INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS, VOL. 19, 275–294 (1994)

A METHOD FOR FINITE ELEMENT PARALLEL VISCOUS COMPRESSIBLE FLOW CALCULATIONS

LAURA C. DUTTO AND WAGDI G. HABASHI

Department of Mechanical Engineering, CERLAC, Concordia University, Montreal, Quebec, Canada

MICHEL P. ROBICHAUD

Numerical Methods Group, Pratt & Whitney Canada, Longueuil, Quebec, Canada

AND

MICHEL FORTIN

Département de Mathématiques et de Statistique, Université Laval, Ste-Foy, Quebec, Canada

SUMMARY

This paper presents the parallelization aspects of a solution method for the fully coupled 3D compressible Navier–Stokes equations. The algorithmic thrust of the approach, embedded in a finite element code NS3D, is the linearization of the governing equations through Newton methods, followed by a fully coupled solution of velocities and pressure at each non-linear iteration by preconditioned conjugate gradient-like iterative algorithms. For the matrix assembly, as well as for the linear equation solver, efficient coarse-grain parallel schemes have been developed for shared memory machines, as well as for networks of workstations, with a moderate number of processors. The parallel iterative schemes, in particular, circumvent some of the difficulties associated with domain decomposition methods, such as geometry bookkeeping and the sometimes drastic convergence slow-down of partitioned non-linear problems.

KEY WORDS Navier Stokes Fully-coupled solution Iterative solvers Finite-element methods Parallel algorithm Block diagonal preconditioning CFD

1. INTRODUCTION

The fully-coupled solution of the Navier–Stokes equations is an attractive approach. When embedded in a Newton algorithm for the overall non-linear problem, its convergence characteristics are hard to match. The approach, however, yields large banded and non-symmetric, non-positive definite systems of linear equations that need to be solved at each non-linear step. The equation solver in a classical fully-coupled finite element approach can consume between 80%–90% of the overall solution time. Its parallelization is therefore seen as the key to the success of any parallel finite element methodology. Solvers for such large-scale systems, originally developed exclusively for supercomputers, can now be contemplated on workstations, as architecture, processor speed and memory size have significantly evolved.

Preconditioning techniques have also come of age and are now widely used to guarantee and accelerate convergence of such iterative solvers.[1,2] In particular, preconditioners based on an incomplete factorization of the Jacobian matrix yield a good acceleration of iterative solvers[3] for a wide range of problems while being easy to generate and use. If the factors are constrained

CCC 0271–2091/94/160275–20
© 1994 by John Wiley & Sons, Ltd.
Received August 1993
Revised January 1994

to have the same sparsity pattern as the original matrix, one obtains the simple but powerful incomplete LU preconditioner ILU(0), originally proposed by Meijernik and van der Vorst[4] to accelerate the resolution of symmetric, positive definite linear systems by the conjugate gradient method. Unfortunately, both the factorization and triangular system solution steps in such preconditioners are essentially sequential processes. Thus for large-scale solutions on parallel computers, ILU(0) preconditioners are less attractive than more easily parallelizable and vectorizable ones such as relaxation techniques, diagonal preconditioners or element-by-element factorization schemes.[5]

In this paper a block-diagonal preconditioner that can be easily parallelized on coarse-grain architectures is proposed. The numerical results show that it is suitable for parallel computations on a moderate number n of processors, i.e. $n \approx O(10)$. Knowing n, the system matrix A is partitioned into $n \times n$ blocks. Solely for the purpose of defining the preconditioning matrix, A is approximated to $\tilde{A}[n]$, keeping only the non-zero coefficients lying on its diagonal blocks. With a suitable nodal reordering, $M[n]$, the incomplete ILU(0) factorization of $\tilde{A}[n]$, can be shown to be an effective preconditioner for iterative solvers such as the generalized minimum residual[1] (GMRES) and conjugate gradient squared[2] (CGS). Because of its strictly diagonal block structure, it is possible to parallelize both the factorization and triangular system solution steps associated with $M[n]$, the truncated preconditioning matrix, without losing the robustness of the often finicky iterative algorithms.

Therefore one can see that the parallelization scheme is based on two steps: first, a partition into blocks of the system matrix A based *only* on its sparsity structure (and not on the geometry of the problem as in domain decomposition methods); second, to define the preconditioning matrix, an approximation of the global matrix into a block-diagonal one, followed by an ILU(0) factorization of the latter. A major feature of such a parallelization scheme is that it offers the advantages of domain decomposition methods while overcoming their bookkeeping and slow-down hurdles.

The methodology of efficient parallelization of such iterative solvers on shared memory parallel workstations is studied here in the framework of a fully parallel approach to viscous flow problems.

In the following sections the governing equations, discretization method and a hybrid artificial viscosity algorithm, necessary for obtaining stable solutions at high Reynolds numbers are presented. The issues associated with the block-diagonal approximation of the system matrix used to build the parallelizable preconditioner are discussed. Finally, demonstration of the methodology for viscous three-dimensional, compressible, laminar flows is presented.

2. GOVERNING EQUATIONS AND DISCRETIZATION

The 3D compressible, variable property, Navier–Stokes equations can be written as

continuity

$$\partial\rho/\partial t + \nabla\cdot(\rho\mathbf{u}) = 0, \tag{1}$$

momentum

$$\partial\rho\mathbf{u}/\partial t + \rho(\mathbf{u}\cdot\nabla)\mathbf{u} + \mathbf{u}(\nabla\cdot\rho\mathbf{u}) = -\nabla p + (1/Re)[-\tfrac{2}{3}\nabla(\mu_t\nabla\cdot\mathbf{u}) + \nabla\times\mu_t(\nabla\times\mathbf{u}) + 2(\nabla\cdot\mu_t\nabla)\mathbf{u}], \tag{2}$$

energy equation (constant total enthalpy)

$$H_0 = \frac{\gamma}{\gamma - 1}\frac{p}{\rho} + \tfrac{1}{2}\mathbf{u}\cdot\mathbf{u}, \tag{3}$$

equation of state

$$p/\rho = RT. \tag{4}$$

Here ρ, $\mathbf{u}$, p, T and H_0 denote the density, velocity vector, pressure, temperature and total enthalpy of the fluid respectively, while Re and μ_t are the Reynolds number and turbulent viscosity respectively.

To overcome the well-known odd–even decoupling or checker-boarding effect in a simple manner, a pressure dissipation term $\lambda\Delta p$ can be added to the continuity equation and (1) may be rewritten as

$$\partial\rho/\partial t + \nabla\cdot(\rho\mathbf{u}) = \lambda\Delta p, \tag{1'}$$

where λ is a small coefficient. For the numerical discretization of the system (1')–(4) the Galerkin approach is taken, whereby the governing equations are satisfied in an average sense with respect to a weight function. The isoparametric finite element approach is then introduced with all variables and geometry described by trilinear shape functions.[6]

For high Reynolds number flows turbulence is modelled in NS3D with the classical high Reynolds number k–ε model. Near walls a special wall element, using logarithmic shape functions in the direction normal to the wall, is employed.

After discretization, Newton linearization and assembly, one must solve a linear system of equations at each Newton iteration for the four unknowns ($\rho\mathbf{u}$, p), followed by an update of density. For turbulent flows the strategy would be to solve the Navier–Stokes equations for a few Newton iterations (say five) before each update of the turbulent viscosity through the k–ε equations.

3. SOLUTION PROCEDURE

3.1 The iterative equation solver

The system of equations to be solved at each non-linear iteration is of the form

$$A\Delta\mathbf{X} = -\mathbf{R}_\mathbf{X}, \tag{5}$$

where $\Delta\mathbf{X} = (\Delta\rho\mathbf{u}, \Delta p)^\mathrm{T}$ and $\mathbf{R}_\mathbf{X} = (\mathbf{R}_{\rho\mathbf{u}}, R_p)^\mathrm{T}$, with Δ indicating the change in a variable and $\mathbf{R}_\mathbf{X}$ being the vector of residuals of the equations calculated at the previous iterative level.

This system, here involving four variables for 3D flows, can be solved using direct or iterative methods. Direct methods have been successfully applied to the solution of the above system of equations,[7] running at speeds of 2.3 GFLOPS on a Cray YMP-8 and currently running at speeds exceeding 10 GFLOPS on an NEC-SX3 with four processors. However, for a typical $N = n \times n \times n$ grid where the bandwidth varies as n^2, such direct methods require $O(N^{2.33})$ operations for the factorization step and $O(N^{1.67})$ operations for the substitution step, with storage requirements proportional to $O(N^{1.67})$. Memory limitations on supercomputers therefore severely limit the application of such direct solvers in three-dimensional flow situations.

Iterative methods for the Newton correction, on the other hand, can offer the advantage of $O(N)$ storage and, under certain conditions, preconditioned conjugate gradient (PCG) methods

can produce a machine-accurate solution in $O(N^{1.17})$ operations. Because of their low storage requirement, linear growth and moderate operation count, iterative methods are therefore well adapted to engineering workstations which are characterized by large memory and fast scalar performance. These advantages must, however, be tempered by the sensitivity of iterative methods to matrix conditioning.

Therefore, in order to improve the convergence of the iterative algorithm, the linear system in (5) is replaced by

$$M^{-1}A\Delta\mathbf{X} = -M^{-1}\mathbf{R}_x, \tag{6}$$

where M is a preconditioning matrix approaching A, selected here on the basis of the incomplete factorization of A.

For parallel processing, the matrix A is partitioned as explained in the next sections. It is also found that a reordering of its non-zero elements may have a great influence on its incomplete Gauss factorization.[8,9] The natural ordering (associated with an $I \times J \times K$ structured grid) is therefore compared with the minimum neighbouring (MINNEIG) ordering[9,10] to assess the sensitivity of the solver to the ordering strategy.

3.2. *A hybrid artificial viscosity unloading scheme*

Because of the convective terms of the Navier–Stokes equations, the resulting linear system is ill-conditioned at high Reynolds numbers. On coarse grids, therefore, a streamline diffusion may be necessary on both sides of the equation system to stabilize the solution. This can be written in the symbolic form

$$A(\lambda, \mu)\Delta\mathbf{X} = -\mathbf{R}_x(\lambda, \mu), \tag{7}$$

with $\mathbf{R}_X(\lambda, \mu) = (\mathbf{R}_{\rho u}(\mu), R_p(\lambda))^T$.

In the present work a strategy is used whereby the iteration matrix A is computed with progressively lower values of the parameters λ and μ, referred to as $(\lambda^{LHS}, \mu^{LHS})$, but higher than those in the residual $\mathbf{R}$, denoted by $(\lambda^{RHS}, \mu^{RHS})$. The residual, representing the physics of the problem, is therefore always computed with the smallest possible values of the parameters for which the outer Newton iteration converges. Thus the hybrid artificial viscosity algorithm can be described as follows.

1. Set $\mu^{RHS} = \mu^{LHS}$ and $\lambda^{RHS} = \lambda^{LHS}$.
2. $\mathbf{X}_0 = (\rho\mathbf{u}_0, p_0)^T$ being given, compute $\|\mathbf{R}_{x_0}\|$, with $\mathbf{R}_{x_0} = (\mathbf{R}_{\rho u_0}, R_{p_0})^T$. Assemble the matrix $A(\lambda^{LHS}, \mu^{LHS})$ and calculate the preconditioning matrix M.

Newton iteration

3. Solve $\Delta\mathbf{X}_i = (\Delta\rho\mathbf{u}_i, \Delta p_i)^T$ with a preconditioned iterative solver at each Newton iteration,

$$M^{-1}A(\lambda^{LHS}, \mu^{LHS})\Delta\mathbf{X}_i = -M^{-1}\mathbf{R}_{x_i}(\lambda^{RHS}, \mu^{RHS}),$$

   by reducing the initial residual norm by 10^{-3}.
4. Update $\mathbf{X}$,

$$\mathbf{X}_{i+1} = \begin{pmatrix} \rho\mathbf{u}_{i+1} \\ p_{i+1} \end{pmatrix} = \begin{pmatrix} \rho\mathbf{u}_i \\ p_i \end{pmatrix} + \begin{pmatrix} \Delta\rho\mathbf{u}_i \\ \Delta p_i \end{pmatrix},$$

   till $\|\mathbf{R}_{x_{i+1}}\|/\|\mathbf{R}_{x_0}\| < 10^{-m}$; repeat from step 3 ($m = 6$ for the last cycle; otherwise $m = 1$).

5. Turbulence equations, when needed, are solved every five Newton iterations.
6. Lower $(\lambda^{RHS}, \mu^{RHS})$ and $(\lambda^{LHS}, \mu^{LHS})$; repeat from step 2 if necessary.

It is important to note that the hybrid artificial viscosity scheme has proven a key element in allowing large time steps, making the use of iterative methods viable for steady-state problems with the present scheme.

4. PARALLELIZATION ASPECTS

4.1. Matrix block partitioning

A simple idea to build a parallelizable preconditioner is to approximate the Jacobian matrix A by a point- or block-diagonal one. However, this approximation cannot be done with impunity, because the preconditioner will lose robustness. To address this concern, a block-diagonal preconditioner suitable for a large class of problems is proposed.

The strategy used is as follows. A coarse-grain structure of the system matrix A is calculated using only the connectivities between the nodes. Let n be the available number of processors, assumed not to be large, $n \approx O(10)$. A regular partition of the coarse-grain structure of A into an $n \times n$ block matrix is carried out, i.e. consecutive unknowns are put in the same block by a sequential process. A local reordering of the nodes in each block is done, if desired, yielding a zero–non-zero structure adapted to a subsequent ILU(0) factorization.[9] After this preliminary step the complete structure of the system matrix A is calculated in the usual manner using the basis functions and the finite element grid defining the discretization scheme. Each block i $(i = 1, \ldots, n)$ has associated with it m_i unknowns, where $m_i \approx N/n$ and $\Sigma_{i=1}^{N} m_i = N$. In general there is a slight difference in the sizes of the blocks because one does not send to different blocks the unknowns associated with the same node at a block boundary node. Therefore the system matrix A is seen as a block-structured one,

$$A = \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{pmatrix} = \begin{pmatrix} A^1 \\ A^2 \\ \vdots \\ A^n \end{pmatrix}, \tag{8}$$

where $A^i = [A_{i1}\ A_{i2}\ \ldots\ A_{in}]$ is the ith block row of A, $i = 1, \ldots n$. Solely for the purpose of defining the parallelizable preconditioner, the system matrix A is approximated by a block-diagonal one $\tilde{A}[n]$, neglecting the non-zero coefficients outside the diagonal blocks. Thus this auxiliary matrix $\tilde{A}[n]$ is composed of n block-diagonal submatrices of size $m_i \times m_i$,

$$\tilde{A}[n] = \begin{pmatrix} \tilde{A}_1 & 0 & \cdots & 0 \\ 0 & \tilde{A}_2 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & \tilde{A}_n \end{pmatrix} = \begin{pmatrix} A_{11} & 0 & \cdots & 0 \\ 0 & A_{22} & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & A_{nn} \end{pmatrix}, \tag{9}$$

where $\tilde{A}_i = A_{ii}$, $i = 1, \ldots, n$. If $n = 1$, one has $\tilde{A}[1] = A$, and if $n = N$, $A[N]$ would consist only of the diagonal coefficients of A. With this partition algorithm and maintaining $n \ll N$, the number of non-zero coefficients dropped from the original matrix to approximate it by a block-diagonal one is not significant for the test cases presented here.

An incomplete factorization of the matrix $\tilde{A}[n]$ (i.e. of the individual submatrices $\tilde{A}_i$, $i = 1, \ldots, n$) is performed by ignoring any fill-in produced during the factorization. The resulting

triangular matrices therefore have the same sparsity pattern as $\tilde{A}[n]$, but when multiplied together no longer yield $\tilde{A}[n]$. The product of the triangular factors obtained, even if in practice it is never calculated explicitly, is denoted by $M[n]$, an effective, or approximate, preconditioning matrix.

Since $\tilde{A}[n]$ is block-diagonal, one can calculate its ILU(0) factorization for all blocks simultaneously. As well as $\tilde{A}[n]$, $M[n]$ is block-diagonal, so one can solve in parallel the triangular systems associated with it. It is clear that the intermediate matrix $\tilde{A}[n]$ is overwritten by its (incomplete) triangular factors defining $M[n]$. From the previous development, $M[n]$ approaches $A[n]$ which in turn approximates A; thus $M[n]$ is an approximation of A, more or less efficient depending on the influence of the neglected coefficients.

Figure 1 shows the structure of a banded matrix and Figure 2 the structure of the associated approximate matrix $\tilde{A}[n]$ given by this algorithm, with $n = 4$ and $N = 3114$. From the banded structure of the matrix A it is clear that only some blocks on each row are non-zero: A_{11} and A_{12} from A^1; A_{21}, A_{22} and A_{23} from A^2; A_{32}, A_{33} and A_{34} from A^3; A_{43} and A_{44} from A^4. When the matrix is not banded but only sparse, a similar topology can sometimes be obtained by reordering the unknowns in an appropriate way.

Another example is shown in Figures 3 and 4. The pattern of matrix A is shown in Figure 3. Figure 4 shows the differences in structure between matrix A and the approximate matrix $\tilde{A}[3]$ obtained by this algorithm.

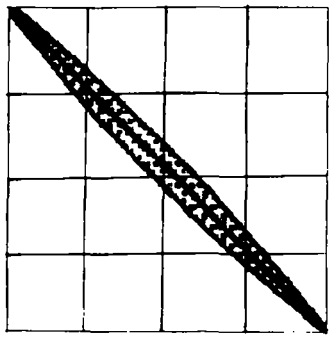

Figure 1. Structure of a banded matrix

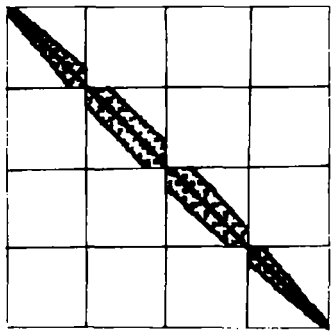

Figure 2. Partition into $n = 4$ blocks associated with the matrix of Figure 1

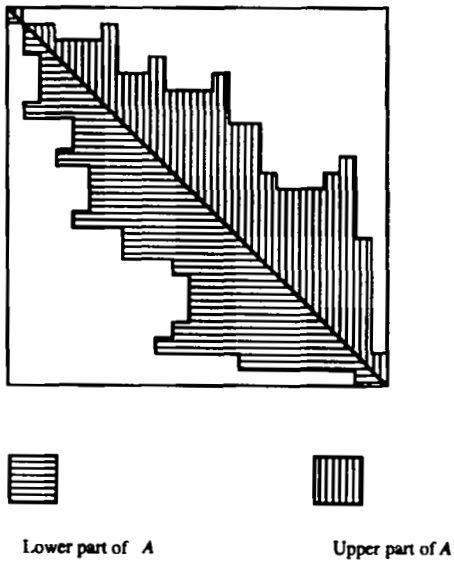

Figure 3. Structure of a banded matrix

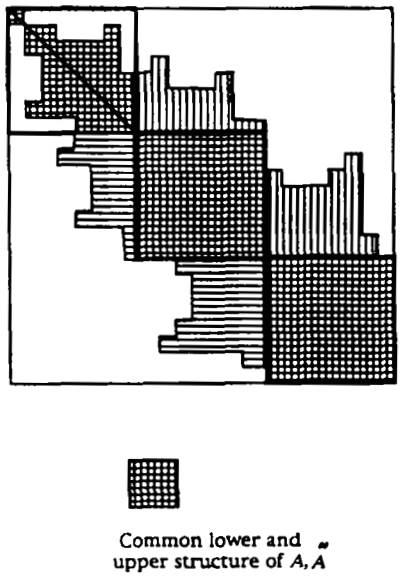

Figure 4. Partition into $n = 3$ blocks associated with the matrix of Figure 3

4.2. Matrix and vector storage

In order to solve equations (6) by an iterative method, two matrices need to be stored: the matrix A defining the linear system to solve, and the preconditioning matrix $M[n]$ (in fact, one stores its triangular factors rather than $M[n]$ itself). Both matrices are stored in 32-bit precision in a condensed form using a skyline storage mode. The non-zero entries of matrix $A = (a_{ij})$ are

stored in three real vectors, here called VD, VL and VU. The array VD has the N diagonal coefficients of A, i.e. $\mathrm{VD}(i) = a_{ii}$. The non-zero coefficients of the lower triangle of A are stored in VL, row by row, while the non-zero entries of the upper triangle of A are in VU, column by column. For the finite element method the zero–non-zero structure of A is symmetric, thus allowing one to define for VL and VU the same integer pointer arrays for the position of VL's and VU's coefficients in A. Thus a vector KL can be used to indicate the cumulated number of non-zero entries in a row and a vector KCOL the column number associated with them. In other words,

$$\mathrm{VL}(q) = a_{ij} \quad \Leftrightarrow \quad \mathrm{VU}(q) = a_{ji} \quad \Leftrightarrow \quad [\mathrm{KL}(i) \leqslant q < \mathrm{KL}(i+1) \text{ and } \mathrm{KCOL}(q) = j].$$

The length of VL, VU and also KCOL is

$$v_A = \mathrm{KL}(N+1) - 1. \tag{10}$$

The total number of non-zero coefficients in A is therefore $2v_A + N$.

The preconditioning matrix is stored in a similar way. First let VDP, VLP, VUP, KLP and KCOLP be initialized by VD, VL, VU, KL and KCOL respectively. The parameter n being given, one can determine the regular partition into n blocks of A and neglect in the preconditioning matrix the non-zero coefficients not included in the diagonal blocks. Thus one obtains a first approximation of A: the n-block-diagonal matrix $\tilde{A}[n]$. The vectors VLP, VUP, KLP and KCOLP are modified if necessary, destroying some entries from the original vectors. The number of non-zero coefficients, v_P, of the lower and upper triangles of $\tilde{A}[n]$ decreases in general, and one has

$$v_P = \mathrm{KLP}(N+1) - 1 \leqslant v_A, \tag{11}$$

with v_A given by (10).

At this stage the structure of the preconditioning matrix given by KLP and KCOLP will no longer be modified, but only the entries of it. For this an ILU(0) factorization of $\tilde{A}[n]$ is carried out, positioning the triangular and diagonal factors in the storage used by the matrix $\tilde{A}[n]$ itself.

The knowledge of the partition of A into n blocks enables one to prepare matrices and vectors for parallel computation. Thus one can write the matrix A and the preconditioning matrix $M[n]$ in n rows as

$$A = \begin{pmatrix} A^1 \\ A^2 \\ \vdots \\ A^n \end{pmatrix}, \qquad M[n] = \begin{pmatrix} M^1 \\ M^2 \\ \vdots \\ M^n \end{pmatrix},$$

where A^i and M^i are $m_i \times N$ matrices, $m_i \approx N/n$ and $\Sigma_{i=1}^{n} m_i = N$. As previously remarked, each row of A has a few non-zero blocks depending on its structure. When A is a banded matrix, each row probably has some zero blocks according to the bandwidth, as seen in Figure 1, and this fact is taken into account in the parallelization of the whole algorithm, especially on matrix–vector products.

The matrices M^i have by definition only one non-zero block, corresponding to the diagonal blocks:

$$M[n] = \begin{pmatrix} M^1 \\ M^2 \\ \vdots \\ M^n \end{pmatrix} = \begin{pmatrix} M_1 & 0 & \cdots & 0 \\ 0 & M_2 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & M_n \end{pmatrix}. \tag{12}$$

An arbitrary vector $\mathbf{x} \in \mathbb{R}^N$ can be written as

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}^1 \\ \mathbf{x}^2 \\ \vdots \\ \mathbf{x}^n \end{pmatrix}, \tag{13}$$

where $\mathbf{x}^i$ is a vector of order m_i corresponding to A^i, $i = 1, \ldots n$.

With the splitting of matrices A and $M[n]$ and vector $\mathbf{x} \in \mathbb{R}^N$ described above, one stores A^i, M_i, and $\mathbf{x}^i$ in processor i, $i = 1, \ldots, n$.

4.3. Parallelization of the hybrid artificial viscosity algorithm

Although the test cases presented here were run on a shared memory parallel computer, the aim of this work is to develop a general iterative algorithm that can also be implemented in parallel on distributed memory machines or on a network of workstations. Communications between processors has therefore to be considered. In this subsection the parallelization of the hybrid artificial viscosity algorithm is discussed, keeping in mind its general implementation on distributed memory machines with a moderate number of processors. The parallelization maintains the effectiveness of the original scheme owing to the algorithmic equivalence of the sequential and parallel versions.

From the hybrid artificial viscosity algorithm described in a previous section and the CG-like Krylov subspace methods used in this paper, the main parallelization tasks are

(i) the assembly of the matrix A
(ii) the computation of the block-diagonal preconditioner
(iii) the product of a matrix and a vector
(iv) the inner product of two vectors.

These aspects are described in the following subsections, but it should be remarked at the outset that the assembly of the matrix and all operations involving the preconditioning matrix *do not need communication* between processors. Only the matrix–vector products involving the full matrix A require limited communication.

4.3.1. Assembly of the matrix A. At each time that step 2 of the hybrid artificial viscosity scheme is repeated, one needs to assemble the matrix A. For each row A^i of A this is done by adding the contribution of each element of the finite element grid. The parallelization of this step is easily achieved, considering on each processor all the elements having at least one unknown associated with the current row. It is clear that some of the elements will be considered by more than one processor at the same time, but in general this duplicate work is negligible compared with the overall work done by each processor.

4.3.2. Parallelization of the preconditioner. When the matrix A is updated, the approximate matrix $\tilde{A}[n]$ and consequently the preconditioning matrix $M[n]$ need to be modified. This occurs at step 2 of the hybrid artificial viscosity scheme. Thus it is important to parallelize the ILU(0) factorization of $\tilde{A}[n]$ in order to minimize the CPU time used in this step. From the definition of $\tilde{A}[n]$ by (9) it is clear that the ith processor ($1 \leqslant i \leqslant n$) needs only the diagonal block $\tilde{A}_i$ in order to produce the lower and upper factors defining M_i. In other words, one simultaneously carries out the ILU(0) factorization of all diagonal blocks of $\tilde{A}[n]$, obtaining the whole ILU(0) factorization of $\tilde{A}[n]$ *without communication* between the processors.

The same idea is used to solve linear systems associated with $M[n]$ (i.e. to multiply a vector by $M[n]^{-1}$), the main task done with the preconditioning matrix. For a given vector $\mathbf{v}$ the preconditioning step consists of solving for $\mathbf{w}$ the linear system associated with $M[n]$,

$$M[n]\mathbf{w} = \mathbf{v}.$$

Since the matrix $M[n]$ is given by (12) and the vectors $\mathbf{w}$ and $\mathbf{v}$ are given by (13), the parallelization is automatic. The ith processor ($i = 1, \ldots, n$) solves a linear system or order m_i,

$$M_i\mathbf{w}^i = \mathbf{v}^i, \tag{14}$$

and the solution of $\mathbf{w}$ is obtained *without communication* between the processors as

$$\mathbf{w} = \begin{pmatrix} \mathbf{w}^1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ \mathbf{w}^2 \\ \vdots \\ 0 \end{pmatrix} + \cdots + \begin{pmatrix} 0 \\ 0 \\ \vdots \\ \mathbf{w}^n \end{pmatrix}.$$

In fact, the blocks M_i ($i = 1, \ldots, n$) are never calculated. After performing the ILU(0) factorization of $\tilde{A}_i$, the lower and upper triangular factors defining M_i are stored as

$$M_i = L_iU_i, \quad i = 1, \ldots, n.$$

Thus the solution of (14) is no more than the solution of two triangular systems

$$L_i\mathbf{y}^i = \mathbf{v}^i, \qquad U_i\mathbf{w}^i = \mathbf{y}^i.$$

4.3.3. Parallelization of a matrix–vector product. Let $\mathbf{y} = A\mathbf{x}$, with A given by (8) and $\mathbf{y}$ and $\mathbf{x}$ given by (13). As already mentioned the matrix A has in general more than one non-zero block per row. Thus, to obtain $\mathbf{y}^i$, processor i needs to know $\mathbf{x}^j$ from the neighbouring processors,

$$\mathbf{y}^i = A^i \begin{pmatrix} \mathbf{x}^1 \\ \mathbf{x}^2 \\ \vdots \\ \mathbf{x}^n \end{pmatrix},$$

and corresponding communication needs to be established by the system. However, owing to the sparsity of the matrix A, this communication is of a limited nature on the order of the bandwidth.

On the other hand, the parallelization of the matrix–vector product for a distributed memory machine can be described as follows:

(i) upper triangle multiply: needs local $\mathbf{x}^i$, updates part of $\mathbf{y}^j$ of previous processor ($j < i$)
(ii) lower triangle multiply: needs part of $\mathbf{x}^j$ from previous processor ($j < i$), updates local $\mathbf{y}^i$
(iii) diagonal multiply: needs local $\mathbf{x}^i$, updated local $\mathbf{y}^i$.

4.3.4. Parallelization of inner product of vectors. The calculation of the inner product of two vectors $\mathbf{x}$ and $\mathbf{y}$ is equal to the sum of the inner products of their corresponding components and therefore can easily be parallelized in the usual way:

$$(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} (\mathbf{x}^i, \mathbf{y}^i). \tag{15}$$

The matrix–vector products and the inner product of vectors are the crucial steps for coupling the equations between processors. Of course, communication needs to be arranged in the two cases. First, one needs to communicate between adjacent processors a vector of the order of the bandwidth and, second, communication is needed only to add the partial results from each processor (one scalar per processor at the end of the operation).

5. NUMERICAL RESULTS

The computations are done in 64-bit precision on a Silicon Graphics IRIS 240 GTX with eight 25 MHz processors at the Concordia CFD Lab. The dimension of the Krylov space is 60 for all GMRES test cases. In order to compare the CGS and GMRES solvers, one iteration at each matrix–vector product is counted. For the GMRES method the required precision for the solver is attained in most cases before completing a Krylov iteration. In all test cases only one step of the hybrid artificial viscosity algorithm is carried out.

The tests are done on two grids of a 90° square bend (inlet area h^2; length upstream of bend, $5h$; length downstream of bend, $2h$; radius of bend, $2h$). Grid 1 has 8550 nodes and grid 2 has 15,300 nodes. A finite element surface grid for the bend is shown in Figure 5. The total number of equations considered is 27,084 for grid 1 and 49,044 for grid 2. The associated system matrices have about $2{\cdot}5 \times 10^6$ and $4{\cdot}6 \times 10^6$ non-zero coefficients respectively stored in 32-bit

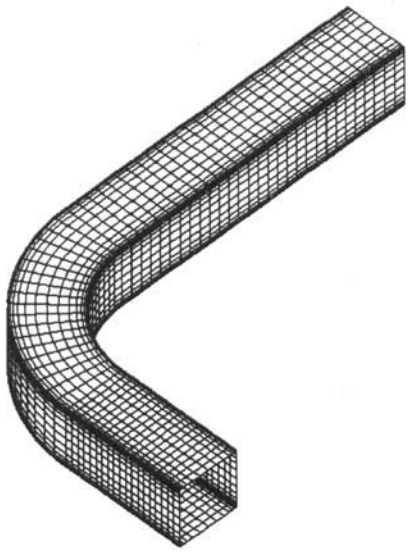

Figure 5. Geometry and surface grid on 90° bend

precision. In the validation of the approach only laminar cases have been considered up to now. The approach is tested out at $Re = 100$ and 1000. In the latter case one solves at $Re = 800$ as an initial solution.

The title of all the following figures is '*Convergence history of the Newton algorithm with different preconditioners*'. For each one the ordering (natural or MINNEIG), the solver (CGS or GMRES), the test case ($Re = 100$ or 1000) and the number of nodes (8550 or 15,300 nodes) used are indicated.

The numbers written beside each curve indicate how many blocks were used in the definition of the preconditioning matrix. When the term '*pfa*' (parallel FORTRAN accelerator) appears, the corresponding curve was obtained using a number of processors equal to the number of blocks. The remaining curves were obtained in a sequential manner using only one processor of the computer.

The unit used on the CPU axis of all these figures is seconds.

Figures 6–11 show the convergence histories of the Newton algorithm for grid 1 using different orderings of equations and the two iterative solvers. In Figures 12–15 the corresponding results on grid 2 are shown. Figures 6–9, 12 and 13 correspond to the first test case ($Re = 100$), while Figures 10, 11, 14 and 15 show the results obtained for $Re = 1000$, the second test case.

The results presented in Figures 6–15 concern the solver part of the hybrid artificial viscosity scheme only. In Tables I–III some timings are presented concerning the entire code, as well as the solver part. The solver speed-up is affected by two factors: the parallel efficiency of the code and the degradation of the non-linear Newton convergence because of the approximation of the decomposition of A by $M[n]$. In the following discussion the word 'resp.' in parentheses signifies 'respectively'.

Figures 6 and 7 show that with the original ordering of equations and the CGS algorithm the use of $M[8]$ can accelerate the solution step 2.6 times, while with the MINNEIG ordering and the CGS solver $M[8]$ is 2.9 times faster than $M[1]$. However, if one compares the total time to obtain the final solution, including assembly and factorization of the preconditioning matrix (not shown in Figures 6–15), $M[8]$ with the MINNEIG ordering (resp. natural ordering) and CGS solver is 4.6 (resp. 3.0) times faster than $M[1]$ with the same ordering of unknowns. The results obtained with the GMRES algorithm are similar but a little slower.

Figures 10 and 11 show the convergence of the Newton algorithm for the second test case ($Re = 1000$) on grid 1 with the GMRES algorithm. This test case is more difficult than the previous one and it can be seen that the new block preconditioners work well. From these figures $M[8]$ with the MINNEIG ordering (resp. natural ordering) is 4.0 (resp. 2.0) times faster than $M[1]$. Considering the total CPU time (and not only the CPU time spent in the solver step), $M[8]$ with the MINNEIG ordering (resp. natural ordering) is 4.6 (resp. 2.7) times faster than $M[1]$.

The results obtained using grid 2 are similar. Figure 13 (resp. Figure 15) shows that for the first test case $Re = 100$ (resp. $\mathrm{Re} = 1000$) the preconditioner $M[8]$ with the MINNEIG ordering and CGS algorithm is 4.6 (resp. 3.7) times faster than $M[1]$. Considering the total CPU time, the previous preconditioner is 6.1 (resp. 4.6) times faster than $M[1]$.

Tables I and II present the results obtained on grids 1 and 2 respectively with the parallelized code for all the test cases. For each preconditioner $M[n]$ the total execution time is indicated (including Jacobian construction, ILU(0) factorization of its diagonal blocks and matrix solution) and, in parentheses, the time spent in the solution of the linear system associated with the Newton iteration. To facilitate comparison, a different normalizing factor is used for the execution time in each test problem. In all cases the normalization factor is given by the time spent by $M[1]$.

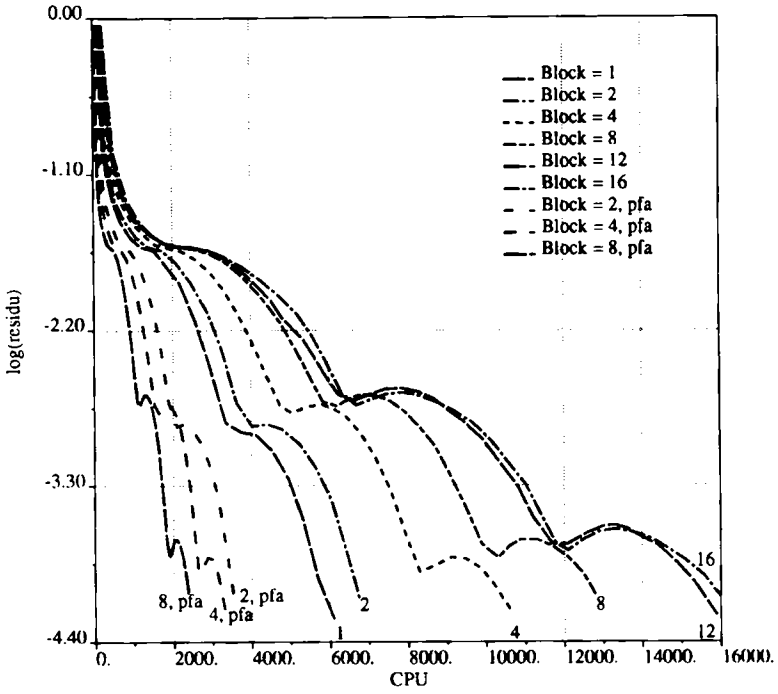

Figure 6. Natural ordering, CGS linear solver. $Re = 100$, 8550 nodes

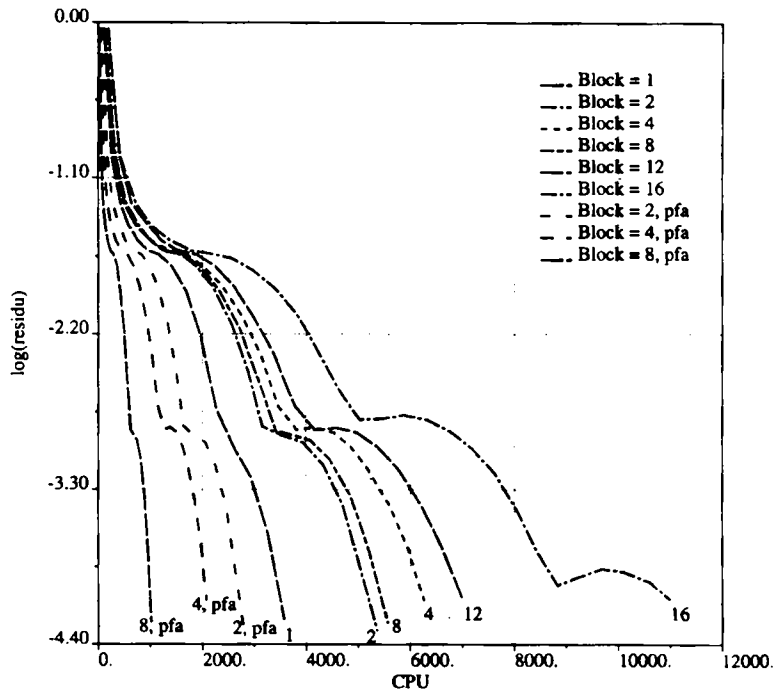

Figure 7. MINNEIG ordering, CGS linear solver. $Re = 100$, 8550 nodes

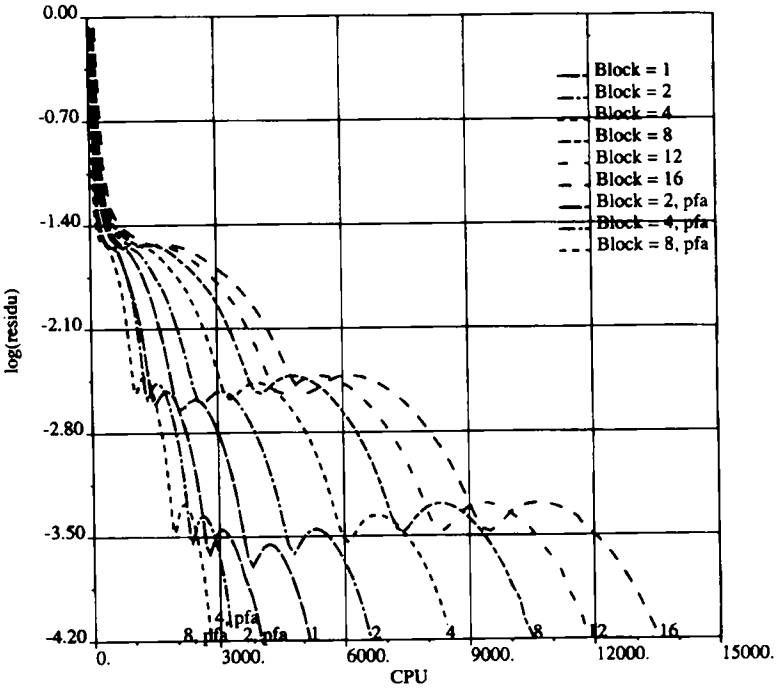

Figure 8. Natural ordering, GMRES linear solver. $Re = 100$, 8550 nodes

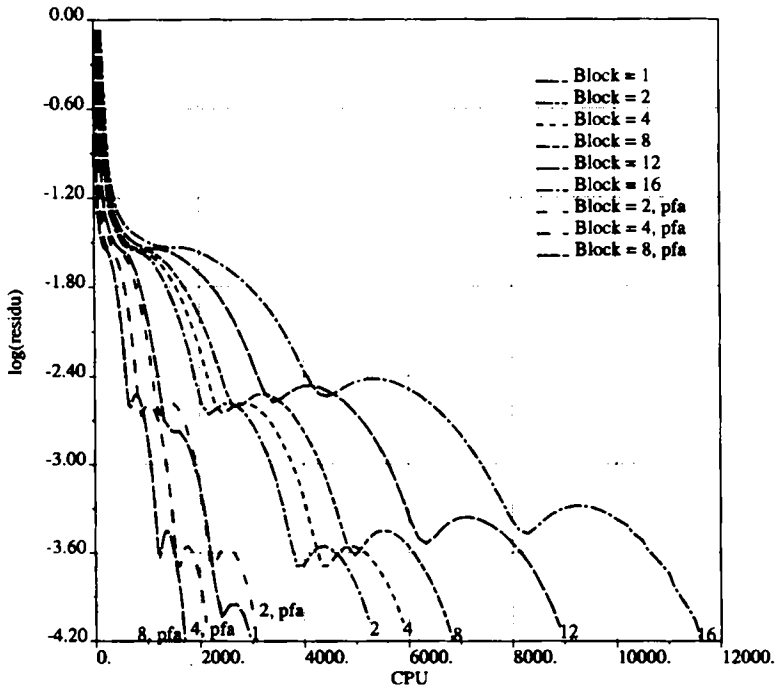

Figure 9. MINNEIG ordering, GMRES linear solver. $Re = 100$, 8550 nodes

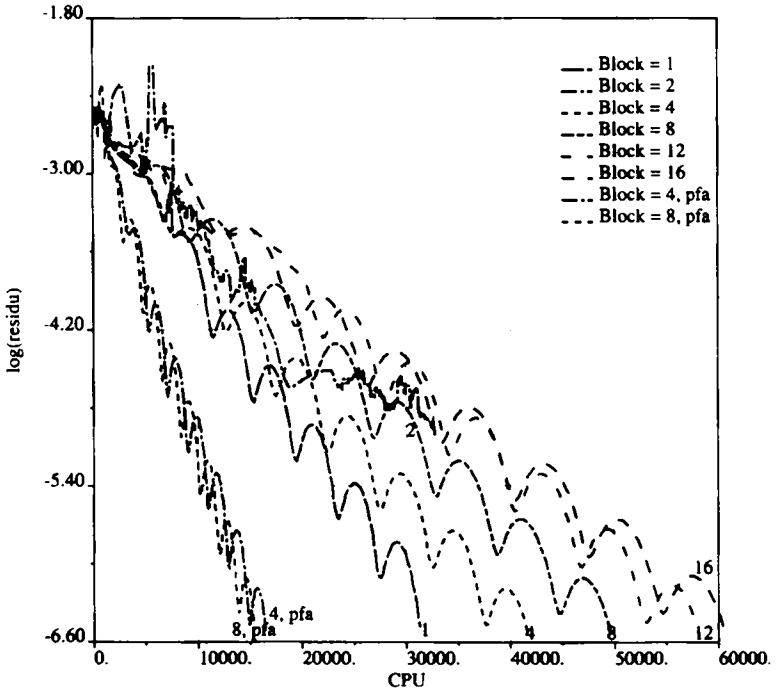

Figure 10. Natural ordering, GMRES linear solver. $Re = 1000$, 8550 nodes

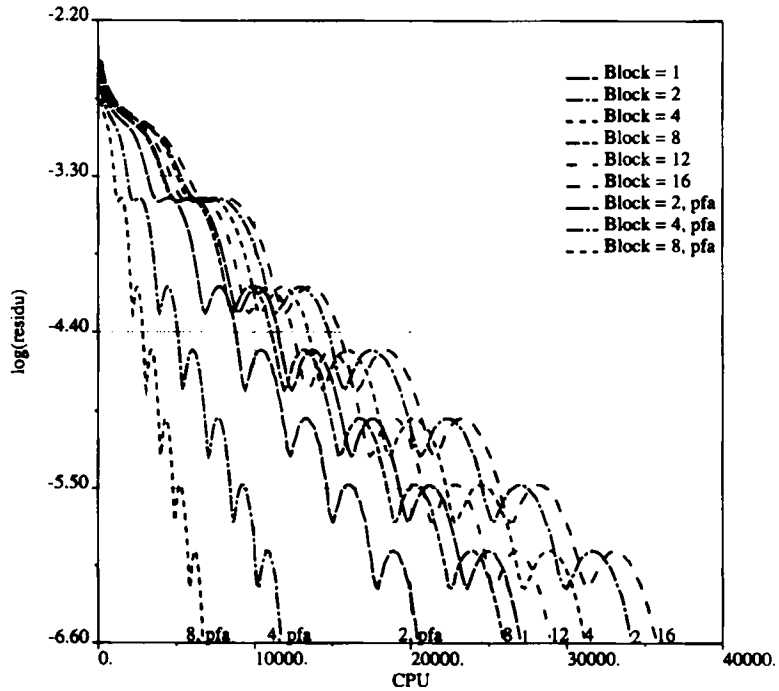

Figure 11. MINNEIG ordering, GMRES linear solver. $Re = 1000$, 8550 nodes

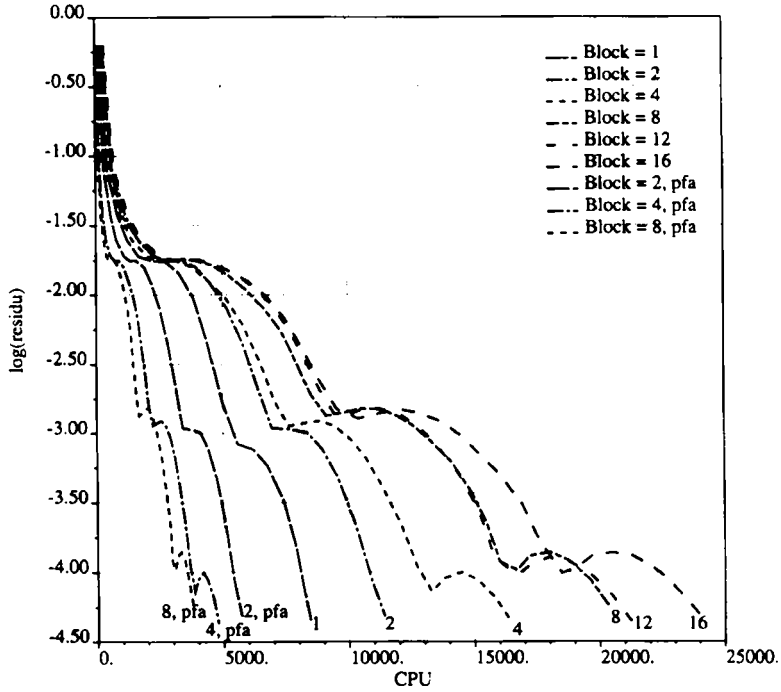

Figure 12. Natural ordering, CGS linear solver. $Re = 100$, 15,300 nodes

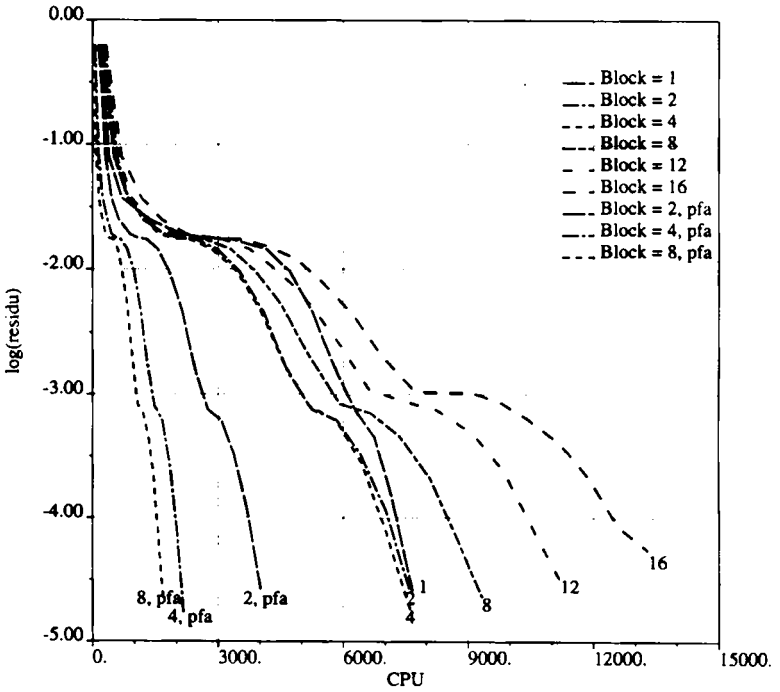

Figure 13. MINNEIG ordering, CGS linear solver. $Re = 100$, 15,300 nodes

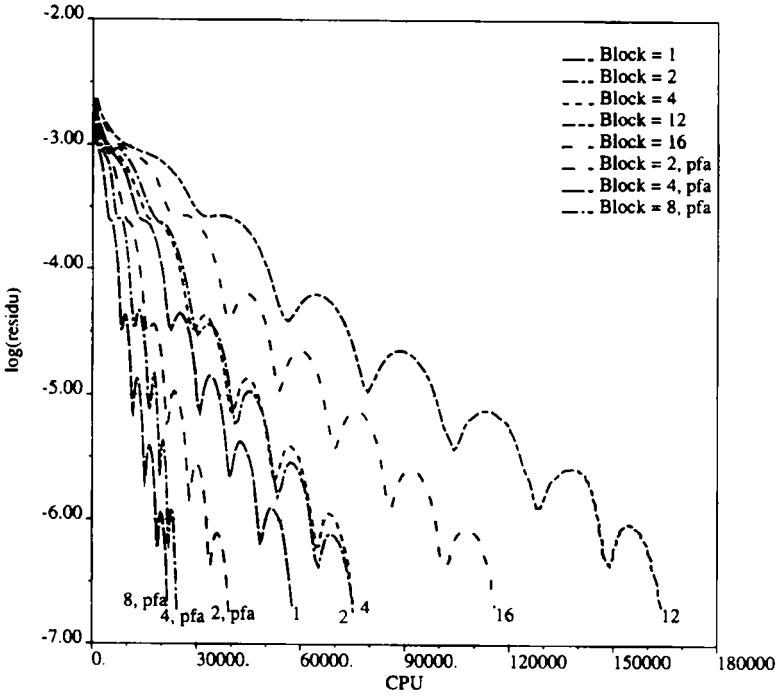

Figure 14. Natural ordering, CGS linear solver. $Re = 1000$, 15,300 nodes

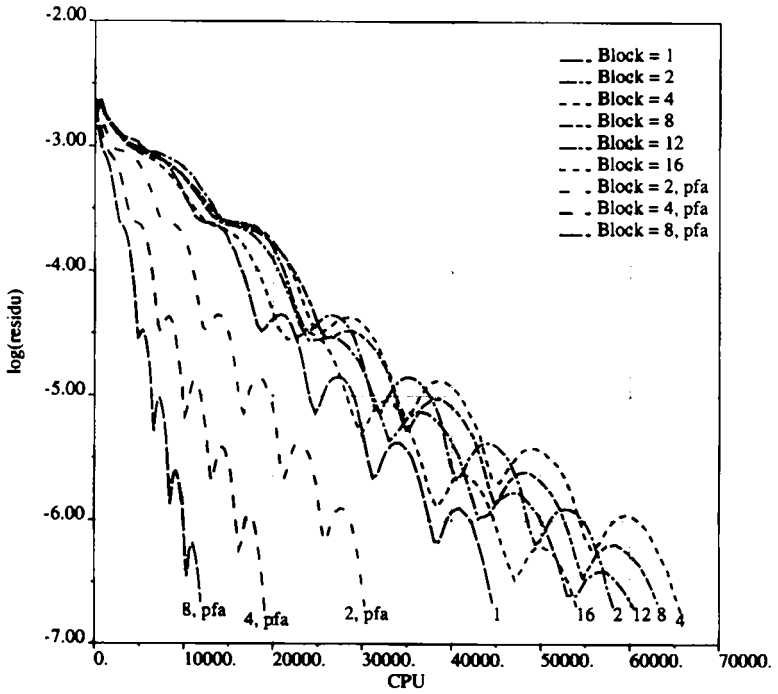

Figure 15. MINNEIG ordering, CGS linear solver. $Re = 1000$, 15,300 nodes

Table I. Normalized total execution time (and normalized time spent by the iterative solvers) for various test cases on grid 1

Re	Solver	Ordering	$M[1]$	$M[2]$	$M[4]$	$M[8]$
100	CGS	Natural	1·0 (1·0)	0·55 (0·58)	0·47 (0·54)	0·33 (0·39)
		MINNEIG	1·0 (1·0)	0·63 (0·78)	0·41 (0·58)	0·22 (0·35)
100	GMRES	Natural	1·0 (1·0)	0·72 (0·78)	0·53 (0·63)	0·42 (0·54)
		MINNEIG	1·0 (1·0)	0·81 (1·03)	0·51 (0·72)	0·38 (0·58)
1000	GMRES	Natural	1·0 (1·0)	Failed*	0·45 (0·52)	0·37 (0·49)
		MINNEIG	1·0 (1·0)	0·68 (0·76)	0·39 (0·44)	0·22 (0·25)

Table II. Normalized total execution time (and normalized time spent by the iterative solvers) for various test cases on grid 2

Re	Solver	Ordering	$M[1]$	$M[2]$	$M[4]$	$M[8]$
100	CGS	Natural	1·0 (1·0)	0·60 (0·68)	0·45 (0·57)	0·33 (0·45)
		MINNEIG	1·0 (1·0)	0·49 (0·53)	0·26 (0·28)	0·16 (0·22)
1000	GMRES	Natural	1·0 (1·0)	0·63 (0·68)	0·35 (0·38)	0·36 (0·43)
		MINNEIG	1·0 (1·0)	0·63 (0·68)	0·34 (0·43)	0·22 (0·27)

Table III. Total number of seconds and normalized time spent by the iterative solvers for $Re = 100$ on grid 1

	CGS		GMRES		CGS/GMRES	
Preconditioner	Natural	MINNEIG	Natural	MINNEIG	Natural	MINNEIG
$M[1]$	6120	3580	5130	2950	1·19	1·21
$M[8]$	12800	5570	10500	6850	1·22	0·81
$M[16]$	16000	11000	13500	11600	1·19	0·95

The results associated with the preconditioner $M[n]$ were obtained using n processors of the computer.

A comment is necessary to qualify the 'failure' of GMRES preconditioned by $M[2]$ with the natural ordering for grid 1 and $Re = 1000$. In fact, the algorithm converges but does not attain the prescribed precision within the maximum number of iterations allowed. The curve of convergence is also rather irregular. This phenomenon is all the more surprising since the algorithm works well when preconditioned by $M[4]$ and $M[8]$. However, it must be kept in mind that the matrices $\tilde{A}[n]$ are constructed independently in a sequential manner and it is not guaranteed that each block of nodes of $\tilde{A}[2]$ contains exactly two blocks of nodes from $\tilde{A}[4]$ and so on. The only thing that can be ensured is that the first group of nodes of $\tilde{A}[8]$ is inside the first one of $\tilde{A}[4]$ and the latter is in the first group of nodes of $\tilde{A}[2]$. Thus it is possible that in this particular test some important connectivities were preserved on $\tilde{A}[4]$ and $\tilde{A}[8]$ but were lost on $\tilde{A}[2]$, producing its loss of efficiency. This phenomenon, however, was isolated and did not repeat itself with the other test cases.

From Figures 6–8 one can compare the relative efficiency of the CGS and GMRES algorithms at $Re = 100$ on grid 1. The results are summarized in Table III. The CPU times spent in solving this test with the natural and MINNEIG orderings are compared using $M[1]$. $M[8]$ and $M[16]$ as preconditioners. The first column shows the number of seconds spent by CGS in solving the Newton algorithm, the second column the CPU time associated with GMRES and the third column the relative speed of CGS with respect to GMRES. All the results presented in this table were obtained using only one processor of the computer. To facilitate comparison, the time spent by GMRES is taken as a normalization factor in each case, in the third column. One can see that GMRES is faster than CGS when the natural ordering is used and even sometimes when the MINNEIG ordering is used.

The relative importance of choosing a good ordering of unknowns is illustrated. For $Re = 100$ it is found that for grid 1, $M[1]$ + MINNEIG ordering is 1.7 times faster than $M[1]$ + natural ordering for solving the problem with both GMRES and CGS algorithms. At the same Reynolds number for grid 2, $M[1]$ + MINNEIG ordering with the CGS algorithm is 1.1 times faster than $M[1]$ + natural ordering. For $Re = 1000$ $M[1]$ + MINNEIG ordering is 1.2 times faster than $M[1]$ + natural ordering using the GMRES algorithm on grid 1 and the CGS algorithm on grid 2. It can therefore be concluded that even for structured grids an efficient reordering of the nodes produces an interesting reduction in solution time. For a general (unstructured) grid, reordering of the unknowns is expected to have a more important effect.

6. CONCLUSIONS

In this paper a parallelizable hybrid artificial viscosity scheme applied to solve viscous compressible 3D flows using finite element methods is presented. A good parallelizable block-diagonal preconditioner is developed in order to accelerate the solution of the linear system associated with the Jacobian matrix at each Newton iteration. The preconditioner presented in this paper depends only on the matrix defining the linear system to solve. It could be interesting to apply it to other linear problems arising from computational fluid dynamics. To build the preconditioning matrix $M[n]$, the system matrix A is partitioned into n blocks using a simple matrix-structure-dependent method, n being the number of available processors. In contrast with the more complex domain decomposition methods, parallelization is achieved by simply restricting the overall system matrix to its diagonal blocks and only for the purpose of the preconditioner.

In the numerical tests it has been shown that for a moderate number of processors (n being $O(10)$) it is possible to choose an appropriate ordering algorithm giving a good rate of convergence of both preconditioned GMRES and CGS algorithms. Even after accounting for the loss of efficiency of the new preconditioner $M[n]$ when $n > 1$ in comparison with $M[1]$, it is shown that it is possible to converge up to six times faster on an eight-processor system with $M[8]$ than on a single processor using the usual ILU(0) factorization of the matrix $A = M[1]$. This speed, coupled with the memory storage advantages of iterative methods, makes them competitive on parallel workstations with direct methods on supercomputers.

ACKNOWLEDGEMENTS

The support of NSERC (Natural Sciences and Engineering Research Council of Canada), Silicon Graphics Inc., Pratt & Whitney Canada and CERCA (Centre for Research on Computation and its Applications) is gratefully acknowledged.

REFERENCES

1. Y. Saad and M. H. Schultz, 'GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems', *SIAM J. Sci. Stat. Comput.*, **7**, 856–869 (1986).
2. P. Sonneveld, 'CGS: a fast Lanczos-type solver for nonsymmetric linear systems', *SIAM J. Sci. Stat. Comput.*, **20**, 36–52 (1989).
3. O. Axelsson, S. Brinkkemper and V. P. Il'in, 'On some versions of incomplete block-matrix factorization iterative methods', *Linear Algebra Appl*, **58**, 3–15 (1984).
4. J. A. Meijerink and H. A. van der Vorst, 'An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix', *Math. Comput.*, **31**, 148–162 (1977).
5. T. E. Tezduyar, J. Liou, T. Nguyen and S. Poole, 'Adaptive implicit-explicit and parallel element-by-element factorization schemes', in T. F. Chan *et al.* (eds), *Domain Decomposition Methods*, SIAM, Philadelphia, PA 1989, pp. 443–463
6. M. F. Peeters, W. G. Habashi, B. Q. Nguyen and P. L. Kotiuga, 'Finite element solutions of the Navier–Stokes equations for compressible internal flows', *AIAA J. Power Propuls.* **8**, 192–198 (1992).
7. W. G. Habashi, M. Robichaud, V.-N. Nguyen, W. S. Ghaly, M. Fortin and J. W. H. Liu, 'Large-scale computational fluid dynamics by the finite element method', to appear, *Int. j. numer. methods fluids* (1994).
8. I. S. Duff and G. A. Meurant, 'The effect of ordering on preconditioned conjugate gradients'. *BIT*, **29**, 635–657 (1989).
9. L. C. Dutto, 'The effect of ordering on preconditioned GMRES algorithms, for solving the compressible Navier–Stokes equations', *Int. j. numer. methods eng.*, **36**, 457–497 (1993).
10. G. Martin, 'Méthodes de préconditionnement par factorisation incomplète', *Mémoire de Maîtrise*, Département de Mathématiques et de Statistique, Université Laval, Québec (1991).