NUMERICAL LINEAR ALGEBRA WITH APPLICATIONS
Numer. Linear Algebra Appl. 2000; **7**:429–447

Fast pressure calculation for 2D and 3D time dependent incompressible flow

J. van Kan, C. Vuik and P. Wesseling*,†

J.M. Burgers Center and Faculty of Information Technology and Systems, Department of Applied Mathematical Analysis, Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands

SUMMARY

A black box multigrid preconditioner is described for second order elliptic partial differential equations, to be used in pressure calculations in a pressure correction method. The number of cells in a block is not restricted as for standard multigrid, but completely arbitrary. The method can be used in a multiblock environment and fine tuning for cache hits is described. A comparison is made with wall clock times of conventional preconditioners.

KEY WORDS: multigrid; preconditioner; black box; alternating line Jacobi; incomplete block LU factorization; Krylov methods; pressure correction; incompressible flow

1. INTRODUCTION

1.1. Computation of time dependent incompressible flows

We will describe a black box multigrid method for second order elliptic partial differential equations discretized on structured grids. To provide motivation and background, we consider the non-stationary incompressible Navier–Stokes equations

$$\mathbf{u}_t + (\mathbf{u} \cdot \nabla)\mathbf{u} + \nabla p - \frac{1}{\mathrm{Re}}\nabla^2 \mathbf{u} = 0, \quad \mathrm{div}\, \mathbf{u} = 0 \tag{1}$$

After discretization in space we obtain a differential-algebraic system of the following form:

$$u_t + N(u) + Gp_h = 0 \tag{2}$$

$$Du = 0 \tag{3}$$

* Correspondence to: P. Wesseling, Department of Applied Mathematical Analysis, Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands
† E-mail: p.wesseling@math.tudelft.nl

Received 9 November 1999
Revised 10 May 2000

where u and p_h are algebraic vectors containing the velocity and pressure unknowns, respectively. A staggered grid has been assumed; the prototype of this kind of scheme has been proposed in Reference [1]. N is a nonlinear algebraic operator, whereas G and D are linear. Recent implementations on boundary fitted curvilinear grids with further references to the literature can be found in References [2–5].

Solution methods for differential algebraic systems are necessarily different from those for systems of ordinary differential equations. For example, they cannot be fully explicit. An efficient method is the pressure correction method [1]. Taking the explicit Euler method as an example, we obtain

$$\begin{aligned} &\frac{u^* - u^n}{\tau} + N(u^n) + Gp^{n-1/2} = 0 \\ &DG\,\delta p = \frac{Du^*}{\tau} \\ &u^{n+1} = u^* + \tau G\delta p, \quad p^{n+1/2} = p^{n-1/2} + \delta p \end{aligned} \tag{4}$$

The principles and time accuracy of pressure correction methods are discussed in References [6,7]. In curvilinear coordinates, it is not necessarily true that $D = G^{\mathrm{T}}$ so the operator DG is not necessarily symmetric.

Almost all of the computational effort to complete a time step goes into computing δp. Experience shows that this remains true for implicit time stepping schemes also. Hence an efficient method is required to solve these equations and since the operator DG is similar to the discretization of a second order partial differential equation, multigrid seems attractive. In our case this is the Laplace equation, but we will not restrict ourselves to this case, since in curvilinear coordinates variable coefficients are involved. A disadvantage of conventional multigrid is the requirement that the number of cells in every direction be divisible by a power of 2. This power has to be at least equal to the number of coarse grids that will be used. One of our primary aims is to remove this restriction. This greatly facilitates incorporation of a multigrid pressure solver in a general Navier–Stokes code, especially when a multiblock approach is used to generate grids in geometrically complicated domains. Such incorporation is further facilitated if the multigrid algorithm is wrapped in a black box, by which we mean that the user need only provide a matrix and a right-hand side to the multigrid subroutine and remains unaware of the inner workings of the method. For this reason we shall describe a black box *single step* multigrid algorithm that can be used for defect correction but also as a *preconditioner* for Krylov space methods.

2. A BLACK BOX MULTIGRID PRECONDITIONER

The idea of wrapping a multigrid solver in a black box has proven its worth in recent decades. In References [8–11], black box implementations of free standing solvers are described. As already has been remarked, we have opted for a black box multigrid *preconditioner*, rather than a free standing solver. A *solver* of a set of linear equations gives the solution to a specified accuracy. A *preconditioner* P gives an approximate solution $\tilde{\mathbf{u}} = P^{-1}\mathbf{b}$ to the system of equations $A\mathbf{u} = \mathbf{b}$. In our case, the preconditioner takes the form of *one* multigrid V-cycle (for terminology see Reference [12]), which can be used either as a single iteration step in defect correction [13] or as a preconditioner for a Krylov subspace method such as BiCGSTAB [14] or GMRES [15].

The direct demand for such a preconditioner came from our code for solving incompressible flow problems [2]. Our current implementation uses GMRES with incomplete LU factorization as a preconditioner [16–18] for solving the pressure equations. We expected to be able to do better with a multigrid preconditioner, especially in large problems.

2.1. Problems the preconditioner can handle

The preconditioner handles second order elliptic equations on a rectangle (block). It is able to handle 9 (27) point discretizations (quantities between brackets are for the 3D case). So it will handle (say) the Laplacian on a region that can be mapped smoothly onto a rectangle (block). This block is divided into $n_x \times n_y (\times n_z)$ cells. The equation can be discretized using a finite difference, finite volume or finite element method (bi(tri)- linear elements). Generically, this will lead to a set of equations with 9 (27) unknowns per equation with the obvious modifications at the boundaries.

First order derivatives will cause the discretization matrix to be non-symmetric. Furthermore, as noted in the preceding section, even the pressure matrix is not necessarily symmetric in curvilinear coordinates. This asymmetry may be caused by the discretization of both the curvilinear terms and the boundary conditions. So symmetry will not be assumed.

As mentioned before, a special feature is that there is no limitation on n_x, n_y (or n_z), unlike standard multigrid methods. Our preconditioner handles all values of n_α equally well. Since the purpose is that the solver can be deployed in existing environments we do not want to be hampered by any limitations in this respect.

The problem under consideration is of the form

$$A\mathbf{u} = \mathbf{f} \tag{5}$$

in which A is a block tridiagonal matrix of tridiagonal matrices (2D) or a block tridiagonal matrix of block tridiagonal matrices of tridiagonal matrices (3D). The effect of boundary conditions is put into the right-hand side $\mathbf{f}$.

To a given right-hand side $\mathbf{f}$, the preconditioner P gives an approximate solution $\tilde{\mathbf{u}} = P^{-1}\mathbf{f}$. P^{-1} must be an approximation to A^{-1}, but the cost to compute $P^{-1}\mathbf{f}$ should be much lower than for $A^{-1}\mathbf{f}$, the exact solution. This can be cast into a formal iteration process called *defect correction* ([13]), the pseudo Pascal code of which is as follows:

presets $\mathbf{u} = 0$, $\mathbf{r} = \mathbf{f}$, $n = 0$
begin (**Defect Correction**)
  while $\|\mathbf{r}\| > \epsilon$ **do**
    $\mathbf{c} := P^{-1}\mathbf{r}$;
    $\mathbf{u} := \mathbf{u} + \mathbf{c}$;
    $\mathbf{r} := \mathbf{r} - A\mathbf{c}$;
  end while
end

It is easy to show that this results in an iteration on the residual of the form $\mathbf{r} := (I - AP^{-1})\mathbf{r}$, which will converge if and only if $\rho(I - AP^{-1}) < 1$. Hence the eigenvalues of AP^{-1} must have positive real parts, and the closer they are to unity the better.

3. MULTIGRID ALGORITHM

For an introduction to multigrid methods, see Reference [12]. We shall only sketch the basic idea. Suppose one has to solve a discrete problem with generic discretization parameter h:

$$A_h \mathbf{u}_h = \mathbf{f}_h, \quad \mathbf{u}_h, \mathbf{f}_h \in U_h$$

Most iterative methods are very well suited to reducing the high frequency (or *rough*) component in the error, and ill suited to reducing the low frequency (or *smooth*) component. If, on the other hand, a solution would be known to a discretization with a coarser generic discretization parameter H

$$A_H \mathbf{u}_H = \mathbf{f}_H, \quad \mathbf{u}_H, \mathbf{f}_H \in U_H,$$

then this *coarse grid solution* could be used as the basis for an approximation to the smooth component of the fine grid solution $\mathbf{u}_h$. For this an interpolation procedure is needed, or more generally a map from U_H to U_h called the *prolongation* P. Now $\mathbf{v}_h = P\mathbf{u}_H$ is used as an initial estimate for an iterative process on the fine level h. Since the error in this initial estimate will mainly consist of a rough component (interpolation represents the smooth component well) this will converge rapidly.

The multigrid algorithm exploits this fact recursively, by solving a sequence of problems on grids of increasing coarseness. On the coarsest grid, the problem is solved exactly.

There are in principle many ways to obtain a coarse grid discretization, but one that works out well is a Galerkin-like method. Suppose you have a map from U_h to U_H (called the *restriction* R), then a natural way of obtaining a coarse discretization would be to take the basis of U_H to approximate any element of U_h (in other words use the prolongation) and for an appropriate subspace T_H of U_h (the test space) demand that the following relation be satisfied:

$$(\mathbf{t}_H, A_h P \mathbf{u}_H) = (\mathbf{t}_H, \mathbf{f}_h) \quad \forall \mathbf{t}_H \in T_H \tag{6}$$

Clearly, the most natural subspace to take would be the orthogonal complement of null(R) or the range of R^{T}. In other words

$$(R^{\mathrm{T}}\mathbf{v}_H, A_h P \mathbf{u}_H) = (R^{\mathrm{T}}\mathbf{v}_H, \mathbf{f}_h) \quad \forall \mathbf{v}_H \in U_H \tag{7}$$

As may easily be checked, this finally comes down to solving

$$R A_h P \mathbf{u}_H = R\mathbf{f}_h \tag{8}$$

In other words, $A_H = R A_h P$ and $\mathbf{f}_H = R\mathbf{f}_h$.

It would appear that a natural choice for R would be P^{T}. Unfortunately, this choice, though natural, is not always possible, owing to the following limitations

1. It is undesirable that the difference molecule 'spreads out' at coarser levels. The 9 (27) point molecule must remain like that at all levels.
2. The restriction and prolongation have to satisfy an accuracy requirement in order that the smooth component is represented sufficiently well on the coarse grid. This accuracy requirement depends on the order of the differential operator m. In our case, where m is 2, the sum of the orders of accuracy of P and R^{T} must be at least 3. That is, if one is $O(h)$ the other must be at least $O(h^2)$.

The choice $R = P^{\mathrm{T}}$ is possible in so-called *vertex centred* methods. A vertex centred method, however, requires that the number of points in all directions is odd. In a *cell centred* method, the footprint of the molecule will increase if the interpolation accuracy of P is $O(h^2)$ and $R = P^{\mathrm{T}}$. A lower accuracy is in violation of the accuracy requirement. Hence either R or P have to be of order $O(h)$, but not *both*. For that case, $R = P^{\mathrm{T}}$ is not possible. For further details see Reference [12].

More details about our choice of restriction/prolongation are provided in Section 3.5.

3.1. Multigrid as a preconditioner

It is very common to implement multigrid as a defect correction method and in fact all of References [8–11] are designed that way.

As noted in Reference [19] it is a natural step to use it as a preconditioner for a conjugate gradient method, since in a way defect correction is the dumbest iteration method and the extra computational effort of a Krylov method is almost negligible compared with that of the preconditioner. For pure CG one has to be very careful, since both the matrix and the preconditioner have to be symmetric. Apart from a symmetric matrix to start with (not true in our case) one would also need:

- $R = P^{\mathrm{T}}$ to preserve symmetry of the operator at the coarser levels;
- the post- and pre-smoothing operators have to be each others transpose. See Section 3.2 for a discussion.

These two requirements stand in the way of the flexibility we aim at and we have used BiCGSTAB [14] and GMRES [15] as Krylov methods that do not need the symmetry that CG needs. Compared to defect correction, the use of a Krylov method typically saves half the number of iterations and 40% computation time.

3.2. Smoothers

The typical two-grid algorithm for solving $A_h\mathbf{u}_h = \mathbf{f}_h$ looks like (see Reference [12]):

presets $\mathbf{u}_h = 0$, $\mathbf{r}_h = \mathbf{f}_h$, n_{pre}, n_{post}, $A_H = RA_hP$.
begin (**Presmoothing**)
  for $i := 1$ **to** n_{pre} **do**
    $\mathbf{c} := S_{\mathrm{pre}}\mathbf{r}_h$
    $\mathbf{u}_h := \mathbf{u}_h + \mathbf{c}$
    $\mathbf{r}_h := \mathbf{r}_h - A_h\mathbf{c}$
  end for
(**Coarse grid correction**)
  $\mathbf{r}_H := R\mathbf{r}_h$
  Solve $A_H\mathbf{c}_H = \mathbf{r}_H$
  $\mathbf{c}_h := P\mathbf{c}_H$
  $\mathbf{u}_h := \mathbf{u}_h + \mathbf{c}_h$
  $\mathbf{r}_h := \mathbf{r}_h - \mathbf{A}_h c_h$
(* *Postsmoothing* *)
  for $i := 1$ **to** n_{post} **do**
    $\mathbf{c} := S_{\mathrm{post}}\mathbf{r}_h$

$\mathbf{u}_h := \mathbf{u}_h + \mathbf{c}$
$\mathbf{r}_h := \mathbf{r}_h - A_h\mathbf{c}$
end for
end

It will be noted that there is a great similarity between the defect correction with a preconditioner as sketched in Section 2.1 and the pre- and post-smoothing process. This similarity is not coincidental: although the aim of preconditioning and smoothing is quite different, it often turns out that a good preconditioner is also a good smoother.

The form of the two-grid preconditioner follows directly from the observation that the residual of the two-grid algorithm satisfies

$$\mathbf{r}_h = (I - A_h S_{\text{post}})^{n_{\text{post}}}(I - A_h P(RA_h P)^{-1}R)(I - A_h S_{\text{pre}})^{n_{\text{pre}}}\mathbf{f}_h \tag{9}$$

Let us denote this by $\mathbf{r}_h = \mathcal{P}_r\mathbf{f}_h$ for short. We see immediately that the preconditioner satisfies

$$\tilde{\mathbf{u}}_h = A_h^{-1}\mathbf{f}_h - \mathbf{r}_h = A_h^{-1}(I - \mathcal{P}_r)\mathbf{f}_h \tag{10}$$

From this we conclude, that to make the preconditioner symmetric we must satisfy $A_h^{-1}\mathcal{P}_r = \mathcal{P}_r^{\mathrm{T}}A_h^{-1}$. The reader will have little trouble verifying that for this to be true it is sufficient that $R = P^{\mathrm{T}}, n_{\text{pre}} = n_{\text{post}}$ and $S_{\text{pre}} = S_{\text{post}}^{\mathrm{T}}$. That this condition is also necessary follows from the observation that if AB is symmetric then $B = CA^{\mathrm{T}}$, with C symmetric.

3.3. Choice of smoothers

We experimented with 3 types of smoothers: alternating damped line Jacobi, alternating zebra and incomplete block factorization. The first two could operate in principle in a parallel environment, are robust and are good smoothers. If parallelism is not a consideration, incomplete block factorization wins out over both Jacobi and Zebra by about a factor of two. This might be even more in practice where the pressure equation must be solved at every time step, and preliminary calculations to determine the incomplete factorization have to be done only once. However, timings have shown us that these preliminary calculations typically take about 10% of the total time. The same is true for Jacobi and Zebra. In our 2D calculations there was almost no difference between Jacobi and Zebra. We took $n_{\text{pre}} = 0$ and $n_{\text{post}} = 1$ or 2. These choices give the smallest wall clock times, though not necessarily the smallest number of iterations. Note that one smoothing step for Jacobi/Zebra consists of *two* iterations, one horizontal sweep, one vertical sweep (and three sweeps in 3D). The reason for this is to make the algorithm more robust in the presence of stretched cells (see Reference [12]). With block factorization as smoother this is not necessary except in very stretched grids (200:1 or worse). The advantage of this smoother (fewer operations) will be lost in that case.

3.4. Jacobi damping parameter

In Reference [12] an empirical damping parameter of 0.7 is given to use for alternating line Jacobi and the following reasoning provides a theoretical foundation for that. Consider a discretization of the

anisotropic Laplace equation on the square $(0, 1) \times (0, 1)$:

$$\epsilon_x(u_{j-1,k} - 2u_{jk} + u_{j+1,k}) + \epsilon_y(u_{j,k-1} - 2u_{jk} + u_{j,k+1}) = f_{jk} \tag{11}$$

with $\epsilon_x = e_x/\Delta x^2$, $\epsilon_y = e_y/\Delta y^2$. The resulting matrix is $A = \epsilon_x D_x + \epsilon_y D_y$. Let $N_x = \epsilon_x D_x - 2\epsilon_y$. The *damped* line Jacobi iteration in the x-direction with damping parameter ω is given by

$$N_x \mathbf{u}^{n+1} = N_x \mathbf{u}^n - \omega(A\mathbf{u}^n - \mathbf{f}) \tag{12}$$

For the *error* $\mathbf{e}^n$ we have

$$\mathbf{e}^{n+1} = \mathbf{e}^n - \omega N_x^{-1} A \mathbf{e}^n \tag{13}$$

Let us first consider the eigenvalues μ_x of the matrix $M_x = N_x^{-1}A$. The eigenvalues λ_x of the Jacobi iteration matrix will then be given by $\lambda_x = 1 - \omega\mu_x$.

We use von Neumann analysis on the generalized eigenvalue problem:

$$\mu_x\big(\epsilon_x(u_{j-1,k} - 2u_{jk} + u_{j+1,k}) - 2\epsilon_y u_{jk}\big)$$

$$= \epsilon_x(u_{j-1,k} - 2u_{jk} + u_{j+1,k}) \quad + \quad \epsilon_y(u_{j,k-1} - 2u_{jk} + u_{j,k+1}) \tag{14}$$

Letting $u_{jk} = e^{ij\rho_x} e^{ik\rho_y}$, we obtain after dividing by $e^{ij\rho_x} e^{ik\rho_y}$

$$\mu_x\left(\epsilon_x(2\cos\rho_x - 2) - 2\epsilon_y\right) = \epsilon_x(2\cos\rho_x - 2) + \epsilon_y(2\cos\rho_y - 2) \tag{15}$$

hence

$$\mu_x = 2\frac{\epsilon_x \sin^2\rho_x/2 + \epsilon_y \sin^2\rho_y/2}{2\epsilon_x \sin^2\rho_x/2 + \epsilon_y} \tag{16}$$

The eigenvectors belonging to these eigenvalues are precisely those of the Jacobi iteration matrix. If at least one of ρ_x, $\rho_y \in (\pi/4, \pi/2)$ this eigenvalue corresponds to a 'rough' eigenvector. We denote such an eigenvalue as μ_{xr}. Consequently, a lower bound for the 'rough' part of the spectrum R_x is obtained for $\rho_y = 0$ and $\rho_x = \pi/4$ and we have $\mu_{xr} \geq \epsilon_x/(\epsilon_x + \epsilon_y)$. Since the smoothing properties of line Jacobi with damping factor ω are given by the factor $1 - \omega\mu_x$, $\mu_x \in R_x$, we see that we cannot hope to give a damping factor ω that is effective uniformly in ϵ_x and ϵ_y. More specifically, when $\epsilon_x/\epsilon_y \to 0$ the smoothing effect will vanish. If we use alternating line Jacobi, however, we get for the vertical sweep a similar expression for the rough spectrum: $\mu_{yr} \geq \epsilon_y/(\epsilon_x + \epsilon_y)$. The smoothing effect of a complete alternating Jacobi cycle can, assuming that the two smoothing operators commute, be given by the factor $\sigma = (1 - \omega\mu_x)(1 - \omega\mu_y)$ with $\mu_x \in R_x$, $\mu_y \in R_y$. For a specific pair of ϵ_x and ϵ_y the rough spectrum will be the rectangle $R_x \times R_y$ with lower left-hand corner $(\epsilon_x/(\epsilon_x + \epsilon_y), \epsilon_y/(\epsilon_x + \epsilon_y))$ and upper right-hand corner $(2, 2)$. To get a robust smoother for a variety of ϵ_x and ϵ_y we apparently must solve a minimax problem:

$$\min_{\omega \in (0,1)} \max_{(x,y) \in G} |(1 - \omega x)(1 - \omega y)| \tag{17}$$

in which G is the pentagonal region defined by $x \geq 0$, $x + y \geq 1$, $y \geq 0$, $x \leq 2$, $y \leq 2$ (see Figure 1)

The reader will have little trouble in verifying that for values of ω above 0.5 the smoothing factor has negative extremes for the points $(0, 2)$ and $(2, 0)$ (these are equal because of the symmetry) and a

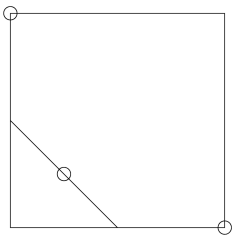

Figure 1. The region G with extremal points.

positive extreme for $(\frac{1}{2}, \frac{1}{2})$. For an optimal value these should be equal in absolute value, hence

$$(2\omega - 1) = (1 - \frac{1}{2}\omega)^2 \tag{18}$$

with solutions $\omega_{1,2} = 6 \pm 2\sqrt{7}$. For damped Jacobi we need $0 < \omega < 1$, so $\omega = 6 - 2\sqrt{7} \approx 0.7085$ is the one we were looking for.

A similar reasoning for the 3D case shows that ω has to satisfy

$$(2\omega - 1) = (1 - \frac{1}{2}\omega)^3 \tag{19}$$

giving $\omega \approx 0.6528$.

3.5. Restriction and prolongation

The usual implementations of black box multigrid use odd numbers of grid points in both the x- and y-directions at all levels [8–11]. The advantage of this way of doing things becomes apparent if we look at a 1D interval: the two extreme points of the interval belong to the grid at *all* levels. These are not necessarily *boundary* points but unknowns adjacent to or on the boundary, depending on the type and implementation of the boundary condition. Since a black box multigrid method bases itself solely on matrices and right-hand sides it should work independently of the type of boundary condition.

As we have implemented the black box solver, we allow *any* number of points in either direction. We describe prolongation to a fine level and restriction to a coarse level in one dimension only. The actual restriction/prolongation in two or three dimensions is obtained by chaining several of these restrictions/prolongations along different coordinate directions.

3.5.1. The fine level has $2N + 1$ points. If the fine level has $2N + 1$ points, the restriction will be standard vertex centred to $N + 1$ points (see Figure 2). The prolongation P will be obtained by linear interpolation and the restriction will be its transpose: $R = P^{\mathrm{T}}$. The formulae for the prolongation are

$$u_{2j} = U_j, \quad j = 0, \ldots, N \tag{20}$$

$$u_{2j+1} = \frac{1}{2}U_j + \frac{1}{2}U_{j+1} \quad j = 0, \ldots, N-1 \tag{21}$$

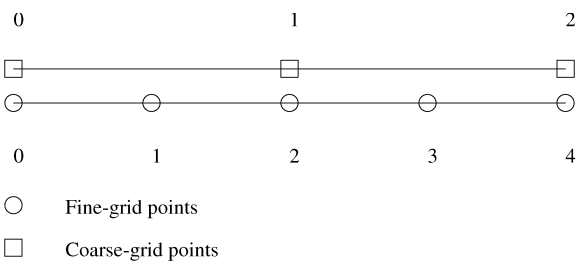

Figure 2. Placement of coarse- and fine-grid points for odd number of fine-grid points.

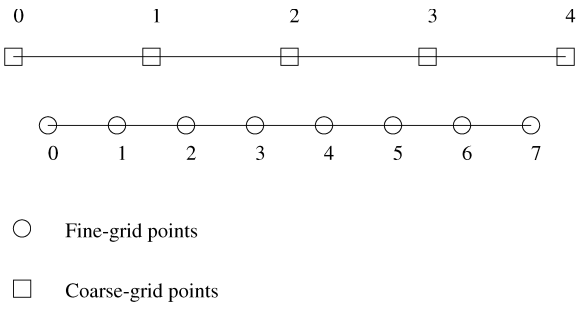

Figure 3. Placement of coarse- and fine-grid points for even number of fine-grid points.

and for the restriction

$$W_0 = w_0 + \frac{1}{2}w_1 \tag{22}$$

$$W_j = \frac{1}{2}w_{2j-1} + w_{2_j} + \frac{1}{2}w_{2j+1} \quad j = 1, \ldots, N-1 \tag{23}$$

$$W_N = \frac{1}{2}w_{2N-1} + w_{2N} \tag{24}$$

3.5.2. The fine level has $2N$ *points.* If the fine level has an even number of points, care must be taken that the prolongation of a constant function is again the same constant. The easiest way to do this is to take $N+1$ cell centred points (see Figure 3) and to take the prolongation by linear interpolation as follows:

$$u_{2j} = \frac{3}{4}U_j + \frac{1}{4}U_{j+1} \tag{25}$$

$$u_{2j+1} = \frac{1}{4}U_j + \frac{3}{4}U_{j+1}, \quad j = 0, \ldots, N-1 \tag{26}$$

The restriction is standard first order:

$$W_0 = \frac{1}{2}w_0 \tag{27}$$

$$W_j = \frac{1}{2}w_{2j-1} + \frac{1}{2}w_{2j}, \quad j = 1, N-1 \tag{28}$$

$$W_N = \frac{1}{2}w_{2N-1} \tag{29}$$

We apply these rules recursively and this completely determines the dimension of the grids at all coarse levels. For instance: if (say) the x-dimension of a problem is 76 the respective x-dimensions on the various coarse grids will be 39, 20, 11, 6, 4, 3 and 2.

4. IMPLEMENTATION CONSIDERATIONS

4.1. Stand-alone preconditioners

For a stand-alone preconditioner to be useful in a variety of applications it is necessary that it is configurable in a number of ways. First and foremost: the way the matrix A is stored must have a certain flexibility. One key choice must be made however: to store it as a rectangle (block) of molecules or to store it as a molecule of rectangles (blocks). Or, in matrix notation, if the 9 point difference molecule has indices $p_0 : p_8$, do we have $A(0 : n_x, 0 : n_y, p_0 : p_8)$ or $A(p_0 : p_8, 0 : n_x, 0 : n_y)$. Our flow solving package (coded in FORTRAN) uses the first option, which is equivalent (since in FORTRAN the first index varies most rapidly) to a molecule of rectangles. In the section about speed we shall say a word or two about the wisdom of this choice, but since it just does not pay to copy large structures of data into other large structures, we went along with it.

Within the molecule, however, the user of the package is free to use his own indexing. The default (standard) order would be

$$\begin{matrix} 7 & 8 & 9 \\ 4 & 5 & 6 \\ 1 & 2 & 3 \end{matrix}$$

but the user is free to provide the preconditioner with a different permutation. This simple feature should make the preconditioner easily embeddable in existing software packages.

A second measure to provide optimal flexibility consists in adding *slack space* to the matrix. The matrix can be extended to $A(-s_x : n_x + u_x, -s_y : n_y + u_y, p_0 : p_8)$, where s_x, u_x, s_y and u_y are slack parameters. The actual data is stored between the limits $0 : n_x, 0 : n_y$, but the slack space makes sure that (say) Dirichlet boundary conditions on certain boundaries can be accommodated without having to copy the whole structure. This feature is also useful in the context of *staggered grids*, where there will be a difference in dimensions of horizontal velocity points, vertical velocity points and pressure points. This can be implemented in various ways, but the slack space sees to it that, whatever method is chosen, the resulting matrix can be used as input for the preconditioner without an extra matrix copy.

4.2. Fine tuning for speed

A profile of the preconditioner reveals that it spends about 75 to 80% of its time doing matrix vector multiplications, usually in the evaluation of residuals. So, optimizing the matrix vector multiplication is of prime importance in achieving optimal speed. This will be largely machine dependent, but on a variety of computers good use can be made of the *cache*, auxiliary fast memory. If an operand is in the cache it can be accessed an order of magnitude faster than when it has to be got from conventional memory. On most architectures, when an element is accessed in conventional memory a whole contiguous block is loaded into the cache. There are cache optimization techniques that exploit this fact very elegantly [20] but there one has to have control over what goes into the cache and what does not. This is usually not true in standard programming languages, but a few general guidelines will help to optimize cache use also in this case.

- Minimize reloads. If one has an element in the cache, do all the things one has to do with it.
- Organize loops such that contiguous array elements are used in subsequent steps to maximize cache hits.

Independently of cache size, it will always be advantageous to have the most rapidly varying index in the innermost loop. This is well known for vector machines but it applies to cache hits as well. The rest of the fine tuning depends very much on the size of the blocks that will be loaded into the cache and whether this size is fixed or optimized during execution time.

Superficially it might seem that, using our configuration of molecules of blocks, the following implementation of $\mathbf{v} = A\mathbf{u}$ (2D) would be optimal in FORTRAN:

```
 c preset: v(i,j) = 0, i = 0...nx, j = 0...ny
do ip = 1, 9
  do j = 1, ny-1
    do i = 1, nx-1
      ix = i - 1 + mod (ip - 1, 3)
      iy = j - 1 + (ip - 1) / 3
      v (i, j) = v (i, j) + a(i, j, ip) * u (ix, iy)
    end do
  end do
end do
```

Experiments have shown, however, that making the ip-loop the innermost loop makes the multiplication about twice as fast; and unrolling the ip-loop (that is, writing it out) gains another factor of about 1.3. The explanation for this typical behaviour has to be found in that the above implementation reloads the vectors v and u nine times, whereas the alternative implementation reloads them only three times.

Let us call the chunk of memory that is loaded into the cache a vector (matrix) *cache block*. Having the ip-loop as innermost loop gives 12 cache blocks simultaneously in the cache (9 for the matrix and 3 for the vector). When we get to the end of a cache vector block, at some point in time we have 6 vector cache blocks and 9 matrix cache blocks simultaneously in the cache.

In 3D these numbers are 18 and 27, respectively. If all these blocks *do not* fit in the cache, performance drops dramatically. From this point of view it must be better to change the design, and make a block of molecules instead of a molecule of blocks. In other words, in FORTRAN the array structure should be

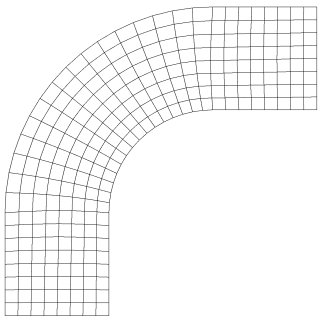

Figure 4. The configuration of the curved channel.

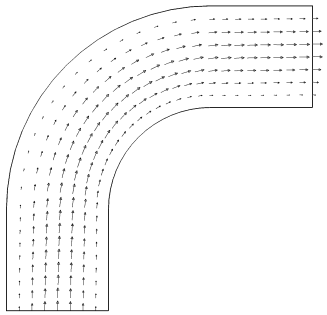

Figure 5. The flow in the curved channel.

Table I. Number of iterations and CPU time for different numbers of smoothing steps.

Grid	CPU to build preconditioner	$nsmooth = 1$		$nsmooth = 2$	
		Iterations	CPU	Iterations	CPU
8×32	< 0.01	7	0.02	4	0.02
16×64	0.01	8	0.08	5	0.08
32×128	0.03	8	0.42	5	0.42
64×256	0.15	8	2.16	6	2.47

`a(1:9,0:nx,0:ny)` and not `a(0:nx, 0:ny, 1:9)`. The former structure will give only one block for the matrix coefficients in the cache instead of 9 (27).

5. NUMERICAL EXPERIMENTS

In this section we present results of the computing time incurred to solve the pressure equations in various flow problems. Since the Reynolds number has no bearing on the pressure equation, it will not be given.

5.1. Curved channel

We consider the flow in a curved channel. The configuration of the channel is shown in Figure 4, together with its (8×32) grid. On the fixed walls a no-slip condition is given, whereas on the inflow a uniform velocity is given. Finally, a free outflow condition is proposed at the outlet. The resulting flow is given in Figure 5.

From previous experiments [21] it appears that a large gain can be obtained for the pressure equation. Therefore we restrict ourselves to this equation. First some experiments are done to investigate the efficiency of the GMRES accelerated multigrid method with respect to the number of smoothing steps. The measurements are given in Table I. GMRES was not restarted, so that the number of auxiliary

Table II. Number of iterations and CPU time measured in milliseconds per unknown for various grid-sizes.

Grid	Iterations	CPU
16×64	5	0.08
13×60	5	0.06
15×63	5	0.08
17×66	6	0.07
18×65	5	0.07
23×87	6	0.17

Table III. Number of iterations with and without Krylov acceleration.

	Iterations	
	Multigrid	GMRES/multigrid
8×32	5	4
16×64	7	5
32×128	8	5
64×256	8	6

vectors that has to be stored is equal to the number of iterations. It appears that the time to build the multigrid preconditioner is negligible with respect to the total solution time. Note that the CPU times for both choices of $nsmooth$ are comparable, but that the number of iterations is fewer for $nsmooth = 2$ than for $nsmooth = 1$. Therefore we take the choice $nsmooth = 2$ in all the following experiments.

The multigrid method is made such that every grid-size can be handled. It appears from experiments that for this problem the number of iterations and the efficiency is indeed independent on the grid-size (see Table II). The final grid-size 23×87 is a worst case. For this choice, the coarse grid-sizes are odd and even alternately. The number of iterations is more or less the same for all grid-sizes, whereas the CPU time per unknown increases by a factor of two in the worst case problem. In Table III we summarize the number of iterations with and without Krylov acceleration. It appears that Krylov acceleration helps to lower the number of iterations. Since the overhead is small there is also a gain in CPU time.

Finally, we compare the GMRES and Bi-CGSTAB acceleration methods with the multi-grid preconditioner and an ILU preconditioner with 8 diagonals of fill-in in the upper- and lower triangular matrix. The results are given in Figure 6. It appears for this problem that the combination of GMRES with a multigrid preconditioner is also optimal for relatively small problems. It is to be noted, however, that BiCGSTAB requires fewer intermediate vectors to be stored (4). But in the context of *flow* problems the number of vectors required by GMRES to solve the pressure never presents a problem: the storage used for the momentum matrix on the previous time level can be used for that.

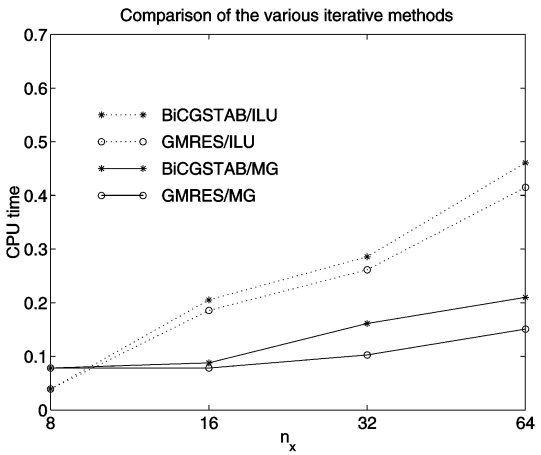

Figure 6. The efficiency of the various iterative methods, measured in CPU time in milliseconds per unknown.

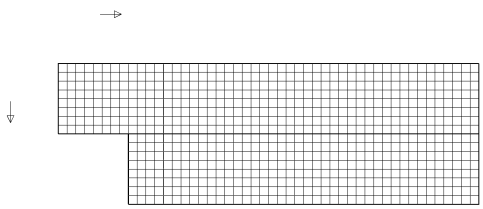

Figure 7. The configuration of the multiblock backward facing step.

5.2. Multiblock backward facing step

We also do some experiments with a flow in a backward facing step, which is computed by a multiblock algorithm [22]. In each outer iteration of this algorithm, the subdomain problems are solved by an inner iteration method. For the inner iteration method we consider the GMRES accelerated multigrid method and the GMRES/ILU method. The geometry of the backward facing step together with the 40×8 grid is shown in Figure 7. The number of outer iterations is the same for both methods. The number of inner iterations and the CPU time are given in Table IV. Again, the number of inner multigrid iterations is

Table IV. Number of inner iterations and CPU time.

	GMRES/multigrid		GMRES/ILU	
Grid	Iterations	CPU	Iterations	CPU
40×8	4	0.55	5	1.37
80×16	4	3.6	8	9.4
160×32	4	30	11	66

Table V. Number of iterations and CPU time for the pressure equation of the NACA airfoil problem.

Method	Iterations	CPU
multigrid	n.c.	—
GMRES/multigrid	58	5.22
BiCGSTAB/multigrid	41	5.94
GMRES/ILU	239	10.44
BiCGSTAB/ILU	86	4.2

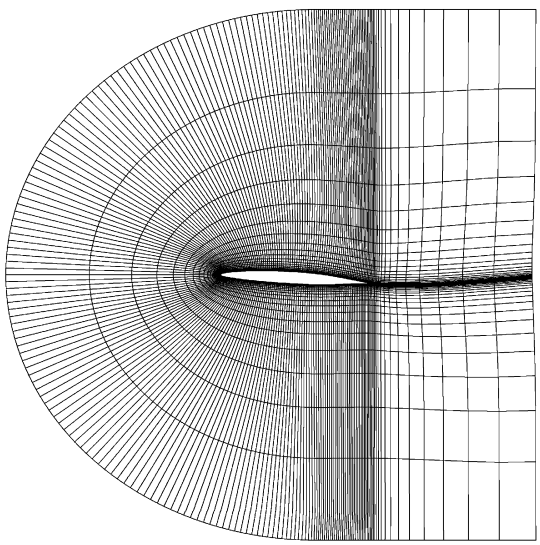

Figure 8. The grid used around the NACA 66-209 airfoil profile.

independent of the grid size, whereas the number increases for the GMRES/ILU method. Furthermore GMRES/multigrid costs less CPU time than GMRES/ILU.

5.3. Flow around an airfoil

In order to investigate the efficiency in more practical problems with non-orthogonal and stretched grids we also compute the flow around a NACA 66-209 airfoil. The 224×20 grid used in this simulation is presented in Figure 8. Note that there are large differences in the size of neighbouring grid cells and the cells at the profile or at the trailing edge are severely stretched. In Table V, the number of iterations and the CPU time are given for various methods. Note that multigrid with defect correction is not convergent. Using Krylov acceleration, multigrid is convergent and efficient. For this problem it appears that an ILU preconditioner without fill-in leads to a good performance. The combination GMRES/ILU needs many iterations. The reason is that GMRES is restarted after 40 iterations, which leads to slow convergence.

Table VI. Number of iterations and CPU time for the cube.

	GMRES/multigrid		GMRES/ILU	
nx	Iterations	CPU	Iterations	CPU
8	4	0.07	14	0.07
16	4	0.85	17	0.91
32	4	8.1	24	16.4

Note that with the ILU preconditioner, BiCGSTAB is the fastest method, whereas with the multigrid preconditioner, GMRES is the fastest method. This is as expected because the number of iterations is small for GMRES/multigrid but the matrix–vector product (including the preconditioner times vector product) is expensive, whereas for the ILU preconditioner the number of iterations is large and the matrix–vector product is relatively cheap (cf. Reference [23]). For this problem, BiCGSTAB/ILU is the fastest method, but the difference from GMRES/multigrid is not significant. The reason Defect Correction/multigrid does not converge (at least not rapidly) is probably that the smoother is not robust enough and does not perform well on this grid. We see here that instead of making the smoother more robust (probably at the cost of its excellent parallelizability) a good alternative is to replace defect correction with GMRES.

5.4. Flow in three-dimensional configurations

In this section we summarize some results for three-dimensional flow problems.

5.4.1. Flow in a cube. We consider the flow in a cube on an $nx \times nx \times nx$ grid. The flow is prescribed at the left, there are no-slip boundary conditions on the bottom, top, front and back faces and a free outflow on the right side of the cube.

Table VI contains the number of iterations and CPU time to solve the pressure equation. Again, the number of iterations for the multigrid method is independent of the grid-size. Furthermore GMRES/multigrid is also an efficient method for small grid sizes.

5.4.2. Flow in a skew cube. In this section we investigate the influence of the skewness of the grid on the convergence. In Figure 9 a skew cube is shown. The upper plane of this cube is translated along the vector (0,0.5,0). The convergence results are given in Table VII for a $16 \times 16 \times 16$ grid. Note that for both methods the convergence deteriorates. However the convergence of GMRES/multigrid depends less on the skewness than that of GMRES/ILU.

5.4.3. Flow in a rectangular channel. Finally, we consider the flow in a rectangular channel with length l, width w and height h. The configuration of the channel is given in Figure 10. The flow of the fluid is from left to right. The boundary conditions are: a uniform inflow at the left-hand plane, an outflow condition at the right-hand plane, and a no-slip condition at the other planes. In Table VIII the results for various values of l, w and h are given using a $16 \times 16 \times 16$ grid. Varying h in the same way as w leads to comparable results. It appears that the rate of convergence of GMRES/multi-grid deteriorates considerably when the grid cells are stretched in the direction of the flow. Below a possible explanation

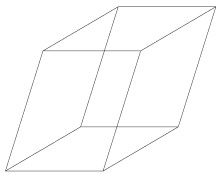

Figure 9. The configuration of the skew cube.

Table VII. Number of iterations and CPU time for the skew cube.

Translation vector	GMRES/multigrid		GMRES/ILU	
	Iterations	CPU	Iterations	CPU
(0,0,0)	4	0.85	17	0.91
(0,0.5,0)	4	0.86	23	1.30
(0,1.5,0)	7	1.34	37	2.30

Figure 10. The rectangular channel with length 10, width 1, and height 1.

Table VIII. Number of iterations and CPU time for the rectangular channel.

$l \times w \times h$	GMRES/multigrid		GMRES/ILU	
	Iterations	CPU	Iterations	CPU
$1 \times 1 \times 1$	4	0.85	17	0.91
$5 \times 1 \times 1$	14	2.64	19	1.03
$10 \times 1 \times 1$	24	4.30	22	1.26
$1 \times 5 \times 1$	8	1.5	19	1.03
$1 \times 10 \times 1$	9	1.65	24	1.33

Table IX. Number of iterations and CPU time for the rectangular channel with outflow boundary conditions.

	GMRES/multigrid		GMRES/ILU	
$l \times w \times h$	Iterations	CPU	Iterations	CPU
$1 \times 1 \times 1$	4	0.85	14	0.71
$5 \times 1 \times 1$	11	1.98	11	0.61
$10 \times 1 \times 1$	12	2.14	11	0.61

of this phenomenon is given. Owing to the boundary conditions for the velocity, the resulting system for the pressure resembles a discretized Poisson equation with a Dirichlet boundary condition at the right-hand plane and Neumann boundary conditions at the remaining planes. As a smoother we used a damped line Jacobi method. When a line is perpendicular to the flow direction and the grid is stretched in the flow direction the resulting system of equations per line converges to a discrete version of the one-dimensional Poisson equation with Neumann boundary conditions. Such a system is singular, which may be the origin of the bad smoothing property.

To verify this explanation the same problem is solved with a uniform inflow at the left plane and an outflow condition at all other planes. The results are given in Table IX and indeed the GMRES/multigrid results are now also acceptable for a stretched grid.

6. CONCLUSIONS

We have shown how restrictions on the number of grid cells can be removed in multigrid methods without incurring an efficiency penalty. A black box implementation of the multigrid method has been incorporated in an existing flow code to solve the pressure equation. A simple smoother with excellent parallelization potential is used, namely alternating line Jacobi with fixed optimal damping parameter. Efficiency and robustness are enhanced by Krylov subspace acceleration, using GMRES or BiCGSTAB. Usually GMRES is more efficient, but if storage is at a premium, BiCGSTAB is to be preferred. Cache performance has a significant impact on efficiency. Some considerations have been presented on the implementation that will improve cache usage. Applications to two- and three-dimensional flows have been presented.

The code is available by anonymous ftp at

```
ftp://ta.twi.tudelft.nl/pub/nw/vankan/multigrid
```

REFERENCES

1. Harlow FH, Welch JE. Numerical calculation of time-dependent viscous incompressible flow of fluid with a free surface. *The Physics of Fluids* 1965; **8**:2182–2189.
2. Wesseling P, Segal A, Kassels CGM. Computing flows on general three-dimensional nonsmooth staggered grids. *Journal of Computational Physics* 1999; **149**:333–362.

3. Shyy M, Vu TC. On the adoption of velocity variable and grid system for fluid flow computation in curvilinear coordinates. *Journal of Computational Physics* 1991; **92**:82–105.
4. Demirdžić I, Gosman AD, Issa RI, Perić M. A calculation procedure for turbulent flow in complex geometries. *Computers & Fluids* 1987; **15**:251–273.
5. Rosenfeld M, Kwak D, Vinokur M. A fractional step solution method for the unsteady incompressible Navier–Stokes equations in generalized coordinate systems. *Journal of Computational Physics* 1991; **94**:102–137.
6. Chorin AJ. Numerical solution of the Navier–Stokes equations. *Mathematics of Computation* 1968; **22**:745–762.
7. Van Kan JJIM. A second-order accurate pressure correction method for viscous incompressible flow. *SIAM Journal on Scientific and Statistical Computing* 1986; **7**:870–891.
8. Bandy V, Sweet R. A set of three drivers for BOXMG: A black-box multigrid solver. *Communications on Applied Numerical Methods* 1992; **8**:563–571.
9. Dendy JE. Black box multigrid. *Journal of Computational Physics* 1982; **48**:366–386.
10. de Zeeuw PM. Matrix-dependent prolongations and restrictions in a blackbox multigrid solver. *Journal of Computational and Applied Mathematics* 1990; **33**:1–27.
11. Goertzel B. An adaptive multilevel connectionist scheme for black box global optimization. In *Preliminary Proceedings of the 4th Copper Mountain Conference on Multigrid Methods*, vol. 2. Mandel J, McCormick SF (eds). Computational Mathematics Group, University of Colorado: Denver, 1989; 69–89.
12. Wesseling P. *An Introduction to Multigrid Methods*. Wiley: Chichester, 1992.
13. Auzinger W, Stetter HJ. Defect correction and multigrid iterations. In *Multigrid Methods*, Hackbusch W, Trottenberg U (eds), *Lecture Notes in Mathematics*, vol. 960. Springer-Verlag: Berlin, 1982; 327–351.
14. Van der Vorst HA. Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for solution of non-symmetric linear systems. *SIAM Journal on Scientific and Statistical Computing* 1992; **13**:631–644.
15. Saad Y, Schultz MH. GMRES: a generalized minimal residual algorithm for solving non-symmetric linear systems. *SIAM Journal on Scientific and Statistical Computing* 1986; **7**:856–869.
16. Meijerink JA, Van der Vorst HA. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Mathematics of Computation* 1977; **31**:148–162.
17. Vuik C. Solution of the discretized incompressible Navier–Stokes equations with the GMRES method. *International Journal for Numerical Methods in Fluids* 1993; **16**:507–523.
18. Vuik C. Fast iterative solvers for the discretized incompressible Navier–Stokes equations. *International Journal for Numerical Methods in Fluids* 1996; **22**:195–210.
19. Sonneveld P, Wesseling P, de Zeeuw PM. Multigrid and conjugate gradient acceleration of basic iterative methods. In *Numerical Methods for Fluid Dynamics II*, Morton KW, Baines MJ (eds). Clarendon Press: Oxford, 1986; 347–368.
20. Hu J. Cache based multigrid on unstructured grids. In *Virtual Proceedings of the 9th Copper Mountain Conference on Multigrid Methods*, Copper Mountain, Colorado USA, 11-16 April, 1999; 1–11, Manteuffel T, McCormick S (eds). http://www.mgnet.org
21. Zeng S, Vuik C, Wesseling P. Numerical solution of the incompressible Navier–Stokes equations by Krylov subspace and multigrid methods. *Advances in Computational Mathematics* 1995; **4**:27–50.
22. Brakkee E, Vuik C, Wesseling P. Domain decomposition for the incompressible Navier–Stokes equations: solving subdomain problems accurately and inaccurately. *International Journal for Numerical Methods in Fluids* 1998; **26**:1217–1237.
23. Vuik C. Further experiences with GMRESR. *Supercomputer* 1993; **55**:13–27.

