INT. J. NUM. METH. HEAT FLUID FLOW, VOL. 3, 3–14 (1993)

AN ALGEBRAIC MULTIGRID SOLVER FOR THE NAVIER–STOKES EQUATIONS ON UNSTRUCTURED MESHES

R. D. LONSDALE

Computational Fluid Dynamics Services, AEA Technology, Harwell Laboratory, Didcot, Oxfordshire, OX11 0RA, UK

ABSTRACT

A simple algebraic multigrid (AMG) solver for linear equations is presented, and its performance compared with a conjugate gradient scheme. This multigrid method is extended to solve the discrete Navier–Stokes equations, obtained by applying a finite volume approach to three-dimensional incompressible flow on a finite element mesh. The resulting multigrid solver is incorporated into a general purpose flow code (ASTEC), where it proves faster than the original solution algorithm, based upon SIMPLE. The linear AMG solver is both efficient and robust, but the extension to include coupling in the Navier–Stokes equations does not converge on all problems.

KEY WORDS Multigrid Unstructured mesh Navier–Stokes SIMPLE Incompressible flow

INTRODUCTION

In recent years, much effort has been devoted to developing multigrid solution algorithms[1]. This is because the multigrid approach offers the prospect of excellent scaling with problem size. Specifically, the CPU time should only increase in proportion to the number of unknowns in the equations. Other solution strategies (for example, Gaussian elimination or conjugate gradient methods) suffer from worse than linear scaling with problem size, so that as computing power continues to increase, enabling larger problems to be tackled, their performance becomes relatively much poorer.

The aim of this paper is to present a simple algebraic multigrid (AMG) algorithm, and show how this can be used to solve either linear equations, or the discretized Navier–Stokes equations, on an unstructured (finite element) mesh. The scheme is relatively economical on computer storage and CPU time, both scaling almost linearly with problem size.

It is important to note that on fully *unstructured* meshes it is not possible to use the most efficient iterative linear solvers, such as Stone's method[2] or incomplete Cholesky conjugate gradients[3], since these rely on there being some structure in the mesh. We are limited to diagonally preconditioned conjugate gradients for symmetric matrices, and little better than Gauss–Seidel for unsymmetric matrices. Thus a fast multigrid solver, for linear equations from unstructured meshes, is a valuable tool.

First, it is necessary to briefly explain the ideas behind algebraic multigrid. Multigrid schemes rely upon using a hierarchy of grids (from fine to coarse) to solve a set of discrete equations, with each grid proving particularly effective for removing errors of wavelength characteristic of the mesh spacing on that grid. Often the hierarchy of grids is chosen *a priori*, making use of the

0961–5539/93/010003–12$2.00

Received April 1992
Revised October 1992

known structure of the original (finest) grid—this is geometric multigrid. However, discretization on an unstructured or finite element mesh demands a multigrid solver capable of automatically generating its own hierarchy of coarser grids. Algebraic multigrid solvers do just this, making use only of the equation matrix itself to create the corase grids.

It is important to realise that the algebraic multigrid approach is not just a way of obtaining geometric multigrid performance on unstructured meshes. Because the grid coarsening is not *blind*, the algebraic multigrid solver should actually prove better than a geometric multigrid solver, even where the latter can be employed, on *difficult* highly anisotropic problems.

A simple but effective algebraic multigrid algorithm is developed for solving a set of linear equations. The performance of this algorithm is evaluated next, where almost linear scaling with problem size is demonstrated, and the effectiveness of AMG on a nasty problem is proven. Results from a conjugate gradient solver are also presented for comparison.

The ideas developed here are used later to produce an algebraic multigrid solver for the Navier–Stokes equations, discretized on an unstructured mesh. The performance of this Navier–Stokes solver is evaluated at the end of the paper.

BASIC ALGEBRAIC MULTIGRID SCHEME

The multigrid scheme

Consider the linear equation set:

$$\mathbf{A}x = b \tag{1}$$

where $\mathbf{A}$ is a sparse $N \times N$ matrix, with the diagonal entry at least the size of the sum of the absolute values of the off-diagonal entries, for every row. Given a latest guess for x, x_l, it is possible to improve upon this by using point-by-point Gauss–Seidel sweeps. This method reduces short wavelength errors very effectively, but it is necessary to obtain a coarse grid representation of $\mathbf{A}$ in order to use Gauss–Seidel for removing long wavelength errors.

Assume that there is a coarsening $M \times N$ matrix, $\mathbf{K}$, which can be used to create a coarse $M \times M$ (M being less than N) representation of (1):

$$\mathbf{A}^c x^c = b^c \tag{2}$$

where

$$\mathbf{A}^c = \mathbf{K}\mathbf{A}\mathbf{K}^{\mathrm{T}} \tag{3}$$

Obviously, there are N points on the original (fine) grid, and M points on the new (coarse) grid. The construction of $\mathbf{K}$ will be described later. Meanwhile, note that (2) will be used to provide a correction to x_l, so the right hand side is derived from the residual in (1):

$$b^c = \mathbf{K}(b - \mathbf{A}x_l) \tag{4}$$

Having approximately solved (2), to obtain x_l^c, an improved solution for x on the fine grid is obtained:

$$x_{ll} = x_l + K^{\mathrm{T}} x_l^c \tag{5}$$

The multigrid strategy now becomes clear. Given (1), a fine-to-coarse grid mapping $\mathbf{K}$ is constructed, and used to produce a coarse grid equation for corrections. The right hand side of this coarse grid equation is derived from the residuals in the original fine grid equation corresponding to the latest guess for the solution. Having approximately solved the coarse grid

equation, the coarse grid solution is used to correct that on the fine grid. The fine grid solution is further improved by applying a smoothing scheme, such as Gauss–Seidel sweeps, to remove errors of a wavelength associated with the fine grid.

The algorithm becomes recursive, in that the coarse grid equation is itself solved by creating a still coarser grid, and so on down until the grids can be coarsened no further. Thus a single cycle of the multigrid solver consists of moving through the grids from finest to coarsest, passing residuals down at each stage, and then sweeping to improve the solution on each grid, now moving from coarsest to finest, while passing corrections up at each stage. This is called a V-cycle.

It has been found that the scheme described in this paper actually performs better when a slightly more complex cycling strategy is used, as illustrated in *Figure 1*. This consists of first passing the residuals down to all grids, and then solving on each coarse grid in turn, from coarsest to finest, by applying a V-cycle as described above, followed by a single Gauss–Seidel sweep. This may be called a F-cycle.

Finally note that this method in no way relies upon the matrix A being symmetric.

The coarsening algorithm

The heart of the algebraic multigrid solver is the construction of the coarsening matrix K for each grid. The construction of the coarse grid will be described geometrically, after which the form of K should become clear.

A typical two-dimensional fine grid is represented in *Figure 2*. There is a discrete equation associated with each node, where the value of the variable at the node is related to the values at all connected nodes. Coarse grid nodes are created by lumping together nodes from the fine grid, as illustrated in *Figure 2*. Note that this contrasts with the approach adopted in earlier algebraic multigrid work[4,5] where the coarse grid is a subset of the fine grid. The equation for a node on the coarse grid is obtained by simply adding the fine grid equations from the contributing fine grid nodes. These equations are in terms of the fine grid nodes, but they are easily translated into coarse grid terms by replacing any reference to a fine grid node by a reference to the coarse grid node to which it belongs.

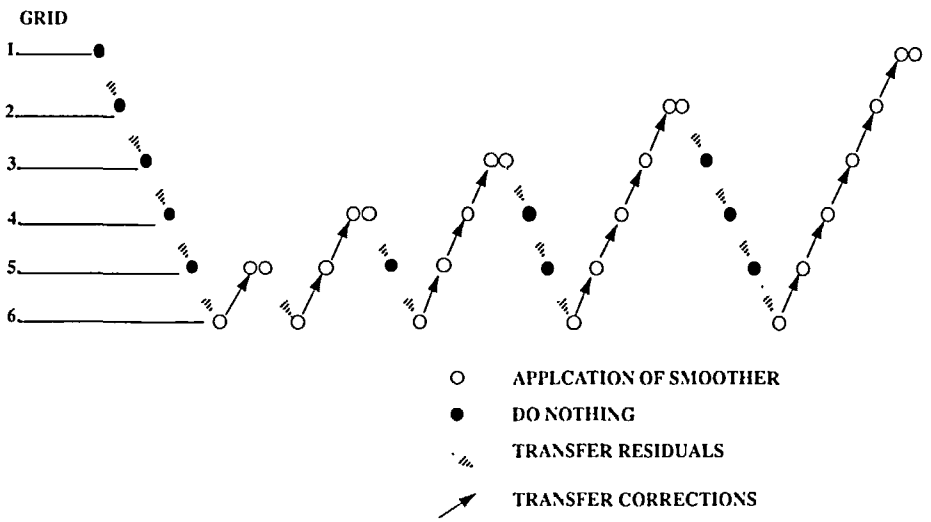

Figure 1 Multigrid cycling strategy

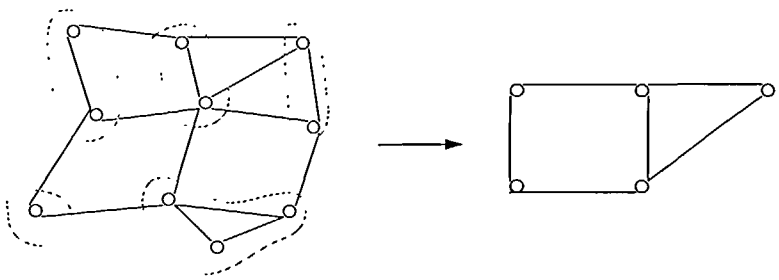

Figure 2 Grid coarsening process

The matrix **K** performs the operation of adding fine grid equations to produce coarse grid equations, and $\mathbf{K}^T$ translates fine grid references into appropriate coarse grid references. Thus given a fine grid matrix **A**, the matrix $\mathbf{KAK}^T$ gives a coarse grid representation. Suppose the fine grid contains N nodes and the coarse grid M nodes, then **K** is a $M \times N$ matrix where the entry K_{ij} contains a 1 if the jth fine grid node is contained in the ith coarse grid node, and a 0 otherwise.

Note that it is important to have a sensible strategy for deciding which fine grid nodes should be joined to create coarse grid nodes. Generally, it is necessary to join together nodes which are strongly connected, in the following sense: nodes a and b are strongly connected if the coefficient of node b in the equation for node a is relatively large, and/or *vice versa.*

The coarsening algorithm used in this work operates as follows. All fine grid nodes are ordered according to the strength of their strongest connections, that is according to the maximum value of $|a_{ij}/a_{ii}|$ as j varies, where $j \neq i$ and the equation for node i is:

$$a_{ii}x_i + \sum a_{ij}x_j = b_i \tag{6}$$

the sum being over j. This list of nodes is processed once, starting with the most strongly connected node. Each node is joined to that neighbour to which it is most strongly connected to create a new coarse grid node, provided that neither of the two fine grid nodes has already been incorporated into a coarse grid node. Then the list of fine grid nodes is subjected to a second pass, again starting with the most strongly connected node. Any remaining fine grid node which does not already belong to a coarse grid node is now attached to the coarse grid node which *owns* the neighbour to which the fine grid node is most strongly connected. This second pass does not process the whole node list, but is stopped once the coarse grid reaches a particular size. Any remaining unallocated fine grid node is assigned to be a coarse grid node in its own right. Note that the reason for having two passes through the mesh, with the first pass unable to attach a fine grid node to an existing coarse grid node, is to avoid a whole string of fine grid nodes being collected into a single coarse grid node.

This algorithm creates a coarse grid where each node is made up of an arbitrary number of fine grid nodes. It is used recursively to generate a hierarchy of grids.

Each coarse grid has approximately half the number of nodes of the previous (fine) grid, and the coarsening stops once all nodes on the coarsest grid are isolated from each other (that is, all off-diagonal entries are zero in the equation matrix for the coarsest grid). The coarsening algorithm is simple and cheap, and takes due account of the strength of inter-node connections. Note that since it is quite acceptable to have the node list only approximately ordered according to connection strengths, a cheap approximate ordering algorithm is used, and the computer time required for the complete coarsening algorithm scales only linearly with the number of fine grid nodes.

PERFORMANCE OF THE BASIC SCHEME

The primary aim of this section is to demonstrate that, for a simple problem at least, the algebraic multigrid scheme gives almost linear scaling with mesh size. A very simple test problem was constructed, consisting of solving the discretized heat conduction equation to determine the temperature distribution in a cube. The surface of the cube is given a zero heat flux boundary condition, except that the temperature is fixed at two opposite corners of the cube, to be 0 and 1 respectively. Five different test computations are performed, on five different finite element meshes using only 8-node linear brick elements, but consisting of between 1000 and 64,000 nodes. *Figure 3* shows, as a function of mesh size (number of nodes), the CPU time in seconds required by the multigrid scheme to solve the conduction equation, starting with an initial guess of zero temperature everywhere and reducing the residuals to near round-off error on a Cray-2 computer. This CPU time includes the initialization time, required to create the hierarchy of coarse grids.

It appears that the CPU time scales approximately as the number of nodes to the power 1.1, but the points do not lie precisely on a straight line in the log-log plot.

In another set of tests, the linear AMG solver has been compared to a diagonally preconditioned conjugate gradient (CG) solver, on 3 different computers, and on 2 different problems. The application consists of solving a discretized Poisson equation for potential flow along a square duct which undergoes a right-angled bend. This represents the pressure-correction stage of one iteration within an iterative Navier–Stokes solver, and since tight convergence is rarely required for this pressure-correction step, we only demand a 10^{-3} reduction in the sum of absolute residuals for convergence. The computational cells in the first problem had an aspect ratio of approximately 2, while in the second problem the legs of the duct were made much longer (without increasing the number of mesh cells) so increasing the aspect ratio to approximately 40. Each problem was solved with various numbers of mesh cells, always keeping the aspect ratios constant, using both AMG and CG solvers on a Cray-2. *Tables 1* and *2* give the recorded CPU times (in secs) for problems 1 and 2 respectively, along with an estimate of scaling with mesh size which was derived from these figures.

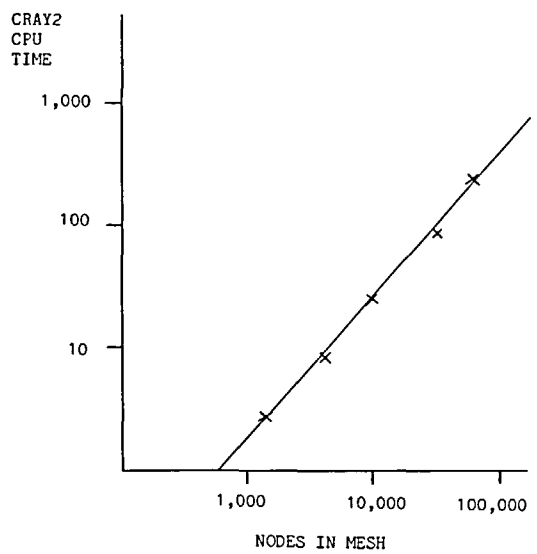

Figure 3 CPU time vs. mesh size for AMG solver

Table 1 CPU time vs. mesh size for first problem

Mesh size (N)	AMG time	CG time
288	0 22	0.04
2304	1.96	0.42
18,432	22.49	6.27
36,000	52.86	12.85
Scaling	$N^{1.16}$	$N^{1.19}$

Table 2 CPU time vs. mesh size for second problem (high aspect ratio)

Mesh size (N)	AMG time	CG time
288	0 20	0 09
2304	1.31	1.89
18,432	14 97	53.45
36,000	36.43	137.47
Scaling	$N^{1\ 12}$	$N^{1.52}$

Table 3 CPU time on various machines for first problem

Computer	AMG time	CG time
Cray-2	1.96	0.42
IBM RS-6000/320	1.46	2.77
SUN Sparc-IPC	6 84	11.34

Most striking is the severe degradation of the conjugate gradient scheme on the high aspect ratio problem—diagonal preconditioning alone is clearly inadequate. In contrast, the multigrid solver actually performs *better* on the (harder) high aspect ratio problem than multigrid on the first problem.

When the mesh cells have a high aspect ratio, the AMG coarsening scheme automatically joins together nodes across the grain of the mesh, so creating coarse grids with lower aspect ratios which can more effectively propagate information down the length of the mesh. Perhaps the high degree of anisotropy in the second problem actually helps the coarsening algorithm to create more effective coarse grids than are produced for the first problem—hence the improvement in performance.

The conjugate gradient scheme vectorizes very well, while the AMG solver is not amenable to vectorization, and this means that the CG scheme performs relatively much better on a Cray-2 as compared with a desktop workstation. This is illustrated in *Table 3*, which gives timings for problem 1 with a mesh of 2304 cells on 3 different machines.

Even on the *easy* problem AMG outperforms CG on both workstations, and the multigrid solver actually runs faster on the RS-6000 than it does on the Cray-2!

This linear AMG solver has been applied to many other problems, and has generally proved very robust.

MULTIGRID FOR THE NAVIER–STOKES EQUATIONS

The discrete equations

The method proposed in this section could be applied to any Navier–Stokes discretization based on primitive variables, which is amenable to solution by an iterative method such as SIMPLE[6]. However, the method will actually be described in the context of the partly staggered mesh approach used in the ASTEC code[7], where pressures are stored at the centres of 8-node linear elements, and all other variables, including velocity components, are stored at the corner nodes[8].

The Navier–Stokes equations for velocity $u = (u_1, u_2, u_3)$ and pressure p are:

$$\frac{\partial}{\partial t}\rho u_i + \frac{\partial}{\partial x_j}\rho u_i u_j - \frac{\partial}{\partial x_j}\mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) + \nabla p = \mathbf{F} \tag{7}$$

$$\frac{\partial}{\partial x_j}\rho u_j = -\frac{\partial \rho}{\partial t} \tag{8}$$

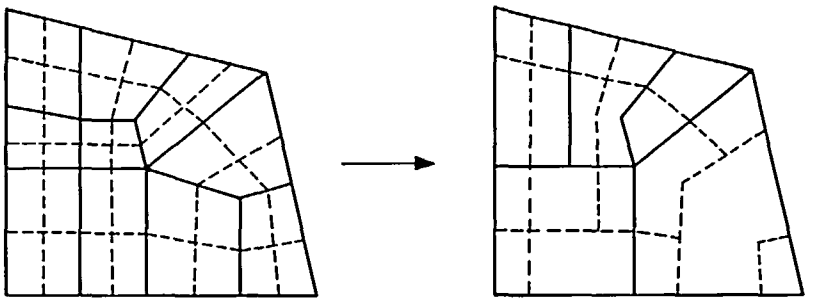

Figure 4 ASTEC mesh arrangement in two dimensions, with illustration of mesh coarsening

where ρ is the fluid density, μ the dynamic viscosity and **F** an arbitrary body force. Equation (7) applies for each velocity component $i = 1, 2, 3$, and the summation convention is employed for index j.

The discrete equations are derived by applying a finite volume approach to a finite element mesh of 8-node cuboidal elements. The left hand picture in *Figure 4* illustrates how the control volumes are constructed around each node for quadrilateral elements in 2 dimensions. The solid lines surround elements, and there is a node at each corner of each element (wherever solid lines meet). The control volume around each node is contained within the broken lines, which simply join mid-points of opposite sides within each element. The generalization to cuboidal elements in 3 dimensions is straightforward.

A proportion of upwind differencing is used for representing advection, that proportion depending upon the local mesh Peclet number. The discretization scheme is fully implicit.

Now the linearized discrete Navier–Stokes equations can be written as:

$$\mathbf{A}u + \mathbf{G}p = a \tag{9}$$

$$\mathbf{C}u + \Omega(\mathbf{CNG} - \mathbf{D})p = b \tag{10}$$

with u now a vector of length $3 \times$ *number of nodes* representing the discrete velocity field in 3 dimensions, and p a vector of length *number of elements* representing the discrete pressure field.

Equation (9) is the discretized momentum equation for velocity u, with the operator **G** mapping the element pressure field p into a pressure gradient vector for each node. The matrix **A** arises from the transient, advection and diffusion terms of (7), with **C**u (10) representing $(\partial/\partial x_j)\rho u_j$ in (8). General source terms (including **F** or $\partial\rho/\partial t$) are represented by a and b.

Equation (10), the discretized continuity equation, is written in an unfamiliar form in order to illustrate the pressure-correction solution method. The terms involving pressure have no physical basis—they are included only for numerical reasons. **D** is the operator used to determine pressure corrections in the solution procedure, as will become clear later, and **N** is the inverse of the diagonal of **A**. It would be nice to have **D** = **CNG** but in order to avoid spurious checker-board oscillations in the pressure solution, arising because of this mesh arrangement[8], it is necessary to choose a different form[7] for **D**. **CNG** represents a Laplacian-like operator, and **D** is simply an alternative representation of this operator which does not allow checker-board solutions.

To illustrate the difference between **CNG** and **D**, consider a two-dimensional rectilinear mesh of quadrilateral cells. The discrete representation of **CNG**, for the cell at location (I, J), will involve the neighbouring cells at locations (I + 1, J + 1), (I + 1, J − 1), (I − 1, J + 1) and (I − 1, J − 1). However, the discrete representation of **D** at location (I, J) would involve

neighbouring cells (I + 1, J), (I − 1, J), (I, J + 1) and (I, J − 1). It is well known that the former discretization[8] allows checker-board solutions, while the latter does not.

Because of these extra terms, the nodal velocities do not exactly satisfy continuity $\mathbf{C}u = b$, although the error $\Omega(\mathbf{CNG} - \mathbf{D})p$ is second order in the mesh size, and generally proves acceptably small.

The pressure-correction scheme

Before discussing the multigrid approach, it is worth describing how a pressure-correction scheme such as SIMPLE would be used to solve (9) and (10). There follows a summary of one SIMPLE iteration.

Let u^0, p^0 be initial guesses for u and p. Then (9) is used to obtain an improved velocity solution u^1, by solving the linear equation:

$$\mathbf{A}u^1 = -\mathbf{G}p^0 + a \tag{11}$$

Since **A** is a diagonally dominant matrix, an iterative point by point method such as Gauss–Seidel can be used. These corrections to u, obtained by solving (9), are usually under-relaxed with an under-relaxation factor ω_u, thus the latest velocity field is actually taken to be:

$$u^2 = (1 - \omega_u)u^0 + \omega_u u^1 \tag{12}$$

Having used the momentum equation to improve the velocity solution, the continuity equation is used to improve the pressure solution. The new pressure field p^1 is obtained by solving:

$$\Omega\mathbf{D}p^1 = \mathbf{C}u^2 + \Omega\mathbf{CNG}p^0 - b \tag{13}$$

$\Omega = \omega_u/\omega_p$ where ω_p is effectively an under-relaxation factor for pressure corrections. Provided **D** is a diagonally dominant matrix, an iterative linear solver can be used to obtain p^1 from (13).

To ensure that the pressure corrections arising from (13) do not later produce a large increase in the residuals for (9), the velocity field is also corrected according to

$$u^3 - u^2 = \delta u = -\Omega\mathbf{NG}\delta p \tag{14}$$

where $\delta p = p^1 - p^0$. The relationship between δu and δp given by (14) is chosen so as to leave the right hand side of (13) unchanged when u^2 and p^0 are replaced by u^3 and p^1 respectively.

u^3 and p^1 represent the latest guesses for velocity and pressure after one iteration of the SIMPLE method, and these would be provided as u^0 and p^0 for the next iteration. A converged solution can be obtained after many iterations providing suitable values are chosen for the relaxation factors ω_u and ω_p. The method is relatively robust, with the majority of problems converging for values of $\omega_u = 0.6$ and $\omega_p = 0.4$.

Note that after each iteration, non-linearities are accounted for by modifying A, and perhaps a and b. The method can be improved by using more sophisticated linear solvers to calculate the momentum and continuity corrections to velocity and pressure.

However, the SIMPLE method scales poorly with mesh size as regards CPU time.

Application of multigrid

It is not sufficient to apply the multigrid method separately to the momentum correction and continuity correction steps of each SIMPLE iteration. Although this gives multigrid performance to the linear equation solvers within each iteration, the number of iterations required to obtain convergence still increases with mesh size. To obtain multigrid performance, the coupled equations (9) and (10) must be subjected to multigrid together.

In order to avoid having to discretize on the coarse grids, the multigrid method is used to solve only the linearized Navier–Stokes equations. So this is *not* a full approximation scheme, although the velocity–pressure coupling is treated implicitly on the coarse grids. For Stokes flow, where the advection terms are neglected, just one application of the multigrid solver would give the final solution. However, because the non-linear advection terms are linearized, several 'outer iterations' are usually required to obtain convergence. Each outer iteration consists of solving the linearised Navier–Stokes equations using the multigrid scheme.

A hierarchy of grids is produced, on each of which there is a representation of (9) and (10), and on each of which the SIMPLE method is used to help solve these equations (just as Gauss–Seidel is used for the single variable situation). In other words, SIMPLE is used as the smoother.

Given (9) and (10) on the fine grid, it is necessary to construct a coarse grid, of nodes and elements, and so coarse grid equivalents for (9) and (10).

Equation (9) is based at nodes, with the diagonally dominant sparse matrix **A** used to determine velocity corrections. Thus the coarse grid nodes, and the coarsening matrix **K**, can be determined by applying the earlier methods to **A**. Similarly, since (10) applies to elements, the matrix **D** is used to determine coarse grid elements and the element coarsening matrix **M**.

Once **K** and **M** have been determined, then coarse grid representations of (9) and (10) can be constructed

$$\mathbf{KAK}^{\mathrm{T}}u^c + \mathbf{KGM}^{\mathrm{T}}p^c = \mathbf{K}r \tag{15}$$

$$\mathbf{MCK}^{\mathrm{T}}u^c + \Omega\mathbf{M}(\mathbf{CNG} - \mathbf{D})\mathbf{M}^{\mathrm{T}}p^c = \mathbf{M}s \tag{16}$$

For reasons to be described later, (16) is approximated to:

$$\mathbf{MCK}^{\mathrm{T}}u^c + \Omega(\mathbf{MCK}^{\mathrm{T}}\mathbf{N}^c\mathbf{KGM}^{\mathrm{T}} - \mathbf{MDM}^{\mathrm{T}})p^c = \mathbf{M}s \tag{17}$$

where $\mathbf{N}^c$ is the inverse of the diagonal of $\mathbf{KAK}^{\mathrm{T}}$.

The right hand sides of these equations are determined by residuals from the latest guess on the fine grid:

$$r = a - \mathbf{A}u - \mathbf{G}p \tag{18}$$

$$s = b - \mathbf{C}u - \Omega(\mathbf{CNG} - \mathbf{D})p \tag{19}$$

where u and p represent the latest guesses on the fine grid. Once (15) and (17) are approximately solved, then corrections to the fine grid solution are given by:

$$\delta u = \mathbf{K}^{\mathrm{T}}u^c; \quad \delta p = \mathbf{M}^{\mathrm{T}}p^c \tag{20}$$

The coarse grid equations have exactly the same form as the fine grid equations. This can be seen more clearly by defining:

$$\mathbf{A}^c = \mathbf{KAK}^{\mathrm{T}}; \quad \mathbf{G}^c = \mathbf{KGM}^{\mathrm{T}}; \quad \mathbf{C}^c = \mathbf{MCK}^{\mathrm{T}}; \quad \mathbf{D}^c = \mathbf{MDM}^{\mathrm{T}} \tag{21}$$

whereupon (15) and (17) look very similar to (9) and (10). Clearly, the SIMPLE scheme can be used to reduce errors in the coarse grid equations just as it is used for the fine grid equations.

If the strict coarse grid element equation were used, that is (16), then the definition of $\mathbf{D}^c$ would actually be:

$$\mathbf{D}^{c*} = \mathbf{MDM}^{\mathrm{T}} + \Omega(\mathbf{MCNGM}^{\mathrm{T}} - \mathbf{C}^c\mathbf{N}^c\mathbf{G}^c) \tag{22}$$

This formulation has two drawbacks. First, much more computational work is required to calculate $\mathbf{D}^{c*}$ compared with $\mathbf{D}^c$, and secondly it is necessary to have a more complex method

of determining M in order to avoid $\mathbf{D}^{c*}$ completely losing diagonal dominance and so making error smoothing with SIMPLE impossible.

The coarse grid can be visualized geometrically, with each coarse grid element formed by combining fine grid elements, and each coarse grid node formed by combining fine grid nodes. Each coarse grid element contains an arbitrary number of coarse grid nodes, and a coarse grid node can belong to an arbitrary number of coarse grid elements. The particulars of the discretization in ASTEC mean that on the finest grid every element contains exactly eight nodes, but this is not true for the coarse grids. *Figure 4* illustrates the mesh coarsening for a 2D mesh of 4-node quadrilateral linear elements. Note that the elements and nodes are essentially coarsened independently.

Just as before, a hierarchy of coarse grids is created from the original fine grid equations, and the coarsening stops once there are no off-diagonal terms in either the node or element equations. The same F-cycle multigrid strategy is used to solve the linearized discrete Navier–Stokes equations, with outer iterations used to take care of non-linear terms.

PERFORMANCE OF NAVIER–STOKES MULTIGRID

Calculation procedure

For each multigrid calculation the following solution strategy is employed.

The initial guess consists of zero pressure and zero velocity everywhere, except that the velocity is set at inlet nodes. A divergence-free velocity field ($\nabla \cdot u = 0$ everywhere) is established by applying 12 cycles of the simple multigrid solver to solve (10) for the fine grid pressure corrections, and thus correcting velocities according to (14). These pressure corrections are not actually applied to the pressure field—only the flow is modified. Then several outer Navier–Stokes multigrid iterations are performed, with the advective terms updated after each one, until appropriate convergence criteria are satisfied. Approximately 12 outer iterations are generally required, with each outer iteration containing 8 multigrid cycles. The smoothing operation on each grid is just 1 SIMPLE iteration, consisting of 1 Gauss–Seidel sweep on the momentum equation, and 3 Gauss–Seidel sweeps on the pressure-correction equation. The under-relaxation factors used for SIMPLE as a smoother in the multigrid solver are $\omega_u = \omega_p = 0.8$, but the performance of the scheme proved relatively insensitive to the values prescribed for ω_u and ω_p.

The test problem

The multigrid solver is applied to determine laminar incompressible flow in the geometry shown in *Figure 5*. A uniform inlet flow is prescribed at the start of the small curved duct, with an outlet boundary condition (constant pressure and zero normal derivatives for velocity) applied on the downstream face of the larger tank. All other boundaries are walls, with a no slip condition on velocity. The Reynolds number for the inlet duct flow is 100. Several calculations are performed to determine how the CPU time scales with mesh size.

For comparison the standard SIMPLE method[7], using a diagonally preconditioned conjugate gradient solver for continuity, is also applied to this problem.

A lax convergence criterion is applied for these calculations, specifically that the maximum momentum and continuity residuals are reduced by three orders of magnitude. *Figure 6* shows maximum momentum residual against Cray-2 CPU time for both methods on the 11,000 element problem, and illustrates that applying more rigorous convergence criteria would inflate the SIMPLE timings much more than the AMG timings.

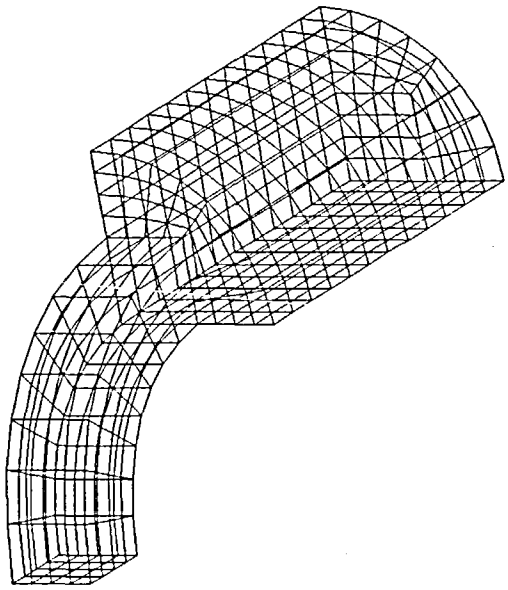

Figure 5 Mesh for the Navier–Stokes test problem

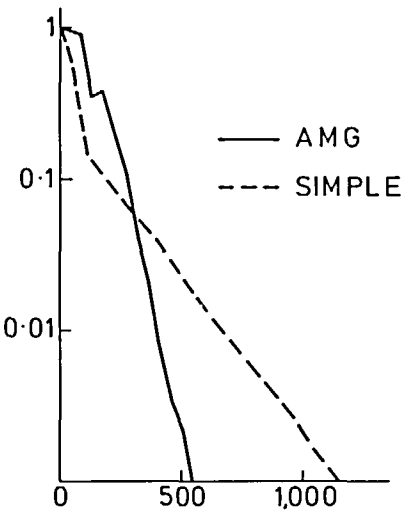

Figure 6 Residual vs. CPU time for Navier–Stokes test problem

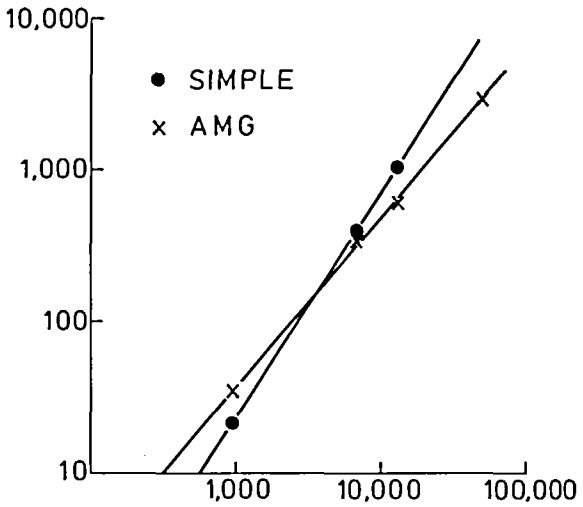

Figure 7 CPU time vs. mesh size for Navier–Stokes test problem

A log–log plot of CPU time on a Cray-2 machine, against mesh size, is given in *Figure 7*. The CPU time for the SIMPLE algorithm scales approximately as $n^{1.55}$, where n is the number of nodes in the mesh. By contrast the multigrid solver scales almost linearly, as $n^{1.15}$, using about 1 minute of Cray-2 CPU time for every 1000 nodes.

The memory requirements of the multigrid scheme also vary linearly, with approximately 250 words of extra storage required for every node when the multigrid solver is used. Note that all the matrices are sparse, and only the non-zero entries are stored and processed.

The Navier–Stokes AMG scheme is not robust however; it has been found to fail completely on some problems. The approximation involved by using $\mathbf{D}^c$ instead of $\mathbf{D}^{c*}$ ((21) and (22)) is probably responsible for reducing the robustness of the Navier–Stokes multigrid solver. Sweeps on the coarse grids actually introduce errors to the fine grid solution, because $\mathbf{D}^c \neq \mathbf{D}^{c*}$, and on many problems the fine grids cannot remove these errors as fast as they are generated on the coarse grids. The algorithm is numerically unstable and quickly diverges.

CONCLUSIONS

A simple algebraic multigrid scheme has been presented which can be used to solve linear equations arising from discretization on a 3D unstructured (finite element) mesh. This scheme has been extended to allow solution of the linearized discrete Navier–Stokes equations for incompressible flow, again on an unstructured mesh in three dimensions, taking account of the velocity–pressure coupling.

The multigrid scheme shows almost linear scaling of both CPU time and memory requirements with mesh size, both for single variable and velocity–pressure solutions.

The linear AMG solver proves particularly effective for problems where the mesh has a high aspect ratio, and is also generally faster than diagonally preconditioned conjugate gradients on non-vector machines.

On the given test problem, the multigrid Navier–Stokes solver proves faster than the standard SIMPLE algorithm for all but the smallest meshes.

The single variable AMG solver has proved very robust, and is a valuable tool for solving linear discrete equations on unstructured meshes.

However, the Navier–Stokes AMG solver has been found to fail on some problems—the method requires improvement before it can be considered for general use.

ACKNOWLEDGEMENTS

The author wishes to express his gratitude to Dr B. R. MacGregor, Mr G. Robinson and Dr R. Webster of AEA Technology Dounreay, and also Dr G. Lonsdale of GMD (Germany), for much helpful advice and discussion. This work was supported by Applications Development funding from AEA Technology.

REFERENCES

1 Brandt, A. Multi-level adaptive computations in fluid dynamics, *AIAA Paper 79-1455* (1979)
2 Stone, H. L. Iterative solution of implicit approximations of multidimensional partial differential equations, *SIAM J. Num. Anal.* 5, 530–558 (1968)
4 Ruge, J. and Stuben, K. Efficient solutions of finite difference and finite element equations by algebraic multigrid (AMG), *Arbeitspap. GMD No. 89* (1984)
5 Ruge, J. and Stuben, K. Algebraic multigrid, *Arbeitspap. GMD No. 2010* (1986)
6 Patankar, S. V. *Numerical Heat Transfer and Fluid Flow*, Hemisphere, New York (1980)
7 Lonsdale, R. D. and Webster, R. The application of finite volume methods for modelling three-dimensional incompressible flow on an unstructured mesh, *Proc. 6th Int. Conf. Numerical Methods in Laminar and Turbulent Flow*, Pineridge Press, Swansea (1989)
8 Gresho, P. M. *et al.* A modified finite element method for solving time-dependent, incompressible Navier–Stokes equations. Part 1: theory, *Int. J. Num. Meth. Fluids*, 4, 557–598 (1984)