JOURNAL OF AIRCRAFT
Vol. 36, No. 6, November–December 1999

Large-Scale Parallel Unstructured Mesh Computations for Three-Dimensional High-Lift Analysis

D. J. Mavriplis* and S. Pirzadeh†
NASA Langley Research Center, Hampton, Virginia 23681-0001

A complete geometry to drag polar analysis capability for three-dimensional high-lift configurations is described. The approach is based on the use of unstructured meshes to enable rapid turnaround for complicated geometries that arise in high-lift configurations. Special attention is devoted to creating a capability for enabling analyses on highly resolved grids. Unstructured meshes of several million vertices are initially generated on a workstation and subsequently are refined on a supercomputer. The flow is solved on these refined meshes on large parallel computers using an unstructured agglomeration multigrid algorithm. Good prediction of lift and drag throughout the range of incidences is demonstrated on a transport takeoff configuration using up to 24.7×10^6 grid points. The feasibility of using this approach in a production environment on existing parallel machines is demonstrated, as well as the scalability of the solver on machines using up to 1450 processors.

I. Introduction

THE computation of three-dimensional high-lift flows constitutes one of the most challenging steady-state aerodynamic analysis problems today. Three-dimensional high lift is typically characterized by complicated geometries, involving flaps, slats, and hinge fairings, in addition to very complex flow physics that must be captured adequately to provide a useful predictive capability for the design process.

Unstructured grid techniques offer the potential for greatly reducing the grid generation time associated with such problems. Furthermore, unstructured mesh approaches enable the use of adaptive meshing techniques that hold great promise for increasing solution accuracy at minimal additional computational cost. However, unstructured mesh solvers require significantly higher computational resources than their structured grid counterparts, thus limiting their applicability for large three-dimensional calculations.

Because of the inherent complexities of high-lift flows (both geometric and flow physical), the accurate analysis of such flows requires highly resolved grids. Current estimates place the requirements for accurate Reynolds-averaged Navier–Stokes high-lift analysis of a complete transport aircraft configuration in the range of 10^7-10^8 grid points. This represents problems over an order of magnitude larger than are currently considered practical for unstructured mesh methods. The consideration of such problems gives rise to significant challenges not only in the flow solution process, but also in the grid generation procedure, which is usually performed locally on a workstation.

The rapid advances in parallel computer architectures, with their large aggregate memory capacity and CPU power, have reached the point where such calculations can be considered in a practical sense. This is particularly true for unstructured mesh techniques, which have been shown to scale very favorably on massively parallel machines using hundreds of processors.[1–3]

The goal of this work is to demonstrate a complete geometry to drag polar practical high-lift analysis capability, based on unstructured mesh techniques, using up to 25×10^6 grid points for a full aircraft configuration. The approach involves the generation of multimillion point unstructured meshes on a workstation, the refinement of these grids by an order of magnitude on a supercomputer, and the solution of the flow on these refined grids in a matter of hours on large parallel computers. Scaling of the solver on machines using up to 1450 processors is demonstrated.

II. Tetrahedral Mesh Generation

The computational grids employed were generated with the NASA Langley Research Center unstructured grid generation codes GridTool and VGRIDns. The subject geometry, known as the energy efficient transport (EET), consists of a fuselage, wing, leading-edge slats, and trailing-edge flaps in a high-lift position and involves complexities such as sharp corners and small gaps between multiple components.

The geometry, defined in the IGES format, is first converted into a number of contiguous patches with the grid-utility interface GridTool. The union of all patches, including the outer boundaries, forms a solid surface that is used by VGRIDns for triangulation. The grid characteristics such as spacings, grid stretching, rate of growth, etc., are also prescribed by the user with GridTool to complete the grid input file. The processes of geometry preparation and grid parameter setup with GridTool constitutes 50–90% of the total grid-generation time depending on the complexity of the geometry definition.

The grid-generation technique used in VGRIDns is based on the advancing-front method (AFM)[4] and the advancing-layers method (ALM)[5] and produces fully tetrahedral meshes. The generation of viscous grids, containing thin layers of tetrahedral cells, is divided into three main steps: 1) generation of a triangular surface grid by the ALM and/or AFM, 2) generation of thin tetrahedral cells in the boundary layer by the ALM, and 3) generation of a regular (inviscid) tetrahedral grid outside the boundary layer by the AFM. Although the entire process is completed in separate stages with this approach, it is performed in a single run with automatic transitions from one stage to another. Generation of inviscid grids is accomplished in a similar fashion by simply skipping the second step.

Both the AFM and the ALM are based on the marching techniques. Grids are generated with these methods by forming tetrahedral cells originating from triangulated boundaries and marching into the computational domain. Unlike the conventional AFM, which introduces cells in the field in a totally unstructured manner, the ALM generates layers of thin tetrahedral cells in a more orderly fashion while maintaining many advantageous features of the AFM. The new strategy reduces the grid-generation complexities which usually arise from floating-point operations (FLOPs) of small numbers associated with extremely small viscous grid spacings.

During the AFM and ALM marching processes, information regarding the grid point distributions is provided by a transparent

Received 6 February 1999; revision received 5 April 1999; accepted for publication 6 April 1999.

*Research Fellow, Mail Stop 132C, Institute for Computer Applications in Science and Engineering.

†Senior Research Engineer, Mail Stop 499, Configuration Aerodynamics Branch.

Cartesian background grid overlaying the entire domain.[6] Included in the background grid are a number of point and line sources prescribed by the user. The grid characteristics are smoothly diffused from the sources onto the background grid nodes by solving an elliptic equation. The problem is similar to that of the heat transfer in a conducting medium. The smoothed parameters are then interpolated from the background grid (and sources) during the unstructured grid generation.

Two main operations are involved in the ALM: 1) computation of surface vectors along which the grid points are distributed and 2) construction of a pattern of compatible tetrahedral cell connectivities within the thin layers. The surface vectors are calculated using a robust iterative algorithm based on an equal-angle criterion followed by a Laplacian smoothing operation. The connectivities among the tetrahedral cells are predetermined by an efficient, all-integer algorithm that eliminates the need for a series of computer-intensive, floating-point calculations as used in the conventional AFM.

Thin layers of tetrahedral cells are formed by inserting new points along the surface vectors and connecting the points according to the predetermined connectivity pattern. The distribution of points along surface vectors is determined by the stretching function

$$\mathrm{d}z_{i+1} = \mathrm{d}z_1 \times [1 + a \times (1 + b)^i]^i \qquad (1)$$

where $\mathrm{d}z_i$ is the normal spacing of the ith layer, $\mathrm{d}z_1$ is the first layer spacing prescribed by the user, and the factors a and b are constants determining the rate of stretching.

The grid layers continue marching in the field until a unit aspect ratio is reached, or some other limiting criteria, based on the background grid information and/or proximity of the approaching fronts, prevent them from further advancement. At this point, the process automatically switches from the advancing-layers to the advancing-front mode to generate a regular grid outside the boundary layer. With a common background grid controlling both methods, the transition from thin layers to the regular grid becomes gradual and continuous. Also, the fact that the number of layers varies from one location to another adds to the flexibility of the method and the smoothness of the generated grids.

A salient feature of VGRIDns is its ability to generate multidirectional, anisotropically stretched grids in which the surface triangles and tetrahedra in the field are elongated in user-prescribed directions.[7] With this capability, fewer points are distributed in the directions of reduced flow gradients without loss of grid resolution in other essential directions. Grid stretching with VGRIDns is achieved through prescribing a stretching direction and two spacings (along and normal to the stretching direction) for each background grid source. To minimize complications due to the marching of highly stretched cells, a local transformation is performed for each new cell being generated. The physical (stretched) space is mapped into an isotropic frame, where equilateral grid elements are formed, and the generated grid is back transformed to the physical space. Spanwise grid stretching for transport-type configurations results in at least a factor of 3 reduction in the total number of grid points.

For the grids presented, the parameter $\mathrm{d}z_1$ was set to $1.35E-06$ wing chords, and the values of a and b in Eq. (1) were set to 0.4 and 0.01, respectively. Anisotropic grid stretching was mainly applied at the leading and trailing edges of the main wing and flaps in the spanwise direction. The aspect ratios of the generated-grid elements (triangles on the surface and tetrahedra in the volume) vary from a maximum value of about 100:1 at the midspan leading edges, where the concentration of points is maximum, to 1:1 at the surface corners, fuselage, and in the field away from the geometry.

Several tetrahedral grids were generated for the present study. The grid used as the basis for the coarse grid computations described subsequently contains 115,489 boundary nodes, 3,107,075 total grid nodes, and 18,216,138 tetrahedral cells. Figure 1 shows the surface grid along with the triangulation on the symmetry plane. As evident, the triangles are highly stretched at the leading edges in the spanwise direction. Also, the thin viscous layers are shown on the symmetry plane around the geometry. The level of spatial resolution is illustrated by Fig. 2, where a cross section of the volume grid is given, showing tetrahedral cells around the multielement airfoil geometry. Subdivision of this grid into 24.7×10^6 points (as explained in the following sections) results in a doubling of the spatial resolution in all three coordinate directions.

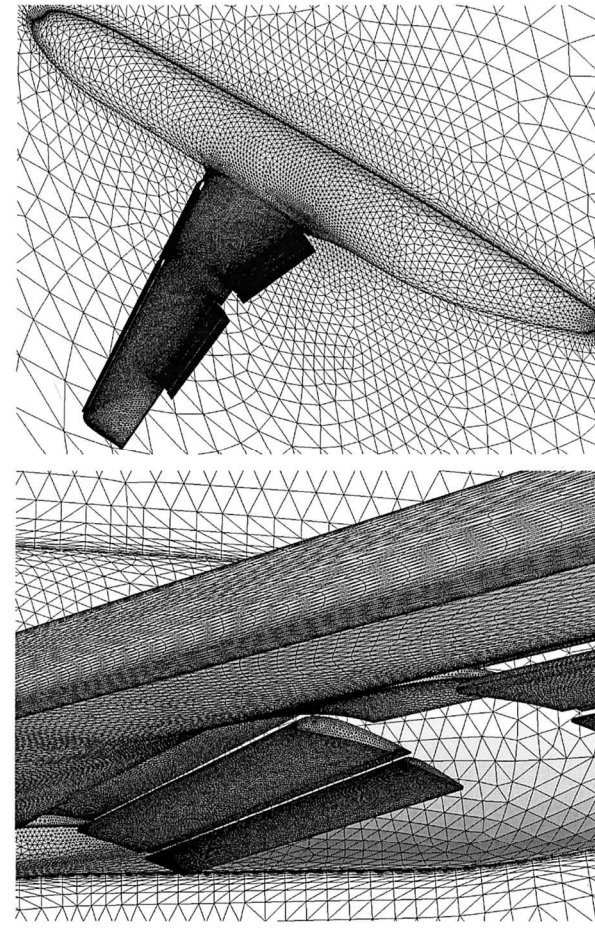

Fig. 1 Surface grid for high-lift configuration with spanwise stretching along slat leading edges.

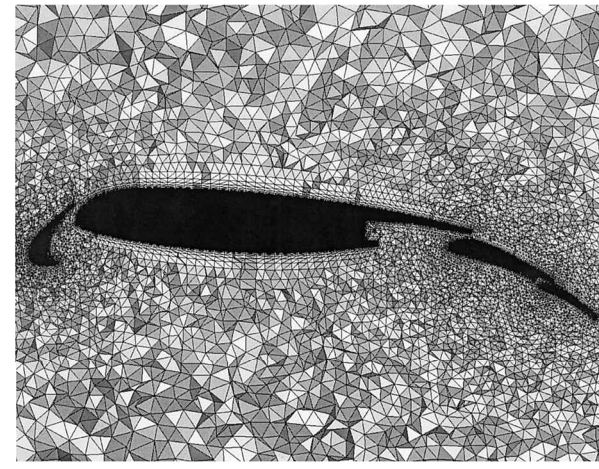

Fig. 2 Two-dimensional cross section of volume grid at spanwise station along wing.

The grids were generated using a Silicon Graphics (SGI) Octane workstation with a 195-MHz (R10000) processor. The entire process, from CAD definition to the final postprocessed volume grid, was completed in about 60 cumulative labor hours. As mentioned earlier, a substantial percentage (in this case, more than 90%) of the total grid-generation time was spent on the geometry preparation. The surface and volume grids (fine mesh) were generated in about 2.5 CPU h using the workstation.

III. Prismatic Element Merging

The advancing-layers phase of the initial grid-generation procedure produces regularly shaped thin tetrahedral elements in the boundary-layer regions near the geometry surface, which can easily be merged into well-shaped prismatic elements.

There are several reasons why the use of prismatic elements rather than tetrahedral elements in these regions is advantageous. First, for the vertex-based discretization employed, prismatic elements result in fewer interconnecting edges than tetrahedral elements, for the same set of unknowns. This has the effect of reducing the memory and computational overheads because the residual assembly in the solver is based on an edge data structure. Because prismatic elements contain almost half as many edges as tetrahedral elements, and up to two-thirds of the grid elements are often merged into prisms, the savings can be substantial.

Second, the use of prismatic elements provides a distinct decoupling in the discretization between the normal and tangential directions in the boundary-layer regions, which is essential for the success of the directional line-implicit multigrid algorithm to be described. For highly stretched tetrahedral elements, the strong normal connections (edges) and the strong diagonal connections cannot be solved implicitly simultaneously, leading to a reduced effectiveness of the algorithm.[8]

Any accuracy benefits of using prismatic elements over tetrahedral elements in boundary-layer regions have not been proven conclusively. On the other hand, it is generally acknowledged that prismatic element discretizations are at a minimum no less accurate than the corresponding tetrahedral element discretizations.[9]

The output of the VGRIDns grid-generation program identifies each point generated in the viscous advancing-layers region with the surface grid point (surface normal) from which it originated. With this information, an algorithm can be devised for merging successive triplets of tetrahedra into prismatic elements, as shown in Fig. 3. Because the height of the advancing layers is nonuniform across the geometry surface, special care must be taken to ensure a consistent grid when merging the new prismatic elements with the remaining tetrahedral elements. Grid inconsistencies can arise in the form of a hanging diagonal on a prismatic face that borders on two tetrahedral neighbors, as shown in Fig. 4. Such situations are handled by considering the total number of hanging diagonals on a given prismatic element. Prisms containing three hanging diagonals (one on each quadrilateral face) are resubdivided into three tetrahedra. Prisms containing two compatible hanging diagonals are divided into one tetrahedron and one pyramid. In cases where this is not possible, that is, two incompatible hanging edges, or when there is only one hanging diagonal, an additional vertex is inserted at the middle of the prismatic element, which is then subdivided into one pyramid and six tetrahedra (for two hanging edges) or two pyramids and four tetrahedra (for one hanging edge). In practice, only a very small number of additional vertices are required.

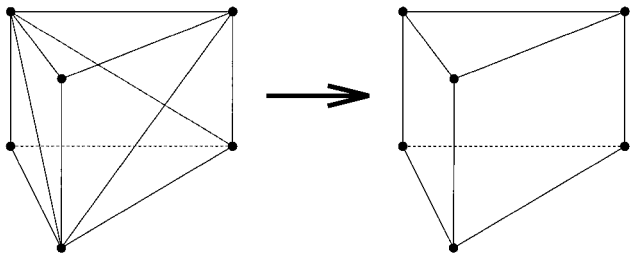

Fig. 3 Merging of three suitable tetrahedra into a single prismatic element.

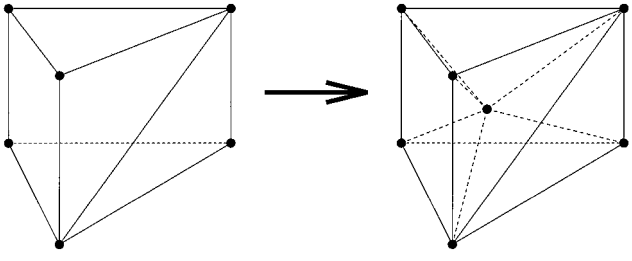

Fig. 4 Insertion of new vertex and subdivision of prismatic element with one noncompatible face diagonal.

IV. Global Grid Refinement

The generation of and flow solution on large unstructured grids, that is, more than 10^7 grid points, are not practical on current high-end workstations. The memory and CPU-time requirements associated with grids of this size dictate the use of large parallel supercomputers. Whereas unstructured mesh flow solvers have been shown to parallelize effectively on such machines, success in grid-generation parallelization has not been as forthcoming. In spite of several research efforts in this area,[10] parallelization of the grid-generation process remains hindered by the complicated logic, the attention devoted to special cases, and the need to easily and rapidly access geometrical information and preprocessing tools, which most often reside on workstations.

The strategy developed in this work consists of generating relatively coarse unstructured grids on a workstation and refining these grids on a large-memory supercomputer. The adjective relatively needs to be stressed here because these initial grids usually contain several million vertices (up to 20×10^6 tetrahedra) and already represent the limit of what can be achieved on a high-end workstation.

Once the initial tetrahedral grid has been generated, it is merged into a mixed prismatic–tetrahedral mesh, as described earlier. This hybrid mesh is then shipped to a supercomputer, such as an SGI Origin 2000, where it is globally refined by subdividing each grid element (tetrahedra and prisms) into eight smaller self-similar elements.[11] This results in a factor of 8 increase in grid size. As an example, the generation of a 24.7×10^6 point grid through global refinement of the 3.1×10^6 point grid described earlier required about 30 min of CPU time on a single processor of the SGI Origin 2000.

Ultimately, the mesh refinement operation should be implemented in parallel. However, for expediency at this stage, mesh refinement is performed sequentially on a single processor of the SGI Origin 2000. However, the access to a large central memory provided by the cc-NUMA shared memory architecture of the SGI Origin 2000 is a key enabling feature for the refinement of large unstructured grids. An additional benefit of this approach is that the resulting large grid is then available directly on the parallel machine for the flow computations, obviating the need to transfer large amounts of data across the network. This approach naturally extends to adaptive meshing strategies, which are planned for future work.

One drawback of the current approach is that newly generated surface points do not lie exactly on the original surface description of the model geometry, but rather along a linear interpolation between previously existing coarse grid surface points. For a single level of refinement, this drawback is not expected to have a noticeable effect on the results. An interface for reprojecting new surface points onto the original surface geometry is currently under consideration.

V. Base Solver

The Reynolds-averaged Navier–Stokes equations are discretized by a finite volume technique on meshes of mixed element types that may include tetrahedra, pyramids, prisms, and hexahedra. All elements of the grid are handled by a single unifying edge-based data structure in the flow solver.[12]

The governing equations are discretized using a central difference finite volume technique with added artificial dissipation. The thin-layer form of the Navier–Stokes equations is employed in all cases, and the viscous terms are discretized to second-order accuracy by finite difference approximation.[12] For multigrid calculations, a first-order discretization is employed for the convective terms on the coarse grid levels.

The basic time-stepping scheme is a three-stage explicit multistage scheme with stage coefficients optimized for high-frequency damping properties[13] and a Courant–Friedrichs–Lewy (CFL) number of 1.8. Convergence is accelerated by a local block Jacobi preconditioner, which involves inverting a 5×5 matrix for each vertex at each stage.[14–17] A low Mach number preconditioner[18–20] is also implemented. This is imperative for high-lift flows that may contain

large regions of low Mach number flow particularly on the lower surfaces of the wing. The low Mach number preconditioner is implemented by modifying the dissipation terms in the residual as described in Ref. 8 and then taking the corresponding linearization of these modified terms into account in the Jacobi preconditioner, a process sometimes referred to as preconditioning[2] (Refs. 8 and 21).

The single-equation turbulence model of Spalart and Allmaras[22] is utilized to account for turbulence effects. This equation is discretized and solved in a manner completely analogous to the flow equations, with the exception that the convective terms are only discretized to first-order accuracy.

VI. Directional Implicit Multigrid Algorithm

An agglomeration multigrid algorithm[12,23,24] is used to further enhance convergence to steady state. In this approach, coarse levels are constructed by fusing together neighboring fine grid control volumes to form a smaller number of larger and more complex control volumes on the coarse grid. Although agglomeration multigrid delivers very fast convergence rates for inviscid flow problems, the convergence obtained for viscous flow problems remains much slower, even when employing preconditioning techniques as described in the preceding section. This slowdown is mainly due to the large degree of grid anisotropy in the viscous regions. Directional smoothing and coarsening techniques[8,25] can be used to overcome this aspect-ratio-induced stiffness.

Directional smoothing is achieved by constructing lines in the unstructured mesh along the direction of strong coupling, that is, normal to the boundary layer, and solving the implicit system along these lines using a tridiagonal line solver. A weighted graph algorithm is used to construct the lines on each grid level, using edge weights based on the stencil coefficients for a scalar convection equation. This algorithm produces lines of variable length. In regions where the mesh becomes isotropic, the length of the lines reduces to zero (one vertex, zero edges), and the preconditioned explicit scheme described in the preceding section is recovered. An example of the set of lines constructed from the two-dimensional unstructured grid in Fig. 5 is shown in Fig. 6.

In the agglomeration multigrid algorithm, coarse-level grids are constructed by fusing together or agglomerating neighboring control volumes to form a coarser set of larger but more complex control volumes. A multigrid cycle consists of performing a time step on the fine grid of the sequence, transferring the flow solution and residuals to the coarser level, performing a time step on the coarser level, and then interpolating the corrections back from the coarse level to update the fine grid solution. The process is applied recursively to the coarser grids of the sequence.

In previous work, a directional agglomeration strategy has been shown to speed convergence for problems involving highly stretched meshes.[3,25] However, coarsening factors are presently limited to a ratio of 4:1 between fine and coarse levels, when employing a directional coarsening algorithm. This leads to additional memory overheads for the storage of these levels. In the interest of reducing overall memory requirements, to enable the solution of larger grid sizes, the directional coarsening strategy has been temporarily abandoned in favor of the simpler isotropic coarsening strategy, which produces coarsening ratios of the order of 8:1 (Ref. 12). Whereas overall memory requirements are reduced due to the more rapid coarsening rates, observed convergence rates are somewhat slower than those reported previously using the directional coarsening strategy.[3,25]

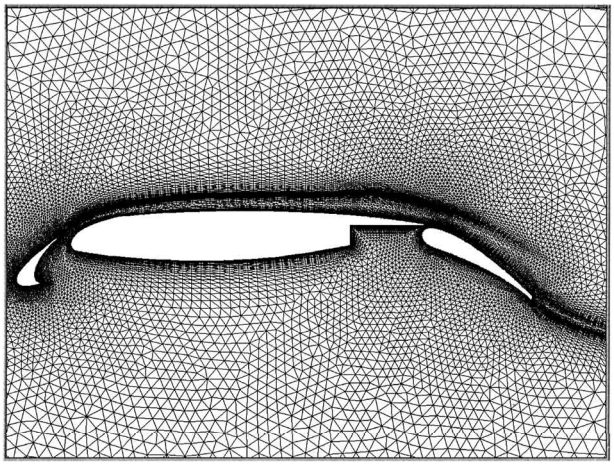

Fig. 5 Unstructured grid for three-element airfoil; number of points = 61,104, wall resolution = 10^{-6} chords.

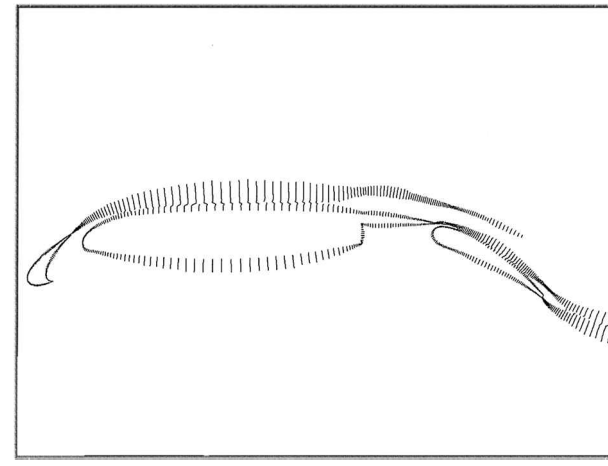

Fig. 6 Directional-implicit lines constructed on grid of Fig. 5 by weighted-graph algorithm.

VII. Parallel Implementation

The unstructured multigrid solver is parallelized by partitioning the domain using a standard graph partitioner[26,27] and by communicating between the various grid partitions running on individual processors using the Message-Passing-Interface (MPI) library.[28] Distributed-memory explicit message-passing parallel implementations of unstructured mesh solvers have been discussed extensively in the literature.[29–31] In this section we focus on the nonstandard aspects of the present implementation that are particular to the directional-implicit agglomeration multigrid algorithm.

In the multigrid algorithm, the vertices on each grid level must be partitioned across the processors of the machine. Because the mesh levels of the agglomeration multigrid algorithm are fully nested, a partition of the fine grid could be used to infer a partition of all coarser grid levels. Whereas this would minimize the communication in the intergrid transfer routines, it affords little control over the quality of the coarse grid partitions. Because the amount of intragrid computation on each level is much more important than the intergrid computation between each level, it is essential to optimize the partitions on each grid level rather than between grid levels. Therefore, each grid level is partitioned independently. This results in unrelated coarse and fine grid partitions. To minimize intergrid communication, the coarse level partitions are renumbered such that they are assigned to the same processor as the fine grid partition with which they share the most overlap.

For each partitioned level, the edges of the mesh that straddle two adjacent processors are assigned to one of the processors, and a ghost vertex is constructed in this processor, which corresponds to the vertex originally accessed by the edge in the adjacent processor, as shown in Fig. 7. During a residual evaluation, the fluxes are computed along edges and accumulated to the vertices. The flux contributions accumulated at the ghost vertices must then be added to the flux contributions at their corresponding physical vertex locations to obtain the complete residual at these points. This phase incurs interprocessor communication. In an explicit (or point-implicit) scheme, the updates at all points can then be computed without any interprocessor communication once the residuals at all points have been calculated. The newly updated values are then communicated to the ghost points, and the process is repeated.

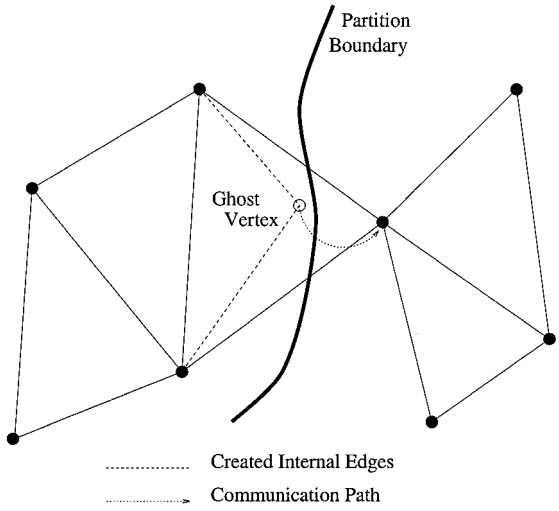

Fig. 7 Creation of internal edges and ghost points at interprocessor boundaries.

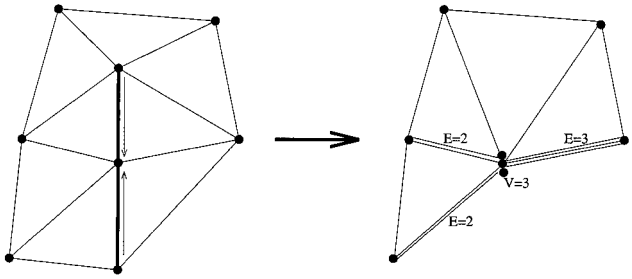

Fig. 8 Line edge contraction and creation of weighted graph for mesh partitioning; *V* and *E* are vertex and edge weights, respectively.

The use of line solvers can lead to additional complications for distributed-memory parallel implementations. Because the classical tridiagonal line solve is an inherently sequential operation, any line that is split between multiple processors will result in processors remaining idle while the off-processor portion of their line is computed on a neighboring processor. However, the particular topology of the line sets in the unstructured grid permit partitioning the mesh in such a manner that lines are completely contained within an individual processor, with minimal penalty (in terms of processor imbalance or additional numbers of cut edges). This can be achieved by using a weighted-graph-based mesh partitioner such as the Chaco[26] or MeTiS[27] partitioners. Weighted-graph partitioning strategies attempt to generate balanced partitions of sets of weighted vertices and to minimize the sum of weighted edges that are intersected by the partition boundaries.

To avoid partitioning across implicit lines, the original unweighted graph (set of vertices and edges) that defines the unstructured mesh is contracted along the implicit lines to produce a weighted graph. Unity weights are assigned to the original graph, and any two vertices that are joined by an edge that is part of an implicit line are then merged together to form a new vertex. Merging vertices also produce merged edges, as shown in Fig. 8, and the weights associated with the merged vertices and edges are taken as the sum of the weights of the constituent vertices or edges. The contracted weighted graph is then partitioned using one of the partitioners described in Refs. 26 and 27, and the resulting partitioned graph is then decontracted, that is, all constituent vertices of a merged vertex are assigned the partition number of that vertex. Because the implicit lines reduce to a single point in the contracted graph, they can never be broken by the partitioning process. The weighting assigned to the contracted graph ensures load balancing and communication optimization of the final uncontracted graph in the partitioning process.

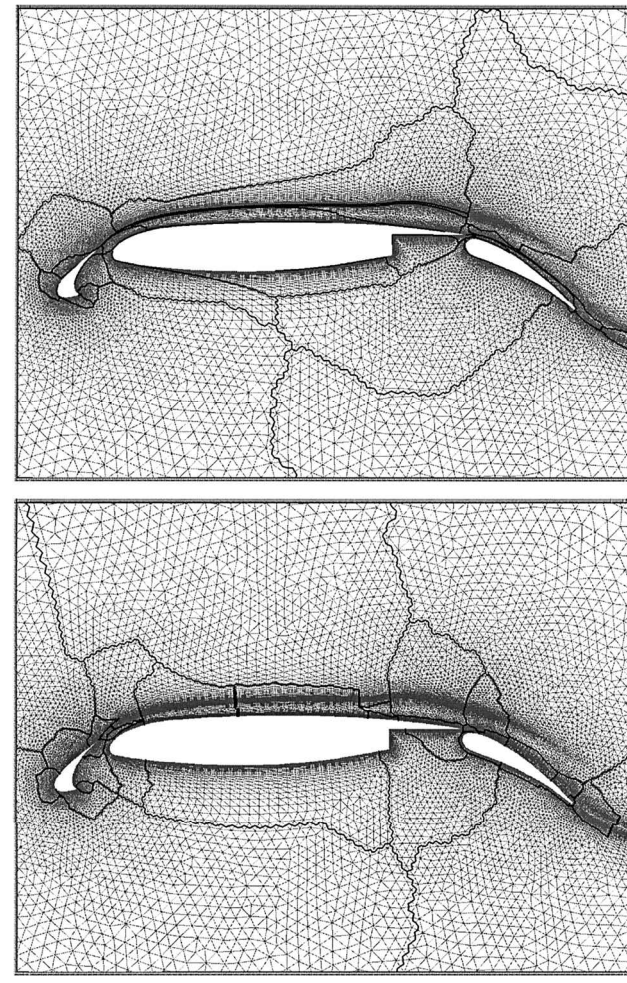

Fig. 9 Comparison of unweighted (upper) and weighted (lower) 32-way partition of two-dimensional mesh.

As an example, the two dimensional mesh in Fig. 5, which contains the implicit lines shown in Fig. 6, has been partitioned both in its original unweighted uncontracted form and by the graph contraction method just described. Figure 9 shows the results of both approaches for a 32-way partition. The unweighted partition contains 4760 cut edges (2.6% of total), of which 1041 are line edges (also 2.6% of total), whereas the weighted partition contains no intersected line edges and a total of 5883 cut edges (3.2% of total), that is, a 23% increase over the total number of cut edges in the nonweighted partition.

Because of the large size of the grids considered, all preprocessing operations must be performed on a large parallel supercomputer. This includes the global grid-refinement procedure, the agglomeration procedure, the partitioning of the various coarse and fine multigrid levels, and the determination of the interprocessor communication schedules. This is mostly due to the large memory requirements of these procedures, (which run between 50 and 75% of the memory requirements of the flow solver, that is $\approx$1 kB per grid point), rather than the CPU time requirements, which are small compared to those of the flow solver. At present, these procedures are executed sequentially on a single processor of an SGI Origin 2000, but use large portions of the memory of the entire machine. For example, the various preprocessing operations for a 24.7×10^6 point grid required between 10 and 20 GB of memory and between 45 and 90 min for each of the operations mentioned earlier. The sequential execution of large jobs of this nature is made possible by the shared memory architecture of the SGI Origin 2000 and cannot be performed on purely distributed memory machines such as the Cray T3E. The complete parallelization of these procedures for distributed-memory machines is planned for the near future.

The run times just given also apply to the Chaco partitioner. The exception was the MeTiS partitioner, which managed to partition the 24.7×10^6 point grid in approximately 10 min, about four times faster than the Chaco partitioner. On the other hand, the Chaco partitioner provided better load balancing in the final partitions, albeit with a slightly larger number of cut edges, particularly for the coarse grid levels of the multigrid sequence. The differences were still small enough that a detailed study of the effect of the two partitioners on parallel efficiency was not considered essential.

VIII. Scalability Study

The scalability of the directional-implicit multigrid algorithm is examined on an SGI Origin 2000 and a Cray T3E machine. The SGI Origin 2000 machine contains 128 MIPS R10000 195-MHz processors with 286 MB of memory per processor, for an aggregate memory capacity of 36.6 GB. The Cray T3E contains 512 DEC Alpha 300-MHz processors with 128 MB of memory per processor, for an aggregate memory capacity of 65 GB.

The test case involves a grid of 1.98×10^6 points over an ONERA M6 wing at a Mach number of 0.1, an incidence of 2.0 deg, and a Reynolds number of 3×10^6 and is reproduced from Ref. 3. Figures 10 and 11 show the relative speedups achieved on the two target hardware platforms for this case. For Figs. 10 and 11, perfect speedups were assumed on the lowest number of processors for which each case was run, and all other speedups are computed relative to this value. In all cases, timings were measured for the single-grid (nonmultigrid) algorithm, the multigrid algorithm using a V cycle, and the multigrid algorithm using a W cycle.

Figures 10 and 11 reveal good scalability on both machines up to the maximum number of processors. The better scalability of the single-grid vs the multigrid algorithms is indicative of the increased volume of communication generated by the coarse-grid-level time stepping in the multigrid algorithm. Although the multigrid W-cycle algorithm displays slightly inferior scalability than the V-cycle or single-grid algorithm, it provides the most rapid convergence and is, thus, used exclusively for all subsequent calculations.

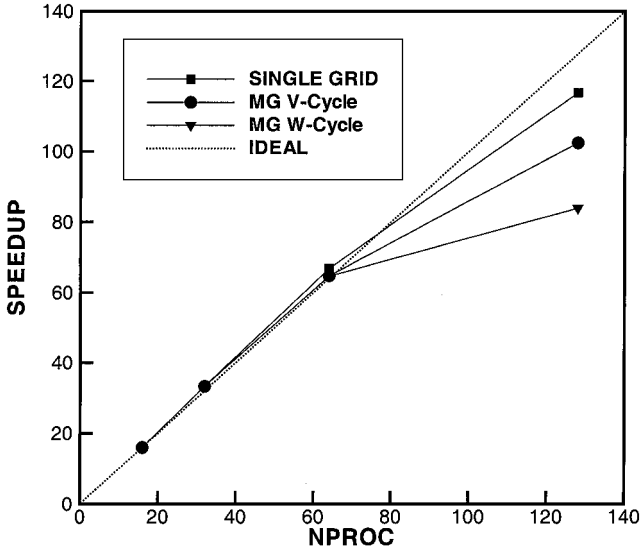

Fig. 10 Observed speedups for ONERA M6 wing case (1.98×10^6 grid points) on SGI Origin 2000.

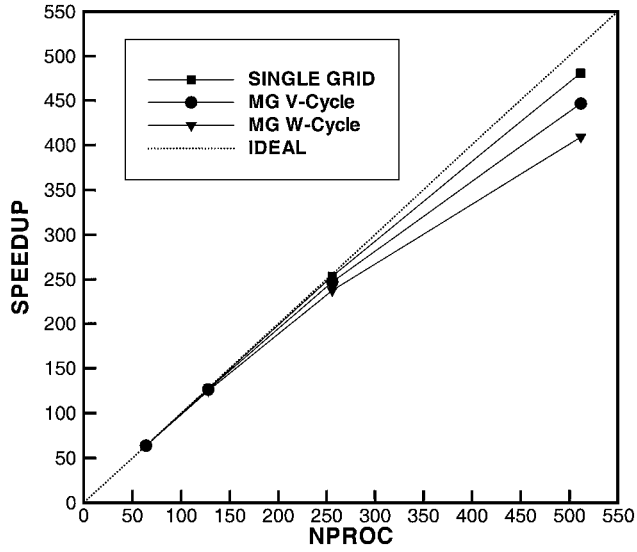

Fig. 11 Observed speedups for ONERA M6 wing case (1.98×10^6 grid points) on Cray T3E.

IX. Results

The high-lift flow over a complete aircraft configuration has been computed for an entire range of incidences on two grids of different resolution. The geometry consists of a twin-engine transport known as the EET configuration, which has been tested both as a full span and semispan model in the NASA Langley Research Center 14×22 ft subsonic wind tunnel.[32] The geometry studied in this work contains no pylon or nacelle. The wing has an aspect ratio of 10, a leading-edge sweep of 28.8 deg, and consists of a supercritical airfoil section with a slat and double-slotted flap. The case studied in this work consists of a take-off configuration, with a slat deflection of 50 deg, a vane deflection of 15 deg, and a flap deflection of 30 deg, with respect to the main airfoil. The freestream Mach number is 0.2, the Reynolds number is 1.6×10^6 based on the wing reference chord, and the experimental flow incidence varies over a range of -4 to 24 deg.

Experimental results are available in the form of force and moment coefficients as a function of angle of attack and chordwise pressure distributions at three spanwise locations. At the time of writing, only the pressure distributions at 10-deg incidence were accessible for comparison. Whereas Ref. 32 describes the experiments on a semispan model with pylon and nacelle, the full-span nacelle-off results used for comparison in this paper have not been previously published. The computations are all performed at zero yaw angle and, therefore, only include one-half of the symmetric aircraft geometry, delimited by a symmetry plane. The coarse grid for this case contained 3.1×10^6 vertices and 18.2×10^6 tetrahedra. This grid was generated directly on an Silicon Graphics Octane workstation, which required about 60 cumulative labor hours for geometry and grid-generation parameter setup and 2.5 h for the actual surface and volume grid generation. This tetrahedral grid was then

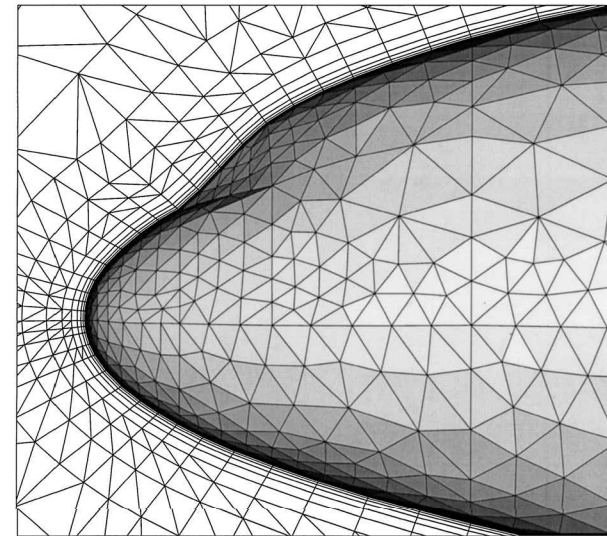

Fig. 12 Mixed prismatic–tetrahedral elements near fuselage nose region for 3.1×10^6 point grid.

merged into a mixed-element grid of 3.9×10^6 prisms, 6.6×10^6 tetrahedra, and 46,899 pyramids. The fine grid was obtained by uniform refinement of this mixed prismatic-tetrahedral grid, resulting in a grid of 24.7×10^6 vertices, with 53×10^6 tetrahedra, 31×10^6 prisms, and 281,000 pyramids. The refinement operation was performed sequentially on a single processor of an SGI Origin 2000 and required approximately 10 GB of memory and 30 min of CPU time. Figure 12 shows the mixed prism–tetrahedral nature of the 3.1×10^6 point grid in the vicinity of the nose of the fuselage on the symmetry plane. An illustration of the computed solution at 10-deg incidence on the 3.1×10^6 point grid is given in Fig. 13 as a set of surface pressure contours on the wing and flaps.

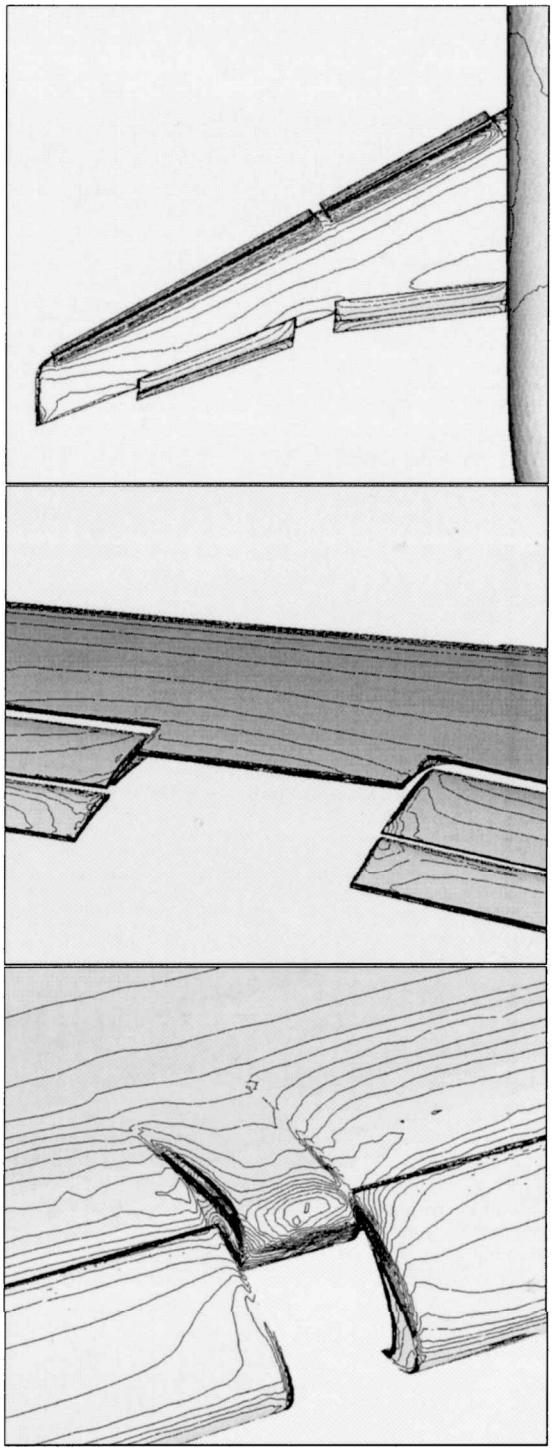

Fig. 13 Computed pressure contours on wing, slat, and flap corner areas on 3.1×10^6 point mesh; Mach = 0.2, incidence = 10 deg, $Re = 1.6 \times 10^6$.

The preconditioned line-implicit agglomeration multigrid algorithm was employed to converge the solution to a steady state. The isotropic agglomeration routine was used to reduce memory overheads of the flow solver, and scalar dissipation was employed in the discretization. Five multigrid levels were employed in the 3.1×10^6 point case, with the coarsest level containing only 1651 vertices. The convergence history on this grid for a flow incidence of 10 deg is shown in Fig. 14. A residual reduction of 4 orders of magnitude is observed in 600 multigrid cycles. However, the lift coefficient remains within 0.1% of its final value after only 180 multigrid

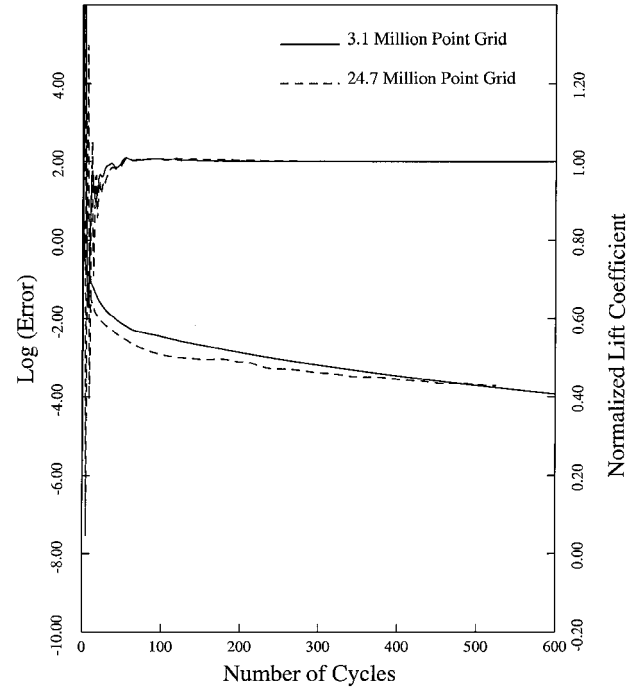

Fig. 14 Convergence rate for coarse (3.1×10^6 point) grid using five multigrid levels and low Mach number preconditioning and fine (24.7×10^6 point) grid using six multigrid levels and no preconditioning at 0.2 Mach number and 10-deg incidence.

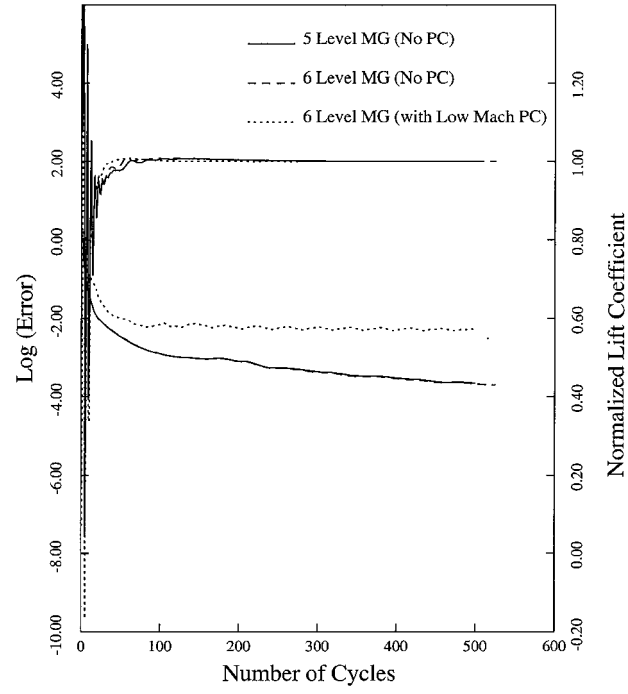

Fig. 15 Convergence rate for fine (24.7×10^6 point) grid using five and six multigrid levels with and without low Mach number preconditioning at 0.2 Mach number and 10-deg incidence.

cycles, so that an engineering result could be obtained with confidence in about 300 cycles.

For the 24.7×10^6 point grid case, convergence was hindered by the behavior of the low Mach number preconditioner. Figure 15 shows a comparison between the convergence history obtained using six multigrid levels with and without the low Mach number preconditioner, as well as using five multigrid levels without the low Mach number preconditioner. In the preconditioned case, convergence stalls after roughly 2.5 orders of magnitude decrease in the density residuals, whereas the nonpreconditioned case converges 4 orders of magnitude in 500 cycles. In fact, the convergence history for the nonpreconditioned case is nearly identical to that obtained on the coarser grid, as shown in Fig. 14, which provides a good illustration of the grid-independent convergence properties of the multigrid algorithm. The addition of a sixth multigrid level in the 24.7×10^6 point grid case (which contains only 718 points, vs 2208 points on the fifth level) has almost no effect on the overall convergence; therefore five multigrid levels were used in most of the computations.

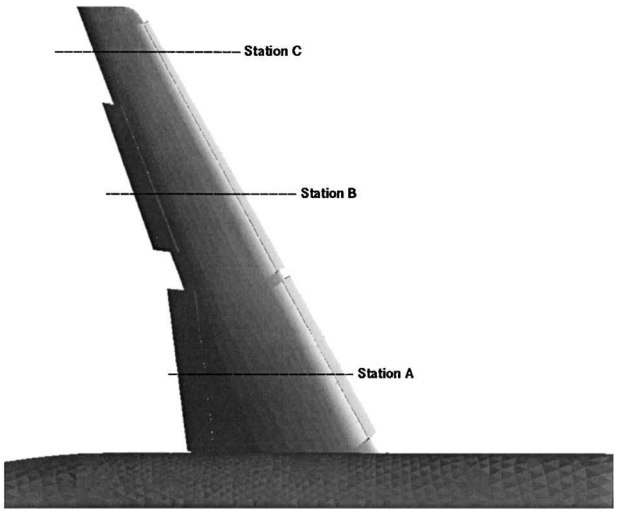

Fig. 16 Definition of three spanwise stations for comparison of computed and experimental pressure distributions.

Although the preconditioned case on the fine grid failed to converge the residuals monotonically, the lift coefficient actually approaches its final value faster than in the nonpreconditioned case, achieving a deviation smaller than 0.1% of its final value in just 110 multigrid cycles, as opposed to 260 cycles for the nonpreconditioned case. It is speculated that the convergence problems of the preconditioned case may relate to the limiter required by such techniques in regions of stagnating flow and may, therefore, be of a local nature. However, because of the uncertainty generated by the lack of residual convergence in the preconditioned cases, the complete lift curve has been computed both with and without preconditioning for the 24.7×10^6 point grid. This also affords the opportunity to study the effect of the preconditioner on the final solution at various angles of attack. Low Mach number preconditioning affects solution accuracy through a modification of the eigenvalues used to scale the artificial dissipation. Experimental values at the three spanwise locations shown in Fig. 16 for an incidence of 10 deg are compared in Figs. 17–19 with the computed surface pressures on the coarse and fine grids. For the fine grid, the preconditioned and nonpreconditioned results are almost indistinguishable; hence, only one set of fine grid results is plotted. The differences between the coarse and fine grid values are rather small, with the fine grid computations providing slightly higher suction peaks at the main and flap leading edges. Both computational results compare favorably with experimental values at all three stations.

A comparison between computed and experimental lift coefficients as a function of angle of attack is given in Fig. 20 for both grids. As expected, the fine grid produces slightly higher lift coefficients than the coarse grid. The effect of preconditioning on the fine grid lift values is very small. The experimental lift values are overpredicted by both fine and coarse grid results. However, the slope of the lift curve is reproduced very accurately by both computations. The maximum lift point, which experimentally occurs at 16-deg incidence, is well predicted by the coarse grid. The fine grid overpredicts the maximum lift incidence by 1 deg, giving a value of 17 deg. In both cases the value of $Cl_{\max}$ is overpredicted. These

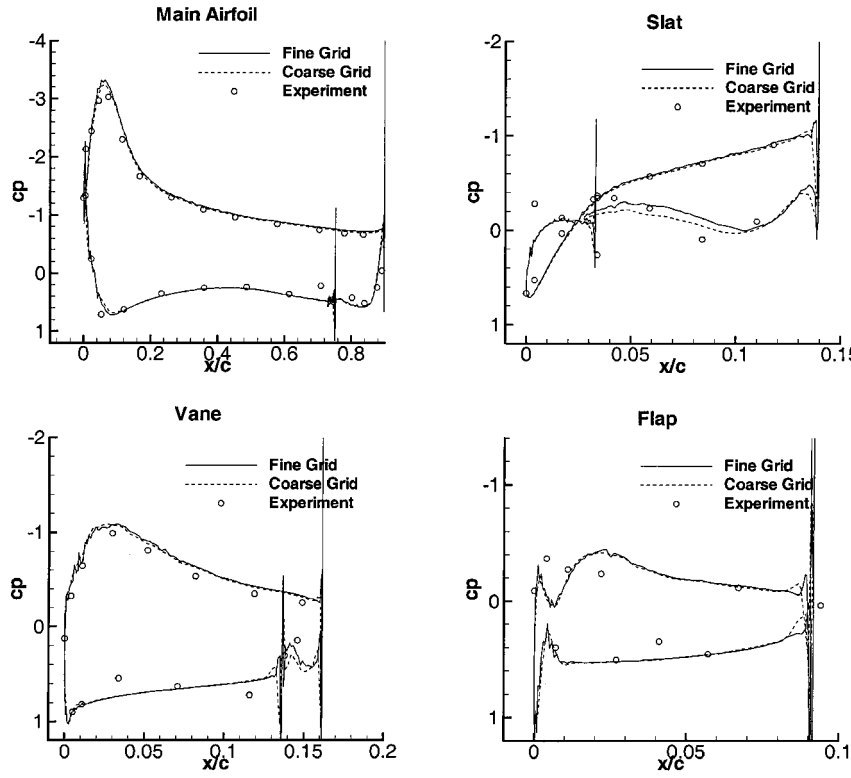

Fig. 17 Comparison of computed and experimental surface pressure distributions at spanwise station A for 10-deg incidence case.

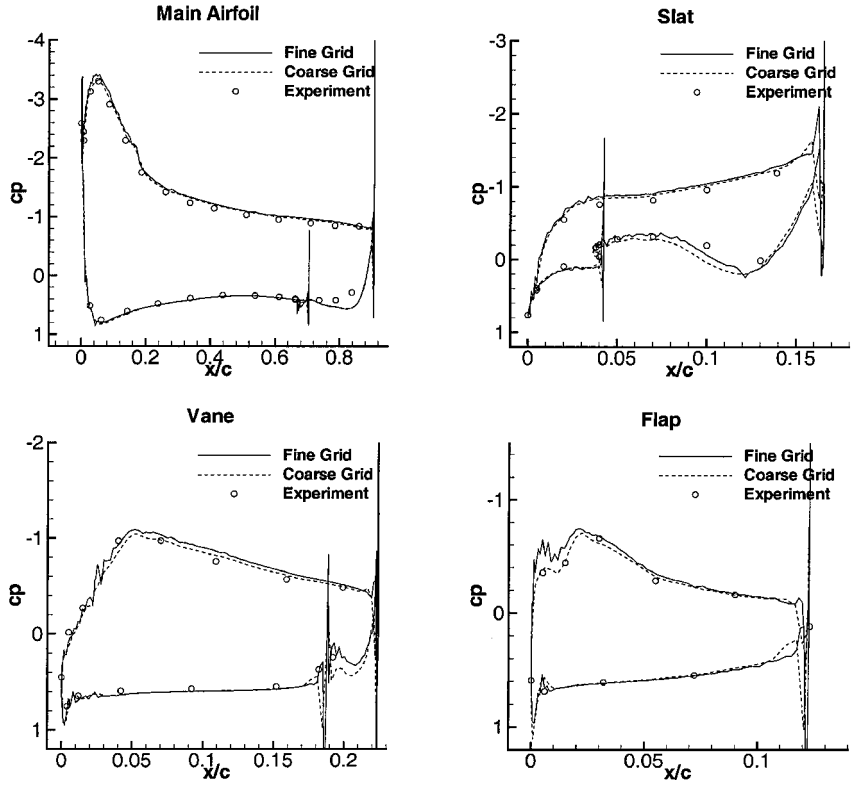

Fig. 18 Comparison of computed and experimental surface pressure distributions at spanwise station B for 10-deg incidence case.

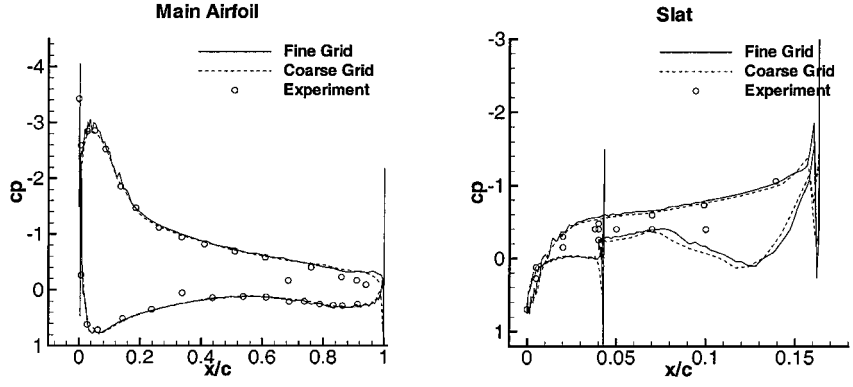

Fig. 19 Comparison of computed and experimental surface pressure distributions at spanwise station C for 10-deg incidence case.

characteristics are similar to those observed in two-dimensional calculations on highly resolved grids.[33]

After stall, the computations fail to converge adequately, producing large variations in the lift coefficients. The average, as well as the minimum and maximum of these computed lift coefficients are plotted in Fig. 20. Note that poststall computed values averaged in this manner follow the experimental values fairly closely, although the range of computed min–max values is rather large. That the coarse grid computations provide a closer prediction of the $Cl_{\max}$ point in this case must be considered fortuitous and probably cannot be generalized to other cases.

Figure 21 provides a comparison of computed and experimental drag coefficients vs angle of attack, whereas the corresponding drag polar plots are given in Fig. 22. The drag values appear to be reasonably well predicted, with the fine grid values agreeing most closely with the experimental values. The shape of the curves is well predicted on both grids up to stall, as are the absolute values of drag. In this case, the low Mach number preconditioning has a nonnegligible effect on the drag values, providing even closer agreement with experiment for the fine grid case than the unpreconditioned case. In fact, the agreement with experiment is very close. For example, at 10-deg incidence, the fine grid preconditioned value is 0.1810, whereas the experimental value is 0.1800, with only 10 counts difference between the two values. More experience with large-scale unstructured grid computations will be required to determined whether this level of agreement can be relied on or is simply fortuitous. Although computed pitching moments have not been examined, the level of agreement between computed and experimental pitching moments can be expected to be similar to that achieved for the lift coefficients.

A two-dimensional cut of the computed Mach contours on the fine grid is shown in Fig. 23. Qualitatively, these solutions appear to approach the type of resolution typically used in common two-dimensional calculations. However, isolated flow features such as the slat wake are not captured adequately because there has been no effort to anisotropically increase grid resolution in these areas.

The 3.1×10^6 point grid cases were run on an SGI Origin 2000 machine and a Cray T3E-600 machine. The scalability of this case on these machines is similar to that shown in Figs. 10 and 11. This case requires a total of 7 GB of memory and 80 min on 128 processors (250 MHz) of the Origin 2000 or 62 min on 256 processors of the Cray T3E-600 for a 500 multigrid cycle run.

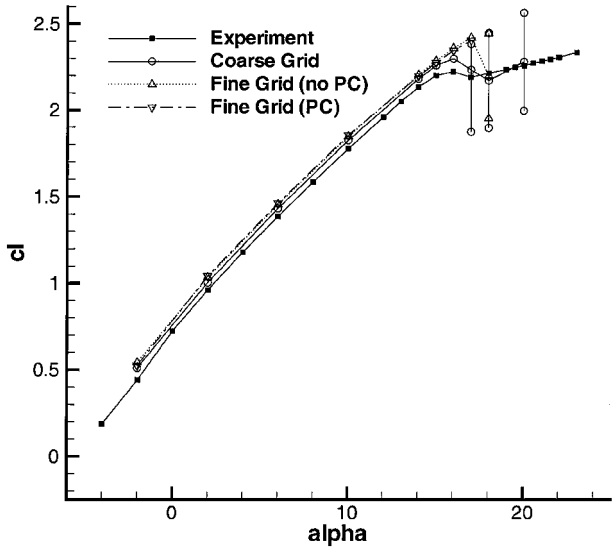

Fig. 20 Comparison of computed and experimental lift coefficients as a function of angle of attack.

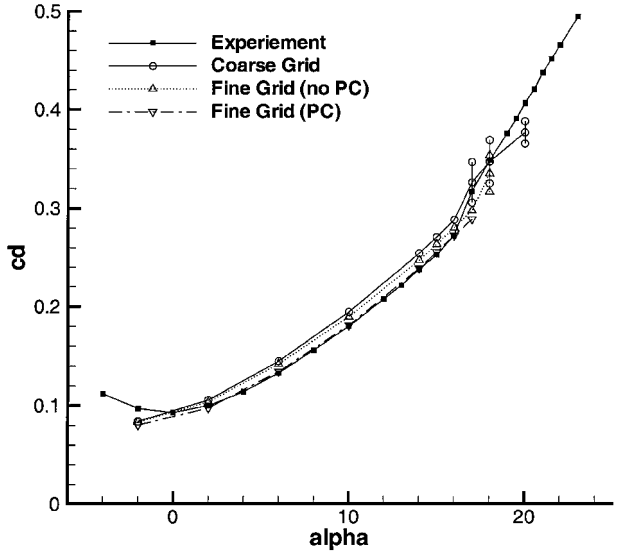

Fig. 21 Comparison of computed and experimental drag coefficients as a function of angle of attack.

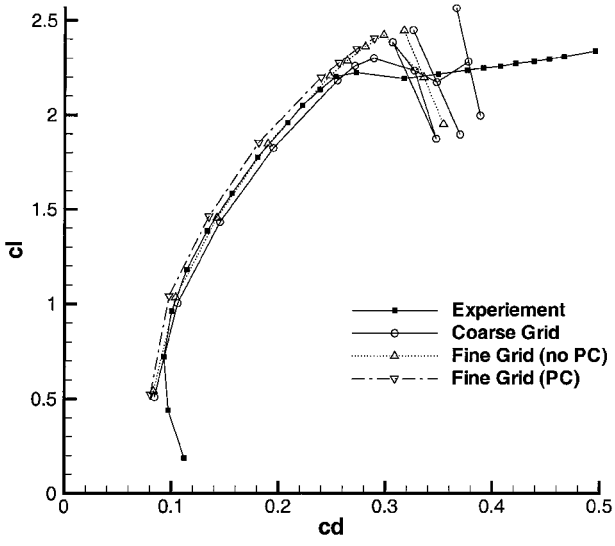

Fig. 22 Comparison of computed and experimental drag polars.

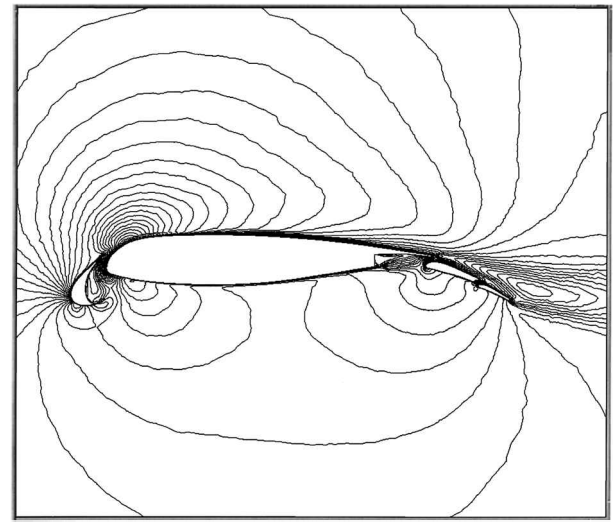

Fig. 23 Computed Mach contours on two-dimensional plane defined by spanwise station A for 16-deg incidence case on fine (24.7×10^6 point) grid.

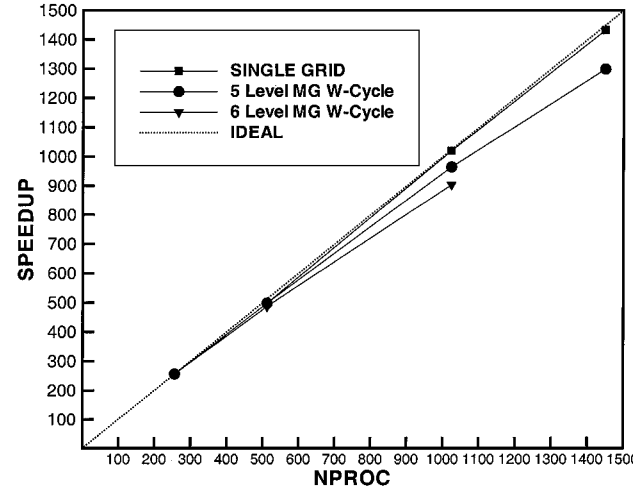

Fig. 24 Observed speedups for 24.7×10^6 point grid case on 1520-processor Cray T3E-1200e.

The 24.7×10^6 point case was run mostly on the Cray T3E-600 machine using 512 processors. This case requires 52 GB of memory, and 4.5 h for 500 multigrid cycles, which includes 30 min of I/O time to read the grid file (9 GB) and write the solution file (2 GB). A total of 18 different cases have been computed on this fine grid. Although these are large calculations, they are entirely feasible on existing production machines, such as the NASA Goddard Research Center T3E or the National Aerospace Simulation (NAS) 256-processor SGI Origin 2000. Furthermore, in a production mode, the solution times could be further reduced (by up to a factor of 2) by using solutions at previously computed angles of attack as the starting point for additional incidence cases and using fewer multigrid cycles.

During the course of this work, a unique opportunity was afforded to perform benchmark runs on a large Cray T3E-1200e machine. The Cray T3E-1200e contains 600-MHz DEC Alpha processors, as well as an upgraded communication chip, as compared to the aforementioned T3E-600 (300-MHz processors). This particular machine contained 1520 processors, each with a minimum of 256 MB per processor. Figure 24 shows the speedups obtained by the single grid and by the five-level and six-level multigrid runs on the 24.7×10^6 point grid running on 256, 512, 1024, and 1450 processors. The single-grid computations achieve almost prefect scalability up to 1450 processors, whereas the speedups achieved by the multigrid runs are only slightly below the ideal values. The six-level multigrid case could

Table 1 Timing and estimated computational rates for 24.7 $\times 10^6$ point grid case on various Cray T3E configurations[a]

Platform	Number of processors	Time/Cycle	GFLOP/s
T3E-600	512	28.1	22.0
T3E-1200e	256	38.3	16.1
T3E-1200e	512	19.7	31.4
T3E-1200e	1024	10.1	61.0
T3E-1200e	1450	7.54	82.0

[a]Five multigrid levels.

not be run on the maximum number of processors because the partitioning of the coarsest level resulted in empty processors with no grid points. Whereas this does not represent a fundamental problem, the software was not designed for such situations. In any case, the five-level multigrid runs are the most efficient overall because there is little difference in the convergence rate between the five- and six-level multigrid runs, as shown in Fig. 15. The single-grid results are included simply for comparison with the multigrid algorithm and are not used for actual computations because convergence is extremely slow.

The computation times are given in Table 1. Computational rates are obtained by linear scaling according to wall clock time with smaller problems run on the Cray C90 using the hardware performance monitor for million floating-point operations ratings. On 512 processors, the five-level multigrid case requires 19.7 s per cycle, as compared to 28.1 s per cycle on the 512-processor Cray T3E-600, which corresponds to an increase in speed of over 40% simply due to the faster individual processors. On 1450 processors, the same case required 7.54 s per cycle, or 63 min of computation for a 500-multigrid cycle run. A complete run required 92 min, which includes 29 min of I/O time. No attempt at optimizing I/O was made, and it is felt that substantial reductions in the I/O could be achieved by changes in both hardware and software configuration.

Because the 24.7×10^6 point grid was run on 256 processors of this machine, simple scaling arguments show that such a machine would be capable of solving over 100×10^6 grid points in several hours. However, for computations of this size, the bottleneck of sequential preprocessing must first be addressed.

X. Conclusions

A complete approach for simulating high-lift flows using highly resolved unstructured grids has been presented. Unstructured grid generation for complex high-lift geometries can be accomplished in a matter of days, and the flow solution can be achieved in several hours on existing production supercomputers. Good agreement between computed and experimental pressure coefficients and force coefficients as a function of angle of attack has been shown for a full aircraft configuration. Good drag prediction has been obtained, although the lift values and $Cl_{\max}$ location are slightly overpredicted. Good scalability of these computations has been demonstrated using up to 1450 processors, although the solution of substantially larger problems will require the resolution of bottlenecks in preprocessing, I/O time, and network file transfer.

Additional work is required to enhance the robustness of the low Mach number preconditioner particularly for fine grids. Accurate resolution of isolated flow features, such as slat wakes, will require more research into novel grid-generation techniques to increase resolution in these regions and/or increased use of adaptive meshing techniques.

Acknowledgments

This work was made possible thanks to dedicated computer time donated by Cray Research, Eagan, Minnesota. Special thanks are due to David Whitaker for his assistance, Leland Bigger for help with file archiving and transfer, and the support staff of the T3E-1200e.

References

[1]Barth, T. J., "Parallel CFD Algorithms on Unstructured Meshes," *Special Course on Parallel Computing in CFD*, Rept. 807, AGARD, May 1995, pp. 7-1, 7-41.

[2]Keyes, D. E., Kaushik, D. K., and Smith, B. F., "Prospects for CFD on Petaflops Systems," *CFD Review*, edited by M. Hafez and K. Oshima, Wiley, New York, 1997, pp. 1079–1096.

[3]Mavriplis, D. J., "Three Dimensional High-Lift Analysis Using a Parallel Unstructured Multigrid Solver," *Proceedings of the 16th AIAA Applied Aerodynamics Conference*, AIAA, Reston, VA, 1998; also AIAA Paper 98-2619, 1998, pp. 378–391.

[4]Lohner, R., and Parikh, P., "Generation of Three-Dimensional Unstructured Grids by the Advancing Front Method," *Journal of Numerical Methods in Fluids*, Vol. 8, 1988, pp. 1135–1149.

[5]Pirzadeh, S., "Three-Dimensional Unstructured Viscous Grids by the Advancing-Layers Method," *AIAA Journal*, Vol. 34, No. 1, 1996, pp. 43–49.

[6]Pirzadeh, S., "Structured Background Grids for Generation of Unstructured Grids by Advancing Front Method," *AIAA Journal*, Vol. 31, No. 2, 1993, pp. 257–265.

[7]Pirzadeh, S., "Progress Toward a User-Oriented Unstructured Viscous Grid Generator," AIAA Paper 96-0031, Jan. 1996.

[8]Mavriplis, D. J., "Multigrid Strategies for Viscous Flow Solvers on Anisotrpic Unstructured Meshes," *Proceedings of the 13th AIAA CFD Conference*, AIAA, Reston, VA, 1997, pp. 659–675; also AIAA Paper 97-1952, 1997.

[9]Aftosmis, M., Gaitonde, D., and Tavares, T. S., "On the Accuracy, Stability, and Monotonicity of Various Reconstruction Algorithms for Unstructured Meshes," AIAA Paper 94-0415, Jan. 1994.

[10]Lohner, R., "Finite-Element Methods in CFD: Grid Generation, Adaptivity and Parallelization," von Kármán Inst. Lecture Series, AGARD Publ. R-787, 1992.

[11]Mavriplis, D. J., "Adaptive Meshing Techniques for Viscous Flow Calculations on Mixed-Element Unstructured Meshes," AIAA Paper 97-0857, Jan. 1997.

[12]Mavriplis, D. J., and Venkatakrishnan, V., "A Unified Multigrid Solver for the Navier–Stokes Equations on Mixed Element Meshes," *International Journal for Computational Fluid Dynamics*, Vol. 8, 1997, pp. 247–263.

[13]van Leer, B., Tai, C. H., and Powell, K. G., "Design of Optimally Smoothing Multistage Schemes for the Euler Equations," AIAA Paper 89-1933, June 1989.

[14]Riemslagh, K., and Dick, E., "A Multigrid Method for Steady Euler Equations on Unstructured Adaptive Grids," *Proceedings of the 6th Copper Mountain Conference on Multigrid Methods*, NASA CP 3224, 1993, pp. 527–542.

[15]Morano, E., and Dervieux, A., "Looking for O(N) Navier–Stokes Solutions on Nonstructured Meshes," *Proceedings of the 6th Copper Mountain Conference on Multigrid Methods*, NASA CP 3224, 1993, pp. 449–464.

[16]Ollivier-Gooch, C., "Towards Problem-Independent Multigrid Convergence Rates for Unstructured Mesh Methods I: Inviscid and Laminar Flows," *Proceedings of the 6th International Symposium on Computational Fluid Dynamics*, 1995, pp. 913–930.

[17]Pierce, N., and Giles, M., "Preconditioning on Stretched Meshes," AIAA Paper 96-0889, Jan. 1996.

[18]Weiss, J. M., and Smith, W. A., "Preconditioning Applied to Variable and Constant Density Time-Accurate Flows on Unstructured Meshes," AIAA Paper 94-2209, June 1994.

[19]Turkel, E., "Preconditioning Methods for Solving the Incompressible and Low Speed Compressible Equations," *Journal of Computational Physics*, Vol. 72, No. 2, 1987, pp. 277–298.

[20]van Leer, B., Turkel, E., Tai, C. H., and Mesaros, L., "Local Preconditioning in a Stagnation Point," *Proceedings of the 12th AIAA CFD Conference*, AIAA, Washington, DC, 1995, pp. 88–101; also AIAA Paper 95-1654, 1995.

[21]Turkel, E., "Preconditioning-Squared Methods for Multidimensional Aerodynamics," *Proceedings of the 13th AIAA CFD Conference*, AIAA, Reston, VA, 1997, pp. 856–866; also AIAA Paper 97-2025, 1997.

[22]Spalart, P. R., and Allmaras, S. R., "A One-Equation Turbulence Model for Aerodynamic Flows," *La Recherche Aérospatiale*, Vol. 1, 1994, pp. 5–21.

[23]Lallemand, M., Steve, H., and Dervieux, A., "Unstructured Multigridding by Volume Agglomeration: Current Status," *Computers and Fluids*, Vol. 21, No. 3, 1992, pp. 397–433.

[24]Smith, W. A., "Multigrid Solution of Transonic Flow on Unstructured Grids," *Recent Advances and Applications in Computational Fluid Dynamics*, edited by O. Baysal, American Society of Mechanical Engineers, Fairfield, NJ, 1990.

[25]Mavriplis, D. J., "Directional Agglomeration Multigrid Techniques for High-Reynolds Number Viscous Flows," AIAA Paper 98-0612, Jan. 1998.

[26]Hendrickson, B., and Leland, R., "The Chaco User's Guide: Version 2.0," TR SAND94-2692, Sandia National Lab., Albuquerque, NM, July 1995.

[27]Karypis, G., and Kumar, V., "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs," TR 95-035, Univ. of Minnesota, Minneapolis, MN, 1995.

[28]Gropp, W., Lusk, E., and Skjellum, A., *Using MPI: Portable Parallel Programming with the Message Passing Interface*, MIT Press, Cambridge, MA, 1994.

[29]Mavriplis, D. J., Das, R., Saltz, J., and Vermeland, R. E., "Implementation of a Parallel Unstructured Euler Solver on Shared and Distributed Memory Machines," *Journal of Supercomputing*, Vol. 8, No. 4, 1995, pp. 329–344.

[30]"Special Course on Parallel Computing in CFD," AGARD Rept. 807, May 1995.

[31]Venkatakrishnan, V., "Implicit Schemes and Parallel Computing in Unstructured Grid CFD," von Kármán Inst. Lecture Series, VKI-LS 1995-02, 1995.

[32]Gatlin, G. M., and McGhee, R. J., "Experimental Investigation of Semispan Model Testing Techniques," *Journal of Aircraft*, Vol. 34, No. 4, 1997, pp. 500–505.

[33]Lynch, F. T., Potter, R. C., and Spaid, F. W., "Requirements for Effective High Lift CFD," *Proceedings of the 20th ICAS Congress*, International Council of the Aeronautical Sciences, 1996, pp. 1479–1492; also ICAS Paper 96-2.7.1, Sept. 1996.