A Distributed Computing Environment For Dynamic Traffic Operations

Abstract: *This study explores distributed computing techniques in the context of dynamic network algorithms for Intelligent Transportation Systems (ITS) applications. ITS technologies such as Advanced Traveler Information Systems (ATIS) and Advanced Traffic Management Systems (ATMS) provide traffic-related information on-line for a new generation of models and methodologies that aim at the real-time enhancement of network performance and efficiency. However, existing sequential processing solution techniques and/or single processor computational environments preclude translation of these methodological and algorithmic advances to on-line operability due to the severe computational burden of processing network-wide time-dependent traffic-related data (such as volume, speed, occupancy, and classification). To address this problem, we investigate two Remote Procedure Call (RPC) based distributed computing techniques on a network of workstations. Computational results indicate that the distributed implementation performs better than sequential computation, though trade-offs between communication overheads and computational savings become critical with the number of processors. This suggests a threshold number of processors for optimal computational performance depending on the specific problem and its solution logic. The results also illustrate the effects of congestion levels, loading profiles, and output data size on computational performance.*

1 INTRODUCTION

Severe traffic congestion in recent years has led to costly delays, lost productivity, and harmful side effects such as reduced safety, increased pollution and wasteful fuel consumption. Traditional traffic engineering approaches to reduce congestion are proving unequal to the growing mobility challenge. Investment in new infrastructure is not a viable solution because of the prohibitively high economic costs, lack of space in urban areas, as well as social and environmental concerns associated with disruption of neighborhoods. In recent years, evolving technologies known as Intelligent Transportation Systems (ITS), which integrate advances in telecommunications, information systems, automation, and electronics to enhance the efficiency of existing road networks, are being proposed to address the problem.

ITS techniques are currently addressing several operational problems that require on-line solutions, such as network-wide optimal routing information provision to users under Advanced Traveler Information Systems (ATIS), and traffic management during incidents using advanced sensor systems under Advanced Traffic Management Systems (ATMS). These on-line approaches are motivating a new realm of control strategies and opportunities to influence the dynamic traffic conditions in congested networks. However, the associated solution procedures require the processing of network-wide time-dependent information in real-time. Existing sequential processing solution techniques and/or single processor computational environments do not meet the intensive computational requirements for on-line data processing. Although dedicated high performance computing systems (for example,

CRAY and Connection Machines) can significantly reduce the computational time, most cities and transportation agencies cannot afford them. The most optimistic and affordable computational environment in the near future for most traffic control centers consists of a network of personal computers or workstations. In such an environment, it is more realistic to add relatively inexpensive new personal computers or workstations into the network to relieve the increased computation burden on the existing system than to fundamentally overhaul the computational environment with expensive dedicated multiple-processor machines. Even if affordable small scale parallel computers or multiple-processor machines were available, they represent a more restrictive environment than a distributed network of computers vis-à-vis future computational needs of the traffic control center. These needs can range from the increased network size and/or travel demand to the processing of additional large time-dependent data sets when incorporating additional functional capabilities in the traffic control center. In such situations, parallel computers may need upgradation or replacement with substantially more expensive and powerful machines whereas a distributed environment logic can be retained with incremental expansion. The proposed distributed environment is also synergistic with the current understanding on ITS strategies for on-line traffic guidance operations which focus on reactive distributed schemes[8,16] to circumvent the significant computational burden. These issues motivate our investigation of ITS related models and algorithms using distributed computing techniques on a network of workstations.

This paper focuses on the analysis of the computational performance of ITS related models and algorithms using two distributed computing approaches known as the multitasking and asynchronized RPC techniques, by exploring the trade-offs between computational gains and communication overheads. Our distributed computing environment consists of a network of SUN SPARC S5/70 workstations operating under SOLARIS 2.3 and connected by a 10 Mbps local area network (LAN) within a client-server framework. The RPC programming mechanism[2,15] and a RPC implemented client-server model are adopted. The overall logic of the RPC distributed mechanism is controlled by one workstation, designated as the client, and the computations are distributed via the LAN to the other workstations, called servers. The multitasking and asynchronized RPC mechanisms are compared by developing distributed codes for small parts of a traffic simulator for ITS applications, DYNASMART[10,14]. An asynchronized RPC version of a dynamic traffic assignment algorithm, MUCTDTA[17,19] is developed, and its computational performance under alternative congestion levels, loading profiles, and output data sizes, is investigated.

The paper is organized as follows. Section 2 reviews literature on high performance computing environments and some prior applications to transportation engineering. Section 3 describes the distributed computing techniques and associated performance measures used in this study. Section 4 compares the computational performance of the multitasking and asynchronized RPC approaches using DYNASMART. Section 5 discusses the application of the asynchronized technique to the MUCTDTA algorithm and simulation experiments to analyze its computational performance under alternative congestion levels and loading profiles. Concluding comments are presented in Section 6.

2 BACKGROUND REVIEW

Most existing traffic simulation software tools such as NETSIM[13], CORSIM[7], and DYNASMART use sequential processing. However, such environments preclude the exploitation of recent advances in computational hardware and software, and may provide inefficient implementations and severely slow down problem solving, especially for large scale problems with intensive data needs. Also, a sequential environment prevents the exploitation of parallelisms inherent to the problem or the solution logic. Recent technological advances have motivated the development of high performance computing environments[1,12,21,22] and the investigation of their potential and limitations[5]. There are two types of high performance computing environments: parallel and distributed computing. Parallel computing systems consist of processors located in close proximity and aim at executing a computational task jointly by exploiting concurrent events in the computational process[9]. A customized architecture with central coordination and control ensures very efficient, reliable, and predictable interprocessor communication. However, their special architectures, with intelligent memory organization and pipelining, make these supercomputers (for example, CRAY-YMP and CM-5) prohibitively expensive. Hence, most cities and the transportation agencies cannot afford to buy them. Distributed computing implies a collection of geographically separated computers or nodes that cooperate and communicate with each other in addressing a task. Occasionally, the term is applied to an operation that involves multiple minicomputers, workstations, or PCs connected by a computer network. To execute sequential algorithms in a distributed environment, the original programs require modifications so that different tasks can be allocated to different processors while ensuring the original sequential logic. A distributed framework is attractive because a significant enhancement of computational power could potentially be achieved on a pre-existing network of PCs or workstations without substantial additional monetary investment.

Till recently, models and solution methodologies developed in the traditional transportation arena have focused mostly on the algorithmic logic. Little attention has been paid to the computational efficiency of algorithms and/or their implementations, or to the computational environment. This is because for most problems addressed in the past, computational efficiency is not a primary concern and/or off-line solution methodologies suffice. Additionally, the field of high performance computing is itself in its embryonic stages[12]. The advent of ITS and the integration of advances in computers, automation, electronics, telecommunications, and information processing systems in the transportation area have led to a new generation of models and algorithms that seek to improve traffic flow performance on-line. They include such diverse strategies as real-time route guidance, dynamic congestion pricing, incident detection and route diversion, dynamic ramp metering, and real-time network-wide adaptive traffic control. From a computational perspective, these strategies have two key characteristics: (i) they require on-line implementation, and (ii) they involve the processing of large amounts of on-line data.

Chang et al.[3,4] implemented a real-time network traffic simulation model for ATMS applications on a massively parallel computing system, the Connection Machine CM-2. The model is parallelized for application to large scale networks by developing parallel data structures whose elements can be processed simultaneously. Three parallel variables are considered including link, node, and vehicle data which are shaped as one-dimensional parallel

variables. Habbal et al.[6] implemented a parallel algorithm for solving a shortest path problem on the CM-2 computer.

Mahmassani et al.[14] developed traffic assignment and simulation procedures for ATIS/ATMS applications and implemented them on the CRAY-YMP supercomputer. Ziliaskopoulos and Mahmassani[23] developed time-dependent shortest path algorithms for real-time ITS applications and implemented them on the CRAY-YMP. These procedures use CRAY's vectorization capabilities to gain computational time savings. Another model implemented on the CRAY-YMP is the DYNASMART (Dynamic Network Assignment-Simulation Model for Advanced Road Telematics) simulation model, designed to simulate the traffic flow patterns resulting from time-varying traffic demands, and evaluate the overall network performance under ITS. It uses the macrotasking, microtasking, and autotasking capabilities of CRAY to gain computational time savings. However, DYNASMART is a sequential model and the use of special purpose hardware features of CRAY to gain speedup restricts its portability to other computing platforms.

3 METHODOLOGY

3.1 Distributed Computing Techniques

Distributed computing systems predominantly use the client-server model. Under this model, one processor called the client coordinates the other processors in the system, called servers, to function as a single computational unit. The client (the host workstation) distributes the tasks to the servers (the remote workstations). Coordination is executed through the communication of messages from the client to the servers. The servers reply the results of the computations to the client, which exerts the overall control of executing the algorithmic code. A client-server model implemented by RPC is used to design our distributed computing environment. Two communication mechanisms using RPC are used to implement our distributed environment: (i) multitasking RPC approach, and (ii) asynchronized RPC approach.

3.1.1 RPC (Remote Procedure Call)

A distributed computer system can be viewed as a set of software components running on a number of computers in a network and RPC is a method for interprocessor communication over the network. Users interact with the application programs which play the role of clients who may need any of the services available in the network. The service programs, or the servers, may themselves be clients of other service programs.

3.1.2 Multitasking RPC

A multitasking RPC (MRPC) approach generates several processes for concurrent serving. It invokes the *fork* system call through a *parent* process, and creates new processes called the *child* processes. Every process has only one parent process, but a process can have many child processes. A parent process can continuously create other new child processes. Fig. 1 shows the multitasking RPC logic. After the successful fork system call, each child

process makes a remote procedure call to distribute the desired part of the routine to a remote workstation for execution. When the remote workstations finish computations, they write the results into files to the network file server, which are then accessed and combined, if necessary, by the client. From a computational perspective, there are several communication overheads associated with the MRPC approach. The creation of each process adds some communication overhead. All child processes issue remote procedure calls to the corresponding remote workstations and wait for them to finish the computations. After a remote workstation acknowledges the completion of the task, the associated child process terminates. The fork system call has significant overheads because each child process needs to make a copy of the parent's environment.

3.1.3 Asynchronized RPC

RPC is a synchronous language level transfer of control between programs in disjoint addresses. The synchronous nature of RPC makes it difficult to take advantage of the parallelisms inherent in a distributed environment. We implement an asynchronous approach under the limit of RPC synchronous mechanism that allows execution to proceed locally in parallel with remote execution. Fig. 2 illustrates our asynchronized RPC (ARPC) approach as a three-step process. When the client issues a request, a server procedure is called. The server first attempts to retrieve the required calling parameters from the request packet. When it succeeds, it immediately acknowledges the call before executing its instructions. The client is now free to issue a request to another remote workstation so that concurrent execution can be performed using RPC. Each server writes its results into files which the client can access and combine together, if necessary, at the end of the distribution process.

From a computational perspective, the ARPC approach circumvents the need for processes which are required in the MRPC approach to perform concurrent computations. Hence, the MRPC approach has significant additional communication overheads compared to the ARPC approach as the number of processors increase.

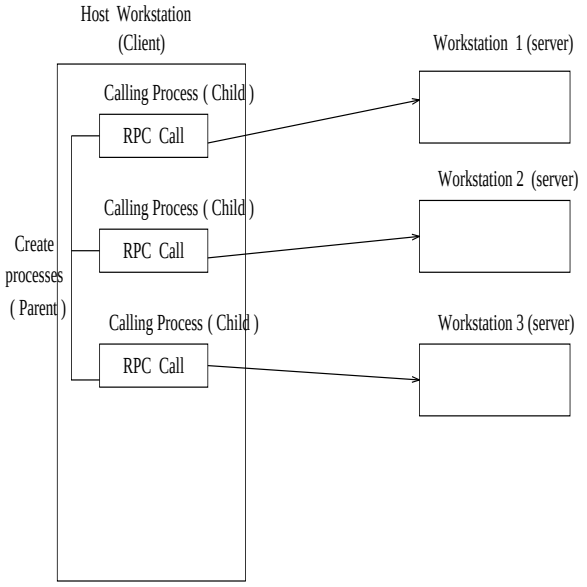

Fig. 1. Multitasking RPC System Structure.

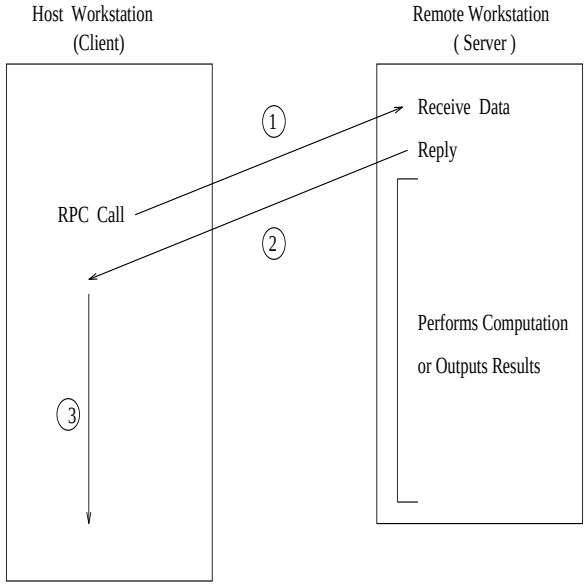

Fig. 2. Asynchronized RPC System Structure.

3.2 Performance measures

In this section, we define speedup and efficiency, which have been used as abstract measures of computational performance for over twenty years[11]. Both these measures depend on a direct indicator of system performance, the total CPU time, and are defined as follows:

1. **Total CPU time**: The total CPU time is the total time spent in solving the specified problem. Denoted by T(n) for a sequential processor, where n is the data size of the problem, it consists of two components: (i) the total computation time, $T_{cp}(n)$, and (ii) the total communication time, $T_{cm}(n)$. Hence, T(n) = $T_{cp}(n)$ + $T_{cm}(n)$.

- **Total computation time,** $T_{cp}(n)$: is the amount of time attributed just to computation. It can be viewed as the time required to solve the problem if all communication were instantaneous.

- **Total communication time** $T_{cm}(n)$: is the time spent on communication overheads or delays. In many parallel/distributed algorithms and systems, the time spent on interprocessor communication can be a substantial fraction of the total time needed to solve the problem. To analyze communication issues, it is helpful to view the distributed computing system as a network of processors connected by communication links. Each processor uses its own local memory for sharing some problem data and intermediate algorithm results, and exchanges information with other processors via communication links. The total communication time in our study consists of time spent on sending/receiving data and creating the associated processes to enable communication.

2. **Speedup**: indicates how much faster a problem will be solved with a parallel/distributed system as opposed to a single processor. If $T^*(n)$ denotes the best sequential processor total CPU time known for solving the problem and $T_P(n)$ is the total CPU time to solve the same problem under distributed computing with P processors, the speedup $S_P(n)$, is defined as the ratio:

$$S_P(n) = \frac{T^*(n)}{T_P(n)}$$

Ideally, $S_P(n)$ = P. However, this ideal bound is practically unattainable because a number of architectural, algorithmic, and software issues intrude before the bound is reached. For example, some programs have some sections that are easily parallelizable and other sections that are inherently sequential, limiting the speedup (Amdahl's law). Also, for the parallel/distributed system, system-level architectural constraints such as memory capabilities lead to communication overheads. In practice, it is difficult to determine the best sequential CPU time for even seemingly simple computational problems. Hence, the commonly used proxy for speedup is the practical speedup which is defined as the ratio of the sequential time complexity of a problem and its parallel/distributed time complexity. We use speedup and practical speedup interchangeably in our experiments. Kuck[12] presents a comprehensive discussion on the issues that limit the practical speedup. He specifies acceptable performance levels and ranges for speedup based on the current state of technology, as illustrated in Table 1. These performance measures and

Table 1. Performance Levels and Ranges for Speedup[12]

($logP \equiv log_2 P$)

Name	Speedup
High Performance Range	$\frac{P}{2} \leq S_P \leq$ P
Minimum High-Performance Level	$S_P = \frac{P}{2}$
Intermediate Performance Range	$\frac{P}{2logP} \leq S_P < \frac{P}{2}$
Threshold Performance Level	$S_P = \frac{P}{2logP}$
Unacceptable Performance Range	$1 \leq S_P \leq \frac{P}{2logP}$

threshold values reflect the current understanding in the high performance computing field and may change with future progress.

3. **Efficiency**: measures the fraction of time that a typical processor is usefully employed. It represents the percentage of busy time spans of a typical processor employed in solving the problem vis-à-vis the total time spans. Denoted by $E_P(n)$, it is the ratio

$$E_P(n) = \frac{S_P(n)}{P} = \frac{T^*(n)}{PT_P(n)}$$

Ideally, $E_P(n)$=1, implying a speedup of P with P processors. For this to occur, no processor should remain idle or perform non-related work. This ideal situation is practically unattainable. A more realistic objective is to aim at an efficiency that stays bounded away from zero, as n and P increase.

4 ANALYSIS OF THE MPRC AND ARPC TECHNIQUES

This section analyzes the MRPC and ARPC distribution techniques. Two small loops in one of the routines of DYNASMART are used to analyze their computational performance. Three experiments are performed. The first experiment analyzes the performance of ARPC and MRPC approaches with the number of processors. The second examines the performance of the ARPC approach with data size. The last experiment compares the performance of the sequential and distributed cases with the data size and number of processors.

4.1 The DYNASMART Model

DYNASMART is a fixed time step mesoscopic simulation model designed to assign time-varying traffic demand and model the corresponding flow pattern and evaluate the overall performance of traffic networks under ITS technologies such as ATIS/ATMS. The modeling approach integrates a traffic flow simulator, a network path processing component, user behavior rules, and information strategies[14].

DYNASMART consists of a main program that controls the overall logic of the model, and several associated subroutines. The particle movement (Partco) subroutine, which models vehicle movements on links and nodes, is

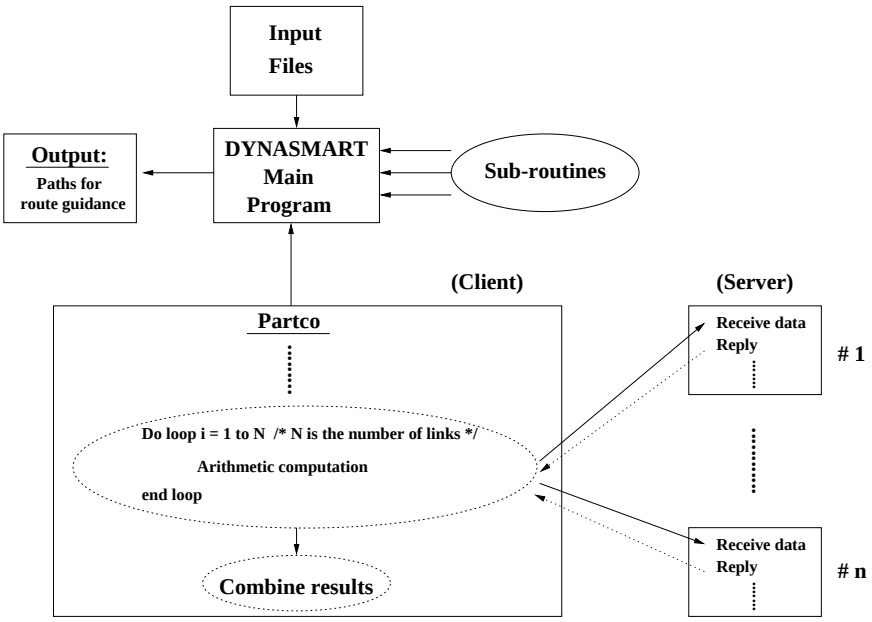

Fig. 3. A control diagram for the distributed code.

used to compare the MRPC and ARPC approaches. Among the DYNASMART routines, Partco ranks first in the total CPU time per call. Two small loops in the Partco routine, which perform computations at the link level, are distributed as illustrated in Fig. 3. The two loops are chosen for their generic nature. Both represent arithmetic computations at the link level in our experiments. However, in the general context of dynamic transportation algorithms, they are representative of the commonly encountered intensive computational needs, especially for large networks (with large numbers of links, nodes, vehicles, etc.). Hence, the corresponding insights are expected to have implications for a broader spectrum of dynamic transportation methodologies. The larger loop, designated Loop 1, performs a few arithmetic computations for all links. The other one, called Loop 2, performs an arithmetic computation for all links. In the distributed environment, these computations are distributed over the various servers.

4.2 Experimental Setup

4.2.1 Experimental Parameters

The following parameters are used in the various experiments:

- number of processors: as stated earlier, the distributed computing system for the study consists of a set of UNIX-based SUN SPARC workstations (S5/70) connected by a local area network. The maximum number of processors used is nine (based on the available computational facilities).
- congestion levels: different traffic congestion levels are achieved using different network loads. The network loading factor is defined as the ratio of the total number of vehicles generated in the network during the 35-minute demand generation horizon of interest compared to a base number. In our experiments, this

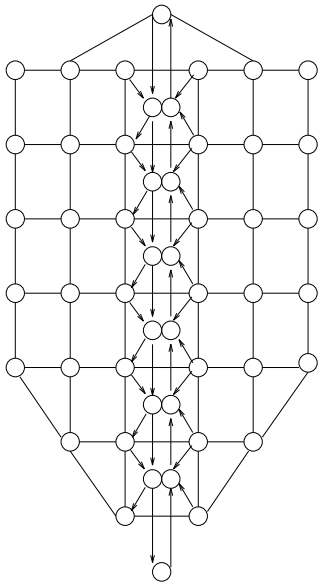

Fig. 4. Network Structure.

base number is 9554 vehicles generated over a 35-minute period, which represents a loading factor of 1.0. Thereby, a loading factor 2.0 generates approximately twice the number of vehicles as loading factor 1.0. In all experiments, the time-dependent loading profile of the O-D flows is conserved. The congestion level is a proxy for the data size as the computations depend on the number of vehicles in the network.

- distribution techniques: the ARPC and MRPC techniques are used to implement the proposed distributed environment.
- number of output files: the maximum number of time-dependent link level output files in the Partco routine used in the study is four. Hence, the number of output files distributed range from 1 to 4. These output files are used to store the intermediate results.
- size of loop distributed: two loops in the Partco routine are distributed. The larger loop is Loop 1 and the smaller one is Loop 2.

4.2.2 Test Network

The test network, illustrated in Fig. 4, consists of a freeway with a grid street network on both sides. The network consists of 168 links and 50 nodes, and has 32 origins and 10 destinations. All arcs are two-directional (and are actually modeled as two directed arcs) and have two lanes in each direction, except the entrance and exit ramps that connect the street network to the freeway; these are directed arcs with one lane as shown in the figure. The free flow speed on the freeway links is 88 kmph and 48 kmph for all other links. Each link is 0.4 kilometers long. In terms of signal control, there are 25 nodes with pre-timed control, 8 with actuated control; the remaining have no form of signal control.

4.3 Description of Experiments

The first experiment analyzes the performance of the MRPC and ARPC approaches with the number of processors. The loading factor is maintained constant at 2.2. The multitasking approach is used for Loop 1 only while the asynchronized approach is used for both loops.

The second and third experiments are performed using the asynchronized approach only. The second experiment examines the performance of the ARPC approach with data size, represented by the network congestion level. As discussed earlier, a higher loading factor reflects greater congestion, and hence, a larger data size (in terms of the number of vehicles and associated parameters). The total CPU time is compared for the sequential and distributed cases as the loading factor increases. The number of output files and processors is fixed at four.

In the third experiment, output files instead of the loops in the routine are distributed to workstations. The number of processors is equal to the number of output files. The total CPU times for the sequential and distributed cases are compared with the number of output files. The network congestion level is kept constant at loading factor 1.0.

4.4 Results

4.4.1 Multitasking RPC Results

In the multitasking approach, Loop 1 in the Partco routine is distributed by invoking the *fork* system calls associated with the RPC mechanism. Fig. 5 depicts the total CPU and communication times of the Partco routine with the number of processors. The value 0 on the x-axis indicates the sequential case. As illustrated in the figure, the total CPU time and the communication time increase with the number of processors. The total CPU time increase is caused almost entirely by additional communication overheads as indicated by the parallel plots beyond two processors. This is because the number of processes created by the fork system call to distribute computations and the number of RPC calls invoked to communicate with remote processors are equal to the number of processors. Hence, the MRPC technique can perform worse than the sequential code in a distributed environment, primarily due to communication overheads resulting from creation of processes by the fork system call. It is important to note that Loop 1 represents a small fraction of the Partco routine computation, and better trade-offs may exist when significant fractions of the computation are distributed.

4.4.2 Asynchronized RPC Results

The results of the asynchronized RPC experiments are illustrated in Figs. 6 through 12.

- Number of processors

Fig. 6 illustrates the total CPU and communication times for Loops 1 and 2 with the number of processors. The total CPU time decreases in the distributed environment for Loops 1 and 2 though there is no significant CPU performance gain by using a large number of processors. As stated earlier, this is because Loops 1 and 2 represent very small fractions of the Partco computation time. While the communication time increases with the number

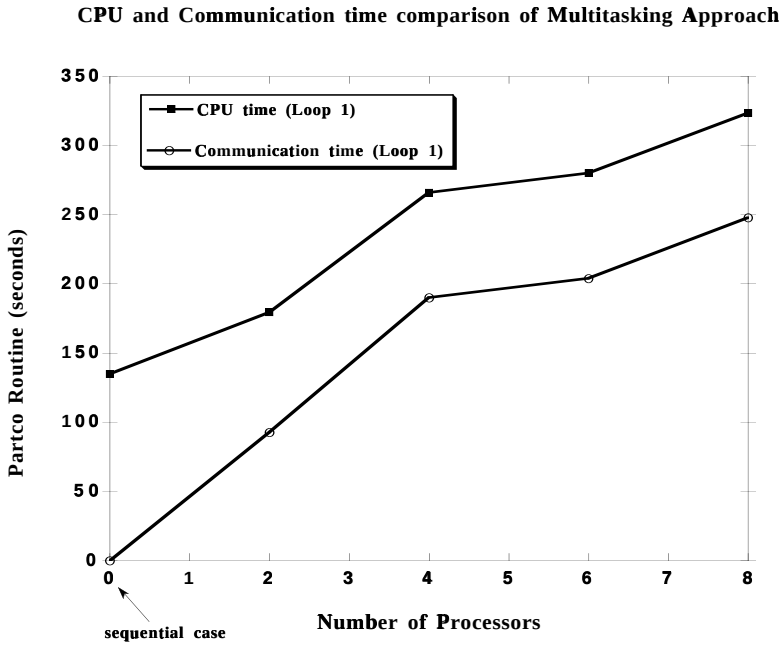

Fig. 5 Multitasking total CPU time and communication time for Loop 1.

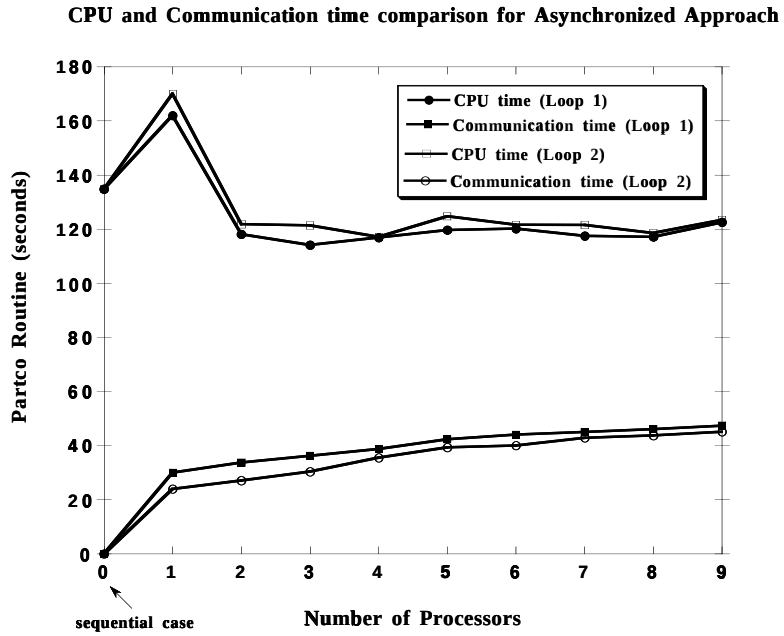

Fig. 6. Comparison of Loop 1 and Loop 2 Performance.

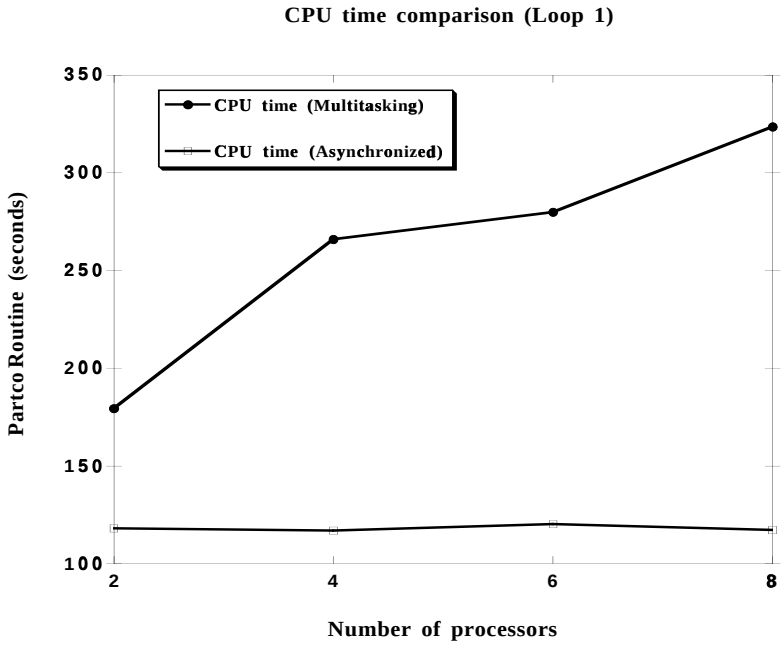

Fig. 7. Comparison of total CPU times of Multitasking and Asynchronized Approaches.

of processors, the ARPC approach performs substantially better than the MRPC technique as illustrated by Figs. 7, 8, and 9. Under the ARPC technique, the communication time increases only marginally with the number of processors, indicating its potential advantages in a distributed environment. This motivates the use of the ARPC approach in Section 5 to study a large scale problem, the distributed version of the dynamic traffic assignment (MUCTDTA) algorithm.

The degradation of performance due to communication overheads under both techniques is further emphasized by comparing the performance on a single workstation in Table 2, where about 15-25 percent overheads are introduced just due to the communication aspects. Also, the communication overheads are lower for the ARPC approach compared to the MRPC approach.

The total CPU time for the bigger loop (Loop 1) is less than the total CPU time for the smaller loop (Loop 2) even though the communication time is slightly higher for Loop 1 (Fig. 6). While neither Loop 1 nor Loop 2 is large enough to warrant significant overall (CPU) savings in a distributed environment, distributing larger loops may result in greater computational efficiency vis-à-vis the communication overheads.

A general conclusion from these results is that a distributed environment provides the best overall computational savings at a threshold number of processors when the tradeoffs between the computational time savings and the communication overheads are highest. For example, the best CPU time for the Loop 1 case is obtained when we use three processors (Fig. 6). It should be noted that this number is problem specific and dependent on the particular computational environment.

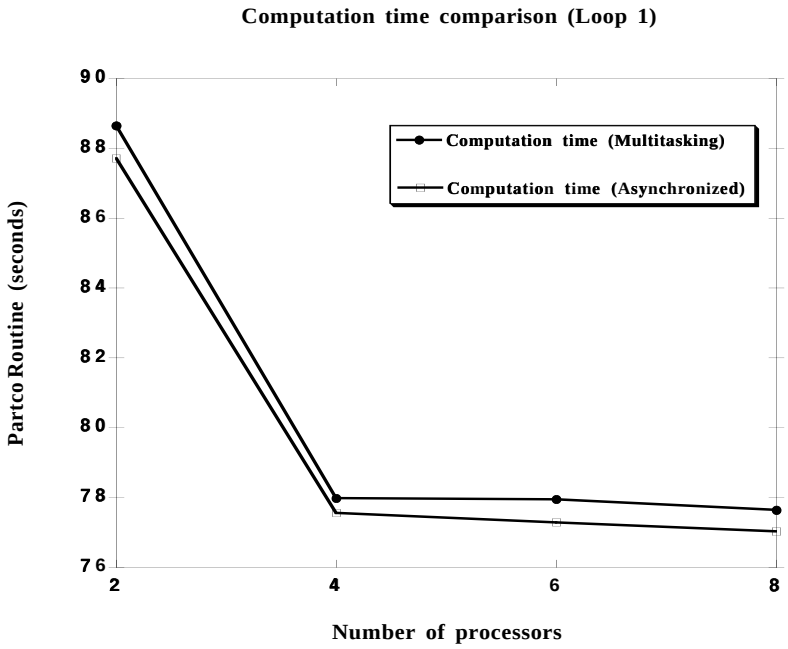

Fig. 8. Comparison of computation times of Multitasking and Asynchronized Approaches.

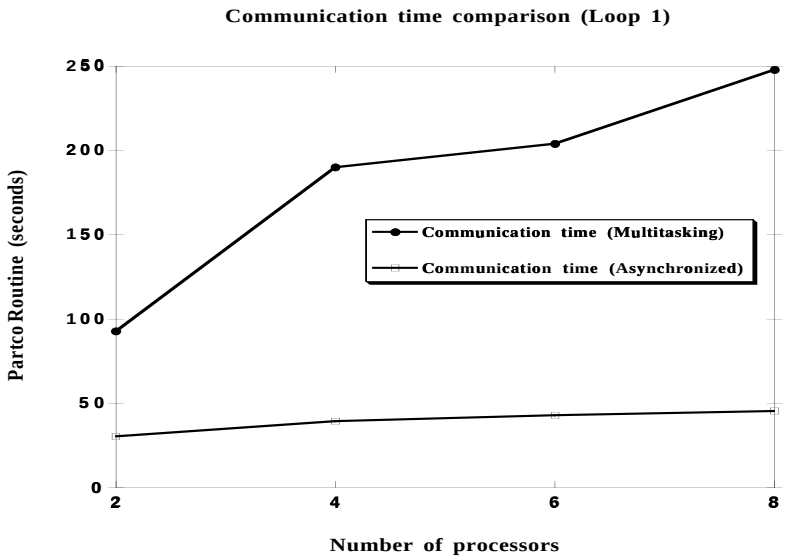

Fig. 9. Comparison of communication times of Multitasking and Asynchronized Approaches.

Table 2. Partco routine CPU time on one workstation

Sequential(secs)	MRPC(secs)	ARPC(secs)
134.86	168.35	153.03

• Data size (congestion level)

Fig. 10 highlights the CPU savings obtained in a distributed environment as the network load increases. The percentage savings over the sequential Partco routine increases from about 12% at loading factor 3.0 to about 25% for a factor 4.0, indicating greater savings with more intensive computations. The corresponding speedup and efficiency increase with loading factor (Fig. 11). The speedup varies from 1.22 to 1.32, placing it in the intermediate performance range (Table 1). This performance is obtained despite distributing a very small fraction of the algorithmic code.

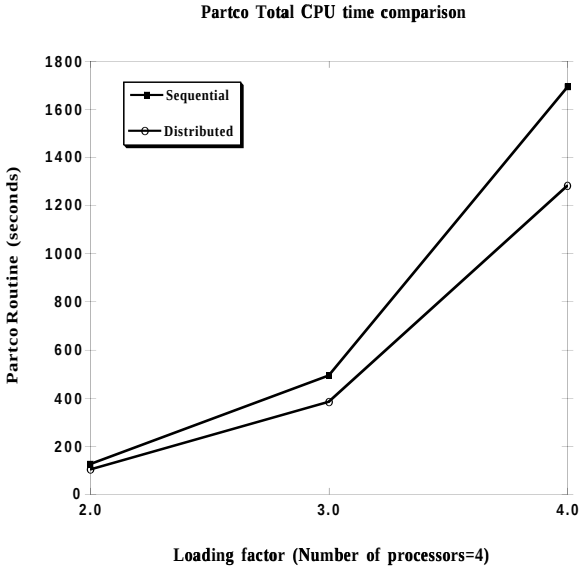

Fig. 10. Comparison of sequential and distributed performance.

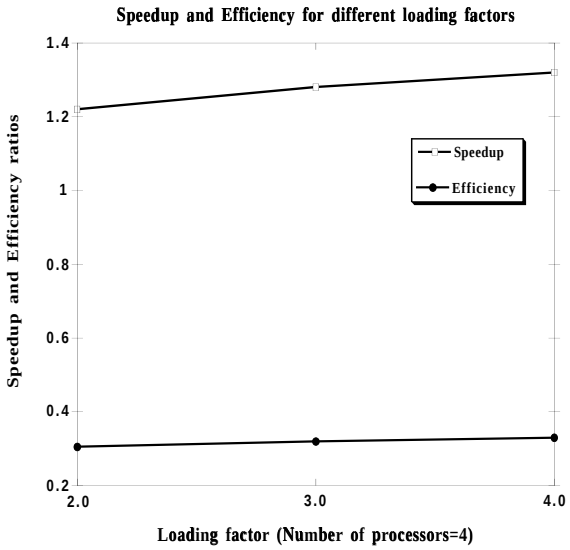

Fig. 11. Speedup and Efficiency.

• Number of output files

Distributing output files can be beneficial, as illustrated by Fig. 12, indicating that it may represent a viable alternative strategy to distributing the code for problems with large data processing needs.

5 APPLICATION OF THE ARPC TECHNIQUE TO A DYNAMIC TRAFFIC ASSIGNMENT ALGORITHM

The previous section highlighted the advantages of the ARPC distributing approach over the MRPC approach. Also, the distribution of very small parts of the DYNASMART code (Loop 1 and Loop 2) leads to speedups in the intermediate performance level range. The above motivate the application of the ARPC approach to a large scale optimization solution algorithm with on-line applications; the Multiple User Classes Time-Dependent Traffic Assignment (MUCTDTA) algorithm, developed for providing optimal routes to users on-line, for traffic networks with ATIS/ATMS capabilities[17,19]. Given a set of time-dependent origin-destination vehicle trip desires for the horizon of interest, the MUCTDTA algorithm determines a time-dependent assignment of vehicles to network paths and corresponding arcs so as to achieve some systemwide objectives and/or satisfy certain conditions for the

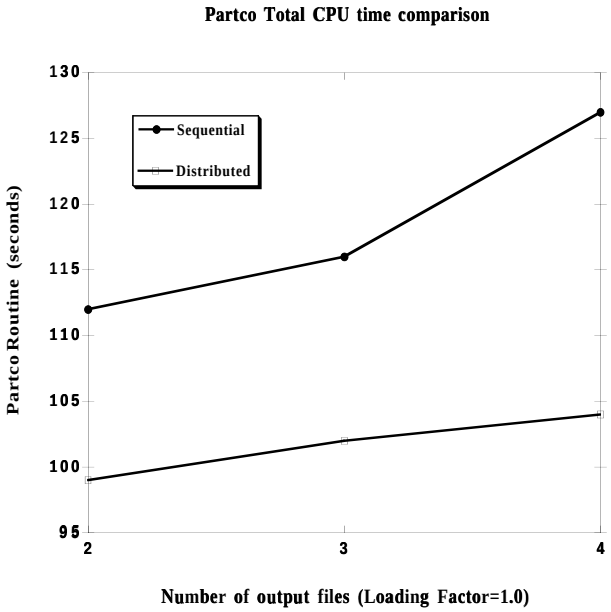

Fig. 12. Computational performance with number of output files.

user classes. Four user classes are considered based on information availability, information supply strategy, and response behavior[17,18]. They are the system optimal (SO), user equilibrium (UE), boundedly rational (BR), and pre-specified (PS) classes. In our experiments, each user class is assumed to constitute twenty-five percent of the total network load.

Fig. 13 depicts the simulation-based iterative solution algorithm for the MUCTDTA problem. It consists of an inner loop that incorporates a direction finding mechanism for the system optimal (SO) and user equilibrium (UE) classes based on the simulation results of the current iteration (the experienced trip times and the time-dependent link vehicle numbers). The boundedly-rational (BR) user class is not directly involved in the search process. The paths of this user class are obtained based on the traffic pattern that evolves in the network in real time and can change en-route, unlike the SO and UE paths which are obtained based on search directions from the experience of previous iterations. Therefore, there is no direct guiding mechanism involved in obtaining the BR paths, other than their being predicated on the assignment strategy for the SO and UE class vehicles. They form the outer loop of the iterative procedure as illustrated in Fig. 13. The unequipped users (PS class) are exogenous to the search and represent constant background information for each iteration as their paths are pre-specified and remain unchanged.

The two sets of programs inside the dashed line boxes (Fig. 13), denoted as SO and UE programs, are distributed and executed on one remote workstation each. Akin to the experiments discussed in Section 4, loops within the path processing algorithms or the update algorithms of the SO and UE programs could also be distributed, in addition to these programs themselves. However, this was not done in the experiments discussed here. A host

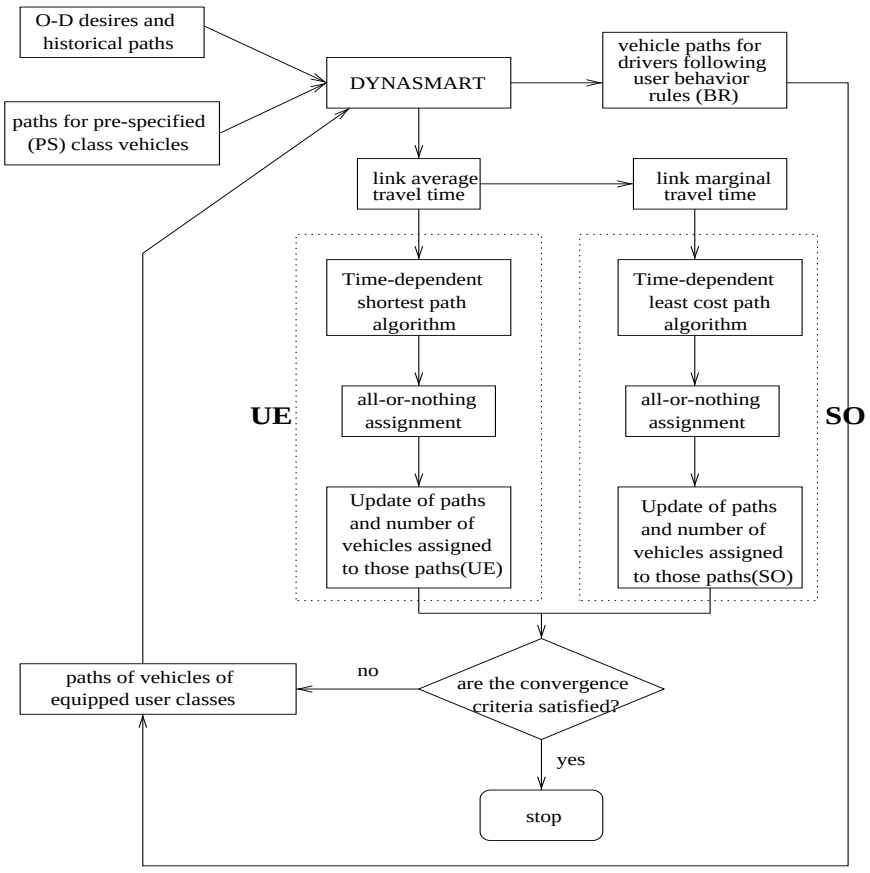

Fig. 13. The MUCTDTA Algorithm[17].

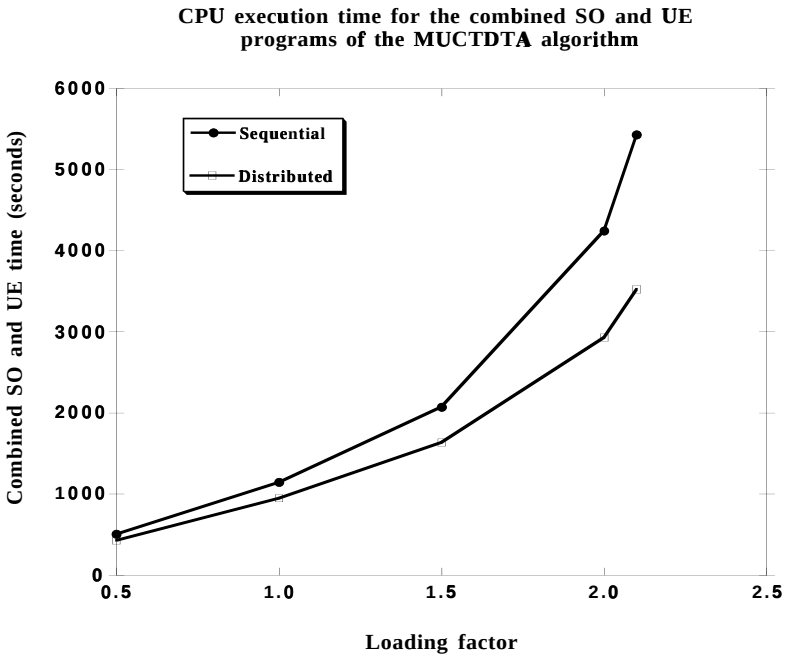

Fig. 14. Comparison of sequential and distributed combined SO and UE CPU times under different loading factors.

workstation executes the overall logic, and coordinates and communicates with the remote workstations. Together, the SO and UE programs constitute about one-third of the total CPU time of the algorithm. These percentages may vary slightly with the data size (number of vehicles) and other network-related parameters (network topology, spatial and temporal O-D demand pattern).

Fig. 14 illustrates the combined CPU time of the SO and UE programs with loading factor for the sequential and distributed cases. The rate of increase of the combined CPU time in the sequential case is larger than that of the distributed case with congestion level. As highlighted in Fig. 15, both the speedup and efficiency of the combined SO and UE programs increase with loading factor. Also, they represent substantially better performance especially at higher congestion levels compared to the results in Section 4, emphasizing that the performance enhancement depends on the fraction of the computation that is distributed.

The original sequential MUCTDTA code does not focus on computational efficiency and is not programmed for a truly distributed approach. Also, it is written in FORTRAN, requiring a C language interface to implement the RPC which reduces its efficiency because of the interface overheads involved. Despite these limitations, the distributed version of the MUCTDTA algorithm provides a speedup of up to 1.55 when compared to the sequential version. From Table 1, this represents a high performance range speedup indicating substantial computational gains. Thus, even discounting a distributed perspective for the algorithmic logic in developing on-line strategies, a distributed computational environment seems beneficial for on-line data processing under ITS applications.

Experiments are conducted to analyze the computational performance of the distributed version of the MUCT-DTA algorithm under various loading profiles. An analysis of loading profiles is important for the dynamic per-

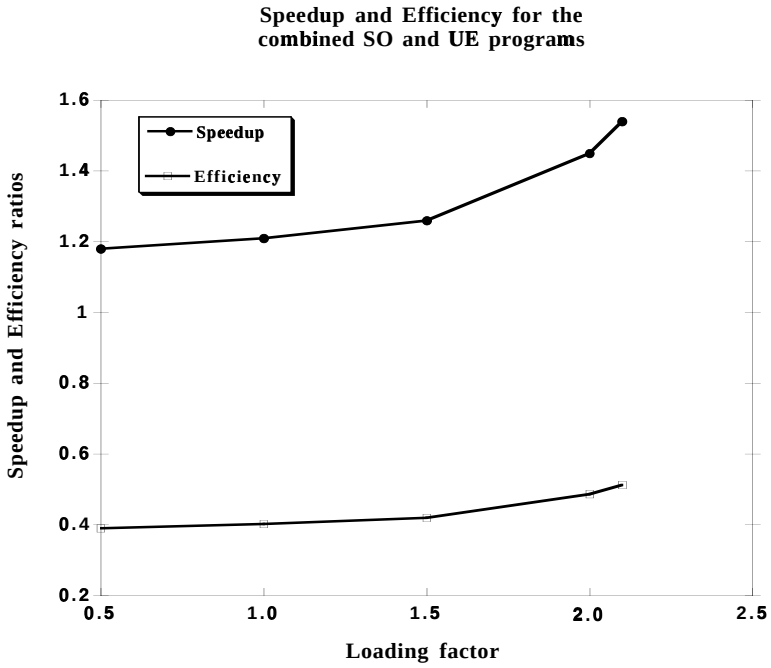

Fig. 15. Speedup and efficiency of the combined SO and UE programs.

formance of transportation networks from two perspectives: (i) the evolving traffic pattern can provide important input to various time-dependent ITS control strategies to enhance network performance, and (ii) an understanding of the performance under various loading profiles can aid in the development of appropriate peak-spreading demand management strategies. In the following experiments, each of the four user classes constitutes twenty five percent of the total network load. The experimental factors considered are:

• *Loading profiles:* Six temporal loading profiles are considered in the experiments, and are referred to as loading profiles 1 through 6. Figs. 16 through 21 depict the various loading profiles for loading factor 1.2. In all cases, vehicles are generated over a 35-minute period, and are shown as 5-minute aggregate vehicle numbers for visual convenience. Approximately the same number of vehicles are generated for all loading profiles. Loading profile 1 generates a typical peak-period pattern over the peak travel period duration, as illustrated in Fig. 16. Loading profile 2, shown in Fig. 17, generates vehicles uniformly over the peak travel period duration. Loading profile 3, shown in Fig. 18, impacts the network with relatively large number of vehicles over a five minute period which is preceded and succeeded by low levels of uniform loading for the rest of the duration (as is typical during special events such as football games and concerts). Loading profile 4 is left skewed; it generates a higher volume of vehicles towards the beginning of the peak period duration. Loading profile 5 is right skewed and generates higher volume of vehicles towards the end of the peak period duration. Loading profile 6 is similar to loading profile 3, except that it generates relatively large number of vehicles over two five minute periods. All cases include a five minute start-up generation time in order get the network reasonably occupied, followed by a thirty minute generation of vehicles for which statistics are accumulated.

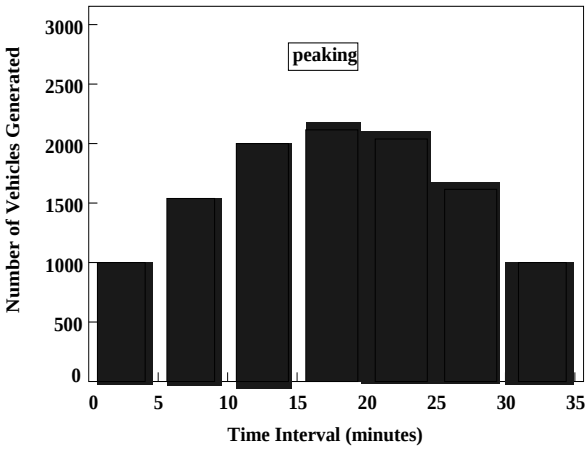

Fig. 16. Loading profile 1.

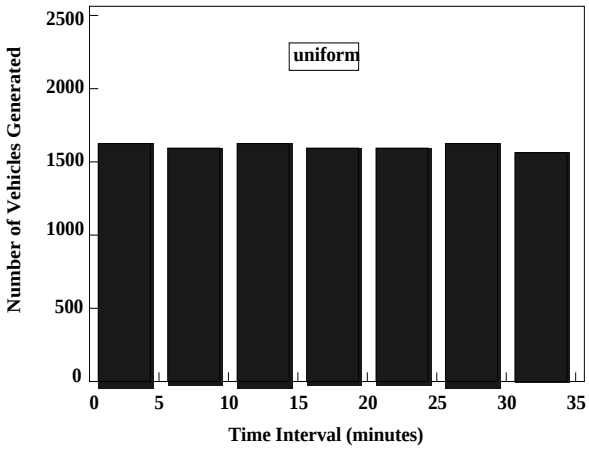

Fig. 17. Loading profile 2.

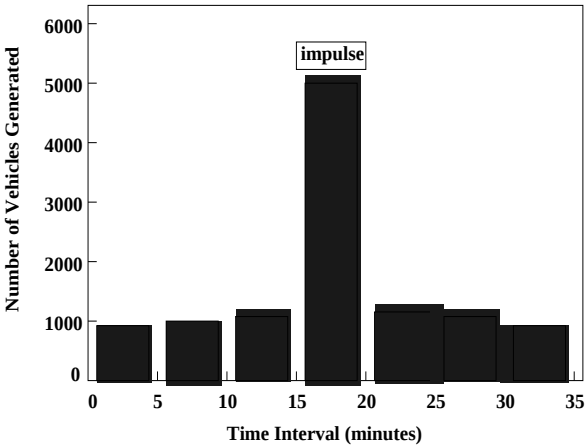

Fig. 18. Loading profile 3.

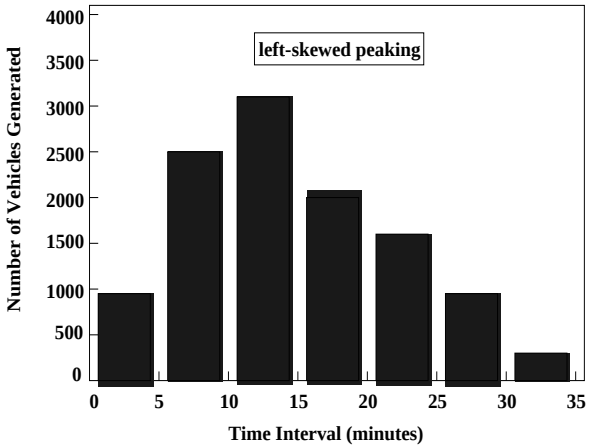

Fig. 19. Loading profile 4.

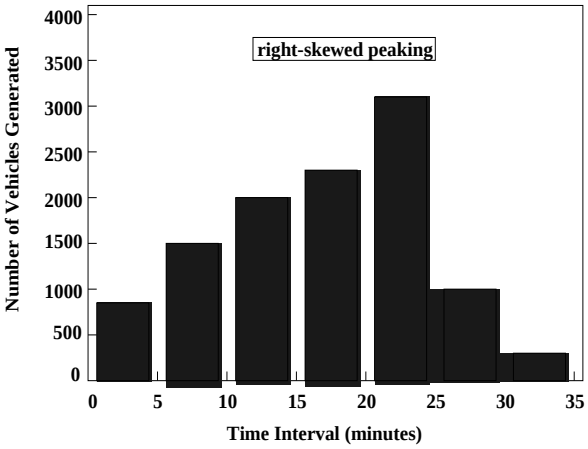

Fig. 20. Loading profile 5.

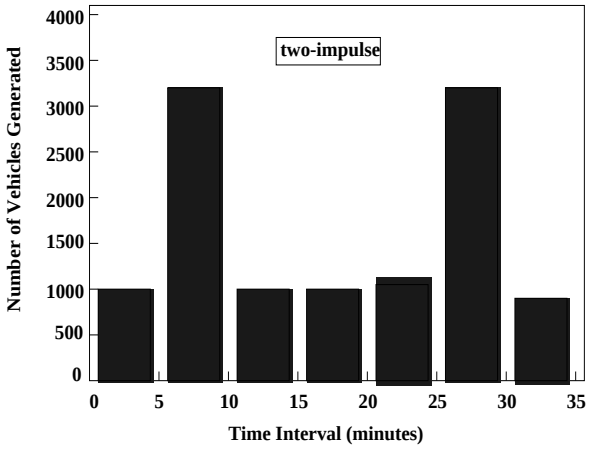

Fig. 21. Loading profile 6.

Table 3. Comparison of total CPU time of sequential and distributed cases for loading profile 1.

Loading factor	Sequential CPU time (sec.)	Distributed CPU time (sec.)	Speedup	Efficiency	Iteration number at convergence
1.20	1308.92	1064.64	1.23	0.41	5
1.60	2213.33	1738.14	1.27	0.42	8
2.00	4248.62	2933.26	1.45	0.48	10

Table 4. Comparison of total CPU time of sequential and distributed cases for loading profile 3.

Loading factor	Sequential CPU time (sec.)	Distributed CPU time (sec.)	Speedup	Efficiency	Iteration number at convergence
1.20	2225.94	1787.87	1.25	0.42	8
1.60	4444.07	3026.19	1.47	0.49	12
2.00	4684.76	3125.47	1.50	0.50	15

• *Network load levels:* Three loading factors are considered in the experiments: 1.2, 1.6, and 2.0. They represent various levels of network congestion ranging from low (loading factor 1.2) to medium (loading factor of 2.0).

The results from the various experiments form the basis for the comparison of system and computational performances under various loading profiles. The three principal measures used to describe the overall system performance are average travel time (ATT), average trip distance (ATD), and average network concentration. The average network concentration (vehicles per lane-kilometer) is defined as the average of the number of vehicles per unit length in the system over the duration of interest. The time-dependent network concentration is obtained by taking five minute averages of number of vehicles per unit length in the system. The average network concentration over the duration of interest is obtained by taking the arithmetic average of the five minute averages.

• Computational Performance

Tables 3 and 4 indicate that the time per iteration and the number of iterations till convergence increase with the loading factor. These are characteristics of the solution algorithm and can be explained by the fact that a higher loading factor translates to a large number of vehicles in each iteration of the MUCTDTA algorithm. Figs. 22 and 23 compare the total CPU times of the sequential and distributed computing mechanisms for loading profiles 1 and 3, respectively. The combined CPU time of the SO and UE programs increases with the loading factor. This is because a larger number of vehicles implies a greater number of computations at the link (to model movement of vehicles on links) and node (to ensure that node vehicle conservation constraints are satisfied) levels. Fig. 24 illustrates that speedup increases with the loading factor, indicating that distributed computing generates

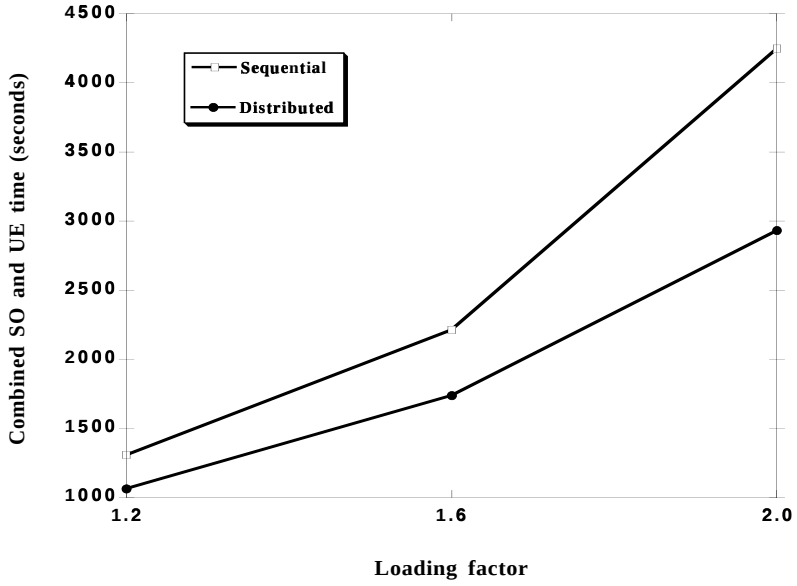

Fig. 22. Combined CPU times for the SO and UE programs for the sequential and distributed cases under loading profile 1.

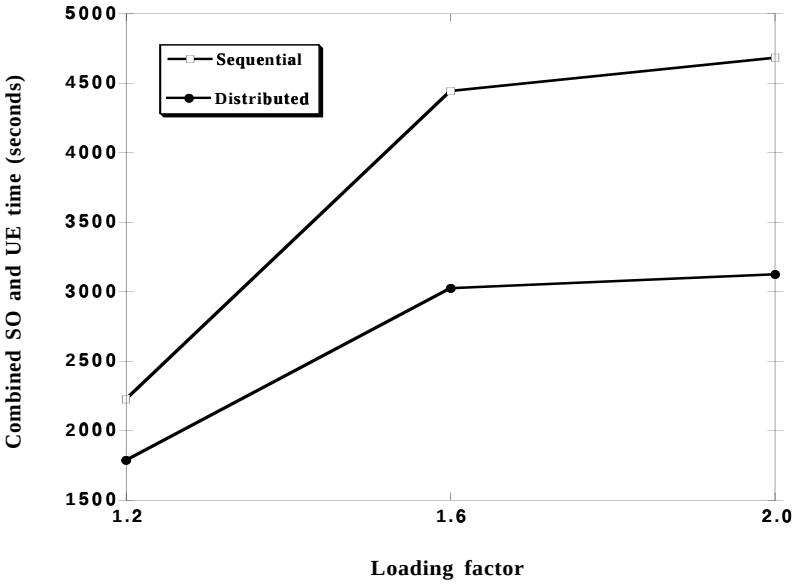

Fig. 23. Combined CPU times for the SO and UE programs for the sequential and distributed cases under loading profile 3.

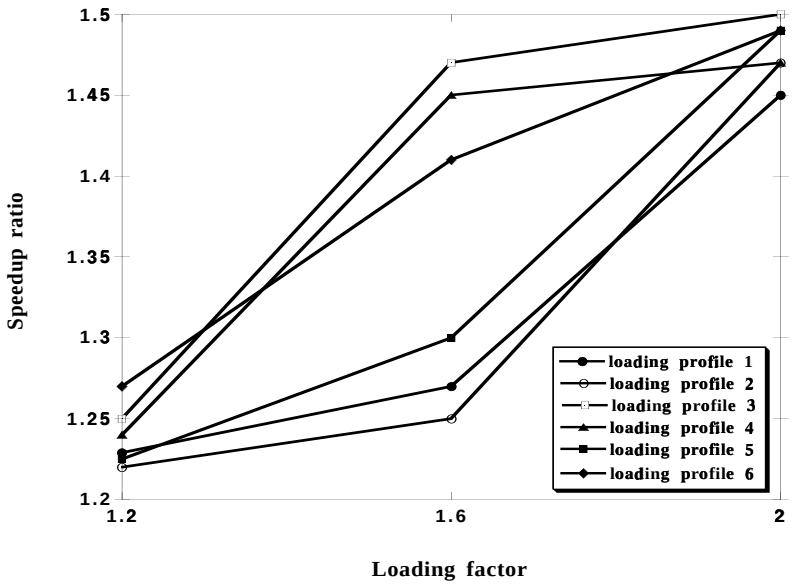

Fig. 24. Speedup of the SO and UE programs of the MUCTDTA Algorithm for different loading profiles.

higher savings when more intensive computation is involved. The figure also indicates that the rate of increase of speedup with loading factor depends on the loading profile. This trend is highlighted by comparing loading profiles 1 and 3 which exhibit different rates of increase of speedup. A comparison of Figs. 22 and 23 illustrates that the CPU times for loading profile 3 increase substantially from loading factor 1.2 to 1.6 unlike in the case of loading profile 1. This is because the network is not sufficiently congested under loading profile 1 unlike loading profile 3 at loading factor 1.6, indicated by an average network concentration of 9 vehicles/lane-kilometer in Table 5 for loading profile 1 versus 14 vehicles/lane-kilometer for loading profile 3. Loading profile 1 reaches a higher congestion level at loading factor 2.0, at which point the network is well congested under both the loading profiles, leading to comparable speedups. Since loading profile 1 is hardly congested at loading factor 1.6 and well congested at loading factor 2.0, its speedup increases substantially leading to the higher rate. In comparison, loading profile 3 has a congested network at loading factor 1.6 itself, thereby, its speedup does not substantially increase from loading factor 1.6 to loading factor 2.0. Hence, the average network concentration profile is a proxy indicator of the computational performance with congestion levels.

6 CONCLUSIONS

This paper investigates two types of RPC distributed mechanisms, MRPC and ARPC, as well as the computational performance of a distributed dynamic traffic assignment algorithm, MUCTDTA, under various congestion levels and loading profiles. The ARPC approach performs better than the MRPC approach due to its superior

Table 5. Average network concentration (vehicles/lane-kilometer)

Loading profile \ Loading factor	1.2	1.6	2.0
1	6	9	14
2	6	9	12
3	8	14	23
4	7	13	22
5	7	11	18
6	7	11	16

communication mechanism, primarily because it circumvents the creation of processes. Hence, an ARPC based distributed environment is used to analyze the computational performance of a distributed version of the MUCT-DTA algorithm. The results indicate that the input-output data can represent a significant component of the total CPU time, especially in the context of time-dependent on-line data under ITS. Also, they suggest that distributed computing at a higher level may result in significant savings, even without distributing/parallelizing algorithmic code at a lower level.

The MUCTDTA results under different time-dependent loading profiles and congestion levels indicate that though the CPU times for both the sequential and distributed cases increase, the rate of increase in the sequential case is larger than that of the distributed case as the network load increases. In addition, the speedup increases with the congestion levels. Of importance to on-line ITS strategies, the computational gains (speedup) depend not only on the congestion levels, but also the temporal distribution of the demand (loading profiles). Also, the average network concentration can be a reliable proxy for computational performance in terms of speedup in a distributed environment.

Another result with key implications for system managers at traffic control centers and agencies is that a large number of workstations (processors) does not necessarily translate into superior computational performance since communication overheads increase with the number of workstations. Hence, substantial computational gains may be achieved with marginal additional investments in computer hardware, depending on the on-line problem and the associated solution logic. In practice, the optimal computational resource allocation depends on the problem being addressed. Thus, it is necessary to investigate the tradeoffs between the computational performance requirements and the available monetary resources.

It should be noted that only a few specific problems are investigated using these distributed techniques, precluding generalization of results. For example, we do not investigate the actual gain under distributed computing by varying network sizes. Also, guidelines on how to choose the distributed routines or segments are not discussed in this paper. However, the results provide key insights, trends, and an understanding of the issues involved in the

application of distributed environments to real-world transportation problems, especially in the context of on-line operations. They suggest that a distributed network of computers provides a realistic, economical, and flexible computing environment for addressing real-time transportation problems. Given that technologies under ITS are still evolving vis-à-vis applications to on-line traffic operations, the distributed environment provides seamless scalability to incorporate future functional capabilities with large data needs (such as dynamic ramp metering, VMS-based information, etc.) without a fundamental revamping of the computational system. The results also highlight the need for fundamental shifts in simulation and modeling methodologies[8,16,20] and paradigms for fully exploiting the potential of parallel/distributed computing. For example, the computational perspective should be integral to the design of the algorithms or methodologies and their implementation, especially if on-line functional capabilities are desired. This requires an understanding of parallelisms inherent to the problem being addressed, and the associated solution approach.

7 REFERENCES

1. Bertsekas, D. P. & Tsitsiklis, J. N., *Parallel and Distributed Computation*, Prentice-Hall, 1989.

2. Birrell, A. D. & Nelson, B. J., Implementing Remote Procedure Calls. *ACM Trans. Comput. Syst.* Vol. 2 No.1, 1984, pp. 39-59.

3. Chang, G. L., Junchaya T., & Santiago A. J., A Real-Time Network Traffic Simulation Model for ATMS Applications: Part I-Simulation Methodologies, *IVHS Journal*, Vol. **1**(3), 1994a, pp. 227-241.

4. Chang, G. L., Junchaya T., & Santiago A. J., A Real-Time Network Traffic Simulation Model for ATMS Applications: Part II-Massively Parallel Model, *IVHS Journal*, Vol. **1**(3), 1994b, pp. 243-259.

5. Greenlaw R., Hoover, H. J., & Ruzzo, W. L., *Limits to Parallel Computation: P-completeness Theory*, Oxford University Press, 1995.

6. Habbal, M., Koutsopoulos, H. N., & Lerman, S., A decomposition algorithm for the all-pairs shortest path problem on massively parallel computer architectures, *Transportation Science*, Vol. 28, No. 4, 1994, pp. 292-308.

7. Halati, A., Lieu, H., & Walker, S., CORSIM - Corridor Traffic Simulation Model, *The 76th Annual Meeting of the Transportation Research Board, Washington, D. C.*, 1997.

8. Hawas, Y. E. and Mahmassani, H. S., Comparative Analysis of Robustness of Centralized and Distributed Network Route Control System in Incident Situations, *Transportation Research Record* **1537**, 1997, pp. 83-90.

9. Hwang, K. & Briggs, F. A., *Computer Architecture and Parallel Processing*, McGraw-Hill, 1984.

10. Jayakrishnan, R., Mahmassani, H. S., & Hu, T., An evaluation tool for advanced traffic information and management systems in urban networks, *Transportation Research*, 2C, 1994, pp. 129-147.

11. Kuck, D. J., Muraoka, Y., & Chen, S. C., On the number of operations simultaneously executable in Fortran-like programs and their resulting speedup, *IEEE Trans. Comp.* Vol. c-21, No. 12, 1972, pp. 1293-1310.

12. Kuck, D. J., *High performance computing*, Oxford University Press, 1996.

13. Lieberman, E. B., Wicks, D. A., Woo, J., & Wu, A., Network Flow Simulation for Urban Traffic Control System - Phase II, Vol. 2, Program Documentation for NETSIM, Network Simulation Model, Part I, Report FHWA-RD-77-42, FHWA, U.S. Department of Transportation, 1977.

14. Mahmassani, H. S., Hu, T., Peeta, S. & Ziliaskopoulos, A., Development and testing of dynamic traffic assignment and simulation procedures for ATIS/ATMS applications, Technical Report DTFH61-90-C-00074-FG, Center for Transportation Research, The University of Texas at Austin, 1994.

15. Nelson, B. J., *Remote Procedure Calls*, Ph.D. dissertation, Carnegie Mellon University, 1981.

16. Pavlis, Y. and Papageorgiou, M., Investigation of Some Decentralized Feedback Strategies for Route Guidance in Traffic Networks, presented at TRISTAN-III, Puerto Rico, June, 1998.

17. Peeta, S., *System optimal dynamic traffic assignment in congested networks with advanced information systems*, Ph.D. Dissertation, The University of Texas at Austin, 1994.

18. Peeta, S. & Mahmassani, H. S., System optimal and user equilibrium time-dependent traffic assignment in congested networks, *Annals of Operations Research 60*, 1995a, pp. 81-113.

19. Peeta, S., & Mahmassani, H. S., Multiple user classes real-time traffic assignment for on-line operations: A rolling horizon solution framework, *Transportation Research*, Vol. 3, No. 2, 1995b, pp. 83-98.

20. Peeta, S. and Zhou, C., Effectiveness of A Priori Stochastic Dynamic Traffic Assignment in Congested Networks under Real-Time Traffic Management and Information Systems, Proceedings of the 8^{th} International Federation of Automatic Control Symposium on Transportation Systems, Chania, Greece, 1997.

21. Smith, J. R., *The Design and Analysis of Parallel Algorithms*, Oxford University Press, 1993.

22. Zenios, S. A., Parallel numerical optimization: current status and an annotated bibliography, *ORSA Journal on Computing* **1**, 1989, pp. 20-43.

23. Ziliaskopoulos, A. & Mahmassani, H. S., A time-dependent shortest path algorithm for real-time intelligent vehicle/highway systems applications, *Transportation Research Record 1408*, 1994, pp. 94-100.