Traffic Flow Simulation Through Parallel Processing

ANTHONY THEODORE CHRONOPOULOS AND GANG WANG

Numerical methods for solving traffic flow continuum models have been studied and efficiently implemented in traffic simulation codes in the past. Explicit and implicit methods have been used in traffic simulation codes in the past. Implicit methods allow a much larger time step size than explicit methods to achieve the same accuracy. However, at each time step a nonlinear system must be solved. The Newton method, coupled with a linear iterative method (Orthomin), is used. The efficient implementation of explicit and implicit numerical methods for solving the high-order flow conservation traffic model on parallel computers was studied. Simulation tests were run with traffic data from an 18-mile freeway section in Minnesota on the nCUBE2 parallel computer. These tests gave the same accuracy as past tests, which were performed on one-processor computers, and the overall execution time was significantly reduced.

Macroscopic continuum traffic model flow based on traffic flow volume, density, and speed has been used in studying the behavior of freeway traffic in the past (*1–6*). These models involve partial differential equations defined on appropriate (road) domains with suitable boundary conditions, which describe various traffic phenomena and road geometries.

Such a model is the semi-viscous model (SVM) (*4*). This mathematical model for the traffic flow is adopted for our research. Our main goal is to derive efficient methods to parallelize the numerical solution of SVM. This choice is not a restriction. The same methods can be applied to other continuum models (*2,5*).

We next outline the SVM equations. Let $k(x,t)$, $q(x,t)$, and $u(x,t)$ be the traffic density, volume and speed, respectively, at the (road)space-time point (x,t). Let u_f be the free flow speed and k_{jam} the jam density (*7*). Let the relaxation time T (which is assumed to vary with density k) be

$$T = t_0\left(1 + \frac{rk}{k_{jam} - rk}\right) \tag{1}$$

where $t_0 > 0$ and $0 < r < 1$ are model parameters. Let u_f be the traffic free flow speed.

The generation term $g(x,t)$ represents the number of vehicles entering or leaving the traffic flow in a freeway with entries and exits.

$$\frac{\theta U}{\theta t} + \frac{\theta E}{\theta x} = Z \tag{2}$$

where U, E, and Z are the following vectors:

$$U = \begin{pmatrix} k \\ q \end{pmatrix} \tag{3}$$

A. T. Chronopoulos, Department of Computer Science, Wayne State University, 5143 Cass Ave., Detroit, Mich. 48202. G. Wang, EE/CSCI Bldg., University of Minnesota, 200 Union Street S.E., Minneapolis, Minn. 55455.

$$E = \begin{pmatrix} ku \\ u^2k + \dfrac{\nu}{\beta + 2} k^{\beta+2} \end{pmatrix} \tag{4}$$

$$Z = \begin{pmatrix} g \\ \dfrac{k}{T}[u_f(x) - u] + gu \end{pmatrix} \tag{5}$$

where ν and β are model parameters. If speed data are not available from the real traffic data measurements, then a $q - k$ curve based on occupancy data (*7*) is used to generate the speed data.

The improvement of computational efficiency in the continuum traffic models has been the focal point in the development of traffic simulation programs. It is understood that the computer execution time to solve traffic flow problems depends not only on the size of the freeway and the complexity of roadway geometries but also on the model equations and numerical schemes used in their discretization. Explicit and implicit numerical methods have been used to compute the solution of traffic flow continuum models (*3,8*). Implicit numerical methods require the solution of nonlinear systems of equations at each time step.

In this work, the Lax method (explicit) and the Euler method (implicit) applied to solve the momentum conservation model on the nCUBE2 parallel computer were parallelized. Implicit methods allow a much larger time step size than explicit methods for the same accuracy. However, at each time step a nonlinear system must be solved. The Newton method coupled with a linear iterative method (Orthomin) was used. The convergence of Orthomin with parallel incomplete LU factorization preconditionings was accelerated. A code (in C) was written simulating a freeway with multiple entry/exit traffic flow on the nCUBE2 parallel computer located at the Sandia National Laboratory. The nCUBE2 is a distributed-memory MIMD parallel system. Communication of data between processors is more time consuming than computation. Thus, a method is fast not only if its tasks can be computed in parallel but also if it does not require frequent data exchanges among the processors.

Tests were conducted with real data from an 18-mile freeway section of I-494 in Minneapolis, Minnesota; 427 space grid points were used in the discretization, only parallelizing the space dimension and computing serially along the time dimension. The Lax method requires very little communication and thus achieves very good speedup, which scales well up to 128 processor nodes. The Euler method on a 16-processor configuration runs about four times faster than on a two-processor configuration. Because of the construction of the preconditioning, we could not run the Euler case on a single processor. However, the Euler does not scale well when the number of processors increases because of the frequent communication required by the iterative method and the preconditioning.

FREEWAY MODEL

Two schemes were used to add and subtract entry-exit (ramp) traffic volumes to the main-lane traffic volume in SVM.

1. Point entry-exit—Ramp volumes are assumed to merge into (diverge from or exit from) the freeway main lane at a single space node. This treatment is necessary to simplify the modeling and reduce computation time at such main-lane nodes.
2. Weaving entry/exit—This is used when the ramp is directly connected to another freeway and is explained in more detail below.

The weaving scheme is outlined as follows: consider the traffic flow volume in a freeway section, presented in Figure 1, at a fixed discrete time. In Figure 1, volume v_1 represents the through traffic volume flow from link A to link B, and volume v_2 represents the diverging volume from link A to link F, and $q_A = v_1 + v_2$; v_3 is the merging volume from link E to link B, and volume v_4 is the through volume from link E to link F; and $q_E = v_3 + v_4$. It is obvious that $q_F = v_2 + v_4$ and $q_B = v_1 + v_3$. Because there are interchanges of v_2 and v_3, traffic friction at link B and link E in this case is greater than the case of a single-entrance ramp or exit ramp. Thus, this must be taken into account by calibrating (locally) the u_f parameter in the mathematical model for these space nodes. Also, only merging dynamics at an entrance ramp must be employed if $v_2 = 0$. Similarly, only diverging dynamics must be employed if $q_E = 0$.

When the distance between links E and F is less than 180 m (600 ft), merging and diverging movements must be completed within a short distance. However, since both q_E and q_F require lane changing in the same limited length of roadway at the same time, the sum of q_E and q_F must be included in the generation term of the model. If the generation term $g > 0$, the short weaving section is treated as a single on-ramp; if the generation term $g < 0$, it is treated as a single off-ramp. The generation term then becomes

$$g = (q_E - q_F)/\Delta x. \qquad (6)$$

PARALLEL IMPLEMENTATION ON NCUBE2

In this section, we outline the basic features of nCUBE2 and discuss the parallelization of the Lax and Euler methods.

nCUBE2 PARALLEL COMPUTER

The nCUBE2 is a multiple instruction multiple data (MIMD) hypercube parallel computer system. A hypercube model is an example of a distributed-memory, message-passing parallel computer. In a hypercube of dimension *ndim*, there are $p = 2^{ndim}$ processors. These processors are labeled by $0, 1, \ldots, p - 1$. Two processors i and j are directly connected if the binary representation of i and j differs in exactly one bit. Each edge of the hypercube graph represents a direct connection between two processors. In a hypercube of dimension *ndim*, each processor is connected to *ndim* other processors. Thus, any two processors, in a hypercube graph, are connected by a maximum distance of *ndim* edges. Figure 2 shows a hypercube graph of dimension $ndim = 4$. The number of processors to be active is chosen by the user but must be a power of two.

Table 1 summarizes interprocessor communication times for neighbor processors and the basic floating point operation times for nCUBE2 (*9*). It indicates that communication even between neighbor processors is several times slower than floating point operations.

In a hypercube with a high communication latency, the algorithm designer must structure the algorithm so that large amounts of computation are performed between communication steps.

The two important factors that influence the delivered performance of this machine are load balancing and reduction of communication overhead. A program is load balanced if all the processors are kept busy. An efficient algorithm is one for which both computations and data are distributed among the processors so that the computations run in parallel, balancing the computational loads of the processors as well as possible.

PARALLELIZATION OF LAX AND EULER METHODS

Let p be the number of processors available in the system. The parallelization of the discrete model is obtained by partitioning the space domain (freeway model) into equal segments $\text{Seg}_0, \ldots, \text{Seg}_{p-1}$ and assigning each segment to the processors $P_{j_0}, \ldots, P_{j_{p-1}}$. The choice of indexes $j_0, \ldots, j_{p-1}$ is called mapping of the segments to the processors.

The computations associated with each segment have as their goal to compute the density, volume, and speed over that segment. The time dimension is not parallelized. This essentially means that the quantities k_j^i, q_j^i, u_j^i *are computed by processor* P_{j_k} *if the space node* $j\Delta x$ *belongs to the segment* Seg_{j_k}. This segment-processor mapping must be such that the communication delays for data exchanges, required in the computation, are minimized. Such a mapping of a linear array of sets onto a hypercube is achieved by the Gray mapping, described elsewhere (*9*). For example, assume that V_0, V_1, V_2, V_3 are four vectors mapped on the processors of a hypercube. Also, assume that the computations for vector V_j may require

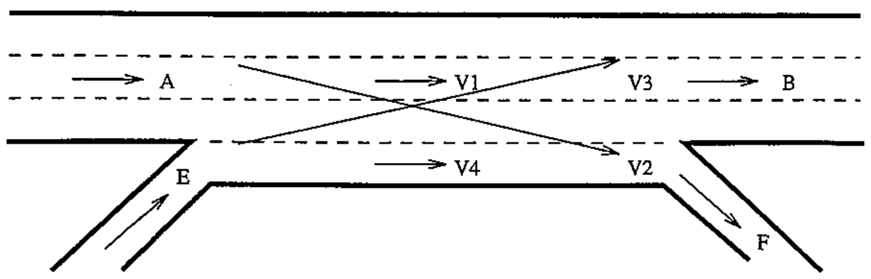

FIGURE 1 Weaving flows in freeway.

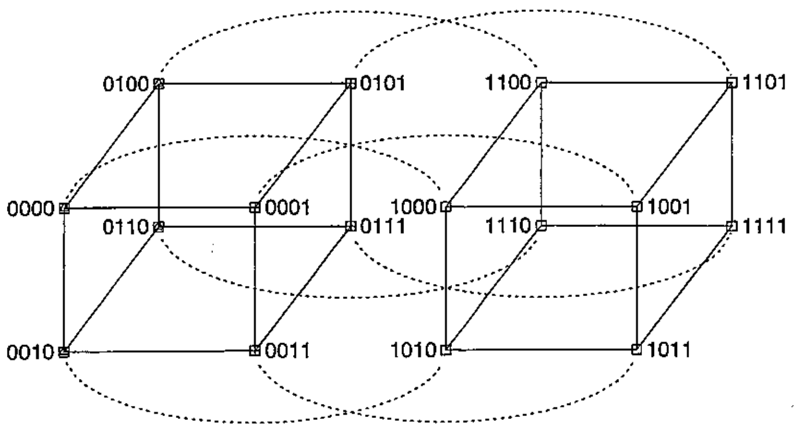

FIGURE 2 Hypercube graph of dimension *ndim* = 4.

some components of the vector V_{j-1} or V_{j+1} (whenever the indexes are defined). Then the sequence of vectors V_0, V_1, V_2, V_3 is mapped, in order, to the processor sequence P_0, P_1, P_3, P_2 on the hypercube by the Gray mapping.

The Lax method is parallelized naturally by mapping the road segments onto processor nodes. The only data exchange occurs between immediately adjacent processors nodes (as indexed in the Gray mapping). The data at the boundaries of the road segments are exchanged at each time step. This means that only one communication point is required during each time step of the method.

The Euler method advances each time step by solving a system of nonlinear equations. This system is solved using Newton and a linear iterative method [preconditioned Orthomin (*8,10,11*)]. Thus the computation of the Jacobian matrix and right-hand side are required together with the computations of the iterative method. The computation of the Jacobian *A* and right-hand-side vector (***rhs***) for the linear systems, which are solved in the discrete model, requires data that may be located in an adjacent segment. Because of the Gray mapping, the processors storing these data are immediate neighbors. Thus, the least amount of communication will be required in computing *A* and ***rhs***. Computation of ***rhs*** is performed at each time step, but the Jacobian *A* is computed very infrequently (*10*). The Jacobian *A* and the ***rhs*** vector are computed, in parallel, by the processor, which is assigned the corresponding space nodes.

Most of the execution time in the Euler-Momentum method is spent in the solution of linear systems. In the parallel implementation of the Orthomin method to solve the linear system, four main computations are parallelized: a linear combination, a dotproduct, a matrix times vector product, and a preconditioning step. Only the matrix times vector and the preconditioning will be explained. The parallelization of the dotproducts and linear combination is studied elsewhere (*9,12*).

TABLE 1 Computation and Communication Times on nCUBE2

Operation	Time	Comm/Comp
8 Byte transfer	111 μ sec	-
8 Byte Add	1.23 μsec	90 times
8 Byte Multiply	1.28 μsec	86 times

Let $n = 2pq + s$ be the dimension of the linear system, where $1 \leq q$ and $s = 2(p - 1)$. The special choice of n is not really a restriction, and it is made for discussing the parallelization of domain decomposition preconditioning. We distribute the matrix *A* and ***rhs*** data in rows to the *p* processors so that each processor gets $2q + 2$ rows, except for the last (in the Gray processor mapping) processor, which gets $2q$ rows. Figure 3 shows an example of the matrix of coefficients and the vector of unknowns for max$j$ = 12(number of space nodes), or $n = 22$ (dimension of matrix and ***rhs*** vector), which is assigned to four processors of the nCUBE.

Although nCUBE2 is an MIMD computer, these algorithms are data parallel. This means the implementation is such that each processor executes the same instructions on different data. Also, care has been taken to divide the data equally and assign them to processors so that the processors are load balanced. The only imbalance occurs when a few processors handle the ramp data calculations while the rest of the processors remain idle. However, these calculations take less than 1 percent of the total execution time.

INCOMPLETE LOWER-UPPER PRECONDITIONINGS

In determining a preconditioner (matrix) P_r, we look for a matrix so that either $P_r \approx A^{-1}$ or AP_r has clustered eigenvalues. The matrix P_r must be easily computable. Equivalently stated, the system $P_r^{-1}x = b$ must be easy to solve.

The basis for the incomplete lower-upper (ILU) factorization method is that the matrix *A* is decomposed into upper (*U*) and lower (*L*) triangular matrices such that $A = LU + \Theta$, where $P_r^{-1} = LU$ and Θ is an error matrix. Also, we assume that if the entry $A_{j1,j2}$ is zero, then both $U_{j1,j2}$ and $L_{j1,j2} = 0$. In other words, *L* and *U* have the same sparsity patterns as *A*. This is the ILU(0) preconditioning method. ILU preconditioning can improve the convergence rate of the iterative solvers considerably if it is implemented properly on a parallel computer (*13*).

Two different parallel implementations of ILU reconditioning are: (a) overlapped submatrix regions (OSR) ILU by Radicati di Brozolo and Robert (*14*) and (b) domain decomposition (DD) ILU (*11*), discussed earlier.

In method (a), the matrix is partitioned into a number (p, equal to the number of PEs) of overlapping submatrix regions. The ILU of

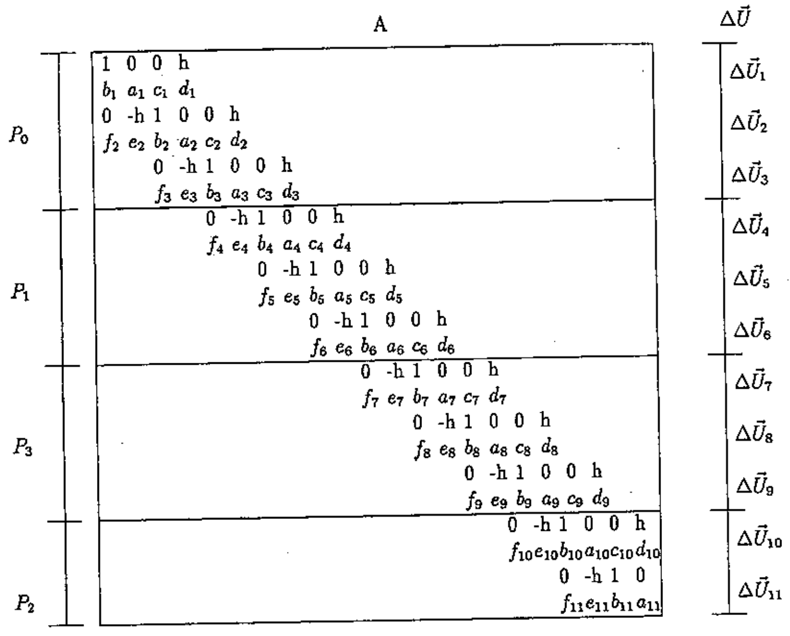

FIGURE 3 Original structure of matrix A and vector ΔU.

each region is computed independently and the average solution is computed on the overlapped regions. More details on (a) may be found elsewhere (*13,14*). Figure 4 illustrates OSR ILU.

Figure 5 presents the coefficient matrix and vector of unknowns after being reordered and mapped onto the four processors (*11*). In Figure 5, only the nonzero entries are shown. The matrix and right-hand-side are partitioned into sets and assigned to the processor sequence by the Gray mapping (P_0, P_1, P_3, P_2).

RESULTS

The Euler-Momentum method has been implemented on the nCUBE2 parallel computer located at the Sandia National Laboratory. The freeway test site was a multiple entry-exit freeway section in the Minneapolis, Minnesota, freeway network: a section of eastbound I-494 that extends from the Carlson Parkway to Portland Avenue. It is 18 miles long and has 21 entry ramps and 18 exit ramps. To test the program, the time stepsize selection was made as follows. For the Euler method, the time stepsize was increased so

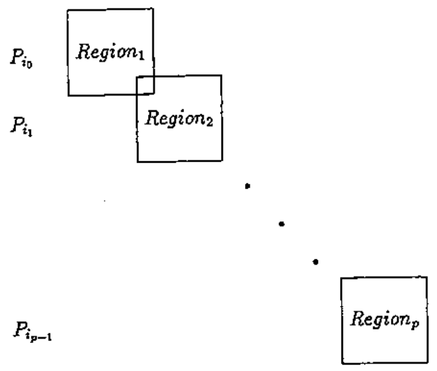

FIGURE 4 Processor assignment for OSR ILU.

that the maximum error did not exceed that of the Lax method (*8*). To test the program, the time and space mesh sizes were $\Delta t = 10$sec and $\Delta x = 61$m (200 ft). For the Lax method $\Delta t = 1$sec and $\Delta x = 61$m (200 ft). Our results are distinguished into two units: comparisons with real data and performance timings.

Let N be the number of discrete time points at which real traffic flow data are collected. Traffic data are collected at the upstream-downstream boundaries of the freeway section and at check-station sites inside the freeway section. The simulation computed volume and speed data are compared with the check-station site data. The following error is used to measure the effectiveness of the simulation in comparison with actual data:

Maximum Relative Error with 2 − Norm

$$= \left[\frac{\sum_{i=1}^{N}(\text{Observed}_i - \text{Simulated})^2}{\sum_{i=1}^{N}\text{Observed}_i^2}\right]^{1/2} \quad (7)$$

The error statistics are summarized in Table 2. The relative errors are at about 10 percent for the volume but are lower for the speed measurements. This accuracy range of agreement with the measured data is acceptable (*3,4,8*). The simulation errors are independent of the number of processors. This is expected because of the parallel implementation of data and the fact that only the space dimensions were parallelized.

The execution times for a varying number of processors are listed in Table 3. It is clear that the lowest times are attained by the Lax method. The Euler method program was not run on more that 16 processors because there was a slowdown in performance.

Speedup is defined as the ratio

$$S_p = \frac{T_1}{T_p} \quad (8)$$

where T_1 is the execution time on two processors, and T_p is the execution time on p processors. For the Euler method, we consider the relative speedup:

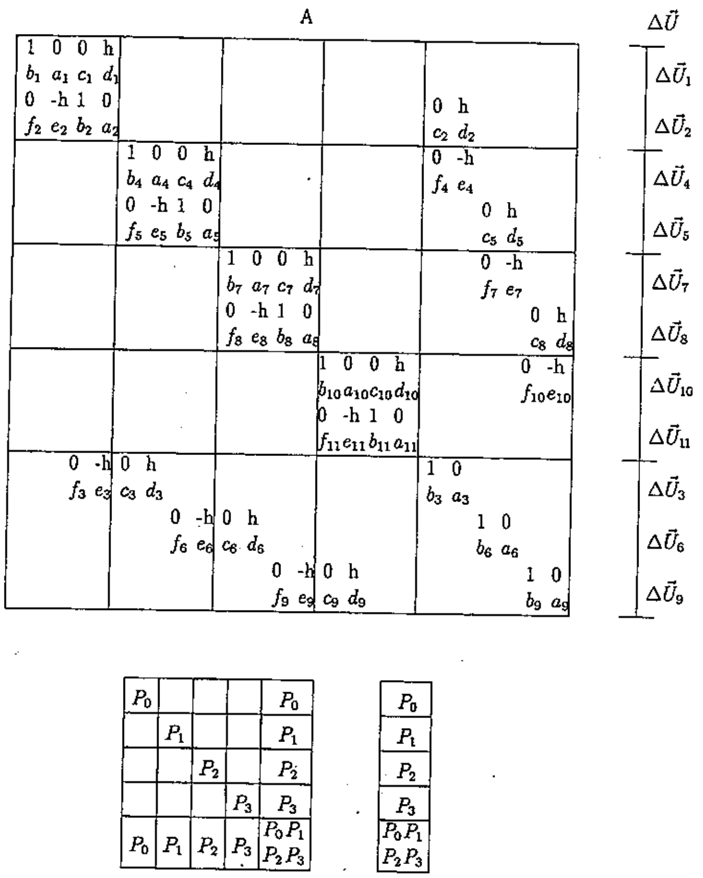

FIGURE 5 Partition of linear system and mapping to processors in DD ILU.

$$S_p = \frac{T_2}{T_p} \tag{9}$$

where T_2 is the execution time on two processors, and T_p is the execution time on p processors. Relative speedup is considered because the preconditioning code was designed to run on more than one processor, and thus the timing could not be obtained on a single processor.

The speedups on nCUBE2 are summarized in Figures 6–8. For the Euler method, the communication costs are higher in the DD ILU case in the matrix and ***rhs*** construction and in the ILU computations. Also, the dotproducts seem to take an unusually long time because of the high communication cost involved in computing them (*9*). Because the size of the problem is kept fixed, the relative speedup decreases as the number of processors increases. The Lax

TABLE 2 Error Statistics for Traffic Flow Volume (I-494)

Error Statistics for Traffic Flow/Speed						
	Volume error $(veh/5min)$			Speed error $(mile/hour)$		
Sites	*Lax*	*OSR ILU*	*DD ILU*	*Lax*	*OSR ILU*	*DD ILU*
1	.12	0.12	0.12	.03	.03	.03
2	.10	0.11	0.13	.04	.07	.08
3	.09	0.09	0.13	.04	.08	.08
4	.05	0.05	0.16	.03	.07	.07
5	.11	0.11	0.17	.05	.13	.11

TABLE 3 Simulation Execution Times

Procs No.	*Lax*	*OSR ILU*	*DD ILU*
1	284.6	-	-
2	145	487.8	544.2
4	75.7	295.6	254.3
8	40.4	180.7	159.1
16	22.9	135.5	175.1
32	14	-	-
64	9.6	-	-
128	7	-	-

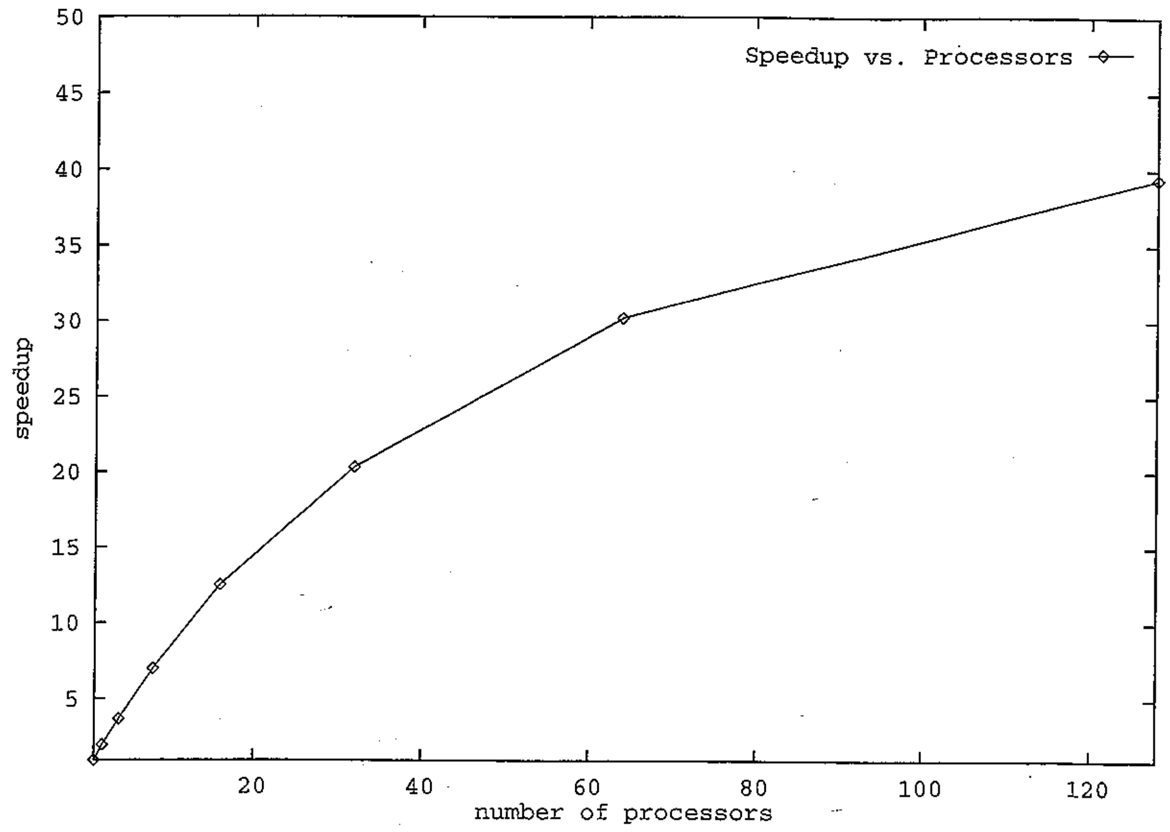

FIGURE 6 Speedup for parallel Lax on nCUBE2.

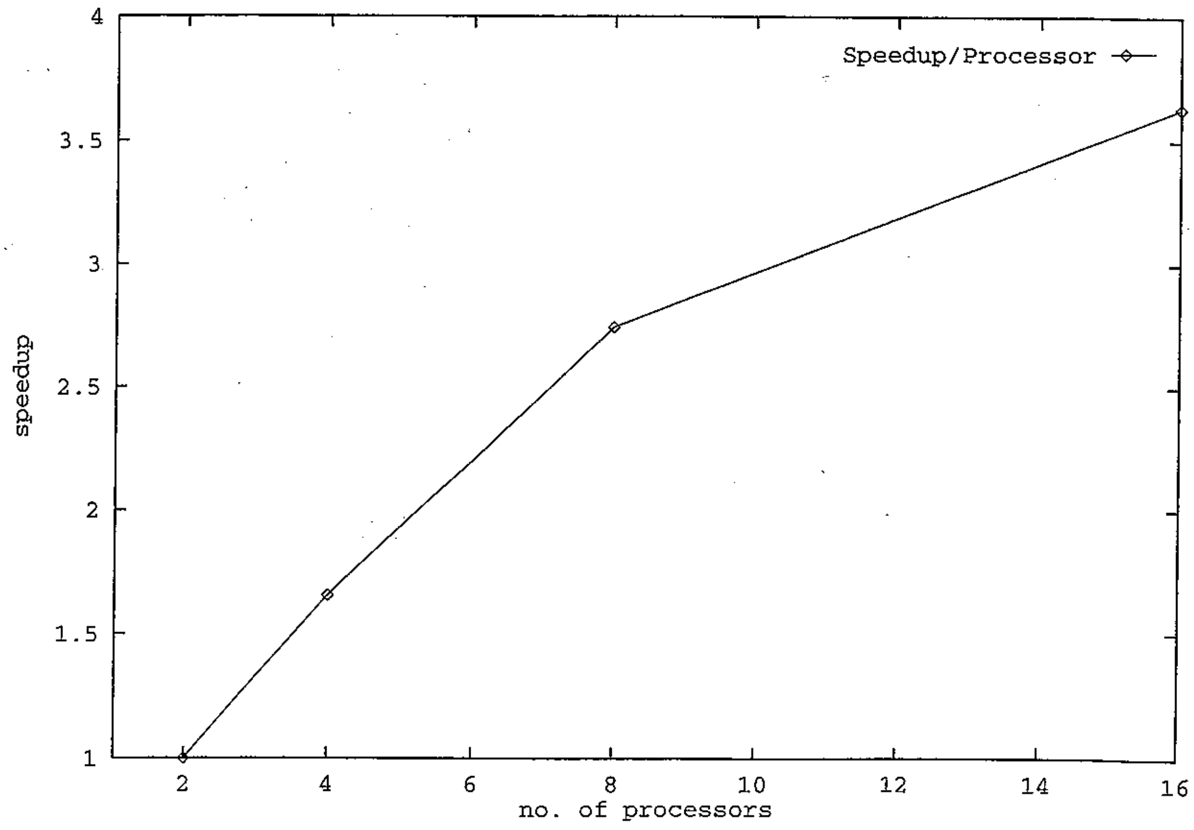

FIGURE 7 I-494 speedup for parallel ILU version on nCUBE.

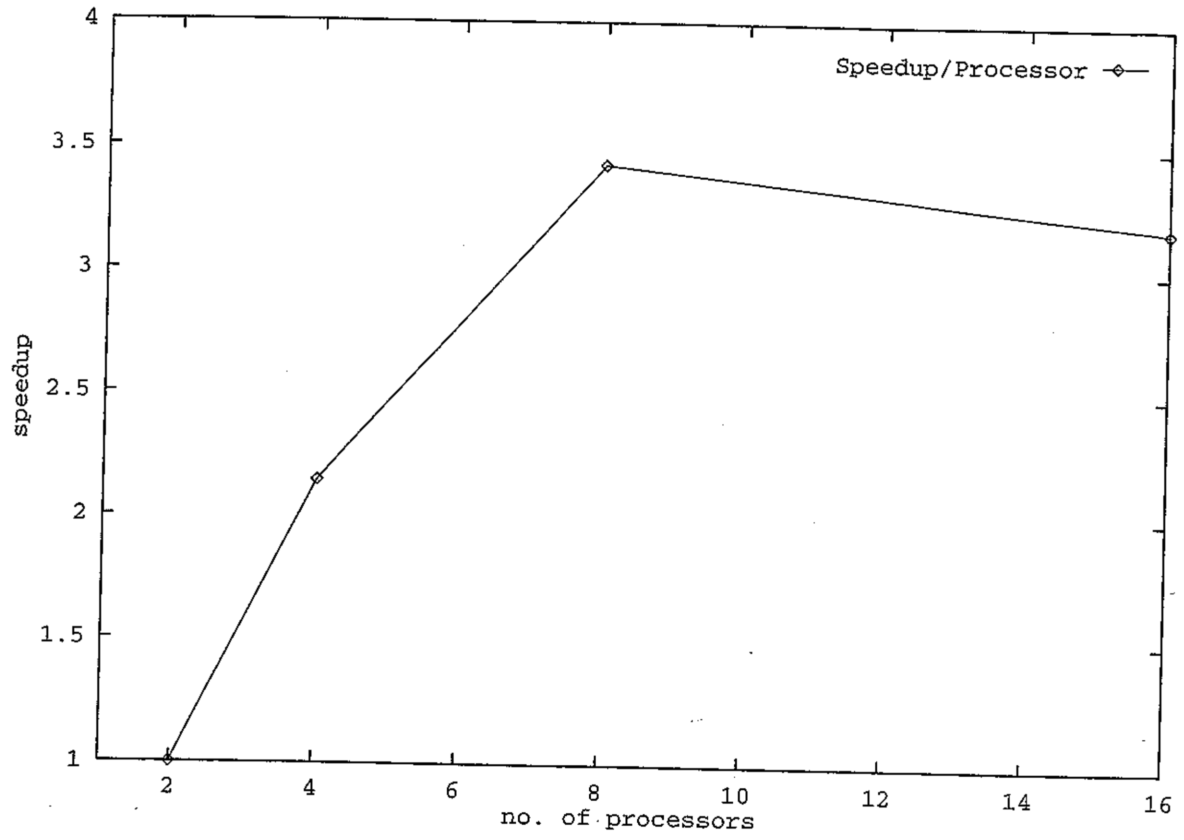

FIGURE 8 I-494 speedup for domain decomposition version on nCUBE.

method requires very little communication, and thus it parallelizes very well up to 128 processors.

CONCLUSIONS AND FUTURE WORK

We studied the parallelization of the solution of a traffic flow continuum model using the Lax and the Euler methods. For the same accuracy, implicit methods allow much larger time step size than explicit methods. However, at each time step a nonlinear system of equations must be solved; this study used the Newton method coupled with a linear iterative method (Orthomin); the convergence of Orthomin was accelerated with two parallel incomplete LU factorization preconditionings.

These methods were implemented on the nCUBE2 parallel computer located at the Sandia National Laboratory. On the nCUBE2, the parallel Euler-Momentum method on the 16 processors runs about four times faster than on the two processors. However, the Euler does not scale well when the number of processors increases because of the frequent communication required by the iterative method and the preconditioning. The Lax method requires very little communication, and thus it achieves very good speedup, which scales well up to 128 processor nodes.

In terms of future work, one could investigate other types of preconditioning and extend the simulation of traffic flow in a network of two or more freeways. For the solution of these problems, a larger number of processors would be used. Another area of future work would be in the use of these methods in applications such as real-time traffic prediction and control.

APPENDIX

Orthomin Iterative Method

Orthomin iterative method is next described to solve $Ax = \boldsymbol{rhs}$, where A is nonsymmetric matrix of order N. The Orthomin applies to nonsymmetric linear systems with the symmetric part of A being positive definite. Let k_0 be a positive integer. We describe the Orthomin iterative method with preconditioning as follows. In this algorithm, $j_i = \max(0, i - k_0 + 1)$, P_r is the right preconditioner ($P_rA \approx I$) which is obtained by the LU decomposition of the matrix A (*10*). Algorithm 1, Orthomin (k_0):

```
 1. Choose $x_0 = 0$
 2. Compute $r_0 = \boldsymbol{rhs} - Ax_0$
 3. $p_0 = r_0$

For i = 0 step 1 Until convergence Do
    4. $a_i = \frac{(r_i, Ap_i)}{(Ap_i, Ap_i)}$
    5. $x_{i+1} = x_i + a_i p_i$
    6. $r_{i+1} = r_i - a_i Ap_i$
    7. Compute $AP_r r_{i+1}$
    8. $b_j^i = \frac{(AP_r r_{i+1}, Ap_j)}{(Ap_j, Ap_j)}$
    9. $p_{i+1} = P_r r_{j+1} + \Sigma_{j=j_i}^{i} b_j^i p_j$
   10. $Ap_{i+1} = AP_r r_{i+1} + \Sigma_{j=j_i}^{i} b_j^i Ap_j$
Endfor
```

Domain Decomposition

The parallel algorithms are given for the complete *LU* factorization a matrix and solution of a linear system using the domain decomposition method (*11,15*). In this study, the *LU* factors were obtained only once and were used to solve the linear systems of the preconditioning step in the Orthomin iterations. This is a domain decomposition ILU (0)-type method (*16*).

Consider the partition of the linear system in Figure 5. Let $s = 2(p - 1)$ be the number of unknowns in the separator set $S = \cup S_i$, where S_i are the separator nodes between the sets Φ_i and Φ_{i+1}. Denote by $\overline{A}_1, \ldots, \overline{A}_p$ the first p diagonal submatrix blocks (each of size $2q \times 2q$) and by $\overline{A}_s$ the last diagonal submatrix block (of size $s \times s$). Denote by B_i the vertical border submatrix blocks (each of size $2q \times s$) and by the horizontal border submatrix blocks $\overline{C}_i$ (each of size $s \times 2q$). Assume, for simplicity, that $n = 2pq + s$.

Algorithm (Block LU Factorization)

The parallel domain decomposition complete *LU* factorization is the following:

1. On every processor, do *LU* decomposition $\overline{A}_i = L_iU_i$, $i = 1, \ldots, p$
2. On every processor, solve the systems $\overline{A}_i\overline{Z}_i = \overline{B}_i$, $i = 1 \ldots, p$
3. On every processor, form $\overline{C}_i\overline{Z}_i$, $i = 1, \ldots, p$
4. On every processor, broadcast $\overline{C}_i\overline{Z}_i$ so that every processor has the data $\overline{C}_i\overline{Z}_i$, $i = 1, \ldots, p$
5. On all processors, form $\hat{A} = \overline{A}_s - \Sigma_{i=1}^{p} \overline{C}_i\overline{Z}_i$ and compute the *LU* decomposition of $\hat{A}$

Algorithm (Block LU Linear System Solution)

The parallel domain decomposition forward elimination and backsubstitution is the following:

1. On every processor, solve the system $\overline{A}_i\overline{z}_i = \overline{b}_i$, $i = 1 \ldots, p$
2. On every processor, form $\overline{C}_i\overline{z}_i$, $i = 1, \ldots, p$
3. On every processor, broadcast $\overline{C}_i\overline{z}_i$ so that every processor has the data $\overline{C}_i\overline{z}_i$, $i = 1, \ldots, p$
4. On all processors, form $\hat{b} = \overline{b}_s - \Sigma_{i=1}^{p} \overline{C}_i\overline{z}_i$
5. On all processors, solve the system $\hat{A}\overline{x}_s = \hat{b}$
6. On every processor, form $\overline{c}_i = \overline{b}_i - \overline{B}_i\overline{x}_s$, $i = 1, \ldots, p$
7. On every processor, solve the system $\overline{A}_i\overline{x}_i = \overline{c}_i$, $i = 1, \ldots, p$

ACKNOWLEDGMENTS

This work was funded in part by the Minnesota Department of Transportation (MNDOT) and in part by the National Science Foundation (CCR–9496327). The Sandia National Laboratory is acknowledged for providing access to its nCUBE2 parallel computer system. David Berg of MNDOT provided the traffic data.

REFERENCES

1. Leo, C. J., and R. L. Pretty. Numerical Simulation of Macroscopic Continuum Traffic Models. *Transportation Research*, Vol. 26B, No. 3, 1990, pp. 207–220.
2. Lighthill, M. H., and G. B. Witham. On Kinematic waves: II A Theory of Traffic Flow on Long Crowded Roads. *Proc., Royal Society, London*, Series A229, No. 1178, 1955, pp. 317–345.
3. Lyrintzis, A. S., et al. Continuum Modeling of Traffic Dynamics. *Proc., 2nd International Conference on Applications of Advanced Technologies in Transportation Engineering*, ASCE, Minneapolis, Minn., Aug., 1991, pp. 36–40.
4. Michalopoulos, P. G., P. Yi, and A. S. Lyrintzis. Development of an Improved High Order Continuum Traffic Flow Model. In *Transportation Research Record 1365*, TRB, National Research Council, Washington, D.C., 1992, pp. 125–132.
5. Payne, H. J. FREFLO: A Macroscopic Simulation Model of Freeway Traffic. In *Transportation Research Record 722*, TRB, National Research Council, Washington, D.C., 1979, pp. 68–75.
6. Mikhailov, L., and R. Hanus. Hierarchical Control of Congested Urban Traffic—Mathematical Modeling and Simulation. *Mathematics and Computers in Simulation*, Vol. 37, 1994, pp. 183–188.
7. Gerlough, D. L., and M. J. Huber. *Special Report 165: Traffic Flow Theory.* TRB, National Research Council, Washington, D.C., 1975.
8. Chronopoulos, A. T., et al. Traffic Flow Simulation Through High Order Traffic Modelling. *Mathematical Computing Modelling*, Vol. 17, No. 8, 1993, pp. 11–22.
9. Kim, S. K., and A. T. Chronopoulos. A Class of Lanczos-like Algorithms Implemented on Parallel Computers. *Parallel Computing*, Vol. 17, 1991, pp. 763–778.
10. Chronopoulos, A. T., and C. Pedro. Iterative Methods for Nonsymmetric Systems in DAEs and Stiff ODEs Codes. *Mathematics and Computers in Simulation*, Vol. 35, 1993, pp. 211–232.
11. Ortega, J. M. *Introduction to Parallel and Vector Solution of Linear Systems.* Plenum Publishing Company, 1988.
12. McBryan, O. A., and E. F. Van Der Velde. Matrix and Vector Operations on Hypercube Parallel Processors. *Parallel Computing*, Vol. 5, 1987, pp. 117–125.
13. Ma, S., and A. T. Chronopoulos. Implementation of Iterative Methods for Large Sparse Nonsymmetric Systems on Parallel Vector Computers. *International Journal of Supercomputer Applications*, Vol. 4, 1990, pp. 9–24.
14. Di Brozolo, G. R., and Y. Robert. Parallel Conjugate Gradient-like Algorithms for Sparse Nonsymmetric Systems on a Vector Multiprocessor. *Parallel Computing*, Vol. 11, 1989, pp. 223–239.
15. Rodrigue, G. Domain Decomposition: A Unified Approach for Solving Fluid Mechanics Problems on Parallel Computers. In *Parallel Processing in Computational Mechanics* (H. Adeli, ed.), Dekker, 1991, pp. 297–330.
16. Meurant, G. Domain Decomposition Methods for Solving Large Sparse Linear Systems. In *Computer Algorithms for Solving Linear Algebraic Equations: The State of the Art* (E. Spedicato, ed.), NATO ASI Series, Series F: Computer and Systems Sciences, Vol. 77, Springer-Verlag, 1991, pp. 185–206.