AIAA JOURNAL
Vol. 36, No. 6, June 1998

Parallel Implicit Time-Accurate Navier–Stokes Computations Using Coarse Grid Correction

Carl B. Jenssen*
SINTEF—Foundation for Scientific and Industrial Research, Trondheim 7034, Norway
and
Per Å. Weinerfelt†
Saab AB, Linköping 58188, Sweden

A parallel implicit Navier–Stokes solver is constructed using a block-by-block inversion of the resulting linear system of equations followed by a global coarse grid correction to account for the block coupling. This solver is used to compute two different cases of vortex shedding flows. The efficiency of our method is compared to the efficiency of the method based on pseudotime stepping and multigrid acceleration. This comparison is based on direct comparison of the two methods applied to identical test cases, as well as results reported in the literature. Emphasis is put on scalability as well as mesh sensitivity. It is concluded that the solver is not sensitive to grid stretching and scales well with both problem size and number of processors, and it is shown that measured CPU times compare well with the multigrid approach.

Introduction

As pointed out by several authors, the use of implicit methods for computing unsteady flow can lead to a considerable gain in efficiency over explicit methods.[1–3] This is because the unsteady timescales for many engineering-type flows can be several orders of magnitude larger than the time-step limitation imposed by the stability criterion for explicit schemes. Thus, implicit schemes, where the time step is governed by accuracy and not stability, may follow a given time evolution in only a fraction of the time steps required by an explicit method.

In 1991, Jameson[4] proposed an implicit algorithm based on the multigrid method for the solution of unsteady flow. This method, which has since been adopted by many others,[2,5,6] is based on introducing a pseudotime and reformulating the nonlinear system of equations to be solved at each time step as a steady-state problem. An existing multigrid steady-state solver then can be used with minor modifications to solve the series of steady-state problems in pseudotime.

Along the same lines, an implicit steady-state solver based on the solution of a linearized problem at each time step also can be modified easily to solve unsteady problems. The multigrid method used to advance the solution in pseudotime simply is replaced by implicit time stepping. For infinitely long time steps in pseudotime, this method is reduced to Newton's method in the case of no linearization errors and exact solution of the linearized system. In practice, approximations are used in both the linearization and the solution phases, yielding only an approximate version of Newton's method. The approximate Newton iterations sometimes are referred to as subiterations on a given time step.

A straightforward way of parallelizing a computational fluid dynamics code is by means of the multiblock concept. The most common method is to advance the solution one time step independently in each block, followed by an update of the block boundary conditions. For implicit schemes, this yields a method in which implicit time stepping is used within each block but with explicit coupling between the blocks. This can be considered a block Jacobi method because the equations are solved for the unknowns in each block on the basis of values of the unknowns from the previous time step in the neighboring blocks. Although used with success by several authors, it can be demonstrated that, for subsonic flows, the number of iterations necessary to reach convergence increases as the number of blocks is increased, and thus the method is not scalable to a large number of processors.[7]

To remedy this problem, a coarse grid correction scheme (CGCS)[8,9] is added to the block Jacobi method to retain global influence in the solution procedure. The CGCS has proven to be very efficient for steady-state Euler problems and does not suffer to any great extent from the decay in the convergence rate often experienced with implicit methods as the number of blocks or processors is increased. We investigate the efficiency of the CGCS for unsteady Navier–Stokes problems by comparison with multigrid results reported in the literature and from computations obtained with our own multigrid solver.[10]

Although measurements of CPU times on several parallel computers are included, the emphasis is on the numerical efficiency of the underlying method rather than the speed obtained on a given computer architecture because an efficient scalable parallel algorithm is the main requirement for any parallel code.

Governing Equations and Numerical Scheme

We are solving the compressible Navier–Stokes equations for an ideal gas with constant viscosity. A finite volume approximation on a structured multiblock grid is formulated in the usual manner by considering the integral form of the equations. The convective part of the fluxes is discretized with a third-order, upwind biased method based on Roe's scheme. Usually, limiters are applied to the extrapolation procedure in this scheme to obtain a total variation diminishing scheme, but this has not been necessary for the low-Mach-number flow cases considered in this work. The viscous fluxes are calculated using central differencing. Derivatives of second-order accuracy are first calculated with respect to the grid indices and then transformed to derivatives with respect to the physical spatial coordinates.

Time Integration

Implicit and second-order-accurate time stepping is achieved by a three-point, A-stable, linear multistep method[11]:

$$\tfrac{3}{2}(V_i/\Delta t)U_i^{n+1} - 2(V_i/\Delta t)U_i^{n} + \tfrac{1}{2}(V_i/\Delta t)U_i^{n-1} = R\left(U_i^{n+1}\right) \quad (1)$$

Here, $R(U_i^{n+1})$ is the sum of the flux contributions, V_i is the volume of the grid cell i, Δt is the time step, and the superscript n refers to the time level. Equation (1) is solved by an approximate Newton

Presented as Paper 96-2402 at the AIAA 14th Applied Aerodynamics Conference, New Orleans, LA, June 17–20, 1996; received Aug. 8, 1996; revision received Nov. 14, 1997; accepted for publication Jan. 9, 1998.

*Research Scientist, Department of Applied Mathematics. Member AIAA.
†Research Scientist, Department of Computational Aerodynamics. Member AIAA.

iteration. Using l as the iteration index, it is customary to introduce a modified residual

$$R^*\left(U_i^{n+1}\right) = R\left(U_i^{n+1}\right) - \tfrac{3}{2}(V_i/\Delta t)U_i^{n+1} + 2(V_i/\Delta t)U_i^n - \tfrac{1}{2}(V_i/\Delta t)U_i^{n-1} \quad (2)$$

that is to be driven to zero at each time step by the iterative procedure

$$\left(\frac{-\partial R}{\partial U} + \frac{3}{2}\frac{V_i}{\Delta t}\right)\Delta U_i = R^*\left(\left[U_i^{n+1}\right]^l\right) \quad (3)$$

updating at each iteration $[U_i^{n+1}]^{l+1} = [U_i^{n+1}]^l + \Delta U_i$. In our implementation, as well as in most other similar codes, the Newton procedure is approximate because inevitably some approximations are used in the linearization of the flux vectors and also because an iterative solver is used to solve the resulting linear system. In particular, we use first-order approximation for the implicit operator.

Thus, for each iteration of the Newton procedure, we have to solve a linear system of equations that is septadiagonal in each block. Ignoring the block interface conditions, this system is solved concurrently in each block using a line Jacobi procedure.[7] In this work, except where otherwise stated, five iterations of the line Jacobi solver have been used, where for each iteration a tridiagonal system is solved along lines in each spatial dimension. The iteration count of five was chosen from experience to give convergence at each time step in the lowest possible CPU time.

A CGCS[9] is used to compensate for ignoring the block interface conditions by adding global influence to the solution. The coarse mesh is obtained by dividing the fine mesh into a suitable number of cells by removing grid lines. The size of the coarse grid system is chosen as the maximum that can be solved efficiently with a parallel solver. In practice, it is the CPU time that sets the limit for how fine a coarse grid can be used, although eventually memory also would impose a limit. In our implementation, constructing a coarse grid by merging four to five grid cells in each of the three spatial dimensions increases the CPU time per iteration by 15–20% compared with when the CGCS is not applied.

The CGCS is reviewed briefly. Using the subscript h to denote the fine mesh and H for the coarse mesh, the method is described in the following. In matrix form, we can write the linear system of equations that we wish to solve at each Newton iteration as

$$A_h \Delta U_h = R_h \quad (4)$$

However, by ignoring the coupling between blocks, we actually solve a different system that we can write symbolically as

$$\tilde{A}_h \widetilde{\Delta U}_h = R_h \quad (5)$$

Defining the error obtained by not solving the correct system as $\varepsilon_h = \Delta U_h - \widetilde{\Delta U}_h$, we have

$$A_h \varepsilon_h = R_h - A_h \widetilde{\Delta U}_h \quad (6)$$

In a classic multigrid fashion, we now formulate a coarse grid representation of Eq. (6) based on a restriction operator I_h^H. Thus we define

$$A_H = I_h^H A_h I_H^h \quad (7)$$

$$R_H = I_h^H (R_h - A_h \widetilde{\Delta U}_h) \quad (8)$$

and solve for the correction to $\widetilde{\Delta U}_h$:

$$A_H \Delta U_H = R_H \quad (9)$$

The solution to Eq. (9) is transformed to the fine grid by means of the prolongation operator I_H^h, and finally the fine grid solution is updated by

$$\widetilde{\widetilde{\Delta U}}_h = \widetilde{\Delta U}_h + I_H^h \Delta U_H \quad (10)$$

As restriction operator, we use summation, and as prolongation operator injection.[12]

To solve the coarse grid system, any suitable parallel sparse matrix solver can be applied. We have used a Jacobi-type iterative solver that at each iteration inverts only the diagonal coefficients in the coarse grid system. In most calculations, we have used 25 iterations of the Jacobi solver to solve the coarse grid problem.

Reference Multigrid Code

To enable comparisons of the CGCS with the multigrid-based approach, we have performed identical calculations with the CGCS code and a multigrid code,[10] based on the Jameson scheme.[13] Our multigrid code is based on a conservative cell–vertex finite volume formulation. This type of scheme has the advantage, compared to a standard cell-centered, finite volume scheme, of giving a fairly accurate solution even on an irregular mesh.

The convective flux integrals are obtained by averaging the fluxes from the four nearest-neighbor cells. For the viscous terms, we first form the gradient ∇U, using a staggered grid, and then we compute the viscous flux integral in the same way as the convective ones. The discretizations result in a central-difference, nine-point approximation of both the convective and the viscous terms. A fourth-order difference operator is added throughout the computational domain to provide a base level of dissipation that will eliminate nonlinear instabilities.

The space discretization forms a system of nonlinear ordinary differential equations that is integrated in time by means of a Runge–Kutta multistep scheme. We use the following two-stage scheme, having a maximum Courant–Friedrichs–Lewy (CFL) number of 1, together with a local time-stepping technique:

$$U^{n+\frac{1}{2}} = U^n + (\Delta t/V)R(U^n)$$

$$U^{n+1} = U^n + (\Delta t/V)R\left(U^{n+\frac{1}{2}}\right) \quad (11)$$

A short description of the multigrid technique is given next. Consider the steady Navier–Stokes equations written in the form

$$R_h(U_h) = f_h \quad (12)$$

where h denotes the mesh size; R_h is the sum of the convective, physical, and artificial dissipation operator; and f_h is a source term that is zero on the finest grid. Equation 12 can be approximately solved by the two-step Runge–Kutta scheme (11).

The h grid residuals, $f_h - R_h(U_h^{n+1})$, are transferred to the next coarser grid by an area-weighed average operator A_h^{2h}:

$$R_{2h}(U_{2h}) = R_{2h}\left(I_h^{2h} U_h^{n+1}\right) + A_h^{2h}\left[f_h - R_h\left(U_h^{n+1}\right)\right] \quad (13)$$

The term $I_h^{2h} U_h^{n+1}$ denotes the point restriction of the h grid solution, and R_{2h} is the restriction of the operator R_h to the $2h$ grid. The Runge–Kutta scheme also is applied to Eq. (13) with frozen value of the right-hand side. At the first time step on grid $2h$, U_{2h} is initialized to the value $I_h^{2h} U_h^{n+1}$, which implies that the iterations on coarser grids are driven by the h-grid residuals. An improved h-grid solution to Eq. (12) finally is obtained by applying bilinear interpolation I_{2h}^h to the coarse grid correction:

$$U_h = U_h^{n+1} + I_{2h}^h\left(U_{2h} - I_h^{2h} U_h^{n+1}\right)$$

The procedure is repeated on a sequence of grids, using a full approximation scheme, until Eq. (12) is solved.

For time-dependent problems, we again write the equation in terms of the modified residual 2 and introduce a pseudotime τ, leading to

$$\frac{dU}{d\tau} = R^*(U) \quad (14)$$

The equation is integrated to steady state using the multigrid technique. At each global time step $R^*(U) = 0$, resulting in a time-accurate solution.

Flow Around a Circular Cylinder

As a first test case, we use the vortex shedding from a circular cylinder at Reynolds numbers from 50 to 200 for which the flow is unsteady, laminar, and, at least in the lower range of Reynolds numbers, truly two dimensional. The Mach number in these calculations was set to 0.2 to simulate nearly incompressible flow, where both experimental data and computations by other authors are available. An example of a calculated flowfield is shown by entropy isolines

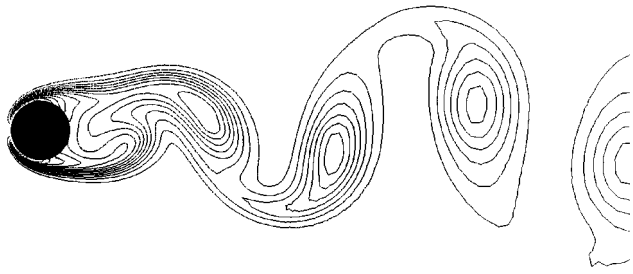

Fig. 1 Calculated instantaneous entropy field behind a circular cylinder.

in Fig. 1. In all of our computations, we used a nondimensional time step of 0.1, which corresponds to about 50 time steps per shedding cycle. The nondimensional time step $\overline{\Delta t}$ is defined as

$$\overline{\Delta t} = \Delta t v / D \tag{15}$$

where Δt is the physical time step, v the freestream velocity, and D the diameter of the cylinder.

In the following evaluation of our implicit scheme, we emphasize scalability with grid size and number of blocks or processors as well as mesh sensitivity. Ideally, one seeks a method in which the number of Newton iterations required to reach a given convergence criterion is independent of the mesh size. That way, the required CPU time would be proportional to the problem size. In the framework of parallel computing, it is equally important that the required number of iterations does not grow with increasing number of processors. Also, the number of iterations should not depend on the cell aspect ratio if the solver is to be used for highly stretched Navier–Stokes meshes or for complex geometries.

A common problem with time-accurate implicit schemes is what to use as a convergence criterion for the Newton iterations at each time step. In theory, the error in the solution of the nonlinear system should be significantly less than the local truncation error, but such a criterion is difficult to construct in practice. Therefore, one usually ends up with a criterion based on a reduction of the modified residual down to a specified tolerance. However, a reduction of the residual does not in all cases imply a reduction of the error. Indeed, a too-weak residual-based convergence criterion sometimes can hide the fact that some error modes are still present, although the solution appears to have converged. Therefore, we have performed two sets of tests. In the first test, an unusually strict convergence criterion was used over a short integration period to make sure that all error modes are damped out evenly. Once this was confirmed, a second test was carried out, using a tolerance of 0.01 for the relative reduction of the modified residual. This tolerance was found to be sufficient for time accuracy and is what one would use for normal production runs. It is also of the same magnitude as what other authors have used for similar problems.[14]

Thus, we first test the CGCS and the reference methods on a series of circular grids, each consisting of 128 × 128 cells, with the ratio of the height of the innermost cell and the cylinder diameter set to 0.01, 0.001, 0.0001, and 0.00001. In each case the far-field boundary was at 15 cylinder diameters from the center of the cylinder, and the grid spacing in the radial direction was controlled by a constant stretching factor. With a fully developed periodic solution as the initial condition, a few time steps were run with the convergence criterion set to 10^{-6} of the initial residual.

As shown in Fig. 2, when using the CGCS, the number of iterations to reach the convergence criterion is almost independent of the grid stretching. In this case we used 64 blocks of size 4 × 64, with 64 being the number of cells in the direction normal to the cylinder surface. In an equivalent test with the CGCS turned off, i.e., with a block-Jacobi-type subiteration at each time step, the residual flattens out after about three decades of reduction, as can be seen in Fig. 3. This is almost certainly due to inferior damping of error modes with long wavelengths because in this case there is no global coupling except for the updating of block interface values between each subiteration. Thus, we can clearly see the improvement obtained by using CGCS and the need for a globally coupled solution procedure.

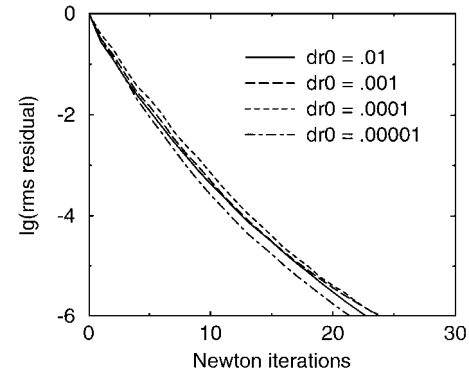

Fig. 2 Convergence history using CGC for different grid stretchings.

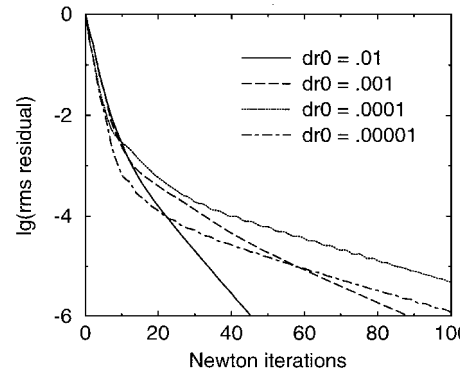

Fig. 3 Convergence history using block Jacobi for different grid stretchings.

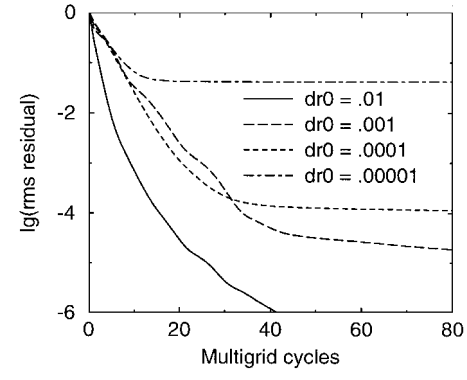

Fig. 4 Convergence history using multigrid code for different grid stretchings.

Figure 4 shows a third set of computations on the same series of grids, this time using the multigrid code. In the case of smallest grid stretching, this approach works well, but with increased cell aspect ratios, the residual curves flatten out before the convergence criterion is reached. From the analysis by Allmaras,[15] it is clear that the multigrid method does not perform efficiently on very stretched meshes. This is explained by the fact that smooth waves are not sufficiently damped by the coarsest grid because of the grid stretching. Heuristically, the poor convergence can be explained by time-step limitations in pseudotime imposed by the CFL condition for the explicit Runge–Kutta scheme.

Next, we consider the scalability of the CGCS by a sequence of three computational grids consisting of 64 × 64, 128 × 96, and 256 × 128 cells in the circumferential and radial directions, respectively. The height of the innermost grid cells in this case was set to 0.002 cylinder diameters. These grids then were partitioned into a number of blocks, and the code was run on an Intel Paragon parallel computer. A series of partitionings into a different number of blocks was tested; the numbers of Newton iterations needed at each time step to drive the modified residual down two decades are shown in Fig. 5. As we can see, for a given grid size, the number of iterations is virtually independent of the number of blocks. Also, the number of iterations

Table 1 CPU times per time step and grid point

Reference	CPU time, s	Machine	Linpack benchmark, Mflop	Relative CPU time, s
CGCS code	0.02	Paragon	10	1.0
Multigrid code	0.002	DEC Alpha	52	0.5
Belov et al.[14]	0.01	IBM RISC	38	1.9
Alonso et al.[3]	0.0045	IBM RISC	38	0.9

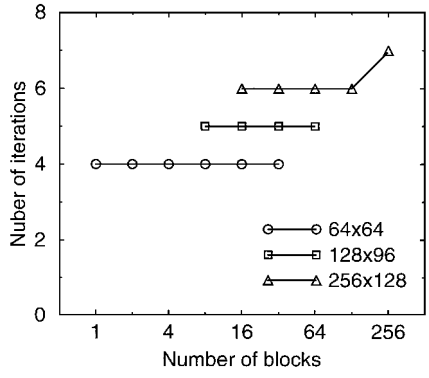

Fig. 5 Number of Newton iterations as a function of number of blocks for different grid sizes, for a convergence criterion of 10^{-2}.

increases only slightly with growing grid size. In going from the 64×64 grid to the 256×128 grid—an increase of the number of cells by a factor eight—the number of iterations goes from four to six.

An estimate of the efficiency of the present method compared to that of the multigrid approach can be obtained by comparing the CPU time required by the CGCS to that required by the multigrid code and to results reported in the literature. In our case, with the CGCS code, a two-decade reduction of the residual at each time step was obtained in six Newton iterations using 32 blocks on the finest grid, corresponding to less than 0.02 s of CPU time per time step and grid point when multiplied by the number of processors used on the Intel Paragon. The multigrid code was run on a DEC 3000-900 Alpha Axp, where the equivalent CPU time per time step and grid point was 0.002 s, corresponding to about 20 multigrid cycles on a total of six grid levels.

Almost identical computations have been reported by Belov et al.[14] They performed similar studies using pseudotime stepping and multigrid to solve the incompressible two-dimensional Navier–Stokes equations and reported a CPU time per time step and grid point of less than 0.01 s on an IBM RISC 6000-580. This was based on relative reduction of the residual of about two decades, which was obtained in 15 multigrid cycles.

In another paper, Alonso et al.[3] used a similar approach to solve the compressible Navier–Stokes equations around an oscillating circular cylinder for $Re = 500$ and a Mach number of 0.2. Using 36 time steps per oscillation period and 40 multigrid cycles per step, they report a CPU time corresponding to 0.0045 s on an IBM RISC 6000-580. They also performed a parallel computation on a 12-processor IBM SP2 in which the accumulated CPU time on all processors was reported to be the equivalent of 0.0075 s per time step and grid point.

The measured CPU times are summarized in Table 1, together with the Linpack benchmark[16] for the computing speed of the respective machines. A relative CPU time has been calculated by scaling the reported CPU times with the CPU time of the CGCS code and the ratio of the computing speed for the utilized computer and that of a single Paragon processor. When comparing the relative CPU time, it should be taken into account that the CPU time for the CGCS code includes the parallel overhead of running on 32 processors as well as the overhead associated with calculating two-dimensional flow with a general three-dimensional code. Also, to compare computation of incompressible flow, as in the case of Belov et al.,[14] with computation of compressible flow at Mach 0.2 leaves many uncertainties. We therefore simply conclude that the two different methods have comparable efficiency.

Table 2 Strouhal number, lift, and drag for $Re = 200$

Reference	Sr	C_l	C_d
CGCS code	0.196	± 0.70	1.38 ± 0.055
Multigrid code	0.195	± 0.68[a]	1.15 ± 0.050[a]
Belov et al.[14]	0.193	± 0.64	1.19 ± 0.042
Experiments	0.197	——	——

[a]Pressure forces only.

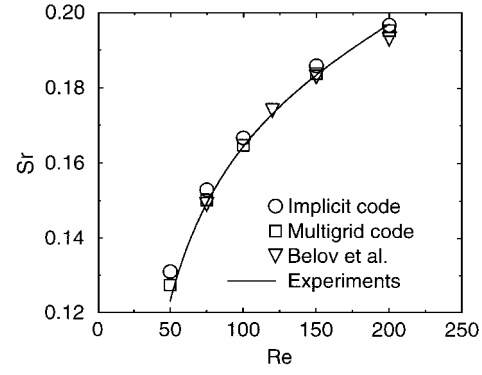

Fig. 6 Calculated Strouhal number; circular cylinder.

Validation results, in terms of the nondimensionalized frequency or Strouhal number of the lift, are shown in Fig. 6 and compared to a curve fit of experimental data[17] and calculations by Belov et al.[14] Because the flow for these low Reynolds numbers develops a completely periodic behavior, the frequency of the computed global forces can be determined easily. In Table 2, the Strouhal number, lift, and drag are shown for $Re = 200$ and again compared to data from the same sources. We refer to the paper of Belov et al.[14] for an overview of additional computations and experiments for this case.

Flow over a Suspension-Bridge Deck

The flow over a section of a suspension-bridge deck is used as an example of where the CGCS has been used for a realistic large-scale, time-dependent computation. The aerodynamic properties of a bridge deck are important factors in the design of long and slender bridges because aeroelastic effects can result in various kinds of vibrations and flutter.[18] Some experimental data exist for the Great Belt bridge currently under construction in Denmark,[19] and we use the cross section of this bridge as our test case. The geometry of the bridge deck, with a typical view of the computed flowfield, can be seen in Fig. 7. The Reynolds number used in the computations was that of the wind-tunnel experiments, $Re = 4.5 \times 10^4$.

A large-eddy simulation (LES) was carried out on a three-dimensional mesh consisting of a total of 3×10^6 grid cells obtained by repeating a two-dimensional mesh 40 times in the third dimension over a distance of 0.2 times the width of the bridge deck. A standard Smagorinsky subgrid-scale turbulence model was used in this simulation. The dimensionless time step based on the width of the profile was set in this case to 0.005 to capture turbulent fluctuations. As before, the Mach number was set to 0.2 to simulate nearly incompressible flow. The results in terms of time-averaged pressure distribution on the profile are shown in Fig. 8, and we see quite good agreement with the measurements.

The computation was carried out on a Cray T3E using 48 processors. However, the mesh was divided into a total of 664 blocks so that each processor was assigned several blocks. This was done to reduce the memory requirement by reducing the block size and thereby also the size of the temporary workspace needed. The coarse grid was defined as consisting of $5 \times 5 \times 5$ grid cells on the fine grid. A total of 1000 iterations were carried out, using close to 20 CPU hours. At each Newton iteration, 2 iterations of the linear solver were applied in each block, together with 10 iterations of the global coarse grid solver.

A convergence study on the number of Newton iterations required at each time step was carried out for this test case, using a coarser

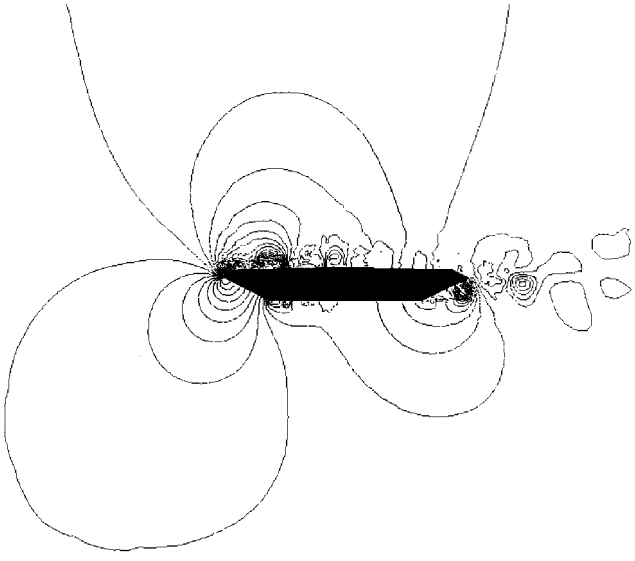

Fig. 7 Computed instantaneous pressure distribution in a two-dimensional plane around the Great Belt bridge deck.

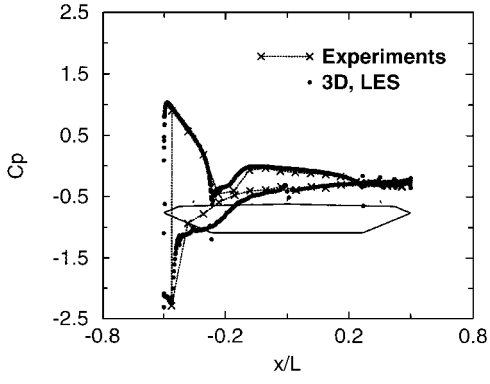

Fig. 8 Time-averaged pressure distribution on the suspension-bridge deck for 6-deg angle of attack, computed using LES: $Re = 4.5 \times 10^4$, and $\alpha = 6$.

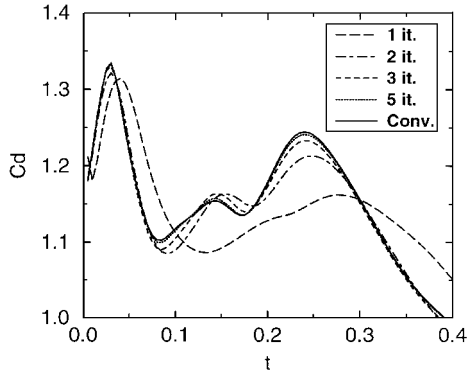

Fig. 9 Time history of calculated drag coefficient for the suspension-bridge deck using the CGCS with different numbers of Newton iterations.

mesh consisting of about 750,000 grid cells, still divided into 664 blocks and with the coarse grid defined in the same way, i.e., by merging 5 grid cells in each direction. In this case, 5 iterations of the linear solver were applied, and 25 iterations were used for the global coarse grid solver.

A series of short runs from the same initial conditions were performed, and in each run, a fixed number of Newton iterations per time step was specified. In Fig. 9, we can see the computed drag over a dimesionless time of 0.4, using the CGCS with one, two, three, or five Newton iterations as well as the converged result. The result we consider as converged was obtained by using 20 Newton iterations, which produced a curve that could not be distinguished

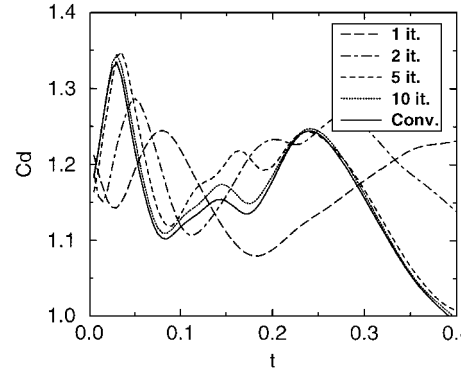

Fig. 10 Time history of calculated drag coefficient for the suspension-bridge deck using the block Jacobi method with different numbers of Newton iterations.

from a curve obtained by 10 iterations. As we can see, there is almost no noticeable difference between the curves corresponding to five iterations and the converged result. Even for two iterations, the general shape of the curve is obtained, and for three iterations the results seem nearly converged. Only for one iteration is there a significant deviation from the converged result.

In a similar test, using the block Jacobi method, we see in Fig. 10 that for one, two, and five iterations there are still significant discrepancies from the converged results. It can be seen that, even for 10 iterations the results are not completely converged, with an accuracy that seems to correspond to between 2 and 3 CGCS iterations. Only after 20 Newton iterations were we able to obtain a curve that exactly matched that of the converged result.

In these computations, the Jacobian was recomputed for each iteration, meaning that the CPU time is directly proportional to the number of iterations. It was found that, whereas the block Jacobi method required 42.8 s on 16 processors, the GCGS needed 51.5 s, or an increase of about 20%. We therefore can conclude that, for this test case, the CGCS requires around one-third of the CPU time needed by the block Jacobi method to converge to a given level of accuracy.

Conclusions

We have seen that the CGCS is well suited for implicit time-accurate Navier–Stokes computations. In our test, the CPU time requirements of the solver compare well with the popular multigrid-based approach. The number of Newton iterations needed depends only to a very small degree on the grid size and the number of blocks used. Thus, the method is scalable in terms of both the problem size and number of processors utilized.

It is demonstrated that the CGCS converges significantly faster than the block Jacobi method for the test cases considered. It also is shown that, for a realistic three-dimensional computation, the computational overhead of the CGCS was around 20% and the CPU time needed for a given level of accuracy was about one-third of the CPU time required by the block Jacobi method.

Compared to the implicit multigrid approach, our method appears to be less sensitive to the cell aspect ratios in the computational mesh. Thus, our method is particularly attractive for problems involving highly stretched grids, as in the case of most Navier–Stokes computations or for grids in complex geometries.

As seen, we have used fairly high numbers of iterations for both the point Jacobi solver on the coarse grid and the line relaxation on the fine grid in each block. We therefore believe that our results are not greatly influenced by our choice of solvers and that any other suitable iterative linear solvers could have been used with similar results. Thus, it should be emphasized that the CGCS is not specific to our particular implicit scheme; it could be built on top of any implicit multiblock solver based on the solution to the linearized equations in each block.

Acknowledgments

The work of the first author was supported by the EC-ESPRIT IV program under Project No. 2011, and a part of the work was carried out while the first author was visiting Electricité de France

in Chatou, France. The work of the second author was supported by the Swedish Board for Industrial and Technical Development.

References

[1]Pulliam, T., "Time Accuracy and the Use of Implicit Methods," AIAA Paper 93-3360, June 1993.

[2]Venkatakrishnan, V., and Mavriplis, D., "Implicit Method for the Computation of Unsteady Flows on Unstructured Grids," AIAA Paper 95-1705, June 1995.

[3]Alonso, J., Martinelli, L., and Jameson, A., "Multigrid Unsteady Navier–Stokes Calculations with Aeroelastic Applications," AIAA Paper 95-0048, Jan. 1995.

[4]Jameson, A., "Time Dependent Calculations Using Multigrid, with Applications to Unsteady Flows Past Airfoils and Wings," AIAA Paper 91-1596, June 1991.

[5]Arnone, A., Liou, M., and Povinelli, L., "Multigrid Time-Accurate Integration of Navier–Stokes Equations," AIAA Paper 93-3361, June 1993.

[6]Crumpton, P., and Giles, M., "Implicit Time Accurate Solutions on Unstructured Dynamic Grids," AIAA Paper 95-1671, June 1995.

[7]Jenssen, C. B., "Implicit Multiblock Euler and Navier–Stokes Calculations," *AIAA Journal*, Vol. 32, No. 9, 1994, pp. 1808–1814.

[8]Jenssen, C. B., and Sørli, K., "A Parallel Implicit Time Accurate Navier–Stokes Solver," *Parallel Computational Fluid Dynamics: Implementations and Results Using Parallel Computers*, Elsevier, Amsterdam, 1996, pp. 625–632.

[9]Jenssen, C. B., and Weinerfelt, P. Å., "Coarse Grid Correction Scheme for Implicit Multiblock Euler Calculations," *AIAA Journal*, Vol. 33, No. 10, 1995, pp. 1816–1821.

[10]Weinerfelt, P. Å., "An Efficient Euler and Navier–Stokes Solver and Its Implementation on Parallel Computers with Local Memory," Dept. of Mathematics, Linköping Univ., LiTH-MAT-R-91-40, Linköping, Sweden, Aug. 1991.

[11]Hirsch, C., *Numerical Computation of Internal and External Flows*, Vol. 1, Wiley, New York, 1988, pp. 423–455.

[12]Wesseling, P., *An Introduction to Multigrid Methods*, Wiley, New York, 1992, pp. 60–78.

[13]Martinelli, L., and Jameson, A., "Validation of a Multigrid Method for the Reynolds Averaged Navier–Stokes Equations," AIAA Paper 88-0414, Jan. 1988.

[14]Belov, A., Martinelli, L., and Jameson, A., "A New Implicit Algorithm with Multigrid for Unsteady Incompressible Flow Calculations," AIAA Paper 95-0049, Jan. 1995.

[15]Allmaras, S. R., "Analysis of Semi-Implicit Preconditioners for Multigrid Solution of the 2-D Compressible Navier–Stokes Equations," AIAA Paper 95-1651, June 1995.

[16]Dongarra, J., "Performance of Various Computers Using Standard Linear Equations Software," Dept. of Computer Science, CS-89-85, Univ. of Tennessee, Knoxville, TN, March 1996.

[17]Williamson, C. H. K., "Defining a Universal and Continuous Strouhal–Reynolds Number Relationship for the Laminar Vortex Shedding of a Circular Cylinder," *Physics of Fluids*, Vol. 31, 1988, pp. 2742–2744.

[18]Ostenfeld, K. H., and Larsen, A., "Bridge Engineering and Aerodynamics," *Aerodynamics of Large Bridges*, Balkema, Rotterdam, The Netherlands, 1992, pp. 3–22.

[19]Larose, G. L., "The Response of a Suspension Bridge Deck to Turbulent Wind: the Taut Strip Model Approach," Ph.D. Thesis, Dept. of Civil and Environmental Engineering, Univ. of Western Ontario, London, ON, Canada, 1992.

J. Kallinderis
Associate Editor