European Congress on Computational Methods in Applied Sciences and Engineering
ECCOMAS 2000
Barcelona, 11-14 September 2000
©ECCOMAS

A BLACK BOX MULTIGRID PRECONDITIONER FOR SECOND ORDER ELLIPTIC PARTIAL DIFFERENTIAL EQUATIONS

C. Vuik[⋆], J.J.I.M. van Kan, and P. Wesseling

[⋆] J.M. Burgers Center and Delft University of Technology, Faculty of Information Technology and Systems, Department of Applied Mathematical Analysis, Mekelweg 4, 2628 CD Delft, The Netherlands e-mail: c.vuik@math.tudelft.nl, web page: http://ta.twi.tudelft.nl/users/vuik/

Key words: Multigrid, preconditioner, black box, alternating line Jacobi, Krylov methods, pressure correction, incompressible flow.

Abstract. *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 each direction is not restricted as for standard multigrid, but completely arbitrary. Fine tuning for cache hits is described. A comparison is made with wall clock times of conventional preconditioners.*

1 INTRODUCTION

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 nonstationary incompressible Navier-Stokes equations. After discretization in space we obtain a differential-algebraic system of the following form:

$$u_t + N(u) + Gp_h = 0, \tag{1}$$
$$Du = 0, \tag{2}$$

where u and p_h are algebraic vectors containing the velocity and pressure unknowns respectively. N is a non-linear algebraic operator, whereas G and D are linear. Recent implementations of boundary fitted curvilinear grids with further references to the literature can be found in [9].

An efficient method to solve (1) is the pressure correction method [2]. Taking the explicit Euler method as an example, we obtain:

$$\begin{aligned}
&\frac{u^* - u^n}{\tau} + N(u^n) + Gp^{n-1/2} = 0,\\
&DG\,\delta p = \frac{Du^*}{\tau},\\
&u^{n+1} = u^* + \tau G\delta p, \;\; p^{n+1/2} = p^{n-1/2} + \delta p.
\end{aligned} \tag{3}$$

The principles and the time accuracy of pressure correction methods are discussed in [1, 5]. In curvilinear coordinates it is not necessarily true that $D = G^T$ 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. Hence an efficient method is required to solve these equations and since the operator DG is similar to a 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.

2 MULTIGRID PRECONDITIONING

For an introduction to multigrid methods the reader is referred to [8]. We shall only sketch the basic idea. Suppose one has to solve a discrete problem with generic discretization parameter h:

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

Most iterative methods are very well suited to reduce the high frequency (or *rough*) component in the error and ill suited to reduce 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:

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

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 this will converge rapidly. We use a Galerkin-like method to obtain a coarse grid discretization. This comes down to solving

$$RA_hP\mathbf{u}_H = R\mathbf{f}_h,$$

so $A_H = RA_hP$ and $\mathbf{f}_H = R\mathbf{f}_h$. It is a natural step to use multigrid as a preconditioner for a Krylov subspace method (Bi-CGSTAB [4], or GMRES [3]).

We experimented with two smoothers: alternating damped line Jacobi and alternating 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 and one vertical sweep (three sweeps in 3D). The reason for this is to make the algorithm more robust in the presence of stretched cells (see [8]).

3 THEORETICAL RESULTS

In this section we present optimal values for the damping parameter. Thereafter a restriction and prolongation are given for arbitrary number of points in each direction.

Jacobi damping parameter

Consider a discretization of the anisotropic Laplace equation on the square $(0,1)\times(0,1)$:

$$\epsilon_x(u_{j-1,k} - 2u_{jk} + u_{j+1,k}) + \epsilon_y(u_{j,k-1} - 2u_{jk} + u_{j,k+1}) = f_{jk}, \tag{4}$$

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 x-direction with damping parameter ω is given by

$$N_x\mathbf{u}^{n+1} = N_x\mathbf{u}^n - \omega(A\mathbf{u}^n - \mathbf{f}). \tag{5}$$

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

$$\mathbf{e}^{n+1} = \mathbf{e}^n - \omega N_x^{-1}A\mathbf{e}^n. \tag{6}$$

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 $\lambda_x = 1 - \omega\mu_x$. The smoothing effect of a complete alternating Jacobi cycle can, assuming that the two smoothing operators commute, be given by the factor $\sigma = (1 - \omega\mu_x)(1 - \omega\mu_y)$.

In [6] it is proven that for damped alternating line Jacobi the optimal value of the damping parameter satisfies $(2\omega - 1) = (1 - \frac{1}{2}\omega)^2$ in 2D and $(2\omega - 1) = (1 - \frac{1}{2}\omega)^3$ in 3D. Since $0 < \omega < 1$ this leads to $\omega_{opt} = 0.7085$ in 2D and $\omega_{opt} = 0.6528$ in 3D.

Restriction and prolongation

The usual implementations of black box multigrid use odd numbers of grid points in both x– and y–direction on all levels. The advantage of this becomes apparent if we look at a 1D interval: the two extreme points of the interval belong to the grid on *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 matrix and right-hand side 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.

If the fine level has 2N+1 points, the restriction will be standard vertex centered to N+1 points. The prolongation P will be obtained by linear interpolation and the restriction will be its transpose : $R = P^T$. If the fine level has an even number of points we take for the restriction N+1 points cell centered. The interpolation is given by

$$u_{2j} = \frac{3}{4}U_j + \frac{1}{4}U_{j+1}, \tag{7}$$

$$u_{2j+1} = \frac{1}{4}U_j + \frac{3}{4}U_{j+1}, \quad j = 0, \dots, N-1, \tag{8}$$

and the restriction by

$$W_0 = \frac{1}{2}w_0, \tag{9}$$

$$W_j = \frac{1}{2}w_{2j-1} + \frac{1}{2}w_{2j}, \quad j = 1, N-1, \tag{10}$$

$$W_N = \frac{1}{2}w_{2N-1}. \tag{11}$$

4 IMPLEMENTATION CONSIDERATIONS

A profile of the preconditioner reveals that it spends about 80% of its time doing matrix vector multiplications. 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*. 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, but there you have to have control over what goes into the cache and what not. This is usually impossible in standard programming languages.

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 gains another factor of about 1.3. The explanation for this typical behavior 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.

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 does not influence the pressure equation it will not be given.

5.1 Flow in a curved channel

We consider the flow in a curved channel [7]. 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.

Our multigrid preconditioner 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 1). 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 with a factor 2 in the worst case problem.

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 1: Number of iterations and CPU time measured in milliseconds per unknown for various grid-sizes

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 1. It appears for this problem that the combination GMRES with a multi-grid preconditioner is optimal also for relatively small problems. It is to be noted, however, that Bi-CGSTAB requires fewer intermediate vectors to be stored. 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.

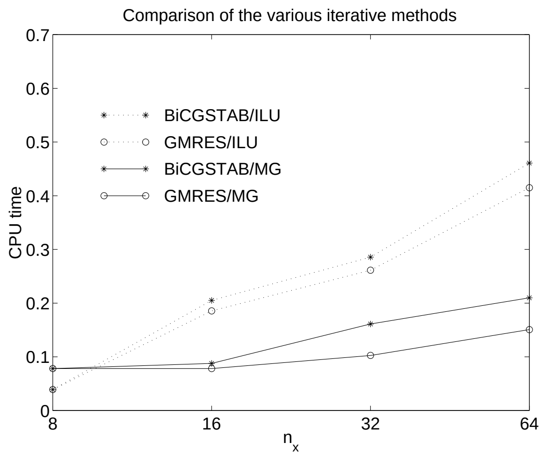

Figure 1: The efficiency of the various methods, measured in CPU time in milliseconds per unknown

5.2 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 bottom, top, front and back faces and a free outflow on the right side of the cube. Table 2 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 an efficient method also for small grid sizes.

nx	GMRES/multigrid		GMRES/ILU	
	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

Table 2: Number of iterations and CPU time for the cube

5.3 Flow in a rectangular channel

Finally we consider the flow in a rectangular channel, with length l, width w, and height h. The flow of the fluid is in the length direction. 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 3 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. At this moment we are trying to alleviate this drawback.

$l \times w \times h$	GMRES/multi-grid		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 3: Number of iterations and CPU time for the rectangular channel

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. 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] A.J. Chorin. Numerical solution of the Navier-Stokes equations. *Math. Comp.*, 22:745–762, 1968.

[2] F.H. Harlow and J.E. Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with a free surface. *The Physics of Fluids*, 8:2182–2189, 1965.

[3] Y. Saad and M.H. Schultz. GMRES: a generalized minimal residual algorithm for solving non-symmetric linear systems. *SIAM J. Sci. Stat. Comp.*, 7:856–869, 1986.

[4] H.A. Van der Vorst. Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for solution of non-symmetric linear systems. *SIAM J. Sci. Stat. Comp.*, 13:631–644, 1992.

[5] J.J.I.M. Van Kan. A second-order accurate pressure correction method for viscous incompressible flow. *SIAM J. Sci. Stat. Comp.*, 7:870–891, 1986.

[6] J.J.I.M. Van Kan, C. Vuik, and P. Wesseling. Fast pressure calculation for 2D and 3D time dependent incompressible flows. submitted.

[7] C. Vuik. Solution of the discretized incompressible Navier-Stokes equations with the GMRES method. *Int. J. Num. Meth. Fluids*, 16:507–523, 1993.

[8] P. Wesseling. *An Introduction to Multigrid Methods.* John Wiley & Sons, Chichester, 1992.

[9] P. Wesseling, A. Segal, and C.G.M. Kassels. Computing flows on general three-dimensional nonsmooth staggered grids. *J. Comp. Phys.*, 149:333–362, 1999.