UNIVERSITY OF MINNESOTA

PB99-163818

CENTER FOR TRANSPORTATION STUDIES

ITS INSTITUTE

DEVELOPMENT OF A PARALLEL SIMULATION ALGORITHM FOR FREEWAY TRAFFIC FLOWS ON A DISTRIBUTED PERSONAL COMPUTER SYSTEM

Eil Kwon

Center for Transportation Studies

REPRODUCED BY: NTIS
U.S. Department of Commerce
National Technical Information Service
Springfield, Virginia 22161

HUMAN-CENTERED TECHNOLOGY TO ENHANCE SAFETY AND MOBILITY

DISCLAIMER

The contents of this report reflect the views of the authors, who are responsible for the facts and the accuracy of the information presented herein. This document is disseminated under the sponsorship of the Department of Transportation, University Research Institute Program, in the interest of information exchange. The U.S. Government assumes no liability for the contents or use thereof.

PROTECTED UNDER INTERNATIONAL COPYRIGHT
ALL RIGHTS RESERVED.
NATIONAL TECHNICAL INFORMATION SERVICE
U.S. DEPARTMENT OF COMMERCE

Technical Report Documentation Page

1. Report No. CTS 97-08	2.	3. Recipient's Accession No.

4. Title and Subtitle Development of a Parallel Simulation Algorithm for Freeway Traffic Flows on a Distributed Personal Computer System (Phase I)	5. Report Date December, 1997 6.
7. Author(s) Eil Kwon, Ph.D.	8. Performing Organization Report No.
9. Performing Organization Name and Address Center for Transportation Studies 200 Transportation and Safety Building 511 Washington Ave. S.E. Minneapolis, MN 55455	10. Project/Task/Work Unit No. 11. Contract (C) or Grant (G) No. (C) (G)
12. Sponsoring Organization Name and Address Center for Transportation Studies 200 Transportation and Safety Building 511 Washington Ave. S.E. Minneapolis, MN 55455	13. Type of Report and Period Covered Final Report 1995 14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract (Limit: 200 words)

This research investigates the feasibility of developing low-cost, personal computer-based parallel processing procedures that can be applicable to real time simulation of freeway flows.
Specific objectives include,

- Development of a framework for PC-based parallel simulation system
- Enhancements of existing freeway traffic models for parallel simulation
- Development of a PC-based, parallel simulation algorithm for freeway flows.
- Development of a prototype version of a PC-based parallel simulation system
- Performance evaluation of the PC-based parallel freeway simulation system.

17. Document Analysis/Descriptors PC-based parallel simulation system	PC-based parallel freeway simulation system	18. Availability Statement No restrictions. Document available from: National Technical Information Services, Springfield, Virginia 22161

19. Security Class (this report) Unclassified	20. Security Class (this page) Unclassified	21. No. of Pages	22. Price

DEVELOPMENT OF A PARALLEL SIMULATION ALGORITHM FOR FREEWAY TRAFFIC FLOWS ON A DISTRIBUTED PERSONAL COMPUTER SYSTEM

Final Report for Phase 1

Prepared by

Eil Kwon, Ph.D.
Center for Transportation Studies
University of Minnesota
Minneapolis, MN 55455

Sejun Song, Back Young Choi, and KyeongA Yoo
Haesun Park, Ph.D.
Department of Computer Science
University of Minnesota
Minneapolis, MN 55455

July 1999

Published by

Intelligent Transportation Systems Institute of the
Center for Transportation Studies
University of Minnesota
200 Transportation and Safety Building
511 Washington Avenue S.E.
Minneapolis, MN 55455-0375

The opinions, findings, and conclusions expressed in this publication are those of the authors and not necessarily those of the Center for Transportation Studies.

Acknowledgment
This research was financially supported by the ITS Institute, Center for Transportation Studies, and the University of Minnesota. The Traffic Management Center, Minnesota Department of Transportation has provided the data for this research.

TABLE OF CONTENTS

List of Tables

Table Number Page

List of Figures

EXECUTIVE SUMMARY

A personal computer-based, parallel simulation algorithm for freeway traffic flows was developed and implemented on two different types of hardware configurations. The parallel algorithm has distributed memory architecture, where each processor or personal computer simulates a subsection of freeway and interchanges the internal boundary data with the adjacent processor or personal computer.

The macroscopic freeway flow models developed in previous studies were enhanced in this research to fit into the parallel simulation structure. A set of basic freeway segments was identified and macroscopic simulation models were developed for each segment type by enhancing the previous models. The computational cost of each segment was then calculated and stored to be used in estimating the total computational cost of a given freeway.

The parallel simulation algorithm first decomposes a given freeway section into these basic segments and determines the locations of the internal boundaries between subsections by balancing the computational cost of all subsections. The algorithm was first implemented on a network of two personal computers communicating through the Named Pipe communication function under the Windows NT operating system. To test the performance of the PC-network prototype parallel simulation systems, a set of hypothetical freeway sections with different lengths were developed using a 20 mile section of the I-494 freeway in Minneapolis as a basis.

The test results show a speedup of 1.94 with the two-PC network parallel system over the single PC sequential simulation, i.e., 49% reduction of the execution time for one-hour simulation of the 20-mile freeway section. The speedup for one-hour simulation of a 80-mile section was increased to 1.96, which is close to the upper limit of the parallel processing considering the ideal speedup with two processors is 2.0. Next, the parallel algorithm was implemented on a multi-processor personal computer equipped with two Pentium processors using the multithread programming approach. The communication between two processors was handled through global variables, which do not require intermachine communication hardware. The performance test using the same test freeway sections with the two-PC network case show that, for one-hour simulation period, the speedup with two threads over the single thread simulation ranged from 1.94 with a 20-mile section to 1.98 for an 80-mile case. The speedups with three threads was almost the same as those with two threads, since there were only two processors available with the PC used in this testing. It was also noted that, while the speedups with the dual processor PC were constantly better than those with the two-PC network parallel simulation system, the speedup differences were not significant, indicating the possibility of using low-cost personal computers for more computationally extensive applications. Further, the proposed parallel simulation algorithm showed consistent speedups on both hardware platforms with different freeway lengths.

Future research needs include the development of a large scale, real time freeway network simulation system that can quickly analyze traffic conditions using current day traffic data and evaluate alternative operational strategies for a given problem in real time. This requires processing of various computations in parallel, such as simulation, optimal parameter calibration, demand estimation and analysis of results. The availability of the low-cost personal computerbased parallel computing environment provides an important first step in developing such a real time, large scale freeway network analysis system. Further, more diverse applications that can take advantage of the proposed parallel computing structure should be studied. The development of a more efficient communication module with different procedures also needs to be continued. This includes the upgrade of the network card and hub device from 10 Mb to 100 Mb and use of high speed PCs.

I. INTRODUCTION

I.1 Background

One of the key elements in developing Advanced Traffic Management Systems is the ability to determine the best management strategy through on-line assessment of the various alternatives prior to implementation. Such an analysis requires reliable models that can realistically represent traffic behavior and efficient computational algorithms that can generate control solutions in real time. While there has been substantial progress in developing traffic flow models during last decades, there still exists the lack of an efficient, on-line computational environment that can be easily accessed by practicing engineers. To be sure, most existing freeway ramp metering systems use automatic rate-selection strategies, which select appropriate metering rates in real time without employing rigorous calculation.

Although parallel processing has long been recognized as the most efficient approach in reducing computational load, the development of parallel processing algorithms that can be implemented in real time traffic simulation is still in its early stage. A recent approach by German researchers with parallel supercomputers used a single-bit coding scheme to simulate individual vehicle behavior (Nagel, et al., 1994). A more notable attempt to apply a distributed, discrete-event simulation approach to microscopic simulation using a network of Sun workstations was reported by the researchers in MITRE Corp. (Wang, et al., 1993). In this research, a large traffic model was decomposed into submodels, which were distributed over a network of Sun workstations with minimum inter-processor interactions to achieve parallelism.

As reviewed above, the existing applications of parallel processing algorithms for traffic simulation use either mainframe computers or a network of workstations, which are not easily accessible by practicing engineers. Furthermore, most parallel processing algorithms developed to date are machine-specific restricting the portability of the algorithms. Developing an efficient parallel simulation system that can be easily accessible by traffic engineers in the field would make a substantial contribution in determining alternative management strategies, thus improving traffic performance.

I. 2 Research Objectives

This research investigates the feasibility of developing low-cost, personal computer-based parallel processing procedures that can be applicable to real time simulation of freeway flows. Specific objectives include,

- Development of a framework for PC-based parallel simulation system
- Enhancements of existing freeway traffic models for parallel simulation
- Development of a PC-based, parallel simulation algorithm for freeway flows.
- Development of a prototype version of a PC-based parallel simulation system
- Performance evaluation of the PC-based parallel freeway simulation system.

I.3 Report organization

First, the framework for personal computer based parallel computing is developed in Chapter II using the Named Pipe intermachine communication function and the multithread programming approaches. The interprocess communication procedures available in the Windows NT environment are also reviewed in this chapter. Based on the PC-based parallel computing structures, a parallel simulation algorithm for freeway traffic flows is developed in Chapter III. The enhancements of the macroscopic traffic models and the performance analysis of the proposed parallel algorithm using the pseudo code are also included in Chapter III. Chapter IV includes the development of the prototype parallel simulation systems with two different hardware configurations. The testing of the performance for the prototype systems is also included in Chapter IV. Finally, Chapter V contains the conclusions and future research needs.

II. FRAMEWORK FOR PC-BASED PARALLEL COMPUTING

II.1 Overview of Parallel Computing

Parallel computing has made a tremendous impact on many areas of computer applications. By using more than one processor simultaneously with efficient communication between subtasks, parallel computing makes it possible to address many computation-intensive applications, which have been until recently beyond the capability of conventional computing techniques (Kumar, et.al, 1994).

The key elements in effective parallel computing include:

1) Hardware platform that can support fast communications, data sharing among processors and scale up to a large number of processors,

2) Efficient parallel algorithm that can take advantage of a given parallel system architecture,

3) Flexible programming environment and performance evaluation tools.

Since all problems are not equally amenable to parallel processing, i.e., for some problems, assigning partitions to other processors might be more time-consuming than performing the processing sequentially, designing efficient parallel algorithms largely depends on the nature of a given problem and the particular hardware architecture to be employed. However, the following indices are commonly used to evaluate the performance of a parallel algorithm (Kumar, et.al., 1994);

Run time

The serial tun time of a program is the time elapsed between the beginning and the end of its execution on a sequential computer. The parallel run time is the time that elapsed from the moment that a parallel computation starts to the moment that the last processor finishes execution.

Speedup

Speedup is a measure that reflects the relative benefit of executing a program in parallel system. It is defined as the ratio of the time taken to solve a problem on a single processor to the time required to solve the same problem on a parallel system with multiple processors. While an ideal parallel system with n processors can show a speedup equal to n, in practice, due to the communication overhead, the processors in a parallel system can not contribute 100 percent of their time in implementing the given algorithm.

Efficiency

Efficiency is a measure of the fraction of time for which a processor is usefully employed. It is defined as the ratio of speedup to the number of processors, i.e.,

Efficiency = Speedup / # of processors

While the ideal efficiency is equal to one, in practice efficiency is between zero and one.

Cost

Cost of solving a problem on a parallel system can be defined as the product of parallel run time and the number of processors used. Cost reflects the sum of the time that each processor spends solving the problem. The cost of solving a problem on a single processor is the execution time of the sequential algorithm. A parallel system is said to be cost-optimal if the cost of solving a problem on a parallel computer is proportional to the execution time of the sequential algorithm on a single processor. Since efficiency is the ratio of sequential cost to parallel cost, a cost-optimal parallel system has an efficiency of $\Theta(1)$.

In the following sections, a general parallel computing framework using personal computers is developed using the interprocess communication procedures available in the Windows NT operating system. Windows NT is a full 32-bit preemptive multitasking operating system with the multithreading capabilities and can operate a network of Intel-based personal computers(Cowart, 1995., Microsoft Corporation, 1994). First, the interprocess communication procedures in personal computer environment are summarized and the framework for PC-based parallel computing based on interprocess communications are developed.

II.2 Interprocess Communication Procedures in Windows NT

The major communication methods available in the Windows NT environment are Remote Process Control (RPC), Windows Sockets, Named Pipe, Mailslots, NetBIOS, IPX/SPX (Shinha 1996, Cowart, 1995, Andrews, 1996). *RPC* is a collection of libraries and tools that allows the client/server application writer to focus on the application itself, rather than on the low-level communication code. With this tool, the application writer designates one or more functions in the server code that can be remotely called by one or more clients. RPC provides the "glue" that bonds client/server communications. The client calls some function that looks no different than a local procedure call, except that the remote procedure is executed at the server's end. With the aid of RPC tools and libraries, applications ca become independent of transport protocols, and can use the Windows NT security that is built into Microsoft Windows NT RPC. Most important, RPC provides the framework for handling client/server transactions through a simple, familiar programming model.

Named Pipe is another interprocess communication method available in Windows NT. This mechanism is much closer to first in, first out(FIFO) than the "Pipe" mechanism in UNIX. It allows bi-directional flow of data between the client and the server, who can be on the waits to read requests sent by the client. The client opens the pipe and sends requests to the server by writing data into the pipe.

The applications that do not need bi-directional communication can use *Mailslots.* Mailslots provides a unidirectional guaranteed (called "first-class" Mailslots) or non-guaranteed ("second class") flow of data. However in the Windows NT environment, only second class Mailslots are available. Any process can create a Mailslots, and any other process can send it a message by writing a message into the first process's Mailslots. Mailslots always use broadcast datagrams to send and receive data.

Windows NT can function in Novell NetWare LANs. In this environment, an application can also use the same interfaces to achieve peer-to-peer communication. IPX/SPX can be used for both connection-oriented and connection-less communication.

One of the low-level communication methods used in some PC-networking environment is the NetBIOS protocol. NetBIOS exposes a set of interfaces that can be used to write client/server or peer-to-peer applications, and for both connection-oriented and connection-less communication.

Windows NT supports also the Windows Socket interface, which, unlike other PC interfaces, allows applications to use transport protocols(for example, TCP/IP, UDP, AppleTalk) directly, without imposing any "on-the-wire" protocol overhead.

Another important feature of Windows NT is its multitasking capability, which can manage multiple processes that appear to be executing at the same time (Shnha 1996, Andrews, 1996). With its multithread-based, multitasking feature, the Windows NT running on a multiprocessor system can execute multiple processes simultaneously. This multitasking feature also ensures that malfunctioning in one executing application cannot cause unexpected behavior in another application and, more importantly, that a malfunctioning application cannot cause a system crash.

II.3 Development of PC-based Parallel Computing Structures

In this research, two types of PC-based parallel computing structures are developed: 1) a distributed memory structure with a network of personal computers connected with the Named Pipe communication procedure, and 2) a shared memory structure with a multiprocessor PC using the multithreading feature of the Windows NT. The following sections describe the outline of each framework.

II.3.1 Distributed Memory Structure with a Network of Personal Computers

Network Communication Structure with Named Pipe

A pipe is an application-level programming construct or interface that can be used to build an IPC(InterProcess Communication) channel between a client and a server. In Windows NT, there are two types of pipes for IPC. Named Pipes, as the name implies, are named, and can be used for inter-machine communication. Another type of pipe is an anonymous pipe, which can only be used on the same machine.

Named Pipes are used to transmit data between two distinct types of entries, servers and clients. The main difference is that only a server can "create" a pipe. Once a pipe has been created by the server, one or more clients can open it. After the pipe is opened, both the server and the client can call the normal read/write functions to read data from, and to write data into the pipe.

Named Pipe-based Parallel Computing

Using Named Pipe, two adjacent personal computers can interchange data after each PC performs its own calculation. This structure resembles the distributed memory structure in parallel computing and can be applicable for simulating a long freeway section or a large network of intersections by having each PC handle a subsection of freeway or a subnetwork of a large network and interchange the data at the boundaries between two adjacent subsections or subnetworks. Figure 2-1 shows the simplified structure of the Named Pipe client and server transferring data.

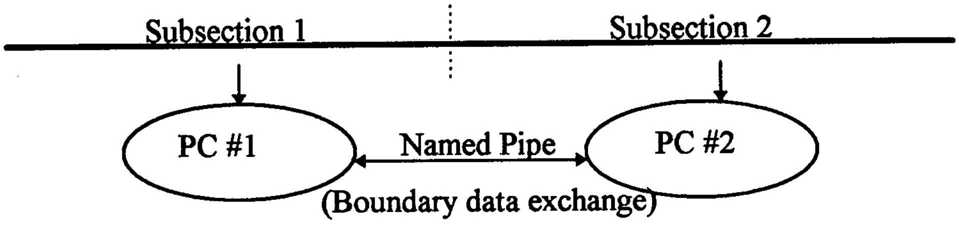


Figure 2-1. Framework based on Named Pipe

II.3.2 Shared Memory Structure with Multithreads-based Programming

Multithreads

When a program starts up, the operating system creates a process for the application and the application starts to execute the instructions in a linear sequence until a jump or call instruction changes the sequence. Each process contains its own private address space, data and code segments, and any resources created or opened by the application. A process can start one or more child processes. Each child process executes in its own address space and contains its own data and code segments. A child process may share one, some, or all of the resources of its parent process. In an operating system such as MS-DOS, each process contains a single thread of execution.

The multithread feature of Windows NT enables a process to have multiple threads of executions. Further, with a multiprocessor hardware platform, multiple threads of executions can be run on multiple processors with each thread representing one process. Every thread shares the process' global variables and virtual address space and any resources owned by the process. Thus, a file opened by one thread can easily be read by another thread within the same process, provided that the second thread can get the handle of the file in some way. Usually, this is done through a global variable. Each thread has its own unique call stack, CPU state and its own thread local storage(TLC) in which threads-specific static data is stored. So undoubtedly, an additional overhead is added to the system overhead for the process with multiple threads, but the additional overhead per thread is generally lower than the overhead caused by each new child process (Cowart 1995, Sinha 1996, Andrews 1996).

With the presence of two or more threads, synchronization and mutual exclusion among threads become crucial for an effective execution multiple processes. More than one process can have a handle to the same synchronization object to make the interprocess synchronization possible. The following objects are available in Windows NT to resolve the synchronization problem;

Critical section, which protects shared resources by ensuring that only one thread can modify the resource at any given time.

Mute, which works much like critical section objects and have additional features that enable them to protect resources when multiple threads are used across different processes and different applications.

Semaphore, which limits the number of threads that can be executed simultaneously.

Event, which prevents a thread from starting until the execution of another thread is complete.

In our application, we need critical section objects for some global variables.

Multithreads-based Parallel Computing

In this research, a parallel computing structure based on the multithreading feature of Windows NT is developed using the critical section object as the synchronization device with a multiprocessor personal computer. The communication between threads are handled through global variables, which resembles a shared memory structure in parallel computing. With a multiprocessor personal computer, a task is divided into multiple subtasks and each subtask is handled by a thread interacting with other threads through global variables. The Windows NT automatically assigns the execution of each thread to different processors. Figure 2-2 illustrates the simplified structure of the multithread-based parallel computing.

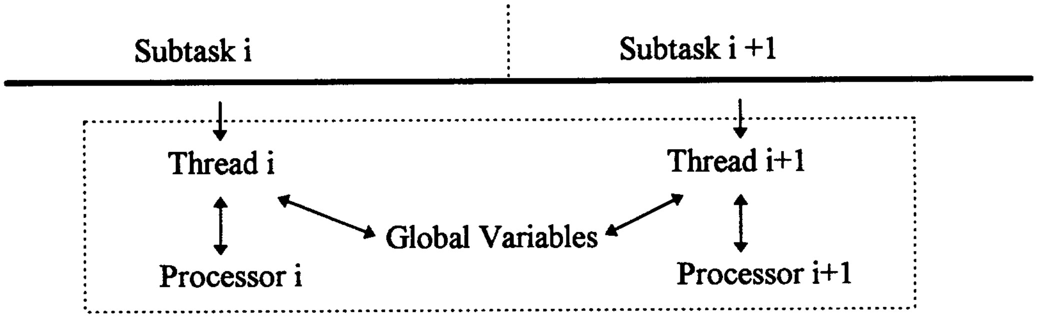
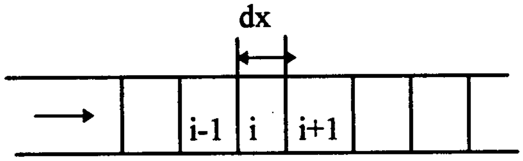


Figure 2-2 Simplified structure of multithread-based parallel computing structure

III. DEVELOPMENT OF A PC-BASED PARALLEL SIMULATION ALGORITHM FOR FREEWAY TRAFFIC FLOWS

III.1 Overview of Macroscopic Freeway Traffic Simulation

Traffic simulation models can be classified into two groups; macroscopic and microscopic. Macroscopic approach treats traffic as a stream and develops algorithms that estimate traffic parameters describing the aggregate behavior of traffic flow through time. Whereas microscopic approach models the behavior of each individual vehicle considering speeds and spacing among adjacent vehicles. In this research, to achieve maximum computational efficiency, a macroscopic approach based on simple continuum modeling is adopted. The basic simulation procedure used in this research for a normal pipeline segment is shown in Figure 3-1.



$$k_i^{n+1} = \frac{1}{2}(k_{i+1}^n + k_{i-1}^n) + \frac{dt}{2dx}(q_{i-1}^n - q_{i+1}^n)$$

$$q_i^{n+1} = q(k_i^{n+1}),$$

$$u_i^{n+1} = q_i^{n+1} / k_i^{n+1}$$

Figure 3-1 Simulation procedure for pipeline segment

In the above figure, k_i^n , q_i^n and u_i^n denotes density, flow and speed of dx i at time step n, respectively. The above finite difference scheme was designed to solve time-dependent compressible flows containing strong shocks (Lax, 1954) and has been applied to develop a freeway simulation software (Kwon, et. al., 1994) in the previous studies. As indicated in the above equations, the simulation algorithm estimates the density of dx_i for the next time step using the information only from dx_{i-1} and dx $_{i+1}$. This makes it possible to develop a distributed computing algorithm, which divides the whole freeway section into a number of subsections, depending on the number of processors, and simulate each subsection independently with the exchange of the boundary dx's information at each time step. The parallel simulation algorithm developed in this research is based on the above principle and takes advantages of the recent developments in the communication functions of the personal computers.

III.2 Development of a PC-based parallel simulation algorithm for freeway flows

Figure 3-2 illustrates the simplified structure of the parallel simulation algorithm developed in this research. For a given freeway section, the algorithm first determines the total computational cost of the whole freeway. The optimal distribution of the computational load to each processor is then determined and the given freeway is divided into multiple subsections based on the number of processors, i.e., number of PCs available in a network or the number of processors in a multiprocessor PC. At each time step, the data at the internal boundaries between two adjacent subsections are first exchanged and each processor or PC simulates the traffic flows of its own subsection with the boundary data from the adjacent subsection.

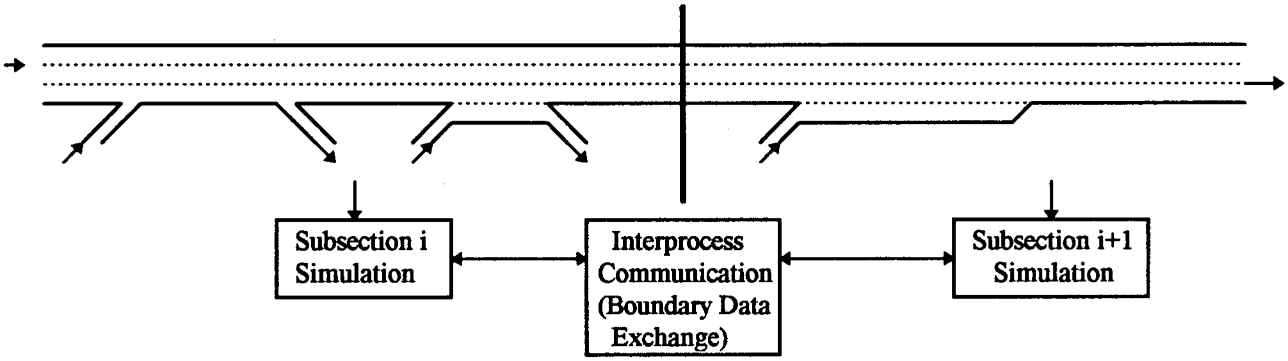


Figure 3-2. Simplified structure of the Parallel Simulation with Distributed Computing

Treatment of boundary data exchange

The simulation algorithm developed in this study requires the data exchange at the internal boundaries between two adjacent subsections at every time step. For example, in Figure 3-3, the traffic data, i.e., density, of dx i (j) needs to be sent to subsection II (I) to determine the density of dx j (i) for the next time step.

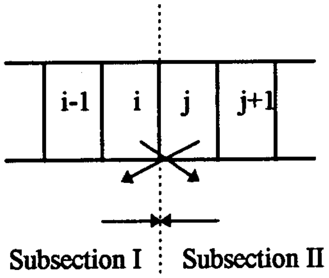


Figure 3-3. Date exchange at internal boundary

In this study, two types of data communication procedures are developed depending on the hardware platform. First, for a multiple computer network-based parallel simulation, a data communication module was developed using the Windows NT-based, Named Pipe function, which handles intermachine communication by treating the first processor as a server and the next processor as a client (Shinha 1996). After the server creates a pipe, both the server and the client can call the normal read/write functions to read data from, and to write data into the pipe. Therefore, at each internal boundary, two communication pipe functions handle the data exchange every one second. For the multithread-based approach with a multiprocessor PC, the communication among multiple threads is performed through global variables, which can be shared by different threads (Shinha 1996, Andrews 1996).

Parallel Simulation Procedures

Figure 3-4 and 3-5 outlines the parallel simulation algorithms developed in this research for two different personal computer hardware platforms, i.e., multiple PC-network and multiprocessor PC. As indicated in these figures, two algorithms are identical except the boundary communication procedure. With the multithread-based approach, the optimal distribution of the computational load to each 'thread' is determined depending on the number of threads to be used, and a given freeway is divided into the same number of subsections as the number of threads. Usually, the number of threads to be used is same as the number of processors available in a given multiprocessor PC. At each time step, the data at the internal boundaries between two adjacent subsections are first exchanged through global variables, and each thread simulates the traffic flows of its own subsection with the boundary data from the adjacent subsection.

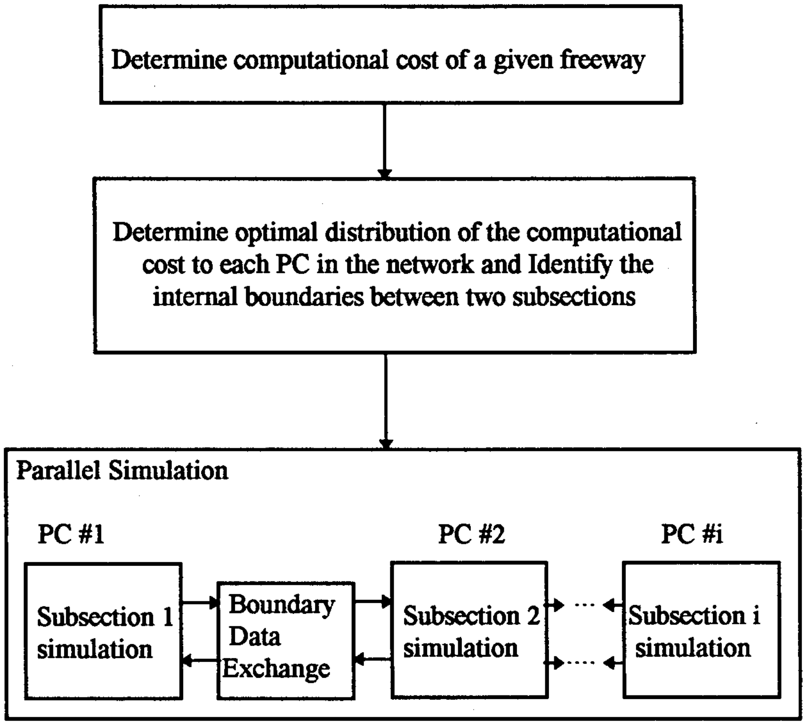

Determine computational cost of a given freeway
Determine optimal distribution of the computational cost to each PC in the network and Identify the internal boundaries between two subsections
Parallel Simulation
PC #1
PC #2
PC #i
Subsection 1 simulation
Boundary Data Exchange
Subsection 2 simulation
Subsection i simulation

Figure 3-4 Outline of the parallel simulation procedure on a PC-Network

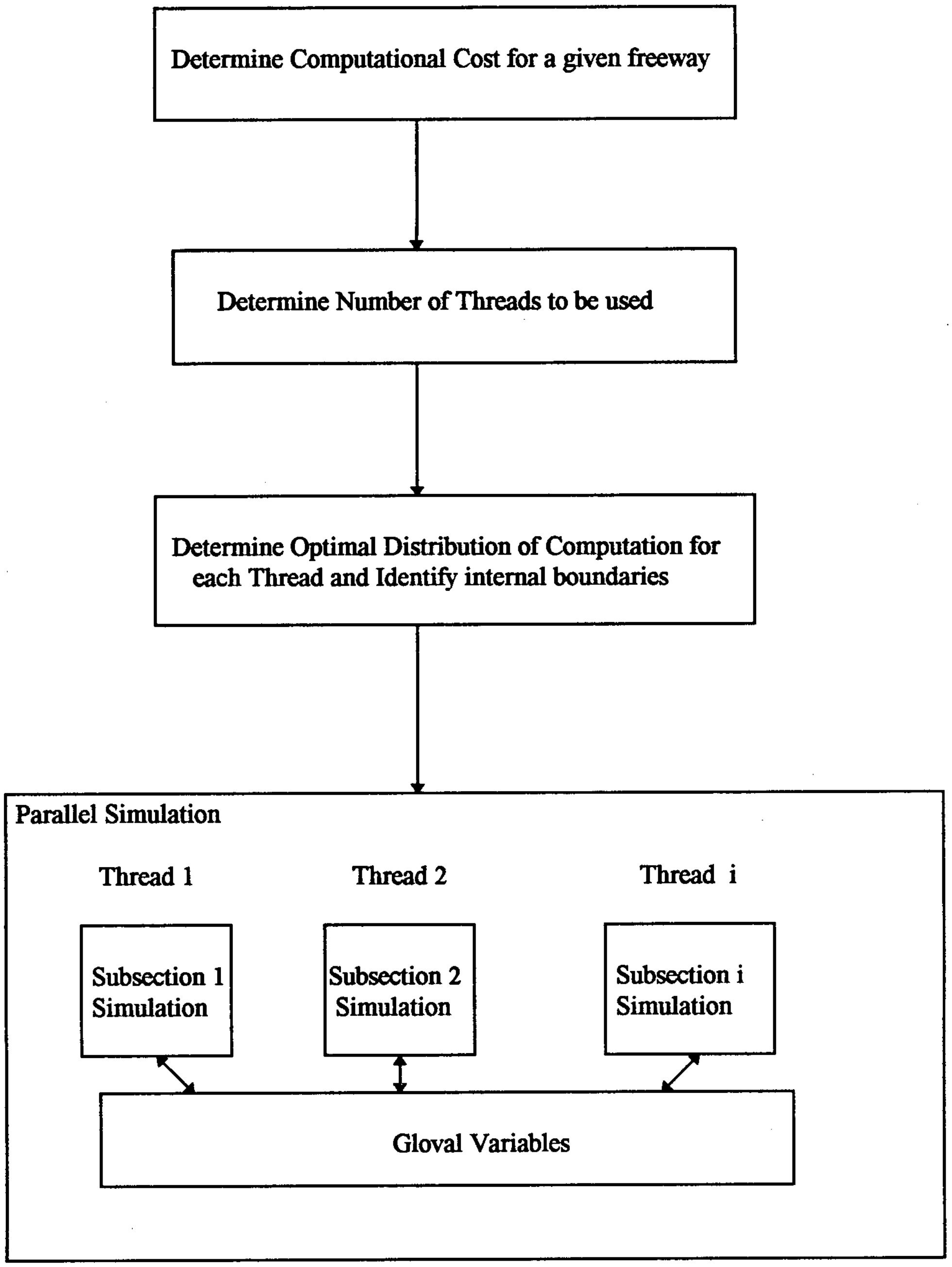
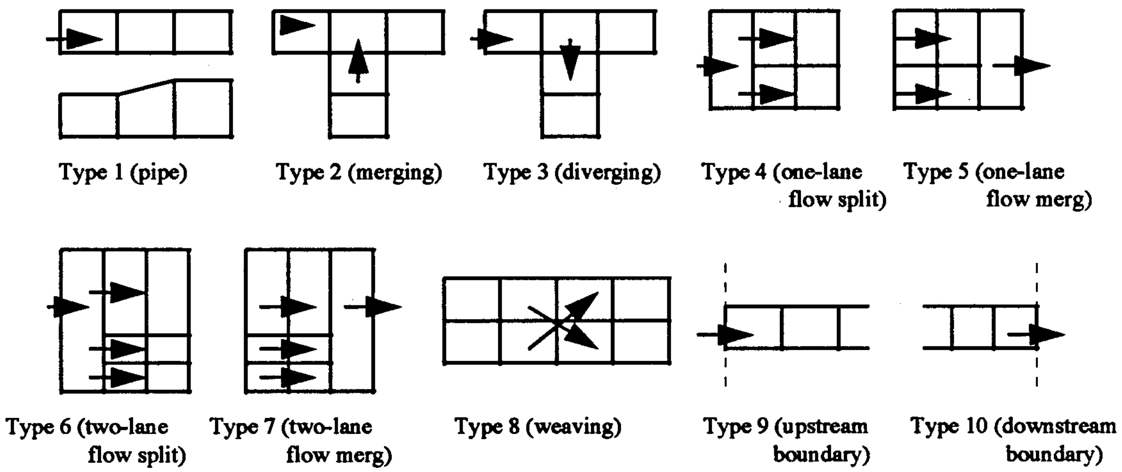

Determine Computational Cost for a given freeway
Determine Number of Threads to be used
Determine Optimal Distribution of Computation for each Thread and Identify internal boundaries
Parallel Simulation
Thread 1
Thread 2
Thread i
Subsection 1 Simulation
Subsection 2 Simulation
Subsection i Simulation
Gloval Variables

Figure 3-5. Outline of the parallel simulation procedure on a multiprocessor PC

III. 3 Enhancement of Freeway Traffic Flow Models for Parallel Simulation

III.3.1 Overview of Modeling Enhancements

This section presents the macroscopic freeway models enhanced for the parallel simulation algorithm developed in the previous section. First, a set of basic freeway segment types was identified and, for each segment type, a macroscopic simulation model is developed by enhancing the existing traffic models developed in the previous research (Kwon, et. al., 1994). The segmentation developed in this research facilitates the calculation of the computational cost for a given freeway, so that the optimal distribution of the computational load can be performed efficiently. Figure 3-6 illustrates the basic segments that can be combined to represent most freeways in the U.S. The modeling enhancements conducted in this research enables the explicit consideration of the effects of interrupted flow, such as merging, diverging and weaving using continuum modeling approach. The detailed algorithm for each segment is described in the next section.



Type 1 (pipe)

Type 2 (merging)

Type 3 (diverging)

Type 4 (one-lane flow split)

Type 5 (one-lane flow merg)

Type 6 (two-lane flow split)

Type 7 (two-lane flow merg)

Type 8 (weaving)

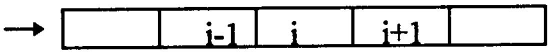
Type 9 (upstream boundary)

Type 10 (downstream boundary)

Figure 3-6 Basic segment types

III.3.2 Enhanced traffic models for freeway segments

Type 1 segment:

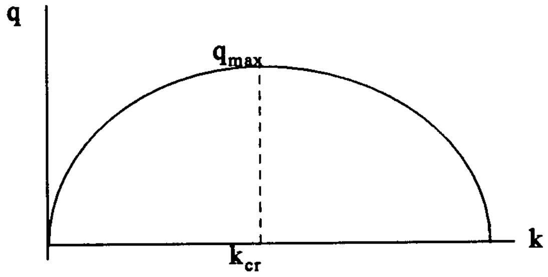


$$k_i^{n+1} = \frac{1}{2}(k_{i+1}^n + k_{i-1}^n) + \frac{dt}{2dx}(q_{i-1}^n - q_{i+1}^n)$$

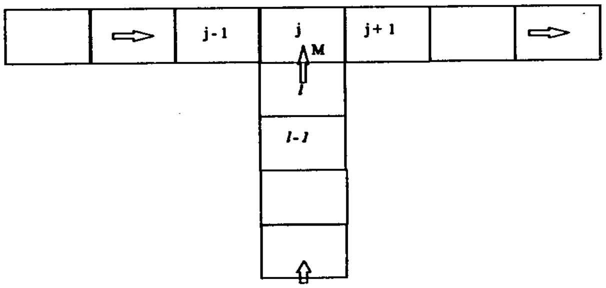
$$q_i^{n+1} = q_i(k_i^{n+1}),$$

$$u_i^{n+1} = q_i^{n+1} / k_i^{n+1}$$

where, $q_i(k)$ = q - k relationship for dx i



Type 2 segment:



$$k_j^{t+1} = \frac{1}{2}(k_{j-1}^t + k_{j+1}^t) + \frac{\Delta t}{\Delta x}(\frac{1}{2}(q_{j-1}^t - q_{j+1}^t) + M_j^t)$$

$$k_l^{t+1} = \frac{1}{2}(k_l^t + k_{l-1}^t) + \frac{\Delta t}{\Delta x}(\frac{1}{2}(q_{l-1}^t + q_l^t) - M^t)$$

Where M^t_j = Minimum [Q_j, F_l]

$Q_j = \alpha * q_{allow,j}$; Available Space at J for M

$q_{allow\ j} = k_j * u_j$ if $k_j <= k_{cr,j}$

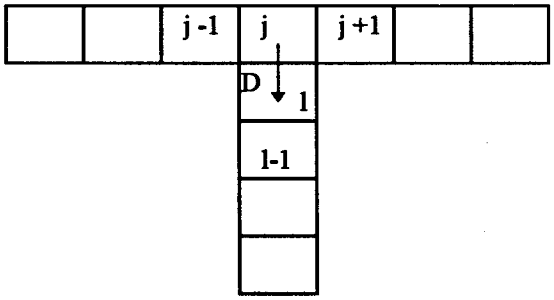
$= q_{max}$ else

$\alpha = r + (1.0 - r) * k_l/k_{jam}$; $r = 0.2$

$F_l = q_l(k_l)$ if $k_l <= k_{cr,l}$

$= q_{max,l}$ else

Type 3 segment



$$k_j^{t+1} = \frac{1}{2}(k_{j-1}^t + k_{j+1}^t) + \frac{\Delta t}{\Delta x}(\frac{1}{2}(q_{j-1}^t - q_{j+1}^t) - D^t)$$

$$k_l^{t+1} = \frac{1}{2}(k_l^t + k_{l-1}^t) + \frac{\Delta t}{\Delta x}(D^t - \frac{1}{2}(q_{l-1}^t + q_l^t))$$

where, D^t = Minimum [$Q_{j,potential}$, $Q_{l,available\ space}$, Exit Demand at t]

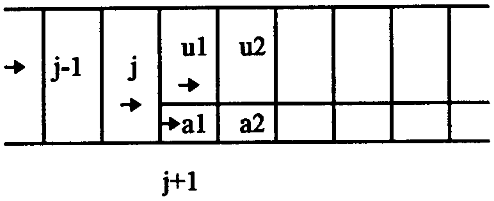
$Q_{j,potential} = k_j * u_j$ if $k_j <= k_{cr,j}$

$= q_{max,j}$ else

$Q_{l,available\ space} = q_{max,l}$ if $k_l <= k_{cr,l}$

$= k_l * u_l$ else

Type 4 segment



$$k_i^{n+1} = \frac{1}{2}(k_{i+1}^n + k_{i-1}^n) + \frac{dt}{2dx}(q_{i-1}^n - q_{i+1}^n)$$

qj+1 = qu1 + qa1

Get k_{j+1} from q-k relationship of j+1 with q_{j+1}

For k_{u1} and k_{a1};

$$k_{j+1}^{n+1} = \frac{1}{2}(k_i^n + k_{i+2}^n) + \frac{dt}{2dx}(q_i^n - q_{i+2}^n)$$

where, $q_{j+2} = q_{u2} + q_{a2}$

k_{j+2} to be obtained from q-k function of j+2 with q_{j+2}

Get q_{i+1}^{n+1} with k_{j+1}^{n+1} using q-k function for j

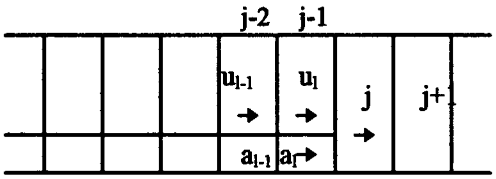
Split q_{i+1}^{n+1} into q_{a1}^{n+1} and q_{u1}^{n+1} ;

q_{a1}^{n+1} = Maximum [Min(Exit demand at n+1, q_{j+1}, available), q_{j+1}/(# of lanes)]

$q_{u1}^{n+1} = q_{i+1}^{n+1} - q_{a1}^{n+1}$,

Get k_{a1} and k_{u1} from respective q-k functions with q_{a1} and q_{u1}.

Type 5 Segment



For k_{ul}^{t+1} & k_{al}^{t+1}

1. Get k_{j-1}^{t+1} first

$k_{j-1}^{t+1} = 1/2\ (k_{j-2}^t + k_j^t) + \Delta t/2\Delta x\ (q_{j-2}^t - q_j^t)$

to do as above, we need k_{j-2} & q_{j-2}

$q_{j-2}^t = q_{ul-1}^t + q_{al-1}^t$

get k_{j-2}^t with q_{j-2}^t from the q-k function for j

if k_{ul-1}^t is congested, then $k_{j-2}^t \rightarrow$ congested.
else uncongested

get k_{j-1}^{t+1} and determine q_{j-1}^{t+1} using q-k function for j

2. $q_{al}^{t+1} = q_{j-1}^{t+1}$/(number of lanes in j-1)

get k_{al}^{t+1} with q_{al}^{t+1} from q-k for a_l

$q_{ul}^{t+1} = q_{j-1}^{t+1} - q_{al}^{t+1}$

get k_{ul}^{t+1} with q_{ul}^{t+1} from q-k fro u_l

For k_j^{t+1}

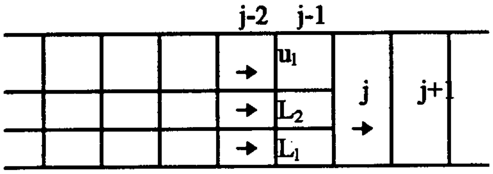
$k_j^{t+1} = 1/2\ (k_{j-1}^t + k_{j+1}^t) + \Delta t/2\Delta x\ (q_{j-1}^t - q_{j+1}^t)$

$q_{j-1}^t = q_{ul}^t + q_{al}^t$

get k_{j-1}^t from q-k for j with q_{j-1}^t

if k_{ul}^t is congested, i.e., $k_{ul}^t >= k_{ul,\ cr}$ then $k_{j-1}^t \rightarrow$ congested.

Type 6 Segment



1) $k_j^{t+1} = 1/2\ (k_{j-1}^t + k_{j+1}^t) + \Delta t/2\Delta x\ (q_{j-1}^t - q_{j+1}^t)$

where $q_{j-1}^t = q_{ul}^t + q_{l2}^t + q_{l1}^t$

get k_{j-1}^t from the q-k curve of j with q_{j-1}^t (Same as other case)

2) k_{ul}^{t+1}, k_{l2}^{t+1}, k_{l1}^{t+1}

Get k_{j-1}^{t+1}

$$k_{j-1}^{t+1} = 1/2\ (k_{j-2}^t + k_j^t) + \Delta t/2\Delta x\ (q_{j-2}^t - q_j^t)$$

get q_{j-2}^t & k_{j-2}^t same as 1)

Get $q_{j-1}^{t+1} = q_{j-1}(k_{j-1}^{t+1}) \rightarrow$ from q-k for j

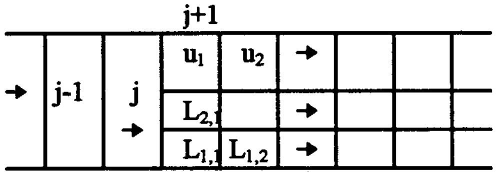
$q_{L1}^{t+1} = q_{L2}^{t+1} = q_{j-1}^{t+1}/$ (number of lanes)

$q_{ul}^{t+1} = q_{j-1}^{t+1} - (q_{L1}^{t+1} + q_{L2}^{t+1})$

get k_{L1}, k_{L2}, k_{ul} from the corresponding q-k curves.

if k_{j-1}^{t+1} is congested, then all k's are congested
else all k's uncongested.

Type 7 Segment



1) $k_j^{t+1} = 1/2\ (k_{j-1}^t + k_{j+1}^t) + \Delta t/2\Delta x\ (q_{j-1}^t - q_{j+1}^t)$

$q_{j+1} = q_{u2} + q_{L2,2} + q_{L1,2}$

get k_{j+1} from the q-k function of j with q_{j+1} (same as other case)

2) k_{u1}, $k_{L2,1}$, $k_{L1,1}$

$k_{j+1}^{t+1} = 1/2\ (k_j^t + k_{j+2}^t) + \Delta t/2\Delta x\ (q_j^t - q_{j+2}^t)$

where $q_{j+2}^t = q_{u2}^t + q_{L2,2}^t + q_{L1,2}^t$

get k_{j+2}^t from the q-k curve function of j+1 (same as other case)

get k_{j+1}^{t+1} & q_{j+1}^{t+1}

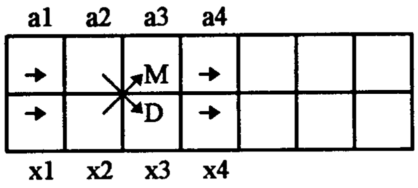
$q_{L1,1} = \text{Min}\ [x_t, q_{L1}, \text{capacity}]$

where x_t = Exit demand at t (given by user)

$q_{L2,1} = \text{Max}\ [(x_t - q_{L1,1}),\ (q_{j+1}^{t+1} - q_{L1,1})/\ (\text{number of lanes} - 1)]$

get $k_{L1,1}$ and $k_{L2,1}$ from the q-k functions for L_1, L_2.

Type 8 Segment



$$k_{a2}^{t+1} = 1/2\,(k_{a1}^{t} + k_{a3}^{t}) + \Delta t/2\Delta x\,(q_{a1}^{t} - q_{a3}^{t})$$

$$k_{a3}^{t+1} = 1/2\,(k_{a2}^{t} + k_{a4}^{t}) + \Delta t/\Delta x\,[1/2\,(q_{a2}^{t} - q_{a4}^{t}) + (m^{t} - d^{t})]$$

$$M^{t} = \mathrm{Min}\,[(\alpha_{t} * q_{a3}, \text{avail. space}), (q_{x2}, \text{potential})]$$

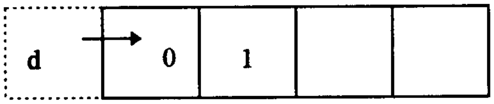
$\alpha_{t} \Rightarrow$ determined same as type 2 segment.]

$$D^{t} = \mathrm{Min}\,[\text{Exit Demand}^{t}, (q_{a2}, \text{potential}), (q_{x3}, \text{available space})]$$

$$k_{x2}^{t+1} = 1/2\,(k_{x1}^{t} + k_{x2}^{t}) + \Delta t/\Delta x\,[1/2\,(q_{x1}^{t} + q_{x2}^{t}) - M^{t}]$$

$$k_{x3}^{t+1} = 1/2\,(k_{x3}^{t} + k_{x4}^{t}) + \Delta t/\Delta x\,[D^{t} - 1/2\,(q_{x3}^{t} + q_{x4}^{t})]$$

Type 9 Segment : first Δx (Main freeway & on-ramp)



$$k_{0}^{t+1} = 1/2\,(k_{0}^{t} + k_{1}^{t}) + \Delta t/\Delta x\,(q_{I}^{t} - 1/2\,(q_{0}^{t} + q_{1}^{t}))$$

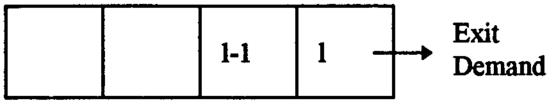
$q_{I}^{t} = q_{0}(k_{d})$ if $k_{d} <= k_{cr,o}$
else $q_{I}^{t} = q_{0,\,max}$

$$k_{d}^{t+1} = k_{d}^{t} + \Delta t/\Delta x\,(\text{Entering Demand}^{t} - q_{I}^{t})$$

If the condition of the entering flow is given, i.e., congested or not,

then, k_{d}^{t} is determined from the q-k function of dx 0 with the given entering flow value.

Type 10 Segment : Downstream boundary dx (mainline and off-ramp)



Option #1: No exit demand specification

$$k_l^{t+1} = 1/2\ (k_{l-1}^t + k_l^t) + \Delta t/2\Delta x\ (q_{l-1}^t - q_l^t)$$

Option #2: Exit demand given

$$k_l^{t+1} = k_l^t + \Delta t/\Delta x\ [1/2\ (q_{l-1}^t + q_l^t) - \text{Exit demand}^t\,]$$

Option #3: Exit flow condition given

$$k_l^{t+1} = 1/2\ (k_{l-1}^t + k_x^t) + \Delta t/2\Delta x\ (q_{l-1}^t - q_x^t)$$

where, q_x^t = Exit flow at t given by user

k_x^t determined by the q-k function of l with q_x^t and its condition.

III. 4 Complexity Analysis of Parallel Simulation Algorithm

To determine the computational complexity of each segment type, the number of floating point operations was counted for each model using the pseudo code with the most complex traffic conditions, which resulted in the upper bound in terms of computation cost for each segment type. For example, the type 1 requires 5n additions, 5n multiplication's and n divisions where n is the number of time steps. Based on the test performed on a 200MHz Pentium 686 processor, the following execution time ratio was obtained;

$$+ : - : * : / : \text{sqrt} = 1 : 1 : 1 : 11.5 : 69.0$$

Table 3-1 includes the computational load of each segment type in Figure 3-6. Appendix includes the complete set of pseudo code for each segment type, which was used to estimate the computational cost.

Table 3-1. Computation Cost for each segment type

Type	Computation cost	Type	Computation cost
1	12.6 n C	6	119 n C
2	40.2 n C	7	124.6 n C
3	39.2 n C	8	80.4 n C
4	99.9 n C	9	22.6 n C
5	94.9 n C	10	13.6 n C

C: execution time for one addition,

n: number of time step.

Performance analysis

The theoretical performance of the parallel simulation algorithm was analyzed using a 20 mile section of I-494 freeway in Minneapolis, Minnesota, as an example. Figure 3-7 illustrates the geometry of the example freeway. A performance index quantifying the benefit of parallel simulation is defined as the ratio of the sequential execution time on a single PC to the execution time on a parallel system with multiple PCs. Assuming that the parallel simulation is performed on a three-PC network, the complexity analysis was performed and the example freeway was divided into three subsections with approximately same computational load. Table 3-2 shows the number of different segment types for each subsection. Based on the complexity analysis results, the sequential and parallel execution times for the example freeway are estimated as follows;

Sequential execution time, $T_s = 20690Cn + 98473.4C$,

Parallel execution time, $T_p = 7034.1Cn + 34092.8C + T_c$,

where, T_c is the communication time between two adjacent processors.

When T_c is negligible, the speedup ratio, i.e., T_s/T_p, becomes 2.94, which is close to 100% efficiency. However, the actual performance of the parallel system can be substantially affected by the communication time between two PCs or two processors in a two-processor PC.

Table 3-2. Number of segment types for each subsection

Segment Type	**Section I**	**Section II**	**Section III**
1	311	376	355
2	3	1	3
3	1	3	3
4	10	7	8
5	9	8	8
6	0	0	0
7	0	0	0
8	10	4	4
9	9	5	5
10	7	6	6

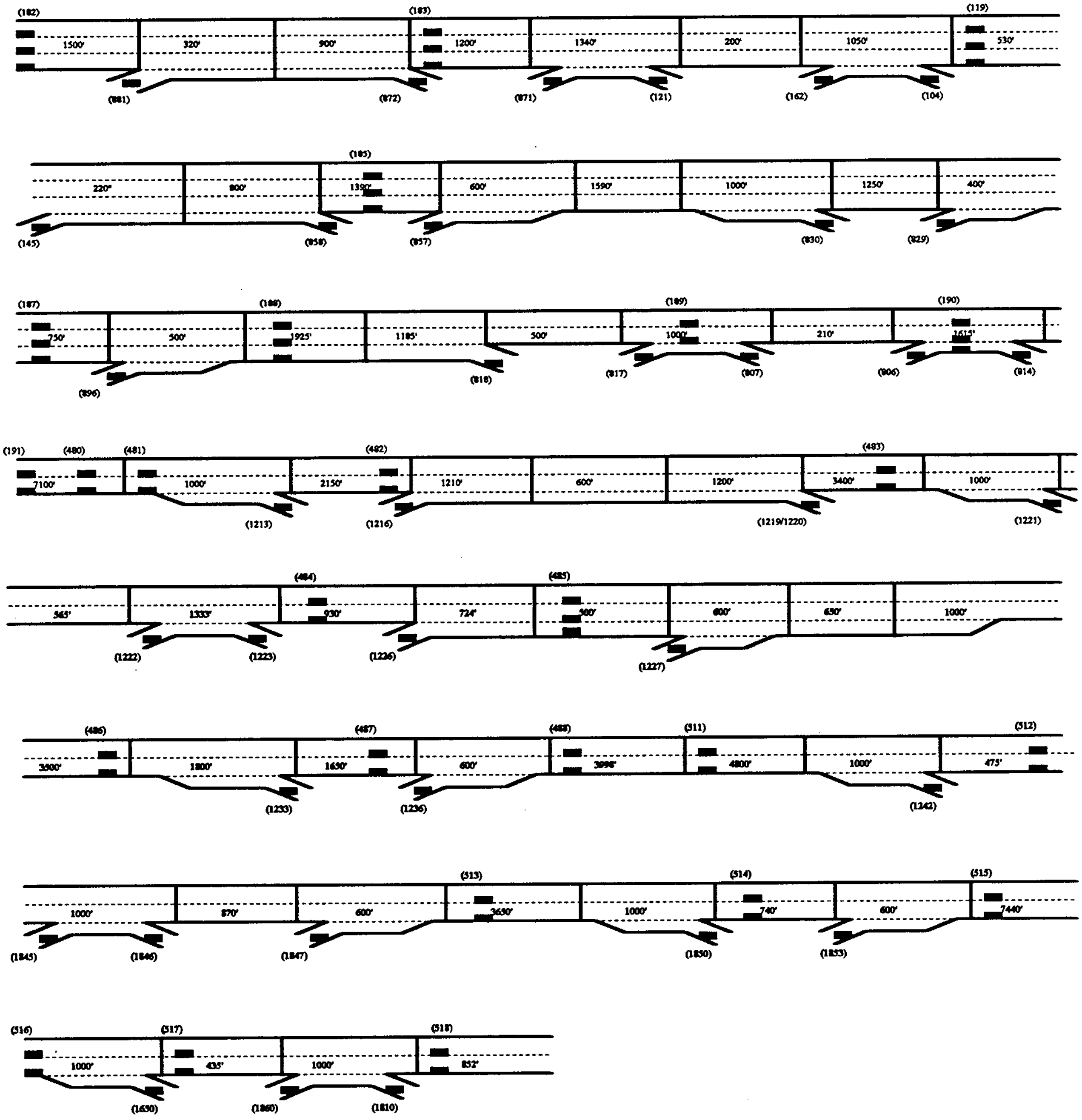


Figure 3-7 I-494 test freeway section

IV. DEVELOPMENT AND EVALUATION OF PROTOTYPE PC-BASED PARALLEL SIMULATION SYSTEM

IV.1 Prototype parallel simulation system with a two-PC network

Using the data communication module with the Named Pipe function, a prototype parallel simulation system was developed by networking two IBM-PC compatible personal computers equipped with a Pentium Pro 686-200 MHz processor. Figure 4-1 shows the structure and pseudo code of the two-PC network parallel simulation system. After determining the internal section boundary for a given freeway section by balancing the computational load between two PCs, the server creates a Named Pipe. Once the simulation starts, the server waits for the client to connect and reads the data requests from the client. After sending the boundary traffic data, i.e., density and flow rate for the boundary dx's, to the client, the server starts the computation for the simulation of the remaining freeway segments allocated to the server. After the server finishes the simulation of its subsection traffic flows for the current time step, it goes back to the waiting mode until the hand shaking with the client is established. The operation of the client is similar to that of the server. The above server-client interaction through the Named Pipe communication procedure continues until the end of the simulation period. Table 4-1 shows the hardware specification of the prototype parallel simulation system developed in this research.

Table 4-1 Prototype System Hardware Specification(PC-Network)

	Server and Client
CPU	Intel Pentium Pro 686-200 MHz
RAM	16 MB
Ethernet Card (Twisted Pair)	SMC 10 Mb
Operating System	Windows NT v3.51
Network software	Microsoft Windows Network
Hub	SMC TigerHub TP6B (10 Mb)
Compiler	Microsoft Visual C++ 2.0

Pseudo Code for Server

```
Read Data
Determine internal section
Create a Named Pipe
Begin Simulation
  Wait for client to respond
   Send boudary data to client
   Receive boundary data from
   Acknowledge receipt to client
   Simulate other segments with boundary
End Simulation
Close the Named Pipe
```

Pseudo Code for Client

```
Read Data
Determine internal section
Begin Simulation
   Open a Named Pipe
   Send boudary data to server
   Receive boundary data from server
   Close the Named Pipe
   Simulate other segments with boundary
End Simulation
```

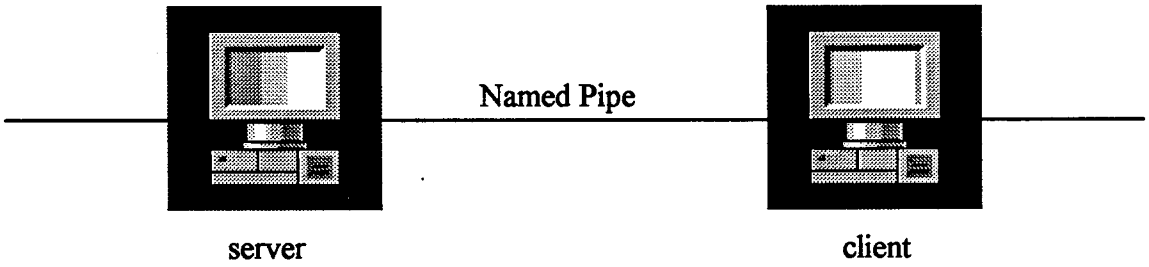


Figure 4-1 Structure of the two PC-Network parallel simulation system

IV.2 Dual-processor PC-based, parallel simulation System

For the multiprocessor PC system with the multithreading approach, a two-processor PC from ALR with 2 Pentium Pro 686-200 MHz processors was used in this research. After determining the internal section boundaries for a given freeway section by balancing the computational load between threads, the algorithm creates threads according to the specified number by user. After storing the boundary traffic data, i.e., density and flow rate for the boundary dx's, into the pre-specified global variables, each thread starts the computation for the simulation of the assigned freeway subsection. When each thread finishes the simulation of its subsection traffic flows for the current time step, it stores the new boundary data into the global variables and reads the new boundary data of the adjacent threads from the global variables. Table 3 shows the hardware specification of the multiprocessor PC prototype parallel simulation system developed in this research.

Table 4-2 Prototype System Hardware Specification (multiprocessor PC)

	Specification
CPU	Intel Pentium Pro 686-200 MHz (2 CPU)
RAM	16 MB
Operating System	Windows NT v3.51
Compiler	MS Visual C++ 2.0

IV.3 Performance evaluation of the prototype parallel simulation systems

The performance of the prototype systems was evaluated by comparing the execution time of the parallel algorithm with that of the single processor sequential algorithm implemented with each hardware platform. The I-494 test section used in the previous chapter, shown in Figure 3-9, for the complexity analysis was also used for testing the performance of each prototype system. To evaluate the performance of the two-PC system with different freeway lengths, hypothetical freeway sections up to 80 miles were created using the 20 mile I-494 section. Table 4-3, 4 and Figure 4-2 show the comparison of the execution times with single PC and two-PC network parallel simulation for different lengths of freeways. As indicated in these tables, the speedup with the two-PC network ranges from 1.94 for the 20 mile section to 1.96 for the 80 mile section for one hour simulation, which indicates almost perfect performance of the two-PC parallel system. It can be noted that the speedup and efficiency with longer simulation time and freeway lengths are slightly better than those of shorter simulation time and freeways, while the differences are not significant.

Table 4-3 Execution time (in seconds) for single PC and Two-PC parallel simulation

Freeway Length	1 Hour Simulation		20 Minutes Simulation	
	1 PC	2 PC	1 PC	2 PC
20 miles	42.781	22.031	12.359	7.11
40	87.656	44.938	25.281	14.234
60	132.484	68.031	38.907	21.375
80	179.735	91.64	53.828	28.969

Table 4-4 Speedup and Efficiency with two-PC Network parallel simulation

Freeway Length (miles)	1 Hour simulation		20 Minutes simulation	
	Speedup	Efficiency	Speedup	Efficiency
20	1.94	0.97	1.74	0.87
40	1.95	0.98	1.78	0.89
60	1.95	0.97	1.82	0.91
80	1.96	0.98	1.86	0.93

Speedup = Single PC execution time / Two-PC parallel system execution time
Efficiency = Speedup / number of processors

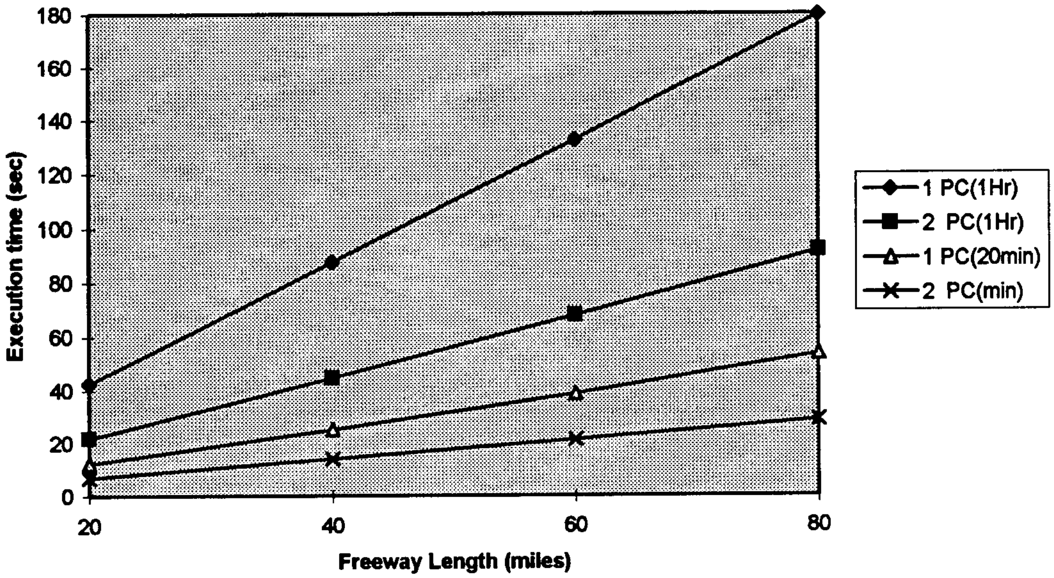


Figure 4-2 Comparison of the execution times between single PC and two-PC parallel simulation

Table 4-5 & 6 and Figures 4-3 & 4 show the execution times and the speedup/efficiency with the dual processor PC with different number of threads. The execution time with single thread represents the execution time with the single processor. As indicated in these tables, the execution time with two threads on a dual processor personal computer is in average 49% faster than with the single thread approach for one hour simulation, while 46% faster for the 20 minute simulation. However, the execution time with three threads are almost same as that of two threads, indicating the limitations with a dual processor personal computer platform. It can be also noted that the execution time of two threads is slightly, but constantly faster than that of two-PC network system, which uses an external communication cable to link two PCs. Overall, the comparison of the execution times between two-PC network and two-thread parallel simulation systems demonstrates the strong performance of the prototype parallel simulation systems developed in this research and thus the feasibility of the personal computer-based parallel processing.

Table 4-5 Execution time with multithread-based parallel simulation on a dual processor PC

Freeway Length	1 Hour Simulation			20 Minutes Simulation		
	1 Thread	2 Thread	3 Thread	1 Thread	2 Thread	3 Thread
20 miles	42.781	22	19.859	12.359	6.859	6.75
40	87.656	44.578	46.187	25.281	13.14	13.671
60	132.484	66.75	66.734	38.907	19.344	21.125
80	179.735	90.641	89.125	53.828	28.641	27.625

Table 4-6 Speedup and Efficiency with multithread-based parallel simulation

Freeway	1 Hour Simulation				20 Minutes Simulation			
Length	2 Threads		3 Threads		2 Threads		3 Threads	
(miles)	S	E	S	E	S	E	S	E
20	1.94	0.97	2.15	1.08	1.80	0.90	1.83	0.92
40	1.97	0.98	1.90	0.95	1.92	0.96	1.85	0.92
60	1.98	0.99	1.99	0.99	2.01	1.01	1.84	0.92
80	1.98	0.99	2.02	1.01	1.88	0.94	1.95	0.97

Speedup = Single PC execution time / Two-PC parallel system execution time
Efficiency = Speedup / number of processors

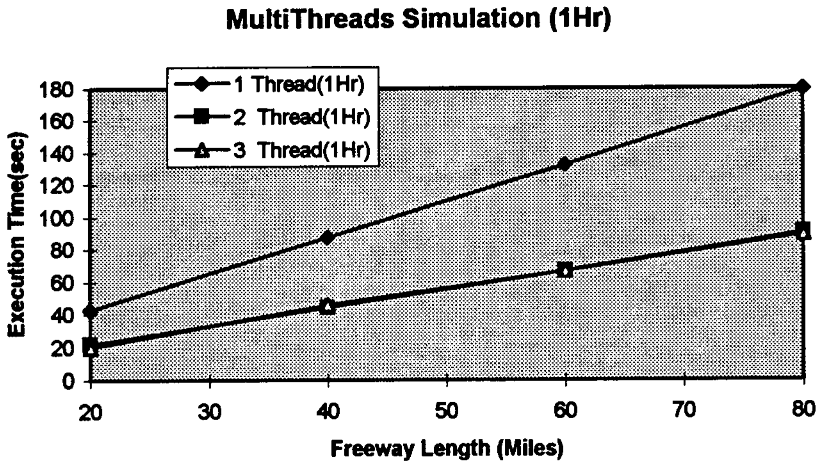


Figure 4-3 Comparison of the execution times for one-hour simulation with multithreads

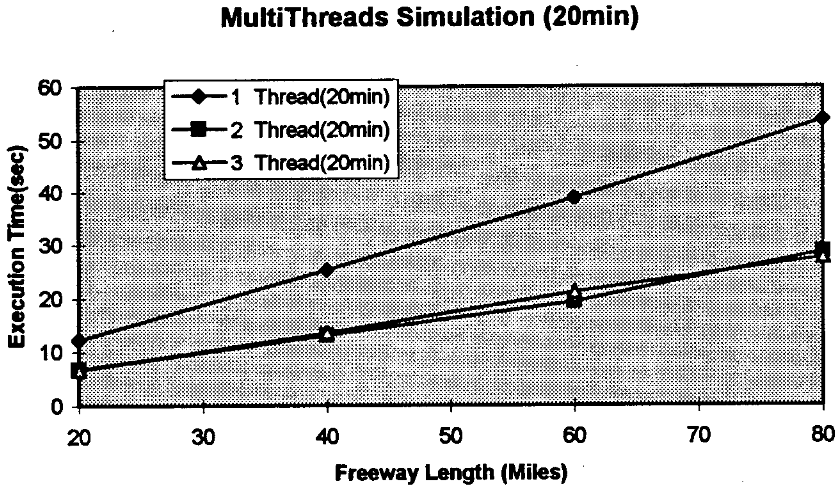


Figure 4-4 Comparison of the execution times for 20-minute simulation with multithreads

V. CONCLUSIONS AND FUTURE RESEARCH NEEDS

This report summarized the final results of the research to develop a prototype system for personal computer (PC)-based parallel simulation of freeway traffic flows. First, the interprocess communication procedures and the multithread programming approaches in a PC network environment including multiprocessor PCs were reviewed and a framework for the PC-based parallel computing was developed using the distributed memory architecture. Next, a parallel simulation algorithm was developed for freeway traffic flows using the enhanced macroscopic traffic flow models. The parallel simulation algorithm first divides a given freeway into a set of subsections by optimally distributing the total computational cost to each subsection depending on the available number of PCs or processors to be used. Each processor simulates a subsection of freeway and interchanges the internal boundary data with the adjacent processor, which simulates its own freeway subsection. The macroscopic freeway flow models developed in the previous studies were enhanced in this research to be suitable for the parallel simulation structure. A set of basic freeway segments was identified and macroscopic simulation models for each segment type were developed by enhancing the previous models. The computational cost of each segment was then calculated and stored to be used in estimating the total computational cost of a given freeway. The parallel simulation algorithm decomposes a given freeway section into these basic segments and determines the locations of the internal subsection boundaries by distributing the total computation cost to each processor.

The parallel simulation algorithm was implemented with two types of hardware configurations: a network of PCs and a single multiprocessor PC. First, two Intel Pentium-based personal computers were networked together with the Named Pipe communication function under the Windows NT operating system. The other type takes advantage of the multithread-based programming approach with a multiprocessor personal computer, where the Windows NT allocates each computation thread to each processor. The communication between threads is performed through global variables, which eliminates the need for external intermachine communication hardware. In this research, a dual processor PC with the Intel Pentium processors was used.

The performance of both prototype systems were tested and compared with that of the single processor-based, sequential simulation algorithm. A 20 mile section of the I-494 freeway in Minneapolis, Minnesota, was used as the base test freeway section and a set of hypothetical freeway sections with different lengths were created to evaluate the performance of the parallel algorithm with various conditions. The test results show the speedup of 1.94 with the two-PC network parallel system over the single PC sequential simulation, i.e., 49% reduction of the execution time for one hour simulation of the 20 mile freeway section. The speedup for one hour simulation of a 80 mile section was increased to 1.96, which is close to the upper limit of the parallel processing considering the ideal speedup with two processors is 2.0. The speedup values for a 20 minute simulation with different lengths range from 1.74 for 20 mile to 1.86 for the 80 mile case, indicating the increased benefit of the proposed parallel simulation system with longer simulation periods.

The performance of the dual processor PC-based parallel simulation showed similar results to that of the two-PC network system. For one hour simulation period, the speedup with two-threads over the single thread simulation ranged from 1.94 for a 20 mile section to 1.98 for a 80 mile case. The speedups with three-threads was almost same as those with two-threads, since there were only two processors available with the PC used in this testing. It was also noted that the speedups with the dual processor PC were slightly, but constantly better than those with the two-PC network parallel simulation system, which indicates the effects of the external communication. Further, the proposed parallel simulation algorithm showed consistent speedups on both hardware platforms with different freeway lengths.

Further research needs include the development of a large scale, real time freeway network simulation system that can quickly analyze the traffic conditions using current day traffic data and evaluate alternative operational strategies for a given problem in real time. This requires real time processing of various computations in parallel, such as simulation, optimal parameter calibration, demand estimation and analysis of results. The availability of the low cost personal computer-based parallel computing environment provides an important first step in developing such a real time, large scale freeway network simulation system. Further, more diverse applications that can take advantage of the proposed parallel computing structure should be studied. The development of more efficient communication module with different procedures also needs to be continued. This includes the upgrade of the network card and hub device from 10 Mb to 100 Mb and use of high speed PCs.

REFERENCES

1. Nagel, K. and Schleicher, A. "Microscopic traffic modeling on parallel high performance computers", Parallel Computing v.20, n1, pp. 125-146, 1994.
2. Wang P., Niedringhaus, W. "Distributed/Parallel traffic simulation for IVHS applications", Proceedings of the Winter Simulation Conference, IEEE, pp. 1225-1330, 1993.
3. Microsoft Corporation, "WindowsNT Workstation Installation Guide", 1994.
4. Cowart, "WindowsNT^{TM} 3.5, 2nd Edition", 1995.
5. Sinha, A., "Network programming in Windows NT", Addison Wesley, 1996.
6. Andrews, M., "Winsows NT programming", M&T Books, 1996.
7. Kumar, V., et. al., "Introduction to Parallel Computing", University of Minnesota, 1994.
8. Lax, "Weak solution of non-linear hyperbolic equation and their numerical computations.", Comm. Pure. Appl. Math. 7, pp. 159-193, 1954.
9. Kwon, E., Michalopoulos, P., Xie, H. and Tong, S., "Enhancements of the Kronos simulation package and database for geometric design planning, operations and traffic management in freeway network/corridors, Phase II", Final Report for Minnesota Department of Transportation, December, 1994.

APPENDIX

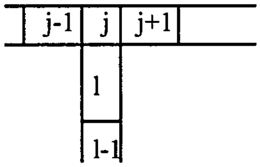
PERFORMANCE ANALYSIS OF THE PROPOSED PARALLEL SIMULATION ALGORITHM WITH PSEUDO CODE

1. General description of the parallel simulation algorithm

```
Set total_cost := 0
Initialize the default q-k curve

Repeat
   Read input cell
   Identify module type of the cell
   Identify current_cost
   total_cost := total_cost + current_cost
   accu_cost[cell_id] := total_cost
Until end_of_data

div_cost := total_cost/# of processors

Search the boundary inside B1 pipeline where accu_cost is approximately div_cost
Broadcast the chosen boundary
Do in parallel
   Identify the first cell and the last cell for parllelization

For cell := first to last, in parallel
   Initialize each cell
EndFor

For time := 1 to current
   Do in parallel
         Send & Receive 1 constant between neighboring processors

   For cell := first to last, in parallel
         Perform the computation according to module type
   EndFor
EndFor
```

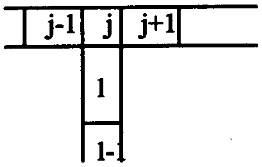
2. Pseudo Code for Basic Freeway Segment Simulation Module

```
        deltax = 100ft
        deltat = 1 sec
        delta = deltat/deltax
        f_j : q-k curve for jth cell


***** B1 ******

-----------------------
  |j-1| j  |j+1|
-----------------------

For time := 1 to current

   k[j,t+1] := 1/2*(k[j-1,t]+k[j+1,t]) + delta/2*(q[j-1,t]-q[j+1,t])
   q[j,t+1] := f_j(k[j,t+1])
   u[j,t+1] := q[j,t+1] / k[j,t+1]

EndFor


**** B2 ****
```



```
Read in alpha

Define a q-k curve q_avail for jth cell
Define a q-k curve q_allow := alpha * q_avail for jth cell
Define a q-k curve q_potential for lth cell

For time := 1 to current

   M := min(q_allow(k[j,t]), q_potential(k[l,t]))
   k[j,t+1] := 1/2*(k[j-1,t]+k[j+1,t]) + delta*(1/2*(q[j-1,t]-q[j+1,t])+M)
   q[j,t+1] := f_j(k[j,t+1])
   u[j,t+1] := q[j,t+1] / k[j,t+1]

   k[l,t+1] := 1/2*(k[l,t]+k[l-1,t]) + delta*(1/2*(q[l,t]+q[l-1,t])-M)
   q[l,t+1] := f_l(k[l,t+1])
   u[l,t+1] := q[l,t+1] / k[l,t+1]

EndFor
```

**** B3 ****



```
Define a q-k curve q_potential for jth cell
Define a q-k curve q_avail for lth cell

For time := 1 to current

   Read in Exit_Demand

   D := min(q_potential(k[j,t]), q_avail(k[l,t]), Exit_Demand)

   k[j,t+1] := 1/2*(k[j-1,t]+k[j+1,t]) + delta*(1/2*(q[j-1,t]-q[j+1,t])-D)
   q[j,t+1] := f_j(k[j,t+1])
   u[j,t+1] := q[j,t+1] / k[j,t+1]

   k[l,t+1] := 1/2*(k[l,t]+k[l-1,t]) + delta*(D-1/2*(q[l,t]+q[l-1,t]))
   q[l,t+1] := f_l(k[l,t+1])
   u[l,t+1] := q[l,t+1] / k[l,t+1]

EndFor
```

**** B4 ****

```
------------------------
  |   |   |u_1 | u_2 |
 |j-1| j  |-----|-----|---
  |   |   |a_1 | a_2 |
------------------------
          j+1  j+2
```

```
Define a q-k curve q_avail for (j+1)th cell

For time := 1 to current

   Read in Exit_Demand

   k[j,t+1] := 1/2*(k[j-1,t]+k[j+1,t]) + delta/2*(q[j-1,t]-q[j+1,t])
   q[j,t+1] := f_j(k[j,t+1])
   u[j,t+1] := q[j,t+1] / k[j,t+1]

   Given q[j+2,t], get k[j+2,t]

   k[j+1,t+1] := 1/2*(k[j,t]+k[j+2,t]) + delta/2*(q[j,t]-q[j+2,t])
   q[j+1,t+1] := f_{j+1}(k[j+1,t+1])
```

```
    q[a_1,t+1] := max(min(Exit_Demand, q_avail(k[j+1,t+1]), q[j+1,t+1]/# of lanes)
    Get k[a_1,t+1]
    u[a_1,t+1] := q[a_1,t+1] / k[a_1,t+1]

    q[u_1,t+1] := q[j+1,t+1] - q[a_1,t+1]
    Get k[u_1,t+1]
    u[u_1,t+1] := q[u_1,t+1] / k[u_1,t+1]

EndFor


**** B5 ****

--------------------------
   |u_{l-1}|u_1 |    |    |
---- ---------|-----|---| j  | j+1|
   |a_{l-1}|a_1 |   |   |
--------------------------
           j-2  j-1

For time := 1 to current

    Given q[j-2,t], get k[j-2,t]

    k[j-1,t+1] := 1/2*(k[j-2,t]+k[j,t]) + delta/2*(q[j-2,t]-q[j,t])
    q[j-1,t+1] := f_{j-1}(k[j-1,t+1])

    q[a_1,t+1] := q[j-1,t+1] / # of lanes
    Get k[a_1,t+1]
    u[a_1,t+1] := q[a_1,t+1] / k[a_1,t+1]

    q[u_1,t+1] := q[j-1,t+1] - q[a_1,t+1]
    Get k[u_1,t+1]
    u[u_1,t+1] := q[u_1,t+1] / k[u_1,t+1]

    k[j,t+1] := 1/2*(k[j-1,t]+k[j+1,t]) + delta/2*(q[j-1,t]-q[j+1,t])
    q[j,t+1] := f_j(k[j,t+1])
    u[j,t+1] := q[j,t+1] / k[j,t+1]

EndFor
```

*****B6******

```
------------------
    |U_1|   |   |
    |    |    |    |
----|---| j  |j+1|
    |L_2|  |  |
----|---|   |  |
    |L_1|  |  |
------------------
    j-1
```

```
For time := 1 to current

   k[j,t+1] := 1/2*(k[j-1,t]+k[j+1,t]) + delta/2*(q[j-1,t]-q[j+1,t])
   q[j,t+1] := f_j(k[j,t+1])
   u[j,t+1] := q[j,t+1] / k[j,t+1]

   Given q[j-2,t], get k[j-2,t]
   k[j-1,t+1] := 1/2*(k[j-2,t]+k[j,t]) + delta/2*(q[j-2,t]-q[j,t])
   q[j-1,t+1] := f_{j-1}(k[j-1,t+1])

   q[L_1,t+1] := q[j-1,t+1] / # of lanes
   q[L_2,t+1] := q[L_1,t+1]
   q[U_1,t+1] := q[j-1,t+1] - (q[L_1,t+1]+q[L_2,t+1])
   Get k[L_1,t+1], k[L_2,t+1] and k[U_1,t+1]

   u[L_1,t+1] := q[L_1,t+1] / k[L_1,t+1]
   u[L_2,t+1] := q[L_2,t+1] / k[L_2,t+1]
   u[U_1,t+1] := q[U_1,t+1] / k[U_1,t+1]

EndFor
```

**** B7 ****

```
-------------------
  |  |   |U_1  |
  |  |   |       |
 |j-1| j |------|-----
  |  |   |L_2  |
  |  |   |------|-----
  |  |   |L_1  |
-------------------
             j+1
```

```
Compute q_capacity for L_1 cell
   q_capacity := q_max / # of lanes

For time := 1 to current

   k[j,t+1] := 1/2*(k[j-1,t]+k[j+1,t]) + delta/2*(q[j-1,t]-q[j+1,t])
   q[j,t+1] := f_j(k[j,t+1])
```

```
    u[j,t+1] := q[j,t+1] / k[j,t+1]

    Given q[j+2,t], get k[j+2,t]
    k[j+1,t+1] := 1/2*(k[j,t]+k[j+2,t]) + delta/2*(q[j,t]-q[j+2,t])
    q[j+1,t+1] := f_{j+1}(k[j+1,t+1])

    Read in Exit_Demand

    q[L_1,t+1] := min(Exit_Demand, q_capacity)
    q[L_2,t+1] := max((Exit_Demand-q[L_1,t+1]), (q[j+1,t+1]-q[L_1,t+1])/(# of lanes-1))
    q[U_1,t+1] := q[j+1,t+1] - q[L_1,t+1] - q[L_2,t+1]
    Get k[L_1,t+1], k[L_2,t+1] and k[U_1,t+1]
    u[L_1,t+1] := q[L_1,t+1] / k[L_1,t+1]
    u[L_2,t+1] := q[L_2,t+1] / k[L_2,t+1]
    u[U_1,t+1] := q[U_1,t+1] / k[U_1,t+1]

EndFor


**** B8 ****

----------------------------
  |a_1|a_2|a_3| ... |a_n|
----------------------------
  |x_1|x_2|x_3| ... |x_n|
----------------------------

Read in alpha

Define a q-k curve q_potential for a_2 cell
Define a q-k curve q_avail for a_3 cell
Define a q-k curve q_allow := alpha * q_avail for a_3 cell

Define a q-k curve q_potential for x_2 cell
Define a q-k curve q_avail for x_3 cell

For time := 1 to current

    Read in Exit_Demand

    M := min(q_allow(k[a_3,t], q_potential(k[x_2,t]))
    D := min(Exit_Demand, q_potential(k[a_2,t]), q_avail(k[x_3,t]))

    k[x_2,t+1] := 1/2*(k[x_1,t]+k[x_2,t]) + delta*(1/2*(q[x_1,t]+q[x_2,t])-M)
    q[x_2,t+1] := f_x_2(k[x_2,t+1])
    u[x_2,t+1] := q[x_2,t+1] / k[x_2,t+1]

    k[x_3,t+1] := 1/2*(k[x_3,t]+k[x_4,t]) + delta*(D-1/2*(q[x_3,t]+q[x_4,t]))
    q[x_3,t+1] := f_x_3(k[x_3,t+1])
    u[x_3,t+1] := q[x_3,t+1] / k[x_3,t+1]

    k[a_2,t+1] := 1/2*(k[a_1,t]+k[a_3,t]) + delta/2*(q[a_1,t]-q[a_3,t])
    q[a_2,t+1] := f_a_2(k[a_2,t+1])
    u[a_2,t+1] := q[a_2,t+1] / k[a_2,t+1]
```

```
   k[a_3,t+1] := 1/2*(k[a_2,t]+k[a_4,t]) + delta*(1/2*(q[a_2,t]-q[a_4,t])+(M-D))
   q[a_3,t+1] := f_a_3(k[a_3,t+1])
   u[a_3,t+1] := q[a_3,t+1] / k[a_3,t+1]

EndFor


*****B9******

|-------------
| d | 0 | 1 |
|-------------

Define a q-k curve q_I for a dummy cell d

Initialize k[d,0]

For time := 1 to current

   Read in Ent_Demand

   k[0,t+1] := 1/2*(k[0,t]+k[1,t]) + delta*(q_I(k[d,t]-1/2*(q[0,t]+q[1,t]))
   q[0,t+1] := f_0(k[0,t+1])
   u[0,t+1] := q[0,t+1] / k[0,t+1]

   k[d,t+1] := k[d,t] + delta*(Ent_Demand-q_I(k[d,t]))

EndFor


**** B10 ****

-----------|
      |l-1| l |
-----------|

Option 1:   For time := 1 to current

   k[l,t+1] := 1/2*(k[l-1,t]+k[l,t]) + delta/2*(q[l-1,t]-q[l,t])
   q[l,t+1] := f_l(k[l,t+1])
   u[l,t+1] := q[l,t+1] / k[l,t+1]

EndFor

  Option 2 :   For time := 1 to current

   Read in Exit_Demand

   k[l,t+1] := k[l,t] + delta*(1/2*(q[l-1,t]+q[l,t])-Exit_Demand)
   q[l,t+1] := f_l(k[l,t+1])
   u[l,t+1] := q[l,t+1] / k[l,t+1]

EndFor
```

3. Running Time Analysis

The following running time analysis was performed with a Pentium 90 Mhz Personal Computer from Blue Star.

3.1 Worst Case Running Time Analysis

The following is the worst case computational complexity for each module type.

B1 : 12.6cn (5+ 5* 1/)n
B2 : 40.2cn (16+ 19* 2/)n
B3 : 39.2cn (16+ 18* 2/)n
B4 : 99.9cn (28+ 30* 10/ 3sqrt)n
B5 : 94.9cn (26+ 27* 10/ 3sqrt)n
B6 : 119cn (32+ 32* 13/ 4sqrt)n
B7 : 124.6cn (35+ 32* 14/ 4sqrt)n
B8 : 80.4cn (33+ 37* 4/)n
B9 : 22.6cn (10+ 10* 1/)n
B10 : 13.6cn(option 1) or 12.6cn(option 2)

where,

1. n : # of time step
2. c : execution time for 1 addition
   + : - : * : / : sqrt = c : c : c : 2.6c : 5.3c

3.2 Best Case Running Time Analysis

The following is the best case computational complexity for each module type.

B1 : (3+ 3* 1/)n
B2 : (8+ 9* 2/)n
B3 : (8+ 8* 2/)n
B4 : (10+ 8* 4/)n
B5 : (10+ 8* 4/)n
B6 : (12+ 8* 5/)n
B7 : (15+ 8* 6/)n
B8 : (17+ 17* 4/)n
B9 : (6+ 5* 1/)n

4. Initialization Cost for Each Module Type

Initialization for each cell j involves 3 steps.
1. Get the parameter q_max
2. Modify the default q-k curve and obtain new coefficients
3. Given q[j,0], compute k[j,0].

The running time analysis was done based on the existing codes.
1. 0
2. 56.6c
3. 19.5c

Therefore, the initialization cost for each cell is 76.1c

B1 : 76.1c
B2 : 76.1c * 2
B3 : 76.1c * 2
B4 : 76.1c * 3
B5 : 76.1c * 3
B6 : 76.1c * 4
B7 : 76.1c * 4
B8 : 76.1c * 2
B9 : 76.1c (will change if k_d needs to be computed)
B10 : 76.1c

1. B2(B3,B8) involves 2 cells, so multiply 2
   B4(B5) involves 3 cells, so multiply 3
   B6(B7) involves 4 cells, so multiply 4

2. c : execution time for 1 addition
   + : - : * : / : sqrt = c : c : c : 2.6c : 5.3c

3. Note : The costs could change depending on how the code is implemented

5. Performance Analysis of the Proposed Parallel Simulation Algorithm with the Example Freeway Section

Peformance Indices

Speedup, S : ratio of the sequential execution time on a uniprocessor to the parallel execution time on a parallel computer with p processors

1. measure of the relative benefit of solving a problem in parallel
2. theoretically, S cannot be > p

Efficiency, E : ratio of speedup to the number of processors

1. measure of the fraction of the time a processor is usefully employed
2. theoretically, E cannot be > 1

Parallel Cost : the product of parallel execution time and the number of processors used

=> reflects the sum total of the time spent by all the processors in solving the problem.

Cost Optimal : A parallel algorithm is Cost Optimal if the parallel cost is proportional to the execution time of the best known serial algorithm on a single processor.

Notations
----

p : number of processors
c : time cost(execution time) for 1 addition
t_s : startup time for communication
t_w : time to send/receive 1 constant
T_s : sequential execution time
T_p : parallel execution time

The ratio of execution time among +,-,*,/,sqrt (for the 90 Mhz Pentium processor):
+ : - : * : / : sqrt = c : c : c : 2.6c : 5.3c

Example Peformance Analysis with the test freeway section

In this example, the parallel architecture is linear array with p processors.

The given example freeway can be decomposed as follows:

Segment Type	Total number of segments
B1	1042
B2	7
B3	7
B4	25
B5	25
B6	0
B7	0
B8	18
B9	19
B10	19

< On a Uniprocessor >

Therefore, the sequential execution time on a single processor is

T_s = 20690cn + 98473.4c

< On 3 processors >

	I	II	III
B1	311	376	355
B2	3	1	3
B3	1	3	3
B4	10	7	8
B5	9	8	8
B6	0	0	0
B7	0	0	0
B8	10	4	4
B9	9	5	5
B10	7	6	6

Computational cost	7034.1cn	6870.1cn	6785.8cn
Initializaion cost	31353.2c	34092.8c	33027.4c
Communication cost		t_s+2t_w	t_s+2t_w

Therefore, the parallel execution time is

T_p = 7034.1cn + 33928.8c + t_s+2t_w
(7034.1cn + 34092.8c + t_s+2t_w)

This algorithm is one of the best because

1. S is near p (in this example, p=3)
   S = T_s/T_p ---> 2.94 as n gets large(goes to infinity)

2. E is near 1
   E = S/p ---> 0.98 as n gets large(goes to infinity)

3. Cost optimal because pT_p = 3 * (7034.1cn + 33928.8c + t_s+2t_w)
   is proportional to T_s