INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS, VOL. 22, 225–240 (1996)

ITERATIVE SOLUTION OF THE INCOMPRESSIBLE NAVIER–STOKES EQUATIONS ON THE MEIKO COMPUTING SURFACE

B. A. TANYI* AND R. W. THATCHER

Manchester Centre for Computational Mathematics, Department of Mathematics, UMIST, Manchester, U.K.

SUMMARY

The numerical discretization of the equations governing fluid flow results in coupled, quasi-linear and non-symmetric systems. Various approaches exist for resolving the non-linearity and couplings. During each non-linear iteration, nominally linear systems are solved for each of the flow variables. Line relaxation techniques are traditionally employed for solving these systems. However, they could be very expensive for realistic applications and present serious synchronization problems in a distributed memory parallel environment. In this paper the discrete linear systems are solved using the generalized conjugate gradient method of Concus and Golub. The performance of this algorithm is compared with the line Gauss–Seidel algorithm for laminar recirculatory flow in uni- and multiprocessor environments. The uniprocessor performances of these algorithms are also compared with that of a popular iterative solver for non-symmetric systems (the GMRES algorithm).

KEY WORDS: domain decomposition; line Gauss–Seidel; conjugate gradient

1. INTRODUCTION

Computational fluid dynamics (CFD) is a major application area of high-performance computing. Despite recent advances in both hardware and software design, realistic simulations still require long run times. Using the most sophisticated machine and software tools on inefficient algorithms or poor solution procedures blurs the potential gains of the resources. A better appreciation of the benefits of such resources therefore requires research into improving the basic solution procedures.

There are various ways of pursuing parallelism; the one described here uses identical copies of the programme running in parallel on each processor acting on a local data subset. For the target machine (the Meiko Computing Surface) each processor has its own local memory and thus the global solution is realized by occasional data exchanges (communications) between the processors. This geometric decomposition, generally referred to as domain decomposition, is particularly suitable for porting CFD applications onto distributed memory machines because of the local character of the governing differential operator. Generally, however, the main bottleneck with such machines is the cost of effecting global communications necessitated by, for example, the formation of inner products in the conjugate gradient algorithm or parallel convergence monitoring. Also, depending on the nature of the algorithm being parallelized, it may be necessary for a rethinking of the sequential algorithm in a parallel environment. Because of the possible differences in the sequential and parallel implementa-

* Present address: Department of Computing, Imperial College, 180 Queens Gate, London SW7 2BZ, U.K.

CCC 0271–2091/96/040225–16

Received July 1994
Revised March 1995

tions, the numerical efficiency $\varepsilon_{\mathrm{num}}$ is generally employed to measure the convergence rate of the parallel implementation and the parallel efficiency $\varepsilon_{\mathrm{par}}$ is used to measure the success of the interprocessor communications, load balancing, etc.

In this paper the incompressible Navier–Stokes equation is solved using a pressure-correction-based (SIMPLE-type) procedure.[1] During each non-linear iteration, approximate solutions to the nominally linear discrete systems are obtained using the CG algorithm.[2,17] Here we accelerate the algorithm using a very simple preconditioner and compare its performance with the traditional tridiagonal matrix algorithm (TDMA) in uni- and multiprocessor distributed memory environments. To appreciate the effectiveness of the CG algorithm, we implemented and compared its convergence rate with that of a similar solver (the GMRES algorithm) with the same preconditioner on a laminar recirculatory flow problem.

2. NUMERICAL FORMULATION AND SOLUTION PROCEDURE

The fluid flow algorithm employed in this study is capable of solving two-dimensional laminar and turbulent incompressible flows. The standard k–ε model of Launder and Spalding[3] is used with the wall function approach for near-wall regions.[3,4] The general differential equation governing the transport of the general flow variable ϕ can be written in Cartesian co-ordinates in the form

$$\frac{\partial}{\partial x}(\rho u\phi)+\frac{\partial}{\partial y}(\rho v\phi)=\frac{\partial}{\partial x}\left(\Gamma\frac{\partial\phi}{\partial x}\right)+\frac{\partial}{\partial y}\left(\Gamma\frac{\partial\phi}{\partial y}\right)+S_\phi, \tag{1}$$

where u and v are the velocity components in the two co-ordinate directions, Γ is the diffusion coefficient and S_ϕ is the source term for the variable ϕ. Thus, from the above, turbulent flow modelling requires solving five dependent variables: the x- and y-momentum equations, the continuity constraint and the turbulent kinetic energy and dissipation equations (k and ε respectively). In a laminar flow situation only the first three variables are considered.

The solution domain is discretized into finite volume cells.[1] A staggered grid system is adopted where scalar variables (e.g. pressure, density) are located at the centre of the control volumes and the velocity components are located at the faces of the control volumes.[1,4] The combined convection and diffusion fluxes across the control volume faces are computed using the hybrid scheme.[1] For each control volume the finite volume scheme results in an algebraic equation of the form

$$a_{\mathrm{P}}\phi_{\mathrm{P}}=\sum_{nb=\mathrm{E,W,N,S}}a_{nb}\phi_{nb}+S_{\mathrm{U}}, \tag{2}$$

for each dependent variable, where $\phi=u,\ v,\ k,\ \varepsilon$ or p' (see below). The discrete coefficients are such that

$$a_{\mathrm{P}}=\sum_{nb=\mathrm{E,W,N,S}}a_{nb}-S_{\mathrm{P}}, \tag{3}$$

where S_{P} is the coefficient of ϕ_{P} in the source term linearization expression and is chosen such that S_{P} is unconditionally negative. S_{U} includes the constant part of the discrete form of the original source term in equation (1) plus the additional terms that cannot be approximated by the values of ϕ at the neighbouring grid nodes (E, W, N, S). The term ϕ_{P} refers to the grid point at the centre of the control volume. Thus for all the grid points the following system results:

$$B\boldsymbol{\phi}=\mathbf{c}, \tag{4}$$

where B is the pentadiagonal coefficient matrix in which all spatial couplings depending on the convective and diffusive transport of ϕ are involved and $\mathbf{c}$ contains the Dirichlet boundary conditions as well as couplings that are not considered in B. Here a pressure correction scheme is employed, the full details of which are given in References 1 and 5. The momentum equations are solved for a given pressure distribution p^* to yield a tentative velocity field u^*, v^*. Since u^* and v^* do not satisfy the continuity constraint, they and the guessed pressure must be updated. This involves solving a pressure correction equation (p'-equation). For turbulent flow, solution of the p'-equation is followed by solution of the turbulent kinetic energy and dissipation equations. A complete execution of each of the above steps constitutes a non-linear or outer iteration. The process is repeated with the corrected pressure as the new guessed pressure until convergence. Global convergence monitoring takes place after each non-linear iteration as described later.

Because of the non-linearities and couplings, underrelaxation is necessary. It is introduced in the following way:

$$\frac{a_P}{\omega}\phi_P = \sum a_{nb}\phi_{nb} + S_U + \frac{1-\omega}{\omega}a_P\phi_P^{\text{old}}, \tag{5}$$

where $0 < \omega < 1$ is the underrelaxation factor. The value of ω strongly influences the convergence rate of the outer iteration, but in general an optimal value can only be found experimentally. In the notation of equation (4) the resulting system solved for the $(n+1)$th iteration is

$$\left(B + \frac{1-\omega}{\omega}D\right)\boldsymbol{\phi}^{n+1} = \mathbf{c} + \frac{1-\omega}{\omega}D\boldsymbol{\phi}^n, \tag{6}$$

where D is the diagonal matrix whose elements are those of the main diagonal of matrix B in equation (4). Equation (6) can be written as

$$A\boldsymbol{\phi}^{n+1} = \mathbf{b}. \tag{7}$$

Since $0 < \omega < 1$, the diagonal dominance of matrix A is enhanced by the above underrelaxation procedure. During each non-linear iteration, systems such as (7) are solved for each of the flow variables. Line relaxation techniques are generally employed for solving such systems.

We consider the portion of the Cartesian grid illustrated in Figure 1. The points P, E, W, N and S are grid nodes on the vertical grid lines. Focusing attention on grid line i, in the x-direction the discrete equation for points on this line can be written in the form

$$\alpha_j\phi_j = \beta_j\phi_{j+1} + \gamma_j\phi_{j-1} + \tau_j, \tag{8}$$

where, from equation (2), $\alpha_j = a_P$, $\beta_j = a_N$, $\gamma_j = a_S$ and $\tau_j = a_E\phi_E + a_W\phi_W + S_U$. The grid system employed here is such that ϕ_1 and ϕ_{NJ} are known for an $NI \times NJ$ grid. Thus for $j = 2$ to $NJ - 1$ a system of the form

$$\begin{bmatrix} -\gamma_2 & \alpha_2 & -\beta_2 & & & \\ & -\gamma_3 & \alpha_3 & -\beta_3 & & \\ & & -\gamma_4 & \alpha_4 & -\beta_4 & \\ & & \ddots & \ddots & \ddots & \\ & & & -\gamma_{nj-1} & \alpha_{nj-1} & \beta nj - 1 \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_{nj} \end{bmatrix} = \begin{bmatrix} \tau_2 \\ \tau_3 \\ \tau_4 \\ \vdots \\ \tau_{nj-1} \end{bmatrix}$$

results for points on line i.

The line-by-line solution procedure begins with a guess of the ϕ-values over the whole grid. These are then improved from one line to the next. When solving any particular line, the values of ϕ on the neighbouring lines are their latest ϕ-values or initial guessed values; thus the τs are known. The tridiagonal systems for each grid line are solved using the tridiagonal matrix algorithm (see References 1 and 6 for details). Depending on the nature of the flow situation, it may be necessary to implement

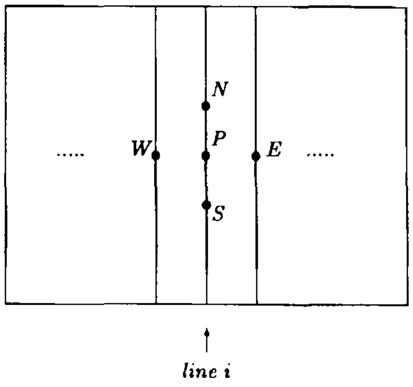

Figure 1. Vertical grid line

different directions of traverse across the flow domain (e.g. in ADI). For each flow variable a tentative solution is obtained by employing a number of 'line sweeps', the optimal number (which can only be obtained experimentally) depending on the grid size and the flow variable in question.

Despite the reliability and ease of implementation of line relaxation techniques, they can be very time-consuming for realistic applications. As a result, researchers are turning towards Krylov or conjugate direction methods. However, since systems such as equation (7) are in general non-symmetric (except for the pressure correction system[1,4], the standard CG algorithm for symmetric, positive definite systems[7] is inapplicable. A number of algorithms have been developed for general non-symmetric systems. Two such algorithms were used in this study; they are the GMRES algorithm of Saad and Shultz[8] and the conjugate gradient (CG) algorithm for linear systems developed by Concus and Golub.[2]

Each iteration of the GMRES algorithm, i.e. steps 2–4 in Algorithm 1, consists of two main steps. The first step is an Arnoldi step which consists of finding an orthogonal basis of the Krylov subspace K_m via Gram–Schmidt orthogonalization. The second step involves finding the approximate solution $\mathbf{x}_m$ in the affine space $(\mathbf{x}_0 + K_m)$ which minimizes the residual norm. This is found by solving the least squares problem whose coefficient matrix is the $(m+1) \times m$ upper Hessenberg matrix H_m. Because of the special structure of H_m,[8,9] the least squares problem is solved by a QR factorization[7] of matrix H_m which is updated at each step of the Arnoldi process in step 2. The details of the practical implementation of the algorithm are explained in References 9 and 10.

The performance of the GMRES algorithm generally depends on the number of steps performed between restarts m. Large values of m are expected to yield faster convergence but require more storage and more time per iteration. A compromise between the two is therefore necessary. For fluid flow problems the dimension of the Krylov subspace does not need to be large, since we seek only a tentative improvement in the solution during each outer iteration. In the implementation of the GMRES algorithm here the value of m was fixed, with a different value (possibly) for each of the flow variables solved.

As in the implementation of the GMRES algorithm described above, $k_{\max}$ in the CG algorithm is kept small during each non-linear iteration, with a different value (generally) for each of the flow variables (Algorithm 2). The optimal vlaues for m and $k_{\max}$ can only be obtained experimentally and are usually a function of the grid size and the flow variable being solved. A residual termination criterion can be used to control these values, but the strong non-linearity and coupling often result in erratic residual patterns.[11]

Algorithm 1 – GMRES algorithm

```
1. Start : choose an initial solution $\underline{x}_0$ and a
           dimension $m$ of the Krylov subspace
2. Arnoldi process :
     • solve $M\underline{r}_0 = (\underline{b} - A\underline{x}_0)$ for $\underline{r}_0$
     • set $\beta = \|\underline{r}_0\|$ and $\underline{v}_1 = \underline{r}_0/\beta$
     • for $j = 1, 2, ...., m$ do
         solve $M\underline{w} = A\underline{v}_j$ for $\underline{w}$
         for $i = 1, 2, ...., j$ do
           $h_{i,j} = (\underline{w}, \underline{v}_i)$
         endfor
         $\hat{v}_{j+1} = \underline{w} - \sum_{i=1}^{j} h_{i,j}\underline{v}_i$
         $h_{j+1,j} = \|\hat{v}_{j+1}\|$
         $\underline{v}_{j+1} = \hat{v}_{j+1}/h_{j+1,j}$
     endfor
     /* Define $H_m$ as the $(m+1)$ by $m$ upper Hessenberg matrix
        whose non-zero entries are the coefficients $h_{i,j}$
3. Form the approximate solution :
     • find the vector which minimizes $\|\beta\underline{e}_1 - H_m\underline{y}\|$,
         where $\underline{e}_1 = [1, 0, 0, ..., 0]^T$
     • compute $\underline{x}_m = \underline{x}_0 + \underline{V}_m\underline{y}_m$   /* $\underline{V}_m = [v_1, v_2, ...v_m]$ */
4. Restart : if satisfied stop, else set $\underline{x}_0 \leftarrow \underline{x}_m$
           and goto step 2
```

Preconditioning is crucial in the performance of Krylov solvers. Two simple preconditioners were employed in this study. We implemented both the point Jacobi and symmetric line Gauss–Seidel algorithms as preconditioners. The matrix M in the GMRES and CG algorithms is the preconditioning matrix. Concus and Golub[2] assumed a positive definite matrix M, but this assumption is not necessary.[11] If the matrix A in equation (7) is split into its lower triangular, diagonal and upper triangular parts, i.e. $A = L + D + U$, then the Jacobi method corresponds to $M = D$. The symmetric line Gauss–Seidel preconditioning employed here involved applying the TDMA line solver described above in alternating directions in the horizontal plane. Implementation details and a more mathematical presentation are described in Reference 5.

The global convergence is assessed at the end of each non-linear iteration on the basis of a 'residual source' criterion. The iterative procedure is considered to have converged if the sum of the absolute normalized residuals for u, v and all the variables as well as the mass source of the pressure correction system is less than a prescribed tolerance:

$$\max(resor_u, resor_v, resor_m, \ldots) < tol, \tag{9}$$

where

$$resor_\phi = [\sum |a_P\phi_P - (\sum a_{nb}\phi_{nb} + S_U)|]/F_{in,\phi}, \quad \phi = u, v, k, \varepsilon, \tag{10}$$

$$resor_m = (\sum |[(\rho u)_e - (\rho u)_w]\Delta y + [(\rho v)_n - (\rho v)_s]\Delta x|)/M_{in}. \tag{11}$$

Here e, w, n and s are the faces of the control volume and $\Delta x \times \Delta y \times 1$ is its volume. $F_{in,\phi}$ and M_{in} are respectively the characteristic momentum and mass flow scales.[4]

Algorithm 2 – CG algorithm

```
$\underline{x}_0$ is given
$\underline{r}_0 = \underline{b} - A\underline{x}_0$
for $k = 0, 1, 2, 3, \ldots, k_{max}$
    solve $M\underline{z}_k = \underline{r}_k$ for $\underline{z}_k$
    if $(k = 0)$ then
        $\underline{p}_0 = \underline{z}_0$
    else
        $\beta_k = -(\underline{z}_k, \underline{d}_{k-1})/(\underline{p}_{k-1}, \underline{d}_{k-1})$
        $\underline{p}_k = \underline{z}_k + \beta_k \underline{p}_{k-1}$
    endif
    $\underline{d}_k = A\underline{p}_k$
    $\alpha_k = (\underline{p}_k, \underline{r}_k)/(\underline{p}_k, \underline{d}_k)$
    $\underline{x}_{k+1} = \underline{x}_k + \alpha_k \underline{p}_k$
    $\underline{r}_{k+1} = \underline{r}_k - \alpha_k \underline{d}_k$
endfor
```

3. PROBLEM SPECIFICATION AND PROGRAMMING ENVIRONMENT

The test problem used here is that of laminar flow in a square lid-driven cavity. The problem specification is depicted in Figure 2. The lid slides across the cavity, inducing a zone of recirculatory fluid. If the grid is sufficiently fine, small counter-rotating eddies can be expected in the corners. The square cavity flow problem was chosen because of its highly recirculatory nature and the fact that the expected flow field is well understood (it has been studied by many workers; see e.g. Reference 12). Also, the singularity in the problem excites bad behaviour in an algorithm. The mesh sizes used are 36 × 36, 72 × 72 and 144 × 144 with a regular distribution.

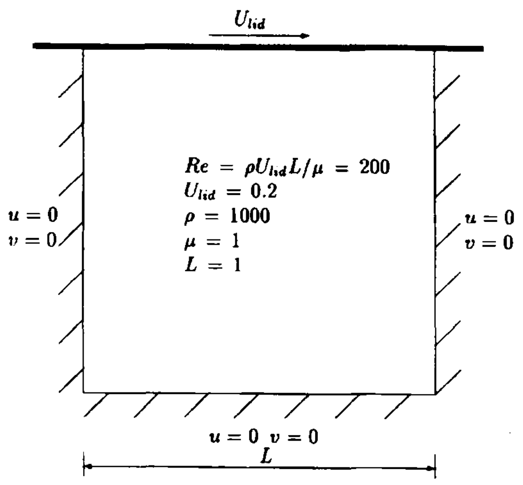

Figure 2. Flow specification

All the numerical results were obtained using the Meiko Computing Surface (MCS). The MCS is a distributed memory transputer-based machine with T800 transputers for which the link speed is 20 Mbits s^{-1}. A full description of this machine can be found in Reference 13. The programming language used here is FORTRAN. This was facilitated in the parallel implementations by the message-passing harness on the MCS called CsTools;[13] it allows parallel implementations to be undertaken in FORTRAN or C. The distance between the processors is an important consideration when there is much interprocessor communication. The choice of an appropriate connection topology can reduce this overhead. For the square cavity solution domain the processor lay-out shown in Figure 5(b) (see Section 5) was employed. Each column and row of processors has a ring connection. These rings are used for implementing global communications. From the configuration this is a two-step process. First, all the processors in the same column form the partial sum for control volumes in their respective subdomains using a systolic loop. These partial sums are then added up across the rows to form the global variable on each processor. Interprocessor communications are effected via transports[13] and all the communications are of the blocked synchronous mode.

4. SEQUENTIAL IMPLEMENTATION RESULTS

Laminar flows are commonly used to test the performance of numerical algorithms, since the effects of the pressure–velocity coupling which usually controls the convergence of the algorithm are most clearly evident for such flows. For the segregated solution approach adopted in the fluid flow code used here, the convergence rate is a function of the underrelaxation factor ω employed and the degree to which each of the variables is solved during each non-linear iteration. The optimal underrelaxation factor is usually a function of the mesh size. For the 36×36 mesh flow the optimal value is approximately $0{\cdot}8$ $(=\omega_u=\omega_v)$ and for the 72×72 and 144×144 grid problems it is approximately $0{\cdot}9$ $(=\omega_u=\omega_v)$. The pressure correction system is not relaxed in the procedure used in the code (SIMPLEC). For all the results presented here, these values were used for all the iterative solvers—line Gauss–Seidel, GMRES and CG algorithms.

Table I–III give the iterations for convergence of the various solvers for the same convergence criterion ($tol=0{\cdot}5 \times 10^{-3}$) using different inner iteration combinations. The initial flow field used is $u_{i,j}=v_{i,j}=10^{-5}$ and $p_{i,j}=0$. For pressure-correction-type procedures a higher number of iterations on the pressure correction system generally results in a faster and more stable solution.[1] Table I presents the numbers of iterations for various sweep combinations of the line Gauss–Seidel solver. Two implementations of this solver are reported. In the first (represented by TDMA) the Gauss–Seidel iteration is applied progressively on the vertical grid lines from the left of the solution domain to the right. In the other case (SGS, symmetric Gauss–Seidel) the iteration is applied in alternating directions, i.e. right to left, left to right, right to left, etc. The results show that sweeping successively from the left of the grid to the right maximizes the rate of flow of information across the solution domain for the test problem used here. The optimal application pattern of the line Gauss–Seidel iteration generally depends on the nature of the flow in question. However, because of the coupling and non-linearity of the governing equations, it is very difficult to predict the convergence behaviour. A mathematical analysis of the effect of flow direction on the convergence rates of line relaxation techniques (on constant coefficient matrices) is presented in Reference 14.

The numbers of iterations using the GMRES and CG algorithms are presented in Table II and III for different inner iteration combinations (implementing the GMRES algorithm on a 144×144 mesh problem requires mor than 4 MB of memory). Determining the exact sweep or inner iteration pattern for optimal convergence is obviously very difficult. There is a trade-off between the gain in convergence rate and the increased time spent in the equation solver. However, as explained above, the objective is to keep these to a minimum. The results clearly show the superiority of the CG algorithm

Table I. Comparison of TDMA and SGS

	No. of sweeps			TDMA		SGS	
Grid size	u	v	p'	Iterations	Time	Iterations	Time
36 × 36	2	2	3	123	176·8	176	241·6
	2	2	6	103	168·2	124	186·4
	4	4	6	89	169·7	113	184·2
	4	4	8	77	155·3	101	171·2
	4	4	12	65	148·1	85	155·2
72 × 72	2	2	6	262	1737·9	372	2270·9
	4	4	12	168	1556·6	240	1781·0
144 × 144	4	4	12	558	21235·9	885	26947·8

Table II. Performance of CG algorithm

	k_{max}				
Grid size	u	v	p'	Iterations for convergence	Time (s)
36 × 36	1	1	3	58	129·2
	2	2	3	57	153·5
72 × 72	1	1	3	116	1058·1
	2	2	3	94	1039·3
144 × 144	2	2	3	267	12086·8

Table III. Performance of GMRES algorithm

	m				
Grid size	u	v	p'	Iterations for convergence	Time (s)
36 × 36	1	1	3	71	214·6
	2	2	3	65	232·7
	2	2	6	64	298·9
	4	4	6	62	368·6
	4	4	12	62	548·4
72 × 72	1	1	3	136	1695·4
	2	2	3	110	1629·1
	2	2	6	102	1974·6
	4	4	12	91	3348·5

for the test problem. Each GMRES iteration is approximately 1·5 times a CG iteration. The performances of the Krylov solvers are, however, heavily dependent on the particular preconditioner used and the test problem in question.[15] To study the effect of the preconditioner used here on the convergence rates, the application was run using both Krylov solvers but with point Jacobi iteration as the sole preconditioner. Figures 3 and 4 show the variation in the normalized mass source $resor_m$ with iteration number on a 72 × 72 mesh problem using the same iteration patterns on both solvers and Table IV presents the relative computational efforts and iterations for convergence for the other meshes. It is worth mentioning that restarting the GMRES iteration resulted in insignificant convergence improvements and was too expensive to be of practical use for the test problem.

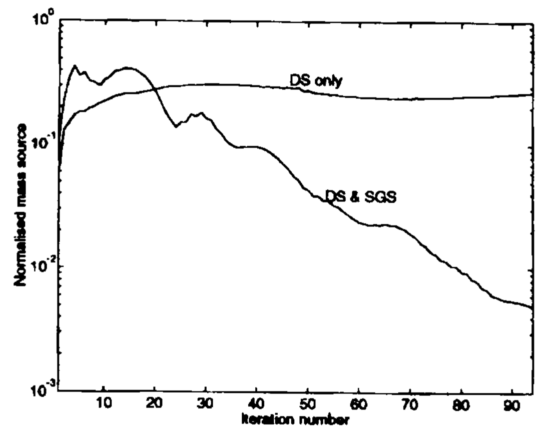

Figure 3. Effect of SGS preconditioning on convergence rate of CG algorithm

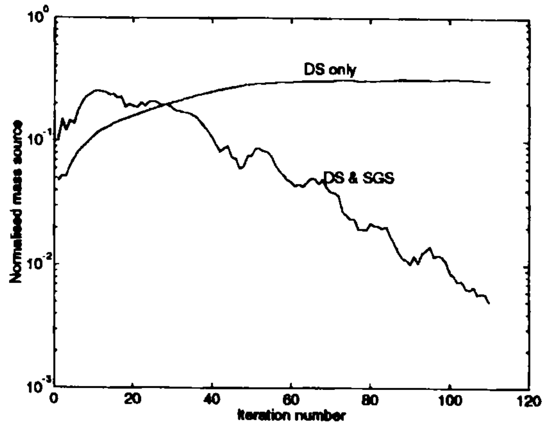

Figure 4. Effect of SGS preconditioning on convergence rate of GMRES algorithm

Table IV. Effect of preconditioning on CG and GMRES algorithms

		DS only		DS and SGS	
Solver	Grid size	Iterations	Time (s)	Iterations	Time (s)
CG	36 x 36	195	359·8	57	153·5
	72 x 72	756	5599·7	94	1039·3
	144 x 144	2636	80296·4	267	12086·8
GMRES	36 x 36	199	465·1	65	232·7
	72 x 72	1123	10712·2	110	1629·1

5. CONCURRENT IMPLEMENTATION

The parallelization of the overall fluid flow procedure can be broadly divided into three parts: coefficient assembly, equation solving and global convergence monitoring. The strategy adopted here is a domain decomposition approach.[16,20] The solution domain is split up into non-overlapping subdomains in such a way that no control volume belongs to more than one subdomain. For the test problem the domain can be split into the vertical or horizontal strips or into tiles with each subdomain allocated to a separate processor (Figure 5). Each processor forms and solves the discrete linear systems for control volumes within its subdomain.

Our parallelization strategy is based on data parallelism. The same programme runs on each processor with different data. The idea (or aim) behind the strategy is to run the subdomains in parallel and, with the communication of halo data, obtain coefficients and hence flow fields that at any stage in the execution are equal to those obtained at that same stage in the serial implementation. This can be viewed as dividing the equation system generated by the serial implementation into subsystems each running on a separate chip. In order to approximate the exact serial algorithm, the same relaxation parameters are used in each subdomain (processor) with the same number of 'sweeps' or inner iterations of the equation solver.[18,19]

For the five-point control volume scheme, control volumes next to the subdomain boundaries require data from the neighbouring processors to form the complete discrete coefficients on the respective processors. This is slightly complicated by the staggered grid system used, which introduces a diagonal data dependence alongside the nearest-neighbour data dependence. Thus processor 5 in Figure 5(a), for example, require data from processors 1, 9, 3 and 7 alongside those required from processors 2, 4, 6 and 8. For the processor interconnection scheme described in Figure 5(b), these diagonal transfers introduce a synchronization problem since the data must first be moved onto a nearest-neighbour processor before their final destination.

The line Gauss–Seidel iteration uses the latest cell approximations as the line sweeps progress. Any attempt to 'fully parallelize' the global algorithm reduces to a sequential implementation. If the line solver were to be strictly applied, then a consideration of Figure 5(a) shows that the processors would have to be synchronized in such a way that processors 2, 5 and 8 would wait for the calculations on processors 1, 4, and 7 to be completed. Furthermore, processor 4 would have to halt while the forward recursion on processor 7 proceeded for a given line, while processor 7 would similarly be blocked before the backward recursion had been completed on processor 4 in the following step, etc. Here three strategies for parallelized line Gauss–Seidel iteration are explored.

The first strategy (Strategy I) involves applying the solver to the local systems and updating halo data only on completion of each line sweep. The second strategy (Strategy II) attempts to simulate the global sequential algorithm by updating some of the lines using the most recent values calculated during the current line sweep. These approximations of the equation solver in the distributed memory environment result in convergence degradation because of the reduction of the solution implicitness along the subdomain boundaries. In order to remedy this problem, the third strategy (Strategy III)

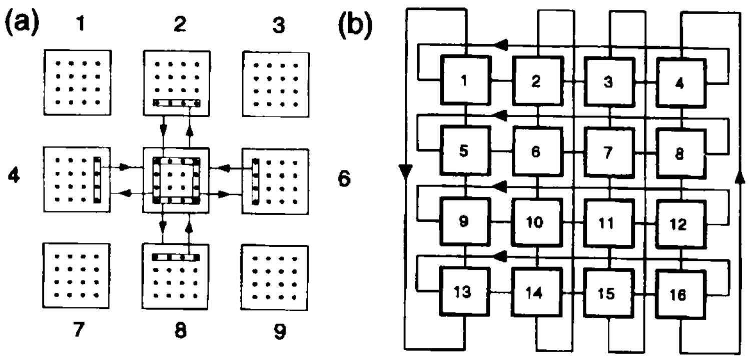

Figure 5. Interprocessor communications for (a) local and (b) global data

Strategy II

```
do i = 1,ni

    update line i in the sweep direction

    if ( i = 1 ) then
        if ( proc. is not in col. 1 ) then
          Tx. values on line 1 to proc. on left.

          if ( proc. is not in last col. ) then
            Rx. halo. data from proc. on right.
          end if
        else
          Rx. halo. data from proc. on right.
        end if
    end if

end do

update boundaries that have not been updated
```

overlaps the subdomains such that each processor solves an extra column of control volumes on the neighbouring processor's subdomain with halo data updated as in the second strategy described above. Thus, in Figure 5(a), processor 5 solves 16 extra control volumes and processor 4 solves 12 extra control volumes, etc.

The 'red–black' version of the solver is fully parallelizable if implemented within a slab decomposition. However, for tile decompositions some of the boundary nodes are updated using data from the previous sweep, thus decreasing the convergence rate of the solution procedure. This, coupled with the slow convergence rate of the red–black scheme, made it impractical for tile decompositions.[5] The line Jacobi iteration is also fully parallelizable, but the slow convergence rate made it too expensive for practical applications.[5]

On the other hand, the CG algorithm in its original form (i.e. without preconditioning) is fully explicit, with the main components being fully parallelizable. However, some of these components require global communications. The main components are the formation of inner products, matrix–vector multiplication and vector updates. Inner product formation is the most inefficient operation in the CG algorithm since it requires global communications. Each processor forms the inner product contribution for control volumes within its subdomain and uses the processor row and column rings (Figure 5(b)) to form the global value as described earlier. Matrix–vector multiplication requires only near-neighbour communications. Using the notation of equation (2), $[A \cdot \boldsymbol{\phi}]_P = a_P\phi_P - \sum a_{nb}\phi_{nb}$, where $nb = \mathrm{E, W, N, S}$. This means that ϕ is required from the neighbouring processors. Vector updates are fully local operations requiring no communications.

From the domain decomposition scheme used, the point Jacobi preconditioning operation is fully local, requiring no communications. The SGS preconditioning can be implemented using any of the strategies described above. However, for the test problem, updating subdomain boundaries did not result in any significant covergence improvement. The SGS preconditioning implementation here is therefore fully local, requiring no communications.

The global convergence monitoring is performed after each outer or non-linear iteration. Each processor forms the residual contribution for control volumes within its subdomain and, using the global communication scheme described above, forms the global residual.

6. RESULTS OF TEST CALCULATION

6.1. Performance evaluation

We saw earlier that the efficient implementation of an algorithm on a distributed memory architecture may require modifications to the original serial implementation. As a result of the possible differences in the serial and parallel implementations, a number of performance parameters are generally employed to measure various aspects of the concurrent implementation. Here we define the efficiency of a parallel implementation as

$$\varepsilon = \frac{T_1}{nT_n}, \tag{12}$$

where T_1 is the execution time for the best serial algorithm and T_n is the time taken by the parallel implementation using n processors. The efficiency is generally less than unity because of losses arising from interprocessor communications, load balancing, extra effort for overlap regions, etc.

If cv_x and cv_y are respectively the number of control volumes in the x- and y-directions of the solution domain, t_{op} is the time for one floating point operation and i_n is the average number of floating point operations per outer iteration, then the time taken by the parallel implementation using n processors can be written as

$$T_n = \frac{cv_x cv_y}{n} t_{\text{op}} i_n itn_n + t_{\text{comm}} itn_n, \tag{13}$$

where t_{comm} is the communication time per outer iteration and itn_n is the number of outer iterations required for the same convergence as the best sequential algorithm. Substituting equation (13) into equation (12), we get

$$\begin{aligned}\varepsilon &= \frac{k_1}{k_n} \frac{1}{1 + t_{\text{comm}}/tcal_n}, \\ &= \varepsilon_{\text{num}} \varepsilon_{\text{par}}\end{aligned}$$

where

$$\varepsilon_{\text{num}} = \frac{k_1}{k_n} = \frac{itn_1 i_1}{itn_n i_n}, \tag{16}$$

$$\varepsilon_{\text{par}} = \frac{1}{1 + t_{\text{comm}}/tcal_n}. \tag{17}$$

The quantities k_1 and k_n are respectively the number of floating point operations required by the serial and parallel implementations to reach a converged solution. In equation (14), $tcal_n = cv_x cv_y t_{\text{op}} i_n / n$. Equation (14) can be written as

$$\varepsilon = \frac{k_1}{k_n} \frac{1}{1 + O + C} \tag{18}$$

where O and C are respectively the overlapping and communication penalties. From the above analysis, $\varepsilon_{\text{num}} = 1$ if the parallel implementation uses non-overlapping subdomains and $itn_1 = itn_n$. This parameter does not depend on the performance characteristics of the computer. The parallel efficiency

ε_{par} gives a measure of the communication and synchronization overhead of the concurrent algorithm and from the above can be measured by fixing the number of outer iterations to be the same for both the parallel and serial implementations (in which cae $\varepsilon_{\text{num}} = 1$ and $\varepsilon = \varepsilon_{\text{par}}$). Equation (18) is generally referred to as the total efficiency ε_{tot} and is a measure of the success of the overall concurrent implementation. From this equation the communication and overlapping penalties become smaller as the problem gets bigger, with everything else fixed. Thus for a large enough problem the main departure from a linear performance will derive from any reduction in the convergence rate that may result.

6.2. *Square cavity flow results*

Although the effect of the decoupling between the subdomains introduced by the approximation of the parallelized Gauss–Seidel iteration cannot be predicted because of the nature of the governing equations, experience shows that the rate of convergence of the parallel algorithm is mostly affected by the ratio of the number of inner boundary nodes to the total number of nodes. Thus for a given number of processors the rate of convergence is less affected on a finer mesh. Therefore, to appreciate the performances of the concurrent line Gauss–Seidel strategies described above, they were tested on a coarse mesh (36×36) with a maximum of 36 processors (giving a minimum subproblem of size 6×6). The iterations for convergence and computational efforts (in seconds) are presented in Table V. For strategy III the extra cost incurred in solving the overalp nodes more than compensates for the convergence improvement. Therefore, for the test problem here, strategy II appears to be the most suitable and is used in the rest of this paper.

Unlike the implicit Gauss–Seidel iteration, the CG algorithm in its original form is fully explicit, with the main components being fully parallelizable. Also, the point Jacobi preconditioning step has no communication overheads. Thus a parallel implementation of the CG algorithm with diagonal scaling as the sole preconditioner converges in the same number of iterations (as the serial implementation) on any number of processors or processor configuration. Figure 6 presents the variation in the normalized absolute mass source $resor_m$ with time for different numbers of processors on a 36×36 mesh problem. The symmetric Gauss–Seidel (SGS) preconditioning used to accelerate the CG algorithm introduces the convergence problems mentioned above. However, for the test problem used, a fully local application of the preconditioner (i.e. no communications between neighbouring processors) resulted in approximately the same number of iterations as that employing any of the coupling strategies described above. To minimize communication costs, the SGS preconditioning implementation here is fully local.

A direct comparison of the equation solvers is very difficult considering the large number of parameters involved for optimal performance. Our comparison here is based on the results of the

Table V. TDMA coupling strategies

	Processor configuration		Strategy I		Strategy II		Strategy III	
n_{proc}	x_{proc}	y_{proc}	Iterations	Time	Iterations	Time	Iterations	Time
1	1	1	65	148·1	65	148·1	65	148·1
4	2	2	66	40·1	66	40·1	65	42·9
8	2	4	67	22·5	67	22·5	66	27·1
	4	2	66	22·4	64	21·7	65	26·9
16	4	4	68	13·8	66	13·4	66	17·3
36	6	6	81	10·2	68	8·5	66	11·2

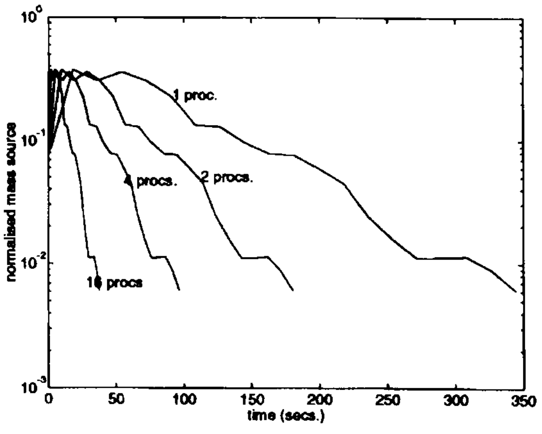

Figure 6. Convergence behaviour using point Jacobi preconditioner

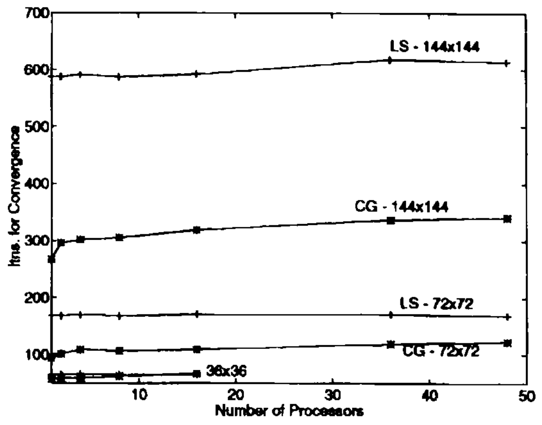

Figure 7. Relative convergence rates of TDMA(LS) and CG algorithms

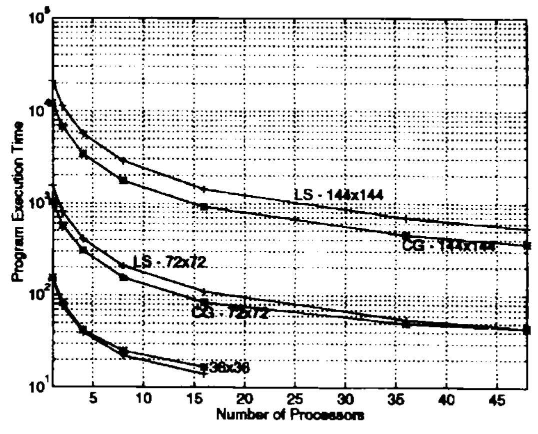

Figure 8. Relative computational efforts

Table VI. Parallel efficiencies (%) of laminar flow procedure

	144 × 144 grid		72 × 72 grid		36 × 36 grid	
n_{proc}	LS	CG	LS	CG	LS	CG
1	100	100	100	100	100	100
4	98·9	98·7	96·5	95·7	93·0	89·9
16	98·7	98·0	89·8	88·2	69·6	67·6
36	94·2	93·8	80·0	74·4	—	—
48	92·1	90·0	78·1	72·4	—	—

parallel implementation of the optimal sequential results presented above for each mesh and solver. The main objective behind the use of parallel processing is the reduction of the execution time of an implementation. The numbers of iterations for convergence and the computational efforts of the optimal sequential implementation on multiple processors using both equation solvers are presented in Figures 7 and 8. The computed parallel efficiencies ε_{par} are presented in Table VI for both solvers (ε_{par} is evaluated using $itn_1 = itn_n$). The global communication overhead in the CG algorithm is evident in the results.

7. CONCLUSIONS

The effectiveness of the generalized CG algorithm for linear systems has been demonstrated for a laminar recirculatory flow problem using a very simple preconditioner. The uni- and multiprocessor performance of the algorithnm was compared with that of traditional line Gauss–Seidel iteration. We also compared its uniprocessor performance with that of a similar Krylov solver for non-symmetric systems—the GMRES algorithm. For the test problem used, the line Gauss–Seidel solver has the lowest cost per outer iteration, but this advantage is lost by the extremely slow convergence rate. The convergence rates of the generalized CG algorithm and the GMRES iteration are very close, with each outer iteration of the latter being approximately 1·5 times more expensive.

Parallelization of the line Gauss–Seidel iteration results in convergence degradation because of the reduction of the solution implicitness along the subdomain boundaries. Three strategies for dealing with this problem have been presented and compared. Obviously the choice of strategy will depend on the test problem used. On the other hand, the generalized CG algorithm in its original form is fully explicit, but requires global communications in a distributed memory parallel environment. However, it has been demonstrated that for a large enough problem these communication overheads are minimal.

ACKNOWLEDGEMENT

The authors would like to thank Dr. David Golby of the Department of Mechanical Engineering at UMIST for his useful ideas and suggestions throughout the course of this work.

REFERENCES

1. S. V. Patankar, *Numerical Heat Transfer and Fluid Flow*, Hemisphere, Washington, DC, 1980.
2. P. Concus and G. H. Golub, 'A generalised conjugate gradient method for non-symmetric systems of linear equations', in R. Glowinski and J. L. Lions (eds), *Computing Methods in Applied Sciences and Engineering*, Lecture Notes in Economics and Mathematical Systems, Vol. 134, Springer, Berlin, 1976, pp. 56–65.

3. B. E. Launder and D. B. Spalding, *Mathematical Models of Turbulence*, Academic, London, 1972.
4. A. D. Gosman and F. J. K. Ideriah, 'TEACH-T: a general computer program for two-dimensional, turbulent, recirculatory flows', *Rep*ME-952-76. Department of Mechanical Engineering, Imperial College, London, 1976.
5. B. A. Tanyi, 'Iterative solution of the incompressible Navier-Stokes equations on a distributed memory parallel computer'. *Ph.D Thesis*, University of Manchester Institute of Science and Technology, 1993.
6. G. D. Smith, *Numerical Solution of Partial Differential Equations: Finite Difference Methods*, 3rd edn, Oxford University Press, Oxford, 1985.
7. G. H. Golub and C. F. Van Loan, *Matrix Computations*, 2nd edn, Johns Hopkins University Press, Baltimore, MD, 1989.
8. Y. Saad and M. H. Shultz, 'A generalized minimal residual algorithm for solving non-symmetric linear systems', *SIAM J. Sci. Stat Comput.*, **7**, 856–869 (1986).
9. F. Shakib, T. J. R. Hughes and Z. Johan, 'A multi-element group preconditioned GMRES algorithm for non-symmetric systems arising in finite element analysis', *Internal Rep.*, Institute for Computer Methods in Applied Mechanics and Engineering, Stanford University, 1988; *Methods Appl. Mech. Eng.*, in press; *Proc. IMA Workshop on Iterative Methods*, in press.
10. X. Xu and B. E. Richards, 'α-GMRES: a new parallelisable iterative solver for large sparse non-lineaer systems arising from CFD', *GU Aero Rep. 9110*, Department of Aerospace Engineering, University of Glasgow, 1991.
11. B. Noll and S. Wittig, 'Generalised conjugate gradient method for the efficient solution of three-dimensional fluid flow problems', *Numer. Heat Transfer B*, **20**, 207–221 (1991).
12. B. R. Latimer and A. Pollard, 'Comparison of pressure-velocity coupling solution algorithms', *Numer. Heat Transfer*, **8**, 635–652 (1985).
13. *CSTools Reference Manual*, Meiko, Bristol, 1989.
14. H. C. Elman and M. P. Chernesky, 'Ordering effects on relaxation methods applied to the discrete convection-diffusion equation', Tech. Report UMIACS-TR-92-40, Institute for Advanced Computer Studies, University of Maryland, 1992.
15. J. N. Shadid and R. S. Tuminaro, 'A comparison of preconditioned nonsymmetric Krylov methods on a large-scale MIMD machine', *Rep. SAND91-0333*, Sandia National Laboratories, Albuqueque, NM, 1991.
16. M. E. Braaten, 'Development of a parallel computational fluid dynamics algorithm on a hypercube computer', *Int. j. numer. methods fluids*, **12**, 947–963 (1991).
17. M. Hestenes and E. Stiefel, 'Methods of conjugate gradients for solving linear systems', *J. Res. Natl. Bur. Stand. B*, **49**, 409–436 (1952).
18. E. Schreck and M. Peric, 'Computation of fluid flow with a parallel multi-grid solver', *Rep.* LSTM 327/N/91, University of Erlangen, 1991.
19. B. A. Tanyi and R. W. Thatcher, 'On the parallelisation of the TEACH-T code for computing fluid flows', *Tech. Rep. 216*, University of Manchester/UMIST Joint Numerical Analysis Reports, 1992.
20. B. A. Tanyi and R. W. Thatcher, 'Fluid flow calculation on a distributed memory parallel computer', in K. I. McKinnon and F. Plab (eds), *Proc. Second Workshop on Parallel Numerical Analysis, EPCC-TR92-22*, Edinburgh, June 1992.