Journal of Computational Physics **161,** 182–203 (2000)
doi:10.1006/jcph.2000.6497, available online at http://www.idealibrary.com on IDEAL®

A Higher-Order Compact Method in Space and Time Based on Parallel Implementation of the Thomas Algorithm

Alex Povitsky* and Philip J. Morris†

**ICASE, M.S. 132C, NASA Langley Research Center, Hampton, Virginia 23681-0001; †Department of Aerospace Engineering, The Pennsylvania State University, University Park, Pennsylvania*
E-mail: aeralpo@icase.edu

Received August 9, 1999; revised March 10, 2000

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 such a 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 nonlocal 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 basic pipelined algorithm and close to that for the explicit DRP algorithm. Use of the algorithm is demonstrated and comparisons with other schemes are given.

Key Words: parallel computing; processor scheduling; high-order numerical method; compact scheme; pipelined Gauss elimination; banded linear systems.

1. INTRODUCTION

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

0021-9991/00 $35.00

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 nonlinear models with nonuniform grids.

While compact schemes of the sixth order are widely used ([4, 5], for example), explicit DRP schemes [6] are used only up to the formal fourth order, to the best of the authors' knowledge. Typically, the wide stencils of DRP schemes lead to more severe problems of stability than compact schemes.

The number of arithmetic operations per grid node and accuracy properties are practically equal for DRP and compact formulations of the fourth order [7]; therefore, we compare the parallelization efficiency of compact schemes with that of DRP schemes using fourth-order schemes as an example. Then, we show that the parallelization efficiency of our algorithm, being applied to a higher (sixth) order compact scheme, is practically equal to that of the fourth-order compact scheme.

Whereas efficient parallelization of explicit central-difference schemes has been implemented by several authors [8, 9] the efficient implementation of compact schemes on parallel computers remains an open problem.

In a multidimensional 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 method for the solution of a linear banded system on 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 interdomain interfaces. Nordstrom and Carpenter [10] 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 unidirectional partitioning.

As an alternative to pipelining, several concurrent direct linear banded solvers have been developed (see [11], [12], and bibliography in these references). These algorithms are based on matrix-vector multiplications instead of the forward and backward recursive steps of the Thomas algorithm. For matrices with narrow bands these factorizations have a higher degree of parallelism than the basic 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 [11].

Hofhaus and van de Velde [12] 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 [13] 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 [14] developed a chained (pipelined) algorithm for the case of periodic boundary conditions. For nonperiodic 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 noncontiguous 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 basic pipelined Thomas algorithm processors stay idle at some stages of the solution of the 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 basic pipelined Thomas algorithm they stay idle waiting for data from neighboring processors 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 basic 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 [15] 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 basic pipelined Thomas Algorithm, and an explicit scheme.

2. COMPACT NUMERICAL SCHEME

Consider a multidimensional first-order partial differential equation (PDE)

$$\frac{dU}{dt} = \sum_k S_k \frac{\partial U}{\partial x_k}, \tag{1}$$

where t is the time, in the 3-D case $k = 1, 2, 3$ denote spatial coordinates x, y, and z. 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]

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

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 straightforward as the compact formulation for second derivatives and the method for their computation is similar to those for the first derivatives. For nonperiodic boundaries, one-sided near-boundary discretizations have the form

$$U'_1 + \alpha_b U'_2 = \frac{1}{\Delta x} \sum_{i=1,\ldots,N_b} a_{bi} U_i, \tag{3}$$

where N_b is the size of the near-boundary stencil and a_{bi} are discrete 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 substages [5]

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

where $M = 1, \ldots, Q$ are the number of substages; $i = 1, \ldots, L$ 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 $N_x \times N_x$

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

where $k = 1, \ldots, N_x$, $l = 1, \ldots, N_y \times N_z$, $a_{k,l}$, $b_{k,l}$, $c_{k,l}$ 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

$$d_{1,l} = b_{1,l}, \qquad d_{k,l} = b_{k,l} - a_{k,l}\frac{c_{k-1,l}}{d_{k-1,l}}, \qquad k = 2, \ldots, N_x, \tag{6}$$

and forward substitution (FS)

$$g_{1,l} = \frac{f_{1,l}}{d_{1,l}}, \qquad g_{k,l} = \frac{-a_{k,l}g_{k-1,l} + f_{k,l}}{d_{k,l}}, \qquad k = 2, \ldots, N_x. \tag{7}$$

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

$$x_{N_x,l} = g_{N_x,l}, \qquad x_{k,l} = g_{k,l} - x_{k+1,l}\frac{c_{k,l}}{d_{k,l}}, \qquad k = N_x - 1, \ldots, 1. \tag{8}$$

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 Eq. (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 Eq. (1) using the spatial derivatives computed on Step 2 and update governing variables by the Runge–Kutta scheme.
4. Repeat computational steps 1–3 for all Q stages of the 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 subdomain 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 [16, 17].

Overlap regions on each side of the subdomain 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 transfered 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 among the processors (see Fig. 1). Computing its part of the lth line, the pth processor: receives coefficient $g_{N(p-1)/P,l}$ from the $(p-1)$th processor and puts it in an overlap node $\{0, l\}$; computes the forward step coefficients $g_{k,l}$, where $k = N(p-1)/P + 1, \ldots, Np/P$; sends coefficients $g_{Np/P,l}$ to the $(p+1)$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 pth processor (except the last) has to wait for the completion of the forward step computations by all processors ahead of it. The last outermost (Pth) processor starts the backward step computations (8) first. Other processors proceed with the backward step computations in a manner similar to the forward step computations. An overlap layer of nodes $N+1$ is used for backward computations.

In the literature [14, 16, 18] 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. (11)).

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 basic 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 basic 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

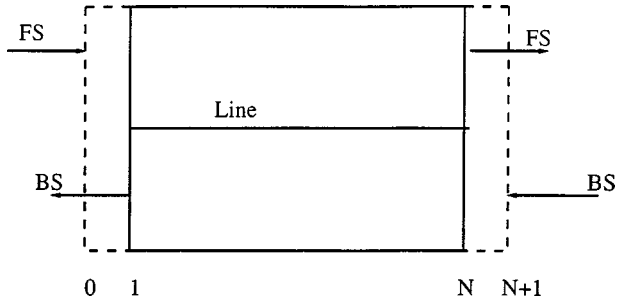

FIG. 1. Data traffic in a single direction.

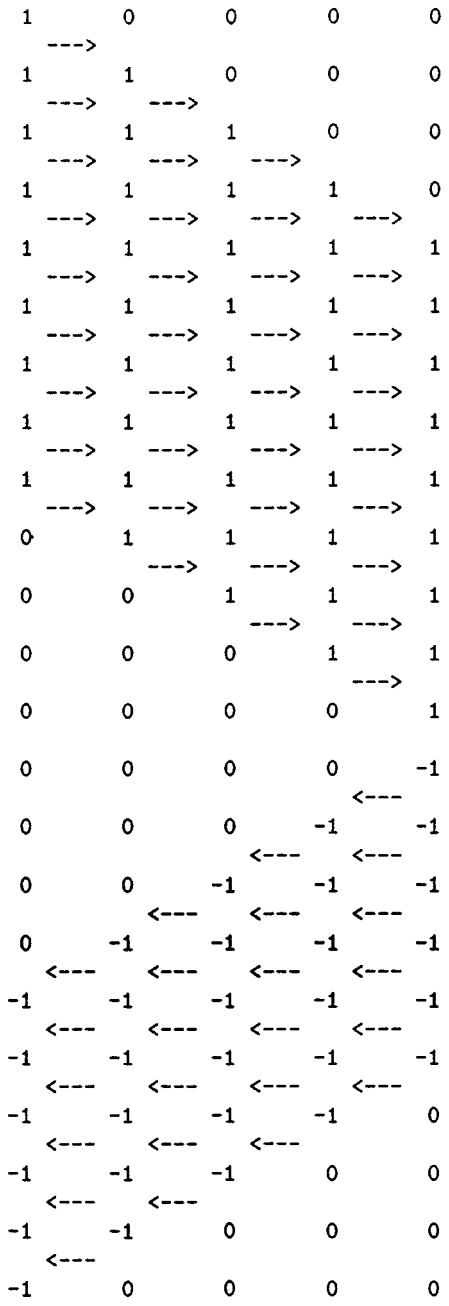

FIG. 2. Schedule of processors for the PTA in a single direction. Here each column corresponds to a processor, "0", "1", and "−1" denote idle stage, forward, and backward computations; arrows - - - >, < - - - denote the send and receive communications.

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.

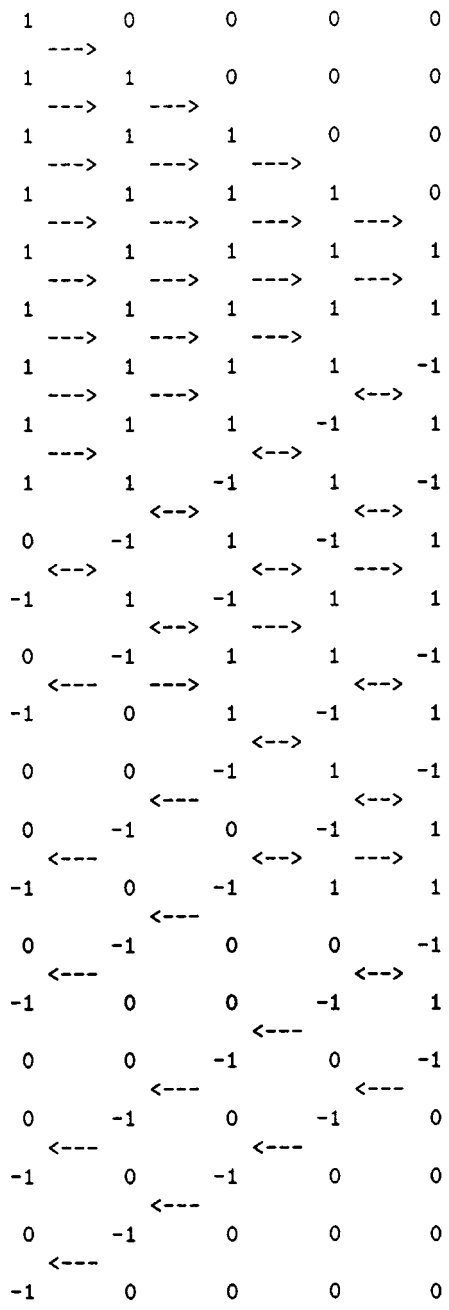

FIG. 3. Schedule of processors for the IB-PTA. The legend is the same as in the previous figure, < - - > denotes the send–receive communications.

The Immediate Backward Pipelined Thomas Algorithm (IB-PTA) has been developed by Povitsky [15] and is implemented here for the computations of the spatial derivative in the last direction. The processor schedule is shown in Fig. 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 basic 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 [15] that the idle time is the same for the IB-PTA and the basic PTA when these algorithms are used in a single direction (compare Figs. 2 and 3). The advantage of the IB-PTA over the basic PTA is that processors become idle after completion of the subset of lines; i.e., zeros appear after "-1"s in Fig. 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 and systems of equations with mixed spatial derivatives.

3.3. *Optimal Size of Packet of Lines*

In the previous subsection, the way to use potential idle stage of processors was shown. Actually, the use of idle time in the proposed algorithm leads to such a trade-off (described in this subsection) between the communication latency and the data dependency delay that considerably increases the optimum number of lines solved per message and, therefore, the proposed algorithm has smaller latency time and fewer messages than the basic algorithm.

The additional (penalty) time in a multiprocessor system originates from (1) communication time due to the transfer of the forward step coefficients and the backward step solution of the Thomas Algorithm, and (2) the processor idle time due to the pipelined nature of the Thomas algorithm.

The former reason for penalty time exists in the proposed algorithm and in the basic algorithm and is computed as a sum of communication latency time and communication transfer time

$$F_1 = \sum_{i=1}^{i=3} \lceil N^2/K_{1,i} \rceil (b_0 + L_f b_1 K_{1,i}) + \lceil N^2/K_{2,i} \rceil (b_0 + L_b b_1 K_{2,i}), \tag{9}$$

where $i = 1, 2, 3$ are spatial coordinates, $K_{1,i}$ and $K_{2,i}$ are the size of packets for the forward and backward steps in the ith spatial direction, respectively, $\lceil N^2/K_{1,i} \rceil$ and $\lceil N^2/K_{2,i} \rceil$ are the number of messages, L_f and L_b is the number of words per numerical grid point transfered on the forward and backward step (see Table II). Here the linear model of communication time between processors is adapted, i.e., time $= b_0 + b_1 K$, where K is the length of the string in words.

The latter component of penalty time for the basic algorithm is given by

$$F_2 = \sum_{i=1}^{i=3} (N_{d,i} - 1) N (K_{1,i} g_1 + K_{2,i} g_2), \tag{10}$$

where $N_{d,i}$ is the number of processors per pipeline in the ith direction, g_1 and g_2 are the computational times per grid node for the forward and backward steps. This component of penalty may be avoided for the proposed algorithm (see the previous subsection).

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 basic PTA, the latency and the processor idle time tradeoff for sets of linear banded systems leads to the expression [16, 18]

$$K_1 = \sqrt{\frac{N\gamma}{\rho(N_d - 1)}}, \qquad K_2 = \sqrt{\frac{N\gamma}{N_d - 1}}, \tag{11}$$

where $\gamma = b_0/g_2$ 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.

For the IB-PTA, which is used in the last spatial direction, the time to perform forward step computations per portion of lines is equal to that for backward step computations:

$$NK_1g_1 = NK_2g_2. \tag{12}$$

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

$$K_1N(N_d - 1)g_1 + K_2N(N_d - 1)g_2 = N^2 \times Ng_1. \tag{13}$$

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. (13) and (12) we obtain

$$K_1 = \frac{N^2}{2(N_d - 1)}, \quad K_2 = \rho K_1. \tag{14}$$

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

In most cases, the K values computed above are bigger than those computed by Eq. (11) and, therefore, communication latency time is smaller for the proposed algorithm than for the basic one. Otherwise, the K values for proposed algorithm are chosen as maximum of those defined by Eqs. (14) and (11). In this case, communication latency times are equal for the basic and the proposed algorithm. Here the idle time for the proposed algorithm is no longer equal to zero and is given by

$$F_2 = \sum_{i=1}^{i=3}((N_{d,i} - 1)N(K_{1,i}g_1 + K_{2,i}g_2) - N^3g_1). \tag{15}$$

Actually, this is the time before completion of all forward-step computations on the first processor and beginning of the backward-step computations for the first packet of lines. Still, this idle time is smaller than that for the basic algorithm (Eq. (10)) because of substraction of the second term in the above equation.

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. (13).

To sum up, the parallelization penalty time includes the processor idle time, communication latency time, and communication transfer time. The communication transfer time is equal for the basic algorithm and for the proposed algorithm as the same amount of information $N^2(L_f + L_b)$ is transfered. This time plays a minor role as $b_1 \ll b_1$ for modern multiprocessor systems (see Fig. 8c). Substituting the optimal K for the basic algorithm (Eq. (11)) to the formulas for idle time (9) and for communication time (10), these components of parallelization penalty appear to be equal.

If Eq. (14) is valid, the proposed algorithm avoids the idle time and decreases the communication latency time, therefore, it suppresses parallelization penalty more than two times. Otherwise, the communication latency times are equal for both algorithms and idle time is still better for proposed algorithm (see Eq. 15). This happens for big N_d and small N; however, our computational experiments show that the former situation occurs in most cases and parallelization penalty reduces two times or more (see Section 4).

3.4. *Scheduling Algorithm*

A unit that the proposed schedule addresses is defined as the time for the treatment of a packet of lines by either forward- or backward-step computations in any spatial direction (see the previous subsection).

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

$$J(p, i, dir) = \begin{cases} +1 & \text{forward step computations} \\ 0 & \text{processor is idle} \\ -1 & \text{backward step computations,} \end{cases} \tag{16}$$

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 [15]

$$\begin{aligned} J(p, l_{min}, dir) &= 1 && \text{if } J(p+1, l, dir) = 1 \\ J(p, l+2, dir) &= -1 && \text{if } J(p+1, l, dir) = -1 \\ J(p, l, dir) &= 0 && \text{otherwise,} \end{aligned} \tag{17}$$

where $l_{min} = \min(1 \le j \le l \mid J(p, j, dir) = 0)$. The corresponding valid schedule must be assigned to the last outermost processor prior to the above recursive computations (see [15] for more details). Thus, different pipelined algorithms (for example, the IB-PTA and the basic 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 basic 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:

$$C(p, i, right[dir]) = \begin{cases} 0 & \text{processors } p \text{ and } p+1 \text{ do not communicate,} \\ 1 & \text{send to processor } p+1, \\ 2 & \text{receive from processor } p+1, \\ 3 & \text{simultaneous send and receive.} \end{cases} \tag{18}$$

The end of the ith time unit on the pth processor corresponds to the beginning of the ith 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 [15]:

$$C(p, i+1, right[dir]) = \begin{cases} 1 & \text{if } J(p+1, i-1, dir) \neq -1 \;\&\; J(p+1, i, dir) = 1, \\ 2 & \text{if } J(p+1, i-1, dir) = -1 \;\&\; J(p+1, i, dir) \neq 1, \\ 3 & \text{if } J(p+1, i-1, dir) = -1 \;\&\; J(p+1, i, dir) = 1, \\ 0 & \text{otherwise.} \end{cases} \tag{19}$$

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

$$T(p, i) = \begin{cases} dir & \text{FS computations in the direction } dir, \\ -dir & \text{BS computations in the direction } dir, \\ 4 & \text{local RK computations.} \end{cases} \tag{20}$$

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
   $while(J(p, i, dir) = 0)\ \{l_b = l_b + 1;\ i = i + 1\}$
2. Assign the partial schedule to the earliest available time unit
   $T(p, l_{min}) = J(p, i, dir) \times dir;\ l_b = l_{min}$
   where $l_{min} = \min(l_b < j \mid T(p, j) = 0)$.
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 Fig. 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 pth 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 pth 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 I. 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.

TABLE I

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

TABLE II
Algorithm *B*: Forward-Step Computations for Interfacial Nodes

Order	Current processor	Next processor	L_f	L_b
4th	$g^{un}_{N,l} = \frac{-a_N g_{N-1,l} - \frac{a}{2\Delta x} U_{N-1,l}}{d_N}$	$(g_{0,l} =)\, g_{N,l} = g^{un}_{N,l} + \frac{a}{2\Delta x} \frac{U_{N+1,l}}{d_N}$	2	2
6th	$g^{un}_{N-1,l} = \frac{-a_{N-1} g_{N-2,l} + \frac{a}{2\Delta x}(U_{N,l} - U_{N-2,l}) - \frac{b}{4\Delta x} U_{N-3,l}}{d_{N-1}}$	$(g_{-,l} =)\, g_{N-1,l} = g^{un}_{N-1,l} + \frac{b}{4\Delta x} \frac{U_{N+1,l}}{d_{N-1}}$	4	3
	$g^{un}_{N,l} = \frac{-a_N g^{un}_{N-1,l} - \frac{a}{2\Delta x} U_{N-1} - \frac{b}{4\Delta x} U_{N-2}}{d_N}$	$(g_{0,l} =)\, g_{N,l} = \frac{-a_N g_{N-1,l}}{d_N} + \frac{b}{4\Delta x} \frac{U_{N+2}}{d_N} + \frac{a}{2\Delta x} \frac{U_{N+1}}{d_N}$		

Computations of the right-hand sides of Eq. (2) requires the exchange of interfacial values of the governing variables. A straightforward way to parallelize the algorithm (Algorithm A) 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 B:

1. Compute the uncompleted forward-step coefficients $g^{un}_{N,l}$ ($g^{un}_{N,l}$ and $g^{un}_{N-1,l}$ for the 6th order scheme) for interfacial nodes (see Table II).
2. Transfer values $g^{un}_{N,l}$ and U_N ($g^{un}_{N,l}$, $g^{un}_{N-1,l}$, U_N and U_{N-1} for the 6th order scheme) to the next processor and put them in the overlap layer 0 (layers 0 and - for the 6th order scheme).
3. Complete computation of $g_{N,l}$ ($g_{N-1,l}$ and $g_{N,l}$ for the 6th order scheme) on the next processor.
4. Using $U_{N-1,l}$ ($U_{N-1,l}$ and $U_{N-2,l}$ for the 6th order scheme), compute right-hand side of Eq. (2) for the first node (first two nodes for the 6th order scheme) on the next processor.
5. Performing the backward-step computations, transfer values $g_{N,l}$ ($g_{N,l}$ and $g_{N-1,l}$ for the 6th order scheme) and solution $x_{1,l}$ back to the current processor.

Algorithm B 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 Algorithm A.

3.5. *Computational Algorithm*

The generation of the processor schedule includes (i) computation of the size of packet by Eq. (14), (ii) computation of the processor schedule for the last processor in a current direction [15], (iii) recurrent computation of the schedule for all processors in the pipeline by Eq. (17), (iv) computation of the scheduling variables by Eqs. (18)–(20), and (v) binding of schedules in spatial directions as described in the previous subsection.

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

After assignment of the processor schedule on all processors, the computational part of the method runs on all processors by an algorithm presented in Fig. 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. (14) in the x direction is given by

$$K_{1,x} = \frac{N^2}{2(N_{d,x} - 1)}. \tag{21}$$

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. (13) 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. Results of parallelization efficiency for the case are presented in Section 4.

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 [19] 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 basic 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 inverse 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 every 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 (noncubic) subdomain 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 multizone 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 subdomains. For example, a cubic zone is perfectly (i.e., in a load-balanced way) covered by cubic subdomains 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 nonaligned pieces of the same grid) to ensure load balance and idle-less performance.

For problems with mixed and one-directional second derivatives, Eq. (1) appears as

$$\frac{dU}{dt} = \sum_k S_k \frac{\partial U}{\partial x_k} + \sum_k \sum_j S_{k,j} \frac{\partial^2 U}{\partial x_k \partial x_j}. \tag{22}$$

If $S_{k,j} = 0$ when $k \neq j$, the parallelization strategy is the same as for Eq. (1) as second derivatives are computed solving sets of linear banded systems (2) simultaneously with those systems for first derivatives.

If all $S_{k,j}$ are assumed different from zero, the proposed parallelization algorithm is used in all three spatial directions to compute first derivatives. Then, the same algorithm is applied to compute derivatives of derivatives $(\partial^2 U)/(\partial x_k \partial x_j)$, where the computed first derivatives are used in the right-hand sides of systems (2). The IB-PTA in the z direction

must be used to obtain $(\partial U)/(\partial z)$ before computing the $(\partial^2 U)/(\partial x \partial z)$ while processors are idle. Additionally, lines in the z direction must be treated in such a way that grid points with computed first derivative by z form continuous lines in the x direction. This problem was solved by Povitsky [20] for parallel alternative direction implicit (ADI) algorithms.

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*. [21]. The problem satisfies the linearized Euler equations with no basic flow and constant thermodynamic basic properties. If the linearized Euler equations are nondimensionalized with respect to the basic density, the speed of sound and the grid spacing as a length scale they may be written as,

$$\begin{aligned} \frac{\partial u}{\partial t} &= -\frac{\partial p}{\partial x}, \\ \frac{\partial v}{\partial t} &= -\frac{\partial p}{\partial y}, \\ \frac{\partial w}{\partial t} &= -\frac{\partial p}{\partial z}, \\ \frac{\partial p}{\partial t} &= -\frac{\partial u}{\partial x} - \frac{\partial v}{\partial y} - \frac{\partial w}{\partial z}. \end{aligned} \tag{23}$$

The initial conditions are given by

$$p = \epsilon \exp\left[-\frac{x^2 + y^2 + z^2}{a}\right], \tag{24}$$

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

The analytical solution for an infinite domain is given by

$$p_{\text{anal}} = \frac{\epsilon}{2r}\{(r - t)\exp[-a(r - t)^2] + (r + t)\exp[-a(r + t)^2]\}. \tag{25}$$

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 [6]. According to Colonius [7] this scheme has approximately the same dispersion behavior and computational count as the considered 4th order compact scheme. A constant coefficient 6th order artificial dissipation is added to the DRP scheme [21]. The volumetric average of the absolute error $\sum_{i,j,k} |p_{comp} - p_{anal}|/\Omega$ 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.

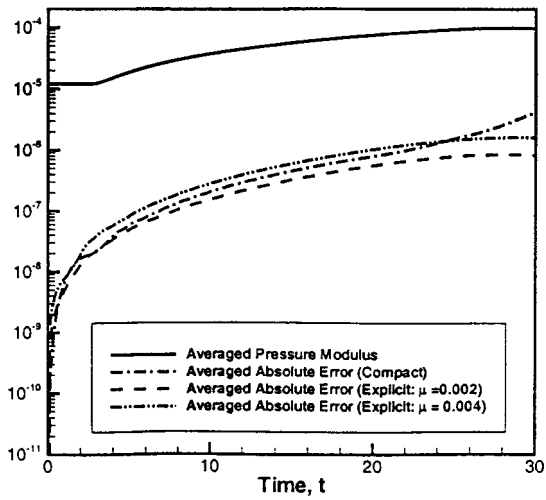

FIG. 5. Temporal behavior of absolute error for explicit DRP and compact schemes.

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 basic 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

$$S = \frac{T_{serial}}{T_{parallel}}, \tag{26}$$

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

$$T_P = \frac{T_{parallel}P - T_{serial}}{T_{serial}} \times 100\% = \left(\frac{P}{S} - 1\right) \times 100\%. \tag{27}$$

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 the parallelization penalty exceeds 100% for concurrent parallel solvers for banded matrices (see the Introduction).

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

The level of parallelization penalty is 25–30% when the subdomain 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

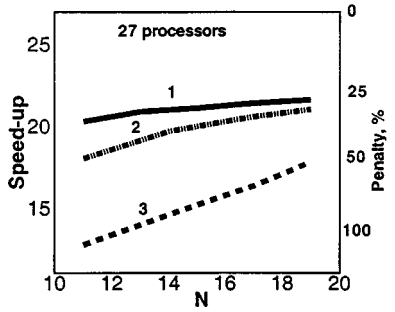

FIG. 6. Speed-up (S) and parallelization penalty T_P for explicit and compact schemes, (1) explicit scheme, (2) proposed algorithm, and (3) basic PTA.

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 basic PTA algorithm are substantially lower.

Parallelization efficiency for the 6th order compact scheme appears to be very close to that for the 4th order compact scheme. The only difference between the two schemes is the amount of transfered information per numerical grid node (see Table II). This communication penalty time is small in comparison with other components of parallelization penalty, i.e., communication latency time and processor idle time (see below).

Sample computer runs on $4 \times 4 \times 4 = 64$ 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 basic parallel compact algorithm (see Figs. 7a and 7b).

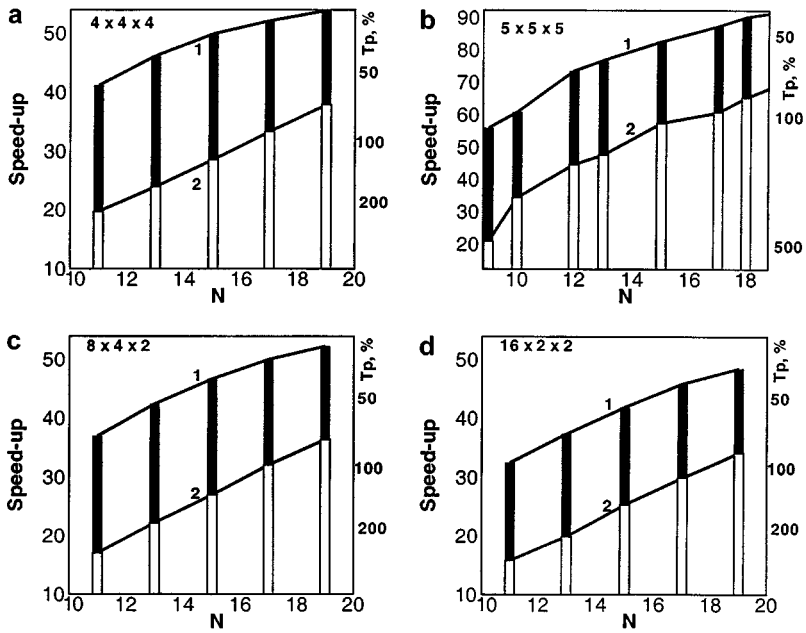

FIG. 7. Speed-up and parallelization penalty (T_P) for 3-D partitionings of computational domain, N is the number of numerical grid points per processor per direction, (a) $4 \times 4 \times 4 = 64$, (b) $5 \times 5 \times 5 = 125$, (c) $8 \times 4 \times 2 = 64$, (d) $16 \times 2 \times 2 = 64$; (1) basic PTA and (2) proposed algorithm.

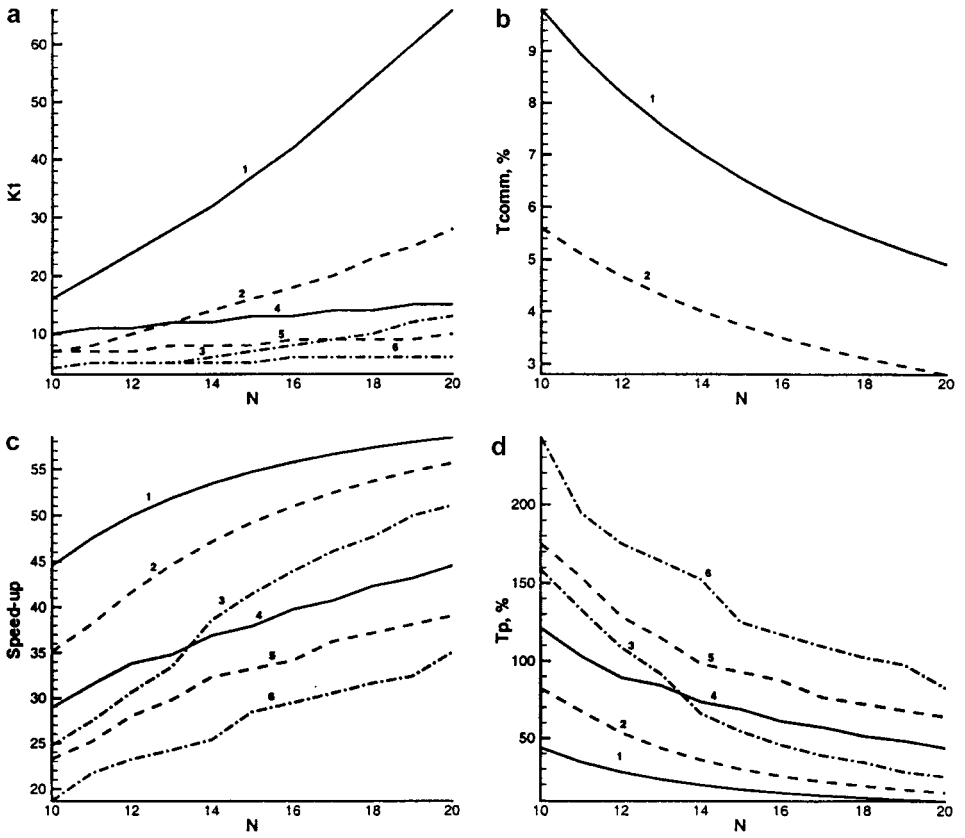

FIG. 8. Theoretical estimation of parallelization efficiency: (a) the optimal number of lines solved per message (for longest pipeline of three), (b) communication part of parallelization penalty T_{comm}, (c) Speed-up, and (d) overall parallelization penalty T_p. Here K_1 is the number of lines solved per message for forward-step computations, N is the number of numerical grid points per processor per direction, pairs of curves (1,4), (2,5), and (3,6) correspond to 3-D partitioning of cubic domains on 4^3, 8^3, and 16^3 processors, curves (1,2,3) correspond to the proposed algorithm (Eq. (21)), and curves (4,5,6) correspond to the basic algorithm (Eq. (11)).

For $8 \times 4 \times 2$ and $16 \times 2 \times 2$ partitionings on 64 processors the speed-up somewhat decreases for both algorithms (see Figs. 7c and 7d). Still, the proposed algorithm outperforms the basic algorithm in terms of speed-up. The novel algorithm and the basic algorithm are used with corresponding optimal numbers of lines solved per message (Eqs. (14), (21), and (11)).

To analyze the obtained speed-ups, some results of theoretical model of parallelization (Subsection 3.3) for 4^3, 8^3, and 16^3 partitioning are presented in Fig. 8. Theoretical and experimental speed-ups for 4^3 partitioning have reasonable correspondence. Experimental speed-ups are lower than theoretical speed-ups due to the assumption of a linear communication time model.

The size of the packet is substantially larger for the proposed algorithm than that for the basic algorithm for 4^3 and 8^3 partitionings (Fig. 8a). For the 16^3 partitioning, the optimal size of packet become equal for both algorithms when $N < 14$. Even in this case, the obtained speed-up for the basic algorithm outperforms that for the basic one (compare curves 3 and 6 in Figs. 8c and 8d). However, the difference in speed-ups decreases when N approaches 10. Note, that the latter case corresponds to $16^3 = 4096$ processors and a

small number of numerical grid nodes per processor ($10^3 = 1000$). For two-sided Gauss Elimination (see Subsection 3.6) this critical number of processors doubles and the overall number of processors becomes equal to 32768 that is considerably larger than a number of processors in any real multiprocessor system. For a nonequal number of processors in different directions, substantial parallelization penalty in the direction with the longest number of processors in a pipeline softens by smaller parallelization penalties in other directions (see Fig. 7d).

In Fig. 8b the communication penalty time $T_{comm} = N^2(L_f + L_b)/T_{serial}$ is presented. As explained in Subsection 3.3, the parallelization penalty for 4th and 6th order compact schemes differs only by means of T_{comm}. This term contributes less than 10% penalty for the 6th order scheme and no more than 5% for the 4th order scheme.

5. CONCLUSION

The pipelined Thomas algorithm has been applied to a multidimensional 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 transfered 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 basic 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 μs) 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 gratefully acknowledges 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. Ch. Hirsch, *Numerical Computation of Internal and External Flows, Vol. 1: Fundamentals of Numerical Discretizations* (Wiley, Chichester, 1994).
2. S. K. Lele, Compact finite difference schemes with spectral like resolution, *J. Comput. Phys.* **103**, 16 (1992).

3. D. V. Gaitonde and M. R. Visbal, Further development of a Navier–Stokes solution procedure based on higher-order formulas, in *37th Aerospace Sciences Meeting & Exhibit, Reno, NV, 1999*. [AIAA Paper 99-0557]
4. O. Inoue and Y. Hattori, Sound generation by shock-vortex interactions, *J. Fluid Mech.* **380**, 81 (1999).
5. R. V. Wilson, A. O. Demuren, and M. Carpenter, *High-Order Compact Schemes for Numerical Simulation of Incompressible Flows* (ICASE Report No. 98-13, 1998).
6. C. K. W. Tam and J. C. Webb, Dispersion-relation-preserving finite difference schemes for computational acoustics, *J. Comput. Phys.* **107**, 262 (1993).
7. T. Colonius, *Lectures on Computational Aeroacoustics* (technical report, von Karman Institute of Fluid Dynamics, Belgium, 1997). [http://green.caltech.edu/colonius]
8. J. S. Shang, J. A. Camberos, and M. D. White, Advances in time-domain computational electromagnetics, in *30th AIAA Plasmadynamics and Lasers Conference, Norfolk, VA, 1999*. [AIAA Paper 99-3731]
9. D. P. Lockard and P. J. Morris, A parallel implementation of a computational aeroacoustic algorithm of airfoil noise, *J. Comput. Acoustics* **5**, 337 (1997).
10. J. Nordstrom and M. Carpenter, Boundary and interface conditions for high order finite difference methods applied to the Euler and Navier–Stokes equations, *J. Comput. Phys.* **148**, 621 (1999).
11. D. C. Sorensen J. J. Dongarra, I. S. Duff, and H. A. van der Vorst, *Numerical Linear Algebra for High-Performance Computers* (SIAM, New York, 1998).
12. J. Hofhaus and E. F. Van De Velde, Alternating-direction line-relaxation methods on multicomputers, *SIAM J. Sci. Comput.* **17**, 454 (1996).
13. X.-H. Sun, *Applications and Accuracy of the Parallel Diagonal Dominant Algorithm* (ICASE Report No. 93-6, 1993).
14. T. M. Eidson and G. Erlebacher, *Implementation of a Fully-Balanced Periodic Tridiagonal Solver on a Parallel Distributed Memory Architecture* (ICASE Report No. 94-37, 1994).
15. A. Povitsky, Parallelization of pipelined algorithms for sets of linear banded systems, *J. Parallel Distrib. Comput.* **59**, 68 (1999).
16. N. H. Naik, V. K. Naik, and M. Nicoules, Parallelization of a class of implicit finite difference schemes in computational fluid dynamics, *Int. J. High Speed Comput.* **5**, 1 (1993).
17. F. F. Hatay, D. C. Jespersen, G. P. Guruswamy *et al*., A multi-level parallelization concept for high-fidelity multi-block solvers, in *Proceedings of the SC97: High Performance Networking and Computing, San Jose, CA, 1997*, pp. 448–456. [http://www.hal.com/users/hatay]
18. C.-T. Ho and L. Johnsson, Optimizing tridiagonal solvers for alternating direction methods on boolean cube multiprocessors, *SIAM J. Sci. Comput.* **11**, 563 (1990).
19. I. Babuska, Numerical stability in problems of linear algebra, *SIAM J. Numer. Anal.* **9**, 53 (1972).
20. A. Povitsky, *Parallel Directionally Split Solver Based on Reformulation of Pipelined Thomas Algorithm* (ICASE Report No. 98-45, 1998).
21. P. J. Morris, L. N. Long, A. Bangalore, and Q. Wang, Three-dimensional computational aeroacoustics method using nonlinear disturbance equations, *J. Comput. Phys.* **133**, 56 (1997).