AIAA JOURNAL
Vol. 38, No. 11, November 2000

# Viscous Flow Analysis Using a Parallel Unstructured Multigrid Solver

Dimitri J. Mavriplis*
*NASA Langley Research Center, Hampton, Virginia 23681-0001*

**A directional implicit unstructured agglomeration multigrid solver is ported to shared and distributed memory massively parallel machines using the explicit domain-decomposition and message-passing approach. Because the algorithm operates on local implicit lines in the unstructured mesh, special care is required in partitioning the problem for parallel computing. A weighted partitioning strategy is described that avoids breaking the implicit lines across processor boundaries, while incurring minimal additional communication overhead. Good scalability is demonstrated on a 128 processor SGI Origin 2000 machine and on a 512 processor Cray T3E machine for reasonably fine grids. The feasibility of performing large-scale unstructured grid calculations with the parallel multigrid algorithm is demonstrated by computing the flow over a partial-span flap wing high-lift geometry on a highly resolved grid of $13.5 \times 10^6$ points in approximately 4 h of wall clock time on the Cray T3E.**

## I. Introduction

THE use of Reynolds-averaged Navier–Stokes analysis methods is rapidly becoming commonplace in the aircraft design process. Such methods require the use of highly stretched grids in the thin boundary-layer and wake regions, as well as a large overall number of grid points to resolve the flow physics adequately. Unstructured grid techniques offer the potential for greatly reducing the grid generation time associated with complex geometries. Several unstructured grid generation packages suitable for viscous flow solvers have been demonstrated recently.[1–5] Furthermore, unstructured mesh approaches enable the use of adaptive meshing techniques, which hold great promise for increasing solution accuracy at minimal additional computational cost. On the other hand, unstructured mesh solvers require significantly higher computational resources than their structured grid counterparts. In particular, the large memory requirements of unstructured mesh solvers have made three-dimensional high-lift analysis, which already strains the capabilities of structured grid solvers, impractical with unstructured grids.

One of the often overlooked advantages of unstructured mesh approaches is their superior scalability on massively parallel computer architectures. In contrast with block-structured or overset-mesh methods, the data structures used in unstructured mesh methods are homogeneous across the entire computational domain, thus enabling near perfect load balancing and good overall scalability on parallel machines, even for small problems on large numbers of processors.

Although the cost of specialized memory used in vector computer architectures has changed very little over the last several years, the cost of commodity memory used in parallel microprocessor-based machines has dropped dramatically, leading to the appearance of parallel machines with large shared or distributed memory systems. This development raises the possibility of performing large-scale unstructured-mesh computations suitable for full configuration analysis on such machines. The purpose of this paper is to demonstrate the feasibility of performing such calculations by porting a previously developed low-memory directional-implicit agglomeration-multigrid algorithm[6,7] to modern parallel computer architectures.

## II. 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. In general, prismatic elements are used in the boundary-layer and wake regions, whereas tetrahedra are used in the regions of inviscid flow. All elements of the grid are handled by a single unifying edge-based data structure in the flow solver.[8]

The governing equations are discretized using a central difference finite volume technique with added matrix-based artificial dissipation. The matrix dissipation approximates a Roe Riemann-solver-based upwind scheme,[9] but relies on a biharmonic operator to achieve second-order accuracy, rather than on a gradient-based extrapolation strategy.[6] 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. This is acheived with an edge-based loop using predetermined edge coefficients on the fine mesh level, and agglomerated edge coefficients on the coarser levels. For multigrid calculations, a first-order discretization is employed for the convective terms on the coarse grid levels for all cases.

The basic time-stepping scheme is a three-stage explicit multistage scheme with stage coefficients (0.1918, 0.4929, 1.0) optimized for high-frequency damping properties[10] and a Courant–Friedrichs–Lewy number of 1.8. Convergence is accelerated by a local block Jacobi preconditioner, which involves inverting a $5 \times 5$ matrix for each vertex at each stage.[11–14] A low-Mach-number preconditioner[15–17] 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 Ref. 6 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. 6 and 18).

The single equation turbulence model of Spalart and Allmaras[19] is utilized to account for turbulence effects. This equation is discretized and solved using multigrid in a manner completely analogous to the flow equations, with the exception that the convective terms are only discretized to first-order accuracy and the production term is only computed on the fine grid and restricted onto the coarser grid levels.

## III. Directional-Implicit Multigrid Algorithm

An agglomeration multigrid algorithm[8,20,21] 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. Whereas multigrid methods deliver 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

Presented as Paper 98-2619 at the 16th Applied Aerodynamics Conference, Albuquerque, NM, 15–18 June 1998; received 30 October 1998; revision received 15 May 2000; accepted for publication 30 May 2000. 

*Research Fellow, Institute for Computer Applications in Science and Engineering, Mail Stop 132C; dimitri@icase.edu. Member AIAA.

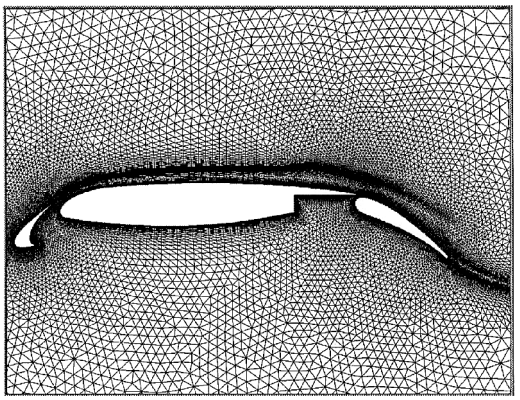


**Fig. 1 Unstructured grid for three-element airfoil; number of points = 61,104, wall resolution = $10^{-6}$ chords.**

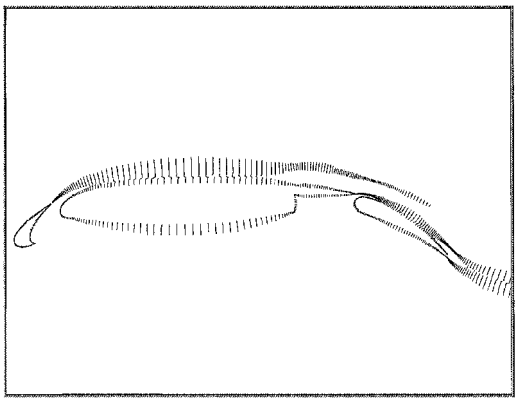


**Fig. 2 Directional implicit lines constructed on grid of Fig. 1 by weighted graph algorithm.**

coarsening techniques[6,7] 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 control volume face areas associated with each mesh edge. 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. 1 is shown in Fig. 2.

In addition to using a directional smoother, the agglomeration multigrid algorithm must be modified to take into account the effect of mesh stretching. The unweighted agglomeration algorithm that groups together all neighboring control volumes for a given fine grid vertex[8] is replaced with a weighted coarsening algorithm that only agglomerates the neighboring control volumes that are the most strongly connected to the current fine grid control volume, as determined by the same edge weights used in the line construction algorithm. This effectively results in semicoarsening-type behavior in regions of large mesh stretching and regular coarsening in regions of isotropic mesh cells. To maintain favorable coarse grid complexity, an aggressive coarsening strategy is used in anisotropic regions, where for every retained coarse grid point, three fine grid control volumes are agglomerated, resulting in an overall complexity reduction of 4:1 for the coarser levels in these regions, rather than the 2:1 reduction typically observed for semicoarsening techniques. In isotropic regions an 8:1 coarsening ratio is obtained. However, because most of the mesh points reside in the boundary-layer regions, the overall coarsening ratios achieved between grid levels is only slightly higher than 4:1. An example of the first directionally agglomerated level on a two-dimensional mesh is shown in Fig. 3

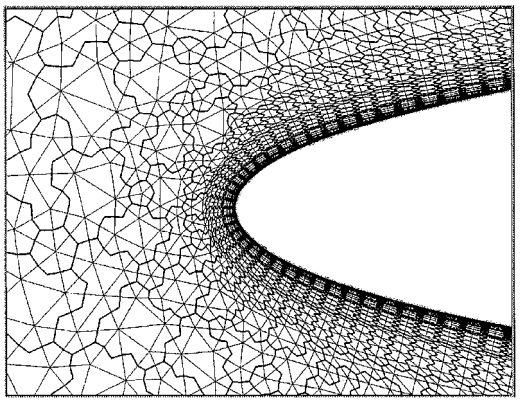


**Fig. 3 First agglomerated multigrid level for two-dimensional unstructured grid illustrating 4:1 directional coarsening in boundary-layer region (taken from Ref. 22).**

(from Ref. 22), where the aggressive agglomeration normal to the boundary layer is observed.

## IV. Aspects Related to Parallel Computer Architectures

For the purposes of this paper, parallel computer architectures can be classified by the configuration of their memory systems. These include multiprocessors with uniform shared-memory access, cache-based processors with uniform (out-of-cache) shared-memory access, cache-based processors with nonuniform (out-of-cache) shared-memory access, and cache-based processors with distributed memory.

In the following we consider the computational model of an unstructured grid solver as an edge-based loop updating vertex-based values. For shared-memory architectures, parallelization can be achieved simply by reordering and grouping the edges of the loop, whereas for distributed-memory systems, explicit domain decomposition and message passing must be performed. Traditionally, only vector machines have been available with uniform shared-memory access. On these machines, vectorization is achieved by sorting the edges of the mesh into groups (colors), such that within each group no two edges access the same vertex. This enables vectorization of the edge loop over each color. Each color can then itself be split into multiple subgroups, each of which is assigned to a particular processor. This approach is simple to implement and has been shown to work well on moderate numbers of processors, delivering speedups of up to 13 on 16 processors of the Cray C90 (Ref. 23). Because each edge color extends across the entire mesh, this approach is only effective when a uniform memory access is available.

In the case of cache-based processors, locality becomes important, and alternative ordering strategies such as those suggested in Ref. 24 are required. In these methods, vertices and edges are reordered for locality and separated into groups, which can then be processed in parallel. This strategy can be used on cache-based machines with uniform shared-memory access such as SGI Power Challenge architectures, as well as on cache-based machines with nonuniform shared-memory access, such as SGI Origin 2000 architectures. In the latter case, the memory architecture is shared, in that it is globally addressable, but the access is nonuniform, in that it is physically distributed, and references to off-processor memory locations are more expensive than references to local processor memory locations. For such memory architectures, (which can be viewed as an additional cache level), data locality is even more important for performance than for cache-based uniform shared-memory architectures. Locality ordering strategies are attractive because they can be implemented through a relatively simple run-time data reordering operation, for any solver that already makes use of edge groupings (as for earlier vectorized codes). This approach works relatively well for small numbers of processors but has been found to scale poorly for larger numbers of processors.

For distributed-memory architectures, memory is not globally addressable, and an explicit domain-decomposition/message-passing implementation must be performed. Unstructured mesh domain

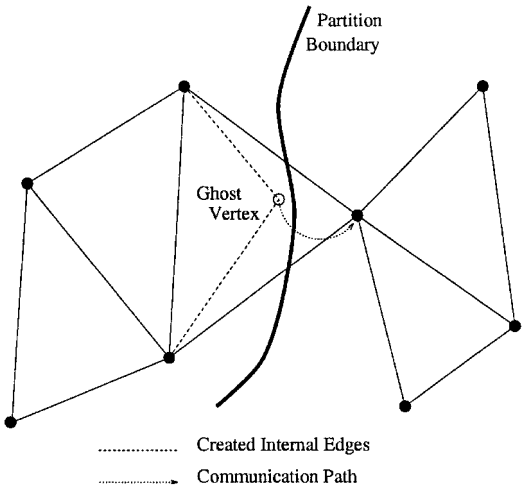



**Fig. 4 Creation of internal edges and ghost points at interprocessor boundaries.**

decomposition or partitioning can be accomplished by a number of well-documented and available partitioning strategies, which attempt to maintain load balancing and reduce communication volume by producing cuts that minimize the total number of edges that are intersected by the partition separators.[25–27] The partitioned data must also be optimized for cache efficiency by reordering the vertices and edges in each partition using a bandwidth minimization technique, such as the Cuthill–McKee[28] reordering algorithm applied locally in each partition.

Interprocessor communication is generated by mesh edges that straddle two adjacent mesh partitions. Such edges are assigned to one of the partitions, and a ghost vertex is constructed in this partition that corresponds to the vertex originally accessed by the edge in the adjacent partition, as shown in Fig. 4. 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 neighboring partitions to obtain the complete residual at these points. This phase incurs interprocessor communication. A standard technique to reduce the latency associated with this communication consists of packing all messages to be transmitted between adjacent pairs of communicating processors into a single large message buffer, which is then transmitted and unpacked on the receiving processor.[27,29,30]

The explicit domain-decomposition and message-passing approach can also be implemented on shared-memory machines. In fact, for large numbers of processors, this approach generally scales more favorably than the simple reordering strategies discussed earlier. This is largely because reordering-based techniques on shared-memory machines implicitly generate communication at the loop level, each time an off-processor or nonlocal memory reference is encountered. In the explicit approach, duplicate remote memory references are omitted through the use of local ghost points and precomputed optimal communication patterns, which are invoked only at critical locations in the program, such as at the end of a residual evaluation.

The implementation described in the remainder of this paper makes use of the explicit message-passing approach using the message-passing interface (MPI) library.[31] This makes for a portable program that can be run on distributed-memory parallel machines such as the Cray T3E, as well as shared-memory machines such as the SGI Origin 2000. For shared-memory architectures, the isolated communication routines, which make use of the MPI libraries, could be replaced with simple copies of data from local to remote memory locations, although this has not been attempted. (The shmem routines on the Cray T3E and SGI Origin 2000 also provide a similar functionality, although this has not been exploited in the present implementation, in the interest of portability.)

Although the flow solver implementation does not make use of the globally addressable memory capability of shared-memory machines such as the SGI Origin 2000, this feature has been instrumental in enabling the execution of the various preprocessing operations that are required by the multigrid algorithm before the flow solution phase. In particular, the implicit line construction and multigrid agglomeration algorithms, as well as the mesh partitioning procedure,[26] require little overall CPU time (usually several minutes, running interactively), but significant memory resources, of the order of one-third to one-half those required by the flow solver. Additionally, these routines involve considerable amount of logic, and their parallelization using explicit message passing is a relatively involved task. The shared-memory architecture of the SGI Origin 2000 enables the execution of these routines using a single processor, but accessing large portions of the entire 128 CPU machine memory (36 GB). These partitioned preprocessed results were then employed as input data for the flow solver running either on the SGI Origin 2000 or the Cray T3E. For increasingly massively parallel applications, the parallel implementation of these preprocessing applications will eventually be required. However, the complexity of this task should be substantially reduced using the shared-memory paradigm.

## V. Parallel Implementation

Distributed-memory explicit message-passing parallel implementations of unstructured mesh solvers have been discussed extensively in the literature.[23,27,32] 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, we choose 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 (cf. Fig. 4). 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.

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 permits a partitioning of 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 partitioner.[26] 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 to form a new vertex. Merging vertices also produce merged edges, as shown in Fig. 5 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 the partitioner described in Refs. 26 and 33, 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.

The Chaco partitioner[26] provides options for various partitioning algorithms. We use the multilevel bisection partitioning option exclusively[33] in this work because it provides good partitions at low computational cost. As an example, the two-dimensional mesh in Fig. 1, which contains the implicit lines shown in Fig. 2, has been partitioned both in its original unweighted uncontracted form and by the graph contraction method described earlier. Figure 6 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. Although the additional number of cut edges in line-partitioned cases may affect overall solution time by increasing overall communication volume, this effect is generally small and depends mainly on the number of partitions and the computer architecture on which the simulation is run. On the other hand, the line-constrained partitioning strategy ensures the exact same numerical algorithm is obtained for sequential as well as parallel runs.

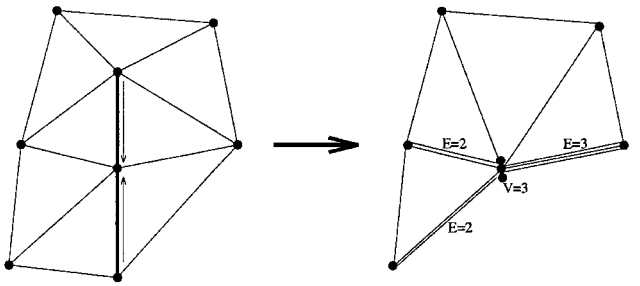
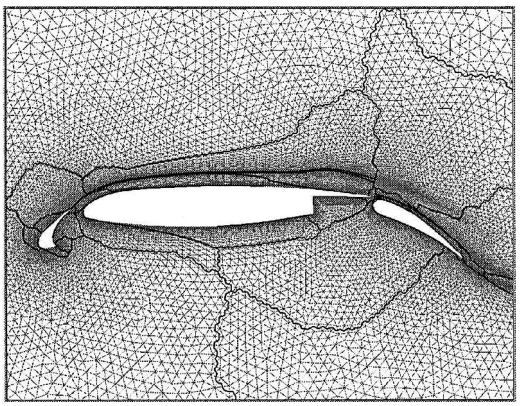
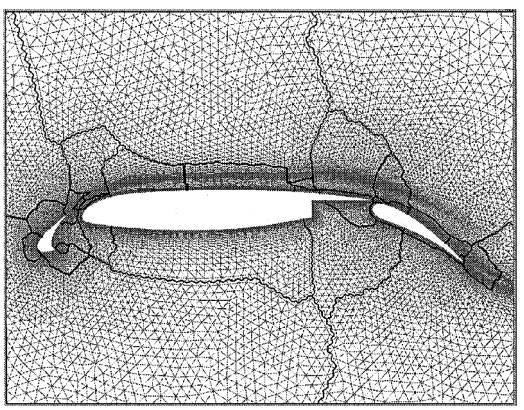



**Fig. 5 Line edge contraction and creation of weighted graph for mesh partitioning; *V* and *E* values denote vertex and edge weights, respectively.**



**Unweighted**



**Weighted**

**Fig. 6 Comparison of 32-way partition of two-dimensional mesh.**

Although the resulting partitions are balanced in terms of the number of vertices in each partition, no attempt is made to balance the number of line edges in each partition. In fact, most often there is great disparity in the number of line edges in the various partitions. Furthermore, it is not feasible to attempt to modify the vertex partition balance to account for the extra work incurred by the line-solves. This is because each stage of a multistage time step consists of a residual evaluation followed by a point or line-solve, with communication (and synchronization) occurring at the end of the residual evaluation and the point/line-solve. Thus, ideally, the partitions must be vertex balanced for the residual evaluation phase, but line balanced for the solution phase. Fortunately, the amount of work involved in a residual evaluation is much larger than that involved in the solution phase, and the additional work of a (block) line-solve vs a (block) point-solve is such that the line imbalance does not appreciably affect the overall computational efficiency.

## VI. Results

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. All of the cases reported in this section were run in dedicated mode, with no other users or processes present on the machines.

The first case consists of a relatively coarse 177,837 point grid over a swept and twisted wing, constructed by extruding a two-dimensional grid over an Royal Aircraft Establishment (RAE) 2822 airfoil in the spanwise direction. Figure 7 shows the grid for this case along with the implicit lines used by the solution algorithm on the finest level. The grid contains hexahedra in the boundary layer and (spanwise) prismatic elements in regions of inviscid flow and exhibits a normal spacing at the wing surface of $10^{-6}$ chords. Approximately 67% of the fine grid points are contained within an implicit line, and no implicit lines on any grid levels were intersected in the partitioning process for all cases. This case was run at a freestream Mach number of 0.1, an incidence of 2.31 deg, and a Reynolds number of $6.5 \times 10^6$. Figure 8 shows the computed solution obtained on this grid as a set of density contours on the surface. The convergence (as measured by the rms averaged density

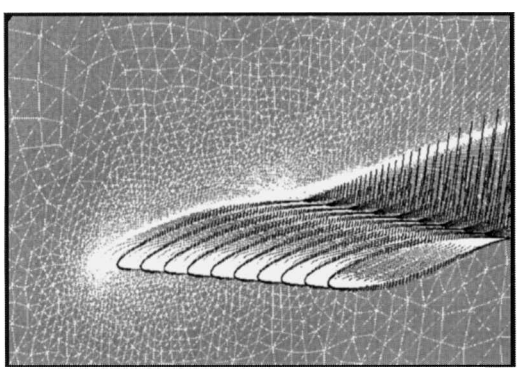


**Fig. 7 Unstructured grid and implicit lines employed for computing flow over three-dimensional swept and twisted RAE wing.**

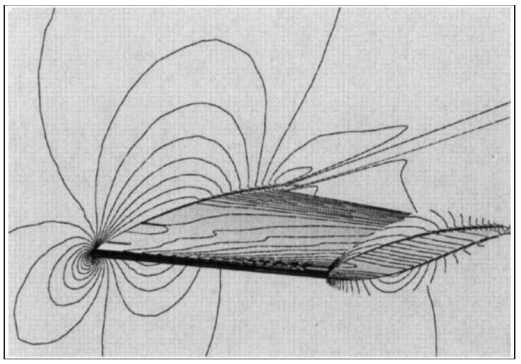


**Fig. 8 Computed density contours for flow over three-dimensional swept and twisted RAE wing; Mach = 0.1, incidence = 2.31 deg, $Re = 6.5 \times 10^6$.**

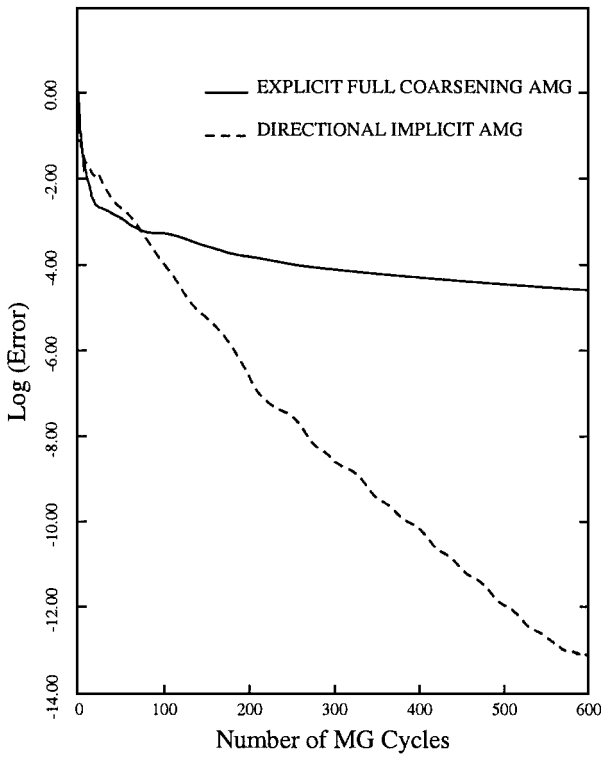



**Fig. 9 Comparison of convergence rate achieved by directional implicit agglomeration multigrid vs explicit agglomeration multigrid for flow over swept and twisted wing; Mach = 0.1, incidence = 2.31 deg, $Re = 6.5 \times 10^6$.**

residual vs the number of multigrid cycles) of the directional implicit multigrid algorithm is compared with that achieved by the explicit isotropic multigrid algorithm[8] on the equivalent two-dimensional problem in Fig. 9. The directional implicit multigrid algorithm is seem to be much more effective than the isotropic algorithm, reducing the residuals by 12 orders of magnitude over 600 multigrid W-cycles, using four levels. In the three-dimensional case, the turbulence model was frozen after 200 multigrid cycles, to isolate any convergence slowdown due to the turbulence model.

The second test case involves a finer grid of $1.98 \times 10^6$ points over an ONERA M6 wing. The grid was generated using the VGRID program.[1] A postprocessing operation was employed to merge the tetrahedral elements in the boundary layer region into prisms.[8] The final grid contains $2.4 \times 10^6$ prismatic elements and $4.6 \times 10^6$ tetrahedral elements and exhibits a normal spacing at the wall of $10^{-7}$ chords. Approximately 62% of the fine grid points are contained within an implicit line, and no implicit lines on any grid levels were intersected in the partitioning process for all cases. The freestream Mach number is 0.1, the incidence is 2.0 deg, and the Reynolds number is $3 \times 10^6$. The convergence for this case is shown in Fig. 10,

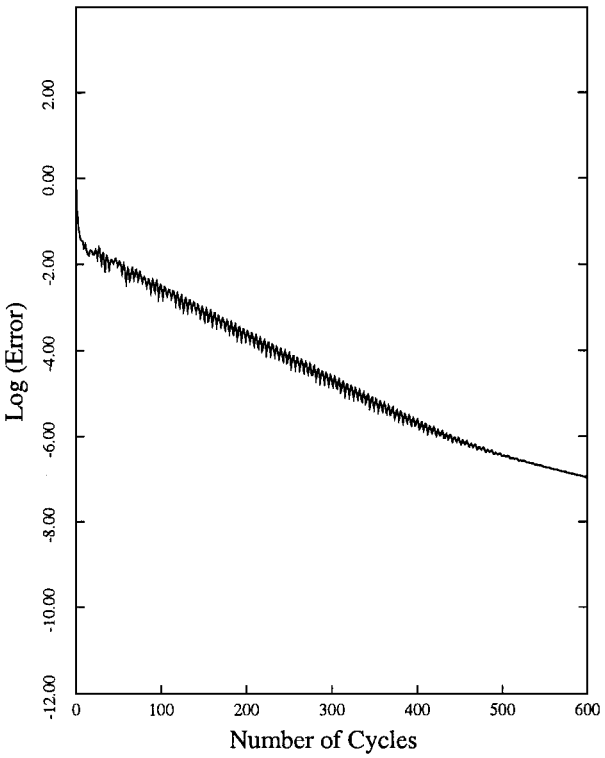



**Fig. 10 Multigrid convergence rate for $1.98 \times 10^6$ point grid over ONERA M6 wing; Mach = 0.1, incidence = 2.0 deg, $Re = 3 \times 10^6$.**

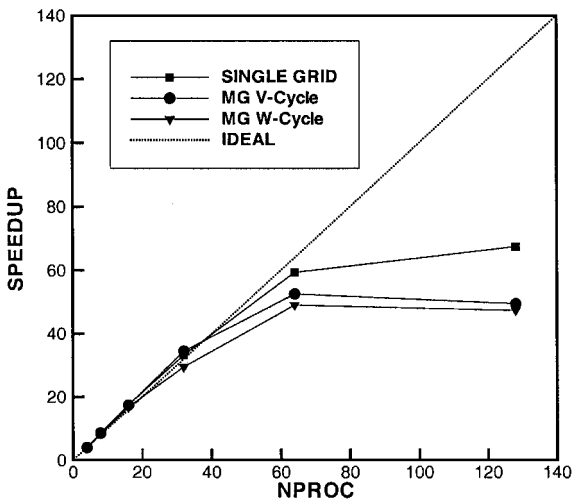



**Fig. 11 Observed speedups for RAE wing case (177,837 grid points) on SGI Origin 2000.**

where the residuals are reduced by seven orders of magnitude over 600 multigrid W-cycles, using five levels.

Figures 11 and 12 show the relative speedups achieved on the two target hardware platforms for the RAE wing case, whereas Figs. 13 and 14 show the corresponding results for the ONERA M6 wing case. For the purposes of Figs. 11–14 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 (using four levels for the RAE case, five levels for the ONERA M6 case). Note that the best convergence rates, that is, those shown in Figs. 9 and 10, are achieved using the W-cycle multigrid algorithm.

For the coarse RAE wing case, the results show good scalability up to a moderate number of processors, whereas the finer ONERA M6 wing case shows good scalability up to the maximum number

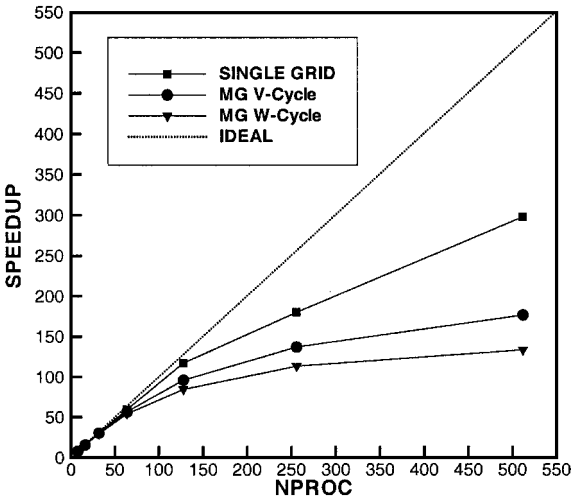



**Fig. 12 Observed speedups for RAE wing case (177,837 grid points) on Cray T3E.**

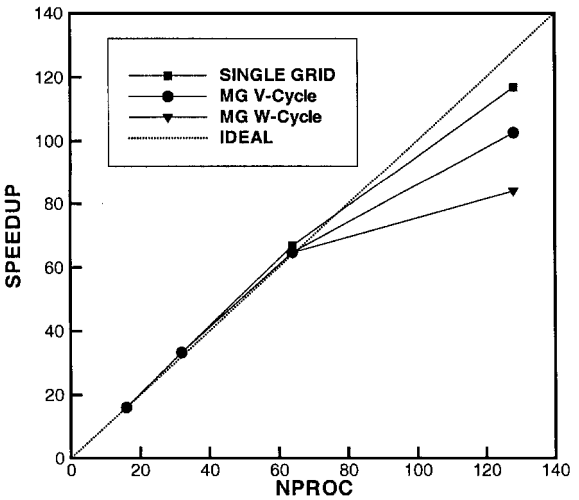



**Fig. 13 Observed speedups for ONERA M6 wing case ($1.98 \times 10^6$ grid points) on SGI Origin 2000.**

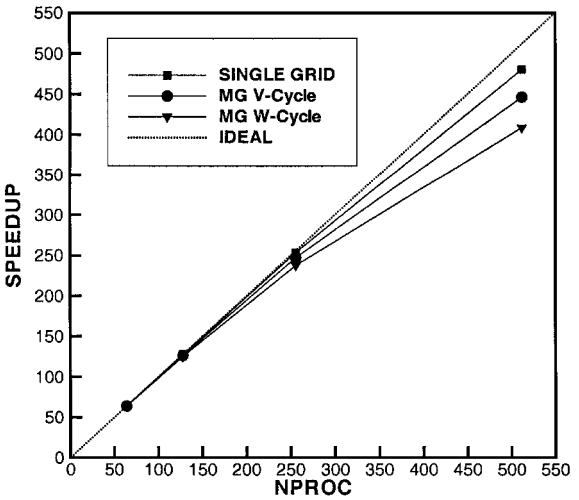



**Fig. 14 Observed speedups for ONERA M6 wing case ($1.98 \times 10^6$ grid points) on Cray T3E.**

of processors on each machine, with only a slight dropoff at the higher numbers of processors. This is to be expected because the relative ratio of computation to communication is higher for finer grids. This effect is also demonstrated by the superior scalability of the single grid algorithm vs the multigrid algorithms, and of the V-cycle multigrid algorithm over the W-cycle multigrid algorithm, that is, the W-cycle multigrid algorithm performs additional coarse grid sweeps compared to the V-cycle algorithm. Note that, for the RAE wing test case on 512 processors of the T3E, the fine grid contained only 348 vertices per processor, whereas the coarsest level contained a mere 13 points per processor. Whereas the W-cycle algorithm suffers somewhat in computational performance for coarser grids on high processor counts, the parallel performance of the W-cycle improves substantially for finer grids. Numerically the most robust and efficient convergence rates are achieved using this cycle.

Figures 12 and 14 clearly illustrate the superior scalability on the T3E for these cases. In Tables 1–4, the timings per cycle and estimated computational rates, as giga-floating-point operations per second (GFLOPS), are shown. These computational rates were estimated by running the same problem on a Cray C90 and scaling the computational rates measured by the hpm command by the ratio of wall clock time on the target machine to the total CPU time on the Cray C90. As Tables 1–4 indicate, the individual processors of the Origin 2000 are up to 70% faster than those of the T3E. For

**Table 1 Timings and estimated computational rates for single grid[a] ONERA M6 wing case on SGI Origin 2000 architecture**

| Number of processors | Time/cycle | GFLOPS | Speedup |
|---|---|---|---|
| 16 | 44.6 | 1.27 | 1.0 |
| 32 | 21.24 | 2.67 | 2.10 |
| 64 | 10.18 | 5.56 | 4.38 |
| 128 | 5.82 | 9.73 | 7.66 |

[a]Number of points: $1.98 \times 10^6$.

**Table 2 Timings and estimated computational rates for W-cycle multigrid[a] ONERA M6 wing case on SGI Origin 2000 architecture**

| Number of processors | Time/cycle | GFLOPS | Speedup |
|---|---|---|---|
| 16 | 94.02 | 1.04 | 1.0 |
| 32 | 45.12 | 2.16 | 2.08 |
| 64 | 22.36 | 4.36 | 4.20 |
| 128 | 14.22 | 6.87 | 6.61 |

[a]Number of points: $1.98 \times 10^6$.

**Table 3 Timings and estimated computational rates for single grid[a] ONERA M6 wing case on Cray T3E architecture**

| Number of processors | Time/cycle | GFLOPS | Speedup |
|---|---|---|---|
| 64 | 17.13 | 3.30 | 1.0 |
| 128 | 8.60 | 6.57 | 1.99 |
| 256 | 4.32 | 13.07 | 3.96 |
| 512 | 2.28 | 24.8 | 7.51 |

[a]Number of points: $1.98 \times 10^6$.

**Table 4 Timings and estimated computational rates for W-cycle multigrid[a] ONERA M6 wing case on Cray T3E architecture**

| Number of processors | Time/cycle | GFLOPS | Speedup |
|---|---|---|---|
| 64 | 31.3 | 3.12 | 1.0 |
| 128 | 16.02 | 6.10 | 1.95 |
| 256 | 8.43 | 11.57 | 3.71 |
| 512 | 4.90 | 20.00 | 6.39 |

[a]Number of points: $1.98 \times 10^6$.

example, on 64 processors, the ONERA M6 wing single grid case (a case which incurs little communication overhead) requires 10.2 s per cycle on the ORIGIN 2000, whereas the same case requires 17.1 s per cycle on 64 processors of the T3E.

However, the better scalability demonstrated on the T3E is indicative of its lower latency and higher bandwidth communication capability. (Additionally, the newer T3E-1200 using 600-MHz processors should provide faster computational rates.) In all cases, the fastest overall computational rates are achieved on the 512 processor configuration of the T3E. In this configuration, the RAE wing test case can be solved to machine zero in just 10 min, that is, 600 W-cycles as per Fig. 9. The $1.98 \times 10^6$ point grid for the ONERA M6 wing case requires just 4.9 s per cycle on the 512 processor T3E, or 14.2 s per cycle on the 128 processor Origin 2000. From Fig. 10 it can be deduced that a calculation to engineering accuracy can be performed in approximately 300 W-cycles, which requires 25 min on the T3E or 71 min on the Origin 2000.

The next test case involves the computation of flow over a three-dimensional high-lift component geometry. The geometry consists of an unswept wing with a partial span flap, which has been the subject of both experimental and computational investigations.[34–36] The wing is mounted between two end walls, and the flap extends to the midspan location. The flap deflection is 30 deg, while the overall incidence of the geometry with respect to the undisturbed flow is 10 deg. The freestream Mach number is 0.2, and the Reynolds number is $3.7 \times 10^6$. The flow over this geometry has been computed on an initial grid of $1.7 \times 10^6$ points, and on a finer grid of $13.5 \times 10^6$ points. The coarse grid exhibits a normal spacing at the wing surface of $10^{-6}$ chords, and the height of the cells in the boundary-layer region follows a geometric progression with a growth rate of 1.2. This grid was generated using the VGRID program and originally contained $10 \times 10^6$ tetrahedra, which were merged into $2.1 \times 10^6$ prisms and $3.6 \times 10^6$ tetrahedra through postprocessing. The finer grid was obtained through a uniform subdivision of the earlier grid, using the mixed element adaptive meshing program described in Ref. 37, resulting in $17 \times 10^6$ prisms and $29 \times 10^6$ tetrahedra. The high degree of resolution of this mesh is shown in Fig. 15. Figure 16 shows the computed density contours. For both grids approximately 70% of the vertices belong to implicit lines. The scalability of the coarser mesh is very similar to that displayed by the earlier ONERA M6 wing case and is therefore not reproduced here. This case was

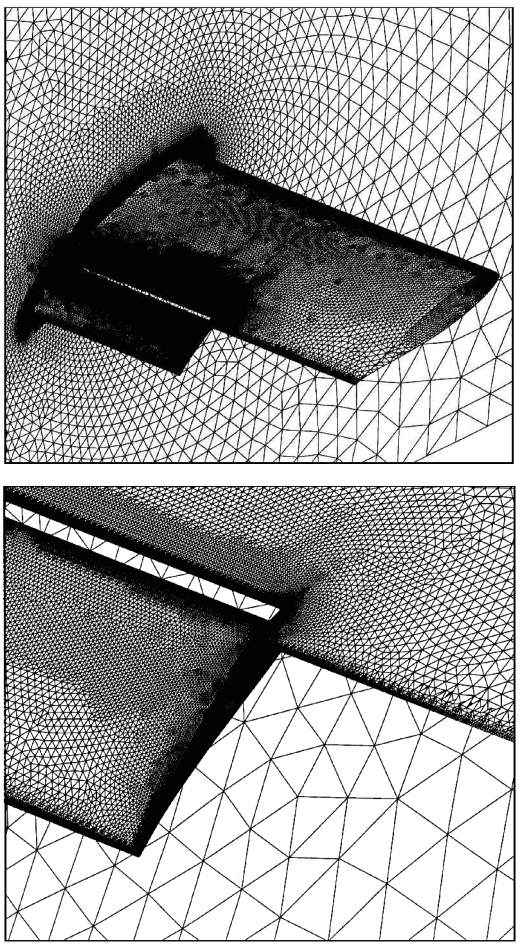


**Fig. 15 Grid with $13.5 \times 10^6$ points used to compute flow over partial-span flap geometry.**

**Table 5 Comparison of coarse and fine grid resource requirements for partial-span flap high-lift geometry on Cray T3E**

| Grid | Memory requirements, GB | Time/cycle | Time to solution[a] |
|---|---|---|---|
| $1.7 \times 10^6$ | 7.1 | 3.76 | 13 min |
| $13.5 \times 10^6$ | 35.5 | 27.75 | 236 min |
| Ratio (= 8) | 5.0 | 7.39 | 7.39 |

[a]Wall-clock time required for 220 multigrid W cycles.

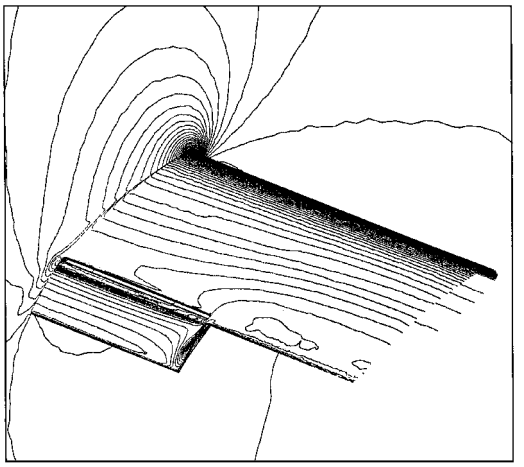


**Fig. 16 Computed density contours on $13.5 \times 10^6$ point grid for flow over part-span flap geometry; Mach = 0.2, incidence = 10 deg, Reynolds number = $3.7 \times 10^6$.**

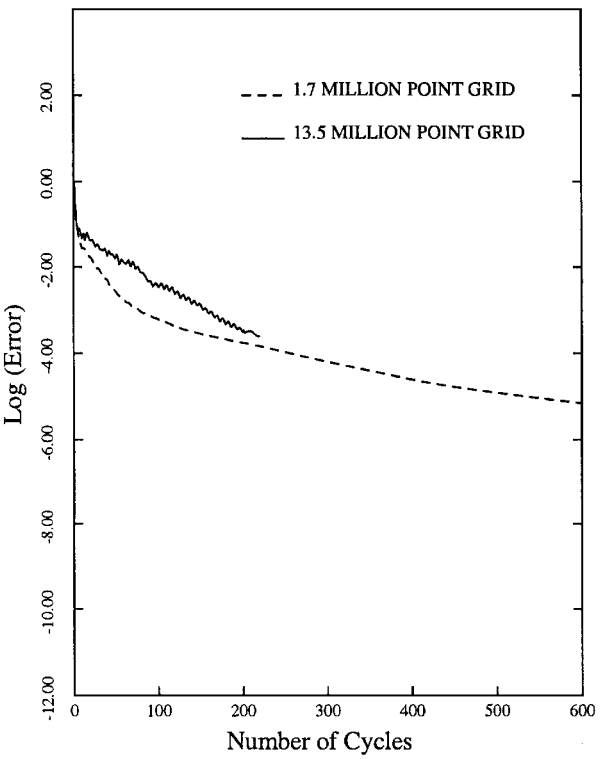



**Fig. 17 Multigrid convergence rates for fine ($13.5 \times 10^6$ points) and coarse ($1.7 \times 10^6$ points) grids over part-span flap geometry.**

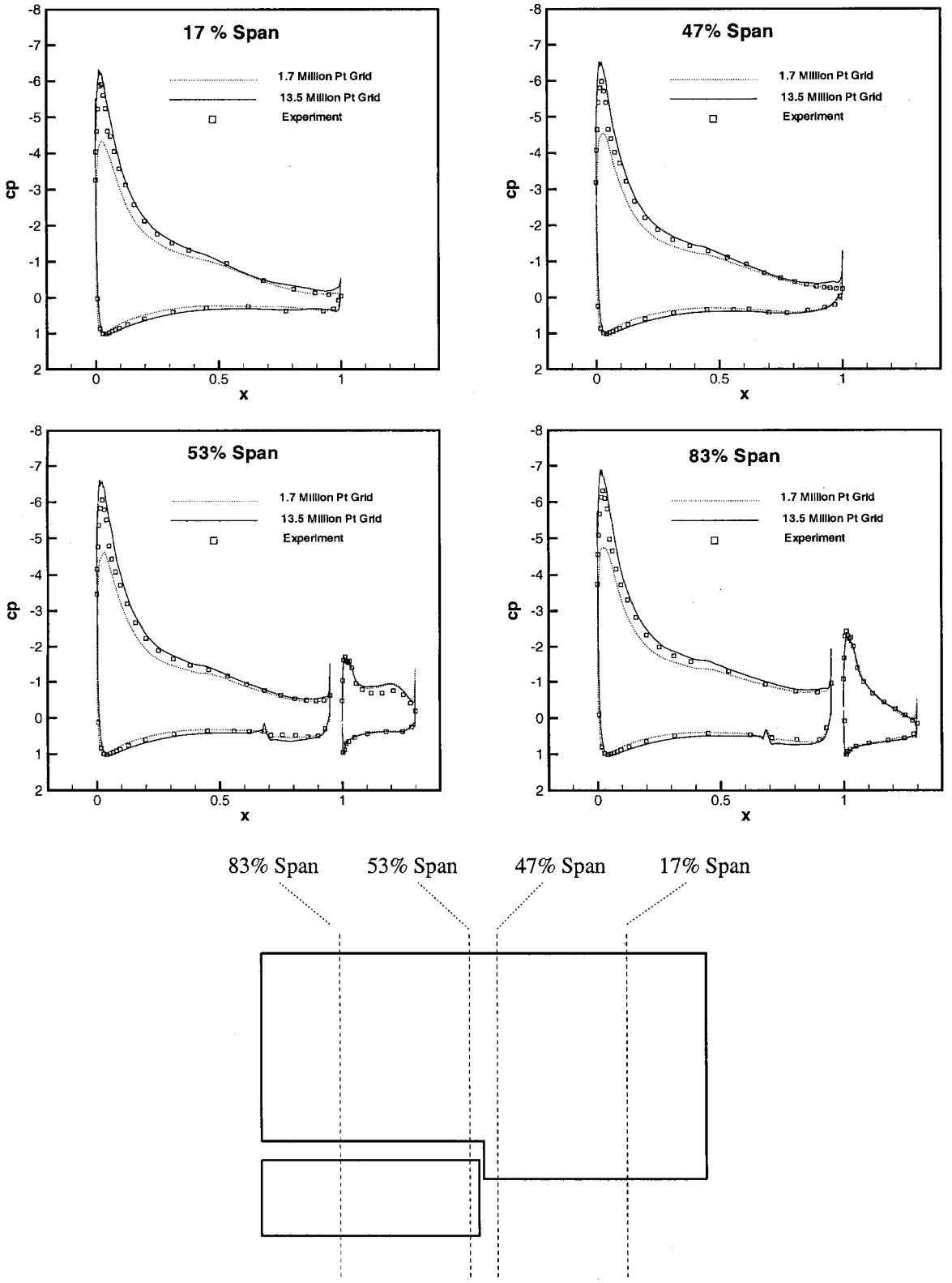



**Fig. 18 Comparison of computed surface pressure distribution with experimental results at four spanwise locations on partial-span flap geometry for fine and coarse grids.**

run for 600 W-cycles on 512 processors of the T3E, which required 3.76 s per W-cycle, or 38 min for the entire run, during which the residuals were reduced by just over five orders of magnitude. The fine grid case required 28 s per cycle on 512 processors of the T3E, or 236 min for a run of 220 W-cycles, during which the residuals were reduced by just under four orders of magnitude, as shown in Fig. 17. At this point, the fine grid achieves approximately the same level of convergence as the coarser grid for the same number of cycles, which is deemed sufficient for engineering accuracy. Although the computation has not been carried out further due to computational expense, an asymptotic slowdown similar to that observed on the coarser grid can be expected for the fine grid. For the fine grid case, the solver required approximately 69 MB of memory per processor, or a total of 35.5 GB of memory, which constitutes just over half the available memory on the machine. This total is approximately 50% higher than the memory requirements of the sequential solver on a per grid point basis and is most likely due to imbalance between the processors (dimensioning

uses the maximum processor data-set size on all processors), as well as additional memory required for the ghost points and other message-passing data structures.

A summary of the resource requirements for the coarse and fine grid runs on the partial-span flap geometry (multigrid W-cycle) is given in Table 5. Although the fine grid contains eight times the resolution of the coarse grid, the memory and CPU time required by the fine grid run are less than eight times that of the coarse grid run. This is due to the larger ratio of computation to communication in the fine grid case. Additionally, the fixed (nondata) memory requirement of the solver instruction set occupies 2.8 MB/processor, which represents a smaller portion of the total memory requirements in the fine grid case.

The computed surface pressures for both grids are compared with experimental values at four spanwise locations in Fig. 18. The coarse grid values tend to underpredict the suction peaks on the leading edge of the main wing. Much better agreement is obtained on the finer grid. The suction peaks are slightly overpredicted on the fine grid, although this is in agreement with previous numerical computations of this flow on block structured grids, which also overpredict the lift in these regions.[36]

The poor agreement of the coarse grid suction peaks with experimental values may be somewhat surprising, considering the better agreement obtained on previous unstructured computations performed with grids of similar overall size.[8] However, in these previous computations, the grids contained much lower boundary-layer resolution, which was traded off for increased chordwise resolution. The boundary-layer resolution employed in the current grids was derived from the resolution requirements established in two-dimensional high-lift computations[38,39] and is believed to represent the necessary level for capturing detailed flow physics. On the other hand, no spanwise grid stretching was employed, and thus a substantial savings in the total number of grid points with minimal accuracy degradation could probably be obtained by reducing the spanwise resolution. The fine grid calculation on the partial-span flap geometry is intended to demonstrate the size of problems that can be attempted using this type of solver on present-day massively parallel computer architectures, rather than to define the number of grid points required for an accurate calculation of this type.

## VII. Conclusion

The combination of a low-memory rapidly converging directional implicit multigrid algorithm and massively parallel computer architectures with large shared or distributed memory has enabled the solution of very large unstructured grid problems in reasonable turnaround time. The $13.5 \times 10^6$ grid point case described, which requires approximately half the available memory of a 512 processor T3E with 128 MB per processor, is believed to be the largest unstructured grid problem attempted to date. An accurate solution of a full transport aircraft high-lift configuration will require several times more resolution. Such a calculation should be feasible on some of the largest currently available parallel machines. Furthermore, the relatively low cost of commodity memory employed by microprocessor-based parallel computers implies that it is now feasible to configure cost-effective midsize machines with 32–64 processors and 1 GB of memory per processor, thus enabling large-scale unstructured grid computations suitable for high-lift analysis with overnight turnaround.

Future work will concentrate on improving the performance of the three-dimensional multigrid algorithm, which often produces suboptimal convergence rates as compared to the equivalent two-dimensional algorithm.[7] Furthermore, many of the preprocessing operations required for the parallel computations, such as mesh partitioning, implicit line construction, and multigrid agglomeration are currently performed sequentially and must eventually be parallelized. Finally, adaptive meshing techniques must be incorporated into the overall parallel solution strategy to take full advantage of the potential of unstructured meshes.

## Acknowledgments

This research was supported by NASA under Contract NAS1-19480 while the author was in residence at the Institute for Computer Applications in Science and Engineering. 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, as well as J. Dawson and R. Vermeland. Acknowledgments are also due to S. Pirzadeh and J. Garriz, who provided the grids generated with the VGRID program.

## References

[1]Pirzadeh, S., "Viscous Unstructured Three-Dimensional Grids by the Advancing-Layers Method," AIAA Paper 94-0417, Jan. 1994.

[2]Kallinderis, Y., Khawaja, A., and McMorris, H., "Hybrid Prismatic/Tetrahedral Grid Generation for Viscous Flows Around Complex Geometries," *AIAA Journal*, Vol. 34, No. 2, 1996, pp. 291–298.

[3]McMorris, H., and Kallinderis, Y., "Octree-Advancing Front Method for Generation of Unstructured Surface and Volume Meshes," *AIAA Journal*, Vol. 35, No. 6, 1997, pp. 976–984.

[4]Marchant, M. J., and Weatherill, N. P., "Unstructured Grid Generation for Viscous Flow Simulations," *Numerical Grid Generation in Computational Fluid Dynamics and Related Fields*, edited by N. P. Weatherill, P. R. Eiseman, J. Hauser, and J. F. Thompson, Pineridge, Swansea, Wales, U.K., 1994, pp. 151–162.

[5]Marcum, D. L., "Generation of Unstructured Grid for Viscous Flow Applications," AIAA Paper 95-0212, Jan. 1995.

[6]Mavriplis, D. J., "Multigrid Strategies for Viscous Flow Solvers on Anisotrpic Unstructured Meshes," *Proceedings of the 13th AIAA Computational Fluid Dynamics Conference*, AIAA, Reston, VA, 1997, pp. 659–675.

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

[8]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.

[9]Roe, P. L., "Approximate Riemann Solvers, Parameter Vectors and Difference Schemes," *Journal of Computational Physics*, Vol. 43, No. 2, 1981, pp. 357–372.

[10]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.

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

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

[13]Ollivier-Gooch, C., "Towards Problem-Independent Multigrid Convergence Rates for Unstructured Mesh Methods I: Inviscid and Laminar Flow," *Proceedings of the 6th International Symposium on CFD*, 1995.

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

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

[16]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.

[17]van Leer, B., Turkel, E., Tai, C. H., and Mesaros, L., "Local Preconditioning in a Stagnation Point," *Proceedings of the 12th AIAA Computational Fluid Dynamics Conference*, AIAA, Washington, DC, 1995, pp. 88–101.

[18]Turkel, E., "Preconditioning-Squared Methods for Multidimensional Aerodynamics," *Proceedings of the 13th AIAA Computational Fluid Dynamics Conference*, AIAA, Reston, VA, 1997, pp. 856–866.

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

[20]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.

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

[22]Mavriplis, D. J., "On Convergence Acceleration Techniques for Unstructured Meshes," AIAA Paper 98-2966, June 1998.

[23]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.

[24]Lohner, R., "Renumbering Strategies for Unstructured Grid Solvers Operating on Shared-Memory Cache-Based Parallel Machines," *Proceedings of the 13th AIAA Computational Fluid Dynamics Conference*, AIAA, Reston, VA, 1997, pp. 1015–1025.

[25]Pothen, A., Simon, H. D., and Liou, D. P., "Partitioning Sparse Matrices with Eigenvectors of Graphs," *SIAM Journal on Matrix Analysis and Applications*, Vol. 11, No. 2, 1990, pp. 430–452.

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

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

[28]Cuthill, E., and McKee, J., "Reducing the Band Width of Sparse Symmetric Matrices," *Proceedings of the ACM National Conference*, Association for Computing Machinery, New York, 1989, pp. 157–172.

[29]Das, R., Mavriplis, D. J., Saltz, J., and Ponnusamy, R., "The Design and Implementation of a Parallel Unstuctured Euler Solver Using Software Primitives," AIAA Paper 92-0562, Jan. 1992.

[30]Venkatakrishnan, V., "Parallel Implicit Unstructured Grid Euler Solvers," *AIAA Journal*, Vol. 32, No. 10, 1994, pp. 1985–1991.

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

[32]Venkatakrishnan, V., "Implicit Schemes and Parallel Computing in Unstructured Grid CFD," *VKI Lecture Series*, VKI-LS 1995-02, von Kármán Inst. for Fluid Dynamics, Rhode Saint Genese, Belgium, 1995.

[33]Hendrickson, B., and Leland, R., "A Multilevel Algorithm for Partitioning Graphs," *Proceedings of the 1995 ACM/IEEE Supercomputing Conference*, http://www.supercomp.org/sc95/proceedings/509_BHEN/SC95.HTM [cited 3 December 1995].

[34]Storms, B. L., Takahashi, T. T., and Ross, J. C., "Aerodynamic Influence of a Finite-Span Flap on a Simple Wing," Society of Automotive Engineers Aerotech Conf., Sept. 1995.

[35]Mathias, D. K., Roth, K. R., Ross, J. C., and Rogers, S. E., "Navier–Stokes Analysis of the Flow About a Flap Edge," AIAA Paper 95-0185, Jan. 1995.

[36]Takallu, M. A., and Laflin, K. R., "Reynolds-Averaged Navier–Stokes Simulations of Two Partial-Span Flap Wing Experiments," AIAA Paper 98-0701, Jan. 1998.

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

[38]Lynch, F. T., Potter, R. C., and Spaid, F. W., "Requirements for Effective High Lift CFD," *Proceedings of the 20th ICAS Congress*, AIAA, Reston, VA, 1996, pp. 1479–1492.

[39]Valarezo, W. O., and Mavriplis, D. J., "Navier–Stokes Applications to High-Lift Airfoil Analysis," *Journal of Aircraft*, Vol. 32, No. 3, 1995, pp. 618–624.

J. Kallinderis
*Associate Editor*