INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS, VOL. 10, 907–923 (1990)

SOLVING NUMERICALLY THE NAVIER–STOKES EQUATIONS ON PARALLEL SYSTEMS

AVI LIN

Department of Mathematics, Temple University, Philadelphia, PA 19122, U.S.A.

SUMMARY

A new general approach for numerically computing flow fields on parallel computing environments is presented, discussed and analysed. The hierarchy presented here is based on a parallel split of operators. A portion of the theory is presented together with its application to two- and three-dimensional flows. This strategy is applied to a two-dimensional problem for which a specific parallel split, called a parabolized split, is given. The parallel algorithm that results from this split is analysed, leading to reasonably good parallel efficiency, which is close to 50%. Actual experiments lead to similar conclusions. This parallel strategy can also be used together with other parallel computing algorithms, such as domain decomposition, to give an optimal-type parallel algorithm for the Navier–Stokes equations.

KEY WORDS Parallel computing Parallel iterative methods Numerical splitting techniques

1. INTRODUCTION

It is well accepted today that the Navier–Stokes (NS) equations exhibit a good approximation for the behaviour of the motion of flows and for aerodynamics in general. Since the area of computational aerodynamics is very important, a variety of numerical schemes for solving these equations for many different cases, for different instances of the flow fields and for different geometries have been devised during the last decade. Thus naturally, computational fluid dynamics (CFD) has emerged, especially in the last few years, as an essential element for progress in two related fields:

(a) a deeper understanding of the physics of the flow
(b) the design process of most types of aerospace vehicles.

This very important revolution has been driven by advances in computer power that have led to advances in numerical techniques. The result is that better computational simulations are achieved more cost-effectively. This remarkable progress (mostly made during the last 10 years or so), which was sparked by the availability of supercomputers, was focused primarily on the development of inviscid flow field simulation techniques.

Obviously, there are some advantages and disadvantages to most of the known schemes, the available algorithms for computing the flow quantities using the chosen scheme, and the computing techniques used to implement these schemes and algorithms. One of the main bottlenecks of many of these schemes is the huge amount of CPU time consumption needed to get satisfactory solutions for the system of approximate equations. It is obvious that there exists a coupling between the computing time consumption and the nature of the scheme, but usually any

0271–2091/90/080907–17$08.50

Received May 1989
Revised October 1989

algorithm to solve these equations will require large resources of computing time and memory. This is not the only problem these schemes face in using the present computing facilities, but this paper will concentrate mainly on this issue.

Supercomputing is a resource that many computational researchers are using today to overcome this problem. Since 'supercomputing' is not a well defined mathematical notion, it is viewed here as computational hardware which consists of vector-like computers and parallel-like computers. Although it has been shown for a few cases that vectorizing the numerical scheme while using vector machines to execute the numerical algorithm gives a reasonable speed-up when compared to the standard scalar machines, it has also been shown[1] that in some cases the application of vector-like schemes to the fluid flow numerical equations when using more reasonable computing algorithms produces only a limited speed-up over scalar machines. It will be shown here that a better way of achieving a good speed-up using current schemes, which are theoretically unbounded in some sense, is by using a parallel machines as the main carrier for the computations. This is especially important since now it is quite evident that forthcoming (super) computers will have substantial capabilities of parallelism.

The present paper focuses on extending the simulation capabilities to more complex flow fields as well as obtaining a more complete treatment of aerodynamic phenomena. A major goal that combines both of these directions is the simulation of three-dimensional fluid flow around a complex body or three-dimensional internal flow. When trying to achieve such a goal, it is necessary to be cognizant of the available types of computing tools. One of the problems that appears in mathematical computing, especially in the CFD area, when a new concept of computer hardware arrives, is how to redesign efficiently the existing numerical schemes or how to invent new numerical schemes to take advantage of the special capabilities of the new machine.

This paper presents a new parallel/distributed strategy to solve numerically steady state two- and three-dimensional incompressible viscous fluid flow problems. One of the main goals of this paper is to develop a good combination of a numerical scheme and a parallel computational technique that, together with the appropriate parallel engines which are available (or will soon be available) on the market, will produce high-performance codes for computational fluid dynamics and will make it possible to examine more closely finer and finer structures that a fluid may produce during its flow. It is known that it is very difficult to solve computationally the governing (Navier–Stokes) equations for these steady state problems, while for other cases (such as unsteady fields or compressible flows) the numerical problem is a bit easier and thus simplified versions of the present schemes may be used. Although it is treated extensively in Reference 2, special attention is given in this paper to high-order numerical schemes for flow fields, which can be used for high-Reynolds-number flows. The formulation of the schemes is affected by the nature of the computer to be used. Basically, it may be extrapolated from this work that any MIMD machine with a fast interconnection network between the processors or a fast access to the shared memory can serve as a reasonable parallel computer to implement the suggested strategy, while there is no real limitation on the number of processors in this machine.

After some definitions and the problem lay-out, a new parallel/distributed scheme to compute three-dimensional viscous fluid flow is described and discussed. This scheme is quite general and does not require specific machines.

2. THE PARALLEL COMPUTATIONAL MODEL

In this section the general computational model of a parallel machine will be described. The basic features are needed in order to implement efficiently the parallel CFD algorithms. For the present

family of algorithms it is enough to describe the parallel machine as a single computer that includes:[3]

(a) multiple processors
(b) processors which may communicate and cooperate at different levels to solve a given computational problem
(c) multiprocessors controlled by the same operating system—this operating system also controls the other computer resources.

Two architecturally different parallel machines are available on the market:

(a) loosely coupled multiprocessors where each processor has a large local memory in which an appropriate part of the algorithm and the appropriate data reside—the processors communicate by exchanging messages through an interconnection network
(b) tightly coupled multiprocessors where all of the processors share a common memory—here the communication between the processors is done via the shared money.

The main bottleneck of the loosely coupled systems is in the low bandwidth of the interconnection network,[4] and although the rate at which data can communicate from one processor to another in a tightly coupled system is of the order of the bandwidth of the memory, the main reason for the performance deterioration in these systems is memory contention.[5] Since the primary breakdown in efficiency for parallel systems is in the extensive need for communication between the various resources, one of the main goals in formulating the parallel numerical scheme is to minimize the communication between the different tasks.

3. THE DISCRETE NUMERICAL MODEL

In order to solve the NS equations on a digital computer, they have to be discretized. There are several methods which transform the continuous problem to a discrete one (via finite differences, finite elements, spectral approximations, etc.). However, before discussing this area it is important to decide which mathematical description of the NS equations is the most appropriate for a parallel environment.

3.1. The governing equations

The steady state incompressible flow is governed by the continuity equation and the three momentum equations, which form the following elliptic system. We call this system I:

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}+\frac{\partial w}{\partial z}=s, \tag{1}$$

$$u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}+w\frac{\partial u}{\partial z}=-\frac{\partial p}{\partial x}+\varepsilon\nabla^2 u+f_1, \tag{2}$$

$$u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}+w\frac{\partial v}{\partial z}=-\frac{\partial p}{\partial y}+\varepsilon\nabla^2 v+f_2, \tag{3}$$

$$u\frac{\partial w}{\partial x}+v\frac{\partial w}{\partial y}+w\frac{\partial w}{\partial z}=-\frac{\partial p}{\partial z}+\varepsilon\nabla^2 w+f_3, \tag{4}$$

where u, v and w are the Cartesian components of the velocity vector $\mathbf{V}=(u, v, w)^{\mathrm{T}}$ in the x-, y- and z-directions respectively, p is the pressure function and ε is the inverse of the flow Reynolds

number. The operator ∇^2 is defined as usual

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}.$$

This system has two driven terms: s is the mass source term and $\mathbf{f} = (f_1, f_2, f_3)^T$ is the external force vector which acts on the flow. Usually these source terms are zero. System I is of the elliptic type and has to be solved in some (given) domain $\Omega \in \mathscr{R}^3$ with a well defined boundary $\partial\Omega$. The boundary conditions over $\partial\Omega$ are usually given in terms of the velocity vector and/or its gradients:

$$\mathbf{a}\mathbf{V} + \mathbf{b}\frac{\partial \mathbf{V}}{\partial \mathbf{n}} = \mathbf{c}, \tag{5}$$

where $\mathbf{a}$ and $\mathbf{b}$ are (local) 3×3 matrices and $\mathbf{c}$ is a three-element column vector. Each of the entries of the coefficients in (5) is a given function of $\partial\Omega$ and $\mathbf{V}$, while the value of the pressure is given at some point in Ω:

$$p(\mathbf{x}_0) = p_0. \tag{6}$$

In fact, (5) can be replaced by any kind of three relations between $\mathbf{V}$ and p on $\partial\Omega$ which constitute a non-singular system at each point of $\partial\Omega$.

3.2. The computational system

From the computational point of view, system I is usually referred to as the primitive variables system. In the past some difficulties have been reported when trying to solve numerically the primitive system. Roache[6] discusses some major difficulties in solving the primitive variables system. The main conclusion from this discussion is that the drawbacks of the other possibilities are much more severe than those of the primitive system. (Indeed, recent numerical procedures for the NS equations use the reduced variables formulation.) The primitive equations (system I) are an elliptic system by definition, since it can be shown that the determinant of the principal symbol of this system does not vanish for non-zero values of the dual variables[7] (this condition results from the regularity requirement of the elliptic system[8]). This definition of ellipticity can be extended to define an elliptic computational system (for the finite difference case see References 9–11), which leads to the conclusion that if $2k$ is the order of the principal values' determinant, then the elliptic system needs k boundary conditions at each point of $\partial\Omega$. Thus the NS equations need only d boundary conditions[10] at every point of $\partial\Omega$, where d is the number of dimensions of the flow domain.

In the present paper we will solve the discrete version of the primitive system I of the NS equations. This two- or three-dimensional system is non-linear and thus to obtain accurate solutions it is necessary to employ an iterative technique.[12] Next we briefly describe the possible iterative modes that will be considered later.

4. THE ITERATIVE APPROACH

The concept that will be used here for solving system I numerically is a version of the iterative factorization procedure[13] which is suited for a parallel environment. It is not the only possible approach, but this one enjoys several nice features which may make it more useful. We discuss this in more detail later.

4.1. The general technique

We start with some standard definitions.[14]

Definition 1. The *time complexity* of a scheme or of an algorithm is the time needed by the scheme to solve a problem, expressed in terms of:

(a) the size of the problem,
(b) the machine's units time.

Assume now that $\mathbf{L}$ is a partial differential operator of the elliptic type, which operates on functions ϕ, $\phi \in \mathbf{R}^d$.

Definition 2. A discrete size of $\mathbf{L}$ is the number of discrete elements in $\mathbf{R}^d$ that are considered in order to approximate $\mathbf{L}$.

For example, it is said that the discrete size of $\mathbf{L}$ is N if N is the number of grid points in Ω, when $\mathbf{L}$ is approximated by some finite difference procedure, or if N is the number of modes taken into consideration, when $\mathbf{L}$ is approximated by some kind of a spectral procedure.[15] Suppose that the following elliptic PDE has to be solved:

$$\mathbf{L}\boldsymbol{\Phi} = \mathbf{R}, \tag{7}$$

where $\mathbf{R}$ is a known right-hand side (RHS). Most of the time it is very difficult or even impossible to get an analytical solution or a direct numerical solution for this problem (especially when $\mathbf{L}$ is a non-linear operator). The usual way of solving (7) is by using some iterative procedure. Generally speaking, when using an iterative procedure, one has to start with some initial guess for the solution, say $\boldsymbol{\Phi}^{(0)}$, and then generate a sequence $\mathbf{S} = \{\boldsymbol{\Phi}^{(1)}, \boldsymbol{\Phi}^{(2)}, \boldsymbol{\Phi}^{(3)}, \dots\}$ that will converge to the solution $\boldsymbol{\Phi}$ of (7). Most of the methods usually suggest that the nth element in this sequence (the one that is calculated in the nth iteration level) is a linear combination of some k previous elements:

$$\boldsymbol{\Phi}^{(n)} = \sum_{i=n-k}^{n-1} \mathbf{L}_{n,i} \boldsymbol{\Phi}^{(i)}, \tag{8}$$

where k is the *depth* of the recursive scheme. These procedures differ from one another:

(a) in the value of k
(b) in the way the operators $\mathbf{L}_{i,j}$ are constructed.

In the present approach a depth 1 ($k = 1$) recurrence iterative procedure will be considered, while each iteration level consists of a fixed number of substeps as will be explained later. Thus instead of (8) the following iteration procedure will be considered,

$$\boldsymbol{\Phi}^{(n+1)} = \mathbf{M}^{(n)} \boldsymbol{\Phi}^{(n)} + \mathbf{N}^{(n)}, \tag{9}$$

and in order to satisfy the original equation (7), the operator $\mathbf{M}$ and the function $\mathbf{N}$ have to satisfy the following relation at each level n:

$$\mathbf{L}(\mathbf{I} - \mathbf{M}^{(n)})^{-1} \mathbf{N}^{(n)} = \mathbf{R}. \tag{10}$$

We will restrict the present iterative procedure to be of the stationary type, i.e. $\mathbf{M}$ and $\mathbf{N}$ are not functions of the iteration level n. Although this assumption may contradict the non-linear fashion of the operator $\mathbf{L}$ as defined by the NS equations, it holds at least locally in the iterative directions and therefore the conclusions may be applied only for a certain number of iterations (this

approach is sometimes called the quasi-linear iterative approach). The main difference between the intermediate results of a substep and the one-step results of the iterative procedure in a given iteration level is that the substep's results do not have to be a solution to (7), not even in the convergent state, while the iteration procedure is designed to satisfy (7) in the convergent state. If **N** is a known stationary function (and thus should not depend on n), then in general it can be assumed that it depends only on **R**. For example,

$$\mathbf{N}=\mathbf{GR}, \tag{11}$$

where **G** is a given operator. If **M** is not a function of n, then it can only be a function of **L**. Assume

$$\mathbf{M}=\mathbf{M}_1-\mathbf{HL}. \tag{12}$$

Thus from (10) it follows that

$$(\mathbf{G}-\mathbf{H})\mathbf{L}=\mathbf{I}-\mathbf{M}_1. \tag{13}$$

This equation defines a family of iterative methods which differ from one another in their choices for the three operators **G**, **H** and $\mathbf{M}_1$. We will concentrate on two different approaches for generating these operators.

4.2. Splitting and factoring

The techniques for finding **M** and **N** are divided into two major classes: the factoring class and the splitting class. In the first technique the operator **L** is approximated by a product of several operators whose inverse is well defined and can be computed directly and easily, and the iteration loop is performed in order to 'bridge' over the difference between the approximation to the operator **L** and the operator **L** itself. In the second technique several different operators which are 'close' to the operator **L** are identified so that their inverses can be computed easily on the given computer system. Then the approximate solution to (7) will consist of some (linear) combination of these approximate solutions. Again the iterations are performed to narrow the gap between the approximate solution to (7) and the exact one.

4.2.1. The factoring techniques. This procedure is designed based on the assumption that **L** can be decomposed into d operators as

$$\mathbf{L}\equiv\sum_{i=1}^{d}\mathbf{A}_i. \tag{14}$$

Given a sequence of scalars $\lambda=\{\lambda_i\}_{i=1}^{d}$, let us define the ($\hat{}$) operators as

$$\hat{\mathbf{A}}_i\equiv\mathbf{A}_i+\lambda_i\mathbf{I}. \tag{15}$$

It can be shown that if it is known how to reasonably and quickly (numerically) invert the operator $\mathbf{A}_i$, then the same procedure can be used to find the inverse of $\hat{\mathbf{A}}_i$. Based on this observation the present iterative technique is constructed as follows. Denoting by $\boldsymbol{\Phi}^{(n)}$ the approximation for $\boldsymbol{\Phi}$ at the nth iteration level, and by $\boldsymbol{\Psi}_j$ the results of the jth substep at this level, the computations at the nth level of the algorithm start by setting

$$\boldsymbol{\Psi}_0\equiv\boldsymbol{\Phi}^{(n-1)}; \tag{16}$$

then d substeps are executed, where the ith substep is given by

$$\hat{\mathbf{A}}_i\boldsymbol{\Psi}_i=\sum_{j=0}^{i-1}\mathbf{E}_{i,j}\boldsymbol{\Psi}_j+\mathbf{F}_i,\quad i=1,2,\ldots,d, \tag{17}$$

and the result of the last substep is stored in $\mathbf{\Phi}^{(n)}$:

$$\mathbf{\Phi}^{(n)}=\mathbf{\Psi}_d. \tag{18}$$

Here $\{\mathbf{E}_{i,j}\}$ is a given sequence of operators and $\{\mathbf{F}_i\}$ is a given sequence of functions. At each substep of the algorithm the appropriate **E** operators and **F** function have to be determined based on the given factoring of the set of operators $\hat{\mathbf{A}}$ and the values of $\mathbf{\Phi}$ and $\mathbf{\Psi}$ that are known up to the current iteration level. It is reasonable to assume that the **F** function is related to **R** as follows:

$$\mathbf{F}_i=\mathbf{Q}_i\mathbf{R}. \tag{19}$$

The more common case is when $\mathbf{E}_{i,j}=0$ for $j<i-1$. Let us denote $\mathbf{E}_{i,i-1}\equiv\mathbf{E}_i$ and

$$\mathbf{D}_i\equiv\hat{\mathbf{A}}_i^{-1}\mathbf{E}_i; \tag{20}$$

then it follows that

$$\mathbf{M}=\prod_{i=1}^{d}\mathbf{D}_i \tag{21}$$

and

$$\mathbf{G}=\sum_{i=1}^{d}\left(\prod_{j=i+1}^{d}\mathbf{D}_i\right)\hat{\mathbf{A}}_i^{-1}\mathbf{Q}_i. \tag{22}$$

This factorization scheme is quite general and has $2d$ free operators. Application of the condition given in (13) for this case is

$$\sum_{i=1}^{d}\left(\prod_{j=1}^{i}\hat{\mathbf{A}}_j^{-1}\mathbf{E}_j\right)\hat{\mathbf{A}}_i^{-1}\mathbf{Q}_i-\prod_{i=1}^{d}\mathbf{E}_i^{-1}\hat{\mathbf{A}}_i=\mathbf{I}. \tag{23}$$

Later this condition will be stated clearly. Figure 1 depicts schematically the iteration loop when using this technique.

Most of the implicit iterative methods discussed in the literature are of the factorization type. The classical implicit methods have been discussed by Douglas[16] and Douglis and Nirenberg.[17] The classical factoring method is that of Peaceman and Rachford[18] and was developed for the case where

$$\mathbf{A}_1=\frac{\partial^2}{\partial x^2},\qquad \mathbf{A}_2=\frac{\partial^2}{\partial y^2},\qquad \lambda=\text{function }(\Delta),$$

with Δ the time step. For this case the following was chosen:

$$\mathbf{\Psi}_0=\mathbf{\Phi}^{(n)},$$
$$(\lambda_1\mathbf{I}-\mathbf{A}_1)\mathbf{\Psi}_1=(\lambda_1\mathbf{I}+\mathbf{A}_2)\mathbf{\Psi}_0,$$
$$(\lambda_2\mathbf{I}-\mathbf{A}_2)\mathbf{\Psi}_2=(\lambda_2\mathbf{I}+\mathbf{A}_1)\mathbf{\Psi}_1.$$

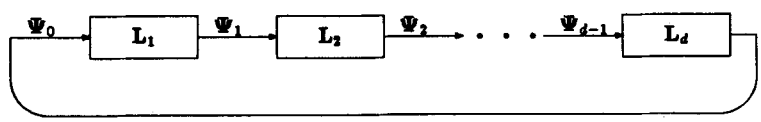

Figure 1. Pipeline scheme for the factorizing technique

They gave to the $\lambda\mathbf{I}$ the meaning of marching in the time-like direction (see Section 4.1.3) as follows. Assume that instead of (7) the following equation is considered:

$$\left(\frac{\partial}{\partial t}-\mathbf{L}\right)\mathbf{\Phi}=-\mathbf{R},$$

where t is a time-like ('parabolic') co-ordinate. Then if central differences are considered for the time term, the λs are proportional to the inverse of the marching time step. For this case the above set of equations presents an approximation of the order of two in the time-like direction. The main disadvantage of this technique, as pointed out by D'Yakonov[19] and Mitchell and Griffiths,[20] is that it will lose its accuracy if the boundary conditions become time-dependent (which is irrelevant in the present case).

4.2.2. The splitting technique. The splitting techniques are similar to the factoring techniques. Assume as before that $\mathbf{L}$ can be split into d operators. Similar to the development done before for the factoring technique, each iterative level of the splitting technique has also d substeps. The main difference between these two approaches is that the results of each substep of the splitting method should not rely on any of the results from the previous substeps. Thus the *d independent* substeps are given by

$$\mathbf{A}_i\mathbf{\Psi}_i=\mathbf{R}-\mathbf{B}_i\mathbf{\Phi}^{(n-1)}, \quad i=1, 2, \ldots, d, \tag{24}$$

and

$$\mathbf{\Phi}^{(n)}=\sum_{i=1}^{d}\boldsymbol{\alpha}_i\mathbf{\Psi}_i, \tag{25}$$

where $\boldsymbol{\alpha}=\{\boldsymbol{\alpha}\}_{i=1}^{d}$ are d operators with the condition

$$\sum_{i=1}^{d}\boldsymbol{\alpha}_i=\mathbf{I} \tag{26}$$

and the d operators $\mathbf{B}_i$ satisfy the condition given in the original equations, or after some algebraic manipulation:

$$\sum_{i=1}^{d}\boldsymbol{\alpha}_i\mathbf{A}_i^{-1}(\mathbf{L}-\mathbf{B}_i)=\mathbf{I}. \tag{27}$$

A simple schematic parallel implementation of this approach is shown in Figure 2. Another condition that the free operators $\boldsymbol{\alpha}_i$ have to satisfy is that the iterative processes will converge in the maximum speed-up. It can be shown that this condition means

$$\lambda\left[\left(\sum_{i=1}^{d}\boldsymbol{\alpha}_i\mathbf{A}_i^{-1}\right)\mathbf{A}\right]\to 1, \tag{28}$$

where λ is now the spectral radius of an operator (or a matrix). Several splitting-type techniques are used in the literature (examples appear in References 21 and 22). It is evident that not much work has been done in this direction, probably owing to the fact that splitting methods are much slower on serial computers than factoring methods. However, versions of the approximate correction procedures[23] and of the Hopscotch procedure[24] can be presented as splitting methods and not only in the factoring version. The latest observation as it appears in the literature is that often more than one factoring or splitting formula can be chosen to obtain a computationally attractive process.

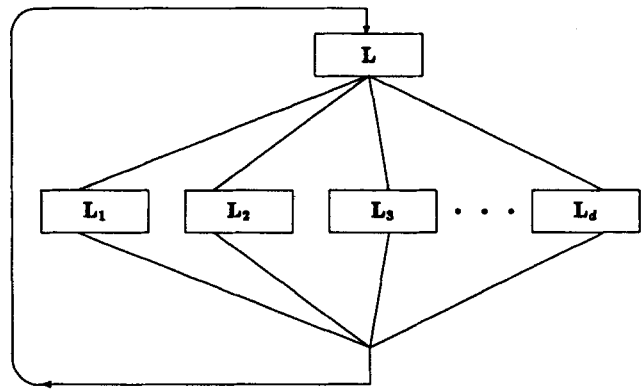

Figure 2. Parallel scheme for the splitting technique

4.3. *Parallel iterative procedure*

From the above discussion it seems that the splitting procedure is appropriate to be implemented on parallel machines. Let us demonstrate this technique as is described in Section 5.2.2 for $d=2$.

Assume that the operator $\mathbf{L}$ in (14) is split into

$$\mathbf{L}=\mathbf{A}_1+\mathbf{A}_2. \tag{29}$$

Then each parallel nth iteration can be written as follows:

$$(\mathbf{A}_1-\lambda_1\mathbf{I})\psi_1=\mathbf{b}-(\mathbf{A}_2-\omega_1\mathbf{I})\boldsymbol{\phi}^{(n)}, \tag{30}$$

$$(\mathbf{A}_2-\lambda_2\mathbf{I})\psi_1=\mathbf{b}-(\mathbf{A}_1-\omega_2\mathbf{I})\boldsymbol{\phi}^{(n)}, \tag{31}$$

and the next approximation is given as

$$\boldsymbol{\phi}^{(n+1)}=\alpha_1\psi_1+\alpha_2\psi_2, \tag{32}$$

where α_1 and α_2 are two known coefficient matrices. This scheme can be written as

$$\begin{aligned}\boldsymbol{\phi}^{(n+1)}=&[\alpha_1(\mathbf{A}_1-\lambda_1\mathbf{I})^{-1}+\alpha_2(\mathbf{A}_2-\lambda_2\mathbf{I})^{-1}\mathbf{b}-[\alpha_1(\mathbf{A}_1-\lambda_1\mathbf{I})^{-1}(\mathbf{A}_2-\omega_1\mathbf{I})\\&+\alpha_2(\mathbf{A}_2-\lambda_2\mathbf{I})^{-1}(\mathbf{A}_1-\omega_2\mathbf{I})]\boldsymbol{\phi}^{(n)}.\end{aligned} \tag{33}$$

If the exact solution to this problem is $\boldsymbol{\phi}$ and we denote the error function by

$$\varepsilon^{(n)}\equiv\boldsymbol{\phi}-\boldsymbol{\phi}^{(n)}, \tag{34}$$

then it can be shown that

$$\varepsilon^{(n+1)}=\mathbf{M}\varepsilon^{(n)}, \tag{35}$$

where

$$\mathbf{M}=\alpha_1(\mathbf{A}_1-\lambda_1\mathbf{I})^{-1}(\mathbf{A}_2-\omega_1\mathbf{I})+\alpha_2(\mathbf{A}_2-\lambda_2\mathbf{I})^{-1}(\mathbf{A}_1-\omega_2\mathbf{I}) \tag{36}$$

and is subject to the condition

$$(\alpha_1+\alpha_2)+(\omega_1+\lambda_1)\alpha_1(\mathbf{A}_1-\lambda_1\mathbf{I})^{-1}+(\omega_2+\lambda_2)\alpha_2(\mathbf{A}_2-\lambda_2\mathbf{I})^{-1}=\mathbf{I}. \tag{37}$$

Let us consider as a simple example the standard serial ADI schemes.[11] Here the scalar λs are the inverse of the half time step size, and the ωs are just the negative of the respective λs, i.e.

$$\lambda_1 = -\omega_1, \qquad \lambda_2 = -\omega_2. \tag{38}$$

Then a relationship similar to (28) holds:

$$\mathbf{M} = [\alpha_1(\mathbf{A}_1 - \lambda_1\mathbf{I})^{-1} + \alpha_2(\mathbf{A}_2 - \lambda_2\mathbf{I})^{-1}]\mathbf{L} - \mathbf{I}. \tag{39}$$

The main problem is to find a set of six parameters $\lambda_{1,2}$, $\omega_{1,2}$ and $\alpha_{1,2}$ such that the rate of convergence will be maximum or $|\lambda(\mathbf{M})|$ will be as small as possible. This by itself is a very difficult mathematical problem. In addition, it can be seen intuitively that there exists a specific split of $\mathbf{L}$ that will also maximize the convergence rate of the iterative process. Given that, the optimization problem is almost not solvable and some assumptions or approximations for the values of some of the parameters are needed.

In the rest of this paper we will analyse a parallel numerical scheme for the two-dimensional NS equations which is optimal over a subset of these parameters.

5. A PARALLEL SCHEME FOR TWO-DIMENSIONAL FLOWS

In this section we will develop a very specific and simple parallel algorithm for two-dimensional viscous flows based on the theory outlined in the preceding section, along with some assumptions which make the analysis easier. This algorithm was actually implemented on parallel computing environments.

5.1. The computational scheme

The governing equations are extracted from equations (1)–(3):

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0, \tag{40}$$

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{\partial p}{\partial x} + \varepsilon\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right), \tag{41}$$

$$u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{\partial p}{\partial y} + \varepsilon\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right). \tag{42}$$

A parallel solution of these equations is obtained by splitting the differential operator defined by this system into two differential suboperators which are solved concurrently. From the previous discussion it follows that the more the equations describing each of the suboperators resemble the original operator the better is the scheme (in terms of its stability and rate of convergence). On the other hand, these suboperators should be invertible in reasonable CPU time and direct access memory consumption. The following split was chosen.

1. A_x is an operator of the parabolized type in the x-direction.
2. A_y is a parabolized operator in the y-direction.

The notion of 'parabolized' operator is well defined[24] and is well understood.[25] The main reason for this choice is that there are many cases for which the parabolized NS approximation can capture most of the flow's important features in most of the flow domain.[12,26] Let us denote by the subscript 1 the results of the suboperator A_x and by the subscript 2 the results of the

suboperator A_y. Thus the equations which result from A_x are

$$\frac{\partial u_1}{\partial x}+\frac{\partial v_1}{\partial y}=0, \tag{43}$$

$$u\frac{\partial u_1}{\partial x}+v\frac{\partial u_1}{\partial y}=-\frac{\partial p_1}{\partial x}+\varepsilon\left(\frac{\partial^2 u_1}{\partial y^2}+R_1\right), \tag{44}$$

$$u\frac{\partial v_1}{\partial x}+v\frac{\partial v_1}{\partial y}=-\frac{\partial p_1}{\partial y}+\varepsilon\left(\frac{\partial^2 v_1}{\partial y^2}+R_2\right) \tag{45}$$

The equations which result from the suboperator A_v are

$$\frac{\partial u_2}{\partial x}+\frac{\partial v_2}{\partial y}=0, \tag{46}$$

$$u\frac{\partial u_2}{\partial x}+v\frac{\partial u_2}{\partial y}=-\frac{\partial p_2}{\partial x}+\varepsilon\left(\frac{\partial^2 u_2}{\partial x^2}+S_1\right), \tag{47}$$

$$u\frac{\partial v_2}{\partial x}+v\frac{\partial v_2}{\partial y}=-\frac{\partial p_2}{\partial y}+\varepsilon\left(\frac{\partial^2 v_2}{\partial x^2}+S_2\right). \tag{48}$$

The source terms R and S are designed to bridge over the difference between each of the operators A_x and A_y and the original operator (given by (43)–(48)). Numerically, these equations are solved iteratively, where the calculated source terms are lagged one step behind the fields of $u_{1,2}$, $v_{1,2}$ and $p_{1,2}$. Thus the iterative systems can be described in general as follows.

Given $R_{1,2}$ and $S_{1,2}$ with u and v until convergence.

1. Iterate until the source terms converge.
   (a) Calculate the new fields $u_{1,2}$, $v_{1,2}$ and $p_{1,2}$.
   (b) Calculate the new source terms $R_{1,2}$ and $S_{1,2}$.
2. Calculate the new convection coefficients u and v.

One of the main advantages of this choice for A_x and A_y is that the solution of each operator fulfils the continuity condition, which is a very important conservation law to be kept during the solution process of flow fields in general.[12,24]

It is not clear as of now what values should be substituted for the convection coefficients u and v in the above equations. Several of the possibilities are:

(a) the current values of u and v
(b) the current values of (u_1, v_1), v_1) and (u_2, v_2) respectively.
(c) the values of u and v from the last global iteration.

Of course there are more possibilities. The discussion of this issue is delayed until the next section where numerical solutions for the system are discussed. Now the two solutions, that of A_x and that of A_y, have to be combined so that the result may serve as a good approximation for the solution of the original system of equations. In the present paper a linear combination is assumed:

$$\begin{pmatrix} u \\ v \\ p \end{pmatrix} = \mathbf{C}_1 \begin{pmatrix} u_1 \\ v_1 \\ p_1 \end{pmatrix} + \mathbf{C}_2 \begin{pmatrix} u_2 \\ v_2 \\ p_2 \end{pmatrix}.$$

In order for this procedure to converge we have to have

$$\mathbf{C}_1 + \mathbf{C}_2 = \mathbf{I}.$$

In the present paper we further assume

$$\mathbf{C}_1=\begin{bmatrix}\alpha_1 & 0 & 0\\ 0 & \alpha_2 & 0\\ 0 & 0 & \alpha_3\end{bmatrix}, \qquad \mathbf{C}_2=\begin{bmatrix}\beta_1 & 0 & 0\\ 0 & \beta_2 & 0\\ 0 & 0 & \beta_3\end{bmatrix}.$$

Obviously this is quite a limited approach and one can assume a more general combination. However, it turns out that it gives very encouraging results. If continuity has to be preserved automatically, then

$$\alpha_1=\alpha_2=\alpha, \qquad \beta_1=\beta_2=\beta.$$

If the coefficients of u and v in both operators are the same, then

$$\alpha_3=\alpha, \qquad \beta_3=\beta$$

and the conditions on the source terms are

$$\alpha R_1+\beta S_1=\alpha\frac{\partial^2 u_1}{\partial x^2}+\beta\frac{\partial^2 u_2}{\partial y^2}, \qquad \alpha R_2+\beta S_2=\alpha\frac{\partial^2 v_1}{\partial x^2}+\beta\frac{\partial^2 v_2}{\partial y^2}. \tag{49}$$

For the present set-up we look for the appropriate values of the convection terms and the source terms so that the rate of convergence will be as large as possible.

5.2. The parallel computing method

The model of computation that will be assumed and used here was described in Section 2. It is assumed that the parallel environment consists of several loosely coupled processors which are clustered into two groups: one that is used by the operator A_x and the other that is used by the operator A_y.

The flow of the computation is such that there are two levels of parallelism. In the coarse-grain level the two subsystems (43)–(45) and (46)–(48) are solved concurrently. Each of these subsystems is solved by marching along the splitting direction (which is, for example, x for the A_x operator) and solving implicitly along the other direction. The marching procedure is performed in the parabolized-like fashion which is well documented in Reference 25. The governing equations along the normal direction are of the boundary value problem (BVP) type.[15] Here the fine-grain parallelism enters as the BVP equations are solved locally in a fine parallel manner. This parallel procedure is based on the co-ordinate-split strategy which is well documented in References 1 and 27. After the two subsystems are solved, the expressions which construct the source terms are calculated. Since this is an explicit or a passive step, these elements can be calculated in parallel using the appropriate processor, almost without shuffling data around.

Here the processors under the control of A_x compute (in parallel) the values of $\partial^2 u_1/\partial x^2$ and $\partial^2 v_1/\partial x^2$ and, concurrently, those under the control of A_y compute in parallel the values of $\partial^2 u_2/\partial y^2$ and $\partial^2 v_2/\partial y^2$.

Finally there is one step which is a scalar-like operation: the calculation of the flow velocities and the source terms themselves. But since this is a passive operation (i.e. all the elements involved carry old values), these calculations can also be executed in parallel. However, if this step is executed concurrently by all the available processors, it may need a lot of data transfer which may affect the time complexity dramatically.

The numerical scheme used to solve the parabolized equations for given convection coefficients and source terms is similar to the one used in References 24 and 25. However, an appropriate correction was added to the computational system in order to maintain second-order accuracy for

all the finite-differenced terms. This procedure is unconditionally stable in the linear case and converges relatively fast in the general case.

5.3. *Stability of the global iteration*

Here we wish to check the stability situation due to the lagged calculations of the source terms. Most of the analysis will be done for the linear case. For the non-linear case, since its solution is obtained by using a second-order rate of convergence algorithm (such as the quasi-linear algorithm), this analysis can be applied to each of its linear steps.

In order to get some general results on the stability condition, let us start with the Poisson equation

$$f_{xx}+f_{yy}=0.$$

The parabolized iterative version for this equation is

$$\phi_{xx}^{(n+1)}-G(\phi^{(n+1)})=-[\alpha\psi_{yy}^{(n)}+(1-\alpha)\phi_{yy}^{(n)}]-G(\phi^{(n)}),$$
$$\psi_{yy}^{(n+1)}-H(\psi^{(n+1)})=-[\beta\phi_{xx}^{(n)}+(1-\beta)\psi_{xx}^{(n)}-H(\psi^{(n)}),$$

where α and β are two scalar coefficients. Although we will assume hereafter that G and H are two scalar coefficients, in general they can be operators which operate on the scalar values of the two functions ϕ and ψ.

In order to analyse this simple iterative system, we have to be more specific about the numerical procedure. We choose here to solve this system by using a finite difference scheme. Let h and k be the (constant) grid spacings and N_x and N_y be the number of grid points in the x- and y-direction respectively. In the case of convergence the error functions are defined as

$$\varepsilon^{(n)}=\phi^{(\infty)}-\phi^{(n)}, \qquad \partial^{(n)}=\psi^{(\infty)}-\psi^{(n)}.$$

The analysis of the stability and convergence is done here using the von Neumann approach.[28] Let

$$\gamma_i=\frac{\pi i}{N_x}, \quad 1\leqslant i<N_x, \qquad \theta_j=\frac{\pi j}{N_y}, \quad 1\leqslant j\leqslant N_y.$$

Then the error functions are expressed as

$$\varepsilon_{i,j}^{(n)}=\sum_{k,l} A_n\exp(\mathrm{i}\gamma_k i)\exp(\mathrm{i}\theta_l j), \qquad \delta_{i,j}^{(n)}=\sum_{k,l} B_n\exp(\mathrm{i}\gamma_k i)\exp(\mathrm{i}\theta_l j),$$

where $\mathrm{i}^2=-1$ and A_n and B_n are the amplification coefficients. The main assumption that we will use later in connection with the amplification coefficients is

$$A_n=\bar{\phi}\lambda^n, \qquad B_n=\bar{\psi}\lambda^n.$$

Although it is not shown explicitly, λ obviously depends on the (i,j)th mode of the error function. Now we need to estimate the values of H, G, α and β that will minimize the amplification factor λ over all the possible modes. It can be shown that for each of the modes, λ fulfils the following quadratic equation:

$$A\lambda^2+B\lambda+C=0,$$

where

$$\begin{aligned} A&=(H+\bar{\theta})(G+\bar{\gamma}),\\ B&=(H+\bar{\theta})[(1-\alpha)\bar{\theta}-1]+(G+\bar{\gamma})[(1-\beta)\bar{\gamma}-1],\\ C&=[H-(1-\beta)\bar{\gamma}][G-(1-\alpha)\bar{\theta}]-\alpha\beta\bar{\gamma}\bar{\theta}, \end{aligned} \tag{50}$$

with

$$\bar{\theta}=\frac{4}{k^2}\sin^2\left(\frac{\theta_j}{2}\right), \qquad \bar{\gamma}=\frac{4}{h^2}\sin^2\left(\frac{\gamma_i}{2}\right).$$

It can be shown that the maximum values of λ occur when the angles γ and θ are minimum. Unfortunately it can be proven that λ cannot have any local minimum with respect to (real values of) G and H and thus the optimum situation has to be sought in some other way. However, it can be proven that in order for this scheme to be stable and convergent the following inequalities have to be satisfied:

$$\bar{\gamma}(G-\bar{\theta})\beta+\bar{\theta}(H-\bar{\gamma})\alpha<G\bar{\gamma}+H\bar{\theta},$$

$$\sin\bar{\gamma}(G+\bar{\gamma})\beta+\bar{\theta}(H+\bar{\theta})\alpha<(G+\bar{\gamma})(H+\bar{\theta}+\bar{\gamma}-1)+(H+\bar{\theta})(G+\bar{\theta}+\bar{\gamma}-1).$$

From examining the results of several test cases we observed that as far as the dependence of the rate of convergence on α and β is concerned, the minimum value of $|\lambda|$ happens to occur for very small positive values of these parameters. For this case it can be shown that

$$|\lambda^2|=\frac{(H-\bar{\gamma})(G-\bar{\theta})}{(H+\bar{\theta})(G+\bar{\gamma})}+\frac{\beta\bar{\gamma}(G-\bar{\theta})+\alpha\bar{\theta}(H-\bar{\gamma})}{(G+\bar{\gamma})(H+\bar{\theta})}.$$

If indeed the values of $\bar{\gamma}$ and $\bar{\theta}$ are very small compared with those of H and G, then

$$\lambda=1-(\bar{\theta}+\bar{\gamma})\frac{G+H}{2GH}+\ldots.$$

Table I depicts some examples of this dependence of the best λ on the values of G and H as reflected by equation (50).

It should be noted that since the equations for λ are non-linear, we may expect to have more than one optimal couple (G, H) that will give the same value for λ. In practice we choose

$$4\bar{\theta}=\frac{G}{G+H}, \qquad 4\bar{\gamma}=\frac{H}{G+H}.$$

It was found that these values give a near-optimal rate of convergence in actual calculations.

5.4. The two-dimensional calculations

Two cases for which this strategy was applied are discussed: the well known square driven cavity problem 2 and the flow over a backstep.[29] Refering to the governing equations (40)–(42), in order

Table I

$N_x \times N_y$	G	H	λ
10 × 10	1·32	1·5	0·8269
	1·404	1·404	
20 × 20	1·48	1·35	0·8560
	2·0	1·09	
	1·411	1·411	
40 × 40	1·46	1·37	0·8635
	1·86	1·14	
	1·4137	1·4135	

to reduce the amount of calculation while not damaging the numerical scheme, we have chosen $S_1 = S_2 = 0$. Also a term of the type $\mathbf{G}\phi$ was subtracted from both sides of (44), (47) and a similar term $\mathbf{H}\phi$ was subtracted from both sides of (45), (48). Here $\phi = (u, v, p)^T$ and $\mathbf{H}$ and $\mathbf{G}$ are two 3×3 constant matrices. We have chosen these matrices as diagonal with a zero value for their first entry:

$$\mathbf{G} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & G_2 & 0 \\ 0 & 0 & G_3 \end{bmatrix}, \qquad \mathbf{H} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & H_2 & 0 \\ 0 & 0 & H_3 \end{bmatrix}.$$

The iterative procedure consists of several loops. The outside loop is due to the quasi-linearization.[2] At this level of iteration no parallel procedure can be enforced since it is an inherent serial recursive system of depth 1 (i.e. $g_i = f(g_{i-1})$). We start applying the parallel strategy at the linear level. The stability analysis of these equations is far more complicated than that in the previous section. In any event it is possible to get some bounds on the parameters $\mathbf{G}$ and $\mathbf{H}$. For example it can be proven that

$$\sin\left(\frac{\pi}{N}\right) \leqslant 1 - 2\varepsilon \frac{H_{2,3} + G_{2,3}}{H_{2,3} G_{2,3}} - \sqrt{\left[\left(1 - \frac{u}{h}\bar{\theta} - \frac{v}{k}\bar{\gamma}\right)^2 + \left(G + H + \frac{\varepsilon}{2}\bar{\theta}^2\right)\right]} + O(\bar{\theta}^4, \bar{\gamma}^4),$$

where $N = N_x = N_y$ and $G_{2,3}$ means $\mathrm{MAX}(|G_2|, |G_3|)$. Of course, better and more accurate values of G and H can be obtained but the procedure is very involved. In Table II values of G and H for various Reynolds numbers are depicted for both flows (the square cavity and the backstep flow) along with some other details. The calculations were performed on the Allaint parallel computer. In order to exemplify the flexibility of this approach, three processors were devoted to the A_s operator and four processors were devoted to the A_y operator. The results are presented for 401 $\times$ 763 grid points. We denote the values of G and H for the x-momentum equation by (G_1, H_1). Similarly, (G_2, H_2) are the values of G and H for the y-momentum equation. The convection terms were treated by second-order upwind differencing.[2] We measure here the quality of the parallel algorithm by the ratio between the pure parallel CPU time and the CPU time needed to solve this problem on one of the processors in the parallel environment, using a similar serial algorithm. Defining this quantity as an efficiency, the numerical experiments presented in Table II suggest that the parallel efficiency approaches 50%. This result coincides with a similar result for parallel computing of elliptic operators and boundary value problems[15] for which we have gotten a similar result.

6. CONCLUSIONS

A new approach for the parallel numerical computation of the two- and three-dimensional Navier–Stokes equations is developed and discussed. The method is based on the idea of a parallel operator split. This new direction for parallel computing is not fully developed yet, and the good results presented in the present paper suggest that more careful attention and effort should be devoted to this idea. Although the theory of such a split is not complete yet, we have shown how this approach can be implemented in two-dimensional simple flows, for which we have been able to get theoretical results for the approximated rate of covergence of the parallel iterative methods.

It is expected that the parallel split theory will spin off in the near future and will resolve some of the questions that were encountered during the course of this work, such as: Since there are several possibilities for the parallel split for a given set of PDEs (which are stable and consistent), what is the best split (in terms of parallel rate of convergence)? How does the parallel hardware influence

Table II

	Re		20	50	100	200	500	1000
Square cavity	G_1	min	1·53	1·42	1·39	1·38	1·38	1·37
		max	3·41	3·35	3·29	3·24	3·19	3·16
		opt	2·54	2·51	2·48	2·46	2·45	2·44
	G_2	min	1·44	1·42	1·41	1·40	1·40	1·40
		max	1·78	1·75	1·73	1·71	1·70	1·69
		opt	1·63	1·61	1·60	1·60	1·59	1·59
	H_1	min	1·51	1·50	1·49	1·49	1·49	1·49
		max	1·93	1·90	1·88	1·87	1·87	1·86
		opt	1·74	1·74	1·73	1·73	1·73	1·72
	H_2	min	2·05	2·02	2·01	1·99	1·98	1·98
		max	2·61	1·57	2·54	1·51	1·49	2·48
		opt	2·35	2·33	2·32	2·31	2·31	2·30
	Efficiency		0·48	0·47	0·44	0·41	0·40	0·40
Back step	G_1	min	1·74	1·67	1·61	1·55	1·49	1·46
		max	4·23	4·15	4·09	4·04	4·01	3·98
		opt	2·94	2·89	2·86	2·84	2·83	2·83
	G_2	min	1·40	1·37	1·35	1·33	1·32	1·31
		max	1·71	1·67	1·64	1·62	1·60	1·59
		opt	1·57	1·54	1·52	1·50	1·49	1·49
	H_1	min	1·43	1·42	1·41	1·41	1·40	1·40
		max	1·82	1·80	1·79	1·78	1·78	1·78
		opt	1·67	1·66	1·66	1·66	1·65	1·65
	H_2	min	3·37	3·35	3·33	3·32	3·31	3·31
		max	3·77	3·74	3·71	3·69	3·68	3·67
		opt	3·56	3·55	3·54	3·53	3·52	3·52
	Efficiency		0·48	0·48	0·46	0·43	0·42	0·41

the choice of the appropriate split? Although some of the answers are underway,[30] much work remains to be done.

REFERENCES

1. A. Lin, 'Parallel and super computing of elliptic operators', in L. Kartashev and S. Kartashev (eds), *Supercomputing*, The International Supercomputing Institute, 1987, pp. 497–502.
2. A. Lin, 'Stable second-order-accurate iterative solutions for second-order elliptic problems', *Int. j. numer. methods fluids*, **9**, 87–102 (1989).
3. J. M. Ortega and R. G. Voight, 'Solution of partial differential equations on vector and parallel computers', *SIAM Rev.* **27**, 149–240 (1985).
4. G. Lee, 'Another combining scheme to reduce hot spot contention in large scale shared memory parallel computers, in E. N. Houstis *et al.* (eds), *Supercomputing, Lecture Notes in Computer Science, Vol. 297*, Springer, 1987, pp. 112–137.
5. P. A. Franaszek and C. J. Georgiou, 'Multipath hierarchy in interconnection networks', *Supercomputing*, in E. N. Houstis *et al.* (eds), *Lecture Notes in Computer Science, Vol. 297*, Springer, 1987, pp. 523–534.
6. P. Rouche, *Computational Fluid Dynamics*, Hermosa, Alburquerque, NM, 1976.
7. J. C. Strikwerda, 'Finite difference methods for the Stokes and Navier Stokes equations, *SIAM J. Sci. Comput.* **5**, 56–68 (1984).
8. S. Agmon, A. Douglis and L. Nirenberg, 'Estimates near the boundary of solutions of elliptic partial differential equations satisfying general boundary conditions, III', *Commun. Pure Appl. Math.*, **17**, 35–92 (1964).

9. A. Brant and N. Dinar, 'Multi-grid solutions to elliptic flow problems', *Proc. Conf. on Numerical Solutions of Partial Differential Equations*, Madison, WI, October 1978.
10. K. Bube and J. C. Strikwerda, 'Interior regularity estimates for elliptic systems of differenced equations', *SIAM J. Numer. Anal.*, **20,** 639–656 (1983).
11. V. Thomee and B. Westergren, 'Elliptic difference equations and interior regularity', *Numer. Math.*, **11,** 196–210 (1968).
12. A. Lin and S. G. Rubin, 'Three dimensional supersonic flow over a cone at incidence', *AIAA J.*, **20,** 1500–1507 (1982).
13. A. R. Gourlay and S. McKee, 'The construction of Hopscotch methods for parabolic and elliptic equations in two space dimensions with mixed derivatives', *J. Comput. Appl. Math.*, **3,** 201–206 (1977).
14. J. Schwartz, 'Ultracomputers', *ACM Trans. Program. Lang. Syst.*, **2,** 484–521 (1980).
15. A. Lin, 'Parallel algorithms for boundary value problems', *Int. J. Parallel and Distributed Computing*, to appear.
16. J. Douglas Jr., 'On the numerical integration of $u_{xx}+u_{yy}=u_t$ by implicit methods', *J. Soc. Indust. Appl. Math.*, **3,** 42–65 (1955).
17. A. Douglis and L. Nirenberg, 'Interior estimates for elliptic systems of partial differential equations',*Commun. Pure Appl. Math.*, **8,** 503–538 (1955).
18. D. W. Peaceman and H. H. Rachford, 'The numerical solution of parabolic and elliptic differential equations', *J. Soc. Indust. Appl. Math.*, **3,** 28–41 (1955).
19. Ye G. D'Yakonov, 'On the application of disintegrating difference operators', *Z. Vycisl. Mat. Mat. Fiz.*, **3,** 385–388 (1963).
20. A. R. Mitchell and D. F. Griffiths, *The Finite Difference Method in Partial Differential Equations*, Wiley, N.Y., 1985.
21. P. J. van der Houwen and J. G. Verwer, 'One-step splitting methods for semi-discrete parabolic equations', *Computing*, **22,** 291–309 (1979).
22. B. P. Sommeijer, P. J. van der Houwer and J. G. Vewer, 'On the treatment of time dependent boundary conditions in splitting methods for parabolic differential equations', *Int. j. numer. methods eng.*, **17,** 335–346 (1981).
23. N. N. Yanenko, *The Method of Fractional Steps*, Springer, 1971.
24. A. Lin and S. G. Rubin, 'Marching with the parabolized Navier Stokes equations', *Israel J. Technol.*, **18,** 61–78 (1981).
25. A. Lin and M. Israeli, 'Iterative numerical solution and boundary conditions for the parabolized Navier Stokes equations', *Int. J. Comput. Fluids*, **13,** 397–409 (1985).
26. S. G. Rubin, 'A review of marching procedures for parabolized Navier–Stokes equations', *Proc. Symp. on Numerical and Physical Aspects of Aerodynamic Flows*, Long Beach, CA, Springer, 1981.
27. A. Lin, 'Parallel algorithms for three dimensional flows', *Numerical Methods in Flows, Vol. 5*, Part 1, Pineridge Press, Swansea, 1987, pp. 48–56.
28. H. B. Keller, *Numerical Methods for Two-point Boundary Value Problems*, Blaisdell, Waltham, MA, 1968.
29. U. Ghia, K. N. Ghia and C. T. Shin, 'High-*Re* solutions for incompressible flow using the Navier–Stokes equations and a multigrid method', *J. Comput. Phys.*, **48,** 59 (1982).
30. A. Lin, 'Fast parallel schemes for computational fluid flows', *Int. Conf. on Numerical Methods in Laminar and Turbulent Flows*, July 1989, invited talk, Swansea, England.