ELSEVIER

Parallel Computing 22 (1997) 1965–1983

PARALLEL COMPUTING

Practical aspects and experiences

Parallel solution of a traffic flow simulation problem [1]

Anthony Theodore Chronopoulos [*,a], Gang Wang [b]

[a] Department of Computer Science, Wayne State University, 5143 Cass Avenue, Detroit, MI 48202, USA
[b] University of Minnesota, Minneapolis, USA

Received 23 May 1995; revised 3 January 1996, 23 September 1997

Abstract

Computational Fluid Dynamics (CFD) methods for solving traffic flow continuum models have been studied and efficiently implemented in traffic simulation codes in the past. This is the first time that such methods are studied from the point of view of parallel computing. We studied and implemented an implicit numerical method for solving the high-order flow conservation traffic model on parallel computers. Implicit methods allow much larger time-step than explicit methods, for the same accuracy. However, at each time-step a nonlinear system must be solved. We used the Newton method coupled with a linear iterative method (*Orthomin*). We accelerated the convergence of Orthomin with parallel incomplete LU factorization preconditionings. We ran simulation tests with real traffic data from an 12-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.

Keywords: Computational fluid dynamics; Traffic flow simulation; Newton method; LU factorization; Preconditioning; nCUBE2

1. Introduction

Macroscopic continuum traffic flow models based on traffic flow *volume*, *density* and *speed* have been used in studying the behavior of the freeway traffic in the past; see

[*] Corresponding author. Email: chronos@cs.wayne.edu

[1] This work was in part funded by NSF under grant CCR-9496327. The Sandia National Laboratory, P.O. Box 5800, Albuquerque, NM 87185 is acknowledged for providing access to its nCUBE2 Parallel Computer System.

PII S0167-8191(97)00070-1

for example [15–17,27,25,22]. These models involve partial differential equations (PDEs) defined on appropriate (road) domains with suitable boundary conditions, which describe various traffic phenomena and road geometries. 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 [17,7]. Implicit methods allow much larger time-step than explicit methods, for the same accuracy. Many results exist on study of algorithms for solving linear systems and mapping of algorithms on parallel computers; see for example [10,9,29,30,31].

In this work we parallelize an implicit numerical method (Backward Euler) to solve the momentum conservation model on the nCUBE2 parallel computer. Implicit numerical methods require the solution of (non)linear systems of equations at each time-step. We use the Newton method coupled with a linear iterative method (Orthomin). We accelerate the convergence of Orthomin with two different types of parallel incomplete LU factorization preconditionings. The first preconditioning is based on the vectorizable incomplete LU method (see [3]). The second preconditioning is based on domain decomposition methods (see [1,4,5,9,12,19,21,24,26,28]).

We wrote a code (in C) simulating a freeway with multiple entry/exit traffic flow on the nCUBE2 parallel computer located at the nCUBE2 located at Sandia National Laboratory. Tests with real data from the I-494 freeways in Minneapolis were conducted. On the nCUBE2, the parallel method, on a 16-processor configuration, runs about 4 times faster than on the 2-processor configuration.

The outline of the article is as follows. In Section 2, a traffic flow model is described. In Section 3, the parallel implementation of the traffic model is discussed. In Section 4, the test results are shown.

2. A traffic flow model

In this section, we outline a *traffic flow model*. Such a model is based on a *continuum traffic flow model*, a *discrete model* and a *freeway model*.

2.1. A continuum model

Lighthill and Whitham [16] first proposed the following *simple continuum conservation equation model* for the traffic flow problem.

$$\frac{\partial k}{\partial t} + \frac{\partial q}{\partial x} = g(x, t), \tag{1}$$

where $k(x, t)$ and $q(x, t)$ are the traffic density and flow respectively at the space-time point (x, t). The generation term $g(x, t)$ represents the number of cars entering or leaving the traffic flow in a freeway with entries/exits. The traffic flow, density and speed are related by the equation

$$q = ku, \tag{2}$$

where the equilibrium speed $u_e(x, t) = u(k)$ must be provided by a theoretical or empirical$u$–k model. The theoretical u–k model, can take the general form

$$u_e = u_f\left[1 - \left(k/k_{\text{jam}}\right)^{\alpha}\right]^{\beta}, \tag{3}$$

where u_f is the *free flow speed* and k_{jam} the *jam density* model parameters. More information on this and other forms of the u–k relationships can be found elsewhere (see [6] and the references therein).

Since the simple continuum model does not consider acceleration and inertia effects, it does not describe accurately non-equilibrium traffic flow dynamics. *High-order continuum traffic models* that include the momentum equation have also been developed. Such a model is the *semi-viscous model* [27] (*SVM*). This mathematical model for the traffic flow is adopted for our research. Our main goal is to derive efficient methods to parallelize the implicit methods applied to SVM. This choice is not a restriction. The same parallel implicit methods can be applied to other continuum models (e.g. [16,25]).

SVM takes into account acceleration and inertia effects by replacing Eq. (3) with a momentum equation. For example the equation in [27] has the following form,

$$\frac{du}{dt} = \frac{1}{T}\left[u_f(x) - u\right] - \nu k^{\beta}\frac{\partial k}{\partial x}, \tag{4}$$

where du/dt is the acceleration of an observer moving with the traffic stream and is related to the acceleration $\partial u/\partial t$ of the traffic stream as seen by an observer at a fixed point of the road, i.e.

$$\frac{du}{dt} = \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x}. \tag{5}$$

The *first term* on the right-hand side of (4), $T^{-1}[u_f(x) - u]$, represents the relaxation term, the tendency of traffic flow to adjust speeds due to changes in free-flow speed $u_f(x)$ along the roadway, where relaxation time T is assumed to vary with density k according to

$$T = t_0\left(1 + \frac{rk}{k_{\text{jam}} - rk}\right), \tag{6}$$

where $t_0 > 0$ and $0 < r < 1$ are model parameters. The *second term* on the right-hand side of (4), $\nu k^{\beta}(\partial k/\partial x)$, represents the anticipation term which is the effect of drivers reacting to downstream traffic conditions. In this term ν is the anticipation parameter. As implied in this example, if downstream density is higher due to congestion, speed has to be decreased accordingly. Conversely, if downstream density is lower, speed can be increased. From Eqs. (4)–(6) one derives a momentum model for the traffic flow described by the following system of PDEs,

$$\frac{\partial \boldsymbol{U}}{\partial t}+\frac{\partial \boldsymbol{E}}{\partial x}=\boldsymbol{Z}, \tag{7}$$

where $\boldsymbol{U}$, $\boldsymbol{E}$, and $\boldsymbol{Z}$ are the following vectors:

$$\boldsymbol{U}=\begin{pmatrix} k \\ q \end{pmatrix},\quad \boldsymbol{E}=\begin{pmatrix} ku \\ u^2k+\dfrac{\nu}{\beta+2}k^{\beta+2} \end{pmatrix},\quad \boldsymbol{Z}=\begin{pmatrix} g \\ \dfrac{k}{T}\left[u_{\mathrm{f}}(x)-u\right] \end{pmatrix}.$$

A typical choice of parameters is: $u_{\mathrm{f}}=60$, $k_{\mathrm{jam}}=180$, $\beta=-1$, $\nu=180$, $t_0=50$, $r=0.80$. These parameters depend on the geometry of the freeway but also on the time of the day and even on the weather conditions.

We note that the momentum conservation model does not require a $q-k$ curve as in the case of the simple continuum model. However, speed data are required for the boundary conditions. If speed data are not available from the real traffic data measurements then a $q-k$ curve based on *occupancy* data [27] is used to generate the speed data.

2.2. A discrete model

We now apply an implicit numerical method (implicit Euler), which is used in computational fluid dynamics [11] to discretize SVM. For each traffic model the road section (the space dimension) is discretized using a uniform mesh. This method provides an efficient alternative to the explicit methods (e.g. Lax or Upwind) in integrating the model equations [7]. Let Δt and Δx be the time and space mesh sizes. We use the following notation:

k_j^i = density (vehicles/mile/lane) at space node $j\Delta x$ and at time $i\Delta t$,
q_j^i = flow (vehicles/hour/lane) at space node $j\Delta x$ and at time $i\Delta t$,
u_j^i = speed (mile/hour) at space node $j\Delta x$ and at time $i\Delta t$.

The implicit Euler method applied to the nonlinear PDEs (1) generates, at each time-step, a nonlinear system involving all space nodes. Newton's method is applied to solve numerically this system [11]. Each Newton step constructs a *linear system* $A\Delta U_j = rhs$, where the Jacobian A is a *narrowly banded coefficient matrix*, the right-hand side vector is *rhs* and the vector of unknowns is $\Delta \boldsymbol{U}_j=\boldsymbol{U}_j^{i+1}-\boldsymbol{U}_j^i$. This linear system has the following form,

$$-\frac{\Delta t}{2\Delta x}\left(\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{U}}\right)_{j-1}^{i}\Delta \boldsymbol{U}_{j-1}+\left[1-\Delta t\left(\frac{\partial \boldsymbol{Z}}{\partial \boldsymbol{U}}\right)_j^i\right]\Delta \boldsymbol{U}_j+\frac{\Delta t}{2\Delta x}\left(\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{U}}\right)_{j+1}^{i}\Delta \boldsymbol{U}_{j+1}$$

$$=-\frac{\Delta t}{2\Delta x}\left(\boldsymbol{E}_{j+1}^i-\boldsymbol{E}_{j-1}^i\right)+\Delta t\boldsymbol{Z}_j^i+\Delta t\left(\frac{\partial \boldsymbol{Z}}{\partial \boldsymbol{U}}\right)_j^i\boldsymbol{U}_j^i,$$

where

$$\left(\frac{\partial E}{\partial U}\right)_j^i = \begin{pmatrix} 0 & 1 \\ -u^2 + \nu k^{\beta+1} & 2u \end{pmatrix}_j^i$$

and

$$\left(\frac{\partial Z}{\partial U}\right)_j^i = \begin{pmatrix} 0 & 0 \\ \frac{u_f}{t_0 k_{\text{jam}}}\left(k_{\text{jam}} - 2rk\right) + \frac{r}{t_0 k_{\text{jam}}} q & -\frac{k_{\text{jam}} - rk}{t_0 k_{\text{jam}}} \end{pmatrix}_j^i .$$

Let m be the number of interior space nodes in a freeway lane (space domain). The Jacobian A has dimension $n = 2m$ and it is a *block tridiagonal* matrix with each submatrix block of size 2×2. A has the form *tridiag*$[C_{j-1}, D_j, B_{j+1}]$, where $C_0 = B_{m+1} = 0$ and for $1 \leqslant j \leqslant m$ we have

$$B_j = \begin{pmatrix} 0 & \frac{\Delta t}{2\Delta x} \\ \frac{\Delta t}{2\Delta x}\left(-u^2 + \nu\right) & \frac{\Delta t}{\Delta x}\frac{q}{k} \end{pmatrix}_j^i, \quad C_j = \begin{pmatrix} 0 & -\frac{\Delta t}{2\Delta x} \\ -\frac{\Delta t}{2\Delta x}\left(-u^2 + \nu\right) & -\frac{\Delta t}{\Delta x}\frac{q}{k} \end{pmatrix}_j^i,$$

$$D_j = \begin{pmatrix} 1 & 0 \\ -\frac{u_f \Delta t}{t_0 k_{\text{jam}}}\left(k_{\text{jam}} - 2rk\right) - \frac{r\Delta t}{t_0 k_{\text{jam}}} q & 1 + \Delta t\left(\frac{k_{\text{jam}} - rk}{t_0 k_{\text{jam}}}\right) \end{pmatrix}_j^i .$$

The Jacobian A is also banded with *upper / lower bandwidth* equal to 3.

This linear system is solved by *preconditioned Orthomin*(*korth*) (see Appendix A), which is an iterative method for linear systems. The solution is then advanced to the next time-step simultaneously at all space nodes by computing $U_j^{i+1} = U_j^i + \Delta U_j$. At each time-step, artificial smoothing is added to reduce oscillatory behavior in the numerical solution [11]. This is achieved by adding to each term U_j the following fourth order damping term

$$-\frac{\omega}{8}\left(U_{j-2} - 4U_{j-1} + 6U_j - 4U_{j+1} + U_{j+2}\right).$$

We have tested several damping coefficients with ω ranging from $\omega = 0$ (no damping) to $\omega = 1$. The choice $\omega = 1$ gave the best results.

In Orthomin(*korth*), we choose *korth* $= 4$, because it gives the fewest number of iterations for convergence of Orthomin(*korth*), in our tests. Each iteration of Orthomin consists of vector operations of the following types: *linear combinations*, *dotproducts*, *one matrix times vector product* and *one preconditioning step* [18]. While A is computed at each time-step, its LU factorization is stored and used over many Newton/time steps. We recompute LU, if the nonlinear residual exceeds the *error tolerance* (ε) after a *maximum number of iterations* (*maxiter*) of Orthomin(4).

Let us define a *flop* to be a multiplication combined with/without an addition. We next give the serial flops count for the various computations of the discrete model.

1. Compute A: $3n$
2. Compute *rhs*: $10n$
3. Compute the banded LU factorization of A: $9n$
4. Compute and apply artificial smoothing: $4n$
5. Apply the preconditioned Orthomin(4) (per inner iteration cost):
   (a) Compute matrix (A) times vector: $7n$
   (b) Compute dotproducts: $7n$
   (c) Compute linear combinations: $10n$
   (d) Apply preconditioning step (i.e. forward substitution and backward elimination using L and U): $7n$

The implicit Euler method is of second order accuracy with respect to Δt and it is unconditionally stable. This method will be called the *Euler–Momentum* method. This method involves more computations per time-step than an explicit method. However, the implicit Euler method allows much larger step-sizes, which makes the overall (serial) computer execution time faster than an explicit method, see [7].

2.3. A freeway model

We have used two schemes to add/subtract entry/exit (ramp) traffic volumes to/from the main-lane traffic volume in SVM.

(1) Point entry/exit scheme: 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 scheme: This is used when the ramp is directly connected to another freeway and it is explained in more detail below.

The weaving scheme is outlined as follows. In the following discussion, let us consider the traffic flow volume in a freeway section shown in Fig. 1 at a fixed discrete time. In Fig. 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 $q_F = v_2 + v_4$

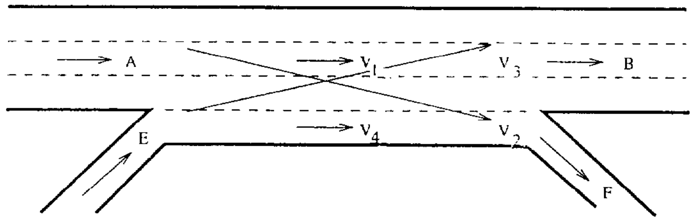

Fig. 1. Weaving flows in a freeway.

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 600 ft (this is an empirical choice), 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

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

3. Parallel implementation on the nCUBE2

In this section we outline the basic features of nCUBE2 and discuss the parallelization of Euler-Momentum.

3.1. The nCUBE2

The nCUBE2 [23] is an *MIMD* (Multiple Instruction Multiple Data) *hypercube* parallel computer system. In a hypercube of dimension *ndim*, there are $p = 2^{ndim}$ processors. These processors are labeled by $0, 1, \ldots, p-1$. Two processors P_{j_1} and P_{j_1} are directly connected iff the binary representations of j_1 and j_2 differ in exactly one bit. For more details on the hypercube routing functions and mapping (onto a hypercube) of basic computations see [14].

The number of processors to be active is chosen by the user, but must be a power of 2. In Table 1, we show a summary of inter-processor communication times for neighbor processors and the basic floating point operation times [13]. We see 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 the communication steps.

Table 1
Computation and communication times on the nCUBE2

Operation	Time	Comm. comp.
8 byte transfer	111 μsec	–
8 byte add	1.23 μsec	90
8 byte multiply	1.28 μsec	86

3.2. Mapping of the problem and parallelization of Orthomin

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* $\mathrm{Seg}_0, \ldots, \mathrm{Seg}_{p-1}$ and assigning each segment to the processors $P_{j_0}, \ldots, P_{j_{p-1}}$. The choice of indices $j_0, \ldots, j_{p-1}$ defines a 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 computation in 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* code mapping (for a description see [14]).

The computation of the Jacobian A and right-hand side vector *rhs* require data which may be located in an adjacent segment. Due to the Gray code 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 (see [2,8]). The Jacobian and the right-hand side 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. Thus, we now concentrate our discussion on the parallel implementation of the Orthomin method to solve the linear system. We have four main computations to be parallelized: *a linear combination*, *a dotproduct*, *a matrix times vector product* and *a preconditioning step*. We will only explain how the matrix times vector and the preconditioning are parallelized. The parallelization of the dotproducts and linear combination is studied in [13,20].

Let $n = 2pq + s$ be the dimension of the linear system, where $1 \leqslant q$ and $s = 2(p-1)$. The special choice of n is not really a restriction and it is made for discussing the parallelization of the domain decomposition preconditioning. We distribute the matrix A and *rhs* data row-wise to the p processors so that each processor gets $2q+2$ rows, except for the last (in the Gray code mapping) processor, which gets $2q$ rows. Fig. 2 shows an example of the matrix of coefficients and the vector of unknowns for *maxj* = 11 (number of space nodes), or $n = 22$ (dimension of matrix and right-hand side vector), which is assigned to four processors of the nCUBE.

Let the entries (with subscripts omitted for simplicity of notation) of the matrix (shown in Fig. 2) be as follows:

$$a = 1 + \Delta t \frac{k_{\mathrm{jam}} - rk}{t_0 k_{\mathrm{jam}}}, \qquad b = -\Delta t\left[\frac{u_{\mathrm{f}}}{t_0 k_{\mathrm{jam}}}\left(k_{\mathrm{jam}} - 2rk\right) + \frac{r}{t_0 k_{\mathrm{jam}}} q\right],$$

$$c = \frac{\Delta t}{2\Delta x}\left[-u^2 + \nu\right], \qquad d = \frac{\Delta t}{\Delta x}\frac{q}{k}, \qquad e = -\frac{\Delta t}{\Delta x}\frac{q}{k},$$

$$f = -\frac{\Delta t}{2\Delta x}\left[-u^2 + \nu\right], \qquad h = \frac{\Delta t}{2\Delta x}.$$

3.3. Parallelization of the preconditioning

Preconditioners are used to accelerate the convergence of iterative methods. In determining a preconditioner (matrix) P_r, we look for a matrix that is a close approximation to the inverse of A, so that

$$P_r \approx A^{-1} \tag{8}$$

or AP_r has clustered eigenvalues. The matrix P_r must be computed easily. 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 and lower triangular matrices U and L such that $A = LU + \Theta$, where $P_r^{-1} = LU$ and Θ is an error matrix. Also, we assume that if the entry A_{j_1,j_2} is zero, then both U_{j_1,j_2} and $L_{j_1,j_2} = 0$. In other words, L and U have the same sparsity patterns as A. This is the $ILU(0)$ preconditioning method. ILU(0) must be implemented properly on a parallel computer, in order to maintain its effectiveness (see [3,18]). We consider two different parallel implementations of ILU preconditioning:

(a) Overlapped Submatrix Regions (OSR ILU) by Radicati di Brozolo and Robert in [3].

(b) The Domain Decomposition (DD ILU) [24].

In method (a) the matrix is partitioned into a number (p, equal to the number of PEs) of overlapping horizontal *submatrix regions* (see [3]). We consider overlapped regions consisting of 2 matrix rows. The L and U factors are computed independently for each region. When the ILU step is applied the (forward elimination)/back-substitution is computed independently for each region and then the average of the two results is

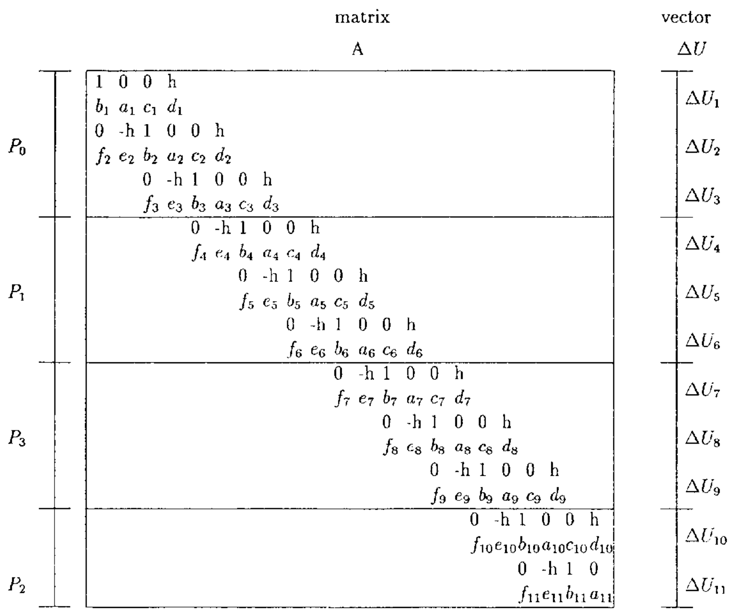

Fig. 2. Original structure of the matrix A and the vector ΔU.

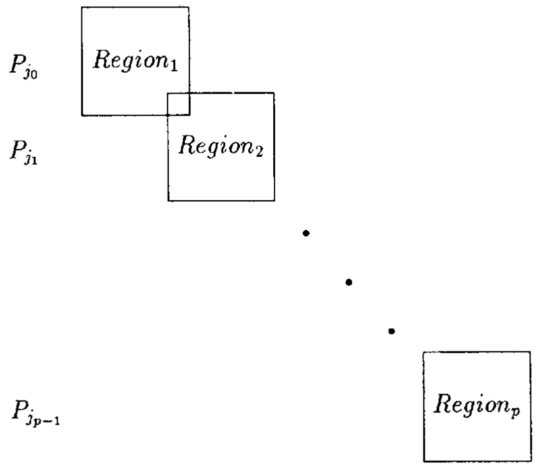

Fig. 3. Processor assignment for OSR ILU.

computed for the overlapped regions. More details on (a) are in [3] or [18]. Fig. 3 illustrates the OSR ILU matrix mapping onto the hypercube.

Method (b) is based on domain decomposition. The space domain is partitioned into p disjoint subdomains. The grid nodes of each subdomains and the corresponding

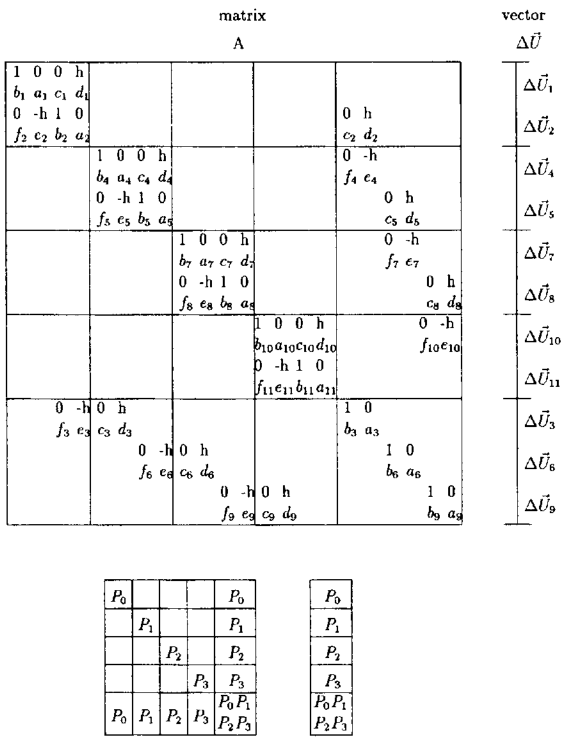

Fig. 4. Partition of the linear system and mapping to processors in DD ILU.

unknowns are mapped to each processor via the Gray code mapping. The set of $(p-1)$ grid nodes, which separate the subdomains define the *separator set*. Let us assume (for simplicity) that each subdomain consists of q nodes. Then A and *rhs* have dimension $n = 2pq + 2(p-1)$. Fig. 4 shows the partitioning of the matrix/(vector of unknowns) (according to the domain decomposition) across the $(p = 4)$ processors of a hypercube, with $q = 2$. After a reordering of the unknowns we move the separator set unknowns to the last $2(p-1)$ rows of the unknowns' vector. If $p \ll q$, computing the separator unknowns involves much less cost than computing each of the p other sets of (q) unknowns. Thus, we assign this task to all the processors in order to reduce the communication cost. The parallel LU, based on domain decomposition, is outlined in the Appendix B (see [24]).

The parallel implementation introduces flops overhead only in the preconditioning step. For OSR ILU it is due to the overlapped submatrix regions computations. Since each such region only consists of 2 rows this overhead is of order p. The flops overhead for DD ILU is due to the Schur complement computations, which is of order p^3. In both cases the overhead is negligible, if $p \ll q$. For large p, the computations in the Schur complement must be distributed across the processors of the system. Also, it may be faster to solve the Schur complement iteratively.

4. Results

We have implemented the Euler–Momentum method on the nCUBE2 parallel computer located at the Sandia National Laboratory.

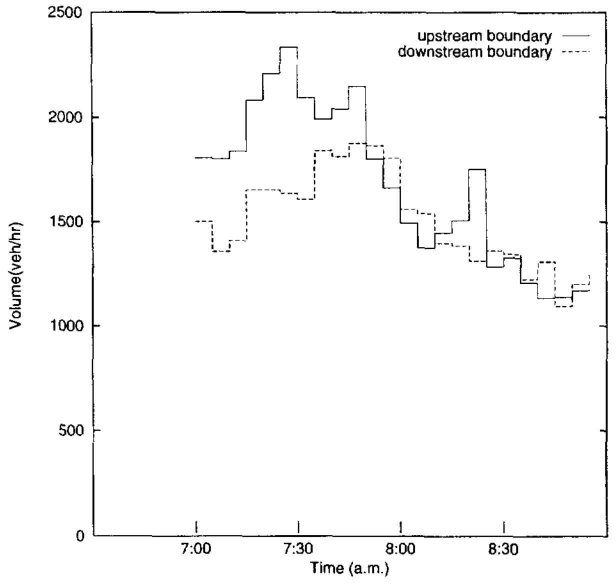

Fig. 5. Boundary conditions input data.

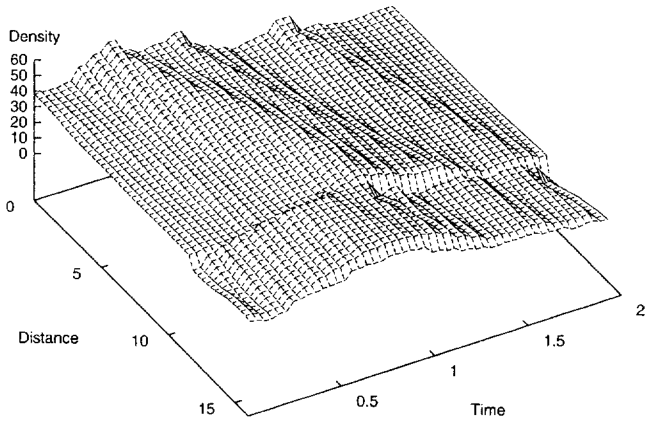

Fig. 6. Density (cars/miles) vs. (time: hours, distance: miles).

As our freeway test-site, we considered Eastbound I-494, which is a multiple entry/exit freeway section in the Minneapolis, Minnesota freeway network. The Eastbound I-494 section extends from the Carlson Pwy to Portland Avenue. We simulated the traffic in a 12-mile stretch of the freeway, which has 12 entry and 10 exit ramps, for a period of 2 hours. Simulation input data are available from measurements collected by the Minnesota Department of Transportation. These data contain 5-minute measurements of the traffic volume at the entry/exit ramps, the upstream/downstream boundaries of the freeway segment and at a number ($nst = 18$) of intermediate check-station sites. Fig. 5 shows the upstream/downstream volume input data for the period of simulation (2 hours). The rest of the input data are not shown but could be plotted similarly.

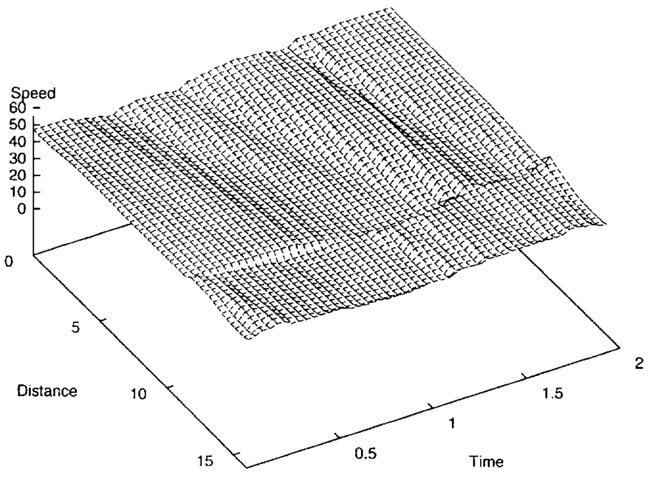

Fig. 7. Speed (miles/hours) vs. (time: hours, distance: miles).

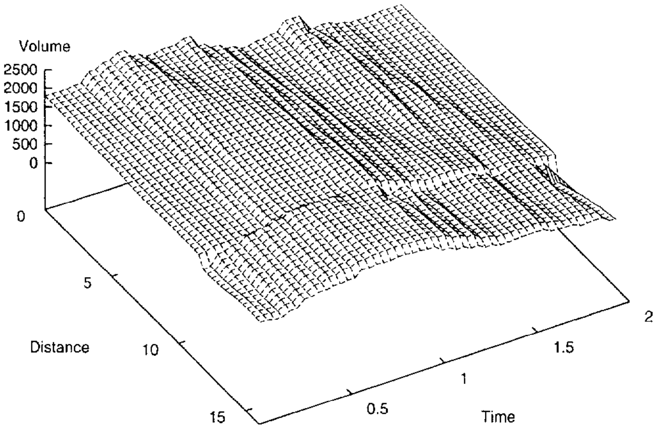

Fig. 8. Volume (cars/hour/lane) vs. (time: hours, distance:miles).

To test the Euler–Momentum, the time-step selection was made as follows. The time-step was increased so that the maximum error did not exceed that of the errors reported in past research (see [7,17,27]), for explicit methods. After testing we selected $\varepsilon = 10^{-6}$ and *maxiter* = 40, for Orthomin(4). The time and space mesh sizes were $\Delta t = 10$ sec and $\Delta x = 200$ ft. There are 313 interior space nodes in the discrete model. The dimension of A is $n = 626$ and 720 time-steps are taken for the 2 hour long simulation.

Figs. 6–8 show the computed density, speed and volume plotted versus the space domain and the time. We compare volume and speed data computed by the simulations with the check-station sites' data. Let N be the number of discrete time points at which real traffic flow data are collected. We use the following error function to measure the effectiveness of the simulation in comparison with actual data:

$$\textit{Relative error with 2-norm} = \left[\frac{\sum_{i=1}^{N} (\textit{Observed}_i - \textit{Simulated}_i)\,u2}{\sum_{i=1}^{N} \textit{Observed}_i^2} \right]^{1/2}. \tag{9}$$

The error statistics are summarized in Table 2. The relative errors are about 10% for the

Table 2
Errors for traffic volume and speed (I-494)

Sites	OSR ILU		DD ILU	
	Volume (veh/5 min)	Speed (miles/hour)	Volume (veh/5 min)	Speed (miles/hour)
1	0.12	0.03	0.12	0.03
2	0.11	0.07	0.13	0.08
3	0.09	0.08	0.13	0.08
4	0.05	0.07	0.16	0.07
5	0.11	0.13	0.17	0.11

Table 3
Time (in seconds) in OSR ILU version for I-494

No. of PEs	Matrix/rhs	ILU	Dotproducts	Matrix × vec.	Lin. Comb.	Total Time
2	32.0	111.4	95.5	75.0	93.4	483.1
4	16.9	66.7	71.2	45.4	54.0	296.2
8	9.2	36.8	53.9	26.3	27.7	177.8
16	4.4	23.8	57.0	17.2	22.1	133.6

Table 4
Time (in seconds) in DD ILU version for I-494

No. of PEs	Matrix/*rhs*	ILU	Dotproduct	Matrix × vec.	Lin. Comb.	Total Time
2	32.2	202.1	82.2	67.3	148.0	536.3
4	17.8	95.0	43.5	31.0	59.1	247.5
8	12.2	68.8	27.8	15.0	24.5	152.2
16	13.1	102.6	24.8	8.6	11.8	168.6

volume but they are lower for the speed. This accuracy range of agreement with the measured data is acceptable (see [7,17,27]). We consider the relative speedup:

$$S_p = \frac{T_2}{T_p},$$

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

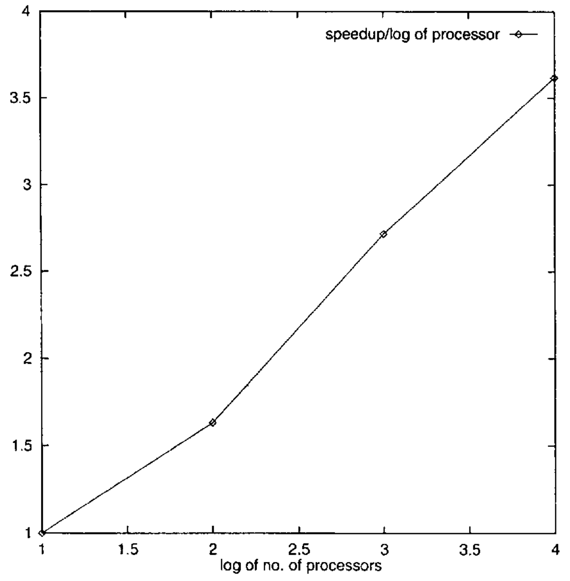

Fig. 9. Speedup for the OSR ILU method on nCUBE2.

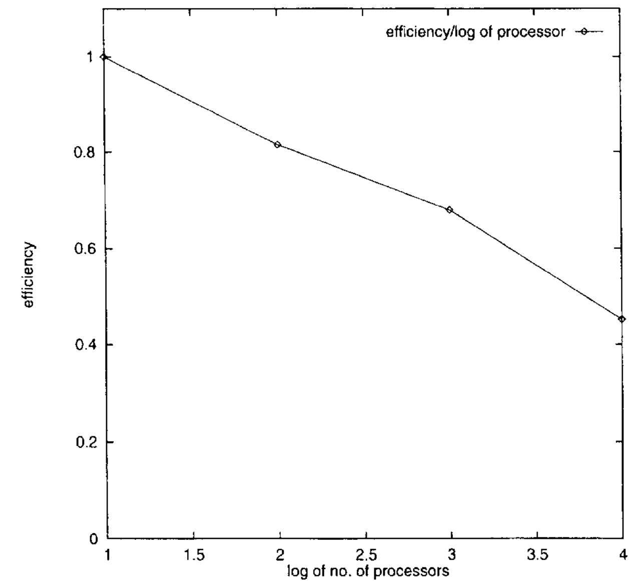

Fig. 10. Efficiency for the OSR ILU method on nCUBE2.

designed to run on more than one processor and thus we could not obtain the timing on a single processor.

Parallel execution times (on nCUBE2), for OSR ILU and DD ILU, are summarized in Tables 3 and 4. The speedups and efficiencies on nCUBE2 are summarized in Figs.

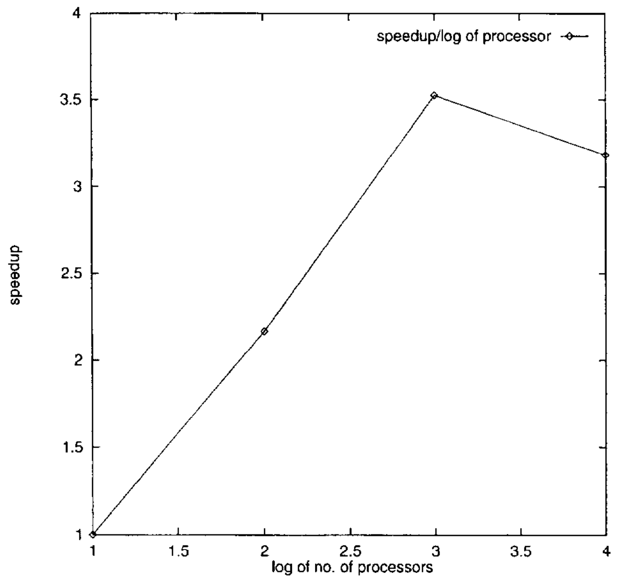

Fig. 11. Speedup for the DD ILU method on nCUBE2.

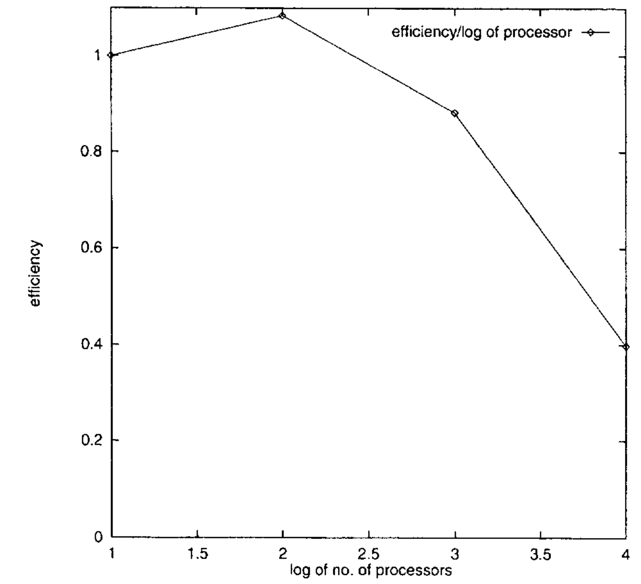

Fig. 12. Efficiency for the DD ILU method on nCUBE2.

9–12. From Tables 3 and 4, the times for the DD ILU parts (Matrix/rhs/ILU/Dotproducts) are greater than the corresponding OSR ILU parts. This is due, in part, to the fact that these parts require more communication in DD ILU than the in OSR ILU. Also, due mainly to the communication costs, in setting up the problem, the time on the 2 processors is so long that it leads to a superlinear speedup for the 4-processor case.

Since the size of the problem is kept fixed the speedup drops as the problem is solved with more processors. Overall, the performance of the parallel implicit methods, studied here, is satisfactory and it is expected to improve for simulation of longer freeway segments and networks of freeways. At the present time the input data available are for single freeway sections.

Acknowledgement

The comments of the anonymous referees, which helped improve the presentation of some parts of the paper are gratefully acknowledged. We are grateful to Mr David Berg of Minnesota Department of Transportation for providing the traffic measurements data.

Appendix A. The Orthomin iterative method

We next describe the Orthomin iterative method to solve $Ax = rhs$, where A is nonsymmetric matrix of dimension n. Orthomin applies to nonsymmetric linear systems with the symmetric part of A being positive definite. Let *korth* be a positive integer. We describe the orthomin iterative method with preconditioning as follows. In this algorithm, $j_i = \max(0, i - korth + 1)$, P_r is the right preconditioner, which is obtained by the LU decomposition of the matrix A. For more details, see [18].

Algorithm 1. Orthomin (*korth*)

```
1. Choose $x_0 = 0$.
2. Compute $r_0 = rhs - Ax_0$.
3. $p_0 = r_0$.
   For i = 0 step 1 Until convergence Do
4. $a_i = (r_i Ap_i)/(Ap_i Ap_i)$
5. $x_{i+1} = x_i + a_i p_i$
6. $r_{i+1} = r_i - a_i Ap_i$
7. If $\| r_{i+1} \| <$ tolerance, then stop
8. Compute $AP_r r_{i+1}$
9. $b_j^i = (AP_r r_{i+1} Ap_j)/(Ap_j Ap_j)$, $j_i \leqslant j \leqslant i$
10. $p_{i+1} = P_r r_{i+1} + \Sigma_{j=j_i}^{i} b_j^i p_j$
11. $Ap_{i+1} = AP_r r_{i+1} + \Sigma_{j=j_i}^{i} b_j^i Ap_j$
   Endfor
```

Appendix B. Domain decomposition

We will outline the parallel algorithms for the complete *LU* factorization of a matrix and the solution of a linear system using the domain decomposition method (see [28,24]). In our implementation, we obtain the LU factors only once and we use them to solve the linear systems of the *preconditioning step* in the Orthomin iterations, over several Newton/time steps. This is a domain decomposition ILU(0) type method (see [21]).

Let us consider the partition of the linear system in Fig. 4. Let us assume, for simplicity, that $n = 2pq + s$ and let $s = 2(p-1)$ be the number of unknowns in the separator set. Let us denote by $\bar{A}_1, \ldots, \bar{A}_p$ the first p diagonal submatrix blocks (each of size $2q \times 2q$) and by $\bar{A}_s$ the last diagonal submatrix block (of size $s \times s$, which is related to the separator set unknowns). Let us denote by $\bar{B}_i$ the vertical border submatrix blocks (each of size $2q \times s$) and by the horizontal border submatrix blocks $\bar{C}_i$ (each of size $s \times 2q$).

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

Algorithm (Block LU Factorization)

1. on every processor, compute the LU decomposition $\bar{A}_i = L_i U_i$, $i = 1, \ldots, p$.
2. on every processor, solve the system $\bar{A}_i \bar{Z}_i = \bar{B}_i$, $i = 1, \ldots, p$.
3. on every processor, form $\bar{C}_i \bar{Z}_i$, $i = 1, \ldots, p$.
4. on every processor, broadcast $\bar{C}_i \bar{Z}_i$ so that every processor has the data $\bar{C}_i \bar{Z}_i$, $i = 1, \ldots, p$.
5. on all processors, form the Schur Complement $\hat{A} = \bar{A}_s - \Sigma_{i=1}^{p} \bar{C}_i \bar{Z}_i$ and compute its LU decomposition.

The parallel domain decomposition forward elimination and back-substitution is the following.

Algorithm (Block LU Linear System Solution)

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

References

[1] A. Brambilla, C. Carlenzoli, G. Gazzaniga, P. Gervasio and G. Sacchi, Implementation of domain decomposition techniques on the nCUBE2 parallel machine, in: A. Quarteroni, J. Periaux, Y.A. Kuznetsov and O.B. Widlund, eds., *Domain Decomposition Methods in Science and Engineering*, Contemporary Mathematics, Vol. 157 (American Mathematical Society, 1994) 345–351.
[2] P.N. Brown and A.C. Hindmarsh, Reduced storage matrix methods in stiff ODE systems, *Appl. Math. Comput.* 31 (1989) 40–91.
[3] G.R. Di Brozolo and Y. Robert, Parallel conjugate gradient-like algorithms for sparse nonsymmetric systems on a vector multiprocessor, *Parallel Computing* 11 (1989) 223–239.
[4] T.F. Chan and T.P. Mathew, Domain decomposition algorithms, *Acta Numerica* (1994) 61–143.
[5] P.E. Bjorstad and O. Widlund, Iterative methods for the solution of elliptic problems on regions partitioned into substructures, *SIAM J. Numer. Anal.* 23 (6) (1986) 1097–1120.
[6] A.T. Chronopoulos, P. Michalopoulos and J. Donohoe, Efficient traffic flow simulation computations, *Math. Comput. Modelling* 16 (5) (1992) 107–120.
[7] A.T. Chronopoulos et al., Traffic flow simulation through high order traffic modelling, *Math. Comput. Modelling* 17 (8) (1993) 11–22.
[8] A.T. Chronopoulos and C. Pedro, Iterative methods for nonsymmetric systems in DAEs and stiff ODEs codes, *Math. Comput. Simulation* 35 (1993) 211–232.
[9] J.J. Dongarra and R.E. Hiromoto, A collection of parallel linear equations routines for the Denelcor HEP, *Parallel Computing* 1 (1984) 133–142.
[10] J. Gustafson, G. Montry and R. Benner, Development of parallel methods for a 1024-processor hypercube, *SIAM J. Sci. Stat. Comput.* 9 (1988) 609–638.
[11] C. Hirsch, *Numerical Computation of Internal and External Flows*, Vol. 2 (Wiley, 1988).
[12] D.E. Keyes and W.D. Gropp, A comparison of domain decomposition techniques for elliptic partial differential equations and their parallel implementation, *SIAM J. Sci. Statist. Comput.* 8 (2) (1987) 166–202.
[13] S.K. Kim and A.T. Chronopoulos, A class of Lanczos-like algorithms implemented on parallel computers, *Parallel Computing* 17 (1991) 763–778.
[14] V. Kumar et al., *Introduction to Parallel Computing Design and Analysis of Algorithms* (Benjamin/Cummings, 1994).
[15] C.J. Leo and R.L. Pretty, Numerical simulation of macroscopic continuum traffic models, *Transportation Research* 26B (3) (1990) 207–220.
[16] M.H. Lighthill and G.B. Witham, On kinematic waves: II A theory of traffic flow on long crowded roads, *Proc. Royal Soc. Ser. A* 229 (1178) (1955) 317–345.

[17] A.S. Lyrintzis et al., Continuum modeling of traffic dynamics, in: *Proc. 2nd Internat. Conf. on Applications of Advanced Technology in Transportation Engineering*, ASCE, Minneapolis, MN (1991) 36–40.

[18] S. Ma and A.T. Chronopoulos, Implementation of iterative methods for large sparse nonsymmetric systems on parallel vector computers, *Internat. J. Supercomputer Appl.* 4 (1990) 9–24.

[19] J. Mandel and M. Brezina, Balancing domain decomposition for problems with large jumps in coefficients, *Math. Comput.* 65 (1996) 1387–1401.

[20] O.A. McBryan and E.F. van der Velde, Matrix and vector operations on hypercube parallel processors, *Parallel Computing* 5 (1987) 117–125.

[21] G. Meurant, Domain decomposition methods for solving large sparse linear systems, in: E. Spedicato, ed., *Computer Algorithms for Solving Linear Algebraic Equations: The State of the Art*, NATO ASI Series, Series F: Computer and Systems Sciences, Vol. 77 (Springer, Berlin, 1991) 185–206.

[22] L. Mikhailov and R. Hanus, Hierarchical control of congested urban traffic-mathematical modeling and simulation, *Math. Comput. Simulation* 37 (1994) 183–188.

[23] nCUBE2 Programmers Guide, nCUBE, 919 E. Hillsdale Guide Boulevard, Foster City, CA 94404, 1992.

[24] J.M. Ortega, *Introduction to Parallel and Vector Solution of Linear Systems* (Plenum Press, 1988) 120–124.

[25] H.J. Payne, FREEFLO: A macroscopic simulation model of freeway traffic *Transportation Research Record* 772 (1979) 68–75.

[26] A. Quarteroni, Domain decomposition and parallel processing for the numerical solution of partial differential equations, *Surv. Math. Industry* 1 (1991) 75–118.

[27] P. Yi et al., Development of an improved high order continuum traffic flow model, *Transportation Research Record* 1365 (1993) 125–132.

[28] G. Rodrigue, Domain decomposition: A unified approach for solving fluid mechanics problems on parallel computers, in: H. Adeli, ed., *Parallel Processing in Computational Mechanics* (Dekker, 1991) 297–330.

[29] U. Schendel, Basic concepts for the development of parallel algorithms and parallel solution of linear systems, in: H. Adeli, ed., *Parallel Processing in Computational Mechanics* (Dekker, 1991) 33–67.

[30] J.H. Shadid and R.S. Tuminaro, A comparison of preconditioned nonsymmetric Krylov methods on large-scale MIMD machine, *SIAM J. Sci. Comput.* 15 (2) (1994) 440–459.

[31] S.G. Ziavras, Efficient mapping of algorithms for a class of hierarchical systems, *IEEE Trans. Parallel Distributed Systems* 4 (11) (1993) 1230–1245.