AIAA JOURNAL
Vol. 42, No. 9, September 2004

New Migratory Memory Algorithm for Implicit Finite Volume Solvers

Nikhil Shende* and N. Balakrishnan†
Indian Institute of Science, Bangalore 560 012, India

This work deals with the issues regarding the convergence acceleration and the memory management of unstructured data-based cell-center finite volume flow solvers. The convergence acceleration to the steady state is achieved using three implicit relaxation procedures cast in a matrix-free framework, namely, point Jacobi, symmetric Gauss–Seidel, and lower-upper-symmetric Gauss–Seidel. The formulation of the point Jacobi procedure presented is different from the symmetric Gauss–Seidel and the lower-upper-symmetric Gauss–Seidel procedures in the sense that the point Jacobi procedure is cast in terms of a face-based algorithm, whereas the other two are cell-based procedures. It is observed that the performances of all of the three implicit relaxation procedures are comparable. Inspired by the developments in the parallelization of unstructured data-based codes, we have presented a new memory-saving device called the *migratory memory algorithm* (MMA). It is shown that the MMA drastically reduces the memory requirement of the class of codes just mentioned. The MMA is presented in the context of the least-squares-based linear reconstruction and the point Jacobi procedures. Particularly in the case of computational-fluid-dynamics codes to be used in routine design cycle, the use of MMA can considerably increase the problem size a given machine can handle.

I. Introduction

A COMPUTATIONAL-FLUID-DYNAMICS (CFD) code used in industry for generating routine design data should have three primary features, as follows:

1) It should be robust, that is, it should be able to handle complex flow physics within the regime of validity of its mathematical model.

2) It should be able to handle geometric complications frequently arising in the industrial computations.

3) It should have a realistic run time for sufficiently large size problems on the present-day PCs, that is, it should provide the designers with the estimates of reliable design data in reasonable time.

CFD codes based on finite volume methodology have reached a high level of maturity in terms of each one of the aforesaid features. Finite volume codes used in conjunction with upwind schemes are robust and are ideally suited for industrial computations. The present-day unstructured finite volume algorithms not only permit the use of arbitaray polyhedral volumes but also ensure good solution quality on the resulting distorted meshes. Two important challenges arising out of the use of unstructured meshes are listed here:

1) Such codes when used in conjunction with an explicit time-integration procedure have poor convergence characteristics. This problem is accentuated on finer grids resulting from a grid-adaptation procedure.

2) The use of unstructured data makes the computations enormously memory intensive.

In the present work, we address these twin issues associated with unstructured mesh computations. Whereas the use of an implicit time-integration procedure is an obvious choice to overcome the stability-related problems resulting in poor convergence characteristics, the memory associated problems are overcome employing a novel algorithm called *migratory memory algorithm* (MMA).[1]

In the present work, the code HIFUN-3D (high-resolution flow solver on unstructured meshes),[2] employing a cell-center finite volume methodology admitting arbitary polyhedral volumes is utilized. It is well known that on a tessellated computational domain the use of cell-vertex finite volume framework can be considered to be superior to that of a cell-center finite volume framework for a variety of reasons including memory, cost, and ease with which viscous fluxes can be determined.[3] Also, the fact that boundary conditions can be satisfied more accurately in a cell-vertex framework with relative ease does not generally find a mention in the literature.[4] In spite of all of the advantages that accrue out of the use of a cell-vertex framework, we chose the cell-center formulation because of the singular advantage it offers in terms of the use of arbitrary polyhedral volumes. Such volumes can result as a consequence of an embedded mesh refinement with hanging nodes[5,6] or the use of a cartesian mesh resulting in trimmed hexahedral volumes in the vicinity of the body.[7] As a result, the code thus developed has enormous flexibility in terms of geometric complexity, grid, and accuracy, particularly when used in conjunction with adaptive refinement.

It is imperative that any such code making use of unstructured data, which are inherently memory intensive, should employ apart from convergence acceleration devices certain memory-saving techniques like MMA discussed in the paper. Such an algorithm can still be relevant in the present days of memory boom, in the sense that the problem size a given machine can handle can be many fold bigger than the case not employing such a device. It is brought out in this work that it is possible to achieve such a goal by making only certain marginal changes in a given serial code. Also, in the context of implicit time integration few attempts have been made in the past to study the relative performance of relaxation procedures such as symmetric Gauss–Seidel (SGS) and lower–upper-symmetric Gauss–Seidel (LU-SGS)[8] techniques in a matrix-free framework. Here, in addition to SGS and LU-SGS procedures, for the first time a matrix-free formulation of point Jacobi (PJ) procedure,[9] amenable to a face-based algorithm, is considered. In particular, we discuss the aforesaid implicit procedures in the framework of MMA. At this stage, it is worth mentioning that many of the popular relaxation procedures[10] used in the structured mesh computations cannot be used with unstructured data.

In Sec. II, different implicit relaxation schemes employed in this paper are introduced. In Sec. III, a brief description of HIFUN-3D code is presented as a prelude to the discussions on the MMA. In Sec. IV, the motivation behind the use of MMA and its details are presented. The results pertaining to the relative performance of

Received 28 October 2002; revision received 1 January 2004; accepted for publication 6 January 2004.

*Research Student, Computational Fluid Dynamics Center, Department of Aerospace Engineering.

†Assistant Professor, Computational Fluid Dynamics Center, Department of Aerospace Engineering; nbalak@aero.iisc.ernet.in.

different relaxation procedures, particularly in light of the use of MMA, are presented in Sec. V. Concluding remarks are made in Sec. VI.

II. Implicit Formulation

The Euler equation in conservation form can be written as

$$\frac{\partial U}{\partial t} + \nabla \cdot F = 0 \tag{1}$$

where U stands for the vector of conserved variables and F stands for the flux vector. Expressing this conservation equation in integral form over a finite volume, we arrive at the following space discretized equation:

$$\frac{\mathrm{d}U_i}{\mathrm{d}t} = -\frac{1}{\Omega_i}\sum_J F_{\perp J}\Delta S_J \tag{2}$$

In the preceding equation, i refers to the cell under consideration, j refers to a neighbor of cell i sharing an inteface J, U_i and Ω_i represent the cell averaged state and volume corresponding to the cell i, respectively. For a given interface J, the normal flux is represented by $F_{\perp J}$ and the area by ΔS_J. Introducing notations,

$$R_i = -\frac{1}{\Omega_i}\sum_J F_{\perp J}\Delta S_J \tag{3}$$

$$W = [U_1 \quad U_2 \quad \cdots \quad U_{nc}]^T$$

$$R = [R_1 \quad R_2 \quad \cdots \quad R_{nc}]^T$$

where nc represents the number of finite volumes in the computational domain, we have

$$\frac{\mathrm{d}W}{\mathrm{d}t} = R(W) \tag{4}$$

which represents a system of ordinary differential equations in time for all of the finite volumes in the computational domain. This system of equations can be solved using implicit time-stepping procedure, which leads to the following vector-matrix equation:

$$M\Delta_t W = R^n \tag{5}$$

with

$$\Delta_t(\cdot) = (\cdot)^{n+1} - (\cdot)^n$$

where n denotes time step. At this stage, it is worth mentioning that several attempts have been made in the past to solve the system given in Eq. (5), in the context of unstructured meshes.[11–13] In the present work, an upwind scheme using the following expression for the split normal flux Jacobian given by Yoon and Jameson[14] is made use of:

$$A^{\pm} = \frac{1}{2}[A \pm \rho_A I] \tag{6}$$

where ρ_A represents the spectral radius of the normal flux Jacobian A. Defining ϕ_i as a set of face-sharing neighbors of cell i, we have

$$\begin{aligned} M_{ij} &= \frac{1}{2\Omega_i}(A_j - \rho_{A_j} I)\Delta S_J \qquad \text{for} \qquad j \in \phi_i \\ &= 0 \qquad \text{for} \qquad j \notin \phi_i \\ M_{ii} &= \left[\frac{I}{\Delta t} + \frac{1}{2\Omega_i}\sum_J (A_i + \rho_{A_i} I)\Delta S_J\right] \end{aligned} \tag{7}$$

The matrix M is now decomposed in three parts such that $M = C + D + E$, where the diagonal matrix D carries the diagonal entires of M and the off-diagonal terms form the upper and lower triangular matrices C and E, respectively. With this decomposition, Eq. (5) can be written as follows:

$$(C + D + E)\Delta_t W = R^n \tag{8}$$

The actual implicit finite volume state update formula after introducing the definition given by Yoon and Jameson[14] for the split normal flux Jacobian reads

$$\left(\frac{I}{\Delta t} + \frac{I}{2\Omega_i}\sum_J \rho_{A_i}\Delta S_J\right)\Delta_t U_i + \frac{1}{2\Omega_i}\sum_J (\Delta_t F_{\perp j} - \rho_{A_j}\Delta_t U_j)\Delta S_J = R_i^n \tag{9}$$

where

$$R_i^n = -\frac{1}{\Omega_i}\sum_J F_{\perp}\left(U_i^n, U_j^n\right)\Delta S_J$$

The vector R_i^n can be computed using any suitable numerical flux formula. In Secs. II.A and II.B, we will follow a convention of using $\propto$ for the quantities at the current iteration in the relaxation procedure and $*$ for the quantities at the preceding iteration.

A. PJ Relaxation Procedure

In this relaxation procedure, Eq. (8) is solved as follows:

$$D\Delta_t W^{\propto} = R^n - (C + E)\Delta_t W^* \tag{10}$$

For point Jacobi iterations, Eq. (9) is written as

$$\left[\frac{I}{\Delta t} + \frac{I}{2\Omega_i}\sum_J \rho_{A_i}\Delta S_J\right]\Delta_t U_i^{\propto} = R_i^n - \frac{1}{2\Omega_i}\sum_J \left(\Delta_t F_{\perp j}^* - \rho_{A_j}\Delta_t U_j^*\right)\Delta S_J \tag{11}$$

In spite of the fact that preceding equation represents a system of algebraic equations, it is matrix free. This is because the matrix on the left-hand side is a diagonal matrix and the right-hand side does not involve any flux Jacobian. This obviates any matrix inversion to obtain the solution of this system of equations. This not only reduces the computational time, but also reduces the memory requirement considerably. The PJ procedure, unlike the other relaxation procedures described next, is amenable to a face-based algorithm. Therefore, the same data structure used for computation of residue R^n can be used in the case of PJ iterations also.

B. SGS Relaxation Procedure

In this relaxation procedure,[8] Eq. (8) is solved as follows.

Forward sweep:

$$(E + D)\Delta_t W^{\propto} = R^n - C\Delta_t W^*$$

Reverse sweep:

$$(D + C)\Delta_t W^{\propto} = R^n - E\Delta_t W^* \tag{12}$$

The use of forward and reverse sweeps cancels the accumulated numerical errors of an iterative procedure. The equation solved in forward sweep reads as follows:

$$\left(\frac{I}{\Delta t} + \frac{I}{2\Omega_i}\sum_J \rho_{A_i}\Delta S_J\right)\Delta_t U_i^{\propto} = R_i^n - \frac{1}{2\Omega_i}\sum_{\substack{J \\ j<i}} \left(\Delta_t F_{\perp j}^{\propto} - \rho_{A_j}\Delta_t U_j^{\propto}\right)\Delta S_J - \frac{1}{2\Omega_i}\sum_{\substack{J \\ j>i}} \left(\Delta_t F_{\perp j}^* - \rho_{A_j}\Delta_t U_j^*\right)\Delta S_J \tag{13a}$$

In the reverse sweep, we solve

$$\left(\frac{I}{\Delta t}+\frac{I}{2\Omega_i}\sum_J \rho_{A_i}\Delta S_J\right)\Delta_t U_i^{\kappa}$$

$$= R_i^n - \frac{1}{2\Omega_i}\sum_{\substack{J\\ j<i}}\left(\Delta_t F_{\perp j}^* - \rho_{A_j}\Delta_t U_j^*\right)\Delta S_J$$

$$-\frac{1}{2\Omega_i}\sum_{\substack{J\\ j>i}}\left(\Delta_t F_{\perp j}^{\kappa} - \rho_{A_j}\Delta_t U_j^{\kappa}\right)\Delta S_J \tag{13b}$$

In the preceding equations too, no matrix inversion is involved. The algorithm of SGS relaxation procedure is cell based. Hence the performance of this relaxation procedure is sensitive to the ordering of the finite volumes in the computational domain.

C. LU-SGS Relaxation Procedure

In this relaxation procedure, using an approximate lower upper decomposition of M (Refs. 13 and 15), we have

$$(E+D)D^{-1}(D+C)\Delta_t W = R^n \tag{14}$$

This results in the LU-SGS procedure described next.

Forward sweep:

$$(E+D)\Delta_t W^* = R^n$$

Reverse sweep:

$$(D+C)\Delta_t W = D\Delta_t W^* \tag{15}$$

The forward sweep for LU-SGS is cast as follows:

$$\Delta_t U_i^* = d_i^{-1}\left[R_i^n - \frac{1}{2\Omega_i}\sum_{\substack{J\\ j<i}}\left(\Delta_t F_{\perp j}^* - \rho_{A_j}\Delta_t U_j^*\right)\Delta S_J\right] \tag{16a}$$

and the reverse sweep is cast as follows:

$$\Delta_t U_i = \Delta_t U_i^* - d_i^{-1}\left[\frac{1}{2\Omega_i}\sum_{\substack{J\\ j>i}}(\Delta_t F_{\perp j} - \rho_{A_j}\Delta_t U_j)\Delta S_J\right] \tag{16b}$$

where

$$d_i = \left(\frac{1}{\Delta t}+\frac{1}{2\Omega_i}\sum_J \rho_{A_i}\Delta S_J\right)$$

Like previous relaxation procedures, in this procedure also, no matrix inversion is involved. The algorithm of LU-SGS relaxation procedure is also cell based. Thus the performance of this relaxation procedure is sensitive to the ordering of finite volumes in the computational domain.[16]

At this point, it should be remarked that in all of the three relaxation procedures just mentioned we employ an inconsistent linearization, where we make use of van Leer's split fluxes[17] for the explicit part and the split flux Jacobian of Yoon and Jameson[14] for the implicit part. Also, a simple Cut-Hill McKee type of reordering[18] is employed to make the SGS and LU-SGS procedures more effective.

III. HIFUN-3D

The code HIFUN-3D (Ref. 2) is a general purpose flow solver based on cell-centered finite volume methodology. The code is unstructured in terms of the data structure it employs, that is, it requires an explicit definition of face-based element connectivity. The basic mesh element that it can handle might be hexahedral, tetrahedral, prismatic, pyramidal, or a combination of the aforesaid elements. In general, the code is capable of handling arbitrary polyhedral volumes that can arise out of the use of embedded mesh refinement with hanging nodes.[6] This flexibility is achieved by the use of a face-based algorithm for the flux computations. The code uses a least-squares-based linear reconstruction procedure[19–21] with Venkatakrishnan limiter.[22] On a regular grid, this procedure is second-order accurate. The linear reconstruction procedure for computing the solution gradients involves the inversion of a geometric matrix. Though this matrix, in general, is nonsingular, it can pose serious computational difficulties in terms of solution convergence to steady state, if not properly conditioned.[20,23] This problem can be avoided by the use of a diagonal preconditioner in the framework of weighted least squares procedure. The diagonal preconditioning matrix is obtained by writing the inverse of the diagonal entries of the geometric matrix as diagonal entries. The real symmetric matrices (such as the geometric matrix involved in the least-squares procedure) are known to respond to such a preconditioning strategy.[24] The aforesaid geometric matrix is obtained by using a weighted least-squares procedure, where the weights are made proportional to the inverse of the square of the distance between two points under consideration. The idea has been successful in the context of inviscid flow calculations. Its extension to viscous flow calculation involving highly stretched grids, particularly in the context of a quadratic reconstruction procedure is currently being studied. The code supports a variety of numerical schemes for interfacial flux computation and a variety of boundary treatments. The convergence to steady state is accelerated using the implicit relaxation procedures such as PJ, SGS, and LU-SGS described in the preceding section.

IV. MMA

A. Motivation

Computational effort and memory are the two important issues involved in the development of a computer code. Often, their requirements are contradictory, and optimizing the code for both can be a great challenge to the developer. The MMA presented in this paper precisely addresses this question. This new algorithm aims at reducing the memory requirement of a serial version of the cell-center unstructured mesh codes without compromising on the computational effort by employing certain features of a parallel code. The main idea behind this algorithm is to divide the computational domain into a number of subdomains. The choice of the number of subdomains is user specified, and this should be generally based on the memory specifications associated with a given machine. This algorithm is based on the fact that the memory associated with a face-based solver like HIFUN-3D can be divided into two parts. The first one can be called the *basal memory* M_b and the other the *floating memory* M_f. The basal memory includes the grid data (geometric data, face-based connectivity for the flux computations, and the element-based connectivity for the reconstruction procedure) and very few flow-dependent arrays (conserved or primitive variables). The floating memory is associated with the flux computations and the implicit procedure. Although, the data associated with the basal memory are indispensable, those associated with the floating memory have only a local utility. In MMA, we exploit this feature, by storing the data associated with the basal memory for the entire computational domain and storing the data associated with the floating memory for a given subdomain. As can be easily seen, in the present case the maximum problem size a given machine can handle would be determined by M_b.

The importance of such an algorithm can be best realized when we look at the memory requirement associated with the reconstruction algorithm. In HIFUN-3D, storing the gradient information in all of the cells would be a memory-intensive implementation of this algorithm but requiring less computational effort. This requires 15 real numbers to be stored for every cell. The other way, which is less memory intensive, but computationally expensive, would be to store six elements of the symmetric geometric matrix involved in the least-squares-based linear reconstruction procedure. This would involve additional vector-matrix operations for determining the gradients, every time a face is accessed. The problem can be more intensive if a quadratic reconstruction[23,25] procedure is used for the viscous flow computations. The use of MMA in the gradient computations would involve storing the gradient information only for a given subdomain. The storage requirement, thus, becomes many times smaller, while avoiding additional effort in recomputing the gradients.

In the code HIFUN-3D, the basal memory involves the storage of three real numbers per node, 32 integers and 19 real numbers per cell, and seven integers and three real numbers per face. Let the number of nodes, cells, and faces in the computational domain be np, nc, and nf, respectively. Then, the total basal memory will involve the storage of $32nc + 7nf$ integer and $3np + 19nc + 3nf$ real numbers. Similarly, the floating memory involves the storage of 30 real numbers per cell. Let nc_m be the number of cells in the largest subdomain, in terms of number of cells. Then, the total floating memory involves the storage of $30nc_m$ real numbers. At this point, it should be mentioned that when the MMA is not incorporated nc_m will be equal to nc. A detailed description of various variables stored in HIFUN-3D is presented in the Appendix.

B. Algorithm

For a solution adaptive procedure employing an embedded mesh refinement with hanging nodes, the graph-based algorithms become the most obvious choice for partitioning the resultant grid. Here we make use of METIS,[26] a software based on multilevel graph partitioning for unstructured graphs. The first step involved is to convert the mesh into a dual graph, where each volume becomes a vertex of the graph and the volume interface becomes an edge. The partitioning objective is to minimize the number of edges that straddle partitions (often referred to as the edge cuts) subject to the constraint that each subdomain has approximately the same number of vertices. When the grid involves arbitrary polyhedral volumes, in the parallel computation terminology, load balancing would demand the use of a weighted graph, where weights are associated with the vertices and are proportional to the degree of the vertices. Whereas, in the present case, the fact that it is only memory that needs to be equitably distributed among subdomains permits the use of equal weights for all of the vertices. Subsequent to obtaining the partitioning information from the METIS, the vertices and the edges of the dual graph are renumbered.

Consider a graph $G = G(V, E)$, where V is the set of vertices and E is the set of edges. Let V_i and E_i represent the set of the vertices and the edges belonging to the subdomain i. Define D, a vector of size $|V|$ such that $D[u]$ represents the number of the subdomain that vertex u belongs to. Let E_b be a set of boundary edges, such that for an edge $e(u, v) \in E_b$, $D[u] \neq D[v]$. The MMA requires following constraints to be satisfied:

1)

$$\left.\begin{array}{ll} \text{a)} & V_i \cap V_j = \phi \\ \text{b)} & E_i \cap E_j = \phi \end{array}\right\} \qquad \text{for} \qquad i \neq j$$

2a) For any two vertices u and v, where $u \in V_i$ and $v \in V_j$, for $i < j$, necessarily $u < v$; b) for any two edges u and v, where $u \in E_i$ and $v \in E_j$, for $i < j$, necessarily $u < v$; and c) for an edge $e(u, v) \in E_b$, with $D[u] < D[v]$, necessarily $e(u, v) \in E_{D[u]}$.

The constraint 1a is automatically satisfied by the output from the METIS partitioner. Constraint 1b is satisfied as a consequence of the renumbering of the edges. Conditions 2a–2c are important for a code like HIFUN-3D employing a face-based procedure. Using a dummy array of size $|E_b|$, for storing face-based data computed in the preceding subdomains (storage of which involves five real numbers corresponding to the fluxes for a given face or six real numbers for point Jacobi based implicit procedure) repetitions in the computations are completely avoided. Therefore, the use of MMA involves an additional storage of $|E_b|$ integer and a maximum of $6|E_b|$ real numbers. But this additional memory requirement of MMA is very small compared to the memory saving it offers.

Another important aspect of MMA is that the spatial locality inherently exhibited by the algorithm effectively enhances the cache utilization.

1. Explicit Update Algorithm

Description of MMA as implemented for an explicit update procedure is given here:

1) Loop over the cells belonging to a given subdomain; compute and store the gradients of the state variables.

2) Loop over the faces in the subdomain, compute the interfacial flux, and pass this flux information to the finite volumes sharing the face. If the face belongs to E_b, perform the reconstruction in the finite volume, which belongs to the neighboring subdomain, calculate the fluxes, and store this information in the dummy array for future use.

3) Check for the flux data pertaining to the current subdomain in the dummy array, and pass on this information to the appropriate volumes.

4) Perform state update of all of the cells belonging to the subdomain.

5) Move to the next subdomain.

2. Point Jacobi Relaxation Procedure

The MMA as applied to PJ update for a given subdomain, reads as follows:

1) Loop over the faces. Compute the terms appearing within the parenthesis on the left-hand side and the terms appearing in the summation on the right-hand side of Eq. (11) for the PJ update, and pass on the information to both the finite volumes sharing the face. If the face belongs to E_b, store the information in the dummy array for future use.

2) Check for the implicit data pertaining to the current subdomain in the dummy array, and pass on this information to the appropriate volumes.

3) Perform the state update for a given PJ iteration for all of the cells belonging to the subdomain.

4) Proceed to the next subdomain.

V. Results and Discussion

Convergence acceleration achieved by an implicit relaxation procedure depends on two important factors, namely, the number of subiterations and the cost of each subiteration in terms of computational time. The more the number of subiterations, the faster will be the convergence to steady state in terms of total number of iterations, while the cost of each subiteration decides the cost of the implicit procedure. Therefore, an increase in the number of subiterations does not necessarily mean a faster convergence to steady state. One of the major advantages of using the split flux Jacobian of Yoon and Jameson[14] in the implicit relaxation procedures is that the resulting matrix-free subiterations are extremely cheap. This permits us to make use of many subiterations (say, of the order of 10) in the relaxation procedure. In this context, it is worthwhile to remark that the cost per subiteration of the point Jacobi procedure amenable to a face-based algorithm is less compared to that of a SGS procedure. Also, it can be expected that the convergence characteristics (in terms of number of iterations) of SGS and PJ procedures should be identical when large number of subiterations are used. For both the point Jacobi and symmetric Gauss–Seidel procedures, it is observed that a performance close to the best is achieved for about 12 subiterations for the class of problems considered in this paper. This can be used as a general guiding principle for choosing the number of subiterations.

The total memory M_t (in bytes) required for the code HIFUN-3D as described in Sec. IV, assuming that the storage of an integer number takes four bytes and that of a real number takes eight bytes, is given by the following formulas:

$$M_b = (32nc + 7nf) \times 4 + (3np + 19nc + 3nf) \times 8$$

$$M_f = 30nc_m \times 8$$

$$M_a = |E_b| \times 4 + 6|E_b| \times 8$$

$$M_t = M_b + M_f + M_a$$

where M_b, M_f, and M_a stand for the basal memory, the floating memory, and the additional memory (required for MMA only), respectively.

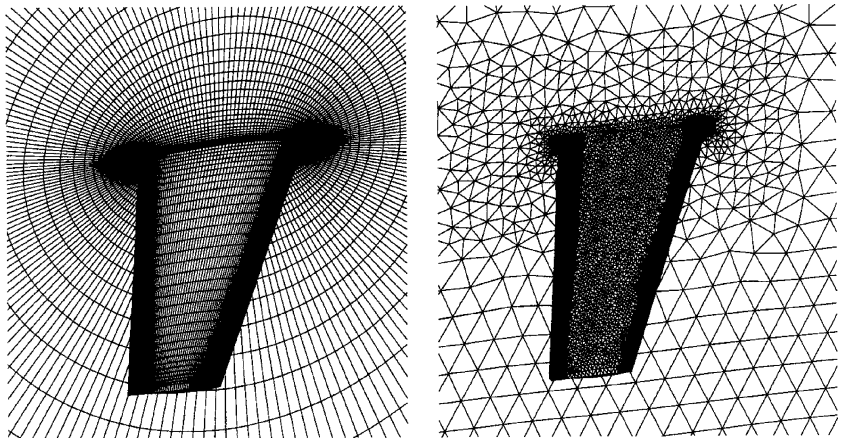
In this paper, the capability of MMA is demonstrated by considering two test cases. In both these test cases, a weighted

Table 1 Grid details

Case	Number of points np	Number of cells nc	Number of faces nf	Number of subdomains	Maximum number of cells in a subdomain nc_m	Number of faces on subdomain interface $\|E_b\|$
Grid 1: unpartitioned	418,200	400,000	1,216,400	1	400,000	——
Grid 1: partitioned	418,200	400,000	1,216,400	32	13,000	55,152
Grid 2: unpartitioned	223,383	1,184,118	2,405,853	1	1,184,118	——
Grid 2: partitioned	223,383	1,184,118	2,405,853	200	6,100	91,953
Grid 3: unpartitioned	208,230	1,100,949	2,249,884	1	1,100,949	——
Grid 3: partitioned	208,230	1,100,949	2,249,884	200	5,669	81,381

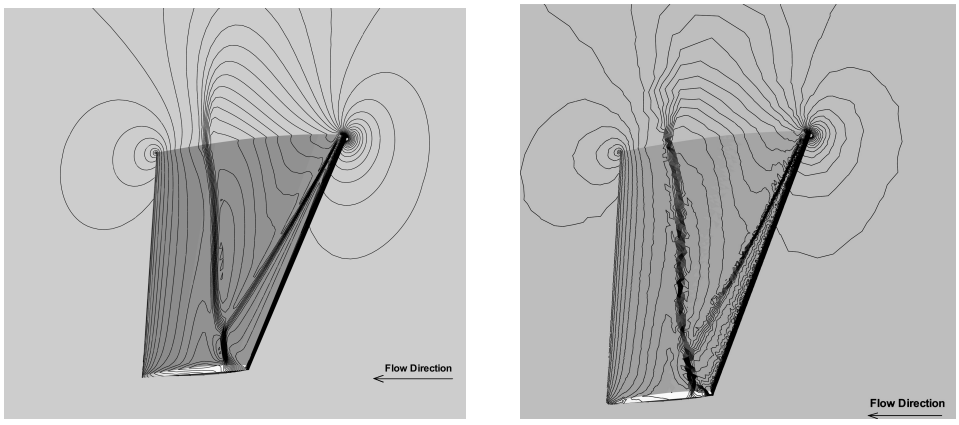
Table 2 Memory requirement of HIFUND-3D with and without MMA

Case	Basal memory M_b, MB	Floating memory M_f, MB	Additional memory M_a, MB	Total memory M_t, MB	% Reduction in total memory
Grid 1: without MMA	185.29	96	——	281.29	——
Grid-1: with MMA	185.29	3.12	2.87	191.28	32.00
Grid-2: without MMA	462.02	284.19	——	746.21	——
Grid-2: with MMA	462.02	1.46	4.78	468.26	37.25
Grid-3: without MMA	430.26	264.23	——	694.49	——
Grid-3: with MMA	430.26	1.36	4.23	435.85	37.24



a) Grid 1 **b) Grid 2**

Fig. 1 Surface grid on ONERA M6 wing and symmetry plane.



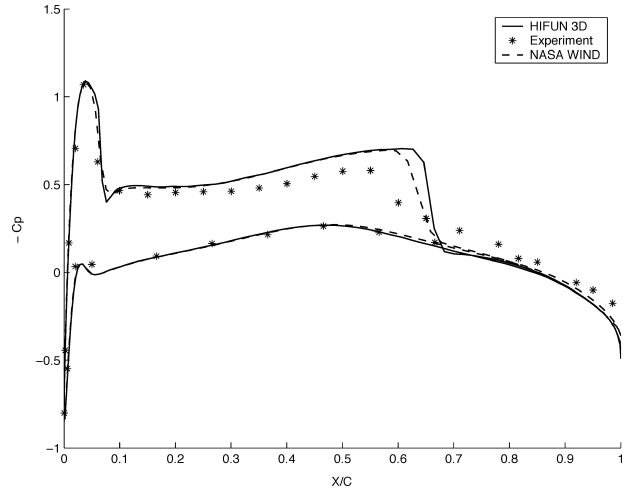
a) Grid 1 **b) Grid 2**

Fig. 2 Pressure contours on wing and symmetry plane ($p$min: 0.46, Δp: 0.026, $p$max: 1.48).

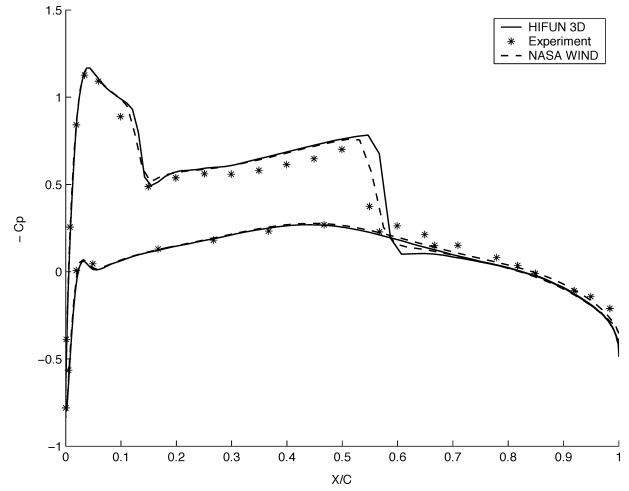
least-squares-based linear reconstruction procedure is employed. Solution is declared to have converged to steady state when residue defined based on density falls by six decades.

A. Test Case 1: ONERA M6 Wing

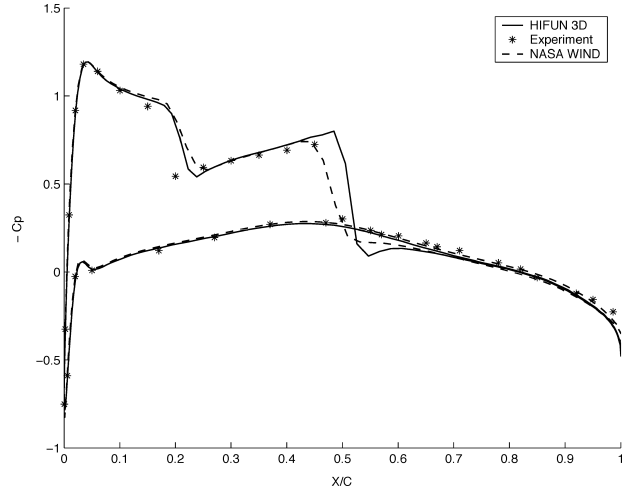
In this test case, transonic flow past ONERA M6 Wing with $M_\infty = 0.84$ and $\alpha = 3.06$ deg is solved using two grids, namely, grid 1 and grid 2. Whereas the grid 1 has hexahedral elements, grid 2 has tetrahedral elements. The surface grids on the wing and the symmetry plane are shown in Fig. 1. The grid details and partitioning details as provided by the METIS are presented in Table 1. The memory requirement for the two grids considered are presented in Table 2. Figure 2 shows the static-pressure contours on the wing surface along with minimum contour level, uniform increment between successive contour levels, and maximum contour level. The static pressure in present study, is nondimensionalized using freestream dynamic pressure $\rho_\infty |\boldsymbol{q}_\infty|^2$, where ρ_∞ and $\boldsymbol{q}_\infty$ are freestream density



a) 20% of wing span

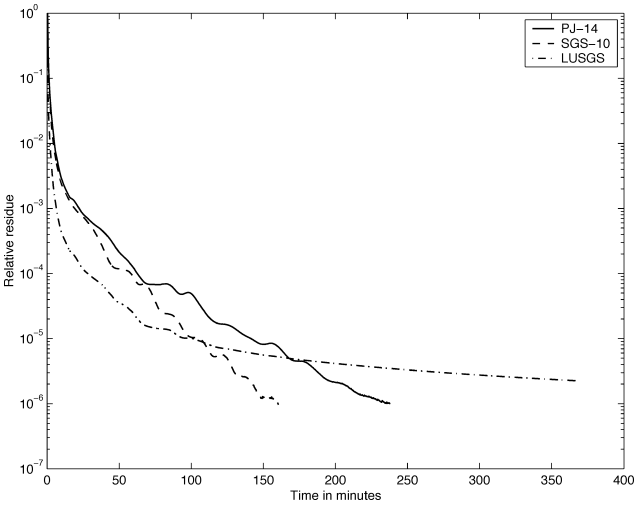


b) 44% of wing span

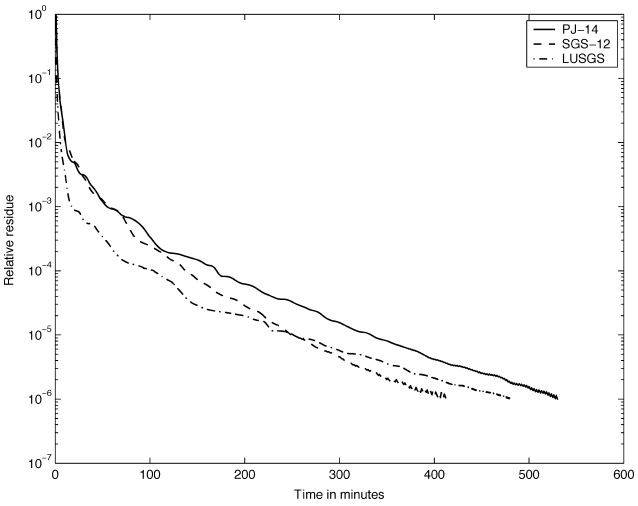


c) 65% of wing span

Fig. 3 Pressure distribution at different stations along the wing span obtained using grid 1.



a) Grid 1



b) Grid 2

Fig. 4 Comparison of convergence history of PJ, SGS, and LU-SGS relaxation procedures.

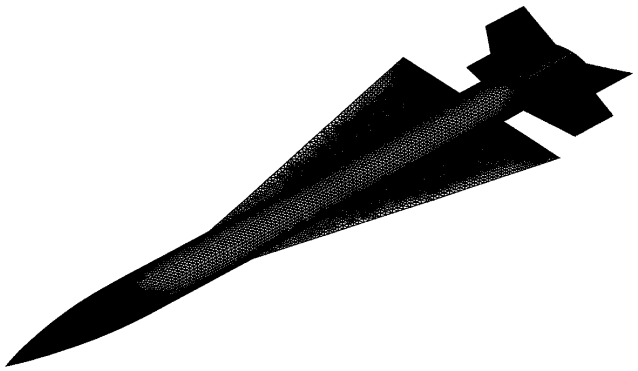


Fig. 5 Surface grid on missile.

and velocity, respectively. Figure 3 shows the Cp distribution at 20, 44, and 65% of the wing span compared with the experimental values and computational results obtained using the code NASA WIND.[‡] Figure 4 shows the relative performance of all of the three implicit relaxation procedures. This figure also indicates the number of subiterations used in the point Jacobi and symmetric Gauss–Seidel procedures corresponding to their best performance. For example, PJ-14 indicates that 14 subiterations are used in the point Jacobi procedure. For this test case, SGS relaxation procedure has best performance followed by LU-SGS and PJ relaxation procedures. The residue convergence for the LU-SGS relaxation procedure tends to saturate after five decades for grid 1.

B. Test Case 2: Monoplanar Missile Configuration

In this test case, a standard monoplanar missile with conventional-cruciform tail arrangement[27] is chosen. The supersonic flow at

[‡]Data available online at http://www.grc.nasa.gov/www/wind/valid/m6wing/m6wing01 [cited July 2002].

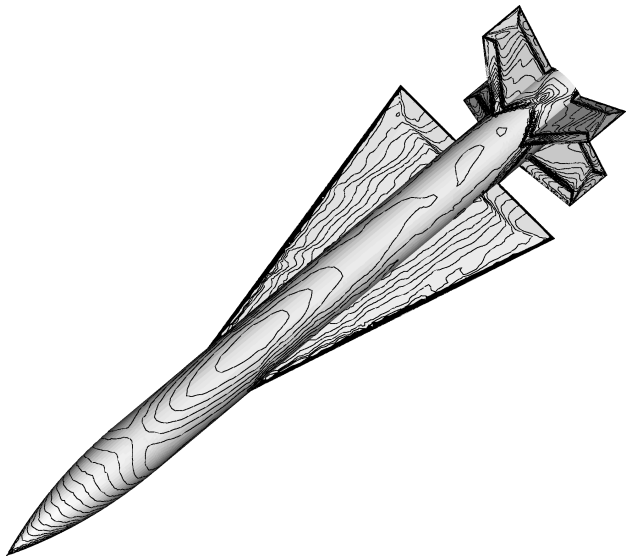


Fig. 6 Pressure contours: leeward side (*p*min: 0.06, Δp: 0.01, *p*max: 0.21).

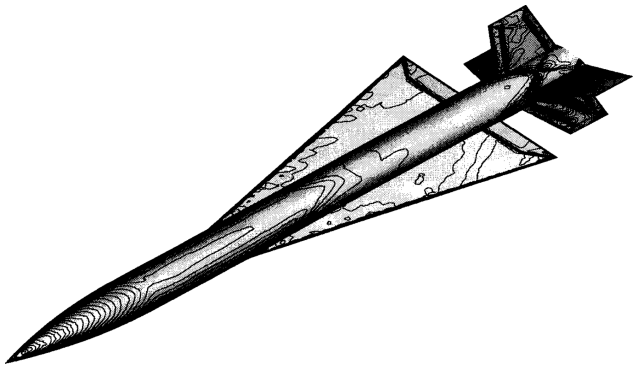
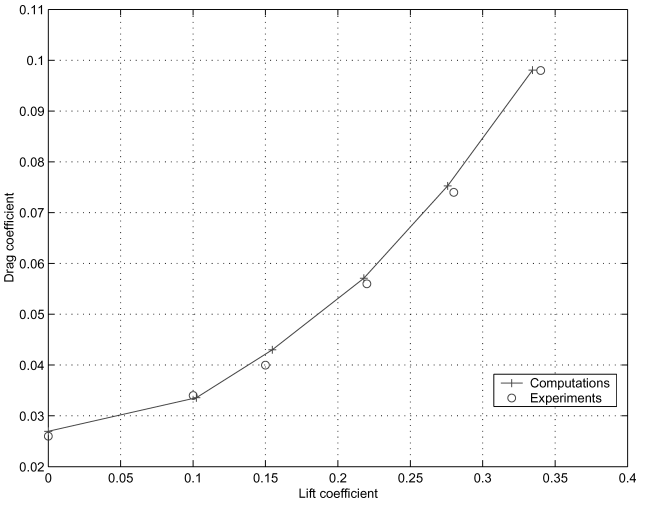
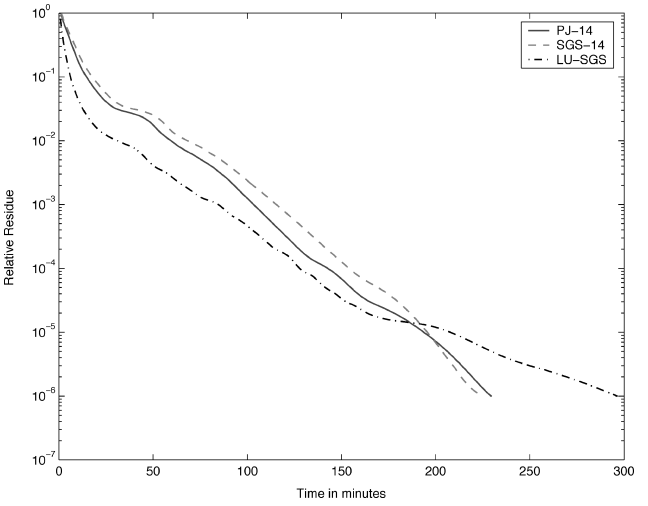


Fig. 7 Pressure contours: windward side (*p*min: 0.09, Δp: 0.01, *p*max: 0.35).



a) Lift–drag curve



b) Comparison of convergence history

Fig. 8 Drag polar and convergence history comparison for missile configuration.

$M_\infty = 2.16$ is solved at angles of attack ranging from 0 to 12 deg. The details of the tetrahedral grid, grid 3, along with the partitioning information provided by METIS are presented in Table 1. The memory requirement for the problem is presented in Table 2. Figure 5 shows the surface grid on body, wings, and tail of the missile. Whereas Fig. 6 shows the pressure contours on leeward side, Fig. 7 shows the pressure contours on windward side of the missile at an angle of attack of 8 deg. These figures also indicate the minimum contour level, uniform increment between successive levels, and maximum contour level. Figure 8a shows the comparison of the lift–drag curve obtained numerically with that obtained experimentally.[27] In supersonic flows over slender bodies at moderate angles of attack, wave drag forms a major component of total drag, while the contribution of the skin-friction drag is much smaller. Hence, even the inviscid codes can predict the drag in such cases fairly accurately. Figure 8b shows the relative performance of all of the three implicit relaxation procedures at an angle of attack of 8 deg. For this test case, all of the three relaxation procedures have a comparable performance. Here also, the LU-SGS procedure exhibits a tendency to saturate after five decades of residue fall. This feature of LU-SGS procedure has been consistently observed for many fine grid calculations.

From the data presented in Table 2, the substantial memory cut MMA offers becomes evident. In fact, the use of MMA can become all the more important in the context of the viscous codes, where quadratic reconstruction procedure[23,25] is essential. In this case, the gradients and Hessians of the state variables involve a storage of 45 real numbers per cell. In the case with quadratic reconstruction, estimated memory-saving MMA offers for three grids are 48.47, 54.41, and 54.39, respectively. These figures clearly bring out the importance of MMA in the viscous flow computations.

VI. Conclusions

In this paper, we have proposed a new migratory memory algorithm (MMA) for implicit finite volume solvers. The MMA is successfully implemented in the code HIFUN-3D, and we have demonstrated that it offers considerable memory saving. This is expected to substatially increase the size of a problem a given machine can handle and be of significance in case of codes used in routine design cycle. In the context of cell-center finite volume computations, PJ iterations compatible with face-based data, akin to flux computations, are compared with symmetric Gauss–Seidel (SGS) and lower-upper-symmetric Gauss–Seidel (LU-SGS) procedures that basically require cell-based data. Though the SGS iterations have shown to perform best, the convergence history of all three of the procedures presented are comparable to each other. The LU-SGS procedure shows a tendency of residue saturation on fine grids. The additional memory requirement for the point Jacobi (PJ) iterations is effectively handled using MMA. Also, it is expected that parallelizing the code based on PJ relaxation procedure should be easier and more cost effective.

Appendix: Variables Used in the HIFUN-3D Code

Table A1

	Basal memory		Floating memory	
Type	Real	Integer	Real	Integer
Node	Coordinates (3)	——	——	——
Cell	Values of the state variables in the previous and current time steps (10), R_i^n appearing in Eq. (3) (5), centroid (3), volume (1)	Support for reconstruction (26), faces forming cell to be used in SGS and LU-SGS iterations (6)	Gradients of the state variables (15), memory associated with limiting (15)	——
Face	Area normals (3)	Nodes constituting face (4), cells sharing face (2), boundary tag (1)	——	——

Acknowledgments

The authors thank Fluent India Private, Ltd., for their support in the use of GAMBIT for the purpose of grid generation. The authors also thank B. Theerthamalai, Scientist, Defence Research and Development Laboratory, Hyderabad, India, for referring us to test case 2, presented in this paper, and helping us in generating the surface model for the same.

References

[1]Shende, N., and Balakrishnan, N., "On Issues Regarding Convergence Acceleration and Memory Management of Unstructured Cell Centre Finite Volume Codes," *Proceedings of the Ninth Asian Congress of Fluid Mechanics*, edited by E. Shirani and A. Pishevar, Isfahan Univ. of Technology, Isfahan, Iran, 2002.

[2]Shende, N., and Balakrishnan, N., "HIFUN-3D-1: Users Manual," Dept. of Aerospace Engineering, Fluid Mechanics Rept. 2004 FM 10, Indian Inst. of Science, Bangalore, India, June 2004.

[3]Venkatakrishnan, V., "Perspective on Unstructured Grid Flow Solvers," *AIAA Journal*, Vol. 34, No. 3, 1996, pp. 533–547.

[4]Balakrishnan, N., and Fernandez, G., "Wall Boundary Conditions for Compressible Inviscid Flows on Unstructured Meshes," *International Journal for Numerical Methods in Fluids*, Vol. 28, 1998, pp. 1481–1501.

[5]Kallinderis, Y., "Grid Adaptation by Redistribution and Local Embedding," CFD Lecture Series: 1996-06, von Kármán Inst., Brussels, March 1996.

[6]Shende, N., Garg, U., Karthikeyan, D., and Balakrishnan, N., "An Embedded Grid Adaptation Strategy for Unstructured Data Based Finite Volume Computations," *Computational Fluid Dynamics 2002,* edited by F. Armfield, P. Morgan, and K. Srinivas, Springer, Berlin, 2002, pp. 94–99.

[7]Coirier, W. J., "An Adaptively-Refined Cartesian Cell Based Scheme for the Euler and Navier–Stokes Equations," Ph.D. Dissertation, Dept. of Aerospace Engineering, Univ. of Michigan, Ann Arbor, MI, 1994.

[8]Luo, Hong, Sharov, Dimtri, Baum, Joseph, D., and Lohner, Rainald, "A Class of Matrix-Free Implicit Methods for Compressible Flows on Unstructured Grids," *Computational Fluid Dynamics 2000*, edited by N. Satofuka, Springer, Berlin, 2001, pp. 93–98.

[9]Shende, N., Arora, K., and Balakrishnan, N., "Convergence Acceleration Using Implicit Relaxation Procedures for Cell Centre and Cell Vertex Finite Volume Schemes," Dept. of Aerospace Engineering, Fluid Mechanics Rept. 2001 FM 03, Indian Inst. of Science, Bangalore, India, March 2001.

[10]Hirsch, C., "Numerical Computation of INTERNAL AND EXTERNAL FLOWS," *Fundamental of Numerical Discretization*, 8th ed., Vol. 1, Wiley, New York, 2000, p. 437.

[11]Batina, J. T., "Implicit Upwind Solution Algorithms for Three-Dimensional Unstructured Meshes," *AIAA Journal*, Vol. 31, No. 5, 1993, pp. 801–805.

[12]Venkatakrishnan, V., and Mavriplis, D. J., "Implicit Solvers for Unstructured Meshes," *Journal of Computational Physics*, Vol. 105, 1993, pp. 83–91.

[13]Luo, Hong, Baum, Joseph D., and Lohner, Rainald, "A Fast, Matrix-Free Implicit Method for Compressible Flows on Unstructured Grids," *Journal of Computational Physics*, Vol. 146, 1998, pp. 664–690.

[14]Yoon, S., and Jameson, A., "Lower–Upper Symmetric-Gauss-Seidel Method for the Euler and Navier–Stokes Equations," *AIAA Journal*, Vol. 26, No. 9, 1988, pp. 1025, 1026.

[15]Jameson, A., and Turkel, E., "Implicit Schemes and LU Decomposition," *Mathematics of Computations*, Vol. 37, No. 156, 1981, pp. 385–397.

[16]Sharov, Dmitri, and Nakahashi, K., "Reordering of Hybrid Unstructured Grids for Lower–Upper Symmetric Gauss–Seidel Computations," *AIAA Journal*, Vol. 36, No. 3, 1998, pp. 484–486.

[17]Van Leer, B., "Flux Vector Splitting for Euler Equations," NASA Rept. 82-30, 1982.

[18]Barth, T. J., "Aspects of Unstructured Grids and Finite Volume Solvers for the Euler and Navier Stokes Equations," CFD Lecture Series: 1994-05, von Kármán Inst., Brussels, March 1994.

[19]Barth, T. J., "A 3-D Least-Squares Upwind Euler Solver for Unstructured Meshes," *Proceedings of 13th International Conference on Numerical Methods in Fluid Dynamics*, edited by M. Napolitano and F. Sabetta, Lecture Notes in Physics, No. 414, Springer-Verlag, Berlin, 1992, p. 240.

[20]Ashford, G. A., and Powell, K. G., "An unstructured Grid Generation and Adaptive Solution Technique for High Reynolds Number Compressible Flows," CFD Lecture Series: 1996-06, von Kármán Inst., Brussels, March 1996.

[21]Balakrishnan, N., and Deshpande, S. M., "Reconstruction on Unstructured Meshes with Upwind Solvers," *Proceedings of the First Asian CFD conference*, Vol. 1, edited by W. H. Hui, Y.-K. Kwok, and J. R. Chasnov, Dept. of Mathematics, Hong Kong Univ. of Science and Technology, Hong Kong, 1995, pp. 359–364.

[22]Venkatakrishnan, V., "Convergence to Steady State Solutions of the Euler Equations on Unstructured Grids with Limiters," *Journal of Computational Physics*, Vol. 118, 1995, pp. 120–130.

[23]Delaunay, M., and Essers, J. A., "An Accurate Finite-Volume Scheme for Euler and Navier–Stokes Equations on Unstructured Grids," AIAA Paper 95-1710, June 1995.

[24]Trefethen, L. N., and Bau, D., *Numerical Linear Algebra*, Society for Industrial and Applied Mathematics, Philadelphia, 1997, p. 313.

[25]Ravikumar, D., "2D Compressible Viscous Flow Computations Using Acoustic Flux Vector Splitting (AFVS) Scheme," M.S. Thesis, Dept. of Aerospace Engineering, Indian Inst. of Science, Bangalore, India, Sept. 2001.

[26]Karypis, G., and Kumar, V., "A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes and Computing Fill-Reducing Ordering of Sparse Matrices," Ver. 4.0, Dept. of Computer Science/Army HPC Research Center, Univ. of Minnesota, Minneapolis, MN, 1998.

[27]Blair, A. B., Jr., "Stability and Control Characteristics of a Monoplanar Missile Configuration with Two Low-Profile Tail Arrangements at March Numbers from 1.70 to 2.86," NASA TM X-3533, Aug. 1977.

S. Mahalingam
Associate Editor