

Multiblock Approach for Calculating Incompressible Fluid Flows on Unstructured Grids

Chunhua Sheng* and David L. Whitfield†
Mississippi State University, Mississippi State, Mississippi 39762
and
W. Kyle Anderson‡
NASA Langley Research Center, Hampton, Virginia 23681

A multiblock approach is presented for solving two-dimensional incompressible turbulent flows on unstructured grids. The artificial compressibility form of the governing equations is solved by a node-based, finite volume implicit scheme, which uses a backward Euler time discretization. Point Gauss-Seidel relaxations are used to solve the linear system of equations at each time step. A multiblock strategy to the solution procedure is introduced, which greatly improves the efficiency of the algorithm by significantly reducing the memory requirements while not increasing the CPU time. Results presented show that the current multiblock algorithm requires 73% less memory than the single-block algorithm.

Introduction

IN the past decade, much progress has been made in developing computational techniques for predicting flowfields about complex configurations. These techniques include both structured and unstructured grid algorithms, which have their own advantages and disadvantages. The biggest advantage of the unstructured grid approach over the structured grid approach is that the process of grid generation for complex geometries is greatly simplified. Another advantage is that unstructured grids lend themselves to adaptive grid methods because new nodes can be added to a localized region of the mesh by modifying a small subset of the overall grid data structure. Although the unstructured grid approach enjoys these advantages over structured grids, flow solvers that utilize it suffer several disadvantages. These primarily include a factor of 2–3 increase in memory requirements and computer run times on a per-grid-point basis.¹

This work introduces a multiblock approach to calculate the two-dimensional incompressible Euler and Navier-Stokes flows based on unstructured grids. The multiblock technique has been widely and successfully used in solution algorithms of structured grids to solve flows about complex configurations.^{2,3} The application of the multiblock approach in the unstructured grid area seems not to have received much attention. This is partly because computational meshes for complex geometries can be relatively easily generated using unstructured grids without the aid of the multiblock technique. However, studies of structured grid algorithms by the present authors found that the multiblock approach can also significantly reduce the memory requirements by almost one order of magnitude in real-world flow computations.^{2,3} These studies and findings motivated the current work to use the same multiblock concept to reduce the memory requirements of an unstructured grid algorithm.

The baseline code FUN2D¹ was originally developed at NASA Langley Research Center. It solves the artificial compressibility form of the two-dimensional incompressible Euler and Navier-Stokes equations on unstructured meshes. Two different turbulence models^{4,5} are included in the code for turbulent flow computations,

although the current work only uses the latter model. The discretized scheme uses a vertex-centered finite volume upwind approximation with edge-based data structure. The numerical fluxes for the inviscid part are evaluated using the flux-difference splitting scheme. The viscous terms are evaluated with a finite volume formulation that is equivalent to a Galerkin type of approximation.⁶ The time-advancement algorithm is based on the linearized backward Euler time-difference scheme, which yields a linear system of equations for the solution at each time step. The Gauss-Seidel procedure is used to solve the linear system of equations at each time step.

The idea to use the multiblock approach to reduce the memory requirements of the algorithm is quite simple. Instead of solving the equations for all of the nodes of the grid at one time, the solution process is broken down into several pieces (blocks) on the grid, which are performed one by one. Because only one block is processed at any time, the memory is allocated to store the data for that block, which requires much less memory than that to store the data for the whole grid. Special attention should be paid to the block interfaces where the information must be properly passed from the other blocks.

This paper is organized as follows: The artificial compressibility form of the two-dimensional Reynolds-averaged Navier-Stokes equations is first outlined, followed by the numerical procedures used in Ref. 1. The multiblock algorithm is introduced next, which mainly includes grid decomposition, multiblock implementation, and boundary treatment at block interfaces. Solutions of turbulent flows about a NACA 4412 airfoil and two four-element airfoils are presented to demonstrate the efficiency and accuracy of the current multiblock algorithm. Some conclusions are summarized in the last section.

Governing Equations

The unsteady two-dimensional incompressible Reynolds-averaged Navier-Stokes equations without body forces are written in Cartesian coordinates and in conservative form. A pseudo-time derivative of pressure is added to the continuity equation. The resulting set of equations in integral form represents a system of conservation laws for a control volume that relates the rate of change of a vector of average state variables q to the flux through the volume surface, which can be written as

$$V \frac{\partial q}{\partial t} + \oint_{a\Omega} f_i \cdot \hat{n} \, dl - \oint_{a\Omega} f_v \cdot \hat{n} \, dl = 0 \quad (1)$$

where \hat{n} is the outward-pointing unit normal to the control volume V . The vector of dependent state variables q and the inviscid and

Received May 9, 1997; presented as Paper 97-1866 at the AIAA 13th Computational Fluid Dynamics Conference, Snowmass Village, CO, June 29–July 2, 1997; accepted for publication July 1, 1998. Copyright © 1998 by the American Institute of Aeronautics and Astronautics, Inc. All rights reserved.

*Research Engineer, Computational Fluid Dynamics Laboratory, National Science Foundation Engineering Research Center. Member AIAA.

†Professor, Computational Fluid Dynamics Laboratory, National Science Foundation Engineering Research Center. Member AIAA.

‡Senior Research Scientist, Aerodynamic and Acoustic Methods Branch, Fluid Mechanics and Acoustics Division. Member AIAA.

viscous fluxes normal to the control volume f_i and f_v are given as

$$q = \begin{bmatrix} p \\ u \\ v \end{bmatrix}, \quad f_i \cdot \hat{n} = \begin{bmatrix} \beta \Theta \\ u \Theta + n_x p \\ v \Theta + n_y p \end{bmatrix}$$

$$f_v \cdot \hat{n} = \begin{bmatrix} 0 \\ n_x \tau_{xx} + n_y \tau_{xy} \\ n_x \tau_{xy} + n_y \tau_{yy} \end{bmatrix}$$

where β is the artificial compressibility parameter; u and v are the Cartesian velocity components in the x and y directions, respectively; Θ is the velocity normal to the surface of the control volume, where

$$\Theta = n_x u + n_y v$$

and p is the pressure. Note that the variables in the preceding equations are nondimensionalized with the characteristic length, freestream values of velocity, density, and viscosity. Pressure is normalized using the relationship $(p - p_\infty)/p_\infty V_\infty^2$, where the subscript denotes a freestream or reference value. The shear stresses in Eq. (1) are given as

$$\tau_{xx} = (\mu + \mu_t)(2/Re) u_x, \quad \tau_{yy} = (\mu + \mu_t)(2/Re) v_y$$

$$\tau_{xy} = (\mu + \mu_t)(1/Re) (u_y + v_x)$$

where μ and μ_t are the laminar and turbulent viscosities, respectively, and Re is the Reynolds number.

Solution Algorithm

Finite Volume Scheme

The baseline flow solver is a node-based finite volume implicit scheme. The computational domain is divided into a finite number of triangles from which control volumes are formed that surround each vertex in the mesh. The flow variables are stored at the vertices of the triangle. Equation (1) is then numerically integrated over the closed boundaries of the control volumes surrounding each node. These control volumes are formed by connecting the center of each triangle to the midpoint of the edges, as shown in Fig. 1. These nonoverlapping control volumes combine to completely cover the domain and are considered to form a mesh that is dual to the mesh composed of triangles formed from the vertices.

Numerical Flux Evaluation

The numerical evaluation of the surface integrals in Eq. (1) is conducted separately for the inviscid and viscous contributions. The inviscid fluxes are obtained on the faces of each control volume with a flux-difference-splitting scheme, whereas the viscous terms are evaluated with a finite volume formulation that is equivalent to a Galerkin type of approximation.¹ The inviscid fluxes on the boundaries of the control volume around the central node 0 are given by

$$\Phi = \frac{1}{2} [f_i(q^+; \hat{n}) + f_i(q^-; \hat{n})] - \frac{1}{2} |\bar{A}| (q^+ - q^-) \quad (2)$$

where Φ is the numerical flux and f_i is the flux vector given in Eq. (1). A nonsingular eigensystem for the matrix $|\bar{A}|$ was obtained in Ref. 1 and will not be repeated here. Quantities q^- and q^+ are the values of the dependent variables on the left and right sides of the

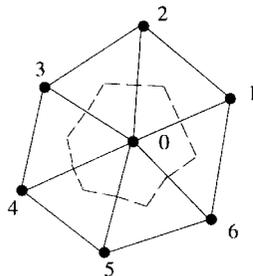


Fig. 1 Control volume surrounding a node: —, mesh, and - - -, median dual (boundary of control volume).

boundary of the control volume. For first-order-accurate differencing, q^- and q^+ are set equal to the data at the nodes lying on either side of the boundary face. For higher-order differencing, these values are computed with a Taylor series expansion about the central node 0 of the control volume:

$$q_{\text{face}} = q_{\text{node}} + \nabla q \cdot \mathbf{r} \quad (3)$$

where \mathbf{r} is the vector that extends from the central node to the midpoint of each edge and ∇q is the gradient of the dependent variables at the node.

The gradient ∇q can be evaluated with a least-squares procedure in which the data surrounding each node are assumed to behave linearly. The data at each node surrounding the center node may be expressed as

$$q_i = q_0 + q_{x_0}(x_i - x_0) + q_{y_0}(y_i - y_0) \quad (4)$$

By expressing the data in a like manner at each of the N surrounding nodes, an $N \times 2$ system of equations is formed as

$$\begin{bmatrix} \Delta x_1 & \Delta y_1 \\ \Delta x_2 & \Delta y_2 \\ \vdots & \vdots \\ \Delta x_N & \Delta y_N \end{bmatrix} \begin{Bmatrix} q_{x_0} \\ q_{y_0} \end{Bmatrix} = \begin{bmatrix} q_1 - q_0 \\ q_2 - q_0 \\ \vdots \\ q_N - q_0 \end{bmatrix} \quad (5)$$

This equation represents an overdetermined system, which can be solved by using the normal equation approach⁷ to obtain the gradients at the nodes. Another approach to calculate the gradients at the nodes, as suggested in Ref. 6, is to use the Gram-Schmidt process because of the sensitivity of the normal equation approach to the condition number squared of the solution matrix.⁶ The resulting formulas for calculating the gradients at the center node in Fig. 1 are given as

$$q_{x_0} = \sum_{i=1}^N W_i^x (q_i - q_0), \quad q_{y_0} = \sum_{i=1}^N W_i^y (q_i - q_0) \quad (6)$$

where the summation is over all of the edges that connect to the node and the weights are given by

$$W_i^x = \frac{[r_{yy}(x_i - x_0) - r_{xy}(y_i - y_0)]}{r_{xx}r_{yy} - r_{xy}^2}$$

$$W_i^y = \frac{[r_{xx}(y_i - y_0) - r_{xy}(x_i - x_0)]}{r_{xx}r_{yy} - r_{xy}^2}$$

with

$$r_{xx} = \sum_{i=1}^N (x_i - x_0)^2, \quad r_{yy} = \sum_{i=1}^N (y_i - y_0)^2$$

$$r_{xy} = \sum_{i=1}^N (x_i - x_0)(y_i - y_0)$$

Note that Eq. (5) gives unweighted gradients in which all of the data surrounding the central node are given equal consideration. It was found in Ref. 6 that weighted gradients, which are evaluated by using inverse distance or Green's theorem, provide better accuracy than unweighted gradients, when actual gradients are required, as in the production terms for the turbulence model. However, for reconstructing nonlinear data on highly stretched meshes, such as viscous grids, an unweighted formulation is far superior to either inverse distance weighting or the use of gradients calculated with Green's theorem.⁶

The viscous flux contribution to the residual is obtained using a finite volume approach. In this approach, quantities such as velocity derivatives are first evaluated in each triangle of the mesh, and the viscosity is computed as an average of the three nodes making up the triangle.

Time-Advancement Scheme

The time-advancement algorithm is based on the linearized, backward, Euler time-differencing scheme, which yields a linear system of equations for the solution at each time step:

$$[A]^n \{\Delta q\}^n = -\{r\}^n \quad (7)$$

where $\{r\}^n$ is the vector of steady-state residuals, $\{\Delta q\}^n$ is the change in the dependent variables, and the solution matrix $[A]^n$ is written as

$$[A]^n = \frac{V}{\Delta t} I + \frac{\partial r}{\partial q} \quad (8)$$

The solution of this system of equations is obtained by a relaxation scheme in which $\{\Delta q\}^n$ is obtained through a sequence of iterations $\{\Delta q\}^i$, which converge to $\{\Delta q\}^n$. There are several variations of classic relaxation procedures that have been used in the past for solving this linear system of equations.^{8,9} In this work, a point implicit Gauss-Seidel procedure as described in Ref. 1 is used. To clarify the scheme, $[A]^n$ is first written as a linear combination of two matrices representing the diagonal and off-diagonal terms:

$$[A]^n = [D]^n + [O]^n \quad (9)$$

and the solution to the linear system of equations is obtained by adopting a Gauss-Seidel type of strategy in which all odd-numbered nodes are updated first, followed by the solution of the even-numbered nodes. This procedure can be represented as

$$[D]^n \{\Delta q\}^{i+1} = [-\{r\}^n - [O]\{\Delta q\}^{(i+1)/i}] \quad (10)$$

where $\{\Delta q\}^{(i+1)/i}$ is the most recent value of Δq , which will be at subiteration level $i+1$ for the odd-numbered nodes that have been previously updated and at level i for the even-numbered nodes. Normally, 15–20 subiterations are adequate at each time step.

Turbulence Modeling

For the current study, the one-equation turbulence model of Spalart and Allmaras⁵ is used. Attention is paid to the distance function when coupling this turbulence model with the multiblock algorithm described next. That is, the distance function for each node of the grid is computed based on the closest distance to the wall across all blocks, prior to processing each block. In the solution process, the equation for turbulent viscosity is solved using a backward Euler time-stepping scheme similar to that used for the flow variables but separated from the flow equations. This results in a loosely coupled solution process that allows easy interchange with new turbulence models. For more details about the turbulent model and its implementation, see Refs. 1 and 5.

Multiblock Algorithm

Grid Decomposition

In the process to generate a multiblock structured grid, the physical domain is first divided into several subdomains with prescribed block boundaries. The grid in each block is then generated separately. To generate a multiblock unstructured grid, however, the already described process does not apply because prescribed block boundaries will degrade the grid quality in these regions. The multiblock unstructured grid is generated through a grid-decomposition process. First, the computational grid of the whole physical domain is constructed within a single block using the grid-generation program TRIGIT.¹ Then a domain-decomposition method is used to break the mesh into several subdomains (or blocks) by selecting all of the cells that fall into the region set for each subdomain. Each block contains a complete set of node information, just like a separate grid. A file that contains information to connect block-to-block interfaces is also created. This domain-decomposition method has also been used for other purposes in unstructured grid algorithms, for example, see Ref. 10 for parallel computations.

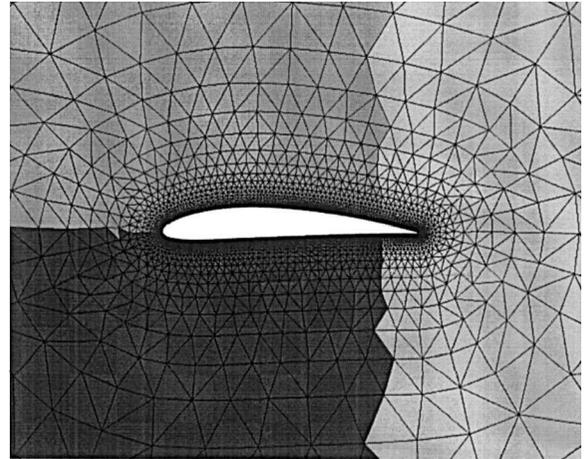
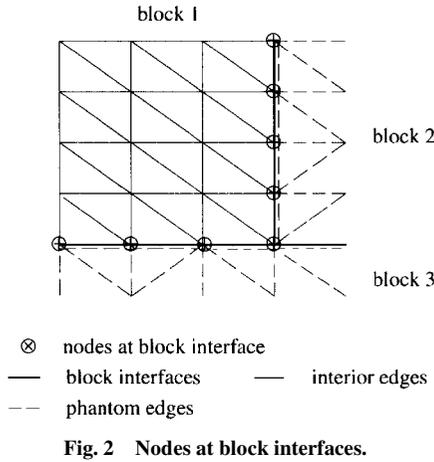
There are several criteria that are used to guide the process of the grid decomposition. The most important one is the number of nodes or cells in each block. Because the overall memory requirements

of the solver are determined by the largest block, each block grid should contain a nearly equal number of nodes or cells to achieve the best reduction in memory usage. Other criteria, such as selecting the fewest interfaces for a given number of blocks to minimize the data communication between blocks, should also be considered. A readily available software (METIS)¹¹ can partially fulfill these requirements and was used in the current work to partition the single-block mesh into multiblock grids.

Multiblock Strategy and Memory Allocation

The implementation of the multiblock algorithm on unstructured grids adopts a similar strategy as used in structured grids in Ref. 2, i.e., a vertical mode in which a complete cycle is completed in each block before proceeding to the next block. The advantage of this approach is that the solution process (nonlinear and linear procedure) in each block is local and, thus, does not depend on the solution in other blocks. The data at block interfaces are treated as block boundary conditions, which are updated after each time step, as described subsequently. This nature of independence of the solution to other blocks offers great flexibility in both implementation and memory allocation for the algorithm and also provides a natural platform for parallel implementations. (In fact, the main difference between the multiblock algorithm and the parallel implementation is that, in the former, the solution in each block is performed sequentially in a prescribed order, whereas in the latter, all blocks are solved simultaneously.) The disadvantage of the vertical mode is that, if a grid has a lot of blocks, the convergence rate could suffer because of the explicit nature of the data interchange between blocks. Because the main purpose of this work is to reduce the memory requirements of the unstructured grid algorithm, the memory allocation in the code must be done in a special way to achieve the best efficiency in both memory usage and CPU time. In the current work, all memory is allocated either locally or globally. For local memory, only the storage needed for the current block is allocated when the solution process enters that block, and this storage is freed at the time when the solution process leaves the block. The new storage is reallocated when the solution process moves to the next block. The local memory allocation is mainly for variables that do not need to be stored for all blocks, such as the flux Jacobian matrix (most costly part in the memory usage), which is updated after each time step in each block. On the other hand, the global memory allocation means that storage is allocated for all blocks and is not freed until the solution process is complete. Some data, such as flow variables, grid coordinates, and the distance function in the turbulence model, must be stored in a global way for all blocks. Therefore, by adopting the described strategy for memory allocation, the memory requirements of the multiblock algorithm will be much less than that of the single-block algorithm where all memory is allocated globally.

Note that the way to compute and save the metric quantities of the grid has great impact on the overall memory usage in the multiblock algorithm because the storage for these metric terms is the second largest part in the memory cost. If the metric terms of the grid are computed and saved in each block, the memory reduction by this method is limited to the maximum of 45% of the single-grid algorithm. The reason is that the memory needed to store these metric terms will exceed the memory reduced by the just-described multiblock algorithm when a certain number of blocks is exceeded. An improvement can be made by allocating these metric terms locally for each block and recomputing them after each time step. Although this will introduce some CPU time overhead, it is the most universal way for general unsteady problems with dynamic moving grids because for these problems the metric terms of the grid must be recalculated after each time step following the grid movement anyway. In fact, because the memory usage of the algorithm is further reduced by this method, the overall CPU time to execute the code may be even less because more data in the main memory can now reside in a much faster accessible data cache device, which allows more efficient implementation of the code. Results in this work show that, by adopting the described strategy, the memory savings by the multiblock algorithm is significantly increased to up to 73% of the single-block algorithm, whereas the CPU time cost is even 12% less than that of the multiblock code in which the metric terms of the grid are allocated globally for all blocks.



Interface Treatment

Two major issues in the multiblock implementation are the data structure and the treatment of the block interfaces where two or more blocks are next to each other. A natural choice is to use the same edge-based data structure to store the data at block interfaces and to treat the nodes at the interface in the same way as the interior nodes (Fig. 2). In the algorithm, a set of buffer arrays is declared for each block to store the data on phantom edges that surround the current block and have the connection to the block interfaces (dashed lines in Fig. 2). In the actual computation, the nodes at block interfaces (circles in Fig. 2) are computed in two steps. In the first step, contributions from interior edges (solid lines in Fig. 2) are collected, and values are stored at the nodes. This procedure is the same as for all interior nodes. In the second step, contributions from other blocks are added to these nodes at block interfaces by looping over the phantom edges surrounding the current block using the data stored in the buffer arrays. The data in the buffer arrays are updated after each time step when all blocks have been processed. Note that the method provides synchronized boundary conditions at the block interfaces because the gradient of flow variables is computed based on the values at the same previous time step. This ensures that the performance of the multiblock algorithm will be close to that of the single-block algorithm.

Results

Results are given for three airfoil cases, a NACA 4412 airfoil and two four-element airfoils. The unstructured grids are first generated in a single block with a code described in Ref. 1, and a domain decomposition method is used to divide the grid into several blocks. Solutions are obtained with the single-block and the multiblock algorithms. Comparison is made for the memory usage and CPU time between the two methods. Note that, for both multiblock and single-block algorithms, the metric quantities of the grid are recomputed after each time step. For each case, the Courant-Friedrichs-Lewy number has been linearly ramped from 20 to 200 over 100 iterations, and 15 subiterations were used at each time step to obtain an approximate solution of the linear system. All computations were carried out on an SGI R10000 194-MHz single processor with 2-GB in-core memory and 2-MB data cache size.

NACA 4412 Airfoil

The first case selected to test the current algorithm is the viscous flow over a NACA 4412 airfoil. Computations are performed with both the single-block and multiblock algorithms, and results are compared with the experimental data obtained in Ref. 12. The flow conditions include an angle of attack of 13.87 deg and a Reynolds number of 1.52×10^6 (based on the chord length of the airfoil). The unstructured grid contains 10,347 nodes and 20,694 cells. The normalized grid spacing at the wall is about 1.0×10^{-5} . The multiblock grid is obtained by dividing the single-block grid into three blocks, with the number of nodes of 3030, 3625, and 3864 in each block, respectively (Fig. 3).

Figure 4 shows the convergence histories for both single-block and multiblock algorithms. Both solutions have about the same

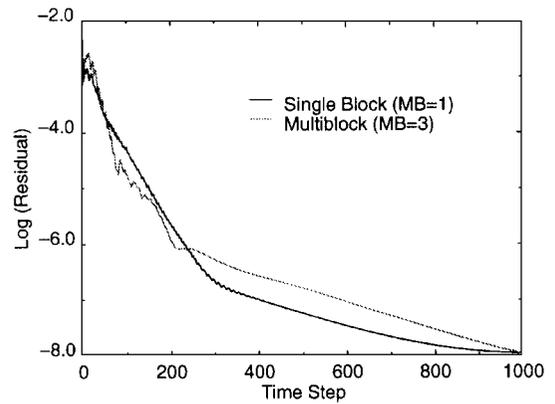


Fig. 4 Convergence histories of single-block and multiblock solutions.

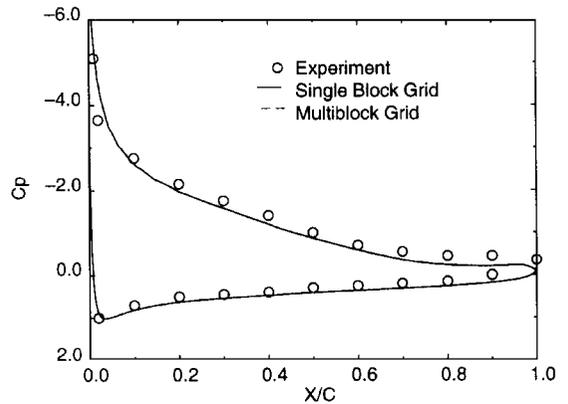
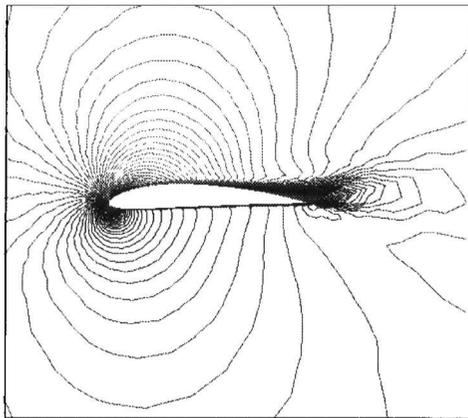


Fig. 5 Comparison of Cp distributions between the experimental data and computations.

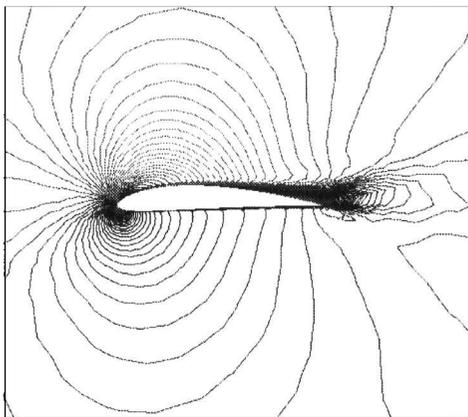
convergence rate, which indicates that the efficiency of the multiblock algorithm is not degraded with the current multiblock strategy and the treatment of block interfaces. Figure 5 shows the comparison of the C_p distributions on the airfoil surface between the experiment and computations. The result shows that both multiblock and single-block solutions are very close to each other, which are in good agreement with the experimental data. Figure 6 shows u velocity contours of both the single-block and multiblock solutions. The continuity of the contour lines across the interfaces indicates the proper treatment of passing data among blocks. Table 1 shows the memory usage and CPU times for 1000 time iterations for both single-block and multiblock solutions. It is seen that the memory requirements for the three-block multiblock code are 41% less than that for the single-block grid.

Table 1 Memory usage and CPU time of single-block and multiblock solutions for NACA 4412 airfoil

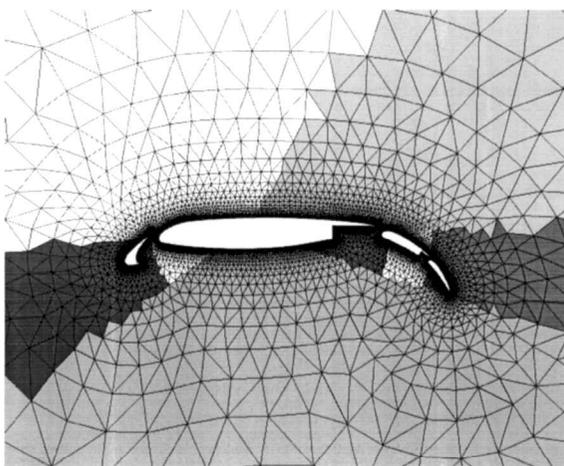
Grid, nodes	Block	Memory, MB	CPU time, min
10,347	1	22	42
10,347	3	12	38



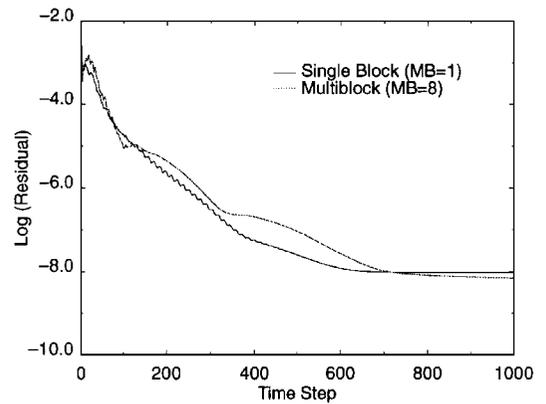
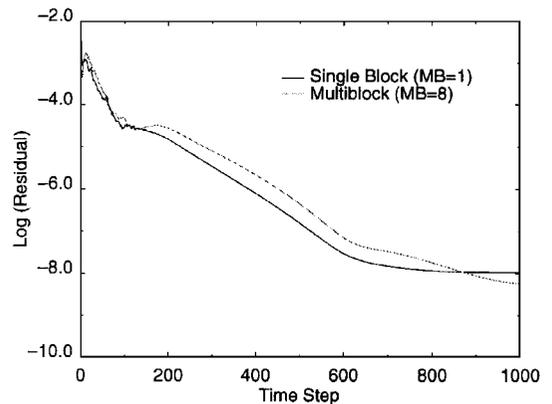
a) Single block



b) Multiblock

Fig. 6 Computed u velocity contours around the airfoil.**Fig. 7** Eight-block grid for the four-element airfoil with 34,295 nodes.**Four-Element Airfoil**

To further demonstrate the efficiency of the current multiblock algorithm, computations were performed on two four-element airfoils. The unstructured mesh for the first configuration was built with 34,295 nodes and 68,596 cells, decomposed into eight blocks (Fig. 7). The block boundaries are placed in several critical wake regions to test the robustness and efficiency of the current code, as well as the accuracy of the block boundary condition as described

**Fig. 8** Convergence histories for the four-element airfoil at $Re = 9 \times 10^6$ and 0-deg angle of attack.**Fig. 9** Convergence histories for the four-element airfoil at $Re = 9 \times 10^6$ and 16-deg angle of attack.

before. The normalized grid spacing on the wall is 1.0×10^{-5} , and the Reynolds number is 9×10^6 based on the length of the airfoil with the slat and flaps retracted. Computations are performed for the flow at 0- and 16-deg angles of attack, and results are compared with the experimental data in Ref. 13.

Figures 8 and 9 show the convergence histories of both multiblock and single-block solutions at 0- and 16-deg angles of attack. The convergence rates of the multiblock and single-block solutions are very similar for both flow conditions. The computed and measured C_p distributions on the slat, main, main flap, and auxiliary flap surfaces are shown in Figs. 10 and 11. Computed results are matched very well with the experimental data at both 0- and 16-deg angles of attack. Note that the current multiblock solver reproduces the same solution as the single-block solver does.

A numerical test was also conducted on another four-element airfoil for which the unstructured mesh was built with 132,331 nodes and 264,668 cells, partitioned into 20 blocks (Fig. 12) using METIS.¹¹ The normalized grid spacing on the wall is 1.0×10^{-6} , which yields a y^+ value of less than 1 for the nodes off the wall surface. The inflow conditions include 20.318-deg angle of attack and a Reynolds number of 9×10^6 . Figure 13 shows the contours of the u velocity components obtained by both single-block and multiblock solutions. It is seen that overall flow patterns between the two solutions are very similar. Figure 14 shows the convergence histories of both multiblock and single-block solutions. It shows that the performance of the multiblock solution is not degraded when the number of blocks is increased.

Table 2 summarizes the CPU times for 1000 time steps and memory requirements for the last two grids. It is seen that the memory savings by the multiblock solution over the single-block solution is 63% for the 8-block grid case and 73% for the 20-block grid case. The CPU time for the multiblock solutions is about 10–40% less than that required in the single-block solutions. The preceding single-block solutions require about 8–22% more

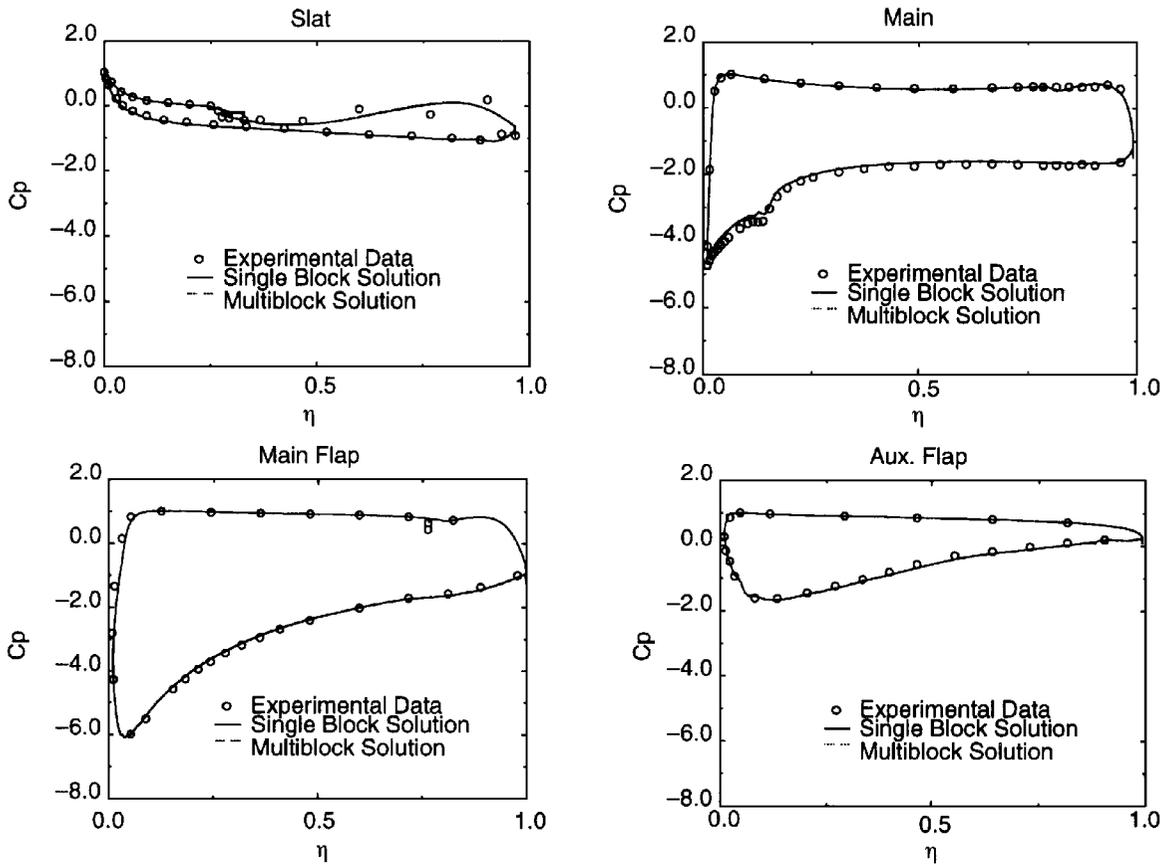


Fig. 10 Comparison of C_p distributions for the four-element airfoil at $Re = 9 \times 10^6$ and 0-deg angle of attack.

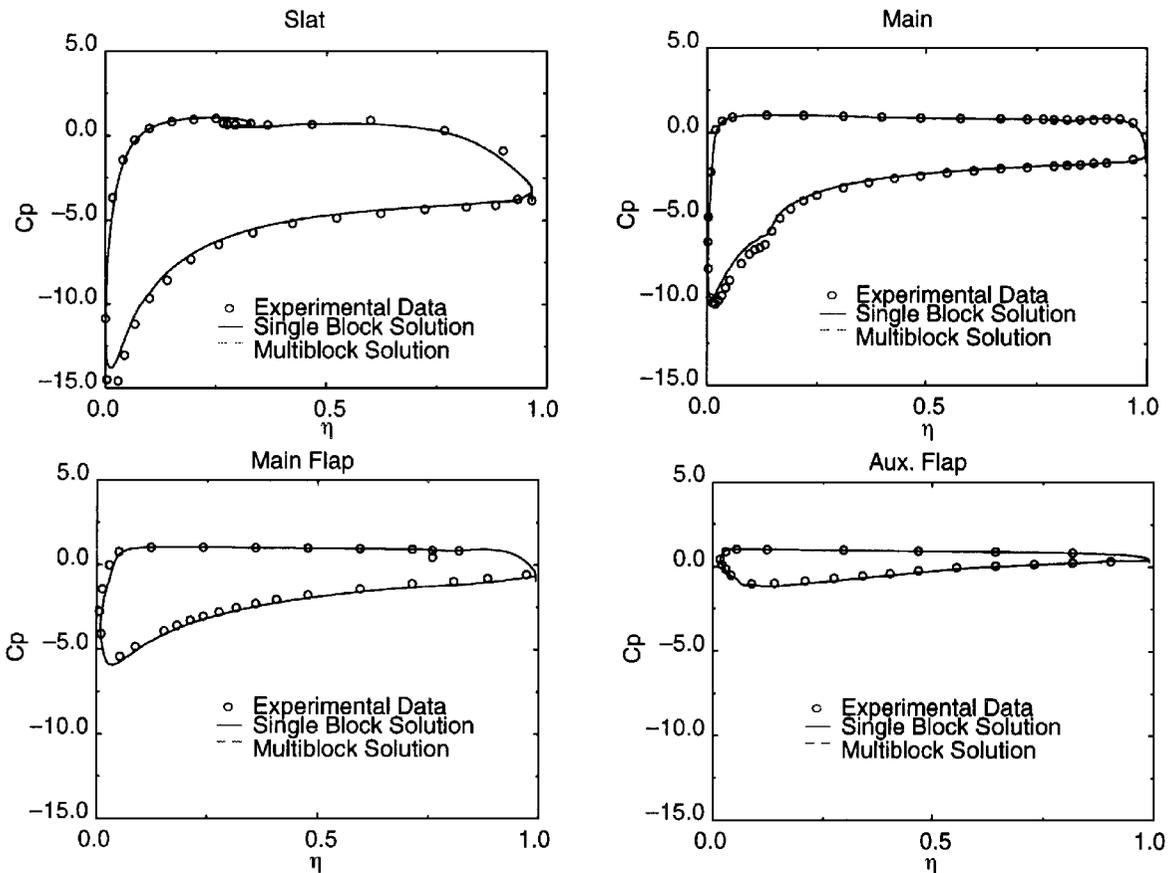


Fig. 11 Comparison of C_p distributions for the four-element airfoil at $Re = 9 \times 10^6$ and 16-deg angle of attack.

Table 2 Memory usage and CPU times of single-block and multiblock solutions for four-element airfoils

Grid, nodes	Blocks	Memory, MB	CPU time, min
34,295	1	66	229 (188) ^a
34,295	8	25	138
132,331	1	241	864 (799) ^a
132,331	20	65	776

^aCPU time of solutions where the metric terms of the grid are not recomputed.

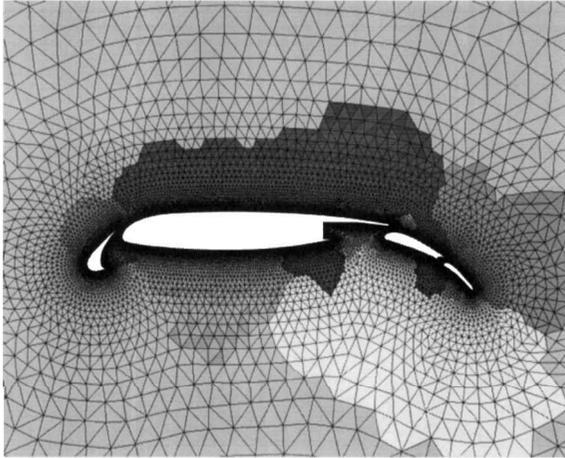
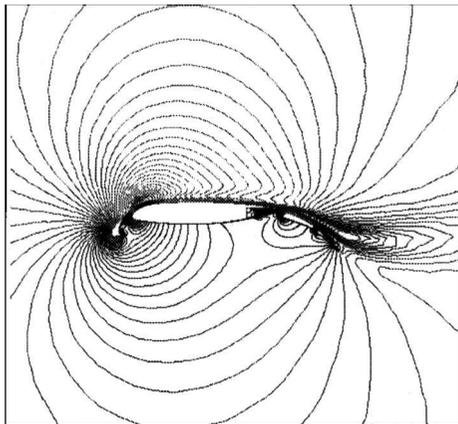
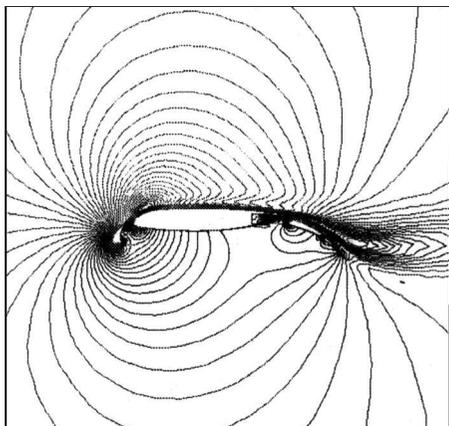


Fig. 12 Four-element airfoil with 132,331 nodes partitioned into 20-block grid.



a) Single block



b) Multiblock

Fig. 13 Computed u velocity contours around the four-element airfoil on grid with 132,331 nodes.

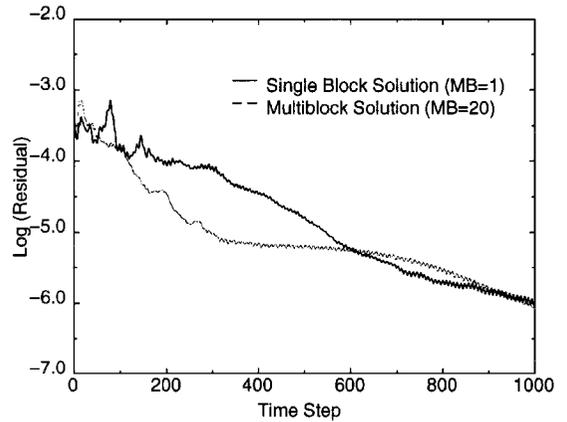


Fig. 14 Convergence histories for four-element airfoil with 132,331 nodes.

CPU time than the original code because of the overhead of the CPU time introduced by recomputing metric quantities of the grid. Nonetheless, the multiblock algorithm still takes less CPU time than the original single-block algorithm.

Conclusions

The development of a multiblock algorithm to solve the two-dimensional incompressible Reynolds-averaged Navier-Stokes equations is presented. The multiblock technique has been used in many computational fluid dynamics applications for structured grids. However, little attention has been given to applying this approach to the unstructured grid area. Results presented show that, by properly allocating the memory for the code, the multiblock solution may save up to 73% of the memory of the single-block solution. Furthermore, the current multiblock algorithm does not introduce any overhead of the CPU time (it even reduces the CPU run time compared with the single-block solutions) on the computer used for the current work. It is expected that the CPU time may go up some on computers where cache management is less important. Nevertheless, the features demonstrated from the current multiblock algorithm prove that this technique is a useful alternative to the disadvantage associated with the large memory requirements of unstructured grid algorithms. Although the results presented here are for two-dimensional cases, the concepts and methods of the algorithm are readily applicable to general three-dimensional compressible and incompressible flow solvers. It is believed that the combination of the advantages from the multiblock technology and the unstructured grid technology makes it possible to provide a highly efficient and cost-effective computational tool to predict realistic complex flows about complex configurations.

Acknowledgments

This research was supported by the NASA Langley Research Center under NASA Grant NAG-1-1990. This support is gratefully acknowledged. The authors wish to thank Gregory Henley of the National Science Foundation Engineering Research Center for Computational Field Simulation for helpful discussions.

References

- Anderson, W. K., Rausch, R. D., and Bonhaus, D. L., "Implicit/Multigrid Algorithms for Incompressible Turbulent Flows on Unstructured Grids," AIAA Paper 95-1740, June 1995.
- Sheng, C., Taylor, L. K., and Whitfield, D. L., "Multiblock Multigrid Solution of Three-Dimensional Incompressible Turbulent Flows About Appended Submarine Configurations," AIAA Paper 95-0203, Jan. 1995.
- Sheng, C., Chen, J. P., Taylor, L. K., Jiang, M. Y., and Whitfield, D. L., "Unsteady Multigrid Method for Simulating 3-D Incompressible Navier-Stokes Flows on Dynamic Relative Motion Grids," AIAA Paper 97-0446, Jan. 1997.
- Baldwin, B., and Barth, T., "A One-Equation Turbulence Transport Model for High Reynolds Number Flows of Unstructured Meshes," AIAA Paper 91-0721, Jan. 1991.
- Spalart, P., and Allmaras, S., "A One-Equation Turbulence Model for Aerodynamic Flows," AIAA Paper 92-0439, Jan. 1992.

⁶Anderson, W. K., and Bonhaus, D. L., "An Implicit Upwind Algorithm for Computing Turbulent Flows on Unstructured Grids," *Computers and Fluids*, Vol. 23, No. 1, 1994, pp. 1-21.

⁷Golub, G., and Loan, C. V., *Matrix Computations*, Johns Hopkins Univ. Press, Baltimore, MD, 1989, pp. 224-231.

⁸Anderson, W. K., "Grid Generation and Flow Solution Method for Euler Equations on Unstructured Grids," NASA TM-4295, April 1992.

⁹Batina, J. T., "Implicit Flux-Split Euler Schemes for Unsteady Aerodynamic Analysis Involving Unstructured Dynamic Meshes," AIAA Paper 90-0936, April 1990.

¹⁰Barth, T. J., and Linton, S. W., "An Unstructured Mesh Newton Solver for Compressible Fluid Flow and Its Parallel Implementation," AIAA Paper 95-0221, Jan. 1995.

¹¹Karypis, G., and Kumar, V., "METIS—A Software Package for Par-

tititioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices," User's Manual, Dept. of Computer Science/Army HPC Research Center, Univ. of Minnesota, Minneapolis, MN, Nov. 1997.

¹²Coles, D., and Wadcock, A. J., "Flying-Hot-Wire Study of Flow Past an NACA 4412 Airfoil at Maximum Lift," *AIAA Journal*, Vol. 17, No. 4, 1979, pp. 321-329.

¹³Valarezo, W., Dominik, C., McGhee, R., Goodman, W., and Paschal, K., "Multi-Element Airfoil Optimization for Maximum Lift at High Reynolds Number," AIAA Paper 91-3332, Sept. 1991.

J. Kallinderis
Associate Editor