IMPLICIT SCHEMES AND PARALLEL COMPUTING IN UNSTRUCTURED GRID CFD

V. Venkatakrishnan
Institute for Computer Applications in Science and Engineering
MS 132C, NASA Langley Research Center
Hampton, VA 23681-0001
United States

Contents

1 Summary

The development of implicit schemes for obtaining steady state solutions to the Euler and Navier-Stokes equations on unstructured grids is outlined. Following a brief review of spatial discretization methods, the principal time discretization techniques that are available are reviewed. The techniques for unstructured grids are contrasted with those used for structured, body-fitted grids. Applications are presented that compare the convergence characteristics of various implicit methods.

Next, the development of explicit and implicit schemes to compute unsteady flows on unstructured grids is discussed. Methods to improve the efficiency of explicit methods for time-accurate computations are reviewed. The development of an implicit scheme that makes use of nonlinear multigrid techniques to compute unsteady flows is outlined. The resulting method allows for arbitrarily large time steps and is efficient in terms of computational effort and storage. The issue of mass matrix that arises with vertex-based finite volume schemes is addressed.

Lastly, the issues involved in parallelizing finite volume schemes on unstructured meshes in an MIMD (multiple instruction/multiple data stream) fashion are outlined. The techniques for partitioning unstructured grids among processors are discussed. Parallelism in the flow solvers is addressed next. As a candidate explicit scheme, a four-stage Runge-Kutta scheme is used to compute steady two-dimensional flows. Implicit schemes are also investigated to solve these problems, where the linear system that arises at each time step is solved by preconditioned iterative methods. The choice of the preconditioner in a distributed-memory setting is discussed. The methods are compared both in terms of elapsed times and convergence rates. It is shown that the implicit schemes offer adequate parallelism at the expense of minimal sequential overhead. Following domain decomposition ideas, the use of a global coarse grid to further minimize this overhead is also investigated. The schemes are implemented on distributed-memory parallel computers. Finally, some dynamic load balancing ideas, which are very useful in adaptive transient computations, are presented.

2 Governing equations

The equations governing compressible fluid flow in integral form for a control volume $\mathcal{V}(t)$ with boundary $\mathcal{S}(t)$ are given by

$$\frac{\partial}{\partial t}\int_{\mathcal{V}(t)} W\,dv + \oint_{\mathcal{S}(t)} [F(W,\boldsymbol{n},\boldsymbol{s}) - G(W,\nabla W,\boldsymbol{n})]da = 0, \tag{1}$$

where

$$\begin{aligned} W &= [\rho, \rho\boldsymbol{V}, \rho e]^T \\ F(W,\ \boldsymbol{n},\boldsymbol{s}) &= (\boldsymbol{V}-\boldsymbol{s}).\boldsymbol{n}W \\ G(W,\nabla W,\ \boldsymbol{n}) &= [0,\ \boldsymbol{t}, \boldsymbol{t}.\boldsymbol{V} - \boldsymbol{q}.\boldsymbol{n}]^T, \\ \boldsymbol{t} &= \boldsymbol{n}.\overline{\overline{T}} \\ \overline{\overline{T}} &= \left[(-p+\lambda\Theta)\overline{\overline{I}} + 2\mu\overline{\overline{D}}\right] \end{aligned}$$

In the formulas given above ρ is the density, $\boldsymbol{V}$ is the velocity vector with Cartesian components V_i, e is the specific total energy, $\boldsymbol{n}$ is the outward unit normal vector of the boundary $\mathcal{S}(t)$ and $\boldsymbol{s}$ is the velocity vector of the boundary. Also, μ is the molecular viscosity, λ is the bulk viscosity related to μ by Stokes' hypothesis, $\lambda = -2/3\mu$, $\overline{\overline{I}}$ is the identity tensor, $\overline{\overline{T}}$ is the stress tensor and $\overline{\overline{D}}$ is the deformation tensor given by

$$D_{ij} = \frac{1}{2}(V_{i,j} + V_{j,i}) \tag{2}$$

where $V_{i,j}$ denotes the partial derivative of the ith component of $\boldsymbol{V}$ with respect to the Cartesian coordinate x_j, i.e., $V_{i,j} = \frac{\partial V_i}{\partial x_j}$. Θ stands for the divergence of $\boldsymbol{V}$ given by $V_{i,i}$ with the usual summation convention. $\boldsymbol{q}$ is the heat flux given by Fourier's law

$$\boldsymbol{q} = -K\nabla T, \tag{3}$$

where K is the thermal conductivity of the fluid and T is the temperature. These equations are augmented by the equation of state, which for a perfect gas is given by

$$p = (\gamma - 1)(\rho e - \frac{1}{2}\rho|\boldsymbol{V}|^2) \tag{4}$$

Eqn. (1) represents the conservation laws for the mass, momentum (the Navier-Stokes equations) and energy. It holds for any volume and in particular, holds for a specific volume associated with each grid point or a cell, termed *the control volume*.

3 Spatial discretization methods

The computational domain is first tessellated using a grid composed of *simplices*, which are triangles in two dimensions and tetrahedra in three dimensions. Unstructured grids provide flexibility for tessellating about complex geometries and for adapting to flow features, such as shocks and boundary layers. They are generated by using any of a number of techniques reviewed in [118, 82]. It is also possible to use hybrid grids, which include in addition structured, body-fitted quadrilateral grids in two dimensions and prismatic grids in three dimensions in the vicinity of the solid boundaries for resolving viscous regions as opposed to using stretched triangles and tetrahedra. In three dimensions, the prismatic grids near the boundaries are typically generated by marching-out procedures [93, 67].

On a given grid, one has the option of locating the variables at the cell centers or at the vertices of the grid, giving rise to cell-centered and cell-vertex schemes. Alternatively, it is possible to deal strictly with averages defined over volumes [11, 27]. This approach has certain advantages for higher order schemes, but is not considered in the present work. In the case of finite volume schemes, the governing equations in integral form (Eqn. (1)) are discretized. This allows discontinuities to be captured as part of the solution. Eqn. (1) expresses the rate of change of the conserved quantities (mass, components of momentum and energy) to be the negative of the net flux out of the control volumes. This net flux through the control volume boundary is termed the *residual*. For steady-state computations, this residual vanishes over all control volumes. Starting from an initial guess, typically freestream conditions, Eqn. (1) is marched in time until the solution W does not change. Thus global conservation in space is guaranteed because of cancellation of the interior fluxes, resulting in flux contributions only at the physical boundaries. The control volume for a cell-centered scheme is typically the triangle or the tetrahedron itself, whereas for a cell-vertex scheme it is taken to be the median dual, composed of segments of the median. The control volumes and the stencils associated with first order cell-centered and cell-vertex schemes in two dimensions are illustrated in Figures 1 and 2. The definition of the median dual as the control volume in a cell-vertex scheme comes about by showing the equivalence with a Galerkin finite element method employing piecewise linear basis functions while computing the gradient of a function [107, 9]. An alternative definition of a control volume for a cell-vertex scheme is a Voronoi cell which is composed of perpendicular bisectors drawn between pairs of grid points. The Voronoi cell of a site (grid point) is defined as that region of space containing points which are closer to the site than to any other site. The dual to the Voronoi tessellation is the well-known Delaunay triangulation. The Voronoi cell is guaranteed to be convex, but the approach imposes a Delaunay triangulation and requires care at the boundaries. Another alternative is the containment dual [142, 13] which has nice properties for stretched triangulations.

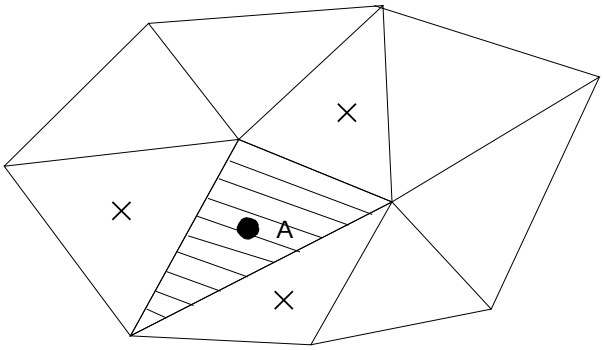

Figure 1: Control volume and stencil for a first order accurate cell-centered scheme.

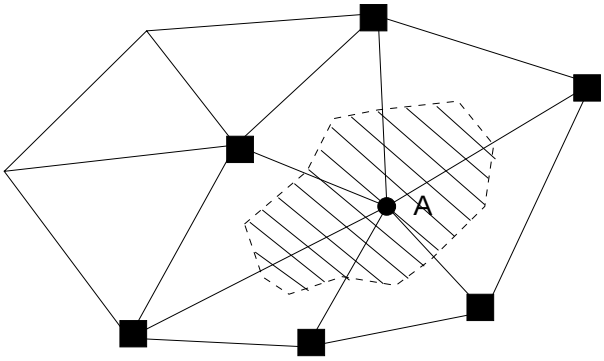

Figure 2: Control volume and stencil for a first order accurate cell-vertex scheme.

An excellent review of spatial discretization methods for advection-dominated flows may be found in [1]. Here we will briefly outline some of the popular methods. The interest here will be confined to the bearing they have on the choice of time integration methods. In [64, 79] a Galerkin finite element scheme using piecewise linear basis functions is augmented with a blend of dissipative terms made up of Laplacian and biharmonic terms. The first order Laplacian term acts only in the vicinity of shocks whereas the third order biharmonic term used away from shocks serves as background dissipation to eliminate odd-even decoupling. This scheme can be thought of as an extension of the scheme of Jameson et al. [65] to unstructured grids. A space-time formulation has been derived by Donea [35] called the Taylor-Galerkin family of schemes for the linear advection equation. Adopting an FEM approach, he has shown how a Galerkin scheme (a centered scheme) could be stabilized by using a Taylor-series expansion for the time derivative $\frac{\partial u}{\partial t}$, similar to the procedure used to derive the Lax-Wendroff scheme. He demonstrated that the resulting schemes had good phase error and dissipation properties and that they could be easily extended to multi-dimensions. Morgan et al. in [1] have used such a procedure for discretizing the Euler equations. Another method of realizing higher order accuracy is provided by adopting the MUSCL formulation of Van Leer [121] as outlined by Barth and Jespersen [12] for unstructured grids. In this approach, first a piecewise polynomial reconstruction is performed from the given data within each control volume and the polynomial is then interpolated to the its faces. The jumps that occur at the control volume faces are reconciled by using an approximate Riemann solver such as Roe's solver [106]. During the reconstruction stage, monotonicity principles are invoked so that oscillation-free solutions can be obtained. Another way to design higher order accurate schemes is to construct schemes of the form

$$\frac{du_i}{dt} = \sum_{j \in \mathcal{N}_i} C_{ij}(u_j - u_i), \tag{5}$$

where the C_{ij}'s are non-negative, and $\mathcal{N}_i$ denotes the set of neighbors of i. It is easy to see that under these conditions, the maxima can never increase and the minima can never decrease. If the C_{ij}'s are constants, the scheme is linear and hence, only first order accurate. Higher order versions of such schemes have been developed [38, 28, 63] which add limited amounts of anti-diffusive fluxes to the lower order fluxes. Higher order accuracy in the case of cell-vertex schemes can also realized by using the fluctuation-splitting approach [31, 96]. These schemes consider the average time evolution of a complete cell (a triangle or a tetrahedron) with the unknowns located at its vertices, and then update the values at the vertices by the effect of linear wave solutions evolving the piecewise linear data over a cell. Finite element methods, such as SUPG and Galerkin least-squares methods [60, 59], and discontinuous Galerkin methods [73] also permit higher order accuracy to be realized by utilizing appropriate basis functions.

It is assumed that a spatial discretization has been performed by using one of the approaches outlined above. The typical stencils for the higher order cell-centered and cell-vertex schemes on

triangular meshes are illustrated in Figures 3 and 4. The stencil for the cell-vertex scheme involves all the nearest-neighbors and the next-to-nearest neighbors. Whereas a first order cell-centered scheme involves only the three nearest neighbors, note that the higher order scheme employs a stencil that is comparable to that of a cell-vertex scheme [12, 48]. The more compact stencils for cell-centered schemes outlined in [130, 38] have difficulties dealing with general triangulations. The viscous terms are discretized in cell-vertex schemes by using a Galerkin finite element approximation and only involve the nearest neighbors. In the case of cell-centered discretizations, Frink et al. [48] compute the first derivatives at the vertices of the triangulation, which are then averaged to obtain the viscous fluxes at the faces. The stencils shown in Figures 3 and 4 are thus unaltered when (laminar) viscous terms are included. One of the advantages of the fluctuation-splitting approach is that the stencil for the second order accurate scheme only involves the nearest neighbors.

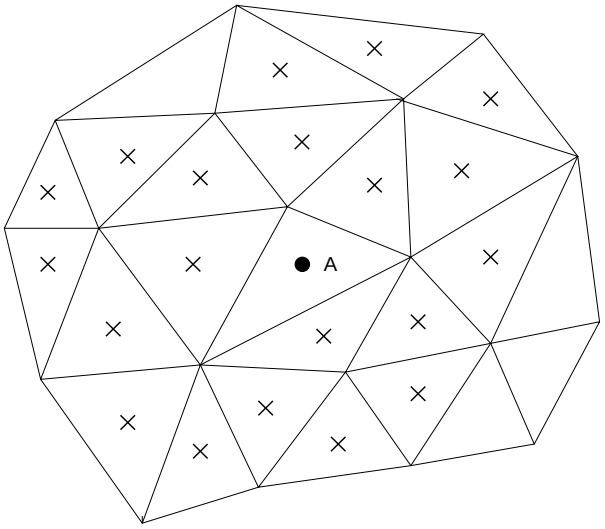

Figure 3: Stencil for a second order accurate cell-centered scheme.

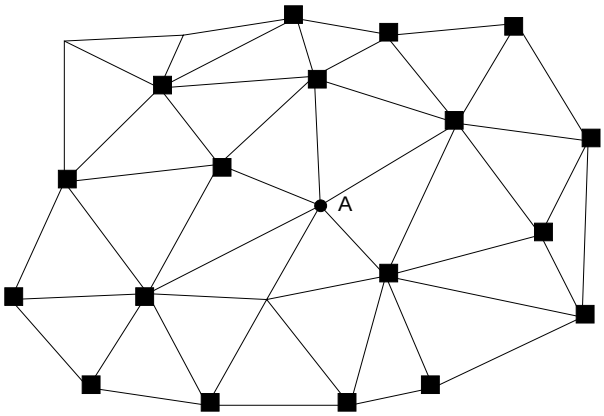

Figure 4: Stencil for a second order accurate cell-vertex scheme.

All the spatial discretization methods outlined above only utilize data from a local neighborhood in order to compute the residual at a point/cell. Thus the operations involved in computing the residual involve compact stencils. The residual is computed by summing the fluxes. This operation can be vectorized in one of two ways. This will be explained in the context of cell-centered schemes in three dimensions. The first method computes the fluxes at the triangular faces and stores them. This is followed by a loop over the tetrahedral cells to accumulate the fluxes to form the residuals. This scheme would require face-to-cell and cell–to–face pointers. Alternatively, it is possible to calculate the fluxes and accumulate the residuals in one loop over the faces, which would only require face-to-cell pointers. This operation can be vectorized by coloring the triangular faces which groups the faces such that no two members of the group point to the same cell. With an outer (sequential) loop over the number of colors, the inner loop over the members of the color can be vectorized. In the case of cell-vertex schemes, Barth [10] and Mavriplis [81] have shown that even in three dimensions, the computation of the residuals can be cast as loops over edges in inviscid computations. Therefore, edges are colored in both two and three dimensions in the case of cell-vertex schemes so that no two edges within the same group point to the same vertex. It is important to keep the number of colors as small as possible to maximize vector lengths and minimize the number of vector startups. Coloring is a problem in graph theory. See section 6.1 for some definitions. The minimum number of colors required to color the edges of a graph is Δ or $\Delta + 1$, where Δ is the maximum degree of a vertex in the graph. This is known as Vizing's theorem [50]. For graphs arising from grids, the minimum number of colors is Δ. The graph for a cell-vertex scheme is the grid itself, whereas for the cell-centered scheme it is the dual graph,

where each tetrahedron is represented by a vertex and each triangular face is represented by an edge. The minimum number of colors required in a cell-vertex scheme is then Δ, which can be arbitrarily large. In a cell-centered scheme, on the other hand, the minimum number of colors is 3 in two dimensions and 4 in three dimensions.

4 Steady state solution techniques

After discretization in space, the following system of coupled ordinary differential equations results:

$$\frac{d(VMW)}{dt} + R(W) = 0. \tag{6}$$

Here W is the vector of unknowns over the grid points for a vertex-based formulation and over cells for a cell-based formulation, and V is the volume of the polyhedral control volume associated with the grid point/cell. In the case of a cell-vertex scheme, M is the mass matrix which represents the relationship between the average value in a control volume and the values at the vertices (the vertex representing the control volume and its nearest neighbors). It arises from adopting a strict finite volume viewpoint since the update as given by the residual should be made to the time derivative of the average value within a control volume. It is only a function of the mesh and hence, a constant matrix for a static mesh. If a steady state solution is sought, time-accuracy is not an issue and M can be replaced by the identity matrix. This technique, known as mass-lumping, yields the following system of ordinary differential equations for the vector of unknowns W :

$$V\frac{dW}{dt} + R(W) = 0. \tag{7}$$

Assuming that the grid is static, cell-centered schemes (up to second order accuracy) and schemes dealing strictly with cell-averages (to any order of accuracy) do not yield a mass matrix and thus lead to Eqn. (7) for steady and unsteady problems. The effect of the mass matrix in time-accurate flow simulations is addressed in Section 5.1. Time-space formulations, such as the Taylor-Galerkin schemes, do not lead to the system of ODE's of Eqn. (6). Rather, they result in coupled system of nonlinear equations which involve previous time levels depending on the order of accuracy of the schemes. The easiest way to solve these equations is by using explicit methods, discussed in the next section.

4.1 Explicit schemes

The simplest means of integrating the system of ODE's in Eqn. (7) is by the use of the explicit schemes. In their simplest form, the time derivative is discretized using a finite difference formula at time step n, and the residual $R(W)$ is evaluated at time step n as well. Thus, explicit schemes possess the advantage of requiring only simple updates. Given that the residual computation is vectorizable, explicit schemes are therefore vectorizable (parallelizable as well). For example, using forward differences in time, we obtain:

$$V\frac{W^{n+1} - W^n}{\Delta t} + R(W^n) = 0. \tag{8}$$

Higher order accurate difference formulas that make use of previous time levels may also be used for discretizing the time derivative at time step n. For reasons of stability, it is necessary to consider a sequence $W^0, W^1, ..W^N, N \rightarrow \infty$ in Eqn. (8). The topic of stability of difference schemes is covered extensively in texts. Therefore, it is assumed, that a proper discretization of the time derivative has been performed so that the resulting scheme is stable. Typically, the nondimensional time step permitted by explicit schemes is $O(1)$. Perhaps the most popular method used to integrate

the system of ODE's in CFD is the Runge-Kutta method. The classical form of Runge-Kutta methods requires solutions from previous levels to be stored. The following form of an $m-$stage Runge-Kutta scheme is preferred for solving the system of ODES's given by Eqn. (7):

$$\begin{aligned} Q_0 &= W^n \\ .... & \\ VQ_k &= VQ_0 - \alpha_k \Delta t R(Q_{k-1}) \\ .... & \\ VQ_m &= VQ_0 - \alpha_m \Delta t R(Q_{m-1}) \\ W^{n+1} &= Q_m \end{aligned} \tag{9}$$

For consistency, we require $\alpha_m = 1$ and the scheme is second order accurate in time for linear and nonlinear problems if $\alpha_{m-1} = 1/2$. The coefficients α_k's can be optimized to expand the stability region so as to allow for the maximum time step with a particular spatial discretization [61]. They can also be optimized so that the resulting explicit scheme acts as a good multigrid smoother [61, 123].

4.1.1 Acceleration techniques

For steady-state computations, several acceleration strategies are often employed in concert with explicit schemes. Local time stepping allows each cell to take the maximum possible time step; as a result the scheme is no longer consistent in time. It is also possible to use characteristic time stepping. Viewing the Euler equations as a superposition of waves, each wave is allowed to propagate at its own maximum allowable time step. This is highly effective in one dimension. Van Leer et al. [122] have extended this concept to deal with two-dimensional Euler equations. Another technique is to introduce a moderate degree of implicitness in the scheme by using the technique of residual averaging [61]. Residual averaging is covered in Section 5.2. It leads to a system of linear equations that is solved by a few Jacobi iterations on unstructured grids [79]. Finally, for constant-enthalpy steady solutions it is possible to use the technique of enthalpy damping provided that the spatial discretization allows for a constant-enthalpy solution [61].

In spite of these acceleration techniques, explicit schemes are not competitive. Convergence to steady state is usually unacceptably slow, especially as the problem sizes and complexities grow. Therefore, either multigrid methods or implicit schemes are required to accelerate the convergence. Multigrid methods for unstructured grids are covered in the chapter by Mavriplis.

4.2 Implicit schemes

We return to Eqn. (7) which represents a system of ODE's. $R(W)$ is in general a nonlinear vector of the components of the solution vector W:

$$V\frac{dW}{dt} + R(W) = 0. \tag{10}$$

After employing a backward Euler discretization in time we obtain:

$$V\frac{W^{n+1} - W^n}{dt} + R(W^{n+1}) = 0. \tag{11}$$

Linearizing R about time level n, we obtain

$$\begin{aligned} \left[\frac{V}{\Delta t} + \frac{\partial R}{\partial W}\right] \Delta W &= -R(W^n) \\ \Delta W &= (W^{n+1} - W^n) \end{aligned} \tag{12}$$

Eqn. (12) represents a large sparse linear system which needs to be solved at each time step. As Δt tends to infinity, the method reduces to the standard Newton's method which yields quadratic convergence for isolated roots of the nonlinear system. The term $\frac{\partial R}{\partial W}$ symbolically represents the implicit side upon linearization and involves the Jacobian matrices of the flux vectors with respect to the conservative variables. Based on the work of Mulder [92], the time step in Eqn. (12) is allowed to vary inversely proportional to the L_2 norm of the residual, so that the time step increases rapidly as the steady state solution is approached.

In the case of structured, body-fitted grids, the linear system Eqn. (12) is seldom solved; indeed it is seldom even assembled. Instead, approximations are made to the linear system itself. For example, the Alternating Diagonal Implicit (ADI) method [23] and Approximate Factorization (AF) [19] result in a product of simpler factors. Each of the factors can be easily solved by direct methods. For example, ADI results in nested block tri-diagonal systems which are solved by Thomas algorithm. In addition, lower order discretizations and approximations to the flux Jacobians are often employed in approximating the left hand side. As a result of these approximations, these methods are only moderately implicit. Typically they allow for CFL number (the nondimensional time step) of the order of 10–100.

In the case of unstructured grids, one has no recourse to techniques such as AF or ADI. Instead one has to deal with the solution of the sparse linear system. The system of equations can be solved by direct or iterative means. We will first review the direct methods applicable to general sparse matrices.

4.2.1 Direct methods

The linear system of Eqn. (12) has been solved by Gaussian elimination for the two-dimensional compressible Euler and Navier-Stokes equations using structured grids [126, 17, 20, 95]. For logically rectangular regions, the matrix assumes a banded form, that can be solved efficiently by banded solvers or by employing direct methods for general sparse matrices. The complexity for a $n \times n$ grid with a banded solver is $O(Nm^2)$ where m is the half-bandwidth and $N = n^2$ is the number of grid points. For example, if a first order discretization is employed, the stencil involves only the nearest neighbors and therefore, $m = n$. Thus, the number of operations is $O(n^4)$. The storage required is $O(n^3)$ since for a banded matrix, almost the entire region between the bands gets filled in during the Gaussian elimination. Some sparse-matrix methods result in more favorable operation counts and require less storage because they try to minimize fill-in. Nested dissection [49], for example, only requires $O(n^3)$ operations and $O(n^2 log_2 n)$ storage for this problem. By solving the linear system using direct methods, quadratic convergence has been demonstrated for compressible Euler and Navier-Stokes equations. Techniques to improve the convergence process such as grid-sequencing to establish a good initial guess and super-convergence are outlined in [126]. Newton methods have been used to solve difficult problems which are not amenable to conventional solution techniques such as hysterisis phenomenon with airfoils [17] and high Reynolds number laminar flows [18]. The Jacobian matrix $\frac{\partial R}{\partial W}$ can be evaluated analytically or numerically. The latter option, while expensive, is attractive when dealing with nondifferentiable functions such as algebraic turbulence models [17]. However, for most flux functions and field turbulence models, the Jacobian evaluation can be done either analytically or by using symbolic packages.

On unstructured grids, direct solvers have been utilized in two-dimensions to solve the compressible Navier-Stokes equations [130]. Use was made of the minimum-degree algorithm [49] to minimize the fill-in during the factorization. It is also possible to use other techniques such as spectral nested dissection [101] for this purpose. In [130] a cell-centered second order scheme was used that only made use of 10-point stencils. A stencil such as the one shown in Figure 4 would require far more computational effort and storage.

In summary, direct methods for compressible flow solvers are limited to two dimensions. Pro-

hibitive computational costs and large memory requirements severely limit the usefulness of the method in three dimensions. In addition, the gain that can be realized with techniques such as nested dissection are not as great in three dimensions as in two dimensions. However, direct methods in can be used to study problems in two dimensions that are difficult to solve by other means. The fact that unstructured grids have larger stencils compared to structured grids make direct methods even less attractive.

4.2.2 Standard iterative methods

Iterative methods are often used to solve the linear system given by Eqn. (12). Due to storage considerations and computational complexity, typically only a lower order representation is employed in the left hand side of Eqn. (12). This matrix is also better conditioned compared to the higher order discretization. A consequence of this approximation is that the resulting can never approach Newton's method (with its associated quadratic convergence property) due to the mismatch of the right- and left-hand side operators. Therefore, it does not pay to solve the linear system well either. Since there is a mismatch of operators in Eqn. (12), it is also necessary to limit the maximum time step.

For solving the linear system

$$AX = b, \tag{13}$$

the matrix A is first split as $A = M + N$ so that

$$(M + N)X = b. \tag{14}$$

A general iterative method is obtained as follows:

$$MX^{k+1} = -NX^k + b. \tag{15}$$

or equivalently,

$$M(X^{k+1} - X^k) = -AX^k + b = -r^k, \tag{16}$$

where $r^k = AX^k - b$ is called the residual for the linear system at kth step. The matrix M should be close to matrix A in some sense while being easily invertible. Several well-known iterative methods are obtained by making appropriate choices for M:

1. $M = I$ – Richardson's method
2. $M = D$, D being the diagonal – Jacobi iteration.
3. $M = D + E$, D being the diagonal and E, the lower triangular part of A – Gauss-Seidel iteration.
4. $M = D + E$ step followed by $M = D + F$, F being the upper triangular part of A – Symmetric Gauss-Seidel.

Variants of these basic schemes can be obtained by using relaxation factors. For unstructured grids, all these techniques can be easily applied. Fezoui [43], Batina [15] and Slack et al. [113] have used a Gauss-Seidel relaxation technique. It is also possible to generalize a red-black Gauss-Seidel technique for unstructured grids leading to the multi-color Gauss-Seidel method. After coloring the vertices (for cell-vertex schemes) and cells (for cell-centered schemes) the Gauss-Seidel algorithm is applied to all the unknowns in each color. As the colors are processed sequentially, use is made of the latest available values of the neighbors. This algorithm has been used by Anderson [5] to compute compressible Euler flows on unstructured grids.

For efficient implementation on vector computers, the kernels that are of interest in Eqn. (16) are the evaluation of b which is the negative of the the residual vector R^k in Eqn. (12), the matrix

vector product AX and the inversion of M. The vectorization of the residual has already been covered in Section 3. The matrix vector product AX can be vectorized as explained in Section 4.3. Regarding the inversion of M, vectorization for the Richardson's method and Jacobi procedure is trivial whereas vectorization for the Gauss-Seidel method can be achieved by using wavefront ordering [4] also described in Section 4.3. Vectorization of the multi-color Gauss-Seidel method is straight-forward since all the entities (cells or vertices) belonging to the same color can be processed simultaneously.

4.2.3 Line-implicit methods

As mentioned earlier, line-implicit methods are among the most successful implicit methods in use for structured grids. However, ADI and AF are not appropriate for grids that possess no structure. Hassan et al. [57] have developed a line-implicit procedure which we now describe for a cell-vertex scheme. A line or a set of lines is passed through the grid such that each line passes through each vertex exactly once. In graph theoretic terms, such a path or circuit which visits each vertex exactly once is called a *Hamiltonian* tour [50]. For a given graph, there may exist many Hamiltonian tours or none at all. Graphs arising out of triangulations in two and three dimensions appear to permit multiple tours. Hassan et al. [57] have developed an efficient incremental algorithm to find these lines. They specify in addition, the orientations of the lines, so that in two-dimensional applications two lines result, one being 'vertical' and the other being 'horizontal'. The algorithm is then made implicit along each line, thus yielding a tridiagonal approximation for matrix M in Eqn. (15) which can be easily solved. At each time step, Morgan et al. in [1] only perform one iteration in Eqn. (15) for each line. Vectorization is an issue for this scheme since there is no nesting of tri-diagonal systems as in ADI methods. It is possible to achieve vectorization by using a cyclic reduction method [52] at the expense of higher operation counts. Morgan et al. in [1] have demonstrated the effectiveness of this algorithm over an explicit method for solving inviscid problems in two dimensions.

Figure 5(a): A 'snake' with horizontal orientation for an unstructured grid.

Figure 5(b): Linelets with horizontal orientation.

Martin and Löhner [77] refer to the Hamiltonian tours as 'snakes'. They have observed that there is a significant folding of these lines. This means that the flow of information in the predominant direction may be slowed down. Rather than use multiple lines as done by Hassan et al. [57], in

order to obtain a steady flow of information they reconnect the line in the direction of interest. Thus the line is broken into multiple linelets and the scheme is made implicit along these linelets. Figure 5(a) and 5(b) taken from [77] illustrate a 'snake' and linelets, aligned roughly with the horizontal direction. Martin and Löhner describe a general algorithm that creates these linelets on unstructured meshes. However, the matrix no longer has a tri-diagonal structure and therefore, they carry out a direct factorization, processing multiple linelets simultaneously to improve the performance on vector computers. They report that the vector performance is still unacceptably low. They have used this technique as a preconditioner for a conjugate gradient method when solving the Poisson equation for pressure in incompressible flows.

One shortcoming of the line-implicit methods is that the direction of propagation of information is predetermined. Although the sensitivity to the orientation can be alleviated by using multiple lines with different orientations, the approach is not satisfactory because the flow of information, in general, varies locally. The methods advocated in the chapter by Mavriplis, may be attractive in this regard. If the residual R_i at vertex i can be expressed as

$$R_i = \sum_{j \in \mathcal{N}_i} C_{ij}(u_j - u_i), \tag{17}$$

with $C_{i,j} \geq 0$, where $\mathcal{N}_i$ is the set of neighbors of i, the coefficients $C_{i,j}$ are interpreted as edge-coefficients that signify the strength of the connection between vertices i and j. It may be possible to use this information to decide how to group vertices so that these can be treated implicitly. It may also be possible to extend these ideas to systems of equations, where $C_{i,j}$ becomes a matrix.

4.2.4 Incomplete LU factorization methods

A family of iterative schemes arises out of an incomplete LU factorization and is referred to as ILU(n) [88]. A symbolic viewpoint is adopted in that LU factorization is carried out with a prespecified nonzero pattern. During the factorization, all the entries that fall outside of this pattern are ignored. Here n represents the level of fill-in. $n = 0$ implies no fill-in beyond the original nonzero pattern. $n = 1$ refers to a case where fill-in caused by the original nonzero pattern is allowed but not the fill-in caused by the newly filled-in entries. In practice, $n = 0$ is often used especially for general sparse matrices since it is quite robust and has lower storage requirements. Note that in terms of the general iterative framework, this scheme does not explicitly define matrix M in Eqn. (15); rather, the ILU factorization directly defines M^{-1}. The Symmetric Successive Over Relaxation (SSOR) iterative method with the relaxation factor set to 1 looks exactly like the ILU(0) scheme, except that the lower and the upper factors are read off directly from the matrix A rather than by an incomplete factorization. Incomplete factorization is a nonvectorizable procedure (although parallelizable by using wavefront ordering described later); SSOR method dispenses with this sequential procedure. ILU factorization and SSOR as iterative techniques by themselves will be tested for solving the linear sub-problems at each time step.

In contrast to the symbolic factorization viewpoint adopted in the definition of ILU given above, a second method of obtaining incomplete factorization relies on the numeric values of the entries. Gaussian elimination is carried out and entries are dropped if they fall below a certain threshold [144]. In general, it is more expensive compared to the symbolic procedure. Saad [108] has proposed a method that combines the two approaches which makes use of simpler data structures and is also less expensive.

4.2.5 Advanced iterative methods

Multigrid methods. Multigrid methods can also be used to solve Eqn. (12). Multigrid methods require the operators to be defined on a sequence of coarser grids, an iterative method that evolves the solution (called a smoother) and interpolation operators that transfer information between

the grids. The principle behind the algorithm is that the high-frequency errors are damped by the smoother on a given grid, whereas the low frequency errors are damped on coarser grids, where these frequencies manifest themselves as high frequencies. In the case of unstructured grids, the coarse grids can either be formed independently [79] or by using agglomeration methods [71, 114, 132] by fusing fine grid control volumes. The linear system is either determined by rediscretization or by combining the fine grid equations as in algebraic multigrid. Multiple Jacobi or Gauss-Seidel iterations perform the role of a smoother. Lallemand et al. [71] have used the agglomeration procedure and Anderson [5] has used the nonnested multigrid procedure to solve the linear systems arising out of two- and three-dimensional inviscid flows on unstructured grids. For a detailed exposition on multigrid, see the notes in this lecture series by Mavriplis.

Krylov methods. One of the most effective ways of dealing with the solution of symmetric, positive-definite matrix systems is by using a preconditioned conjugate gradient method devised by Hestenes and Stiefel – see Strang [115]. The issue of preconditioning is covered in the next section. The idea for conjugate gradient comes about from the observation that with a symmetric, positive-definite matrix A, the solution X of the linear system

$$AX = b, \tag{18}$$

minimizes the functional

$$F(z) = \frac{1}{2}(Az, z) - (b, z). \tag{19}$$

The steepest descent method for this problem is defined by a one-dimensional minimization of F in the direction of the gradient of F, given by r_k :

$$\begin{aligned} x_{k+1} : F(x_{k+1}) &= \min_{\alpha} F(x_k - \alpha r_k) \\ r_k &= Ax_k - b. \end{aligned} \tag{20}$$

In the step $x_k \rightarrow x_{k+1}$ of the conjugate gradient method, instead, a $(k+1)$-dimensional minimization is carried out:

$$\begin{aligned} x_{k+1} : F(x_{k+1}) &= \min_{\alpha_0 ..... \alpha_k} F(x_k - \alpha_0 r_0 ..... - \alpha_k r_k) \\ r_i &= Ax_i - b, i \leq k. \end{aligned} \tag{21}$$

Because of the symmetry of A, an orthogonal basis of the ith Krylov subspace, defined below, can be derived with only three-term recurrences. This is also sufficient for generating the residuals. Thus, the residuals r_i's and α_i's need not all have to be stored. This results in constant work and storage requirements at each iteration of the conjugate gradient method. Conjugate gradient method overcomes the difficulties in convergence associated with steepest descent method and for an $n \times n$ matrix A, it converges in n iterations in exact arithmetic.

For nonsymmetric systems, in some circumstances, it is possible to apply the conjugate gradient method to the normal equations. The drawbacks are that the condition number worsens and that a multiplication with the matrix transpose is required. It also eliminates the option of using matrix-free methods discussed in Section 4.4. Several generalizations of conjugate gradient method to solve nonsymmetric systems have been proposed in the literature. These can be classified into Arnoldi-based methods and Lanczos-based methods. The Generalized Minimal Residual Method (GMRES) [109] belongs to the first category whereas the Transpose-free Quasi-minimum Residual method(TFQMR) [45] and Bi-conjugate Gradient Stabilized method (Bi-CGSTAB) [34] belong to the second category. For nonsymmetric systems, the optimality condition of Eqn. (21) is replaced by the minimization of the residual norm at each step. The Arnoldi-based methods maintain this optimality condition but sacrifice the recursion relationships, whereas the Lanczos-based methods relax the optimality condition. Compared with the Arnoldi-based methods, these methods require

less work and storage per iteration. GMRES is quite robust and is probably the most widely used method and we describe it below.

Let x_0 be an approximate solution of the system

$$Ax = b, \tag{22}$$

where A is an invertible matrix. The solution is advanced from x_0 to x_k as

$$x_k = x_0 + y_k. \tag{23}$$

GMRES(k) finds the best possible solution for y_k over the Krylov subspace $< v_1, Av_1, A^2v_1$,...., $A^{k-1}v_1 >$ by solving the minimization problem

$$||r_k|| = \mathrm{m}in_y||v_1 + Ay||, \tag{24}$$

$$r_0 = v_1 = Ax_0 - b, r_k = Ax_k - b. \tag{25}$$

Here $||c||$ stands for the L_2 norm of vector c. The GMRES(k) procedure forms an orthogonal basis $\{v_1, v_2, ...v_k\}$ (termed search directions) spanning the Krylov subspace by a modified Gram-Schmidt method. The Gram-Schmidt process is a potential source of numerical error. An alternative implementation of GMRES using Householder transformation is given by Walker [138]. The search directions need to be stored. As k increases, the storage increases linearly and the number of operations, quadratically. Good solutions can however be found in small subspaces ($k \ll n$) if the $n \times n$ matrix A is well-conditioned. To mitigate the storage requirements and the operation counts, Saad and Schultz [109] also describe a GMRES(k, m), which is a restarted GMRES(k) where the k search directions are discarded and recomputed every m cycles. Substituting $v_1 = r_0$ into Eqn. (24) we obtain

$$||r_k|| = \min_{P(A)}||P(A)r_0||, \tag{26}$$

where $P(A)$ is of the form $I + a_1A + a_2A^2 + ...a_kA^k$. GMRES can be thought of as an optimal polynomial acceleration scheme [141]. Some insight can be gained by considering the special case of A being a diagonal matrix. Eqn. (26) then becomes

$$||r_k|| \leq E(k)||r_0||, \tag{27}$$

where $E(k) = \min_{p \in P(k)} \max_{\lambda \in \sigma(A)}|p(\lambda)|$ and $\sigma(A)$ is the eigenvalue spectrum of A. Like the conjugate gradient method, GMRES also satisfies a finite-stopping criterion, i.e., in the absence of roundoff errors, it will converge in at most n iterations for an $n \times n$ matrix. Preconditioning greatly improves the performance of GMRES and other related methods. It decreases the size of the spectrum so that the optimal polynomial generated by GMRES can annihilate the errors associated with each eigenvalue. For most large scale CFD problems, preconditioning is essential to achieve convergence of the linear problem.

4.2.6 Preconditioning

Instead of Eqn. (22) the preconditioned iterative methods solve the following systems:

$$PAx = Pb, \tag{28}$$

or

$$AQ(Q^{-1}x) = b \tag{29}$$

The systems of linear equations in Eqn. (28) and Eqn. (29) are referred to respectively as, left preconditioned and right preconditioned systems and P and Q as left and right preconditioners. The role of the preconditioner is to cluster the eigenvalues around unity. Thus, we require the

preconditioner to be a good approximation to A^{-1} (the ideal preconditioner) while being easy to compute. Thus, the requirements for a preconditioner are not different from those for choosing M in the general iterative method given by Eqn. (15). Thus all of the candidates for M fulfill the role of a preconditioner.

There is an important difference between right and left preconditioning. In the iterative methods, one sets a tolerance and when the residual for the linear problem is reduced to this tolerance relative to the initial residual, the linear system is declared solved. In the case of right preconditioning, this residual is the actual residual of the linear system $Ax - b$, whereas in left preconditioning it is the scaled residual $P(Ax - b)$. Therefore, when left preconditioning is employed, it is possible that the actual residual is not reduced as well as the scaled residual. As a result, we could terminate the solution procedure prematurely. Figure 6, taken from [127], shows the convergence histories obtained using left and right preconditioned GMRES for laminar flow over an NACA0012 airfoil. The flow conditions are $M_\infty = 0.3$, $\alpha = 3°$ and Reynolds number of 5000. The structured grid computation employed grid sequencing and the convergence histories are plotted as a function of CPU time on a Cray Y-MP. It is seen that on the fine grid (128×32) convergence deteriorates with left preconditioning.

Figure 6: Convergence histories with left and right preconditioning for laminar viscous flow over an NACA0012 airfoil.

Preconditioned GMRES method has been used to solve the compressible Euler and Navier-Stokes equations by a number of researchers. GMRES with block-diagonal preconditioning has been used by Shakib et al. [110] to solve the linear systems arising out of a finite element discretization of the Euler equations. Slack et al. [113] and Whitaker [140] have also used GMRES with block-diagonal preconditioning in two and three-dimensional applications. Slack et al. [113] have observed when solving the two-dimensional Euler equations, that the diagonally-preconditioned iterative methods perform better than the other methods as the number of elements in the mesh increases. Venkatakrishnan and Mavriplis [133] examined the use of GMRES with three preconditioners, namely diagonal preconditioning, ILU factorization and SSOR for solving compressible Euler and Navier-Stokes equations on unstructured grids. The discussion and results that follow are taken from [133]. The preconditioners and the optimizations carried out to extract the best vector performances out of them are described below.

4.3 Data structures

In this section the data structures and kernels employed are described for cell-vertex schemes in two dimensions; these can be easily modified to deal with cell-centered schemes. They are critical in reducing memory requirements and obtaining good performance. In the course of the GMRES method with preconditioning as per Eqn. (29), two kernels need to be addressed.

The first kernel is a sparse matrix - dense vector multiplication to compute Ax. The most commonly used data structures [49] are not suitable for this purpose since they have poor vectorization properties. The compressed row-storage scheme that is suitable for LU factorization yields short vector lengths because the vector lengths are limited to the number of nonzeros in each row. The data structure that is used for storing the sparse matrix A is most easily explained by interpreting the underlying triangular mesh as an undirected graph. Associated with each edge are the two vertices, say n_1 and n_2, which are incident to the edge. The spatial discretization operator typically utilizes this data structure and therefore, this information is already available. The two 4×4 matrices which contain the influence of n_2 on n_1 (entry in block row n_1 and block column n_2 in A) and the influence of n_1 on n_2 are stored. The diagonal blocks are stored separately. With such a data structure, a matrix vector multiplication can be carried out efficiently by employing a coloring algorithm to color the edges of the original mesh to get vector performance. Such a data structure is possible since the graph of the sparse matrix for the lower order linear system is equivalent to that of the supporting unstructured mesh.

The second kernel deals with the effect of the preconditioner Q on a vector. Q is D^{-1} for block-diagonal preconditioning and $(\tilde{L}\tilde{U})^{-1}$ for ILU preconditioning, where the $\tilde{}$ indicates approximate factors. For SSOR preconditioning, L and U are read off directly from the matrix A. The block-diagonal preconditioner computes the inverse of the 4×4 diagonal block associated with a grid point. Good vectorization when using this preconditioner is easy to achieve by unrolling the LU decomposition of the 4×4 diagonal matrix as well as the forward and back solves over all the grid points. The ILU and SSOR preconditioners require repeated solutions of sparse triangular systems. By using a wavefront reordering algorithm [4] it is possible to obtain good vector performance. Under this permutation of the matrix, unknowns within a wavefront are eliminated simultaneously. The key step in this procedure is an off-diagonal rectangular matrix - vector multiplication. This requires that $\tilde{L}$ and $\tilde{U}$ be stored in a convenient form. A data structure similar to that for A is chosen for $\tilde{L}$ and $\tilde{U}$. In addition to the nonzero blocks and the block column numbers which are provided by the compressed row storage scheme in the factorization, we store the block row numbers. With this additional information, the data structure becomes similar to the edge-based data structure employed for the A matrix except that we only store one block per edge. The off-diagonal matrix vector multiplication can then be vectorized by interpreting the rectangular matrix as a directed graph and coloring the directed edges. The performances are further enhanced by performing all the operations on blocks of size 4×4.

The memory requirements for the implicit schemes are now given starting with the storage requirements for the matrix A. A cell-centered scheme that makes use of the lower-order representation on the left-hand side of Eqn. (12) requires an array of size $(\alpha+1) \times 4 \times 4N$ in two-dimensions and $(\alpha+1) \times 5 \times 5N$ in three dimensions, where N is the number of triangular or tetrahedral cells and α is the number of neighbors of each cell. $\alpha = 3$ in two dimensions and $\alpha = 4$ in three dimensions. In two dimensions, a cell-vertex scheme requires an array of size $7 \times 4 \times 4N = 112N$ to store the nonzeros of the matrix, where N is the number of vertices. The factor of 7 arises from having 3 times as many edges as vertices (valid for all two-dimensional triangular grids, neglecting boundary effects); we store two blocks per edge plus the diagonal matrix for all the vertices. The factor of 7 can also be arrived at as $(\alpha+1)$, where the average number of neighbors in a triangulation is 6. In three dimensions β, defined as the ratio of the number of tetrahedra and the number of vertices, N, could be arbitrarily large; however in practice this ratio is $5-8$. Meijering [87] shows that for

a three-dimensional Delaunay triangulation of randomly distributed points $\beta \approx 6.77$ and that the number of edges in the triangulation varies as $(\beta+1)N \approx 7.77N$. Thus the memory required to store the nonzeros of the matrix in three dimensions is roughly $2(\beta+1)+1 \approx 17 \times 5 \times 5N = 425N$. This is prohibitive even given the large memory available on current supercomputers. Barth [13], however, has proposed some interesting techniques that bring down the memory requirements for dealing with the linear system derived from a higher order discretization. The standard iterative methods require only one array of the sizes given above, namely to store the matrix A. ILU-preconditioned GMRES method requires three such arrays. For our cell-vertex scheme, one of these arrays stores the matrix A in the edge-based data structure that is suitable for computing the matrix-vector product. A second copy stores the matrix in a compressed row format [49] that is suitable for the factorization and a third contains the $\tilde{L}$ and the $\tilde{U}$ factors. The second array is reused for storing the search directions in GMRES, permitting up to 27 search directions to be stored in two dimensions. Block-diagonal as well as multi-color Gauss-Seidel preconditioners dispense with one of these arrays.

We conclude this section with the topic of reordering of unknowns. The ordering of unknowns has a bearing on the convergence properties of many iterative methods that involve a directional bias such as the SSOR and ILU techniques. Batina [15] reordered the unknowns in the direction of the freestream flow while using a Gauss-Seidel iteration on unstructured grids to great advantage. For structured meshes it was found [37] that a column-major ordering which minimized the bandwidth (the "most local" ordering) yielded the best convergence rates. For unstructured meshes we have settled on the Reverse Cuthill-Mckee (RCM) ordering [49]. This is a standard ordering used in sparse direct methods to reduce fill-in, but it also appears to be the "most local" ordering. Various orderings based on coordinates of the vertices (sorting the vertices by the x coordinates, y coordinates or some combination of x and y coordinates) were also tested in the solution of the compressible Navier-Stokes equations in [133]. The RCM ordering gave slightly better convergence rates over a wide range of problems. RCM is also more efficient in that it creates fewer wavefronts, thus producing longer vectors. Dutto [39] has carried out a systematic study of the effect of various orderings on convergence and has reported similar results.

4.4 Newton-Krylov methods

All the iterative methods discussed so far sacrifice convergence properties by making a lower order approximation on the left hand side of Eqn. (12). If on the other hand, a consistent second order approximation were employed on the left hand side, the convergence rates in terms of iterations would vastly improve although involving higher computational and storage costs at each iteration. If the linear system is solved well at each time step, it is possible to realize quadratic convergence associated with Newton's method. As discussed earlier, the memory requirements for the higher order matrix representation are prohibitive. Therefore, unless one has access to very large-memory computers, this is not a viable approach. The Newton-Krylov approach bypasses this issue by never forming the matrix. Instead the effect of the Jacobian matrix on a vector is approximated by one-sided finite differences:

$$\frac{\partial R}{\partial W}(x)p \approx \frac{R(x+\epsilon p) - R(x)}{\epsilon}, \tag{30}$$

or by the more expensive central-difference approximation:

$$\frac{\partial R}{\partial W}(x)p \approx \frac{R(x+\epsilon p) - R(x-\epsilon p)}{2\epsilon}, \tag{31}$$

where ϵ is the step size. Newton-Krylov methods, proposed by Brown and Saad [24], have been investigated for compressible Euler and Navier-Stokes equations using unstructured grids by Tidriri

[119], Johan et al. [66], Nielsen et al. [94] and for the full potential equation by Cai et al. [26]. Finite-differencing with ϵ makes the matrix-free methods somewhat susceptible to numerical difficulties. To ensure that the derivative is reasonably well approximated, ϵ cannot be too large. It cannot be too small as the derivatives will be susceptible to precision errors. Guidelines for choosing ϵ are provided in the text by Dennis and Schnabel [33]. It appears that GMRES may have an advantage over other Krylov methods when used in a matrix-free context because the vectors p that arise in GMRES have unit-norm and are hence well-scaled. McHugh and Knoll [86] have observed this to be the case when solving the incompressible Navier-Stokes equations. The performance of GMRES did not degrade much when switching from a standard implementation to the matrix-free implementation, whereas those of TFQMR and Bi-CGSTAB degraded.

Although the matrix-free method is attractive because it does not form the matrix explicitly, the matrix is still required for preconditioning purposes. This is true for ILU, SSOR and multi-color SOR preconditioners. Zohan et al. [66] settle for a compromise that uses a block-diagonal preconditioner to enable them to solve three-dimensional problems. Therefore, for other preconditioners that require the matrix, the advantage of the matrix-free methods comes not from a savings in storage but from the fact that a true Newton's method can be approached. To this end, we can use the lower order system to precondition the higher order system. In [26], ILU preconditioning of the lower order system is employed in concert with a matrix-free GMRES in order to realize fast convergence in the solution of nonsymmetric elliptic problems. Barth [13] and Nielsen et al. [94] have employed an ILU preconditioning of the lower order system to solve three-dimensionals Euler equations on unstructured grids. Whereas Barth uses a higher order matrix-based GMRES, Nielsen et al. employ a Newton-Krylov framework.

4.5 Applications

The iterative methods discussed require a few parameters. The start-up CFL number (nondimensional time step) and the maximum CFL number that can be used need to be specified. It is also possible to freeze the ILU factorization after a few time steps (or after a prescribed reduction in the residual) and increase the efficiency of the code, since it eliminates the assembly and/or the approximate LU factorization of the matrix. This introduces an additional parameter. GMRES requires a few parameters. It requires the maximum number of search directions k, the number of restart cycles m and a tolerance level which specifies the desired order of reduction of the residual of the linear sub-problem. In all the problems, the tolerance is set to 10^{-5}. The solution to the linear system is terminated when the number of iterations exceeds the specified maximum whether or not the tolerance criterion is met, opting to rely on the outer inexact Newton iteration for convergence. Experience indicates that the tolerance criterion is easy to meet in the initial stages of the flow solution, but becomes extremely difficult to satisfy in the latter stages.

We illustrate the applications of iterative methods for a variety of flows and problem sizes in two dimensions. The first case studied is a standard airfoil flow, namely inviscid flow over the ubiquitous NACA0012 airfoil at a freestream Mach number of 0.8 at 1.25° angle of attack. The unstructured grid contains 4224 vertices or 8192 triangles. A close-up of the nearly uniform grid is shown in Figure 7. The solution (not shown here) agrees with standard results. The computed lift, drag and moment coefficients are 0.3523, 0.0226 and -0.0452 respectively. The convergence histories of five different methods are shown in Figure 8 as a function of CPU time. Since we are dealing with different methods which require varying amounts of work at each time step we believe that CPU time is the only true measure for comparing them. Since there are quite a few parameters involved in each of these methods, what we have shown is the "best" convergence history obtained with each method. GMRES with ILU preconditioning (GMRES/ILU) uses 5 search directions, CFL $20 - 10^6$ and freezes the factorization after 30 time steps. GMRES/SSOR, wherein SSOR is used as the preconditioner, employs 15 search directions, CFL $20 - 10^6$ and freezes the matrix after 30 time steps.

GMRES/DIAG, which uses the block-diagonal preconditioner, employs 25 search directions with 3 restarts, CFL 10-500,000 and freezes the preconditioner after 25 time steps. The ILU iteration uses CFL 1-50 and freezes the matrix after 25 steps. Finally, the SSOR iteration uses CFL 1-25 and freezes the matrix after 30 time steps. Using multiple "inner" sub-iterations with the ILU and the SSOR iteration schemes in order to be able to use larger time steps turns out be less efficient for this problem. The number of time steps taken by GMRES/ILU, GMRES/SSOR, GMRES/DIAG, ILU and SSOR are 75, 100, 75, 700 and 700 respectively. The parameters given above for the five methods, we believe, are nearly optimal for this problem and yield the best convergence history for each of the methods. Having to choose many parameters is a major drawback in using iterative methods to solve the approximate linear systems arising from nonlinear problems. However, we will be able to provide some guidelines for choosing these parameters for the best of these methods, namely GMRES/ILU, by solving a few more representative problems. In Figure 8, we notice that GMRES/DIAG is quite slow even for this simple problem, while ILU iteration appears to be quite good. SSOR iteration and GMRES/SSOR have similar convergence histories. SSOR as a preconditioner is not as effective as the ILU preconditioner; GMRES/ILU appears to be the best of all the methods. As we shall see, as the problems get bigger and more stiff, GMRES/ILU performs much better than the other four methods.

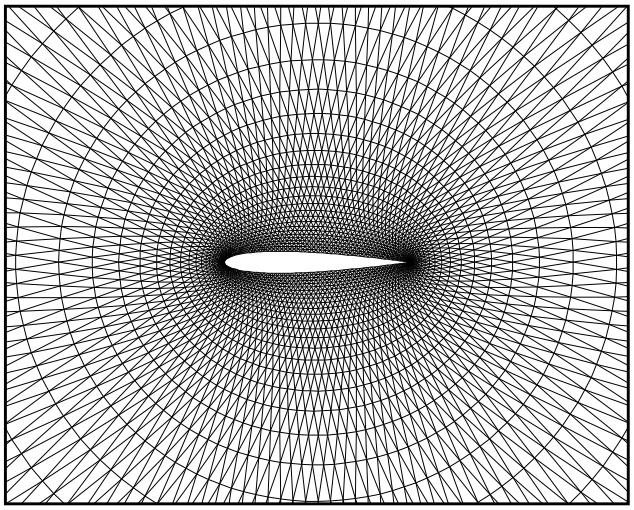

Figure 7: Grid about an NACA0012 airfoil with 4412 vertices.

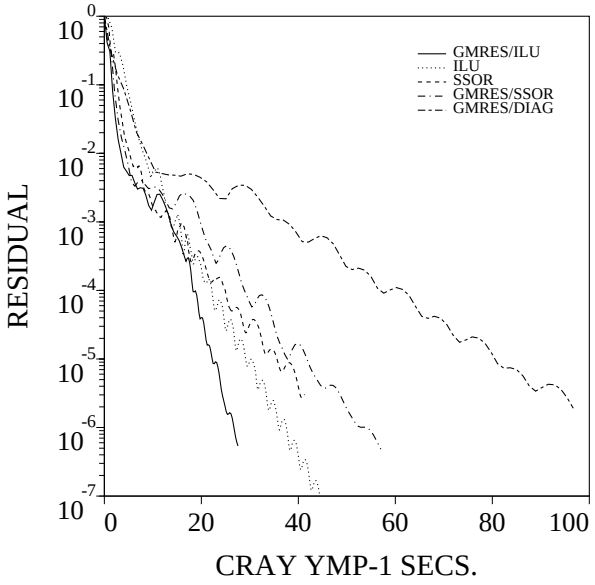

Figure 8: Convergence histories for inviscid flow over an NACA0012 airfoil ($M_\infty = 0.8, \alpha = 1.25°$).

The next flow considered is inviscid subcritical flow over a four-element airfoil at a freestream Mach number of 0.2 and angle of attack of 5°. The triangular mesh employed has 10395 vertices. The grid is shown in Figure 9. The solution is not shown here and may be found in [78]. In Figure 10 we present the convergence histories of GMRES/ILU, GMRES/DIAG and ILU and SSOR iteration. GMRES/SSOR had great difficulties in the initial stages and is not shown. GMRES/ILU converges much better than the other methods. The parameters for GMRES/ILU are 10 search directions and CFL $20 - 10^6$, the factorization being frozen after 30 time steps. GMRES/DIAG employs 25 search directions with 2 restarts, CFL $10 - 5 \times 10^5$ and freezes the preconditioner after 30 time steps. ILU iteration uses CFL $1 - 50$, freezes the matrix after 50 time steps and does not use sub-iterations. SSOR iteration uses CFL $0.5 - 5$ and freezes the matrix after 100 time steps. The number of time steps taken by GMRES/ILU, GMRES/DIAG, ILU and SSOR are 100, 70, 400 and 400 respectively. SSOR, either by itself or as a preconditioner, is clearly unsatisfactory for this

problem.

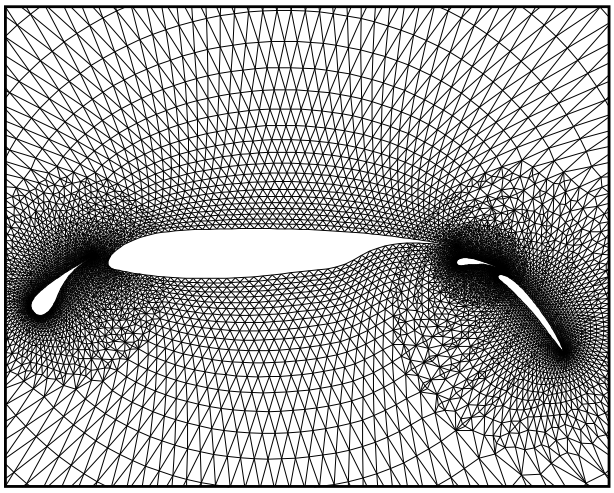

Figure 9: Grid about a four-element airfoil with 10395 vertices.

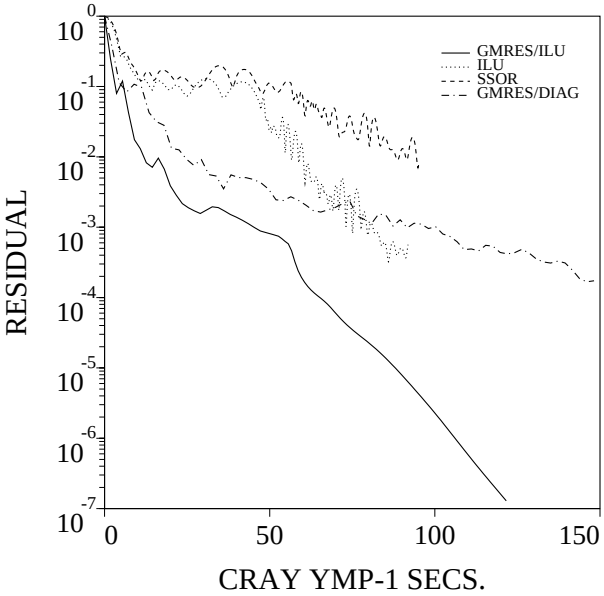

Figure 10: Convergence histories for inviscid flow over the four-element airfoil ($M_\infty = 0.2, \alpha = 5°$).

The performances of the methods are compared for transonic turbulent flow over an RAE2822 airfoil, referred to as Case 6. The flow conditions are $M_\infty = 0.729$, $\alpha = 2.31°$ and Reynolds number 6.5×10^6 based on the chord. The flow is computed on a mesh with 13751 vertices which contains cells in the boundary layer and the wake region with aspects ratios up to 1000:1. For this turbulent calculation, the unstructured mesh implementation of the Baldwin-Lomax model developed in [80] is used. The grid is shown in Figure 11. The pressure plot and skin friction distribution and experimental data are shown in Figures 12 and 13. The lift, drag and moment coefficients are 0.7342, 0.0132 and -0.0978. Figure 14 shows the convergence histories of the various methods. Only GMRES/ILU and GMRES/DIAG converge, the latter doing so much more slowly. GMRES/SSOR diverges for any reasonable CFL numbers at all and its convergence history is not shown. The parameters for GMRES/ILU are 25 search directions and CFL $5-25000$. The factorization is frozen after 80 time steps. The turbulence model is also frozen after nearly six orders of reduction in the residual; otherwise, the residual hangs and the convergence of the method slows down. The effect of freezing the turbulence model in this fashion has minimal effect on the aerodynamic coefficients (less than 0.02% change in lift coefficient). The parameters for GMRES/DIAG are the same as for GMRES/ILU. The number of time steps taken by both GMRES/ILU and GMRES/DIAG is 150. The unstructured multigrid algorithm of Mavriplis [79] takes nearly 300 secs. on the YMP to reduce the L_2 norm of the residual to $.3 \times 10^{-3}$ and GMRES/ILU takes about 450 secs. to get to the same level (7 orders of reduction in residual) for this problem. In the full multigrid algorithm, the problem is first solved on coarser grids, whereas GMRES/ILU starts from freestream conditions on the fine grid. The ILU and SSOR iterations use 10 sub-iterations, CFL $.5 - 2.5$ and still do not converge after 200 time steps.

Based on this study, we draw the following conclusions regarding the five candidate implicit schemes. For inviscid problems, with a small number of vertices and grids with low cell-aspect ratios, most of the methods work well, GMRES with ILU preconditioning performing the best. For larger problems, especially at high Reynolds numbers, almost all the methods except for GMRES/ILU converge extremely slowly, if at all. Regarding the parameters, for inviscid flows, we find that 5–10 GMRES search directions are usually sufficient, whereas the turbulent viscous flows require 25 search directions. The start-up CFL number is usually about 20 for inviscid flows and 5 for

turbulent viscous flows and the CFL number is allowed to increase up to 500-50,000 fold. A non-restarted GMRES is used whenever possible.

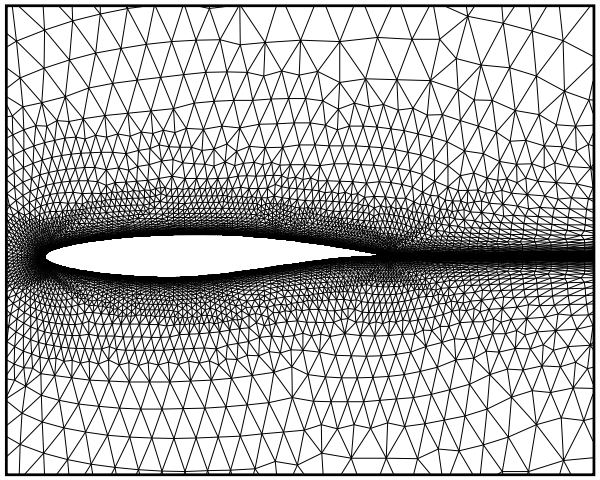

Figure 11: Mesh for computing transonic turbulent flow over an RAE2822 airfoil with 13,751 vertices.

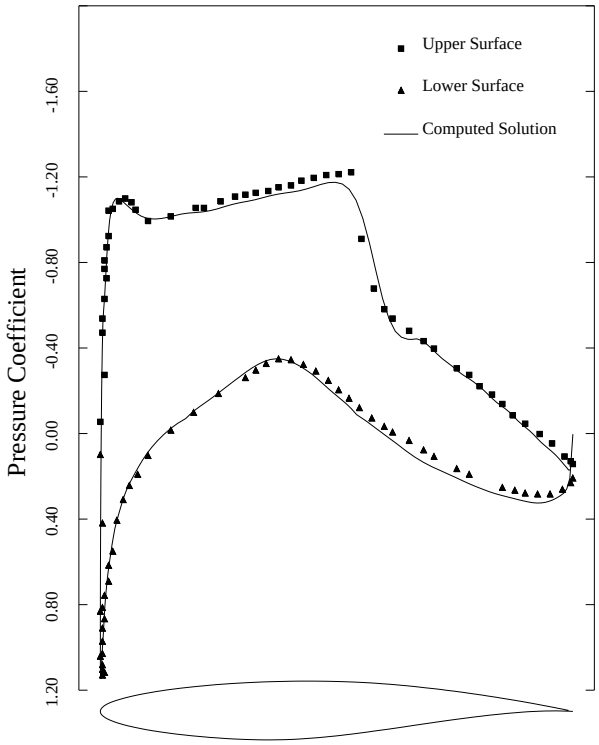

Figure 12: Computed surface pressure distribution for Case 6 ($M_\infty = 0.729, \alpha = 2.31°$) and Reynolds number = 6.5×10^6).

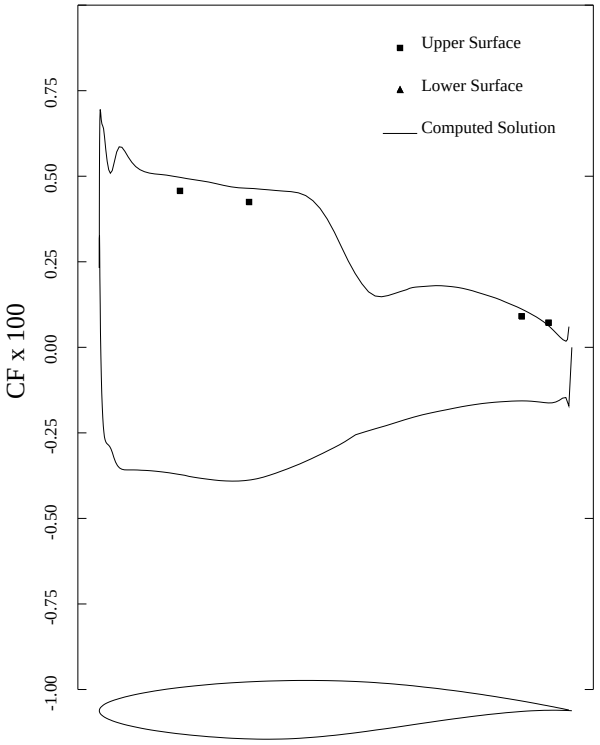

Figure 13: Skin friction distribution for Case 6.

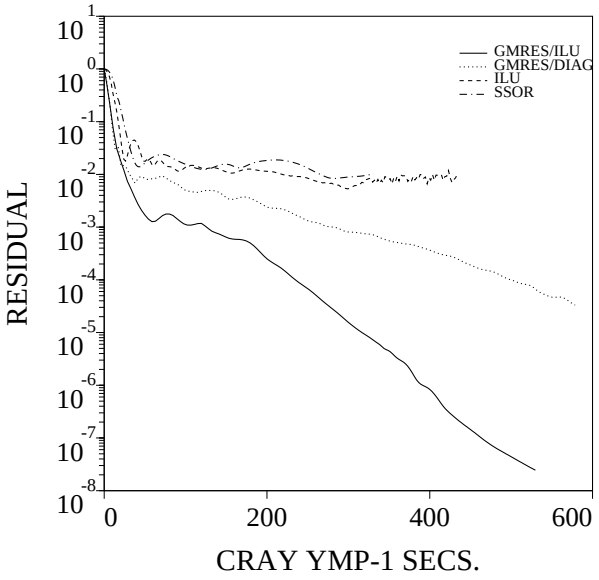

Figure 14: Convergence histories for Case 6 with the various implicit schemes.

An application of the Newton-Krylov method to solve three-dimensional Euler equations is presented. This result is taken from a forthcoming paper by Nielsen et al. [94]. The Euler code employs linear reconstruction and uses Roe's approximate Riemann solver to compute the fluxes. The Newton-Krylov method employs an ILU preconditioning of the lower order system. Figure 15 shows the surface grid for an ONERA M6 wing with 139,356 nodes. The freestream conditions are $M_\infty = 0.699$ and $\alpha = 3.06°$. The convergence histories of the Newton-Krylov and the multi-color Gauss-Seidel iterative techniques are shown in Figures 16(a) and 16(b) in terms of iterations and CPU times on the Cray Y-MP. The superior convergence of the Newton-Krylov method is apparent. The multi-color Gauss-Seidel employed 20 iterations. The Newton-Krylov method used 15 GMRES search directions.

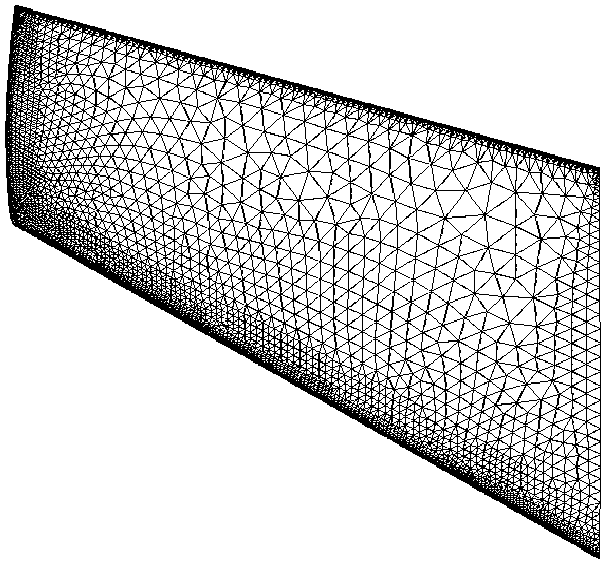

Figure 15: Surface grid for inviscid flow over the ONERA M6 wing (139,356 nodes).

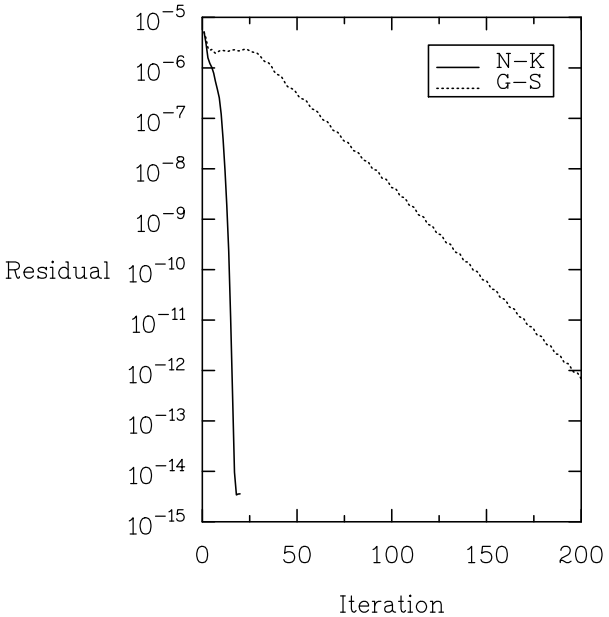

Figure 16(a): Convergence histories of the Newton-Krylov and multi-color Gauss-Seidel techniques as a function of iterations.

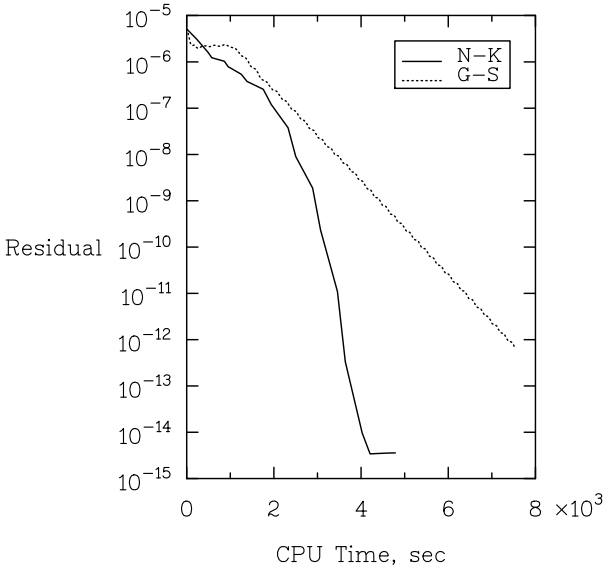

Figure 16(b): Convergence histories of the Newton-Krylov and multi-color Gauss-Seidel techniques as a function of CPU time.

5 Solution techniques for unsteady flows

While solution techniques for computing steady flows have evolved to a high degree of sophistication, those for dealing with unsteady flows have lagged behind. A comprehensive survey of methods for computing unsteady flows using structured grids may be found in the survey paper by Edwards and Thomas [40]. In this section, we concentrate on techniques applicable for unstructured grids.

5.1 Finite volume discretization

After applying the finite volume procedure, the following system of coupled differential equations is obtained:

$$\frac{d}{dt}(VMW) + R(W) = 0. \tag{32}$$

Here W is the solution vector over the whole field, $R(W)$ is the residual vector approximating the boundary integral in Eqn. (1), V is the cell volume associated with the vertex and M is the mass matrix.

The mass matrix arises because the update indicated by the residual $R(W)$ should be made to the average value in the control volume. It thus relates the average value of a control volume associated with a vertex to the point value of the vertex and those of its immediate neighbors. Note that this definition differs from the way the mass matrix is defined in finite element formulations where the mass matrix arises naturally from requiring the PDE, with the solution expanded in a set of basis functions, to be orthogonal to a set of trial functions; in a Galerkin method the trial functions are also the basis functions. It is well known that the use of a consistent mass matrix in a finite element method results in excellent phase properties [136, 35]. However, in the case of finite volume schemes employing a reconstruction procedure and upwinding, such a definition does not extend readily and therefore we will use an alternative definition. For those schemes employing a polynomial reconstruction procedure within a cell, the mass matrix is determined by computing the average of this polynomial over the cell. When cell-centered approximations are employed, the average value in the control volume and the point value at the centroid of the cell match to second order accuracy, and therefore the mass matrix may be omitted, decoupling the system of ODE's in Eqn. (32). However, when cell-vertex discretizations are employed, in general, the centroid of the control volume is not represented by the vertex in question. The mass matrix M couples the system of ODE's. The effect is that even when an explicit scheme such as a multi-stage Runge-Kutta scheme is used, one has to deal with the solution of a coupled linear system at each stage of the Runge-Kutta scheme. A technique called "mass-lumping" often used in finite element approach [116], replaces the matrix M by the identity matrix. While this has no effect on steady-state solutions, for time-accurate computations, it would appear that such an approximation introduces locally a first order spatial error. This approximation is routinely adopted for unsteady flows as well [16], and does not appear to adversely affect the quality of the solutions obtained. Davis and Bendiksen [30] have observed little discernible differences in the unsteady solutions when using the full and the lumped mass matrices. However, since they used an explicit scheme, the time steps were quite small and furthermore, their grids appeared to be fairly uniform. For an equilateral grid, the mass matrix can be lumped without any adverse impact, because the vertex locations coincide with the centroids of the control volumes defined by the median dual. The technique employed to solve the mass matrix (a few Jacobi iterations) is clearly not efficient, especially when larger grids are used. Also, when higher order spatial discretizations are employed, the mass matrix has to be reckoned with, even when using cell-centered discretizations. An efficient means of inverting the mass matrix is yet to be found. Direct inversion will entail a substantial effort, and is clearly unattractive, especially in three dimensions.

One way to avoid the mass matrix altogether is to never deviate from the concept of cell averages. This would require that the reconstruction procedure only make use of cell averages and

not point values. This is an attractive proposition for higher order schemes. Higher order accurate schemes based cell averages have been proposed and tested in [11, 27, 55].

In the following, we will review explicit schemes for unstructured grids and some acceleration techniques. The techniques of residual averaging and temporal adaptation that relax time step restrictions are briefly reviewed. Finally, the development of implicit schemes that allow for arbitrarily large time steps is outlined. Much of the discussion on the implicit scheme for unstructured grids is excerpted from a forthcoming paper [134].

5.2 Explicit schemes

Explicit schemes are the schemes of choice for certain unsteady applications when the time scales of interest are small or more precisely, that they are comparable to the spatial scales. The grids should be clustered only in regions of interest; otherwise, the size of the explicit time step could become unnecessarily small. However, when dealing many low frequency phenomena such as flutter, explicit schemes lead to large computing times. Also, for a variety of practical viscous flows, the time step restrictions imposed by small cells deep inside the boundary layer are excessively small. Since the boundary layer is quasi-steady, implicit methods which allow for larger time steps may be more suitable methods for such flows.

Assuming that the mass matrix has been lumped, the explicit schemes reviewed in Subsection 4.1 can be applied to solve the system of uncoupled ODE's Eqn. (32) in a time-accurate manner. The standard Runge-Kutta schemes are attractive since they can be designed to have a temporal order of accuracy comparable to the spatial order of accuracy, without the need to store many solution levels. When the spatial discretization possesses the TVD (Total Variation Diminishing) or ENO (Essentially Nonoscillatory) properties, the Runge-Kutta schemes designed by Shu and Osher [111] are often employed since they preserve these properties while maximizing the CFL number.

When the mass matrix is present, the system of ODE's is coupled. When using a two-step explicit Finite Element - Flux Corrected Transport (FEM-FCT) algorithm, Parikh et al. [99] used a few Jacobi iterations since the mass matrix is well-conditioned. Davis and Bendiksen [30] when employing a multi-stage Runge-Kutta algorithm have used a similar procedure at every stage of the RK scheme. Donea [35] advocated a two-pass procedure where the beneficial effect of the mass matrix is exploited in a lumped-explicit context. Splitting the mass matrix $M = I + B$, the two pass procedure for Eqn. (32) becomes

$$\begin{aligned} V(W_j^{n+1} - W_j^n)^{(1)} &= -R_i(W^n) \\ V(W_j^{n+1} - W_j^n)^{(2)} &= -R_i(W^n) - B(W_j^{n+1} - W_j^n)^{(1)} \end{aligned} \tag{33}$$

Donea also showed that such a scheme possesses the same order of accuracy as the scheme employing a consistent mass matrix, while suffering some degradation in phase properties on uniform grids.

One way to relax the stability restrictions of explicit schemes is by using the technique of residual averaging [61]. However, in its original form it is only applicable for steady computations. Venkatakrishnan and Jameson [131] proposed an extension of residual averaging for time-accurate computations. This is outlined for a one-dimensional example. To solve

$$\frac{\partial W}{\partial t} + R(W) = 0, \tag{34}$$

replace the residual $R(W)$ by $\bar{R}(W)$ given implicitly by the relation

$$-\epsilon_{i+1/2}(\bar{R}_{i+1} - \bar{R}_i) + \bar{R}_i + \epsilon_{i-1/2}(\bar{R}_i - \bar{R}_{i-1}) = R_i \tag{35}$$

ϵ is required to satisfy the following inequality which can be derived by assuming a central difference approximation to the first order spatial derivative, $R(W) = \frac{\partial u}{\partial x}$:

$$\epsilon \geq \frac{1}{4}[(\Delta t/\Delta t^*)^2 - 1]. \tag{36}$$

Here Δt is the global time step used and Δt^* is the local time step allowed for the unsmoothed scheme. This is similar to the implicit scheme of Lerat et al. [72] where use is made of the spectral radius of the Jacobian matrix which is inversely related to the local time step. However, Lerat et al. [72] interpret their method as correcting the truncation error term in an implicit manner. They were able to show unconditional stability for this implicit method. Even though unconditional stability is proven for such implicit schemes, in practice only a CFL number of the order of 10 is used. This practical limit arises out of convergence considerations for steady problems and temporal truncation error in time-accurate computations. For steady state applications, ϵ is assumed to be constant given by Eqn. (36), with Δt replaced by λ, the CFL number with residual averaging, and Δt^* replaced by λ^*, the CFL number of the unsmoothed scheme.

Another technique that can be used to improve the performance of explicit schemes is temporal adaptation. Standard explicit schemes use a globally minimum time step for stability reasons. This implies that many of the cells such as the cells having large volumes, are being advanced at a fraction of maximum time steps permitted locally by stability considerations. Kleb et al. [70] have derived a procedure that enables different cells to take varying number of local time steps to get to a particular time level. The residual calculations make use of time-consistent fluxes which are either available or are obtained via interpolation. Kleb et al. [70] have demonstrated savings in computational effort of factors up to 10 over explicit schemes that employ globally minimum time steps when solving a variety of two-dimensional transonic flows on unstructured grids.

Multigrid time-stepping schemes have been developed primarily to accelerate the convergence to steady-state. They rely on approximations of the governing equations on a sequence of successively coarser grids. In contrast to the elliptic viewpoint given in Section 4.2.5, the hyperbolic interpretation of multigrid is that by using successively coarser grids and maintaining a constant CFL number, and thereby taking increasingly larger time steps, the disturbances are rapidly dispatched out of the domain. One effort arising as a result of adopting the hyperbolic viewpoint is the unsteady multigrid algorithm of Jespersen [46]. While he was able to show theoretically that the solution obtained by using this procedure was time-consistent to a given order, he observed experimentally that the quality of the solution deteriorated as the number of coarse grids used was increased.

In spite of these acceleration techniques, explicit schemes are not viable for many unsteady computations. An implicit method that allows for arbitrarily large time steps is desirable since the time steps would then be solely determined by flow physics. Akin to a spatial grid refinement, a temporal refinement should be done to ensure that the solution is converged in time. Such a method is outlined in the next section.

5.3 Implicit schemes

When an implicit scheme is used to solve for unsteady flows, one has to drive the unsteady residual, defined below, to zero or at least to truncation error. In the context of factored implicit schemes, this is usually done by employing inner iterations [103, 102]. It is the role of these inner iterations to eliminate errors due to factorization, linearization, and also errors arising from employing a lower order approximation on the implicit side. The number of inner iterations required may be large depending on the flow situation and the size of the time step employed.

Brennis and Eberle [22] and Jameson [62] have advocated a different approach for deriving an efficient implicit scheme for unsteady flows. The idea is to define an unsteady residual, following a backward difference approximation to the time derivative and then use the same method as for

the steady state problem. In [22] a relaxation method is used whereas in [62] a multigrid procedure is used as a driver for the fully implicit scheme when using structured grids. The significant advantage of the approach when multigrid is used to solve the nonlinear problem is that it incurs no storage overheads associated with traditional implicit schemes, and is particularly attractive for unstructured grid computations in three dimensions. It allows the time step to be determined solely based on flow physics. This method has been used to compute two- and three-dimensional flows over airfoils and wings [62, 89, 3] using structured grids. Vassberg [125] has applied this method to compute flow solutions over oscillating airfoils using unstructured grids where a sequence of triangulations was generated by removing points from the fine grid triangulation.

We first outline the implicit scheme as developed by Jameson [62] for cell-centered structured grids. Therefore the mass matrix was not present in his formulation. Replacing the mass matrix in Eqn. (32) by the identity matrix and making a 3-point backward-difference approximation for the time derivative yields

$$\frac{3}{2\Delta t}V^{n+1}W^{n+1} - \frac{2}{\Delta t}V^{n}W^{n} + \frac{1}{2\Delta t}V^{n-1}W^{n-1} + R(W^{n+1}) = 0. \tag{37}$$

As argued in [62], when applied to a linear differential equation of the form,

$$\frac{dW}{dt} = \alpha W \tag{38}$$

this particular discretization is A-stable i.e., stable for all values of $\alpha\Delta t$ in the left-half of the complex plane [29]. Eqn. (37) is now treated as a steady state equation by introducing a pseudo-time t^*. The multigrid scheme then solves the following system to steady state using local time steps Δt^*:

$$\frac{\partial VU}{\partial t^*} + R^*(U) = 0, \tag{39}$$

where U is the approximation to W^{n+1}. Here the *unsteady residual* $R^*(U)$ is defined as

$$R^*(U) = \frac{3}{2\Delta t}VU + R(U) - S(V^nW^n, V^{n-1}W^{n-1}) \tag{40}$$

with the source term

$$S(V^nW^n, V^{n-1}W^{n-1}) = \frac{2}{\Delta t}V^nW^n - \frac{1}{2\Delta t}V^{n-1}W^{n-1} \tag{41}$$

remaining fixed through the multigrid procedure. We would like to drive R^* to zero at each time step.

A multi-stage Runge-Kutta scheme is now applied to solve Eqn. (39). A low-storage second order accurate m-stage Runge-Kutta scheme to advance U is given by

$$\begin{aligned} Q_0 &= U^l \\ &.... \\ V^{n+1}Q_k &= V^{n+1}Q_0 - \alpha_k\Delta t^* R^*(Q_{k-1}) \\ &.... \\ U^{l+1} &= Q_m \end{aligned} \tag{42}$$

Starting with $U^1 = W^n$, the sequence of iterates $U^l, l = 1, 2, 3....$ converges to W^{n+1}.

However, the way the scheme has been formulated has been observed by Arnone et al [6] to be unstable for small physical time steps, Δt. This is counter-intuitive because when using small Δt, the multigrid procedure should converge fast and ideally, in the limit of explicit time steps, the multigrid procedure should converge in just a few iterations. Otherwise, one has to depart from the

implicit framework for small time steps and switch to an explicit scheme and, this is not desirable. Melson et al. [89] showed that the problem is due an instability that arises when small Δt is used. They modified the scheme to get rid of this instability. The problem is that the unsteady residual $R^*(W)$ includes the term $\frac{3}{2\Delta t}VU$ and, is therefore, treated explicitly in the Runge-Kutta scheme. Their analysis showed that if this term were treated implicitly in the Runge-Kutta scheme, the stability region would grow as Δt decreased. It is easy to treat the term implicitly since it is only a diagonal term. Splitting the residual $R^*(U)$ as

$$R^*(U) = \frac{3}{2\Delta t}VU + R(U) - S, \tag{43}$$

the Runge-Kutta scheme now becomes,

$$\begin{aligned} Q_0 &= U^l \\ &.... \\ \left[I + \frac{3}{2\Delta t}\alpha_k \Delta t^*\right] V^{n+1} Q_k &= V^{n+1} Q_0 - \alpha_k \Delta t^* \left[R(Q_{k-1}) - S\right] \qquad (44) \\ &.... \qquad (45) \\ U^{l+1} &= Q_m \end{aligned}$$

With the modified scheme, Melson et al. [89] have shown that arbitrarily large or small time steps Δt may be employed.

As in [62, 89], we employ a full approximation storage multigrid scheme. The source term is computed only on the fine grid and the coarse grids are driven by the fine grid residuals. For the generation of coarse grids we follow the agglomeration multigrid procedure [71, 114, 132, 84, 85]. In this method, the sequence of coarse grids is generated *a priori* using efficient graph-based algorithms. This method has certain advantages when dealing with rigidly moving or deforming meshes. Since the edges that comprise the coarse grid volumes are subsets of the fine grid control volume edges, when the grid moves rigidly or deforms, the projections of the control volume faces onto the coordinate directions are easily computed from those of the fine grid. Also, as long as no grid points are added or removed, and the triangulation remains valid, and the grid connectivity remains the same, the interpolation operators stay the same. Multigrid schemes based on non-nested triangulations would have to recompute the transfer operators when the grids deform.

5.4 Treatment of the mass matrix

When employing a vertex-centered approximation, making a 3-point backward-difference approximation for the time derivative yields

$$\frac{3}{2\Delta t}V^{n+1}M^{n+1}W^{n+1} - \frac{2}{\Delta t}V^n M^n W^n + \frac{1}{2\Delta t}V^{n-1}M^{n-1}W^{n-1} + R(W^{n+1}) = 0. \tag{46}$$

The multigrid scheme now solves the following system to steady state using local time steps Δt^*:

$$\frac{\partial VU}{\partial t^*} + R^*(U) = 0, \tag{47}$$

where U is the approximation to W^{n+1} where $R^*(W)$ now includes the mass matrix terms. Notice that the first term $\frac{\partial U}{\partial t^*}$ does not involve the mass matrix uncoupling the system of equations. The explicit Runge-Kutta scheme can be applied exactly as before. Thus, the inversion of the mass matrix is thus accomplished indirectly during the multigrid procedure. However, the modified scheme of Melson et al. [89] poses a serious problem. Their modification would require the term

$\frac{3}{2\Delta t}VMU$ to be treated implicitly which is no longer a diagonal term. A modification has been devised that solves this problem and is detailed below. The implicit Runge-Kutta scheme that is stable for all Δt is given by

$$\begin{aligned} Q_0 &= U^l \\ &.... \\ \left[I + \frac{3}{2}\Delta t \alpha_k \Delta t^* M^{n+1}\right] V^{n+1} Q_k &= V^{n+1} Q_0 - \alpha_k \Delta t^* \left[R(Q_{k-1}) - S\right] \\ &.... \\ U^{l+1} &= Q_m \end{aligned} \tag{48}$$

where the source term S is now given by

$$S = \frac{2}{\Delta t} V^n M^n W^n - \frac{1}{2\Delta t} V^{n-1} M^{n-1} W^{n-1} \tag{49}$$

If we simply replace the mass matrix M by the identity on the left hand side of Eqn. (49), we have observed that the instability at small time steps persists. In our modification, we first add and subtract $\frac{3}{2\Delta t}\alpha_k \Delta t^* M^{n+1} V^{n+1} Q_{k-1}$ on the right hand side of Eqn. (49) to obtain

$$\left[I + \frac{3}{2\Delta t}\alpha_k \Delta t^* M^{n+1}\right] V^{n+1} Q_k = V^{n+1} Q_0 - \alpha_k \Delta t^* R^*(Q_{k-1}) + \frac{3}{2\Delta t}\alpha_k \Delta t^* M^{n+1} V^{n+1} Q_{k-1}, \tag{50}$$

where use has been made of the equation

$$R^*(U) = \frac{3}{2\Delta t} VMU + R(U) - S. \tag{51}$$

Note that the same term $\frac{3}{2\Delta t}\alpha_k \Delta t^* MVQ$ appears on the left and the right hand sides of Eqn. (50), except that they are valuated at the $k-1$ and k stages . Recall that $R^*(U)$ is being driven to zero. The mass matrix M can now be replaced by βI, where I is the identity matrix and β is a constant yielding the following equation:

$$\left[1 + \frac{3}{2\Delta t}\alpha_k \Delta t^* \beta\right] V^{n+1} Q_k = V^{n+1} Q_0 - \alpha_k \Delta t^* R^*(Q_{k-1}) + \frac{3}{2\Delta t}\alpha_k \Delta t^* \beta V^{n+1} Q_{k-1} \tag{52}$$

The method can always be stabilized by increasing β and is akin to using a damped Jacobi method. The implicit Runge-Kutta scheme no longer requires a matrix inversion. For small time steps of the order permitted by the explicit scheme, we find that the choice of $\beta = 2$ stabilizes the scheme.

5.5 Grid adaptation for transient problems

One of the principal advantages of unstructured grids is the ease of adaptation. Adaptive grids are increasingly being used to compute complex unsteady and steady flows. Grid adaptation is particularly useful in transient flows, where features, such as shocks, move through the domain ; it is impractical to refine the grid everywhere. There are three distinct ways the grid can be adapted to the solution. These are r-refinement, h-refinement and p-refinement. In r-refinement, the nodes are simply redistributed so that regions of importance are better resolved. In h-refinement or mesh-enrichment, the cells are locally subdivided or merged or in some instances, a complete remeshing is done to reduce the grid spacing in regions of interest. In p-refinement, the degree of the basis function is adjusted locally to match the variation in solution.

R-refinement is probably the simplest in concept, but is burdened with practical difficulties in multi-dimensions particularly when dealing with highly stretched grids customarily employed for viscous calculations. The difficulties include excessive grid skewness, crossing of lines, arbitrarily

small cell volumes etc. The advantage of r-refinement is that if a valid grid results from it, all that is required is interpolation of variables from the old to the new grid. This could be done in a conservative manner if desired. A way to avoid the interpolation, which introduces errors that could accumulate, is to introduce the grid movement terms in the governing equations Eqn. (1). These terms need to be discretized carefully so that freestream is preserved. In other words, simply moving the grid through the domain should not change the freestream solution. The Geometric Conservation Law (GCL) [117, 143] formalizes this procedure. It can be derived from the continuity equation in Eqn. (1) by assuming the control volumes to be the simplices themselves, both for cell-vertex and cell-centered schemes. Assuming a uniform velocity field and a constant density field, we obtain

$$\frac{\partial \mathcal{V}}{\partial t} + \oint_{\mathcal{S}(t)} [\boldsymbol{V} - \boldsymbol{s}].\boldsymbol{n}\, da = 0, \tag{53}$$

where $\boldsymbol{V}$ is the velocity field and $\boldsymbol{s}$ is the velocity of the boundary $\mathcal{S}(t)$. Since $\boldsymbol{V}$ is constant and the control volume is assumed to be closed at all times so that $\oint_{\mathcal{S}(t)} \boldsymbol{n}\, da = 0$, the equation becomes

$$\frac{\partial \mathcal{V}}{\partial t} - \oint_{\mathcal{S}(t)} \boldsymbol{s}.\boldsymbol{n}\, da = 0. \tag{54}$$

The discrete form of this equation should hold at all time steps and for all the simplices and is called the GCL. Using a forward Euler approximation for the time derivative, we obtain

$$\begin{aligned} \mathcal{V}_I^{n+1} - \mathcal{V}_I^n &= \oint_{\mathcal{S}_I(t)} \boldsymbol{s}.\boldsymbol{n}\, \Delta t\, da \\ &= \sum_j \int_{\Omega_{Ij}} \boldsymbol{s}.\boldsymbol{n}\, \Delta t\, da, \end{aligned} \tag{55}$$

where $\mathcal{S}_I = \sum_j \Omega_{I,j}$ is the surface enclosing the volume $\mathcal{V}_I$ of simplex I. As observed in [143], the term inside the summation represents the volume swept out by the boundary $\Omega_{I,j}$ as the grid points forming that segment move. If grid points are allowed to move arbitrarily, the GCL enables the velocities $\boldsymbol{s}$ to be determined so that the GCL is obeyed. Since simplices are convex, the volumes $\mathcal{V}^n, \mathcal{V}^{n+1}$ are uniquely determined by the positions of the points at time levels n and $n+1$. If the velocity $\boldsymbol{s}_i$ for grid point i is computed by the simple formula

$$\boldsymbol{s}_i = \frac{\boldsymbol{X}_i^{n+1} - \boldsymbol{X}_i^n}{\Delta t}, \tag{56}$$

where $\boldsymbol{X}$ is the position vector, it turns out that the GCL is satisfied. Eqn. (56) simply means that a linear motion of the grid points is assumed between time levels n and $n+1$.

Recently, r-refinement has been used to great advantage with Roe's upwind scheme [106] to obtain "fitted" shock resolution for steady and unsteady two-dimensional flows [98, 100, 124] by aligning the edges of the triangulation with discontinuities. The "fitting" is done in a shock-capturing framework by utilizing that property of Roe's scheme which allows isolated discontinuities aligned with the mesh to be captured exactly. Many of the methods for moving grid points such as the use of exponentially-varying scaling factors [30], tension spring analogies [14, 47, 56] create valid triangulations only when small time steps are used. For large time steps, especially when multiple bodies are present and also for highly clustered viscous grids, these methods usually result in invalid triangulations with crossing grid lines. Retriangulation techniques proposed in [47, 56] would have to be incorporated to recover valid triangulations, but could become expensive. Palmerio [97] presents some interesting techniques for adapting the grid to flow solutions which could be extended to deal with large scale motions of the bodies. A fast regridding procedure will also have applications in design optimization, where the geometry changes during the design cycle.

H-refinement is by far the most popular means of adaptation in compressible flows. This is especially true for inviscid flows dominated by interactions of shock waves where p-refinement techniques are of limited value. The regions of interest are first identified either through a combination of heuristic criteria such as density gradients (undivided) or through estimation of the truncation error. For transient problems, adaptation is performed frequently and therefore the regridding process is required to be efficient. Efficient h-refinement techniques have been developed in [75, 105]. The problem of flow past bodies in relative motion has also been addressed in the literature [74, 44, 120, 56, 68, 27]. Typically, mesh point movement and efficient mesh restructuring are employed to obtain valid, good-quality grids about the moving bodies. Many impressive simulations of flows about bodies in relative motion have been carried out using unstructured grids e.g., [16].

5.6 Applications

First, results from a one-dimensional example are presented illustrating the role of the mass matrix. Observe that for a second order accurate scheme on a uniform mesh with constant Δx, the vertex and the centroid of its control volume coincide. Therefore, the mass matrix can be lumped without suffering any adverse consequences. The situation is different if a mesh with variable mesh widths is considered. In particular, random perturbations about a uniform mesh are considered. Starting with a uniform mesh with constant Δx, each vertex moves randomly towards its left or right neighbor a random amount. The distribution of Δx is shown in Figure 17 with 100 grid points and displays a considerable deviation from the uniform mesh $\Delta x = 0.01$.

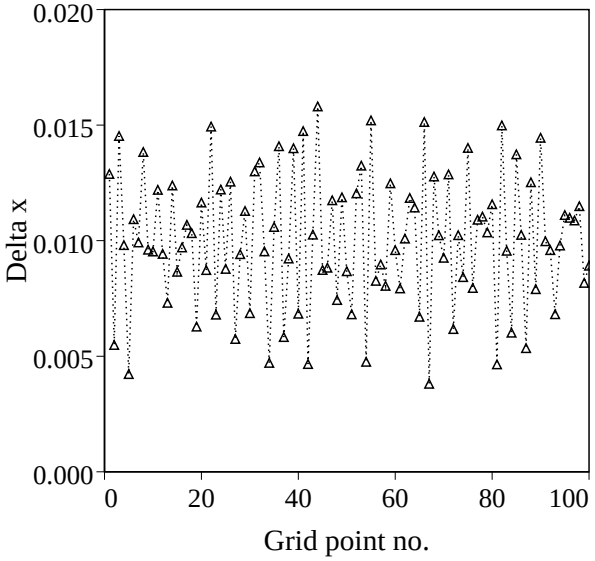

Figure 17: Distribution of the grid spacing in the non-uniform grid with 100 grid points.

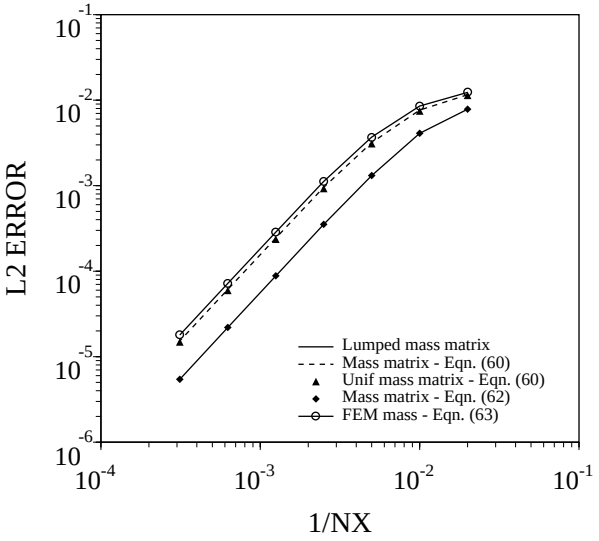

Figure 18: L_2 norms of the errors with various schemes.

The spatial derivative $\frac{\partial u}{\partial x}$ is approximated in a MUSCL scheme [121] as

$$\left(\frac{\partial u}{\partial x}\right)_i = \frac{1}{\Delta x}(u^L_{i+1/2} - u^L_{i-1/2}) \tag{57}$$

The one-dimensional advection equation is solved first using a scheme that is spatially second order accurate. It employs a linear reconstruction procedure:

$$u^L_{i+1/2} = u_i + (x_{i+2} - x_i)\frac{(u_{i+1} - u_{i-1}}{(x_{i+1} - x_{i-1}}. \tag{58}$$

On a uniform grid, this formula reduces to the $\kappa = 0$ scheme. Recall that the formula for the κ scheme [121] is given by

$$u^L_{i+1/2} = u_i + \frac{1-\kappa}{4}(u_i - u_{i-1}) + \frac{1+\kappa}{4}(u_{i+1} - u_i), \tag{59}$$

which on the random grid is only first order accurate for the spatial derivative. The initial condition is a Gaussian and the profile is advected by marching to a fixed time. A grid refinement study is carried out by using a constant CFL number of 0.5 and doubling the mesh size starting from 50 grid points. The mass matrix, which is tridiagonal, is inverted using Thomas algorithm. We have experimented with two definitions of the mass matrix. The first one assumes a piecewise-linear distribution of data. The entries in the mass matrix for vertex i are given by deriving the formula for the average value $\overline{u}$ in the interval $\left[x_{i-1/2}, x_{i+1/2}\right]$:

$$\frac{1}{4}\frac{x_i - x_{i-1}}{(x_{i+1} - x_{i-1})}u_{i-1} + \frac{3}{4}u_i + \frac{1}{4}\frac{x_{i+1} - x_i}{(x_{i+1} - x_{i-1})}u_{i+1} \tag{60}$$

A second definition of the mass matrix is derived as the average of the reconstruction polynomial within a cell which for this scheme is

$$u(x) = u_i + (x - x_i)\frac{u_{i+1} - u_{i-1}}{x_{i+1} - x_{i-1}}. \tag{61}$$

The mass matrix then becomes

$$-\frac{1}{8}\frac{x_{i-1} - 2x_i + x_{i+1}}{(x_{i+1} + x_{i-1})}u_{i-1} + u_i + \frac{1}{8}\frac{x_{i-1} - 2x_i + x_{i+1}}{(x_{i+1} + x_{i-1})}u_{i-1} \tag{62}$$

Figure 18 compares the errors in L_2 norm with the mass matrices given by Eqn. (60) and Eqn. (62), and with the lumped mass matrix. All the schemes exhibit second order accuracy and the errors are larger with the mass matrix given by Eqn. (60). The results obtained with the lumped mass matrix are almost identical to those obtained with Eqn. (62). As per the earlier discussion, Taylor series expansion would imply a first order error with the lumped mass matrix, whereas Figure 18 clearly indicates second order accuracy. The results therefore reveal the inadequacy of local analysis. Figure 18 also shows the results when we assume a uniform grid just for evaluating the mass matrix in Eqn. (60), so that the entries become $1/8, 3/4, 1/8$ and exhibits almost no difference. Note that when such an assumption is made with Eqn. (62) it results in a lumped mass matrix. The results obtained with the usual finite element mass matrix

$$\frac{1}{6}u_{i-1} + \frac{4}{6}u_i + \frac{1}{6}u_{i-1}, \tag{63}$$

are also shown in Figure 18 and again displays larger errors compared to the lumped mass approximation. The reason for this is that the finite element mass matrix is consistent with a Galerkin method which results central difference discretization, whereas the spatial differencing employed here is upwind-biased. After experimenting with a one-parameter family of mass matrices, we have found that the lumped mass matrix gives the lowest errors with this particular spatial discretization.

It is well known in finite element literature [116] that in some cases the lumping of the mass matrix does not compromise the solution accuracy but that the mass matrix may play a crucial role when higher-order discretizations are considered. To this end, the $\kappa = 1/3$ is used to discretize spatial derivative to third order accuracy, on a uniform grid. The finite volume mass matrix with a quadratic distribution within each cell now reads

$$\frac{1}{24}u_{i-1} + \frac{11}{12}u_i + \frac{1}{24}u_{i+1} \tag{64}$$

Figure 19 shows the error plots for the $\kappa = 1/3$ scheme with the lumped and the full mass matrices. The use of the mass matrix degrades the scheme to second order accuracy whereas the lumped mass matrix yields third order accuracy. At first glance, this would appear surprising. However, if we recall that the κ formula assumes cell-averaged quantities, it is clear that the mass matrix should be equal to the identity matrix. It is *wrong* to use any other mass matrix when dealing with schemes based on cell-averaged values. To obtain a third order accurate scheme based on point values, the following formula should be used:

$$u^L_{i+1/2} = -\frac{1}{8}u_{i-1} + \frac{3}{4}u_i + \frac{3}{8}u_{i+1} \tag{65}$$

Figure 19 also shows the error plots for this third order accurate scheme with the full and the lumped mass matrices. It shows that with the lumped mass matrix only second order accuracy is achieved, whereas using the full matrix given by Eqn. (64) yields the third order accuracy of the spatial discretization. We have observed that using any other definition for the mass matrix degrades the accuracy to second order. The standard Runge-Kutta scheme that is fourth order accurate in time is used for the higher order computations.

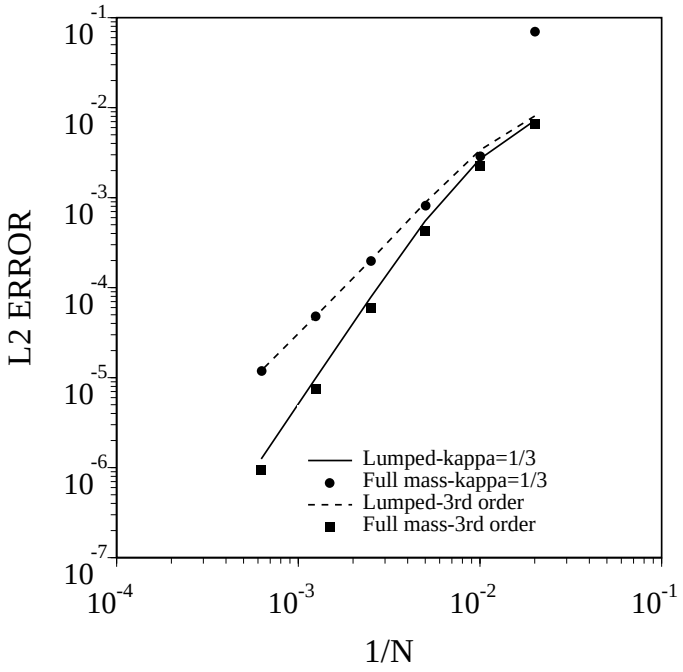

Figure 19: L_2 norms of the errors with higher-order schemes.

The implications for the scheme in multiple dimensions are clear. As long as only a second order accurate scheme is used and we operate with either cell-vertex or cell-averaged data, the mass matrix may be lumped without any loss of order of accuracy. The mass matrix can also be ignored for third (and higher) order accurate schemes as long as only cell-averages are used. If point values are used to construct third and higher order accurate schemes, the accuracy will degrade if the mass matrix is lumped. For higher order accurate schemes based on point values, the indirect mass matrix inversion technique discussed earlier will help preserve the order of accuracy of the scheme.

We next present results from two-dimensional inviscid calculations over pitching airfoils. The transonic flow is over a sinusoidally oscillating NACA0012 airfoil where the angle of attack $\alpha(t)$ varies according to the formula

$$\alpha(t) = \alpha_m + \alpha_0 sin(\omega t) \tag{66}$$

For the test case chosen, $\alpha_m = 0.016°$, $\alpha_0 = 2.51°$, $\kappa = \frac{\omega c}{2U_\infty} = 0.0814$ and the freestream Mach number, $M_\infty = 0.755$. Computing this flow using an explicit scheme is very time-consuming because of the low frequency. Flows are computed using two meshes, referred to as GRID1 and GRID2, each having 6336 vertices. These are shown in Figures 20 and 21, respectively. GRID1 is generated by drawing diagonals in a structured C-mesh and is fairly uniform. GRID2 is generated by random perturbations on GRID1 by a procedure similar to that employed in the one-dimensional example described earlier. Figure 22 shows the lift histories during the third cycle of oscillation. Four curves are depicted, namely, the histories with the lumped and full mass matrices for GRID1 and GRID2. The mass matrix is derived by using a definition similar to Eqn. (60).

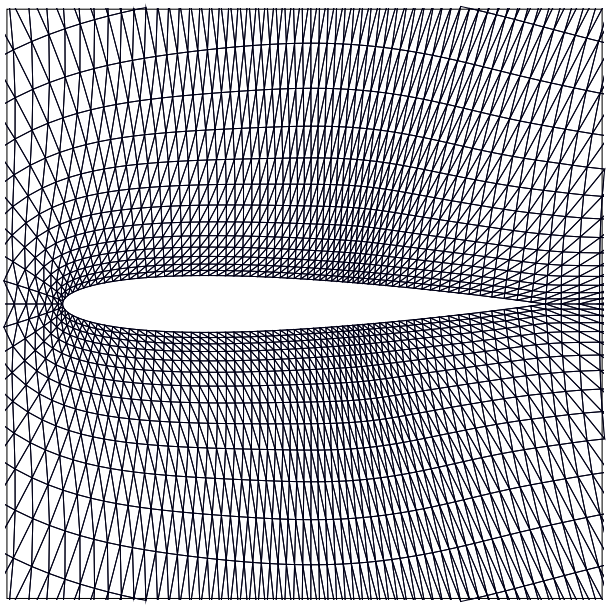

Figure 20: GRID1 about an NACA0012 airfoil with 6336 vertices.

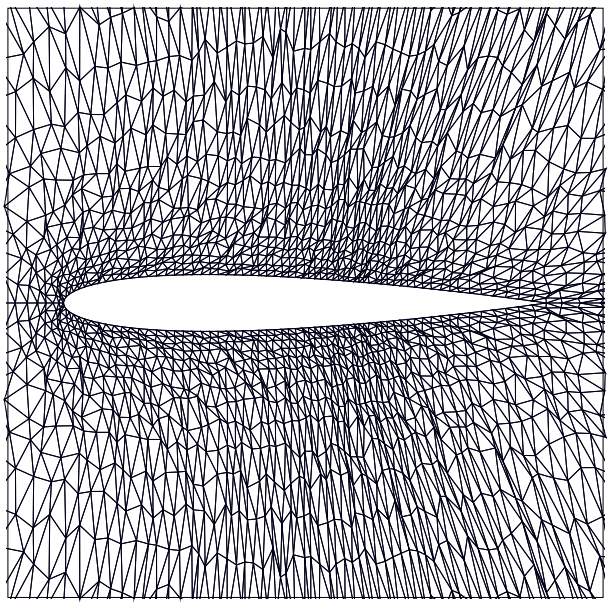

Figure 21: GRID2 about an NACA0012 airfoil with 6336 vertices.

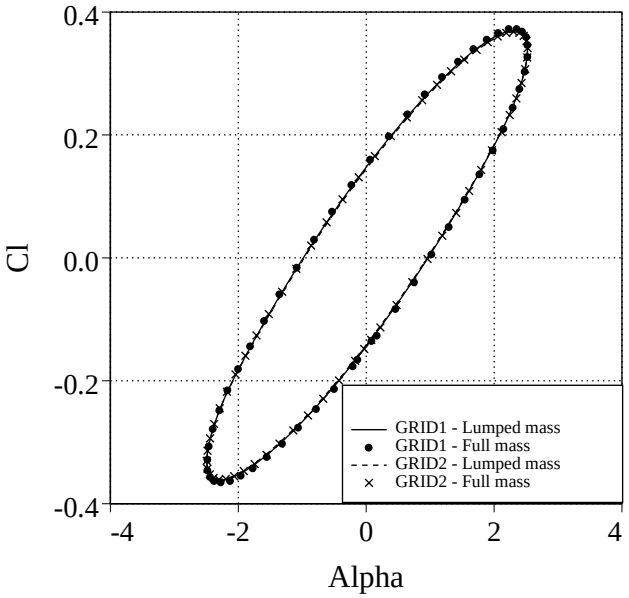

Figure 22: Lift histories during the third cycle of motion.

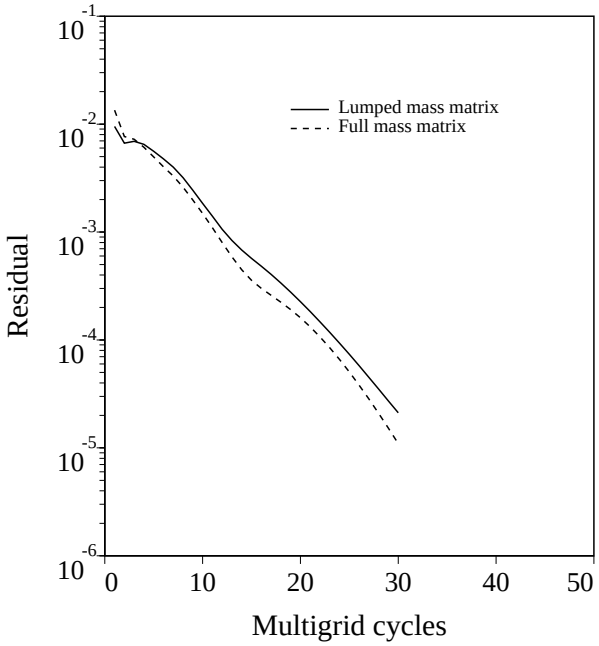

Figure 23: Histories of the unsteady residual at a particular time step.

As expected, the mass matrix has little impact on the integrated quantities even in the random mesh. The differences in the solutions between the two grids are likewise insignificant. The CPU time increases by about 15% when the full mass matrix is included. These examples have been run with a maximum physical CFL number of 500, corresponding to using 54 time steps per sinusoidal oscillation of the airfoil. The number of iterations for the inner multigrid procedure is fixed at 30. Figure 23 shows the convergence of the agglomeration multigrid procedure during a particular time step with the lumped and the full matrices. The L_2 norm of the unsteady residual R^* is plotted as a function of the multigrid cycles. The convergence improves slightly when the mass matrix is included. The reason for this improvement is furnished by inspecting a one-dimensional situation. The mass matrix for a second order accurate scheme given by Eqn. (60) becomes on a uniform grid

$$\frac{1}{8}u_{i-1} + \frac{3}{4}u_i + \frac{1}{8}u_{i+1} \tag{67}$$

This can be rewritten as

$$u_i + \frac{1}{8}u_{i-1} - \frac{2}{8}u_i + \frac{1}{8}u_{i+1} \approx \left[I + \frac{1}{8}\Delta x^2 \frac{d^2}{dx^2} \right] u_i, \tag{68}$$

where a centered second order accurate difference formula is used to approximate $\frac{d^2u}{dx^2}$. Written in this form, the equation is similar to that used in residual averaging technique, discussed in Section 5.2. Finally, Figure 24 shows the effect of the physical time step size. Two lift histories are shown, one for a CFL number of 500 and the other for 1000 using the lumped mass matrix. The integrated quantities show slight discrepancies near the ends of the oval region. This may be due to two causes – one, that the physical time step is too large and second, that the multigrid procedure is not converged. The number of inner multigrid cycles is fixed at 30 and the convergence is worse with the higher CFL number. This opens up the question of when to declare the inner iteration converged. Ideally, this system should be solved only until the residual matches the truncation error.

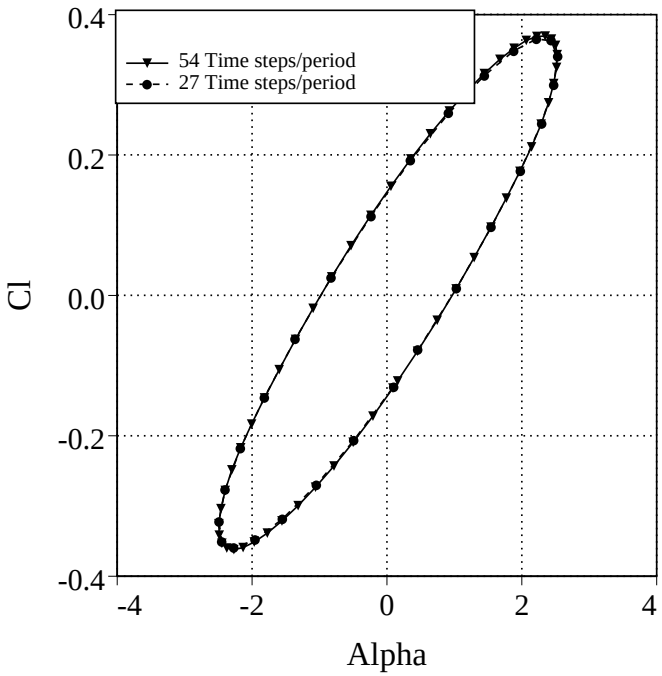

Figure 24: Lift histories during the third cycle of motion with CFL = 500 and CFL = 1000.

6 Parallel computing issues

Computational fluid dynamics (CFD) as its name implies is inevitably linked to computing issues. Among these are processing power, memory technology, networking and accessibility. Ability to compute the solutions to problems in finite time always being the goal, CFD has benefited immensely from the revolution that has taken place in the last 15 years in these areas. Vector supercomputers have provided much of the computing power that has been harnessed to compute complex three-dimensional flows. It is anticipated that distributed-memory parallel computers will offer the next cost-effective leap forward in terms of computing power. For the goal of sustained high performance on these machines to be realized, however, many fundamental issues need to be addressed. Among these are scalable algorithms and software.

In the case of unstructured grid computations on parallel parallel platforms, a number of issues need to be addressed. Interfacing with geometry packages and grid generation should be ideally done on the parallel computer itself. Following this, the grid and the data may need to repartitioned so that communication is minimized. The next stage is the flow solver, which could be explicit or implicit. Finally, the parallel aspects of flow visualization and other postprocessing techniques cannot be overemphasized. Each of the areas mentioned above could become a sequential bottleneck, limiting performance. Parallel unstructured grid generation has been investigated by a number of researchers. Löhner et al. [76] have implemented advancing front grid generation algorithm in two dimensions on Intel iPSC/860. Merriam [90] has implemented a Delaunay triangulation method on the Intel iPSC/860 for three-dimensional point-sets. Still, parallel grid generation is not commonplace. The reason for this seems to be that although grid generation is a complicated, time-consuming procedure, the stumbling block is not excessive computational effort. Rather, interfacing with geometry packages and ensuring high quality of grids are the pacing items. Thus, modern workstations that can generate up to 1000 tetrahedra a second appear to be adequate for the task of grid generation. Parallel grid generation may be of more importance in unsteady simulations involving motion of bodies where the grids may have to be regenerated periodically. Partitioning the grid among processors in a judicious manner is important since it has a significant impact on the parallel performance of the flow solver. When unstructured grid computations are carried out on parallel computers, extracting parallelism out of the flow solver is an important task. In the case of adaptive grid computations, maintaining load balance among the processors is also an important consideration. Finally, although it is possible to concatenate the data from different processors on a workstation for post-processing and visualization, it is clearly not a viable option as the memories of the processors and problem sizes continue to grow. Therefore, parallel visualization techniques have to be utilized.

Explicit schemes used in computational fluid dynamics possess almost complete parallelism. They require only simple update procedures that involve local dependencies. On a parallel computer, such schemes typically require communication only to nearest neighbors. Implicit schemes, on the other hand, require the solution of coupled equations which involves global dependencies. On distributed-memory parallel computers, the design of implicit schemes is more difficult since parallelism and load balance during the implicit phase are additional considerations. In reference [129], the implicit schemes discussed in Section 4.2 were investigated for parallelism on the Intel iPSC/860. The results from this study are reported in this section. In reference [66] an implicit iterative solution strategy based on the diagonal-preconditioned matrix-free GMRES algorithm [24] was implemented on the Connection Machine. Ramamurthi et al. [104] have developed and tested on an Intel iPSC/860 an implicit incompressible flow solver that uses a "linelet-based" preconditioner (see Section 4.2.3) Ajmani et al. [2] have investigated the use of a preconditioned GMRES implicit method for the solution of the Navier-Stokes equations using structured grids on the Intel iPSC/860. Venkatakrishnan et al. [135] and Das et al. [83] have shown that it is possible to obtain supercomputer performance when solving explicit unstructured grid problems

on the Intel iPSC/860. By paying careful attention to the partitioning of the mesh, communication schedule and data structures, they have been able to show that nearly 2-4 times the speed of a Cray Y-MP/1 could be obtained with 128 processors of the iPSC/860. The effects of using various strategies for partitioning the unstructured grids on communication costs have been examined as well. Unsteady and steady viscous flows have been computed in [41] using an explicit scheme on unstructured grids. More recently, Barth [13] has also obtained excellent performance when solving the three-dimensional Euler equations using the implicit scheme discussed in Section 4.4 on the IBM SP2.

In this section, we first discuss in detail, the problem of partitioning of the grid and the data for parallel computers. Next, the issues involved in parallelizing finite volume schemes for solving the Euler equations on triangular unstructured meshes in MIMD (multiple instruction/multiple data stream) fashion are outlined. As a candidate explicit scheme, a four-stage Runge-Kutta scheme is used to solve two-dimensional flow problems. The implicit schemes outlined in Section 4.2 are explored as candidate schemes to solve these problems on parallel computers. The issues in implementing the GMRES algorithm and the preconditioners in a distributed-memory environment are addressed. The methods are compared both in terms of elapsed times and convergence rates. Results for a typical flow around a multi-element airfoil are presented and the performances of the explicit and implicit schemes on the Intel iPSC/860 are compared. It is shown that the implicit schemes offer adequate parallelism at the expense of minimal sequential overhead. Following domain decomposition ideas, the use of a global coarse grid to further minimize this overhead is also investigated. The full details of the parallel implementations may be found in [129]. Finally, we present some techniques for load balancing, which are important when adaptive grid computations are carried out on parallel computers.

6.1 Partitioning of grids

We begin with a few definitions. An *undirected graph* is defined as a set of vertices joined by edges. It is symmetric in that if vertex A is connected to B, B is connected to A as well. In the context of unstructured grid flow solvers, the graph can thus be viewed as a collection of first order stencils. Vertices A and B are termed *nearest neighbors* if there exists an edge in the graph linking A and B. Recall that the stencil for a higher order accurate cell-vertex scheme involves next-to-nearest neighbors as well. However, in most finite volume schemes, the information from the next-to-nearest neighbors enters in the form of gradients evaluated at nearest neighbors. Thus the graph of the problem for a cell-vertex scheme is the underlying grid itself. If the scheme made use of information from vertices other than nearest neighbors directly, the proper graph to consider should include edges connecting the vertex in question to those vertices as well. This is seldom done in practice because the problem graph would become more dense in such cases.

The partitioning of unstructured grids among processors should be carried out in a manner as to minimize the execution time. The execution or wall clock time is the maximum over all processors the sum of the times required for computation and communication. The computational work (load) is typically a function of the number of grid points and sometimes, a function of the shape of the domain as well. The dependence on the shape of the domain arises, for example, when a banded solver is used to invert a linear system of equations within each processor. For most practical CFD computations, however, the computational work is only a function of the number of grid points contained within each processor.

$$t_i^{comp} = \alpha N_i^a, \tag{69}$$

where α is the time taken to process one grid point and N_i is the number of grid points in the partition. In most CFD solvers, the work is directly proportional to the number of vertices, so that $a = 1$in Eqn. (69). In the case of cell-vertex schemes on unstructured grids, the computational work involved in the computation of residuals is directly proportional to the number of edges which

is linearly related to the number of vertices in the grid. A typical model for communication time t_{ij}^{comm} between two processors i and j is given by

$$t_{ij}^{comm} = t_s + \beta d_{ij} + \gamma m_{ij}. \tag{70}$$

Here t_s is the cost of start-up (also known as latency), β is the time required for communication between nearest neighbors in the given topology of the computer, d_{ij} is the number of hops between the two processors in the topology, γ is the time required to communicate one byte and m_{ij} is the number of bytes being communicated. Thus the total execution time for processor i is given by

$$\begin{aligned} t_i^{tot} &= t_i^{comp} + \sum_{j \in \mathcal{N}_i} t_{ij}^{comm} \\ &= \alpha N_i^a + |\mathcal{N}_i| t_s + \sum_{j \in \mathcal{N}_i} \beta d_{ij} \\ &+ \sum_{j \in \mathcal{N}_i} \gamma m_{ij}, \end{aligned} \tag{71}$$

where $\mathcal{N}_i$ is the set of neighboring partitions of i with a cardinality of $|\mathcal{N}_i|$. Minimizing the maximum t_i^{tot} over all processors is a very difficult problem because it ties in the characteristics of the parallel computer, such as the topology and the communication model, to the algorithm used to solve the problem. Rather than solve this difficult problem, we will examine each of the terms in Eqn. (71) individually. There is no guarantee that the piecemeal approach to the partitioning problem will minimize the execution time.

The first term in Eqn. (71) deals with the time to carry out the computations. This is minimized if the partitioning problem guarantees that N_i is equal across all processors. The last term deals with the transmission costs and is related to the number of cut-edges. The number of cut-edges is a good metric for assessing the various partitioning strategies with the goal of minimizing it. In a cell-vertex scheme, this metric is only a rough measure since the message lengths are proportional to the number of vertices that are on either side of the cut-edges.

The penultimate term in Eqn. (71) is usually dealt with separately and is referred to as the *embedding* problem. This term is getting less important with switches and wormhole routing in modern parallel architectures. Embedding deals with the assignment of partitions to processors. More precisely, it is the embedding of the partition communication graph to the processor graph. An embedding of a graph G onto a graph H is a one-to-one assignment of a vertex in G to a vertex in H. A *partition communication graph* (PCG) is defined as an undirected graph with vertices representing the partitions and edges representing communication link between two neighboring partitions. Figure 25 shows a decomposition of a domain into eight partitions and the corresponding PCG. An optimal assignment of partitions to processors is an embedding that minimizes the dilation cost, which is defined as the maximum distance in H between the images of vertices that are adjacent in G [58]. Figure 26 shows a processor graph, which is a hypercube interconnect for 8 processors. Figure 26 also shows an embedding of the PCG shown in Figure 25. The processor numbers are shown as binary numbers while the partition numbers are shown in parantheses as Arabic numerals. It is easy to see that the dilation cost for this mapping is 2. Heuristic techniques are usually employed to derive good embeddings. On parallel computers with hypercube interconnect, embedding does not appear to be much of an issue. The assignment of partitions to processors could be more critical if the processor network is less dense e.g. a two-dimensional mesh.

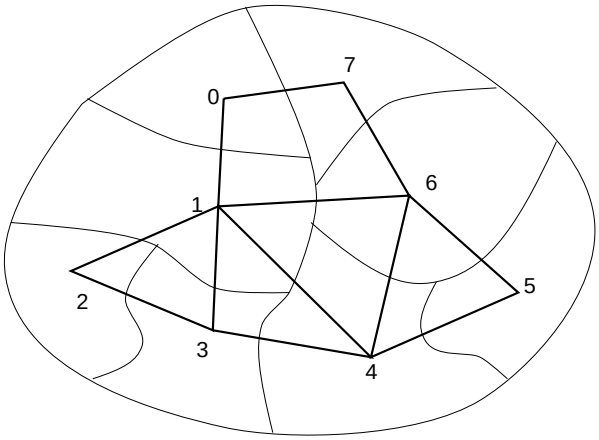

Figure 25: 8-way decomposition of domain and its associated partition communication graph.

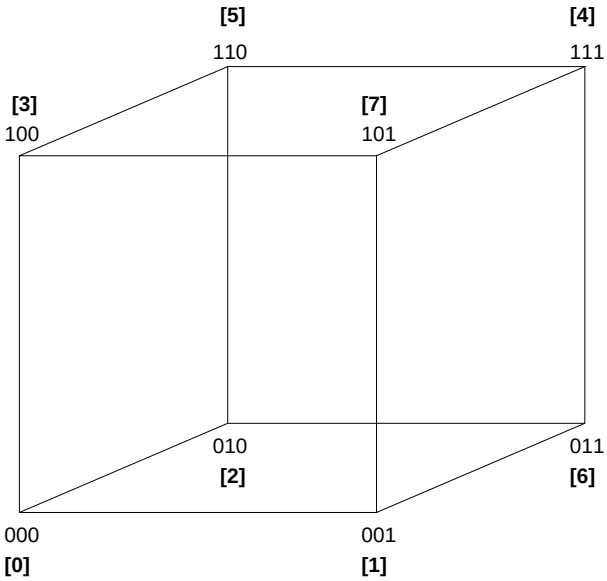

Figure 26: 8-processor hypercube interconnect and an assignment of processors.

The second term in Eqn. (71) depends on the number of neighbors of a partition. This can be minimized if desired by using the so-called stripwise partitioning strategies[135]. Usually, however, minimizing this leads to an inordinate increase in the cut-edges [135] and communication costs. An example is given for a simple square domain. Figure 27(a) shows a 16-way domainwise partitioning of a square whereas Figure 27(b) shows a 16-way strip-wise partitioning of the domain.

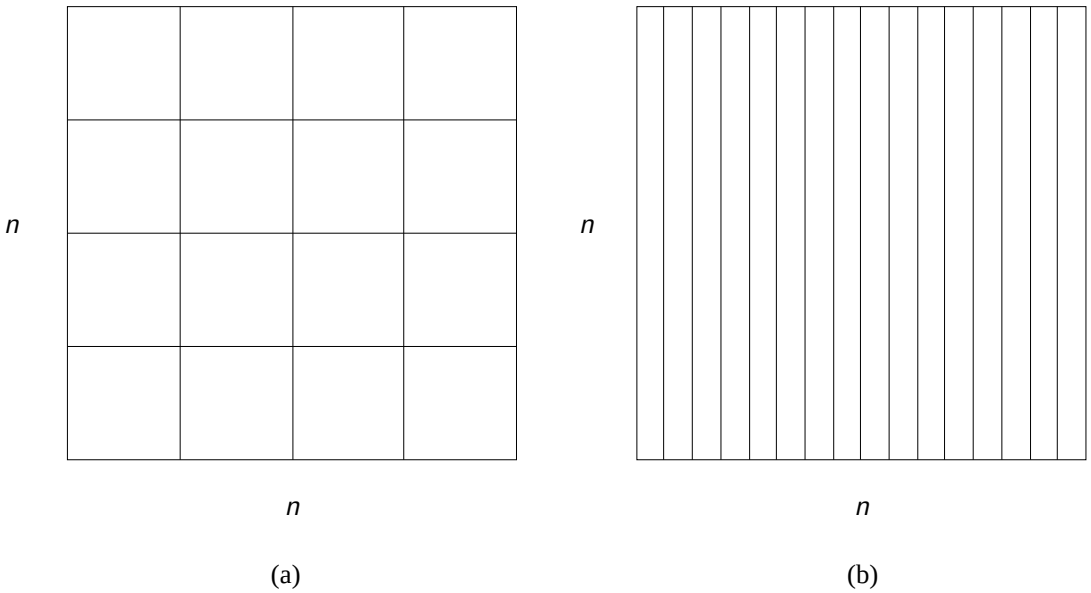

Figure 27: 16-way partitioning of a square (a) domainwise (b) stripwise.

Assume that the domain has $n \times n$ grid points and that the communication takes place across the edges of the partitions and that the length of the messages is equal to the number of grid points along the boundaries between partitions. The domainwise partitions have a maximum of 4 neighbors and the stripwise partitions, a maximum of 2. The computational time is the same,

whereas the communications times are different. Assuming a simple communication model, Eqn. (70) with $\beta = 0$, the communication cost with domainwise partitioning for an interior processor is

$$t^{comm} = 4(t_s + \gamma n/4), \tag{72}$$

and for stripwise partitioning

$$t^{comm} = 2(t_s + \gamma n), \tag{73}$$

This example shows that unless $t_s > \gamma n/2$, the domainwise partitioning strategy is better. On most modern parallel computers, the latency is small enough that minimizing the number of neighbors is not necessary, although it should be reasonably bounded.

The partitioning algorithms discussed below, create partitions that have the same computational loads and are applied to two-dimensional triangular grids. The efficacies of the partitioning strategies are assessed by inspecting the number of cut edges and also by measuring the communication times in applications. Das et al. [83] and Johan et al. [66] have applied the algorithms to three-dimensional problems. It is assumed that a cell-vertex scheme is employed. One has the choice of either partitioning triangles or the vertices themselves. Vertices are partitioned by applying the algorithms to the graph represented by the triangulation itself, whereas triangles are partitioned by considering the dual, where the triangles are represented by vertices which are connected by dual edges. In [129], we have examined using both these strategies for a cell-vertex scheme and find both the schemes lead to similar execution times. In the examples shown below however, the triangles will be assinged uniquely to partitions; therefore, the dual graph is partitioned. Figure 28 shows a triangulation and Figure 29 shows the corresponding centroidal dual; each vertex in Figure 28 corresponds to a triangle in Figure 28.

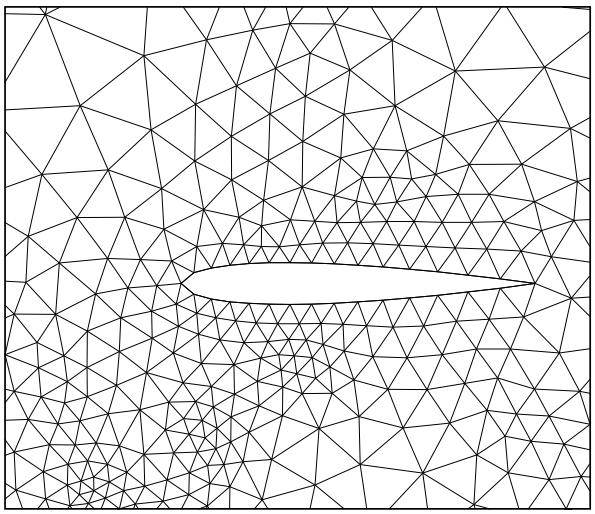

Figure 28: A triangulation.

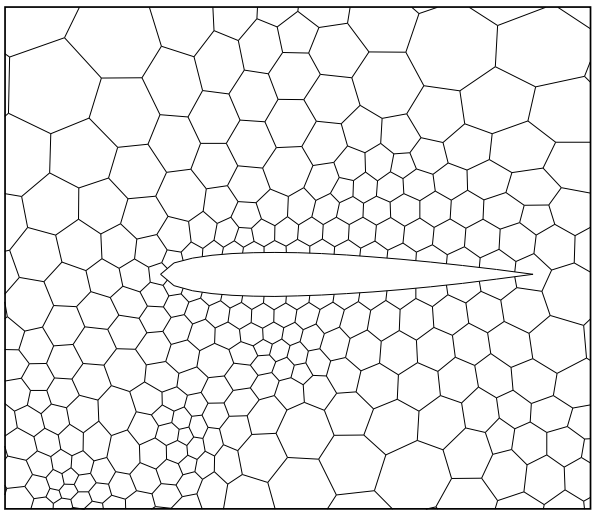

Figure 29: Dual graph.

Partitioning is done recursively starting with the problem of dividing one domain into two, almost equal subdomains. The number of vertices in the two subdomains differs at most by one. There are two classes of partitioning schemes. The first class utilizes the coordinates of the vertices and does not make use of the problem graph. The second class does not use the coordinates but uses only the graph information.

The *coordinate bisection* strategy uses the coordinate information associated with each vertex. The coordinates are then sorted in a particular coordinate direction (either x or y). Typically, the direction containing more number of points is chosen as the direction in which to sort. One half of the ordered vertices define the first partition and the remaining vertices define the second partition. The advantage of the coordinate bisection method is that it is extremely efficient because

the sorting can be done in $logN$ operations, where N is the total number of vertices. Variants of this method include inertial bisection method [42] and parametric binary dissection [21]. In the inertial bisection method, instead of sorting in the coordinate directions, a different coordinate system is used. In the case of parametric binary dissection, load balance is sacrificed in order to improve the total execution time.

The *graph bisection* techniques view the unstructured grid as an undirected graph and partition the graph by finding the graph separator by any of a number of methods. A separator is a set of nodes that subdivides the original connected graph into two disjoint subgraphs. The sizes of the separators have a direct bearing on the fill-in that occurs during the factorization of sparse matrices, but are also important in the context of partitioning since they form the interpartition boundaries. One way to derive a separator is to first form the rooted level structure defining level sets. These represent the neighbor lists starting with a root, neighbors of the root, neighbors of neighbors of the root and so on. The Cuthill-McKee algorithm [49] generates a rooted level structure as a first step. The two partitions are defined when one half of the domain has been traversed. Another way to find a graph separator is called the *spectral bisection* method and is based on the spectral partitioning algorithm of Pothen et al. [101]. Their algorithm induces the partitions from the eigenvector corresponding to the second smallest eigenvalue of the Laplacian matrix associated with the graph. The elements of $n \times n$ Laplacian matrix L_{ij} of an undirected graph with n vertices are defined as

$$\begin{aligned} L_{ij} &= -1 \quad \text{if } i \neq j \text{ and an edge connects } i \text{ and } j \\ L_{ij} &= 0 \quad \text{if } i \neq j \text{ and no edge connects } i \text{ and } j \\ L_{ij} &= D \quad \text{if } i = j, \end{aligned} \tag{74}$$

where D is the the degree of vertex i. The smallest eigenvalue of this matrix is 0 with an eigenvector of $(1, ...1)$. The eigenvector corresponding to the second smallest eigenvalue is determined by a Lanczos algorithm. The entries of this eigenvector are sorted and split along the median to produce equally-sized partitions. Pothen et al. [101] have shown that the separators produced by this algorithm are shorter than those produced by other techniques. Barth in [1] presents a simple proof that the spectral bisection technique minimizes number of cut-edges.

Simon [112] has applied these three partitioning algorithms to a variety of two- and three-dimensional grids and has shown that the spectral bisection technique yields better partitions in that it produces subdomains with shorter boundaries. He has observed that the coordinate bisection technique leads to disconnected partitions, thereby greatly increasing the lengths of the boundary segments. Disconnected partitions also have the undesirable effect of increasing the number of adjacent partitions, and each adjacent partition requires a message to be generated. Therefore, disconnected partitions imply higher start-up and transmission costs. The graph bisection technique using level sets produces partitions with long boundaries since it uses a breadth-first search to define the level sets. The spectral bisection technique produces uniform, mostly connected subdomains with short boundaries. Theoretical results by Fiedler (summarized in [101]) show that one of the two subdomains formed by the spectral partitioning is always connected. Spectral partitioning results in fewer shorter length messages and reduced communication costs.

Figures 30(a), 30(b), and 30(c) show eight-way decompositions for a mesh around a four-element airfoil obtained with coordinate bisection, graph partitioning based on level sets and spectral partitioning, respectively. The interpartition boundaries are shown by the thick lines in these figures. We observe from Figure 30(a) that even with only eight partitions, there are instances when the subdomains degenerate to zero thickness. This is a direct consequence of the variable density of the grid within a rectangular coordinate strip and leads to disconnected domains for a larger number of partitions. In Figure 30(b) we see that the partitions produced by the level sets strategy have long boundaries and are connected, but as the number of partitions increases, we have observed

that the partitions become disconnected. In Figure 30(c) we notice that the partitions produced by the spectral bisection technique are compact, and this property seems to hold even as the number of partitions is increased.

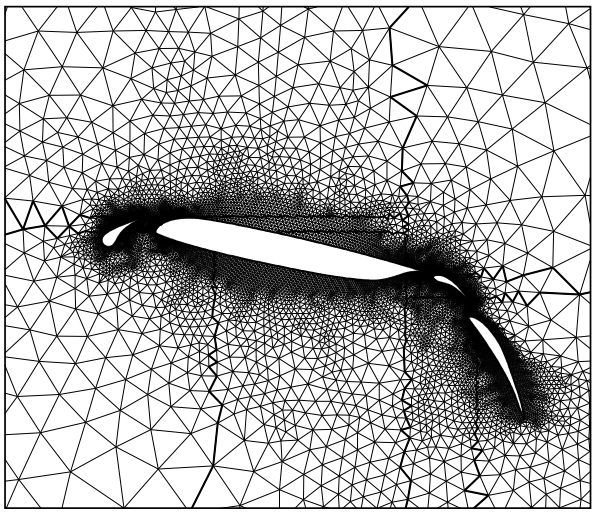

Figure 30(a): 8-way decomposition with coordinate bisection. communication graph.

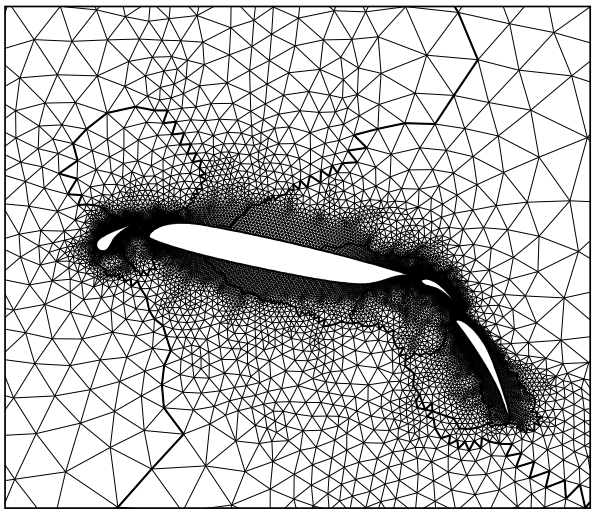

Figure 30(b): 8-way decomposition with graph bisection using level sets. processors.

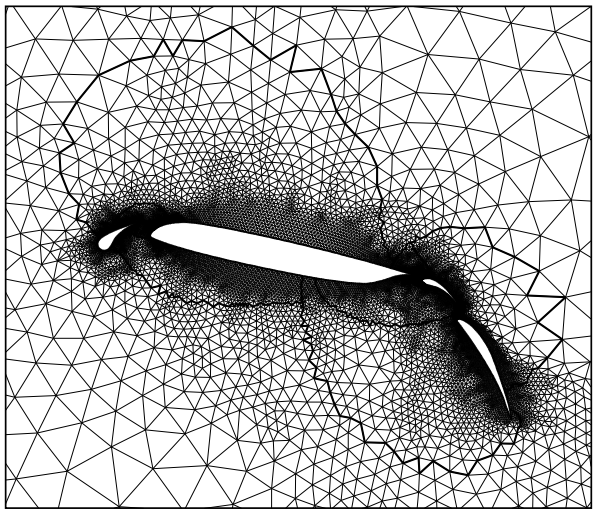

Figure 30(c): 8-way decomposition with graph bisection using level sets. processors.

The execution times for the coordinate, level sets, and spectral bisection techniques for a 64-way partitioning of a triangular mesh with 15606 vertices on a Silicon Graphics workstation (Iris 4D/70) in 32-bit arithmetic are 4, 3, and 1750 seconds, respectively. On the Cray Y-MP/1 the timings without vectorization are 3.70, 3.96, and 399.26 seconds and 0.76, 1.04, and 26.6 seconds with vectorization. The performance of spectral bisection improves considerably with vectorization because the matrix-vector products are vectorized on the Cray Y-MP. The spectral bisection is thus expensive. The multi-level spectral bisection scheme of Barnard and Simon. [7] can be used to improve the efficiency. The execution time on the workstation is reduced to 200 seconds to partition the same grid. The multi-level spectral technique does not create the same partitions as the original spectral algorithm due to round-off and sensitivity to stopping criteria in the Lanczos algorithm.

A different partitioning strategy based on coordinates has been tested by Gilbert et al. [51] based on the theoretical work of Miller et al. [91]. A brief description of this relatively new partitioning algorithm, called *geometric partitioning* is given here.

Project Up. A stereographic projection of the point set in $\mathbb{R}^d$ onto a higher-dimensional unit sphere is carried out. Assume $\mathbb{R}^d$ is embedded in $\mathbb{R}^{d+1}$ as the $x_{d+1} = 0$ coordinate plane and assume a unit sphere U_d embedded in $\mathbb{R}^{d+1}$ centered at the origin. Given a point p in $\mathbb{R}^d$, construct a line L in $\mathbb{R}^{d+1}$ passing through p and through the north pole of U_d. The line L must pass through another point q of U_d; the point $ST(p) = q$ is defined as the image under the stereographic projection mapping. Thus the entire point set $P = \{p_1, ..., , p_n\}$ in $\mathbb{R}^d$ is mapped onto $ST(P) = \{ST(p_1), ..., ST(p_n)\}$.

Find Centerpoint. This is a special point in the interior of U^d. The *centerpoint* of a point set is defined as one such that every hyperplane passing through it about evenly divides the point set. A more precise definition may be found in [51].

Conformal map: Rotate and Dilate. This step moves the centerpoint conformally to the origin. It is accomplished in two steps. First a rotation on U_d is carried out so that the center-point c is mapped onto the diameter between the north and south poles of U_d i.e., the new center point is $c_1 = (0, 0, ...r)$. Following a mapping back to $\mathbb{R}^d$ by using the inverse transformation ST^{-1}, the points in $\mathbb{R}^d$ are scaled by a factor $\sqrt{(1-r)/(1+r)}$. The scaled points are projected back onto U^d by another application of ST.

Find Great Circle. A random great circle (a sphere in $\mathbb{R}^d$) is chosen on the unit sphere U^d.

Unmap and Project Down. The great circle is transformed to a circle in $\mathbb{R}^d$ by applying the inverse of the transformations. The resulting circle in $\mathbb{R}^d$ represents the boundary between partitions.

The center-point computation is computationally expensive involving $(O(N^d))$ operations and Miller et al. [91] have proposed a sampling strategy. The theoretical results in [91] indicate that provably good separators can be obtained.

The advantage of geometric partitioning over coordinate bisection is that it produces separators that are arcs of circles (in two-dimensions) as opposed to straight lines. It is also much less expensive compared to graph bisection techniques. Since it only deals with the coordinates of the point set, there are many situations where the graph bisection techniques will be better, e.g. a two-dimensional graph embedded in 3-dimensional space.

6.2 Communication issues

After partitioning, global values of the data structures required to define the unstructured mesh are given local values within each partition in a preprocessing step. We thus dispense with any references to global indices. In the present implementation, each local data set also contains the information that a partition requires for communication at its interpartition boundaries. The information required for communication at the interpartition boundaries is precomputed using sparse matrix data structures. These are outlined for a cell-vertex scheme where vertices are partitioned.

The data structures required for communication and stored by each processor consist of:

nadjproc - no. of adjacent processors (processors handling adjacent partitions)

iadjproc - list of adjacent processors; length nadjproc

ibvs - pointers to the cumulative number of interior boundary vertices that need to send information in common with the adjacent processors; length nadjproc+1

nbvs - number of boundary vertices in common with processor iadjproc(j) that need to send information. This can be derived from ibvs and is not stored; nbvs(j) = ibvs(j+1)–ibvs(j)

nintbvs(.,1) - Local indices for the vertices sending information on current processor; length ibvs(nadjproc+1)–1

nintbvs(.,2) - Local indices on adjacent processor receiving information; length ibvs(nadjproc+1)–1

ibvr - pointers to the cumulative number of interior boundary vertices in common with the adjacent processors that need to receive information; length nadjproc+1

nbvr - number of boundary vertices in common with processor iadjproc(j) that need to receive information. This can be derived from ibvr and is not stored; nbvr(j) = ibvr(j+1)–ibvr(j)

nintbvr(.,1) - Local indices on current processor receiving information; length ibvr(nadjproc+1)–1

nintbvr(.,2) - Local indices on adjacent processor sending information; length ibvr(nadjproc+1)–1

The arrays **nintbvs(.,2)** and **nintbvr(.,2)** can be dispensed with if the numberings in the adjacent processors are done in a consistent manner. The data structures are illustrated by means of an example. Figure 31 shows a three-way partition with the inter-partition boundaries indicated by the thick lines. Each of the vertices shown is stored by two or three processors. The entries of the data structures for processor 0 are also shown in the figure.

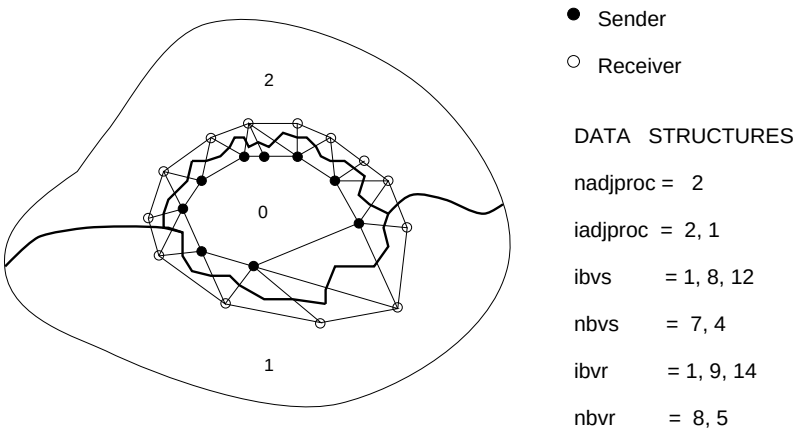

Figure 31: 3-way decomposition showing only the triangles intersecting the partition boundaries and data structures for processor 0.

Regarding the assignment of partitions to processors, a naive mapping is done. This simply maps partition 0 to processor 0 and so on. As was discussed in the last section, it is possible to do a near-optimal mapping by heuristics. In [135], we evaluated the naive mapping against a random mapping for an unstructured problem on 64 nodes of the Intel iPSC/860 and observed little difference in performance. Reasons are also given [135] as to why the assignment of partitions to processors is not crucial. We simply note here that since the partitioning is done in a recursive manner, spatial locality is imposed on the partitions. For example, the first cut in a 64-way partioning of the domain ensures that the first 32 partitions (0–31) are spatially separated from the second 32 (32–63) except for the boundary between the two halves. A similar locality property also exists in some processor networks, such as the hypercube.

The PCG defined in Section 6.1 only reveals the communication pattern. It does not contain any information on the order in which messages could be received. Therefore, asynchronous receives can be posted for all the messages that a processor expects to receive. This would entail providing storage for buffers to receive all the messages. This would also imply that the exchange of information between two processors A and B takes place serially, the first to transfer information

between A and B and the second between B and A. On many parallel computers, such as the Intel iPSC/860, a bidirectional communication facility is provided. If the processors are synchronized, a two-way exchange of information takes place in parallel, thus reducing communication costs. For this to be utilized, the edges in the PCG needs have to be colored. This approach also reduces memory requirements since storage is not required for all the messages that a processor receives; the buffer only needs to be as large as the maximum message length. Thus a schedule of messages is derived. Table 1 presents a schedule of messages for the PCG of Figure 25. It is a coloring of the edges of the PCG. As a result, processors are organized into pairs so that the bidirectional communication can take place between pairs of processors at each stage of the schedule.

Table 1: Communication schedule.

Processor	Permuted iadjproc				
0	1	7	-	-	-
1	0	2	3	4	6
2	3	1	-	-	-
3	2	4	1	-	-
4	5	3	6	1	-
5	4	6	-	-	-
6	7	5	4	-	1
7	6	0	-	-	-

Partitioning, conversion from global to local addresses, and generation of the data structures required for communication at the interpartition boundaries are all done presently on a workstation as a preprocessing step. This is justified when the same geometric case will be run for a variety of analyses, varying freestream Mach number, angle of attack, etc. In adaptive grid situations, where the grid evolves with the flow solution, such an approach requires constant repartitioning and is clearly not viable; procedures such as those outlined in Section 6.6 need to adopted. It is also possible to parallelize the partitioning algorithms e.g., Johan et al. [66] have successfully parallelized the spectral partitioning method.

6.3 Parallelism in explicit schemes

In a vertex-partitioned mesh, each vertex of the triangulation is assigned uniquely to a partition and the interpartition boundaries consist of the edges of the control volumes. In the case of upwind schemes based on projection-evolution techniques [12], two communication phases are required for the evaluation of the residuals, one during the computation of the gradients and the other, during the formation of the fluxes. The processors exchange the dependent variables at two rows of vertices that are incident to the interpartition boundary edges for the computation of the gradients. Next, during the reconstruction phase, gradients are exchanged so that each processor can compute the interpolated variables on each side of the the dual edges forming the interpartition boundaries. If limiters such as the one presented in [12] are employed, another communication step is necessary. Each processor can thus compute the entire residuals for all the vertices it owns. Duplication of the flux calculations occurs at the interpartition boundary edges, but it is not a crucial issue on medium- and coarse-grained parallel computers. As discussed in [135], this duplication can be avoided on fine-grained parallel computers at the expense of more communication. Hammond and Barth [54] when implementing an explicit scheme on a fine-grained parallel computer, assign orientations to the edges of the triangulation so that no vertex has an out-degree greater than 3. Each processor is assigned a vertex and redundant flux calculations are avoided by assigning flux evaluations to the processors containing the outgoing edges. We conclude this subsection by observing that there is ample parallelism in stencil-based operations on fine- and coarse-grained parallel computers with

the associated communication cost increasing with the sizes of the stencils.

6.4 Parallelism in implicit schemes

This section deals with the issues involved in parallelizing implicit schemes on unstructured grids. Ramamurthi et al. [104] have implemented a conjugate gradient method with a linelet-based preconditioner (see Section 4.2.3) and have addressed the issues involved in parallelizing an incompressible flow solver. They settle on a weaker parallel preconditioner that limits the linelets to be contained entirely within the processor. Ajmani et al. [2] have also settled for a weaker preconditioner when solving the Navier-Stokes equations with a GMRES implicit method grids on the Intel iPSC/860. As was discussed in Section 4.2, the preconditioned GMRES method was shown to be quite efficient for solving two-dimensional flow problems using unstructured grids on sequential and vector computers. Given that the sequential algorithm is satisfactory, the techniques to extract the best parallel performance are examined.

On distributed-memory parallel computers, the same least squares problem of Eqn. (24) is solved by each of the processors. While this results in some duplication of work, the main nonlocal kernels of the GMRES are distributed across multiple processors. These kernels include the sparse matrix-vector multiplication, dot products, and L_2 norm evaluations. On a Cray Y-MP, vectorization for the sparse matrix-vector product was achieved by using an edge-oriented data structure for the matrix and coloring the edges of the graph. Coloring of edges destroys locality while allowing for vectorization, and is attractive on a computer such as the Cray Y-MP, because of the fast gather/scatter functions it possesses. However, this is not the optimal way to compute the matrix vector product on a parallel computer with hierarchical memory, where locality is of utmost importance. The compressed-row storage scheme [49], which affords more locality, is used instead. We have found that even on a single node this approach outperforms the one that uses the edge-based data structure by a factor of two, because of the increased locality. Alternatively, the edges and vertices could be reordered such that the new ordering possesses much more locality [83]. The rows are uniquely assigned to processors. The communication step consists of exchange of the vector components at the two rows of vertices incident to the interpartition boundary edges. Akin to the explicit scheme, each processor computes its share of the matrix vector multiplication. More details on the implementations of the matrix-vector product on vector-parallel and distributed-memory computers may be found in [128].

In most problems of interest, the choice of the preconditioner is very important, but the effort involved in applying the preconditioner should not be prohibitive. The implicit scheme without preconditioning possesses almost complete parallelism, except for the duplication of some work when solving the least squares problem in GMRES, and the communication associated with the formation of the residual. On a parallel computer, the parallelism in the preconditioning phase is an important additional consideration. A simple choice is a block-diagonal preconditioner that computes the inverse of the 4×4 diagonal block associated with a mesh point. The LU decomposition of the 4×4 blocks and the forward and back solves are local and, hence, are inherently parallel.

With ILU(0) preconditioning, it is possible to obtain parallelism by using a level scheduling [4]. Under this permutation of the matrix, unknowns within a wavefront can be eliminated simultaneously. However, since the degree of parallelism varies with the wavefront, it cannot be easily exploited on a distributed-memory parallel computer. A fixed partitioning strategy for the mesh incurs substantial load imbalance, while a dynamic partitioning strategy entails substantial data movement and hence, increased communication costs. It has been found in [53, 8] that using a fixed partitioning strategy when solving triangular systems of equations on a regular grid results in low upper bounds on efficiency even in the absence of communication. A higher degree of parallelism in ILU(0) can be achieved by using a different ordering of unknowns, but typically such an ordering adversely affects the convergence of the underlying iterative method. Therefore, for general

sparse matrices, the ILU(0) preconditioner is ill-suited for implementation on a distributed-memory parallel computer.

Therefore, we settle on an ILU preconditioner that is processor-implicit i.e., ILU(0) is carried out for all the vertices internal to a processor. Thus, at a macro-level, the overall preconditioner can be viewed as an approximate block Jacobi iteration, wherein each block is assigned to a processor for which an incomplete LU factorization is carried out. A block here refers to a subdomain consisting of all the unknowns assigned to a processor. In the preconditioning phase, ILU factorization is carried out for each processor by zeroing out the matrix entries whose column numbers lie outside the processor domain. This is equivalent to solving the problem within each processor subject to zero Dirichlet boundary conditions during the preconditioning. This approximation is consistent with the steady state solution $\Delta W \equiv 0$ everywhere. The overall preconditioner is weaker than the global ILU(0), and degenerates to a block-diagonal preconditioner in the limit of one grid point per processor. Thus, as the number of processors increases, degradation in convergence is to be expected. This degradation should be moderate on coarse-grained parallel computers.

In order to minimize the sequential overhead, we appeal to techniques developed in domain decomposition. For an overview of domain decomposition techniques and their suitability to parallel computers see [69]. One of the most successful methods in use in domain decomposition is the Schwarz alternating procedure for overlapping subdomains, which can also be implemented as a preconditioner. Two variants of this procedure have been developed in the literature, the additive and the multiplicative algorithms; see [36]. The term additive denotes that the preconditioning can be carried out independently for each subdomain. The processor-implicit scheme outlined above is an example of an additive Schwarz preconditioner. In contrast, the multiplicative Schwarz method requires that the preconditioner be applied in a sequential way by cycling through the subdomains in some order, as in Gauss-Seidel relaxation. It is possible to extract some coarse-grained parallelism by coloring the subdomains in an additive/multiplicative hybrid, but the potential is limited. Therefore in a parallel context, the additive Schwarz method is preferred.

A powerful idea for elliptic problems advocated in [36], is the use of a coarse grid in order to bring some global influence to bear on the problem, similar in spirit to a two-level multigrid algorithm. The coarse grid operator is applied multiplicatively in our context i.e., the coarse grid problem is solved first. The solution from the coarse grid problem is subsequently used by the processors during the additive (parallel) phase as Dirichlet data at the subdomain boundaries. Applying the coarse grid in this manner does impose a penalty in a parallel setting; it becomes a sequential bottleneck. Additive coarse grid operators are also common [25]. In this reference, the multiplicative and additive Schwarz algorithms are applied to the solution of nonsymmetric elliptic problems. An almost h-independent convergence, where h is the fine grid size, is observed provided the coarse grid is fine enough. In [25] the coarse grid operator was formed by discretizing the partial differential equation on a coarse grid. However, in our application, this would require a triangulation followed by a discretization on this coarse grid. Generation of interpolation operators to transfer information between the coarse and the fine grids would also be necessary. We avoid all these complexities by appealing to an alternative way of obtaining a coarse grid operator described in [139].

A coarse grid Galerkin operator is easily derived from a given fine grid operator by specifying the restriction and prolongation operators. We choose the restriction operator to be a simple summation of fine grid values, and the prolongation operator to be injection. Under this choice, the coarse grid discretization is similar to the one used in an agglomeration multigrid strategy, see [71, 132]. It amounts to identifying all the vertices that belong to a subdomain by one coarse grid vertex, and summing the equations and the right-hand sides associated with them. Thus the coarse grid system has as many vertices as the number of subdomains. At each time step, a coarse grid system is formed and solved by using a direct solver. The data obtained from the coarse grid is used on the boundaries as Dirichlet data for each subdomain. We have found that in practice,

a direct solver is seldom needed to solve the coarse grid system; an iteration of incomplete LU decomposition seems to suffice. The implementation of the coarse grid solver is discussed next. Each processor first forms parts of the coarse grid matrix and the right-hand side at every time step. A global concatenation is performed so that each processor has the entire coarse grid system. This system is solved redundantly by each processor by forming approximate L and U factors. During the preconditioning phase, each processor forms a portion of the right-hand side. After a global concatenation of the right-hand side, each processor carries out forward and backward solves and deduces the appropriate Dirichlet data.

We have found that at least one cycle of implicit smoothing similar to that employed in multigrid context [79] is needed to mitigate the adverse effects of injection of the solution from the coarse to the fine grid. Therefore, on the fine grid, after injection, given the old vector, u^{old}, the following system of implicit equations is solved for the new solution vector, u^{new}:

$$(I + \epsilon d)u_i^{new} = u_i^{old} + \sum_{j=1}^{d} \epsilon u_j^{new}, \tag{75}$$

where ϵ is taken to be 0.5, d is the degree of the vertex i, and the summation is over the neighbors of each vertex. We have found one Jacobi iteration applied to Eqn. (75) to be sufficient. This smoothing step involves communication at the boundaries. We have also developed a weaker smoother that dispenses with the communication associated with the Jacobi smoothing, but yields comparable convergence. This technique termed *modified Jacobi smoothing*, smooths the neighboring coarse grid data (to be used as Dirichlet data) with the data that the processor holds. This step is given by the following relation:

$$(I + \epsilon)U_D^{new} = U_D + \epsilon U_{LOC}, \tag{76}$$

where U_D is the old Dirichlet data, U_D^{new} is the new Dirichlet data and U_{LOC} is the value of the coarse grid vertex assigned to the processor.

6.5 Performance on the Intel iPSC/860

The Intel iPSC/860 is a multiple instruction/multiple data stream (MIMD) parallel computer. The machine used has 128 processor nodes. Each node comprises of a 40 MHz Intel i860 micro-processor, 8 MBytes of memory, and a Direct Connect Module (DCM) which handles communication in the hypercube communication network. Each node has a peak performance of 60 Mflops in 64-bit arithmetic. The bi-directional hypercube interconnect facilitates communication across the nodes.

Flow past a four-element airfoil in a landing configuration at a freestream Mach number $M_\infty = 0.2$ and an angle of attack of 5° is considered as a test case. Performance results are presented for two problem sizes that are representative of two-dimensional inviscid flows. The coarse mesh has 6019 vertices, 17,473 edges, 11,451 triangles, 4 bodies, and 593 boundary edges. The fine mesh has 15,606 vertices, 45,878 edges, 30,269 triangles, 4 bodies, and 949 boundary edges. Figure 32 shows the coarse grid about the four-element airfoil. The Cray implementation of the explicit code [12] runs at 150 megaflops on the Cray Y-MP. The implicit code was not optimized for the Cray Y-MP, since it was developed on the Intel iPSC/860. The result is that it runs in an almost scalar fashion on the Cray, except for the right-hand side computation. However, a similar implicit unstructured mesh Navier-Stokes code was implemented earlier on the Cray Y-MP and optimized [135] to run at approximately 110–120 megaflops. All the megaflop numbers in this section are based on operation counts using the Cray hardware performance monitor. The explicit scheme is a four-stage Runge-Kutta scheme and uses a CFL number of 1.4. With the GMRES/DIAG scheme, the start-up CFL number is 3 and the CFL number is allowed to vary inversely proportional to the L_2 norm of the residual up to a maximum of 30. With GMRES/ILU, the start-up CFL number is 20 and the CFL

number is allowed to vary inversely proportional to the L_2 norm of the residual up to a maximum of 200,000. With both implicit schemes, the number of GMRES search directions is limited to 15. Hence, we use a fixed-storage inexact Newton method [32].

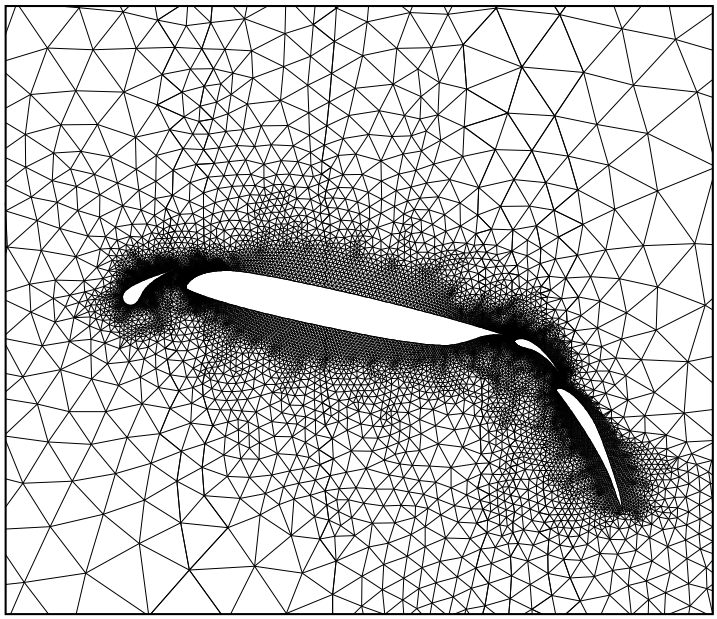

Figure 32: Coarse grid about the four-element airfoil with 6019 vertices.

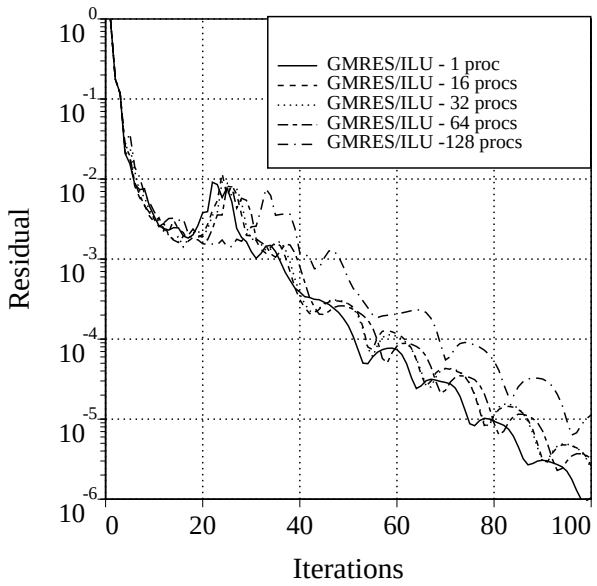

Figure 33: Convergence histories with GMRES/ILU on the fine mesh.

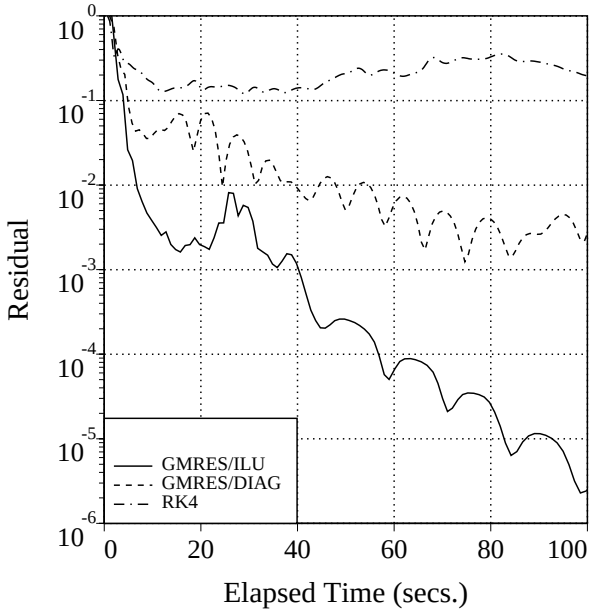

Figure 34: Convergence histories as a function of elapsed times on the fine mesh with 64 processors.

The performances of the explicit and the implicit schemes are compared on the Intel iPSC/860. Tables 2 and 3 show the times per iteration in seconds and the convergence rates for the coarse

and the fine grids, respectively. The convergence rate is defined as

$$Rate = \left(\frac{R_n}{R_1}\right)^{\frac{1}{n-1}}, \tag{77}$$

where R_n is the L_2 norm of the residual of the density equation at the end of nth time step and R_1 is the residual at the end of the first time step. Figure 33 shows the convergence histories for the fine mesh as a function of the number of iterations. It may be observed that the explicit scheme is barely converging while the implicit schemes converge much faster. The GMRES/ILU processor-implicit preconditioning exhibits degradation in convergence as the number of processors increases, but the degradation is moderate. It is also seen that the convergence histories with GMRES/ILU gravitate towards that of GMRES/DIAG as the number of processors increases. In the limit of 1 grid point per processor the two will be identical. Since the problem does not fit on one processor of the Intel iPSC/860, the uni-processor runs were carried out on the Cray Y-MP. Even with 128 processors, the GMRES/ILU scheme requires only about 20% more iterations than the ideal 1 processor scheme to obtain the same level of convergence (5 orders of reduction in the residual norm). Since the time to completion is of ultimate interest, Figure 34 shows the convergence histories as a function of the elapsed times with the number of processors fixed at 64. It clearly shows the superiority of the GMRES/ILU processor-implicit technique over the explicit and the GMRES/DIAG schemes.

Table 2: Performance of the implicit scheme on the Intel iPSC/860 - 6019 vertices.

		No. of processors					
Scheme	Measure	1	4	8	16	32	64
RK4	Time/iter (sec)	-	1.07	0.59	0.32	0.20	0.13
	Conv. rate	0.973	0.973	0.973	0.973	0.973	0.973
GMRES/	Time/iter (sec)	-	3.06	1.66	0.95	0.59	0.42
DIAG	Conv. rate	0.874	0.874	0.874	0.874	0.874	0.874
GMRES/	Time/iter (sec)	-	4.42	2.36	1.32	0.77	0.52
ILU	Conv. rate	0.791	0.795	0.796	0.797	0.797	0.797

Table 3: Performance of the implicit scheme on the Intel iPSC/860 - 15606 vertices.

		No. of processors				
Scheme	Measure	1	16	32	64	128
RK4	Time/iter (sec)	-	0.78	0.43	0.25	0.15
	Conv. rate	0.997	0.997	0.997	0.997	0.997
GMRES/	Time/iter (sec)	-	2.19	1.24	0.75	0.51
DIAG	Conv. rate	0.968	0.968	0.968	0.968	0.968
GMRES/	Time/iter (sec)	-	3.07	1.73	1.07	0.65
ILU	Conv. rate	0.870	0.878	0.878	0.880	0.891

Finally, we examine the effects of using a coarse grid as discussed in Section 6.4 to improve convergence for the 15,606-vertex mesh. Figure 35 shows the convergence histories as a function of iterations for the uni-processor, 32-processor and 128-processor cases, with and without the use of a coarse grid. A cycle of modified Jacobi smoothing is employed as part of the preconditioner in order to stabilize the procedure with the coarse grid system, and the coarse grid system is solved redundantly by all processors using one iteration of incomplete LU factorization. The convergence improves significantly, illustrating the power of the coarse grid; the convergence with 128 processors

is even better than that obtained with the uni-processor scheme. Unfortunately, this improved convergence does not translate into a reduction in the time required to solve the problem, in spite of all the optimizations mentioned in Section 6.4. This is illustrated in Figure 36, which shows the convergence histories as a function of elapsed times on 32 and 128 processors with and without the coarse grid. In both the 32- and the 128-processor cases, it may be observed that the times required to solve the problem are nearly the same with and without the use of the coarse grid. On a per iteration basis, the elapsed times for the 32-processor case are 1.73 and 1.87 seconds respectively, without and with the coarse grid. For the 128-processor case, these times are 0.64 an 1.02 seconds. This points to a major drawback of using the coarse grid system to improve convergence on a parallel computer. With too small a coarse grid system, the effort required to solve the system is minimal, but so is the realized improvement in convergence. With a larger coarse grid system, the gain in convergence is substantial but comes at a greater cost. The coarse grid operator, being sequential in nature, predominates as the number of processors increases. When invoking the coarse grid operator, a large fraction of the time is spent in the global concatenation. Thus, if the parallel computer were to have better communication rates, the technique would be more competitive in terms of elapsed times as well.

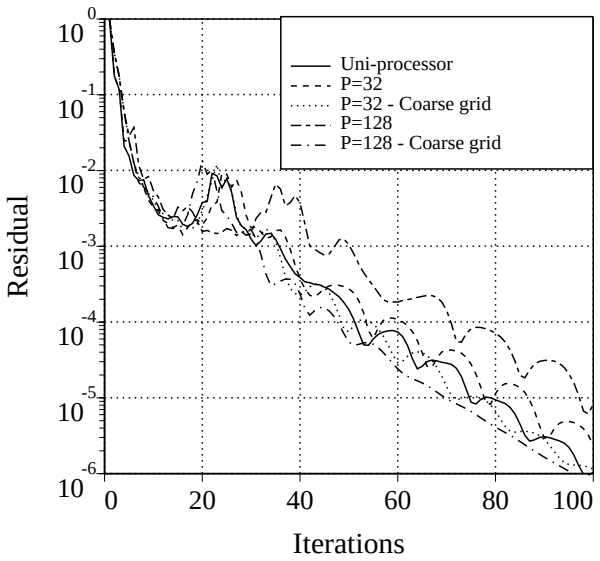

Figure 35: Convergence histories for the 15606-vertex case as a function of iterations with and without the use of a coarse grid.

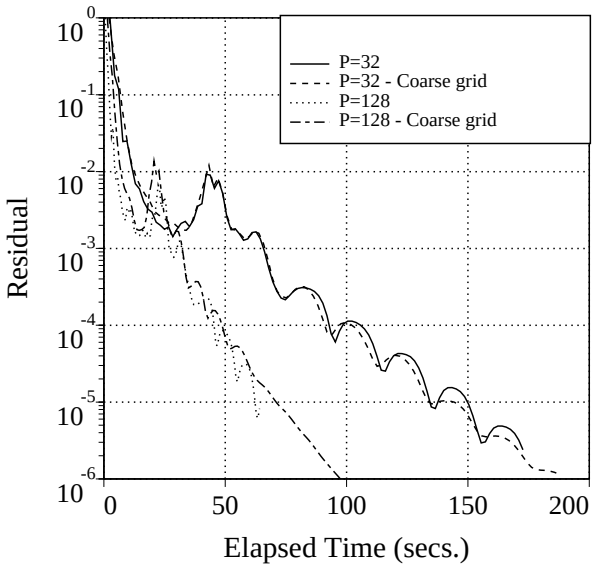

Figure 36: Convergence histories 15606-vertex case as a function of elapsed times with and without the use of a coarse grid.

In order to get an idea of the relative performances of the codes on the Intel iPSC/860 and the Cray Y-MP/1, performance data from the Cray implementation are given. The elapsed times on the Cray Y-MP/1 are respectively, 0.15 and 0.39 seconds per time step for the coarse and fine meshes with the explicit scheme. The explicit code runs at 150 megaflops on the Cray Y-MP/1. The megaflop ratings on the Cray are obtained using the hardware performance monitor. By simple scaling it may be verified that the explicit code runs at nearly 400 megaflops on 128 processors of the Intel iPSC/860 with the larger problem. Timings for the implicit scheme on the Cray Y-MP/1 are not provided since the codes have not been not optimized.

6.6 Adaptive grids

R-refinement, where the grid points are simply repositioned so that regions of importance are better resolved, poses no problems in parallel. P-refinement causes load imbalance in a parallel setting. H-refinement or mesh-enrichment also results in load imbalance. We will be mainly concerned in

this section about redressing the load imbalance caused by h-refinement, with a similar approach being possible for p-refinement.

In the case of steady flows, a global repartitioning using any of the partitioning strategies outlined in Section 6.1 is attractive, because the adaptation is typically only carried out a few times. In the case of unsteady flows, however, the adaptation is usually carried out much more frequently and global repartitioning, even if done in parallel, is time-consuming. Instead, a dynamic load balancing strategy which involves local migration of cells from each processor to/from the neighboring processors is more appealing. The quality of the partitions that result from such a local procedure is dependent on the quality of the initial partitions and also on the number of times the local procedure is carried out. Periodically, we may have no choice but to use a global repartitioning strategy if the quality of the partitions degrades. A tacit assumption is made when dealing with adaptive grids on distributed-memory parallel computers. The assumption is that when the subgrid assigned to a node is refined, it still fits in the memory of that node before the load balancing algorithm is initiated. This may be an unrealistic assumption unless the memory of that node is sizable; it may also limit the amount of refinement that can be done.

A dynamic load balancing strategy presented in [137] for adapted tetrahedral grids is now described. The technique migrates tetrahedral cells between processors so that balanced partitions result. Given an initial partitioning, that becomes unbalanced because of adaptive refinement, the load balancing algorithm consists of two steps. The first step, or the higher-level algorithm, concerns the identification of the processors that need to exchange cells with their face-adjacent neighbors and the specification of the number of cells to be exchanged. The second phase, or the lower-level algorithm, concerns the actual modus operandi of the exchange of cells between any two face-adjacent processors, including the updating of the pertinent data structures.

The higher-level algorithm used for load balancing is a divide-and-conquer strategy. The global problem involving all the processors is split into two similar, independent problems, each of which involves half the number of processors. The two problems are recursively solved in the same fashion, with the recursion terminating when the problem involves only two processors. Thus the algorithm completes in $\log_2 P$ stages where P is the total number of processors. At each stage, processors in each group that are face-adjacent to at least one processor in the other group are identified for the actual migration of cells and are called *candidate processors*. Since a recursive partitioning strategy is adopted (see Section 6.1), the groups of processors at each stage are easily identified by their binary addresses. This will be clarified by means of an example. Figure 37 shows an 8-way partition of the domain. The loads associated with the partitions are shown in parantheses. Assume that a naive mapping of processors is done, so that partition 0 is assigned to processor 0, and so on. At the first stage, the processors are split into two groups, one with 0 in the leading bit of the binary addresses (processors 0, 1, 2, 3) and the other with 1 (processors 4, 5, 6, 7). The candidate senders are processors 0, 1 and 3 and the candidate receivers are 7, 6 and 4 at the first stage. The cumulative loads in these two processor groups are balanced by migrating a total load of 50 from the heavier group using the candidate processors. The next step will involve two problems, one that exchanges data between processor groups (0, 1) and (2, 3), using candidate processors 1, 2 and 3 and a second, that exchanges data between processor groups (4, 5) and (7, 6), using candidate processors 6, 4 and 5. The last stage consists of exchange of information between processors 0 and 1, 2 and 3, 4 and 5, and 7 and 8. At each stage of the load balancing, migration takes place only across face-adjacent neighbors. Because of the recursive initial partitioning, such a scheme would never have to migrate cells between processors that are not face-adjacent. In our applications, the number of cells migrated by each candidate by each candidate sender is given by the following relation:

$$Mig_i = \frac{N_i}{N_{tot}} * Mig_{tot}, \tag{78}$$

where Mig_i is the number of cells to be migrated by the ith candidate processor in the sender

group, N_i is the total number of cells on processor i, Mig_{tot} is the total number of cells to be migrated, and N_{tot} is the cumulative number of cells in all the candidate processors of the sender group. This algorithm will fail when $Mig_i > N_i$. This can happen if, for example, a processor such as processor 2 in Figure 37, has an excess load, but is not a candidate processor in the early stages of the algorithm. However, such situations can be remedied by initially balancing the load of such a processor with its face-adjacent neighbors before applying the divide-and-conquer algorithm.

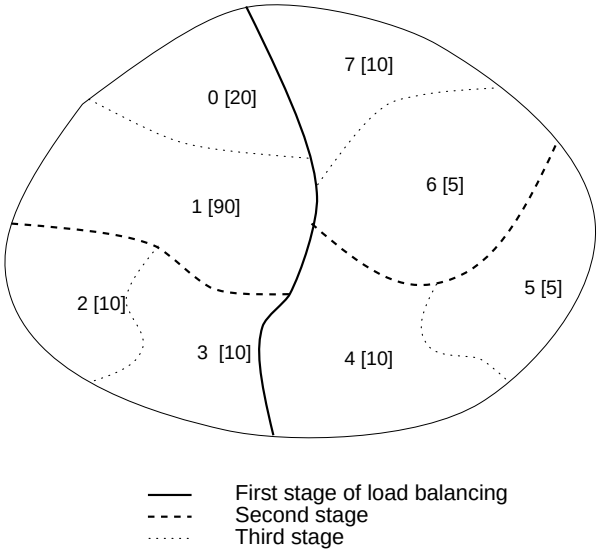

Figure 37: 8-way decomposition with the computational loads shown in parantheses.

After the determination of candidate senders and receivers at each step of the divide and conquer algorithm, the second step consists of the actual migration of cells. The candidate processors in the sender group send the number of cells computed using Eqn. (78) to the receivers. The candidate processors in the receiver group likewise receive the information from the senders. Asynchronous communication models are utilized for this purpose. The candidate processors also have to adjust their inter-partition data structures so as to accurately reflect the change in the assignment of cells to partitions. The local migration procedure is split into two steps:

1. The cell-designation step: The sender processor determines which cells are to be sent. To minimize start-up costs, the sender sends the entire group of cells to the receiver in one message. The cells could be designated by using either a *connectivity-based* or a *coordinate-based* strategy. In the connectivity-based strategy, starting with the cells on the sender that are face-adjacent to the receiver, additional cells are added to the list if they share a face with any of the cells already in the list. In the coordinate-based strategy, cells within a spatial region are designated for transfer. In [137], it was found that the coordinate-based designation scheme was found to produce smooth inter-partition boundaries, whereas the connectivity-based strategy produced jagged boundaries. It should be mentioned that the coordinate-based cell-designation is appropriate only when a coordinate bisection strategy is employed for the initial grid partitioning.

2. Communication step: The sender processor deletes the designated cells and the appropriate faces, edges and nodes from its representation. It sends the information to the receiver processors. Each of the receivers adds the elements to its representation. Then both the sender and receiver update their inter-partition boundary data structures.

An example of the load balancing algorithm taken from [137] for an adapted grid for flow about an ONERA M6 wing is shown in Figure 39(a) and (b). The surface grid has been adapted once near the leading edge, the tip and the shock waves. The initial partitioning using a coordinate bisection

strategy is shown in Figure 39(a) by the thick lines. The surface plot for the balanced grid using the divide and conquer strategy with the coordinate-based cell designation scheme is shown in Figure 39(b). During the first step of the load balancing algorithm the vertical partition boundary moves to the left to balance the number of cells in the two halves. This is followed by an independent movement of each of the horizontal boundaries. The length of the inter-partition boundaries has not changed appreciably due to the balancing. This is contrast to the case where connectivity-based cell-designation is employed where it was found that the inter-partition boundaries became jagged and poorly formed [137]. The load balancing was implemented on an iPSC/860 and was found to have adequate parallelism.

Figure 39(a): Adapted grid with the initial partitioning.

Figure 39(b): Partitions that result from applying the load balancing algorithm.

References

[1] *Special course on unstructured grid methods for advection dominated flows*, vol. AGARD Report 787, AGARD, 1992.

[2] K. Ajmani, M. S. Liou, and R. W. Dyson, *Preconditioned implicit solvers for the Navier-Stokes equations on distributed-memory machines.* AIAA Paper 94-0408, Jan. 1994.

[3] J. Alonso and A. Jameson, *Fully-implicit time-marching aeroelastic solution.* AIAA Paper 94-0056, Jan. 1994.

[4] E. Anderson and Y. Saad, *Solving sparse triangular systems on parallel computers*, Int. J. High Speed Computing, 1 (1989), pp. 73–96.

[5] W. K. Anderson, *A grid generation and flow solution method for the Euler equations on unstructured grids*, J. Comp. Phys., 110 (1994), pp. 23–38.

[6] A. Arnone, M.-S. Liou, and L. A. Povinelli, *Multigrid time-accurate integration of Navier-Stokes equations.* AIAA Paper 93-3361CP, 1993.

[7] S. T. Barnard and H. D. Simon, *A fast multi-level implementation of recursive spectral bisection for partitioning usntructured grid problems*, Concurrency: Practice and Experrience, 6 (1994), pp. 101–117.

[8] E. Barszcz, R. Fatoohi, V. Venkatakrishnan, and S. Weeratunga, *Triangular systems for CFD applications on parallel architectures*, Tech. Rep. NAS Applied Research Branch RNR, NASA Ames Research Center, Apr. 1993.

[9] T. J. Barth, *Numerical aspects of computing viscous high Reynolds number flows on unstructured meshes.* AIAA Paper 91-0721, Jan. 1991.

[10] ——, *Numerical aspects of computing viscous high Reynolds number flows on unstructured meshes.* AIAA Paper 91-0721, Jan. 1991.

[11] T. J. Barth and P. O. Fredrickson, *Higher order solution of the Euler equations on unstructured grids using quadratic reconstruction.* AIAA Paper 90-0013, Jan. 1990.

[12] T. J. Barth and D. C. Jespersen, *The design and application of upwind schemes on unstructured meshes.* AIAA Paper 89-0366, Jan. 1989.

[13] T. J. Barth and S. W. Linton, *An unstructured mesh Newton solver for compressible fluid flow and its parallel implementation.* AIAA Paper 95-0221, Jan. 1995.

[14] J. T. Batina, *Unsteady Euler airfoil solutions using unstructured dynamic meshes*, AIAA J., 28 (1990), pp. 1381–1388.

[15] ——, *Implicit upwind solution algorithms for three-dimensional unstructured meshes*, AIAA J., 31 (1993), pp. 801–805.

[16] J. D. Baum, H. Luo, and R. Löhner, *A new ALE adaptive unstructured methodology for the simulation of moving bodies.* AIAA Paper 94-0414, Jan. 1994.

[17] R. M. Beam and H. E. Bailey, *Newton's method solver for the axisymmetric Navier-Stokes equations*, J. Comp. Phys., 30 (1990), pp. 1507–1514.

[18] R. M. Beam and H. S. Bailey, *Viscous computations using a direct solver*, Computers and Fluids, 18 (1990), pp. 191–204.

[19] R. M. Beam and R. F. Warming, *An implicit finite-difference algorithm for hyperbolic systems in conservation laws*, J. Comp. Phys., 22 (1976), pp. 87–110.

[20] E. E. Bender and P. K. Khosla, *Solution of the two-dimensional Navier-Stokes equations using sparse matrix solvers.* AIAA Paper 87-0603, Jan. 1987.

[21] S. Bokhari, T. W. Crockett, and D. M. Nichol, *Parametric nested dissection.* ICASE Report 94-39, 1994.

[22] A. Brennis and A. Eberle, *Application of an implicit relaxation method solving the Euler equations for time-accurate unsteady problems*, Transactions of ASME - J. Fluids Engrg., 112 (1990), pp. 510–520.

[23] W. R. Briley and H. McDonald, *Solution of the multi-dimensional compressible Navier-Stokes equations by a generalized implicit method*, J. Comp. Phys., 21 (1977), pp. 372–397.

[24] P. N. Brown and Y. Saad, *Hybrid krylov methods for nonlinear systems of equations*, SIAM J. Sci. Stat. Comput., 11 (1990), pp. 450–481.

[25] X. Cai, W. D. Gropp, and D. E. Keyes, *A comparison of some domain decomposition and ILU preconditioned algorithms for nonsymmetric elliptic problems*, Num. Linear. Alg. with Applications, 19 (1994), pp. 477–504.

[26] X. C. Cai, W. D. Gropp, D. E. Keyes, and M. D. Tidriri, *Newton-Krylov-Schwarz methods in CFD*, in Proceedings of the International Workshop on Numerical Methods for the Navier-Stokes Equations, vol. 47 of Notes in Numerical Fluid Mechanics, Braunschweig/Wiesbaden, 1994, Vieweg Verlag.

[27] S. Chakravarthy, K. Y. Szema, S. Ramakrishnan, R. Burman, and R. Schultz, *Inviscid CFD for store separation using unified boundary conditions.* AIAA Paper 93-3404, Aug. 1993.

[28] B. Cockburn, S. Hou, and C. Shu, *TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws IV: the multidimensional case*, Math. Comp., 54 (1990), pp. 545–581.

[29] G. Dahlquist and A. Björck, *Numerical methods*, Prentice Hall, New Jersey, 1974.

[30] G. A. Davis and O. O. Bendiksen, *Unsteady transonic two-dimensional Euler solutions using finite elements*, AIAA J., 31 (1993), pp. 1051–1059.

[31] H. Deconinck, H. Paillere, R. Struijs, and P. L. Roe, *Multidimensional upwind schemes based on fluctuation-splitting of conservation laws*, Comp. Mechanics, 11 (1993), pp. 323–340.

[32] R. S. Dembo, S. C. Eisenstadt, and T. Steihaug, *Inexact Newton methods*, SIAM J. Numer. Anal, 19 (1982), pp. 400–408.

[33] J. E. Dennis and R. B. Schnabel, *Numerical methods for unconstrained optimization and nonlinear equations*, Prentice–Hall, Englewood Cliffs, NJ, 1983.

[34] H. A. V. der Vorst, *Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetic linear systems*, SIAM J. Sci. Stat. Comput., 13 (1992), pp. 631–644.

[35] J. Donea, *A Taylor-Galerkin method for convective transport problems*, Int. J. for Numer. Meth. in Engrg., 20 (1984), pp. 101–119.

[36] M. Dryja and O. B. Widlund, *Towards a unified theory of domain decomposition algorithms for elliptic problems*, in Third International Symposium on Domain Decomposition Methods for Partial Differential Equations, T. Chan, R. Glowinski, J. Periux, and O. B. Widlund, eds., Philadelphia, PA, 1990, SIAM, pp. 3–21.

[37] I. S. Duff and G. A. Meurant, *The effect of ordering on preconditioned conjugate gradients*, BIT, 129 (1989), pp. 635–657.

[38] L. J. Durlofsky, B. Engquist, and S. Osher, *Triangle-based adaptive stencils for the solution of hyperbolic conservation laws*, J. Comp. Phys., 98 (1992), pp. 64–73.

[39] L. Dutto, *The effect of ordering on preconditioned GMRES algorithm, for solving the compressible Navier-Stokes equations*, Intl. J. for Numer. Meth. in Engrg., 36 (1993), pp. 457–497.

[40] J. W. Edwards and J. L. Thomas, *Computational methods for unsteady transonic flows.* AIAA Paper 87-0107, Jan. 1987.

[41] C. Farhat, L. Fezoui, and S. Lanteri, *Two-dimensional viscous flow computation on the Connection Machine: Unstructured meshes, upwind schemes and parallel computation*, Comput. Methods Appl. Mech. Engrg, 102 (1993), pp. 61–88.

[42] C. Farhat and M. Lesoinne, *Automatic partitioning of unstructured meshes for the parallel solution of problems in computational mechanics*, Intl. J. for Numer. Meth. in Engrg., 36 (1993), pp. 745–764.

[43] L. Fezoui and B. Stoufflet, *A class of implicit upwind schemes for Euler simulations with unstructured meshes*, J. Comp. Phys., 84 (1989), pp. 174–206.

[44] L. Formaggia, J. Peraire, and K. Morgan, *Simulation of a store separation using finite element method*, Appl. Math. Modelling, 12 (1988), pp. 175–181.

[45] R. W. Freund, G. H. Golub, and N. M. Nachtigal, *Iterative solutions of linear systems*, Acta Numerica, 1 (1992), pp. 57–100.

[46] N. T. Frink, *A time-accurate multiple-grid algorithm.* AIAA Paper 85-1493CP, June 1985.

[47] ——, *Grid restructuring for moving boundaries.* AIAA Paper 91-1589CP, July 1991.

[48] ——, *Recent progress toward a three-dimensional unstructured Navier-Stokes flow solver.* AIAA Paper 94-0061, Jan. 1994.

[49] A. George and J. W. Liu, *Computer solution of large sparse positive definite systems*, Prentice Hall Series in Computational Mathematics, Englewood Cliffs, N. J., 1981.

[50] A. Gibbons, *Algorithmic graph theory*, Cambridge University Press, New York, NY, 1985.

[51] J. R. Gilbert, G. L. Miller, and S.-H. Teng, *Geometric mesh partitioning: Implementation and experiments*, Tech. Rep. CSL–94–13, Xerox Palo Alto Research Center, 1994.

[52] G. H. Golub and C. F. Van Loan, *Matrix computations*, Johns Hopkins Uniersity Press, Baltimore, MD, 1989.

[53] A. Greenbaum, *Solving sparse triangular linear systems using FORTRAN with parallel extensions on the NYU Ultracomputer prototype*, Tech. Rep. Ultracomputer Note 9, New York Univ., Apr. 1986.

[54] S. W. Hammond and T. J. Barth, *Efficinet massively parallel Euler solver for two-dimensional unstructured grids*, AIAA J., 30 (1992), pp. 947–952.

[55] A. Harten and S. Chakravarthy, *Multi-dimensional ENO schemes for general geometries.* ICASE Report No. 91-76, 1991; submitted to J. Comp. Phys.

[56] J. E. Hase, D. A. Anderson, and I. H. Parpia, *A Delaunay triangulation method and Euler solver for bodies in relative motion.* AIAA Paper 91-1590CP, July 1991.

[57] O. Hassan, K. Morgan, and J. Peraire, *An adaptive implicit/explicit finite element scheme for compressible high speed flows.* AIAA Paper 89-0363, Jan. 1989.

[58] J. Hong, K. Melhorn, and A. L. Rosenberg, *Cost trade-offs in graph embeddings, with applications*, JACM, 30 (1983), pp. 709–728.

[59] T. J. R. Hughes, *Recent progress in the development and understanding of SUPG methods with special reference to the compressible Euler and Navier-Stokes equations*, Intl. J. for Numer. Meth. in Fluids, 7 (1987), pp. 1261–1275.

[60] T. J. R. Hughes, L. P. Franca, and G. M. Hulbert, *A new finite element formulation for computational fluid dynamics: VIII The Galerkin/least-squares method for advective-diffusive systems*, Comput. Methods Appl. Mech. Engrg, 73 (1989), pp. 173–189.

[61] A. Jameson, *Solution of the Euler equations by a multigrid method*, Applied Mathematics and Computation, 13 (1983), pp. 327–356.

[62] ——, *Time-dependent calculations using multigrid, with applications to unsteady flows past airfoils and wings*. AIAA Paper 91-1596, July 1991.

[63] ——, *Analysis and design of numerical schemes for gas dynamics 1. Artificial diffusion, upwind biasing, limiters and their effect on accuracy and multigrid convergence*. Submitted to Intl. J. Comp. Fluid Dynamics, 1994.

[64] A. Jameson and D. J. Mavriplis, *Finite volume solution of the two-dimensional Euler equations on a regular triangular mesh*, AIAA J., 24 (1986), pp. 611–618.

[65] A. Jameson, W. Schmidt, and E. Turkel, *Numerical solution of the Euler equations by finite volume methods using Runge-Kutta time stepping schemes*. AIAA Paper 81-1259, 1981.

[66] Z. Johan, T. J. R. Hughes, K. K. Mathur, and S. L. Johnsson, *A data parallel finite element method for computational fluid dynamics on the Connection Machine System*, Comput. Methods Appl. Mech. Engrg., 99 (1992), pp. 113–124.

[67] Y. Kallinderis and S. Ward, *Hybrid prismatic/tetrahedral grid generation for complex geometries*. AIAA Paper 93-0669, Jan. 1993.

[68] S. R. Kennon, J. M. Meyering, C. W. Berry, and J. T. Oden, *Geometry based Delaunay tetrahedralization and mesh movement strategies for multi-body CFD*. AIAA Paper 92-4575, Aug. 1992.

[69] D. E. Keyes, *Domain decomposition: a bridge between nature and parallel computers*, in Adaptive, multilevel and hierarchical computational strategies, A. K. Noor, ed., New York, 1991, ASME, pp. 293–334.

[70] W. L. Kleb, J. T. Batina, and M. H. Williams, *Temporal adaptive Euler/Navier-Stokes algorithm involving unstructured dynamic meshes*, AIAA J., 30 (1992), pp. 1980–1985.

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

[72] A. Lerat, J. Sides, and V. Daru, *Efficient computation of steady and unsteady transonic flows by an implicit solver*, in Advances in computational transonics, W. G. Habashi, ed., vol. 4 of Recent advances in Numerical Methods in Fluids, Swansea, U.K., 1985, Pineridge Press, pp. 545–576.

[73] P. Lesaint and P. A. Raviart, *On a finite element method for solving the neutron transport problem*, in Mathematical aspects of Finite Elements in Partial Differential Equations, C. de Boor, ed., Academic Press, 1974, pp. 89–123.

[74] R. Löhner, *An adaptive finite element solver for transient problems with moving bodies*, Comp. Struct., 30 (1988), pp. 303–317.

[75] R. Löhner and J. D. Baum, *Adaptive H-refinement on 3-D unstructured grids for transient problems*, Int. J. Num. Meth. Fluids, 14 (1992), pp. 1407–1419.

[76] R. Löhner, J. Cambreros, and M. Merriam, *Parallel unstructured grid generation*, Intl. J. Num. Meth. Engrg., 54 (1992), pp. 545–581.

[77] D. Martin and Löhner, *An implicit linelt-based solver for incompressible flows.* AIAA Paper 93-0668, Jan. 1992.

[78] D. J. Mavriplis, *Multigrid solution of the two-dimensional Euler equations on unstructured triangular meshes*, AIAA J., 26 (1988), pp. 824–831.

[79] ——, *Accurate multigrid solution of the Euler equations on unstructured and adaptive meshes*, AIAA J., 28 (1990), pp. 213–221.

[80] ——, *Algebraic turbulence modeling for unstructured and adaptive meshes*, AIAA J., 29 (1991), pp. 2086–2093.

[81] ——, *Three-dimensional multigrid for the Euler equations*, AIAA Journal, 30 (1992), pp. 1753–1761.

[82] ——, *Mesh generation and adaptivity for complex geometries*, Handbook of Computational Fluid Mechanics, Academic Press, Inc., San Diego, CA, 1995.

[83] D. J. Mavriplis, R. Das, J. Saltz, and R. E. Vermeland, *Implementation of a parallel unstructured Euler solver on shared and distributed memory machines.* ICASE Report No. 92-68, 1992; to appear in The J. of Supercomputing.

[84] D. J. Mavriplis and V. Venkatakrishnan, *Agglomeration multigrid for viscous turbulent flows.* AIAA Paper 94-2332, June 1994; to appear in Computers and Fluids.

[85] ——, *A 3D agglomeration multigrid solver for the Reynolds-averaged Navier-Stokes equations on unstructured meshes.* AIAA Paper 95-0345, Jan. 1995.

[86] P. R. McHugh and D. A. Knoll, *Comparison of standard and matrix-free implementations of several Newton-Krylov solvers*, AIAA J., 32 (1994), pp. 2394–2400.

[87] J. L. Meijering, *Interface area, edge length, and number of vertices in crystal aggregates with random nucleation*, Tech. Rep. 8, Philips Res. Rep, 1953.

[88] J. A. Meijerink and H. A. van der Vorst, *Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems*, J. Comp. Phys., 44 (1981), pp. 134–155.

[89] N. D. Melson, M. D. Sanetrik, and H. L. Atkins, *Time-accurate Navier-Stokes calculations with multigrid acceleration*, in 6th Copper Mountain Conf. on Multigrid Methods, 1993, pp. 423–439.

[90] M. Merriam, *Efficient parallel implementation of an algorithm for Delaunay triangulation.* Paper submitted to the J. Comp. Phys., June 1993.

[91] G. L. Miller, S.-H. Teng, W. Thurston, and S. A. Vavasis, *Automatic mesh partitioning*, in Graph Theory and Sparse Matrix Computation, A. George, J. R. Gilbert, and J. W. H. Liu, eds., Springer Verlag, 1993, pp. 57–84. The IMA Volumes in Mathematics and its Applications, Volume 56.

[92] W. A. Mulder and B. van Leer, *Implicit upwind methods for the Euler equations.* AIAA Paper 83-1930, July 1983.

[93] K. Nakahashi, *A finite-element method on prismatic elements for the three-dimensional Navier-Stokes equations*, in Lecture Notes in Physics, vol. 323, Springer Verlag, 1989.

[94] E. J. Nielsen, R. W. Walters, W. K. Anderson, and D. E. Keyes, *Application of Newton-Krylov methodology to a three-dimensional Euler code.* Paper submitted to the 12th AIAA CFD Conf., San Diego, CA., June 1995.

[95] P. D. Orkwis and D. S. McRae, *Newton's method solver for the axisymmetric Navier-Stokes equations*, AIAA J., 30 (1992), pp. 1507–1514.

[96] H. Paillere, H. Deconinck, R. Struijs, P. L. Roe, L. M. Mesaros, and J. D. Muller, *Computations of compressible flows using fluctuation-splitting on triangular meshes.* AIAA Paper 93-3301CP, July 1993.

[97] B. Palmerio, *An attraction-repulsion mesh adaption model for flow solution on unstructured grids*, Computers and Fluids, 23 (1994), pp. 487–506.

[98] M. Paraschivoiu, J. Y. Trepanier, M. Reggio, and R. Camarero, *A conservative dynamic discontinuity tracking algorithm for the Euler equations.* AIAA Paper 94-0081, Jan. 1994.

[99] P. Parikh, R. Löhner, C. Gumbert, and S. Pirzadeh, *Numerical solutions on a PATHFINDER and other configura tions using unstructured grids and a finite element solver.* AIAA Paper 89-0362, Jan. 1989.

[100] I. Parpia and P. Parikh, *A solution-adaptive mesh generation method with cell-face orientation control.* AIAA Paper 94-0416, Jan. 1994.

[101] A. Pothen, H. D. Simon, and K. P. Liou, *Partitioning sparse matrices with eigenvectors of graphs*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 430–452.

[102] T. H. Pulliam, *Time accuracy and the use of implicit schemes.* AIAA Paper 93-3360CP, July 1993.

[103] M. M. Rai, *Navier-Stokes simulations of blade-vortex interaction using high-order accurate upwind schemes.* AIAA Paper 87-0543, Jan. 1987.

[104] R. Ramamurthi, W. Sandberg, and R. Löhner, *Evaluation of a scalable 3-D finite element incompressible flow solver.* AIAA Paper 94-0756, Jan. 1994.

[105] R. D. Rausch, J. T. Batina, and H. T. Y. Yang, *Spatial adaptation of unstructured meshes for unsteady aerodynamic flow computations*, AIAA J., 30 (1992), pp. 1243–1251.

[106] P. L. Roe, *Characteristic-based schemes for the Euler equations*, Ann. Rev. Fluid Mech., 18 (1986), pp. 337–365.

[107] P. Rostand and B. Stoufflet, *TVD schemes to compute compressible viscous flows on unstructured meshes*, in Notes on numerical fluid mechanics, vol. 24, Braunschweig, Germany, 1988, Vieweg Press.

[108] Y. Saad, *ILUT : A dual threshold incomplete factorization*, Tech. Rep. UMSI 92/38, University of Minnesota Supercomputing Institute, Mar. 1992.

[109] Y. Saad and M. H. Schultz, *Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 856–869.

[110] F. Shakib, T. J. R. Hughes, and Z. Johan, *A multi-element group preconditioned gmres algorithm for nonsymmetric problems arising in finite element analysis*, Comput. Methods Appl. Mech. Engrg, 87 (1989), pp. 415–456.

[111] C.-W. Shu and S. Osher, *Efficient implementation of Essentially Non-oscillatory shock-capturing schemes*, J. Comp. Phys., 77 (1988), pp. 439–471.

[112] H. D. Simon, *Partitioning of unstructured problems for parallel processing*, Computing Systems in Engrg., 2 (1991), pp. 135–148.

[113] D. C. Slack, D. L. Whitaker, and R. W. Walters, *Time integration algorithms for the two-dimensional Euler equations on unstructured meshes*, AIAA J., 32 (1994), pp. 1158–1166.

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

[115] G. Strang, *Introduction to Applied Mathematics*, Wellesley-Cambridge Press, Wellesley, MA, 1986.

[116] G. Strang and G. J. Fix, *An analysis of the finite element method*, Prentice Hall, New Jersey, 1973.

[117] P. D. Thomas and C. K. Lombard, *Geometric conservation law and its applications to flow computations on moving grids*, AIAA J., 17 (1979), pp. 1030–1037.

[118] J. F. Thompson and N. P. Weatherill, *Aspects of numerical grid generation: current science and art.* AIAA-93-3539-CP, 1993.

[119] M. D. Tidriri, *Couplage d'approximations et de modèles de types différents dans le calcul d'ècoulements externes*, PhD thesis, Université de Paris IX, May 1992.

[120] J. Y. Trepanier, H. Zhang, M. Reggio, and R. Camarero, *Adaptive and moving meshes for the computation of unsteady compressible flows*, in Numerical grid generation in Computational Fluid dynamics and related fields, A. S. Arcilla, J. Hauser, P. R. Eiseman, and J. F. Thompson, eds., New York, 1991, North-Holland, pp. 43–54.

[121] B. van Leer, *Towards the ultimate conservative difference scheme V. A second order sequel to Godunov's method*, J. Comp. Phys., 32 (1979), pp. 101–136.

[122] B. van Leer, W. T. Lee, and P. Roe, *Characteristic time-stepping or local preconditioning of the Euler equations.* AIAA Paper 91-1552CP, June 1991.

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

[124] J. van Rosendale, *Floating point shock fitting for the Euler equations.* Paper submitted to the 12th AIAA CFD Conf., San Diego, CA., June 1995.

[125] J. C. Vassberg, *A fast, implicit unstructured-mesh Euler method.* AIAA Paper 92-2693, 1992.

[126] V. Venkatakrishnan, *Newton solution of inviscid and viscous problems*, AIAA J., 27 (1989), pp. 885–891.

[127] ——, *Preconditioned conjugate gradient methods for the compressible Navier-Stokes equations*, AIAA J., 29 (1991), pp. 1092–1100.

[128] ——, *Parallel computation of Ax and $A^T x$*, Int. J. High Speed Computing, 6 (1994), pp. 325–342.

[129] ——, *Parallel implicit unstructured grid Euler solvers*, AIAA J., 32 (1994), pp. 1985–1991.

[130] V. Venkatakrishnan and T. J. Barth, *Application of direct solvers to unstructured meshes for the Euler and Navier-Stokes equations using upwind schemes.* AIAA Paper 89-0364, Jan. 1989.

[131] V. Venkatakrishnan and A. Jameson, *Computation of unsteady transonic flows by the solution of Euler equations*, AIAA J., 26 (1988), pp. 974–981.

[132] V. Venkatakrishnan and D. J. Mavriplis, *Agglomeration multigrid for the three-dimensional Euler equations.* AIAA Paper 94-0069, June 1994; to appear in AIAA J.

[133] ——, *Implicit solvers for unstructured meshes*, Journal of Computational Physics, 105 (1993), pp. 83–91.

[134] ——, *Implicit method for the computation of unsteady flows on unstructured grids.* Paper submitted to the 12th AIAA CFD Conf., San Diego, CA., June 1995.

[135] V. Venkatakrishnan, H. D. Simon, and T. J. Barth, *A mimd implementation of a parallel Euler solver for unstructured grids*, The J. of Supercomputing, 6 (1992), pp. 117–137.

[136] R. Vichnevetsky and J. B. Bowles, *Fourier Analysis of Numerical Approximations of Hyperbolic Equations*, SIAM Studies in Applied Mathematics, SIAM, Philadelphia, PA, 1982.

[137] A. Vidwans, Y. Kallinderis, and V. Venkatakrishnan, *Parallel dynamic load-balancing algorithm for three-dimensional adaptive unstructured grids*, AIAA J., 32 (1994), pp. 497–506.

[138] H. F. Walker, *Implementation of the GMRES method using Householder transformations*, SIAM J. Sci. Stat. Comput., 9 (1988), pp. 152–163.

[139] P. Wesseling, *An introduction to multigrid methods*, John Wiley & Sons, New York, 1992.

[140] D. L. Whitaker, *Three-dimensional unstructured grid Euler computations using a fully-implicit, upwind method.* AIAA Paper 93-3337CP, July 1993.

[141] L. B. Wigton, N. J. Yu, and D. P. Young, *GMRES acceleration fo fluid dynamics codes.* AIAA Paper 85-1494CP, July 1985.

[142] X. Xia and R. Nicolaides, *Covolume techniques for anisotropic media*, Numer. Math., 61 (1992), pp. 215–234.

[143] H. Zhang, M. Reggio, J. Y. Trepanier, and R. Camarero, *Discrete form of the GCL for moving meshes and its implementation in CFD schemes*, Computers and Fluids, 22 (1993), pp. 9–23.

[144] Z. Zlatev, *Use of iterative refinement in the solution of sparse linear systems*, SIAM J. Numer. Anal., 19 (1982), pp. 381–399.