AIAA 99–0537

LARGE-SCALE PARALLEL UNSTRUCTURED MESH COMPUTATIONS FOR 3D HIGH-LIFT ANALYSIS

D. J. Mavriplis
Institute for Computer Applications in Science and Engineering
MS 403, NASA Langley Research Center
Hampton, VA 23681-0001
S. Pirzadeh
Configuration Aerodynamics Branch
MS 499, NASA Langley Research Center
Hampton, VA 23681-0001

37th AIAA Aerospace Sciences Meeting
January 11-14 1999, Reno NV

LARGE-SCALE PARALLEL UNSTRUCTURED MESH COMPUTATIONS FOR 3D HIGH-LIFT ANALYSIS

D. J. Mavriplis
Institute for Computer Applications in Science and Engineering
MS 403, NASA Langley Research Center
Hampton, VA 23681-0001
S. Pirzadeh
Configuration Aerodynamics Branch
MS 499, NASA Langley Research Center
Hampton, VA 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 in order to enable rapid turnaround for complicated geomtries which 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 work-station, and subsequently 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 take-off configuration using up to 24.7 million 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.

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 which must be captured adequately in order 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 which 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.

Due to the inherent complexities of high-lift flows (both geometrical 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 to 10^8 grid points. This represents over an order of magnitude larger problems 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 were 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 million grid points for a full aircraft configuration. The approach involves the generation of multi-million 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.

Tetrahedral Mesh Generation

The computational grids employed in this paper were generated with the NASA Langley 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 which 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 Advancing-front and the Advancing-Layers methods 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 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" 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. First, 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 which 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

$$dz_{i+1} = dz_1 * [1. + a * (1. + b)^i]^i \qquad (1)$$

where dz_i is the normal spacing of the ith layer, dz_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 vary 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 multi-directional, 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

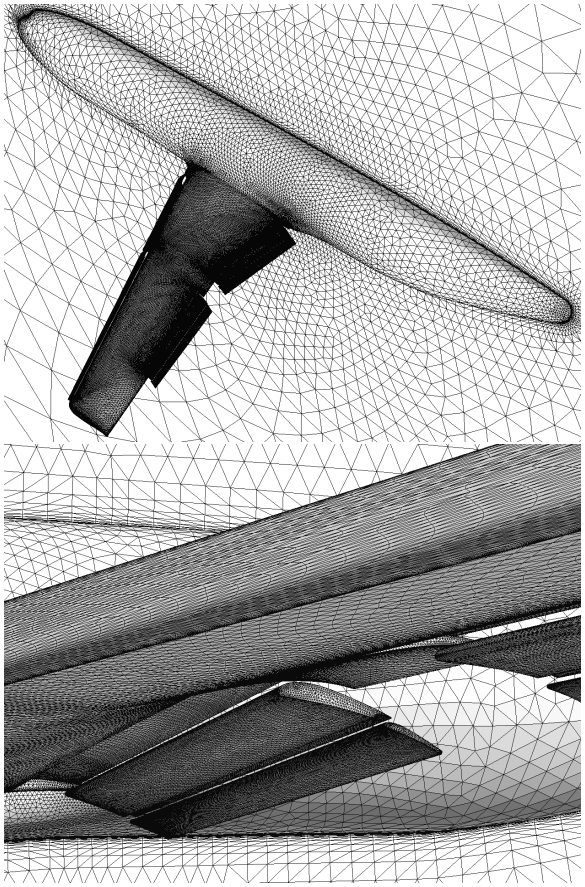

Fig. 1 Illustration of Surface Grid for High-Lift Configuration with Span-Wise Stretching Along Slat Leading Edges

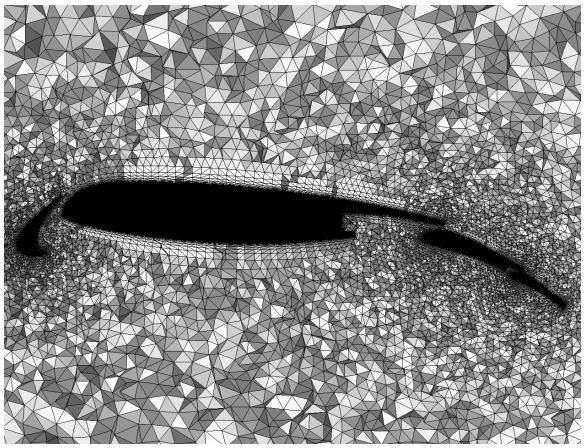

Fig. 2 Illustration of Two-Dimensional Cross-Section of Volume Grid at Spanwise Station Along Wing

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 three reduction in the total number of grid points.

For the grids presented in this paper, the parameters dz1 was set to 1.35E-06 wing chords, and the values of a and b in equation (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 in Figure 2 were a cross-section of the volume grid is depicted, showing tetrahedral cells around the multi-element airfoil geometry. Subdivision of this grid into 24.7 million points (as explained in the following sections) results in a doubling of the spatial resolution in all three coordinate directions.

The grids were generated using a Silicon Graphics Octane workstation with a 195 MHZ (R10000) processor. The entire process, from CAD definition to the final post-processed 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 hours using the workstation.

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. Firstly, for the vertex-based discretization employed herein, 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, since 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.

Secondly, 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 described below. 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 depicted in Figure 3. Because the height of the advancing layers is non-uniform 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 which borders on two tetrahedral neighbors as shown in Figure 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 re-subdivided into three tetrahedra. Prisms containing two compatible hanging diagonals are divided into one tetrahedron and one pyramid. In cases where this is not possible (i.e. 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 1 pyramid and 6 tetrahedra (for two hanging edges) or 2 pyramids and 4 tetrahedra (for one hanging edge). In practice, only a very small number of additional vertices are required.

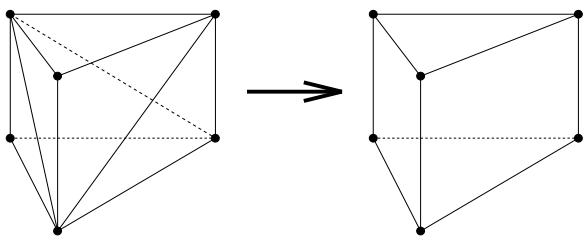

Fig. 3 Illustration of Merging of Three Suitable Tetrahedra into a Single Prismatic Element

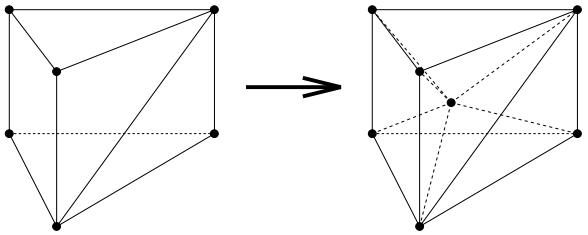

Fig. 4 Illustration of Insertion of New Vertex and Subdivision of Prismatic Element with one non-compatible Face Diagonal

Global Grid Refinement

The generation of and flow solution on large unstructured grids (i.e. $> 10^7$ grid points) is 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 super-computers. While 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, since these initial grids usually contain several million vertices (up to 20 million 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 previously. 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 8 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 million point grid through global refinement of the 3.1 million point grid described earlier required about 30 minutes 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 of the results. An interface for reprojecting new surface points onto the original surface geometry is currently under consideration.

Base Solver

The Reynolds averaged Navier-Stokes equations are discretized by a finite-volume technique on meshes of mixed element types which may include tetrahedra, pyramids, prisms, and hexahedra. All elements of the grid are handled by a single unifying edge-based datastructure 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 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 which 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 reference[8] , and then taking the corresponding linearization of these modified terms into account in the Jacobi preconditioner, a process sometimes referred to as *preconditioning*[2,8,21].

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.

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. While 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 previous 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 (i.e. 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 previous section is recovered. An example of the set of lines constructed from the two-dimensional unstructured grid in Figure 5 is depicted in Figure 6.

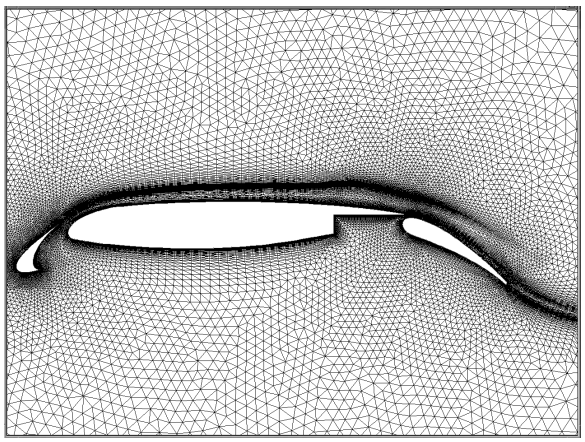

Fig. 5 Unstructured Grid for three-element airfoil; Number of Points = 61,104, Wall Resolution = 10^{-6} chords

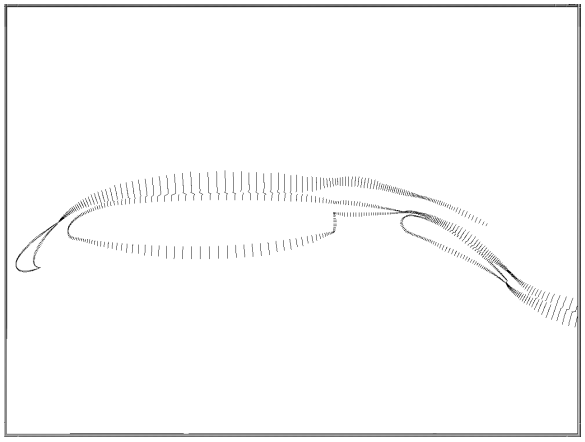

Fig. 6 Directional Implicit Lines Constructed on Grid of Figure 5 by Weighted Graph Algorithm

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, in order 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.[12] While 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]

Parallel Implementation

The unstructured multigrid solver is parallelized by partitioning the domain using a standard graph partitioner[26, 27] and communicating between the various grid partitions running on individual processors using the MPI message-passing 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 non-standard aspects of the present implementation which 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. Since 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. While this would minimize the communication in the inter-grid transfer routines, it affords little control over the quality of the coarse grid partitions. Since the amount of intra-grid computation on each level is much more important than the inter-grid 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. In order to minimize inter-grid 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.

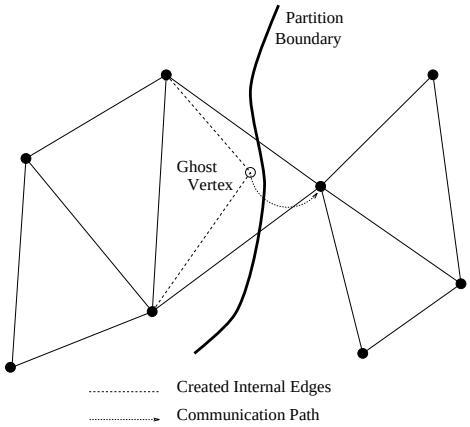

Fig. 7 Illustration of Creation of Internal Edges and Ghost Points at Inter-processor Boundaries

For each partitioned level, the edges of the mesh which 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 (c.f. Figure 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 in order 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.

The use of line-solvers can lead to additional complications for distributed-memory parallel implementations. Since the classical tridiagonal line-solve is an inherently sequential operation, any line which 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 a 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 which are intersected by the partition boundaries.

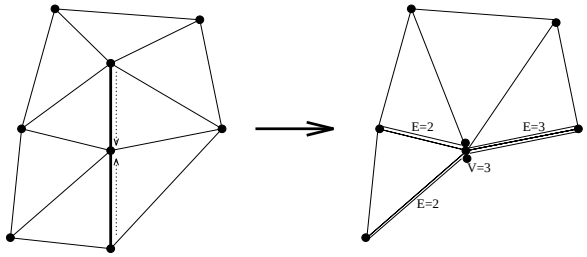

Fig. 8 Illustration of Line Edge Contraction and Creation of Weighted Graph for Mesh Partitioning; V and E Values Denote Vertex and Edge Weights Respectively

In order to avoid partitioning across implicit lines, the original unweighted graph (set of vertices and edges) which 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 which are joined by an edge which is part of an implicit line are then merged together to form a new vertex. Merging vertices also produce merged edges as shown in Figure 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 partitioner described in references[26, 27] , and the resulting partitioned graph is then de-contracted, i.e. all constituent vertices of a merged vertex are assigned the partition number of that vertex. Since 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.

As an example, the two dimensional mesh in Figure 5, which contains the implicit lines depicted in Figure 6, has been partitioned both in its original unweighted uncontracted form, and by the graph contraction method described above. Figure 9 depicts 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), while the weighted partition contains no intersected line edges and a total of 5883 cut edges (3.2 % of total), i.e. a 23% increase over the total number of cut edges in the non-weighted partition.

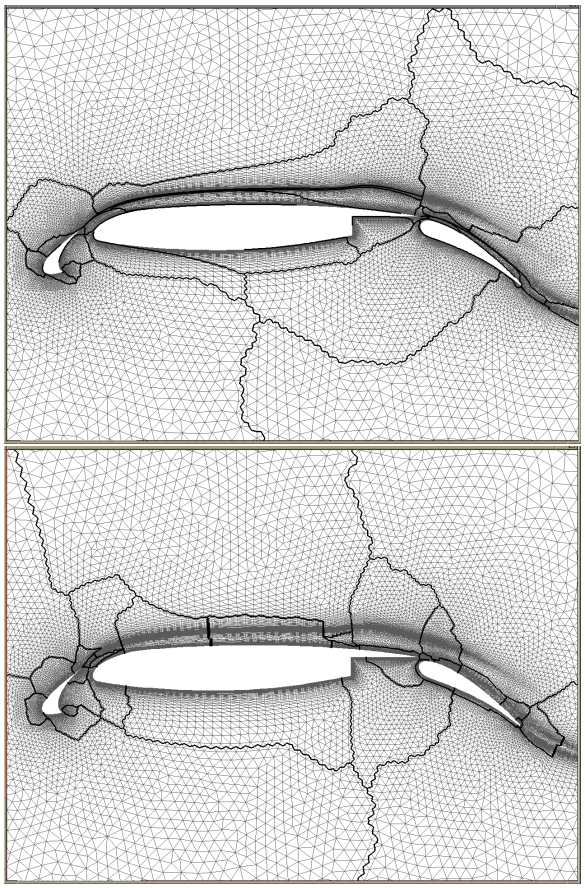

Fig. 9 Comparison of Unweighted (above) and Weighted (below) 32-Way Partition of Two-Dimensional Mesh

Due to the large size of the grids considered in this work, 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 inter-processor 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, i.e. 1 Kbyte 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 using large portions of the memory of the entire machine. For example, the various preprocessing operations for a 24.7 million point grid required between 10 to 20 Gbytes of memory and between 45 minutes to 90 minutes for each of the operations mentioned above. 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 above run times also apply to the CHACO partitioner. The exception was the MeTiS partitioner, which managed to partition the 24.7 million point grid in approximately 10 minutes, about 4 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.

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 Mbytes of memory per processor, for an aggregate memory capacity of 36.6 Gbytes. The CRAY T3E contains 512 DEC Alpha 300 MHz processors with 128 Mbytes of memory per processor, for an aggregate memory capacity of 65 Gbytes.

The test case involves a grid of 1.98 million points over an ONERA M6 wing at a Mach number of 0.1, an incidence of 2.0 degrees, and a Reynolds number of 3 million and is reproduce from reference[3] . Figures 10 and 11 show the relative speedups achieved on the two target hardware platforms for this case. For the purposes of these figures, 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 (non-multigrid) algorithm, the multigrid algorithm using a V-cycle, and the multigrid algorithm using a W-cycle.

The figures reveal good scalability on both machines up to the maximum number of processors. The better scalability of the single grid versus the multigrid algorithms is indicative of the increased volume of communication generated by the coarse grid-level timestepping 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.

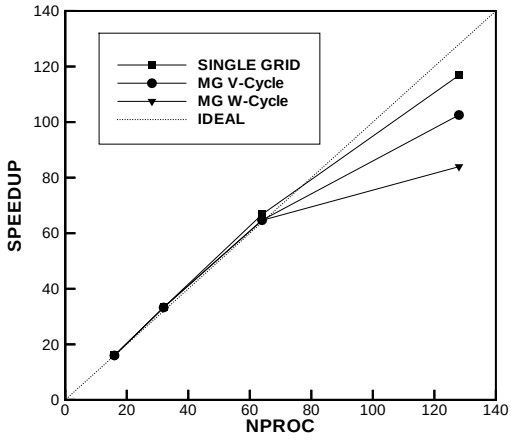

Fig. 10 Observed Speedups for ONERA M6 Wing Case (1.98 million grid points) on SGI Origin 2000

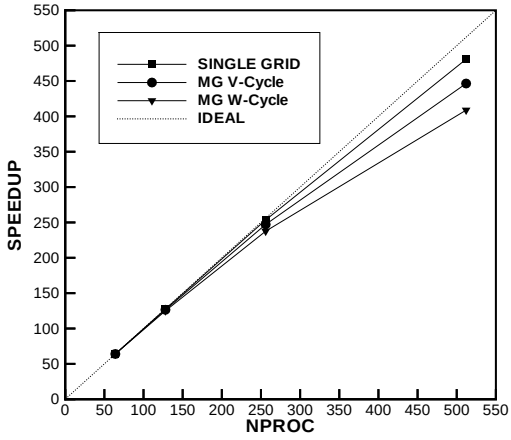

Fig. 11 Observed Speedups for ONERA M6 Wing Case (1.98 million grid points) on CRAY T3E

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 energy efficient transport (EET) configuration, which has been tested both as a full span and semi-span model in the NASA Langley $14 \times 22ft$ 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 degrees, and consists of a super-critical 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 degrees, a vane deflection of 15 degrees,

and a flap deflection of 30 degrees, with respect to the main airfoil. The freestream Mach number is 0.2, and the Reynolds number is 1.6 million based on the wing reference chord, and the experimental flow incidence varies over a range of -4 degrees up to 24 degrees.

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 degrees incidence were accessible for comparison. While reference[32] describes the experiments on a semi-span model with pylon and nacelle, the full-span "nacelle-off" results used for comparison in this paper have not been previously published.

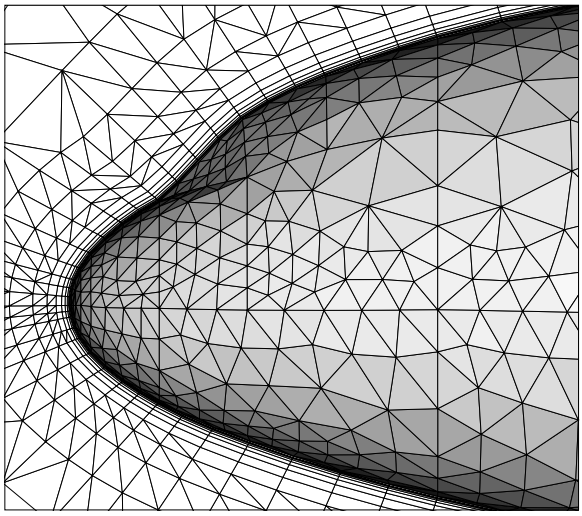

Fig. 12 Illustration of Mixed Prismatic-Tetrahedral Elements near Fuselage Nose Region for 3.1 million Point Grid

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 million vertices and 18.2 million 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 hours for the actual surface and volume grid generation. This tetrahedral grid was then merged into a mixed-element grid of 3.9 million prisms, 6.6 million 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 million vertices, with 53 million tetrahedra, 31 million prisms, and 281,000 pyramids. The refinement operation was performed sequentially on a single processor of an SGI ORIGIN 2000, and required approximately 10 Gbytes of memory and 30 minutes of CPU time.

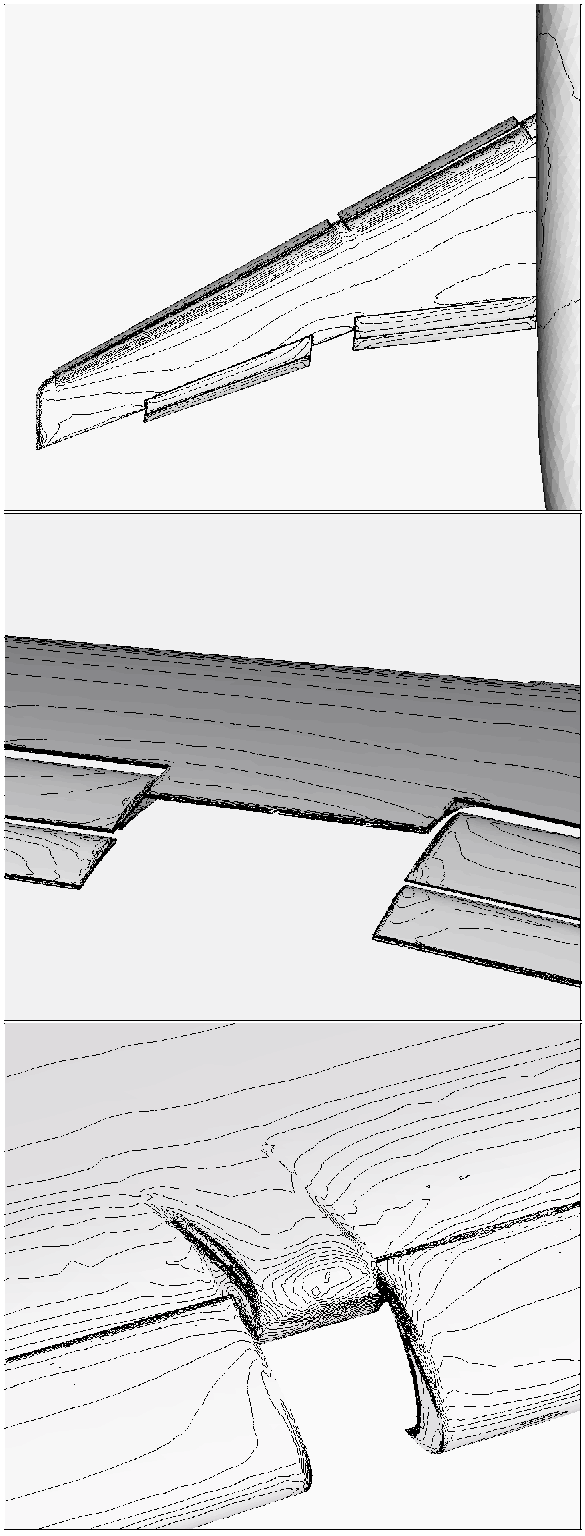

Fig. 13 Illustration of Computed Pressure Contours on Wing, Slat and Flap Corner Areas on 3.1 million point Mesh; Mach = 0.2, Incidence = 10 degrees, Re = 1.6 million

Figure 12 illustrates the mixed prism-tetrahedral nature of the 3.1 million point grid in the vicinity of the nose of the fuselage on the symmetry plane. An illustration of the computed solution at 10 degrees incidence on the 3.1 million point grid is given in Figure 13 as a set of surface pressure contours on the wing and flaps.

The preconditioned line-implicit agglomeration multigrid algorithm was employed to converge the solution to a steady-state. The isotropic agglomeration routine was used in order 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 million point case, with the coarsest level containing only 1,651 vertices. The convergence history on this grid for a flow incidence of 10 degrees is depicted in Figure 14. A residual reduction of four 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 cycles, so that an engineering results could be obtained with confidence in about 300 cycles.

For the 24.7 million point grid case, convergence was hindered by the behavior of the low-Mach number preconditioner. Figure 15 depicts 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 non-preconditioned case converges four orders of magnitude in 500 cycles. In fact, the convergence history for the non-preconditioned case is nearly identical to that obtained on the coarser grid, as shown in Figure 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 million point grid case (which contains only 718 points, versus 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.

While 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 non-preconditioned case, achieving a deviation smaller than 0.1 % of its final value in just 110 multigrid cycles, as opposed to 260 cycles for the non-preconditioned 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 by 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 million 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.

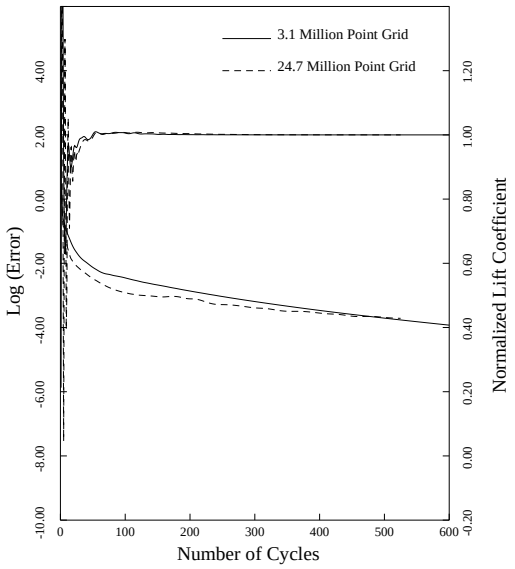

Fig. 14 Convergence Rate for Coarse (3.1 million pt) Grid using 5 Multigrid Levels and Low Mach Number Preconditioning and Fine (24.7 million pt) Grid using 6 Multigrid Levels and no Preconditioning at 0.2 Mach Number and 10 degress Incidence

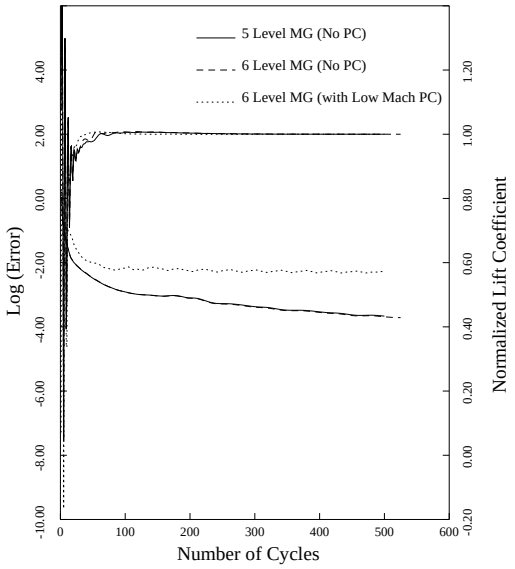

Fig. 15 Convergence Rate for Fine (24.7 million pt) Grid using 5 and 6 Multigrid Levels with and without Low-Mach Number Preconditioning at 0.2 Mach Number and 10 degress Incidence

Figures 17 through 19 provides a comparison of the computed surface pressures on the coarse and fine grids with experimental values at the three spanwise locations illustrated in Figure 16 for an incidence of 10 degrees. For the fine grid, the preconditioned and non-preconditioned 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.

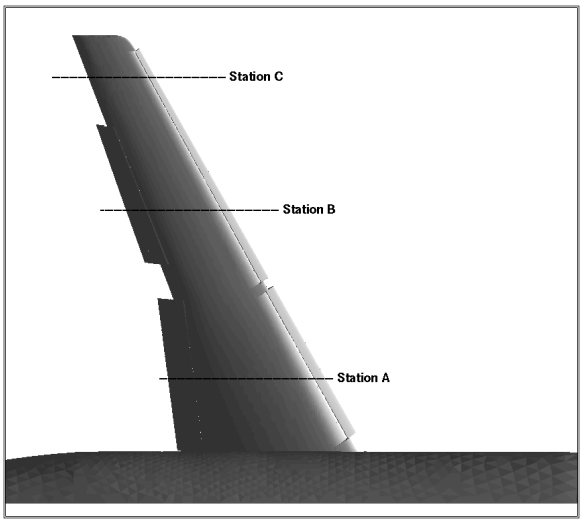

Fig. 16 Definition of Three Spanwise Stations for Comparison of Computed and Experimental Pressure Distributions

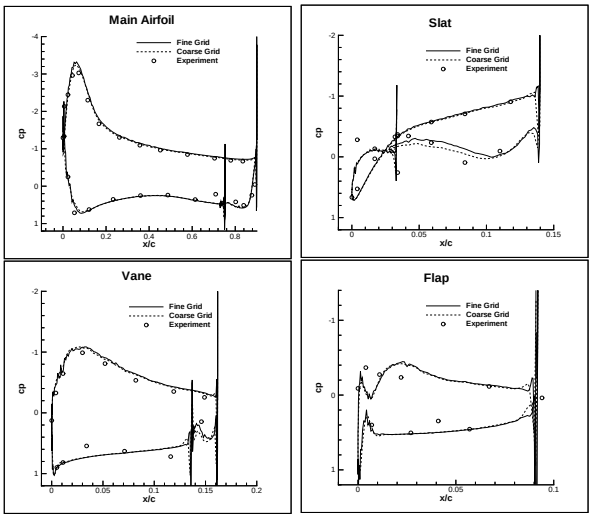

Fig. 17 Comparison of Computed and Experimental Surface Pressure Distributions at Spanwise Station A for 10 Degrees Incidence Case

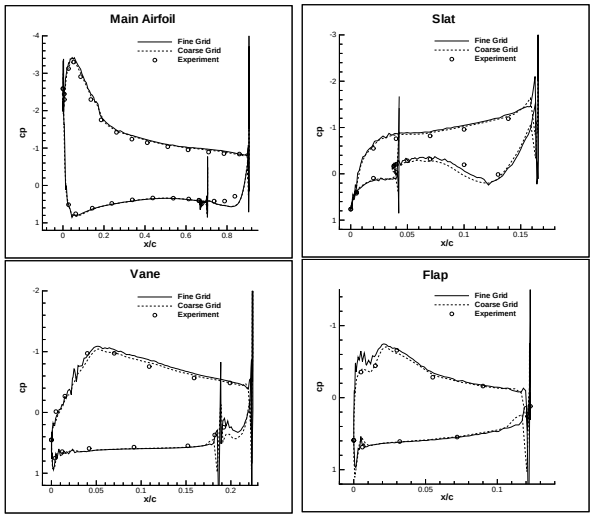

Fig. 18 Comparison of Computed and Experimental Surface Pressure Distributions at Spanwise Station B for 10 Degrees Incidence Case

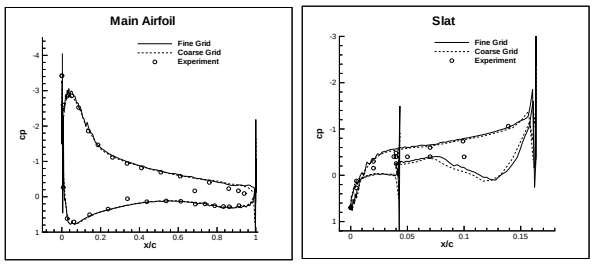

Fig. 19 Comparison of Computed and Experimental Surface Pressure Distributions at Spanwise Station C for 10 Degrees Incidence Case

A comparison between computed and experimental lift coefficients as a function of angle of attack is given in Figure 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 over-predicted 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 degrees incidence, is well predicted by the coarse grid. The fine grid over-predicts the maximum lift incidence by 1 degree, giving a value of 17 degrees. In both cases the value of Cl_{max} is over-predicted. These 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 the figure. It is interesting to note that post-stall computed values averaged in this manner follow the experimental values fairly closely, although the range of computed min-max values is rather large.

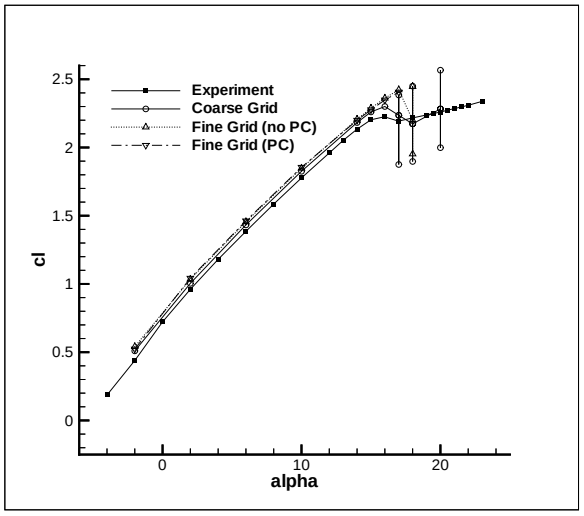

Fig. 20 Comparison of Computed and Experimental Lift Coefficients as a Function of Angle of Attack

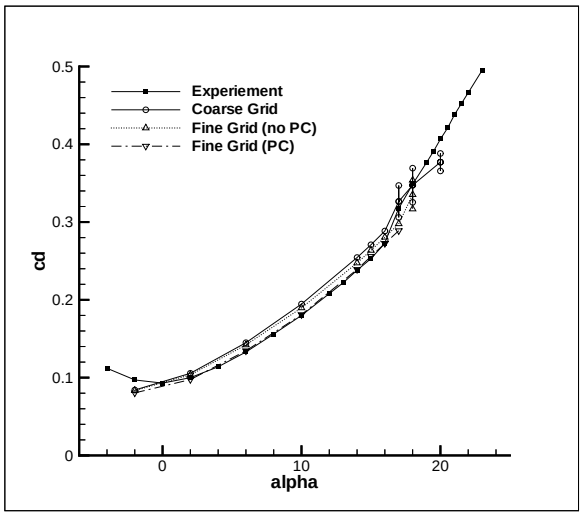

Fig. 21 Comparison of Computed and Experimental Drag Coefficients as a Function of Angle of Attack

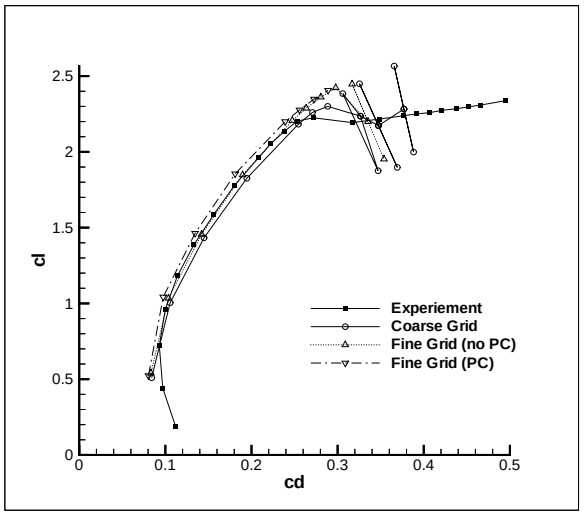

Fig. 22 Comparison of Computed and Experimental Drag Polars

The fact 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 versus angle of attack, while the corresponding drag polar plots are given in Figure 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 non-negligible effect on the drag values, providing even closer agreement with experiment for the fine grid case than the un-preconditioned case. In fact the agreement with experiment is very close. For example, at 10 degrees incidence, the fine grid preconditioned value is 0.1810. while 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 upon, or is simply fortuitous.

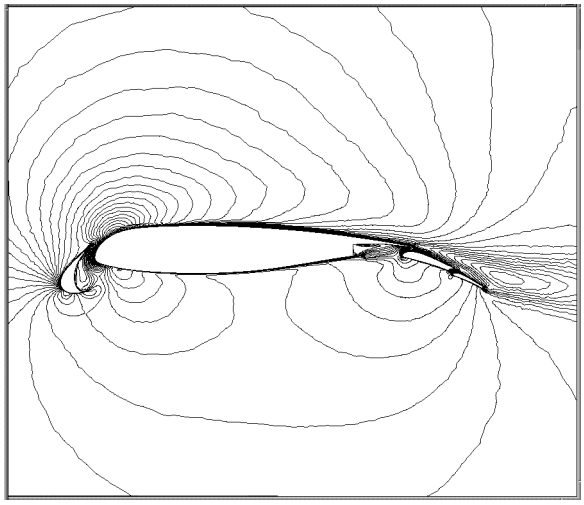

Fig. 23 Computed Mach Contours on Two-Dimensional Plane Defined by Spanwise Station A for 16 degrees Incidence Case on Fine (24.7 million point) Grid

A two dimensional cut of the computed Mach contours on the fine grid is depicted in Figure 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, since there has been no effort to anisotropically increase grid resolution in these areas.

The 3.1 million 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 illustrated in Figures 10 and 11. This case requires a total of 7 Gbytes of memory and 80 minutes

on 128 processors (250 MHz) of the ORIGIN 2000, or 62 minutes on 256 processors of the Cray T3E-600 for a 500 multigrid cycle run.

The 24.7 million point case was run mostly on the Cray T3E-600 machine using 512 processors. This case requires 52 Gbytes of memory, and 4.5 hours for 500 multigrid cycles, which includes 30 minutes of I/O time to read the grid file (9 Gbytes), and write the solution file (2 Gbytes). 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 T3E, or the 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.

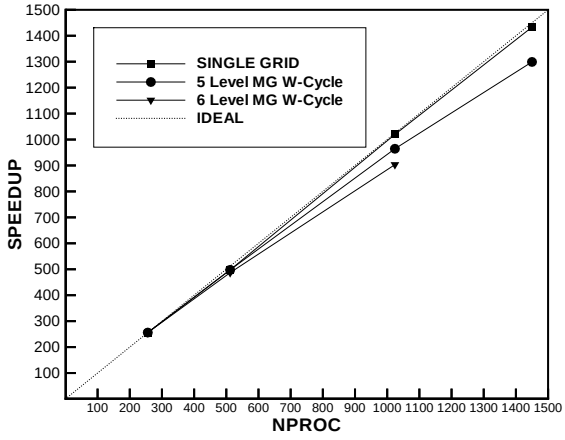

Fig. 24 Observed Speedups for 24.7 million point Grid Case on 1520 Processor CRAY T3E-1200e

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 previously mentioned T3E-600 (300MHz processors). This particular machine contained 1520 processors each with a minimum of 256 Mbytes per processor. Figure 24 depicts the speedups obtained by the single grid, and the five level and six level multigrid runs on the 24.7 million point grid running on 256, 512, 1024 and 1450 processors. The single grid computations achieve almost prefect scalability up to 1450 processors, while the speedups achieved by the multigrid runs are only slightly below the ideal values. The six level multigrid case could not be run on the maximum number of processors, since the partitioning of the coarsest level resulted in empty processors with no grid points. While 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, since there is little difference in the convergence rate between the five and six level multigrid runs, as shown in Figure 15. The single grid results are included simply for comparison with the multigrid algorithm, and are not used for actual computations since convergence is extremely slow.

24.7 Million Pt Case (5 Multigrid Levels)			
Platform	No. of Procs	Time/Cyc	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

Table 1 Timings and Estimated Computational Rates for 24.7 million point Grid Case on Various Cray T3E Configurations; 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 Mflop ratings

The computation times are depicted in Table 1. On 512 processors, the 5 level multigrid case requires 19.7 seconds per cycle, as compared to 28.1 seconds 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 seconds per cycle, or 63 minutes of computation for a 500 multigrid cycle run. A complete run required 92 minutes, which includes 29 minutes 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.

Since the 24.7 million 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 million grid points in several hours. However, for computations of this size, the bottleneck of sequential preprocessing must first be addressed.

Conclusions and Further Work

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,

while the lift values and Cl_{max} location are slightly over-predicted. 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 in order to increase resolution in these regions, and/or increased use of adaptive meshing techniques.

Acknowledgements

This work was made possible thanks to dedicated computer time donated by Cray Research, Eagan MN. 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]T. J. Barth. Parallel CFD algorithms on unstructured meshes. In *Special Course on Parallel Computing in CFD*, pages 7–1,7–41, May 1995. AGARD Report-807.

[2]D. E. Keyes, D. K. Kaushik, and B. F. Smith. Prospects for CFD on petaflops systems. In *CFD Review M. Hafez, et. al., eds., Wiley, New York*, 1997.

[3]D. J. Mavriplis. Threee dimensional high-lift analysis using a parallel unstructured multigrid solver. In *Proceedings of the 16th AIAA Applied Aerodynamics Conference, Albuquerque, NM*, June 1998. AIAA paper 98-2619-CP.

[4]R. Lohner and P. Parikh. Generation of three-dimensional unstructured grids by the advancing front method. *Journal of Numerical Methods in Fluids*, 8:1135–1149, 1988.

[5]S. Pirzadeh. Three-dimensional unstructured viscous grids by the advancing-layers method. *AIAA Journal*, 34(1):43–49, 1996.

[6]S. Pirzadeh. Structured background grids for generation of unstructured grids by advancing front method. *AIAA Journal*, 31(2):257–265, 1993.

[7]S. Pirzadeh. Progress toward a user-oriented unstructured viscous grid generator. AIAA Paper 96-0031, January 1996.

[8]D. J. Mavriplis. Multigrid strategies for viscous flow solvers on anisotrpic unstructured meshes. In *Proceedings of the 13th AIAA CFD Conference, Snowmass, CO*, pages 659–675, June 1997. AIAA Paper 97-1952-CP.

[9]M. Aftosmis, D. Gaitonde, and T. S. Tavares. On the accuracy, stability and monotonicity of various reconstruction algorithms for unstructured meshes. AIAA Paper 94-0415, January 1994.

[10]R. Lohner. Finite-element methods in CFD: Grid generation, adaptivity and parallelization. In *von Karman Institute Lecture Series*, AGARD Pub. R-787, 1992.

[11]D. J. Mavriplis. Adaptive meshing techniques for viscous flow calculations on mixed-element unstructured meshes. AIAA paper 97-0857, January 1997.

[12]D. J. Mavriplis and V. Venkatakrishnan. A unified multigrid solver for the Navier-Stokes equations on mixed element meshes. *International Journal for Computational Fluid Dynamics*, 8:247–263, 1997.

[13]B. van Leer, C. H. Tai, and K. G. Powell. Design of optimally-smoothing multi-stage schemes for the Euler equations. AIAA Paper 89-1933, June 1989.

[14]K. Riemslagh and E. Dick. A multigrid method for steady Euler equations on unstructured adaptive grids. In *6th Copper Mountain Conf. on Multigrid Methods, NASA conference publication 3224*, pages 527–542, 1993.

[15]E. Morano and A. Dervieux. Looking for O(N) Navier-Stokes solutions on non-structured meshes. In *6th Copper Mountain Conf. on Multigrid Methods*, pages 449–464, 1993. NASA Conference Publication 3224.

[16]C. Ollivier-Gooch. Towards problem-independent multigrid convergence rates for unstructured mesh methods i: Inviscid and laminar flows. In *Proceedings of the 6th International Symposium on CFD, Lake Tahoe, NV*, September 1995.

[17]N. Pierce and M. Giles. Preconditioning on stretched meshes. AIAA paper 96-0889, January 1996.

[18]J. M. Weiss and W. A Smith. Preconditioning applied to variable and constant density time-accurate flows on unstructured meshes. AIAA Paper 94-2209, June 1994.

[19]E. Turkel. Preconditioning methods for solving the incompressible and low speed compressible equations. *J. Comp. Phys.*, 72:277–298, 1987.

[20]B. van Leer E. Turkel C. H. Tai and L. Mesaros. Local preconditioning in a stagnation point. In *Proceedings of the 12th AIAA CFD Conference, San Diego, CA*, pages 88–101, June 1995. AIAA Paper 95-1654-CP.

[21]E. Turkel. Preconditioning-squared methods for multidimensional aerodynamics. In *Proceedings of the 13th AIAA CFD Conference, Snowmass, CO*, pages 856–866, June 1997. AIAA Paper 97-2025-CP.

[22]P. R. Spalart and S. R. Allmaras. A one-equation turbulence model for aerodynamic flows. *La Recherche Aérospatiale*, 1:5–21, 1994.

[23]M. Lallemand, H. Steve, and A. Dervieux. Unstructured multigridding by volume agglomeration: Current status. *Computers and Fluids*, 21(3):397–433, 1992.

[24]W. A. Smith. Multigrid solution of transonic flow on unstructured grids. In *Recent Advances and Applications in Computational Fluid Dynamics*, November 1990. Proceedings of the ASME Winter Annual Meeting, Ed. O. Baysal.

[25]D. J. Mavriplis. Directional agglomeration multigrid techniques for high-Reynolds number viscous flows. AIAA paper 98-0612, January 1998.

[26]B. Hendrickson and R. Leland. The Chaco user's guide: Version 2.0. Tech. Rep. SAND94-2692, Sandia National Laboratories, Albuquerque, NM, July 1995.

[27]G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. Technical Report Technical Report 95-035, University of Minnesota, 1995.

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

[29]D. J. Mavriplis, R. Das, J. Saltz, and R. E. Vermeland. Implementation of a parallel unstructured Euler solver on shared and distributed memory machines. *The J. of Supercomputing*, 8(4):329–344, 1995.

[30]Special course on parallel computing in CFD. May 1995. AGARD Report-807.

[31]V. Venkatakrishnan. Implicit schemes and parallel computing in unstructured grid CFD. In *VKI Lecture Series VKI-LS 1995-02*, March 1995.

[32]G. M. Gatlin and R. J. McGhee. Experimental investigation of semispan model testing techniques. *AIAA Journal of Aircraft*, 34(4):500–505, 1997.

[33]F. T. Lynch R. C. Potter and F. W. Spaid. Requirements for effective high lift CFD. In *Proceedings of the 20th ICAS Congress, Sorrento, Italy*, pages 1479–1492, September 1996. paper ICAS-96-2.7.1.