International Journal of Computational Fluid Dynamics, December 2003 Vol. 17 (6), pp. 467–485

Parallel Implementation of Curvilinear High-order Formulas

F. LADEINDE[a,*], X. CAI[b], M.R. VISBAL[c] and D. GAITONDE[c]

[a]*Mechanical Engineering Department, State University of New York, Stony Brook, NY 11794-2300, USA;* [b]*Aerospace Research Corporation, Stony Brook, L.I., NY 11790, USA;* [c]*Air Vehicles Directorate, Wright-Patterson AFB, OH 45433, USA*

High-order formulas are required for differencing and filtering the Navier–Stokes equations in order to obtain the needed accuracy for a variety of CFD applications. The parallel performance issue relevant to one of these methods, the compact scheme, is studied in this paper with emphasis on the associated implicit operators. Three procedures were selected: the one-sided method, the parallel diagonal dominant (PDD) method, and the parallel Thomas algorithm (PTA) method. These parallel procedures were implemented in the AFRL code, FDL3DI. Kernel codes were also developed to extract some inherent performance features of these methods. Some of the calculations combine the methods for compact differencing and filtering. In general, the procedures based on the one-sided schemes produced accurate results and good parallel performance (efficiency, speedup, and scalability). The procedures that combine the one-sided schemes and PDD also performed well. However, parallel calculations that use the PDD method for both compact differencing and filtering produced the wrong results for low-order filters. On the other hand, high-order filters cause PDD to be very expensive. The one-sided method leads to super-scalable calculations when the number of processors is low. For PDD, increasing the number of grid points in the derivative-difference direction leads to better speedup, as does an increase in the number of right-hand side (RHS) columns. In standard implementation (i.e. without engaging the processors during the idle time), the PTA procedure has a very poor parallel performance in comparison to PDD and the one-sided formulations. However, the procedure tends to be more accurate.

Keywords: Parallel Thomas algorithm; High-order formulas; Compact schemes; Parallel diagonal dominant; Computational fluid dynamics; Implicit formulas; Parallel computation

INTRODUCTION

High-order formulas are required for differencing and filtering the Navier–Stokes equations in order to obtain the needed accuracy for direct numerical simulation (DNS), large-eddy simulation (LES) of turbulence and a variety of CFD applications. In this regard, the essentially non-oscillatory (ENO) scheme has received attention (Cai *et al.* 1996; 1997a,b; 1999), as have the compact finite difference schemes for Cartesian (Lele, 1992) and generalized curvilinear formulations (Visbal and Gaitonde, 1998; Gaitonde and Visbal, 1998). Recently, a comparison of the ENO and compact schemes was presented (Ladeinde *et al.*, 2001) in which the focus was placed on the ability of the two procedures to resolve the high wave number end of the turbulence energy spectrum. The present paper deals with the compact schemes.

The computational load associated with the solution of realistic fluid dynamic systems mandates code execution on supercomputers, to take advantage of massive parallelization in such systems. The development of the compact schemes along this line has been initiated (Gaitonde and Visbal, 1999), wherein the performance of 2D, overlapped, multiblock formulations was investigated, leading to the derivation of appropriate high-order one-sided filter formulas with stable and accurate interblock treatments.

Unlike the ENO schemes which are explicit and parallelize relatively easily (Ladeinde, 1992; Ladeinde *et al.*, 1996; Cai *et al.*, 1997), certain numerical issues undermine the parallel performance of the compact differencing schemes. The first and perhaps most obvious is the implicit nature of the spatial differencing operators in the form of tridiagonal or pentadiagonal matrix systems, which are inherently difficult to parallelize efficiently. The parallel diagonal dominant (PDD) algorithm (Sun and Moitra, 1996) and the parallel Thomas algorithm (PTA) (Povitsky, 1998; Povitsky and Morris, 1999) have been proposed for compact tridiagonal operators. However, algorithmic performance, in terms of efficiency, for the Navier–Stokes equations in generalized curvilinear coordinate systems has not

*Corresponding author. Tel.: +1-631-632-9293. E-mail: foluso.ladeinde@sunysb.edu

ISSN 1061-8562 print/ISSN 1029-0257 online © 2003 Taylor & Francis Ltd
DOI: 10.1080/10618560310001615060

received enough attention. The one-sided multidomain approach (Gaitonde and Visbal, 1999) is also an option for parallel implementation of tridiagonal matrix operators. The performance of the one-sided method has also not been reported for 3D or parallel processing.

The second difficulty with the parallelization of the compact schemes pertains to the discretization of the viscous and heat conduction terms in the flow equations. Compact formulas that directly discretize these second derivative terms have been proposed (Lele, 1992). However, their applicability to general curvilinear formulations involves a significant computational penalty. The successive applications of the compact formula, as in Gaitonde and Visbal (1999), is computationally more efficient and, although the procedure may be more prone to numerical instability, this instability has not been observed in several applications of the method. However, message-passing difficulties are possible in parallel implementation because of the difficulty of bundling small messages in the two-pass procedure.

The third difficulty with the parallelization of the high-order compact schemes pertains to the filter, which is often needed for numerical stability. As shown later in this paper, low-order filters may not work well with some of the algorithms, such as the one-sided and PDD algorithms. On the other hand, the use of high-order filters could lead to unacceptably low parallel efficiency because of the relatively large stencil size.

It is the objective of the present paper to investigate the foregoing issues, with a view toward determining the relative parallel performance (efficiency, speedup and scalability) of various algorithms for differencing and filtering. Due to our interest in realistic aerodynamic systems, we particularly focus on procedures that are relatively easy to implement and therefore are applicable to the simulation of practical configurations. The three methods selected for further study are the one-sided, the PDD and the PTA methods. To our knowledge, the present work represents the first application of the PDD procedure to the parallel calculation of the complete Navier–Stokes equations. We are also not aware of any previous application of the PTA procedure in a generalized curvilinear coordinate framework, with or without the use of high-order filters.

This paper is organized as follows. The basic numerical procedures in the AFRL code, FDL3DI, are summarized in the second section, followed by a description of the parallel strategies in the third section. The third section is divided into two subsections, for the one-sided procedure and the parallel diagonal solvers, respectively. The descriptions of the PTA and PDD methods are presented in the latter. Results are given in the fourth section, first for the theoretical performance of the three parallel strategies and then for the kernel procedures developed for investigating the inherent performance of the methods. The last subsection in the fourth section pertains to the parallel results using FDL3DI. Conclusions are presented in the fifth section.

THE NUMERICAL MODEL

The compressible Navier–Stokes equations expressed in the strong conservation form for generalized curvilinear coordinates (ξ, η, ζ) are as follows:

$$\frac{\partial}{\partial t}\left(\frac{U}{J}\right) + \frac{\partial F}{\partial \xi} + \frac{\partial G}{\partial \eta} + \frac{\partial H}{\partial \zeta} = \frac{1}{\mathrm{Re}}\left[\frac{\partial F_v}{\partial \xi} + \frac{\partial G_v}{\partial \eta} + \frac{\partial H_v}{\partial \zeta}\right], \tag{1}$$

where $U = \{\rho, \rho u, \rho v, \rho w, \rho E_t\}$ is the solution vector, J is the Jacobian of the transformation, F, G, H are the inviscid fluxes and F_v, G_v, H_v are the viscous fluxes. The equations are closed with the perfect gas law and Sutherland's viscosity law.

In the framework of compact differencing, the derivative u' for any generic variable u in the transformed coordinate frame is represented as

$$\alpha u'_{i-1} + u_i' + \alpha u'_{i+1} = b\frac{u_{i+2} - u_{i-2}}{4\Delta\xi} + a\frac{u_{i+1} - u_{i-1}}{2\Delta\xi}, \tag{2}$$

where α, a, and b are constants which determine the spatial properties of the algorithm. The base compact differencing schemes used in this paper are the three-point, fourth-order scheme, C4, with $(\alpha, a, b) = (1/4, 3/2, 0)$, the five-point, sixth-order scheme, C6, with $(\alpha, a, b) = (1/3, 14/9, 1/9)$ and the five-point, fourth-order scheme, O5, with $(\alpha, a, b) = (0.430816, 1.6205440, 0.2410880)$. The latter scheme minimizes the dispersion error over the entire range of wave numbers up to two points per wave. Note that the symbol u above also represents components of vector quantities such as the F or G vector defined in Eq. (1).

Equation (2) is used to calculate the various derivatives in the (ξ, η, ζ) plane, as well as the metrics of the coordinate transformation. The inviscid fluxes in the Navier–Stokes equations are formed in the transformed coordinates at each nodal point and the components are differentiated using Eq. (2). The same coefficients (α, a, b) are used for both the metrics and the fluxes; this procedure reduces error on stretched and curvilinear meshes. To calculate the double derivative (e.g. the viscous and heat conduction) terms, the appropriate components of the primitive variable vector $W = \{\rho, u, v, w, T\}$ are first compact differentiated to form the stress tensor and heat flux vector. The viscous and heat conduction terms of the flow equations are then computed by another application of Eq. (2). The use of second derivative compact formulations, as given by Lele (1992), has some advantages but could lead to an excessive computational penalty in curvilinear coordinates.

The compact formulas for the physical boundary points are well known. The present work is based on the boundary formulas as given by Visbal and Gaitonde (1998) and Gaitonde and Visbal (1998). Time integration of the ODE that results from the spatial integration is based on the classical fourth-order Runge-Kutta (RK4) scheme.

Filters are employed to numerically stabilize the compact differencing calculations. In the formulation, the filtered values $\tilde{u}$ for any quantity u in the transformed space are represented as (Gaitonde and Visbal, 1998; 1999):

$$\alpha_f \tilde{u}_{i-1} + \tilde{u}_i + \alpha_f \tilde{u}_{i+1} = \sum_{n=0}^{N} \frac{a_n}{2}(u_{i+n} + u_{i-n}). \qquad (3)$$

This representation provides a non-dispersive, 2*N*th-order filter with a $2N+1$ point stencil. The eighth- and tenth-order filters are mostly used in the present work, for which the coefficients are tabulated by Gaitonde and Visbal (1998). Simulations with lower-order filters are also reported for the purpose of investigating the effect on accuracy and numerical stability in a multiblock and/or parallel environment.

PARALLELIZATION STRATEGIES

The One-sided Formulations

The basic algorithm for the one-sided approach was presented by Gaitonde and Visbal (1999). It involves the advancement of the solution independently in each subdomain with individual interior and boundary formulas in the same manner as in single-domain computations. Data is exchanged between adjacent subdomains at the end of each sub-iteration of the implicit scheme (or each stage of RK4), as well as after each application of the filter. Gaitonde and Visbal (1999) applied the interface algorithm to some 2D inviscid and viscous flow calculations using the compact scheme. It was found that the lower-order one-sided boundary scheme could cause a serious distortion of the flow structure and that this distortion could be reduced by superior higher-order one-sided filter formulas and a deeper overlap size.

Parallel Tridiagonal Solvers

Parallel tridiagonal solvers are an alternative to the one-sided multidomain formulations discussed in the last subsection. They are expected to recover the single-domain results if a conforming mesh system is used. With this requirement, there exist three candidate algorithms, i.e. the transposed (Cai *et al.*, 1997), the pipelined (Povitsky, 1998) or PTA and Sun's distributed PDD methods. The transposed approach (Povitsky, 1998) will not be discussed in this paper.

The Reduced PDD Algorithm

The reduced PDD algorithm was developed by Sun and Moitra, 1996 for the tridiagonal system

$$Ax = d, \qquad (4)$$

where $x = (x_1, \ldots, x_n)^{\mathrm{T}}$ and $d = (d_1, \ldots, d_n)^{\mathrm{T}}$ are n-dimensional vectors and A is a diagonally dominant tridiagonal matrix with order n:

$$A = \begin{bmatrix} b_0 & c_0 & & & & \\ a_1 & b_1 & c_1 & & & \\ & \cdot & \cdot & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & & & a_{n-2} & b_{n-2} & c_{n-2} \\ & & & & a_{n-1} & b_{n-1} \end{bmatrix}.$$

The reduced PDD algorithm partitions A into tridiagonal submatrices A_i and solves

$$(\tilde{A} + \Delta A)x = Ax = d \qquad (5)$$

based on algebraic matrix manipulation with $\tilde{A} \equiv \cup A_i$ being a block tridiagonal matrix with diagonal sub-matrices $A_i (i = 0, \ldots, p-1)$. To illustrate this matrix transformation, we assume $n = pm$ without loss of generality. The sub-matrices $A_i (i = 0, \ldots, p-1)$ are then $m \times m$ tridiagonal matrices. Note that ΔA denotes the matrix containing the extra entries in A that are not contained in $\cup A_i$. By some manipulations, we can express this matrix as follows:

$$\Delta A = [a_m e_m, c_{m-1} e_{m-1}, \ldots, a_{(p-1)m} e_{(p-1)m}, c_{(p-1)m-1} e_{(p-1)m-1}] \begin{bmatrix} e^{\mathrm{T}}_{m-1} \\ e^{\mathrm{T}}_{m} \\ \vdots \\ e^{\mathrm{T}}_{(p-1)m} \\ e^{\mathrm{T}}_{(p-1)m-1} \end{bmatrix} = VE^{\mathrm{T}},$$

where both V and E are $n \times 2(p-1)$ matrices and e_i denotes a column vector with its ith element being one and all the other entries being zero. Therefore, the matrix transformations can be written as

$$\begin{aligned} x &= (A + VE^{\mathrm{T}})^{-1} d \\ &= \tilde{A}^{-1} d - \tilde{A}^{-1} V (I + E^{\mathrm{T}} \tilde{A}^{-1} V)^{-1} E^{\mathrm{T}} \tilde{A}^{-1} d. \qquad (6) \end{aligned}$$

Let

$$\tilde{A}\tilde{x} = d, \tag{7}$$

$$\tilde{A}Y = V, \tag{8}$$

$$Z = I + E^{\mathrm{T}}Y, \tag{9}$$

$$h = E^{\mathrm{T}}\tilde{x}, \tag{10}$$

$$Zy = h, \tag{11}$$

$$\Delta x = Yy, \tag{12}$$

the solution is thus

$$x = \tilde{x} - \Delta x.$$

Note that each step in Eqs. (7)–(12) can be solved independently, block-by-block.

The implementation of the above algorithm can be described by the following steps:

1. Allocate A_i, $d^{(i)}$ and elements a_{im}, $c_{(i+1)m-1}$ to the ith block, where $0 \le i \le p-1$.
2. Solve $A_i[\tilde{x}^{(i)}, v^{(i)}, w^{(i)}] = [d^{(i)}, a_{im}e_0, c_{(i+1)m-1}e_{m-1}]$, where $0 \le i \le p-1$ [For Eqs. (7) and (8)].
3. Send $\tilde{x}_0^{(i)}$, $v_0^{(i)}$ from the ith block to the $(i-1)$th block, where $1 \le i \le p-1$.
4. Solve

$$\begin{bmatrix} 1 & w_{m-1}^{(i)} \\ v_0^{(i+1)} & 1 \end{bmatrix}\begin{pmatrix} y_{2i} \\ y_{2i+1} \end{pmatrix} = \begin{pmatrix} \tilde{x}_{m-1}^{(i)} \\ \tilde{x}_0^{(i+1)} \end{pmatrix},$$

where $0 \le i \le p-2$. [For Eq. (11)], since Z has the form with order of $2(p-1)\times(2p-1)$,

$$Z = \begin{bmatrix} 1 & w_{m-1}^{(0)} & & & & & & \\ v_0^{(1)} & 1 & 0 & w_0^{(1)} & & & & \\ v_{m-1}^{(1)} & 0 & 1 & w_{m-1}^{(1)} & 0 & & & \\ & \cdot & \cdot & \cdot & \cdot & \cdot & & \\ & & \cdot & \cdot & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & 1 & 0 & w_0^{(p-2)} \\ & & & & v_{m-1}^{(p-2)} & 0 & 1 & w_{m-1}^{(p-2)} \\ & & & & & 0 & v_0^{(p-1)} & 1 \end{bmatrix}$$

and $v_{m-1}^{(i)}$, $w_0^{(i)}$ are negligible for a diagonally dominant tridiagonal system when $m \gg 1$ (not more than 10 for Compact scheme).

5. Send y_{2i} from the ith block to the $(i+1)$th block, where $0 \le i \le p-2$.
6. Solve

$$\Delta x^{(i)} = \left[v^{(i)}, w^{(i)}\right]\begin{pmatrix} y_{2(i-1)} \\ y_{2i+1} \end{pmatrix},$$

where $0 \le i \le p-1$.

The PTA Procedure

To understand the PTA procedure (Povitsky, 1998), we first give the basic Thomas algorithm for the linear-algebra system:

$$a_{k,l}x_{k-1,l} + b_{k-1,l}x_{k,l} + c_{k,l}x_{k+1,l} = f_{k,l}, \tag{13}$$

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 from the compact scheme or the implicit filter scheme, $x_{k,l}$ are the unknown variables, and N_x, N_y and N_z are, respectively, the number of grid nodes in the x, y, and z directions.

The first step of the Thomas algorithm is LU factorization:

$$d_{1,l} = b_{1,l}, \quad d_{k,l} = b_{k,l} - a_{k,l}\frac{c_{k-1,l}}{d_{k-1,l}},$$
$$k = 2,\ldots,N_x, \tag{14}$$

and forward substitution (FS):

$$g_{1,l} = \frac{f_{1,l}}{d_{1,l}}, \quad g_{k,l} = \frac{-a_{k,l}g_{k-1,l} + f_{k,l}}{d_{k,l}},$$
$$k = 2,\ldots,N_x. \tag{15}$$

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

$$x_{N_x,l} = g_{(N_x,l)}, \quad x_{k,l} = g_{k,l} - x_{k+1,l}\frac{c_{k,l}}{d_{k,l}},$$
$$k = 1,\ldots,N_x - 1. \tag{16}$$

In the first step, the constants a_k, b_k, c_k are from compact schemes or implicit filters and the LU factorization is performed only once so that the FS dominates the computations. However, to proceed with the FS, the $(i-1)$th data is needed for the evaluation of the ith data. This is an unfavorable situation in parallel computations. For example, suppose a line l in the x-direction is split among the P processors and each processor holds $N = N_x/P$ data. Computing its part of the lth line, the pth processor: receives coefficient $g_{((p-1)N,l)}$ from the $(p-1)$th processor; computes the forward step coefficients $g_{k,l}$, where $k = (p-1)N+1,\ldots,pN$; sends coefficients $g_{(pN,l)}$ to the pth processor. Before the arrival of the data $g_{(pN,l)}$, the pth processor is idle. A similar situation

TABLE I Theoretical CPU times required by various schemes to invert a tridiagonal system of matrix equations

Algorithm	Computation	Communication	Idle
TDMA (Cai, 1999)	$N_1(5N-3)\gamma$	0	0
One-sided (Gaitonde and Visbal, 1999)	$N_1[5(N/P+N_2)-3]\gamma$	$(2\alpha+N_1N_2\beta)$	0
PTA (Povitsky, 1998)	$T_1=N_1(5N/P-3)\gamma$	$T_2=2k\alpha+2N_1\beta$	$(P-1)(T_1/k)$
PDD (Sun and Moitra, 1996)	$N_1(5N/P+3j+1)\gamma$	$2\alpha+2N_1\beta$	0

"j" Is the reduced number in Sun's PDD algorithm and "k" is the number of groups of lines (packets) in the PTA procedure.

occurs during BS. It is obvious that to obtain the least idle time, the pth processor should communicate with its neighbors only after the calculation of every line. The penalty for not doing this is that too many small messages are generated which introduces a significant amount of communication latency. A trade-off between the cost of idle time and the communication latencies can be achieved by finding the optimized number of lines in each message to be solved, which is represented in Eq. (4).

A few comments on the communication modes are in order before the results are presented. Two types of data are communicated: (1) data needed to evaluate the right-hand side of the global matrix system and (2) the data required in order to invert the tridiagonal matrix from compact differencing or filtering. All the procedures (PTA, PDD and one-sided) require communication to calculate the RHS vector. The stencil for the RHS vector could be quite large. Stencil size is determined by the RHS of compact differencing formula [Eq. (2)] and the RHS of the filtering operation in Eq. (3). Clearly, high-order filters will involve a large stencil. Not all the methods need to communicate any data in order to invert the tridiagonal matrix associated with the left-hand side (LHS) of Eqs. (2) and (3). Specifically, a subdomain in the one-sided method is self-contained with respect to the tridiagonal matrix, even if this matrix is approximate. Also, the coefficients of the matrix are constant. Therefore, the one-sided method does not communicate any data in order to invert A. For PDD and PTA, A is exact and global (not subdomain-based); a subdomain contains only a part of A and needs to communicate with neighboring subdomains during the matrix inversion process. For example, for PTA, the coefficients of A as well as the forward results ($g_{i,j}$) and the backward results ($x_{i,j}$) need to be communicated. For PDD, the data $\tilde{x}_0^{(i)}$, $v_0^{(i)}$, and y_{2i} need to be communicated. Note that the depth of communication is one grid point for the purpose of inverting the tridiagonal matrix. It is clear from the foregoing comments that the communication times stated for the one-sided procedure in Tables I and II are actually not used in the present implementation, but it is there in case the coefficients of A are nevertheless communicated.

RESULTS

Theoretical Performance of the Parallelization Strategies

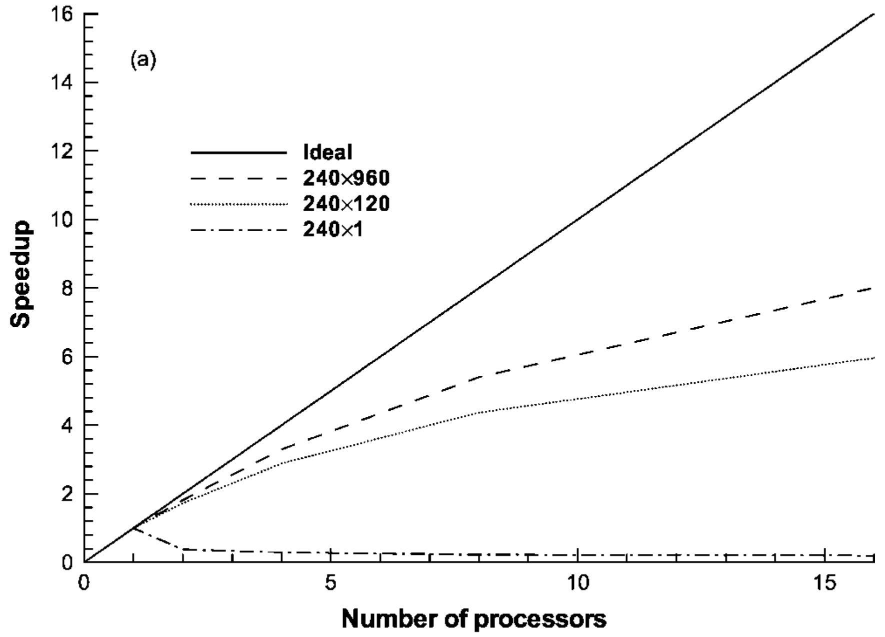
The theoretical performance of the three algorithms investigated in this paper is summarized in Table I. Table II contains the numerical values for the IBM SP2 system at Cornell, where the present calculations were done. While the data in Table I are CPU times, those in Table II are the CPU times normalized by those for the Thomas algorithm (i.e. the sequential computation). Thus, the data in Table II are actually those of speedup. They were generated for a kernel problem with $N=400$ (i.e. total number of nodal points in the x- or derivative-direction), $N_1=400$ (i.e. number of nodal points in the vertical or vector direction). N_2 in the table is the overlap depth in terms of grid points. The numbers in parentheses in Table II are the values actually observed (measured) in our numerical experiments. Note that the domain is decomposed only in the x-direction.

The system data for the IBM SP2 are: start-up latency, $\alpha = 55\,\mu\text{s}$, point-to-point communication, $1/\beta = 17.5\,\text{MWords/s}$, and time to perform one floating point operation, $\gamma = 1/65\,\mu\text{s}$. Note that $\beta = 1/17.5\,\mu\text{s}$, is the time required to send a double precision data. Thus, $\alpha/\beta \approx 966$, which is a large number (compared to unity), indicating that it is costly to initiate the process of sending a message (big or small) in this system and that messages should be bundled. The system is rated at 266 Mflops/s at peak performance, although the measured values are 65 Mflops/s (block tridiagonal matrix calculation) and

TABLE II Theoretical versus observed CPU times on IBM SP2 taken by various schemes to invert a tridiagonal system of matrix equations

Algorithm	$P=2$	$P=4$	$P=8$	$P=16$
TDMA	1	1	1	1
One-sided	1.954 (1.957)	3.826 (3.802)	7.342 (7.30)	13.585 (13.41)
PTA	1.53 (1.445)	2.29 (1.927)	3.07 (2.179)	3.69 (2.169)
PDD	1.89 (1.762)	3.59 (3.025)	6.52 (5.098)	11.01 (6.78)

Only the computation task is included in this table (i.e. no communication or idle time) and the numbers have been normalized by the CPU time for the Thomas algorithm. The numbers in parentheses are the observed (measured) values.



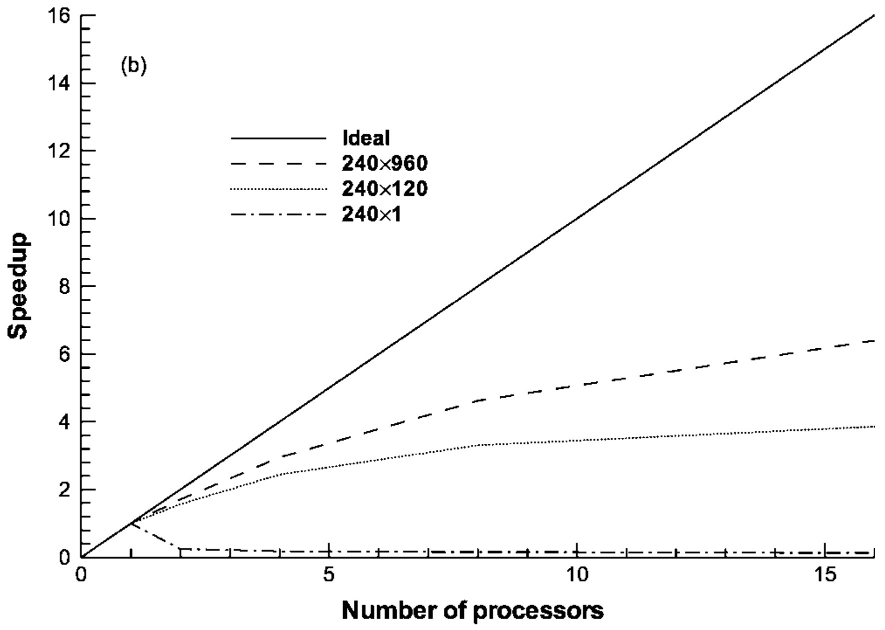
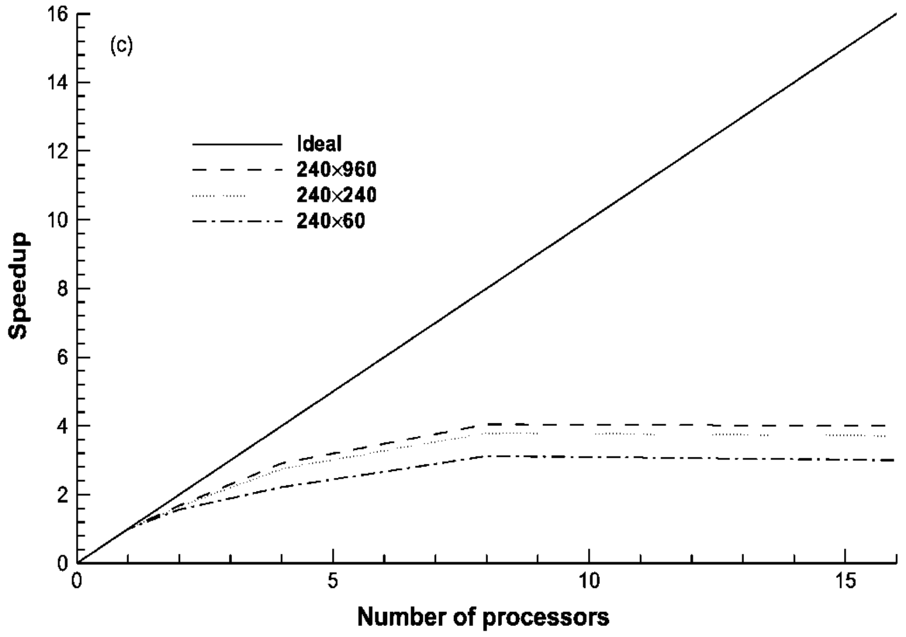


FIGURE 1 Speedup and scalability of parallel computations with a kernal code for PDD: (a) Speedup of the multiple-RHS reduced PDD algorithm. Includes the time for the preparation of TDMA; (b) Speedup of the multiple-RHS reduced PDD algorithm. Excludes the time for the preparation of TDMA; (c) Speedup of the single-RHS reduced PDD algorithm. Includes the time for the preparation of TDMA; (d) Scalability of the multiple-RHS and the single-RHS reduced PDD algorithm without the time for the preparation of TDMA (Problem size in each processor: 60 × 120); (e) Scalability of the multiple-RHS reduced PDD algorithm for different problem sizes.

85 Mflops/s (multi-grid calculation). In Tables I and II, P denotes the number of processors, k is the number of groups of lines solved per packet in the pipelined algorithm and j is the reduced number in the Sun's algorithm, usually no larger than 10 for the compact difference scheme. Note that the optimal parameter k can be expressed as

$$k = \left\lceil \frac{N_1}{\sqrt{\frac{N_1 \cdot P/N \cdot \nu}{\rho(P-1)}}} \right\rceil, \tag{17}$$

where $\nu = \alpha/g_2$, $\rho = g_1/g_2$, and g_1 and g_2 are the forward and backward calculation times for the TDMA per grid point. To produce Tables I and II, we choose Sun's reduced PDD approach to represent the distributed algorithms, which appears to be the most efficient parallel solver for diagonal dominant tridiagonal systems in this category (Table I), and the unoptimized Povitsky method to represent the pipelined algorithms. The transposed algorithm (not shown) was originally developed by Cai *et al.* (1997) for FFT but has also been applied to the tridiagonal system at hand. The "Thomas" algorithm in



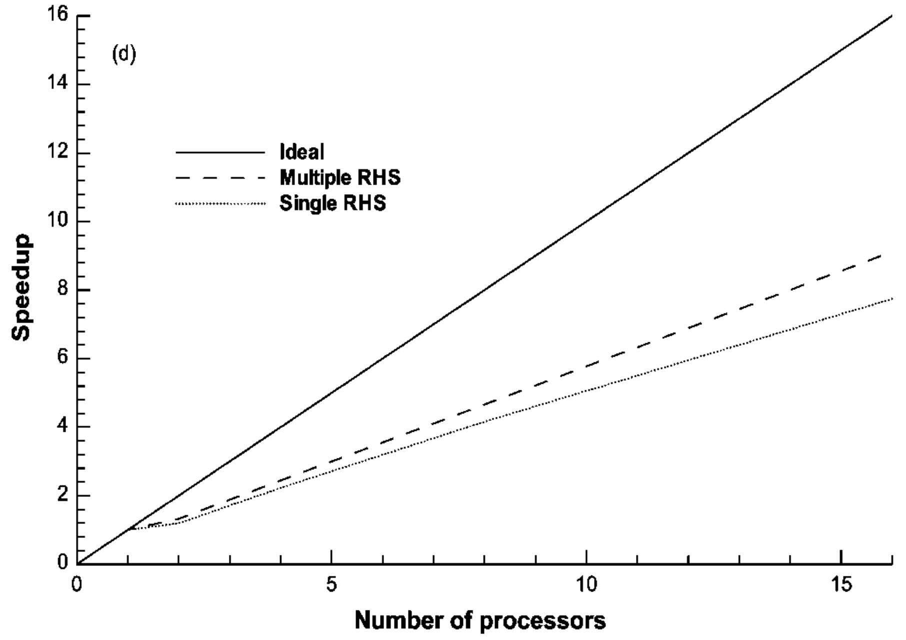


FIGURE 1 Continued.

the table is the standard (sequential) tridiagonal Thomas algorithm (TDMA) for solving the tridiagonal system of matrix equations. From Table I, it can be seen that Sun's algorithm incurs a smaller communication cost compared to PTA, and therefore should be preferred on machines capable of handling computations much faster than they do communication.

Parallel Performance Studies with Kernel Codes

The ultimate goal of the present project is to produce an efficient and scalable parallel version of the AFRL code, FDL3DI. In order to choose among the three parallel strategies, it became necessary to carry out some basic investigations of each of the strategies. These procedures, which we refer to as kernel procedures, focus on the parallelization of the basic tridiagonal system of equations, without the details of the Navier–Stokes equations or the manner in which the tridiagonal system was generated. In the next two subsections, we report on the performance of PDD and compare the parallel performance of PTA to those of PDD and the one-sided approach.

Performance of PDD

The PDD algorithm involves a large amount of communication relative to the one-sided algorithm and

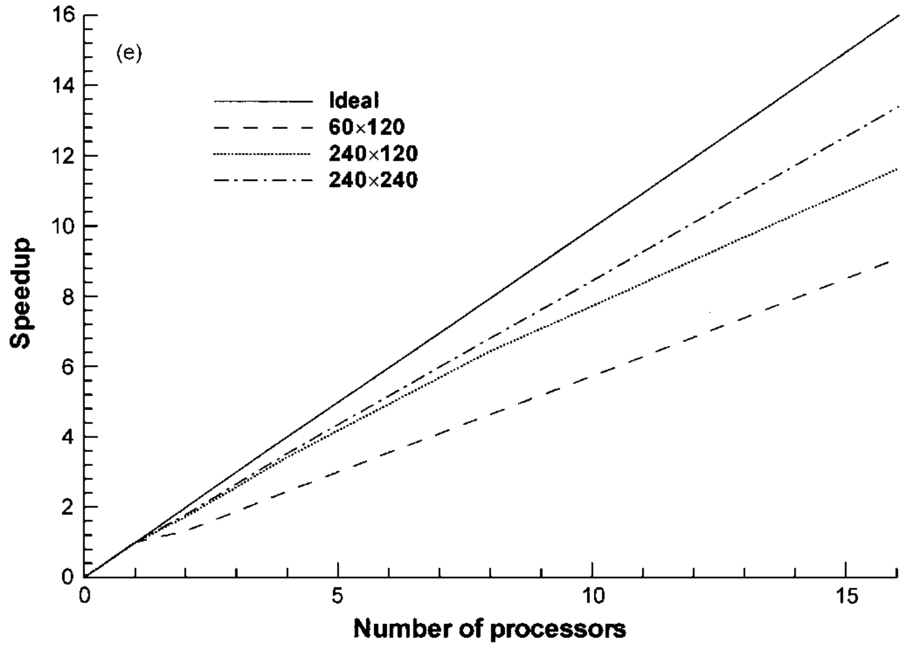


FIGURE 1 Continued.

hence, as shown later in this paper (Table V), shows up very poorly. It therefore became necessary to examine this procedure in more detail, for some insight into the factors that determine its performance. For this purpose, a kernel code suffices, and was developed as a stand-alone code with C4 compact differencing. No filters were used for the kernel codes reported in this paper. To analyze the x-difference, $\partial/\partial x$, the domain is split in the x-direction, and the y-direction is treated as the "vectorization" (vertical) direction. That is, the difference along various lines in y are computed via the multiple right-hand side (RHS) procedure in which the number of RHS columns is equal to the number of lines (nodal points) in y. The x-difference for all x–y planes is obtained by looping over the z-direction and repeating the above procedure. The y- and z-differences, $\partial/\partial y$, $\partial/\partial z$, are analyzed similarly. The results in Fig. 1 pertain only to the x-difference and one x–y plane. Hence, the two dimensions shown for the grid in the figure legend (for example, 240×960). The first index is the x-direction, the second is the "vectorization" (vertical) direction. The results for all z lines can be obtained by a simple scaling of the reported results which, incidentally, is a procedure that is quite consistent with the setup of the FDL3DI code.

Figures 1(a–c) and (d, e) show the speedup and scalability results, respectively. With the number of x grid points fixed, increasing the number of processors reduces the speedup [Fig. 1(a–c)], as would be expected based on the increased amount of communication relative to the time for useful calculations. However, of interest is the fact that increasing the number of RHS columns leads to better performance, both in terms of speedup and scalability. Scalability is calculated as speedup with the number of grid points fixed for each processor, and the number of processors (and hence the problem size) is varied. A highly scalable process will show a straight line with near-unity slope.

Performance of PTA

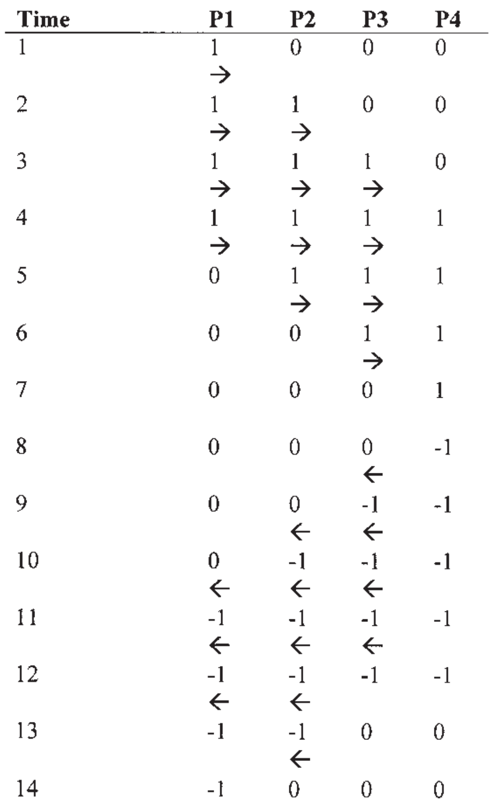
The performance of PTA relative to that of PDD and the one-sided scheme is tested on a kernel problem which is a tridiagonal system with $N_x = 60$, $N_y = 120$ in each processor. Three identical equations were solved instead of one, to mimic the Navier–Stokes equations. The effective N_y [or N_V in Eq. (20) below] is, therefore, equal to 3×120. Some system parameters are required to analyze the performance of PTA. The communication time for a single message between two processors in the network can be approximated by the following linear expression:

$$f(L) = \alpha + L\beta, \tag{18}$$

where L is the number of bytes (words) per message. The computation times at a single point for the FS and BS are g_1 and g_2, respectively. Therefore, the additional (i.e. communication plus idle) CPU time required for PTA can be written as

$$F = \lceil N_V/k_1 \rceil (\alpha + k_1\beta) + \lceil N_V/k_2 \rceil (\alpha + k_2\beta) \\ + (P - 1)N_V(k_1 g_1 + k_2 g_2), \tag{19}$$

where $N_V = N_y \times N_z$, and k_1 and k_2 are the number of lines solved per packet for the forward and backward step computations. (Note that in Table I, it has been assumed



Time	P1	P2	P3	P4
1	1 →	0	0	0
2	1 →	1 →	0	0
3	1 →	1 →	1 →	0
4	1 →	1 →	1 →	1
5	0	1 →	1 →	1
6	0	0	1 →	1
7	0	0	0	1
8	0	0	0 ←	-1
9	0	0 ←	-1 ←	-1
10	0 ←	-1 ←	-1 ←	-1
11	-1 ←	-1 ←	-1 ←	-1
12	-1 ←	-1 ←	-1	-1
13	-1	-1 ←	0	0
14	-1	0	0	0

FIGURE 2 A four-processor sample schedule for the PTA method. The first column is the time sequence while the 2nd to 5th represent the four processors. "0", "1", and " − 1" denote idle stage, forward and backward computations, respectively; →, ← denote send and receive communication, respectively. ($P = 4$, $k_1 = 27$, $k_2 = 31$, $Nv = 120$).

that $k_1 = k_2 = k$). From Eq. (19), the optimal k_1 and k_2 are thus obtained:

$$k_1 = \sqrt{\frac{N_V/N \cdot \sigma_1}{\sigma_2(P-1)}}, \quad k_2 = \sqrt{\frac{N_V/N \cdot \sigma_1}{(P-1)}}, \qquad (20)$$

where $\sigma_1 = \alpha/g_2$ and $\sigma_2 = g_1/g_2$, both of which are machine-dependent parameters. Numerical experiments on the IBM SP2 at the Cornell Theory Center have found that

$$\sigma_1 \simeq 487, \quad \sigma_2 \simeq 1.35. \qquad (21)$$

Figure 2 shows a schedule for the communication and computations within a pipeline, using four processes for the illustration. Lines are gathered in four packets in the forward direction and also in four packets (by chance) for the backward direction. Zeros denote the idle time. Note that the k_1 and k_2 values in this figure, i.e. 27 and 31, respectively, are not necessarily those of Povitsky because they depend on the number of grid points, as per their definitions in Eq. (20) above. It can be seen from Fig. 2 that the idle time and thus inefficiency occurs at the beginning of the forward computations, at the end of the backward computations, and during the switch from the forward to the backward steps. Many techniques have been devised to utilize this idle time. One possibility is to perform data-independent calculations, such as the evaluation of the derivatives for the other directions. A more elaborate approach is proposed by Povitsky, which reschedules the pipeline so that the local but data-dependent Runge-Kutta procedure can be done at the stage where the standard PTA is idle. However, these corrections suffer from the complex code structure by mixing the tridiagonal solvers with the other procedures.

TABLE III The optimal number of lines for FS and BS in PTA

Processors	2	4	8	16
(k_1, k_2)	(47, 54)	(27, 31)	(18, 21)	(12, 14)

Table III presents the optimal k_1 and k_2 for our tests. As these values are functions of some parameters dependent on the "now/then" status of the machine, it is not possible to achieve exactly the least CPU time at each run. For example, Fig. 3(a) presents the CPU times for the case with a different number of lines in the backward-step packets when 16 processors are involved. It can be seen that the optimal k_2 from Eq. (20) corresponds to the case with almost, but not exactly, the least CPU time.

The comparison of speedup from various parallel algorithms is shown in Fig. 3(b). From this figure, it can be seen that all of the three parallel strategies have good scalability but the PTA procedure has the least speedup. However, the advantage of PTA is that it can recover the results of a serial Thomas algorithm exactly while the parallelized one-sided method for both the filter and difference schemes might lose some accuracy near the region of inter-domain interfaces.

Parallel Computations with FDL3DI

In order to establish the feasibility of the parallel strategies discussed in this paper, a simple two-dimensional vortex advection problem was analyzed using the simplest possible partition–a one-dimensional partition in the x-direction. Both sequential multiblock and parallel procedures were implemented in the AFRL code FDL3DI and the computations were carried out on the IBM SP2 system at the Cornell Theory Center. To explore the issues involved, consider the unsteady inviscid flow resulting from a convecting vortex in an otherwise uniform flow at a freestream Mach number $M_\infty = 0.1$. The initial flow field and vortex strength are set as in Visbal and Gaitonde (1998). The domain considered is $-6 < X < 18, -6 < Y < 6$. The z-direction is a dummy for which 7 grid points are used. To test for the accuracy of the calculations, we use a uniform mesh of $60 \times 30 \times 7$, and two processors. The domain decomposition is located at $X = 6$. Figure 4 displays the vorticity contour maps obtained for various compact and filter schemes and boundary condition types (i.e. periodic versus non-periodic).

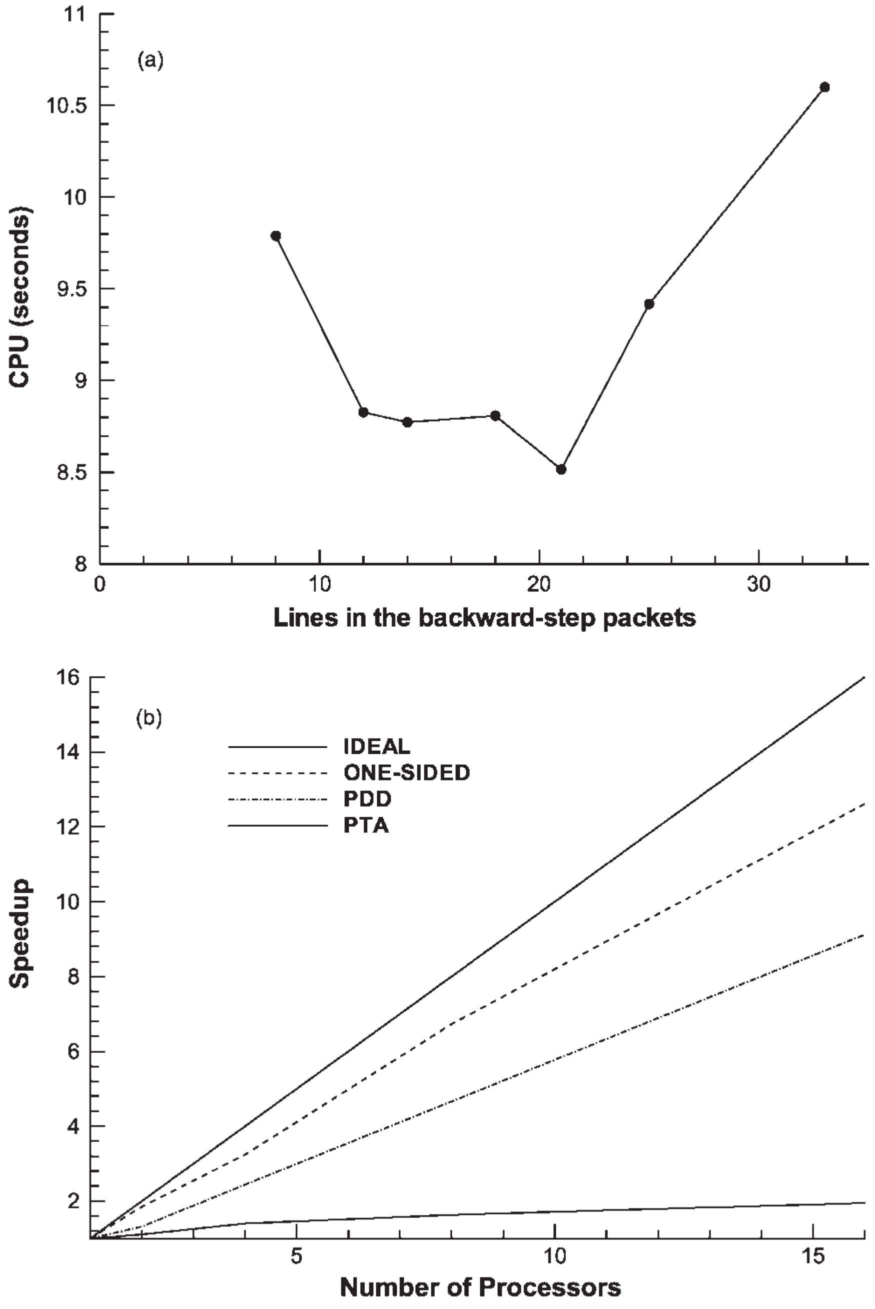


FIGURE 3 CPU time performance of PTA and a comparison of the speedup of PTA with those of OOP and PDD: (a) uses 16 processors with 120×60 points in each processor; (b) also uses 120×60 grid points in each processor.

The notations used in Gaitonde and Visbal (1998) will be adopted to simplify the reference to the different compact and filter schemes. When periodic conditions are employed, the notation C*n* will be used to denote an *n*th-order compact differencing scheme. When the specification of boundary treatment is important, the precise scheme will be denoted with five numbers, a, b, c, d, e where a through e are numbers denoting the order of accuracy of the formulas employed, respectively, at points 1, 2, {3,. . .,IL-2}, IL-1 and IL, where IL is the number of nodal points. For example, 24642 consist of a sixth-order interior compact formula, which degrades to 4th- and 2nd-order accuracy near the boundary. For the filter scheme, the interior order will be designated with its order of accuracy superscripted with the value of a parameter, α_f. For example, $\mathrm{F10}^{0.3}$ represents a tenth-order filter with $\alpha_f = 0.3$. When boundaries are present, the notation for the boundary filter formulas consists of simply appending the interior filter with the order of accuracy of the formulas at each of the boundary points 1,2,. . . in sequence. If a point is left unfiltered, the value 0 is utilized. For example, $\mathrm{F10}^{0.4} - 0$, 6, 8, 8, 8 consists of an $\alpha_f = 0.4$ tenth-order interior filter formula which degrades to 8th-, 8th-, 8th-, and 6th-order accuracy as the boundary is approached

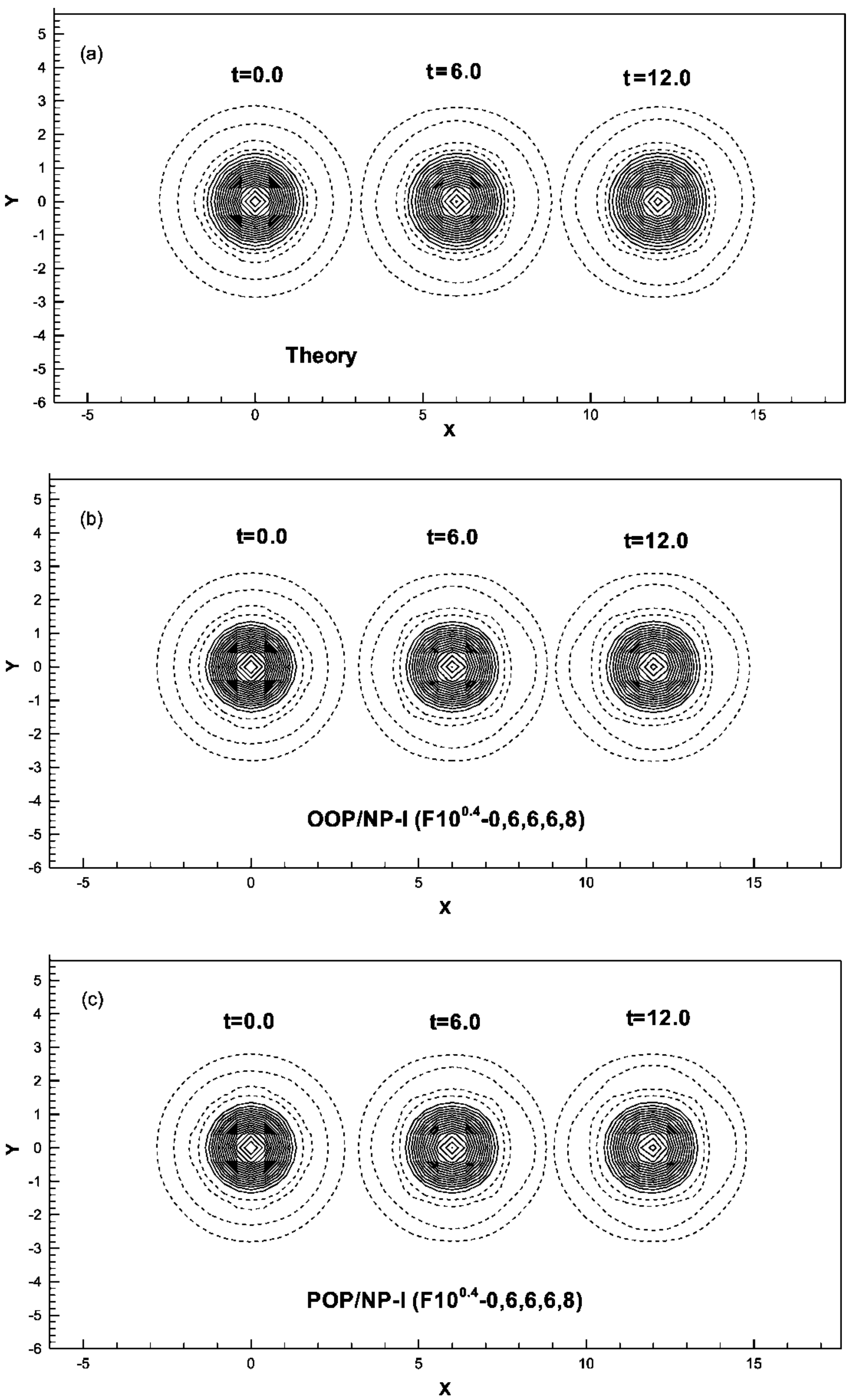
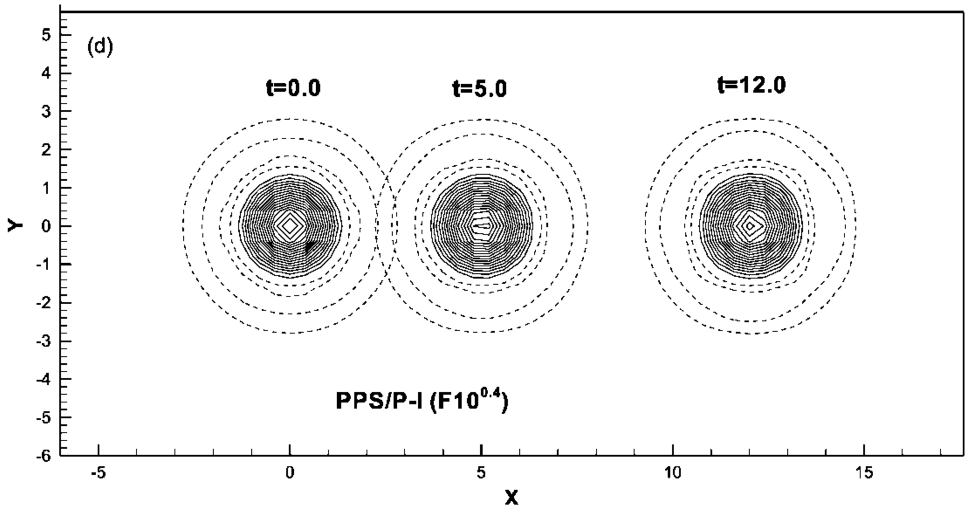
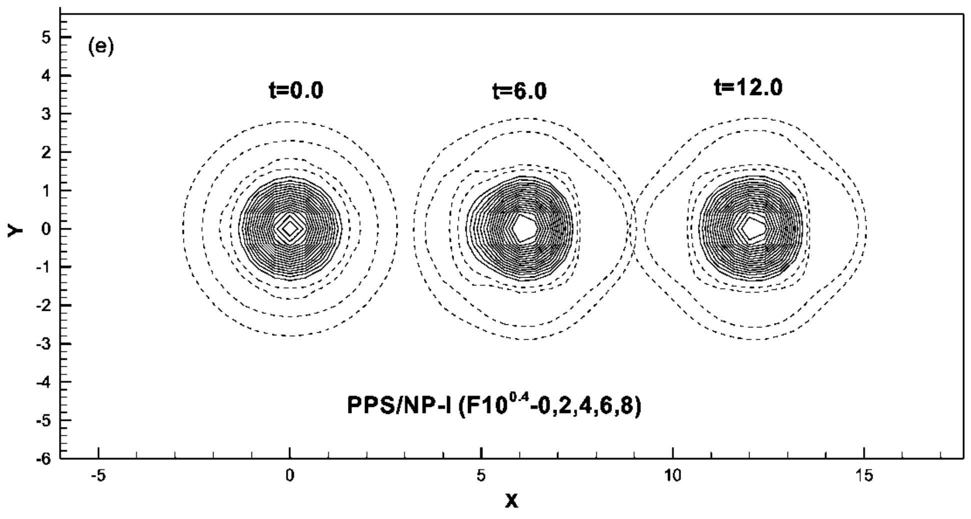


FIGURE 4 Parallel vortex advection computations with FDL3DI, showing qualitative accuracy in procedures that use the one-sided method, the PDD method, and their combination for compact differencing and filtering. The abbreviations used in the figure are defined in the text.

from the interior. For this example, the point on the boundary is not filtered. Furthermore, some of the case studies represent a permutation from the following schemes:

1. Compact differencing using either one-sided (symbol "O") or the PDD algorithm (symbol "P").
2. Compact filtering using either one-sided (symbol "O") or the PDD algorithm (symbol "P").





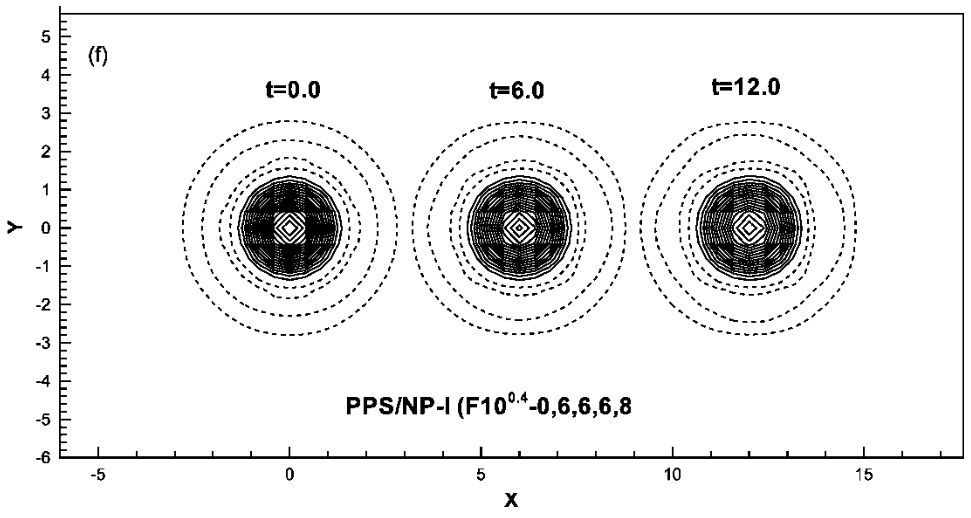


FIGURE 4 Continued.

TABLE IV (a) Total CPU time, T_1, (s) for sequential (multiblock) implementation with two subdomains in x and a grid size of $(N_x, N_y, N_z) = (60 \times 30 \times 7)$. (b) Total CPU time, T_n, (s) for parallel implementation with two processes in x and a grid size of $(N_x, N_y, N_z) = (60 \times 30 \times 7)$. (c) Total CPU time, T_n, (s) for parallel implementation with several process topologies and a grid size of $(N_x, N_y, N_z) = (240 \times 120 \times 14)$

	Periodic (no filter)	Non-periodic (no filter)	Periodic (with filter)	Non-periodic (with filter)
(a)				
OOS/*-I	–	10.71	–	11.84
PPS/*-I	7.69	7.50	8.75	8.66
SD/*-I	7.61	6.09	8.65	7.89
SD/*-V	36.70	32.81	37.75	33.86
(b)				
OPP/*-I	–	3.64 (3.55)	–	4.25 (4.03)
POP/*-I	–	5.89 (3.95)	–	6.31 (4.66)
PPP/*-I	–	5.76 (3.95)	–	6.46 (4.44) (LOC)
PPP/*-V	–	30.87 (19.74)	–	30.84 (20.13)
(c)				
SD/*-I	342.72	305.38	401.47	343.38
OOP/*-I ($2 \times 1 \times 1$ procs)	–	153.11	–	173.72
OOP/*-I ($2 \times 2 \times 1$ procs)	–	79.82	–	90.84
OOP/*-I ($2 \times 2 \times 2$ procs)	–	56.58	–	65.03
OOP/*-I ($4 \times 2 \times 2$ procs)	–	27.52	–	30.92

3. Execution in a sequential multiblock mode (symbol "S") or parallel mode (symbol "P").
4. Periodic and non-periodic boundary conditions (symbols "P" and "NP").
5. Viscous and inviscid computations (symbols "V" and "I").

The notation for a case study is presented in the ordered form $G_1G_2G_3/G_4 - G_5$, where "G_i" stands for the appropriate notation for index "i" chosen from the five schemes above. As an example, the notation POS/P-I implies a case study that uses the PDD algorithm for compact differencing, one-sided algorithm for the filtering, is executed sequentially, involves the use of periodic boundary conditions, and pertains to inviscid computations. Note that the notation for the case study excludes information on the specific compact differencing or the filter scheme, whose notations have been defined above. The use of "SD" in place of $G_1G_2G_3$ implies a single domain calculation. For example, SD/NP-V implies a single domain calculation for a viscous, non-periodic problem. Note that the one-sided and PDD algorithms pertain to multiblock or parallel implementation of TDMA.

The calculation for the vortex advection problem uses the C4 compact scheme coupled with an interior filter of tenth-order accuracy and an LOC (lower-order centered) boundary filter scheme, $\mathrm{F10}^{0.4} - 0,\ 2,\ 4,\ 6,\ 8$. Certain observations are apparent from Fig. 4. The one-sided multiblock schemes, when used for compact differencing and filtering, give acceptable results for parallel computations [Fig. 4(b)]. The combination of PDD for compact differencing and the one-sided scheme for filtering also shows similar behavior [Fig. 4(c)]. However, the case studies that use PDD for both compact differencing and filtering in parallel mode, i.e. PPP, calculate totally wrong results (not shown) when low-order filters such as $\mathrm{F4}^{0.4} - 0,\ 2$ were used. The use of high-order filters for PPP proved to be too expensive and hence could not be pursued. The sequential implementation, PPS, gave accurate results, provided a high-order filter is used. PDD for compact differencing, without filters, works, but is of little interest since filters must be used for most of the Navier–Stokes problems. Finally, in the present study, a successful implementation of PDD in FDL3DI was feasible only when the procedure was combined with the one-sided method, as in POP.

Table IV shows the CPU time performance for selected cases from the inviscid and viscous vortex advection calculations with FDL3DI using the mesh $(60 \times 30 \times 7)$ and carrying out the calculations for five time steps. Tables IV(a) and (b) use two processors whereas Table IV(c) uses 2, 4, 8, and 16 processors. Although viscous calculations of vortex advection are shown in the tables, the intention is not to assess the accuracy, as there are no analytical solutions for the viscous vortex advection case. A comparison of the relative CPU time for the viscous and inviscid calculations is of interest. In the current implementation, the tables show that the cost of the viscous calculations is quite high relative to that for the inviscid calculations. The numbers in the table show that adding filters might not lead to too much computational overhead for realistic problems, which are invariably viscous. Note that the data in parentheses in Table IV(b) excludes the communication time.

Parallel efficiency, T_1/nT_n, speedup $S_n = T_1/T_n$, and overhead, T_c/T_n can be obtained from the data in the table, where n, T_1, T_n, and T_c are the number of processors, the CPU time for a sequential, single-domain calculation, the CPU time for n processors, and the communication time, respectively. In Table IV(b), T_c

TABLE V Performance data for sequential and parallel (OOP) calculation of laminar boundary layer

Proc. No.	1	2	2	4	4	8	8	16
Proc. dim.	(1,1,1)	(2,1,1)	(1,2,1)	(4,1,1)	(2,2,1)	(2,4,1)	(2,2,2)	(2,4,2)
OOP CPU	41.92	43.82	44.85	45.33	48.43	50.29	53.03	54.37
Seq. Grid	54000	108000	108000	216000	216000	432000	432000	884000
Seq. CPU	42.79	93.82	93.33	192.64	198.04	410.07	419.92	862.86

For OOP, the size of the grid in each processor is $60 \times 30 \times 30$. For the sequential calculations, the grid points in (x, y, z) corresponding to each of the eight parallel cases are $(60, 30, 30)$, $(120, 30, 30)$, $(60, 60, 30)$, $(240, 30, 30)$, $(120, 60, 30)$, $(120, 120, 30)$, $(120, 60, 60)$ and $(120, 120, 60)$. The abbreviations "Proc", "dim" and "seq" in the table denote "processor", "dimension" and "sequential", respectively.

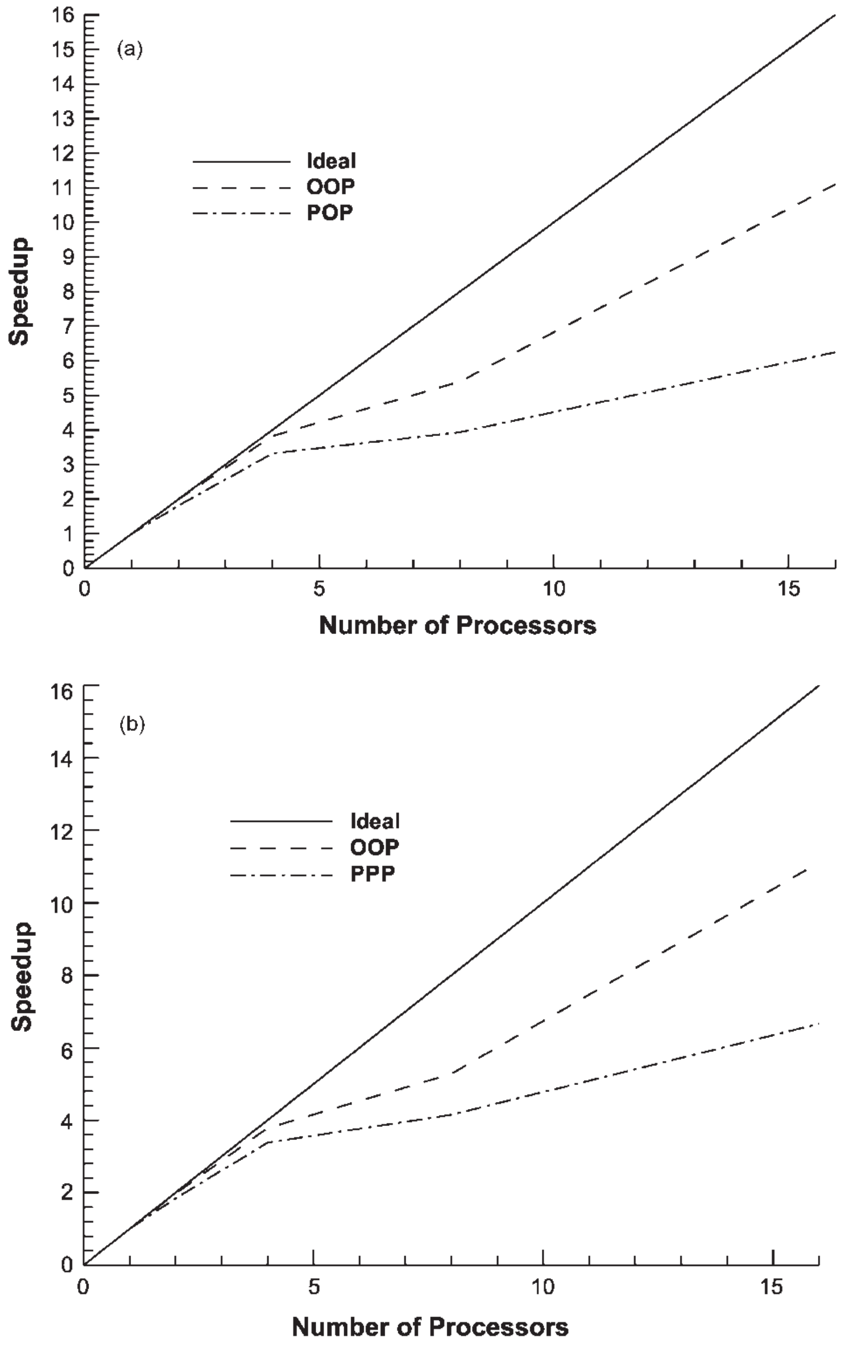


FIGURE 5 The speedup associated with the vortex advection computations with FDL3DI: (a) filter turned off (b) filter turned on.

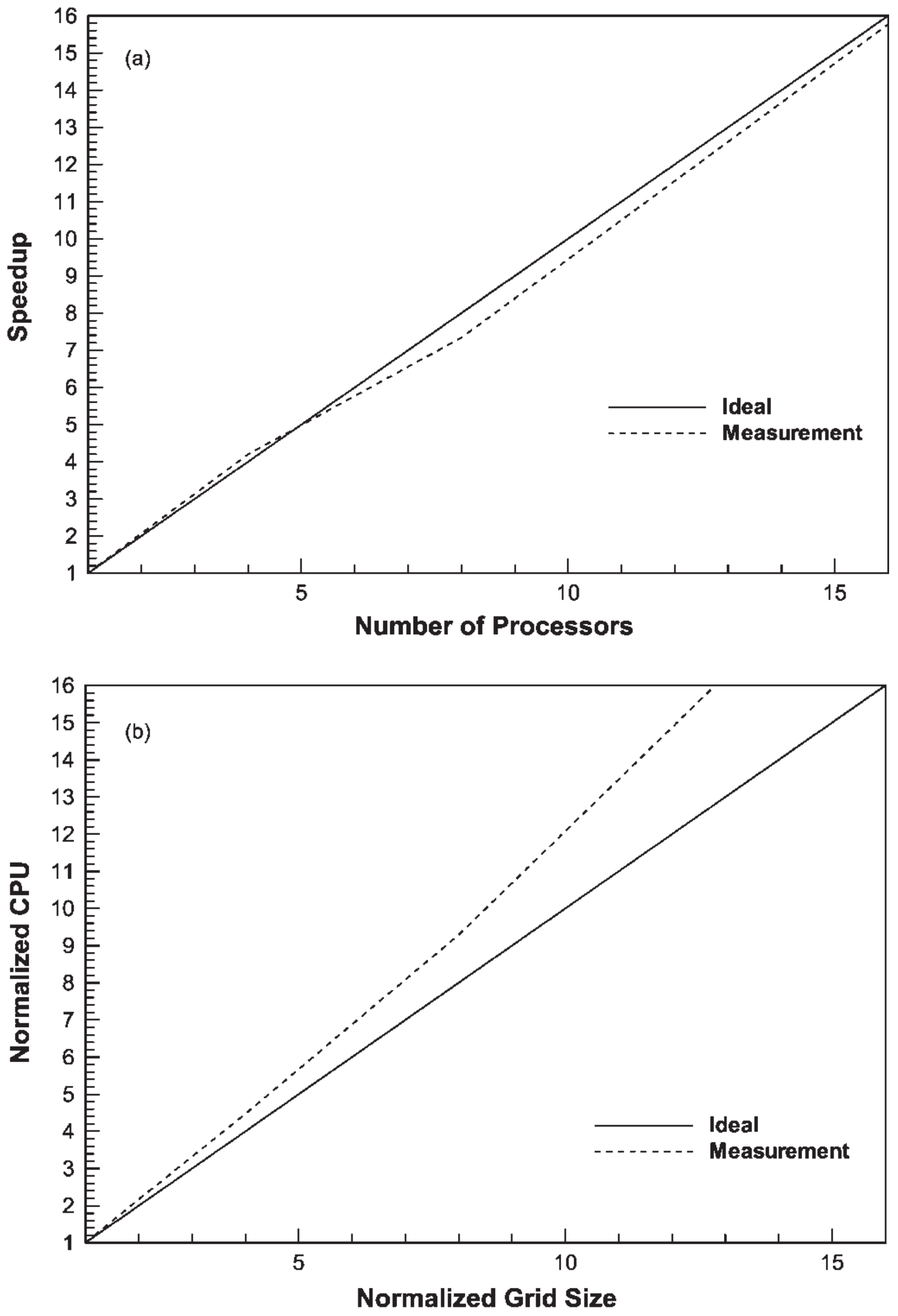


FIGURE 6 Parallel performance of the boundary layer computations with FDL3DI using the OOP parallelization strategy. Note that Fig. 6(b) pertains to the single-domain computations.

for a case study is obtained by subtracting the data in parentheses. From Tables IV and V, only the case study OOP shows interesting parallel performance (efficiency of 88% and 92% for filtered and unfiltered cases, respectively). Efficiency values of 98% and 99% can be observed in Table IV(c) for the case of two processors. Due to the increased communication cost relative to the cost of computation, the parallel performance deteriorates with increasing number of processors since the problem size is fixed. A second reason is that the overlapped depth becomes relatively larger. Some of these observations are contained in Fig. 5(a and b), which show the speedup (without filter), using FDL3DI. The results were also obtained with five time steps for a system with a grid size of $240 \times 120 \times 14$. (The grid size of $60 \times 30 \times 7$ is too small for meaningful parallel computing.)

For the cases shown in the figures, the performance is excellent until $P = 8$ when the z-direction goes through decomposition. Obviously, more grid points are needed in the z-direction in order to obtain a more reasonable performance than the one shown in Fig. 5(a and b). The deviation from the idealized speedup also results

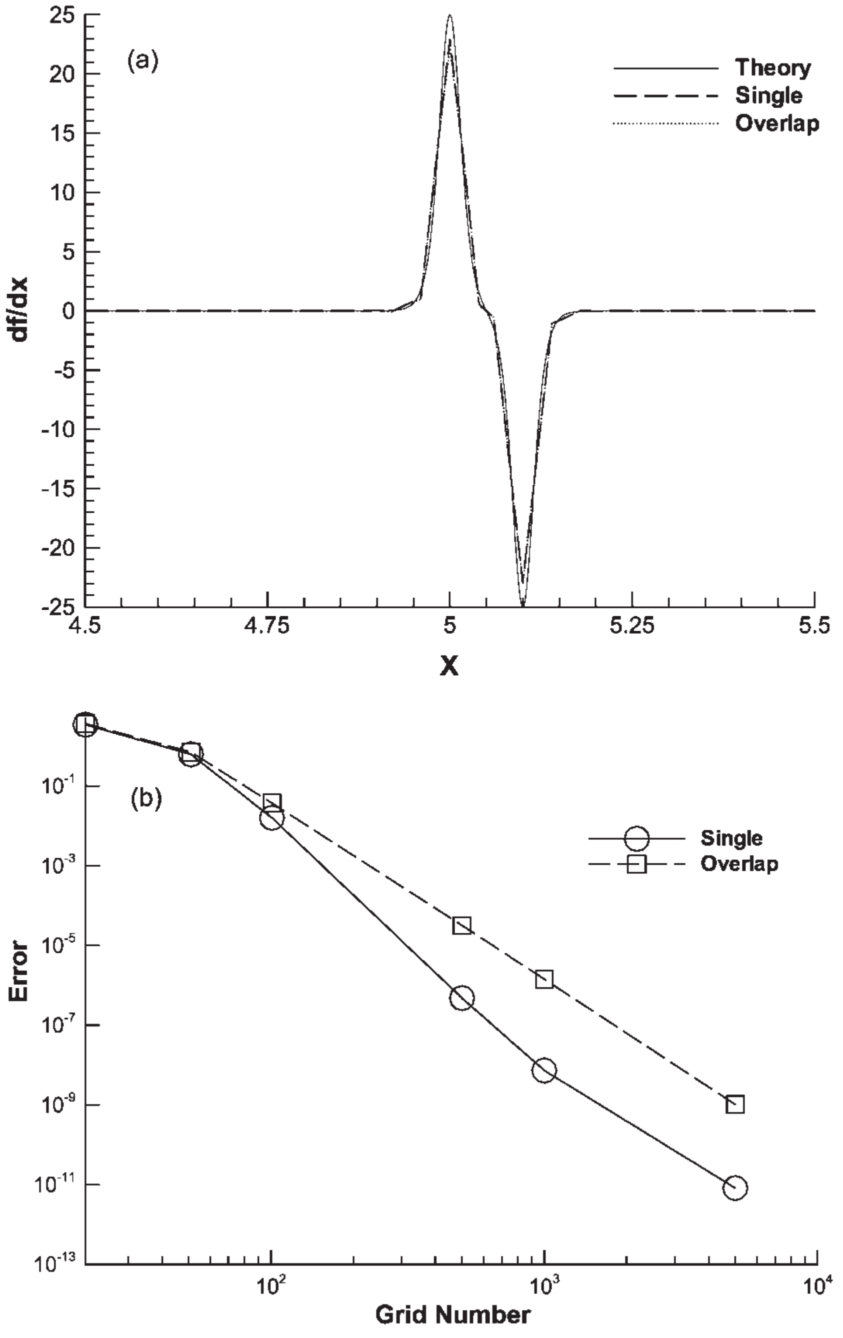


FIGURE 7 Assessment of the error from the interface treatment compared to the single-domain results: (a) Results of the gradient of a sharp signal using a total of 51 grid points with five in the overlapped region; (b) Euclidean norm error versus grid numbers.

from the fact that OOP requires extra calculations for the overlap region. The same situation occurs for PPP. Although the performance of PPP is worse than that of OOP, the situation improves (Fig. 1) as the grid size becomes large, with PPP showing improved performance. The practical observation is consistent with the theoretical analysis shown earlier in this paper. The same can be said of POP since the scheme is exactly PPP with the filter turned off in the calculations. Figure 5(b) shows similar performance as the filtering scheme is turned on.

The parallel performance of the one-sided schemes (OOP), which Fig. 5 shows to be superior to that of PPP, is further investigated for 3D splitting in which each processor computed the grid $60 \times 30 \times 30$ in (x, y, z). This is the useful grid in that it excludes two of the five grid points shared with the neighboring processor in the overlap region on each side of a subdomain. Another reason for our interest in OOP is the simplicity of the procedure, which means that it can be applied to realistic systems. The physical domain calculated here is a laminar boundary layer flow. Sequential calculations

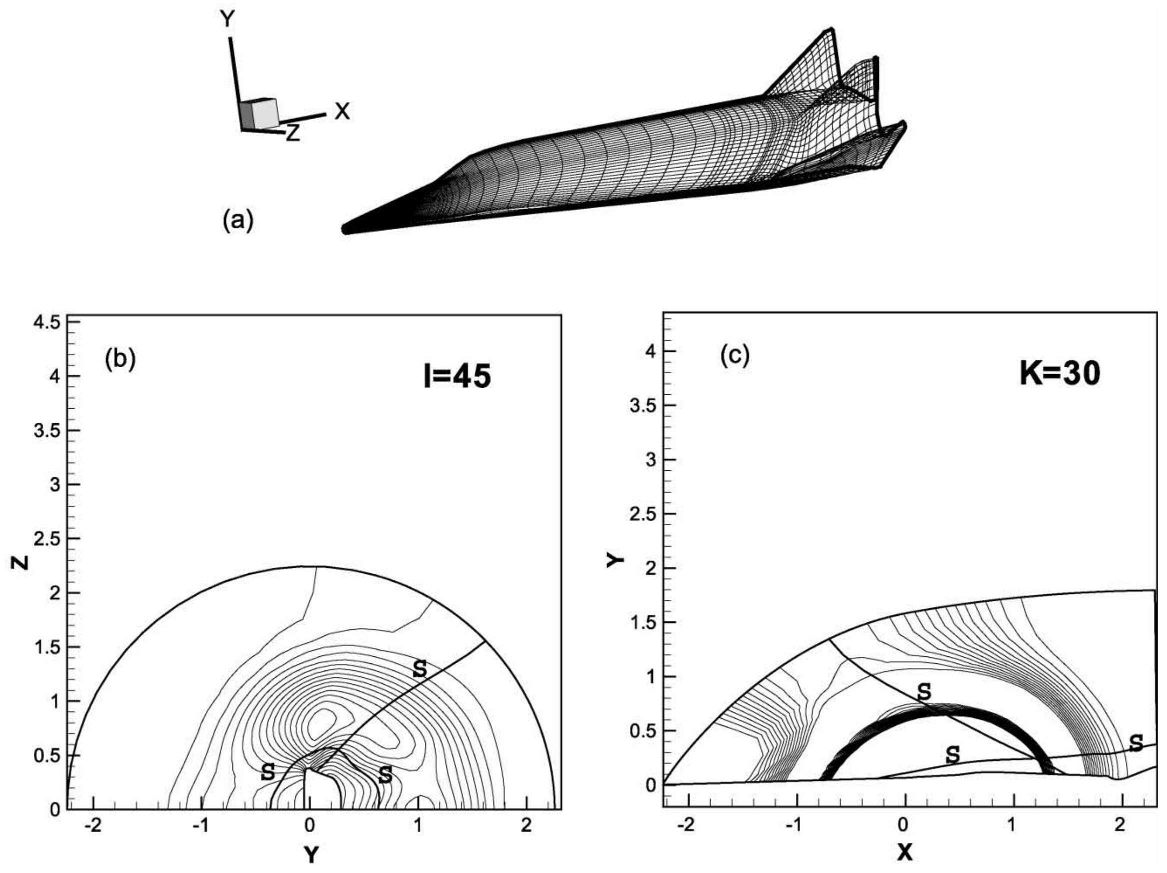


FIGURE 8 (a) Coarse grid model of X24C reentry vehicle showing the surface mesh ($J = 0$); (b) Acoustic pressure distribution on surface along the X (axial) direction; (c) Acoustic pressure distribution on surface along the Z (circumferential) direction. The lines labelled "S" in the figure represent the interface between subdomains in the parallel implementation.

corresponding to each parallel case are needed in order to calculate speedup. The base sequential mesh is $60 \times 30 \times 30$ or 54×10^3 grid points. The results, which are shown in Fig. 6(a and b) are quite interesting, as discussed below. The grid layout in the (x, y, z) directions of the sequential mesh is chosen to mimic the processor decomposition for OOP. Thus, for the OOP decomposition $2 \times 4 \times 1$, for example, the grid layout for the sequential calculation is $(120, 120, 30)$, or $(60 \times 2, 30 \times 4, 30 \times 1)$, which is 432×10^3 nodal points. Table V shows that processor decomposition [e.g. $(2, 1, 1)$ versus $(1, 2, 1)$] does not significantly affect the CPU time performance. Also, for the sequential calculations, the total number of grid points, not the grid layout in (x, y, z), governs the performance. Scalability results are presented in Fig. 6(a), wherein the speedup is plotted against the number of processors; each processor calculates $60 \times 30 \times 30$.

In Fig. 6(b), the CPU time performance is presented as a function of the number of grid points for the sequential calculations. The (ideal) solid line in this figure is based on a linear scaling of the CPU time with the grid point, using the grid $60 \times 30 \times 30$ (or 54×10^3 grid points) as the base for the extrapolation. It is evident that the observed CPU performance (dashed line) does not scale linearly with the number of grid points. In general, if the CPU time for the sequential calculation of the base grid (54×10^3 grid points) is T_0, that for the sequential calculation of $54 \times n \times 10^3$ grid points, say T_{sn}, is greater than nT_0, as shown by the larger values of the dashed line data over the corresponding solid line results [Fig. 6(b)]. The speedup is $T_{sn}/T_0 + T_c$. That is, the speedup can go above n [Fig. 6(a)], depending on the values of $(T_{sn} - nT_0)$ relative to T_c. Note that for all cases, T_0 is the same because, even though the size of the sequential problem (and hence T_{sn}) changes, that in each processor (and hence T_0) is fixed.

The integrity of sub-domain interface treatment in FDL3DI is important for the proposed procedure. The results in Fig. 7(a and b) have been included to show the performance of the interface treatment. The function f_2 defined in Ladeinde *et al.* (2001) and its derivative were computed using the same overlapping scheme as used for the FDL3DI code. The one-dimensional domain is decomposed into two sub-domains, with the interface located within the region of strong gradients. The number of grid points is 51, with five in the overlap region. Figure 7(a) shows very accurate calculations from the interface treatment relative to the single domain results. The L_2 error analysis also supports this observation [Fig. 7(b)].

The application of the proposed method to vortex advection does not demonstrate the validity of the method to problems with strong nonlinearity. Furthermore, it needs to be demonstrated that the algorithm works when the flow direction is not parallel to the interface boundary. In order to demonstrate these capabilities, the proposed method was applied to a complicated aeroacoustic phenomenon over realistic configurations. This involves the scattering of a spherical pulse by a generic aerospace vehicle (the X24C) for which a body-fitted grid system was readily available. The pulse is specified as

$$p = p_\infty + \epsilon e^{-\ln 2\frac{(x-x_c)^2+(y-y_c)^2+(z-z_c)^2}{b^2}},$$

where

$$p_\infty = \frac{1}{\gamma M_\infty^2}, \epsilon = 0.01, b^2 = 0.1$$

and

$$x_c = 0.2978, \quad y_c = 0.2995, \quad z_c = 0.$$

A sixth-order compact scheme is used in the interior with fourth-order on the boundary. The interior filter is tenth-order, whereas the four nodes in the vicinity of the boundary (including interface boundaries) use filter schemes of orders 2, 4, 6 and 8, respectively. The calculations were done with the third-order, implicit Beam-Warming procedure using $\Delta t = 10^{-3}$. Note that the use of RK4 for this problem required a Δt that is two orders of magnitude smaller than this value. The calculations were done for two grids: $120 \times 80 \times 121$ and $60 \times 40 \times 61$. The domain is decomposed as $2 \times 2 \times 2$ and mapped into eight processors on SGI 2100. The transformed curvilinear coordinates $\xi(i)$, $\eta(j)$, $\zeta(k)$ are aligned with the streamline, body normal, and transverse directions, respectively. Figure 8(a) shows the surface grid ($J = 0$). A projected view of the acoustic pressure distribution on the surface $I = 45$ along the X (axial) direction is shown in Fig. 8(b). Figure 8(c) shows the $K = 30$ surface along the Z (circumferential) direction. The integrity of the domain interface treatment is evident, at least qualitatively, from the figures. Note that the acoustic simulation exercise for the X24C re-entry vehicle, as shown above, is preliminary and has not been examined in detail from a numerical perspective. However, the calculations do not show any unusual behavior for this complicated problem. Therefore, the proposed high-order parallelization algorithm appears to hold promise for the analysis of complex aerodynamic configurations.

CONCLUSION

The implicit operators associated with the compact schemes pose difficulties for parallel implementation for the solution of the Navier–Stokes equations. Three methods were selected for study in this paper: the one-sided method, the PDD method, and the PTA method. These parallel procedures were implemented in the AFRL code, FDL3DI. Kernel codes were also developed for these methods to extract some inherent performance features. The main findings from this work can be summarized as follows. Compared with PPP, the OOP procedure leads to faster calculations of the vortex advection problem. Also, the PPP procedure cannot accurately calculate this problem when low-order filters are used. On the other hand, high-order filters cause the procedure to be significantly more expensive. The sequential implementation of the one-sided method showed that the calculation time does not scale linearly with the number of grid points. This leads to a super-scalable parallel performance when the number of processors is few. For PDD, increasing the number of grid points in the difference direction leads to better parallel performance. Also, the use of multiple right-hand side (RHS) columns leads to better performance relative to a series of parallel PDD calculations for single columns. Increasing the number of columns significantly increases the speedup.

In standard implementation (i.e. without engaging the processors during the idle time), the PTA procedure has a very poor parallel performance in comparison to PDD and the one-sided formulations. Note that all three procedures lead to very scalable parallel codes, although the PTA procedure tends to be more accurate. The reasons for the poor parallel performance of PTA include the large idle time and the large number of small messages associated with the procedure. Procedures have been suggested that engage the processors during the idle time. Povitsky recently presented the IBPTA (Immediate Backstep Calculation PTA) procedure, which reschedules the pipelined procedure so that the local (but data-dependent Runge-Kutta procedure) could be done at the temporal regimes where the standard PTA is idle. This correction leads to a very unwieldy procedure for FDL3DI that might not be suitable for realistic aerodynamic systems and was therefore not pursued.

Acknowledgements

This work was supported in part by the DoD HPCMO Common High Performance Computing Software Support Initiative (CHSSI). Aerospace Research Corp., LI, a Subsidiary of Thaerocomp Technical Corp., is a Corporate Partner of Cornell University and some of the parallel computations reported in this paper were carried out at the Cornell Theory Center.

References

Cai, X., O'Brien, E.E. and Ladeinde, F. (1996) "Uniform mean scalar gradient in grid turbulence: asymptotic probability distribution of a passive scalar", *The Physics of Fluids* **8**(9), 2555–2558.

Cai, X., Ladeinde, F. and O'Brien, E.E. (1997) "DNS on SP2 with MPI", In: Liu, C. and Liu, Z., eds, Advances in DNS/LES (Greyden Publishing, Columbus, OH), pp 491–495.

Cai, X., O'Brien, E.E. and Ladeinde, F. (1997) "Thermodynamic behavior in decaying, compressible turbulence with initially dominant temperature fluctuations", *The Physics of Fluids* **9**(6), 1754–1763.

Cai, X. (1999) “Derivation of the theoretical performance of various parallel methods”. *Internal Report*. Report #: ARC-REG-MV99-02R. Aerospace Research Corp., LI.

Gaitonde, D. and Visbal, M.R. (1998) “High-order schemes for Navier–Stokes equations: algorithm and implementation into FDL3DI”. *Technical Report*# AFRL-VA-WP-TR-1998-3060, Air Force Research Laboratory, Wright-Patterson AFB, OH.

Gaitonde, D. and Visbal, M.R. (1999) “Further development of a Navier–Stokes solution procedure based on higher-order formulas”. *AIAA Paper* 99-0557.

Ladeinde, F. (1992) “Challenges posed for parallel processing on the iPSC/860 supercomputer by DNS schemes of supersonic flows”, In: Pelz, R.B., Ecer, A. and Hauser, J., eds, Parallel Fluid Dynamics (Elsevier, Amsterdam), pp 253–266.

Ladeinde, F., O’Brien, E.E., Cai, X. and Liu, W. (1995) “Advection by polytropic compressible turbulence”, *The Physics of Fluids* **48**(11), 2848–2857.

Ladeinde, F., O’Brien, E.E. and Cai, X. (1996) “An efficient parallelized ENO procedure for direct numerical simulation of turbulence”, *The Journal of Scientific Computing* **38**(11), 215–242.

Ladeinde, F., Cai, X.D., Visbal, M.R. and Gaitonde, D. (2001) “Turbulence spectral characteristics of high order schemes for direct and large eddy simulation”, *Applied Numerical Mathematics* **36**, 447–474.

Lele, S.K. (1992) “Compact finite difference schemes with spectral-like resolution”, *Journal of Computational Physics* **103**, 16–42.

Povitsky, A. “Parallel directionally split solver based on reformulation of pipelined Thomas algorithm”. *ICASE Report No*. 98-45.

Povitsky, A., Morris, P.J. “A parallel compact multi-dimensional numerical algorithm with aeroacoustics applications”. *ICASE Report No*. 99-34.

Shu, C.-W., Zang, T.A., Erlebacher, G., Whitaker, D. and Osher, S. (1992) “High order ENO schemes applied to two- and three-dimensional compressible flow”, *Applied Numerical Mathematics* **9**, 45–71.

Sun, X.-H. and Moitra, S. (1996) “A fast parallel tridiagonal algorithm for a class CFD applications”, *NASA TP 3585*.

Visbal, M.R., Gaitonde, D.V. (1998) “High-order accurate methods for unsteady vortical flows on curvilinear meshes”. *Paper AIAA*-98-0131, Reno, NV.