Block Multigrid Implicit Solution of the Euler Equations of Compressible Fluid Flow

Yoram Yadlin* and David A. Caughey†
Cornell University, Ithaca, New York 14853

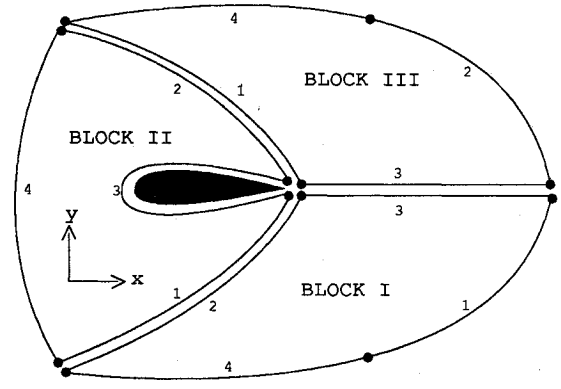
A multigrid diagonal implicit algorithm has been developed to solve the Euler equations of inviscid compressible flow on block structured grids. Two modes of advancing the multigrid cycle have been examined with respect to convergence rates, accuracy, and efficiency. The algorithm has been designed to run on parallel computers. Results are computed for transonic flows past airfoils and include pressure distributions to verify the accuracy of the scheme and convergence histories to demonstrate the efficiency of the method. Efficiencies that were obtained using a modest number of processors in parallel are presented and discussed.

Introduction

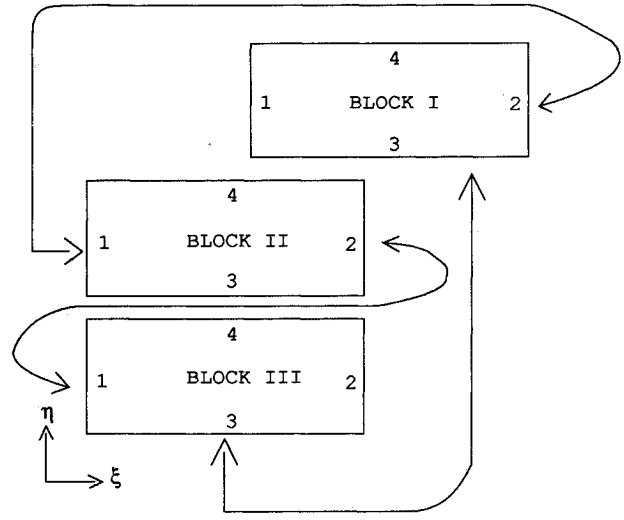
THE development of computers and efficient algorithms has made feasible the simulation of flows around high-performance aircraft, yet a number of difficulties still remain, one of them being the generation of an appropriate grid to be employed in the simulation. One approach to the problem of generating a grid for a complex configuration is the composite block structure employed, for example, in the EAGLE code.[1] In this approach, the physical domain is divided into a set of subdomains, each corresponding to a rectangular computational block. The grid generated by this method can then be used as an input for a flow solver algorithm.[2] The works of Karman et al.[3] and Weatherill et al.[4] are other examples of the block structure approach. A similar approach to handle complex geometries and to take advantage of the accelerated convergence achieved by using multigrid has been developed by Frese[5] for the solution of magnetic diffusion problems. The high degree of accuracy and efficiency of the diagonal implicit multigrid (DIM) algorithm[6,7] has motivated the present implementation of the DIM scheme within the framework of a composite block structure grid. In this paper, the implementation of the DIM scheme on a composite block structure grid will be described, and the results of calculations of flows past two-dimensional airfoils will be presented to demonstrate the accuracy and efficiency of the scheme and to describe the implementation of the algorithm on parallel computers.

Description of Algorithm

The Euler equations are approximated using a finite-volume spatial discretization on the mesh cells corresponding to a boundary-conforming curvilinear coordinate system. Artificial dissipation is add as a blend of second and fourth differences of the solution to prevent excessive oscillations of the solution in the vicinity of shock waves and to ensure convergence to a steady state. The time-linearized implicit operator is approximated as the product of two one-dimensional factors, each of which is then diagonalized by a local similarity transformation, so that only a decoupled system of scalar pentadiagonal equations needs to be solved along each line. The resulting method has good high wave-number damping and thus is a good smoothing algorithm to be used in conjunction with the multigrid method. A complete description of this algorithm for two-dimensional problems is given in Ref. 6. Only those aspects relevant to the composite block structure implementation will be described in this section.



Physical Domain



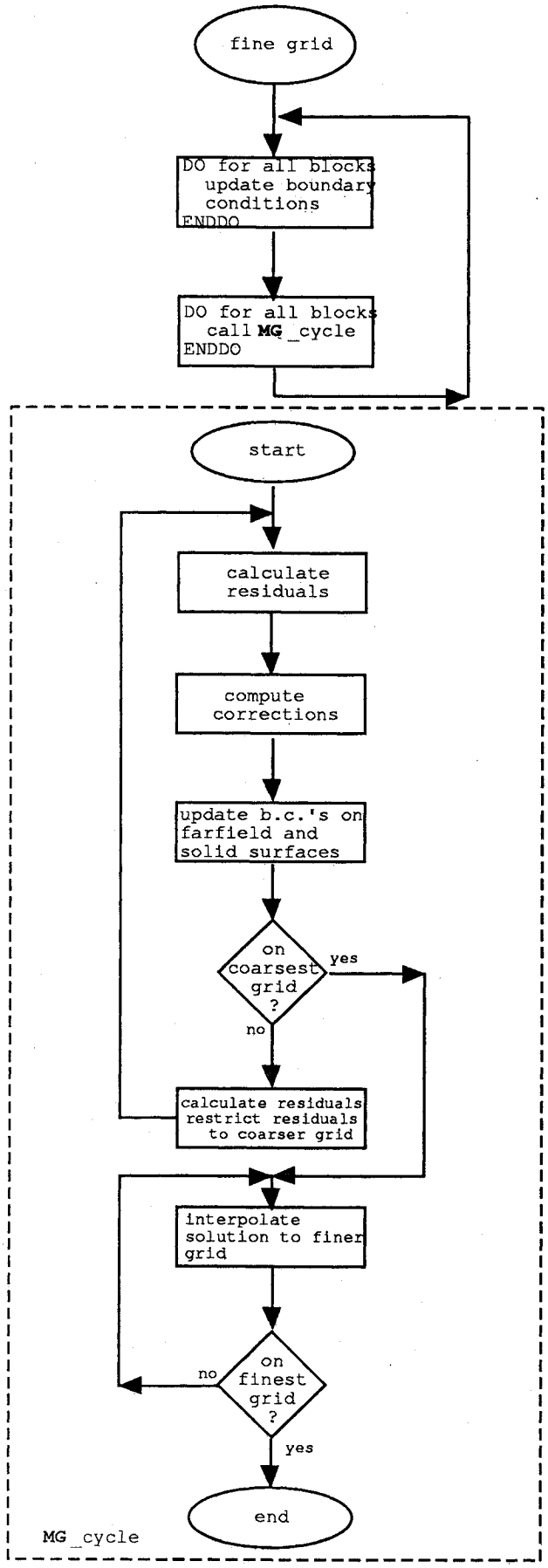
Computational Domain

Fig. 1 Domain decomposition of physical and computational spaces.

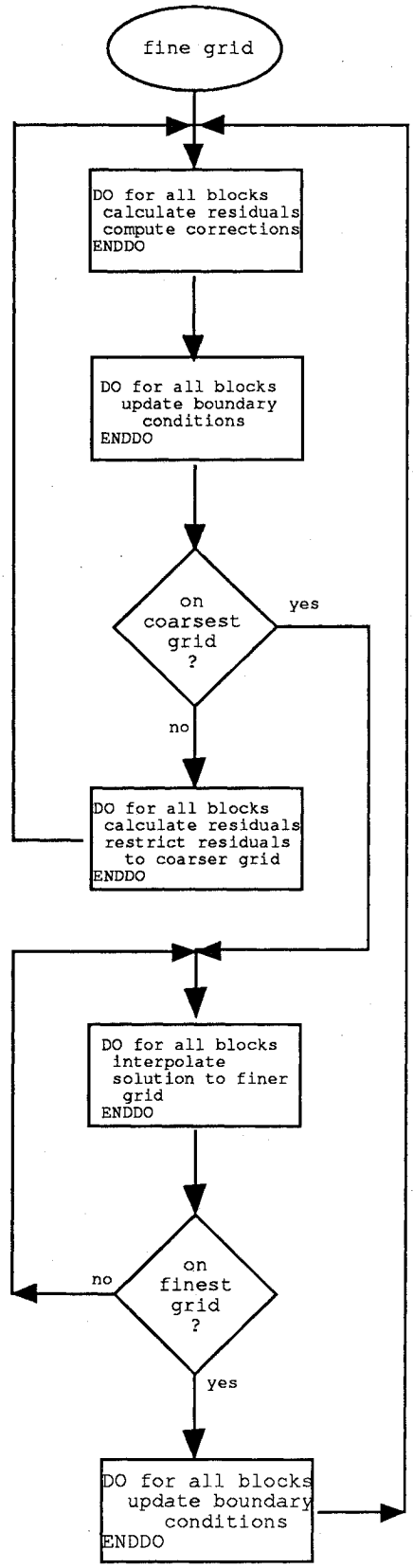
Received Nov. 8, 1989; revision received March 15, 1990.

*Graduate Research Assistant; currently Postdoctoral Associate, Cornell Theory Center. Student Member AIAA.

†Professor. Associate Fellow AIAA.



a) Vertical Mode



b) Horizontal Mode

Fig. 2 Flow chart for multigrid cycles (sawtoothed cycle, one iteration on each grid level): a) vertical mode; b) horizontal mode.

Domain Decomposition

The physical domain is divided into subdomains, which are represented by blocks in the computational domain. Each block has four faces (six in three dimensions), with its own coordinate system (ξ,η) in the computational space. The faces are numbered as illustrated in Fig. 1.

Each block is defined with a layer of dummy cells around it, which are used to enforce the boundary conditions; when two faces of neighboring blocks coincide, these layers create a region of overlapping. The information required at each face is as follows: 1) block number, 2) neighboring face number (if applicable) and the orientation of its coordinate system, and 3) type of boundary conditions. This information is stored in a set of two-dimensional integer arrays, created as input for the flow solver by the grid-generation code. In the present implementation, it is required that the grid distribution on coincident faces be identical.

Multigrid

The multigrid scheme on a composite block structure grid can be implemented in at least two modes: 1) the "horizontal" mode in which the multigrid cycle is advanced concurrently in all the blocks and 2) the "vertical" mode in which the multigrid cycle is advanced in each block independently. The main difference between the two modes involves the interaction between the blocks during the multigrid cycle. In the horizontal mode, all of the blocks are in phase during the cycle, hence the data exchange between the blocks (i.e., the updating of boundary conditions on interfaces) can be done easily at each grid level in the cycle. On the other hand, in the vertical mode the blocks are synchronized only at the beginning and/or end of each cycle, allowing for data exchange only once in the cycle. It is possible to update boundary conditions on the interfaces, but since each cycle advances independently, problems arise, involving asynchronous updating, more message passing when using a distributed memory architecture, and more access contention to main memory in a shared memory architecture. In the present implementation, there is no updating of boundary conditions on interfaces in the vertical mode, which means that those boundary conditions are frozen during the entire cycle. It was also found that for both modes it is unnecessary to update the far-field boundary conditions on the coarser grids; they can be frozen during the entire cycle with no significant degradation in convergence rate. Flow charts depicting examples of horizontal and vertical multigrid cycles are illustrated in Fig. 2.

Data Structure

The data structure for the composite block-multigrid algorithm is an extension of the multigrid data structure. All of the unknowns, coordinates, cell areas (volumes in three dimensions), time steps, etc., are stored in one-dimensional arrays. The arrays are organized by blocks; the unknowns of the first block are stored at the beginning of the array, followed by those of the second block, and so on. Within each block, the data are organized by grid levels, as is the case in the single-block multigrid scheme.

The position of the first entry of each block is calculated according to the number of blocks and grid levels in the multigrid cycle and is stored in an integer array, as illustrated in Fig. 3. This data structure allows access to each block independently and requires no synchronized input/output when the algorithm is executed in parallel.

Boundary Conditions

Each face can have one of the following types of boundary conditions: solid surface, far field, or interface. In the present implementation, the type of boundary condition must be the same for the entire extent of each face of a block. For each step of the multigrid cycle in which an update to the boundary conditions is required, the code loops through the blocks and the faces within each block and updates the boundary conditions according to the appropriate type. The data structure of the solution just described allows this loop to be performed concurrently on all blocks when running the code on a parallel machine.

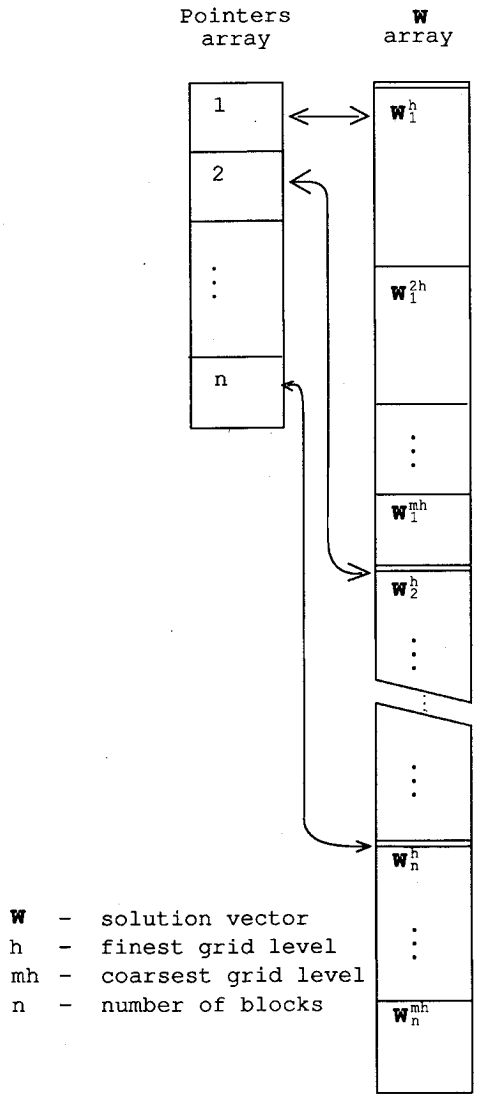


Fig. 3 Structure of data arrays.

The treatment of the boundary conditions on solid surfaces and in the far field is the same as in the original DIM scheme.[6] On an internal interface, the solution vector in the final row or column in the interior of a given block is copied into the dummy row or column of the adjacent block, as illustrated in Fig. 4.

Following the original implementation of the DIM scheme, the implicit boundary conditions are treated in a manner consistent with the characteristic theory. At each face, the appropriate eigenvalues are calculated and used to determine the directions of the characteristics for the one-dimensional problem normal to the boundary. The boundary conditions for those elements of the solution vector that correspond to characteristics entering the domain are taken to be homogeneous Dirichlet conditions, whereas the boundary conditions on those elements that correspond to characteristics leaving the domain are taken to be homogeneous Neumann conditions. Note that there is no extra computational cost for this implicit characteristic boundary condition treatment since the calculation of the eigenvalues is required for the construction of the coefficient matrix in any case.

Since the locations of the block boundaries are determined independently of the solution, it is possible that large gradients (including numerically smeared shocks) may occur in the vicin-

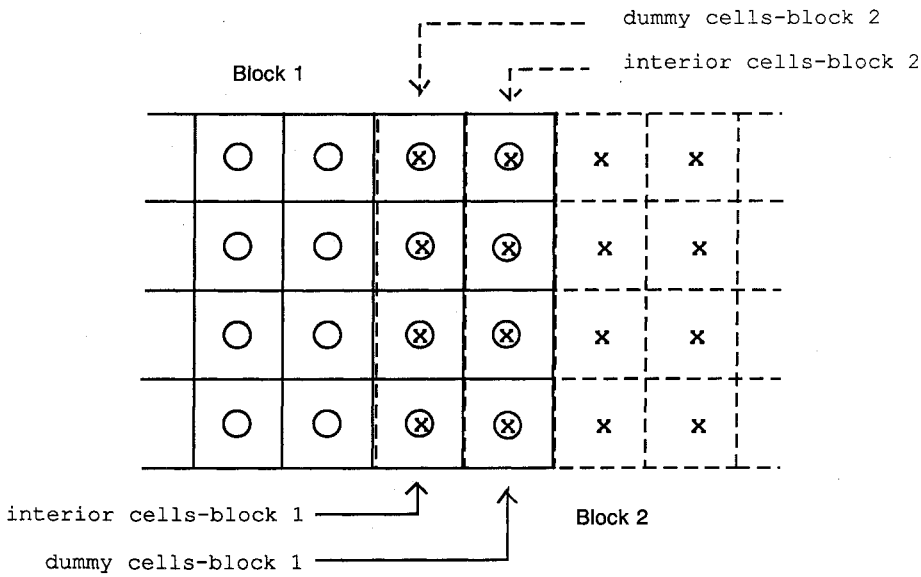


Fig. 4 Data relations at interface boundaries.

ity of the boundaries. This imposes a requirement that no approximations be made in the evaluation of the residuals near these artificial boundaries. In particular, the treatment of the boundary conditions for the dissipation terms is crucial, since these have the largest difference stencil. In the basic algorithm, the dissipative terms include fourth differences of the solution, hence their direct evaluation would require an additional layer of dummy cells. This significantly increases the storage requirements (especially if all blocks are to be stored in the virtual memory simultaneously). In the present implementation, this problem is avoided by calculating the dissipation terms on the interface boundaries when the boundary conditions themselves are updated in the boundary-condition routine. At that time, the two adjacent blocks are in memory, and all the data needed to calculate the dissipation terms are available. The additional storage requirement for the boundary conditions with this implementation is half that for the implementation using an additional layer of dummy cells. In the vertical mode, the treatment of these terms is similar to that of the interface boundary conditions: they are frozen during the entire multigrid cycle.

Parallel Implementation

The structure of the algorithm allows each step in the multigrid cycle, including the updating of the boundary conditions, to be executed concurrently on all the blocks. This suggests that an implementation of the algorithm on a parallel computer would be advantageous. Parallelism can be exploited on two levels: 1) fine-grained parallelism at the do-loop level and 2) coarse-grained parallelism at the subroutine level. In the case of the fine-grained parallelism, each task (a discrete section of computational work to be completed) is relatively small, requiring more frequent communication, more frequent synchronization, and greater overhead expenses. For the coarse-grained parallelism, the tasks are relatively larger, and more computational work is done between synchronizations, but more programming effort is required to implement the parallelism.

In the present implementation, both levels are exploited using the parallel extension of FORTRAN[8] available on multiprocessor versions of the IBM 3090. This extension allows for fine-grained parallelism by invoking a compiler option that generates parallel code for any do-loop found eligible and economic (i.e., when the solution remains the same and the code is predicted to be executed faster). The compiler allows for the nesting of parallel, scalar, and vector loops within a parallel loop and attempts to find the most efficient available combination. The coarse-grained level was implemented by executing each "do-for-all-blocks" loop in Fig. 2 concurrently. This is done using the explicit mode of parallelization available in parallel FORTRAN; a set of tasks equal to the number of blocks is defined, and each time a do-for-all-blocks loop is encountered, each task is assigned work (i.e., the execution of one iteration of the loop). If more than one processor is available, the tasks can be mapped onto different processors, and the jobs can be executed in parallel. A wait statement is provided for synchronization since all tasks or loop iterations must have finished before the next step or loop can be executed.

Results

The algorithm just described has been applied to the problem of transonic flow past an airfoil. Results have been obtained for transonic flow past the NACA 0012 airfoil for a variety of freestream conditions to verify the accuracy of the algorithm. Airfoil surface pressure distributions will be presented first, followed by a comparison of convergence rates for the composite block diagonal multigrid algorithm with those of other schemes and a discussion of the effects of block boundaries on the solution. Finally, results from the parallel implementation of the code will be discussed.

All flowfield results presented here were calculated on a sheared parabolic C-grid, containing 192×32 grid cells in the wraparound and normal directions, respectively. For the first tests, the grid was artificially divided into three blocks, containing 32×32, 128×32, and 32×32 grid cells, as shown in Fig. 5. The calculations were performed on an IBM 3090-600E, and a typical calculation (consisting of 100 work units), in which the residuals were reduced by three orders of magnitude, required about 1.5 m of CPU time. The airfoil surface pressure distribution for the NACA 0012 airfoil at a freestream Mach number of 0.8 and 1.25-deg angle of attack for the vertical mode is presented in Fig. 6. The locations and strengths of the shocks as well as the force coefficients are identical to those obtained in the horizontal mode and in single block calculations.

To examine the effects of an artificial boundary in the vicinity of large gradients, the flow at a freestream Mach number of 0.875 and 0.0-deg angle of attack was calculated. Contours of constant pressure for this case are presented in Fig. 7. It is clear that the crossing of the shocks by the boundary between two blocks has no visible effect on the shock structure.

Convergence rates discussed subsequently are for the NACA 0012 airfoil at a freestream Mach number of 0.80 and 1.25-deg angle of attack. The calculations were performed using a saw-toothed multigrid cycle with grid sequencing, starting with the

undisturbed flow as the initial guess on the coarsest grid. In this strategy, 100 multigrid work units (WU) were first performed on grids containing 8×8, 32×8, and 8×8 cells in the three blocks, respectively, using three levels of multigrid. The solution was then interpolated onto grids containing 16×16, 64×16, and 16×16 cells, and an additional 100 WU were performed. Finally, this solution was interpolated on to the final grid, containing 32×32, 128×32, and 32×32 cells, and a final 300 WU were performed. The additional work on the first and second grids required for this grid sequencing was only 3 and 10%, respectively, of that on the final grid. Figures 8 and 9 present convergence histories on the finest grid for the block scheme in horizontal mode without and with multigrid, respectively. The plotted variables are the logarithm of the average over all the grid cells of the residual of the continuity equation $|\Delta\rho/\Delta t|$, the total number of grid cells in which the local Mach number is supersonic, and the lift and drag coefficients as a function of WU. The latter three quantities are plotted on normalized scales, and one WU is the amount of computational work required for one time step on the fine grid. (One multigrid cycle requires slightly less than 4/3 WU for the saw-toothed cycle used.) The effect of multigrid on the convergence rates is clear; using five levels of multigrid, the lift coefficient reaches its final value in fewer than 100 WU, whereas without multigrid, more than 350 WU are required. With multigrid the error was reduced three orders of magnitude in 100 WU, whereas without multigrid the same reduction in error was achieved only after 300 WU. It is worth noting that these results were obtained using grid sequencing, which reduces considerably the initial error on the final grid; without

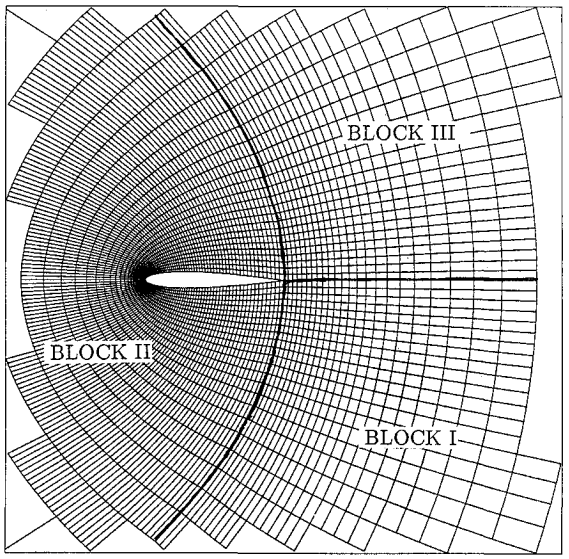


Fig. 5 Block grid boundaries in the vicinity of the airfoil.

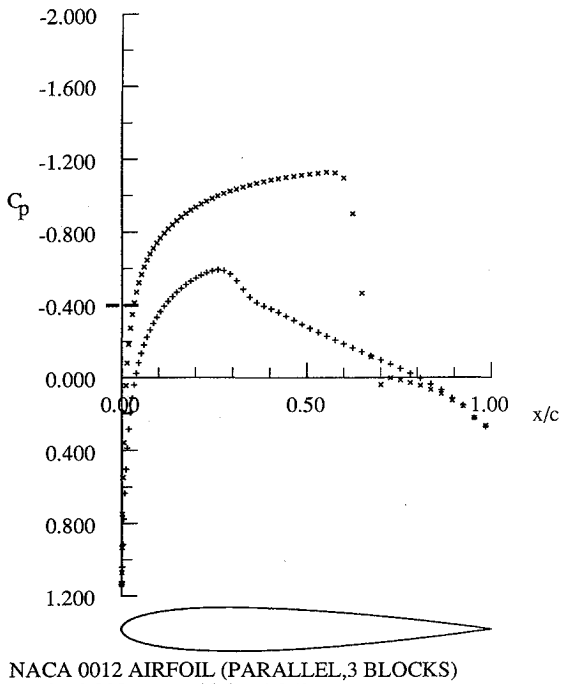


Fig. 6 Surface pressure distribution; NACA 0012 airfoil at $M_\infty = 0.8$, $\alpha = 1.25$ deg.

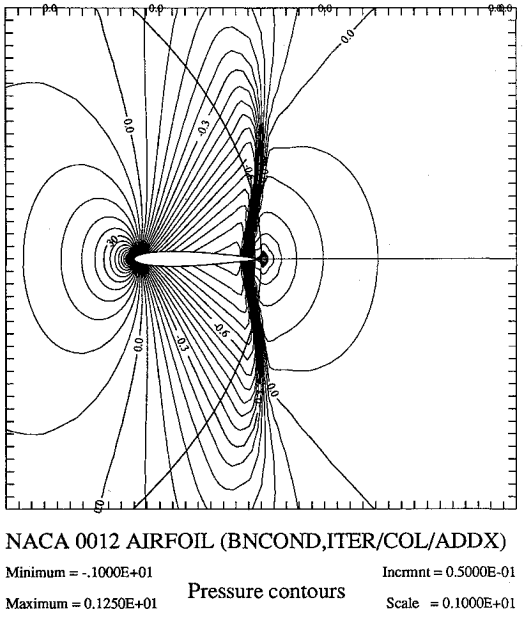


Fig. 7 Lines of constant pressure; NACA 0012 airfoil at $M_\infty = 0.875$, $\alpha = 0.0$ deg.

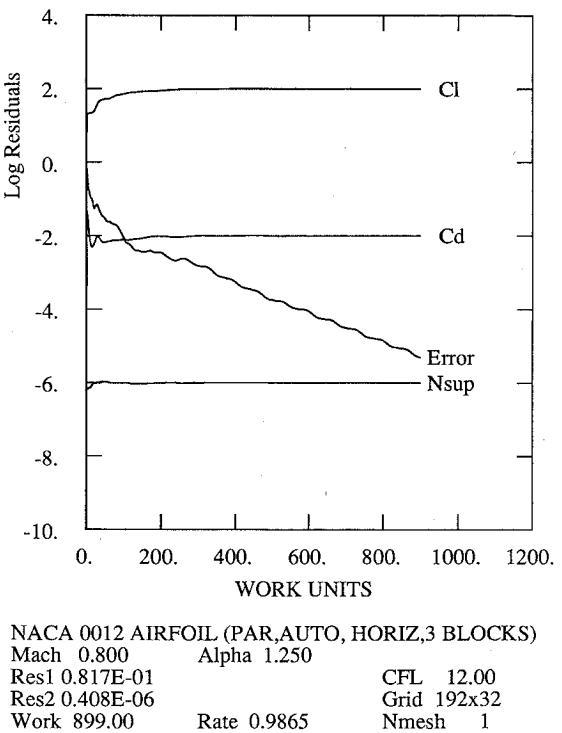


Fig. 8 Convergence history; horizontal mode, single grid; NACA 0012 airfoil at $M_\infty = 0.8$, $\alpha = 1.25$ deg.

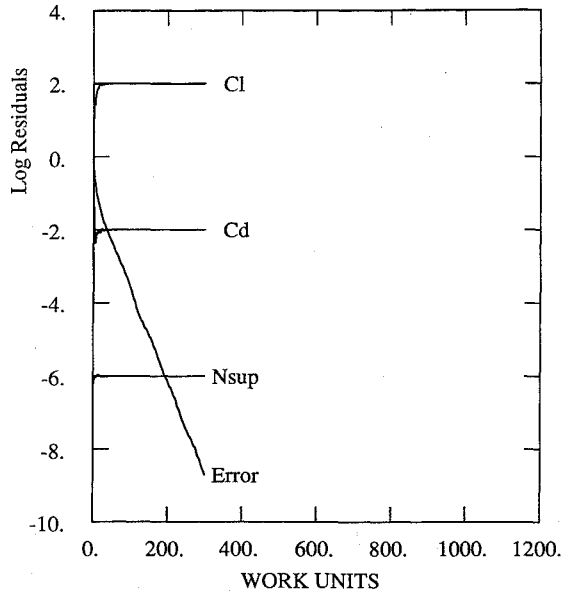


Fig. 9 Convergence history; horizontal mode, five-level multigrid; NACA 0012 airfoil at $M_\infty = 0.8$, $\alpha = 1.25$ deg.

it, the multigrid scheme reached a steady-state solution in about 150 WU whereas the single grid scheme needed about 700 WU. The asymptotic rate of residual reduction was not significantly affected by the use of grid sequencing.

Table 1 presents a comparison between the present scheme in its two modes, the unsteady block implicit Euler (UBIE) scheme developed by Belk and Whitfield[2] and the original DIM scheme of Caughey.[6] Two indicators for convergence have been employed: the iteration number at which the number of supersonic points stops changing (or is "frozen") and the number of iterations needed to reduce the average residual six orders of magnitude. It is clear that even though the number of cells in the present grid is almost 40% larger than that used by Belk and Whitfield, the present scheme is much more efficient per iteration in converging to the steady state. Comparisons in terms of the CPU time required to achieve a steady-state solution are made difficult by the fact that the UBIE code is a completely arbitrary three-dimensional solver, and the CPU time required to solve a two-dimensional problem, as in this case, includes a significant amount of overhead. However, this overhead is unlikely to amount to more than the factor of two difference in CPU time projected on the same machine for the two codes, with the result that both codes in a fair comparison would probably require about the same CPU time per iteration.

Table 1 Convergence results for different schemes

	UBIE	DIM	BDIM(VER)	BDIM(HOR)
Grid size	221×20	192×32	192×32	192×32
CPU time per grid point WU, μs	71.0[a]	32.6[a] 149.0[b]	166.0[b]	166.0[b]
Number of iterations for "frozen" supersonic points	604	31 cycles 41 WU	42 cycles 56 WU	31 cycles 41 WU
Number of iterations to reduce residual by six orders of magnitude	890	145 cycles 193 WU	312 cycles 416 WU	148 cycles 197 WU

[a]CRAY X-MP, time per iteration. [b]IBM 3090-600E.

It is also apparent that the multigrid method is proportionately faster in establishing the correct global features of the solution because of the relative efficiency with which it works on all wave-number components of the error. For example, the present scheme is almost 15 times faster at establishing the correct size for the supersonic pocket but less than five times faster asymptotically than the Belk and Whitfield scheme. The increase in CPU time per time step due to the introduction of blocks to the original scheme is about 10%, but the number of multigrid cycles required to achieve the same level of convergence for the horizontal mode is virtually unchanged.

We can see from the results in Table 1 that the convergence characteristics of the vertical mode are quite different from those of the horizontal mode. Although the number of WU required to reach a steady-state solution is greater by only 25% (but is still considerably less than that required by the UBIE scheme), the asymptotic rate of convergence is much slower. This is probably caused by the fact that in the vertical mode, the boundary conditions on the interfaces in the coarser grids are not the "correct" ones (compared to single-block or horizontal-mode multigrid), and the relaxation operations spread these disturbances into the interior of the blocks, thus impairing smoothing. Hempel and Schüller[9] reported similar behavior in the solution of Poisson equations on the SUPRENUM.

We now turn to the results of the parallel computations. Since most of the execution time is spent in four sets of operations (computing residuals and corrections, restricting residuals to coarser grids, interpolating solutions to finer grids, and updating boundary conditions), only these operations were explicitly executed in parallel. This was accomplished using the "parallel task" option in the parallel extension of FORTRAN on the IBM 3090-600E. The maximum theoretical speedup S_{theor} expected, ignoring the overhead required to manage the parallel tasks, is given by Amdahl's law

$$S_{\text{theor}} = \frac{1}{1-P+P/N}$$

where P is the percentage of work performend in parallel, and N is the number of processors. In the case where the overhead associated with the parallel tasking (which we denote by V) is included, the modified theoretical speedup $\tilde{S}_{\text{theor}}$ is given by

$$\tilde{S}_{\text{theor}} = \frac{1}{1-P+(P/N)(1+V)}$$

The actual speedup S_{act} is given by

$$S_{\text{act}} = \frac{\text{serial CPU time}}{\text{wall clock time}}$$

Table 2 presents the results of timings for calculations using three processors in parallel for the three-block configuration using both do-loop level parallelism (using the AUTO PAR-

Table 2 Speedup results using three processors in parallel for the two modes with and without autoparallelism

	Vertical mode		Horizontal mode	
	No auto	Auto	No auto	Auto
Parallel CPU time, s	331	330	344	353
Wall time, s	214	209	221	218
Serial CPU time, s	307		308	
Overhead V, %	7.2	7.0	10.4	12.7
Theoretical speedup, S_{theor}	2.94		2.83	
Mod. theoretical speedup, $\tilde{S}_{\text{theor}}$	2.75	2.75	2.58	2.53
Actual speedup, S_{act}	1.43	1.47	1.39	1.41

ALLEL option of the compiler) and parallel tasks.

It can be seen that both modes have nearly the same efficiency levels and that these are very low (S_{act} is about 55% of $\tilde{S}_{theor}$). Possible explanations for this fact include the large overhead associated with scheduling tasks and waiting for their completion and the unbalanced load due to the different sizes of the blocks, which causes some processors to sit idle while waiting for others to finish their calculations. To investigate further, the code was rerun using four processors in parallel with four blocks of equal size and with varying grid sizes. For this purpose, the distribution of cells is altered so that block II in Fig. 1 has exactly twice the number of cells as either block I or block III, then it is divide into two equal-sized blocks. Table 3 and Fig. 10 present speedup results for various block sizes and computational environments. These results were obtained without the do-loop level parallelism generated by the compiler, since it did not seem to improve the parallel efficiency of the three-block runs. The stand-alone runs were done by specifically allocating up to 90% of the CPU to the code, while the production runs were executed with no priority over other jobs running at the same time. (Stand-alone results are available only for the horizontal mode.) It is clear that simple load balancing results in an improvement in speedup. The actual speedup S_{act} for the 192×32 grid increased from 55 to 78% of the modified theoretical speedup $\tilde{S}_{theor}$ for the horizontal mode and up to 99% for the vertical mode.

By comparing S_{act} for the two modes, it is clear that the vertical mode is significantly more efficient than the horizontal mode on the coarser grids but only slightly more efficient on the finer grids. This improvement is probably due to the smaller number of tasks needed to be scheduled in the vertical mode; the cost of scheduling is comparable to the cost of computation on the coarse grids but becomes negligible as the grids get finer. The results of the stand-alone runs for the horizontal mode show that there is a good agreement between the actual speedup S_{act} and the modified speedup $\tilde{S}_{theor}$ for all grid sizes. On the other hand, the results of the production runs show good agreement for some cases and not for others. This reflects that for a real production environment, in which other users are competing for system resources, the limiting factor in achieving the modified theoretical speedup is the total CPU resource available; not all the CPU resources that the program needs for an optimal parallel performance can be allocated. For the stand-alone runs, the limiting factor is the overhead associated with the management of the parallel tasks. (This can be seen in the difference between S_{act} and $\tilde{S}_{theor}$.)

Since the structure of the code is unchanged when using parallel tasks, the overhead can be approximated by comparing the CPU time of the serial and parallel codes. This overhead accounts for the cost of originating and scheduling tasks; it does not account for the time during which individual tasks wait for others to be completed, but since the only difference in work load between the blocks is in the boundary conditions, this time should be negligible and should not contribute significantly to the overhead. The cost of originating and scheduling a task depends on the size of the task, and the total cost depends on the number of times the tasks are scheduled. Since the code was compiled for the largest dimensions and the number of schedulings is the same for all grid sizes, the absolute overhead (not the relative) should have been the same for all grid sizes at each mode and significantly higher for the horizontal mode (at least for the coarser grids). Table 3 shows, however, that the parallel overhead is very nearly a constant percentage of the total work and is only slightly less for the vertical mode than for the horizontal mode. With our current understanding of parallel FORTRAN, we cannot explain this fact, but the good agreement between S_{act} and $\tilde{S}_{theor}$ for the stand-alone runs in the horizontal mode shows that the overhead was computed correctly. It is also important to note that the results presented in Table 3 and Fig. 10 were obtained without the do-loop parallelism. By comparing the results from the runs on 192×32 grid with three blocks with do-loop parallelism and four blocks without do-loop parallelism, we observed that the speedup achieved by using load balancing is much better than that achieved by using do-loop parallelism. It is therefore better to concentrate on obtaining good vectorization at the do-loop level, and to create a balanced load, than to use the do-loop level parallelism for problems of this type.

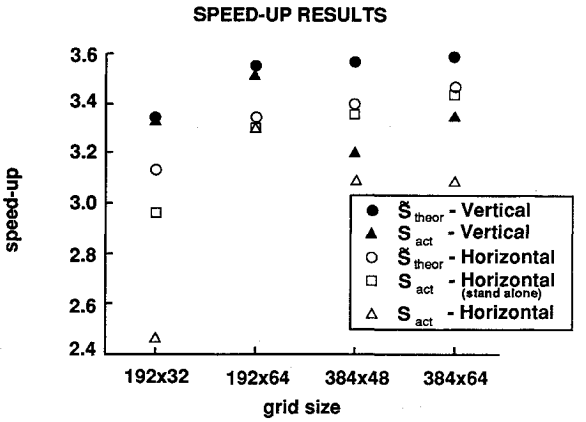


Fig. 10 Parallel speedup for various block sizes; four processors with equal-sized blocks.

Table 3 Speedup results using four processors in parallel; four block configuration, s

	Grid size / Block size	192×32 (48×8)	192×64 (48×16)	384×48 (96×12)	384×64 (96×16)
Horizontal mode	Total CPU serial	306	601	834	1100
	Total CPU parallel	339	671	924	1222
	Wall clock time	124	183	270	355
	Parallel	33	70	90	122
	Overhead, V, %	(9.7)	(10.4)	(9.7)	(10.0)
	Parallel fraction of code, P, %	93.8	96.8	97.3	98.2
	Theoretical speedup, $\tilde{S}_{theor}$	3.13	3.34	3.40	3.47
	Actual speedup, S_{act}[a]	2.96	3.30	3.36	3.44
	Actual speedup, S_{act}	2.47	3.28	3.09	3.10
Vertical mode	Total CPU serial	306	590	835	1085
	Total CPU parallel	333	633	906	1188
	Wall clock time	92	168	261	324
	Parallel	27	43	71	103
	Overhead, V, %	(8.1)	(8.1)	(8.1)	(8.1)
	Parallel fraction of code, P, %	96.0	98.0	98.5	99.0
	Theoretical speedup, $\tilde{S}_{theor}$	3.34	3.55	3.56	3.59
	Actual speedup, S_{act}	3.33	3.51	3.20	3.35

[a]Stand-alone run.

Conclusion

A multigrid diagonal implicit algorithm has been developed to solve the Euler equations of inviscid compressible flow on block structured grids. Two modes of advancing the multigrid cycle have been examined: a horizontal mode in which the multigrid cycle advances concurrently in all blocks, allowing data exchange between them during the cycles, and a vertical mode in which the multigrid cycle advances independently in each block. Results for transonic flows past airfoils verify the accuracy of both modes, in particular the fact that no spurious errors have been introduced at the interblock boundaries. It is also demonstrated that the two modes of the multigrid are effective in accelerating the convergence of the solution even when the domain is partitioned into blocks and that the efficiency of the algorithm is very high compared with other known block schemes. It also has been found that the rate of convergence of the horizontal mode is faster than that of the vertical mode due to the frequent updating of the interface boundary conditions. The algorithm has been implemented on a parallel computer, and speedups approaching the theoretical ones have been obtained for large enough problems.

Acknowledgments

This research has been supported in part by the Independent Research and Development Program of the McDonnell Douglas Corporation and by the U.S. Army Research Office through the Mathematical Sciences Institute of Cornell University. The calculations reported here were performed at the Cornell National Supercomputer Facility, a resource of the Cornell Theory Center, which receives major funding from the National Science Foundation and the IBM Corporation, with additional support from New York State and the Corporate Research Institute.

References

[1]Thompson, J. F., "A Composite Grid Generation for General Three-Dimensional Regions—the EAGLE Code," *AIAA Journal,* Vol. 26, No. 3, 1988, pp. 271, 272.

[2]Belk, D. M., and Whitfield, D. L., "Three-Dimensional Euler Solutions on Blocked Grids Using an Implicit Two-Pass Algorithm," AIAA Paper 87-0450, Jan. 1987.

[3]Karman, S. L., Jr., Steinbrenner, J. P., and Kisielewski, K. M., "Analysis of the F-16 Flow Field by a Block Grid Euler Approach," *Applications of Computational Fluid Dynamics in Aeronautics,* AGARD-CP-412, April 1986.

[4]Weatherill, N. P., Shaw, J. A., Forsey, C. R., and Rose, K. E., "A Discussion on a Mesh Generation Technique Applicable to Complex Geometries," *Applications of Computational Fluid Dynamics in Aeronautics,* AGARD-CP-412, April 1986.

[5]Frese, M. H., "Multiblock Multigrid Solution of the Implicit Time Advance Equations for Magnetic Resistive Diffusion in Geometrically Complex Regions," *Multigrid Methods: Theory, Applications and Supercomputing, Lecture Notes in Pure and Applied Mathematics*, Marcel Dekker, New York, Vol. 110, 1987, pp. 211–227.

[6]Caughey, D. A., "Diagonal Implicit Multigrid Algorithm for the Euler Equations," *AIAA Journal,* Vol. 26, No. 7, 1988, pp. 841–851.

[7]Yadlin, Y., and Caughey, D. A., "Diagonal Implicit Multigrid Solution of the Three-Dimensional Euler Equations," *Proceedings of the Eleventh International Conference on Numerical Methods in Fluid Dynamic, Lecture Notes in Physics*, Springer-Verlag, New York, Vol. 323, 1989, pp. 597–601.

[8]"Parallel FORTRAN Language and Library Reference," IBM Corporation, SC23-0431-0, 1988.

[9]Hempel, R., and Schüller, A., "Experiments with Parallel Multigrid Algorithms using the SUPERNUM Communications Subroutine Library," GMD mbH, GMD Studien Nr. 141, 1988.