AIAA JOURNAL
Vol. 30, No. 7, July 1992

Three-Dimensional Unstructured Multigrid for the Euler Equations

D. J. Mavriplis*
NASA Langley Research Center, Hampton, Virginia 23665

The three-dimensional Euler equations are solved on unstructured tetrahedral meshes using a multigrid strategy. The driving algorithm consists of an explicit vertex-based finite element scheme, which employs an edge-based data structure to assemble the residuals. The multigrid approach employs a sequence of independently generated coarse and fine meshes to accelerate the convergence to steady state of the fine grid solution. Variables, residuals, and corrections are passed back and forth between the various grids of the sequence using linear interpolation. The addresses and weights for interpolation are determined in a preprocessing stage using an efficient graph traversal algorithm. The preprocessing operation is shown to require a negligible fraction of the CPU time required by the overall solution procedure, whereas gains in overall solution efficiencies greater than an order of magnitude are demonstrated on meshes containing up to 350,000 vertices. Solutions using globally regenerated fine meshes as well as adaptively refined meshes are given.

I. Introduction

ALTHOUGH much literature has been published in recent years advocating the use of unstructured meshes, three-dimensional unstructured mesh schemes remain painfully inefficient when compared with available three-dimensional structured techniques. Although recent improvements have been achieved,[1,2] the two main reasons for these inefficiencies persist. The first is due to the indirect addressing performed when computing on random data sets, and the resulting scatter-gather operations required on vector computers that effectively lower the number of achievable Mflops. The second is a result of the simple explicit schemes generally employed for integrating the Euler equations to steady state. In addition, many three-dimensional unstructured schemes require excessive memory overheads that severely limit the size of the meshes that may be processed on present-day supercomputers. To construct an efficient three-dimensional unstructured solver, care must be taken to 1) minimize memory overhead, 2) minimize amount of gather scatter, and 3) provide rapid convergence to steady state.

The first two items can be influenced by the choice of data structures. In this work, we choose a vertex-based scheme, with an edge-based data structure, which minimizes the amount of indirection required in the residual evaluation loops. The third item represents an algorithmic issue. Although much work has been performed in two dimensions on direct,[3] iterative implicit,[4-6] and multigrid methods[7-10] for convergence acceleration, very little of this work has been extended to the three-dimensional setting, using realistically fine grids. Direct methods and many of the iterative implicit methods incur memory overheads that are too large to be practical for three-dimensional problems. Multigrid methods, on the other hand, appear to be well suited for three-dimensional unstructured problems, since they incur minimal overheads. Furthermore, multigrid methods have been shown to be among the most efficient known solvers for structured two- and three-dimensional problems,[11] and this experience has been reproduced in the unstructured context for two-dimensional problems. In this work, the unstructured multigrid algorithm originally developed in two dimensions by the author[7] is extended to three dimensions.

II. Design of a Three-Dimensional Unstructured Solver

Three-dimensional computational fluid dynamics problems often involve very complex geometries and require very high resolution, resulting in a large number of grid points. To be able to handle such complicated problems, and to avoid incurring excessive memory and CPU overheads, which may have the effect of rendering the problem intractable, efficient and unambiguous data structures must be devised for effectively representing the geometry and constructing the flow solver.

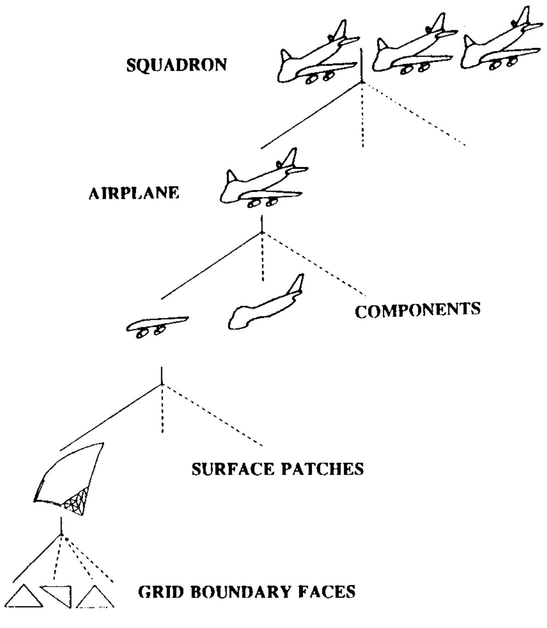

Fig. 1 Illustration of directed-acyclic-graph (DAG) data structure employed to represent the geometry.

Received May 16, 1991; presented as Paper 91-1549 at the AIAA 10th Computational Fluid Dynamics Conference, Honolulu, HI, June 24-26, 1991; revision received Oct. 24, 1991; accepted for publication Nov. 19, 1991.

*Senior Scientist, Institute for Computer Applications in Science and Engineering, MS 132C. Member AIAA.

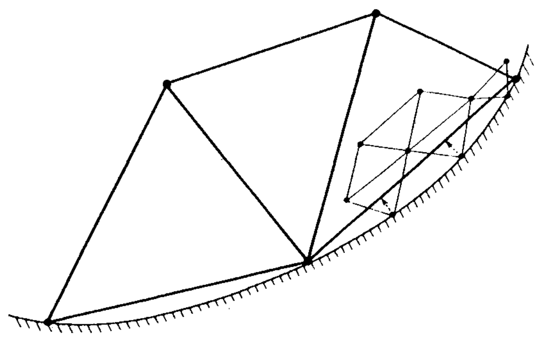

Fig. 2 Illustration of the relative displacement of coarse and fine grid boundary points due to the surface definition of the geometry.

A. Geometry Data Structures

One of the most widely used models for representing complex geometries in the fields of computer-aided design (CAD) and computational graphics is based on the hierarchical structure of a directed-acyclic graph (DAG),[12] in which higher level objects can be built using instances of simpler objects. For example, if we wish to compute the flow over a squadron of airplanes, the top of the tree (labeled squadron) would point to a number of lower level objects called "airplanes." Each airplane in turn points to its major components, such as fuselage, wing, nacelle, etc. These, in turn, point to sub-components, i.e., slat, main wing, flap, etc., and so on down the tree, until the smallest components are reached. These most basic components then point to a number of defining surface patches. We require the basic building blocks of the entire geometry to be surface patches, since we are interested in discretizing (gridding) the surface of the geometry as well as the volume around the geometry. Surface patches are the two-dimensional equivalent of the one-dimensional spline curves generally employed to describe two-dimensional geometries. A large variety of rectangular and triangular patch types has been devised over the years.[13] Thus, the data structures must accommodate any patch type, provided it has been previously defined. Finally, once the geometry has been discretized, each surface patch will point to a number of triangular faces that constitute the surface grid on that patch. A complete discretized geometry DAG is illustrated in Fig. 1. In actual fact, the geometry DAG is stored in reverse order from that described earlier. Hence, each boundary face has a tag associated with it that points to the parent surface patch, and each surface patch is associated with a component, etc. In this manner, when operating on individual boundary faces (e.g., when adding a point inside a boundary face or extracting a subset of the boundary faces for postprocessing), we can immediately determine which surface patch, component, body, etc., this boundary face belongs to, simply by traversing the tree. This data structure has some subtle implications concerning the discretization and solution process. For example, grid boundary faces that cross over surface-patch boundaries are not permitted. This appears to be a reasonable restriction, since patch boundaries usually correspond to regions of discontinuities in surface slope (i.e. a wing break section), or intersecting components (i.e., a wing-fuselage intersection).

A data structure for assigning boundary conditions to the flow solver is also required. A boundary face based structure is chosen, since a boundary face represents the smallest element of the discretized surface geometry. Hence, a tag is associated with each boundary face that identifies its boundary condition. This structure is completely independent from the geometry definition data structure, since we allow for the possibility of multiple boundary conditions on a single surface patch or boundary conditions extending across surface-patch boundaries. When node-based boundary conditions are required, these may be determined by scattering the face-based condition tags to the nodes and setting up a set of priority rules for ambiguous nodes bordering on regions with differing boundary conditions.

B. Flow-Solver Data Structure

To design appropriate data structures for a three-dimensional flow solver, it is first useful to look at some of the properties of general three-dimensional unstructured meshes. Such meshes may contain a mix of arbitrary polyhedral elements. We will restrict ourselves to the subset of these meshes involving only tetrahedra, hexahedra, and "in-between" elements such as prisms and pyramids and argue in favor of meshes containing only tetrahedral elements. For a mesh of hexahedral elements, neglecting boundary effects, if N represents the number of vertices, then we have N elements, $3N$ quadrilateral faces, and $3N$ edges. For a mesh of tetrahedral elements with N vertices, we have αN elements, where it is possible for α to be linearly related to N, thus yielding a quadratic growth in the number of tetrahedra with the number of vertices.[14] In practice, however, most suitable meshes contain $5 < \alpha < 6$, and this could in fact be made a grid-generation constraint. Tetrahedral meshes with N vertices thus yield αN cells, $2\alpha N$ triangular faces, and approximately $(\alpha+1)N$ edges. For meshes containing prismatic and pyramidal elements, the vertex-element-face-edge ratios lie in between these two extremes. The previous ratios point in favor of hexahedral elements. However, tetrahedral meshes offer the most flexibility in dealing with complex geometries, thus their widespread usage. A further property of tetrahedral meshes is that they are fully connected, i.e., each vertex within an element is connected to each other vertex of the element by an edge. Although the relative benefits of employing mixed elements is still an open question, there remains a strong incentive for remaining exclusively with tetrahedral elements for geometric flexibility, simplicity, and due to the fully connected property, as will be shown.

Whereas there appears to be little difference between a cell-centered and a vertex-based scheme in the case of hexahedral meshes, for tetrahedral meshes the situation is quite different. A tetrahedral cell-centered scheme results in αN unknowns, which are joined by $2\alpha N$ faces, across which discrete integrals must be evaluated to form the residuals. A vertex-based scheme results in N unknowns joined by $(\alpha + 1)N$ edges. Hence, a vertex scheme contains $5<\alpha<6$ less unknowns than the cell-centered scheme and thus yields a lower spatial resolution for a given mesh. However, the discrete integration around each unknown is potentially more accurate than in the cell-centered case, due to the high degree of each vertex (number of incident edges). On similar meshes, vertex schemes incur substantially lower memory overheads than cell-centered schemes and appear to yield similar accuracy. However, a detailed accuracy comparison between the two approaches has yet to be carried out.

A basic data structure is also required to gather and scatter information back and forth between neighboring vertices

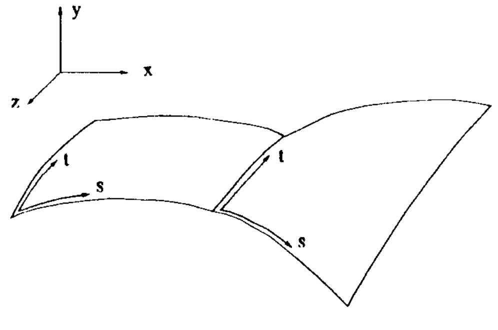

Fig. 3 Definition of physical coordinate system and individual surface-patch parametric coordinate systems.

throughout the solution process. An element-based data structure, traditionally used in the finite element context, requires $4\alpha N$ pointers (i.e., 4 vertices per element). A (triangular) face-based data structure (intuitively appealing for a finite volume scheme) requires $6\alpha N$ pointers and an edge-based data structure $2(\alpha + 1)N$ pointers. Thus, the edge-based data structure is the most compact of all. In fact, this structure, defined by a list of edges with the addresses of the two vertices delimiting each edge, represents the minimum amount of information required to describe the unstructured mesh, from which all other quantities (faces, elements) can be reconstructed. As will be shown, the fact that this data structure can be directly employed as the basis for the flow solver is a result of the fully connected property of tetrahedral meshes.

If we wish to solve the equation

$$Lu = 0 \tag{1}$$

where L is a linear operator, by discretizing this continuous equation in a vertex-based manner on a tetrahedral mesh, we obtain

$$[M_l]v = 0 \tag{2}$$

where v is now a discrete N-dimensional vector, N is the number of grid points, and $[M_l]$ is an $N \times N$ sparse matrix. The graph of a sparse matrix is defined by the graph obtained by drawing a line joining the two vertices corresponding to the row number and column number for each nonzero entry in the matrix.[15] If we employ a finite element discretization procedure, in which values are gathered from all vertices of each element, an update computed, and then scattered back to the element vertices, the sparse matrix $[M_l]$ must contain a nonzero entry for any pair of vertices within an element. It is therefore evident that, in the case of tetrahedral meshes, the fully connected property implies that the graph of the resulting sparse matrix is identical to the graph of the unstructured mesh. Hence, the sparse matrix $[M_l]$ may be represented in an edge-based format in which each edge of the mesh is associated with two nonzero entries (one entry for symmetric matrices) in the matrix $[M_l]$. This property is in fact true for any operator on any mesh that has a nearest neighbor stencil but does not hold for a finite element operator on hexahedral meshes, for example, since the matrix will contain entries relating vertices at diagonally opposed corners of the cell, which are not joined by a mesh edge.

In practice, the edge-based data structure may also be applied to nonlinear problems such as the Euler equations. A residual evaluation for the Euler equations may be recast as a sparse-matrix vector multiplication where the matrix coefficients are formed with edge-based metric quantities and the primitive variables at the vertices at either end of the edge. In fact, even for linear problems, it is usually not desirable to store all coefficients of the matrix. We choose to store only the minimum amount of information at each edge from which all matrix coefficients can be reconstructed. This is especially important for systems of equations and in operators involving a sequence of similar matrices. For the Navier-Stokes equations, the discretized viscous terms may also be constructed using the edge-based data structure, since these also result in nearest neighbor stencils in the case of constant viscosity, as pointed out by Barth.[16] To include the effect of variable viscosity in a purely edge-based data structure, the gradient of viscosity may be precomputed at each vertex, and then an extrapolation procedure using precomputed edge-based coefficients may be employed. Finally, the biharmonic dissipative operator employed to construct artificial dissipative terms in the present scheme, which results in a stencil that extends out to the neighbors of neighbors may be formed by a double application of a sparse matrix:

$$[M_l][M_l]v = [M_l]^2 v \tag{3}$$

Residual evaluations are thus constructed as a sequence of sparse-matrix vector multiplications, which are executed as loops over the edge-based data structures. To vectorize such loops, the edges must be divided into groups (colors) within which no vertex is accessed more than once. The original scalar loop is then replaced by an outer loop over the various colors or groups, with an inner vector loop over the edges of a given color.

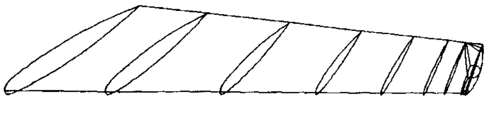
Fig. 4 Geometric surface patch definition employed for the ONERA M6 wing (total number of surface patches = 36).

C. Single-Grid Solver

The base solver employed to drive the multigrid algorithm represents the three-dimensional extension of the two-dimensional solver employed by Mavriplis[7] and is similar to the method employed by Jameson et al.[17] The flow variables are stored at the mesh vertices, and the residuals are assembled using loops over the edges of the mesh. The discretization scheme is based on a Galerkin finite element approach, using piecewise linear flux functions over the individual tetrahedra. This type of discretization corresponds to central differencing in structured-mesh terminology, and thus additional artificial dissipation terms must be introduced to maintain the stability of the scheme. These are constructed as a blend of a Laplacian and biharmonic operator on the conserved variables, the Laplacian dissipation being applied only in the vicinity of a shock, and the higher accuracy biharmonic dissipation being employed elsewhere in the flowfield.

The spatially discretized equations represent a large system of coupled ordinary differential equations, which are then integrated in time to obtain the steady-state solution. A five-stage Runge-Kutta scheme is employed for the time integration, where the convective terms are evaluated at each stage in the time-stepping scheme, and the dissipative terms are only evaluated at the first two stages and then frozen for the remaining stages. A complete multistage time step, in which the solution is advanced from time level n to level $n + 1$, can be written as

$$\begin{aligned}
w^{(0)} &= w^n \\
w^{(1)} &= w^{(0)} - \alpha_1 \Delta t \{Q[w^{(0)}] - D[w^{(0)}]\} \\
w^{(2)} &= w^{(0)} - \alpha_2 \Delta t \{Q[w^{(1)}] - D[w^{(1)}]\} \\
w^{(3)} &= w^{(0)} - \alpha_3 \Delta t \{Q[w^{(2)}] - D[w^{(1)}]\} \\
w^{(4)} &= w^{(0)} - \alpha_4 \Delta t \{Q[w^{(3)}] - D[w^{(1)}]\} \\
w^{(5)} &= w^{(0)} - \alpha_5 \Delta t \{Q[w^{(4)}] - D[w^{(1)}]\} \\
w^{(n+1)} &= w^{(5)}
\end{aligned} \tag{4}$$

with

$$\alpha_1 = 1/4, \quad \alpha_2 = 1/6, \quad \alpha_3 = 3/8, \quad \alpha_4 = 1/2, \quad \alpha_5 = 1$$

where w represents the conserved flow variables, Q is the convective residual, D denotes the dissipative operator, and Δt represents the discrete time step. This type of scheme requires the separate evaluation of the dissipative terms and the convective terms. The former require a sequence of two loops over edges, whereas the convective terms can be assembled in a single edge loop. This particular scheme has been designed to rapidly damp out high frequency error components,[11] which is a necessary characteristic for a multigrid driving scheme. Convergence to steady state is accelerated by employing local time stepping and implicit residual averaging,[7,11] which have previously been described in the context of unstructured meshes.

III. Multigrid Strategy

The idea of a multigrid strategy is to perform time steps on coarser meshes to calculate corrections to a solution on a fine mesh. The advantages of time stepping on coarse meshes are twofold: first, the permissible time step is much larger, since it is proportional to the cell size, and second, the work is much less because of the smaller number of grid points. On the finest grid of the sequence, the flow variables are updated by the five-stage scheme as shown in Eq. (4). The residuals and flow variables are then transferred to the next coarser grid. If R' represents the transferred residuals and w' the transferred flow variables, a forcing function on the coarse grid can be defined as

$$P = R' - R(w') \tag{5}$$

Now on the coarse grid, time stepping proceeds as shown next:

$$w^{(q)} = w^{(q-1)} - \alpha_1 \Delta t \{R[w^{(q-1)}] + P\} \tag{6}$$

for the qth stage. In the first stage, $w^{(q-1)}$ reduces to the transferred flow variable w'. Thus, the calculated residuals on the coarse grid are canceled by the second term in the forcing function P, leaving only the R' term. This indicates that the driving force for the solution on the coarse grid is provided by the fine grid residuals. Thus we are ensured that, when the fine grid solution is fully converged, no further corrections will be generated by the coarser grids. This procedure is repeated on successively coarser grids. When the coarsest grid is reached, the corrections are transferred back to the finer grids. The use of a multigrid method with unstructured meshes presents an additional challenge. Consistent coarse tetrahedral grids can no longer be formed by simply considering subsets of the fine grid vertices. An alternative would be to generate the fine mesh by repeatedly subdividing an initial coarse mesh in some manner. However, generally poor topological control of the fine mesh results from such a procedure. Another approach, known as the agglomeration technique, reconstructs coarse grids from a given fine unstructured grid by grouping neighboring elements together to form large polyhedral coarse-grid cells.[8,9] In the present work, it has been decided to pursue an unstructured multigrid approach in which a sequence of completely unrelated coarse and fine meshes are employed. This approach, which has already been demonstrated in two dimensions,[7] provides great flexibility in determining the configuration of the coarsest and finest meshes. Coarse meshes may be designed to optimize the speed of convergence, whereas fine meshes may be constructed based on solution accuracy considerations. In general, beginning from a fine grid, a coarser level is constructed that contains roughly half the resolution in each coordinate direction throughout the domain (about 1/8 the number of vertices). This process is repeated until the coarsest grid capable of representing the geometry topology is obtained. In the context of adaptive meshing, new finer meshes may be added to the multigrid sequence, using any given adaptive refinement technique, since no relation is assumed between the various meshes of the sequence.

The key to the success of such a strategy lies in the ability to efficiently transfer variables, residuals, and corrections back and forth between unrelated unstructured meshes. In the present context, this is performed using linear interpolation. For each vertex of a given grid, the tetrahedron that contains this vertex on the grid to which variables are to be interpolated is determined. The variable at this node is then linearly distributed to the four vertices of the enclosing tetrahedron. The main difficulty lies in efficiently determining the enclosing cell for each grid point. A naive search over all cells would lead to an $\mathcal{O}(N^2)$ complexity algorithm, where N is the total number of grid points, and would be more expensive than the flow solution itself. In this work, a graph traversal search routine with an expected complexity of $\mathcal{O}(N)$ is employed. This type of algorithm requires two additional data structures. First, a list of the tetrahedra in each mesh must be constructed. For each tetrahedron, the address of the four forming vertices are stored as well as the address of the four neighboring tetrahedra. The search begins by choosing a node on one grid and locating the enclosing tetrahedron on the other grid. This can

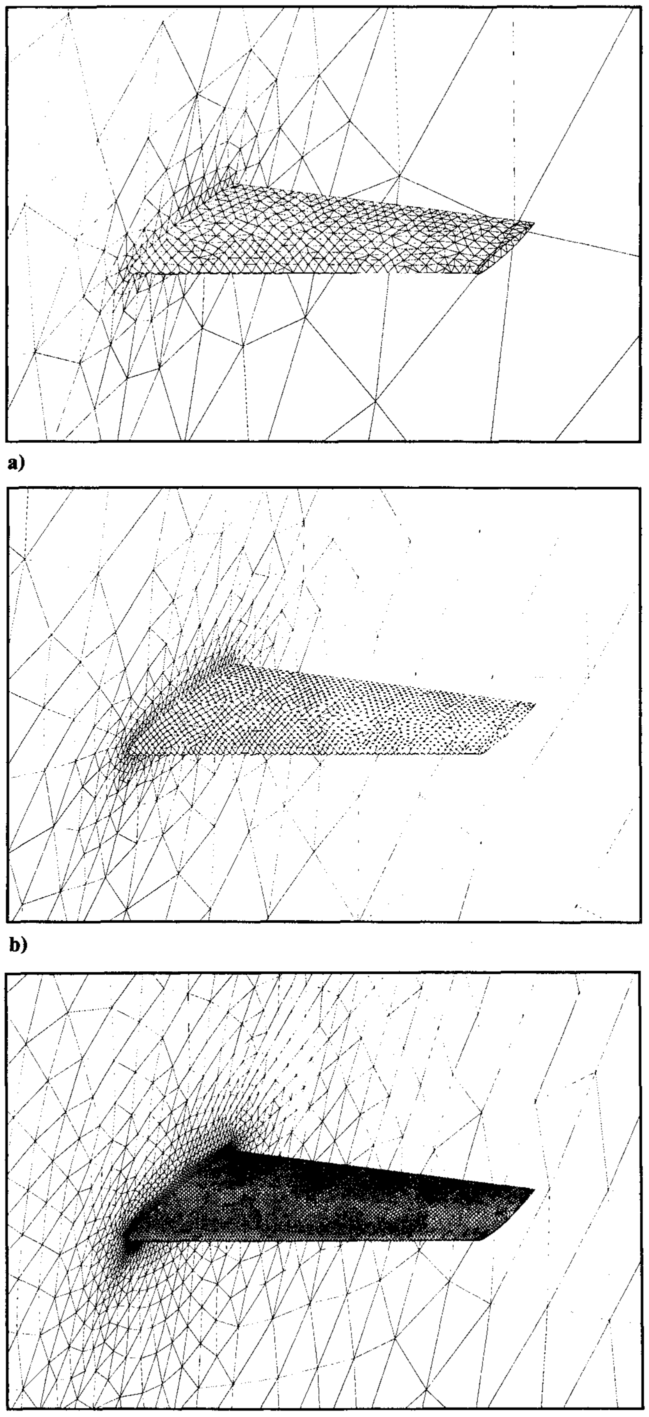
a)
b)
c)

Fig. 5 Sequence of global coarse and fine meshes employed for computing inviscid transonic flow over the ONERA M6 wing. Mesh 1: 2,800 nodes, 13,576 tetrahedra, 2,004 boundary faces. Mesh 2: 9,428 nodes, 47,504 tetrahedra, 5,864 boundary faces. Mesh 3: 53,961 nodes, 287,962 tetrahedra, 23,108 boundary faces. Mesh 4 (not shown): 357,900 nodes, 2,000,034 tetrahedra, 91,882 boundary faces.

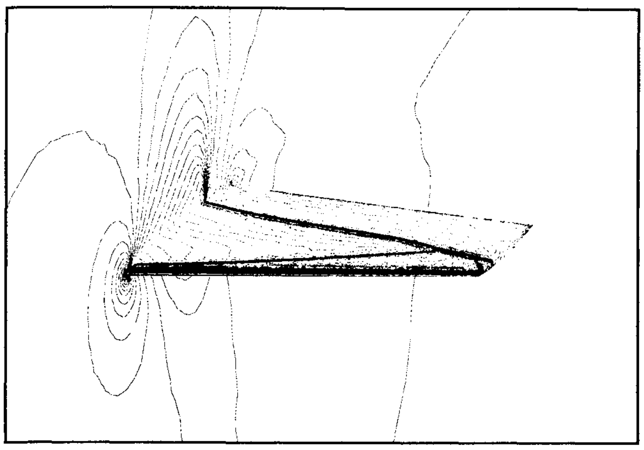

Fig. 6 Computed Mach contours on the finest grid of the multigrid sequence (mesh 4).

usually be determined a priori, for example, by choosing the minimum *x-y-z* node and the minimum *x-y-z* tetrahedron for the respective grids. We next chose a new node for which the enclosing cell is to be searched, and this node is taken as a neighbor of the previous node. As a starting guess we choose the tetrahedron that was previously found to enclose the first node, which is in the same vicinity as the new node. If this cell is not found to enclose the new node, we search the four neighbors of this cell, and then the neighbors of these neighbors, thus traversing through the mesh until the enclosing cell is located, at which point the process is repeated for a new node.

Additional complications arise at geometry boundaries. Since the geometry is described by a series of curved surfaces, the boundary points of a given fine grid will not necessarily be included on the surface defined by a coarser grid. In fact, for concave boundaries, such points may even lie outside of the coarse grid computational domain, as illustrated in Fig. 2, for the two-dimensional case. However, all such points are contained on the surface-patch definition of the geometry. Hence, rather than searching for the boundary faces from a particular grid that enclose a given boundary node from another grid in physical coordinates, this searching and subsequent interpolation are done in the two-dimensional *s-t* parametric space of the surface patch, as illustrated in Fig. 3. Thus, for each surface patch, we loop over the "grid 1" nodes on the patch and locate the "grid 2" boundary faces on that patch that enclose the nodes. This type of search cannot fail, since we have precluded the possibility of having boundary faces extending across surface-patch boundaries.

The interpolation patterns between the various meshes are completely determined by assigning to each mesh vertex four interpolation addresses and four interpolation weights, which are all computed in a preprocessing phase. In practice, this preprocessing has been found to require an amount of CPU time roughly equivalent to one or two flow solution cycles on the finest grid. The extra addresses and weights needed to determine the interpolation patterns represent an additional storage requirement of 8 words/node or about 10%. Since the coarse meshes in three dimensions generally contain one-half as many points in each coordinate direction as the next finer mesh, the total number of mesh points decreases by a factor of 8, when going to a coarser mesh. Thus the total additional memory requirement over a single grid solver is less than 25%. In the flow solution phase, the restriction and prolongation operations consume less than 6% of the total time required, since they merely represent a scatter of data to preassigned locations. Combined with the extra work on coarser meshes, and an extra required fine-grid residual evaluation (prior to the restriction operation), the total cost of a multigrid cycle is about 45% higher in terms of CPU time than the equivalent single fine-grid cycle.

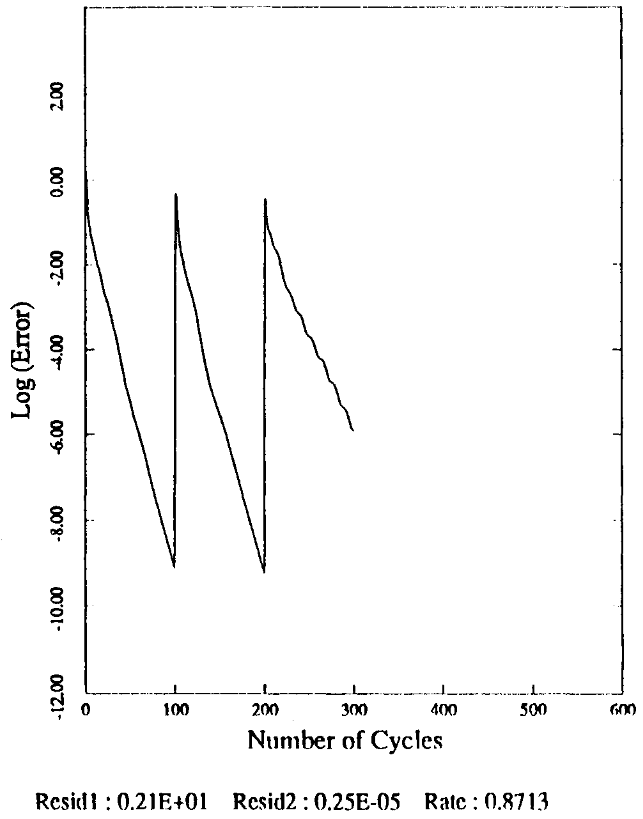

Fig. 7 Convergence rate of the unstructured multigrid algorithm on the three finest grids (meshes 2, 3, and 4) of the four grid sequence about an ONERA M6 wing as measured by the average density residuals vs the number of multigrid cycles.

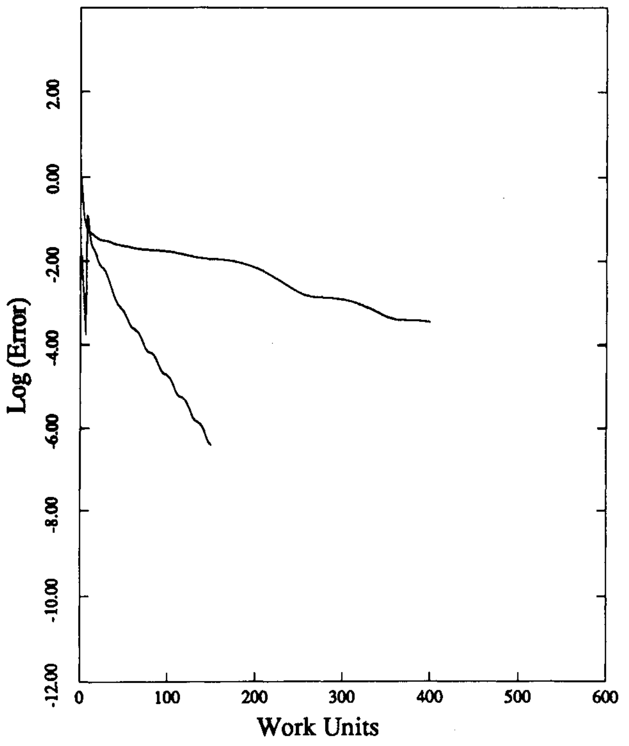

Fig. 8 Comparison of the multigrid convergence rate and the single grid convergence rate on the finest grid of the sequence (mesh 4) about the ONERA M6 wing as measured by the average density residuals vs the number of work units.

IV. Results

A. Globally Generated Meshes

A standard three-dimensional test case has been employed to validate the present code and to evaluate the efficiency of the multigrid strategy. The geometry consists of an ONERA M6 wing delimited by a symmetry plane. The surface-patch geometry definition is illustrated in Figure 4. A total of 36 surface patches were employed to describe the wing, symmetry plane, and outer boundaries of the flowfield. On the wing surface, these consist of mostly rectangular bicubic Coon's patches with several triangular patches near the wing tip. A sequence of four meshes was generated about this configuration using the advancing-front technique.[18] The resulting surface meshes are displayed in Fig. 5. The finest grid, denoted as mesh 4, contains 357,900 vertices and 2,000,034 tetrahedra, whereas the coarsest grid of the sequence, denoted as mesh 1, contains only 2800 vertices and 13,576 tetrahedra. The flow over the ONERA M6 wing has been computed with the multigrid strategy using this sequence of four meshes. The Mach number and incidence for this case are 0.84 and 3.06 deg, respectively. At these conditions, the flow is transonic, and a double shock is formed over most of the span of the wing. The computed Mach contours on the finest grid of the sequence are shown in Fig. 6, illustrating the double shock pattern, which appears to be well resolved on this relatively fine mesh. The multigrid convergence rate for this case is depicted in Fig. 7, where 100 multigrid *W* cycles were successively run on each of the three finest grids of the sequence, after which the solution was interpolated to the next finer grid and multigrid time stepping resumed. In practice, as few as 25 cycles may be performed on the coarser grids, before initializing the fine-grid multigrid solution process. However, an equal number of cycles was run on each grid to compare the multigrid convergence rates on the various grids. As can be seen, the multigrid convergence rate degrades only slightly when going to finer meshes. On the finest mesh, the flowfield residuals were reduced by 6 orders of magnitude over 100 cycles, for an average spectral radius of 0.871. The single grid convergence rate for this case (including local time stepping and residual averaging) is compared to the multigrid convergence rate by plotting both convergence histories in terms of work units in Fig. 8, where a work unit is defined as the time required for a single-grid explicit cycle. Even after 400 cycles, the single-grid calculation is seen to have reduced the residuals by only 3 orders of magnitude, a feat that required over 2 h of CPU time on a single processor of the CRAY-YMP. This single-grid run required 33 MW of memory and ran at a speed of 19 s/cycle (5-stage Runge-Kutta with 2 dissipation evaluations, implicit residual smoothing), which translates to a memory requirement of 92 words per vertex and 53 μs/vertex/cycle. In contrast, the multigrid run required 43 MW of memory and 34 s per multigrid cycle, which translates to 120 words per fine grid vertex and 95 μs/vertex/multigrid cycle. This increase in resource requirements over the single-grid case is somewhat larger than that stated in the previous section and is due to the coarse grids of the sequence containing more than 1/8 the number of nodes of the previous finer grid. It should also be noted that these figures vary somewhat with the ratio of boundary vertices to interior nodes on a given grid. The preprocessing time for setting up the interpolation patterns between the various grids required 40 s of CPU time, and the actual transfer of data back and forth between the various meshes of the sequence within each multigrid cycle consumed less than 6% of the total time for this run. An engineering accuracy calculation could be achieved with this case by performing 25 cycles on each of the coarser grids and 50 cycles on the finest grid, resulting in a residual reduction of 3.5 orders of magnitude, and requiring 30 CPU min on a single CRAY-YMP processor.

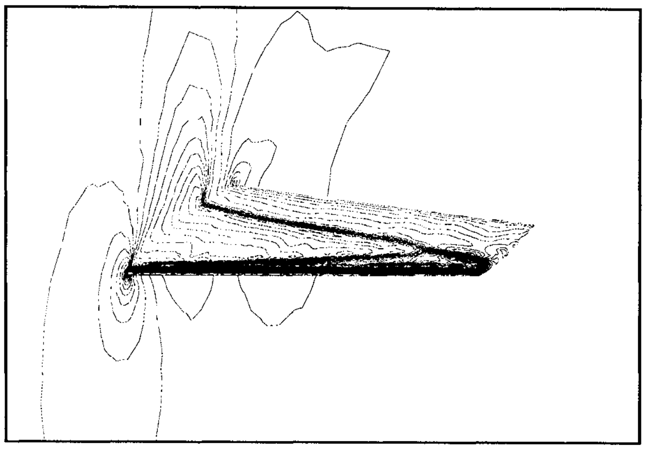
Fig. 9 Computed Mach contours on the second finest mesh (mesh 3) of the multigrid sequence over the ONERA M6 wing.

Figure 9 depicts the computed Mach contours obtained after 50 multigrid cycles on the second finest grid, denoted as mesh 3 in Fig. 5, which contains a total of 53,961 points and 287,962 tetrahedra. The double shock pattern seems to be reasonably well captured on this coarser grid. The values of the lift and drag coefficients for this case are compared with the values obtained on the finer mesh (mesh 4) in Table 1. The coarse grid lift coefficient is seen to be within 2% of the more accurate fine grid value. It appears that, because of the possibility of clustering more grid points near the body surface with the unstructured mesh, reasonably accurate solutions can be obtained with a relatively lower number of mesh points, as compared with a structured grid calculation. The solution on this grid required only 7 MW of memory and 5 CPU minutes, during which the residuals were reduced by 5 orders of magnitude. This type of calculation can easily be performed in the interactive mode on the NAS CRAY-YMP.

B. Adaptively Generated Meshes

One of the main advantages of unstructured grids is the possibility of easily performing adaptive meshing. The present

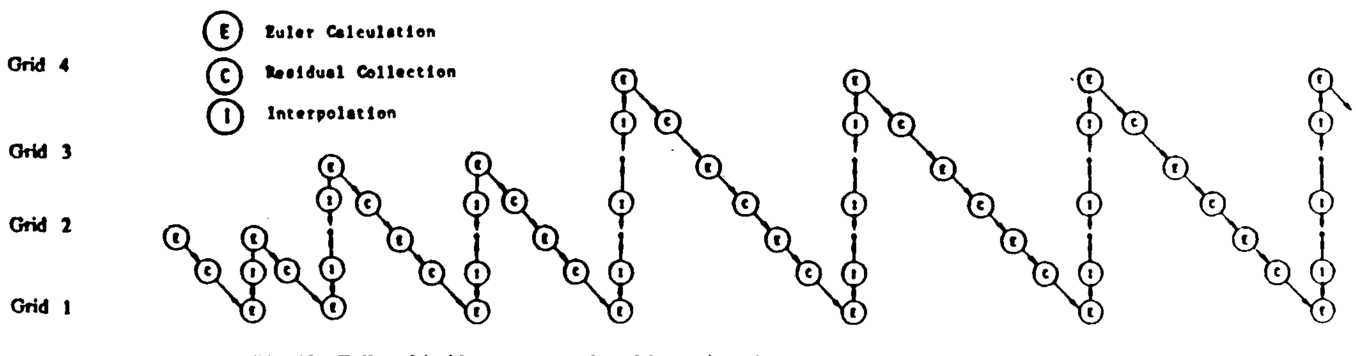

Fig. 10 Full multigrid strategy employed in conjunction with the adaptive meshing strategy.

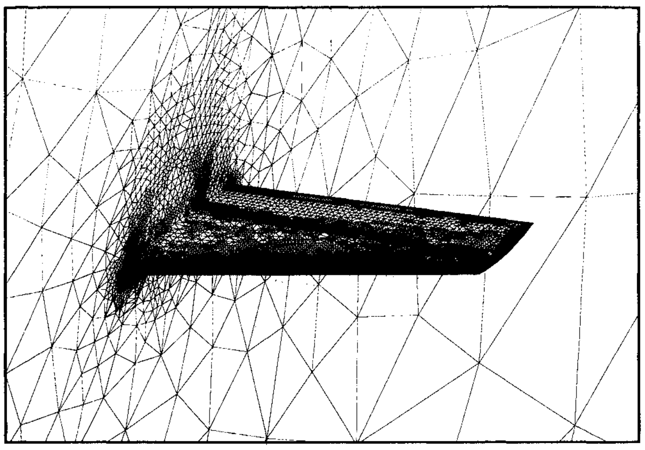

Fig. 11 Finest adapted mesh generated about the ONERA M6 wing (number of nodes = 173,412, number of tetrahedra = 1,013,718).

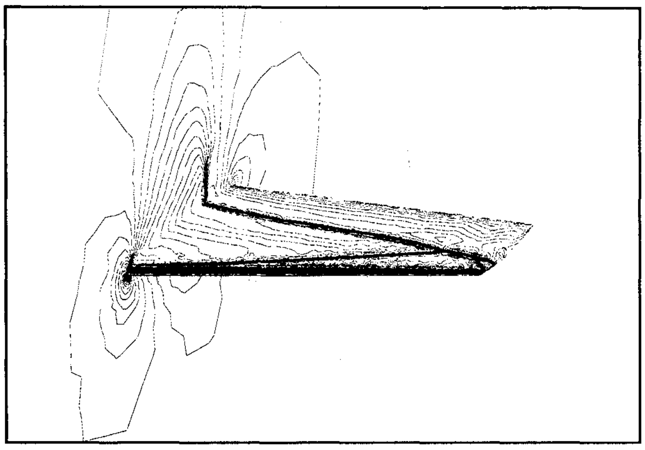

Fig. 12 Computed mach contours on the adaptively generated mesh about the ONERA M6 wing.

multigrid approach has been devised to accommodate adaptive meshing strategies. Since the unstructured multigrid assumes no relation between the various coarse and fine meshes of the sequence, finer meshes may be generated either by global remeshing, as in the previous case, or by adaptive refinement of the current finest mesh. A full multigrid strategy is employed in conjunction with adaptive meshing, as illustrated in Fig. 10. Multigrid time stepping is first performed on a sequence of globally generated meshes. When (partial) convergence is obtained on the finest mesh, a new finer mesh is created by adding new points to the existing fine mesh in regions of high gradients and locally restructuring the mesh. After all new mesh points have been inserted, the patterns for interpolating data between the new mesh and the previous mesh are computed and stored. The solution is then interpolated onto the new fine mesh, and multigrid time stepping is resumed on this augmented sequence of meshes, which contains the new adapted mesh as the finest mesh. This procedure may be repeated, each time adding a new mesh to the sequence, until the desired solution accuracy has been obtained. The undivided gradient of density is employed as a refinement criterion. For each tetrahedron, the density difference along all six edges of the cell is examined. If any of these differences is larger than some fraction of the average density difference over all mesh edges, new mesh points are added at the midpoints of all six edges, thus insuring an isotropic refinement. Each new point is inserted into the mesh using Bowyer's Delaunay triangulation algorithm. This algorithm consists of locating the union of tetrahedra whose circumspheres contain the new point. The mesh is then locally restructured by removing the union of all such tetrahedra and creating new tetrahedra by joining the new point to the vertices of the faces delimiting the region of removed tetrahedra. If the original mesh is a Delaunay triangulation, this procedure is guaranteed to result in a consistent (Delaunay) mesh.[19] However, the original meshes to be refined have been generated using the advancing-front technique,[18] and hence are not necessarily Delaunay triangulations. Therefore, each time a new point is inserted, the resulting local restructured mesh is checked for consistency; if negative cell volumes are created, then the mesh point is rejected. Points that are rejected may often be successfully reintroduced after all other points have been inserted, since the mesh becomes more "Delaunay" as more points are inserted. In practice, the advancing-front meshes do not represent large deviations from the corresponding "Delaunay" meshes and generally less than 2% of the attempted points are rejected. When new boundary points are introduced, they are repositioned onto the analytic surface-patch definition of the geometry by recomputing the physical coordinates of the new point based on the assigned parametric patch coordinates s and t, which are taken as the average of the parametric coordinates of the two vertices at either end of the generating boundary edge. Special care is also taken to preclude the possibility of newly formed boundary faces from crossing over adjacent surface-patch boundaries.

The ONERA M6 wing case has been recomputed using the adaptive meshing strategy. The final adapted mesh is shown in Fig. 11. This represents the fourth mesh in the multigrid sequence, where the two finest meshes have been generated adaptively, and the two coarse meshes correspond to the coarsest meshes (meshes 1 and 2) of the sequence depicted in Fig. 5. The final mesh contains a total of 173,412 vertices and 1,013,718 tetrahedra. Extensive refinement is seen to occur in the leading-edge region and in the vicinity of the shocks. Although this mesh contains roughly half the number of vertices of the finest mesh (mesh 4) of the previous case, it contains slightly higher maximum resolution in the leading-edge and shock regions. The computed Mach contours for this

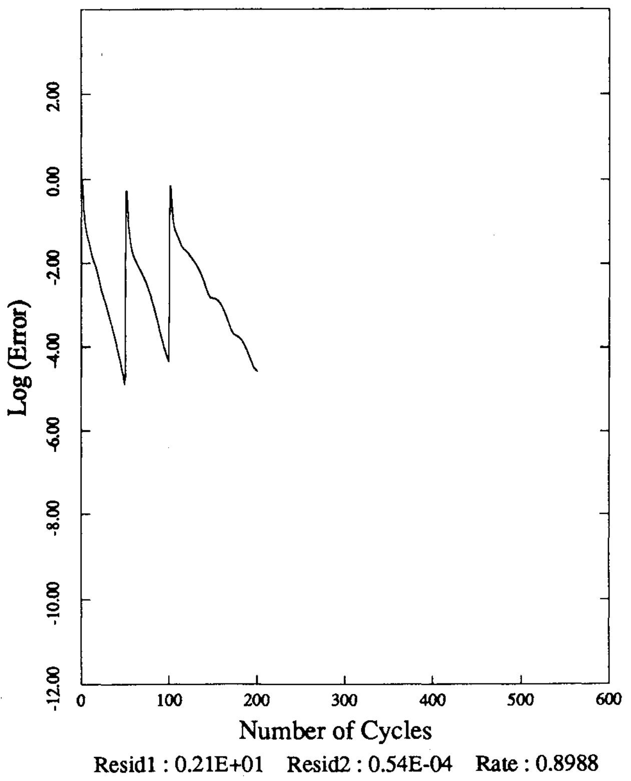

Fig. 13 Convergence rate of the unstructured multigrid algorithm on the adaptively generated sequence of meshes about the ONERA M6 wing as measured by the average density residual vs the number of multigrid cycles.

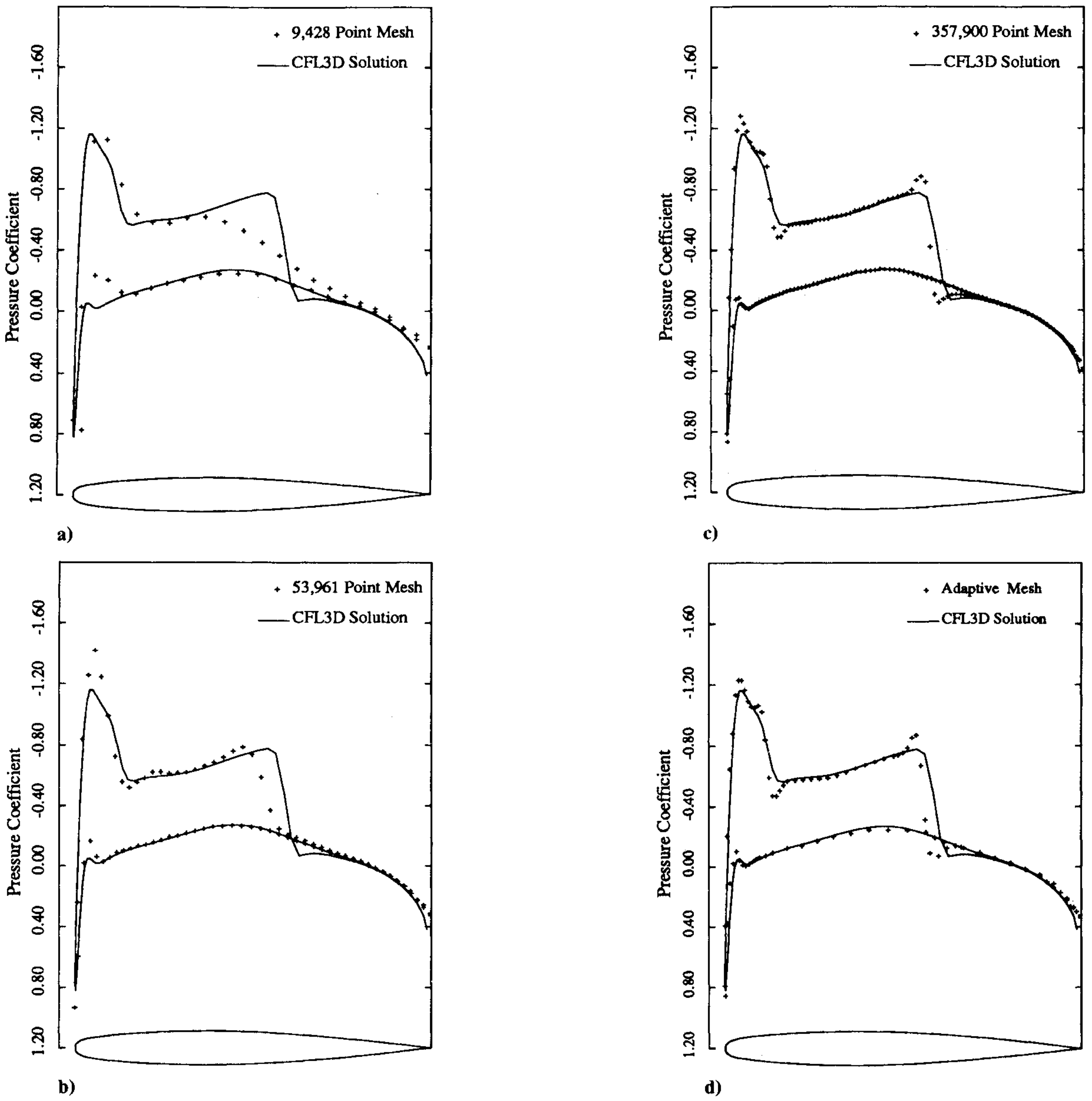

Fig. 14 Computed surface pressure coefficients at the 44% span location for the three finest grids of the globally generated mesh sequence and the finest adaptively generated mesh and comparison with structured grid solution from Ref. 20 on a 193 x 33 x 33 C-O mesh.

Table 1 Computed force coefficients on the three finest meshes of the globally generated mesh sequence and on the finest adaptively generated mesh[a]

No. of nodes	No. of tetrahedra	No. of boundary faces	Lift coefficient	Drag coefficient
9,428	47,504	5,864	0.2713	0.0275
53,961	287,962	23,108	0.2871	0.0188
357,900	2,000,034	91,882	0.2923	0.0131
173,412 (adapted)	1,013,718	32,212	0.2901	0.0130

[a]For this geometry, no. of boundary nodes = no. of boundary faces/2 + 2).

case are depicted in Fig. 12, where good resolution of the shocks and leading-edge expansion are observed. The computed lift and drag coefficients are compared with those obtained on the nonadapted meshes in the previous section in Table 1. Good agreement between the adapted mesh and the global fine mesh drag values is observed. The adapted mesh lift coefficient, however, is somewhat lower than the value obtained on the global fine mesh. This is probably a result of the particular refinement criterion chosen and will be the subject of future investigations. The multigrid convergence rate for this case is shown in Fig. 13. The fine grid residuals were reduced by roughly 5 orders of magnitude over 100 multigrid cycles, for an average reduction of 0.898 per multigrid cycle, with each cycle requiring roughly 20 s of CPU time. This case required a total of 22 MW of memory and 33 min of CPU time, which represents a savings of 2 with respect to the globally refined meshes of the previous section, while providing better shock and expansion resolution. The beneficial effects of adaptive meshing can, of course, be greatly increased through the use of additional adaptive grid levels.

Figure 14 compares the computed surface pressure coefficients at the 44% span location on the wing for the three finest global meshes (meshes 2, 3, and 4 of Fig. 5) and the finest adaptively generated mesh with a solution obtained from a well-established upwind structured mesh Euler solver from Ref. 20 on a 193×33×33 C-O mesh (210,177 points). As the unstructured grid is refined, the solution is seen to converge to the structured-grid solution. In fact, the leading-edge suction peaks appear to be better resolved than in the structured-grid solution. The shock position is, however, slightly forward of the shock location computed on the structured grid. This is due to the proximity of the outer boundary on the unstructured grids, where a uniform flow boundary condition is applied. The effect of the outer boundary on solution accuracy will be revisited in future work. In Table 1, the computed lift and drag coefficients for these same meshes are displayed to give an indication of the rate of grid convergence of the present scheme and the accuracy of the adapted mesh results.

V. Conclusion

Multigrid has proven to be an effective means for accelerating the convergence to steady state of three-dimensional unstructured Euler solvers. Increases in efficiency of over an order of magnitude have been demonstrated in this work. The present approach also offers the possibility of enhancing solution accuracy very effectively through the use of adaptive meshing techniques. At present, the adaptive-meshing procedure and the flow-solving procedure are performed separately. In the future, the integration of these two tasks into a single code is planned. The results obtained in this work suggest that it should be possible to solve most flows around complex geometries with on the order of 100 words/vertex and 75–100 μs/mg-cycle/node in about 100 multigrid cycles. Therefore, it seems that accurate solutions of steady inviscid flows over complete aircraft configurations using unstructured meshes within a reasonable turnaround time are well within the reach of present-day supercomputers, especially given the potential beneficial effects of multitasking on such machines.

Acknowledgments

The help of Shahyar Pirzadeh and Paresh Parikh (of Vigyan, Inc.) is greatly appreciated for providing the surface definition and generating the global coarse and fine meshes about the ONERA M6 wing, without which this work would not have been possible. Special thanks are also due to Clyde Gumbert (Computational Aerodynamics Branch, NASA Langley) for providing assistance with the three-dimensional graphics used in this paper. This work was made possible in large part due to the computational facilities made available by the National Aerodynamic Simulation facility.

References

[1]Vassberg, J. C., and Dailey, K. R., "AIRPLANE: Experiences, Benchmarks and Improvements," AIAA Paper 90-2998, Aug. 1990.

[2]Frink, N. T., Parikh, P., and Pirzadeh, S., "A Fast Upwind Solver for the Euler Equations on Three-Dimensional Unstructured Meshes," AIAA Paper 91-0102, Jan. 1991.

[3]Venkatakrishnan, V., and Barth, T. J., "Application of Direct Solvers to Unstructured Meshes for the Euler and Navier-Stokes Equations Using Upwind Schemes," AIAA Paper 89-0364, Jan. 1989.

[4]Hassan, O., Morgan, K., and Peraire, J., "An Implicit Finite-Element Method for High Speed Flows, AIAA Paper 90-0402, Jan. 1990.

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

[6]Venkatakrishnan, V., and Mavriplis, D., "Implicit Solvers for Unstructured Meshes," *Proceedings of the 10th AIAA Computational Fluid Dynamics Conference* (Honolulu, HI), June 1991 (AIAA Paper 91-1537).

[7]Mavriplis, D. J., "Multigrid Solution of the Two-Dimensional Euler Equations on Unstructured Triangular Meshes," *AIAA Journal,* Vol. 26, No. 7, 1988, pp. 824–831.

[8]Lallemand, M. H, and Dervieux, A., "A Multigrid Finite-Element Method for Solving the Two-Dimensional Euler Equations," *Proceedings of the Third Copper Mountain Conference on Multigrid Methods, Lecture Notes in Pure and Applied Mathematics,* edited by S. F. McCormick, Marcel Dekker, New York, April 1987, pp. 337–363.

[9]Smith, W. A., "Multigrid Solution of Transonic Flow on Unstructured Grids," *Recent Advances and Applications in Computational Fluid Dynamics, Proceedings of the ASME Winter Annual Meeting,* edited by O. Baysal, Vol. 103, American Society of Mechanical Engineers, New York, Nov. 1990, pp. 145–152.

[10]Leclercq, M. P., "Resolution des Equations d'Euler par des Methodes Multigrilles Conditions aux Limites en Regime Hypersonique," Ph.D. Dissertation, Applied Mathematics, Université de Saint-Etiénne, Saint-Etiénne, France, April 1990.

[11]Jameson, A., "Transonic Flow Calculations," Princeton Univ. Rept. MAE 1751, Princeton, NJ, June 1984.

[12]Welling, J., Nuuja, C., and Andrews, P., "P3D: A Lisp-Based Format for Representing General 3D Models," *Proceedings of the Supercomputing '90 Conference,* IEEE Computer Society Press, New York, Nov. 1990.

[13]Farin, G. E., *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide,* 2nd edition, Academic Press, New York, 1990.

[14]Preparata, F. P., and Shamos, M. I., *Computational Geometry, An Introduction,* Texts and Monographs in Computer Science, Springer-Verlag, New York, 1985.

[15]Duff, I. S., Erisman, A. M., and Reid, J. K., "*Direct Methods for Sparse Matrices,* Oxford Science Publications, Clarendon Press, Oxford, England, UK, 1986.

[16]Barth, T. J., "Numerical Aspects of Computing Viscous High-Reynolds Number Flows on Unstructured Meshes," AIAA Paper 91-0721, Jan. 1991.

[17]Jameson, A., Baker, T. J., and Weatherill, N. P., "Calculation of Inviscid Transonic Flow over a Complete Aircraft," AIAA Paper 86-0103, Jan. 1986.

[18]Parikh, P., Pirzadeh, S., and Lohner, R., "A Package for 3-D Unstructured Grid Generation, Finite-Element Flow Solution and Flow-Field Visualization," NASA CR-182090, Sept. 1990.

[19]Baker, T. J., "Three Dimensional Mesh Generation by Triangulation of Arbitrary Point Sets," AIAA 8th Computational Fluid Dynamics Conference, AIAA Paper 87-1124, June 1987.

[20]Anderson, W. K., Thomas, J. L., and Whitfield, D. L., "Multigrid Acceleration of the Flux Split Euler Equations," AIAA Paper 86-0274, Jan. 1986.