AIAA JOURNAL
Vol. 33, No. 10, October 1995

Coarse Grid Correction Scheme for Implicit Multiblock Euler Calculations

Carl B. Jenssen*
SINTEF—Foundation for Scientific and Industrial Research, N-7034 Trondheim, Norway
and
Per Å. Weinerfelt†
Linköping University, S-58183 Linköping, Sweden

A coarse grid correction scheme is used to bring global influence to implicit multiblock calculations. Compared to using only explicit coupling between the blocks, a considerable reduction in the number of time steps needed to reach steady state has been observed for transonic and subsonic flows. The additional CPU time per time step associated with the method is small, resulting in a highly efficient parallel implicit scheme.

Introduction

IMPLICIT schemes generally offer higher efficiency than explicit schemes for steady-state calculations. Implicit schemes, however, have a sequential nature and are, therefore, often difficult to parallelize. As a result, implicit flow solvers usually display a loss of either computational or numerical efficiency as the number of utilized processors increases.

By far the most popular way of designing a parallel flow solver, whether it is based on a structured or an unstructured grid, has been to use some sort of multiblock, or domain decomposition, algorithm. The simplest implicit multiblock method is to use implicit time stepping within each block but with explicit coupling between the blocks. The resulting method, which has been used by several authors,[1–4] can be regarded as a non-linear block Jacobi method.[5] It has been shown by Jenssen[6,7] that this approach can be made to work very well even for a large number of blocks, for both the Euler and Navier–Stokes equations as long as the flow is supersonic. For subsonic flow, however, the number of time steps needed to obtain a converged solution grows as the number of blocks is increased. Although the loss of convergence is only moderate for a small number of blocks, the method is not scalable to a large number of processors.

The reason for this behavior lies in the global, or elliptic, nature of the governing equations in the case of subsonic flow. In the framework of domain decomposition methods, Dryja and Widlund,[8] among others, have proposed solving a global coarse grid system as a way of retaining global influence in implicit multiblock methods. Cai et al.[9] recently applied preconditioners based on the solution of a coarse grid system when solving model problems. Trials have also been made by Venkatakrishnan[10] to incorporate these ideas into algorithms for solving the two-dimensional Euler equations, using a coarse grid consisting of one grid cell per block to obtain boundary conditions for a block incomplete lower–upper factorization (ILU) preconditioner.

We have used a slightly different approach, formulating a standard coarse grid correction scheme, or two-level multigrid algorithm. Thus, at each time step, the equations are first solved independently in each block using explicit boundary conditions obtained from the values at previous time step, and then a correction step is performed in which a solution to a global coarse grid system is used to improve the fine grid solution.

In this paper we describe this method and address the problem of parallel efficiency. Both the number of time steps required to obtain a converged solution, and the total CPU time on an Intel Paragon and a Cray Y-MP will be considered. The results are based on three-dimensional subsonic and transonic flow problems. As the elliptic nature of the subsonic steady-state problem is an effect of the Euler equations, we only consider cases of inviscid flow.

The flow solver is based on a finite volume formulation on a cell centered structured multiblock mesh. Roe's scheme with second-order total variation diminishing (TVD) extrapolation is used for the spatial discretization, whereas the resulting linear system within each block is solved using line relaxation.

Governing Equations

The Euler equations can be found in any textbook on fluid flow; thus, we give only a general symbolic form here. Consider a system of conservation laws written in integral form

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_{\Omega} U\,\mathrm{d}V + \int_{\partial\Omega} \boldsymbol{F}\cdot\boldsymbol{n}\,\mathrm{d}s = 0 \tag{1}$$

Here U is the conserved variable, $\boldsymbol{F}$ the flux vector, and $\boldsymbol{n}$ the outward pointing normal to the surface $\partial\Omega$ enclosing an arbitrary volume Ω. We assume that the Jacobian of $\boldsymbol{F}\cdot\boldsymbol{n}$ with respect to U has real eigenvalues and a complete set of eigenvectors.

Space Discretization

The fluxes are discretized with a TVD scheme based on Roe's scheme with second-order monotone upwind schemes for conservation laws (MUSCL) extrapolation.[11] Considering a system of non-linear hyperbolic equations in one dimension, this scheme is briefly reviewed here. The extension to two or three dimensions is straightforward by applying the one-dimensional scheme independently in each coordinate direction.

For ease of notation we write $F(U) = \boldsymbol{F}(U)\cdot\boldsymbol{n}$ and let $F_{i+1/2}$ denote the flux through the surface $S_{i+1/2}$. By Roe's scheme, the numerical flux is then given by

$$F_{i+\frac{1}{2}} = \tfrac{1}{2}\left[F\left(U^{-}_{i+\frac{1}{2}}\right) + F\left(U^{+}_{i+\frac{1}{2}}\right)\right] - \tfrac{1}{2}\left(A^{+}_{i+\frac{1}{2}} - A^{-}_{i+\frac{1}{2}}\right)\left(U^{+}_{i+\frac{1}{2}} - U^{-}_{i+\frac{1}{2}}\right) \tag{2}$$

Here $U^{-}_{i+1/2}$ and $U^{+}_{i+1/2}$ are extrapolated values of the conserved variable at the left and right side of the surface and $A^{\pm}_{i+1/2}$ are given by

$$A^{\pm}_{i+\frac{1}{2}} = R_{i+\frac{1}{2}}\Lambda^{\pm}_{i+\frac{1}{2}}R^{-1}_{i+\frac{1}{2}} \tag{3}$$

where $R_{i+1/2}$ is the right-eigenvector matrix of $A_{i+1/2}$, which is the

Received Nov. 26, 1994; presented as Paper 95-0225 at the AIAA 33rd Aerospace Sciences Meeting, Reno, NV, Jan. 9–13, 1995; revision received March 20, 1995; accepted for publication March 22, 1995.

*Research Scientist, Department of Industrial Mathematics, Norwegian Institute of Technology. Member AIAA.

†Associate Professor, Department of Applied Mathematics. Member AIAA.

Jacobian of F taken at $U_{i+1/2}$. The state $U_{i+1/2}$ is given by Roe's approximate Riemann solver defined by the equation

$$F\left(U^+_{i+\frac{1}{2}}\right) - F\left(U^-_{i+\frac{1}{2}}\right) = A\left(U_{i+\frac{1}{2}}\right)\left(U^+_{i+\frac{1}{2}} - U^-_{i+\frac{1}{2}}\right) \quad (4)$$

where $A(U)$ is the Jacobian to $F(U)$. The diagonal matrices $\Lambda^{\pm}_{i+1/2}$ are the split eigenvalue matrices corresponding to $R_{i+1/2}$. To avoid entropy violating solutions, the eigenvalues are split according to

$$\lambda^{\pm}_l = \tfrac{1}{2}\left(\lambda_l \pm \sqrt{\lambda_l^2 + \varepsilon^2}\right) \quad (5)$$

where ε is a small parameter.

The first-order version of Roe's scheme is obtained by simply setting $U^-_{i+1/2} = U_i$ and $U^+_{i+1/2} = U_{i+1}$. Schemes of higher order are obtained by defining $U^-_{i+1/2}$ and $U^+_{i+1/2}$ by higher order extrapolation. The scheme used here is based on a second-order minmod limited extrapolation of the characteristic variables, resulting in a scheme that is fully upwind.

Time Integration

The steady-state solution to the discretized equations are obtained by implicit local time stepping. Since time accuracy is not required, several simplifications to the implicit operator can be made. First, consider a fully implicit time discretization again in one dimension expressed in the so-called delta form

$$(V_i/\Delta t)\Delta U_i + \Delta F_{i+\frac{1}{2}} - \Delta F_{i-\frac{1}{2}} = -\left(F^n_{i+\frac{1}{2}} - F^n_{i-\frac{1}{2}}\right) \quad (6)$$

Here, V_i is the volume of the grid cell i and Δt the time step. Based on the first-order version of Roe's scheme, the flux vectors are approximately linearized resulting in a linear system of equations that in three dimensions is septadiagonal in each block

$$\begin{aligned} &C_0\Delta U_{i,j,k} + C_{-1}\Delta U_{i-1,j,k} + C_1\Delta U_{i+1,j,k} \\ &\quad + C_{-2}\Delta U_{i,j-1,k} + C_2\Delta U_{i,j+1,k} \\ &\quad + C_{-3}\Delta U_{i,j,k-1} + C_3\Delta U_{i,j,k+1} = \mathrm{RHS}_{i,j,k} \end{aligned} \quad (7)$$

where for the Euler equations the various C are 5×5 matrices and the unknowns $U_{i,j,k}$ and the right-hand side $\mathrm{RHS}_{i,j,k}$ are vectors of size 5. The indices i, j, and k of the various C have been dropped for clarity.

In each block, the system is solved using a line Jacobi procedure which is a three-dimensional vectorizable extension of the conventional two-dimensional line Gauss–Seidel method.[12] Solving along all lines in a given coordinate direction simultaneously, the procedure can be vectorized across the tri-diagonal systems yielding a vector length equal to the number of points in a computational plane. Assuming we have an approximate solution ΔU^m to Eq. (7), we can find a new approximate in three steps. First, $\Delta U^{m+1/3}$ is found by solving along the i lines:

$$\begin{aligned} &C_{-1}\Delta U^{m+1/3}_{i-1,j,k} + C_0\Delta U^{m+1/3}_{i,j,k} + C_1\Delta U^{m+1/3}_{i+1,j,k} \\ &\quad = \mathrm{RHS}_{i,j,k} - C_{-2}\Delta U^m_{i,j-1,k} - C_2\Delta U^m_{i,j+1,k} \\ &\quad - C_{-3}\Delta U^m_{i,j,k-1} - C_3\Delta U^m_{i,j,k+1} \end{aligned} \quad (8)$$

and, then, similar equations are solved for $\Delta U^{m+2/3}$ and ΔU^{m+1} along the j and k directions, always utilizing the latest available iterate of the solution on the right-hand side. A number of iterations can now be performed by repeating the three steps.

The somewhat high-memory requirement of this method is controlled by solving blocks in a sequential manner, using the same work space. In a multiprocessor environment, this can be achieved by mapping several blocks to each processor.

Coarse Grid Correction Scheme

The block Jacobi method consists of, at each time step, solving Eq. (7) independently in each block assuming $\Delta U = 0$ in the neighbor blocks. The system can be solved using the line relaxation procedure described or any other suitable method. The idea behind the present method is to formulate a coarse grid problem that we are able to solve as a fully coupled global problem. The coarse grid solution will then be used to improve the fine grid solution.

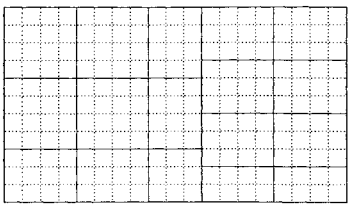

Fig. 1 Schematic view of the fine and coarse grids in two blocks.

Thus, the method can be formulated as a coarse grid correction scheme (CGC) or a two-level multigrid algorithm applied to a linear system of equations. What distinguishes the present method from a conventional multigrid method is that it consists of just one fine and one very coarse grid. In addition a solver, and not a smoother, is employed both on the coarse grid and in each block on the fine grid.

The coarse mesh is obtained by dividing the fine mesh into a suitable number of cells by removing grid lines as shown in Fig. 1. The only additional constraint we impose on the coarse mesh is that no cells overlap block boundaries.

Using the subscript h to denote the fine mesh and H for the coarse mesh, the method is described in the following. For ease of notation we use W for the unknowns ΔU of Eq. (7). The linear system of equations that we wish to solve at each time step can then be written in matrix form as

$$A_h W_h = R_h \quad (9)$$

By ignoring the implicit coupling between blocks, however, we actually solve a different system that we symbolically can write as

$$\tilde{A}_h \tilde{W}_h = R_h \quad (10)$$

Defining the error obtained by not solving the correct system as $\varepsilon_h = W_h - \tilde{W}_h$, we have

$$A_h \varepsilon_h = R_h - A_h \tilde{W}_h \quad (11)$$

In a classic multigrid fashion we now formulate a coarse grid representation of Eq. (11) based on a restriction operator I^H_h and prolongation operator I^h_H. Thus, we define

$$A_H = I^H_h A_h I^h_H \quad (12)$$

$$R_H = I^H_h (R_h - A_h \tilde{W}_h) \quad (13)$$

and solve

$$A_H \Delta W_H = R_H \quad (14)$$

The solution to Eq. (14) is transformed to the fine grid by means of the prolongation operator

$$\Delta W_h = I^h_H \Delta W_H \quad (15)$$

and, finally, the fine grid solution is updated by

$$\tilde{\tilde{W}}_h = \tilde{W}_h + \Delta W_h \quad (16)$$

As the restriction operator we use summation and as prolongation operator injection.[13] Thus, the coarse grid right-hand side is found simply by summing the residual $R_h - A_h \tilde{W}_h$ over all fine grid cells corresponding to a coarse grid cell. Similarly, the left-hand side (LHS) is found by a summation of appropriate elements in the fine grid operator A_h. The interested reader might note that a conservative formulation of the fine grid LHS might simplify the formation of the LHS on the coarse grid, as several terms will cancel out.

In a conventional multigrid approach, the correction step defined by Eq. (16) would have been followed by another fine grid solution to smooth the corrected values. In our case this final step is omitted,

as we allow several blocks to be solved on each processor using the same workspace and thereby overwrite the equation system.

Only the matrix elements corresponding to the coupling between the blocks have to be saved to evaluate the contributions to Eq. (14) from neighboring blocks. To see this, we can write the right-hand side of Eq. (11) as

$$R_h - A_h\tilde{W}_h = R_h - \tilde{A}_h\tilde{W}_h - (A_h - \tilde{A}_h)\tilde{W}_h \qquad (17)$$

Here, the term $R_h - \tilde{A}_h\tilde{W}_h$ is zero by Eq. (10) whereas $(A_h - \tilde{A}_h)$ consists only of the matrix elements we have discarded when solving Eq. (10) instead of Eq. (9). If Eq. (10) is only solved approximately, which to some extent always will be the case, then the term $R_h - \tilde{A}_h\tilde{W}_h$ can be evaluated block by block when the corresponding part of the equation system is in the memory. In the latter case, we also find a correction to the error made in each block, as with a standard multigrid method.

To solve the coarse grid system, any suitable parallel sparse matrix solver can be applied. We have used a Jacobi type iterative solver that inverts only the 5×5 diagonal coefficients in the coarse grid system. In practice, 25 iterations with this solver have been sufficient. Thus, whenever we have used the method, the cost of actually solving the coarse grid system has been smaller than the cost of creating it.

Test Cases

In the following the method is tested with practical calculations of realistic three-dimensional flowfields. In each case we use the block Jacobi method as a reference. Only subsonic flows are considered, as the block Jacobi method can be made to work very well for supersonic flows.[6] For subsonic flows, it has been shown that better results can be obtained using a red–black Gauss–Seidel procedure,[6] however, the improvement is not dramatic, and as the number of blocks is increased, this approach also shows asymptotic behavior similar to that of the block Jacobi method.

Transonic Flow over a Delta Wing

The first test case is the flow over the well-known delta wing model, made up of NACA 64A005 chord sections, having 65-deg sweep and cropped at 85% of the root chord. The transonic flow at Mach 0.85 and 10-deg angle of attack for which experimental data exist[14] have been calculated, as well as the purely subsonic flow at Mach 0.6 and 1.5-deg angle of attack. See Fig. 6 for a view of the wing geometry and surrounding mesh.

First, an extensive study of the convergence rate of the various methods for different number of blocks is carried out on a coarse mesh for the transonic flow case. Then, selected fine mesh computations have been done for both Mach numbers to confirm that the results carry over to larger problems, as well as to validate the correctness of the code.

In Fig. 2, convergence histories for the block Jacobi method using 1, 8, 64, or 512 blocks are shown. The mesh used was made up of $32 \times 18 \times 40$ cells in the streamwise, normal, and circumferential directions. Although convergence is just slightly slower in the 8-block case, the result for 64 blocks is quite discouraging, whereas using 512 blocks causes unacceptably slow convergence.

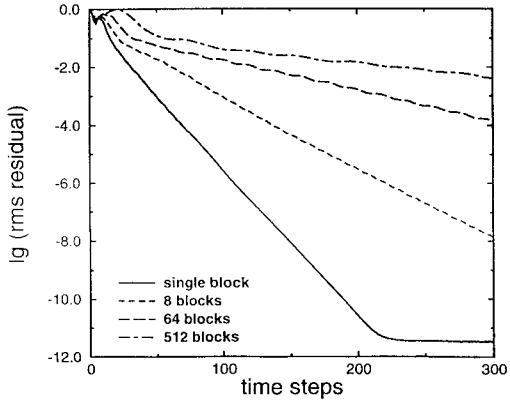

Fig. 2 Convergence histories for the solutions of the Euler equations over a delta wing using the block Jacobi procedure.

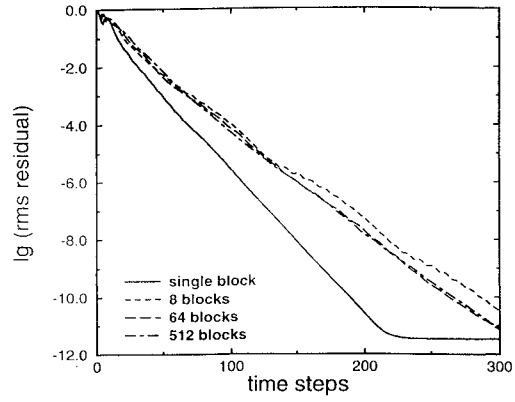

Fig. 3 Convergence histories for the solutions of the Euler equations over a delta wing using the coarse grid correction scheme.

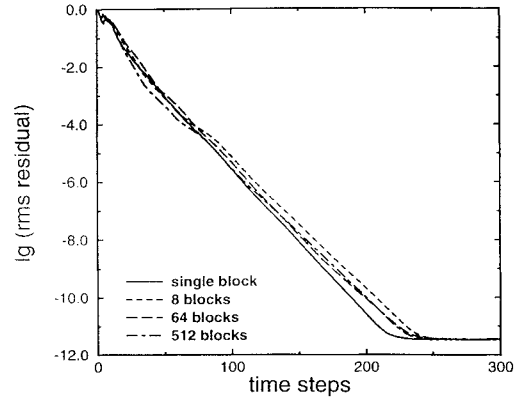

Fig. 4 Convergence histories for the solutions of the Euler equations over a delta wing using 8 coarse grid cells per block.

The convergence histories for the simplest possible form of the CGC scheme, i.e., with only one coarse grid cell per block, can be seen in Fig. 3. As we can see, the CGC scheme leads to a significant improvement. All multiblock computations now require about 50% more time steps than the single-block case. Both when using 64 or 512 blocks it is clear that any computational overhead associated with the method cannot outweigh the observed reduction in number of time steps required.

We have seen that the original uncoupled method works better as the number of blocks decreases. The corrections, however, will become better, although more expensive to solve, as the number of blocks increases and the coarse grid becomes finer. This seems to offer a perfect balance, but it also indicates how to improve the convergence even further. There is no reason why one should restrict the number of coarse grid cells to the number of blocks. If a parallel solver is employed to solve the coarse grid system, one can afford a much larger number of coarse grid cells. The strategy should, therefore, be to choose the number of blocks according to the number of available processors and memory and the size of the coarse grid system as the maximum that can be solved efficiently.

To demonstrate this idea, we have performed another set of calculations with the same number of blocks, but with a coarse grid consisting of 8 cells per block. From Fig. 4 we can see that this indeed leads to a further improvement. Now all multiblock computations require nearly the same number of time steps as the single-block method. This dependency on the size of the coarse mesh is also in agreement with the observations made by other authors[9] using related methods on model problems.

To verify that the effects observed with the coarse grid carry over to finer grids, we now present calculations on a more realistic grid with twice as many grid cells in each direction, or 184,320 in total.

Table 1 Total CPU times on a 56 processor Intel Paragon for the delta wing at Mach 0.85

Method	CG size	s/step	Steps	CPU time, h:min
BJ		13.8	874	3:21
CGC	56	14.6	1227	4:58
CGC	1792	14.9	185	0:46

Table 2 Total CPU times on a 56 processor Intel Paragon for the delta wing at Mach 0.6

Method	CG size	s/step	Steps	CPU time, h:min
BJ		13.8	1044	4:00
CGC	56	14.6	870	3:31
CGC	1792	14.9	151	0:37

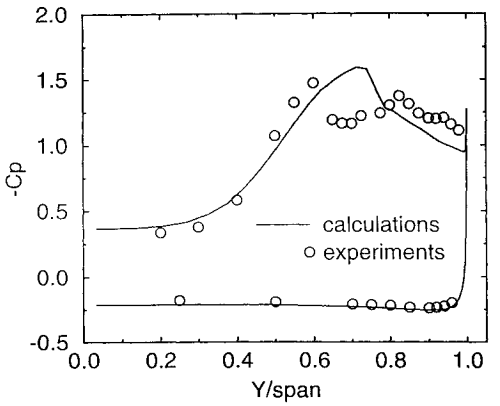

Fig. 5 Pressure coefficient for Mach 0.85 on the delta wing at a spanwise cut at 80% of the root chord.

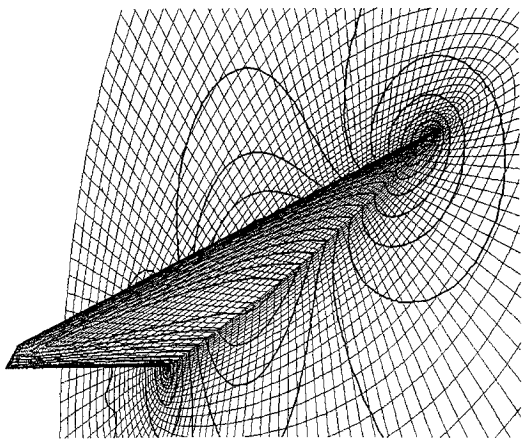

Fig. 6 Pressure contours on the surface of the delta wing and in the symmetry plane at Mach 0.6 and 1.5-deg angle of attack.

A comparison of experimental and calculated values of the pressure coefficient in a spanwise cut along the wing for Mach 0.85 is shown in Fig. 5. The computational results compare with the measurements as expected for an Euler solution which is able to capture a primary vortex induced by the sharp leading edge but not secondary vortices created by viscous effects. Calculated pressure contours for Mach 0.6 are shown in Fig. 6.

Using 56 blocks and 1792 coarse grid cells, which is 36 per block, we see from Tables 1 and 2 that the reduction in time steps as well as total CPU time is significant compared with the block Jacobi (BJ) method. In all cases the convergence criterion was a six decade reduction of the L_2 norm of the residual.

Contrary to what was seen with the coarse mesh, using only one coarse grid cell per block does not result a substantial reduction of time steps. In the transonic test case, this actually increases both the number of time steps and CPU time to reach the convergence criterion. With a finer coarse grid, however, the speedup is considerable, reducing the CPU time down to less than 23% for the Mach 0.85 case and to nearly 15% for the Mach 0.6 test case.

Table 3 Total CPU times on Cray Y-MP for the delta wing at Mach 0.85

Method	CG size	s/step	Steps	CPU time, h:min
BJ		26.7	900[a]	6:40
CGC	512	30.7	186	1:35

[a]Extrapolated.

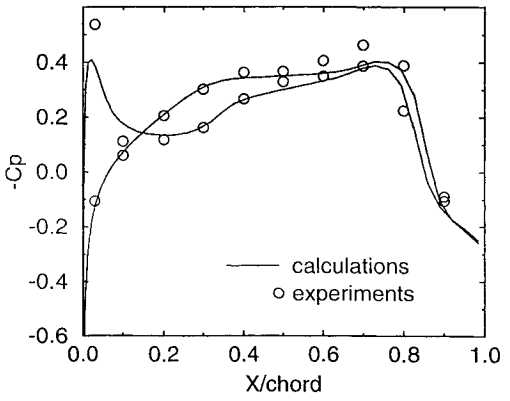

Fig. 7 Pressure coefficient for Mach 0.95 on the NLR F5 wing at 0.841 of span.

The efficiency of the CGC scheme on conventional shared memory machines is demonstrated by two calculations on the Cray Y-MP, one with and one without the CGC scheme. Eight coarse grid cells per block were used. It should be noted that multiple blocks are necessary on the Cray, not only for parallel processing, but also to reduce the memory requirement. Thus, using 64 blocks on four processors, the memory requirement was about 14 Megawords.

The results, in terms of number of time steps and the equivalent single-processor CPU time are shown in Table 3. The calculation with the CGC scheme was terminated after a six decade reduction of the residual corresponding to 1 h 35 min CPU time. In order not to waste too many CPU hours, the results of the block Jacobi calculations were extrapolated based on 300 time steps. We can see that the CGC scheme amounts to an increase in the CPU time per time step of only 15%, again resulting in an overall speedup of more than a factor four.

Subsonic Flow over an NLR F5 Wing

The final test case concerns the flow over the NLR F5 wing. Calculations have been carried out for Mach 0.95, 0.6, and 0.15, all at 0-deg angle of attack. The mesh used was of a C–H type, consisting of 128 × 18 × 24 cells in the streamwise, normal, and spanwise directions, totaling 55,296 grid cells. Experimental results exist for the transonic case of Mach 0.95 and the purely subsonic flow at Mach 0.6. Calculated and experimental values of the pressure coefficient on the wing in a chordwise cut are shown in Figs. 7 and 8 for these flowfields. In spite of the fairly coarse mesh, the calculations agree well with the experiments. Selected mesh lines are shown in Fig. 9, together with computed Mach number contours for Mach 0.15.

The calculations were carried out on the Intel Paragon, using 56 blocks on 56 processors.The CGC scheme was employed with 1, 8, or 36 coarse grid cells per block, corresponding respectively to a total of 56, 448, or 2016 coarse grid cells. In Tables 4–6, the number of time steps and total CPU time to reach steady state are compared to the results obtained when using the block Jacobi method. For this test case, the mesh is small enough to fit a single-block calculation on the Cray Y-MP, and thereby makes a comparison with a fully coupled single block calculation possible. The number of time steps required by the single-block (SB) calculations are, therefore, also included in Tables 4–6. To compensate for the explicit coupling across the wake cut in the C–H mesh, the CGC scheme with 448 coarse grid cells was employed for single-block calculations as well.

Table 4 Total CPU times on a 56 processor Intel Paragon for the F5 wing at Mach 0.95

Method	CG size	s/step	Steps	CPU time, min
BJ		3.88	472	30.5
CGC	56	4.44	286	21.2
CGC	448	4.56	221	16.8
CGC	2016	4.93	193	15.8
SB	448		165	

Table 5 Total CPU times on a 56 processor Intel Paragon for the F5 wing at Mach 0.6

Method	CG size	s/step	Steps	CPU time, min
BJ		3.88	513	33.2
CGC	56	4.44	269	19.9
CGC	448	4.56	186	14.1
CGC	2016	4.93	161	13.2
SB	448		156	

Table 6 Total CPU times on a 56 processor Intel Paragon for the F5 wing at Mach 0.15

Method	CG size	s/step	Steps	CPU time, min
BJ		3.88	445	28.8
CGC	56	4.44	370	27.4
CGC	448	4.56	276	21.0
CGC	2016	4.93	211	17.3
SB	448		185	

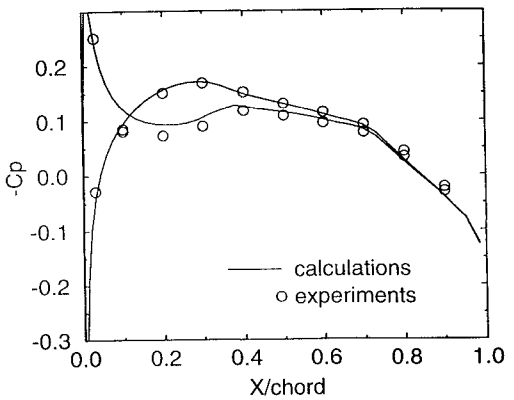

Fig. 8 Pressure coefficient for Mach 0.6 on the NLR F5 wing at 0.841 of span.

As before, the convergence criterion was a six decade reduction of the L_2 norm of the residual.

Using 56 coarse grid cells per block, the reduction in CPU time relative to the block Jacobi method is 52% for the transonic case, 40% for Mach 0.6, and, finally, 60% for the nearly incompressible case with Mach 0.15. Again we see that using only one coarse grid cell per block is clearly less efficient. This is especially apparent for Mach 0.15, where in this case only a very small decrease in CPU time was observed.

We also see that with the CGC scheme, using 56 blocks results in an increase in the time step count compared to the fully coupled single-block case in the range of 3%–14%. Again this is in sharp contrast to what is experienced with the block Jacobi method, where the increase in number of time steps is by a factor of 2.4–3.3.

Conclusion

The use of a coarse grid correction scheme to bring global influence to implicit multiblock calculations has been investigated. We have demonstrated that for subsonic flow this can lead to a significant reduction in the number of time steps for steady-state calculations compared to using the block Jacobi method. The additional CPU time per time step associated with the method is small, resulting in a considerable reduction in total CPU time.

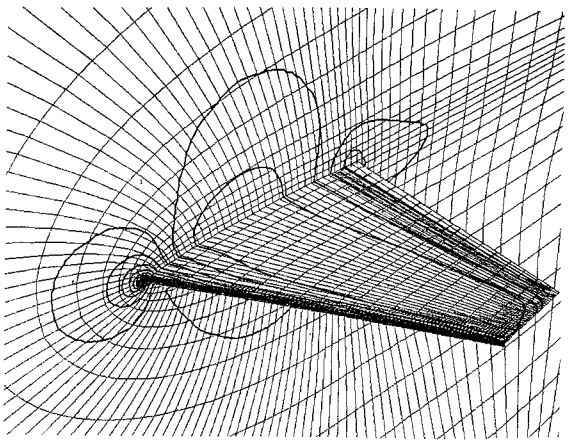

Fig. 9 Mach number contours around the NLR F5 wing at Mach 0.15.

In principle, the CGC scheme can be built on top of any implicit scheme. As the coarse grid operator is formulated based only on the original linear system of equations, no coarse grid has to be explicitly created. Also, the memory requirement of the method can be controlled by allowing several blocks to be solved on each processor using the same workspace. Only the matrix elements corresponding to the coupling between the blocks have to be saved to form the coarse grid system.

In our tests, using 56 or 64 blocks, the increase in CPU time per time step was in the range of 10%–20% on shared as well as distributed memory machines. Thus, an overall speedup of 3–4 was observed in all cases except in the incompressible limit where the speedup was about 1.8.

Whereas the convergence rate for the block Jacobi method deteriorates with increasing number of blocks, the tests we have carried out show that the number of time steps needed to reach steady state with the CGC scheme depends only to a very small degree on the number of blocks used. Thus, the method is scalable to a large number of processors, and the speedup over the block Jacobi method will increase with increasing number of blocks.

A crucial factor in obtaining maximum overall efficiency with the method is to use more than one coarse grid cell per block. In some cases, using only one coarse grid cell per block did not result in any significant reduction of the total CPU time, and in one case it even resulted in an increase in both number of time steps and CPU time. The strategy should, therefore, be to choose the number of blocks according to the number of available processors and memory, and the size of the coarse grid system as the maximum that can be solved efficiently with a parallel solver.

It is also clear that the relative expense of employing the CGC scheme depends on the implicit solver used within each block. For a CPU-intensive solver resulting in convergence in relatively few time steps, the overhead associated with the CGC method would be small, whereas for a solver based on a larger number of time steps, each at a lower cost, the CGC scheme would be relatively more expensive. The line relaxation technique that we have used in this work, therefore, seems like a good choice, allowing for convergence in a limited number of large time steps.

Acknowledgments

The work of the first author was supported by the Research Council of Norway. The work of the second author was supported by the Swedish National Board for Industrial and Technical Development.

References

[1]Flores, J., "Simulation of Transonic Viscous Wing and Wing-Fuselage Flows Using Zonal Methods," *Proceedings of the First International Symposium on Domain Decomposition Methods for Partial Differential Equations*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1988, pp. 181–200.

[2]Walters, R., and Thomas, J., "A Patched-Grid Algorithm for Complex Aircraft Configurations," *Proceedings of the Third International Symposium on Domain Decomposition Methods for Partial Differential Equations*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990, pp. 397–416.

[3]Yadlin, Y., and Caughey, D., "Block Multigrid Implicit Solution of the Euler Equations of Compressible Fluid Flow," *AIAA Journal*, Vol. 29, No. 5, 1991, pp. 712–719.

[4]Sahu, J., and Steger, J., "Numerical Simulation of Three-Dimensional Transonic Flows," *International Journal for Numerical Methods in Fluids*, Vol. 10, No. 8, 1990, pp. 855–873.

[5]Golub, G., and Ortega, J., *Scientific Computing, An Introduction with Parallel Computing*, Academic, New York, 1993, pp. 337, 338.

[6]Jenssen, C., "Implicit Multi Block Euler and Navier–Stokes Calculations," *AIAA Journal*, Vol. 32, No. 9, 1994, pp. 1808–1814; AIAA Paper 94-0521.

[7]Jenssen, C., "A Parallel Block-Jacobi Euler and Navier–Stokes Solver," *Proceedings of the Conference on Parallel Computational Fluid Dynamics '93*, Elsevier, Amsterdam, The Netherlands, pp. 307–314.

[8]Dryja, M., and Widlund, O., "Towards a Unified Theory of Domain Decomposition Algorithms for Elliptic Problems," *Proceedings of the Third International Symposium on Domain Decomposition Methods for Partial Differential Equations*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990, pp. 3–21.

[9]Cai, X., Gropp, W., and Keyes, D., "A Comparison of Some Domain Decomposition Algorithms for Nonsymmetric Elliptic Problems," *Proceedings of the Fifth International Symposium on Domain Decomposition Methods for Partial Differential Equations*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1992, pp. 224–235.

[10]Venkatakrishnan, V., "Parallel Implicit Unstructured Grid Euler Solvers," AIAA Paper 94-0759, Jan. 1994.

[11]Chakravarthy, S., "High Resolution Upwind Formulations for the Navier–Stokes Equations," *Proceedings from the VKI Lecture Series 1988-05*, Von Karman Institute of Fluid Dynamics, Brussels, Belgium, 1989, pp. 58–63.

[12]MacCormack, R., "Current Status of Numerical Solutions of the Navier–Stokes Equations," AIAA Paper 85-0032, Jan. 1985.

[13]Wesseling, P., *An Introduction to Multigrid Methods*, Wiley, New York, 1992, pp. 66–73.

[14]Hirdes, R., "US/European Vortex Flow Experiment: Test Report of Wind Tunnel Measurements on the 65 deg Wing in NLR High Speed Wind Tunnel HST," National Aerospace Lab., NLR TR 85046 L, Amsterdam, The Netherlands, 1985.