

NASA/CR-1999-209518
ICASE Report No. 99-34



A Parallel Compact Multi-dimensional Numerical Algorithm with Aeroacoustics Applications

Alex Povitsky
ICASE, Hampton, Virginia

Philip J. Morris
The Pennsylvania State University, University Park, Pennsylvania

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, VA

Operated by Universities Space Research Association



National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

Prepared for Langley Research Center
under Contract NAS1-97046

August 1999

A PARALLEL COMPACT MULTI-DIMENSIONAL NUMERICAL ALGORITHM WITH AEROACOUSTICS APPLICATIONS

ALEX POVITSKY* AND PHILIP J. MORRIS†

Abstract. In this study we propose a novel method to parallelize high-order compact numerical algorithms for the solution of three-dimensional PDEs in a space-time domain. For this numerical integration most of the computer time is spent in computation of spatial derivatives at each stage of the Runge-Kutta temporal update. The most efficient direct method to compute spatial derivatives on a serial computer is a version of Gaussian elimination for narrow linear banded systems known as the Thomas algorithm. In a straightforward pipelined implementation of the Thomas algorithm processors are idle due to the forward and backward recurrences of the Thomas algorithm. To utilize processors during this time, we propose to use them for either non-local data independent computations, solving lines in the next spatial direction, or local data-dependent computations by the Runge-Kutta method. To achieve this goal, control of processor communication and computations by a static schedule is adopted. Thus, our parallel code is driven by a communication and computation schedule instead of the usual “creative programming” approach. The obtained parallelization speed-up of the novel algorithm is about twice as much as that for the standard pipelined algorithm and close to that for the explicit DRP algorithm.

Key words. parallel computing, high-order numerical method, compact scheme, aeroacoustics, pipelined Thomas Algorithm, banded matrices

Subject classification. Computer Science, Fluid Mechanics

1. Introduction. High order accurate numerical schemes are needed to capture multi-scale phenomena and the long-time integration characteristics required for problems of computational wave propagation and the direct numerical simulation of turbulence.

Implicit finite difference formulas are defined as expressions where derivatives at different mesh points appear simultaneously [1, 2]. The price that must be paid for high order accuracy with low dissipation and dispersion is that compact finite difference schemes require the solution of a linear narrow-banded system of equations for the unknown derivative values. For instance, one can achieve 8th and 10th orders of accuracy solving tridiagonal and pentadiagonal systems [2], respectively. The use of implicit filters [3] enables implementation of compact schemes for non-linear models with non-uniform grids. The number of arithmetic operations per grid node is approximately equal for explicit and compact formulations of the same order [4]. Whereas efficient parallelization of explicit central-difference schemes has been implemented by several authors [5, 6] the efficient implementation of compact schemes on parallel computers remains an open problem.

In a multi-dimensional case the partial derivatives can be found by the solution of the banded linear systems formed by considering each spatial partial derivative separately. The most computationally efficient

*Staff Scientist, ICASE, NASA Langley Research Center, Hampton, VA 23681-2199 (e-mail: aeralpo@icase.edu), this research was supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-97046 while the author was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23681

†Boeing/A. D. Welliver Professor, Department of Aerospace Engineering, The Pennsylvania State University, University Park, PA 16802 (e-mail: pjm@psu.edu)

method for the solution of a linear banded system in a single processor is a version of Gaussian Elimination known as the Thomas algorithm. For a systems with N unknowns this method requires $O(N)$ operations.

Parallel solvers that adopt the Thomas algorithm for sets of independent banded systems are of the pipelined type. Pipelines occur due to the recurrence of data within a loop. The main disadvantage is that during the pipelined process processors will be idle at the beginning of the computations and when the algorithm switches from the forward to the backward computational step. Note, that the idle stage exists even if communications are very fast, because processors must wait for completion of computations on the previous processors. A natural way to avoid far-field data-dependency is to introduce artificial boundary conditions (ABC) at inter-domain interfaces. Nordstrom and Carpenter [7] have shown that multiple interface ABC lead to a decrease of the stability range and accuracy for high-order compact schemes. Additionally, the theoretical stability analysis is restricted to linear PDEs and uni-directional partitioning.

As an alternative to pipelining, several concurrent direct linear banded solvers have been developed (see [8, 9] and bibliography in these references). These algorithms are based on matrix-vector multiplications instead of forward and backward recursive steps of the Thomas algorithm. For matrices with narrow bands these factorizations have a higher degree of parallelism than the standard pipelined Thomas algorithm. These techniques lead to a substantial increase in the number of floating-point operations (a factor of 2-2.5), which effectively reduces the gains obtained by parallelism [8].

Hofhaus and van de Velde [9] compared the pipelined Thomas algorithm with other direct methods (recursive doubling, cyclic reduction, divide and conquer, and partition method) and observed that it has the lowest floating-point operation count and requires the least amount of communication. However, it is less concurrent than some other methods due to the startup time required for all processors to participate in the computation (i.e., the pipelined nature of this algorithm).

Sun [10] developed a Parallel Diagonal Dominant (PDD) algorithm which is specifically designed for the solution of Toeplitz tridiagonal systems arising from compact schemes. Taking into account the constant and diagonally dominant nature of the coefficients of Toeplitz matrices, Sun dropped intermediate coefficients and investigated the accuracy of this approximation, which is a necessary part of PDD. However, the PDD algorithm is an approximation of the original high-order compact schemes and it has a higher computational overhead compared to the Thomas algorithm.

Eidson and Erlebacher [11] developed a chained (pipelined) algorithm for the case of periodic boundary conditions. For non-periodic boundary conditions, they proposed a reordering of the elements within the array in order to avoid idle time. However, in this case the computational field would be partitioned in a non- contiguous way and, therefore, the communication costs are large.

The goal in this paper is to develop a parallel compact algorithm which keeps the same computational cost and produces exactly the same solution as its single-processor analog. The algorithm should also be suitable for any local boundary conditions.

We recall that in the standard pipelined Thomas algorithm processors stay idle at some stages of the solution of linear banded systems in any spatial direction. Compact schemes require the solution of data-independent linear systems in three spatial directions. Therefore, processors can be used for computations of derivatives in the next spatial direction while they cannot proceed with computations corresponding to solutions of linear systems in the current direction. On the other hand, Runge-Kutta computations are local but data-dependent, i.e. all spatial derivatives in a grid node must be computed before the temporal update.

The key feature of the proposed algorithm is that processors are used for the next computational tasks, whereas in the standard pipelined Thomas algorithm they stay idle waiting for data from neighboring proces-

sors at the forward and the backward steps of the Thomas algorithm. As a result, in the proposed algorithm processors run in a time-staggered way performing their computational tasks contiguously. In turn, the optimal number of lines to be solved per message becomes larger than that for the standard pipelined Thomas algorithm. Reduction of the number of messages is especially important for processor networks where the communication latency time is larger than that for MIMD parallel computers. Reduction of idle time and communication latency time leads to a considerable increase in speed-up.

To make this algorithm feasible, a static schedule is used to control processor activities. To assign this schedule before the execution of numerical computations, Povitsky [12] recently developed a recursive scheduling algorithm for a one-dimensional pipeline of processors. Here we adopt this algorithm to obtain an idle-less 3-D high-order parallel method.

The paper contains four sections. In Section 2, we describe compact numerical schemes and the Thomas algorithm in a serial case. In Section 3, we describe our parallelization method for compact solvers. In Section 4, we describe a test case and compare the parallelization efficiency for our algorithm, the standard pipelined Thomas Algorithm and an explicit scheme.

2. Compact numerical scheme. Consider a multi-dimensional first-order partial differential equation (PDE):

$$(1) \quad \frac{dU}{dt} = \sum_k S_k \frac{\partial U}{\partial x_k}$$

where t is the time, $k = 1, 2, 3$ are spatial coordinates. The mixed derivatives are not taken into consideration, to allow a directionally split compact numerical scheme to be used for the solution of above equation.

The first derivative terms, such as $\partial U / \partial x_1$, are approximated using compact finite difference schemes [2]:

$$(2) \quad \beta U'_{i-2} + \alpha U'_{i-1} + U'_i + \alpha U'_{i+1} + \beta U'_{i+2} = \frac{a}{2\Delta x} (U_{i+1} - U_{i-1}) + \frac{b}{2\Delta x} (U_{i+2} - U_{i-2}),$$

where Δx is the grid spacing and primes denote derivatives with respect to x_1 . Expansion to systems with second spatial derivatives (Navier-Stokes type) is straight-forward as the compact formulation for second derivatives and the method for their computation is similar to those for the first derivatives. For non-periodic boundaries, one-side near-boundary discretizations have the form

$$(3) \quad U'_1 + \alpha_b U'_2 = \frac{1}{\Delta x} \sum_{i=1, \dots, N_b} a_{bi} U_i,$$

where N_b is the size of the near-boundary stencil and a_{bi} are discretized coefficients. With this choice the boundary schemes can be used with a tridiagonal interior scheme without increasing the bandwidth [2]. In this study the classical Padé scheme ($\alpha = 0.25$, $a = 1.5$ and $\beta = b = 0$) is taken as an example with a tridiagonal matrix for the right and left sides of (2). The proposed method of parallelization can be easily expanded to any compact scheme described by (2).

Equation (1) is discretized in time with an explicit Runge-Kutta (RK) scheme. The solution is advanced from time level n to time level $n + 1$ in several sub-stages [13]

$$(4) \quad \begin{aligned} H_i^M &= \sum_k S_k \frac{\partial U^M}{\partial x_k} + a^M H_i^{M-1}, \\ U_i^{M+1} &= U_i^M + b^{M+1} \Delta t H_i^M, \end{aligned}$$

where $M = 1, \dots, Q$ is the number of sub-stages; $i = 1, \dots, L$ denotes the unknown variables; and the coefficients a^M and b^M depend upon the order of the RK scheme.

To compute derivatives $\partial U^M / \partial x_k$, we must solve a set of independent linear banded systems of equations where each system corresponds to one line of the numerical grid. For example, a system corresponding to a line in the x direction has a scalar tridiagonal matrix $N_x \times N_x$:

$$(5) \quad a_{k,l}x_{k-1,l} + b_{k,l}x_{k,l} + c_{k,l}x_{k+1,l} = f_{k,l},$$

where $k = 1, \dots, N_x$, $l = 1, \dots, N_y \times N_z$, $a_{k,l}, b_{k,l}, c_{k,l}$ are the coefficients, $x_{k,l}$ are the unknown variables, and N_x, N_y and N_z are the number of grid nodes in the x, y and z directions, respectively.

The first step of the Thomas algorithm is LU factorization

$$(6) \quad d_{1,l} = b_{1,l}, \quad d_{k,l} = b_{k,l} - a_{k,l} \frac{c_{k-1,l}}{d_{k-1,l}}, \quad k = 2, \dots, N_x,$$

and forward substitution (FS)

$$(7) \quad g_{1,l} = \frac{f_{1,l}}{d_{1,l}}, \quad g_{k,l} = \frac{-a_{k,l}g_{k-1,l} + f_{k,l}}{d_{k,l}}, \quad k = 2, \dots, N_x.$$

The second step of the Thomas algorithm is backward substitution (BS)

$$(8) \quad x_{N_x,l} = g_{N_x,l}, \quad x_{k,l} = g_{k,l} - x_{k+1,l} \frac{c_{k,l}}{d_{k,l}}, \quad k = N_x - 1, \dots, 1.$$

The coefficients a_k, b_k and c_k are constant for compact schemes; therefore, LU factorization is performed only once and the first step computations include only forward substitution (7).

The serial algorithm for the compact numerical solution of the system (1) is performed as follows:

1. Compute the right-hand side of equations (2) using values of the governing variable U from the previous time step.
2. Compute the spatial derivatives solving tridiagonal systems in all spatial directions.
3. Compute the right-hand side of equations (1) using the spatial derivatives computed on Step 2 and update governing variables by Runge-Kutta scheme.
4. Repeat computational steps 1-3 for all Q stages of Runge-Kutta scheme.
5. Repeat computational steps 1-4 for all time steps.

3. Parallelization method.

3.1. Partitioning scheme. The computational domain is split into subdomains and each subdomain is loaded on a processor. Steps 1 and 2 require exchange of interfacial data between neighboring processors. For a sub-domain with a given volume (number of grid nodes per processor) and a parallelepiped shape (interface planes parallel to coordinate planes), a cube has the minimum surface-to-volume ratio that secures the most efficient parallelization [15, 16].

Overlap regions on each side of the sub-domain store information that must be transferred from neighboring domains, i.e, the forward-step coefficients, the backward-step solution and the values of the main variables to compute the right-hand sides of the compact formulations (2). For the classical Padé scheme, one layer of nodes from each side should be transferred to neighboring processors. If $\beta \neq 0$ and/or $b \neq 0$, two layers of nodes are required to store the transferred data. Note, that overlap regions are used only for data storage and not for redundant computations, i.e., the computations are exactly the same as in the single processor case.

To parallelize a serial code using 3-D partitioning is a difficult task. However, an object-oriented approach adapted in C++ makes it possible to use the same class for pipelined computations in all spatial directions. Three-dimensional partitioning with cubic subdomains is adopted in the present study.

3.2. Parallelization of direct linear solvers. Consider the parallelization of Step 2. Suppose, a line l in the x direction is split amongst the processors (see Figure 1). Computing its part of the l^{th} line, the p^{th} processor: receives coefficient $g_{N(p-1)/P,l}$ from the $(p-1)^{\text{th}}$ processor and puts it in an overlap node $\{0, l\}$; computes the forward step coefficients $g_{k,l}$, where $k = N(p-1)/P + 1, \dots, Np/P$; sends coefficients $g_{Np/P,l}$ to the $(p+1)^{\text{th}}$ processor; and repeats computations (7) for the next lines until all the forward step computations are completed. After completion of all forward step computations specific to a single processor, the p^{th} processor (except the last) has to wait for the completion of the forward step computations by all processors ahead of it. The last outermost (P^{th}) processor starts the backward step computations (8) first. Other processors proceed with the backward step computations in a similar manner as the forward step computations. An overlap layer of nodes $N+1$ is used for backward computations.

In the literature [11, 14, 15] the parallelization penalty for the solution of sets of linear banded systems has been reduced by sending the necessary information to neighboring processors for groups of computed lines at the forward and backward steps of the Thomas algorithm. The optimal number of lines to be solved per message (the size of packet) has been derived as a function of computation time per grid point and communication time (see Eq. (9)).

Figure 2 presents the communication and computations within a pipeline in a single spatial direction. The pipeline includes five processors. Lines are gathered in nine packets in the forward direction and in six packets in the backward direction. Zeros denote the idle time that occurs at the beginning of computations and when the algorithm switches from the forward to the backward computational step.

We define the standard pipelined Thomas algorithm (PTA) to be the method described above for the solution of sets of linear banded systems on multiple processors. If the computational domain is partitioned in all spatial directions, computations in the next spatial direction are pipelined as well. Therefore, a processor belongs to three pipelines. Global synchronization of processors occurs at each spatial step and processors stay idle waiting for data from immediate neighbors.

In the proposed algorithm we avoid this idle stage by performing computations in the next spatial direction when there is no available data to perform the Thomas algorithm computations in a current spatial direction. In other words, we fill idle time units of the standard pipelined algorithm with useful computations.

The Runge-Kutta computations (Step 3) are local but data-dependent because these computations use spatial derivatives in all directions as input data. Consequently, all spatial derivatives must be computed before RK computations can be performed for corresponding grid nodes. Thus, we cannot perform Runge-Kutta computations (Step 3) while processors are idle between the forward and the backward steps of the Thomas algorithm (see above). By this time spatial derivatives in the last rendered direction are not yet computed.

The Immediate Backward Pipelined Thomas Algorithm (IB-PTA) has been developed by Povitsky [12] and is implemented here for the computations of the spatial derivative in the last direction. The processor schedule is shown in Figure 3. The idea behind this algorithm is that the backward step computations for each group of lines start immediately after the completion of the forward step computations for these lines. Each processor switches between the forward and backward steps of the Thomas algorithm for various groups of lines. As for the standard PTA, a processor communicates with its neighbors to get the necessary data for the beginning of either the forward or backward computations for the next group of lines.

The IB-PTA itself is not an idle-less algorithm. It has been shown [12] that the idle time is the same for the IB-PTA and the standard PTA when these algorithms are used in a single direction (compare Figures 2 and 3). The advantage of the IB-PTA over the standard PTA is that processors become idle after completion of the subset of lines, i.e, zeros appear after “-1”s in Figure 3. In the proposed algorithm, the IB-PTA is used in such a way that the idle time units are filled with the local Runge-Kutta computations. Obviously, one can use the IB-PTA in the first two directions as well.

The two types of interplay between processor activities considered here require the use of a processor schedule to control processor computations and communication. The remainder of this section describes computation of the optimal number of lines solved per message (subsection 3.3), generation of the processor schedule (subsection 3.4) and the computational schedule-driven algorithm (subsection 3.5). In subsection 3.6, we will discuss ways to create the schedule for more general domains.

3.3. Optimal size of packet of lines. For the standard PTA, the latency and the processor idle time tradeoff for sets of linear banded systems leads to the following expression [14, 15]:

$$(9) \quad K_1 = \sqrt{\frac{N\gamma}{\rho(N_d - 1)}}, \quad K_2 = \sqrt{\frac{N\gamma}{N_d - 1}},$$

where K_1 and K_2 are the numbers of lines solved per message on the forward and backward steps of the Thomas algorithm, respectively, $\gamma = b_0/g_2$ is the ratio of the communication latency and the backward step computational time per grid node and $\rho = g_1/g_2$ is the ratio of the forward and the backward step computational times.

To generate the processor schedule for the IB-PTA the number of lines solved per message in either direction should be defined.

In this subsection we consider the 3-D domain with a $N_{tot} \times N_{tot} \times N_{tot}$ numerical grid and an equal number of subdomain partitions in each spatial direction.

For the IB-PTA, the time to perform forward step computations per portion of lines is equal to that for backward step computations:

$$(10) \quad NK_1g_1 = NK_2g_2,$$

where $N = N_{tot}/N_d$ is the number of grid nodes in a single direction per sub-domain, N_d is the number of partitions in one spatial direction and g_1 and g_2 are the computational times per grid node for the forward and backward steps.

The first outermost processor in the x direction computes the forward step of the Thomas algorithm in the y direction while this processor is waiting for the backward step solution from the second processor. The time balance of this processor is given as follows:

$$(11) \quad K_1N(N_d - 1)g_1 + K_2N(N_d - 1)g_2 = N^2 \times Ng_1,$$

The left-hand side of the above equation represents the time between the beginning of the backward-step computations and the completion of the forward-step computations of the Thomas algorithm in the x direction. The right-hand side is the time for the forward step computations in the next (y) spatial direction.

Combining Eqs (11) and (10) we obtain

$$(12) \quad K_1 = \frac{N^2}{2(N_d - 1)},$$

$$(13) \quad K_2 = \rho K_1.$$

The same time balance equation is obtained for the share of processor time between computations in the y direction and in the z direction.

The potential idle stage of the processors performing the IB-PTA in the z direction is used for local Runge-Kutta computations. To have completed lines for RK computations, the first portion of lines must be completed with backward step computations no later than the first outermost processor completes the forward step computations. This leads to a time balance constraint similar to that described by Eq. (11).

3.4. Scheduling algorithm. A unit, that the proposed schedule addresses, is defined as the time for the treatment of a portion of lines by either forward or backward step computations in any spatial direction. At each time unit each processor either performs forward or backward step computations or local Runge-Kutta computations for one packet of lines. To set up this schedule, let us define the “partial schedules” corresponding to sweeps in a spatial direction as follows:

$$(14) \quad J(p, i, dir) = \begin{cases} +1 & \text{forward step computations} \\ 0 & \text{processor is idle} \\ -1 & \text{backward step computations,} \end{cases}$$

where $dir = 1, 2, 3$ denotes a spatial direction, p is the number of processors in a processor row in the dir direction and i is the number of the unit.

A recursive algorithm to compute the schedule in a single spatial direction was proposed by Povitsky [12]:

$$(15) \quad \begin{aligned} J(p, l_{min}, dir) &= 1 & \text{if } J(p+1, l, dir) &= 1 \\ J(p, l+2, dir) &= -1 & \text{if } J(p+1, l, dir) &= -1 \\ J(p, l, dir) &= 0 & \text{otherwise,} \end{aligned}$$

where $l_{min} = \min(1 \leq j \leq l | J(p, j, dir) = 0)$. The corresponding valid schedule must be assigned to the last outermost processor prior to the above recursive computations (see [12] for more details). Thus, different pipelined algorithms (for example, the IB-PTA and the standard PTA) are fully defined by their schedule on the last outermost processor.

In the framework of Cartesian partitioning, a processor (I, J, K) receives the forward-step coefficients from its left neighbors $(I-1, J, K)$, $(I, J-1, K)$ and $(I, J, K-1)$ and sends the forward-step coefficients to its right neighbors $(I+1, J, K)$, $(I, J+1, K)$ and $(I, J, K+1)$. Performing the backward-step computations, the processor sends results of computations to the left neighbors and receives data from the right neighbors.

For the standard PTA, a processor computes “direction-by-direction”, and its activities are controlled by the communications, i.e., a processor waits for available data. For the IB-PTA a processor receives data from a neighbor only when it is necessary to complete the computations. Therefore, the communication schedule is assigned by means of a computations schedule as follows. At the beginning of each time unit a processor communicates with some of its right neighbors according to the value of the scheduling variable C :

$$(16) \quad C(p, i, right[dir]) = \begin{cases} 0 & \text{processors } p \text{ and } p+1 \text{ do not communicate,} \\ 1 & \text{send to processor } p+1, \\ 2 & \text{receive from processor } p+1, \\ 3 & \text{simultaneous send and receive.} \end{cases}$$

The end of the i^{th} time unit on the p^{th} processor corresponds to the beginning of the i^{th} time unit on the $(p+1)^{th}$ processor in the same spatial direction. Therefore, the communication schedules in any spatial direction are computed as described in [12]:

$$(17) \quad C(p, i+1, right[dir]) = \begin{cases} 1 & \text{if } J(p+1, i-1, dir) \neq -1 \text{ \& } J(p+1, i, dir) = 1, \\ 2 & \text{if } J(p+1, i-1, dir) = -1 \text{ \& } J(p+1, i, dir) \neq 1, \\ 3 & \text{if } J(p+1, i-1, dir) = -1 \text{ \& } J(p+1, i, dir) = 1, \\ 0 & \text{otherwise.} \end{cases}$$

The definition of $C(p, i, left[dir])$ and its computation are similar to that for $C(p, i, right[dir])$.

The final computational schedule is defined by

$$(18) \quad T(p, i) = \begin{cases} dir & \text{FS computations in the direction } dir, \\ -dir & \text{BS computations in the direction } dir, \\ 4 & \text{local RK computations.} \end{cases}$$

Partial directional schedules must be combined to form a final schedule. For example, the processors should be scheduled to execute the forward step computations in the y direction while their partial schedules include an idle stage between the forward and the backward step computations in the x direction.

The final schedule is set by merging schedules in all three spatial directions, as follows:

1. Skip the idle time units

$$while(J(p, i, dir) = 0) \quad \{l_b = l_b + 1; i = i + 1\}$$

2. Assign the partial schedule to the earliest available time unit

$$T(p, l_{min}) = J(p, i, dir) \times dir; \quad l_b = l_{min}$$

$$\text{where } l_{min} = \min(l_b < j | T(p, j) = 0).$$

3. Assign communication schedule $C(p, i, left[dir])$ and $C(p, i, right[dir])$ to the time unit l_{min} .
4. Repeat steps 1-3 until all elapsed time units in the current direction are completed.

The first step ensures that the time interval between any computational activities does not become smaller than that for a partial schedule. Otherwise, one might schedule the backward step computations immediately after completion of the forward step computations (see Figure 1) and get an incorrect schedule.

The obtained schedule meets the following requirements of consistency: (i) each processor performs one task per time unit; (ii) the forward step computations on the p^{th} processor begin no earlier than the conclusion of these computations for a current group of lines in the same direction on the $(p-1)^{th}$ processor (the left neighbor); (iii) the backward step computations on the p^{th} processor begin no earlier than conclusion of these computations on the $(p+1)^{th}$ processor (the right neighbor) and (iv) the backward step computations begin after completion of the forward step computations in the same direction for the current group of lines.

An example of a communication and computations schedule for the first outermost processor (1,1,1) is shown in Table 1. Obviously, this processor communicates only with its right neighbors, (2, 1, 1), (1, 2, 1) and (1, 1, 2). Here the IB-PTA is used in all three spatial directions.

Computations of the right-hand sides of Equation (2) requires the exchange of interfacial values of the governing variables. A straightforward way to parallelize the algorithm includes exchange of the near-boundary values before each time step. Each processor exchanges data with its neighbors (I-1,J,K), (I,J-1,K), (I,J,K-1) in all three spatial directions, then waits for the completion of computational tasks by its other neighbors (I+1,J,K), (I,J+1,K), (I,J,K+1) and finally exchanges data with these three processors. The ‘‘asynchronous send - synchronous receive’’ mode of communication is suitable for exchange of interfacial data. The interfacial values are stored in overlap node layers ‘‘0’’ and ‘‘N + 1’’. This communication leads to

local synchronization between processors. Additionally, exchange of boundary values with three processors simultaneously may lead to deterioration of parallelization efficiency. To avoid this synchronization, we propose to transfer these values together with forward-step coefficients in the corresponding directions by means of the following algorithm:

1. Compute the uncompleted forward-step coefficients $g_{i,N}$ for interfacial nodes, as follows:

$$g_{i,N}^{un} = \frac{-a_N g_{N-1,l} - \frac{a}{2\Delta x} U_{N-1}}{d_N}$$

2. Transfer values $g_{N,l}^{un}$ and U_N to the next processor and put them in the overlap layer 0.
3. Complete computation of $g_{N,l}$ on the next processor:

$$(g_{0,l} =) g_{N,l} = g_{N,l}^{un} + \frac{a}{2\Delta x} \frac{U_{N+1}}{d_N},$$

where the $(N + 1)^{th}$ grid node is the first grid node on the next processor.

4. Compute right-hand side of Eq. (2) for the first node on the next processor.
5. Performing the backward-step computations, transfer values $g_{N,l}$ back to the current processor.

This algorithm avoids local synchronization between the neighboring processors and reduces the traffic of messages between the processors. This leads to approximately a 25% reduction of the parallelization penalty in comparison with the straight-forward algorithm (see above).

To sum up, the generation of processor schedule includes (i) computation of the size of packet by Eq. (12), (ii) computation of the processor schedule for the last processor in a current direction `citerep1`, (iii) recurrent computation of schedule for all processors in the pipeline by Eq. (15), (iv) computation of the scheduling variables by Eqs. (16-18) and (v) binding of schedules in spatial directions as it was described above.

3.5. Computational algorithm. After assignment of the processor schedule on all processors, the computational part of the method runs on all processors by an algorithm presented in Figure 4. The static processor schedule governs the consequence of computations and communications on each processor. At the beginning of each unit a processor communicates with its neighbors by the schedule (variable **Com**) and then performs scheduled computations (variable **T**.) The proposed code style fully separates computational routines from communication procedures that allows for easy reuse of the code.

3.6. Some extensions. Let us consider a global domain $N_{tot,x} \times N_{tot,y} \times N_{tot,z}$, where $N_{tot,x} \neq N_{tot,y} \neq N_{tot,z}$. In this case the number of partitions is different for different directions, i.e., $N_{d,x} \neq N_{d,y} \neq N_{d,z}$, and the analog of Eq. (12) in the x direction is given by:

$$(19) \quad K_{1,x} = \frac{N^2}{2(N_{d,x} - 1)}.$$

Assuming $N_{d,x} > N_{d,y} > N_{d,z}$, we round $K_{1,y}$ and $K_{1,x}$ to smaller integers in such a way that $mK_{1,x} = K_{1,y}$ and $nK_{1,y} = K_{1,z}$. So doing, Eq. (11) holds for any direction and processors run idle-less. The processor schedule addresses a packet of size $K_{1,x}$ as a unit. Computing the Thomas algorithm in the x direction, processors use potential idle time for computations in the y direction as in the previous case. Here a processor treats m packets of lines in the y direction before communication with the neighboring processor. Doing computations in the z direction, a processor treats mn packets per communication.

To reduce further the number of processors in a pipeline, we propose to combine our scheduling algorithm with a two-way decomposition, denoted as the TW algorithm in this study. Direct solvers for banded linear

systems based on two-sided Gauss Elimination were introduced by Babuska [17] and are referred to as twisted factorization, two-way decomposition (TW), and “burn from two sides” by various authors. The computational count per grid node for the TW algorithm is the same as for the serial Thomas algorithm. An additional 2×2 system of linear equations should be solved per line. The number of processors in the pipeline is half the number compared to the standard Thomas algorithm; therefore, the parallelization penalty is reduced. For long chains of processors, we propose to combine our scheduling algorithm with the two-way pipelined algorithm. The price for this improvement is programming of the Thomas algorithm in an inversed direction.

In this case the schedule is generated for the rows of the first $P/2$ and the last $P/2$ processors. Then we include exchange of the forward-step coefficients between the $(P/2)^{th}$ and $(P/2 + 1)^{th}$ processors and the solution of a 2×2 system. These tasks are performed immediately after completion of the forward-step computations for each group of lines on middle processors.

For very stretched domains the size of a cubic subdomain becomes bigger than the domain size in some directions. In this case 1-D partitioning by stretched (non-cubic) sub-domain is proposed. Since the linear systems need to be solved in each direction, no matter how the grid is partitioned over the processors, there will be at least one direction in which the recurrence spans across several processors. This direction is taken last, the proposed scheduling algorithm is used to combine the IB-PTA in this direction with the RK computations. The Thomas algorithm computations in the other directions are trivial to solve, since processors contain the full systems.

For practically important multi-zone situations the governing partial differential equations are discretized on sets of numerical grids connecting at interfacial boundaries by ABC. In this case the number of processors in each zone is arbitrary and can be determined to be proportional to the size of zone. Here we cannot always partition a zone with cubic sub-domains. For example, a cubic zone is perfectly (i.e, in a load-balanced way) covered by cubic sub-domains only in a case that the number of processors allocated to this zone is cube of an integer number. Otherwise, a domain partitioning degrades to two- or one- dimensional partitioning with poor surface-to-volume ratio. Our approach allows for the combination of schedules corresponding to different pipelines and, therefore, a processor can handle subsets of different grids (or non-aligned pieces of the same grid) to ensure load balance and idle-less performance.

4. Parallel computations.

4.1. Benchmark problem. As an example of a three-dimensional model problem we consider the development of an acoustic pulse in an unbounded domain. This problem was also considered as a benchmark case by Morris *et al.* [18]. The problem satisfies the linearized Euler equations with no basic flow and constant thermodynamic basic properties. If the linearized Euler equations are non-dimensionalized with respect to the basic density, the speed of sound and the grid spacing as a length scale they may be written as,

$$(20) \quad \begin{aligned} \frac{\partial u}{\partial t} &= -\frac{\partial p}{\partial x}, \\ \frac{\partial v}{\partial t} &= -\frac{\partial p}{\partial y}, \\ \frac{\partial w}{\partial t} &= -\frac{\partial p}{\partial z}, \\ \frac{\partial p}{\partial t} &= -\frac{\partial u}{\partial x} - \frac{\partial v}{\partial y} - \frac{\partial w}{\partial z}. \end{aligned}$$

The initial conditions are given by,

$$(21) \quad p = \epsilon \exp \left[-\frac{x^2 + y^2 + z^2}{a} \right],$$

where $\epsilon = 0.01$ and $a = \ln(2)/9$.

The analytical solution for an infinite domain is given by

$$(22) \quad p_{\text{anal}} = \frac{\epsilon}{2r} \{ (r-t) \exp[-a(r-t)^2] + (r+t) \exp[-a(r+t)^2] \}.$$

Characteristic boundary conditions are applied at $\partial\Omega$. The computational domain is $\Omega = [-30 < x < 30] \times [-30 < y < 30] \times [-30 < z < 30]$.

For comparison, the same problem has been solved using an explicit dispersion-relation-preserving (DRP) spatial discretizations with a seven point stencil [19]. According to Colonius [4] this scheme has approximately the same dispersion behavior and computational count as the considered 4-th order compact scheme. A constant coefficient sixth-order artificial dissipation is added to the DRP scheme [18]. The volumetric average of the absolute error $\sum_{i,j,k} |p_{\text{comp}} - p_{\text{anal}}|/\Omega$ is shown in Fig. 5. As would be expected from their dispersion properties, the error in these two cases is almost equal when $t < 25$. Then, for $t \geq 25$ the accuracy is determined by the implementation of the boundary conditions and not by the interior scheme properties. The error in the explicit scheme is dependent on the artificial dissipation coefficient. Two values have been considered: $\mu = 0.004$ and $\mu = 0.002$. As expected, the absolute error is reduced as the value of μ is decreased.

4.2. Speed-up. The CRAY T3E MIMD computer used in this study is installed in the San Diego Supercomputer Center (SDSC) at the University of California, San Diego.

The scheduling part of the parallel code includes the computation of the optimal number of lines solved per message (i.e., number of portions of lines) and the assignment of a communication and computation schedule (see the previous section). The solver part, which is controlled by the static schedule, includes the Thomas algorithm computations in the spatial directions (step 1 of the serial algorithm); local Runge-Kutta computations (steps 2 and 3) and loops by the stages of the RK computations and by time. Exactly the same compact numerical schemes are used for the standard PTA and the proposed algorithm. The speed-up on an MIMD computer with P processors over a single processor solving the same problem is defined by

$$(23) \quad S = \frac{T_{\text{serial}}}{T_{\text{parallel}}},$$

where T_{parallel} is the actual elapsed time per processor on a MIMD computer and T_{serial} is the actual elapsed time on a single processor. The parallelization penalty is defined as follows:

$$(24) \quad T_P = \frac{T_{\text{parallel}}P - T_{\text{serial}}}{T_{\text{serial}}} \times 100\% = \left(\frac{P}{S} - 1 \right) \times 100\%.$$

Obviously, in the ideal case $S = P$ and $T_P = 0$. For example, 100% parallelization penalty corresponds to the case where a code runs on P processors and its speed-up S is equal to $P/2$. We recall that parallelization penalty exceeds 100% for concurrent parallel solvers for banded matrices (see Introduction).

Parallel speed-ups for explicit and compact computations with the standard PTA algorithm and the proposed algorithm are shown in Figure 6. Speed-up is shown as a function of the number of nodes per processor per direction.

The level of parallelization penalty is 25 – 30% when the sub-domain size varies from 20^3 to 10^3 . The parallelization penalty for explicit schemes is invariant to the number of processors involved in the computations because the only source of parallelization penalty is local communication due to exchange of interfacial values of the governing variables. Speed-ups for the compact scheme combined with our scheduling algorithm are similar to those obtained for explicit computations whereas speed-ups for the standard PTA algorithm are substantially lower.

Sample computer runs on $3 \times 3 \times 3 = 27$, $4 \times 4 \times 4 = 64$ and $5 \times 5 \times 5 = 125$ processors with $10^3 - 20^3$ grid nodes per processor show that the parallel speed-up increases 1.5 – 2 times over the standard PTA (Figure 7). The novel algorithm and the standard one are used with corresponding optimal numbers of lines solved per message. The size of the packet is substantially larger for the proposed algorithm than that for the standard PTA (Figure 5).

5. Conclusion. The pipelined Thomas algorithm has been applied to a multi-dimensional aeroacoustics problem solved by a compact (implicit in space and explicit in time) numerical scheme. To achieve good parallelization efficiency: the computational domain is split into cubic subdomains; the number of lines solved per message is optimal; the values of the governing variables are transferred together with the forward-step coefficients; and the schedule-driven computations are performed in such a way that an idle stage for the processors is avoided. Under this schedule, the processors perform computations in the next spatial direction while otherwise they are idle from recursive computations in the current direction. To get completed data for the Runge-Kutta temporal update, the Immediate Backward Pipelined Thomas Algorithm is used in the last spatial direction. Processors perform their tasks in a contiguous way. The optimal number of lines solved per message is larger than that for the standard Thomas algorithm. The absence of idle time and the reduced latency of the communications lead to a substantial reduction of the parallelization penalty.

Using modern MIMD computers with low communication latency (below $100\mu s$) the parallelization penalty of the proposed PTA is below 100% if the number of grid nodes per processor is more than 10^3 . This is a significant improvement over alternative algorithms for implicit schemes. Thus, one can use the Thomas algorithm “as it is” rather than program concurrent parallel solvers. On the other hand, the obtained parallelization efficiency is comparable to that for the explicit dispersion-relation-preserving scheme with the same order of accuracy.

Acknowledgments. The first author is grateful for Dr. Mark Carpenter (NASA Langley Research Center) for discussion about compact schemes. The authors also thank Mr. Chingwei M. Shieh (the Pennsylvania State University) for performing the computations using the DRP scheme.

REFERENCES

- [1] HIRSCH, CH., *Numerical computation of internal and external flows, Vol. 1: Fundamentals of numerical discretizations*, John Wiley and Sons, Chichester, 1994.
- [2] LELE, S. K., “Compact Finite Difference Schemes with Spectral Like Resolution,” *Journal of Computational Physics*, **103** (1992), pp. 16-42.
- [3] GAITONDE, D. V. AND VISBAL, M. R., “Further Development of a Navier-Stokes Solution Procedure Based on Higher-order Formulas,” *AIAA Paper No. 99-0557*, 1999.

- [4] COLONIUS, T., "Lectures on Computational Aeroacoustics," presented at the lecture series on Aeroacoustics and Active Noise Control, von Karman Institute of Fluid Dynamics, 1997, <http://green.caltech.edu/~colonius>.
- [5] SHANG, J. S., CAMBEROS, J. A., WHITE, M. D., Advances in Time-domain Computational Electromagnetics, *AIAA Paper No. 99-3731*, 1999.
- [6] LOCKARD, D. P. AND MORRIS, P. J., A Parallel Implementation of a Computational Aeroacoustic Algorithm fo Airfoil Noise, *Journal of Computational Acoustics*, **5**, No. 4 (1997), pp. 337-353.
- [7] NORDSTROM, J. AND CARPENTER, M., Boundary and Interface Conditions for High Order Finite Difference Methods Applied to the Euler and Navier-Stokes Equations, *ICASE Report No. 98-19*, 1998.
- [8] DONGARRA, J. J., DUFF, I. S., SORENSEN, D. C. AND VAN DER VORST, H. A., *Numerical Linear Algebra for High-performance Computers*, SIAM, 1998.
- [9] HOFHAUS, J. AND VAN DE VELDE, E. F., "Alternating-direction Line-relaxation Methods on Multi-computers," *SIAM J. of Scientific Computing*, **17**, No. 2 (1996), pp. 454-478.
- [10] SUN, X.-H., "Applications and Accuracy of the Parallel Diagonal Dominant Algorithm," *ICASE Report No. 93-6*, 1993.
- [11] EIDSON, T. M. AND ERLEBACHER, G., "Implementation of a Fully-balanced Periodic Tridiagonal Solver on a Parallel Distributed Memory Architecture," *ICASE Report No. 94-37*, 1994.
- [12] POVITSKY, A., "Parallelization of the Pipelined Thomas Algorithms," *ICASE Report No. 98-45*, 1998 (To appear, *Journal of Parallel and Distributed Computing*, Sept. 1999).
- [13] WILSON, R. V., DEMUREN A. O. AND CARPENTER, M., "High-order Compact Schemes for Numerical Simulation of Incompressible Flows," *ICASE Report No. 98-13*, 1998.
- [14] HO, C.-T. AND JOHNSON, L., "Optimizing Tridiagonal Solvers for Alternating Direction Methods on Boolean Cube Multiprocessors," *SIAM Journal of Scientific and Statistical Computing*, **11**, No. 3 (1990), pp. 563-592.
- [15] NAIK, N. H., NAIK, V. K. AND NICOULES, M., "Parallelization of a Class of Implicit Finite Difference Schemes in Computational Fluid Dynamics," *International Journal of High Speed Computing*, **5** (1993) pp. 1-50.
- [16] HATAY, F., JESPERSEN, D. C., GURUSWAMY, G. P., ET AL., "A multi-level parallelization concept for high-fidelity multi-block solvers," paper presented in SC97: High Performance Networking and Computing, San Jose, California, November 1997.
- [17] BABUSKA, I., "Numerical stability in problems of linear algebra," *SIAM Journal of Numerical Analysis*, **9** (1972), pp. 53-77.
- [18] MORRIS, P. J., LONG, L. N., BANGALORE, A. AND WANG, Q., "A Parallel Three-dimensional Computational Aeroacoustics Method Using Nonlinear Disturbance Equations," *Journal of Computational Physics*, **133** (1997), pp. 56-74.
- [19] Tam, C. K. W. and Webb, J. C., "Dispersion-relation-preserving Finite Difference Schemes for Computational Acoustics," *Journal of Computational Physics*, **107** (1993), pp. 262-281.

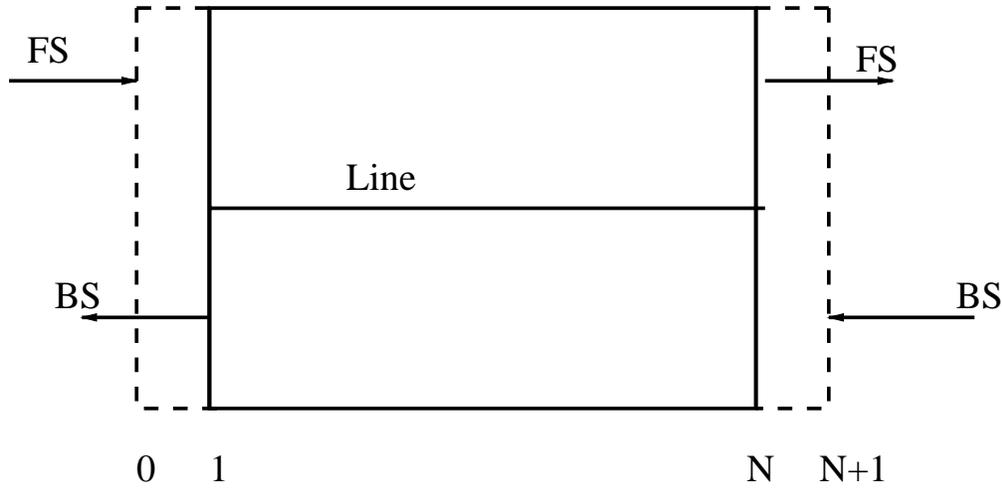


FIG. 1. Data traffic in a single direction

TABLE 1

Schedule of communication and computations for the first outermost processor $(1,1,1)$, where i is the number of time unit, T denotes type of computations, $(2, 1, 1)$, $(1, 2, 1)$ and $(1, 1, 2)$ denote communication with corresponding neighbors

i	T	$(2, 1, 1)$	$(1, 2, 1)$	$(1, 1, 2)$
1	1	0	0	0
2	1	1	0	0
3	1	1	0	0
4	1	1	0	0
5	1	1	0	0
6	1	1	0	0
7	1	0	0	0
8	-1	3	0	0
9	2	0	0	0
10	-1	3	0	0
11	2	0	1	0
12	-1	2	0	0
13	2	0	1	0
14	-1	2	0	0
15	2	0	1	0
16	-1	2	0	0
17	2	0	1	0
18	-1	2	0	0
19	2	0	1	0
20	-1	2	0	0
21	2	0	0	0
22	-2	0	3	0
23	3	0	0	0
24	-2	0	3	0
25	3	0	0	1
26	-2	0	2	0
i	T	$(2, 1, 1)$	$(1, 2, 1)$	$(1, 1, 2)$
27	3	0	0	1
28	-2	0	2	0
29	3	0	0	1
30	-2	0	2	0
31	3	0	0	1
32	-2	0	2	0
33	3	0	0	1
34	-2	0	2	0
35	3	0	0	0
36	-3	0	0	3
37	4	0	0	0
38	-3	0	0	3
39	4	0	0	0
40	-3	0	0	2
41	4	0	0	0
42	-3	0	0	2
43	4	0	0	0
44	-3	0	0	2
45	4	0	0	0
46	-3	0	0	2
47	4	0	0	0
48	-3	0	0	2
49	4	0	0	0
50	4	0	0	0
51	4	0	0	0
52	4	0	0	0
53	4	0	0	0

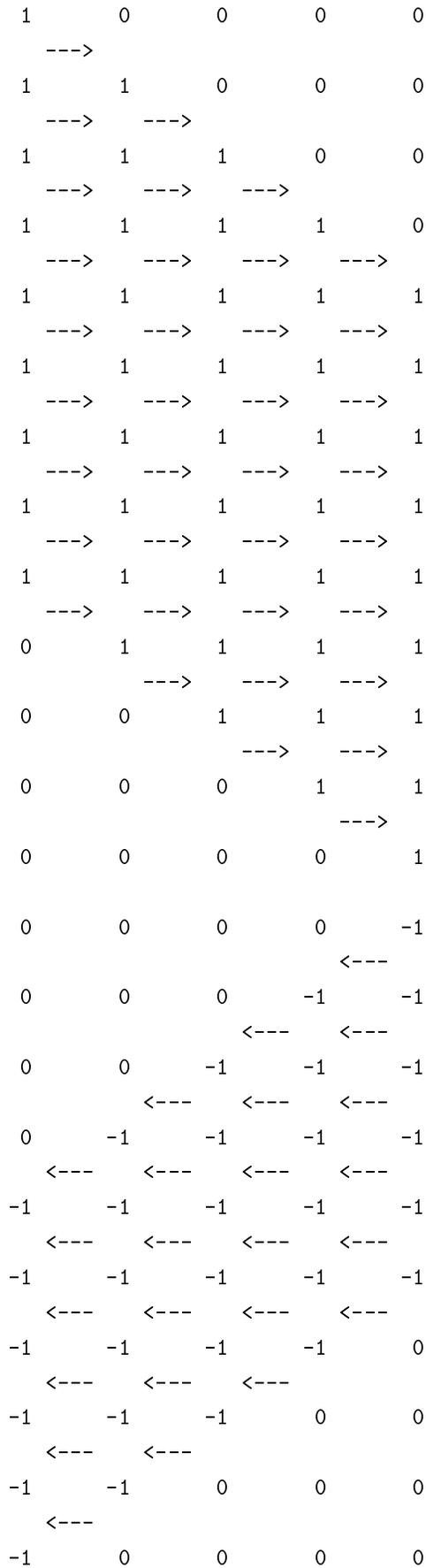


FIG. 2. Schedule of processors for the PTA in a single direction. Here each column corresponds to a processor, “0”, “1” and “-1” denote idle stage, forward and backward computations; arrows $----$, $<----$ denote the send and receive communications.

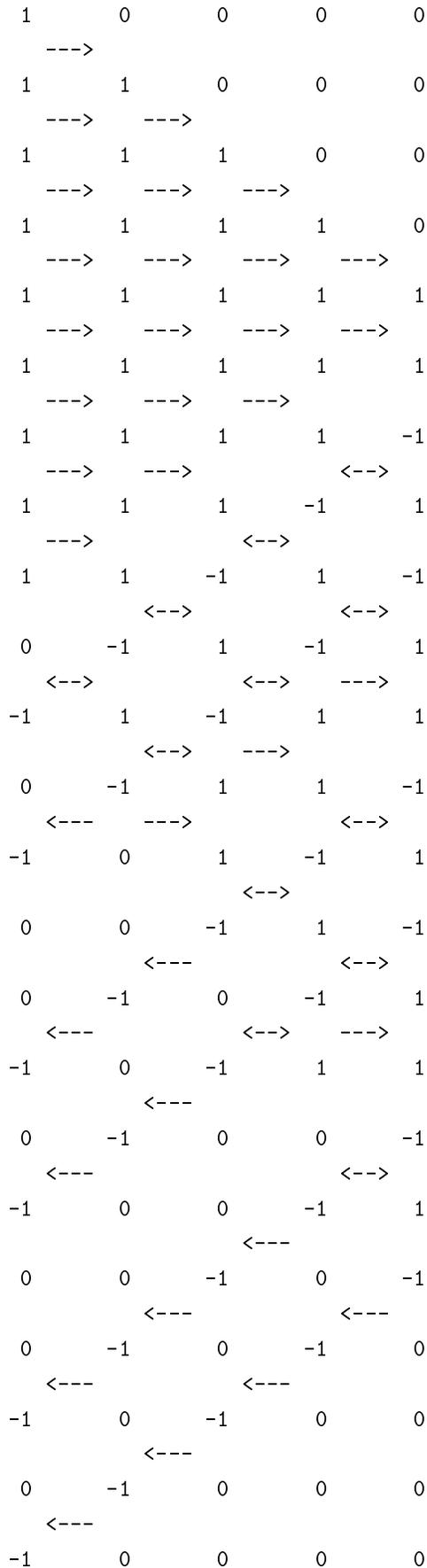


FIG. 3. Schedule of processors for the IB-PTA. The legend is the same as in the previous Figure, < -- > denote the send-receive communications.

```

for  $i=1, \dots, I$ 
{
for  $dir=1, 3$ 
{
if ( $\mathbf{Com}(\mathbf{p}, i, \mathbf{right}[dir]) = 1$ ) send FS coefficients to right processor;
if ( $\mathbf{Com}(\mathbf{p}, i, \mathbf{right}[dir]) = 3$ ) send FS coefficients to right processor
and receive BS solution right processor;
if ( $\mathbf{Com}(\mathbf{p}, i, \mathbf{left}[dir]) = 1$ ) send BS solution to left processor;
if ( $\mathbf{Com}(\mathbf{p}, i, \mathbf{left}[dir]) = 3$ ) send BS solution to left processor
and receive FS coefficients from left processor;
if ( $\mathbf{Com}(\mathbf{p}, i, \mathbf{right}[dir]) = 2$ ) receive BS solution for line lb from right processor;
if ( $\mathbf{Com}(\mathbf{p}, i, \mathbf{left}[dir]) = 2$ ) receive FS coefficients from left processor;
}
for  $dir=1, 3$ 
{
if ( $\mathbf{T}(\mathbf{p}, i) = dir$ ) do FS computations
if ( $\mathbf{T}(\mathbf{p}, i) = -dir$ ) do BS computations
}
if ( $\mathbf{T}(\mathbf{p}, i) = 4$ ) do RK computations
}

```

FIG. 4. Schedule-governed banded linear solver, where $right = p + 1$ and $left = p - 1$ denote left and right neighbors, $dir = 1, 2, 3$ corresponds to $x, y,$ and z spatial directions, T governs computations, Com controls communication with neighboring processors, p is the processor number, and i is the number of group of lines (number of time unit).

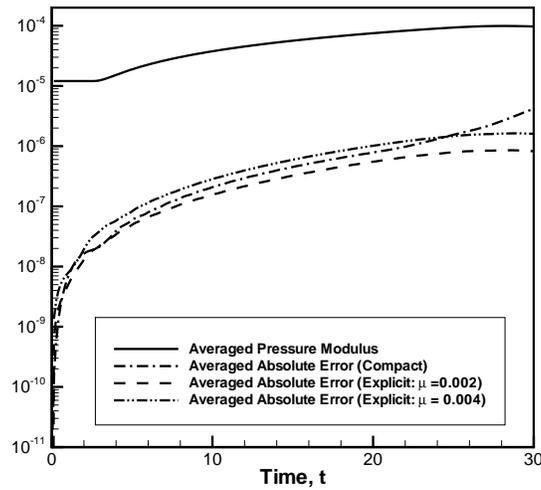


FIG. 5. Temporal behavior of absolute error for explicit DRP and compact schemes

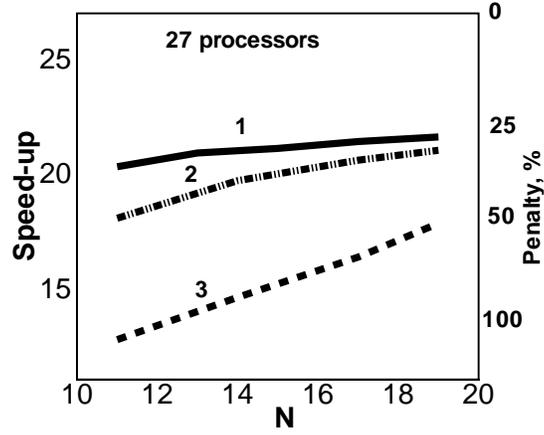


FIG. 6. Speed-up (S) and parallelization penalty T_P for explicit and compact schemes, (1)-explicit scheme;(2)-proposed algorithm and (3)-standard PTA

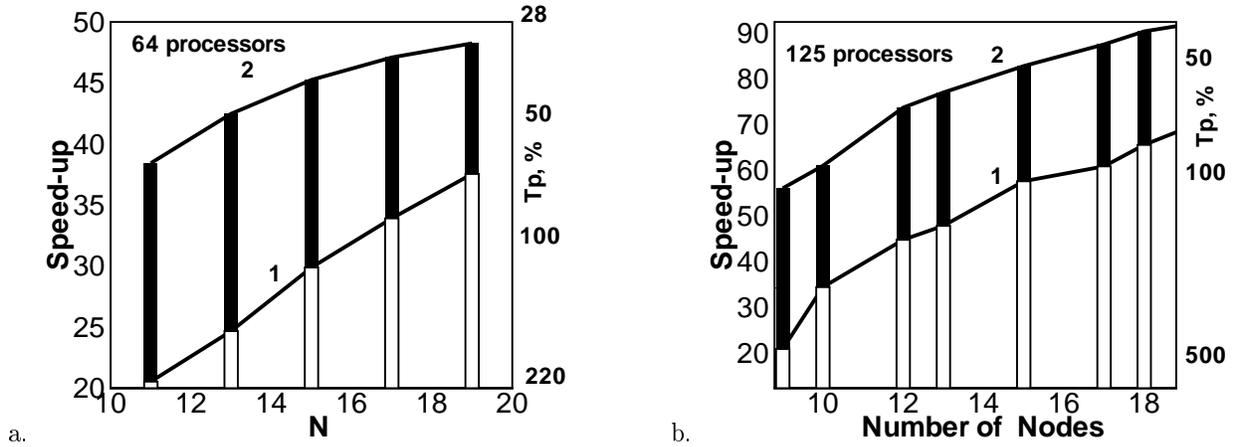


FIG. 7. Speed-up and parallelization penalty for (a) $4 \times 4 \times 4 = 64$ and (b) $5 \times 5 \times 5 = 125$ processors. (1)-standard PTA, and (2)-proposed algorithm

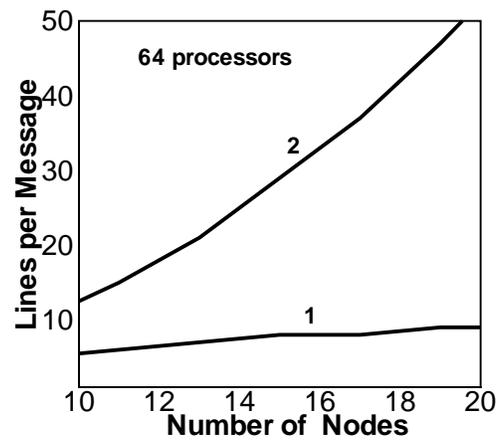


FIG. 8. Number of lines solved per message, (1)-standard PTA, and (2)-proposed algorithm