NASA/CR-1998-208733
ICASE Report No. 98-45

Parallel Directionally Split Solver Based on Reformulation of Pipelined Thomas Algorithm

A. Povitsky
ICASE, Hampton, Virginia

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

Operated by Universities Space Research Association

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

Prepared for Langley Research Center
under Contract NAS1-97046

October 1998

PARALLEL DIRECTIONALLY SPLIT SOLVER BASED ON REFORMULATION OF PIPELINED THOMAS ALGORITHM

A. POVITSKY [*]

Abstract. A very efficient direct solver, known as the Thomas algorithm, is frequently used for the solution of band matrix systems that typically appear in models of mechanics. The processor idle time is a reason for the poor parallelization efficiency of the solvers based on the pipelined parallel Thomas algorithms.

In this research an efficient parallel algorithm for 3-D directionally split problems is developed. The proposed algorithm is based on a reformulated version of the pipelined Thomas algorithm that starts the backward step computations immediately after the completion of the forward step computations for the first portion of lines. This algorithm has data available for other computational tasks while processors are idle from the Thomas algorithm.

The proposed 3-D directionally split solver is based on the static scheduling of processors where local and non-local, data-dependent and data-independent computations are scheduled while processors are idle. A theoretical model of parallelization efficiency is used to define optimal parameters of the algorithm, to show an asymptotic parallelization penalty and to obtain an optimal cover of a global domain with subdomains.

It is shown by computational experiments and by the theoretical model that the proposed algorithm reduces the parallelization penalty about two times over the basic algorithm for the range of the number of processors (subdomains) considered and the number of grid nodes per subdomain.

Key words. parallel computing, parallelization model, directionally split methods, pipelined Thomas Algorithm, banded matrices, ADI and FS methods

Subject classification. Computer Science

1. Introduction. Efficient solution of directionally split banded matrix systems is essential to multi-grid, compact, and implicit solvers. When the implicit schemes are applied to multi-dimensional problems, the operators are separated into one-dimensional components and the scheme is split into two (for 2-D problems) or three (for 3-D problems) steps, each one involving only the implicit operations originating from a single coordinate [1]. These numerical algorithms are denoted as fractional step (FS) or alternating direction implicit (ADI) methods.

According to D. Caughey, the development of implicit CFD algorithms suitable for distributed memory machines will become increasingly important in the future. The parallelization of implicit schemes will require more ingenuity if they are to remain competitive with parallel explicit schemes. The interplay between the data structures, the memory assignment among processors, and its access by the flow solver will require a truly interdisciplinary approach to produce effective algorithms [2].

For block-banded systems the LU decomposition method leads to an efficient serial direct algorithm known as the Thomas algorithm. Parallelization of implicit solvers that use the Thomas algorithm for the solution of banded system of equations is hindered by global data dependencies. Parallel versions of the Thomas algorithm are of the pipelined type [3]. Pipelines occur due to the recurrence of data at the

[*]Staff Scientist, ICASE, NASA Langley Research Center, Hampton, VA 23681-0001 (e-mail:aeralpo@icase.edu) This research was supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-97046 while the author was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23681-2199.

forward and the backward step computations of the Thomas algorithm. During the pipelined process the first processor has to wait for the completion of the forward step computations and completion of the backward step computations for the first group of lines on all processors ahead. Thus, the first processor becomes idle at the switch from the forward to the backward step of the Thomas algorithm. In turn, the last processor has to wait for available data at the beginning of each spatial step.

The use of the pipelined Thomas algorithm (PTA) results in communication between neighboring processors due to the transfer of coefficients on the forward step and transfer of solution on the backward step. One may use a separate message after completion of the forward or backward step for a single line, however, this results in the maximum communication time latency overhead. On the other hand, reducing the latency overhead by sending data from more than one line in each message increases the data dependency delay effect and processor idle time. This latency and data dependency delay tradeoff determines the optimum number of lines to be completed before a message is sent to the neighboring processor [4], [5]. V. Naik et al [4] defined this optimum number of lines and used it in their parallelization of an implicit finite-difference code for solving Euler and thin-layer Navier-Stokes equations. Still, there is the processor idle time between the forward and the backward step computations and global synchronization of processors at each spatial step of the ADI. This parallel implicit directionally split algorithm is referred to as the basic algorithm in this study.

In our recent study [6], we formulated a new version of the Thomas parallel algorithm named the Immediate Backward Pipelined Thomas Algorithm (IB-PTA). The backward step computations begin immediately after the forward step computations have been completed for the first portion of lines. Some lines have been rendered by the Thomas algorithm before processors become idle. Although the IB-PTA cannot reduce the processor idle time, there are data available for other computational tasks while processors are idle. A multi-dimensional numerical algorithm using the IB-PTA in each spatial direction can run on processors in the time-staggered manner without a global synchronization (the first processor finishes its computations first at each time step). The development of such algorithm is addressed in this study.

Various computational tasks are examined for execution while processors are idle from the Thomas algorithm in the current direction. These tasks may include: (i) computation of the left-hand side coefficients for the next spatial stage, including computation of damping functions and non-linear TVD evaluations; (ii) computation of the right-hand side coefficients, for example, multiplications of intermediate FS or ADI functions by transformation metrics of curvilinear grids; and (iii) execution of the forward step computations of the Thomas algorithm in the next spatial direction. These tasks are classified here as either data-dependent (using the last spatial step solution) or data-independent.

We develop a theoretical model to estimate a parallelization efficiency of the proposed algorithm. Such models are recognized as a quantitative basis for the design of parallel algorithms. We base our model on the idealized multicomputer parallel architecture model [7]. Low-level hardware details such as memory hierarchies and the topology of the interconnection network are not introduced in the model. This model is used here to define the optimal number of solved lines per message, to provide asymptotic analysis and to estimate of parallelization efficiency for a large number of processors which are not available yet, and to compare the proposed algorithm with the basic one. First, this model is used for a cubic global domain (equal number of grid nodes in all directions). Then an optimal partitioning for a global domain with unequal number of grid nodes in different directions is obtained based on this model. To get a unified approach for various MIMD computers, results are presented in terms of ratios between communication latency and transfer times to the backward step computation time per grid node. These ratios were used in studies [8] and [9] for the complexity analysis of various parallel algorithms for numerical solutions of partial differential equations (PDE).

As the development of a computer program by the proposed algorithm may represent certain difficulties, we design a parallel computer code using modular design techniques [7] and present it here. We generate static schedule of processors for the IB-PTA in a single direction by methods described in our study [6] and fashion schedules of the pipelined Thomas algorithm in different directions together with local computational tasks. Performing core computations of the algorithm, processors are governed by this static schedule. Computations are separated from communication between processors. One can switch to other type of banded matrix system to be solved, method of computing discretized coefficients of PDE, type of directionally split method, other schedule of processors, or type of message-passing library changing only corresponding modules of the code.

This paper is composed of the following sections. In section 2, the parallelization method is described. In section 3, the theoretical model of parallelization efficiency is developed for 2-D and 3-D computational domains. In section 4, the parallelization method and the theoretical model of the parallelization efficiency are tested and verified by a numerical solution of a benchmark problem on CRAY T3E and IBM SP MIMD computers.

2. Formulation of the algorithm.

2.1. Mathematical formulation. Consider a non-linear partial differential equation

$$\frac{dU}{dt} = S(U)U + Q \tag{1}$$

where t is the time, $S(U) = S_x(U) + S_y(U) + S_z(U)$ is a spatial differential operator and Q is a source term. As an example of a directionally split method we consider the fractional step (FS) method. The method is based on a factorization of the Crank-Nicholson scheme, where the factorized scheme is solved in three steps as a succession of one-dimensional Crank-Nicholson schemes:

$$(1 - 0.5\Delta t S_x)\overline{U^{n+1}} = (1 + 0.5\Delta t S_x)U^n + \Delta t Q$$

$$(1 - 0.5\Delta t S_y)\overline{\overline{U^{n+1}}} = (1 + 0.5\Delta t S_y)\overline{U^{n+1}}$$

$$(1 - 0.5\Delta t S_z)U^{n+1} = (1 + 0.5\Delta t S_z)\overline{\overline{U^{n+1}}}, \tag{2}$$

where S_x, S_y and S_z are linear finite-difference operators in the x, y and z directions, respectively, and $\overline{\overline{U^{n+1}}}, \overline{U^{n+1}}$ are intermediate FS functions.

Second-order finite-difference formulation of this system leads to band tri-diagonal system of linear equations

$$a_{i,j,k}U_{i-1,j,k} + b_{i,j,k}U_{i,j,k} + c_{i,j,k}U_{i+1,j,k} = f_{i,j,k}, \tag{3}$$

where $i,\ j,\ k$ are spatial grid nodes, coefficients $a_{i,j,k}, b_{i,j,k}$ and $c_{i,j,k}$ are functions of U^n and/or time, $f_{i,j,k}$ is the r.h.s. of equations (2). The above system of linear equations corresponds to the first spatial step of Eqs. (2). The similar banded linearized systems must be solved for the second and the third spatial steps of the FS.

This system of N^3 equations is considered as N^2 systems of N equations where each system of N equations corresponds to $j, k = const$. Therefore, the scalar tridiagonal version of the Thomas algorithm is used to solve each system of N equations. The forward step of the Thomas algorithm for this system is

$$d_{1,j,k} = b_{1,j,k}, \quad d_{i,j,k} = b_{i,j,k} - a_{i,j,k}\frac{c_{i-1,j,k}}{d_{i-1,j,k}}, \quad i = 2, ..., N_{totx}$$

$$g_{1,j,k} = \frac{f_{1,j,k}}{d_{1,j,k}}, \quad g_{i,j,k} = \frac{-a_{i,j,k}g_{i-1,j,k} + f_{i,j,k}}{d_{i,j,k}}, \quad i = 2, ..., N_{totx}, \tag{4}$$

where N_{totx} is the number of grid nodes in the x direction. The backward step of the Thomas algorithm is

$$U_{N,j,k} = g_{N,j,k}, \ U_{i,j,k} = g_{i,j,k} - U_{i+1,j,k}\frac{c_{i,j,k}}{d_{i,j,k}}, \quad k = N_{totx} - 1, ..., 1 \tag{5}$$

The solution of this banded system of N equations is also denoted as the rendering of the line (j, k).

2.2. Pipelined Thomas algorithms. To parallelize the Thomas algorithm, the coefficients of Eq. (3) are mapped into processors so that the subset $\{a_{i,j,k}, \ b_{i,j,k}, \ c_{i,j,k} | \quad i = N(I-1)+1, ..., NI, \ j = N(J-1)+1, ..., NJ, \ k = N(K-1)+1, ..., NK\}$ belongs to the $(I, J, K)^{th}$ processor. The computational domain is divided to $N_I \times N_J \times N_K$ subdomains with $N \times N \times N$ grid nodes each one.

The Pipelined Thomas Algorithm (PTA) [4] is cited shortly. The $(I, J, K)^{th}$ processor receives coefficients $d_{N(I-1),j,k}, \ g_{N(I-1),j,k}$ from the $(I-1, J, K)^{th}$ processor; computes coefficients $d_{l,j,k}$ and $g_{l,j,k}$, where $l = N(I-1)+1, ..., NI$ of the forward step of the Thomas algorithm, sends coefficients $d(NI, j, k), \ g(NI, j, k)$ to the $(I+1, J, K)^{th}$ processor and repeats computations (4) until the forward step computations for the N^2 lines are completed. After completion of all the forward step computations specific to a single processor, the $(I, J, K)^{th}$ processor has to wait for the completion of the forward step by all processors ahead. The backward step computations (5) are performed in the similar manner as the forward step computations but in the decreasing direction. The lines (j, k) are gathered in groups and the number of lines is solved per message. Processors are idle between the forward and the backward steps of the Thomas algorithm and there are no data available for other computational tasks by this time.

The Immediate Backward Thomas Pipeline Algorithm (IB-PTA) has been developed in our study [6] and is described here briefly. First, groups of lines are rendered by the forward step computations (see above) till the first group of lines is completed on the last processor. Then the backward step computations for each group of lines begin 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 and communicates with its neighbors to receive necessary data for beginning of either the forward or the backward computations for the next portion of lines. Finally, remaining lines are rendered by the backward step computations and there are no available lines for the forward step computations. It was shown [6] that the idle time is the same for the IB-PTA and the basic PTA when these algorithms are used in a single direction. The advantage of the IB-PTA is that processors become idle after completion of subset lines. At this time processors can be used for other computational tasks requiring these data, as described in the following subsection.

2.3. Schedule of processors for directionally split problem. The basic algorithm is executed in the y and z directions the same way as in the x direction. After completion of all Thomas algorithm computations in the last spatial direction, processors compute $S(U)$ operators. Communications control computational tasks as either the forward step coefficients or the backward step solution must be obtained from the neighboring processors for the beginning of either the forward or the backward step computations, and there are no other computational tasks while processors wait for these data.

This is no longer the case for the proposed algorithm, because processors execute other computational tasks while these processors are idle. Additionally, these tasks might be manifold, and the idle processor times are different for the different processors. Therefore, the static scheduling of processors is adopted in this study, i.e., the communication and computations schedule of processors is computed before numerical computations are executed.

We define a time unit as a time interval when a processor either renders a group of lines by one type of computations or is idle. Processors may communicate only at the beginning of the time unit. The length of a time unit is the same for all processors. This schedule of processors includes the order of computational tasks on each processor and the order of communication with its neighbors.

Types of computations in this study include non-local (the forward step computations and the backward step computations of the Thomas Algorithm) and local (computations of spatial operators $S(U)$ or computations of the right hand side (r.h.s.) of equations (2)). Most of considered computational tasks, namely, all the Thomas algorithm computations, local computations of the r.h.s., and local computations of spatial operators at the end of a time step use results of the Thomas algorithm computations in the last rendered direction, i.e., these tasks are data-dependent. However, computations of the S_y and S_z operators are data-independent from the results of the Thomas algorithm computations in the directions x and y. These operators depend upon solution U^n and do not depend upon intermediate functions $\overline{\overline{U^{n+1}}}$ and $\overline{U^{n+1}}$. Both data-dependent and data-independent tasks may be executed while processors are idle from the Thomas algorithm. However, the practical realization is different for data-dependent, data-independent, local and non-local tasks. Additionally, each type of computations has a different computational time per grid node; therefore, the number of rendered lines per time unit are different for various types of computations (see the next section about calculation of these numbers).

Computations are governed by an integer-valued variable:

$$J(p,i) = \begin{cases} 4 & \text{local computations} \\ +l & \text{forward step computations} \\ 0 & \text{processor is idle} \\ -l & \text{backward step computations} \end{cases} \tag{6}$$

where p is the processor number, i is the number of time unit, $l = 1, 2, 3$ corresponds to the directions x, y and z, respectively.

A scheduling algorithm for PTAs in a single direction has been developed in our study [6]. The processor schedule for a subset of N_K processors $\{(I, J, 1), ..., (I, J, N_K)\}$ that form a pipeline for the Thomas algorithm computations in the z direction is shown in Fig. 1. It is assumed, that the computational time per grid node is equal for the forward and the local computations and it is 1.5 times greater than that for the backward step computations. Therefore, the number of portions of lines for the forward step computations is equal to that for the local computations, whereas the number of portions of lines for the backward step computations is 1.5 times less. The column of values $J(K, i)$ corresponds to the K^{th} processor (from 1 to N_K). Columns are shifted so as each horizontal line corresponds to a single time moment in terms of wall clock. Arrows $---->$, $<----$ and $<-- >$ denote send, receive and send-receive communications between neighboring processors in this processor pipeline.

The processor schedule corresponds to the execution of the Thomas algorithm in the direction z and the local, data-dependent computations of the $S_x(U^n)$ operator for the next time step. The proposed algorithm is based on the IB-PTA and uses the processor idle time for the computations of the operator S_x. Processors run the proposed algorithm in a time-staggered way so that the first outermost processor $(I, J, 1)$ completes its computations first and processors do not become idle (Fig. 1a). For the basic algorithm, the operator S_x can be computed only after the completion of the PTA for all lines (Fig. 1b). The first outermost processor completes computations last due to the idle time between the forward and the backward steps (Fig. 1b). Thus, Fig. 1 illustrates the main advantage of the proposed algorithm over the basic one.

The other schedule of processors includes the forward step computations of the Thomas algorithm in the next direction while processors are idle from the Thomas algorithm in the current direction. To make it feasible, the grid nodes rendered by the Thomas algorithm in the current direction must form a contiguous extend in the next direction.

To execute the Thomas algorithm in the x direction, the set of N^2 lines $\{(j,k), j = 1,...,N,\ k = 1,...,N\}$ is gathered in groups in such a way that non-rendered lines with the minimum value of index k are taken first. For example, consider the subdomain with 14^3 nodes where the $14^2 = 196$ lines are gathered in the 17 groups (see Fig 2a). Each group contains the 12 lines except the last, 17th group. To execute the Thomas algorithm in the x direction, lines are gathered in groups by this method that secures a contiguous extent in the y direction.

The schedule of processors corresponding to this case is shown in Fig. 3. The first two processors have idle time (denoted as 0) between the forward and the backward steps and the first processor does not complete its tasks first. The reason is that there are no completed lines while these processors become idle. The straightforward way to remedy the problem of idle processors is to reduce the number of lines per group. However, the communication latency time increases as more messages have to be transfered.

Consider the case where processors are used for the Thomas algorithm computations in the next direction while they are idle repeatedly for the x and y directions. In addition to the previous case, executing the Thomas algorithm computations in the y direction, the algorithm gathers lines so as to form a contiguous extend of nodes in the z direction. Therefore, the set of N^2 lines $\{(i,k), i = 1,...,N,\ k = 1,...,N\}$ is gathered in groups in such a way that non-rendered lines with the minimum value of index i are taken first.

Consider the previous example (Fig 2a-b). The first 8 groups of lines must be completed in the current direction before processors become idle. Otherwise, processors stay idle when they perform computations in the y direction waiting for a contiguous extend of grid nodes in the z direction. This causes a severe restriction on the maximum number of lines per group.

Thus, scheduling of data-dependent computations while processors are idle from the Thomas algorithm may lead to restrictions of the number of lines per group. By scheduling data-independent computations while processors are idle we avoid these restrictions. In the considered case of FS, computations of the S_y and S_z operators while processors are idle from the Thomas algorithm in the x and y directions are data-independent. Both the IB-PTA and the PTA with the processor scheduling may be adopted for the two first stages of the FS. Still, the IB-PTA is essential for the last stage of the FS.

The following recommendations with regard to the processor scheduling are drawn for non-linear FS methods:

• If computational time per grid node is greater for the local computations than that for the forward step computations, the spatial operators are computed for a subset of grid nodes while processors are idle from the Thomas algorithm. Coefficients for the rest of nodes are computed after the completion of the Thomas algorithm for all lines in the current direction. These computations are local; therefore, they do not contribute to the parallelization penalty time.

• If local computations are partly data-independent (computations of the l.h.s. coefficients) and partly data-dependent (computations of the r.h.s.), then the data-independent computations are scheduled to execute

first while processors are idle.
• If data-independent computation time per grid node is less than the forward step computational time per grid node and greater than a half of the latter time, then the computations of the S_y and S_z are scheduled while processors are idle in the x direction. The Thomas algorithm computations in the z direction are scheduled while processors are idle in the y direction.

For the systems of the Euler or Navier-Stokes equations these suggestions can be applied as follows. A typical large-scale aerodynamics code ARC-3D [4], [11] includes three dimensional solution of a system of five PDE and uses two versions of ADI. The first version is the original Beam-Warming Approximate Factorization scheme [12] leading to the block tri-diagonal Thomas algorithm operating with 5×5 blocks. The other version is based on the diagonalization technique of Pulliam and Chaussee [13], leading to the decoupling of variables and solution of 5 penta-diagonal scalar systems in each direction. Naik et al measured the elapsed CPU time and number of floating point operations for these two versions ([4],Table 2). Presented there are the following results important for our study: (i) the cost of implicit part of the solver (including the setup of coefficients of the linear system) is the dominant cost for the two versions; (ii) the cost of the r.h.s. computations is approximately equal to the total cost of forward step computations in all directions for the second version; (iii) the cost of coefficients setup is greater than the cost of the forward step computations for the second version.

For the scalar penta-diagonal version, the setup of coefficients costs much due to non-linearity of the original system of equations, use of fourth order numerical viscosity in implicit side, and multiplications of intermediate ADI functions by curvilinear derivatives. For the block tri-diagonal version, the forward step computations cost approximately ten times as much as those for the scalar penta-diagonal version.

For the scalar penta-diagonal version, one may use processors for local computations while they are idle from the Thomas algorithm. These local computations include data-independent computations of discretized coefficients and data-dependent multiplications of intermediate FS functions by curvilinear derivatives. For the block tri-diagonal version, the only way to use the processor idle time is to execute the Thomas algorithm in the next direction. For the third stage of the ADI, the data-dependent r.h.s. computations may be performed.

3. Theoretical model of parallelization efficiency. A parallel machine model called the multicomputer [7] is used here for the development of the model. A multicomputer comprises a number of von Neumann computers, or nodes, linked by an interconnection network. Each computer executes its own program. This program may access local memory and may send and receive messages over the network. Messages are used to communicate with other computers or, equivalently, to read and write remote memories. In the idealized network, the cost of sending a message between two nodes is independent of both node location and other network traffic but does depend on message length. Although the most important case for parallel computing is 3-D, we start with the 2-D case and further use the same technique to build the theoretical model for the 3-D case.

3.1. 2-D case. The parallelization efficiency is estimated for a square computational domain covered with $N_d \times N_d$ equal load-balanced subdomains with $N \times N$ grid nodes per subdomain Each subdomain belongs to a different processor. A single PDE to be solved and the FS method (2) with tri-diagonal matrices in each direction are assumed. Extensions to a 3-D case and global computational domain with an unequal number of grid nodes in each direction will be considered in the next subsections.

The communication time for a single message between two processors in the network can be approximated by the following linear expression [9], [10]:

$$f(L) = b_0 + b_1 L, \tag{7}$$

where b_0 and b_1 are communication coefficients and L is the length of the string in words.

In this section, the additional (penalty) time required for a single FS time step due to communication and idle time of processors is estimated. This penalty time is defined as

$$F = T_{parallel} - T_{serial}/N_D, \tag{8}$$

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 function F is composed of three main contributions:

F_1 - the communication time due to the transfer of the forward step coefficients and the backward step solution of the Thomas algorithm.

F_2 - the idle time due to waiting for communication with the neighboring processor.

F_3 - the communication time due to the transfer of the values of the FS variables between neighboring subdomains.

For the square subdomains considered with $N \times N$ grid nodes, the quantities $F_1 - F_3$ are given by

$$F_1 = L_s(\lceil N/K_1 \rceil (b_0 + 2b_1K_1) + \lceil N/K_2 \rceil (b_0 + b_1K_2)), \tag{9}$$

where $\lceil N/K_1 \rceil$ and $\lceil N/K_2 \rceil$ are the number of messages for the forward and backward steps, respectively, L_s is the index of the partitioning scheme (2 for 2-D and 3 for 3-D), K_1 and K_2 are the number of lines per message for the forward and the backward step computations.

The expression of F_1 is the same for the proposed and the basic algorithm as the same data must be transfered. However, the optimal values of K_1 and K_2 are different for these algorithms (see below).

There are two reasons why the current processor has to wait for its neighbors:

1. Neighboring processors are synchronized due to the exchange of values of FS variables: the $(I, J)^{th}$ processor completes its backward step computations for the last K_2 lines later than the $(I, J+1)^{th}$ processor. At the next time step, these processors must exchange interface values of the intermediate FS function. Therefore the $(I, J+1)^{th}$ processor has to wait for the $(I, J)^{th}$ processor.

2. If data-dependent computational tasks are scheduled while processors become idle, processors might be idle waiting for the completion of the first group of lines (see Fig. 3). This idle time is equal to the time difference between the completion of the first portion of lines by the backward step and the completion of all lines by the forward step.

The delay time of the current processor is determined as the maximum of these two delays:

$$F_{2A} = L_s \cdot \max(NK_2g_2, (2(N_d - 1) + \lceil \rho \rceil)NK_1g_1 - N^2g_1). \tag{10}$$

First, let us consider that the first reason of delay dominates and thus

$$F_{2A} = L_sNK_2g_2 \ \ (= L_sNK_1g_1). \tag{11}$$

For the basic algorithm the global synchronization occurs twice per spatial step due to a pipelining property of the Thomas algorithm:

$$F_{2B} = L_s(N_d - 1)N(K_1g_1 + K_2g_2). \tag{12}$$

A processor sends the interface values of the intermediate FS functions to the neighboring processors at each spatial step. Thus, each processor sends $2L_s$ messages with length N per spatial step. The communication time for transfer of the variables between neighboring processors is

$$F_3 = 2L_s(b_0 + b_1 N). \tag{13}$$

The term F_3 does not depend on K.

Generally, computational times per grid point are different for the forward step and for the backward step computations of the Thomas algorithm. For the IB-PTA the computational work per group of lines should be the same for the forward and the backward step computations:

$$NK_1 g_1 = NK_2 g_2, \tag{14}$$

where g_1 and g_2 are the computation times of the forward and backward steps of the Thomas algorithm per grid point. Thus,

$$K_2 = \left\lceil \frac{g_1}{g_2} K_1 \right\rceil. \tag{15}$$

The penalty function F depends on the following parameters: the number of grid points in one direction per subdomain N, the number of subdomains N_D, the computation times the Thomas algorithm per grid point g_1 and g_2 and the communication coefficients b_0 and b_1. This function for the proposed algorithm is given by $F_A = F_1 + F_{2A} + F_3$ and for the basic algorithm $F_B = F_1 + F_{2B} + F_3$. The theoretical value of parallelization penalty function per grid point is defined as

$$PnM = \frac{F}{g_{ov} N^{L_s}} \times 100\%. \tag{16}$$

The way to seek such values of K_2 that minimize F is to solve the equation

$$\partial F / \partial K_2 = 0, \tag{17}$$

where $1 \leq K_2 \leq N$. In order to facilitate this operation the discrete function $\lceil x \rceil$ is replaced by x in the following discussion. For the proposed algorithm using the IB-PTA the optimal K_2 value is given by

$$K_{1,IB-PTA} = \sqrt{(1+\rho)\gamma}/\rho, \quad K_{2,IB-PTA} = \sqrt{(1+\rho)\gamma}, \tag{18}$$

where $\gamma = b_0/g_2$ is the ratio between the communication latency and the characteristic computational time per grid node and $\rho = g_1/g_2$ is the ratio between the forward and the backward step computational times.

The corresponding value of the parallelization penalty F_A is

$$F_A = Ng_2(4\sqrt{\gamma(1+\rho)} + 6\tau + 4(\gamma/N + \tau)). \tag{19}$$

For the basic algorithm the optimal K_1 and K_2 values are given by

$$K_{1,B} = \sqrt{\frac{\gamma}{\rho(N_d - 1)}}, \quad K_{2,B} = \sqrt{\frac{\gamma}{N_d - 1}} \tag{20}$$

and the corresponding value of the parallelization penalty function:

$$F_B = Ng_2(4\sqrt{\gamma(N_d - 1)}(1 + \sqrt{\rho}) + 6\tau + 4(\gamma/N + \tau)). \tag{21}$$

The ratio of first components of the F_B and the F_A which are the leading terms for large γ and N is given by

$$R_1 = \frac{\sqrt{N_d - 1}(1 + \sqrt{\rho})}{\sqrt{1+\rho}}. \tag{22}$$

The multiplier $C_1 = (1 + \sqrt{\rho})/\sqrt{1+\rho}$ reaches its maximum $\sqrt{2}$ if the computational times per grid node are equal for the forward and the backward steps ($\rho = 1$). For the bound case $g_1 >> g_2 (\rho \to \infty)$, the multiplier $C_1 \to 1$.

The ratio of parallelization penalties for the large N is expressed by

$$R_2 = \frac{F_{PTA}}{F_{IB-PTA}} = \frac{F_1 + F_{2,PTA} + F_3}{F_1 + F_{2,IB-PTA} + F_3} \approx \frac{R_1 + C_2}{1 + C_2}, \tag{23}$$

where $C_2 = 2.5\tau/\sqrt{\gamma(1+\rho)} \leq 2.5\tau/\sqrt{2\gamma}$.

Thus, the ratios R_1 and R_2 are $O(N_d^{1/2})$ with the factor $1 < C_1 \leq \sqrt{2}$.

Now we have to verify when the first term in Eq. (10) dominates, i.e.,

$$(2(N_d - 1) + \lceil \rho \rceil) N K_1 g_1 - N^2 g_1 \leq N K_2 g_2. \tag{24}$$

Using Eq. (14), the above inequality becomes

$$K_1 \leq K_{1,r}, \quad K_2 \leq K_{2,r}, \tag{25}$$

where $K_{1,r} = N/(2(N_d - 1) + \lceil \rho \rceil - 1)$, $K_{2,r} = \rho N/(2(N_d - 1) + \lceil \rho \rceil - 1)$. By substituting Eq. (18) for K_{2A} in the above inequality, we obtain a condition when the first term in Eq. (10) dominates:

$$\sqrt{(1+\rho)\gamma} \leq \frac{\rho N}{2(N_d - 1) + \lceil \rho \rceil - 1}, \tag{26}$$

If this inequality is satisfied, Eq. (19) and ratios (22, 23) are valid. The critical value of N depends linearly upon N_d :

$$N_{cr} = (2(N_d - 1) + \lceil \rho \rceil)\sqrt{(1+\rho)\gamma}/\rho. \tag{27}$$

Assuming that the second term in Eq. (10) dominates, we obtain the optimal value of K from the condition $\partial F/\partial K = 0$ to give

$$K'_{2A} = \sqrt{\frac{(1+\rho)\gamma}{2(N_d - 1) + \lceil \rho \rceil}}. \tag{28}$$

For the case considered, the condition $K'_{2A} \geq K_{2,r}$ is expressed by

$$\sqrt{\frac{(1+\rho)\gamma}{2(N_d - 1) + \lceil \rho \rceil}} \geq \frac{\rho N}{2(N_d - 1) + \lceil \rho \rceil}. \tag{29}$$

Therefore, the expression for N'_{cr} is given by

$$N'_{cr} = \sqrt{2(N_d - 1) + \lceil \rho \rceil}\sqrt{(1+\rho)\gamma}/\rho = N_{cr}/\sqrt{2(N_d - 1) + \lceil \rho \rceil}. \tag{30}$$

Thus, there are three cases in terms of the optimal number of lines solved per message. For $N \geq N_{cr}$ or if data-independent computations are scheduled while processors are idle, the optimal K values are defined by (18). For $N'_{cr} \leq N \leq N_{cr}$, the optimal K values are equal to K_r (see (25)). The values $K_{1,r}$ and $K_{2,r}$ correspond to the case when the backward step computations for the first portion of lines are completed immediately after conclusion of all the forward step computations on the first outermost processor. However, for $N'_{cr} > N$ these K_r do not give the minimum to the penalty function. In this case the optimal K are defined by Eq. (28), and processors become idle from the conclusion of the last group of lines by the forward step till the beginning of the first group of lines by the backward step. Finally, the value of parallelization penalty is given by

$$(31) \quad F_A = \begin{cases} Ng_2(4\sqrt{\gamma(1+\rho)} + 6\tau + 4(\gamma/N + \tau)) & \text{if } N \geq N_{cr} \\ Ng_2(\frac{2(1+\rho)\gamma(2(N_d-1)+\lceil\rho\rceil-1)}{\rho N} + \frac{2\rho N}{2(N_d-1)+\lceil\rho\rceil-1} + 6\tau + 4(\gamma/N+\tau)) & \text{if } N'_{cr} \leq N \leq N_{cr} \\ Ng_2(2\sqrt{(1+\rho)\gamma(2(N_d-1)+\lceil\rho\rceil-1)} - N\rho + 6\tau + 4(\gamma/N+\tau)) & \text{if } N < N'_{cr}\ . \end{cases}$$

The asymptotic analysis of the above formulae and Eq. (27 and (30) for N_{cr} and N'_{cr} leads to the following expression for the asymptotic order of the parallelization penalty:

$$(32) \qquad O(F_A) = \begin{cases} O(N) & \text{if } O(N) > O(N_d) \\ O(N_d) & \text{if } O(N_d^{1/2}) < O(N) < O(N_d) \\ O(N)O(N_d^{1/2}) & \text{if } O(N) < O(N_d^{1/2})\ . \end{cases}$$

Thus, for $N_d,\ N \to \infty$ the proposed algorithm has an advantage over the basic algorithm unless $O(N) < O(N_d^{1/2})$. In the last case both algorithms have the same order $O(N)O(N_d^{1/2})$ (see Eq. 21)). The order of the penalty function in terms of the overall number of nodes and the overall number of subdomains (processors) is given by

$$(33) \qquad O(F_A) = \begin{cases} O(N_{tot}^{1/2})/O(N_D^{1/2}) & \text{if } O(N_{tot}) > O(N_D^2) \\ O(N_D^{1/2}) & \text{if } O(N_D^{3/2}) < O(N_{tot}) < O(N_D^2) \\ O(N_{tot}^{1/2})/O(N_D^{1/4}) & \text{if } O(N_{tot}) < O(N_D^{3/2})\ , \end{cases}$$

where $N_{tot} = (N \times N_d)^2$ is the overall number of nodes and $N_D = N_d^2$ is the overall number of processors.

The previous case corresponds to the data-dependent computations scheduled while processors are idle. Now we will analyze a case where part of computations scheduled to be executed while processors are idle is data-independent. In this case the second term in Eq. (10) becomes

$$(34) \qquad (2(N_d-1)+\lceil\rho\rceil)NK_1g_1 - (1+\alpha)N^2g_1,$$

where $\alpha = g_{di}/g_1$ is the ratio of the data-independent computational time g_{di} to the forward step computational time per grid node. The data-dependent computations are scheduled to be executed first while processors are idle from the Thomas algorithm computations. The expression for $K_{1,r}$ is given by

$$(35) \qquad K_{1,r} = \frac{N(1+\alpha)}{2(N_d-1)+\lceil\rho\rceil-1}.$$

The expressions for the critical numbers are divided by $(1+\alpha)$ in Eq. (27,30). Asymptotic results (32) are the same as for the previous case of the data-dependent computations scheduled while processors are idle. However, the parallelization penalty is reduced:

$$F_A = \begin{cases} Ng_2(4\sqrt{\gamma(1+\rho)} + 6\tau + 4(\gamma/N+\tau)) & \text{if } N \geq N_{cr} \\ Ng_2(\frac{2(1+\rho)\gamma(2(N_d-1)+\lceil\rho\rceil-1)}{\rho N(1+\alpha)} + \frac{2\rho N(1+\alpha)}{2(N_d-1)+\lceil\rho\rceil-1} + 6\tau + 4(\gamma/N+\tau)) & \text{if } N'_{cr} \leq N \leq N_{cr} \\ Ng_2(2\sqrt{(1+\rho)\gamma(2(N_d-1)+\lceil\rho\rceil-1)} - N(1+\alpha)\rho + 6\tau + 4(\gamma/N+\tau)) & \text{if } N < N'_{cr}\ . \end{cases}$$

3.2. 3-D case. In this case a cubic domain is divided regularly into cubic subdomains with $N \times N \times N$ grid nodes each one. The parallelization penalty function is composed of the same components as in the previous 2-D case:

$$F_1 = L_s(\lceil N^2/K_1 \rceil (b_0 + 2b_1K_1) + \lceil N^2/K_2 \rceil (b_0 + b_1K_2)), \tag{36}$$

$$F_{2A} = L_s max(NK_2g_2, (2(N_d - 1) + \lceil \rho \rceil)NK_1g_1 - N^3g_1), \tag{37}$$

$$F_3 = L_s(b_0 + b_1N^2). \tag{38}$$

The number of grid nodes at an interface boundary is equal to N^2 and not to N as for the 2-D case; therefore, N is replaced by N^2 in the components of the parallelization penalty.

The parallelization penalty component F_{2B} is the same as in the 2-D case, and the parallelization penalty for the basic algorithm is

$$F_B = Ng_2(6\sqrt{\gamma N(N_d - 1)}(1 + \sqrt{\rho}) + 9\tau N + 6(\gamma/N + \tau N)), \tag{39}$$

where the optimal K values are

$$K_{1,B} = \sqrt{\frac{N\gamma}{\rho(N_d - 1)}}, \quad K_{2,B} = \sqrt{\frac{N\gamma}{N_d - 1}}. \tag{40}$$

If the first term in Eq. (37) dominates, the optimal K values for the proposed algorithm are analogous to (18) and are given by

$$K_{1,A} = \sqrt{N(1 + \rho)\gamma}/\rho, \quad K_{2,A} = \sqrt{N(1 + \rho)\gamma}. \tag{41}$$

The bound values $K_{1,r}$ and $K_{2,r}$ are

$$K_{1,r} = N^2/(2(N_d - 1) + \lceil \rho \rceil), K_{2,r} = \rho N^2/(2(N_d - 1) + \lceil \rho \rceil). \tag{42}$$

Using expressions (41, 42), the critical number of nodes per subdomain in one direction is obtained:

$$N_{cr} = \left[\frac{(2(N_d - 1) + \lceil \rho \rceil - 1)^2(1 + \rho)\gamma)}{\rho^2} \right]^{1/3}. \tag{43}$$

The values of $K'_{2,A}$ and N'_{cr} are

$$K'_{2,A} = \sqrt{\frac{N(1 + \rho)\gamma}{2(N_d - 1) + \lceil \rho \rceil}}, \ N'_{cr} = \frac{N_{cr}}{(2(N_d - 1) + \lceil \rho \rceil - 1)^{1/3}}. \tag{44}$$

The final expression for the parallelization penalty of the proposed algorithm in the 3-D case is

$$(45) F_A = \begin{cases} Ng_2(6\sqrt{N\gamma(1 + \rho)} + 9N\tau + 6(\gamma/N + N\tau)) & \text{if } N \geq N_{cr} \\ Ng_2(\frac{3(1+\rho)\gamma(2(N_d-1)+\lceil \rho \rceil)}{\rho N} + \frac{3\rho N^2}{2(N_d-1)+\lceil \rho \rceil} + 9N\tau + 6(\gamma/N + N\tau)) & \text{if } N'_{cr} \leq N \leq N_{cr} \\ Ng_2(3\sqrt{N(1 + \rho)\gamma(2(N_d - 1) + \lceil \rho \rceil)} - N^2\rho + 9N\tau + 6(\gamma/N + N\tau)) & \text{if } N < N'_{cr} \ . \end{cases}$$

The asymptotic analysis follows a pattern very similar to that of the previous subsection:

$$O(F_A) = \begin{cases} O(N^2) & \text{if } O(N) > O(N_d^{2/3}) \\ O(N^2) & \text{if } O(N_d^{1/2}) < O(N) < O(N_d^{2/3}) \\ O(N_d) & \text{if } O(N_d^{1/3}) < O(N) < O(N_d^{1/2}) \\ O(N^{3/2})O(N_d^{1/2}) & \text{if } O(N) < O(N_d^{1/3}) \ . \end{cases} \tag{46}$$

If N falls into $N'_{cr} \le N \le N_{cr}$ (the second line in (45)) then the following cases are considered: $O(N_d^{1/2}) < O(N) < O(N_d^{2/3})$ and $O(N_d^{1/3}) < O(N) < O(N_d^{1/2})$. In the former case the leading term is the first component whereas in the latter one the leading term is the $N\tau$ component.

The asymptotic analysis of the parallelization penalty for the basic algorithm (39) leads to the following expression

$$O(F_B) = \begin{cases} O(N^2) & \text{if } O(N) > O(N_d) \\ O(N^{3/2})O(N_d^{1/2}) & \text{if } O(N) < O(N_d)\ . \end{cases} \tag{47}$$

Thus, the proposed algorithm has an advantage over the basic one if $O(N_d^{1/3}) < O(N) < O(N_d)$. Otherwise, both algorithms are of the same order. If $O(N) > O(N_d)$, the main component in the parallelization penalty becomes $15N\tau$. This component characterizes the amount of transfered data which is the same for these algorithms. If $O(N) < O(N_d^{1/3})$, the idle time between conclusion of the forward step computations and completion of the backward step computations for the first group of lines becomes large and there is no longer an advantage of the proposed algorithm over the basic one.

The order of the parallelization penalty function in terms of the overall number of nodes and the overall number of subdomains (processors) is given by

$$O(F_A) = \begin{cases} O(N_{tot}^{2/3})/O(N_D^{2/3}) & \text{if } O(N_{tot}) > O(N_D^{3/2}) \\ O(N_D^{1/3}) & \text{if } O(N_D^{4/3}) < O(N_{tot}) < O(N_D^{3/2}), \\ O(N_{tot}^{1/2})/O(N_D^{2/3}) & \text{if } O(N_{tot}) < O(N_D^{4/3})\ , \end{cases} \tag{48}$$

where $N_{tot} = (N \times N_d)^3$ is the total number of grid nodes and $N_D = N_d^3$ is the total number of processors. The proposed algorithm has an advantage over the basic algorithm if $O(N_D^{4/3}) < O(N_{tot}) < O(N_D^2)$.

3.3. 3-D domain with different number of grid points in each direction. The case with different number of grid nodes in the x, y and z directions is considered here ($N_x \ne N_y \ne N_z$). We will obtain theoretically (where it is possible) the optimal cover of the computational domain with subdomains.

Consider first the case where the first term in Eq. (10) dominates; therefore, the parallelization penalty for the proposed algorithm is given by

$$F_A = \sum_{i=1,2,3} \left(\left\lceil \frac{N_j N_k}{K_{i,1}} \right\rceil (b_0 + 2b_1 K_{i,1}) + \left\lceil \frac{N_j N_k}{K_{i,2}} \right\rceil (b_0 + b_1 K_{i,2}) + N_i K_{i,2} g_2 + 2(b_0 + b_1 N_j N_k) \right), \tag{49}$$

where $i, \quad j, \quad k$ are the spatial directions ($i \ne j \ne k$), K_i, K_j and K_k are the numbers of lines solved per message in spatial directions, (in general, $K_i \ne K_j \ne K_k$). The first two terms in the above equation represent the communication time due to the transfer of the forward step coefficients and the backward step solution. Next term corresponds to the processor idle time due to the local synchronization. The last term is equal to the communication time due to the transfer of the FS variables. We group the terms, use Eq. (14) and replace $\lceil x \rceil$ by x, which leads to the following expression:

$$F_A = \sum_{i=1,2,3} \left(b_0(1+\rho)\frac{N_j N_k}{K_{i,2}} + N_j K_{j,2} g_2 \right) + 5b_1 \sum_{i=1,2,3} N_j N_k + 6b_0. \tag{50}$$

Both sums in the above equation will be estimated by the following known inequality:

$$\sum_{i=1}^{n} a_i \ge n\left(\prod_{i=1}^{n} a_i\right)^{1/n}, \tag{51}$$

where $\sum_{i=1}^{n} a_i = n(\prod_{i=1}^{n} a_i)^{1/n}$ for equal a_i.

The estimation for the second sum in (50) is given by

$$\sum_{i=1,2,3} (N_j N_k) \geq 3(N_i N_j N_k)^{2/3} = 3N_{ov}^{2/3}, \tag{52}$$

where $N_{ov} = N_i N_j N_k$ is the overall number of grid nodes per subdomain. This sum is minimal if $N_i = N_j = N_k = N$, i.e., if the subdomains are cubic. The first sum is estimated by

$$\sum_{i=1,2,3} (1+\rho) b_0 N_j N_k / K_{i,2} + N_i K_{i,2} g_2) \geq 6(b_0 g_2 (1+\rho) N_i N_j N_k)^{1/2} = 6(b_0 g_2 (1+\rho) N_{ov})^{1/2}. \tag{53}$$

This inequality turns into equality if $N_i = N_j = N_k = N$ and K_2 satisfies Eq. (41). Thus, the partitioning by cubic subdomains gives the minimum parallelization penalty. In this case, the optimal number of solved lines per message is the same in all directions, and the parallelization penalty is equal to the value stated in the first line of (45).

The parallelization penalty for the basic method is

$$F_B = \sum_{i=1,2,3} \left\lceil \frac{N_j N_k}{K_{i,1}} \right\rceil (b_0 + 2b_1 K_{i,1}) + \left\lceil \frac{N_j N_k}{K_{i,2}} \right\rceil (b_0 + b_1 K_{i,2}) + (N_{d,i} - 1) N_i (K_{i,1} g_1 + K_{i,2} g_2) + 2(b_0 + b_1 N_j N_k),$$

where $N_{d,i}$ is the number of subdomain partitioning in the i^{th} direction. We can show (see above), that the F_B reaches its minimum if $N_i = N_j = N_k$. In this case, optimal K values are given by

$$K_{i,1} = \sqrt{\frac{\gamma N}{\rho(N_{d,i} - 1)}}, \quad K_{i,2} = \sqrt{\frac{\gamma N}{(N_{d,i} - 1)}}. \tag{54}$$

Thus, for the basic algorithm the optimal cover is also composed of cubic subdomains; however, the optimal K values may be different for different directions. Therefore, for both algorithms the number of subdomains in each direction is proportional to the number of grid nodes in this direction, i.e., $N_{d,x} : N_{d,y} : N_{d,z} = N_{totx} : N_{toty} : N_{totz}$.

For the basic algorithm the penalty function for optimal K values is given by (39), where $6(N_d - 1)$ is replaced by $2\sum_{i=1,2,3} \sqrt{N_{d,i} - 1}$. This sum can be estimated by

$$\sum_{i=1,2,3} \sqrt{N_{d,i} - 1} \approx \sum_{i=x,y,z} \sqrt{N_{d,i}} \geq 3N_D^{1/6}. \tag{55}$$

Thus, for the basic algorithm the parallelization penalty reaches its minimum for a cubic global domain. For the proposed algorithm, the parallelization penalty is invariant with respect to the number of nodes in each direction (see the first line in Eq (45)). This represents an additional advantage of the proposed algorithm over the basic one.

If the values of K are equal to bound values K_r (42), then the cubic subdomains no longer give the minimum parallelization penalty for the proposed algorithm. However, partitioning by cubic subdomains is chosen in this case as a fair assumption.

4. Numerical solution of the sample problem. The parallelization method and the model for the parallelization efficiency are tested by a numerical solution of a benchmark problem. The non-stationary Laplace equation in the cube $\Omega = [-1 < x < 1] \times [-1 < y < 1] \times [-1 < z < 1]$ is chosen as the test case. The PDE to be solved is

$$\frac{dU}{dt} = a_x \frac{\partial^2 U}{\partial x^2} + a_y \frac{\partial^2 U}{\partial y^2} + a_z \frac{\partial^2 U}{\partial z^2}, \tag{56}$$

Table 1

Characteristic parameters of MIMD computers

	parameters of communication			non-dimensional parameters		
Computer	$b_0, \mu S$	$b_1, \mu S/Word$	$g_2, \mu S$	$\rho = g_1/g_2$	$\gamma = b_0/g_2$	$\tau = b_1/g_2$
CRAY T3E	18	0.1	0.36	1.71	50.	0.28
IBM SP	70	0.05	0.28	1.42	250.	0.18

where $a_x,\ a_y$ and a_z are non-linear functions of U. There are the following boundary and initial conditions:

$$\forall\ \ (x,y,z) \in \Omega\ \ U(x,y,z,0) = 0, \qquad (57)$$
$$\forall\ \ (x,y,z) \in \partial\Omega\ \ U(x,y,z,t) = 1.$$

The MIMD computers used in this study are installed in the San Diego Supercomputer Center (SDSC) at the University of California, San Diego [14]. SDSC's CRAY T3E has 272 (maximum 128 for a single task) distributed-memory processing elements (PEs), each with 128 megabytes (16 megawords) of memory. Each processor is a DEC Alpha 21164 (300 MHz clock). The T3Es run the UNICOS/mk operating system. The T3E PEs are relatively inexpensive, with fast processing ability but slow main memory. The theoretical limit of 600 Mflops for the 300 Mhz processors of the SDSC T3E applies only to certain operations within the registers of the processor. The IBM SP has 128 (maximum 120 for a single task) thin node POWER2 Super Chip (P2SC) processors with 256 MBytes of memory on each processor. The SP processors are superscalar (implying simultaneous execution of multiple instructions) pipelined chips and are capable of executing up to six instructions per clock cycle and four floating point operations per cycle. These nodes run at 160 Mhz and are capable of a peak performance of 640 MFLOPS each.

Our measurements of communication times (b_0 and b_1) for the CRAY T3E and the IBM SP computers are presented in Table 1. The maximum length of string is 1000 words. The sample size is 100 messages for each string. Communication time is the time required to receive a message of L words which sent from another processor. These measurements confirm the linear approximation (7) for the communication time.

Computational times are obtained from computational experiments on a single processor. Fortran compiler CF90 with the third optimization level is used on the CRAY T3E. The Fortran compiler on the SP is IBM's XL Fortran, also known as xlf. The cash locality is exploited in our computer code, i.e., the "implicit" direction (index i in Eqs. (4,5)) corresponds to the last index in working arrays for the Thomas algorithm computations in all three directions. The values of ρ are different for CRAY T3E and IBM SP computers as ratios between arithmetic operations are compiler- and computer- dependent.

4.1. Description of a multi-processor code. The code is designed as follows. First, the optimal number of grid lines to be solved per message (i.e., the optimal number of groups of lines) is computed as described in the previous section. Then, processors are scheduled in each direction according to the algorithm described in [6]. These one-directional schedules and schedules of the local computations are joined together to use the processor idle time according to recommendations presented in the second section. Thus, the static schedule of processors is formed.

The solver part of the code (Appendix A) does not depend upon a particular schedule of processors. The functions handling communications are referenced in a common form without relation to any specific message-passing system. The external loop with the loop variable IPX execute lines group-by-group. The array $ICOM$ governs communication with six closest neighbors of the current processor. The value of $ICOM(IPX, I)$ controls type of communications, as follows:

$$ICOM(IPX,I) = \begin{cases} 0 & \text{processors do not communicate} \\ 1 & \text{send} \\ 2 & \text{receive} \\ 3 & \text{simultaneous send and receive} \end{cases} \tag{58}$$

where I=1,...,6. The send and receive operations transfer either the forward step computations or the backward step solution. Before the computations for the current group of lines are executed, each processor has completed exchanging data with its neighbors. The only data used for the computations at the current time unit are transfered to the processor. At the backward step of the Thomas algorithm computed values of the solution are transfered to the processor ahead after they have been computed in the current processor at the previous time unit. These values are stored in the array *SF.* At the forward step interface values of the coefficients are not transfered immediately after they have been computed; therefore, these values are extracted from matrices of the forward step coefficients DX and GX where they are stored. Pointers $J3X$, $J3Y$ and $J3Z$ control these data streams in the directions x, y and z, respectively.

After communications have been completed, computations are executed according to the value stored in the array, ITX (see (6)). The local, forward and backward step computations are executed in sub-routines COEF, FRWD and BCWD, respectively.

4.2. Results of multiprocessor runs. Results of multiprocessor computer runs are shown in Tables 2a,b for CRAY T3E and IBM SP computers, respectively. The number of grid nodes per subdomain varies from 10^3 to 20^3 for the CRAY T3E and from 12^3 to 20^3 for the SP computer. The computational domain is covered with $3^3, 4^3$ or 5^3 equal cubic subdomains. Therefore, the total number of grid nodes varies from 27×10^3 (46.5×10^3 for the SP computer) to 10^6.

Processors compute coefficients a_y and a_z while they are idle from the Thomas algorithm in the x and y directions. These computations are data-independent. In this particular case $\alpha = 1.2$ for the CRAY T3E and $\alpha = 1.3$ for the SP computer (see Eq. (34)). The data-dependent computations of the coefficients a_x are executed while processors are idle in the z direction. The optimal K values in each direction for the proposed algorithm are computed by one of the Eqs. (41) (42), (44). These methods and denoted as 1,2 and 3, respectively, in Tables 2a,b. For the basic algorithm, optimal values of K are computed by (40). The values of K for the proposed algorithm are greater than those for the basic algorithm that confirms an advantage of the proposed algorithm in terms of the number of messages.

The parallelization penalty (Pn) is obtained by

$$Pn = \frac{T_P - T_1}{T_1} \times 100\%, \tag{59}$$

where T_P and T_1 are measured computational times on P processors and on a single processor with the same size of subdomain, respectively. The parallelization penalty obtained from computer experiments is in good agreement with that computed by the theoretical model (16). The difference between experimental and theoretical values increases with the decrease in the number of grid nodes per processor. This can be explained by those details of the computer architecture that are not taken into account in the developed theoretical parallelization model. To match theoretical and experimental results for the SP computer, experimental data are filtered. If the elapsed time for a time step is 30% greater than the averaged value, this time step is excluded from the sample. These time steps make $3 - 5\%$ of overall sample. The parallelization penalty for the basic algorithm is approximately twice as much that for the proposed algorithm for the considered range of the number of nodes per processor and the number of processors.

To estimate the parallelization penalty for the large number of processors which are not available yet we compute the theoretical parallelization penalty for 10^3 processors and present these results in Tables 2a,b. The ratio of parallelization penalties of the basic and the proposed algorithm increases with the number of processors.

Finally, the computer runs are executed for an unequal number of grid nodes in various directions. The partitionings are $8 \times 4 \times 2$ and $16 \times 2 \times 2$, the total number of processors is equal to 64. The parallelization penalty for the basic algorithm is greater for the considered cases than for the 4^3 partitioning; however, for the proposed algorithm parallelization penalty is almost equal to that for the 4^3 partitioning. This confirms the theoretical conclusions of subsection 3.3.

5. Conclusions. A parallel implicit numerical algorithm for the solution of directionally split 3-D problems is proposed. This algorithm provides exactly the same solution as its serial analogue at each time step. While executing this algorithm, processors run in a time-staggered way without global synchronization in each direction. The proposed algorithm uses the idle processor time either for computations of discretized coefficients of the PDE to be solved or for the Thomas algorithm computations in the next spatial direction. To make the algorithm feasible, the reformulated version of the pipelined Thomas algorithm is used.

Static scheduling of processors is adopted in this study. Various computational tasks which may be executed while processors are idle from the Thomas algorithm in the current direction are discussed and recommendations about optimal scheduling of processors are drawn.

A theoretical model of the parallelization efficiency is developed. This model is used to estimate the parallelization penalty for the basic and the proposed algorithms. The optimal number of lines to be solved per message is defined by this model. The asymptotic analysis shows the relations between the number of grid nodes per subdomain and the number of processors which ensure an advantage of the proposed algorithm over the basic one. Finally, this model is used for the optimal partitioning of a computational domain with an unequal number of grid nodes in spatial directions.

The parallel computer code uses the modular design technique: a schedule of the processor tasks is assigned before computations by the numerical algorithm and communications are separated from computational modules. Experiments with the multidomain code in distributed memory multiprocessor systems (CRAY T3E and IBM SP) show a reasonable parallelization penalty for a wide range of the number of grid nodes and the number of processors. The parallelization penalty agrees well with that obtained by the theoretical model.

6. Acknowledgment. The author wishes to thank Professor David Keyes (ICASE and ODU) for useful discussion about parallel numerical algorithms.

REFERENCES

[1] Ch. Hirsch, *Numerical computation of internal and external flows, Vol. 1: Fundamentals of numerical discretization*, John Wiley and Sons, 1994.

[2] D.A. Caughey, *Computational Aerodynamics, in Research Trends in Fluid Dynamics*, J.L. Lumley et al., eds., AIP Press, NY, 1996, pp. 55-59.

[3] J. Hofhaus and E.F. Van De Velde, *Alternating-direction Line-relaxation Methods on Multicomputers*, SIAM J. Sci. Comput. **17**, No. 2, (1996), pp. 454-478.

[4] Naik, N.H., Naik, V.K. and Nicoules, M., *Parallelization of a Class of Implicit Finite Difference Schemes in Computational Fluid Dynamics*, Int. J. of High Speed Computing **5**, No. 1, (1993), pp. 1-50.

[5] F.F. Hatay, D.C. Jespersen, G.P. Guruswamy, et al., *A multi-level parallelization concept for high-fidelity multi-block solvers*, Technical paper presented in SC97: High Performance Networking and Computing, San Jose, California, November 1997, http://www.hal.com/users/hatay.

[6] A. Povitsky, *Parallelization of the pipelined Thomas algorithm*, ICASE Report No. 98-48, NASA Langley Research Center, Hampton, VA.

[7] Ian Foster, *Designing and Building Parallel Programs*, Addison-Wesley, 1995, http://www.mcs.anl.gov/dbpp/.

[8] W.D. Gropp and D.E. Keyes, *Complexity of Parallel Implementation of Domain Decomposition Techniques for Elliptic Partial Differential Equations*, SIAM J. Sci. Stat. Comput. **19**, No. 2, (1988), pp. 312-326.

[9] A. Povitsky and M. Wolfshtein, *Parallelization Efficiency of CFD problems on a MIMD computer. Computers and Fluids* **26**, No. 4, (1997), pp. 359-371.

[10] Sh. H. Bokhari, *Multiphase Complete Exchange on Paragon, SP2 and CS-2*, ICASE Report 95-61, NASA Langley Research Center, Hampton, Virginia 23681-0001, 1995.

[11] T.H. Pulliam and J.L. Steger, *Recent improvements in efficiency, accuracy and convergence for implicit approximate factorization algorithms*, AIAA Paper 85-0360, AIAA 23rd Aerospace Sciences Meeting, Reno, 1985.

[12] R.W. Beam and R.F. Warming, *An implicit finite difference algorithm for hyperbolic systems in conservation form*, Journal of Computational Physics **23**, (1976), pp. 87-110.

[13] T.H. Pulliam and D. Chaussee, *A diagonal form of an implicit approximate-factorization algorithm*, Journal of Computational Physics **39**, (1981), pp. 347-363.

[14] A National Laboratory for Computational Science and Engineering, San-Diego, http://www.sdsc.edu.

Appendix A

Fragment of computer code which performs the solver of the proposed algorithm

```
c
c        loop by portions of lines solved per one message
c
 DO IPX=1,IEND
c
c ********** logistics of communications  ***************************
c
       ICOMT=ICOM(IPX,I)
c a. receive from neighboring processors (only one receive is expected)
       DO I=1,6
       IF(ICOMT.GE.2) go to 62
       END DO
go to 64
62     continue
c    receive forward step coefficients from previous subdomain
     IF (I.LE.3) THEN
        IF (ICOMT.EQ.2) THEN
        call recv(RF,2*K1,nbr(I),itagf(I))
                        ELSE
        call sendrecv(SF,K2,nbr(I),itagb(I),RF,2*K1,nbr(I),itagf(I))
END IF
        go to 64
     END IF
c    receive backward step solution from  subdomain ahead
     IF(I.GE.4) THEN
I1=I-3
IF(ICOMT.EQ.2) THEN
        call recv(RF,K2,nbr(I),itagb(I1))
                       ELSE
        IF(I.EQ.4) call STORE(SF,DX,GX,J3X)
IF(I.EQ.5) call STORE(SF,DY,GY,J3Y)
IF(I.EQ.6) call STORE(SF,DZ,GZ,J3Z)
        call sendrecv(SF,2*K1,nbr(I),itagf(I1),RF,K2,nbr(I),itagb(I1))
      END IF
      END IF
64    continue
c b. send to neighboring domains
      DO I=1,6
      IF (ICOM(IPX,I).EQ.1) then
      IF (I.GE.4) THEN
c    send forward step coefficients
I1=I-1
        IF(I.EQ.4) call STORE(SF,DX,GX,J3X)
IF(I.EQ.5) call STORE(SF,DY,GY,J3Y)
IF(I.EQ.6) call STORE(SF,DZ,GZ,J3Z)
        call send(SF,K1,nbr(I),itagf(I1))
      END IF
c    send backward step solution
      IF(I.LE.3) call send(SF,K2,nbr(I),itagb(I))
      END DO
c
c *********** logistics of computations
c
       IF (ITX(IPX).EQ.4) THEN
c computations of coefficients
         IF(IDIR.EQ.1) call  COEF(AX,BX,CX)
         IF(IDIR.EQ.2) call  COEF(AY,BY,CY)
         IF(IDIR.EQ.3) call  COEF(AZ,BZ,CZ)
       END IF
c forward step computations
       IF (ITX(IPX).EQ.1) call FRWD(DX,GX)
       IF (ITX(IPX).EQ.2) call FRWD(DY,GY)
       IF (ITX(IPX).EQ.3) call FRWD(DZ,GZ)
c backward step computations
       IF (ITX(IPX).EQ.-1) call BCWD(VX,DX,GX)
       IF (ITX(IPX).EQ.-2) call BCWD(VX,DY,GY)
       IF (ITX(IPX).EQ.-3) call BCWD(VX,DZ,GZ)
       END DO
```

a

```
  3
    --->
  3       3
    --->    --->
  3       3       3
    --->    --->    --->
  3       3       3       3
    --->    --->
  3       3       3       3
    --->    --->
  3       3       3      -3
    --->            <-->
  3       3      -3       3
            <-->
  3      -3       3      -3
    <-->            <-->
 -3       3      -3       3
            <-->    --->
  3      -3       3       3
    <-->    --->
 -3       3       3      -3
    --->            <-->
  4       3      -3       3
            <-->
  4      -3       3      -3
    <---            <-->
 -3       4      -3       3
            <-->    --->
  4      -3       3       3
    <---
 -3       4       4      -3
                    <-->
  4       4      -3       3
            <---
  4      -3       4      -3
    <---            <---
 -3       4      -3       4
            <---
  4      -3       4       4
    <---
 -3       4       4       4

  4       4       4       4

  4       4       4       4

  4       4       4       4

  3       4       4       4

  3       3       4       4

  3       3       3       4
```

b

```
  3
    --->
  3       3
    --->    --->
  3       3       3
    --->    --->    --->
  3       3       3       3
    --->    --->    --->
  3       3       3       3
    --->    --->    --->
  3       3       3       3
    --->    --->    --->
  3       3       3       3
    --->    --->    --->
  3       3       3       3
    --->    --->    --->
  3       3       3       3
    --->    --->    --->
  0       3       3       3
            --->    --->
  0       0       3       3
                    --->
  0       0       0       3

  0       0       0      -3
                    <---
  0       0      -3      -3
            <---    <---
  0      -3      -3      -3
    <---    <---    <---
 -3      -3      -3      -3
    <---    <---    <---
 -3      -3      -3      -3
    <---    <---    <---
 -3      -3      -3      -3
    <---    <---    <---
 -3      -3      -3       4
    <---    <---
 -3      -3       4       4
    <---
 -3       4       4       4

  4       4       4       4

  4       4       4       4

  4       4       4       4

  4       4       4       4

  4       4       4       4

  4       4       4       4

  4       4       4       1

  4       4       1       1

  4       1       1       1
```

FIG. 1. *The processor schedule: (a) The IB-PTA is used for the z direction, processors compute local coefficients of the non-linear PDE while they are idle from the IB-PTA; (b) the basic PTA algorithm is used, processors are idle between the forward and backward steps.* $N_d = 4, L_f = 9, L_b = 6$

a.

```
  1  1  1  1  1  1  1  1  1  1  1  1  2  2
  2  2  2  2  2  2  2  2  2  2  3  3  3  3
  3  3  3  3  3  3  3  3  4  4  4  4  4  4
  4  4  4  4  4  4  5  5  5  5  5  5  5  5
  5  5  5  5  6  6  6  6  6  6  6  6  6  6
  6  6  7  7  7  7  7  7  7  7  7  7  7  7
  8  8  8  8  8  8  8  8  8  8  8  8  9  9
  9  9  9  9  9  9  9  9  9  9 10 10 10 10
 10 10 10 10 10 10 10 10 11 11 11 11 11 11
 11 11 11 11 11 11 12 12 12 12 12 12 12 12
 12 12 12 12 13 13 13 13 13 13 13 13 13 13
 13 13 14 14 14 14 14 14 14 14 14 14 14 14
 15 15 15 15 15 15 15 15 15 15 15 15 16 16
 16 16 16 16 16 16 16 16 16 16 17 17 17 17
 z
 |__y
```

b.

```
  1  1  2  3  4  5  6  8 10 11 12 13 15 16
  1  1  2  3  4  5  6  8 10 11 12 13 15 16
  1  1  2  3  4  5  6  8 10 11 12 14 15 16
  1  1  2  3  4  5  6  8 10 11 12 14 15 16
  1  1  2  3  4  5  6  8 10 11 13 14 15 16
  1  2  2  3  4  6  6  9 10 11 13 14 15 16
  1  2  2  3  4  6  7  9 10 12 13 14 15 16
  2  2  2  3  4  6  7  9 10 12 13 14 15 16
  3  3  3  3  5  6  7  9 11 12 13 14 15 16
  4  4  4  4  5  6  7  9 11 12 13 14 15 16
  5  5  5  5  5  6  7 10 11 12 13 14 15 17
  7  7  7  7  7  7  7 10 11 12 13 14 15 17
  8  8  8  8  8  8  8 10 11 12 13 14 16 17
  9  9  9  9  9  9  9 10 11 12 13 14 16 17
  z
 |__x
```

FIG. 2. *Gather of lines in groups. (a) The Thomas algorithm in the y direction is executed while processors are idle computing the Thomas algorithm in the x direction; (b) In addition, the Thomas algorithm in the z direction is executed while processors are idle computing the Thomas algorithm in the y direction. Numbers denote the group of lines to which this line belongs.*

```
 1
   --->
 1       1
   --->    --->
 1       1       1
   --->    --->    --->
 1       1       1       1
   --->    --->    --->    --->
 1       1       1       1       1
   --->    --->    --->    --->    --->
 1       1       1       1       1       1
   --->    --->    --->    --->    --->
 1       1       1       1       1       1
   --->    --->    --->    --->
 1       1       1       1       1      -1
   --->    --->    --->            <-->
 1       1       1       1      -1       1
   --->    --->            <-->
 0       1       1      -1       1      -1
                   <-->            <-->
 0       0      -1       1      -1       1
           <-->            <-->    --->
 0      -1       1      -1       1       1
   <---            <-->    --->
-1       2      -1       1       1      -1
           <---    --->            <-->
 2      -1       2       1      -1       1
   <---                    <-->
-1       2       2      -1       1      -1
                   <---            <-->
 2       2      -1       2      -1       1
           <---            <-->    --->
 2      -1       2      -1       1       1
   <---            <---
-1       2      -1       2       2      -1
           <---                    <-->
 2      -1       2       2      -1       1
   <---                    <---
-1       2       2      -1       2      -1
                   <---            <---
 2       2      -1       2      -1       2
           <---            <---
 2      -1       2      -1       2       2
   <---            <---
-1       2      -1       2       2       2
           <---
 2      -1       2       2       2       2
   <---
-1       2       2       2       2       2

 2       2       2       2       2       2

 2      -2      -2       2       2       2

-2      -2      -2      -2       2       2

-2      -2      -2      -2      -2       2
```

FIG. 3. *The processor schedule: processors compute the forward step of the Thomas algorithm in the y direction while they are idle from the Thomas algorithm computations in the x direction*

TABLE 2

Parallelization penalties for the benchmark problem, N_{tot} - total number of nodes, N - number of grid nodes per subdomain in a single direction, K_x, K_y and K_z - number of lines solved by backward step per message, Method - method of computation of K_x, K_y and K_z values, respectively (1-by Eq. (41), 2-by Eq. (42) and 3-by Eq. (44)), Pn− penalty, obtained from computational experiments (Eq. (59)); PnM− theoretical value of penalty (Eq. (16))

a. CRAY T3E multiprocessor system.

		Proposed algorithm						Basic algorithm				
N_{tot}	N	K_x	K_y	K_z	Method	Pn, %	PnM, %	K_x	K_y	K_z	Pn, %	PnM, %
partitioning 3 x 3 x 3												
27000	10	51	51	26	112	52.41	39.90	22	22	22	108.35	96.74
91125	15	63	63	62	112	23.80	19.97	27	27	27	53.60	49.93
216000	20	74	74	74	111	15.01	13.56	31	31	31	33.86	31.33
partitioning 4 x 4 x 4												
64000	10	42	42	20	222	58.71	43.94	18	18	18	132.4	114.53
216000	15	63	63	47	112	31.10	21.34	22	22	22	59.23	57.46
512000	20	74	74	74	111	15.19	13.56	26	26	26	39.46	36.67
partitioning 5 x 5 x 5												
125000	10	35	35	17	222	67.29	54.54	16	16	16	152.20	131.61
421875	15	63	63	38	112	33.84	22.69	19	19	19	70.17	66.33
1000000	20	74	74	67	112	16.12	13.01	22	22	22	45.08	41.59
partitioning 10 x 10 x 10												
1000000	10	19	19	10	222	-	92.66	11	11	11	-	186.33
3375000	15	40	40	19	222	-	29.84	13	13	13	-	93.51
8000000	20	74	74	35	222	-	13.80	15	15	15	-	59.09
partitioning 8 x 4 x 2												
64000	10	22	42	40	222	56.74	42.37	12	18	31	136.83	119.59
216000	15	51	63	63	211	23.12	20.89	15	22	38	61.23	59.77
512000	20	74	74	74	111	15.11	13.56	17	26	44	39.23	37.71
partitioning 16 x 2 x 2												
64000	10	11	51	40	212	63.85	61.17	8	31	31	143.32	129.57
216000	15	26	63	63	211	28.39	25.76	10	38	38	67.56	64.29
512000	20	46	74	74	211	13.08	14.25	12	44	44	42.71	40.74

.

.

b. IBM SP multiprocessor system.

		Proposed algorithm						Basic algorithm				
N_{tot}	N	K_x	K_y	K_z	Method	Pn, %	PnM, %	K_x	K_y	K_z	Pn, %	PnM, %
partitioning 3 x 3 x 3												
46656	12	74	74	73	2 2 3	112.36	95.62	54	54	54	207.19	198.92
91125	15	120	120	53	2 2 2	69.64	49.06	61	61	61	122.64	128.86
216000	20	157	157	96	1 1 2	38.97	27.73	71	71	71	95.23	80.38
partitioning 4 x 4 x 4												
110592	12	62	62	62	3 3 3	129.45	115.14	45	45	45	256.17	232.11
216000	15	87	87	39	2 2 2	93.46	70.73	50	50	50	156.43	147.15
512000	20	157	157	70	1 1 2	43.85	32.04	58	58	58	103.24	93.34
partitioning 10 x 10 x 10												
1728000	12	22	22	37	2 2 3	-	264.81	26	26	26	-	359.85
3375000	15	35	35	43	2 2 3	-	146.53	29	29	29	-	239.19
8000000	20	62	62	27	2 2 2	-	68.04	34	34	34	-	150.08
partitioning 8 x 4 x 2												
110592	12	27	62	91	2 3 3	146.76	125.59	29	45	77	219.87	234.34
216000	15	43	87	103	2 2 3	74.88	64.75	33	50	87	148.63	155.21
512000	20	77	157	151	2 1 2	37.32	28.65	38	58	101	103.26	95.72
partitioning 16 x 2 x 2												
110592	12	30	117	91	3 2 3	182.76	169.87	20	77	77	246.63	253.24
216000	15	22	134	103	2 1 3	99.45	90.93	23	87	87	184.76	167.85
512000	20	39	157	151	2 1 2	52.17	41.49	26	101	101	109.72	102.56