JOURNAL OF COMPUTATIONAL PHYSICS **128,** 223–236 (1996)
ARTICLE NO. 0205

Parallel Multigrid Computation of the Unsteady Incompressible Navier–Stokes Equations

A. T. DEGANI AND G. C. FOX

Northeast Parallel Architectures Center, Syracuse University, Syracuse, New York 13244-4100

Received September 6, 1995

Parallel computation on distributed-memory machines offers the capability of a scalable approach to the solution of large CFD problems. However, in order to fully realize this capability, it is necessary not only to devise parallel methodologies, but also to develop numerical schemes for which the computational effort also scales with problem size. To this end, a parallel multigrid scheme for the calculation of the unsteady incompressible Navier–Stokes equations is considered here. A spatial and temporal second-order accurate implicit discretization scheme on a staggered grid is employed, and a full approximation storage multigrid method, appropriate for nonlinear problems, is used. A parallel solver is developed which smooths the equations at each multigrid level in a fully coupled mode. The programming paradigm is single program multiple data with message passing. In comparison with single-grid calculations, it is demonstrated that the convergence rate for multigrid calculations is considerably superior and dominates the slight degradation in speed up.

1. INTRODUCTION

A variety of fluid-flow phenomena are inherently unsteady and the accurate calculation of flows at high Reynolds number is not only important in engineering design, but also in providing an insight into the fundamental mechanisms at play in these complex flows. However, the computation of unsteady flows at high Reynolds number requires a large number of grid points and sufficiently small time steps to resolve all the important spatial and temporal scales, thus demanding imposing computational power and memory requirements. In the recent past, a consensus seems to have emerged that large-grain distributed-memory machines are most suitable for the execution of such large problems. Many of the solution methodologies that are of interest in CFD may be classified as ''loosely synchronous'' [1] and are thus well-suited for implementation on these machines. If N denotes the problem size and p the number of processors, good speedups and efficiencies are possible for large N/p.

Although it is an important requirement that the parallel implementation scale with p, it is perhaps more important to develop numerical solution schemes that *also* scale with N. A simple illustrative example, where the first requirement is satisfied but not the second, is in the solution of the two-dimensional Poisson equation using the Gauss–Seidel method. With a red–black ordering scheme and a block distribution of data, the parallel implementation of the Gauss–Seidel method is known to be scalable [1]; however, the computational effort required to obtain a converged solution is of $O(N^2 \log N)$ which makes this numerical scheme impractical for large problems. Thus, it is maintained that the two most desirable features of computational algorithms for large problems, typical of those in CFD, are: (i) the computational effort required by the numerical scheme to solve a problem of size N is, say, no more than $O(N \log N)$, and (ii) the parallel implementation of the numerical scheme is scalable with the number of processors p for N/p large and fixed.

Since the early pioneering work by Brandt [2], multigrid methods have been used in a wide variety of problems and a well-designed multigrid method has at most an operation count of $O(N \log N)$ [3]. Most iterative methods are characterized by the fact that they are efficient in eliminating the high-wave number component of the error in the solution, but are poor in removing the low-wave number error. The power of multigrid methods is in their ability to quickly eliminate the low-wave number component of the error through a succession of mesh coarsenings. In most instances, multigrid methods employ relaxation schemes which act upon the local data and are thus well-suited for parallel implementation. But as the multigrid algorithm descends to coarser levels, the parallel efficiency is expected to degrade because the ratio of the time of communication to the time of computation increases; however, since the computational effort at the coarser levels is less than at the finer levels, the overall degradation in efficiency is expected to be minimal for large N/p. In order to establish this assertion, consider first the calculation on the finest grid. A two-dimensional computational domain is assumed here for simplicity although similar results may be obtained for higher dimensions. In a sequential computation, the overall execution time is given by

$$t_{\text{seq}} = N f t_{\text{comp}},$$

0021-9991/96 $18.00

where t_{seq}, N, f, and t_{comp} are the sequential execution time, the total number of points, the floating point operations per point, and the time to execute one floating point operation, respectively. On a distributed-memory machine with p processors and a block-data distribution, the overall execution time t_{par} is given by

$$t_{par} = n f t_{comp} + c\sqrt{n} t_{comm},$$

where $n = N/p$, c is a constant of $O(1)$, and t_{comm} is the time to communicate one byte of data. The processors communicate along the boundary, and, for a block-data distribution, the number of data to be communicated scales with $\sqrt{n}$. The parallel efficiency for the calculations on the single grid η_{single} is then given by

$$\eta_{single}^{-1} = \frac{p t_{par}}{t_{seq}} = 1 + \frac{c\alpha}{f}\frac{1}{\sqrt{n}}, \tag{1}$$

where $\alpha = t_{comm}/t_{comp}$. Next consider the multigrid calculation. In the sequential implementation, this involves obtaining the solution on successively coarser grids of size $N/4$, $N/16$ ... . Assuming N large, the sum of the computational effort may be approximated by an infinite series, in which case the overall execution time of the sequential multigrid scheme is given by

$$t_{seq} = \frac{4}{3} N f t_{comp}.$$

In the parallel implementation, the computational effort and data to be communicated decrease by a factor of $\frac{1}{4}$ and $\frac{1}{2}$, respectively, with each grid coarsening. Once again, assuming an infinite series, the overall parallel execution time is

$$t_{par} = \frac{4}{3} n f t_{comp} + 2c\sqrt{n} t_{comm},$$

and consequently the parallel multigrid efficiency η_{multi} is given by

$$\eta_{multi}^{-1} = 1 + \frac{3c\alpha}{2f}\frac{1}{\sqrt{n}}. \tag{2}$$

Thus, although the multigrid efficiency is lower, the degradation occurs in the higher-order term and only by a factor of 3/2. In three dimensions, it may be easily shown that the degradation factor is 7/6 with $\sqrt{n}$ replaced by $\sqrt[3]{n}$. Although the above development is an over-simplification, it serves to illustrate that the parallel implementation of multigrid schemes on large-grain distributed-memory machines is scalable.

The application of multigrids to the calculation of steady incompressible flow has been successfully demonstrated in a number of studies. Ghia *et al.* [4] employed the strongly implicit relaxation scheme to solve the vorticity and associated Poisson equations in a coupled manner and reported a reduction in the execution time by a factor of 4 by using multigrids. Vanka [5] developed the so-called symmetrical coupled Gauss–Seidel (SCGS) relaxation scheme appropriate for a staggered mesh [6]. A primitive variable formulation is adopted and the equations are solved in a fully coupled mode without forming the intermediate Poisson equation for the pressure. It is reported that this scheme has better stability and smoothing properties than the distributed Gauss–Seidel (DGS) method originally proposed by Brandt [7]. Variants of the SCGS method were also used in later studies [8, 9]. Multigrid calculations based on a staggered curvilinear mesh and employing a decoupled pressure-correction based relaxation scheme have also been carried out [10]. Recently, results on three-dimensional flow calculations in generalized curvilinear coordinates were reported [11].

Here, the parallel implementation of the multigrid method is applied to the time-accurate calculation of the unsteady, incompressible Navier–Stokes equations. As a first step, a regular structured two-dimensional cartesian coordinate system is considered; however, since a primitive variable formulation is adopted, an extension to three-dimensional flow is possible. The multigrid algorithm was first implemented on a sequential machine in Fortran 77 but designed in a fashion to allow the subsequent development of a parallel version using message passing with a minimum number of modifications. This process considerably simplifies the overall development and debugging of the code.

2. NUMERICAL SCHEME

Discretization

Consider the nondimensionalized unsteady incompressible Navier–Stokes equations given by

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u} = \mathbf{g} - \nabla p + \frac{1}{Re}\nabla^2\mathbf{u}, \tag{3}$$

$$\nabla\cdot\mathbf{u} = 0. \tag{4}$$

A spatial and temporal second-order accurate upwind–downwind discretization scheme, originally developed for the calculation of boundary-layer equations [12], is extended here for the computation of the unsteady incompressible Navier–Stokes equations on a staggered grid shown in Fig. 1. A temporal discretization of the momentum equations (3) at the mid-time plane, i.e., at $t + \Delta t/2$, yields

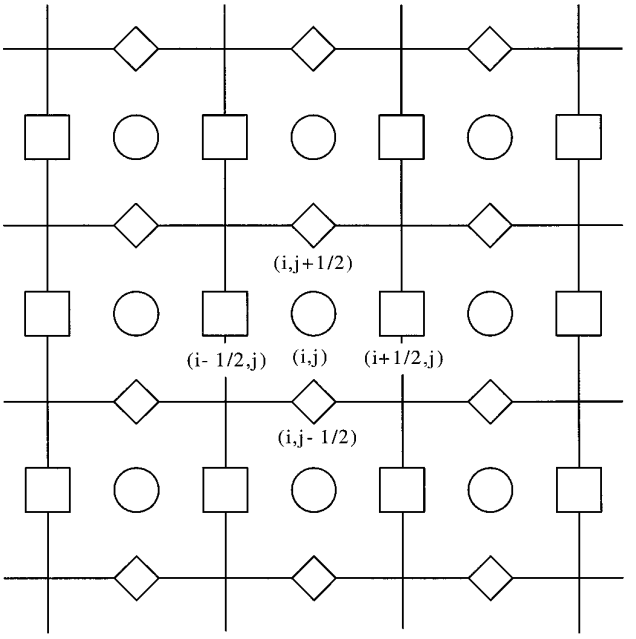

FIG. 1. The staggered mesh showing location of the velocity and pressure: □, u-velocity; ◇, v-velocity; ○, pressure.

$$\frac{u^{n+1}-u^n}{\Delta t} = g_x^{n+1/2} + \overline{Q_x \frac{\partial u}{\partial x}} + \overline{P_x \frac{\partial u}{\partial y}} - \frac{\partial p^{n+1/2}}{\partial x} + \frac{1}{Re}\left[\overline{\frac{\partial^2 u}{\partial x^2}} + \overline{\frac{\partial^2 u}{\partial y^2}}\right], \tag{5}$$

and

$$\frac{v^{n+1}-v^n}{\Delta t} = g_y^{n+1/2} + \overline{Q_y \frac{\partial v}{\partial x}} + \overline{P_y \frac{\partial v}{\partial y}} - \frac{\partial p^{n+1/2}}{\partial y} + \frac{1}{Re}\left[\overline{\frac{\partial^2 v}{\partial x^2}} + \overline{\frac{\partial^2 v}{\partial y^2}}\right], \tag{6}$$

where the overbar denotes the evaluation of the quantities at the mid-time plane, and the superscripts n, $n+1$, and $n+\frac{1}{2}$ indicate the values of the associated variables at times t, $t+\Delta t$, and $t+\Delta t/2$, respectively. The quantities $Q_x = -u$, $Q_y = -u$, $P_x = -v$, and $P_y = -v$ are introduced here to facilitate the subsequent description of the discretization scheme. Note that the pressure is defined only at the mid-time plane.

A uniform computational mesh is assumed and the x- and y-momentum equations are discretized at $(x+\Delta x/2, y)$ and $(x, y+\Delta y/2)$, respectively; these locations are denoted by the subscripts $(i+\frac{1}{2}, j)$ and $(i, j+\frac{1}{2})$ in Fig. 1. To illustrate the upwind–downwind scheme for the convection terms, consider for example, the discretization of $\overline{Q_x \partial u/\partial x}$ on a staggered mesh. It may be confirmed that

$$\overline{Q_x \frac{\partial u}{\partial x}}\bigg|_{i+1/2,j} = \{\overline{Q_x}\}_{i+1/2,j}\left[\frac{1}{2}\left(\frac{u^{n+1}_{i+3/2,j}-u^{n+1}_{i+1/2,j}}{\Delta x} + \frac{u^{n}_{i+1/2,j}-u^{n}_{i-1/2,j}}{\Delta x}\right)\right], \quad \{\overline{Q_x}\}_{i+1/2,j} > 0,$$

$$\overline{Q_x \frac{\partial u}{\partial x}}\bigg|_{i+1/2,j} = \{\overline{Q_x}\}_{i+1/2,j}\left[\frac{1}{2}\left(\frac{u^{n+1}_{i+1/2,j}-u^{n+1}_{i-1/2,j}}{\Delta x} + \frac{u^{n}_{i+3/2,j}-u^{n}_{i+1/2,j}}{\Delta x}\right)\right], \quad \{\overline{Q_x}\}_{i+1/2,j} < 0,$$

provides a spatial and temporal second-order accurate upwind–downwind discretization. The quantity $Q_x = -u$ evaluated at the mid-time plane and at location $(x+\Delta x/2, y)$ is given by

$$\overline{Q_x} = -\mu_t u,$$

where μ denotes the averaging operator. A similar scheme is also used for $\overline{Q_y \partial v/\partial x}$; however, the quantity $Q_y = -u$ is not defined at the location where the y-momentum equation is discretized, i.e., at $(x, y+\Delta y/2)$. In this instance, the required value is obtained from

$$\overline{Q_y} = -\mu_t \mu_x \mu_y u.$$

The other two convection terms are treated in a similar manner and appropriate second-order central differences are employed for the diffusion and pressure-gradient terms. The resulting momentum difference equations are given by

$$\begin{aligned} &M^x_{1+}u^{n+1}_{i+3/2,j} + M^x_{2+}u^{n+1}_{i+1/2,j+1} + M^x_{3+}u^{n+1}_{i+1/2,j} \\ &\quad + M^x_{4+}u^{n+1}_{i+1/2,j-1} + M^x_{5+}u^{n+1}_{i-1/2,j} \\ &\quad + D^x_1 p^{n+1/2}_{i,j} + D^x_2 p^{n+1/2}_{i+1,j} \\ &= M^x_{1-}u^{n}_{i+3/2,j} + M^x_{2-}u^{n}_{i+1/2,j+1} + M^x_{3-}u^{n}_{i+1/2,j} \\ &\quad + M^x_{4-}u^{n}_{i+1/2,j-1} + M^x_{5-}u^{n}_{i-1/2,j} + G_u, \end{aligned} \tag{7}$$

and

$$\begin{aligned} &M^y_{1+}v^{n+1}_{i+1,j+1/2} + M^y_{2+}v^{n+1}_{i,j+3/2} + M^y_{3+}v^{n+1}_{i,j+1/2} \\ &\quad + M^y_{4+}v^{n+1}_{i,j-1/2} + M^y_{5+}v^{n+1}_{i-1,j+1/2} \\ &\quad + D^y_1 p^{n+1/2}_{i,j} + D^y_2 p^{n+1/2}_{i,j+1} \\ &= M^y_{1-}v^{n}_{i+1,j+1/2} + M^y_{2-}v^{n}_{i,j+3/2} + M^y_{3-}v^{n}_{i,j+1/2} \\ &\quad + M^y_{4-}v^{n}_{i,j-1/2} + M^y_{5-}v^{n}_{i-1,j+1/2} + G_v. \end{aligned} \tag{8}$$

The coefficients in (7) and (8) are evaluated at $(i + \frac{1}{2}, j)$ and $(i, j + \frac{1}{2})$, respectively, and are given in the Appendix. Note that for boundary cells, the above difference equations involve quantities that lie outside the computational domain; this issue is taken up later in the discussion of the implementation of the boundary conditions. The discretized momentum equations may be compactly written as

$$M_+^x u + D^x p = M_-^x u^* + G_u = F_u,$$
$$M_+^y v + D^y p = M_-^y v^* + G_v = F_v.$$

Note that (u, v) and p are evaluated at $t + \Delta t$ and $t + \Delta t/2$, respectively, and the superscript "*" denotes the evaluation at the previous time t. The continuity equations (4) is left in its original form, and a second-order discretization at the cell center at time $t + \Delta t$ yields

$$\{D_1^x u_{i-1/2,j}^{n+1} + D_2^x u_{i+1/2,j}^{n+1} + D_1^y v_{i,j-1/2}^{n+1} + D_2^y v_{i,j+1/2}^{n+1}\} = 0, \quad (9)$$

which may be compactly expressed as

$$D^x u + D^y v = G_p = 0.$$

Upon defining

$$\mathbf{q} = \begin{bmatrix} u \\ v \\ p \end{bmatrix}, \qquad \mathbf{G} = \begin{bmatrix} G_u \\ G_v \\ G_p \end{bmatrix}, \qquad \mathbf{F} = \begin{bmatrix} F_u \\ F_v \\ F_p \end{bmatrix},$$

$$\mathbf{M}_+ = \begin{bmatrix} M_+^x & 0 & D^x \\ 0 & M_+^y & D^y \\ D^x & D^y & 0 \end{bmatrix}, \qquad \mathbf{M}_- = \begin{bmatrix} M_-^x & 0 & 0 \\ 0 & M_-^y & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

the difference equations for the unsteady incompressible Navier–Stokes equations may be expressed as

$$\mathbf{M}_+ \mathbf{q} = \mathbf{F} = \mathbf{M}_- \mathbf{q}^* + \mathbf{G}. \quad (10)$$

The implementation of the boundary conditions is taken up next. For simplicity, only Dirichlet boundary conditions on the velocity are considered here in which case no boundary conditions on the pressure are necessary. Although the normal velocity boundary condition can be satisfied exactly, such is not the case for the no-slip condition; however, a second-order accurate discretization of the latter yields

$$\{\mu_y u\}_{i+1/2,1/2}^{n+1} = \frac{u_{i+1/2,0}^{n+1} + u_{i+1/2,1}^{n+1}}{2} = \{u_g\}_{i+1/2,1/2}, \quad (11)$$

where the bottom horizontal wall is considered for illustration and u_g is the specified wall speed. Thus, upon applying the no-slip condition in a similar manner all along the boundary, as many equations are generated as the number of fictitious points required in the discretized momentum equations (10). A general form of the specified boundary conditions may be represented by

$$\mathbf{\Lambda q} = \mathbf{\Phi}, \quad (12)$$

where $\mathbf{\Lambda}$ is an operator and $\mathbf{\Phi}$ is specified. In a typical solution procedure, (10) and (12) are relaxed alternately until convergence.

FAS Multigrid Algorithm

The discretized difference equations and boundary conditions are given by

$$\mathbf{M}_+^k \mathbf{q}^k = \mathbf{F}^k, \qquad \mathbf{\Lambda}^k \mathbf{q}^k = \mathbf{\Phi}^k, \quad (13)$$

where the superscript k denotes discretization on a grid level k. The quantity $\mathbf{F}^k$ on the finest grid $k = M$ is obtained from (10) and is given by

$$\mathbf{F}^k = \mathbf{M}_-^k \mathbf{q}^{*k} + \mathbf{G}^k, \qquad k = M. \quad (14)$$

At the finest grid $k = M$, (13) are relaxed before a subsequent coarsening process outlined next. In the Full Approximation Storage (FAS) method, a coarse-grid approximate solution, which is the sum of the error and the base approximation on the finer grid, is calculated. The resulting equations have the same form as (13), but the right-hand side is now given by

$$\mathbf{F}^k = \mathbf{M}_+^k \mathbf{q}^k + \mathbf{I}_{k+1}^k \mathbf{R}^{k+1} \qquad k < M,$$
$$\mathbf{\Phi}^k = \mathbf{\Lambda}^k \mathbf{q}^k + \mathbf{I}_{k+1}^k \mathbf{\Pi}^{k+1}, \qquad k < M, \quad (15)$$

where $\mathbf{R}$ and $\mathbf{\Pi}$ denote the residuals in the interior and boundary equations, respectively, and $\mathbf{I}_{k+1}^k$ is the restriction operator. The residuals are given by

$$\mathbf{R}^{k+1} = \mathbf{F}^{k+1} - \mathbf{M}_+^{k+1} \mathbf{q}^{k+1},$$
$$\mathbf{\Pi}^{k+1} = \mathbf{\Phi}^{k+1} - \mathbf{\Lambda}^{k+1} \mathbf{q}^{k+1}, \quad (16)$$

and $\mathbf{q}^k$ is obtained from

$$\mathbf{q}^k = \mathbf{I}_{k+1}^k \mathbf{q}^{k+1}. \quad (17)$$

Inspection of the coefficients in (10) in the Appendix indi-

cates that both (u^k, v^k) and (u^{*k}, v^{*k}) are required in order to evaluate the nonlinear operator $\mathbf{M}_+^k$; the former is obtained from (17) and the latter is obtained by storing the velocities at the previous time step at all levels. The resulting equations, with an initial estimate of $\mathbf{q}^k$ given by (17), are relaxed once again and the coarsening process continues until the grid $k = 1$ is reached. In the backward sweep, it is the error in the solution, not the solution itself, that is projected from the coarser grid to the finer grid; thus, the corrected finer grid k solution is calculated by

$$\mathbf{q}^k \leftarrow \mathbf{q}^k + \mathbf{I}_{k-1}^k\{\mathbf{q}^{k-1} - \mathbf{I}_k^{k-1}\mathbf{q}^k\}, \qquad k \neq 1, \tag{18}$$

where $\mathbf{I}_{k-1}^k$ denotes the projection operator. The equations in (13) are relaxed before the next correction step and this process continues until the finest grid $k = M$ is reached, thus completing one "V" cycle iteration of the FAS multigrid algorithm. A V(2, 1) cycling scheme is adopted here, i.e., the equations (13) are relaxed twice and once in the forward and backward sweeps, respectively.

On the staggered mesh considered here, the Dirichlet conditions on the normal velocity can be satisfied exactly and thus the boundary residual at any level is identically zero. For this subset of the boundary conditions, an explicit use of the second of (15) is not necessary since the normal velocity at the coarser grid is obtained from (17). In effect, the Dirichlet conditions on a grid $k < M$ are obtained by restricting the specified values of the normal velocity from the finer grid. A similar approach may also be used for the implementation of the no-slip condition by replacing the appropriate subset of $\mathbf{\Phi}$ in the second equation of (15) by the restricted values of the wall speed, i.e.,

$$\mathbf{\Phi}_\tau^k = \mathbf{I}_{k+1}^k \mathbf{\Phi}_\tau^{k+1},$$

where $\mathbf{\Phi}_\tau$ is the subset of $\mathbf{\Phi}$ that specifies the wall speed. However, this simpler approach is inferior since it does not account for a non-zero boundary residual in the no-slip condition on the finer grid. Indeed, numerical experiments indicate a marked increase in the number of iterations required for convergence when this simpler approach is adopted.

In general, the restriction operators for the residuals and the dependent variables $\mathbf{q}$ may be different; however, in this study, they are assumed to be identical. Due to the nature of the staggered grid, different restriction formulae are necessary for the individual components of the residual $\mathbf{R} = (R_u, R_v, R_p)$ and $\mathbf{q}$. For the pressure p and continuity residual R_p, the restriction operator is chosen to be a simple average of the four adjacent values, viz.

$$\{I_p\}_{k+1}^k = \mu_x^{k+1}\mu_y^{k+1}. \tag{19}$$

There are two commonly used formulae for restricting u and R_u in the interior of the computational domain, viz.

$$\{I_u\}_{k+1}^k = \mu_y^{k+1}, \tag{20}$$

$$\{I_u\}_{k+1}^k = \mu_x^{k+1}\mu_x^{k+1}\mu_y^{k+1}. \tag{21}$$

Equations (20) and (21) result in 2-point and 6-point formulae, respectively; the former has been used in [5] and the latter in [9, 10]. The relative merit of the two alternatives is discussed further in Section 3. The velocity u also needs to be restricted along the boundary Γ and a 2-point formula is appropriate for sections of Γ where u is the normal velocity. From the second of (15), it follows that for the implementation of the no-slip condition considered here, both the boundary residual and fictitious velocity need to be restricted. Thus, along the sections of Γ where u is the tangential velocity, the standard 3-point average is used to restrict the appropriate values. The restriction formulae for the velocity v and y-momentum residual R_v is obtained by a simple 90° rotation of the formulae for u and R_u.

Bilinear interpolation is commonly used to obtain the correction to $\mathbf{q}$ [5, 7, 10]. Although the projected corrections are easily obtained for all the interior cells, some modification to the formulae is necessary in the boundary cells due to the nature of the staggered grid. To address this, a number of options were tried: (i) using the bilinear interpolation stencil to extrapolate the appropriate values in the boundary cells, (ii) assuming a zero normal pressure gradient at the boundary (following the procedure in [5, 10]), (iii) using the values of the fictitious velocity to extend the range of interpolation to include boundary cells, and (iv) 6-point interpolation formulae. However, no consistent and significant differences were observed in the convergence rate of the multigrid algorithm and eventually the simplest option (i) was adopted.

Smoother

The symmetrical-coupled Gauss–Seidel (SCGS) solver [5] is employed here to smooth the system of difference equations (10) in a fully coupled mode. In this scheme, the four velocities at the faces of each cell and the pressure, defined at the cell center, are updated simultaneously in an iterative process that traverses all the cells in the computational domain usually in a lexicographic order. This is accomplished by inverting a 5×5 (for two-dimensional flow) matrix. In this manner, the velocity is updated twice and the pressure once in each sweep. Furthermore, as in the simple Gauss–Seidel method for scalar equations, the most recently updated values are used.

The SCGS method is briefly summarized here, but the details may be found in [5]. Assume that the iterative process is at cell (i, j) in its sweep through the computational domain and denote δu, δv, and δp as the difference

in the updated and old values of u, v, and p, respectively. If the x-momentum equation (7) at $(x + \Delta x/2, y)$ is used to update only $u_{i+1/2,j}$ and $p_{i,j}$, then it follows that in terms of δu and δp,

$$\{M^x_{3+}\}_{i+1/2,j}\delta u^{n+1}_{i+1/2,j} + \{D^x_1\}_{i+1/2,j}\delta p^{n+1/2}_{i,j} = \{R_u\}_{i+1/2,j},$$

where $\{R_u\}_{i+1/2,j}$ is the residual calculated prior to the update and is the difference between the right- and left-hand sides of (7). Similarly, the x-momentum equation at location $(x - \Delta x/2, y)$ is written as

$$\{M^x_{3+}\}_{i-1/2,j}\delta u^{n+1}_{i-1/2,j} + \{D^x_2\}_{i-1/2,j}\delta p^{n+1/2}_{i,j} = \{R_u\}_{i-1/2,j}.$$

Expressing the y-momentum equation (8) at the top and bottom faces of the cell and the continuity equation (9) in a similar manner results in a sparse system of five equations in five unknowns which is inverted analytically and the resulting formulae are used to obtain the changes in the four velocities and pressure.

In the computation of steady incompressible flow, it was found [5] that the SCGS scheme is either slowly convergent or divergent when applied to calculate high Reynolds number flows on fine grids. In this instance, it becomes necessary to provide damping to stabilize the SCGS scheme. This is accomplished by multiplying the diagonal terms of the 5×5 system of equations by constants which are greater than unity. In this study, which considers unsteady flow, it was not found necessary to include damping factors in the calculation of the results presented here. This may be attributed to the appearance of Δt^{-1} in the terms along the diagonal of the 5×5 system of equations which enhances the diagonal dominance of the interative scheme. Although the present methodology is not tailored toward calculating steady flow, it is possible to achieve this objective by considering large time steps. In this instance, the undamped SCGS scheme was found to become unstable for high Reynolds number and fine grids.

3. PARALLEL IMPLEMENTATION

Prior to discussing the specifics of a parallel implementation of the multigrid calculation of the unsteady incompressible Navier–Stokes equations, some general comments are made first on the development and structure of the parallel code. The multigrid method was first developed on a sequential machine in Fortran 77. For efficient memory utilization, the two-dimensional arrays at all levels are mapped onto a single one-dimensional array and a table is generated containing pointers to the first element at each level. This is convenient in Fortran 77 since arrays are passed to functions based on assumed size. Anticipating the subsequent implementation of a parallel version, the sequential code was designed to allow a smooth transition to the development of a single program multiple data (SPMD) code in Fortran 77 with message passing. The SPMD code, which uses synchronous collective communication, was developed by interspersing calls to routines in an interface layer at appropriate locations in the sequential code; the original Fortran 77 statements required very few changes. The routines in the interface layer between the main code and the message-passing library, compose and decompose messages that are sent and received, respectively, make calls to the appropriate routines in the message-passing library and perform peripheral book-keeping tasks. Thus, when porting to other systems, only the interface layer, and not the main code, needs to be modified which considerably simplifies the transition. Portability was a main design consideration in constructing the parallel code.

The data are distributed in blocks on a two-dimensional mesh of abstract processors and the local data in each processor are mapped onto a local 1D array. The second-order accurate discretized equations (7) and (8) and the projection and restriction operators considered here indicate that each processor requires data from the boundary cells of the adjacent processors. Consequently, the two-dimensional array of cells in each processor is augmented by a buffered boundary of one cell thickness for all the multigrid levels. Appropriate flags are set in each processor to indicate whether it lies adjacent to the boundary or in the interior of the computational domain. By choosing the number of cells in the finest grid such that each processor has an equal number of cells, the issue of load balancing is not considered here.

The parallel code was developed on the 32-node CM-5 installed at the Northeast Parallel Architectures Center using the Connection Machine communication library CMMD. Since the code is written in Fortran 77, the four vector units (VUs) on each node may be accessed only by writing non-portable assembly language code [13]. The CMMD library also provides virtual channels and an active message interface which considerably reduce the message latency. However, in keeping with the main design consideration of constructing portable code, none of the above functionalities were incorporated.

The details of the parallel implementation of the multigrid method are now taken up. Consider first the idle-processor problem which occurs at coarse multigrid levels where the total number of cells is less than the number of processors. To simplify the subsequent discussion, let the finest grid contain N cells which are arranged in a two-dimensional array $\sqrt{N} \times \sqrt{N}$ on a processor mesh of $\sqrt{p} \times \sqrt{p}$; furthermore, let $\sqrt{N} = 2^\eta$ and $\sqrt{p} = 2^\sigma$. It is assumed that the coarest grid consists of an array of 2×2 cells; thus the total number of multigrid levels is η. In a

sequential multigrid, the total computational time for η levels is given by $t_{seq} = 4/3\, ft_{comp}(N-1)$, where an infinite series approximation has not been used. In a parallel implementation, each processor is assigned a grid of $\sqrt{n} \times \sqrt{n}$ cells at the finest level, where $n = N/p$. The multigrid calculations may be continued until each processor contains 2×2 cells, which occurs at level $\sigma + 1$. The computational time required for the first $\eta - \sigma$ finest grids is given by $t^f_{par} = 4/3 ft_{comp}\,(n-1)$. The simplest approach for all levels at and coarser than σ is to have each processor solve the same problem. In this case, subsequent to an all-to-all broadcast, each processor contrains $\sqrt{p} \times \sqrt{p}$ cells at level σ. The computational time required to solve this problem is given by $t^c_{par} = 4/3 ft_{comp}(p-1)$. Thus, for this approach, the ideal parallel efficiency (i.e., discounting all communication overhead) may be shown to be given by

$$\eta^{-1}_{ideal} = 1 + \frac{(p-1)^2}{np-1}. \tag{22}$$

The two limiting cases of interest here are:

$$\text{(a)}\ n \gg p \quad \eta^{-1}_{ideal} \sim 1 + O\left(\frac{1}{n}\right),$$

$$\text{(b)}\ n,p \gg 1 \quad \eta^{-1}_{ideal} \sim 1 + O\left(\frac{p}{n}\right).$$

Thus this approach is appropriate for the solution of large problems on a modest number of processors such as a network of workstations. An added advantage of this approach is that besides the all-to-all broadcast, no communication is required for the coarse-grid levels which is an important consideration in this situation where typically the communication latencies are high. On the other hand, the degradation in efficiency is unacceptable when $p \sim O(n)$ and thus this approach is not scalable.

An alternative approach is adopted here for multigrid levels at and coarser than σ which is appropriate for cases in which both n and p are large. Let the coordinates of an abstract processor in the two-dimensional mesh be denoted by (ip, jp) where $0 \le ip, jp \le \sqrt{p} - 1$. At level σ, the four processors identified by the coordinates (ip, jp), $(ip+1, jp)$, $(ip, jp+1)$, $(ip+1, jp+1)$, where $ip, jp \bmod 2 = 0$, coalesce their data onto a 2×2 cell grid and each of these processors solves the same problem; in effect, only one-fourth of the processors are active. The ''adjacent'' neighbors are now a stride of 2 away in each of the coordinate directions. Similarly, at the next coarser level, a group of 16 processors solve the same problem and so on. This may be summarized:

for $\{1 \le k \le \sigma\}$,

$$\text{stride} = 2^{\sigma+1-k}$$

$$\text{coalesce } (ip+i, jp+j) \begin{cases} (ip, jp) \bmod \text{stride} = 0 \\ 0 \le i, j \le \text{stride} - 1. \end{cases}$$

In this manner, the computational effort for each processor is the same for all grid levels at and coarser than σ. Thus, the overall computational time for the coarsest σ levels is given by $t^c_{par} = 4\sigma ft_{comp}$. Noting that $\sigma = 1/2 \log p$, the ideal parallel efficiency may be shown to be given by

$$\eta^{-1}_{ideal} = 1 + \frac{p[\frac{3}{2}\log p - 1] + 1}{np - 1}. \tag{23}$$

Considering the two limiting cases once again, we get

$$\text{(a)}\ n \gg p \quad \eta^{-1}_{ideal} \sim 1 + O\left(\frac{1}{n}\right),$$

$$\text{(b)}\ n,p \gg 1 \quad \eta^{-1}_{ideal} \sim 1 + O\left(\frac{\log p}{n}\right).$$

Thus, the degradation in the ideal parallel efficiency is reduced to an acceptable level in comparison with the previous method for the case where both n and p are large. Although the order of degradation in the ideal efficiency is the same for both approaches considered here for the case where $n \gg p$, the former is preferable especially in a high-latency environment such as a network of workstations.

The parallel implementation of the smoother is discussed next. The overriding theme here is to parallelize the symmetrical coupled Gauss–Seidel scheme [5] with minimal modifications to the original algorithm. With this objective in mind, a parallel version, PAR-SCGS, is developed which is appropriate for distributed-memory machines using message passing. The red–black ordering scheme, commonly employed to solve the discretized second-order Laplace equation with a Gauss–Seidel smoother, is also the basis for PAR-SCGS; however, a two-color scheme is adopted here for a different reason and this is elaborated upon subsequently. Consider Fig. 2a which shows a two-dimensional mesh of cells in a processor along with the augmented buffer boundary of one cell thickness. The processor boundary is indicated by a thick line and we assume that, through prior communication, all data values in the buffered boundary have been obtained from the adjacent processors. The red and black cells may be identified by $(i+j) \bmod 2 = 0$ and $(i+j) \bmod 2 = 1$, respectively, where (i, j) denotes the location of a cell in the global mesh; furthermore, assume that the bottom left cell within the processor is red. In step 1, all the red cells are relaxed. Note that for the cells along the interior boundary of the

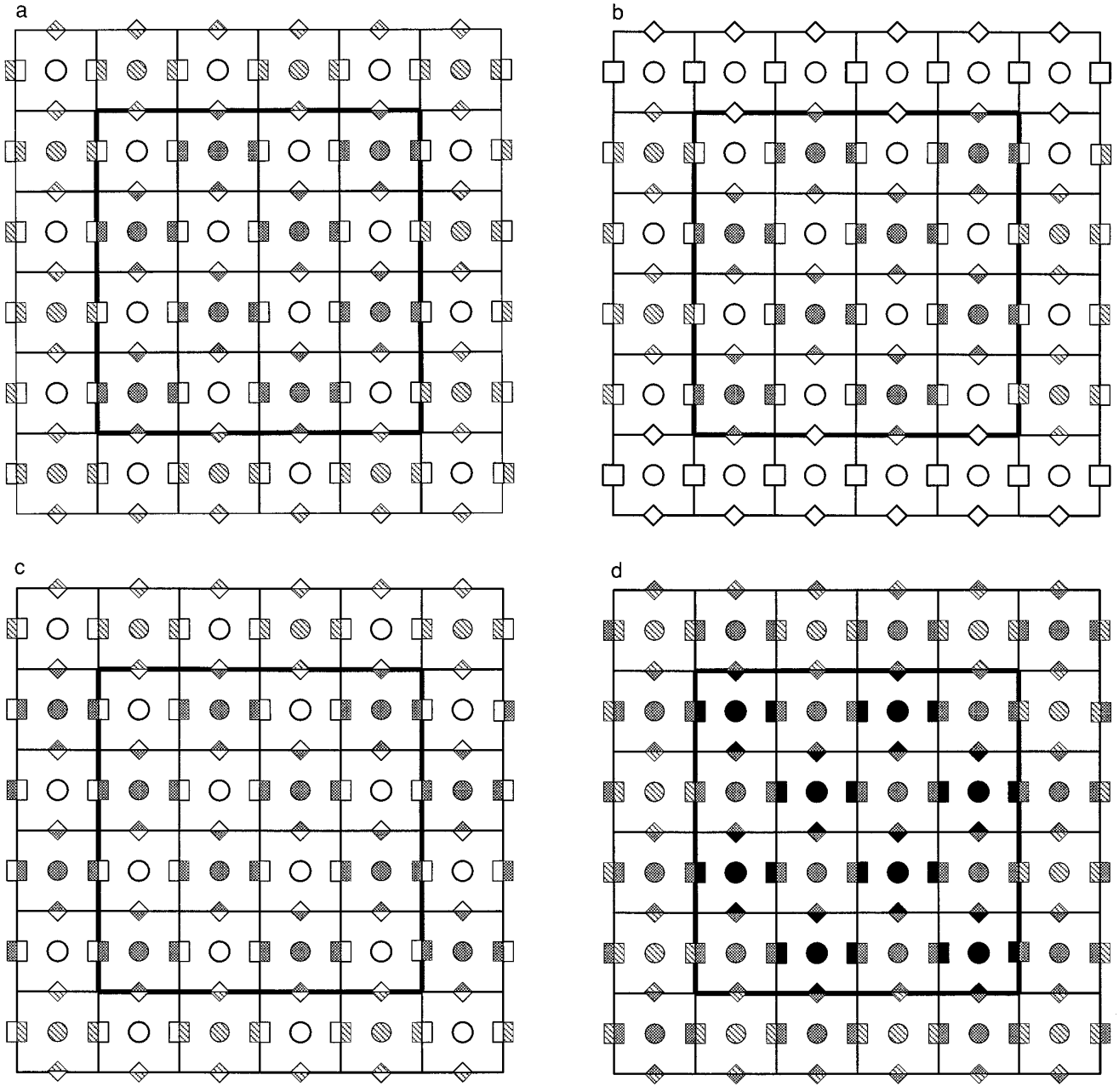

FIG. 2. Stages in the execution of Parallel SCGS. (a) Step 1: Relaxation of "red" cells (⊠). Cells relaxed in the neighboring processors are identified by a cross-hatch. (b) The values at the end of Step 1 are shown in red (⊠). In Step 2(a), the red hatched values are obtained from the left and right neighboring processors. (c) The values at the end of Step 2(a) are shown in red. In Step 2(b), the red hatched values are obtained from the bottom and top neighboring processors. (d) At the end of Step 2(b), all red values are known. In step 3, "black" cells (■) are relaxed and the cells in the neighboring processors are identified by cross-hatching. (e) The values at the end of Step 3 are shown in black. In Step 4(a), the black hatched values are obtained from the left and right neighboring processors. (f) The values at the end of Step 4(a) are shown in black. In step 4(b), the black hatched values are obtained from the bottom and top neighboring processors.

processor, (7) and (8) indicate that data in the buffered boundary are required. Within the processor, the updated values are shown in "red" (shaded). In one sweep of the SCGS scheme, the pressure is updated once and the velocity twice. Thus only one-half of the velocity symbols are shaded red to indicate that the velocity has been updated once by the relaxation of the red cells. Simultaneously, the red cells are also relaxed in the neighboring processors, but the updated values are not yet available and are distinguished by a cross-hatch. The relaxation of

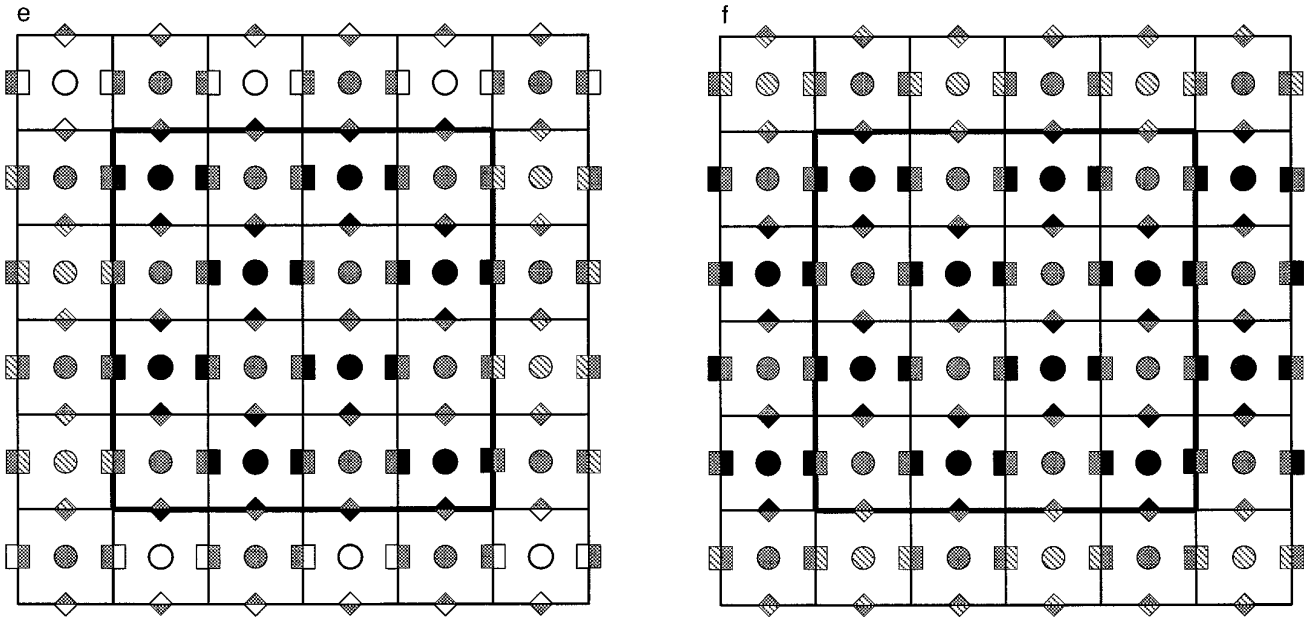

FIG. 2—*Continued*

the red cells is followed by Step 2 which involves two stages of communication with neighboring processors. In the first stage, Step 2(a), shown in Fig. 2b, each processor sends the updated data on the left interior boundary to the left processor and receives the cross-hatched data in the buffered right boundary from the right processor using a synchronous send-and-receive. In a similar manner, the cross-hatched data on the left boundary is received from the left processor. Thus after two send-and-receives, each processor has the updated data shown in Fig. 2c. The second stage in the communication is similar to the first, but now each processor communicates with its top and bottom

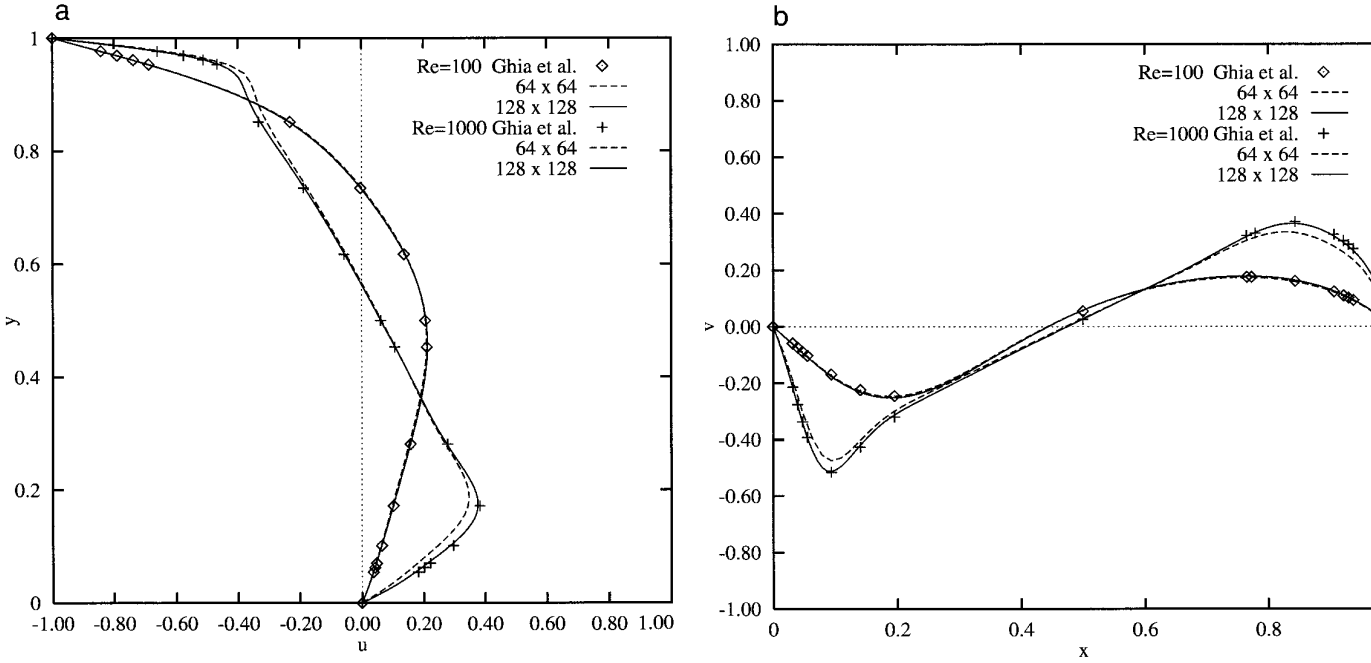

FIG. 3. Steady state solution in a square cavity with the top wall moving at unit speed from right to left, $Re = 100, 1000$: (a) the horizontal velocity along the vertical centerline, (b) the vertical velocity along the horizontal centerline.

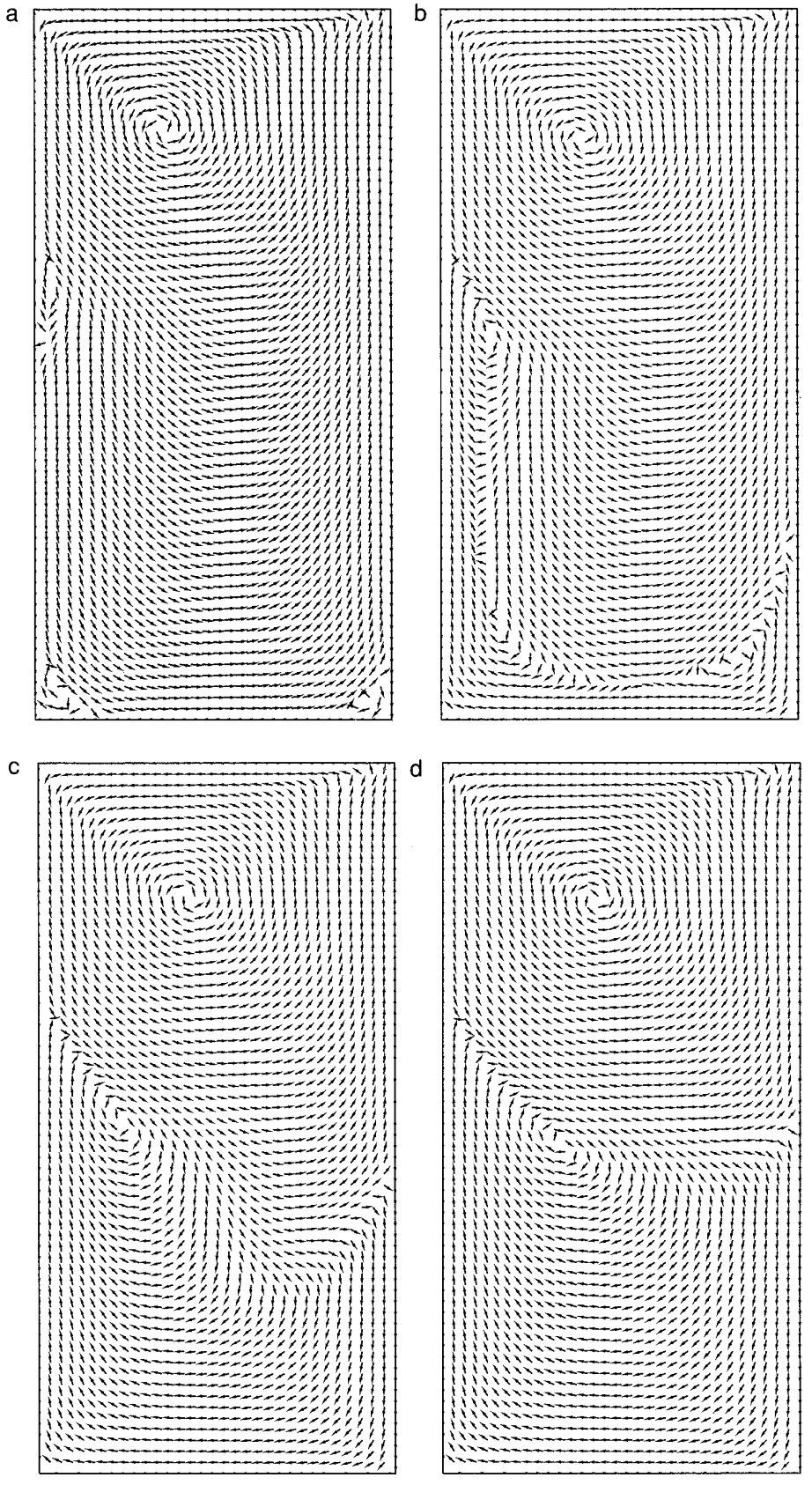

FIG. 4. Normalized velocity vector plots in a cavity of aspect ratio 2 with the top wall moving at unit speed from right to left, $Re = 400$: (a) $t = 6$, (b) $t = 8$, (c) $t = 10$, (d) $t = 12$.

neighbor. Note, however, that some data obtained from both the left and right processors in Step 2(a) are also communicated. In this manner, as indicated in Fig. 2d, data may be obtained from the diagonal neighbors without any explicit communication with them. Step 3 involves the relaxation of the black cells, and once again, the cells relaxed in the neighboring processors but whose updated values are not yet available are indicated by a cross-hatch. In Step 4, the black data values are communicated in an analogous manner to that in step 2 and at its conclusion all the data, including those in the buffered boundary, are updated. This sets the stage for the next iteration at the start of Step 1.

Consider now the differing roles of the two-color schemes in the solution of the second-order accurate Laplace equation and the PAR-SCGS algorithm. If a one-color scheme is adopted for the Laplace equation, a Jacobi-type iteration will result at the processor boundary, a limitation that is easily overcome by a two-color scheme. With a red–black point ordering scheme, the update of the potential at a point depends only on the values of the potential at adjacent points of the other color, and the smoothing rate remains independent of the number of processors. If a one-color scheme is used for the SCGS smoother, it may be confirmed that the values of the velocity on the processor boundary will be different in the neighboring processors at the end of the iteration. Thus, in the PAR-SCGS method described here, a *two*-color scheme is used because the velocity is updated *twice* in each sweep. However, this does not prevent a Jacobi-type iteration at the processor boundary since red (black) values also depend on the red (black) values, and consequently, a degradation in the smoothing rate of PAR-SCGS may be expected as the number of processors p increases for fixed N. However, as is demonstrated later, this degradation in the smoothing rate for the multigrid computation of the Navier–Stokes equations is very small.

It may be noted that an alternative strategy that overcomes the potential difficulty with a Jacobi-type iteration at the processor boundary is to increase the number of colors. Although the total number of data to be communicated in one iteration will remain approximately the same as that in the two-color scheme, smaller messages will be sent more frequently. This will result in an increase in message latency, rendering this alternative inappropriate in light of the fact that the degradation in smoothing rates for PAR-SCGS described above is minimal. A similar argument may also be made against the distributed Gauss–Seidel (DGS) [7] method in which each equation is smoothed individually, thus increasing message latency.

Communication is also required in the intergrid transfer of data. Consider first the restriction of $\mathbf{q}$ and the residual $\mathbf{R}$ in the forward sweep of the V-cycle. As indicated earlier, there are two options commonly used to restrict u and the x-momentum residual R_u. It may be deduced that (20) introduces no communication, whereas (21) does. For the test problems considered here, it was empirically determined that the convergence rate of the multigrid method is insensitive to the choice of restriction operators. Therefore, it would appear that (20) is clearly preferable in the parallel multigrid method. However, (21), which describes a ''full-weighting'' restriction operator, is generally considered to be more robust especially for nonlinear problems and near boundaries [7]. Therefore, more extensive testing is necessary to determine the relative influence of (20) and (21) on the convergence rate, and thus to assess whether the increase (if any) in convergence rate by using the 6-point formula justifies the added overhead in communication. In the meantime, the present implementation uses the 6-point formula and all results have been obtained with this restriction operator; note that by using the alternative 2-point formula, a reduction in communication time by approximately a factor of 30% may be achieved. In the backward sweep, communication is required in order to obtain the value of the error in the buffered boundary at the coarser level prior to prolongation to the finer level. In all cases where data from the diagonally located processors are required, no explicit message passing with these processors is necessary since the data may be obtained indirectly as demonstrated by the implementation of communication in PAR-SCGS. In implementing the approach defined by (23) at the coarser levels, a group of four processors coalesce their data by two successive pairs of data swaps. The only global communication that is required is in testing convergence at the end of each V-cycle.

4. RESULTS AND DISCUSSION

In order to test the parallel multigrid scheme developed here, the classic problem of steady flow in a square cavity is considered first. Although a time-accurate computation is not an efficient way to obtain steady-state solutions, these calculations have been performed here to enable a quantitative comparison with well-established data in the literature. The sides of the cavity are normalized to unity and the top wall is assumed to move from right to left at unit speed. Two Reynolds numbers are considered, viz. $\mathrm{Re} = 100, 1000$, and the time step is chosen to be uniform and equal to $\Delta t = 0.1$. Fig. 3a shows the horizontal velocity along the vertical centerline of the cavity, and Fig. 3b shows the vertical velocity along the horizontal centerline. The computed data by Ghia *et al.* [4] were obtained on a grid of 128×128 using the vorticity-stream-function formulation.

The next test case considers the unsteady flow in a rectangular cavity where the ratio of the vertical to horizontal sides is 2. The top wall is set into impulsive motion at time $t = 0$ and moves from right to left at unit speed. The Reynolds number is $\mathrm{Re} = 400$ and the time step is $\Delta t =$

TABLE I

Residual Tolerance Error for Various Grid Sizes

	Global grid			
	32×32	64×64	128×128	256×256
Tol	1.0×10^{-3}	2.5×10^{-4}	6.0×10^{-5}	1.5×10^{-5}

0.02. The grid size is 64×128, but for clarity only every other point is shown. The instantaneous flow patterns at four selected times are shown in Fig. 4 in the same format as that used by Gustafson and Halasi [14] and show good agreement with their computed results. A quantitative comparison is not possible since the appropriate data were not tabulated in the work cited above.

Next the effectiveness of applying multigrid methods to the solution of the unsteady incompressible Navier–Stokes equations is considered. Table I shows the residual tolerance error that is specified for various fine grids. It is appropriate to choose the tolerance error to be of the same order of magnitude as the discretization error which for a second-order accurate scheme is of the form Kh^2 where h denotes the mesh spacing and $K \sim O(1)$. Choosing $K = 1$, the residual tolerance error is then approximately set to h^{-2}. Convergence is deemed to have occurred at each time step when the residuals of the continuity and momentum equations are *all* less than the residual tolerance. In the context of multigrid methods, it is convenient to quantify the computational effort in terms of work units (WU) [2]. Let the effort in performing one iteration on the finest grid be 1 WU. In a V-cycle, denoted by $V(\nu_1, \nu_2)$, the number of iterations in the forward and backward sweeps at each level besides the coarest are given by ν_1 and ν_2. At the coarsest level, only the forward sweep is performed and thus the number of iterations at this level is ν_1. For a two-dimensional domain, the computational effort reduces by a factor of 4 for each coarsening of the mesh. Therefore, the number of WU's required for one V-cycle with M levels of multigrids is given by

$$\mathrm{WU}_1 = \frac{4}{3}(\nu_1 + \nu_2)\{1 - 2^{-2(M-1)}\} + \nu_1 2^{-2(M-1)}, \quad (24)$$

where the subscript on WU denotes the number of iterations of the V-cycle. If the number of iterations of the V-cycle to obtain a converged solution at a time step is denoted by κ, then the computational effort in terms of work units is given by $\mathrm{WU}_\kappa = \kappa \mathrm{WU}_1$. Note that in this study it is assumed that $\nu_1 = 2$ and $\nu_2 = 1$.

Consider the sequential implementation on a single processor where the cells are traversed in a lexicographic order during the smoothing operation of SCGS. Table II shows the variation of WU_κ per time step with resolution of the finest grid and number of multigrid levels; the results in Table II are obtained by averaging WU_κ over the first 10 time steps. The flow configuration is a square cavity in which the top wall is set into impulsive motion with unit speed at $t = 0$ and the Reynolds number and time step are $\mathrm{Re} = 1000$ and $\Delta t = 0.01$. There is a substantial reduction in computational effort as the number of multigrid levels is increased and this is more pronounced as the number of mesh points in the finest grid is increased. With the same flow conditions and parameters, the computational effort for the parallel multigrid calculation is also obtained. In this case, PARSCGS is used as the smoother over a processor mesh of 8×4 and the implementation defined by (23) is used for the coarser levels. Table III shows WU_κ (summed over all the processors) and the resuults indicate trends similar to those observed for the sequential implementation. It may be noted by comparing the results in Tables II and III that for a grid size and multigrid levels, the number of work units required by the parallel implementation is generally higher.

TABLE II

Number of Work Units WU_κ on a Sequential Machine ($\mathrm{Re} = 1000$, $\Delta t = 0.01$)

	Global grid			
Levels	32×32	64×64	128×128	256×256
1	60	—	—	—
2	24	350	—	—
3	10	99	825	—
4	8	27	220	—
5	8	13	58	315
6	—	13	17	82
7	—	—	17	22
8	—	—	—	21

TABLE III

Number of Work Units WU_κ on a Processor Mesh of 8×4 ($\mathrm{Re} = 1000$, $\Delta t = 0.01$)

	Global grid			
Levels	32×32	64×64	128×128	256×256
1	86	—	—	—
2	30	452	—	—
3	10	119	966	—
4	6	29	243	—
5	—	15	59	347
6	—	—	23	82
7	—	—	—	25

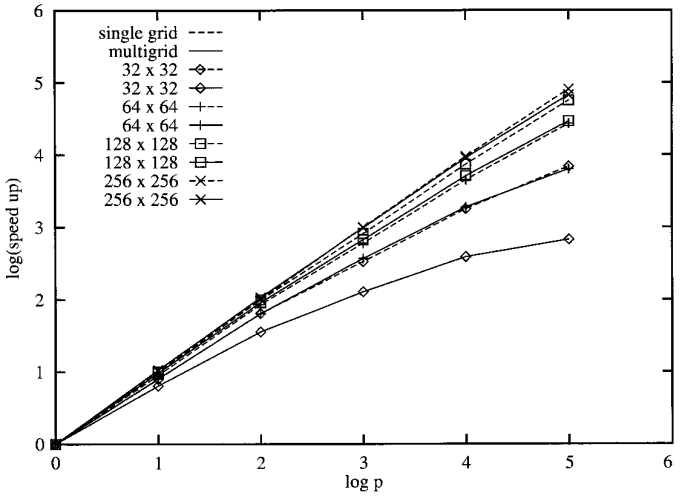

FIG. 5. Variation of speed up with number of processors for single-grid and multigrid calculations.

This is a direct result of the fact that the smoothing rate of PAR-SCGS is inferior to SCGS since the former utilizes a Jacobi-type iteration at the processor boundary; however, this degradation is minimal when the full complement of multigrid levels is utilized.

Finally, the speed up for both single-grid and multigrid calculations is shown in Fig. 5 where the number of processors is varied from 1 to 32 in steps of 2. The speedup is obtained from t_1/t_p where t_p and t_1 denote the measured execution times for calculations with p processors and a single processor, respectively. For the multigrid calculations, the maximum number of levels possible is used for a given fine grid and the approach defined by (23) is implemented at the coarser levels. Three expected trends may be observed: (i) the degradation in speedup for both single-grid and multigrid calculations with increasing number of processors is more pronounced for smaller-sized problems, (ii) the speedup of multigrid is always less than single-grid calculations, and, most importantly, (iii) the difference in speedup between single-grid and multigrid calculations diminishes with increasing problem size.

5. CONCLUSIONS

A parallel multigrid scheme for the time-accurate calculation of the unsteady incompressible Navier–Stokes equations in primitive variables has been investigated. The convergence rate of the multigrid method using the parallel smoother, PAR-SCGS, developed here is comparable to that of a sequential algorithm. It is shown that good speedups are attainable for multigrid calculations as the problem size increases. The reduction in the computational effort by incorporating multigrids dominates the degradation in speedup when compared with single-grid calculations.

APPENDIX

The coefficients in the momentum equations (7) and (8) and the continuity equation (9) are given by

$$
\begin{aligned}
M^z_{1+} &= -\frac{a_1}{\mathrm{Re}} + \overline{Q}_z a_4,\\
M^z_{2+} &= -\frac{a_3}{\mathrm{Re}} + \overline{P}_z a_7,\\
M^z_{3+} &= -\frac{1}{\Delta t} + \frac{(a_1 + a_3)}{\mathrm{Re}} + \overline{Q}_z a_5 + \overline{P}_z a_8,\\
M^z_{4+} &= -\frac{a_3}{\mathrm{Re}} + \overline{P}_z a_9,\\
M^z_{5+} &= -\frac{a_1}{\mathrm{Re}} + \overline{Q}_z a_6,
\end{aligned}
$$

$$
\begin{aligned}
M^z_{1-} &= \frac{a_1}{\mathrm{Re}} + \overline{Q}_z a_6,\\
M^z_{2-} &= \frac{a_3}{\mathrm{Re}} + \overline{P}_z a_9,\\
M^z_{3-} &= \frac{1}{\Delta t} - \frac{(a_1 + a_3)}{\mathrm{Re}} + \overline{Q}_z a_5 + \overline{P}_z a_8,\\
M^z_{4-} &= \frac{a_3}{\mathrm{Re}} + \overline{P}_z a_7,\\
M^z_{5-} &= \frac{a_1}{\mathrm{Re}} + \overline{Q}_z a_4,
\end{aligned}
$$

$$
\begin{aligned}
D^z_1 &= -\Delta z^{-1},\\
D^z_2 &= \Delta z^{-1},
\end{aligned}
$$

$$
\begin{aligned}
G_u &= g_x^{n+1/2},\\
G_v &= g_y^{n+1/2},
\end{aligned}
$$

where $z = x,y$ for the x- and y-momentum equations, respectively. The coefficients a_i are given by

$$
\begin{aligned}
a_1 &= \frac{1}{2\Delta x^2},\\
a_3 &= \frac{1}{2\Delta y^2},\\
a_4 &= -\frac{A_z}{2\Delta x},\\
a_5 &= \frac{(A_z - 1/2)}{\Delta x},\\
a_6 &= \frac{1 - A_z}{2\Delta x},\\
a_7 &= -\frac{B_z}{2\Delta y},\\
a_8 &= \frac{(B_z - 1/2)}{\Delta y},\\
a_9 &= \frac{1 - B_z}{2\Delta y},
\end{aligned}
$$

and

$$A_z = \frac{1 + \mathrm{sgn}(\overline{Q_z})}{2},$$
$$B_z = \frac{1 + \mathrm{sgn}(\overline{P_z})}{2}.$$

Finally, note that

$$\overline{Q_x} = -\mu_t u,$$
$$\overline{Q_y} = -\mu_t \mu_x \mu_y u,$$
$$\overline{P_x} = -\mu_t \mu_x \mu_y v,$$
$$\overline{P_y} = -\mu_t v.$$

ACKNOWLEDGMENT

The authors gratefully acknowledge the support of this work under a research grant from the Alex G. Nason Foundation.

REFERENCES

1. G. C. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon and D. Walker, *Solving Problems on Concurrent Processors* (Prentice Hall, Englewoods Cliffs, NJ, 1988), Vols. I and II.
2. A. Brandt, *Math. Comput.* **31,** 333 (1977).
3. W. L. Briggs, *A Multigrid Tutorial* (SIAM, Philadelphia, 1987).
4. U. Ghia, K. N. Ghia, and C. T. Shin, *J. Comput. Phys.* **48,** 387 (1982).
5. S. P. Vanka, *J. Comput. Phys.* **65,** 138 (1986).
6. R. Peyret and T. D. Taylor, *Computational Methods for Fluid Flow* (Springer-Verlag, New York/Berlin, 1983).
7. A. Brandt, GMD-Studien 85 (Gesellschaft fur Mathematik und Datenverarbeitung MBH, Bonn, 1984).
8. M. C. Thompson and J. H. Ferziger, *J. Comput. Phys.* **82,** 94 (1989).
9. C-H. Bruneau and C. Jouron, *J. Comput. Phys.* **89,** 389 (1990).
10. W. Shyy and C-S. Sun, *Comput. and Fluids* **22,** 51 (1993).
11. C. Sheng, L. Taylor, and D. Whitfield, AIAA Paper 94-2335, in *Proc., AIAA 25th Fluid Dynamics Conference, Colorado Springs, CO, 1994.*
12. T. L. Doligalski and J. D. A. Walker, *J. Fluid Mech.* **139,** 1 (1984).
13. Thinking Machines Corporation, *CMMD Reference Manual* (Thinking Machines Corporation, Cambridge, MA, 1993).
14. K. Gustafson and K. Halasi, *J. Comput. Phys.* **64,** 279 (1986).