THE EFFICIENT USE OF VECTORIZED DIRECT SOLVERS IN COMPUTATIONAL FLUID DYNAMICS

by

David W. Riggins

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

DOCTOR IN PHILOSOPHY

in

Aerospace Engineering

APPROVED

R.W. Walters, Chairman

J.A. Schetz

B. Grossman

W.L. Neu

D. Walker

May, 1988

Blacksburg, Virginia

ACKNOWLEDGEMENTS

I am deeply indebted to my committee chairman and mentor, Dr. Robert Walters, for his patient instruction and guidance in all phases of this work. I would like to express special appreciation to Dr. B. Grossman for kindling my interest in fluid dynamics several years ago. My subsequent work and continuing interest in this area owes much to that inspiration. Thanks are due as well to Dr. Joseph Schetz, Dr. Wayne Neu and Dr. Dan Walker. I also wish to express gratitude to NASA Langley Research Center and the Computational Methods Branch there for sponsoring my work.

Finally, I thank my family; my parents, my very understanding wife and, in particular, my wife's parents, and , whose unreserved support and encouragement throughout the course of this work have been invaluable.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

a	Speed of sound
β	Matrix bandwidth measured from main diagonal to outer diagonal (not including main diagonal)
C_p	Pressure coefficient
CPU	Central Processing Unit
Δt	Time step
ε	Error vector
e	Total energy per unit volume
$\vec{F}$	Flux vector including viscous contribution
F, G	Inviscid flux and pressure components of F
F_v, G_v	Viscous flux components of F
$\hat{F}$, $\hat{G}$	Inviscid flux components in (ξ , η) space
f_o	Ratio of total CPU time spent in an iteration to CPU time spent in a matrix inversion (approximatly 1.0)
f'_o	Ratio of total CPU time spent in an iteration to CPU time spent in forward and back substitution (LU decomposition bypassed)
f_1	Ratio of total CPU time spent in a VLGS iteration to CPU time spent in a VLGS decomposition process
f_2	Weighted-average ratio for VLGS reuse strategy (timing)
γ	Ratio of specific heats (1.4)
I	Number of iterations
I_i^{i+1}	Restriction operator for multi-grid process
$\hat{i}$, $\hat{j}$	Cartesian unit vectors

J	Inverse of cell volume
J	Number of control volumes in streamwise direction
K	Number of control volumes in normal direction
L	Reference length
ℓ	Vector length, i.e. number of elements in vector
λ	Matrix eigenvalue
M_V	Vector operation peak operating rate (in flops)
M	Mach number
m	Block size (4 for 2-D)
μ	Absolute viscosity
n	Matrix dimension, number of equations
$\hat{n}$	Outward unit normal vector
N	Time level or iteration level
$\lVert\ \rVert$	Vector 2-Norm
ν	Order-of-magnitude reduction in residual/error
P	Pressure
ϕ, κ	Parameters controlling bias and order of upwind differencing
Pr	Prandtl number
Q	Vector of conserved state variables
ρ	Fluid density
ρ	Spectral radius of iteration scheme
ρ_c	Critical spectral radius
Re	Reynolds number
$\hat{R}$	Residual vector (RHS)

$\hat{S}$	Viscous flux vector for thin-layer equations
S	Surface of control volume
t	Time
T	Temperature
τ	CPU time required for execution
τ_o	CPU time required per element of a vector in a vector operation
τ_s	Start-up time required for a given vector operation
u,v	Cartesian velocity components
U,V	Contravariant velocity components
V	Cell volume
x,y	Cartesian coordinates
ξ,η	Generalized coordinates
∞	(subscripted) indicates free-stream conditions

I. INTRODUCTION

Current and future aerospace missions require the design of high performance vehicles including transatmospheric craft such as the National Aerospace Plane. Information concerning the fluid flow about such configurations is essential in the design process. For hypersonic vehicles, the physics of the fluid flow is governed by real gas effects and non-equilibrium chemical reactions within the fluid. Even for moderate flight regimes, the flow can very complex, involving shock-boundary layer interactions, boundary layer separation, vortex formation and breakdown, etc.. Experimental and theoretical investigation of these phenomena will be necessary for a successful design process. In addition, the area of computational fluid dynamics (CFD) will have an increasing role throughout the design process due to the decreasing cost of numerical simulation in conjunction with increasing capability.

The study of CFD has been revolutionized in the last three decades by the advent and rapid evolution of powerful computers. These machines, along with state-of-the-art software, now have the capability of simulating many complex two and three-dimensional flows. This, in general, is done by discretizing the domain of interest, applying algebraic

approximations of the governing equations at each discrete point (or cell) in the domain and enforcing physically realistic boundary conditions. This process results in the need for efficient algorithms to solve large numbers of simultaneous equations.

The direct solution of such systems of linear equations resulting from discretized fluid dynamic problems has not been generally practical in the past. Although it has been known that quadratic convergence to the solution of the non-linear compressible flow equations can be obtained by using such an approach, the consensus has been against using the direct method because of the operation count and the large storage needed for the LU decomposition. Consequently, iteration methods have received the almost unanimous attention of the computational fluid dynamics community. In fact, it has been customary in algorithm development to simply state that the direct method is either impractical or inefficient and then proceed to develop an iteration method.

Recently, however, research has been done with both banded direct and sparse matrix methods. This has been for two (related) reasons: 1) advances in computer hardware which allow fast calculation on large arrays of numbers, and 2) the inherent usefulness of the direct method for problems where standard iteration methods either converge slowly or

fail to converge altogether. Such research includes that of Wigton[1], who has implemented sparse matrix technology (through the application of MACSYMA and related software) for efficiently solving 2-D airfoil problems on a Cray X-MP, and Bender and Khosla[2], who have also used a sparse matrix solver with a stream function - vorticity transport analysis for a driven cavity and airfoil problems. Venkatakrishnan[3] at NASA Langley Research Center, has recently performed a study of direct solvers using both a banded solver modified for efficient implementation on a Cray-2 processor and a sparse-matrix solver. He studied the use of mesh sequencing in conjunction with the direct solution of compressible flow problems. In addition, he examined the direct method as applied on both C and O meshes for airfoil problems. Sub-iterations on the wake line and far-field boundary (due to a point vortex representation of the airfoil) are necessary in order to efficiently obtain quadratic convergence for such grids.

Riggins, et al.,[4] examined the relative efficiency of banded direct strategies in comparison to line Gauss-Seidel iteration for a variety of 2-D compressible problems and show that on vector processors the direct method has a wide range of application. Other notable work with banded direct solvers has been performed by a number of researchers.

Dwyer, et al.,[5] have demonstrated the use of a banded direct solver for the solution of model equations and incompressible flow, as well as for a 3-D and time-dependent spinning cylinder problem. Giles and Drela[6] developed a solver which combines a stream-line determined mesh with Newton's method to obtain solutions for airfoil problems. In addition, Dam, Hafez, and Ahmad[7] have shown quadratic convergence for 2-D incompressible flow using a serial direct solver. They are particularly interested in the usefulness of the direct method for problems where conventional iteration methods fail to converge.

Childs and Pulliam[8] applied Newton's method to an airfoil problem. An iterative multigrid scheme was used for 'convergence' at each Newton iteration rather than a direct solver. Liou[9] also demonstrated quadratic convergence using Newton's method with iteration schemes for 1-D and 2-D inviscid problems with flux-vector splitting. The results of his work clearly show that a large number of Newton iterations are needed in the transient region before quadratic convergence is obtained. This problem is addressed in this work and in Reference 4.

The primary purpose of this work is to examine in detail the banded direct method and obtain the full efficiency of the scheme on available vector machines. It

is shown that the speed of the method makes it favorable in comparison with iteration schemes in many applications. The expanded utility shown here for the banded direct method is due entirely to the exploitation of the vector capabilities of modern supercomputers.

Direct solvers which utilize sparse matrix technology were not examined in detail, since it was not the purpose of this work to compare them to the banded direct solver. However, for all problems and strategies developed involving the banded solver, similar results could be obtained using a sparse matrix solver. Both methods solve a linear system of equations by LU decomposition. No computational results are presented using sparse matrix solvers although the method of nested dissection is discussed qualitatively in Appendix II. The method of physically-based nested dissection reorders the computational domain in a systematic manner in order to minimize both storage and operations required during the LU decomposition. Wigton, in particular, has had success with this approach. However, it should be noted that the vectorized banded solver as compared to existing sparse matrix solvers is very simple to implement and easy to understand. It can be written with less than 20 lines of fortran code (see Appendix III) and, unlike sparse matrix solvers, requires no symbolic manipulation or pre-

processing.

An important feature of the direct method is the quadratic convergence to the steady-state that it exhibits. However, the work of a direct solver in CFD applications, like all other methods, is directly dependent on the size of the problem under consideration. On a serial processor, the computational time required for a direct solution increases rapidly with an increase in problem (grid) size. Hence, any advantage obtained by the quadratic convergence of the method is quickly lost due to the increased CPU time required to solve the problem. On the other hand, iteration methods exhibit linear asymptotic convergence with a relatively small computational time per iteration. For the direct method to compete with iteration methods, the overall computational time (or machine work per iteration) must be reduced to a reasonable level.

The advent of high-speed vector processors has dramatically altered the treatment and development of many iteration methods. Large increases in computational efficiency have been found possible by employing iteration schemes with good 'vectorization' characteristics.[10] It is demonstrated in this work that the banded direct method, when suitably constructed to take advantage of presently available vector processing capabilities and large memory,

is often more efficient than the most popular iteration schemes. On a serial processor, the CPU time of an algorithm is directly proportional to the operation count associated with the algorithm. However, on a vector processor, the CPU time also depends in a more complex manner on the algorithm; the operations that can be vectorized, the vector lengths over which the operations can be performed, etc.. For a matrix with n unknowns and bandwidth β , the operation count for performing a full LU decomposition is $0(n\ \beta^2)$. This is true for both serial and vector processors. However, for a vector processor, the decomposition can be programmed such that the method effectively performs $0(n\beta\ )$ operations on vectors of length β . The total CPU time needed for a solution can be accurately estimated from this type of analysis for any algorithm and is described in a later section. Throughout this work, the terms computational work and CPU time are used interchangeably, but, as explained above, are distinct from operation count.

The vertical line Gauss-Seidel algorithm (VLGS) was chosen as the iteration method to compare with the direct method. It is representative of iteration schemes in common use in CFD applications. Even though it does not completely vectorize like the popular approximate-factorization (AF)

scheme, VLGS, in general has a superior rate of convergence to the steady-state. Thus these two iteration schemes are reasonably cost-competitive on current vector processors.

On the VPS-32 at the NASA Langley Research Center, it is shown that for small-to-moderate sized problems, for both inviscid and viscous flows, the vectorized direct method is more efficient than a vectorized version of line Gauss-Seidel in terms of minimizing CPU times as well as in saving dollars. In fact, it has been found that the limiting criteria for use of the direct solver is machine memory. For 2-D problems which require up to the machine memory limit, a strategy utilizing a direct solver can be developed which is faster and less expensive than vectorized VLGS. For problems too large to fit into high-speed memory, the use of a direct solver in a multi-grid context is discussed. The multi-grid strategy using iteration schemes has been shown to be effective for accelerating the convergence rate of flow problems on large grids with excellent overall reductions in computing time.[12,13,27] It will be shown that the direct method can also be useful for large grid CFD applications as the solver on the coarse grid(s) associated with a multi-grid scheme.

In addition to the advantage that few iterations are needed when using the direct method, another important feature of the method is its ability to rapidly converge to

the steady-state solution of difficult problems for which iteration methods converge slowly or simply fail to converge. The direct method is shown here to be very fast- the CPU execution time required for convergence of airfoil problems is relatively unaffected by wide variations in both Reynolds and Mach numbers. For such cases, iteration methods often perform poorly.

This work is organized as follows: Chapter II describes the governing equations for 2-D compressible flow and their numerical treatment. The resulting matrix structure of the linear system and the convergence of the solution process for the non-linear problem are analyzed in Chapter III and Chapter IV. Chapter V discusses the efficient implementation of the direct and the VLGS methods on vector processors. A CPU execution time prediction capability is developed in Chapter VI for solution processes using either the direct or the VLGS schemes. A comparison study based on this capability is then made to determine the range of efficient direct solution. Chapter VII briefly describes mesh-related convergence acceleration techniques used in the computational studies. Finally, Chapter VIII is a compilation of the computational results for three test problems: 1) transonic channel flow, 2) laminar shock-boundary layer interaction and 3) NACA 0012 airfoil Mach and Reynolds numbers variation studies.

II. GOVERNING EQUATIONS

The governing Navier-Stokes equations for compressible flow can be written in integral form as

$$\frac{\partial}{\partial t}\iiint_V \vec{Q}\,dV + \iint_S \vec{F}\cdot\hat{n}\,ds = 0. \qquad (1)$$

$\vec{Q}$ is the vector of conserved state quantities, $\vec{F} = (F - F_V)\hat{i} + (G - G_V)\hat{j}$ (where F and G are the inviscid fluxes and pressure terms and F_V and G_V are the viscous flux contributions) and $\hat{n}$ is the outward unit normal from surface S which bounds an arbitrary volume, V. Equation (1) describes conservation of mass, momentum and energy in the volume. This relationship can be more usefully written in differential form for a given cell in terms of $\hat{Q}$, where $\hat{Q}$ is a cell-averaged value, i.e.,

$$\hat{Q} = \frac{1}{V}\iiint_V \vec{Q}(x,y,z,t)\,dV \qquad (2)$$

rather than a point-wise cell-centered value.[31] Inviscid flux vectors, $\hat{F}$ and $\hat{G}$, are defined such that they are evaluated at cell sides. In addition, the thin-layer assumption is made such that all viscous terms with derivatives in the streamwise direction have been neglected. Such an approximation is valid for flows in which there are relatively thin regions of separated and reversed flow and

is simpler to implement than the complete Navier-Stokes equations. This results in the following expression in generalized coordinates,

$$\frac{\partial \hat{Q}}{\partial t} + \frac{\partial \hat{F}}{\partial \xi} + \frac{\partial \hat{G}}{\partial \eta} = \frac{1}{Re} \frac{\partial \hat{S}}{\partial \eta} , \tag{3}$$

for 2-D flow, where

$$\hat{Q} = \frac{1}{J}\begin{Bmatrix} \rho \\ \rho u \\ \rho v \\ e \end{Bmatrix} , \quad \hat{F} = \frac{1}{J}\begin{Bmatrix} \rho U \\ \rho Uu + \xi_x P \\ \rho Uv + \xi_y P \\ (e + P)U \end{Bmatrix} , \quad \hat{G} = \frac{1}{J}\begin{Bmatrix} \rho V \\ \rho Vu + \eta_x P \\ \rho Vv + \eta_y P \\ (e + P)V \end{Bmatrix}$$

and

$$\hat{S} = \mu J \left(\frac{|\nabla \eta|}{J}\right)^2 \begin{Bmatrix} 0 \\ (1 + \frac{\bar{\eta}_x^2}{3}) u_\eta + (1/3)\ \bar{\eta}_x \bar{\eta}_y v_\eta \\ (1 + \frac{\bar{\eta}_y^2}{3}) v_\eta + (1/3)\ \bar{\eta}_x \bar{\eta}_y u_\eta \\ 1/2\ \ (1 + \frac{\bar{\eta}_x^2}{3})(u^2)_\eta + (1 + \frac{\bar{\eta}_y^2}{3})(v^2)_\eta \\ + 1/3[\bar{\eta}_x \bar{\eta}_y (uv)_\eta] + \dfrac{T_\eta}{(\gamma-1) M_\infty^2 \ Pr} \end{Bmatrix} \tag{4}$$

($\bar{\eta}_x = \eta_x / |\nabla \eta|$, etc.).

The equations are nondimensionalized by the reference density, ρ_∞ , the free-stream velocity, U_∞ , and a reference length, L, where the Reynolds number, Re, is equal to $\rho_\infty U_\infty L / \mu_\infty$.

$\xi , \eta = \xi, \eta\ (x,y)$ define the coordinate axes of the computational domain and for convenience $\Delta \xi$ and $\Delta \eta = 1$.

(1/J) is the cell volume where J is the Jacobian of the metric formulation and U and V are the contravariant velocities defined as

$$U = \xi_x u + \xi_y v \tag{5}$$

$$V = \eta_x u + \eta_y v.$$

The computational coordinates (ξ, η) are defined such that ξ = constant and η = constant lines form the cell faces in the physical domain. Hence, the vectors $\nabla\xi/J$ and $\nabla\eta/J$ represent directed areas of cell interfaces in the ξ and η direction, respectively. For example, the second and third components of $\hat{F}$ are the inviscid flux contributions crossing the ξ = constant cell face in the x direction and in the y direction, respectively. $\hat{F}$ and $\hat{G}$ in equation (3) are differenced across some arbitrary cell (j,k) such that (for example):

$$\frac{\partial\hat{F}}{\partial\xi} = \hat{F}_{j+1/2,k} - \hat{F}_{j-1/2,k} \tag{6}$$

The indices (j+1/2) and (k+1/2) denote (j,k) cell faces in the positive ξ and η directions, respectively.

These equations must be closed by an equation of state. Here, a calorically perfect gas is assumed such that γ (the ratio of specific heats) is constant and

$$P = (\gamma-1)[e- \rho(u^2 + v^2 + w^2)/2]. \tag{7}$$

In this work, the inviscid fluxes are spatially split

using the Van Leer flux-vector splitting technique.[14] $\hat{F}$ and $\hat{G}$ are split such that (for example):

$$\frac{\partial \hat{F}}{\partial \xi} = (\hat{F}^{+}_{j+1/2,k}(Q^{-}) + \hat{F}^{-}_{j+1/2,k}(Q^{+})) \tag{8}$$

$$- (\hat{F}^{+}_{j-1/2,k}(Q^{-}) + \hat{F}^{-}_{j-1/2,k}(Q^{+}))$$

In equation (8), $\hat{F}^{+}$ is the flux contribution associated with the positive eigenvalues (wavespeeds) of the Jacobian, $\partial\hat{F}/\partial\hat{Q}$, and is evaluated at the indicated cell face of cell (j,k). $\hat{F}^{-}$ is the split flux associated with the negative eigenvalues of $\partial\hat{F}/\partial\hat{Q}$. The superscripts on $\hat{Q}$ (+ or -) indicate that the conserved quantities are differenced in an upwind sense. The standard upwind-biased interpolating polynomials[26] are used:

$$Q^{-}_{j+1/2,k} = Q_{j,k} + \frac{\phi}{4}\,[(1-\kappa)\nabla + (1+\kappa)\Delta]Q_{j,k} \tag{9a}$$

$$Q^{+}_{j+1/2,k} = Q_{j+1,k} - \frac{\phi}{4}\,[(1+\kappa)\nabla + (1-\kappa)\Delta]Q_{j+1,k} \tag{9b}$$

where Δ and ∇ denote the forward and backward difference operators (i.e. $\Delta Q_{j,k} = Q_{j+1,k} - Q_{j,k}$, etc.). ϕ is a switch; $\phi = 0$ yields first-order upwind differencing while $\phi = 1$ and the parameter κ determine the order of accuracy of the scheme. Second-order fully upwind differencing corresponds to $\kappa = -1$ and third-order upwind-biased differencing to

$\kappa = 1/3$, etc.. Similar results can be obtained for the flux contribution, $\hat{G}$.

The advantage of the Van Leer splitting over the Steger-Warming splitting[32] is that the fluxes are continuous and differentiable through sonic and stagnation points.[13] This method has been used extensively by a number of investigators[11,29] and allows efficient implementation of higher-order upwind algorithms. Van Leer orginally developed and presented the flux vector splitting for Cartesian coordinates.[14] References 11 and 15 extended this method to generalized coordinates (ξ , η) by evaluating split flux vector contributions perpendicular to cell faces. This is accomplished by the use of a coordinate rotation of flux vectors $\hat{F}$ and $\hat{G}$ into a local Cartesian coordinate system at the cell face; hence, the Cartesian flux split contributions developed in Reference 14 can be applied to the rotated system in a straight-forward fashion. The result is then rotated back to generalized coordinates. $\hat{F}^{\pm}$ is given below. $\hat{G}^{\pm}$ can be evaluated in a similar manner. $\hat{F}$ is split according to the contravariant Mach number in the ξ direction, $M_{\xi} = \bar{u}/a = U/(a\,|\nabla\xi|)$. For supersonic flow where $|M_{\xi}| \geq 1$,

$$\hat{F}^{+} = \hat{F} \text{ and } \hat{F}^{-} = 0 \text{ if } M_{\xi} \geq 1.0 \tag{10}$$

$$\hat{F}^{-} = \hat{F} \text{ and } \hat{F}^{+} = 0 \text{ if } M_{\xi} \leq -1.0. \tag{11}$$

For subsonic flow where $|M_\xi| < 1$, $\hat{F}$ is defined as:

$$\hat{F}^{\pm} = \frac{|\nabla \xi|}{J} \begin{Bmatrix} f^{\pm}_{mass} \\ f^{\pm}_{mass} \left[\frac{\bar{\xi}_x}{\gamma} (-\bar{u} \pm 2a) + u \right] \\ f^{\pm}_{mass} \left[\frac{\bar{\xi}_y}{\gamma} (-\bar{u} \pm 2a) + v \right] \\ f^{\pm}_{mass} \left[\frac{-(\gamma-1)\bar{u}^2 \pm (\gamma-1)\bar{u}a + 2a^2}{(\gamma^2-1)} + \frac{u^2 + v^2}{2} \right] \end{Bmatrix} \tag{12}$$

where $f^{\pm}_{mass} = \pm \frac{\rho a}{4} (M_\xi \pm 1)^2$ and a is the local speed of sound.

The viscous term, $\hat{S}$, in equation (3) is differenced as

$$\frac{\partial \hat{S}}{\partial \eta} = \hat{S}_{j,k+1/2} - \hat{S}_{j,k-1/2} \tag{13}$$

across cell (j,k). Velocity derivatives in $\hat{S}$ are central differenced. The inverse-volume muliplier (1/J) in $\hat{S}$ is treated as the cell volume-average (i.e. the average of cells (j,k) and (j,k+1) for $\hat{S}_{j,k+1/2}$). The state variables are naturally obtained from the cell centers due to the central differencing of the derivatives within $\hat{S}$ with the exception of the viscosity, μ , which is based on an 'averaged' temperature such that

$$\mu\,[T_{j,k+1/2}] = \mu[T(Q_{j,k+1/2})] = \mu\left[T\left(\frac{Q_{j,k+1} + Q_{j,k}}{2}\right)\right] \tag{14}$$

Sutherland's law[16] is used to relate μ and temperature:

$$\frac{\mu}{\mu_\infty} = \left(\frac{T}{T_\infty}\right)^{3/2} \frac{(T_\infty + T')}{(T + T')} \tag{15}$$

where T' = 110.4 K.

III. MATRIX FORMULATION

By applying the Euler implicit time integration scheme to equation (1), the following system can be written for control volume (or cell) (j,k):

$$\frac{1}{J}\ \frac{\Delta\hat{Q}_{j,k}}{\Delta t} + \hat{R}_{jk}^{N+1} = 0 \tag{16}$$

where $\Delta\hat{Q}_{jk} = \hat{Q}_{jk}^{N+1} - \hat{Q}_{jk}^{N}$, N denotes the time (or iteration) level, Δt is the time step, and $\hat{R}_{jk}^{N+1}$ is the steady-state equation residual (the discrete representation of the surface integral in equation (1));

$$\hat{R}_{jk}^{N+1} = \hat{F}_{j+1/2,k}^{N+1} - \hat{F}_{j-1/2,k}^{N+1} + \hat{G}_{j,k+1/2}^{N+1} - \hat{G}_{j,k-1/2}^{N+1} - \frac{1}{Re}\left[\hat{S}_{j,k+1/2}^{N+1} - \hat{S}_{j,k-1/2}^{N+1}\right] . \tag{17}$$

$\hat{R}^{N+1}$ can be written in terms of $\hat{R}^{N}$ using a Taylor series expansion as

$$\hat{R}^{N+1} = \hat{R}^{N} + \left(\frac{\partial\hat{R}}{\partial\hat{Q}}\right)^{N} \Delta\hat{Q} + \left(\frac{\partial^2\hat{R}}{\partial\hat{Q}^2}\right)^{N} \frac{(\Delta\hat{Q})^2}{2} + 0(\Delta\hat{Q})^3. \tag{18a}$$

Note that if $\hat{R}^{N}$ is a linear function of $\hat{Q}$, then $(\partial\hat{R}/\partial\hat{Q})^{N}$ is constant and all higher order terms $(\Delta\hat{Q})^2$, $(\Delta\hat{Q})^3$, etc.,

are zero. For non-linear $\hat{R}^N$, $\hat{R}^{N+1}$ is written as

$$\hat{R}^{N+1} \approx \hat{R}^N + \left(\frac{\partial \hat{R}}{\partial \hat{Q}}\right)^N \Delta \hat{Q} \tag{18b}$$

with truncation error of $0(\Delta t^2)$ since $\Delta \hat{Q}$ is $0(\Delta t)$.

Linearization of equation (16) results in the approximate system

$$\left[\frac{I}{J\,\Delta t} + \frac{\partial \hat{R}}{\partial \hat{Q}}\right]^N \Delta \hat{Q}_{jk} = -\hat{R}^N_{jk}\,. \tag{19}$$

The rectangular computational domain is composed of J control volumes in the ξ direction and K control volumes in the η direction and is numbered as shown in Figure 1. Then equation (19) applied at each cell along with appropriate boundary conditions yields a linear system with a banded matrix with block matrix structure, i.e.

$$A\ \overline{\Delta Q} = \overline{R}. \tag{20}$$

A is a square matrix of overall dimension n (where n=4JK) and $\overline{\Delta Q}$ and $\overline{R}$ are vectors of length 4JK.

For first-order differencing, the computational molecule involves control volumes (j,k), (j+1,k), (j-1,k), (j,k+1) and (j,k-1). Hence, the equation for each volume represented in equation (20) has the form

$$a\Delta\hat{Q}_{j-1,k} + b\Delta\hat{Q}_{j,k-1} + c\Delta\hat{Q}_{j,k} + d\Delta\hat{Q}_{j,k+1} + e\Delta\hat{Q}_{j+1,k} = -\hat{R}_{jk} \quad (21)$$

where a,b,c,d and e are 4x4 coefficient matrices. Thus, assuming that the nodes are sequentially numbered as shown in Figure 1, the bandwidth of the global matrix A is 4K.

For second-order differencing, a nine-diagonal system results. In this case, the computational molecule will include additional contributions from (j,k+2), (j,k-2), (j+2,k) and (j-2,k) leading to

$$f\Delta\hat{Q}_{j-2,k} + a\Delta\hat{Q}_{j-1,k} + g\Delta\hat{Q}_{j,k-2} + b\Delta\hat{Q}_{j,k-1} + c\Delta\hat{Q}_{jk} + d\Delta\hat{Q}_{j,k+1} + h\Delta\hat{Q}_{j,k+2} + e\Delta\hat{Q}_{j+1,k} + f\Delta\hat{Q}_{j+2,k} = -\hat{R}_{jk} \quad (22)$$

The bandwidth of A would be 8K for second-order differencing.

Although this work is concerned with steady-state problems ($\Delta t \rightarrow \infty$ in equation (19)), the time term is kept to facilitate the solution process. The presence of a finite time step increases the diagonal dominance of A which usually enhances the stability of a solution process. For steady-state solutions, time accuracy is not necessary or desirable, thus Δt is, in general, an adjustable (but highly important) parameter for convergence of a given problem. If time accuracy is important then Δt must be kept small due to truncation error.

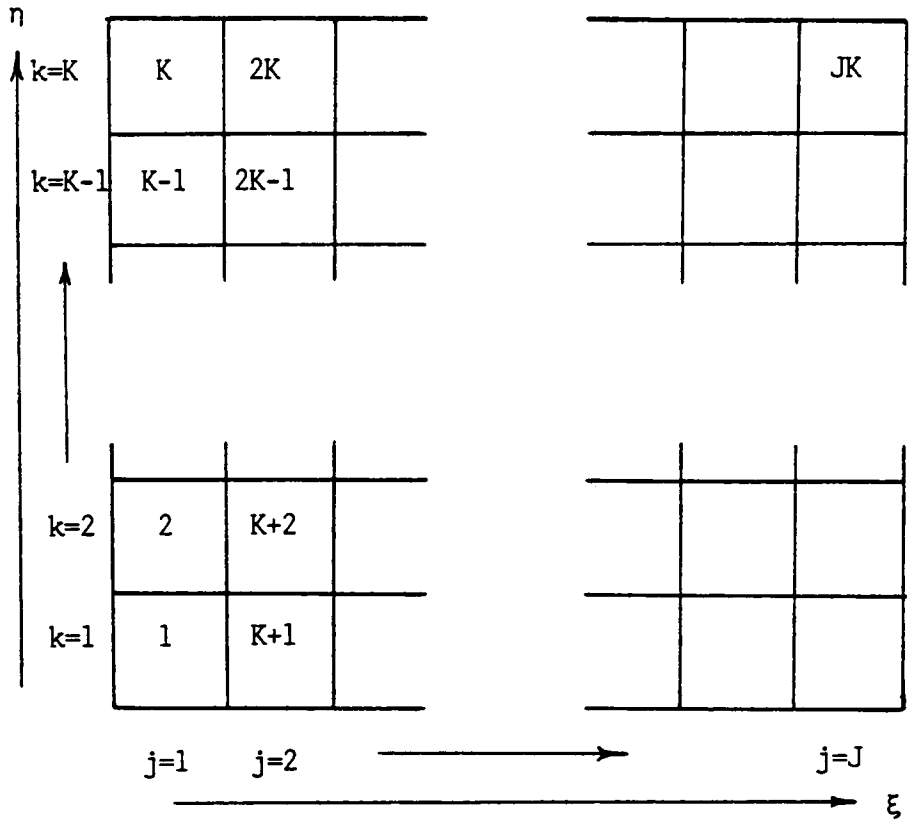


Figure 1: Computational Domain Numbering System

IV. CONVERGENCE OF THE SOLUTION PROCESS FOR NON-LINEAR SYSTEMS

The linear system arising from a finite difference or finite volume discretization of the governing equation of fluid flow, $A\bar{x}=\bar{b}$, has traditionally been solved using iteration methods. An iteration method can be defined as a solution strategy which approximates the left-hand side coefficient matrix, generates an intermediate solution vector $\tilde{x}$, and repeats the process until a converged solution $\tilde{x} = \bar{x}$ is found. The alternative solution strategy is a direct solution procedure (such as LU decomposition). The direct method solves the linear system exactly, either by directly finding the inverse of A (Cramer's rule) or by decomposing A into the product of lower and upper triangular matrices, L and U, and solving the system with forward and back substitutions. Cramer's rule is not even remotely feasible for systems of interest; it takes order $(n+1)!$ operations, where n is the number of unknowns. Hence, in this work, the direct method will mean exclusively the solution of $A\bar{x}=\bar{b}$ by LU decomposition. Due to the non-linear nature of the compressible flow equations, the direct method itself must be applied iteratively in order to fully solve the non-linear problem. For steady-state solutions ($\Delta t \rightarrow \infty$), this is simply Newton's root-finding method applied to a

multi-equation system. On the other hand, iteration methods must resolve the non-linearity in the equations as well as approximate the linear problem itself as noted above. Thus, in general, iteration schemes require many more iterations to solve a given system than the direct method. The following sections analyze the asymptotic rate of convergence for both the iteration and the direct methods.

A. Convergence of Iteration Methods

For iteration methods, the linear system $A\bar{x}=\bar{b}$ is solved by generating successive approximations to $\bar{x}$. This is done by defining a matrix B where B=A-C such that B is easily invertible or readily decomposed into a lower and upper triangular matrix. This yields the system

$$\bar{x}^N = B^{-1}\bar{b} + (I- B^{-1} A)\ \bar{x}^{N-1} \tag{23}$$

$(I-B^{-1}A)$ is called the iteration matrix for the algorithm. The error $\bar{\varepsilon}^N = \bar{x}^N - \bar{x}_{exact}$, at time level N can be written in terms of the error at previous time level N-1 as

$$\bar{\varepsilon}^N = (I-B^{-1}A)\ \bar{\varepsilon}^{N-1} . \tag{24}$$

For an iteration method, convergence is determined by the spectral radius, ρ , of the iteration matrix where ρ is

equal to the magnitude of the largest eigenvalue, λ_{max}, of the iteration matrix $(I-B^{-1}A)$. (The magnitude of the largest eigenvalue of a matrix can be thought of as the maximum radius for the solution space 'envelope'. No possible solution vector will have length exceeding this maximum radius.) Hence, if ρ is less than 1.0, the method will result in linear convergence with the rate of convergence being governed by the magnitude of ρ. For computational fluid dynamics calculations using an iteration method, ρ is typically between 1.0 and 0.9.

Note that for an iteration process with I iterations and with some starting error $\bar{\varepsilon}^o$,

$$\bar{\varepsilon}^I = (I-B^{-1}A)^I \; \bar{\varepsilon}^o \tag{25}$$

and the slowest convergence possible for large I would exhibit

$$\left\| \bar{\varepsilon}^I \right\| \approx \rho^I \left\| \bar{\varepsilon}^o \right\| . \tag{26}$$

To reduce the error ν orders of magnitude such that

$$\frac{\left\| \bar{\varepsilon}^I \right\|}{\left\| \bar{\varepsilon}^o \right\|} = 10^{-\nu}, \tag{27}$$

I iterations are required where

$$I = \frac{-\nu}{\log \rho} . \tag{28}$$

Iteration methods, then, in general, exhibit linear asymptotic convergence. For the actual non-linear problem of interest the fore-going analysis applies approximately since from iteration to iteration the matrix A and the right-hand side vector $\bar{b}$ do not usually change rapidly. This is particularly true in the region of asymptotic convergence.

If, on the other hand, the linearized problem is solved exactly at each time step by using an iteration method, then this is simply Newton's method via an iteration method. However, because of time step restrictions with the iteration schemes, it is usually more efficient to update A and $\bar{b}$ throughout the entire iteration process for steady-state solutions.

For linear problems and point Jacobi or point Gauss-Seidel iteration, it is easy to show that a sufficient condition for convergence is diagonal dominance of the matrix A (such as occcurs with upwind first-order differencing). This is in fact true for all iteration methods - the rate of convergence is closely related to the diagonal dominance of A. For block-type schemes as arise in computational fluid dynamics in which the coefficient matrix itself is approximated and there is non-linear updating on both left and right-hand sides, the analysis is more

complicated but general features remain the same. The criteria of diagonal dominance is seldom met in computational fluid calculations where higher-order differencing is routine. As a result, even the best iteration methods are usually frustratingly slow, often require large numbers of iterations with severe time step limitations and can fail to converge altogether. The advantages of iteration methods are 1) fast CPU time per iteration and 2) very low storage requirements.

B. Convergence of Newton's Method for a System of Equations

The convergence of the direct method is investigated by considering a system of n non-linear equations:

$$f_i(\bar{x}_e) = 0 \qquad i = 1 \ldots n \tag{29}$$

i.e.

$$\begin{aligned}
&f_1(x_1, x_2, x_3, \ldots, x_i, \ldots x_n) = 0 \\
&f_2(x_1, x_2, x_3, \ldots, x_i, \ldots x_n) = 0 \\
&\vdots \\
&f_i(x_1, x_2, x_3, \ldots, x_i, \ldots x_n) = 0 \\
&\vdots \\
&f_n(x_1, x_2, x_3, \ldots, x_i, \ldots x_n) = 0.
\end{aligned}$$

This can be written as $L(\bar{x}_e) = 0$ where L is a non-linear operator and $\bar{x}_e$ is the exact solution vector.

From Taylor's series, assuming $\bar{x}^N$ close to $\bar{x}_e$, we can write

$$\bar{x}_e^{N+1} \approx \bar{x}^N - (H\ \bar{f})^N \tag{30}$$

where superscripts denote iteration and

$$\bar{x} = \begin{Bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{Bmatrix}; \quad \bar{f} = \begin{Bmatrix} f_1 \\ f_2 \\ \cdot \\ \cdot \\ \cdot \\ f_n \end{Bmatrix}; \quad H = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \\ \cdot & \cdot & & \\ \cdot & \cdot & & \\ \cdot & \cdot & & \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}^{-1}$$

Note that at $\bar{x}_e$ (exact), $\bar{f} = 0$, or $(f_1, f_2, \ldots f_n) = 0$.

Consider a single equation (i) from this system such that

$$x_i^{N+1} = x_i^N - \sum_{j=1}^{n} (h_{ij} f_j)^N . \tag{32}$$

Let

$$g_i^N = x_i^N - \sum_{j=1}^{n} (h_{ij} f_j)^N . \tag{32}$$

Then for any k from 1 to n,

$$\left(\frac{\partial g_i}{\partial x_k}\right)^N = \left(\frac{\partial x_i}{\partial x_k}\right)^N - \sum_{j=1}^{n} \left(\frac{\partial h_{ij}}{\partial x_k}\right)^N f_j^N - \sum_{j=1}^{n} h_{ij}^N \left(\frac{\partial f_j}{\partial x_k}\right)^N. \tag{33}$$

when $k = i$

$$\sum_{j=1}^{n} h_{ij}^N \left(\frac{\partial f_j}{\partial x_k}\right)^N = 1 \tag{34}$$

and if $k \neq i$

$$\sum_{j=1}^{n} h_{ij}^N \left(\frac{\partial f_j}{\partial x_k}\right)^N = 0. \tag{35}$$

Then

$$\left(\frac{\partial g_i}{\partial x_k}\right)^N = -\sum_{j=1}^{n} \left(\frac{\partial h_{ij}}{\partial x_k}\right)^N f_j^N \tag{36}$$

and note that $\left(\frac{\partial g_i}{\partial x_k}\right) = 0$ at $\bar{x}_e$ (exact) for all i,k since $f_1, f_2 \ldots f_n) = 0$.

Expand g_i^N in equation (32) about $\bar{x}_e$ (exact):

$$g_i^N \approx g_i(\bar{x}_e) + \sum_{j=1}^{n} \left[\left(\frac{\partial g_i}{\partial x_j}\right)^N_{\bar{x}_e} (x_j^N - x_{j\,exact}) + \left(\frac{\partial}{\partial x_j} \frac{\partial g_i}{\partial x_j}\right)^N_{\bar{x}_e} \frac{(x_j^N - x_{j\,exact})^2}{2} \right] \tag{37}$$

Then since $g_i(\bar{x}_e) = x_i(\text{exact})$, the error, ε_i, is defined as

$$\varepsilon_i^N = x_i^N - x_i(\text{exact}) \tag{38}$$

and

$$\varepsilon_i^{N+1} \approx \sum_{j=1}^{n} \frac{1}{2} \left[\left(\frac{\partial}{\partial x_j} \frac{\partial g_i}{\partial x_j}\right)^N (\varepsilon_j^N)^2 \right]. \tag{39}$$

Since i was chosen arbitrarily, the error on the ith element of the solution vector (in the limit) is proportional to the sum of the squares of the previous errors (element-wise) assuming $\bar{x}^N \approx \bar{x}_e$. This results in asymptotic quadratic convergence to the steady-state (i.e. errors decrease as 10^{-2}, 10^{-4}, 10^{-8}, etc.) and is simply Newton's method for a system of non-linear equations.

Note that for iteration methods, in general, H^N is approximated so that $\left(\partial g_i / \partial x_j\right) \neq 0$ at $\bar{x}_e$ and

$$\varepsilon_i^{N+1} = \sum_{j=1}^{n} \left(\frac{\partial g_i}{\partial x_j}\right)^N_{\bar{x}_e} \varepsilon_j^N + \ldots \tag{40}$$

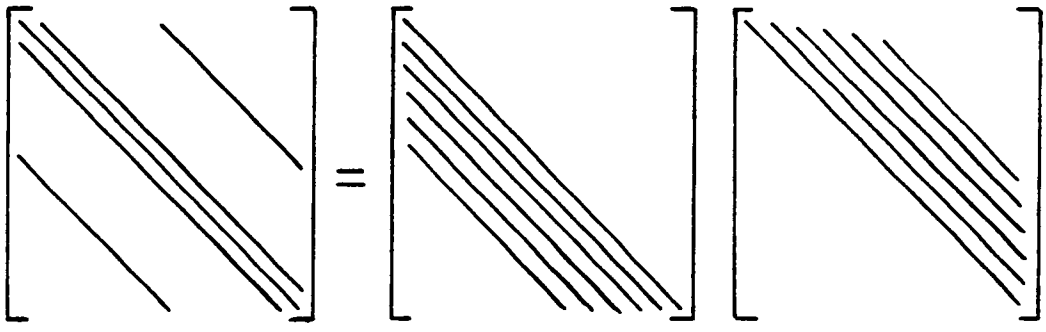
which indicates that iteration methods result in linear convergence.

Quadratic convergence to steady-state solutions (Newton's method) using the direct method is not explicitly affected by the diagonal dominance of A (although main-diagonal divisors in the inversion or LU decomposition process should not be so small as to cause unacceptable numerical errors.) This is not a problem with upwind spatial discretizations of the convective terms in the governing equations. Very large time steps to the steady-state can be taken with the direct method. The disadvantages of the method have been the prohibitive CPU time required per iteration (on serial processors) and the large memory required. Until very recently these disadvantages outweighed the fact that few iterations are necessary with the direct method. Consequently, iteration methods, both implicit and explicit, have been used for almost all CFD calculations. The following sections discuss the efficient implementation of the direct method on a vector processor and a representative iteration method to be used for comparison purposes, the vectorized vertical line Gauss-Seidel scheme (VLGS).

V. IMPLEMENTATION ON VECTOR PROCESSORS

A. The Banded Direct Method

The direct method first decomposes the coefficient matrix, A, into the product of a lower and an upper triangular matrix; A=LU. This is pictured below for a first-order system:



The system $A\bar{x}=\bar{b}$ can then be solved in a two-step process:

$$L\bar{x}^* = \bar{b} \quad \text{(forward substitution) and} \tag{41}$$

$$U\bar{x} = \bar{x}^* \quad \text{(back substitution).} \tag{42}$$

The LU decomposition process is both time and memory intensive. Element fill-in within L and U results in memory requirements of $2n\beta$ for a symmetrically banded matrix A. However, an efficient procedure for performing the

decomposition is obtained by first defining a working matrix C of dimension $(2*\beta+1,n)$ such that the rows of C are the diagonals of the matrix A, with the first row being the far right upper diagonal, the next row being the diagonal directly below the far right upper diagonal, etc.. The main diagonal of A will then lie on the middle ($\beta + 1$) row of C. C is a single computational matrix which combines both L and U. Note that the upper diagonals must be offset with initial zeroes as shown below for a simple penta-diagonal (first-order) case:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 & 0 & 0 & 0\ldots \\ a_{21} & a_{22} & a_{23} & a_{24} & 0 & 0 & 0\ldots \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & 0 & 0\ldots \\ 0 & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & 0\ldots \\ 0 & 0 & a_{53} & \ldots & & & \\ 0 & 0 & 0 & \ldots & & & \\ \vdots & & & & & & \end{bmatrix} ;$$

$$C = \begin{bmatrix} 0 & 0 & a_{13} & a_{24} & a_{35}\ldots\ldots & \\ 0 & a_{12} & a_{23} & a_{34} & a_{45}\ldots\ldots & \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55}\ldots\ldots & a_{nn} \\ a_{21} & a_{32} & a_{43} & a_{54} & a_{65}\ldots\ldots & 0 \\ a_{31} & a_{42} & a_{53} & a_{64} & a_{75}\ldots\ldots 0 & 0 \end{bmatrix} \qquad (43)$$

All intermediate zero diagonals of A between the outermost diagonals and the main diagonal must be included in C. Elements in these locations are subject to fill-in in the LU decomposition process. The algorithm for the decomposition can then be written as (the complete fortran listing for the banded direct solver is given in Appendix III):

```
for I=1, n-1
C( β+1, I; β ) = C( β+1, I; β )/C( β+1, I)
for J=1,
C( β-J+2, I+J; β ) = C( β-J+2, I+J; β )
                   - C( β-J+1, I+J)   C( β+2, I; β )
next J
next I
```

where the 'semi-colon' notation (; β) indicates a vector operation such that the given operation is to be performed over a length, β . (Note that near the end of the I loop when (n-I) is less than or equal to β , the vector length over which the operations are done varies such that it equals (n-I) rather than β .) The decomposition effectively performs O n(β +1) operations on vectors of length β and completely forms the elements of L and U (corresponding to the lower and upper halves of C, respectively).

The forward and back substitution algorithms are as follows (where $\bar{x} = \bar{b}$, initially):

```
for I-1, n-1
x(I+1; β ) = x(I+1; β ) - x(I) * C( β +1, I; β )
next I
```

and

```
for K=1, n-1
I=n-k+1
x(I) = x(I)/C( β +2, I)
x(I- β ; β ) = x(I- β ; β ) - x(I) * C(1,I; β )
next K
x(1) = x(1)/C( β +1, 1).
```

(Again, in the forward substitution, when (n-I) is less than or equal to β , the vector length varies such that it equals (n-I). Similarly, for the back substitution, when I is less than or equal to β , the vector length varies as (I-1).)

These two phases basically perform O(n) operations on vectors of length β . The total operation count for the vectorized direct solution is summarized in Table 1. Notice that the decomposition phase is the most time-intensive, taking $O(n\beta)$ vector operations.

This is a very lean algorithm. Particular computational advantage comes from the simple vector

Table 1. Operation Count for Direct Solver

	Divisions	Linked Triads
Decomposition	$[n-\beta]_{(\beta)} + \sum_{k=1}^{\beta-1} 1_{(k)}$	$[(n-\beta)(\beta)]_{(\beta)} + \sum_{k=1}^{\beta-1} [\beta-k]_{(\beta-k)}$
Forward Substitution	0	$[n-\beta]_{(\beta)} + \sum_{k=1}^{\beta-1} 1_{(k)}$
Back Substitution	n	$[n-\beta]_{(\beta)} + \sum_{k=1}^{\beta-1} 1_{(k)}$

(Note: Subscripts indicate vector length over which operations are performed)

operations used - peak machine operating rates can be obtained for these operations.

B. Vertical Line Gauss Seidel (VLGS)

The VLGS scheme is obtained by first approximating the left-hand side matrix (equation (20)) by setting those coefficients of the matrix to zero in equation (21) which are associated with cells downstream of the sweep direction. The solution is then obtained on each (vertical) line, taking upstream (known) information to the right hand side. This requires the decomposition over the entire domain of a block tri-diagonal matrix for first-order differencing or a block penta-diagonal matrix for second-order differencing. This decomposition can be vectorized over the number of lines (J) in the entire field. However, the back-substitution phase of the method cannot be vectorized. (The method of Approximate Factorization or AF performs the decomposition of the matrix in exactly the same fashion-however, since upstream information is not passed to the right-hand side, the back-substitution phase for AF is fully vectorizable.) It is important to note that the LU decomposition can be reused (or 'frozen') as mentioned earlier for a substantial number of iterations in a solution process. This can reduce the CPU time required by a factor

of two for the VLGS method.[11]

VI. CONVERGENCE TIMING ANALYSIS AND PREDICTION

A. Vector Timing Considerations

Vector processors utilize a pipe-line strategy to efficiently add, multiply, divide, or perform other relatively simple operations on long vectors. For timing analysis purposes, a vector operation is not inherently different than a scalar do-loop over some length. Both have an associated start-up time and a CPU time required per element/operation. The advantage of the vectorized operation is in the rapid CPU time in which each element of the vector is processed. The CPU time, τ , required for a vector operation is given by the linear relationship,

$$\tau = \tau_o \ell + \tau_s \tag{44}$$

where τ_s is the vector instruction 'start-up' time and τ_o is the CPU time required to perform a single operation on an element of the vector of length ℓ . Such a relationship is shown in Figure 2a. To obtain a floating point operations/second (flops) versus vector length curve, find ℓ / τ :

$$\frac{\ell}{\tau} = \frac{1}{\tau_o}\left(\frac{\tau_s}{\tau} + 1\right) \tag{45}$$

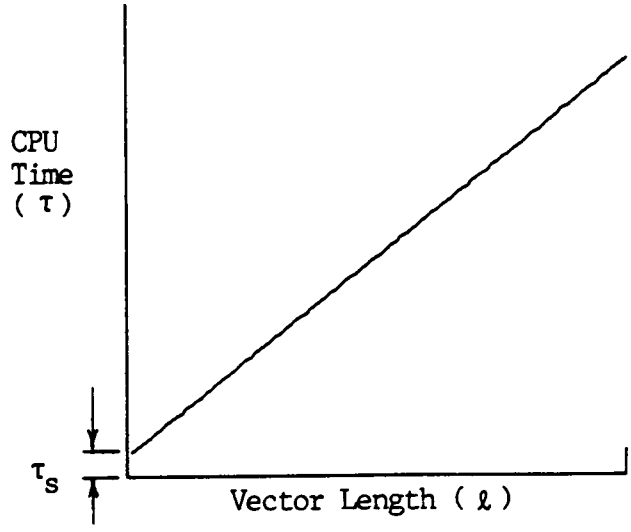
The result is shown in Figure 2b. Note that for very long vector lengths ($\ell \rightarrow \infty$)

$$\lim_{\ell \rightarrow \infty} \frac{\ell}{\tau} = \frac{1}{\tau_o} = M_V \text{ asymptotic flop rate of machine for given operation} \tag{46}$$

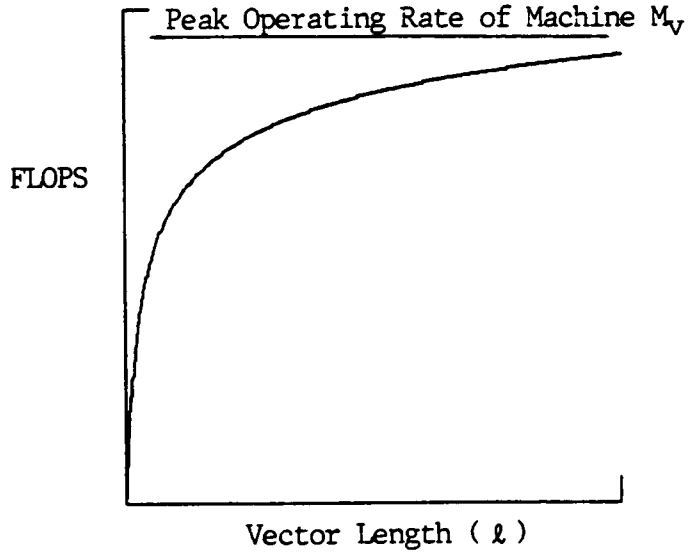
M_V is often expressed in millions of flops (Mflops). Since M_V is a constant for a given vector operation, τ can be written as

$$\tau = \frac{\ell}{M_V} + \tau_s \tag{47}$$

Vector operations of interest in this study are additions (a=b+c), multiplications (a=b*c), divisions (a=b/c) and linked triads (a=b+c*d). Linked triads are apparently not recognized by the Cray 2 compiler (unlike the Cyber 205 or the Cray X-MP). For machines where there is no recognization of linked triads it is trivial to break the linked triad into a separate multiplication and addition for timing analysis. It should be emphasized, however, that the linked triad is highly efficient, and particularly beneficial for efficiency when using the direct solver. It can be almost twice as fast as two separate but equivalent operations. Machine parameters M_V and τ_s for vector operations of interest are given in Appendix I.



(a) CPU Time (τ) vs Vector Length



(b) Flops vs Vector Length

Figure 2. Vector Operation Timing

B. Direct Method Timing

The time, $\tau^{N \to N+1}$, of a complete direct solution can be readily obtained from the description of vector timing in the previous section and the operation count in Table 1. For example, for the leading order term in the matrix decomposition, $[(n-\beta)\beta]$, the CPU time required can be written as

$$[(n-\beta)\beta] \left(\frac{\beta}{M_V} + \tau_s\right) \text{linked triad} \tag{48}$$

The total time required for an iteration with a direct matrix solution is given by

$$\tau^{N \to N+1} = f_o \left(\begin{array}{l} \left[\frac{1}{M_V}(n\beta^2 + 2n\beta - \beta^3 - \beta^2 - \beta) \right. \\ \qquad \left. + \tau_s \left(n\beta - \frac{\beta^2}{2} + 2n - \frac{\beta}{2} - 2\right)\right] \text{linked triad} \\ + \left[\frac{1}{M_V}\left(n\beta - \frac{\beta^2}{2} - \frac{\beta}{2}\right) + \tau_s(n-1)\right] \text{divisions} \end{array} \right) \tag{49}$$

f_o is the ratio of the CPU time required for an entire iteration to the CPU time required for the matrix solution process alone. f_o is approximately 1.0 for all problems examined in the course of this study. This demonstrates that the bulk of the computational effort is in the solution by LU decomposition, as opposed to time spent in Jacobian

evaluations, matrix formulation, boundary condition implementation, residual computation, etc.. Even for very coarse grids in CFD solutions, f_o was measured to be less than 1.1.

It can be seen from equation (49) that the work of a direct solver on a vector processor does not necessarily increase by a factor of 16 when the grid size is doubled. Instead, it increases by a variable factor between 8 and 16, and is often close to 8 for small to moderately sized problems. The factor of 16 is often quoted as one of the disadvantages of the banded solver. This is true only as β becomes very large or τ_s very small. Similarly, when the bandwidth is doubled, as occurs when changing from first to second-order differencing, the computational effort increases by a factor of 4 only for very large bandwidths. For most problems examined here the factor is closer to 2 and rarely exceeds 3.

It is often found efficient to reuse the LU decomposition elements on subsequent iterations. This is identical to the reuse strategy often implemented in iteration methods.[17] When this technique is used, only the forward and back substitution phases of the direct solver are performed. Time for a direct iteration with LU reuse,

$\tau_r^{N \to N+1}$, is approximated as:

$$\tau_r^{N \to N+1} = f'_o \left[\frac{1}{M_v}(2n\beta - \beta^2 - \beta) + 2\tau_s(n-1) \right]_{\substack{\text{linked} \\ \text{triad}}} \quad (50)$$

f'_o is the ratio of CPU time for a complete iteration with reused LU to the CPU time required for the forward and back substitutions alone. Unlike f_o, f'_o is not close to 1.0. It can vary between 5 and 20, depending on problem, problem size, etc.. However, it can be readily determined by a timing measurement.

A comparison of predicted computational time (using equation (49)) versus actual time for a single iteration with a full direct solution is shown in Table 2. Comparisons are made for varying mesh sizes. Actual computational time was measured using an available timing routine on the VPS-32. It was found necessary to increase the predicted CPU times by 20% to account for scalar do-loops and non-vectorizable if and assignment statements in the direct solution process itself. Agreement is good - the prediction analysis has been found to be accurate for problem sizes up to the machine memory limit.

For a complete direct solution process involving possible LU decomposition reuses, an estimate of the CPU time takes the form

Table 2: Estimated Vs. Measured Timings for Vectorized Direct Solution

Mesh	n	β	CPU sec estimated	CPU sec measured
33x17 (first order)	2304	72	.564	.61
33x17 (second order)	2304	144	1.37	1.56
85x41 (first order)	14114	168	10.73	11.71
85x41 (second order)	14114	336	30.72	32.31
31x57 (viscous)	6960	464	24.6	24.0
31x57 (viscous)	6960	240	8.78	8.63

$$\tau^{sol} = I_D \, \tau^{N \to N+1} + I_r \, \tau^{N \to N+1} \tag{51}$$

I_D is the number of iterations with full direct matrix inversion while I_r is the number of direct iterations with reuse of the LU decomposition. For many problems it is possible to perform only one full direct iteration ($I_D = 1$) once the transient phase is passed. This single full iteration is followed by all subsequent 'reuse' iterations until convergence is achieved. With this strategy, I_r has been observed to vary between 5 and 25 for most problems. However, a direct iteration with a LU decomposition reuse is computationally negligible in comparison to a full direct iteration, i.e. $\tau_r^{N \to N+1} << \tau^{N \to N+1}$ always. Hence, provided the CPU time spent in the transient region is relatively small, it is often possible to approximate the CPU time required for a solution as:

$$\tau^{sol} \approx \tau^{N \to N+1} \tag{52}$$

C. VLGS Timing

The VLGS method results in a block tri-diagonal (first-order) or penta-diagonal (second-order) system which must be triangularized and then solved by back substitution. The triangularization phase can be vectorized over the number of

lines in the domain (taken here to be J) while the back substitution process is scalar; hence the back substitution process accounts for much of the computational effort in a typical VLGS iteration. The number of total operations per line in the decomposition can be counted for this method[11] in a straight-forward fashion and are given here for the tri-diagonal scheme as

$$W_T = K\left[(14/3)m^3 - (3/2)m^2 + (5/6)m\right] . \tag{53}$$

m is the block size (4 for two-dimensional problems) and K is the number of control volumes on a vertical line (i.e. $K = \beta/m$ for first-order differencing).

For a single complete tri-diagonal VLGS decomposition process, an estimate of CPU time can be written as

$$\tau_T = W_T\left[\frac{(n/\beta)}{M_V} + \tau_s\right] \tag{54}$$

since $J = n/\beta$ for first-order differencing. It should be noted that the decomposition for the VLGS scheme is not nearly as clean from the standpoint of vector operations as is the direct banded solver. There are numerous multiple-operation statements, as well as many divisions and linked triads. However, the timing has been found to be estimated very well by using the machine parameters for separate

multiplications/additions in equation (54). This is because these operations make up the bulk of the operations present in the decomposition.

τ_T, then, is rewritten as follows:

$$\tau_T = \left[(14/3)m^2 - (3/2)m + (5/6)\right] \left((n/M_V) + \beta\tau_s\right) \quad (55)$$

In order to obtain the time for a complete VLGS iteration, a factor f_1 must be defined such that

$$f_1 = \frac{\text{CPU time (complete iteration)}}{\text{CPU time (decomposition)}} \quad (56)$$

Unlike the ratio f_o for the direct method, f_1 is not close to 1.0 - it has been observed to range from 4 to 11 for a range of problems. This indicates that the decomposition process requires only a fraction of the CPU time required for an iteration - the matrix evaluations, boundary conditions, residual update and the VLGS back substitution step require most of the effort.

The CPU time required by a single iteration with a tri-diagonal VLGS decomposition is expressed in terms of this factor as

$$\tau_T^{N \to N+1} = f_1 \left[(14/3)m^2 - (3/2)m + (5/6)\right] ((n/M_V) + \beta \tau_S) \tag{57}$$

For a solution process where the error (or residual) is reduced to some tolerance, an estimate of τ_T^{sol}, the computational time required, is given as

$$\tau_T^{sol} = I \; \tau_T^{N \to N+1} \tag{58}$$

or

$$\tau_T^{sol} = f_1 (-\nu / \log \rho) \left[(14/3)m^2 - (3/2)m + (5/6)\right] ((n/M_V) + \beta \tau_S) \tag{59}$$

As defined earlier, ν is the order-of-magnitude reduction in the residual and ρ is the asymptotic spectral radius of the scheme. τ_T^{sol} for the iteration method is seen to be far more empirical than the banded direct τ^{sol} insomuch as the parameters f_1 and ρ must be measured or estimated for a particular problem.

As noted in the previous section, a technique frequently used when solving a problem using an iteration method is that of reusing or 'freezing' the decomposition of the left-hand side matrix for some fixed number of iterations. With this technique the computationally

expensive Jacobian evaluations as well as the matrix decomposition itself is bypassed. Van Leer and Mulder[17] first used this method for CFD solutions. Originally, it was a tool used to accelerate root-finding for scalar non-linear equations. The computational time prediction (equation (58)) can be modified to include this strategy such that

$$\tau_T^{sol} = I_T \, \tau_T^{N \to N+1} + I_r \, \tau_{Tr}^{N \to N+1} \tag{60}$$

where I_T and I_r are the number of iterations without and with reuses, respectively, and similarly $\tau_T^{N \to N+1}$ and $\tau_{Tr}^{N \to N+1}$ are the CPU times required for a single iteration without and with reuse, respectively. Equation (59) can then be modified to include the reuse strategy by defining a scaled weighted-average ratio, f_2, such that

$$\tau_T^{sol} = f_1 f_2 \, (-\nu / \log \rho) \left[(14/3)m^2 - (3/2)m + (5/6)\right] ((n/M_v) + \beta \tau_s) \tag{61}$$

where

$$f_2 = \frac{I_T \, \tau_T^{N \to N+1} + I_r \, \tau_{Tr}^{N \to N+1}}{(I_T + I_r) \, \tau_T^{N \to N+1}} \tag{62}$$

(for no reuse, $f_2 = 1.0$). For large number of reuses ($I_r >> I_T$), only the second term in the numerator of equation (62) is significant, and f_2 approaches the simple ratio

$$f_2 \approx \frac{\tau_{Tr}^{N \to N+1}}{\tau_{T}^{N \to N+1}} \tag{63}$$

Note that equation (59) is a special case of (61) where $f_2 = 1.$ (no reuse).

This estimation of CPU time was developed for a block tri-diagonal system. For a block penta-diagonal system arising from consistent second-order differencing, an entirely identical estimation process is possible, except that the tri-diagonal operation count for a line, W_T, must be replaced by the penta-diagonal operation count, W_P, as follows:

$$W_P = K \left[(38/3)m^3 - (5/2)m^2 + (5/6)m \right] \tag{64}$$

This leads directly to the following expression for the CPU time required for a penta-diagonal (second-order) solution process:

$$\tau_P^{sol} = f_1 f_2 (-\nu / \log \rho) \left[(38/3)m^2 - (5/2)m + (5/6)\right] ((n/M_v) + (\beta/2) \tau_s) \quad (65)$$

A comparison of estimated versus actual CPU time for a variety of problems is shown in Table 3. Results are in good agreement with measured times. Parameters f_1 and f_2 and ρ were simply measured and input into the prediction analysis.

D. Comparison of VLGS and the Direct Banded Solver

It is now possible to compare relative work (CPU time) for the direct and the VLGS methods. Such a comparison will serve as an indicator as to when a vectorized direct solver may be feasible.

Figure 3 is a representative result of such a comparison. A family of lines is shown for various matrix dimensions, n. These lines give the estimated CPU time as a function of the matrix bandwidth for one iteration with a direct solver. A second family of curves corresponding to the VLGS method without the reuse strategy is also plotted. The work required for the iteration scheme was computed using the assumption that one full Newton (direct) iteration (with subsequent 'reuses') will decay the residual ten orders-of-magnitude - so that the CPU time shown for VLGS

Table 3: Estimated Vs. Measured Timings for VLGS - (six order-of-magnitude residual reduction)

Mesh	n	β	f_1	f_2	ρ	CPU (sec) estimated	CPU (sec) measured
33x17 (first order)	2304	72	6.6	1.0	.913	6.58	6.6
33x17 (second order) first order LHS	2304	72	7.0	1.0	.930	8.79	8.8
85x41 (first order)	14114	168	9.4	.57	.966	46.4	48.2
85x41 (second order) first order LHS	14114	168	10.4	1.0	.981	162.5	163.1
31x57 (second order) viscous	6960	464	4.1	.46	.952	31.5	32.0
61x113	27360	912	5.1	.48	.982	268.2	284.6

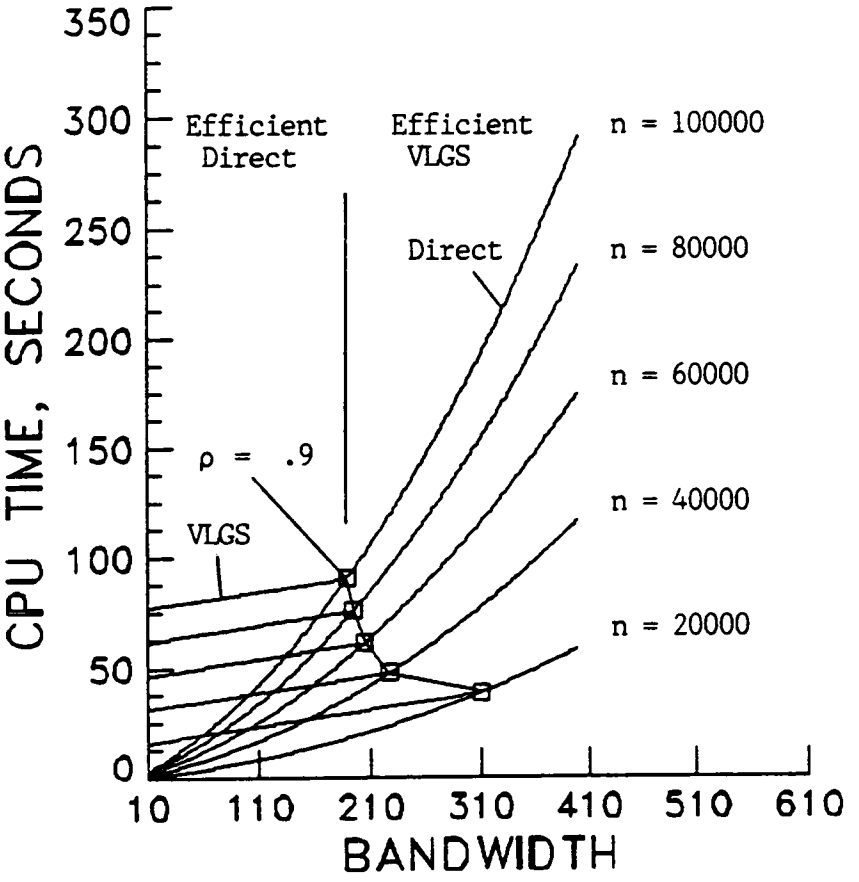


Figure 3. Work Curves for Direct and VLGS (ρ =.9), ν = 9, Tri-diagonal, f_1=5., f_2=1.

corresponds to a residual decrease of ten orders-of-magnitude.

The intersections of the two families of lines in Figure 3 represent the bandwidths where the direct and the VLGS methods yield identical work estimates for a given n. The line of these intersections for a specified spectral radius of the iteration scheme provides an interface between problems which are more efficiently solved using the direct method and those that are more efficiently solved using VLGS. The iterative process is more efficient when matrix parameters n and β lie to the right and above the spectral radius line.

Figure 4 is a similar result for three spectral radii for tri-diagonal VLGS without the reuse strategy and with $f_1 = 5.0$. For purposes of clarity, the curves for the VLGS method have not been shown. Plainly, as the spectral radius increases, holding all other parameters constant, the problem size increases for which the direct method is more efficient than the VLGS scheme. Also, an increase in bandwidth, β , holding n constant, is seen to increase the relative computational cost of the direct method as compared to VLGS.

Figure 5 shows a family of curves for various values of the factor f_1 for the special case of a square domain with

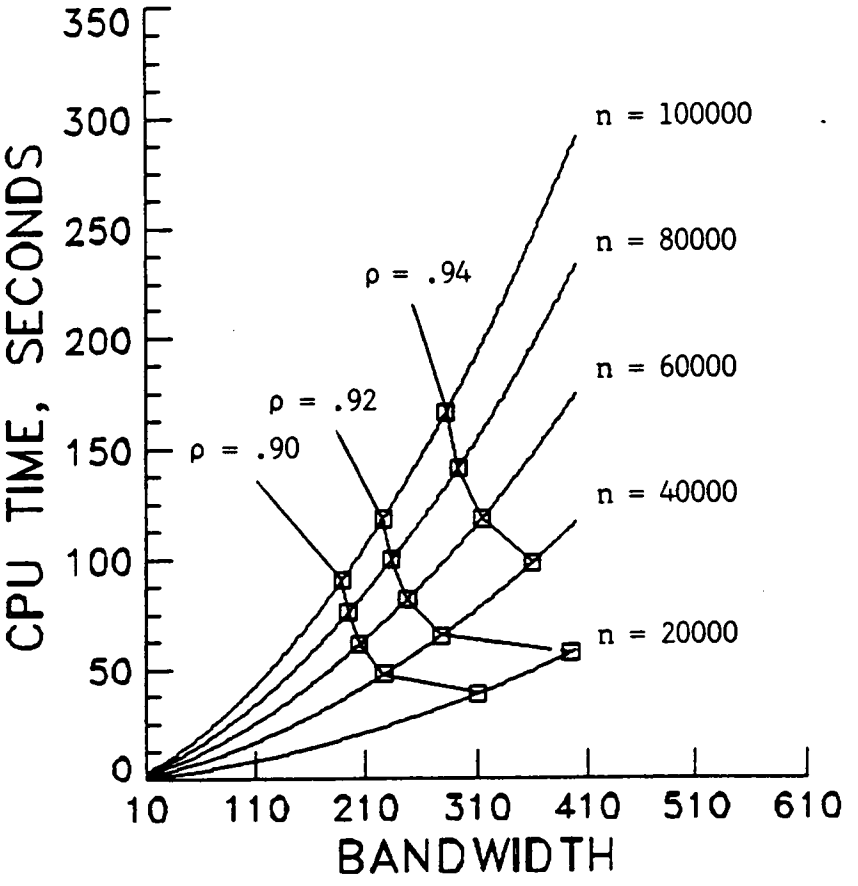


Figure 4. Equal Work Curves for Direct and VLGS, $\nu = 9$, Tri-diagonal, $f_1=5.$, $f_2=1.$

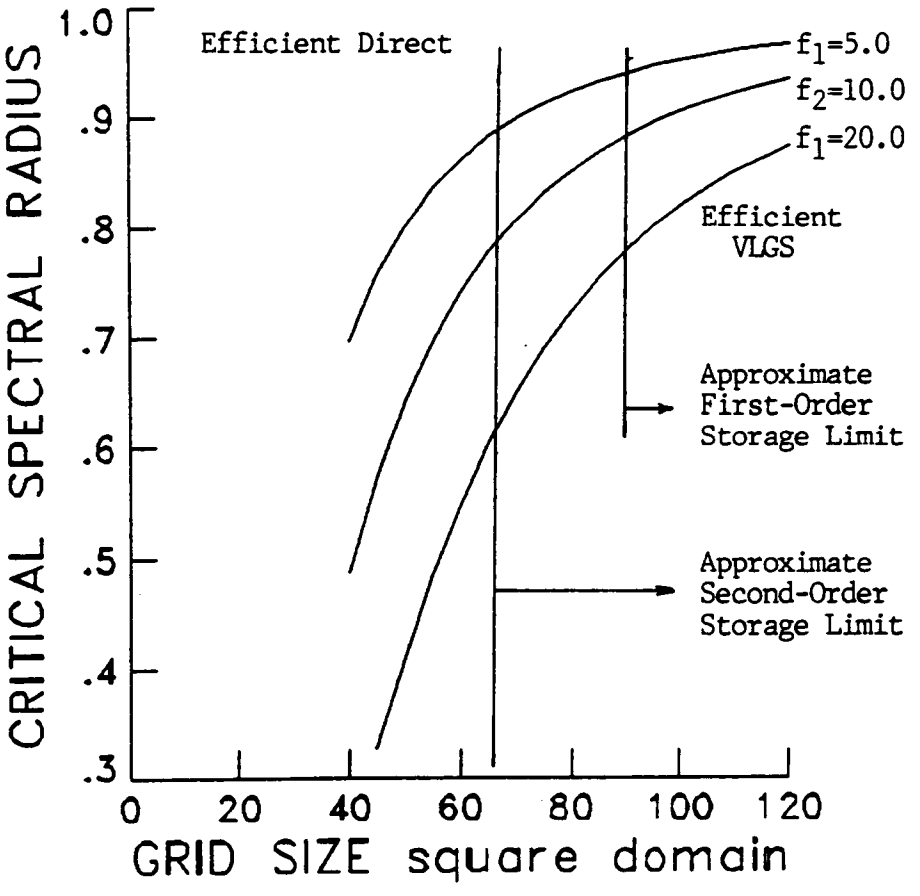


Figure 5. Critical Spectral Radius vs Grid Size (Square Domain) Tri-diagonal, ν =9, f_2=1.

an equal number of cells in both directions (J=K). This represents a worst-case scenario for the direct method since ordinarily one would have a smaller number of grid points in one direction, providing a natural minimum bandwidth for both storage and CPU time considerations. Again, one full direct iteration is assumed to drop the residual ten orders-of-magnitude. The ordinate gives the critical spectral radius, ρ_c, for which the direct and the iteration methods yield identical computational work, and the abscissa is the mesh size ranging from 20x20 to 120x120. For a given square mesh size and ratio f_1, it is possible to simply read the critical spectral radius, ρ_c, and note that for any $\rho < \rho_c$, VLGS will be more efficient than direct inversion, while for any $\rho > \rho_c$, the direct method will be more efficient than VLGS. As can be seen, the region in which the direct method is more efficient is quite large. Also indicated on this figure are the approximate machine memory limits required with use of the direct solver for both first and second-order differencing. These lines show that, for many problems, machine memory, not computational time, is the limiting factor when considering the relative efficiency of the direct method.

There are cases where it may not be feasible to assume that one full direct iteration followed by subsequent reuses

will drop the residual 9 or 10 orders-of-magnitude. In these cases, I_D in equation (51) will not be equal to 1. (Once clearly into the region of asymptotic convergence, however, it has never been observed in this study that $I_D > 1$.) Also, most workers are not interested in dropping the residual any more than 4 or 5 orders-of-magnitude - in fact, needs rarely demand such accuracy. Then it is instructive to examine the computational time required for one full direct iteration versus that for a five order-of-magnitude drop in the residual by the VLGS method. This shifts the work curves for VLGS down by a factor of two and decreases the region of efficient direct solution. A plot of equal work curves for various ρ for such a case is shown in Figure 6. Figure 7 shows a family of curves for various f_1 for ρ_c versus grid size (square domain). Although the range of n and β where the direct method is clearly more efficient is diminished, there is still a wide range of problems where the direct method is faster than VLGS.

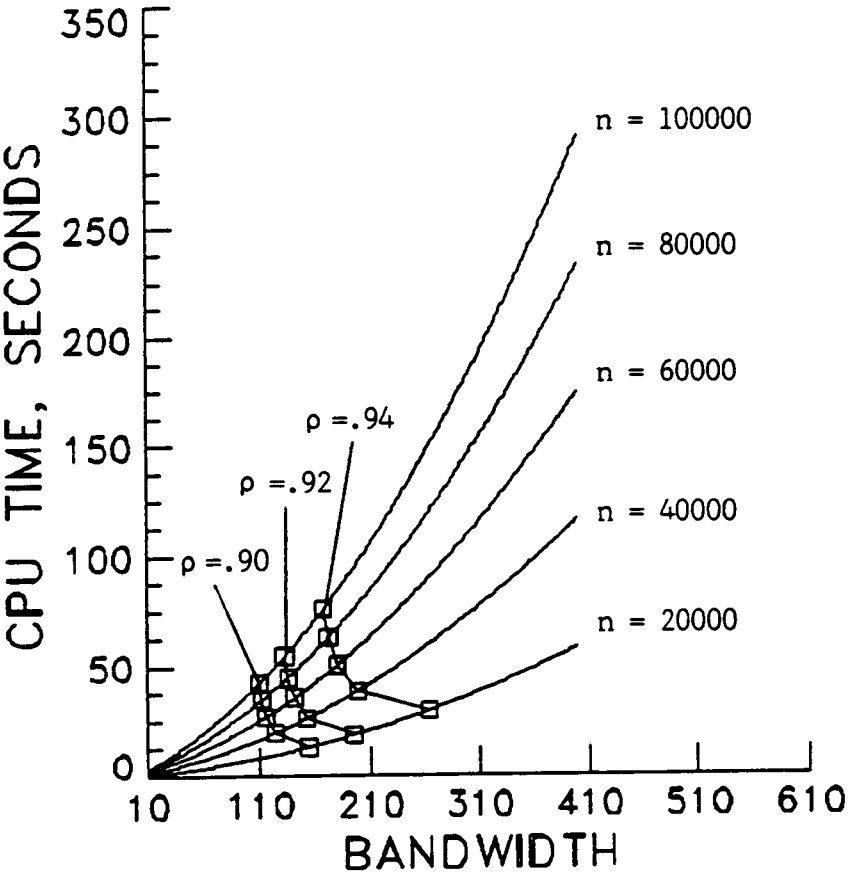


Figure 6. Equal Work Curves for Direct and VLGS, ν =5, Tri-diagonal, f_1=5., f_2=1.

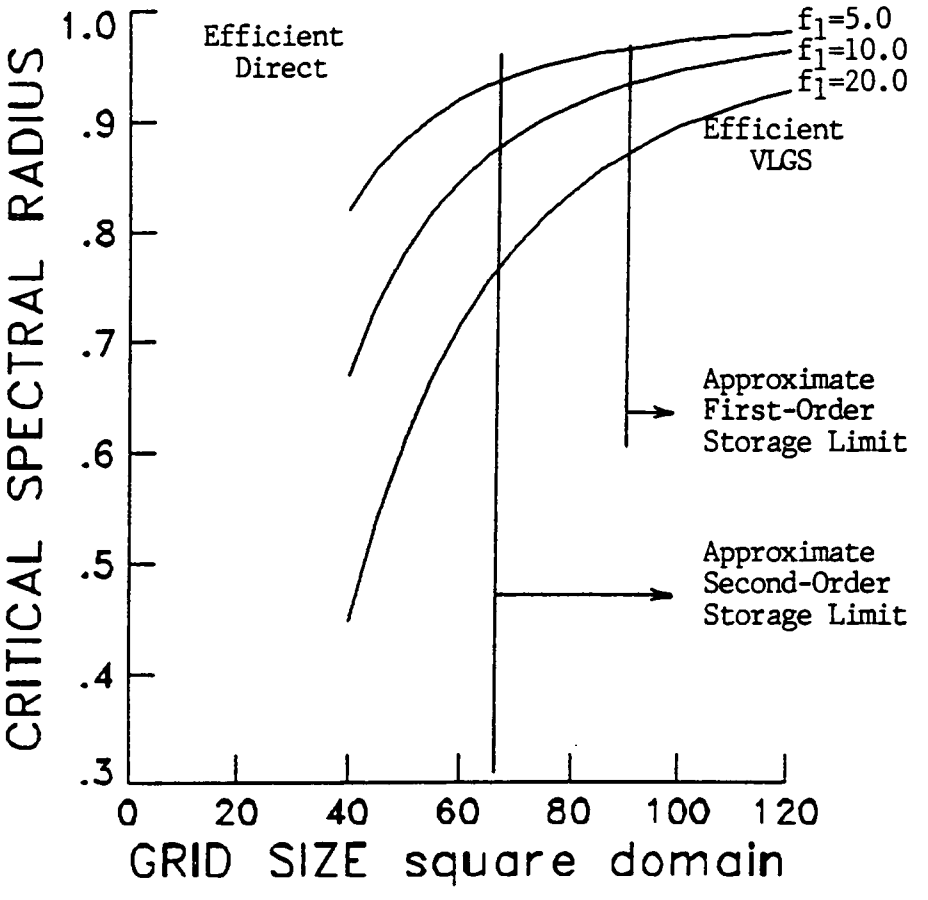


Figure 7. Critical Spectral Radius vs Grid Size (Square Domain) Tri-diagonal, $\nu=5$, $f_2=1$.

VII. MESH-RELATED CONVERGENCE ACCELERATION TECHNIQUES

A. Multi-Grid

Multi-grid strategies are, in general, very effective for reducing the CPU time required to obtain solutions to the governing equations of fluid mechanics, although the method was originally developed for use in solving elliptic boundary value problems.[18,19] Currently, the full approximation storage scheme (FAS) has been implemented in a two dimensional Euler/Navier-Stokes code for the purpose of studying the feasibility of using a direct solver on coarse grids while using a relaxation or other iteration method on the finer grids where the direct method cannot be used due to memory requirements. The advantages of such an approach are 1) efficient and thorough damping of fine-grid low frequency errors by solutions obtained on the coarser grids, 2) low storage requirements for the direct method since only coarse grids are used with the direct solver and 3) a reduction in overall CPU time for a converged solution of the compressible flow equations.

The multi-grid process uses a sequence of grids G_1, G_2, ...G_p, where G_1 represents the finest mesh and G_p is the coarsest mesh with intermediate grid levels G_2, G_3, etc.. A coarse mesh is obtained by deleting every other mesh line in

the domain of the fine mesh. The advantage of the method lies in its ability to smooth the solution on the coarse meshes and pass this information as a correction back up to the finer grid. As a result, the asymptotic convergence rate of the iteration method used on the fine mesh is improved, with a net savings in work performed. The method as programmed follows the development in Reference 12 and uses a standard V cycle or fixed cycling strategy where a fixed number of iterations are performed on each grid level. The number of sub-grids used in each 'fine' iteration can also be varied; this has been observed to have some effect on the rate of convergence.

The dependent (conserved) variables are transferred from fine to coarse grids (i to i+1) by the relationship:

$$Q_{i+1} = I_i^{i+1} Q_i \quad , \tag{66}$$

where I_i^{i+1} is a volume weighted average restriction operation defined by

$$I_i^{i+1} Q_i = \frac{\sum VQ}{\sum V} \quad . \tag{67}$$

The summations are taken over all fine grid cells in each coarse grid cell. This can be shown to conserve mass,

momentum and energy.[13]

Furthermore, the residual on the fine grid is transferred to the coarse grid such that

$$R_{i+1} = R_{i+1} [I_i^{i+1} Q_i] + P_{i+1} , \tag{68}$$

where P_{i+1} is a forcing function defined by

$$P_{i+1} = \hat{I}_i^{i+1} R_i - R_{i+1} (I_i^{i+1} Q_i) . \tag{69}$$

Notice that P_{i+1} will not change for subsequent iterations on any given grid level. P_{i+1} describes the relative truncation error of the coarse grid with respect to the fine grid, and, as noted in Reference 13, allows a progressing solution on the coarse grid to maintain fine-grid accuracy.

The operator $\hat{I}_i^{i+1}$ is the restriction operator for the residual, given by

$$\hat{I}_i^{i+1} R_i = \sum R_i . \tag{70}$$

This process is carried out on successively coarser grids $G_{2,3 \ldots p-1}$. Corrections on each grid level are then computed and passed back to the fine grid, G_1, using bilinear interpolation. No iteration steps are carried out

on intermediate grid levels when passing the correction to the fine mesh (for the V cycles used in the procedure).

B. Mesh-Sequencing

Like multi-grid and the LU decomposition reuse strategy, mesh-sequencing is a technique used to reduce CPU time required for convergence. The solution is obtained on a sequence of grids, coarse to fine, with each coarse grid solution simply providing a starting solution for the next (finer) grid. The advantage of mesh-sequencing is that the computational time spent in the transient region of the solution process on the fine grid can be significantly reduced. This transient region is very often time-consuming, and can lead to a large plateau in the CPU time versus residual history. This plateau is particularly large for Newton's method when the initial condition for a steady-state problem is specified to be free-stream. By obtaining the solution on a coarser grid where the direct solution can be readily obtained and interpolating the 'converged' solution to the fine grid, a great deal of initial transient work on the fine grid is avoided. Mesh-sequencing with direct solvers in conjunction with solutions for airfoil flowfields have been examined recently be Venkatakrishnan.[3]

For this investigation, the same linear 4-point

interpolations employed in the multi-grid process were used to move the solution vector from the coarse to the fine grid.

VIII. COMPUTATIONAL RESULTS

A. Transonic Channel Flow (Case 1)

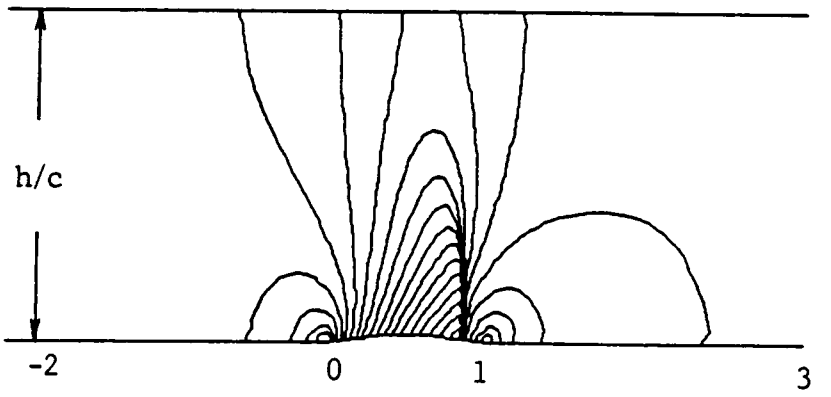
The relative convergence rates using the VLGS scheme and the vectorized direct method have been examined by applying them to both inviscid and viscous problems. The first case studied is that of transonic inviscid flow over a circular arc airfoil (t/c=.042) in a channel. This problem has been studied extensively by a number of investigators in the past.[11,20] Boundary conditions for the problem were imposed by specifying static pressure at the exit and stagnation pressure, stagnation enthalpy, and vertical velocity component (v=0) at the inlet. Flow tangency was enforced on the lower and upper boundaries. The inlet Mach number corresponded to $M_\infty = .85$. Pressure contours obtained on a 85x41 mesh with second-order accurate differencing are shown in Figure 8a. Shown in Figure 8b is a plot of C_p versus x/L. This distribution compares well with the results in Reference 20.

A plot of residual versus iteration with the direct method for a 33x17 mesh is shown in Figure 9. The direct method converges quadratically and displays no time step restriction ($\Delta t=10^{12}$ for this calculation). For this particular grid, it was efficient to begin with Newton's

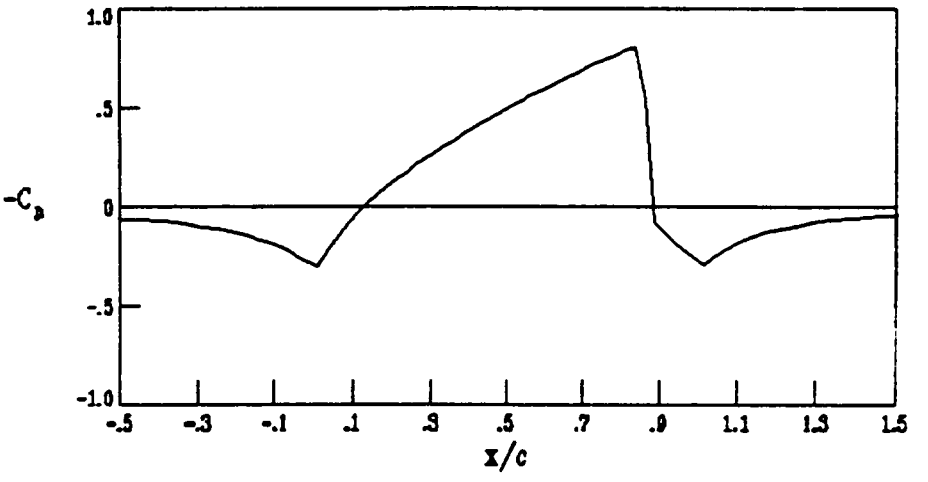
method, however, as the grid was refined, it was found advantageous to initially use VLGS until a two-order drop in the residual occurred. If this was not done, several extra direct iterations were needed resulting in a loss in efficiency. Figure 10 is a comparison of the methods versus CPU time. The VLGS method shown here used a varying time step based on the SER (switched evolution-relaxation) strategy[20] with a maximum Courant number of 10^2. The situation for the direct method can be significantly improved by the reuse of the LU decomposition of the left-hand side coefficient matrix as discussed in previous sections. This is exactly equivalent to the reuse strategy as implemented in iteration methods. It is possible to solve the identical problem with only one full direct iteration (after initial residual reduction) followed by iterations with frozen LU elements as shown in Figure 10. The CPU time required for each subsequent 'reuse' iteration is negligible relative to a complete Newton iteration and yet displays an impressive residual reduction per iteration. This represents the most efficient use of the direct solver for this case. The solution is presented for this coarse grid since it clearly demonstrates the feasibility of using a direct solver. Note that in a multi-grid context, where such coarse grids may be often used, the direct solver may

be preferable for rapid convergence on some grid levels. The transonic case over the circular-arc airfoil was studied further by using a finer (85x41) mesh. With initial smoothing by VLGS, the direct solver can be used with very large time steps. In Figure 11, for the first-order case, the residual has been reduced almost nine orders-of-magnitude in about 33 seconds with the final three full direct iterations.

A plot of residual versus CPU time for the second-order accurate results on the 33x17 mesh is shown in Figure 12. The direct method can be called fast (even with the doubling of the matrix bandwidth) and can be implemented with the reuse strategy as shown. The fully converged solution is obtained in less than five seconds, with only one full direct iteration necessary. With this iteration strategy, it can be seen from the figure that approximately 60% of the time was associated with the VLGS iterations and 40% with the direct solver. Figure 13 shows a similar reuse strategy for the 85x41 mesh with second-order differencing. Here the banded solver took roughly 60% of the computational time, the remainder taken up by the initial VLGS iterations. Although not shown here, quadratic convergence using full direct solutions was obtained for the 85x41 mesh with second-order accuracy. Obtaining quadratic convergence is



(a) Pressure Contours



(b) Wall Pressure Distribution

Figure 8. Transonic Channel Flow
Over a Circular-Arc Airfoil
M_∞ =.85, t/c=.04, 85x41 Mesh

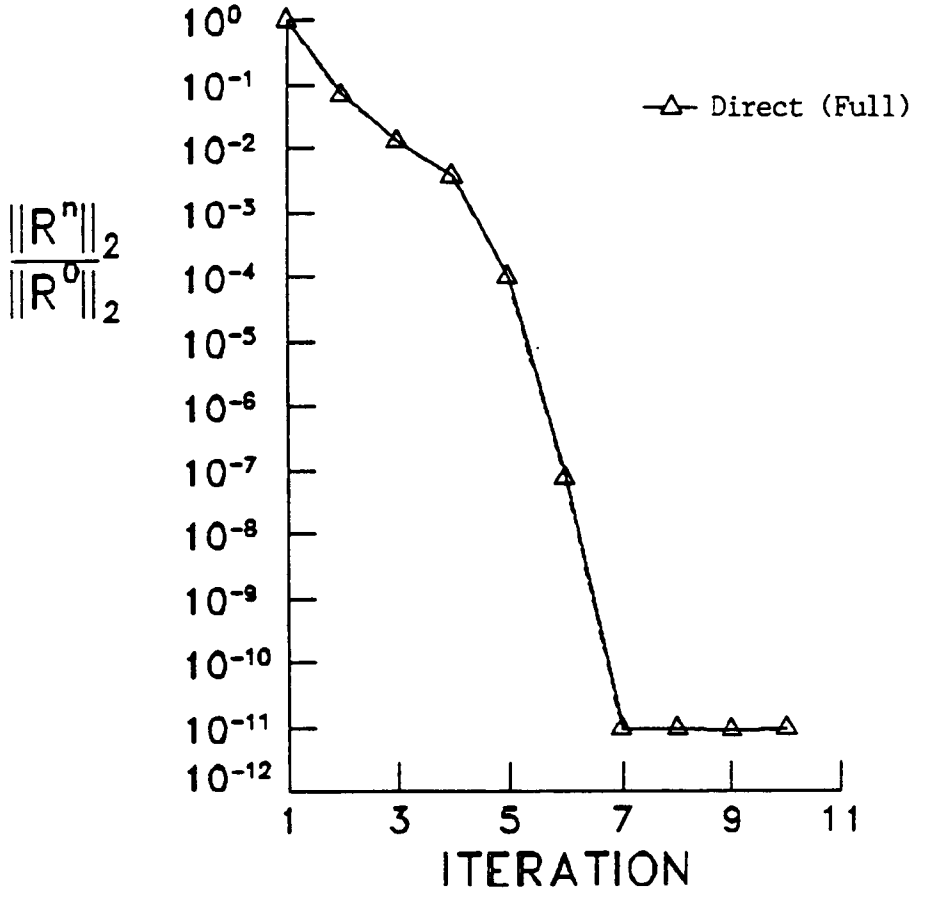


Figure 9. Transonic Channel Flow:
Residual vs Iteration
First-order, 33x17 Grid

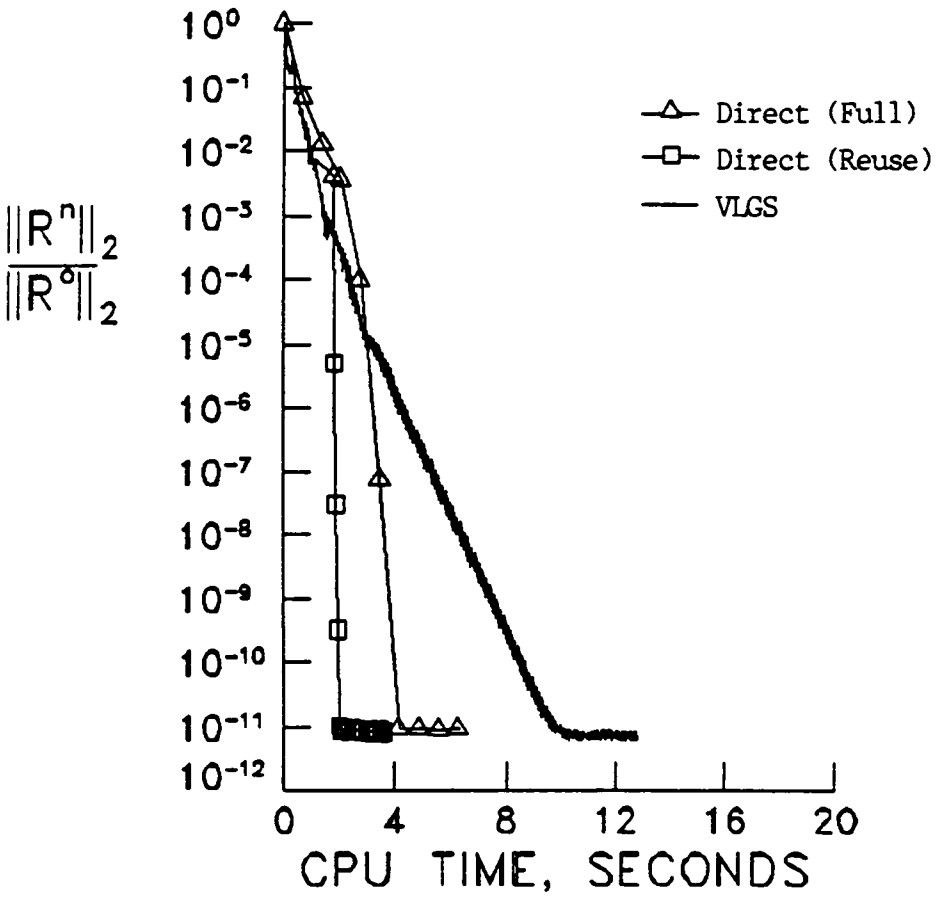


Figure 10. Transonic channel Flow:
Residual vs CPU Time
First-Order, 33x17 Grid

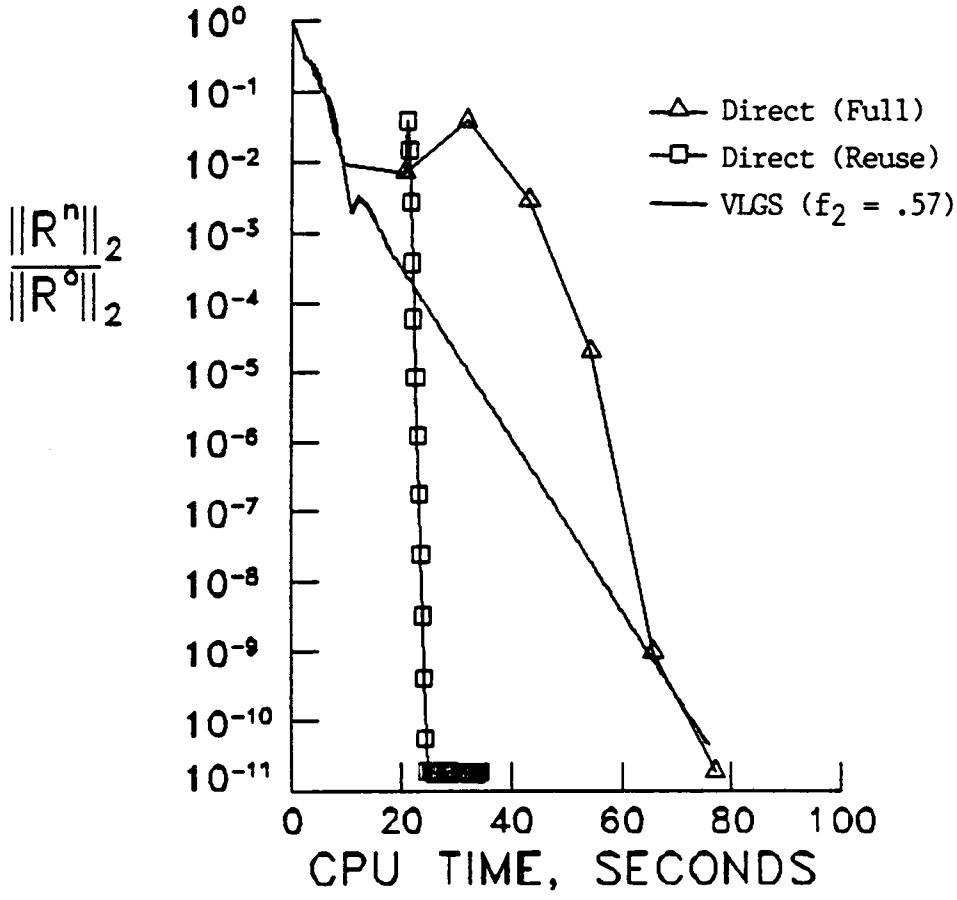


Figure 11. Transonic Channel Flow:
Residual vs CPU Time
First-Order, 85x41 Grid

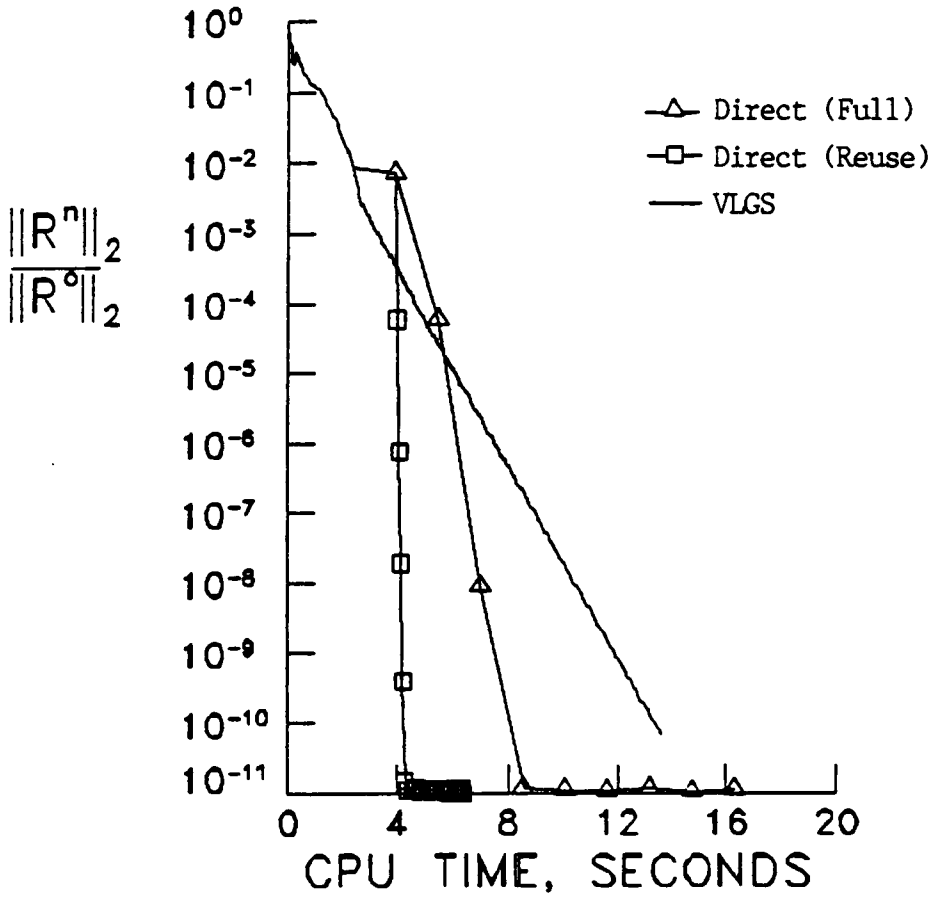


Figure 12. Transonic Channel Flow:
Residual vs CPU Time
Second-Order, 33x17 Grid

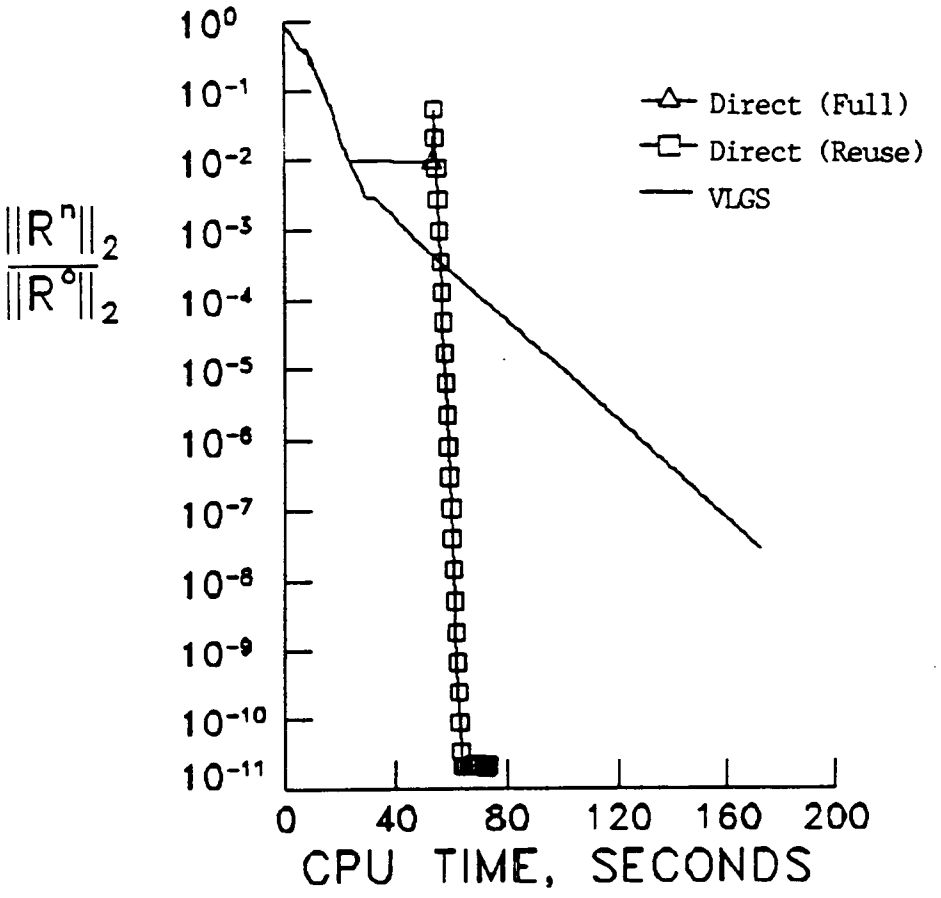


Figure 13. Transonic Channel Flow:
Residual vs CPU Time
Second-Order, 85x41 Grid

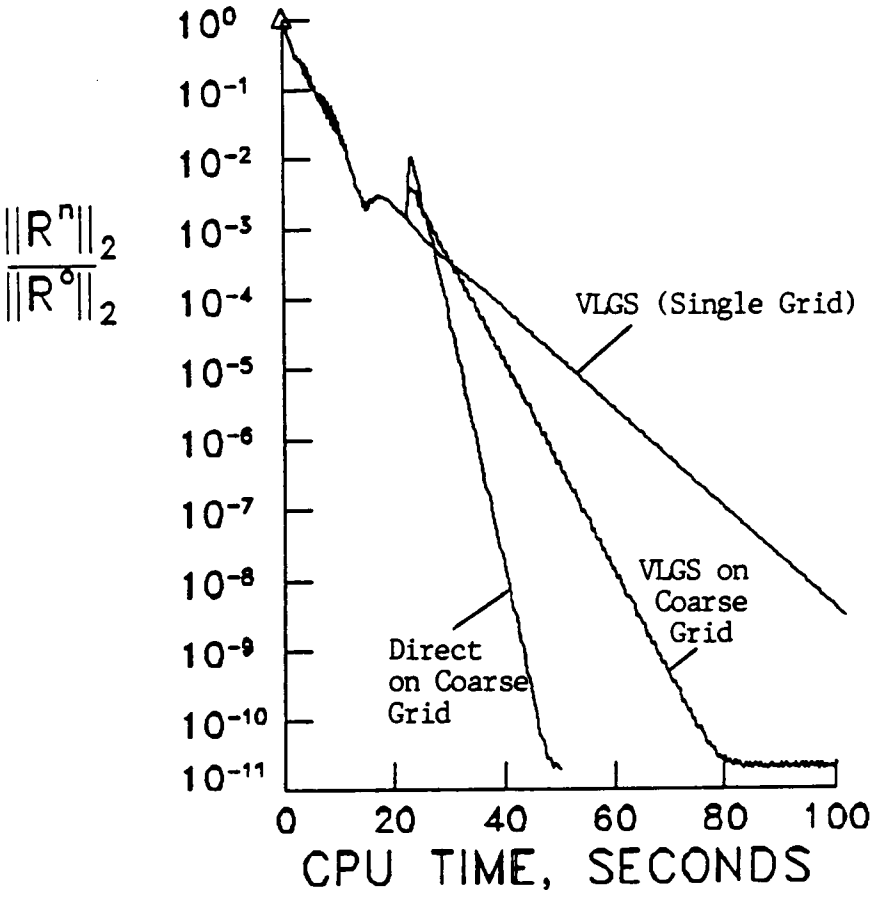


Figure 14. Transonic Channel Flow:
Residual vs CPU Time
Multi-grid, 85x41 Grid

not an efficient strategy with this problem.

Figure 14 is a timing comparison on the 85x41 grid using a three level multi-grid strategy in which a) VLGS iterations were used on all grids and b) VLGS was used on the fine and medium grids and the direct method was used on the coarse grid. One full direct iteration with $\Delta t=10^{12}$ was used on the coarse grid. The multi-grid method with the coarse-grid direct solver is seen to be fast and is easy to implement; performing more than one direct iteration on the coarse grid has been found not to be efficient.

B. Shock-Boundary Layer Interaction (Case 2)

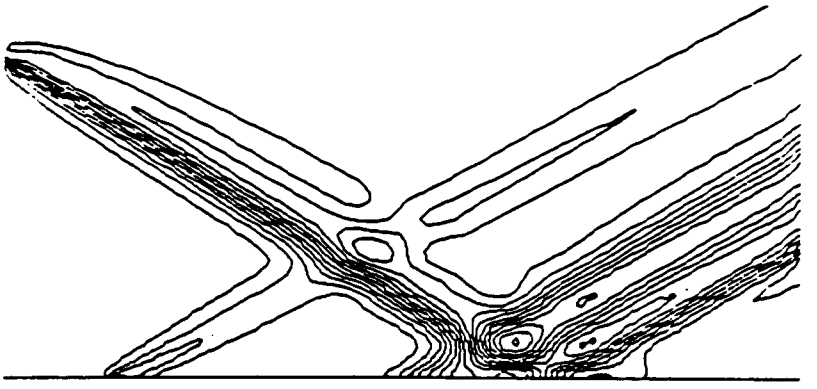
The second case examined in this study was a shock and laminar boundary layer interaction problem. The resulting flow is relatively complex, with separation and reattachment. The freestream Mach number is 2, the shock angle is 32.6 degrees, and the Reynolds number based on freestream velocity and the distance between the leading edge of the plate and the shock reflection point is 2.96×10^5. This problem has been examined by several investigators.[21,22,26,27] For the convective and pressure terms, third-order accurate differencing was used in the streamwise direction. Viscous terms were treated with second-order accurate central differences. Boundary

conditions were specified at the inflow and upper boundaries corresponding to reference and post-shock conditions, no-slip was enforced on the plate, flow tangency imposed before the leading edge of the plate, and second-order extrapolation was used for the supersonic outflow boundary. Except as noted, the viscosity varied with the temperature as given by Sutherland's Law. Pressure contours and the skin friction distribution on the plate obtained on a 61x113 mesh are shown in Figures 15a and 15b. These results compare well with experimental results for this case obtained by Hakkinen, et al.[22] They used a Stanton tube assembly to estimate wall friction. The black circles in 15b indicate experimentally determined separated flow (experimental measurements were not accurate in the separated region itself).

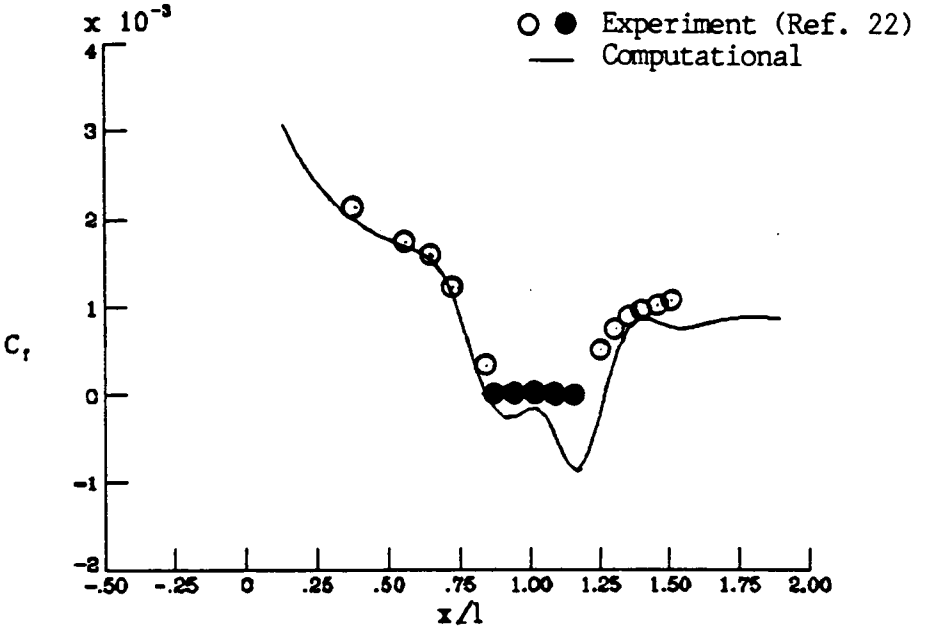
Figure 16 is a comparison of the residual versus iteration (for a 31x57 grid) for the direct solver when the viscosity is both held constant at the freestream value and allowed to vary according to Sutherland's Law. The figure shows that the direct method converges quadratically with μ = constant and converges linearly when it varies with the temperature of the discrete equations. Linearization of the viscosity in the implicit left-hand side was not performed for this work and hence quadratic convergence cannot be

expected when Sutherland's Law is used. Since it has been found that only one full direct matrix solution is necessary for a converged solution, obtaining quadratic convergence using all full direct (no 'reuse') iterations is not necessary or efficient. Figure 17 is a residual versus CPU time plot for the direct and the VLGS methods on the 31x57 mesh. The VLGS scheme is optimized by using the reuse strategy (11 reuses to each full VLGS iteration) and is superior to the direct method without reuses. However, the direct strategy when supplemented by subsequent reuse iterations is faster. It was found more efficient for this problem to initially drop the residual using VLGS three orders-of-magnitude rather than two, as in the previous problem. Figure 18 shows the residual comparison for these cases versus dollars, based on the VPS-32 charging algorithm at NASA Langley as of October 1987. For this problem, the direct method, because of its increased memory requirements, gets billed at a rate roughly 25% greater than that of the iteration scheme for each CPU second of execution time. Nevertheless, by using frozen LU elements, the direct method is still efficient in terms of dollars, in getting to the steady-state. Note that if one seeks only a three or four order-of-magnitude residual reduction, the iteration scheme is more efficient.

Figure 19 shows three level multi-grid results for a finer grid of 61x113. The spectral radius for the single-grid iteration solution (without reuse) is $\rho = .98$. Also shown are the results obtained by using VLGS on the coarse grid as compared to using a single direct iteration on the coarse grid. In this case, using the direct solver on the coarse grid is only marginally faster but, nonetheless, both simple and efficient.



(a) Pressure Contours



(b) Skin Friction Distribution

Figure 15. Shock-Boundary Layer Interaction

M =2.0, 61x113 Grid, Re=2.96x10^5

Laminar Flow

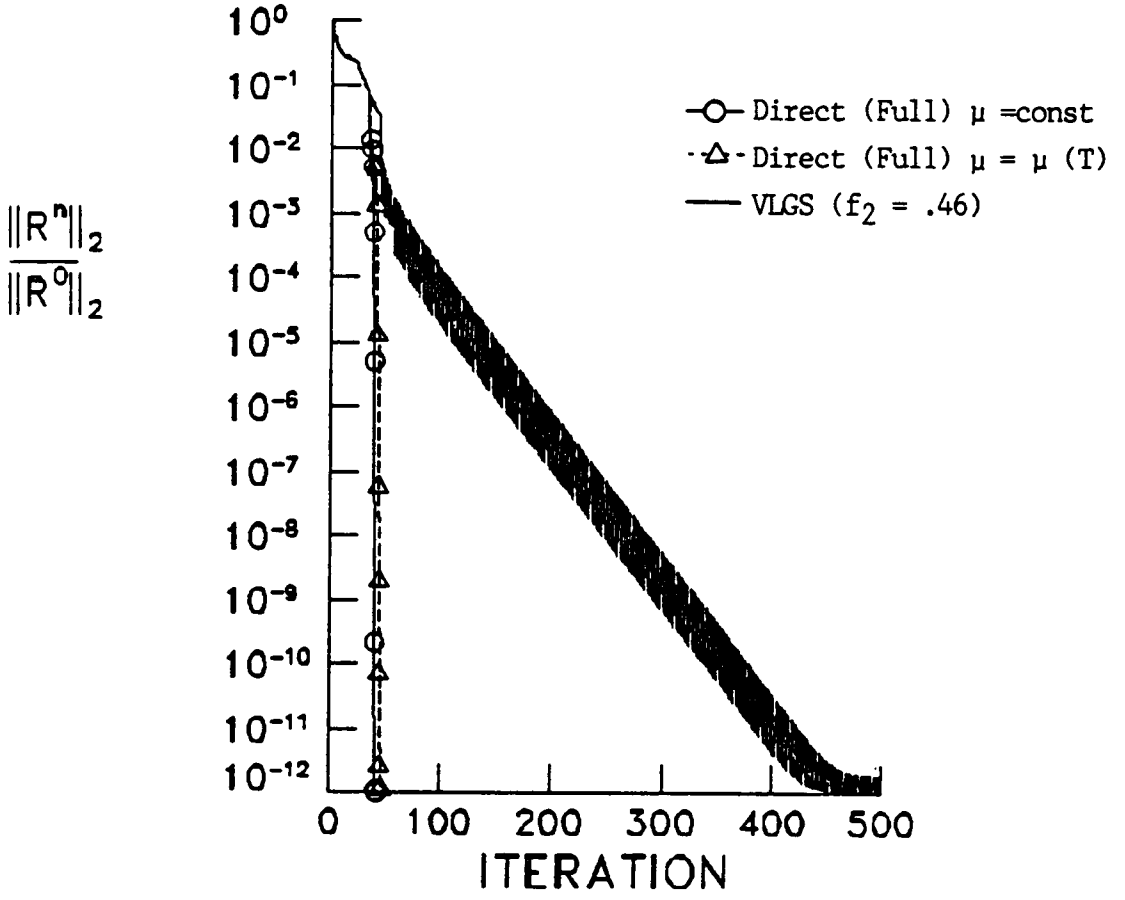


Figure 16. Shock-Boundary Layer Interaction: Residual vs Iteration, 31x57 Grid, Constant and Varying μ

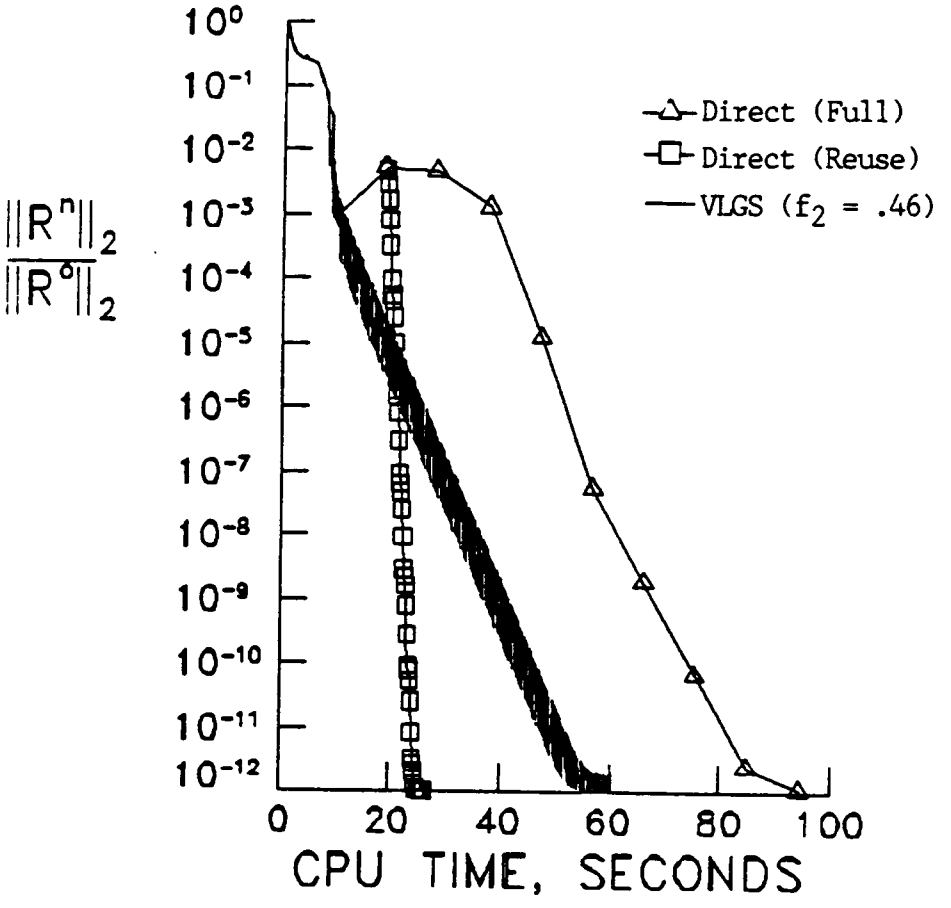


Figure 17. Shock-Boundary layer Interaction: Residual vs CPU Time, 31x57 Grid

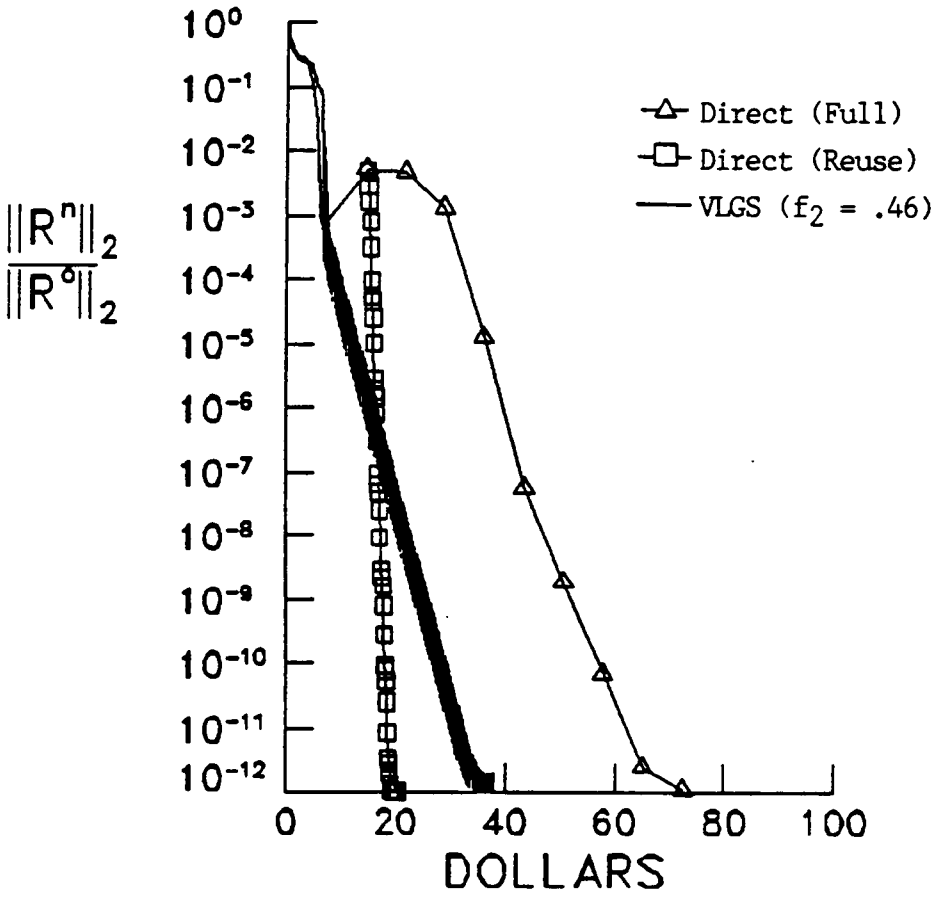


Figure 18. Shock-Boundary Layer Interaction: Residual vs Dollars, 31x57 Grid

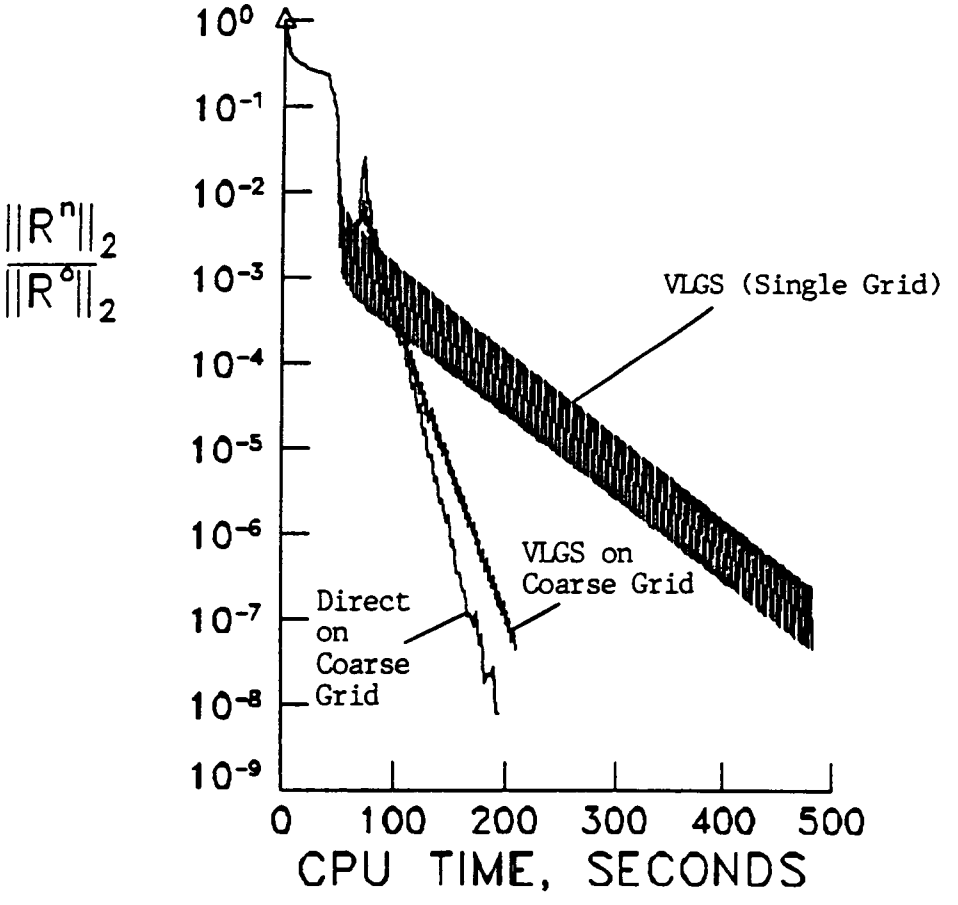


Figure 19. Shock-Boundary Layer Interaction: Residual vs CPU Time, Multi-grid 61x113 Grid

C. NACA 0012 Airfoil Results - Mach and Reynolds Number Variations (Case 3)

The final test problem examined was that of laminar subsonic and transonic viscous flow over a NACA 0012 airfoil at zero angle of attack ($\alpha = 0$). This problem has been recently studied by Venkatakrishnan.[3] He has used direct solvers and sparse matrix technology to investigate this problem at $M_\infty = .5$ and for Reynolds numbers (Re) of 5000 and 10000. The NACA 0012 distribution was obtained from Reference 24. Boundary conditions were no-slip on the airfoil and tangency both before and after the airfoil (on the zero-line). On the far-field boundary, the stagnation enthalpy was held constant while density and velocity were extrapolated. For the subsonic inflow, stagnation pressure, stagnation enthalpy and vertical velocity were specified and the pressure from the interior was extrapolated; for the outflow, the back pressure was specified and velocity and density were extrapolated. The inflow boundary was located at three chord lengths upstream from the leading edge of the airfoil, the outflow boundary at three chord lengths from the trailing edge and the far-field boundary at eight chord lengths above the foil. Since $\alpha = 0$, a standard 81x45 H-grid was used with grid clustering near the airfoil in both

streamwise and crossflow directions (Figure 20). Although the H-grid used is inferior to a C-grid, particularly near the leading edge, it was simple to implement, involved no wake line complications and was considered to be satisfactory for the purpose of convergence studies. It should be emphasized, however, that the actual solution quality obtained is less than satisfactory for some of the range of Mach and Reynolds numbers under consideration. This is due to both the grid type and the relatively coarse grid spacing (particularly in the η direction). The skin friction distribution for M_∞ =.5 and Reynolds numbers of 5000 and 10000 are shown in Figure 21. These results compare well with those obtained by Venkatakrishnan using a finer C type grid.[3] As can be seen, the grid at the leading edge causes a significant problem. However, the features of the solution for M_∞ =.5 are present; separation occurs on the airfoil at a location (in chord lengths) of 18% from the trailing edge while the flow reattaches at 14% from the trailing edge for Re=5000. Separation occurs at 26% from the trailing edge and reattachment at 24% from the trailing edge for Re = 10000. This agrees well with the results in Reference 3 (although the Re=10000 results are poor for both Reference 3 and this work due to inadequate mesh resolution near the airfoil).

For all NACA 0012 airfoil results, mesh-sequencing using the direct solver was used instead of VLGS for initial (transient) residual reduction. This was found to be much faster than using VLGS which has extreme time-step limitations and a very poor convergence rate for this problem - particularly for high Re or very low Mach numbers. (See Figure 22.) The mesh-sequencing strategy (where solutions are found on coarser grids) is the reason that all CPU time versus residual plots shown for this problem begin with a non-zero CPU time. Time for a converged coarse-grid solution needed to initialize the fine-grid process is relatively uniform at between 65 and 85 seconds. For all cases with mesh-sequencing, the fine grid time step could be taken as infinity ($\Delta t = 10^{12}$). It should be noted that (with mesh-sequencing) the coarse grid solution introduces high frequency errors into the fine grid solution process and hence requires more than one full direct iteration in the fine grid solution process. This is in contrast to the previous cases where with initial VLGS residual reduction only one full direct solution is required. However, for this problem, mesh-sequencing is far more efficient than using VLGS for initial residual reduction due to the very poor VLGS convergence in the transient region.

A series of convergence plots are shown for the

transonic case of M_∞ =.8 in Figure 23 for Reynolds numbers of 5000,10000,20000,and 40000. Various strategies of reusing the LU decomposition are shown. Some increase in time for convergence is observed as the Reynolds number is increased but, in general, the CPU time required for convergence varies only slightly. This case (M_∞ =.8) has a very small supersonic region developing in the flow. Δy_{min} for all cases was chosen such that $\Delta y_{min} < .7/\sqrt{Re}$ where Re = 40000, in order to keep some semblance of definition in the boundary layer for extreme cases. Figures 24 and 25 show convergence histories for a similar range of Re for M_∞ =.5 and M_∞ = .05, respectively. It is interesting to note that there is no apparent effect on convergence for increasing Reynolds numbers for cases where the flow is entirely subsonic - in fact there is evidently some slight decrease in CPU time. It should also be noted that the quality of the actual solution for M_∞ =.05 was very poor; it is believed that the lack of resolution at the airfoil leading edge causes a deterioration through the entire flow field for this strongly downstream-influenced case. However, convergence is quite rapid when using the direct solver while the VLGS scheme has an extremely poor asymptotic convergence rate for M_∞ = .05. Table 4 summarizes overall CPU times, time spent in mesh-sequencing

(all direct solutions on coarse grids) and number of full direct iterations required for all cases.

The direct method is seen to be robust over the tested range for the NACA airfoil and does not require an increased amount of CPU time for high Reynolds numbers except for transonic flow regimes. Even for transonic problems, the time required is not severely affected by Reynolds number variations. For this problem, mesh-sequencing using a coarse-grid direct solver is an extremely effective way to 'bypass' the fine-grid transient region and initially lower the residual for input as the starting fine-grid solution. This has been found to be true in general for cases where iteration methods perform poorly in the transient region. Thus mesh-sequencing can provide an efficient alternative to the use of the hybrid iteration-direct with reuse strategy as discussed for the transonic channel flow and the shock-boundary layer interaction problems. For the M_∞ =.8 (transonic) case, CPU times for high Reynolds numbers (i.e. Re=40000) can be dramatically reduced by using a low Re solution (i.e. Re=5000) as an initial (starting) solution. Convergence can then be obtained with one full direct iteration. In addition, the mesh-sequencing CPU times for all cases can be substantially reduced - no effort was made in this work to optimize the coarse-grid solution process.

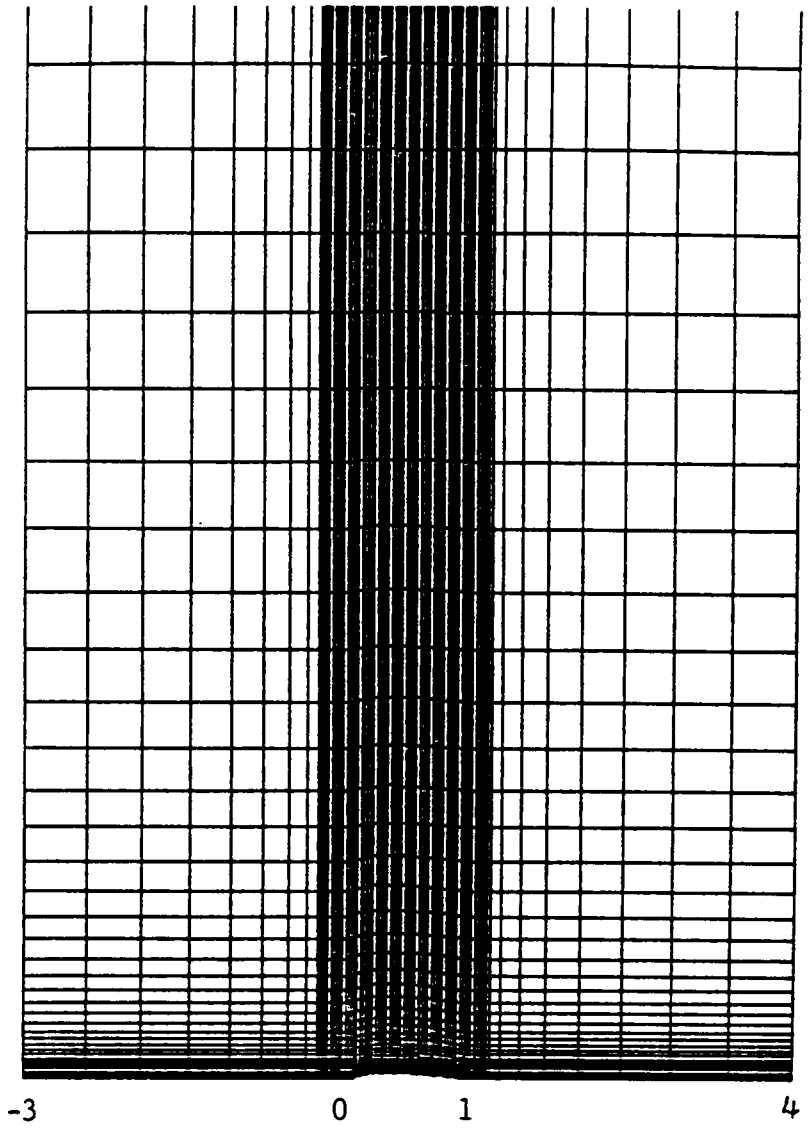
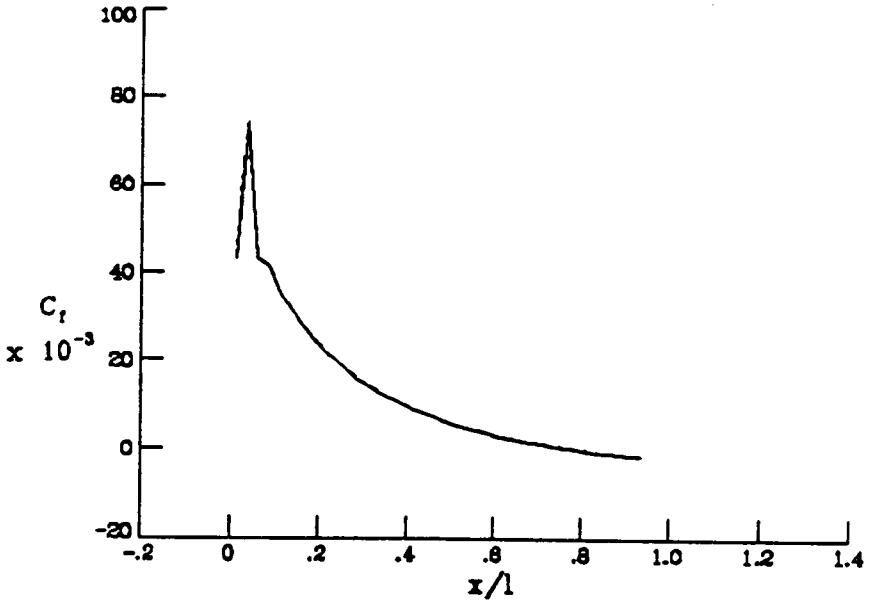
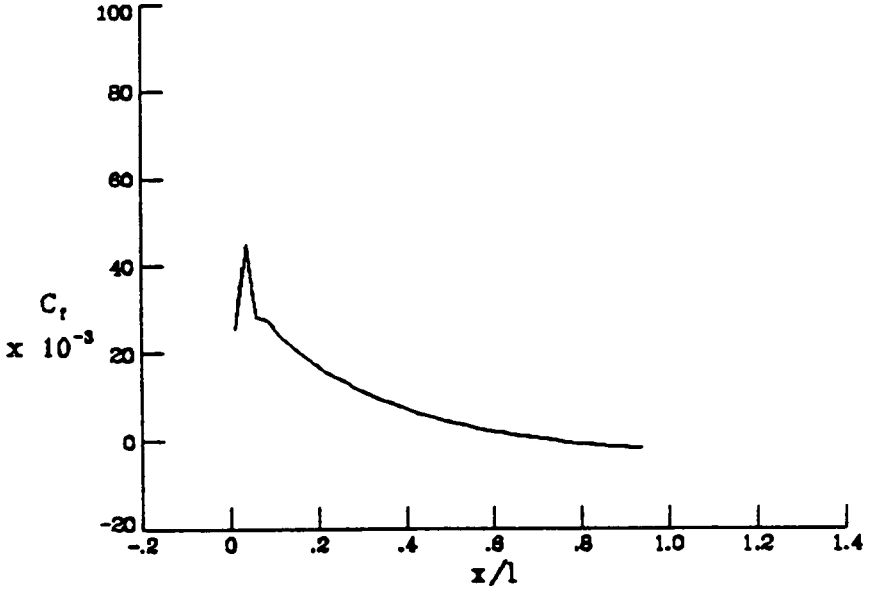


Figure 20. Grid for NACA 0012 Airfoil

(81x45 Grid)



(a) Re = 5000



(b) Re = 10000

Figure 21. NACA 0012 Airfoil: Skin Friction Distribution, M_∞ =.5 81x45 Grid

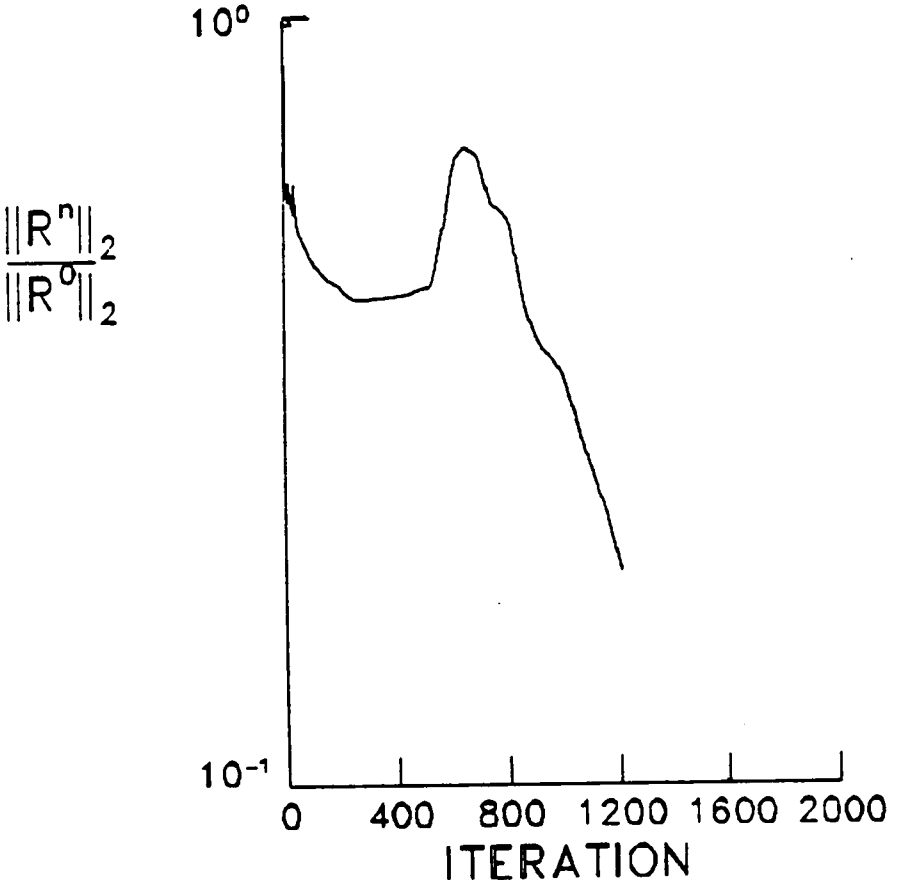


Figure 22. NACA 0012 Airfoil, Residual vs Iteration for VLGS $M_\infty = .5$, Re = 5000

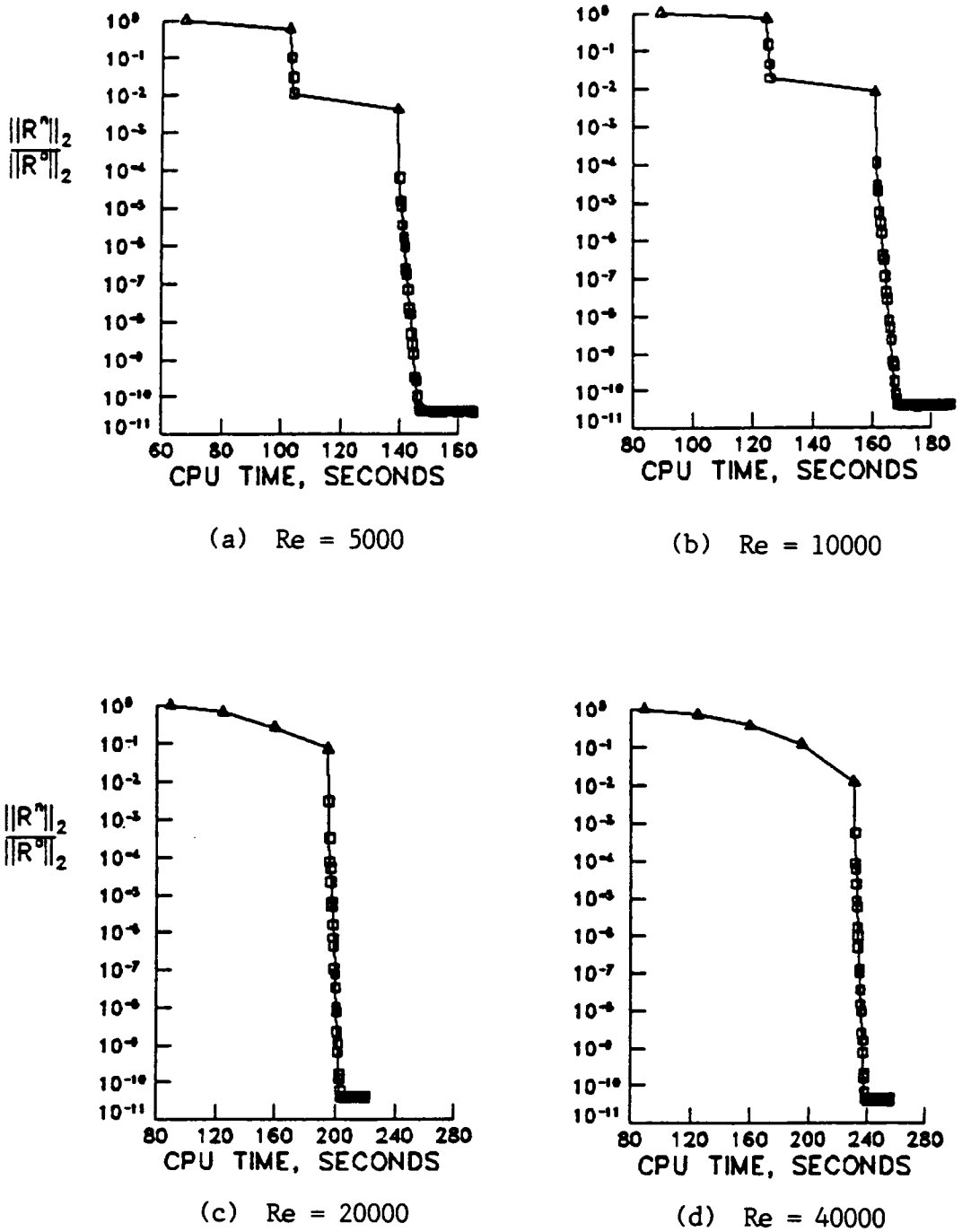
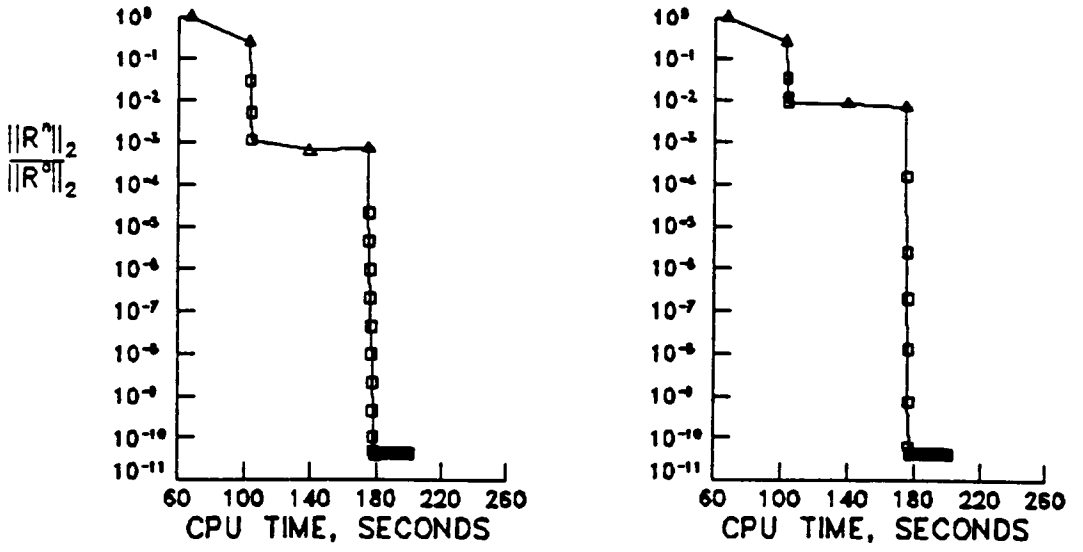
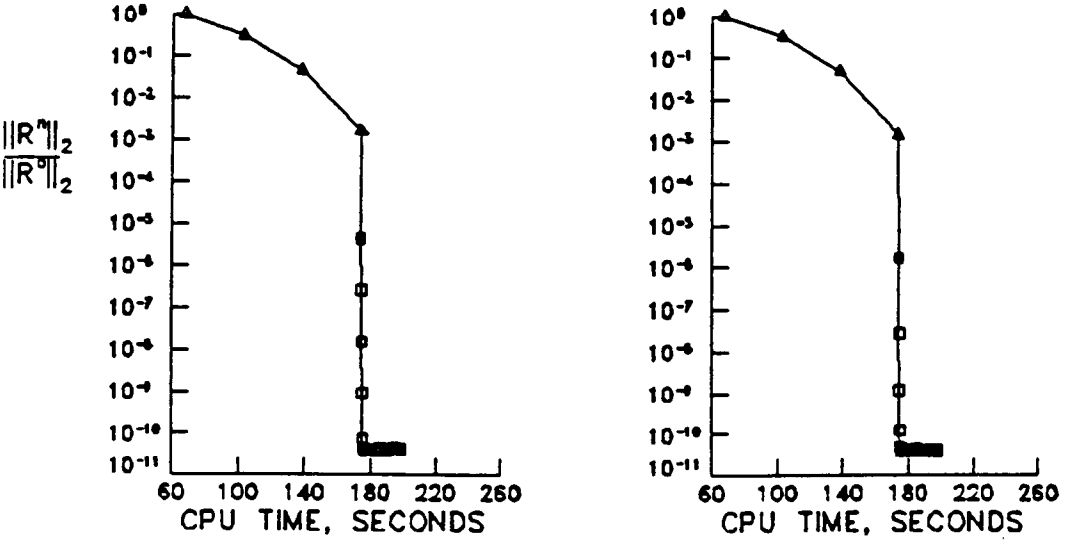


Figure 23. NACA 0012 Airfoil: Residual vs CPU Time, M_∞ = .8, 81x45 Grid



(a) Re = 5000

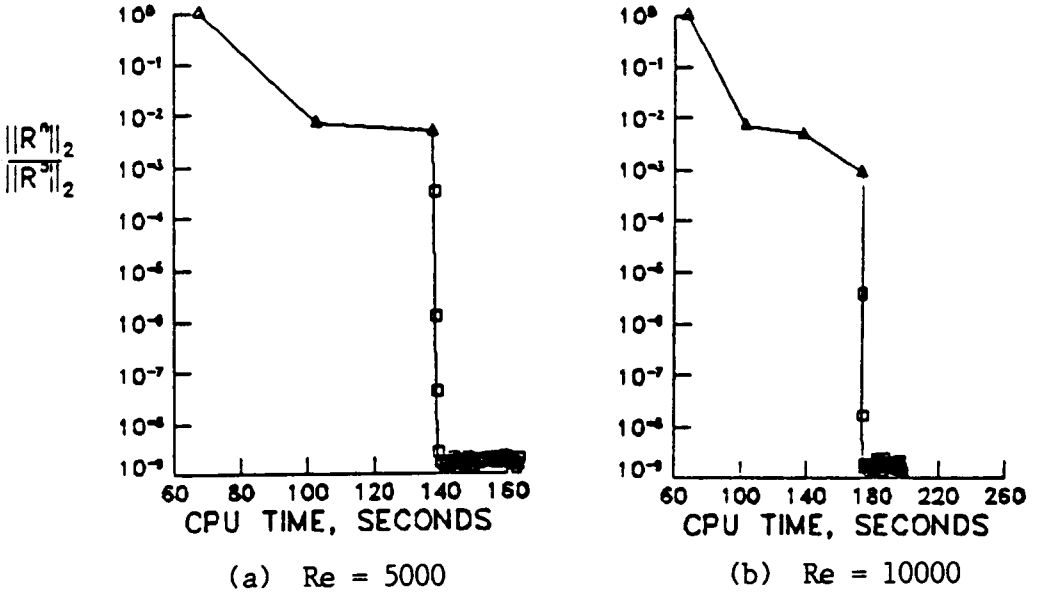
(b) Re = 10000



(c) Re = 20000

(d) Re = 40000

Figure 24. NACA 0012 Airfoil: Residual vs CPU Time, $M_\infty = .5$, 81x45 Grid



(a) Re = 5000 (b) Re = 10000

Figure 25. NACA 0012 Airfoil: Residual vs CPU Time, M_∞ = .05, 81x45 Grid

Table 4: Summary of NACA 0012 Convergence Results

Mach	Re	Total CPU (sec)	Mesh-Sequencing CPU (sec)	Number of Full Direct Iterations
.8	5000	145	65	2
.8	10000	167	84	2
.8	20000	202	84	3
.8	40000	237	84	4
.5	5000	176	65	3
.5	10000	176	65	3
.5	20000	175	65	3
.5	40000	173	65	3
.05	5000	137	65	2
.05	10000	173	65	3

IX. CONCLUSIONS

The primary objective of this study was to develop strategies using the vectorized banded direct solver such that the computational effort is significantly reduced for 2-D CFD solution processes. An approximate method has been developed to predict the computational work of the vectorized VLGS algorithm and the direct method for solving large systems of equations in CFD applications. The analysis indicates and numerical experimentation confirms that the vectorized direct solver with a reuse strategy is more efficient in driving the residual to machine zero than the VLGS method for all problems considered. For either a decrease in vector start-up time or for an increase in machine peak operating rate, the direct method improves in efficiency as compared to the VLGS method. A highly effective strategy investigated was a hybrid approach involving VLGS iterations followed by a single direct inversion with subsequent re-use of the LU decomposition elements from the inversion step. The bulk of the CPU time required for a complete solution is in the single matrix inversion. This indicates that obtaining quadratic convergence with the direct method, long touted as a major advantage of the method, in actuality is not necessary or efficient. If the residual is not reduced enough before the

direct strategy is used, the LU decomposition is relatively inexact and a very large number of reuse iterations may be required.

Alternatively, mesh-sequencing rather than an iteration method can be used to initially reduce the residual. This is particularly efficient for problems where the available iteration method performs poorly in the transient region. The memory required for use of the direct method on fine grids is the major obstacle at this time. The use of direct solvers on coarse grids in a multi-grid strategy can result in a reduction in computational effort. The direct method has been shown to be very robust for all problems. Wide ranges of Mach and Reynolds numbers for viscous flows over airfoils have little effect on time required for convergence when using the method.

An important consideration is that of the tolerance desired in the final solution. For instance, if a four or five order-of-magnitude reduction in the residual is considered to yield an adequate solution for engineering purposes, then an iteration method may be preferred for some problems. It is only when large residual reductions are desired that the direct method becomes efficient relative to existing technology.

For 3-D applications, the direct method has potential

as the solver in cross-flow planes, particularly in conjunction with multi-grid strategies. A sub-domain strategy in which a very large domain is broken into a number of 'pieces' and solved simultaneously using a direct solver also should be investigated fully. This strategy may be extremely useful with multi-processor (parallel) machines.

REFERENCES

1. Wigton, L.B., "Application of MACSYMA and Sparse Matrix Technology to Multielement Airfoil Calculations," AIAA Paper 87-1142, June 1987.

2. Bender, E.E. and Khosla, P.K., "Solution of the Two-Dimensional Navier-Stokes Equations Using Sparse Matrix Solvers," AIAA Paper 87-0603, January 1987.

3. Venkatakrishnan, V., "Newton Solution of Inviscid and Viscous Problems," AIAA Paper 88-0413, January 1988.

4. Riggins, D.W., Walters, R.W., and Pelletier, D., "The Use of Direct Solvers for Compressible Flow Computations," AIAA Paper 88-0229, January 1988.

5. Dwyer, H.A., Matsuno, K., Ibrani, S. and Hafez, M., "Some Uses of Direct Solvers in Computational Fluid Mechanics," AIAA Paper 87-0594, January 1987.

6. Giles, M., Drela, M. and Thompkins, W.T., Jr., "Newton Solution of Direct and Inverse Transonic Euler Equations," AIAA Paper 85-1530, 1985.

7. Van Dam, C.P., Hafez, M. and Ahmad, J., "Calculation of Viscous Flows Using Newton's Method and Direct Solver," AIAA 88-0412, January 1988.

8. Childs, R.E. and Pulliam, T.H., "A Newton-Multigrid Method for the Euler Equations," AIAA Paper 84-0430, January 1984.

9. Liou, M., "An Efficient Method for Solving the Steady Euler Equations," AIAA Paper 86-1079, May 1986.

10. Lambiotte, J.J., Jr., "The Solution of Linear Systems of Equations on a Vector Computer," Ph.D. Dissertation, University of Virginia, May 1975.

11. Thomas, J., Van Leer, B. and Walters, R.W., "Implicit Flux-Split Schemes for the Euler Equations," AIAA Paper 85-1680, July 1985.

12. Anderson, W.K. and Thomas, J.L., "Multi-grid Acceleration of the Flux Split Euler Equations," AIAA Paper 86-0274, January 1986.

13. Anderson, W.K., "Implicit Multigrid Algorithms for the Three-Dimensional Flux Split Euler Equations," Ph.D. Dissertation, Mississippi State, August 1986.

14. Van Leer, B., "Flux-Vector Splitting for the Euler Equations," ICASE Report No. 82-30, September 1982; also: Lecture Notes in Physics, Vol. 170, 1982, pp. 507-512.

15. Anderson, W.K., Thomas, J.L. and Van Leer, B., "A Comparison of Finite Volume Flux Vector Splittings for the Euler Equations," AIAA Paper 85-0122, January 1985.

16. White, Frank, Viscous Fluid Flow, McGraw-Hill, Inc., New York, 1974, pp. 28-30.

17. Van Leer, B. and Mulder, W.A., "Relaxation Methods for Hyperbolic Equations," Report 84-20, Delft University of Technology, 1984.

18. Brandt, Achi, "Multi-Level Adaptive Solutions to Boundary-Value Problems," Mathematics of Computation, Volume 31, Number 138, pp. 333-390, April 1977.

19. Brandt, Achi, "Multilevel Adaptive Computations in Fluid Dynamics," AIAA Journal, Vol. 18, October 1980, pp. 1165-1172.

20. Rizzi, A. and Viviand, H. (Editors), Numerical Methods for the Computation of Inviscid Transonic Flows with Shock Waves: A GAMM Workshop, Vol. 3 (Notes on Numerical Fluid Mechanics), Friedr. Vieweg & Sohn, Braunschweig/Wiesbaden, 1981.

21. Thomas, J.L. and Walters, R.W., "Upwind Relaxation Algorithms for the Navier-Stokes Equations," AIAA Journal, Vol. 25, April 1987, pp. 527-534.

22. Hakkinen, R.J., Greber, I., Trilling, L. and Abarbanel, S.S., "The Interaction of an Oblique Shock Wave with a Laminar Boundary Layer," NASA Memo 2-18-59W, March 1959.

23. Swanson, R.C. and Turkel, E., "Artificial Dissipation and Central Difference Schemes for the Euler and Navier-Stokes Equations," AIAA Paper 87-1107, 1987.

24. Abbott, I.H. and Von Doenhoff, A.E., Theory of Wing

Sections, McGraw-Hill, Inc., New York, 1959, pp. 113-114.

25. George, A. and Lui, J.W., "Computer Solution of Large Sparse Positive Definite Systems," Prentice Hall, 1981.

26. Newsome, R.W., Walters, R.W. and Thomas, J.L., "An Efficient Iteration Strategy for Upwind/Relaxation Solutions to the Thin-Layer Navier-Stokes Equations," AIAA Paper 87-1113, 1987.

27. Chima, R.V., Turkel, E. and Schaffer, S., "Comparison of Three Explicit Methods for the Euler and Navier-Stokes Equations," AIAA Paper 87-0602, January 1987.

28. Walters, R.W. and Riggins, D.W., "CFD Applications of Newton's Method on Supercomputers," Computational Mechanics '88, Proceedings of the International Conference on Computational Engineering Science, Volume 2, pp. 51.vi.1-4, April, 1988.

29. Walters, R.W. and Dwoyer, D.L., "Efficient Solutions to the Euler Equations for Supersonic Flow with Embedded Subsonic Regions," NASA TP 2523, January 1987.

30. Pissanetsky, S., Sparse Matrix Technology, Academic Press, Inc., Orlando, Fla., 1984.

31. Walters, R.W. and Thomas, J.L., "Advances in Upwind Relaxation Methods," State of the Art Surveys in Computational Mechanics, ASME Special Publications.

32. Steger, J.L. and Warming, R.F., "Flux Vector Splitting of the Inviscid Gasdynamic Equations with Application to Finite Difference Methods," Journal of Computational Physics, Vol. 40, No. 2, April 1981, pp. 263-293.

APPENDICES

APPENDIX I: VPS-32 (LANGLEY RESEARCH CENTER) MACHINE INFORMATION AND PARAMETERS

The VPS-32 at LRC is manufactured by the Control Data Corporation. Vectorization is invoked by the software formulation "semi-colon notation".

Memory

33554000 sixty-four bit words of central memory

32505000 (496 Large Pages) available to user

Large Page size = 65536 words

Small Page size = 8192 words

Machine Parameters (Vector Operations)

Vector Operation	M_V (Peak Operating Rate) in Megaflops	τ_S (Vector Instruction Start-up Time in μs
addition (a=b+c)	100	1
subtraction (a=b-c)	100	1
multiplication (a=b*c)	100	1
division (a=b/c)	16.5	1
linked triad (a=b+c*d)*	100	2

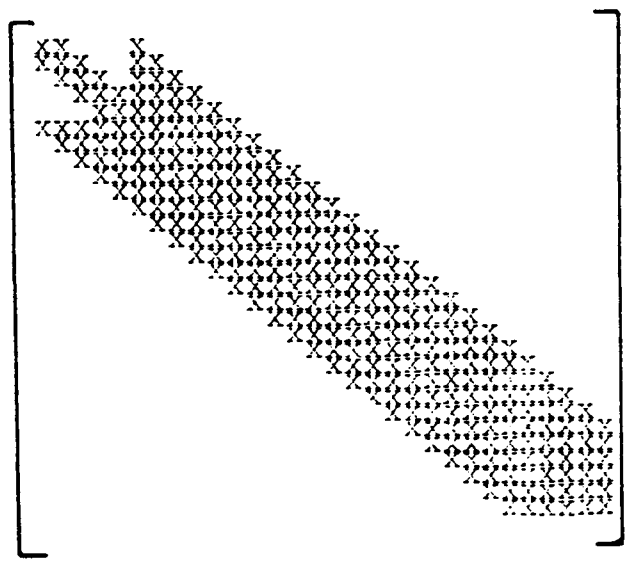
*Note that a linked triad is here considered to be one operation for timing purposes.

APPENDIX II: PHYSICALLY BASED NESTED DISSECTION FOR SPARSE MATRIX SOLUTIONS

The average CFD worker will order the computational domain in such a way to minimize bandwidth as shown in Figure 26. This results in almost complete fill-in in the LU decomposition process (Figure 26b). Nevertheless, as shown in this work, very fast direct matrix solutions are obtained when the method is correctly programmed such that it takes the best advantage of the vector processing capability. Unfortunately, as demonstrated in Reference 4 and this work, the memory required for the LU decomposition is the major drawback for using the method for problems with large numbers of unknowns. However, by strategically renumbering the domain, a significant decrease in the number of non-zero elements produced in the LU decomposition fill-in process can be achieved. When this is coupled with a symbolic Gauss elimination scheme to precisely define the fill-in, the CPU time required for a solution as well as the amount of required memory can be reduced considerably. This is done, of course, at the expense of the simplicity of the banded direct solution approach; sparse vector operations are required with a corresponding increase in complexity. The issue of the 'best' ordering is not resolved. There exists no proof that a method for finding the optimal

5	10	15	20	25	30
4	9	14	19	24	29
3	8	13	18	23	28
2	7	12	17	22	27
1	6	11	16	21	26

(a) Computational Domain
and Conventional Numbering



(b) Resultant Matrix for
First-Order after probable
Fill-in during LU decomposition

Figure 26. Banded System

ordering for an arbitrary domain (i.e. interchanging rows and columns of the coefficient matrix) in order to completely minimize fill-in is possible for the general problem. However, two methods currently used for reordering are the minimum degree algorithm and the method of nested dissection. Both were developed as a graph-based approach for minimizing fill-in and the operations required for the decomposition. Although the minimum degree algorithm produces very good orderings and can be simultaneously performed with symbolic Gauss elimination (to determine the fill-in), it is very complex for many applications and can be expensive. George[25] originally developed the method of nested dissection. Wigton[1] has described a method he calls 'physically based' nested dissection which works quite well for CFD problems and is relatively easy to implement and understand. It also has the advantage that it is fast and need only be performed once (for a given discretization).

The basic idea of physically based nested dissection is to first separate the domain into two approximately equal but 'unconnected' pieces by defining a separator, S, as shown in Figure 27. For first-order differencing, the separator need only be one cell wide, while for second-order differencing S must be at least two cells wide. This separation of the domain into two sub-domains A and B allows

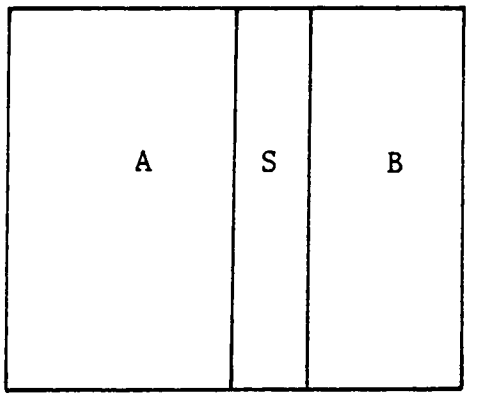


Figure 27. Separation of Domain

the renumbering of cells in both A and B such that the bandwidths associated with the subregions are made as small as possible. An example numbering for the renumbered domain is given in Figure 28a. Notice that those cells in S itself are numbered last. The matrix (shown in terms of subdomains A,B and S) is sketched in Figure 28b. Also note the 'spreading' of the bandwidth evident in Figure 28b for this one application of nested dissection. However, it can be shown that with repeated application of the nested dissection method to each subdomain until there are only undividable cliques of cells left (Figure 29a), a matrix formulation results which for large systems will have significantly less fill-in (and hence operation count) in the LU decomposition than the original banded system ordering. The fill-in that occurs is far less for such a system than is predicted by the traditional 'shadow fill-in' rule. Figure 29b shows the resulting matrix structure for the final ordering.

The advantages of nested dissection become more pronounced for large problems. In Chapter 8 of Reference 25, it is shown that for Laplace's equation on a β by β grid and for $\beta \rightarrow \infty$, the ordering resulting from nested dissection is much better in terms of memory and CPU time in comparison with a banded ordering scheme:

13	14	15	30	24	25
10	11	12	29	22	23
7	8	9	28	20	21
4	5	6	27	18	19
1	2	3	26	16	17

(a) Domain

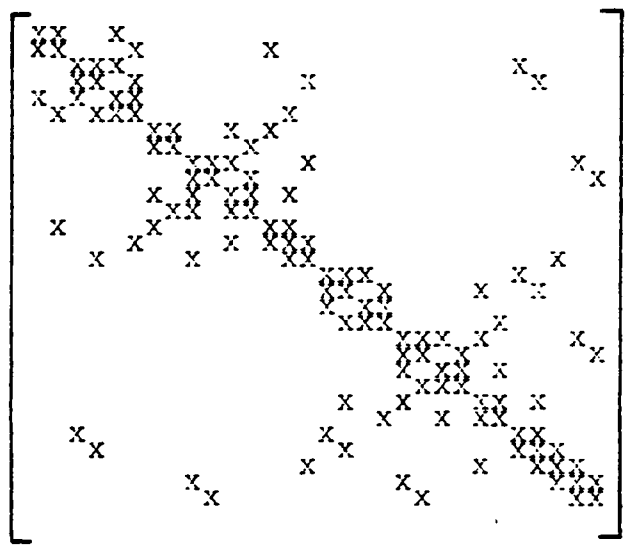
$$\begin{bmatrix} AA & AB=0 & AS \\ BA=0 & BB & BS \\ SA & SB & SS \end{bmatrix}$$

(b) Matrix

Figure 28. Domain and Matrix after One Nested Dissection

8	12	10	30	21	23
7	11	9	29	20	22
13	14	15	28	24	25
2	6	4	27	17	19
1	5	3	26	16	18

(a) Domain



(b) Matrix

Figure 29. Domain and Matrix after Complete Nested Dissection Process

	Storage	Operations
Banded	$0(\beta^3)$	$0(\beta^4)$
Nested Dissection	$0(\beta^2 \ln \beta)$	$0(\beta^3)$

The nested dissection renumbering is actually only the first part of a three step sparse-matrix solution process. These three steps are listed below:

1) Reorder domain using physically-based nested dissection as described - this provides an 'optimal' ordering for minimizing fill-in in the LU decomposition.

2) Perform symbolic Gauss elimination to precisely determine the fill-in

3) Solve for the LU decomposition matrix elements and solve the system using sparse-vector operations.

The second step can be merged with step 3; searching is then done during the LU decomposition itself in order to find non-zero elements created in the process. Wigton has had success with this approach.

The actual implementation of a sparse matrix LU decomposition and solution is far more complex than that necessary for a banded matrix solver insomuch as the non-zero structure of the LU matrix must be determined, stored, indexed, and fetched. However, the basic idea behind the

decomposition remains the same for both methods. With the recent development of very efficient sparse vector operations and searching operations, the sparse matrix solvers are shown to be competitive with banded direct solvers on vector processors.[1]

APPENDIX III: FORTRAN LISTING OF VECTORIZED BANDED DIRECT SOLVER (SEE CHAPTER V FOR DETAILS)

```
DIMENSION C(MAXD,N), B(N)
```

This routine is the solver for a banded system, CX=B, where matrix A is as defined in Chapter V and B is originally the right-hand side vector and is overwritten by solution vector X in the solution process.

Begin LU decomposition; for reuse strategy, this phase should be bypassed.

```
M1 - (MAXD - 1) /2
IAD = M1 + 1

DO 115 I = 1, N - 1
LT = N - 1
IF (LT .GT. M1) LT = M1
C(IAD + 1, I; LT) = C(IAD + 1, I; LT) / C(IAD, I)
DO 114 J = 1, LT
C(IAD - J + 1, I + J; LT) = C(IAD - J + 1, I + J; LT) -
          C(IAD - J, I + J) * C(IAD + 1, I; LT)
CONTINUE
```

Forward substitution; for reuse strategy, begin here.

```
DO 200 I = 1, N - 1
LT = N - I
IF (LT .GT. M1) LT = M1
B(I + 1, J; LT) = B(I + 1, J; LT) - B(I, J) *
          C(IAD + 1, I; LT)
```

Back substitution

```
DO 300 K = N - 1
I = N - K + 1
B(I, J) = B(I, J)/ C(IAD, I)
LT = M1
IF (I. LE. M1) LT = I - 1
B(I - LT, J; LT) = B(I - LT, J; LT) - B(I, J) *
          C(IAD - LT, I; LT)
CONTINUE

B(1,J) = B(1, J) / C(IAD, 1)
```

The vita has been removed from the scanned document

THE EFFICIENT USE OF VECTORIZED DIRECT SOLVERS IN COMPUTATIONAL FLUID DYNAMICS

by

David W. Riggins

Committee Chairman: Robert W. Walters

Aerospace and Ocean Engineering

(ABSTRACT)

The feasibility of using a vectorized banded direct solver for the compressible Euler and Navier-Stokes equations is examined for both single-grid and multi-grid strategies. A procedure is developed for comparing the computational effort required for the direct method with that of the vertical line Gauss-Seidel iteration scheme in order to provide a criteria for choosing between the two techniques. The direct method is shown to have a relatively wide range of application on a vector processor with large memory. Indeed, the primary limitation of the direct method at this time is machine memory. Results for both inviscid and viscous test problems over a range of Mach numbers and Reynolds numbers are examined for two dimensions.