AIAA-95-0221

An Unstructured Mesh Newton Solver for Compressible Fluid Flow and Its Parallel Implementation

Timothy J. Barth
Samuel W. Linton
NASA Ames Research Center
Moffett Field, CA 94035

33rd Aerospace Sciences Meeting & Exhibit
January 9-12, 1995/Reno, NV

An Unstructured Mesh Newton Solver for Compressible Fluid Flow and Its Parallel Implementation*

Timothy J. Barth †
Advanced Algorithms and Applications Branch
NASA Ames Research Center
Moffett Field, CA 94035

Samuel W. Linton ‡
Sterling Software
NASA Ames Research Center
Moffett Field, CA 94035

Abstract

An algorithm is presented for the solution of the fluid flow equations on two- and three-dimensional unstructured meshes. The spatially discretized equations are solved using a Newton iteration procedure. Numerical results are shown for the uniprocessor algorithm in two space dimensions. The algorithm is then extended to three space dimensions and implemented on a multiprocessor architecture using a message passing protocol. Numerical results are shown in three dimensions for computations carried out on the IBM SP2 parallel computer.

Introduction and Motivation

Our goal is to develop methods for solving the Navier-Stokes equations which permit generalization into diverse areas such as aerodynamic shape optimization and the study of fluid flow bifurcation phenomena. This imposes additional requirements not normally placed on the design of computer algorithms for solving the Navier-Stokes equations. For example, in optimization theory and nonlinear programming, an objective function $F(x)$ is minimized subject to constraints $C_i(x)$. A necessary condition for optimality at x^* is that the gradient of the objective function be parallel to the gradient of the active constraints

$$g(x^*) = A^T(x^*)\lambda(x^*)$$

where $g(x)$ is the gradient of the objective function and $A(x)$ is the Jacobian matrix of the active constraints. For this reason, a linear system involving the transposed Jacobian matrix is often solved (perhaps in a least-squares sense) to obtain Lagrange multiplier estimates. Since in aerodynamic shape optimization, the flow equations are treated as constraints, the need arises to solve the transposed Jacobian matrix problem

$$A^T x = b.$$

This is a matrix problem not normally encountered in numerical schemes for solving the discretized flow equations.

A motivation for constructing Newton's method to solve the discretized fluid flow equations concerns the convergence and robustness of current numerical methods near singular points in the Jacobian matrix (characterized by vanishing eigenvalues). These singular points can arise due to a variety of physical flow phenomena such as fluid flow hysteresis and a host of fluid flow bifurcation events. Singularities can also arise from a purely numerical origin. For example, it is known that certain numerical schemes admit nonunique steady-state shockwave structures [Bar89] and consequently produce singular discrete operators. From standard fixed-point theory, we know that numerical schemes which converge at a linear rate depend fundamentally on the eigenvalue spectrum of the Jacobian matrix. Therefore, we expect

†barth@nas.nasa.gov
‡linton@biocomp.arc.nasa.gov

these methods to perform poorly, i.e. sublinearly, near or at these singular points. On the other hand, Newton's method generally converges quadratically and degenerates to linear performance near singularities. Other properties of Newton's method near fixed-points make the method appealing. Let $\{u_k\}$ denote a sequence that convergences quadratically (or at least superlinearly) to the fixed point u^*, i.e.

$$|u_{k+1} - u^*| \leq \beta_k |u_k - u^*|$$

for some $\{\beta_k\}$ that converge to zero. From this property it is easily shown that

$$\lim_{k \to \infty} \frac{|u_{k+1} - u_k|}{|u_k - u^*|} = 1.$$

This theoretical result is of genuine practical value since it provides a means for estimating the distance to the exact solution (fixed-point) given the distance between successive iterates. For methods converging at a linear rate, measuring the distance between successive iterates is not generally sufficient to estimate the distance to the exact solution.

Although we claim to use Newton's method to solve the discretized fluid flow equations, this point requires clarification. We have developed the present algorithm within the framework of a backward Euler integration of the time dependent equations so that as the time step parameter approaches infinity, Newton's method is recovered. This provides a mechanism for reaching the basin of attraction in Newton's method. In practice, Newton's method is approached only for the last few iterations of the scheme.

In related work, an inexact Newton scheme using numerical Fréchet derivatives and a preconditioned iterative matrix solver was introduced by Johan [Joh92] for solving the 3-D compressible fluid flow equations discretized using a finite element approximation. The algorithm was successfully implemented on the Thinking Machines CM-5 computer using data parallel programming. Tezduyar and coworkers have also considered implicit solution strategies for 3-D finite element formulations on the CM-5 computer, see for example [MT93]. Venkatakrishnan [Ven94] has explored the use of preconditioned iterative solvers on the Intel iPSC860 parallel computer in solving the 2-D Euler equations. His procedure uses approximate matrix-vector products yielding an approximate Newton iteration.

The use of Newton's method in solving the Navier-Stokes equations about complex geometries places large demands on computer memory and floating-point performance. This makes current parallel computers a natural platform for this development. We will show that the resulting method which we have devised is well suited to implementation on parallel computer architectures with large memory capacity and fast sequential processing rates. The algorithm can be modularly programmed using a standard message passing protocol.

In the remainder of the paper, we will describe the Navier-Stokes equations with turbulence, the basic discretization technique used for unstructured meshes, and the Newton iteration solution procedure. The performance of the resulting algorithm in two dimensions will be demonstrated by numerical example. The extension of the code to three dimensions and its implementation on a parallel computer using message passing will then be described. Finally, some results will be shown from calculations on the IBM SP2 parallel computer.

Navier-Stokes Equations

We consider the standard compressible Navier-Stokes equations in integral form for a domain Ω with bounding surface $\partial\Omega$

$$\frac{\partial}{\partial t}\int_\Omega \mathbf{u}\, dV + \int_{\partial\Omega} (\mathbf{f}\cdot\mathbf{n}) dS = \int_{\partial\Omega} (\mathbf{g}\cdot\mathbf{n}) dS \quad (1)$$

where $\mathbf{u}$ represents the vector of conserved variables, $\mathbf{f}$ and $\mathbf{g}$ the inviscid and viscous flux vectors respectively. In two dimensions, the vectors are

$$\mathbf{u} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ E \end{pmatrix}, \quad \mathbf{f} = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ u(E+p) \end{pmatrix} \hat{\mathbf{i}} + \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ v(E+p) \end{pmatrix} \hat{\mathbf{j}}$$

$$\mathbf{g} = \begin{pmatrix} 0 \\ \tau_{xx} \\ \tau_{yx} \\ u\tau_{xx} + v\tau_{xy} - q_x \end{pmatrix} \hat{\mathbf{i}} + \begin{pmatrix} 0 \\ \tau_{xy} \\ \tau_{yy} \\ u\tau_{yx} + v\tau_{yy} - q_y \end{pmatrix} \hat{\mathbf{j}}$$

$$\mathbf{q} = -\kappa\nabla T, \boldsymbol{\tau} = \lambda(u_x + v_y)I + \mu \begin{bmatrix} 2u_x & u_y + v_x \\ v_x + u_y & 2v_y \end{bmatrix}$$

with $p = (\gamma - 1)\left(E - \frac{1}{2}\rho(u^2 + v^2)\right) = \rho RT$. In these equations λ and μ are diffusion coefficients, κ the coefficient of thermal conductivity, γ the ratio of specific heats, and R the ideal gas law constant. In addition, at solid walls with no slip boundary conditions the flux reduces to the following form:

$$\mathbf{f}\cdot\mathbf{n} = \begin{pmatrix} 0 \\ n_x p \\ n_y p \\ 0 \end{pmatrix}, \quad \mathbf{g}\cdot\mathbf{n} = \begin{pmatrix} 0 \\ \nabla u \cdot \mathbf{n} \\ \nabla v \cdot \mathbf{n} \\ -\nabla q \cdot \mathbf{n} \end{pmatrix}.$$

The last entry in the viscous flux vanishes for adiabatic flow. These conditions can be enforced weakly. In addition, the strong condition can be applied that the velocity vector vanish at the surface.

High Reynolds Number Flow and Turbulence

In addition to the basic Navier-Stokes equations, we model the effects of turbulence on the mean flow equations using an eddy viscosity turbulence model. In a report with Baldwin [BB90], we proposed a single equation turbulence transport model with the specific application to unstructured meshes in mind. This model was subsequently modified by Spalart and Allmaras [SA92] to improve the predictive capability of the model for wakes and shear-layers as well as to simplify the model's dependence on distance to solid walls. In the present computations, the Spalart model is solved in a form fully coupled to the Navier-Stokes equations. The 1-equation model for the eddy viscosity-like parameter $\widetilde{\nu}$ is written

$$\begin{aligned} \frac{D\widetilde{\nu}}{Dt} &= \frac{1}{\sigma}\left[\nabla\cdot((\nu+\widetilde{\nu})\nabla\widetilde{\nu}) + c_{b2}(\nabla\widetilde{\nu})^2\right] \\ &- \left[c_{w1}f_w - \frac{c_{b1}}{\kappa^2}f_{t2}\right]\left[\frac{\widetilde{\nu}}{d}\right]^2 + c_{b1}\widetilde{S}\widetilde{\nu} \end{aligned} \quad (2)$$

where d is the physical distance to solid walls, c_{b1}, c_{b2}, c_{w1}, and σ are constants, and $\widetilde{S}$ is related to the strain tensor. The complete model also includes terms for simulating the transition to turbulence.

The Solution Algorithm

The flow equations are solved using a finite-volume method. In this technique the solution domain is tessellated into a number of smaller subdomains ($\Omega = \cup\Omega_i$). Each subdomain serves as a control volume in which mass, momentum, and energy are conserved. In the present application, the control volumes are formed from a dual obtained from the triangulation, see Figure 1.

Fundamental to the finite-volume method is the definition of the integral cell average. Componentwise, the integral cell average is defined in each subdomain as:

$$\overline{u}_i = \frac{1}{V_i}\int_{\Omega_i} u\, dV$$

where $V_i = \int_{\Omega_i} dV$. The integral conservation law can then be rewritten in the following form:

$$\frac{\partial}{\partial t}(\overline{\mathbf{u}}V) + \int_{\partial\Omega}(\mathbf{f}\cdot\mathbf{n})dS = \int_{\partial\Omega}(\mathbf{g}\cdot\mathbf{n})dS. \quad (3)$$

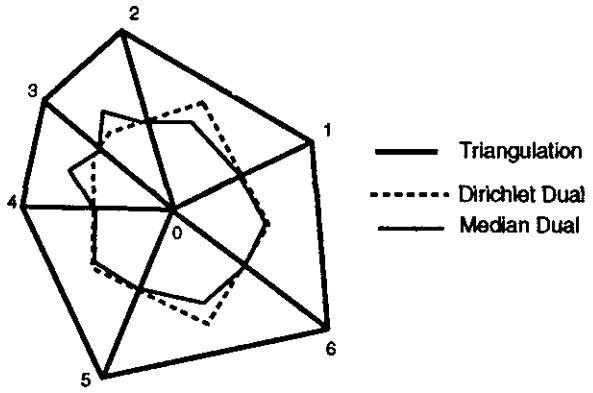

Figure 1: Local triangulation with Dirichlet and median duals.

The integral cell averages are the basic unknowns (degrees of freedom) in the scheme. The task at hand is to evaluate the flux integral given these cell averages of the solution. The basic solution process is summarized in the following steps and further details are given in [Bar91] [BJ89]:

Reconstruction

Given integral averages of the solution in each control volume, reconstruct a piecewise polynomial which approximates the behavior of the solution in each control volume.

Flux Quadrature

From the piecewise polynomial description of the solution, approximate the flux integral in (3) by numerical quadrature. Because the piecewise polynomials are not continuous at control volume boundaries, special flux functions are employed which are functions of multiple solution states. Those flux functions which can be characterized as some approximate and/or exact solution of the Riemann problem of gasdynamics result in upwind biased approximations. Present computations utilize Roe's approximate Riemann solver [Roe81] which is described in a later section.

Evolution

Given a numerical approximation to the flux integral, evolve the system in time using any class of implicit or explicit schemes. This results in new integral cell averages of the solution. The solution process can then be repeated.

It is important to realize that for steady-state calculations, the spatial accuracy of the scheme depends solely on the reconstruction and flux quadrature steps. Moreover, the use of cell averages can be

replaced by pointwise values of the solution associated with each control volume. In our application, we place the solution unknowns at mesh vertices. As we will see, this can greatly simplify the reconstruction step. Unfortunately, schemes based on these reconstructed polynomials are subject to the generation of spurious oscillations near discontinuities and regions of high solution gradient unless additional measures are taken which limit extremum behavior of the reconstructed polynomial. These measures are the basis for the class of MUSCL schemes developed by van Leer [vL79]. This framework of reconstruction followed by monotonicity enforcement extends naturally to unstructured meshes in higher dimensions and sufficient conditions required by the reconstructed polynomial to guarantee monotonicity are generally known, see for example [Bar94].

For purposes of this investigation, we do not undertake the task of introducing and properly incorporating monotonicity preserving measures into the Newton solver scheme. In the following paragraphs, two simple procedures are reviewed for data reconstruction from pointwise values of the solution at mesh vertices. For simplicity, sample algorithms are outlined in two dimensions although all algorithms have three-dimensional counterparts documented in the cited literature.

Green-Gauss Reconstruction

Consider a domain Ω' consisting of all triangles incident to some vertex v_0, see Figure 1, and the exact integral relation

$$\int_{\Omega'} \nabla u \, dV = \int_{\partial\Omega'} u\mathbf{n} \, dS.$$

In [BJ89] we show that given function values at vertices of a triangulation, a discretization of this formula can be constructed which is exact whenever u varies linearly:

$$(\nabla u)_{v_0} = \frac{1}{A_0} \sum_{i \in \mathcal{N}_0} \frac{1}{2}(u_i + u_0)\vec{\mathbf{n}}_{0i}. \qquad (4)$$

In this formula $\vec{\mathbf{n}}_{0i} = \int_a^b d\,\vec{\mathbf{n}}$ for any path which connects triangle centroids adjacent to the edge $e(v_0, v_i)$ and A_0 is the area of the *nonoverlapping* dual regions formed by the median dual region shown in Figure 1. This approximation is extended to three dimensions in [Bar91]. The formula (4) suggests a natural computer implementation using the edge data structure. Assume that the normals $\vec{\mathbf{n}}_{ij}$ for all edges $e(v_i, v_j)$ have been precomputed with the convention that the normal vector points from v_i to v_j. An edge implementation of (4) can be performed in the following way:

Algorithm: Green-Gauss Gradient Calculation

For $k = 1, n(e)$	Loop through edges
$j_1 := e^{-1}(k, 1)$	Pointer to edge origin
$j_2 := e^{-1}(k, 2)$	Pointer to edge destination
$u_{av} := (u(j_1) + u(j_2))/2$	Gather
$u_x(j_1) \mathrel{+}= \vec{\mathbf{n}}_x(k) \cdot u_{av}$	Scatter
$u_y(j_1) \mathrel{+}= \vec{\mathbf{n}}_y(k) \cdot u_{av}$	Scatter
$u_x(j_2) \mathrel{-}= \vec{\mathbf{n}}_x(k) \cdot u_{av}$	Scatter
$u_y(j_2) \mathrel{-}= \vec{\mathbf{n}}_y(k) \cdot u_{av}$	Scatter
Endfor	
For $k = 1, n(be)$	*Loop through boundary edges*
$j_1 := e^{-1}(k, 1)$	Pointer to edge origin
$j_2 := e^{-1}(k, 2)$	Pointer to edge destination
$u_{av} := (5u(j_1) + u(j_2))/6$	Weighted Gather
$u_x(j_1) \mathrel{+}= \vec{\mathbf{n}}_x(k) \cdot u_{av}$	Scatter
$u_y(j_1) \mathrel{+}= \vec{\mathbf{n}}_y(k) \cdot u_{av}$	Scatter
$u_{av} := (u(j_1) + 5u(j_2))/6$	Weighted Gather
$u_x(j_2) \mathrel{+}= \vec{\mathbf{n}}_x(k) \cdot u_{av}$	Scatter
$u_y(j_2) \mathrel{+}= \vec{\mathbf{n}}_y(k) \cdot u_{av}$	Scatter
Endfor	
For $j = 1, n(v)$	Loop through vertices
$u_x(j) := u_x(j)/area(j)$	Scale by area
$u_y(j) := u_y(j)/area(j)$	
Endfor	

It can be shown that the use of edge formulas for the computation of vertex gradients is asymptotically optimal in terms of work done.

Linear Least-Squares Reconstruction

To derive this reconstruction technique, consider a vertex v_0 and suppose that the solution varies linearly over the support of adjacent neighbors of the mesh. In this case, the change in vertex values of the solution along an edge $e(v_i, v_0)$ can be calculated by

$$(\nabla u)_0 \cdot (\mathbf{r}_i - \mathbf{r}_0) = u_i - u_0$$

where $\mathbf{r}$ denotes the spatial position vector. This equation represents the scaled projection of the gradient along the edge $e(v_i, v_0)$. A similar equation could be written for all incident edges subject to an arbitrary weighting factor. The result is the following matrix equation, shown here in two dimensions:

$$\begin{bmatrix} w_1\Delta x_1 & w_1\Delta y_1 \\ \vdots & \vdots \\ w_n\Delta x_n & w_n\Delta y_n \end{bmatrix} \begin{pmatrix} u_x \\ u_y \end{pmatrix} = \begin{pmatrix} w_1(u_1 - u_0) \\ \vdots \\ w_n(u_n - u_0) \end{pmatrix}$$

or in symbolic form $\mathcal{L}\,\nabla u = \mathbf{f}$ where

$$\mathcal{L} = \begin{bmatrix} \vec{L}_1 & \vec{L}_2 \end{bmatrix}$$

in two dimensions. Exact calculation of gradients for linearly varying u is guaranteed if any two row vectors $w_i(\mathbf{r}_i - \mathbf{r}_0)$ span all of 2 space. This implies linear independence of $\vec{L}_1$ and $\vec{L}_2$. The system can then be solved via normal equations

$$\begin{bmatrix} \vec{V}_1 \\ \vec{V}_2 \end{bmatrix} \begin{bmatrix} \vec{L}_1 & \vec{L}_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

The row vectors $\vec{V}_i$ are given by

$$\begin{aligned} \vec{V}_1 &= \frac{l_{22}\vec{L}_1 - l_{12}\vec{L}_2}{l_{11}l_{22} - l_{12}^2} \\ \vec{V}_2 &= \frac{l_{11}\vec{L}_2 - l_{12}\vec{L}_1}{l_{11}l_{22} - l_{12}^2} \end{aligned} \qquad (5)$$

with $l_{ij} = (\vec{L}_i \cdot \vec{L}_j)$.

Note that reconstruction of N independent variables in $\Re^d$ implies $\binom{d+1}{2} + dN$ inner product sums. Since only dN of these sums involves the solution variables themselves, the remaining sums could be precalculated and stored in computer memory. This makes the present scheme competitive with the Green-Gauss reconstruction. Using the edge data structure, the calculation of inner product sums can be calculated for *arbitrary* combinations of polyhedral cells. In all cases linear functions are reconstructed exactly.

Algorithm: Weighted Least-Squares Gradient Calculation

For $k = 1, n(e)$	Loop through edges
$j_1 = e^{-1}(k, 1)$	Edge origin
$j_2 = e^{-1}(k, 2)$	Edge destination
$\Delta x = w(k) \cdot (x(j_2) - x(j_1))$	Weighted Δx
$\Delta y = w(k) \cdot (y(j_2) - y(j_1))$	Weighted Δy
$l_{11}(j_1) = l_{11}(j_1) + \Delta x \cdot \Delta x$	l_{11} orig sum
$l_{11}(j_2) = l_{11}(j_2) + \Delta x \cdot \Delta x$	l_{11} dest sum
$l_{12}(j_1) = l_{12}(j_1) + \Delta x \cdot \Delta y$	l_{12} orig sum
$l_{12}(j_2) = l_{12}(j_2) + \Delta x \cdot \Delta y$	l_{12} dest sum
$\Delta u = w(k) \cdot (u(j_2) - u(j_1))$	Weighted Δu
$f_1(j_1) += \Delta x \cdot \Delta u$	$\vec{L}_1 f$ sum
$f_1(j_2) += \Delta x \cdot \Delta u$	
$f_2(j_1) += \Delta y \cdot \Delta u$	$\vec{L}_2 f$ sum
$f_2(j_2) += \Delta y \cdot \Delta u$	
Endfor	

For $j = 1, n(v)$ Process vertices dividing by det
$det = l_{11}(j) \cdot l_{22}(j) - l_{12}^2(j)$
$u_x(j) = (l_{22}(j) \cdot f_1(j) - l_{12}(j) \cdot f_2(j))/det$
$u_y(j) = (l_{11}(j) \cdot f_2(j) - l_{12}(j) \cdot f_1(j))/det$
Endfor

This formulation provides freedom in the choice of weighting coefficients, w_i. These weighting coefficients can be a function of the geometry and/or solution. Classical approximations in one dimension can be recovered by choosing geometrical weights of the form $w_i = 1./|\Delta \mathbf{r}_i - \Delta \mathbf{r}_0|^t$ for values of $t = 0, 1, 2$.

Geometric Duals for Stretched Triangulations

One significant difference (and apparent advantage) of the finite-volume method over the finite-element method appears to be the flexibility in choosing control volumes for use in the discrete conservation statement. It is known that the median dual geometry appears naturally in the finite-element discretization using linear elements. A finite-volume scheme can also be constructed using this geometry. Qualitative assessments have been made of finite-volume solution accuracy [AG94] on isotropic and stretched element meshes using a median dual geometry, but it is somewhat naive to make the median dual assumption in both of these cases. For example, Nicolaides and coworkers [XN92] have investigated the containment[1] dual in constructing staggered variable discretizations of divergence and curl. The containment dual, shown in Figure 2, is often

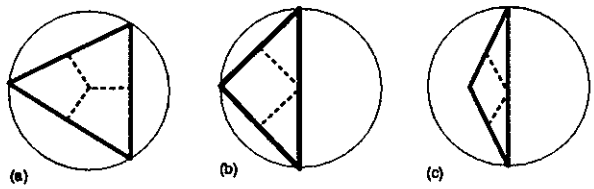

Figure 2: An example of containment duality (dotted lines) for acute (a) and obtuse (c) triangles.

studied in the theory of triangulations as well as error estimation, viz. the Delaunay triangulation in $\Re^d$ is known to minimize the maximum containment sphere [Raj91]. We have studied the containment dual because of its unusual properties on stretched triangulations [Bar94], see Figure 3. Using this

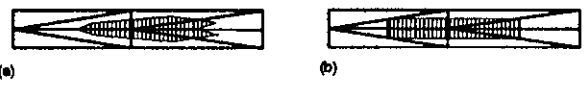

Figure 3: A comparison of median (a) and containment (b) duals for stretched triangulations.

[1] A.k.a. minimum spanning sphere

dual, discretization formulas can be constructed on unstructured meshes that are essentially identical to structured mesh formulas on quadrilateral cells. The use of containment duality does in general require a more sophisticated flux quadrature implementation, but the resulting improvement in accuracy can be dramatic.

Implicit Solution Algorithm

In this section we consider implicit solution strategies for the upwind schemes described in the previous section. Defining the solution vector

$$\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3, ..., \mathbf{u}_N]^T$$

and similarly $\mathbf{R}$ for all mesh vertices. The basic scheme is written as

$$D\mathbf{U}_t = \mathbf{R}(\mathbf{U}) \tag{6}$$

where D is a positive diagonal matrix. Performing a backward Euler time integration, equation (6) is rewritten as

$$D(\mathbf{U}^{n+1} - \mathbf{U}^n) = \Delta t\ \mathbf{R}(\mathbf{U}^{n+1})$$

where n denotes the iteration (time step) level. Linearizing the right-hand-side of the preceding equation in time produces the following form:

$$D(\mathbf{U}^{n+1} - \mathbf{U}^n) = \Delta t \left[\mathbf{R}(\mathbf{U}^n) + \frac{d\mathbf{R}^n}{d\mathbf{U}}(\mathbf{U}^{n+1} - \mathbf{U}^n)\right] \tag{7}$$

By rearrangement of terms, we arrive at the delta form of the backward Euler scheme

$$\left[\frac{D}{\Delta t} - \frac{d\mathbf{R}^n}{d\mathbf{U}}\right](\mathbf{U}^{n+1} - \mathbf{U}^n) = \mathbf{R}(\mathbf{U}^n) \tag{8}$$

Note that for large time steps, the scheme becomes equivalent to Newton's method for finding roots of a nonlinear system of equations. In practice the time step is usually scaled as a exponential function of the norm of the residual

$$\Delta t = f(\|\mathbf{R}(\mathbf{U})\|)$$

so that when $\|\mathbf{R}(\mathbf{U})\|$ approaches zero the time step becomes large. Each iteration of the scheme requires the solution of a linear system of equations of the form $A\mathbf{p} = \mathbf{b}$. The two most difficult tasks in this procedure are:

(1) Calculation of the Jacobian matrix elements

(2) Solution of the sparse linear system, $A\mathbf{p} = \mathbf{b}$

Before discussing these steps, it is worthwhile to assess the storage requirements for this strategy. Assume that the solution unknowns are associated with vertices of the mesh. All of the schemes discussed in previous sections require knowledge of distance-one neighbors in the graph (mesh). The higher order accurate schemes require more distant neighbors in building the scheme. If we consider only distance-one neighbors then the number of nonzero entries in each row of the matrix is related to the number of edges incident to the vertex associated with that row. Or equivalently, each edge $e(v_i, v_j)$ will guarantee nonzero entries in the i-th column and j-th row and similarly the j-th column and i-th row. In addition nonzero entries will be placed on the diagonal of the matrix. From this counting argument we see that the number of nonzero block entries NNZ in the matrix is exactly twice the number of edges plus the number of vertices, $2E + N$ (approximately $7N$ in 2-d). An approximate analysis of our fluid flow solver yields the following storage requirements for storing the Jacobian matrix assuming distance-one and distance-two neighbors:

Dim	NNZ (Dist 1)	NNZ (Dist 2)	Blk Size
2	$7N$	$19N$	5×5
3	$14N$	$55N$	6×6

In two dimensions, we have constructed versions of Newton's method which explicitly store the Jacobian matrix associated with either distance-one or distance-two neighbors of the triangulation. In 3-d this storage cost can be prohibitively large. The strategy taken in 3-d is to store only the matrix associated with distance-one neighbors. We will require two copies of this matrix which are used in two different ways: (1) in the preconditioning of the linear system, (2) partial matrix-vector products in the iterative solver.

A major task in the overall calculation is the linearization of the numerical flux vector with respect to the two solution states. For example, given the Roe flux function

$$\begin{aligned}\mathbf{h}(\mathbf{u}^R, \mathbf{u}^L; \mathbf{n}) &= \frac{1}{2}\left(\mathbf{f}(\mathbf{u}^R, \mathbf{n}) + \mathbf{f}(\mathbf{u}^L, \mathbf{n})\right) \\ &- \frac{1}{2}|A(\mathbf{u}^R, \mathbf{u}^L; \mathbf{n})|\left(\mathbf{u}^R - \mathbf{u}^L\right)\end{aligned} \tag{9}$$

we require the Jacobian terms $\frac{d\mathbf{h}}{d\mathbf{u}^R}$ and $\frac{d\mathbf{h}}{d\mathbf{u}^L}$. Exact analytical expressions for these terms are available [Bar87]. Using the edge data structure mentioned in the previous section, the actual assembly of the sparse matrix from components becomes straightforward. The basic idea is that we can determine

the flux Jacobian terms edge-wise. The edge $e(v_i, v_j)$ then contributes entries in the i-th row and j-th column as well as the j-th row and i-th column.

Solving The Linear System

The next task is to solve the large sparse linear system of the form

$$A\mathbf{p} - \mathbf{b} = 0$$

produced by Newton's method. Owing to the asymmetry of A, the GMRES algorithm of Saad and Schultz [YS86] is used. In the GMRES algorithm the solution is advanced from $\mathbf{p}_0$ to $\mathbf{p}_k = \mathbf{p}_0 + \mathbf{z}_k$. Let $\mathbf{v}_1 = A\mathbf{p}_0 - \mathbf{b}$. GMRES($k$) finds the best possible solution for the $\mathbf{z}$ over the Krylov subspace $[v_1, Av_1, A^2v_1, ..., A^{k-1}v_1]$ by solving the minimization problem

$$\min_{\hat{\mathbf{z}}} \|\mathbf{v}_1 + A\hat{\mathbf{z}}\|.$$

As k increases, the storage increases linearly and the computation quadratically. Thus Saad and Schultz devised a variant of GMRES(k) in which the GMRES algorithm is restarted every m steps.

To enhance the convergence of the basic GMRES algorithm, matrix preconditioning is utilized. In left preconditioned form the matrix problem becomes

$$P(A\mathbf{p} - \mathbf{b}) = 0.$$

If available, the optimal choice of P is A^{-1}, in which case the underlying matrix problem for GMRES is trivially solved with one Krylov vector. Obtaining A^{-1} amounts to a direct solution of the original matrix problem which is considered impractical for both storage and computational cost considerations. In practice, a preconditioner based on the inverse of an incomplete lower-upper factorization of the matrix A is used. More specifically, we employ ILU(n) where n refers to the level of additional fill allowed during the factorization over and above the sparsity pattern of the original matrix. For the present calculations, we use ILU(0) so that no additional storage is required. The preconditioned GMRES algorithm is detailed below:

Algorithm: Preconditioned GMRES(k)

For $l = 1, m$	m restart iterations
$\mathbf{v}_0 := \mathbf{b} - A\mathbf{p}_0$	initial residual
$\mathbf{r}_0 := P\mathbf{v}_0$	preconditioning step
$\beta := \|\mathbf{r}_0\|_2$	initial residual norm
$\mathbf{v}_1 := \mathbf{r}_0/\beta$	define initial Krylov
For $j = 1, k$	inner iterations
$\mathbf{y}_j := A\mathbf{v}_j$	matrix-vector product
$\mathbf{w} := P\mathbf{y}_j$	preconditioning step
For $i = 1, j$	Gram-Schmidt
$h_{i,j} := (\mathbf{w}, \mathbf{v}_i)$	.
$\mathbf{w} := \mathbf{w} - h_{i,j}\mathbf{v}_i$	.
End For	
$h_{j+1,j} := \|\mathbf{w}\|_2$	
$\mathbf{v}_{j+1} := \mathbf{w}/h_{j+1,j}$	define Krylov vector
End For	
$\mathbf{z} := \min_{\hat{\mathbf{z}}} \|\beta\mathbf{e}_1 - H\hat{\mathbf{z}}\|_2$	least squares solve
$\mathbf{p} := \mathbf{p}_0 + \sum_{i=1}^{m} \mathbf{y}_i\mathbf{z}_i$	approximate solution
If $\|\beta\mathbf{e}_1 - H\mathbf{z}\|_2 \leq \epsilon$ exit	convergence check
$\mathbf{p}_0 := \mathbf{p}$	restart
End For	

Matrix-Vector Products

Note that in the GMRES algorithm

$$A \approx \frac{D}{\Delta t} - \overline{A} = \left[\frac{D}{\Delta t} - \frac{d\mathbf{R}}{d\mathbf{U}}\right]$$

and the matrix-vector product $\overline{A}\mathbf{p}$ is actually a projection of the Jacobian matrix in the direction of $\mathbf{p}$ (a directional derivative)

$$\overline{A}\mathbf{p} = \frac{d\mathbf{R}}{d\mathbf{U}}\mathbf{p}.$$

This suggests several possible strategies for computing this term. Two possibilities are discussed in the following paragraphs. The first relies on approximate numerical differentiation using Fréchet derivatives [Joh92]. The second approach is exact and also permits the calculation of the transposed matrix problem.

Approximate Fréchet Derivatives

In the first approach, directional derivatives are obtained using numerical difference approximations. This approach is discussed in [BS94] [EW94]. The directional derivative is a limiting form of the difference approximation

$$\frac{d\mathbf{R}}{d\mathbf{U}}\mathbf{p} = \lim_{\epsilon \to 0} \frac{\mathbf{R}(\mathbf{U} + \epsilon\mathbf{p}) - \mathbf{R}(\mathbf{U})}{\epsilon}.$$

The primary concern with this approach is the accuracy of derivatives and the optimal choice for ϵ. If derivatives are not computed accurately then GMRES iteration may stall or fail. Using a forward difference approximation, ϵ must be carefully chosen.

In general it is insufficient to choose ϵ as a constant such as the square root of machine precision. Johan [Joh92] also mentions this fact and gives some analysis for choosing ϵ but this analysis assumes that $\mathbf{R}(\mathbf{u})$ is well scaled. An alternative is to use higher order accurate formulas such as central differencing at *double the computational cost.*

In the introduction we discussed applications which also require that the transposed matrix problem be solved. In this situation, we know of no Fréchet-like technique for constructing the matrix-vector product

$$\left[\frac{d\mathbf{R}}{d\mathbf{U}}\right]^T \mathbf{p}$$

using numerical difference approximations. We consider this a serious shortcoming of the method.

Exact Product Forms

In this section we will present a new technique for constructing matrix-vector products which is an exact calculation of the directional derivatives. Extension to systems and the inclusion of diffusion terms are also handled using this technique.

Let $G(e,v)$ denote the triangulation in 2-d or 3-d with N vertices and E edges. Next define the $N \times E$ directed connectivity matrix C such that

$$\text{If } e(v_i, v_j) \in G(e,v), \text{ then } C_{ie} = 1,\ C_{je} = -1$$

and zero otherwise. Let $\mathbf{h} = \mathbf{h}(\mathbf{u}^L, \mathbf{u}^R; \mathbf{n})$ denote the numerical flux function as defined by Equation 9. For a system of m coupled differential equations, the Jacobian matrix entries are actually small $m \times m$ blocks. For ease of exposition, we tacitly treat these small blocks as scalar entries. Under these simplifications, the desired matrix-vector product is given by

$$\frac{d\mathbf{R}}{d\mathbf{U}}\mathbf{p} = \mathcal{C}\left[\left[\frac{d\mathbf{h}}{d\mathbf{u}^L}\right]\left[\frac{d\mathbf{u}^L}{d\mathbf{u}}\right] + \left[\frac{d\mathbf{h}}{d\mathbf{u}^R}\right]\left[\frac{d\mathbf{u}^R}{d\mathbf{u}}\right]\right]\mathbf{p} \quad (10)$$

where $\left[\frac{d\mathbf{h}}{d\mathbf{u}}\right]$ is actually an $E \times E$ diagonal matrix, and $\left[\frac{d\mathbf{u}^L}{d\mathbf{u}}\right]$ an $E \times N$ matrix. As mentioned earlier, we have not incorporated monotonicity enforcement into the reconstruction procedure for the calculations performed in this investigation. In this case a considerable simplification occurs in the calculation of matrix-vector products. The main idea is given in the following almost trivial lemma.

Lemma: Let $v = \mathcal{R}(U) = \mathcal{R}(u_1, u_2, ..., u_N)$ denote an arbitrary order reconstruction operator. If $\mathcal{R}$ depends linearly on u_i then

$$\frac{dv}{du}p = \mathcal{R}(p).$$

Proof: Linearity implies that

$$v = \mathcal{R}(u_1, u_2, ..., u_N) = \sum_{i=1}^{N} \alpha_i u_i$$

so that $\frac{dv}{du_i} = \alpha_i$. The desired result follows immediately

$$\frac{dv}{du}p = \sum_{i=1}^{N} \frac{dv}{du_i}p_i = \sum_{i=1}^{N} \alpha_i p_i = \mathcal{R}(p).$$

This lemma suggests the following procedure for calculation of matrix-vector products.

$$\frac{d\mathbf{R}}{d\mathbf{U}}\mathbf{p} = \mathcal{C}\left[\left[\frac{d\mathbf{h}}{d\mathbf{u}^L}\right]\mathcal{R}^L(\mathbf{p}) + \left[\frac{d\mathbf{h}}{d\mathbf{u}^R}\right]\mathcal{R}^R(\mathbf{p})\right] \quad (11)$$

This amounts to a reconstruction of the vectors $\mathbf{p}^L$ and $\mathbf{p}^R$ from $\mathbf{p}$ using the same reconstruction operator used in the residual computation. Next, the linearized form of the flux function is computed:

$$\mathbf{h}_{lin} = \frac{d\mathbf{h}}{d\mathbf{u}^L}\mathbf{p}^L + \frac{d\mathbf{h}}{d\mathbf{u}^R}\mathbf{p}^R.$$

Finally, the linearized fluxes are assembled using the *same procedure as the residual vector assembly.* In actual calculations, the conservative flow variables are not reconstructed, thereby necessitating that a change of variable transformation be embedded in the formulation. This is not a serious complication.

Equation (11) does not reveal how to construct the transposed matrix-vector product

$$\left[\frac{d\mathbf{R}}{d\mathbf{U}}\right]^T \mathbf{p}.$$

But by introducing addition matrices, we can derive the required equation. In addition, the following forms allow the incorporation of monotonicity limiting in the reconstruction process, although we have not done so here. Define the $N \times E$ matrices $\mathcal{A}, \mathcal{S}^+, \mathcal{S}^-$

$$\text{If } e(v_i, v_j) \in G(e,v), \text{ then } \mathcal{A}_{ie} = \mathcal{S}^+_{ie} = 1, \mathcal{A}_{je} = \mathcal{S}^-_{je} = 1$$

and zero otherwise. In addition, define the $E \times E$ diagonal matrices containing edge and edge normal geometry $[\Delta x]$, $[\Delta y]$, $[\mathbf{n}_x]$, and $[\mathbf{n}_y]$ as well as the $N \times N$ diagonal matrix $[D]$ containing dual cell volumes. Using these matrices the left and right reconstructed states obtained by Green-Gauss reconstruction are

given by

$$\mathbf{u}^L = \left[[\mathcal{S}^-]^T + \left[\frac{\Delta x}{4}\right] [\mathcal{S}^-]^T [D^{-1}] \mathcal{C} [\mathbf{n}_x] \mathcal{A}^T + \left[\frac{\Delta y}{4}\right] [\mathcal{S}^-]^T [D^{-1}] \mathcal{C} [\mathbf{n}_y] \mathcal{A}^T \right] \mathbf{u} \tag{12}$$

$$\mathbf{u}^R = \left[[\mathcal{S}^+]^T - \left[\frac{\Delta x}{4}\right] [\mathcal{S}^+]^T [D^{-1}] \mathcal{C} [\mathbf{n}_x] \mathcal{A}^T - \left[\frac{\Delta y}{4}\right] [\mathcal{S}^+]^T [D^{-1}] \mathcal{C} [\mathbf{n}_y] \mathcal{A}^T \right] \mathbf{u} \tag{13}$$

A similar equation exists for the least-squares reconstruction. From these formulas the transposed matrix problem is obtained

$$\left[\frac{d\mathbf{R}}{d\mathbf{U}}\right]^T = \left[[\mathcal{S}^-] + \mathcal{A} [\mathbf{n}_x] \mathcal{C} [D^{-1}] [\mathcal{S}^-] \left[\frac{\Delta x}{4}\right] + \mathcal{A} [\mathbf{n}_y] \mathcal{C} [D^{-1}] [\mathcal{S}^-] \left[\frac{\Delta y}{4}\right] \right] \left[\frac{d\mathbf{h}}{d\mathbf{u}^L}\right]^T \mathcal{C}^T + \left[[\mathcal{S}^+] - \mathcal{A} [\mathbf{n}_x] \mathcal{C} [D^{-1}] [\mathcal{S}^+] \left[\frac{\Delta x}{4}\right] - \mathcal{A} [\mathbf{n}_y] \mathcal{C} [D^{-1}] [\mathcal{S}^+] \left[\frac{\Delta y}{4}\right] \right] \left[\frac{d\mathbf{h}}{d\mathbf{u}^R}\right]^T \mathcal{C}^T \tag{14}$$

Just as equations (12) and (13) have implementations using an edge data structure (one would *never* store the connectivity as $\mathcal{A}$ or $\mathcal{C}$ in dense matrix form), the transpose equation has an implementation using an edge data structure for the calculation of $\left[\frac{d\mathbf{R}}{d\mathbf{U}}\right]^T$. For example, the matrix operation $\mathcal{A}^T v$ performs a *gather and sum* of the two edge vertex values of v for each and every edge. The matrix operation $\mathcal{A}w$ performs a *scatter and accumulate* of an edge quantity w to the two edge vertices locations for each and every edge. Similar edge operations are performed for the directed matrix $\mathcal{C}$. Thus we have constructed a technique for matrix-vector products based on elementary edge operations which also permits constructing the transposed matrix-vector product. As we will see, the ability to write the entire algorithm in terms of a sequence of edge operations makes the parallel implementation straightforward.

An Example Two-Dimensional Calculation

In this section we demonstrate the 2-d performance of the implicit strategy for a computation of viscous flow with turbulence about a multiple-component airfoil geometry which has been triangulated using highly stretched mesh cells, see Figure 4. The experimental flow conditions are $M_\infty = .20$, $\alpha = 16°$, and a Reynolds number of 9 million. Experimental results are given in [WVG92].

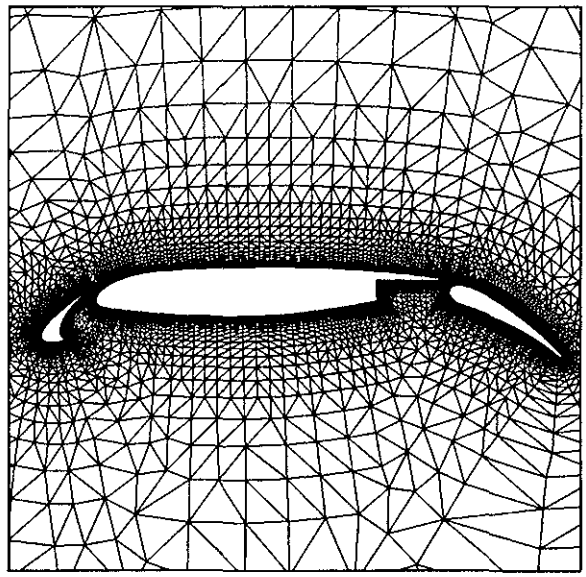

Figure 4: Steiner Triangulation of multiple-component airfoil, 38,000 mesh vertices, 190,000 degrees of freedom.

The numerical computation has been performed using linear least-squares reconstruction and containment dual control volumes. The computed solution is shown in Figure 5 and the surface pressure coefficient is graphed in Figure 6. The calculation requires about 2 minutes CPU time per global iteration on an IBM RS6000 workstation. Although the mesh does not fully resolve the wake-layer leaving the leading edge slat, the calculated surface pressures agree very well with experiment. The convergence history of Newton's method is shown in Figure 7. The computation began using a local time step corresponding to a maximum CFL number of about 100. The time step was exponentially increased to about 10 million as the residual norms decreased. This residual history shows the typical convergence behavior of flow computations which include a turbulence

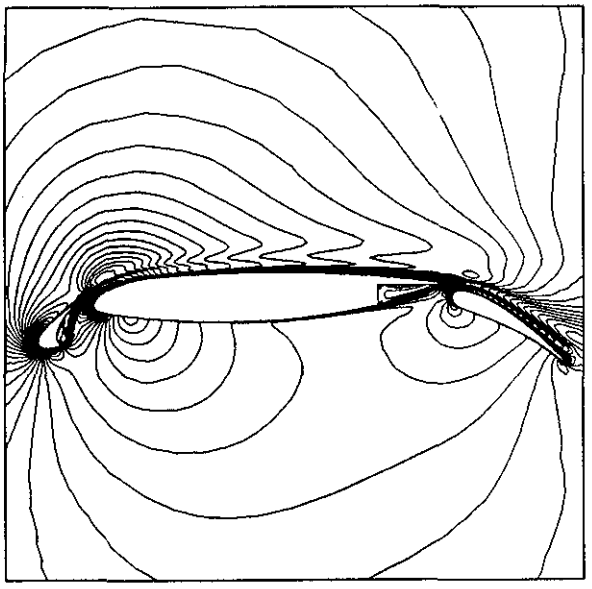

Figure 5: Mach number contours near the airfoil surface.

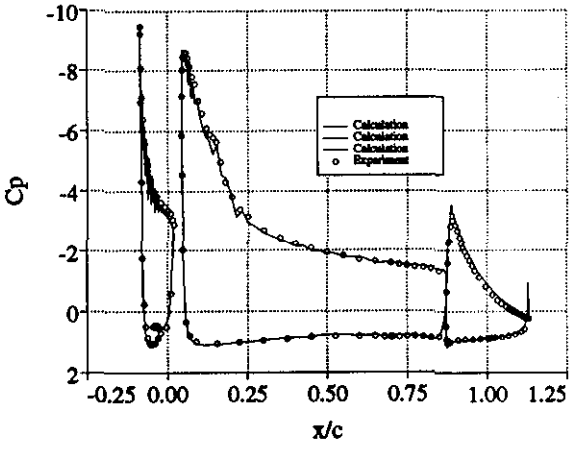

Figure 6: Surface pressure coefficient showing comparison with experiment.

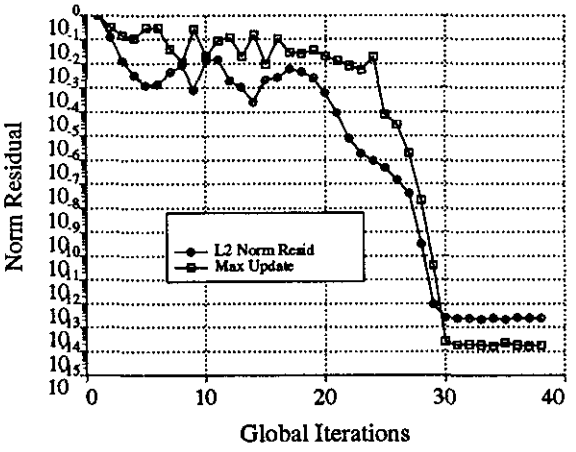

Figure 7: Residual History for Multiple Component Airfoil

model. Many iterations are necessary before sufficient levels of the turbulence variable are obtained, after which convergence is rapid. This is somewhat different from inviscid flow calculations (without discontinuities present) for which Newton's method can be rapidly approached.

Parallel Implementation in Three Dimensions

The implementation of the present numerical algorithm on parallel computers is important because of a need for their large memory and floating point capacity. In order to take full advantage of the parallel architecture, the implementation should be highly scalable. That is, the speedup of a computation on N processors should be as near to N as possible. The challenge is to implement the uniprocessor algorithm on parallel computers as faithfully as possible while minimizing interprocessor communication costs. As will be shown, the present algorithm is well suited for parallel implementation: good scalability is retained while maintaining all the favorable attributes of the uniprocessor algorithm.

Our current platform for parallel computation is an IBM SP2 computer located at the NASA Ames Research Center. The current configuration consists of 160 rack-mounted IBM 590 workstations with total memory capacity exceeding 20 gigabytes. Each processor has a peak theoretical speed of 250 megaflops. For these computations a single processor attains a sustained speed of about 55 megaflops. The processors are interconnected via a fast network

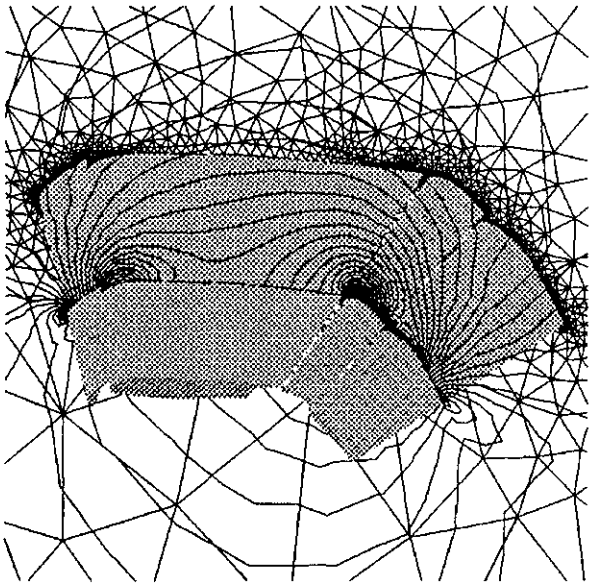

Figure 8: Inviscid flow $M_\infty = .20, \alpha = 0$ over a multiple-component wing geometry (600,000 degrees of freedom).

switch with measured bandwidth of approximately 33 megabytes/second and a measured latency of about 45 microseconds.

For maximum portability we have chosen MPI as the message passing protocol for implementation of the parallel Newton algorithm. Although our current platform for parallel computation is the IBM SP2, the MPI library is available for use on most other parallel computers and workstation clusters.

The overall strategy in the parallel implementation is to reduce the entire algorithm to a sequence of steps requiring only distance-one information on the triangulation. The greatly simplifies the implementation of the algorithm while still replicating uniprocessor results. The implementation contains several algorithmic elements. Each of these elements will be described in the following sections and elucidated using the realistic example of fluid flow over a multiple-component wing geometry. The wing geometry, symmetry plane mesh, and Mach contours at a midspan cutting plane are shown in Figure 8.

Mesh Partitioning

In the parallel algorithm, the mesh is *a priori* partitioned (divided) into N nonoverlapping subdomains, each of which resides on one of N processors. More precisely, mesh volumes (tetrahedra, hexahedra, prisms, etc) lie entirely on a given partition, triangulation vertices are repeated on partition

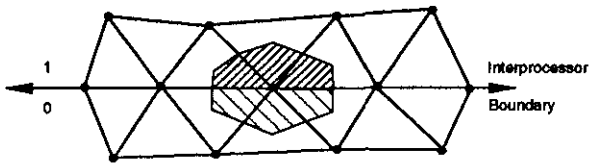

Figure 9: Portion of mesh spanning partition boundary showing control volume subdivision.

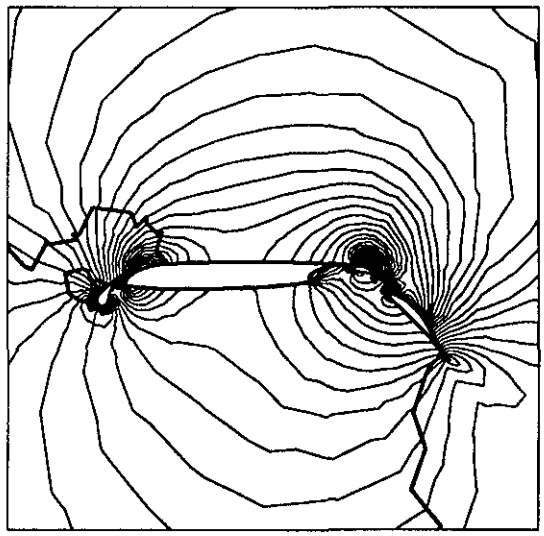

Figure 10: Mach Contours and Partition Boundary (bold lines).

boundaries, and control volumes for the finite-volume scheme span partition boundaries. The situation is depicted in Figure 9.

Although any partitioning strategy suffices, a good partitioning is one that approximately balances the computational load among the processors and minimizes the communication cost between processors. In the paper by Venkatakrishnan, Simon, and Barth [VVB92], we consider mesh partitioning algorithms based on recursive coordinate bisection, Cuthill-McKee, and spectral partitioning strategies. In general we found that the spectral partitioning method outperformed the others but at a higher partitioning cost. Figure 10 shows the mesh subdivisions (bold lines) induced by a spectral partitioning on the midspan cutting plane.

Computation of the Explicit Residual and Reconstruction Gradients

For control volumes completely contained within a single partition domain, the calculation of the resid-

ual is identical to the uniprocessor computation. For the control volumes subdivided by partition boundaries, integral conservation implies that

$$\int_{\Omega_0+\Omega_1} (\mathbf{f}\cdot\mathbf{n})\, dS = \int_{\Omega_0} (\mathbf{f}\cdot\mathbf{n})\, dS + \int_{\Omega_1} (\mathbf{f}\cdot\mathbf{n})\, dS. \quad (15)$$

In Figure 9, Ω_0 and Ω_1 correspond to the control volume portions on processors 0 and 1 respectively such that $\Omega = \Omega_0 \cup \Omega_1$. Therefore residuals can be computed on a processor-by-processor basis followed by an exchange and sum of residuals on interprocessor boundaries. This yields results identical to that obtained on a uniprocessor mesh.

A similar technique can be used for the calculation of reconstruction gradients. The Green-Gauss integral exhibits the same property as Equation (15) so that computations can be carried out on a processor-by-processor basis followed by an exchange and accumulate at interprocessor boundaries. Finally, the resulting accumulated values are divided by dual cell volumes, yielding results identical to a uniprocessor computation. The least-squares reconstruction technique also extends in a similar way if a bit mask is assigned to all edges in the mesh so that edges lying on processor boundaries contribute only once to the accumulation formulas.

GMRES Algorithm

The GMRES algorithm requires three basic operations: vector inner products, matrix-vector products, and preconditioning. The parallel implementation of each of these is described below. In our implementation of GMRES all processors solve the same small least-squares problem. This redundancy is of minor consequence.

Vector Inner Products

Redundancy of boundary vertices in vector inner products is eliminated with a mask bit preassigned to each vertex. The actual inner product is calculated by a local masked inner product followed by a global summation reduction (MPI_REDUCE).

Matrix-Vector Products

Previously, we discussed several strategies for computing matrix-vector products in the uniprocessor case. If Fréchet approximate derivatives are used, then the procedure is straightforward and uses exactly the same communication steps needed in computing the explicit residual. If exact matrix-vector products are desired, we store only the matrix associated with the distance-one neighbors on the triangulation and compute the remaining terms using Equation (11).

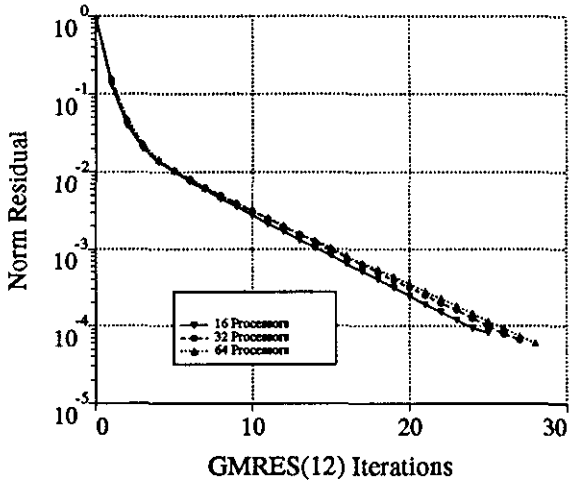

Figure 11: GMRES Iterations (restarts) required.

Processor Local ILU(0) Preconditioning

Our preconditioning matrix for the GMRES solver is based on the Jacobian matrix of the first-order accurate spatial discretization. This matrix has nonzero entries placed at distance-one locations in the connectivity graph. In a departure from the uniprocessor code, we compute and store on a processor nonzero entries in the matrix associated with mesh vertices residing on that processor. As a second step, the diagonal matrix blocks corresponding to interprocessor boundary vertices are exchanged and summed. This yields diagonal block entries in the resulting processor local matrix that are identical to the corresponding uniprocessor matrix. At the cost of increased interprocessor boundary vertex communication, all processor local matrix entries could be made identical to the uniprocessor matrix entries. The processor local matrix is ILU factored and used for preconditioning GMRES iterations. If the off-diagonal entries in rows corresponding to interprocessor boundary vertices are zeroed, then identical solutions at those vertices on all shared processors are obtained in the preconditioning step. If these matrix entries are retained then a unique value must be obtained from a linear combination of the multiple computed values. Our experience has shown that the local processor preconditioning does not significantly impact the effectiveness of the ILU preconditioning. In Figure 11, we show the convergence of GMRES(12) with local ILU(0) preconditioning on 16, 32, and 64 processors for the multiple-component wing calculation at a CFL number of about 20000.

Keep in mind that this departure from the unipro-

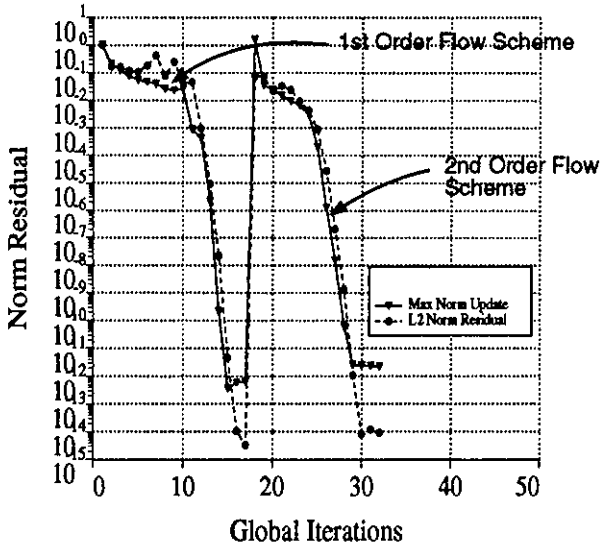

Figure 12: Convergence History for Inviscid Multiple Component Wing Case (16 Processors)

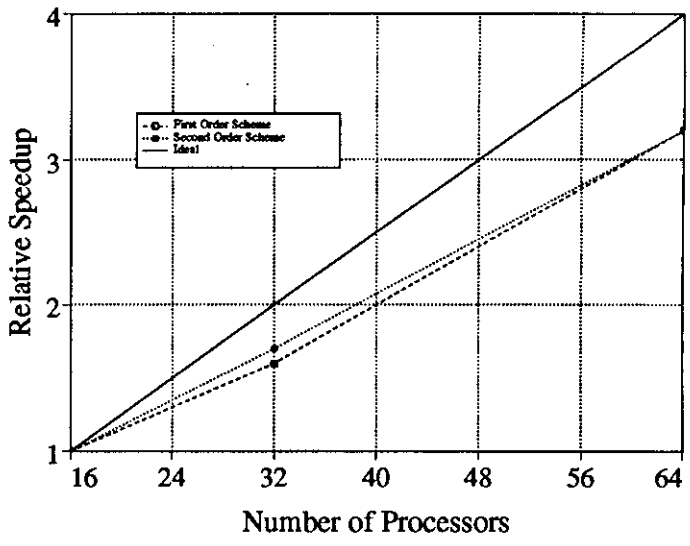

Figure 13: Relative speedup of parallel computations using 16, 32, and 64 processor. Each speedup is normalized by the 16 processor value.

cessor algorithm only affects the GMRES convergence and not the convergence of Newton's method. Figure 12 shows the convergence history of the Newton scheme for the first and second order accurate spatial discretization schemes.

Scalability

The scalability of the current parallel algorithm on the IBM SP2, while not excellent is certainly acceptable. This is particularly true since the parallel algorithm retains the favorable qualities of the uniprocessor algorithm, such as Newton-like convergence. Furthermore, because the parallel algorithm makes very few compromises in implementing the uniprocessor algorithm, the primary contribution to the degradation of scalability is the time taken by interprocessor communication. This implies that the scalability would be better on parallel computers with faster interprocessor communication.

Figure 13 shows the relative speedup of computations on the IBM SP2 for 16, 32, and 64 processors, for both the first order and second order schemes. Once again, the problem being solved is the inviscid flow about a multiple component wing, as described above. The speedups are normalized by the 16 processor value, since the memory requirements made 16 processors a minimum requirement to run the problem.

The table below shows the total wallclock time in minutes taken by the runs corresponding to Figure 13. In each case the second order scheme takes roughly 3.5 times as long as the first order scheme.

Wallclock Time for Multiple Component Airfoil Runs

Number of Processors	First Order Accurate Scheme	Second Order Accurate Scheme
16	50.0	176.0
32	31.0	103.0
64	15.6	55.0

Parallel Computation Results

In this section we will present results for a turbulent viscous computation about the multiple component wing described above. The Mach number of the run is 0.2, with a Reynolds number of 5 Million. The angle of attack is 8°. For this run, 64 processors are used. To minimize storage, the Jacobian matrices are stored using single precision (32 bits on the SP2), although all floating point operations are still performed in double precision.

The tetrahedral mesh about the body has roughly 400,000 vertices and over 2,000,000 tetrahedra. Because of the need to resolve the turbulent boundary layer, the mesh is highly stretched near the wall, with cell aspect ratios of more than 10,000. The mesh on the centerline plane is seen in Figure 14.

The Spalart and Allmaras turbulence model [SA92] is used to simulate the effect of turbulence on the

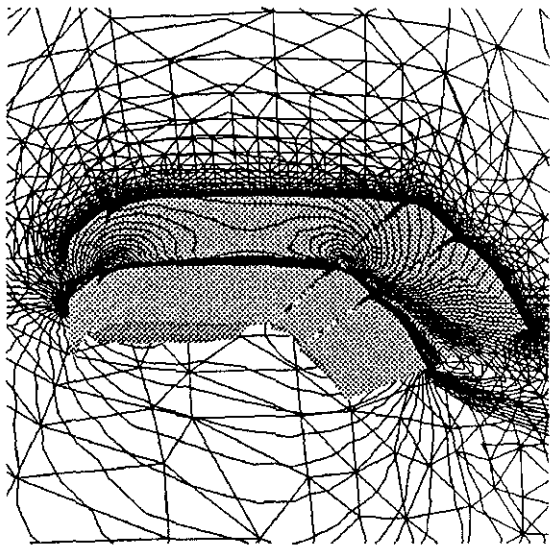

Figure 14: Viscous turbulent flow ($M_\infty = .20, \alpha = 8°$, $Re = 5$ Million) over a multiple-component wing. Mach contours are shown on the midspan cutting plane.

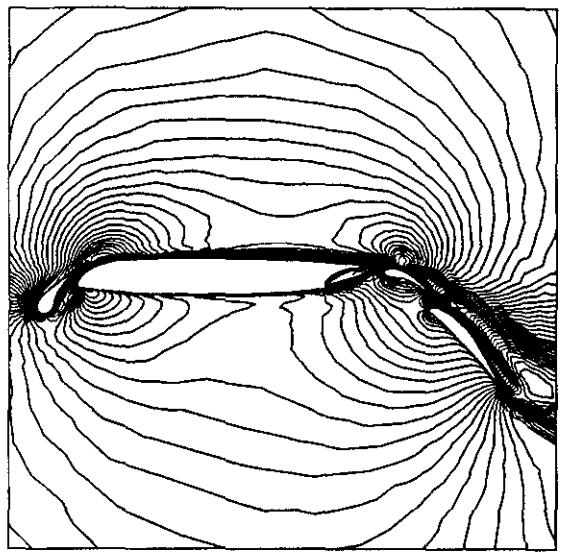

Figure 15: Mach Contours on the midspan cutting plane.

mean flow equations. Although the basic flow equations are solved using linear reconstruction, the turbulence model equation is solved using only first order advection. This is a common procedure used to increase robustness, even in structured mesh codes. The turbulence model is fully coupled with the flow equations in computing the Jacobians. This insures that Newton's method is approached at large timesteps even for turbulent computations.

Figure 15 shows the resulting Mach contours on a cutting plane placed at approximately mid-span. Note the qualitative agreement between these results and those obtained by the corresponding two-dimensional computation shown in Figure 5, albeit at a different angle of attack.

In Figure 16, contours of the eddy viscosity-like turbulence parameter $\tilde{\nu}$ defined earlier are depicted on the mid-span cutting plane. Note the high levels generated downstream of the main wing element over the aft flaps.

Presently, this computation takes about 10 minutes per step, and about 80 steps to converge to steady-state (a relatively large number for Newton's method). This is due to the slow development of turbulence over the wing. This situation is likely to improve in the near future, as we refine our technique for approaching steady-state and compute on a sequence of coarser meshes to accelerate the removal of the initial transient.

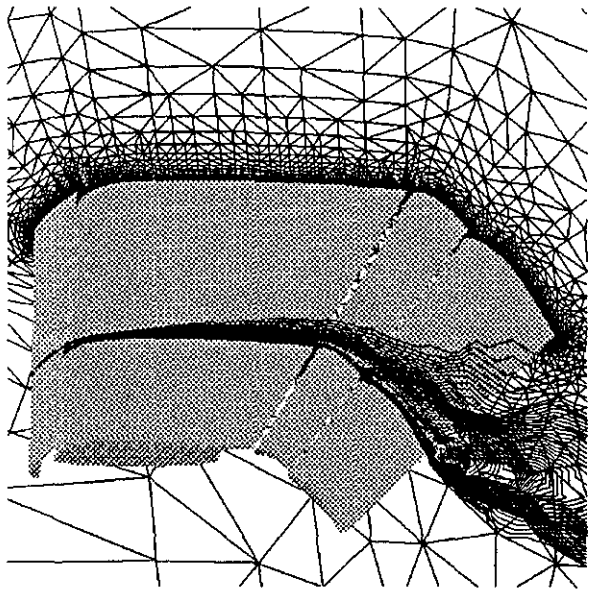

Figure 16: Turbulence quantity contours showing the buildup of turbulence over the aft flap.

Conclusions and Future Work

We have developed an algorithm for solving the Euler and Navier-Stokes equations on unstructured meshes including the effects of turbulence. We have designed a Newton solution strategy which provides a framework for extension to optimization and the study of fluid flow bifurcation events. A code has been implemented on a parallel architecture using a message passing protocol with favorable scalability characteristics and portability.

We believe that the efficiency of the overall procedure can be greatly improved by the use of multiple coarser meshes to accelerate the removal of transient solutions, particularly for turbulent flow computations. In addition, we believe the on-processor performance can be improved by efficient cache memory management on the IBM SP2. Finally, we plan to incorporate an optimization capability into the parallel code.

References

[AG94] M. Aftosmis and D. Gaitonde. *On the Accuracy, Stability and Monotonicity of Various Reconstruction Algorithms for Unstructured Meshes.* Technical Report AIAA 94-0415, Reno, NV, 1994.

[Bar87] T. J. Barth. *Analysis of Implicit Local Linearization Techniques for TVD and Upwind Algorithms.* Technical Report AIAA 87-0595, Reno, NV, 1987.

[Bar89] T. J. Barth. *Some Notes on Shock Resolving Flux Functions Part 1: Stationary Characteristics.* Technical Report TM-101087, NASA Ames Research Center, Moffett Field, CA, May 1989.

[Bar91] T. J. Barth. *A Three-Dimensional Upwind Euler Solver of Unstructured Meshes.* Technical Report AIAA 91-1548, Honolulu, Hawaii, 1991.

[Bar94] T. J. Barth. *Aspects of Unstructured Grids and Finite-Volume Solvers for the Euler and Navier-Stokes Equations*, March 1994. von Karman Institute Lecture Series 1994-05.

[BB90] B. S. Baldwin and T. J. Barth. *A One-Equation Turbulence Transport Model for High Reynolds Number Wall-Bounded Flows.* Technical Report TM-102847, NASA Ames Research Center, Moffett Field, CA, August 1990.

[BJ89] T. J. Barth and D. C. Jespersen. *The Design and Application of Upwind Schemes on Unstructured Meshes.* Technical Report AIAA 89-0366, Reno, NV, 1989.

[BS94] P. Brown and Y. Saad. Convergence Theory of Nonlinear Newton-Krylov Algorithms. *SIAM J. Optimization.*, 4:297–330, 1994.

[EW94] S. Eisenstat and H. Walker. Globally Convergent Inexact Newton Methods. *SIAM J. Optimization.*, 4:393–422, 1994.

[Joh92] Z. Johan. *Data Parallel Finite Element Techniques for Large-Scale Computational Fluid Dynamics.* PhD thesis, Stanford University, Department of Mechanical Engineering, 1992.

[MT93] M. Behr A. Johnson J. Kennedy S. Mittal and T. Tezduyar. Computation of Incompressible Flows with Implicit Finite Element Implementations on the Connection Machine. *Comput. Methods Appl. Mech Engrg.*, 108:99–118, 1993.

[Raj91] V. T. Rajan. *Optimality of the Delaunay Triangulation in* $\mathbf{R}^d$, 1991. Proceedings of the 7th ACM Symposium on Computational Geometry.

[Roe81] P. L. Roe. Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes. *J. Comput. Phys.*, 43, 1981.

[SA92] P. Spalart and S. Allmaras. *A One-Equation Turbulence Model for Aerodynamic Flows.* Technical Report AIAA 92-0439, Reno, NV, 1992.

[Ven94] V. Venkatakrishnan. *Parallel Implementation Unstructured Grid Euler Solvers.* Technical Report ICASE 94-4, January 1994.

[vL79] B. van Leer. Towards the Ultimate Conservative Difference Schemes V. A Second Order Sequel to Godunov's Method. *J. Comput. Phys.*, 32, 1979.

[VVB92] H. D. Simon V. Venkatakrishnan and T. J. Barth. A MIMD Implementation of a Parallel Euler Solver for Unstructured Grids. *J. Supercomput.*, 6:117–137, 1992.

[WVG92] R. McGhee W. Valarezo, C. Dominik and W. Goodman. *High Reynolds Number Configuration Development of a High-Lift Airfoil.* Technical Report AGARD Meeting In High-Left Aerodynamics 10-01, 1992.

[XN92] X. Xia and R. Nicolaides. Covolume Techniques for Anisotropic Media. *Numer. Math.*, 61:215–234, 1992.

[YS86] M. H. Schultz Y. Saad. GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems. *SIAM J. Sci. Stat. Comp.*, 7(3):856–869, 1986.