AIAA JOURNAL
Vol. 30, No. 8, August 1992

Parallel Computing Strategies for Block Multigrid Implicit Solution of the Euler Equations

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

A multigrid diagonal implicit algorithm has been developed to solve the three-dimensional Euler equations of inviscid compressible flow on block-structured grids. An improved method of advancing the multigrid cycle has been examined with respect to convergence rates, accuracy, and efficiency. In this method, the multigrid cycle is advanced independently in each of the blocks, and the information exchange between the blocks is done using buffer arrays, allowing for the asynchronous updating of interface boundary conditions. This updating scheme is used to eliminate the convergence problems found in a previous implementation of the algorithm while retaining its potential for efficient parallel execution. Results are computed for transonic flows past wings 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 also presented and discussed.

Introduction

WHEN one tries to design an algorithm for the simulation of flow around realistic three-dimensional aerodynamic configurations, a major difficulty is the generation of an appropriate grid on which to compute the solution. One approach to this problem is the use of composite block-structured grids.[1] In this approach, the physical domain is divided into a set of subdomains; in each subdomain, relatively simple grids can be generated. In its most general implementation, the subdomains can be adjacent to each other or overlapped and have different degrees of continuity at the interfaces. The block-structured grids also allow different governing equations to be solved on different blocks according to the physical characteristics of the flow. The block-structured grid, in a natural way, also allows the use of a multiprocessor computer since the solution on several blocks can be computed concurrently, resulting in a faster turn-around time.

Flow solvers that take advantage of block-structured grids have been developed in the last few years, employing both explicit[3-7] and implicit schemes.[8-11] In Ref. 12, an implementation of the diagonal implicit multigrid (DIM) algorithm[13,14] on block-structured grids has been described and implemented for two-dimensional problems. The implementation of the multigrid scheme has been done in two modes: a horizontal mode, in which the multigrid cycle is kept in phase in all the blocks; and a vertical mode, in which the multigrid cycle is advanced independently in each block. Both modes were examined with regard to their accuracy and efficiency in serial and parallel execution, and it was found that, although the vertical mode shows more potential for high parallel efficiency (e.g., less synchronization and overhead), it exhibits convergence problems. These findings led to the implementation of only the horizontal mode in a three-dimensional version of the code.[15]

In this paper, an implementation of the vertical mode in a three-dimensional code that addresses these convergence problems will be described. First the numerical algorithm will be described, followed by a description of the implementation of multigrid on block-structured grids. Next, the implementation of the code on parallel computers will be discussed, and finally, results of calculations of flows past three-dimensional geometries will be presented in order to demonstrate the accuracy and efficiency of the algorithm and its applicability for parallel execution.

Received June 20, 1991; revision received Jan. 23, 1992; accepted for publication Jan. 30, 1992.

*Postdoctoral Research Associate, Cornell National Supercomputer Facility; currently Postdoctoral Associate, Department of Mechanical Engineering, Ben-Gurion University, Israel. Member AIAA.

†Professor, Sibley School of Mechanical and Aerospace Engineering. Associate Fellow AIAA.

Description of Algorithm

The block diagonal implicit multigrid (BDIM) algorithm is a direct extension of the DIM algorithm described in Refs. 13 and 14. The Euler equations are approximated using a cell-centered finite volume spatial discretization on the mesh cells corresponding to a boundary-conforming curvilinear coordinate system. Artificial dissipation is added as a blend of second and fourth differences of the solution; the fourth differences are necessary to ensure convergence to a steady state, and the second difference terms are introduced to prevent excessive oscillation of the solution in the vicinity of shock waves. The time-linearized implicit operator is approximated as the product of three one-dimensional factors, each of which is 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. In the following sections, those aspects relevant to the implementation of the DIM algorithm on block-structured grids will be described. Most of the descriptions will be for problems in two dimensions with references to the appropriate extensions to three dimensions.

Domain Decomposition

The physical domain is divided into subdomains, which are represented by rectangular 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 two layers 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 overlap. The information required at each face is as follows: 1) block number, 2) type of boundary conditions, and 3) neighboring face number (if applicable) and the orientation of its coordinate system. 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 distribu-

tion on coincident faces be the same in both blocks sharing the face.

Boundary Conditions

Each face can have one of the following types of boundary conditions: 1) solid surface, 2) far field, or 3) interface. In the present implementation, the type of boundary condition must be homogeneous over each block face. 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 either the solid surface or the far-field boundary conditions accordingly. The update of the interface boundary conditions is done by introducing a set of surface arrays, which act as buffer arrays between the blocks. Each block has a surface array that holds a copy of the solution vector in the two inner layers; from this array, data will be read into the layers of dummy cells of the adjacent blocks (see Fig. 2).

The treatment of the boundary conditions on solid surfaces and in the far field is the same as in the original DIM scheme.[13] 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 cost for this implicit characteristic boundary condition treatment since the calculation of the eigenvalues is already required for the construction of the coefficient matrix.

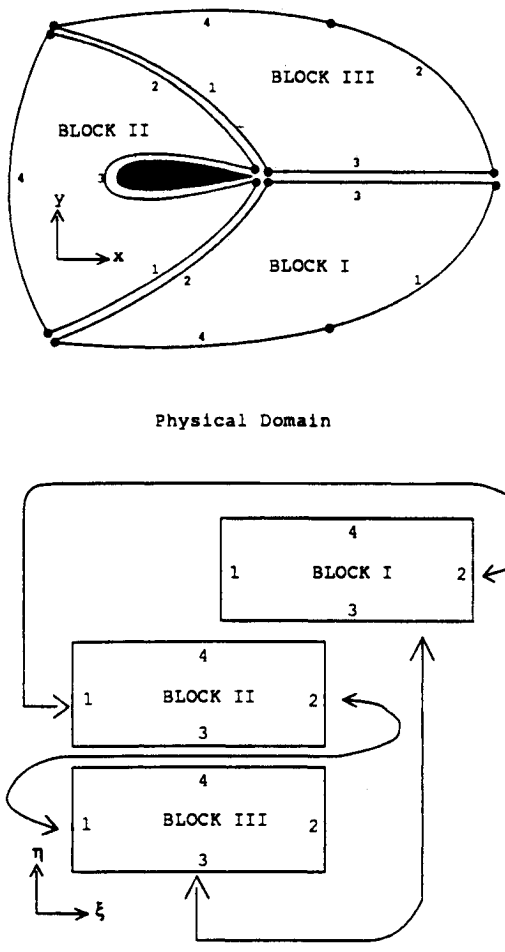


Fig. 1 Decomposition of physical and computational domains into blocks.

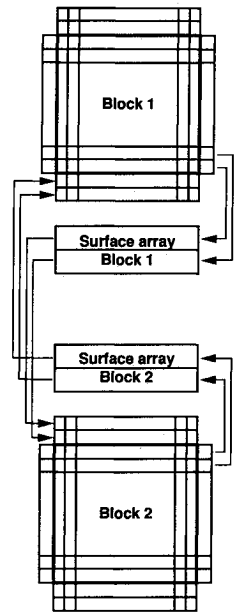


Fig. 2 Surface array.

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 vicinity 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 each block is surrounded by two layers of dummy cells (an increase of 10% in memory requirement for a block sized $64 \times 64 \times 64$).

Data Structure

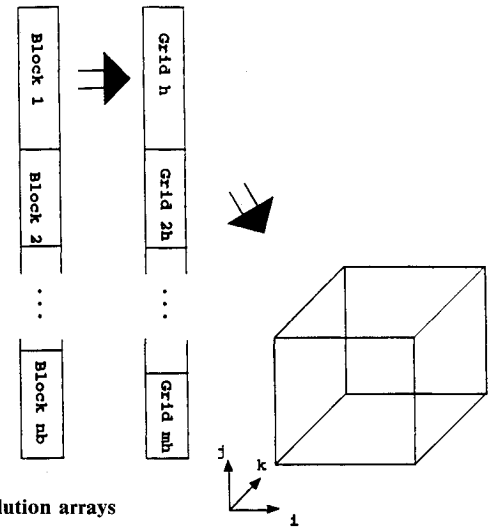
The data structure for the composite block-multigrid algorithm is an extension of the multigrid data structure used for a single block. All of the flow variables, coordinates, cell areas (volumes in three dimensions), time steps, etc., are stored in one-dimensional arrays. The arrays are organized by 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 is organized by grid levels, as is the case in the single-block multigrid scheme. The surface array data structure follows the same arrangement, where at each grid level, data is organized by faces, starting with face 1 ($\xi = 1$ surface) and ending with face 6 ($\zeta = k\max$ surface), as illustrated in Fig. 3.

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

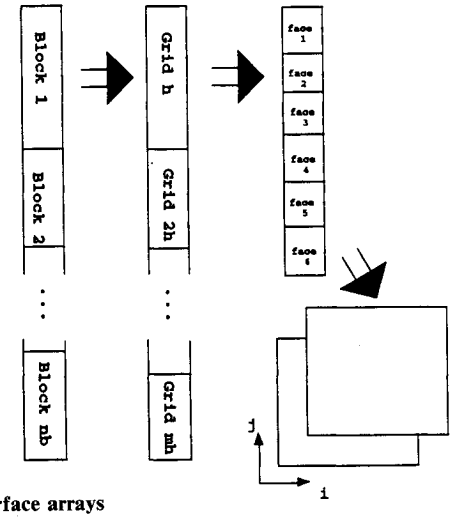
The only exceptions are the surface arrays; for these, the possibility exists that one block will try to write into a surface array while the adjacent block is reading from it. To avoid this possibility, a locking mechanism is used in the parallel code, which allows access to the array by only one block at a time.

Multigrid

The multigrid scheme on a composite block structured grid can be implemented in at least two modes: 1) horizontal — the



a) Solution arrays



b) Surface arrays

Fig. 3 Data structure.

multigrid cycle is advanced in phase in all of the blocks, or 2) vertical — the multigrid cycle is advanced independently in each of the blocks. The main difference between the two modes is the degree of 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/end of each cycle, allowing for data exchange only once in the cycle, resulting in the freezing of the boundary conditions on interfaces during the entire cycle. As discussed in Ref. 12, this updating scheme results in poor convergence rates, probably due to the fact that the interface boundary conditions on the coarse grids are not the correct ones (compared to single block or the horizontal mode multigrid), and the relaxation operations spread these disturbances into the interior of the block, thus impairing smoothing.

One way to improve on this updating scheme is the use of asynchronous updating, in which the interface boundaries are updated with any available data from adjacent blocks. That data can be new or old, depending on the current stage of the multigrid process in the adjacent block. The implementation of the asynchronous updating in the vertical mode has been done by using the surface arrays; any time an update of the interface boundaries is required, the most recently available data from the surface array of the adjacent block will be read into the layers of dummy cells of the block.

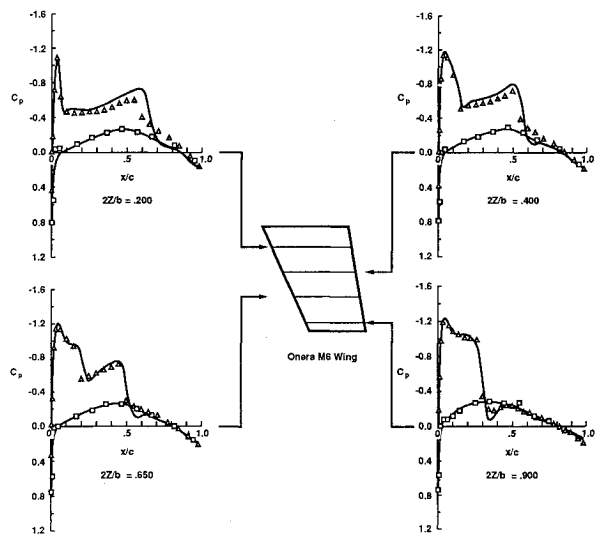
Using these surface arrays, advancing the solution one time step involves the following steps: 1) Update interface boundaries by reading in data from the surface arrays of the adjacent blocks. 2) Advance the solution one time step. 3) Write out the solution in the inner layers into the block surface array.

A similar sequence of steps is taken in the interpolation step of the multigrid cycle.

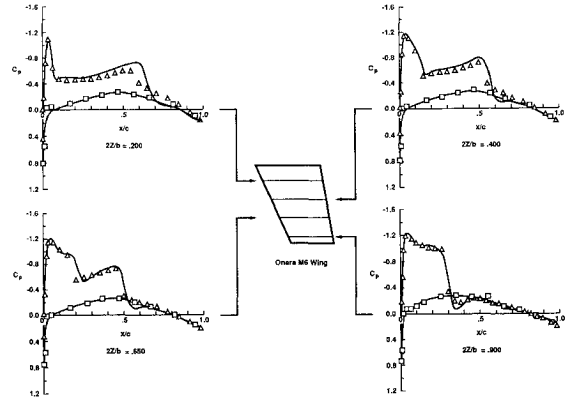
The order in which the blocks are updated dictates which of the blocks will use surface arrays with updated data and which ones will use old data. It is worth noting that the order of updating will affect the intermediate solution on the way to a converged solution (e.g., a symmetric solution may develop asymmetries during the iteration, even though it eventually converges to a symmetric solution). The present code allows for different combinations of updating interface boundary conditions at different stages in the multigrid cycle. The effects of these options will be discussed later.

Implementation on Parallel Computers

Most physical problems may have some level of inherent parallelism. Two common forms of parallelism are spatial and functional. In spatial parallelism, each subdomain interacts weakly with its neighbors and can be assigned to a different processor for updating; information is only occasionally exchanged between the domains. In functional parallelism, the physical phenomena can be represented by different model equations having different time or length scales; e.g., viscous



a)



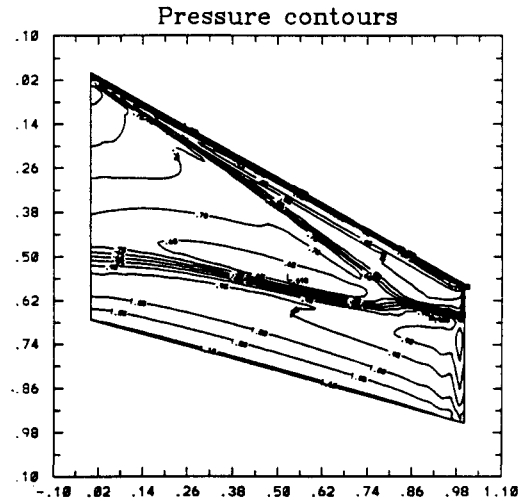
b)

Fig. 4 Comparison of measured and calculated pressure coefficients: $M_\infty = 0.839$; $\alpha = 3.06$ deg.

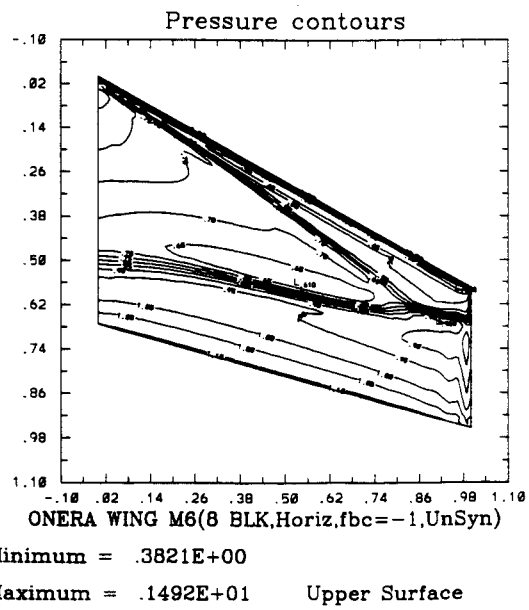
effects taking place in a confined boundary layer or a chemical reaction in a multicomponent flow.

In each of these forms, parallelism can be exploited on different levels: 1) fine-grained parallelism at the do-loop level, and 2) coarse-grained parallelism at the subroutine level. In the case of 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 coarse-grained parallelism, the tasks are relatively larger, and so more computational work is done between synchronizations, but more programming effort usually is required to implement the parallelism.

The BDIM algorithm has an inherent spatial parallelism on at least two levels. Since it is based on domain decomposition, all blocks can be updated concurrently, with exchange of boundary data at specific times. In the vertical mode, each task can be the execution of an entire multigrid cycle in each block, whereas in the horizontal mode, a task might be the execution of one part of the multigrid cycle in each block (e.g., compute corrections or interpolate corrections to a finer grid). This is a coarse-grained level of parallelism, in which the number of tasks is of the order of the number of blocks. On the fine-grained level, many operations can be done concurrently. For example: the calculation of the residual in each cell is independent of the others and can be done concurrently, the line sweeps in the alternating-direction implicit (ADI) algorithm can be done concurrently, the solution of the five decoupled pentadiagonal systems on each line can be done concurrently, etc. On this fine-grained level, the number of tasks can be of the order of the number of cells in the entire domain.



a)



b)

Fig. 5 Plan views of constant pressure contours on upper wing surfaces: a) vertical mode; b) horizontal mode ($M_\infty = 0.839$; $\alpha = 3.06$ deg).

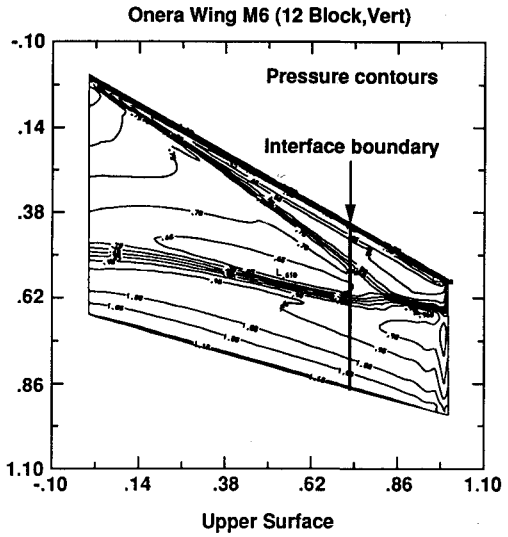


Fig. 6 Lines of constant pressure, 12 blocks: $M_\infty = 0.839$; $\alpha = 3.06$ deg.

The present implementation has used the parallel extension of Fortran (PF)[16] on the IBM 3090-600J. This parallel architecture has six processors, each with a vector facility and access to a shared memory. The language extension allows for fine-grained parallelism by invoking a compiler option that generates a parallel code for any do-loop found eligible and economical (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 is implemented by executing each step in the multigrid cycle concurrently. This is done using the explicit mode of parallelization available in parallel Fortran; a set of parallel Fortran tasks is originated [the number of tasks is the minimum (number of blocks, number of processors)], each having its own local storage and subprograms and the ability to share memory with other tasks. Each time a do-for-all-blocks loop is encountered, each task is dispatched to perform work (i.e., the execution of one iteration of the loop). If more than one real processor is available, the tasks can be mapped onto different processors and the jobs 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. For more details on PF and its execution on the IBM 3090 see Ref. 17.

Results

The algorithm just described has been applied to the problem of transonic flow past wings. Results have been obtained for the transonic flow past an ONERA M6 wing for a variety of freestream conditions to verify the accuracy of the algorithm. Surface pressure distributions will be presented first, followed by a comparison of convergence rates for the two modes of multigrid, and a discussion of the effects of block boundary updates on the solution and convergence rates. Fi-

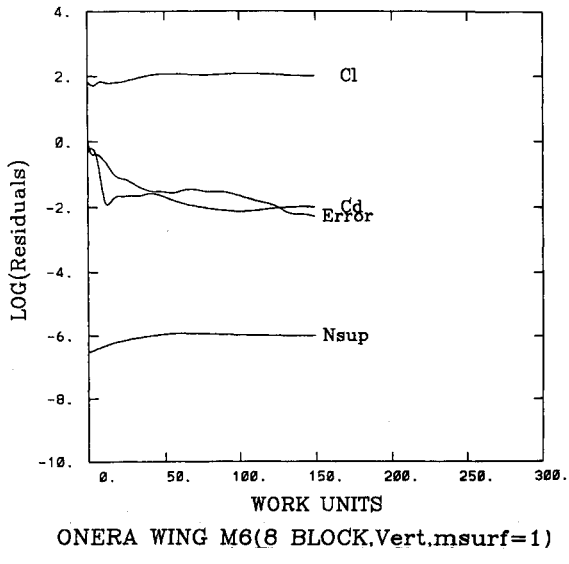


Fig. 7 Convergence history: vertical mode, single grid.

nally, results from the parallel implementation of the code will be discussed.

The grid used for these calculations is a C-grid containing $192 \times 32 \times 32$ mesh cells in the wraparound, normal, and spanwise directions, respectively. The grid was generated by stacking two-dimensional C-type grids in the spanwise direction to produce a three-dimensional grid. The two-dimensional C-type grid was generated by a weak shearing of a square root transformation about a point just inside the leading edge of the wing surface in each plane of constant z. The distribution of mesh cells is such that for each $x - y$ plane 2/3 of the cells in the wraparound direction are on the wing surface and 3/4 of the C-type sections in the spanwise direction intersect the wing. The far-field boundaries are located approximately 10 chords upstream and downstream of the wing, approximately 19 chords laterally from the wing, and at approximately 3.5 semispan from the plane of symmetry in the z direction. This global grid was artificially divided into eight blocks: four blocks containing the wing and its wake and four blocks containing the domain outboard of the wing toward the spanwise far field (blocks 1–4 around the wing and 5–8 outboard of the wingtip). This division resulted in two sets of blocks containing $32 \times 32 \times 24$, $64 \times 32 \times 24$, $32 \times 32 \times 8$, and $64 \times 32 \times 8$ grid points; the ratio of the number of cells in the largest block to that in the smallest block is 6:1. The interface boundaries in each spanwise plane lie along the wing-normal lines leaving the leading and trailing edges of the airfoil and along the cut downstream of the airfoil trailing edge.

The calculations, to be presented here, have been performed on an IBM 3090-600J under AIX, for the serial code, and under CMS for the parallel code. To confirm the accuracy of the vertical mode, results have been calculated for a freestream Mach number of 0.839 and 3.06-deg angle of attack in order to allow comparison with existing wind-tunnel test data. Figure 4 presents a comparison with wind-tunnel data[18] at four spanwise stations. The Reynolds number based on the mean aerodynamic chord Re_c for the wind-tunnel test is approximately 11.7×10^6. The calculated results predict quite accurately the strengths and the locations of the shocks for this flow condition in spite of the neglect of viscous effects in calculation.

Figure 5 presents contours of constant pressure on the upper surface of the wing for both the horizontal mode and the vertical mode. The development of the lambda shock pattern on the wing upper surface, characteristic of supercritical flows past swept wings, is clearly visible and the solutions are identical.

To examine the effects of an interblock boundary in a region of large gradients, the set of blocks containing the wing and the wake was further divided into two sets of blocks, having a common interface at about the 70% semispan of the wing. Contours of constant pressure for this case (12-block configuration) are presented in Fig. 6. It is clear that there is no visible effect on the shock structure caused by its crossing the interblock boundary (indicated by the solid line at the 70% semispan). The force coefficients for this case and the previous one (eight blocks) are exactly the same. The accuracy of the solution in the vicinity of the interface boundaries is a direct result of the introduction of the second layer of dummy cells, which eliminates the need for approximating the fourth-order dissipation terms at the boundaries. This is in contrast to the increased thickness of the shock across the interface described by Atkins,[6] which resulted from his approximation of the dissipation terms on the interblock boundary. The present results also agree exactly with the calculations done using the horizontal mode.[15]

Convergence rates discussed in the following are for the ONERA M6 wing at a freestream Mach number of 0.839 and 3.06-deg angle of attack. The calculations were performed using a sawtooth multigrid cycle with grid sequencing, starting with the undisturbed flow as the initial guess on the coarsest grid. In this strategy, four levels of grids have been used; 100 multigrid work units have been performed on each level, using the interpolated final solution of the coarser grid level as the initial solution on the next finer grid. The additional work on the first three grids required for this grid sequencing was about 14% of that on the final grid (for 100 work units on the final grid). Figures 7 and 8 present convergence histories on the finest grid for the block scheme in vertical mode without and with multigrid, respectively. The plotted variables are the logarithm of the average over all of 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 work units (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 8/7 WU for the sawtooth cycle used here). The effect of multigrid on the convergence rates is clear; using four levels of multigrid, the lift coefficient (as a measure for global convergence) reaches its final value in fewer than 40 WU, whereas without multigrid, more than 150 WU are required; with multigrid the error was reduced four orders of magnitude in 150 WU, whereas without multigrid a reduction

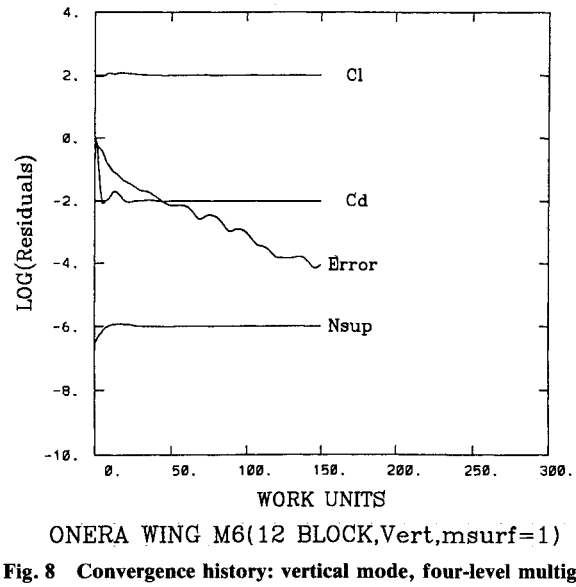


Fig. 8 Convergence history: vertical mode, four-level multigrid.

in error of only two orders of magnitude was achieved with the same number of WU. Figure 9 presents the convergence rates for the horizontal mode for the same test case. It is clear that the two modes have very similar convergence characteristics. This is in contrast to our earlier two-dimensional results presented in Ref. 12, where the boundary conditions on the block interfaces were frozen during the entire multigrid cycle, resulting in degradation of convergence rate for the vertical mode.

The effect of altering the order in which the blocks are solved on the accuracy and stability of the method has been examined by creating a random ordering of the blocks in each multigrid cycle (in contrast to a fixed consecutive ordering). The final results have been found to be exactly the same as for the fixed sequence and the differences in convergence rates were negligible.

As mentioned in the description of the multigrid cycle, the updating of the interface boundary conditions is done at two times during the cycle; before advancing the solution one time step and before interpolating the corrections to the next finer grid. The effect of the updating has been investigated by turning off the updating, when on the coarse grids, before the interpolation step and at both steps altogether. It was found that when no updating of the boundary conditions is done on the coarse grids, the solution diverges (a behavior consistent with the two-dimensional results), but when the updating is turned off only in the interpolation step, no visible effect on convergence can be found. This suggests that, in parallel execution, the overhead incurred by the locking mechanism at the updating step can be avoided by freezing the boundary conditions during the interpolation processes.

We now turn to the results of the parallel computations. Since most of the execution time is spent in the multigrid cycle, and each block can execute its cycle independently of the other blocks, this part of the code has been explicitly executed in parallel. This was accomplished using the parallel task and parallel lock options in the parallel extension of Fortran on the IBM 3090-600J. The same test case (eight block) has been executed in parallel, using up to six processors at a time. The final solution, after 75 WU, has been found to be exactly the same as for the serial execution, independent of the number of processors. The convergence rates were almost identical to the rates of the serial runs as would be expected based on the experiment with the random ordering of the blocks in serial runs.

The parallel performance of the code can be evaluated as follows. The maximum theoretical speedup S_{theor} expected,

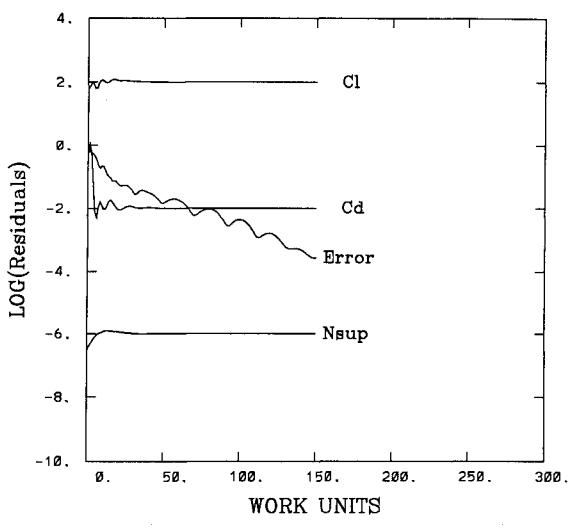


Fig. 9 Convergence history: horizontal mode, four-level multigrid.

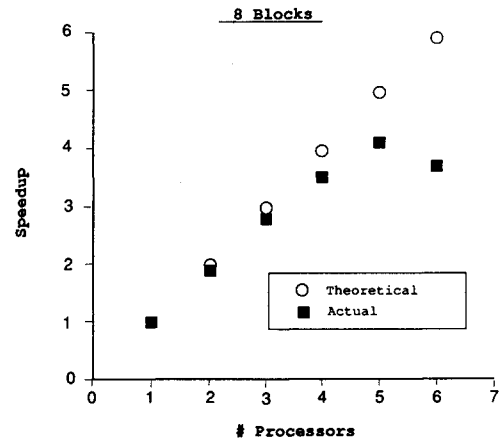


Fig. 10 Parallel speedup for vertical mode.

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 performed in parallel, and N the number of processors.

In the case where the overhead V associated with the parallel tasking is included, the modified theoretical speedup is given by

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

The actual speedup is given by

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

The efficiency of the parallel execution can be defined as

$$E = \frac{S_{\text{act}}}{\tilde{S}_{\text{theor}}}$$

Figure 10 presents speedup results for the eight-block configuration using up to six processors. It has been found that the parallel overhead V is almost negligible, except for the cases in which six processors have been used, for which $V = 2.8\%$. This is similar to the overhead observed for the horizontal mode.[15] It is clear that the parallel speedups achieved by the code are very good when using up to three processors but reduced significantly for any number of processors greater than three. This observation can be explained by looking at the way in which the different blocks are distributed between the processors and the ratio of largest to smallest blocks. For example, in the eight-block decomposition, the total work to be done between synchronization points consists of 24 units (taking the smallest block to be equal to 1 unit). When using six processors, the time to process the two largest blocks (with a size of six units each) is the limiting factor in speedup, i.e., $24/6 = 4$; it is similar for five processors. It can be seen that the actual speedup achieved for five and six processors is, in fact, less than or equal to 4. In Ref. 19, it has been shown that improved load-balancing results in better speedups for the horizontal mode; the same holds for the vertical mode. The reason for lower efficiencies when using a large number of processors is most probably due to system overhead and limited CPU resources when running in a production environment on a multiuser machine. The results presented here have been obtained for the case in which the interblock boundary conditions are not updated at the interpolation step. Updating the interblock boundary conditions at the interpolation step resulted in no significant difference in

parallel performance. This suggests that the amount of time spent in the updating process, which causes locking of some data and increases overhead, is negligible compared to the time spent in advancing the solution. This might change when a larger number of blocks is used and the ratio of block surface area to block volume increases.

Conclusions

A multigrid diagonal implicit algorithm has been developed to solve the Euler equations of inviscid compressible flow on block-structured grids. An improved version of the vertical mode of advancing the multigrid cycle has been examined. In this version, the use of buffer arrays allows for asynchronous updating of interface boundary conditions on coarse grids, thus eliminating the convergence problems encountered when the boundary conditions were frozen throughout the cycle. Results for transonic flows past wings verify the accuracy of the new method, in particular, the fact that no spurious errors have been introduced at the interblock boundaries or due to the asynchronous updating of the boundary conditions. It is also demonstrated that the new version of the vertical mode of the multigrid exhibits the same convergence characteristics as the horizontal mode, but without the need for frequent synchronization required in the horizontal mode. The algorithm has been implemented on a parallel computer, and speedups approaching the theoretical ones have been obtained when using a modest number of processors.

Acknowledgments

This research has been supported in part by the National Science Foundation through the New Technologies Research Associateships program and by the NASA Ames Research Center, under Grant NAG 2-665. 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 Code for General 3-D Regions—the EAGLE Code," *AIAA Journal*, Vol. 26, No. 3, 1988, pp. 271,272.

[2]Sorenson, R. L., "3DGRAPE Book: Theory, Users' Manual, Examples," NASA TM-102224, July 1989.

[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, pp. 18-1-18-14.

[4]Leicher, S., and Vitaletti, M., "Parallel Processing on IBM 3090: Domain Techniques for 3-D Aerodynamics Applications," *Parallel and Distributed Algorithms*, edited by M. Cosnard, Elsevier/North-Holland, New York, 1989, pp. 21–32.

[5]Swisshelm, J. M., "Development of a Navier-Stokes Algorithm for Parallel Supercomputers," NASA TM-102188, May 1989.

[6]Atkins, H. L., "A Multi-Block Multigrid Method for the Solution of the Euler and Navier-Stokes Equations for Three-Dimensional Flows," AIAA Paper 91-0101, Jan. 1991.

[7]Nishida, B. A., Langhi, R. G., and Bencze, D. P., "A Multiblock/Multigrid Euler Analysis of a Propfan Transport with Wing-Mounted Nacelles Including Slipstream Effects," AIAA Paper 91-0706, Jan. 1991.

[8]Flores, J., Reznick, S. G., Holst, T. L., and Gundy, K., "Transonic Navier-Stokes Solutions for a Fighter-Like Configuration," AIAA Paper 87-0032, Jan. 1987.

[9]Flores, J., Holst, T. L., Kaynak, Ü., Gundy, K., and Thomas, S. D., "Transonic Navier-Stokes Wing Solutions Using a Zonal Approach: Part 1. Solution Methodology and Code Validation," *Applications of Computational Fluid Dynamics in Aeronautics*, AGARD-CP-412, April 1986, pp. 30A-1–30A-12.

[10]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.

[11]Chen, C. L., Ramakrishnan, S., Szema, K. Y., Dresser, H. S., and Rajagopal, K., "Multizonal Navier-Stokes Solutions for the Multi-Body Space Shuttle Configuration," AIAA Paper 90-0434, Jan. 1990.

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

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

[14]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 Dynamics*, edited by D. L. Dwoyer, M. Y. Hussaini, and R. G. Voight, Vol. 323, Lecture Notes in Physics, Springer-Verlag, New York, 1989, pp. 597–601.

[15]Yadlin, Y., "Block Implicit Multigrid Solution of the Euler Equations," Ph.D. Dissertation, Cornell University, Ithaca, NY, Aug. 1990.

[16]IBM Corporation, "Parallel Fortran Language and Library Reference," IBM Corp., SC23-0431-0, Kingston, NY, March 1988.

[17]Gentzsch, W., Szelényi, F., and Zecca, V. "Use of Parallel Fortran for Some Engineering Problems on the IBM 3090 VF Multiprocessor," IBM European Center for Scientific and Engineering Computing, TR ICE-0023, IBM Rome, Italy, July 1988.

[18]"Experimental Data Base for Computer Program Assessment," AGARD AR-138, edited by J. Barche, May 1979, pp. BI-1–BI-44.

[19]Yadlin, Y., and Caughey, D. A., *Block Implicit Multigrid Solution of the Euler Equations on a Parallel Computer*, Parallel CFD, edited by H. Simon, MIT Press, Cambridge, MA, 1992, pp. 133–151.