AIAA JOURNAL
Vol. 32, No. 6, June 1994

Three-Dimensional Adaptive Grid-Embedding Euler Technique

Roger L. Davis* and John F. Dannenhoffer III†
United Technologies Research Center, East Hartford, Connecticut 06108

A new three-dimensional adaptive-grid Euler procedure is presented that automatically detects high-gradient regions in the flow and locally subdivides the computational grid in these regions to provide a uniform, high level of accuracy over the entire domain. A tunable, semistructured data system is utilized that provides global, topological unstructured-grid flexibility along with the efficiency of a local, structured-grid system. In addition, this data structure allows for the flow solution algorithm to be executed on a wide variety of parallel/vector computing platforms. An explicit, time-marching, control volume procedure is used to integrate the Euler equations to steady state. In addition, a multiple-grid procedure is used throughout the embedded-grid regions as well as on subgrids coarser than the initial grid to accelerate convergence and properly propagate disturbance waves through refined-grid regions. Upon convergence, high flow gradient regions, where it is assumed that large truncation errors in the solution exist, are detected using a combination of directional refinement vectors that have large components in areas of these gradients. The local computational grid is directionally subdivided in these regions and the flow solution is reinitiated. Overall convergence occurs when a prespecified level of accuracy is reached. Solutions are presented that demonstrate the efficiency and accuracy of the present procedure.

Introduction

THE focus of many computational fluid dynamics researchers has recently been on the development of a new generation of "intelligent" prediction techniques that use grid adaptation and grid enrichment to enhance the quality of the solution. One of the engineering needs that drives the development of these procedures is the requirement of adequate accuracy to determine absolute levels of performance. With this capability, engineers can use computational procedures to reliably determine optimal designs or confidently analyze various configurations without being concerned about truncation error. A second need that also drives this research is the need for a computationally efficient procedure that obtains the required accuracy using the minimal amount of computer resources and, hence, computational grid points. Much has now been achieved in the creation of accurate and efficient two-dimensional adaptive-grid procedures for a wide range of applications. However, less has been accomplished in the development of three-dimensional solution-adaptive procedures.

By far, most of the research in the development of three-dimensional adaptive-grid prediction techniques has focused on totally unstructured-grid schemes utilizing concepts borrowed from finite element theory with linear, tetrahedral elements or cells. Examples of some of the more recent efforts using this approach include those of Lohner and Baum,[1] Dawes,[2] Morgan et al.,[3] and Mavriplis.[4] Grid adaptation occurs from the subdivision of tetrahedral elements into two or more subelements depending on how additional nodes are added. The strength of this type of procedure lies in its applicability to virtually any type of configuration. However, there are several weaknesses to this approach. Because of the use of tetrahedral elements, integration of the governing equations becomes at least three times[5] more expensive than procedures that use hexahedral cells. Application of multiple-grid acceleration techniques with unstructured tetrahedral solvers, although not impossible, becomes difficult to implement and can require a completely separate, additional data structure. Unstructured grid procedures also typically require large amounts of computer storage for cell-to-node connectivity tables for each cell. In addition, due to the use of this connectivity table and the indirect addressing of nodal information, computational efficiency is diminished during the flow integration for these techniques. Finally, tetrahedral elements become increasingly less robust and efficient for the resolution of two-dimensional flow features such as shock sheets and viscous layers.[5]

The research presented in this paper has been focused on the development of a three-dimensional solution-adaptive Euler procedure that avoids many of the problems related to unstructured-grid techniques through the use of a semistructured-grid approach. In this approach, small blocks of hexahedral, bilinear elements are used to discretize the domain. Grid adaptation results from subdivision of cells into two, four, or eight subcells (h-refinement). Similar procedures have been developed by Aftosmis,[6] Berger,[7] Oden et al.,[8] Melton et al.,[9] and Schmidt and Sturler.[10] The features of the present procedure that make it unique are its tunable data structure for parallel/vector computing, directional grid-embedding adaptation, and multiple-grid acceleration technique. This paper will describe the adaptation strategy, the data structure, and the flow integration technique followed by results that demonstrate the accuracy and efficiency of the current procedure.

Solution Strategy

For the prediction of steady, inviscid flows, the time-dependent Euler equations are integrated in time on a prescribed initial computational grid until a steady-state solution is achieved. Upon convergence, regions of high-flow gradients are located in the solution domain, and the computational grid in these regions is marked for enrichment. The computational cells in the vicinity of these regions are subdivided to create an embedded group of cells with grid spacing that are half of the original parent cell's spacing. The Euler equations are then solved on the new computational grid. This process continues until a user-specified level of accuracy is achieved. In general, this loop needs to be executed fewer than five times to obtain engineering accuracy.

Feature Detection

It is assumed, based on series expansion arguments and numerical experience, that the truncation error in the flow

Presented as Paper 93-0330 at the AIAA 31st Aerospace Sciences Meeting, Reno, NV, Jan. 11–14, 1993; received Feb. 6, 1993; revision received Nov. 2, 1993; accepted for publication Nov. 26, 1993.

*Senior Principal Engineer, Computational and Design Methods. Member AIAA.

†Senior Research Engineer, Advanced Software Methods. Senior Member AIAA.

solution procedure is proportional to the primary variable gradients and the grid spacing in the direction of the gradients. Therefore, by identifying high-gradient flow features in regions of coarse grid spacing, areas of high-truncation error can be located and marked for grid enrichment. High-gradient flow features are detected in the current procedure through a two-step process.

First, flow features such as shocks, stagnation points, and velocity-slip lines in regions of coarse grid spacing are identified through the use of refinement parameter vectors that have large components in the immediate vicinity of these features and in the direction(s) normal to these features. In the present scheme, a combination of the first difference of velocity and the first difference of pressure in each of three local grid directions is used to identify features such as those mentioned earlier.

The second phase of the feature finder is the determination of "how large" a refinement parameter needs to be before the grid is refined. Here, a threshold algorithm[11] is used to find the value of the refinement parameter that separates the features from the background. This classical signal-to-noise discriminator detects rapid changes in the cumulative distribution of the refinement parameter to set a division threshold; all cells with values of the refinement parameter exceeding this threshold are flagged for division. In the current procedure, the maximum of the three directional refinement parameter components is used in the threshold algorithm to determine the appropriate division threshold.

Directional Embedding

Once the division threshold is determined, all of the computational cells that have a refinement parameter component greater than the threshold are marked for division in that particular direction. Thus, grid subdivision in multiple directions occurs when more than one refinement parameter component exceeds the threshold (Fig. 1). This technique represents a three-dimensional extension of the two-dimensional directional grid-embedding adaptation approach of Kallinderis.[12] With this approach, the computational grid only subdivides in the directions necessary to reduce the appropriate truncation error. This directional flexibility in grid-embedding adaptation is essential for three-dimensional flows to efficiently resolve critical flow features. Since many high-gradient flow phenomena are one or two dimensional in nature, the addition of grid points along these features greatly increases the solution cost but provides little in accuracy.

Data Structure

The design of an efficient data structure is one of the keys to the effective use of parallel processing in domains that contain locally embedded regions. Specifically, the data structure must 1) allow for a locally structured view of the data and simultaneously a globally unstructured view; 2) allow for the application of a variety of explicit or implicit flow solution algorithms; 3) support hierarchically organized, directionally-divided grid cells; 4) be consistent with general, block-structured grid inputs; 5) support parallel computing of varying granularities and on a broad range of computer architectures; and 6) make efficient use of computer memory.

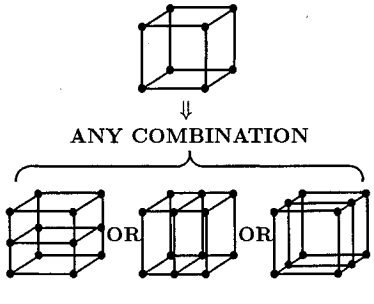

Fig. 1 Directional grid-embedding scheme.

All of the aforementioned requirements can be satisfied through the use of $N \times N \times N$ microblocks, called chunks; the images of chunks on the domain boundaries produce $N \times N$ structures called patches. The chunk (and subsequently patch) edge size parameter N allows one to easily control the granularity of the data, thus allowing the data structure to be easily mapped to any computer architecture.

Chunks

Chunks are created from a general block-structured input description by a simple decomposition process. Directional subdivision as previously discussed is actually performed on a per-chunk basis; that is, all of the cells within a chunk are subdivided in the same directions. Thus, arbitrary subdivision of a chunk produces two, four, or eight children chunks that are hierarchically related to their parent. This hierarchical structure is an essential ingredient in the multiple-grid acceleration scheme that will be described later.

Data within each chunk are structured (providing a locally structured view); however, chunks can be adjoined in any arbitrary manner (providing a globally unstructured view). This unique combination of views is referred to as a semistructured data system.

One of the major disadvantages of most other data structures is that they are extremely memory intensive because of the need to store interconnectivity information for each element. By using chunks, the connectivity information can be stored on a per-chunk basis, thus keeping memory requirements to a minimum. Table 1 lists the storage required by the current data structure as a function of the chunk edge size parameter N. Large values of N result in the lowest data structure memory requirements but also the highest adaptation granularity. Thus, an optimal value of N must be chosen that provides an acceptable data structure overhead with a reasonable adaptation granularity. In the present procedure, this has been accomplished by setting the chunk edge size parameter to 5. As will be discussed later, this edge size also produces nearly optimal intrachunk vector lengths. Thus, the total storage required for the Euler flow solver and the data structure for this chunk size is currently 20 words per node.

Patches

Patches are created from the faces of chunks that lie on a computational domain boundary. Subdivision of a chunk can therefore produce two or four children patches. Information necessary to perform the flow solution boundary conditions, including boundary condition type, surface normals, and auxiliary information such as freestream conditions, is stored at the nodes within a patch. The coordinates of additional nodes created from subdivision that lie on a patch are obtained from splines of the original initial grid. The information to create these splines is part of the general, multiblock input.

Neighbors

In the present procedure, the grid spacing between neighboring chunks is only allowed to change by a factor of two to minimize the discretization error in the numerical solution at these interfaces. Thus, the chunk numbers corresponding to the neighbors of a given chunk must be stored so that this grid-spacing constraint can be maintained. Before division of a particular chunk can occur, each set of neighbors is first interrogated to determine if it first should be subdivided. In

Table 1 Current data structure storage requirements as a function of chunk size

Chunk edge size, N	Words/node
2	53.8
3	10.3
5	4.0
9	2.7
17	2.2
33	2.1
65	2.0

the current procedure, a chunk can have as many as four neighbors corresponding to each of the six chunk faces.

Flow Solution Procedure

The flow solution algorithm for the Euler equations consists of the basic (fine-grid) scheme and a multiple-grid method that is used to couple the various levels of embedded grids as well as to accelerate convergence of the fine-grid scheme.

Governing Equations

The prediction of inviscid flows in general domains can be performed through the time integration of the time-dependent Euler equations. These equations for three-dimensional flow can be written in conservative form as

$$\frac{\partial U}{\partial t} + \frac{\partial (F)}{\partial x} + \frac{\partial (G)}{\partial y} + \frac{\partial (K)}{\partial z} = S \tag{1}$$

where

$$U = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ E \end{bmatrix} \qquad F = \begin{bmatrix} \rho u \\ \rho u^2 + P \\ \rho uv \\ \rho uw \\ \rho uH \end{bmatrix} \tag{2}$$

$$G = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + P \\ \rho vw \\ \rho vH \end{bmatrix} \qquad K = \begin{bmatrix} \rho w \\ \rho uw \\ \rho vw \\ \rho w^2 + P \\ \rho wH \end{bmatrix} \tag{3}$$

where ρ is the density; u, v, and w are the velocity components in the x, y, and z directions, respectively; E is the total energy; P is the static pressure; H is the stagnation enthalpy; and S is a vector containing any source term components such as those contributing to body forces.

Fine-Grid Integration Procedure

An explicit, Lax-Wendroff (Taylor-Galerkin), time-marching procedure as developed by Ni[13] is used to iteratively solve Eq. (1) according to the Taylor series approximation:

$$U^{t+\Delta t} = U^t + \frac{\partial U}{\partial t}\Delta t + \frac{\partial^2 U}{\partial t^2}\frac{\Delta t^2}{2} \tag{4}$$

where t is time, and Δt is the time step. Local time steps are used in the current procedure to accelerate convergence to steady state.

The second term on the right-hand side of Eq. (4), the first-order time rate change, is obtained through an integration of Eq. (1) over primary control volumes constructed from a grid meshing of the computational domain (see Fig. 2). The independent variables U are located at the intersection of the grid mesh lines (nodes). The first-order time rate change at the grid nodes is obtained from an averaging of the adjacent primary control volume values. This step is considered the first part of a transport operator T_Δ:

$$T_\Delta = \delta_C^N + \delta_C^{2N} \tag{5}$$

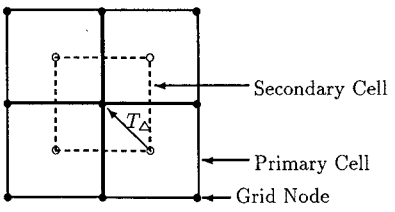

Fig. 2 Flow integration scheme.

The third term on the right-hand side of Eq. (4), the second-order time rate change, could be obtained directly at the grid nodes from an integration of the time derivative of Eq. (1) over a secondary control volume that is constructed by connecting the centroids of the adjacent primary control volumes (Fig. 2). Instead of using this approach, however, the second-order time rate change is calculated from each primary control volume. This piecewise contribution to the second-order time rate change makes up the remainder of the transport operator [i.e., the second term on the right-hand side of Eq. (5)]. Thus, the overall integration of Eq. (1) is performed completely as a cell-based operation in which the first-order time rate changes are computed and then transported along with the piecewise contribution of the second-order changes to the grid nodes.

To capture shocks and eliminate oscillations in the flowfield in regions of coarse grid spacing, a numerical smoothing model that consists of a combination of second and fourth difference operators,[14] is added to the right-hand side of the numerical approximation of the Euler equations. Second differences of the primary variables are obtained by collecting the first differences along the grid lines on each side of a given node. Fourth differences are calculated by taking two consecutive second differences.

Multiple-Grid Procedure

A multiple-grid acceleration technique[13] is used to couple the various levels of embedded grids and accelerate convergence of the fine-grid integration scheme. The basis of the multiple-grid approach is to accelerate the propagation of disturbance waves through the domain, especially through regions of embedded grids, by numerical treatment of the governing equations on a sequence of coarser grids. Thus, this approach can be used to maintain wave propagation speeds through disparate regions of grid spacing, such as arise at the interface between chunks that have and have not been subdivided. In addition, this scheme can be used to accelerate convergence of the basic fine-grid integration procedure.

The present multiple-grid acceleration scheme can be implemented by either directly using the hierarchical chunk data structure or using the children-chunk data structure by successively skipping every other grid node. The former of these implementations is denoted as the interchunk multiple-grid scheme. Interchunk multiple-grid is used whenever the data structure can support its use. It is used exclusively when a chunk data structure can be defined more coarse than the initial grid and for parents of chunks that have been divided in all three directions. The second implementation, denoted as the intrachunk multiple-grid approach, allows for $(N-1)/2$ levels of multiple grid to be performed inside a chunk where, again, N is the prescribed edge length of a chunk. Intrachunk multiple grid is used to allow for multiple-grid acceleration coarser than the initial grid when an initial grid parent data structure is not defined or is not possible due to the dimensions of the initial grid. This implementation is also used for parent chunks that have been directionally subdivided, in those directions that have not been divided. For this case, the parent's spacing is identical to the child's in those directions.

Embedded Edge Treatment

When subdivision of a chunk results in a grid spacing (in computational space) that differs by a factor of 2 from that of a particular neighbor, then an embedded edge results. Face nodes that exist as part of a chunk but do not exist as part of its neighbor are denoted as hanging nodes. Hanging nodes arise when the number of divisions of a chunk face is greater than those in its neighbor. Hanging nodes, however, do not arise when the grid spacing normal to a chunk face differs by a factor of 2 from its neighbor.

An interface zone is created between the embedded edge face nodes and the interior nodes of a chunk. Special discretization treatment is given to the nodes contained in the interface zone to insure conservation in fluxes between all cells

and to account for the change in grid spacing. This treatment is similar to that used in the two-dimensional procedure[11] except that its execution is performed entirely within the chunk to avoid additional data structure storage to define the embedded edge type. This internal-chunk implementation of embedded edge logic is essential for the application of this adaptive-grid approach on parallel computers.

Hierarchical Levels

To capture the essence of the "treelike" structure of the multiple-grid algorithm, the chunk-based information in this procedure has been organized in a hierarchical manner. This data structure consists of chunk levels that are used to delineate the parent-child relationship. In this approach, parent chunks are located at one level lower than their fine-grid chunks in the data structure.[11,15,16] Combinations of fine-grid and parent chunks can therefore exist at any given level except at the highest level where only fine-grid chunks exist and at levels coarser than the initial grid where only parent chunks exist.

Chunks are processed in the flow solution algorithm starting at the highest level descending to the lowest level. At any given level, fine-grid chunks are treated using the fine-grid integration scheme. Simultaneously, parent (coarse) chunks at that level are treated using the multiple-grid algorithm. Once all of the chunks at a given level have been processed, boundary conditions are applied at the domain boundary nodes belonging to chunks at that level, and the primary variables are updated.

Parallel Computing

Several domain decomposition techniques are used in the present procedure to enable chunks to be processed in parallel.[16] In addition to hierarchical levels as previously described, chunk coloring and categorization are used to avoid data dependencies and to reduce the amount of complex logic. These techniques are discussed in Refs. 15 and 16. Processing of the chunks in the flow integration scheme are performed by first gathering the chunk information from global to local memory, performing all mathematical operations such as the flux integration, multiple-grid acceleration, or smoothing on the local arrays, and then scattering the results back to global arrays. These intrachunk operations are all performed with vector processing when possible. With this approach, the penalty associated with gather/scatter and indirect addressing is greatly reduced.

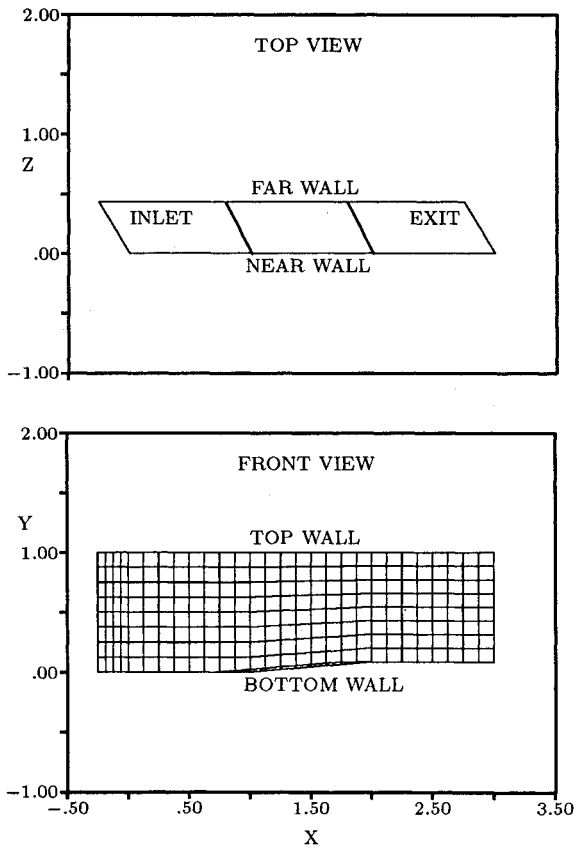

Fig. 3 Domain and initial grid for swept supersonic inlet (10-deg ramp and 30-deg sweep).

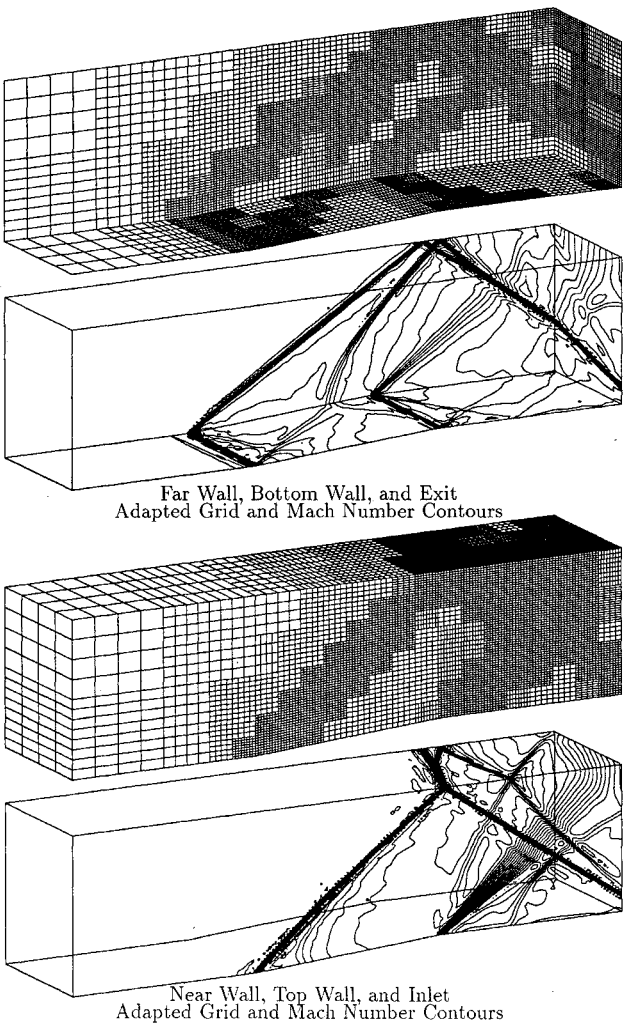

Fig. 4 Directionally adapted solution for swept supersonic inlet (2758 chunks, 165,300 nodes, 176,500 cells, and 1570 patches).

Results

Computed results for a generic supersonic inlet and the ONERA M6 wing are shown to demonstrate the capability of the present procedure to locate high-gradient regions and subdivide the grid normal to those gradients. In addition, results are shown that demonstrate the efficiency of the present procedure. As previously discussed, the chunk edge size was chosen to be 5 for these cases. This resulted in intrachunk node vector lengths of 125 and intrachunk cell vector lengths of 64, which are both nearly optimal for vector computing. Two levels of multiple grid more coarse than the initial grid were also used in both solutions. These calculations were executed on an Alliant FX2800 parallel/vector computer that had six processors linked in concurrency. The calculational speed of this computer as a function of the number of processors executed in parallel for this adaptive-grid procedure is described in Ref. 16.

Supersonic Inlet

Figure 3 shows the $25 \times 9 \times 5$ initial grid for a generic supersonic inlet that is modeled using a duct that has a 10-deg

ramp and is swept 30 deg. By using a tangency boundary condition at the near and far walls, symmetry is imposed in the ramp geometry in the z-direction, creating a "sawtoothed" ramp leading edge. Similarly, a tangency condition enforced at the lower and upper walls is used to model a converging nozzle with twice the height of the domain shown in Fig. 3. The inlet Mach number for this case was set to 1.73 to generate a complex three-dimensional oblique shock/expansion system. The computational grid was adapted to the solution three times.

The computational grid and Mach number contours after three adaptations are shown in Fig. 4. In this case, the dominant flow features all run at oblique angles relative to each of the primary directions, which results in the majority of the grid subdivision being performed in two or all three directions. A comparison of the computational grid and Mach number contours resulting from the third adaptation demonstrates that the shocks in the flowfield have been properly located and the computational grid has been embedded in these regions. Shock planes are generated at the onset of the 10-deg ramp (ramp shock) and at the apex of the wedge (wedge shock) located at the intersection of the near wall and the ramp. An expansion zone is generated by the end of the ramp where the flow returns to axial. The wedge shock extends in the z-direction from the near wall to the far wall at nearly 45 deg relative to the freestream where it runs nearly parallel to the ramp shock along the far wall. This shock continues to reflect back to the near wall and then again to the far wall downstream of the ramp. The wedge and ramp shocks merge midway between the near and far walls at the upper boundary and on the far wall downstream of the ramp. This complex flow solution demonstrates the ability of the current procedure to detect directional flow gradients and properly refine where necessary. In addition, this solution demonstrates the robustness of the flow solution procedure for complex directional embedded grids.

A uniform fine-grid case was executed for comparison with the adaptive-grid solution. A 201 × 73 × 41 mesh was used for this calculation, which corresponded to the finest grid spacing in the adaptive-grid solution. This grid resulted in 7020 chunks and 3400 patches. Comparison between Figs. 4 and 5 shows that the solutions are essentially the same. The adaptive-grid solution, however, took 30% fewer iterations, 6 times less CPU, and 2.5 times less memory than the uniform fine-grid solution.

The convergence history for the adaptive-grid solution is shown in Fig. 6. This figure shows that the rate of convergence is very similar for each level of adaptation and that convergence is reached in 300–1200 iterations for each grid. This convergence behavior is made possible in the current procedure through the use of the multiple-grid acceleration scheme over all of the embedded grids.

ONERA M6 Wing

The flow over the ONERA M6 wing at a freestream Mach number of 0.84 at an angle of attack of 3.06 deg[17] has also

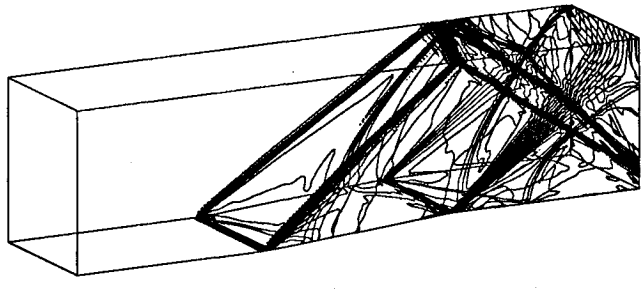

Fig. 5 Uniform fine-grid solution for swept supersonic inlet.

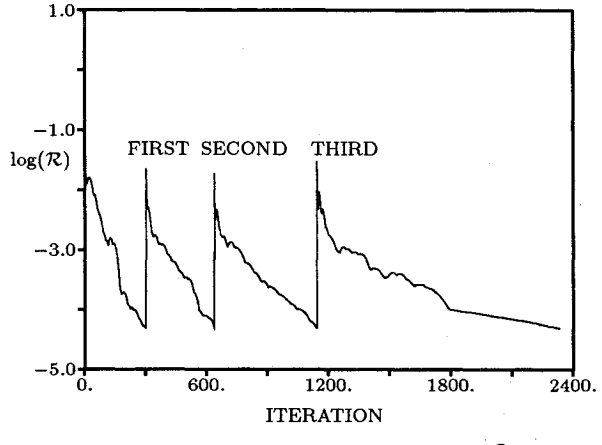

Fig. 6 Convergence history for swept supersonic inlet, $\mathcal{R} = (\Delta\rho V)_{\max}$.

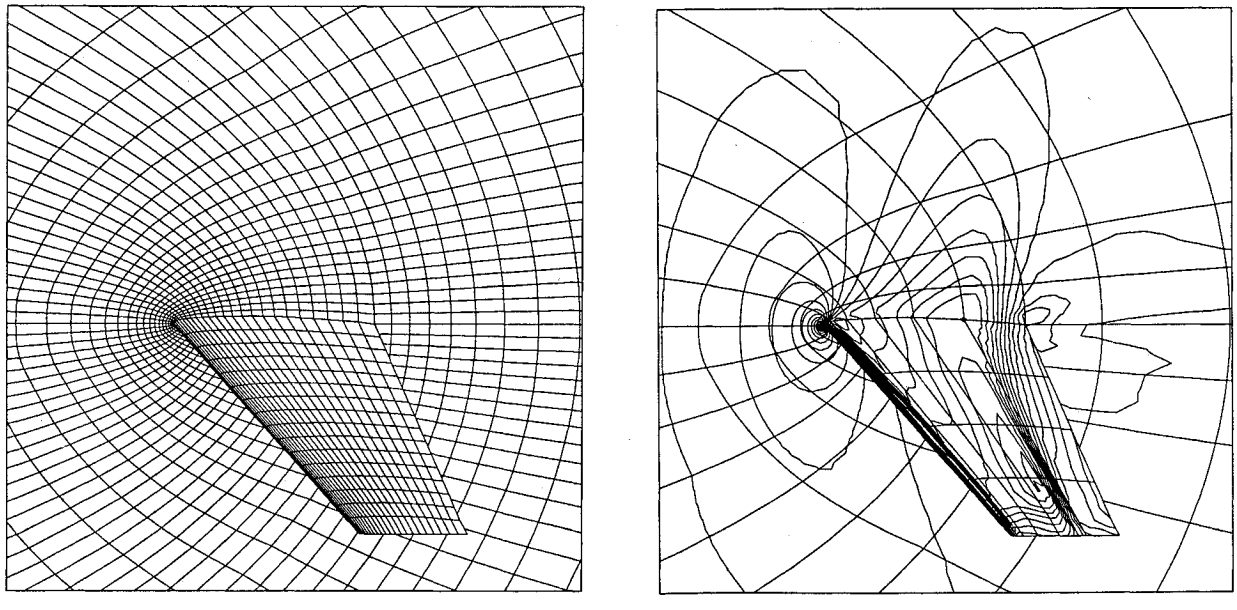

Fig. 7 Initial grid and Mach number contours (with chunk boundaries) for ONERA M6 wing, $M_\infty = 0.84$ and $\alpha = 3.06$ deg (650 chunks, 45,600 nodes, 41,500 cells, and 500 patches).

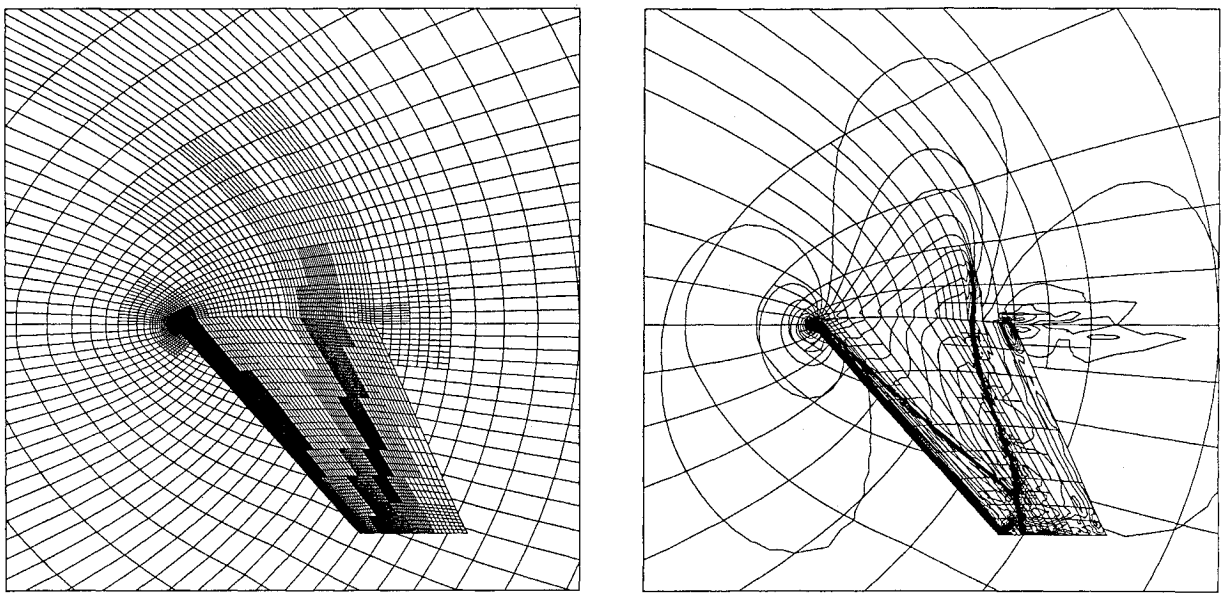

Fig. 8 Adapted grid and Mach number contours (with chunk boundaries) for ONERA M6 wing, $M_\infty = 0.84$ and $\alpha = 3.06$ deg (3900 chunks, 203,800 nodes, 250,100 cells, and 1643 patches).

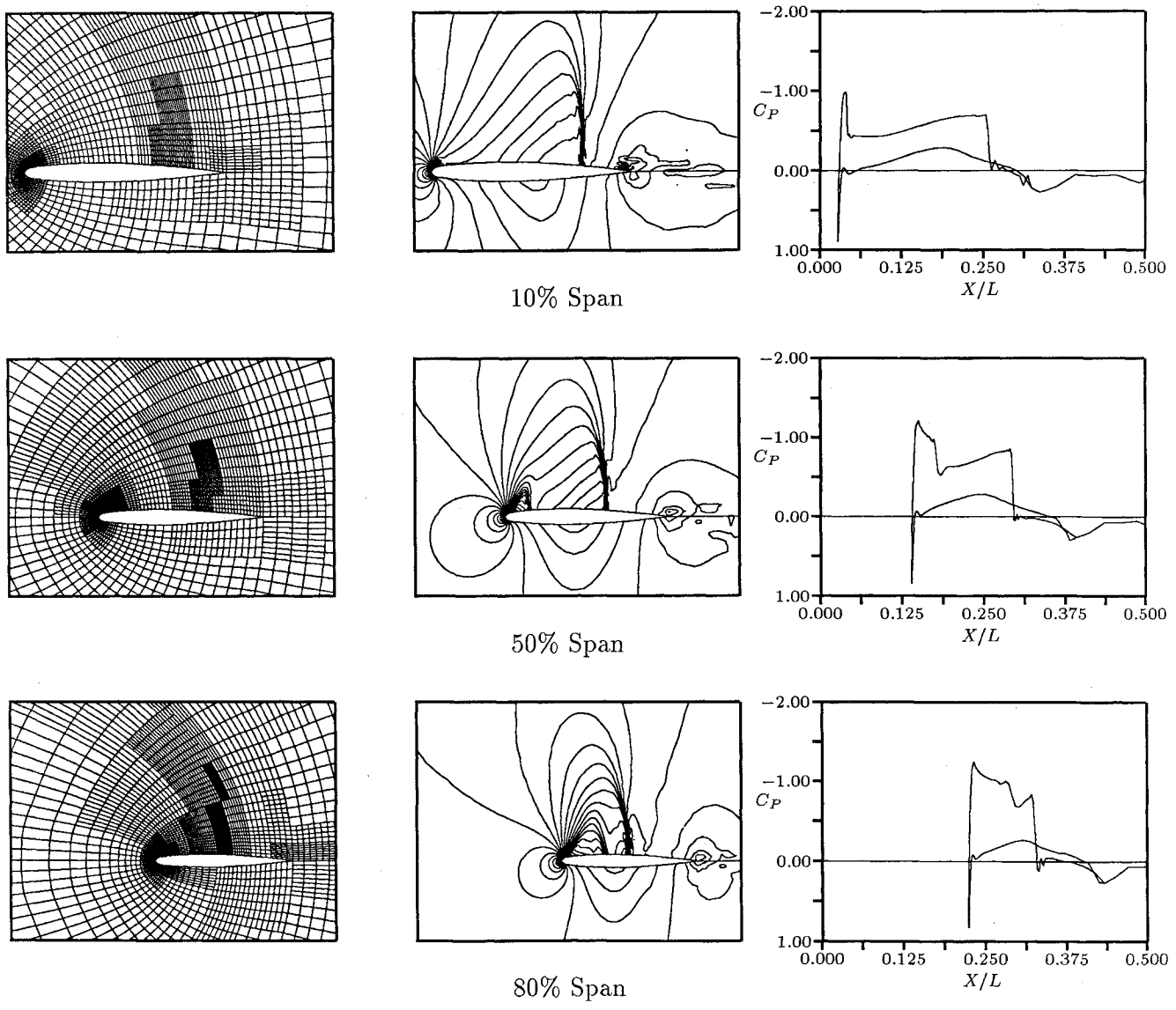

Fig. 9 Adapted grid, Mach number contours, and pressure coefficient distributions at various spanwise locations for ONERA M6 wing, $M_\infty = 0.84$ and $\alpha = 3.06$ deg (3900 chunks, 203,800 nodes, 250,100 cells, and 1643 patches).

Table 2 Lift coefficient for each grid adaptation for ONERA M6 Wing: $M_\infty = 0.84$ and $\alpha = 3.06$ deg

Adapt level	No. of nodes	No. of cells	Lift coefficient
0	45,600	41,500	0.2545
1	67,500	67,200	0.3111
2	105,900	118,000	0.3067
3	203,800	250,100	0.3107

been computed with the present adaptive-grid procedure. This case was chosen because of its well-known upper surface double-shock system. The initial grid for this case, shown in Fig. 7, was generated using the FLO57 Euler procedure.[14] This grid consists of a $73 \times 25 \times 25$ C-grid that is wrapped around the airfoil surface. Figure 7 also shows the predicted Mach number contours on the upper surface of the airfoil corresponding to the initial grid. These contours illustrate that two shocks exist on the upper surface near the leading and trailing edges. The definition of the suction side shocks is poor, however, due to the coarseness of the computational grid.

Figure 8 demonstrates the adapted computational grid and Mach number contours after three adaptations to the flow. For this case, in which the shocks are aligned to the spanwise grid lines, the majority of grid subdivision occurs in the streamwise direction. Comparison of the Mach number contours between this figure and those of Fig. 7 shows the increase in shock resolution brought about by the grid adaptation. Figure 8 also shows that the grid-refinement algorithm of the current procedure has properly detected the gradient directions and has subdivided the grid in the directions normal to the shock. A small region of grid refinement exists aft of the rear shock at the root of the airfoil near the trailing edge corresponding to a jump in the static pressure. This static pressure jump, which has not been observed in previous predictions, is probably due to the Kutta condition used at the trailing edge for this calculation.

The adapted grid, Mach number contours, and pressure coefficient are shown in Fig. 9 for three different spanwise locations.[17] The fine resolution in the shocks is evident by the tightly clustered Mach number contours and the sharp discontinuity in the pressure distributions. The suction side two-shock system that was shown in Fig. 8 is also illustrated in these cross-plane contours and pressure distributions. The computational grid has been subdivided in the streamwise direction at both suction side shocks. In addition, the grid is subdivided in both the streamwise and blade surface-normal directions at the airfoil leading-edge stagnation point and at the base of each suction side shock. Comparisons between the computational grids and the Mach number contours in this figure as well as in Fig. 8 show that the contours pass smoothly through the various embedded grids. This demonstrates the ability of the current procedure to detect high-gradient regions and directionally subdivide the local grid without producing errors at embedded-grid interfaces.

The lift coefficient is listed for each level of adaptation in Table 2. The lift coefficient converges to approximately 0.310, which is greater than the average computed value[17] of 0.285. The reason that the predicted lift is larger than the average computed value is probably due to the manner in which the trailing edge and tip were modeled in the present calculation and the near proximity of the far-field boundary to the airfoil surface.

One thing to note in Table 2 is that the number of cells for each level is nearly the same as the number of nodes, which indicates that the amount of work required for the flow integration calculations in the current procedure is kept at a minimum. A uniform fine-grid calculation with grid spacing equivalent to that after the third adaptation would require over 100 times the CPU and memory of that required for the adaptive-grid solution. Again, this illustrates the computational savings possible from the use of an adaptive-grid procedure such as the one presented here.

Summary

A new three-dimensional adaptive grid-embedding Euler procedure is presented. This procedure features a tunable, semistructured data system for parallel/vector computing, directional grid-embedding adaptation for efficient resolution of high-gradient flow features, and application of multiple-grid convergence acceleration over all embedded grid regions as well as on grids coarser than the initial grid. Microblocks, called chunks, are used as the fundamental computing unit with interchunk connectivity being defined via an unstructured-grid data system and intrachunk flow integration computations being performed using efficient structured grid operations. Pressure and velocity difference grid refinement parameter vectors are used to identify regions of high gradients in the solution. Chunks in the vicinity of high-refinement parameters are directionally divided, and the flow solution procedure is reinitiated. Rapid convergence to steady state is achieved through the application of a multiple-grid convergence acceleration scheme using either the coarse-parent or fine-grid children chunk data structure. Example solutions of internal and external flows are presented to demonstrate solution accuracy and efficiency. As compared with uniform fine-grid solutions, the current procedure requires 3–100 times less computing time and memory for comparable accuracy depending on the complexity of the flowfield.

Acknowledgments

This work was supported by the United Technologies Corporation under the Corporate Sponsored Research Program. The authors wish to thank Dan Dorney from the United Technologies Research Center for his work in grid generation and Bob Haimes from the Massachusetts Institute of Technology for supplying the VISUAL3 code that was used in the visualization of the numerical results.

References

[1]Lohner, R., and Baum, J., "Adaptive *h*-Refinement on 3D Unstructured Grids for Transient Problems," *International Journal for Numerical Methods in Fluids,* Vol. 14, 1992, pp. 1407–1419.

[2]Dawes, W., "The Development of a Solution-Adaptive 3D Navier-Stokes Solver for Turbomachinery," AIAA Paper 91-2469, June 1991.

[3]Morgan, K., Peraire, J., and Peiro, J., "The Computation of Three-Dimensional Flows Using Unstructured Grids," *Computer Methods in Applied Mechanics and Engineering,* Vol. 87, 1991, pp. 335–352.

[4]Mavriplis, D. J., "Three Dimensional Unstructured Multigrid for the Euler Equations," AIAA Paper 91-1549-CP, June 1991.

[5]Ward, S., and Kallinderis, Y., "Hybrid Prismatic/Tetrahedral Grid Generation for Complex 3-D Geometries," AIAA Paper 93-0669, Jan. 1993.

[6]Aftosmis, M. J., "A Second-Order TVD Method for the Solution of the 3D Euler and Navier-Stokes Equations on Adaptively Refined Meshes," Thirteenth International Conf. on Numerical Methods in Fluid Dynamics, July 1992; also *Lecture Notes in Physics,* edited by M. Napolitano and F. Sabetta, Springer-Verlag, Heidelberg, Germany, 1993.

[7]Berger, M., "Adaptive Mesh Refinement on the Connection Machine," Workshop on Adaptive Methods for Partial Differential Equations, Rensselaer Polytechnic Inst., Troy, NY, May 1992.

[8]Oden, J. T., Rachowicz, W., and Kennon, S. R., "Numerical Analysis of Three-Dimensional Compressible Navier-Stokes Equations Using an Adaptive *h-p* Finite Element Method," AIAA Paper 91-0119, Jan. 1991.

[9]Melton, J. E., Thomas, S. D., and Cappuccio, G., "Unstructured Euler Flow Solutions Using Hexahedral Cell Refinement," AIAA Paper 91-0637, Jan. 1991.

[10]Schmidt, G. H., and Sturler, E., "Multigrid for Porus Medium Flow on Locally Refined Three Dimensional Grids," *Multigrid Methods: Special Topics and Applications II,* edited by W. Hackbusch and U. Trottenberg, GMD-studien Nr. 189, Gesellschaft fur Mathematik und Datenverarbeitung, Sankt Augustin, 1991 (ISBN 3-88457-189-3, ISSN 0170-8120).

[11]Dannenhoffer, J. F., "Grid Adaptation for Complex Two-Di-

mensional Transonic Flows," Sc.D. Thesis, Massachusetts Inst. of Technology, Dept. of Aeronautics and Astronautics, Cambridge, MA, Aug. 1987.

[12]Kallinderis, Y., and Baron, J. R., "Adaptation Methods for a New Navier-Stokes Algorithm," *AIAA Journal,* Vol. 27, No. 1, 1989, pp. 37–43.

[13]Ni, R. H., "A Multiple Grid Scheme for Solving the Euler Equations," *AIAA Journal,* Vol. 20, No. 11, 1982, pp. 1565–1571.

[14]Jameson, A., and Baker, T. J., "Solution of the Euler Equations for Complex Configurations," AIAA Paper 83-1929, July 1983.

[15]Dannenhoffer, J. F., and Davis, R. L, "Adaptive Grid Computations for Complex Flows: A Supercomputing Challenge," *Proceedings from the 4th International Conference on Supercomputing,* International Supercomputing Institute, 1989, pp. 206–215.

[16]Davis, R. L., and Dannenhoffer, J. F., "Decomposition and Parallelization Strategies for Adaptive Grid-Embedding Techniques," *Proceedings from the Sixth International Conference on Domain Decomposition Methods,* June 1992.

[17]Yoshihara, H., and Sacher, P., "Test Cases for Inviscid Flow Field Methods," AGARD Advisory Rept. 211, May 1985.