COMPUTATIONAL STRATEGIES FOR THREE-DIMENSIONAL FLOW SIMULATIONS ON DISTRIBUTED COMPUTER SYSTEMS

Final Report
for
NASA Grant NAG 2-869

Attn.: Dr. William Van Dalsem
NASA Ames Research Center
Moffett Field, CA 94035-1000

Prepared By

Lakshmi N. Sankar, Professor
Richard A. Weed, Graduate Research Assistant
School of Aerospace Engineering
Georgia Institute of Technology, Atlanta, GA 30332-0150

August 1995

BACKGROUND

This research effort is directed towards an examination of issues involved in porting large computational fluid dynamics codes in use within the industry to a distributed computing environment. This effort addresses strategies for implementing the distributed computing in a device independent fashion and load balancing. A flow solver called TEAM presently in use at Lockheed Aeronautical Systems Company was acquired to start this effort.

SUMMARY OF WORK DONE

All the objectives of the research proposal submitted to NASA have been accomplished. Specifically, the following tasks were completed.

1. Mr. Richard Weed, a graduate student working on this project ported the TEAM code to a number of distributed computing platforms. These platforms include (a) a cluster of HP workstations located in the School of Aerospace Engineering at Georgia Tech, (b) A cluster of DEC Alpha Workstations in the Graphics visualization lab located at Georgia Tech, (c) a cluster of SGI workstations located at NASA Ames Research Center, and (d) An IBM SP-2 system located at NASA Ames Research center. The public domain PVM software was used to establish communications between the processors.

2. A number of communication strategies were implemented. Specifically, the manager-worker strategy and the worker-worker strategy were tested. The manager-worker strategy was found to be simpler to implement, but required a large amount of manager workstation memory. It was found to be inferior to the worker-worker strategy where worker processors directly exchange information.

4. A variety of load balancing strategies were investigated. Specifically, the static load balancing, task queue balancing and the Crutchfield algorithm were coded and evaluated.

5. The classical explicit Runge-Kutta scheme in the TEAM solver was replaced with an LU implicit scheme. The performance of the implicit scheme on a distributed platform was compared to that of the baseline code. In most instances, the implicit scheme was found to be superior to the explicit scheme.

6. The implicit TEAM-PVM solver was extensively validated through studies of unsteady transonic flow over an F-5 wing, undergoing combined bending and torsional motion.

These investigations are documented in extensive detail in Mr. Richard Weed's Ph. D. dissertation. A copy of this dissertation is enclosed as an appendix. At this writing, this thesis is being reviewed by a committee of 5 Georgia Tech faculty members, and minor changes to the dissertation are likely. A final draft of Mr. Weed's dissertation will all the corrections will be mailed to our technical monitor, Dr. William Van Dalsem, in September 1995.

EXTERNAL INTERACTIONS AND TECHNOLOGY TRANSFER

The implicit PVM-TEAM code has been made available to researchers at Lockheed Martin Corporation, and to researchers at Wright Labs. The following papers were also published.

1. Weed, R. and Sankar, L. N., "Computational Strategies for Three-Dimensional Flow Simulations on Distributed Computer Systems," AIAA Paper 94-2261.

2. Weed, R. and Sankar, L. N., "Computational Strategies for Three-Dimensional Unsteady Flow Simulations on Distributed Computing Systems," Proceedings of the NASA Computational Aerosciences Workshop, March 7-9, 1995.

ACKNOWLEDGMENTS

This work was supported by the NASA Ames Research Renter under Grant NAG-2-869. The technical monitors were Dr. Wiiliam Van Dalsem and Dr. Terry Holst. The authors are thankful to Mr. Frank Witzeman of Wright Labs for providing the TEAM code and sample input deck for starting this effort. The present authors wish to thank Dr. Pradeep Raj and Mr. Brian Goble of Lockheed Martin Corporation for their assistance and encouragement throughout this effort. The authors are also thankful to Merritt Smith of NASA Ames Research Center for valuable assistance on all aspects of the present research.

APPENDIX

COMPUTATIONAL STRATEGIES FOR THREE-DIMENSIONAL FLOW SIMULATIONS ON DISTRIBUTED COMPUTING SYSTEMS

A Thesis

Presented to

The Academic Faculty

by

Richard Allen Weed

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy in Aerospace Engineering

Georgia Institute of Technology

August 1995

COMPUTATIONAL STRATEGIES FOR THREE-DIMENSIONAL FLOW SIMULATIONS ON DISTRIBUTED COMPUTING SYSTEMS

Approved:

_____

Lakshmi N. Sankar, Chairman

_____

Suresh Menon

_____

Stephen M. Ruffin

Date Approved:_____

ACKNOWLEDGMENTS

I would like to express my profound gratitude to my thesis advisor, Dr. L. N. Sankar, whose advice and encouragement has made this effort possible. It has been a distinct honor to have had Dr. Sankar as a teacher, colleague, and friend over the many years I have known him.

I would also like to thank Dr. Suresh Menon and Dr. Stephen Ruffin serving on the thesis advisory committee. In addition, I would also like to thank acknowledge the contributions of Dr. P. K. Yeung and Dr. Karsten Schwann as members of the thesis reading committee. Special thanks are due to the final member of the thesis reading committee, Dr. Pradeep Raj, of the Lockheed Aeronautical Systems Company for his many helpful suggestions and comments throughout the course of this work and for providing access to the baseline computer code and computational grids used in this effort.

This work was supported by the National Aeronautics and Space Administration under grant number NAG 2-869. I would like to express my appreciation to Dr. Terry Holst and Dr. Bill Van Dalsem at NASA's Ames Research Center who served as technical monitors for the grant for providing both monetary support and access to the National Aerodynamics Simulation facility's computer resources. In addition, I would like to express my gratitude to Mr. Merritt Smith of the Computational Aerosciences Branch at NASA Ames and Dr. Christopher Atwood of Overset Methods, Inc. for their helpful discussions and suggestions during my work period at NASA Ames.

I would like to thank the Lockheed Aeronautical Systems Company for providing support for tuition and books during the initial part of my graduate career at Georgia Tech

and for providing me the opportunity to continue my studies while serving as a full-time employee.

I would like to thank my former colleagues at Lockheed, Dr. George Shrewsbury, Dr. Marilyn Smith, and Dr. David Schuster for their friendship and help in preparing for the oral qualifying exam and Dr. Ashok Bangalore for our many discussions about distributed computing. I would like to add a special thanks to Dr. Larry Birckelbaw and Mrs. Luly Birckelbaw for the friendship and kindness they have shown me over the years and for encouraging me to complete my degree requirements.

Finally, I would like to dedicate this work to my parents and express my profound thanks for all the love and encouragement they have given me over the course of my life. This work would not have been completed without their constant moral support. I thank them for all the sacrifices they have made to turn my dreams of becoming an engineer into a reality.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

NOMENCLATURE

A,B,C	flux Jacobian matrices
c	speed of sound, chord length
CD	total wing drag coefficient
CFL	Courant-Friedrech-Lewy number
CL	total wing lift coefficient
CM	total wing moment coefficient
Cp	pressure coefficient
c_p	specific heat at constant pressure
c_v	specific heat at constant volume
D	diagonal operator in LU-SGS scheme
D_ξ, D_η, D_ζ	difference operators
d	dissipative flux
E	total energy
E_1, E_2, etc	matrices in matrix dissipation formulation
e	internal energy
F	flux vector, frequency in Hertz
G	non-convective flux
H	total enthalpy
h	enthalpy
I,J,K	grid indices

I	identity matrix
J	Jacobian of general coordinate transformation
k	coefficient of thermal conductivity
L	reference length
ℓ	turbulent length scale
M	Mach number
n	unit normal vector
p	pressure
P_r	Prandtl number
Q	conservation variables
q	velocity magnitude
R	gas constant, Reimann invariants, residual
R_e	Reynolds number
r	position vector
T	temperature, elapsed time
t	time
U	contravariant velocity
u,v,w	Cartesian velocity components
v_s	surface velocity
V	cell volume
X,Y,Z	Cartesian coordinates in inertial system
x_t,y_t,z_t	grid speeds
α	angle of attack, dissipation eigenvalue factor

β	dissipation factor
γ	ratio of specific heats
Δ	increment, forward difference operator
∇	backwards difference operator, gradient operator
δ	central difference operator
ξ,η,ζ	curvilinear coordinates
$\varepsilon_2,\varepsilon_4$	dissipation coefficients
λ	bulk viscosity coefficient
$\lambda(A)$, etc	eigenvalue of the flux Jacobian matrices
θ	mode shape, temporal accuracy parameter
κ	reduced frequency
Λ	dissipation factor
μ	molecular viscosity
μ_T	eddy viscosity
ρ	density
$\rho(A)$	spectral radius of flux Jacobian matrices
τ	shear stress
τ_{xx}, etc	components of stress tensor

Φ viscous dissipation function

ω vorticity

Superscripts

n time level

k integration stage

Subscripts

I,J,K grid indices

E effective value

T turbulent quantity

t derivative with respect to time

x,y,z derivative with respect to the Cartesian coordinates

ξ,η,ζ derivatives with respect to curvilinear coordinates

∞ freestream quantity

SUMMARY

In the present research, the issues involved in the development and implementation of three-dimensional Computational Fluid Dynamics analysis methods on distributed parallel computing systems are explored. The growing requirements in the aerospace industry for more cost effective aerodynamic simulations has led to research into the use of distributed systems based on loosely and tightly coupled systems of engineering workstations as substitutes for large scale supercomputer systems for large scale aerodynamic simulations. However, effective computational strategies for implementing three-dimensional multi-zone flow solvers on workstation based distributed systems are required to ensure that the distributed parallel flow solvers provide the required levels of performance. In addition, the effects of various computational procedures on the accuracy of both steady and unsteady flow simulations must be evaluated. Finally, efficient procedures for implementing a distributed parallel flow solver on a variety of distributed system needs to be quantified. This research attempts to evaluate and define the most appropriate strategies for implementing a cost effective distributed parallel flow solver for both steady and unsteady flow simulations.

The approach taken in this research was to use an existing industry standard three-dimensional multi-zone flow solver in conjunction with the Parallel Virtual Machine software interface to develop a distributed parallel flow solver that was used to define and evaluate efficient communications and load balancing strategies for workstation based systems. The research was performed in two phases. In the first phase, an explicit flow solver was implemented on small Ethernet based networks of engineering workstations. Steady flow simulations for standard wing and body of revolution geometries were used to

evaluate a Manager/Worker communications strategy and two static load balancing schemes The results of these simulations also demonstrated the effects of the zonal boundary condition update procedures and numerical dissipation models on the accuracy and convergence of the distributed flow solver. It was found that increased levels of dissipation were required to maintain the convergence of the distributed solver. The adverse effects of system load and communications overhead were demonstrated.

In the second phase of the research, an improved version of the baseline code was used to develop both explicit and implicit flow solvers that were implemented on the large scale IBM SP2 distributed system at the NASA Ames Research Center. These solvers were validated for both Euler and Navier-Stokes simulations using a standard test case. The accuracy and performance the explicit and implicit distributed flow solvers were evaluated.

A series of unsteady flow analyses were performed using the implicit flow solver to quantify the effect of the domain decomposition procedure used in the distributed solver on solution accuracy. The test case for these simulations was the modal vibration of the F5 wing oscillating in pitch. The results of these tests led to the implementation and evaluation of a second communications strategy that demonstrated improved performance for increasing numbers of processors on the SP2 system. These results demonstrated the viability of a distributed flow solver for unsteady flow simulations on a large scale distributed system such as the SP2.

Finally, the utility of workstation based distributed flow solvers for real world aerodynamic analyses of both steady and unsteady flows was demonstrated. The distributed flow solvers developed in this research are felt to provide the basis for a production aerodynamic analyses and design tool.

CHAPTER I

INTRODUCTION

During the past decade, an increasing amount of research activity in Computational Fluid Dynamics (CFD) has been devoted to harnessing the power of parallel computing architectures to improve the total throughput of CFD codes. This effort was prompted by several factors. Existing vector supercomputers such as the CRAY systems are approaching the theoretical limit in processing speed obtainable by a single processor. In the same period of time, higher order CFD methods such as flow solvers based on numerical solution of the Euler or Navier-Stokes equations have obtained a wider acceptance in the aerospace community as viable tools for aerodynamic design and analysis. Current production CFD codes in use in the aerospace industry are capable of performing both Euler and Navier-Stokes analyses on complete aircraft [1,2,3]. However, these types of analyses are expensive in both time and money even for current large scale supercomputing systems. This has led to increasing emphasis on reducing the overall costs and increasing the throughput of CFD analyses in order to integrate them into the overall design process. In addition, research is underway to develop multi-disciplinary tools that combine structural analyses, aerodynamic analyses, and optimization procedures into a single analyses method. These multidisciplinary methods require computing power that is beyond the performance capabilities of a single processor. The increased demand for faster

throughput and increasing problem size has led to research in the use of parallel processing systems for CFD simulations.

1.1 Motivation For The Present Research

All parallel computing systems utilize multiple processors of varying levels of power and sophistication to achieve performance improvements over serial computers. Parallel systems can be differentiated by the relationship of the memory subsystem to individual processors and the number of data streams available on the system [4]. Memory subsystems are classified as either distributed memory or shared memory. On distributed memory systems, each processor has direct access to only its own local memory space. Access to information on the other processors is accomplished by passing data messages or packets over a high speed communications network. On shared memory multiprocessors, each processor accesses the same global memory space.

Over the last few years, the Multiple Instruction-Multiple Data (MIMD) systems such as the Cray Y-MP, Cray C-90, and the Intel Paragon and iPSC/860 systems have supplanted the Single Instruction-Multiple Data (SIMD) shared memory machines such as the Connection Machine as the predominant parallel systems for CFD calculations. The Cray systems are examples of shared memory machines while the Intel systems use distributed memory. Shared memory machines are best suited for problems that require fine grain parallelism where work is distributed among the processors at the level of individual loops or vectors. Distributed memory machines are best suited for problems that can utilize coarse grain parallelism where a large problem can be decomposed into smaller problems or tasks that are mapped to the individual processors.

In the mid 1980's, studies performed by Johnson [5] and Gropp and Smith [6] pointed out the advantages and disadvantages associated with the use of parallel processing systems

for CFD simulations. In particular, the performance penalties due to communications overhead between processors and the necessity of maintaining an equal balance of work among processors was emphasized. In addition, the problems associated with transferring existing serial algorithms to the parallel environment were described. Therefore, a significant amount of research has centered on developing new algorithms to take advantage of the task level parallelism inherent in the numerical solutions of the Euler and Navier-Stokes equations and the problems associated with porting existing algorithms to different parallel architectures. Most of this research has been targeted at massively parallel architectures that are composed of large numbers of simple processors such as the Connection Machine CM-2 and CM-5 and the CRAY T3D systems which can utilize thousands of processors, smaller distributed multi-processor versions of vector machines such as the CRAY C-90 which have typically four to eight specialized high-speed processors, and large distributed multi-processor systems such as the Intel Paragon and iPSC/860. However, the difficulties in porting existing codes to these systems along with their high acquisition and overhead costs and limited availability have prevented their wide spread use by the aerospace industry. The rapid development of high-speed engineering workstations has led to large distributed systems such as the IBM SP2 system [7] based on existing workstation processors.

Recently, the increasing performance to price ratios and availability of engineering workstations have led some researchers to explore the viability of linking clusters of workstations together to form distributed parallel systems [8]. This interest was also prompted by the fact that most engineering workstations are idle during off hours and represent an underutilized computing resource of enormous potential. In addition, the recent development of standardized application programming interfaces such as the Parallel Virtual Machine (PVM) system [9,10,11,12] have greatly reduced the effort required to link together workstations into a distributed parallel system. The recent work of Smith and

Palas [13] demonstrated the feasibility of a PVM based parallel distributed flow solver. They pointed out the need for effective computational procedures tailored to workstation based distributed systems. This research was undertaken to help define these procedures and to evaluate the effort required to implement an existing flow solver as a parallel solver on distributed systems using PVM. In addition, the present research was undertaken to determine the viability of a PVM based multi-zone parallel flow solver for unsteady flow simulations

1.2 Survey Of Literature On Parallel Processing And CFD

In the 1980's, early attempts to employ parallel processing in CFD centered on the development of fine grain solution algorithms for the Euler and Navier-Stokes equations that allowed inner loops in the solver to be processed in parallel. The multitasking approach used on Cray computing systems is an example of loop level parallelism. An example of the effectiveness of multitasking for CFD calculations is given in the work of Swisshelm et. al. [14]. Other authors such as Patel, Sturek, and Jordan [15] developed parallel solution algorithms for multiprocessor MIMD systems such as the Denelcor HEP computers. As the decade progressed, parallel algorithm development was focused on data parallel SIMD systems such as the Connection Machine CM2 and large multiprocessor MIMD systems such as the Intel iPSC/860 systems.

In 1989, several researchers presented results for Euler and Navier-Stokes calculations on the Connection Machines. Wake and Eglof [16] ported a hybrid implicit-explicit Navier-Stokes solver for helicopter rotor analysis to the CM-2 and achieved performance that was slightly better than on a single processor Cray-2 system. An improved version of the code [17] was able to achieve twice the performance of a Cray 2 system. Long, Khan, and Sharp [18] implemented an three-dimensional explicit Euler/Navier-Stokes solver on the

CM-2 using the Lisp programming language. Another implementation of a Navier-Stokes solver on the CM-2 was described by Agarwal [19]. Kallinderis and Vidwans [20] have introduced a generic parallel adaptive-grid Navier-Stokes solver that has executed successfully on both the CM-2 and Cray Y-MP/8 systems. These flow solvers were based on structured grid systems. Hammond and Barth [21] developed an unstructured Euler solver for the CM-2. Morano and Mavriplis [22] have also implemented an unstructured Euler solver on the CM-5 machine operating in MIMD mode.

The introduction of large distributed parallel systems such as the Intel iPSC/860 has led to a significant amount of research for these types of machines. Ryan and Weeratunga [23] used a distributed parallel version of an overset grid flow solver to compute Navier-Stokes flowfields for supersonic vehicles on the iPSC/860. Hixon and Sankar [24] used domain decomposition and an implicit flow solver to perform unsteady two-dimensional flow calculations. Otto [25] used the iPSC/860 to perform Navier-Stokes calculations of chemically reacting flows. Das et. al. [26] and Venkatakrishnan et. al. [27,28] have developed parallel flow solvers for unstructured grid systems using the iPSC/860. Unsteady aeroelastic flow simulations have been performed by Promono and Weeratunga [29] and Byun and Guraswamy [30] using the iPSC/860 and Paragon systems. Imlay and Soetrisno [31] studied the problems of porting implicit flow solvers for chemically reacting flows to both shared and distributed memory MIMD machines. Fatoohi [32] has described the effort required to adapt the NASA Ames INS3D flow solver to three different parallel systems.

In the past two years, research in the area of workstation based distributed systems has grown due to the increasing pressure of budget constraints in the aerospace industry and the introduction of software such as PVM. The previously cited work of Smith and Palas [13] proved the viability of a workstation based system for real world applications. This has led to a wide range of workstation based programs for distributed systems. An

overview of these programs was given in Reference 33. Hayden, Jayasimha, and Pillay [34] compared the performance of a workstation based system with other shared and distributed memory systems. Deshpande, Feng, and Merkle [35] studied the effect of various network communications protocols on the performance of a parallel flow solver on a distributed network system. The majority of the distributed flow solvers have been applied to steady-flow only. Recently, Bangalore et. al. [36,37] used a workstation based distributed system to simulate the unsteady viscous flow over rotor systems. Finally, the first phase of the present research was reported in References [38] and [39].

1.3 Overview of the Present Research

1.3.1 Objectives and Approach

The main objectives of this research are to evaluate the issues involved in porting, fine tuning, and improving an existing flow solver for optimal performance on a distributed parallel system. The research emphasizes the development and evaluation of efficient computational strategies for both steady and unsteady flow. The approach taken was to utilize an existing multi-zone solver, the Lockheed/AFOSR Three-Dimensional Euler/Navier-Stokes Aerodynamic Method (TEAM) code [40], and the PVM software interface to develop a parallel distributed flow solver that could run on a variety of hardware platforms.

A variety of multi-zone solvers have been developed in the past decade [41, 42, 43]. The common feature of all these flow solvers is that they break the computational domain into several different grids with either abutting or overlapping interfaces. This natural domain decomposition makes multi-zone solvers such ideally suited for distributed computing systems because each zone or a set of zones can be mapped to separate

processors. The solution in each zone is generated concurrently using the same solution algorithm running on each processor. Therefore, a parallel flow solver can be implemented with relatively minor code modifications. However, the performance of a distributed flow solver can be severely impacted by the communications overhead introduced by the size and number of messages that are exchanged during the solution process to synchronize the solution and transfer zonal boundary information. Therefore, effective communications strategies must be implemented to reduce the overhead and maintain system performance. In addition, efficient procedures are required to maintain as closely as possible an equal level of work on each processor. This reduces the amount of time each processor is idle during the solution cycle. This process is called load balancing.

The present research was performed in two phases. In the first phase, a version of the Lockheed/AFOSR TEAM code was obtained from the U.S. Air Force to serve as a baseline code. Version 3.1.5 of PVM was installed on a system of four Hewlett-Packard workstations on an Ethernet based network. The PVM software interface was used to implement a distributed version of the baseline code (PVMTEAM) that was used as a test bed for evaluating code performance and implementation issues. Standard test cases were used to evaluate overall performance and the effects of such issues as boundary update procedures and the effects of numerical dissipation on the solver performance and accuracy. The code was also run on a network of Digital Equipment Company (DEC) ALPHA workstations. In the second phase, an improved version of the code was implemented on a system of Silicon Graphics Co. (SGI) workstations. This code was later ported to the IBM SP2 distributed supercomputer system at NASA Ames Research center. The improved code was used to evaluate two different load balancing algorithms and two communications strategies. Finally, this version of the code was used to perform a series of unsteady flow simulations for an oscillating wing.

1.3.2 The Three-Dimensional Euler/Navier-Stokes Aerodynamic Method

The baseline code was developed by the Lockheed Aeronautical Systems Company for the United States Air Force. This code is a multi-zone explicit finite volume code based on the mulit-stage Runge-Kutta time stepping scheme of Jameson et. al. [44]. Acceleration techniques such as residual smoothing and enthalpy damping are employed to extend the stability bounds of the explicit scheme and allow the use of larger Courant numbers. The code has been validated for a wide variety of geometries and flow conditions. The multi-zone formulation allows the code to be applied to complex geometries such as complete aircraft. The code has a wide variety of boundary condition options for both external and internal flows. In addition, several numerical dissipation models are available that insure stable solutions over a wide range of Mach numbers. The code employs a dynamic memory allocation scheme that sizes solution arrays at run time. This eliminates the need to recompile the code whenever the size of the grid systems change. These features made the TEAM code an excellent baseline for the present research.

1.4 Organization Of The Dissertation

The remaining chapters of this dissertation are organized in the following manner. A discussion of the mathematical formulation of the conservation law form of the governing equations is given in Chapter 2. The numerical algorithms used in the distributed flow solver is described in Chapter 3. This includes complete discussions of the spatial discretization techniques, dissipation models, boundary conditions and time stepping schemes used in the research. Chapter 4 summarizes the procedures used to implement the distributed version of the solver. This includes discussions of the communications and load balancing strategies used in the code. Results for both steady and unsteady flow simulations on the workstation based systems used in this research are presented in Chapter

5. The results of steady and unsteady simulations on the IBM SP2 system at NASA's Numerical Aerodynamics Simulation (NAS) facility are presented in Chapter 6. The final conclusions and suggestions for future results are presented in Chapter 7.

CHAPTER II

MATHEMATICAL FORMULATION OF THE EULER AND NAVIER-STOKES EQUATIONS

This chapter will discuss the formulation of the Reynolds-Averaged Navier-Stokes equations used in the baseline Three-Dimensional Euler/Navier-Stokes Aerodynamic Method code. The Euler equations will be discussed as a subset of the Navier-Stokes equations. The procedure used to non-dimensionilize the Euler and Navier-Stokes equations will be described. The turbulence model and procedures used to compute the thermodynamic and transport properties required to provide closed systems of equations will also be discussed.

2.1 Integral Form of The Governing Equations

The formulation of the Navier-Stokes Equations used in the baseline code is derived from the integral form of the equations obtained by application of Reynolds Transport Theorem [45] and the principles of conservation of mass, momentum, and energy to a control volume of arbitrary shape surrounding a cell of fluid moving in space and time. Following Vinokur [46], a general conservation law for a fluid cell of volume V(t) moving through a finite region of space over a finite interval of time, $t_2 - t_1$, can be written as

$$\int_{V(t_2)} \vec{Q}dV - \int_{V(t_1)} \vec{Q}dV + \int_{t_1}^{t_2} \oint_{S(t)} \vec{F} \bullet \vec{n}dSdt = 0 \qquad (2.1)$$

where Q is a vector of conserved variables per unit volume, F is the flux of Q per unit volume per unit time, S(t) is the surface bounding the volume, dS is a differential element of S, and n is a outward pointing unit vector normal to dS. The flux F can be written as

$$\vec{F} = (\vec{u} - \vec{v}_s)\vec{Q} + \vec{G} \qquad (2.2)$$

where u is the fluid velocity vector and v_s is the velocity of the surface element dS. The first term in Equation (2.2) represents the convective component of the flux and the quantity G is the non-convective component of the flux due to forces acting on the volume and the dissipation and transfer of energy into and out of the volume. For the Euler and Navier-Stokes equations, the flux terms arise from the fluid pressure acting on the surface, shear stresses due to viscosity, and heat transfer across the cell surface. If all the variables are assumed to be continuous in time, Equation (2.1) can be written in the more familiar integro-differential form

$$\frac{D}{Dt}\int_V QdV + \oint_S \vec{F} \bullet \vec{n}dS = 0 \qquad (2.3)$$

For the compressible Euler and Navier-Stokes equations, the conserved variables are normally taken to the density ρ, the fluid momentum $\rho\vec{u}$, and the total energy per unit volume E. The vector Q can be written as

$$Q=(\rho,\rho u,\rho v,\rho w,E)^T \tag{2.4}$$

where u, v, and w are the Cartesian components of the fluid velocity with respect to an interial coordinate system. The convective component of the flux F can be expanded into its Cartesian components and combined with the pressure force and work contributions from G to give the inviscid flux components F_X, F_Y, and F_Z which can be written as

$$F_X=\begin{pmatrix}\rho(u-x_t)\\ \rho u(u-x_t)+p\\ \rho v(u-x_t)\\ \rho w(w-x_t)\\ E(u-x_t)+pu\end{pmatrix};\ F_Y=\begin{pmatrix}\rho(v-y_t)\\ \rho u(v-y_t)\\ \rho v(v-y_t)+p\\ \rho w(v-y_t)\\ E(v-y_t)+pv\end{pmatrix};\ F_Z=\begin{pmatrix}\rho(w-z_t)\\ \rho u(w-z_t)\\ \rho v(w-z_t)\\ \rho w(w-z_t)+p\\ E(w-z_t)+pw\end{pmatrix} \tag{2.5}$$

where x_t, y_t, and z_t are the Cartesian components of cell surface velocity with respect to an inertial coordinate system and p is the static pressure. The system of equations formed by substituting the inviscid components of F and the conserved variables defined in Equation (2.4) into Equations (2.1) and (2.3) comprise the Euler Equations.

The Navier-Stokes equations are obtained by including the forces due to viscous stresses into the momentum equation components of the flux and the energy transfer into and out of the control volume due to thermal conduction and the dissipation due to the deformation of the volume by viscous stresses into the energy equation. The Cartesian components of the flux due to viscous stresses and heat transfer F_{VX}, F_{VY}, and F_{VZ} can be defined as

$$F_{VX} = \begin{pmatrix} 0 \\ \tau_{XX} \\ \tau_{XY} \\ \tau_{XZ} \\ k\frac{\partial T}{\partial x} + \Phi_X \end{pmatrix}; F_{VY} = \begin{pmatrix} 0 \\ \tau_{YX} \\ \tau_{YY} \\ \tau_{YZ} \\ k\frac{\partial T}{\partial y} + \Phi_Y \end{pmatrix}; F_{VZ} = \begin{pmatrix} 0 \\ \tau_{ZX} \\ \tau_{ZY} \\ \tau_{ZZ} \\ k\frac{\partial T}{\partial z} + \Phi_Z \end{pmatrix} \tag{2.6}$$

and the viscous stress and dissipation terms are defined as

$$\tau_{XX} = (\lambda + 2\mu)\frac{\partial u}{\partial x} + \lambda(\frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}) \tag{2.7a}$$

$$\tau_{YY} = (\lambda + 2\mu)\frac{\partial v}{\partial y} + \lambda(\frac{\partial u}{\partial x} + \frac{\partial w}{\partial z}) \tag{2.7b}$$

$$\tau_{ZZ} = (\lambda + 2\mu)\frac{\partial w}{\partial z} + \lambda(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}) \tag{2.7c}$$

$$\tau_{XY} = \tau_{YX} = \mu(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}) \tag{2.7d}$$

$$\tau_{XZ} = \tau_{ZX} = \mu(\frac{\partial u}{\partial z} + \frac{\partial w}{\partial x}) \tag{2.7e}$$

$$\tau_{YZ} = \tau_{ZY} = \mu(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}) \tag{2.7f}$$

$$\Phi_X = u\tau_{XX} + v\tau_{XY} + w\tau_{XZ} \tag{2.7g}$$

$$\Phi_Y = u\tau_{YX} + v\tau_{YY} + w\tau_{YZ} \tag{2.7h}$$

$$\Phi_Z = u\tau_{ZX} + v\tau_{ZY} + w\tau_{ZZ} \tag{2.7i}$$

where k is the coefficient of thermal conductivity , T is the temperature, μ is the coefficient of molecular viscosity and λ is the bulk viscosity coefficient. The bulk viscosity λ can be related to the molecular viscosity μ by Stokes hypothesis

$$\lambda = -\frac{2}{3}\mu \tag{2.8}$$

The viscous stress and heat flux terms can be be replaced by "thin-layer" approximations that are analogous to the classical boundary layer approximations. The thin-layer approximation is obtained by excluding derivatives of the flow quantities along specified directions. For flows bounded by a solid surface, these directions are usaully taken to be the streamwise coordinates approximately tangent to the surface. Only the derivatives along the coordinate approximately normal to the surface are retained. This approximation has proven to provide a sufficiently accurate simulation of the viscous flux for a wide range of flows.

Closure of the preceding system of equations requires definitions for the thermodynamic variables (ρ, p, T, E) and the transport properties (μ, k). Analytical relationships between these variables are also required. Pressure can be related to density and temperature through the equation of state for a perfect gas

$$p = \rho RT \tag{2.9}$$

where R is the gas constant. The specific internal energy e, the specific enthalpy h, and the ratio of specifics heats γ can be defined as

$$e = c_v T \quad ; \quad h = c_p T \quad ; \quad \gamma = \frac{c_p}{c_v} \tag{2.10}$$

where c_v is the specific heat at constant volume and c_p is the specific heat at constant pressure. Pressure and temperature can be related to internal energy by using Equations (2.9) and (2.10) and the following definitions for the specific heats

$$c_v = \frac{R}{\gamma - 1} \quad ; \quad c_p = \frac{\gamma R}{\gamma - 1} \tag{2.11}$$

This leads to relationships for pressure and temperature of the form

$$p = (\gamma - 1)\rho e \quad ; \quad T = \frac{(\gamma - 1)e}{R} \tag{2.12}$$

The specific heats c_v and c_p are constants when the gas is both thermally and calorically perfect. Under the assumption of a thermally and calorically perfect gas, the ratio of specific heats for air at standard conditions has a value of 1.4. For imperfect gases, the thermodynamic properties must be computed from tables or curve fits or the system of equations must be modified to solve for the constituent species of the gas [47]. The final thermodynamic relationship is the definition of the total energy per unit volume E which is given by

$$E = \rho(e + \frac{u^2 + v^2 + w^2}{2}) \tag{2.13}$$

The coefficient of molecular viscosity μ has been shown empirically to be a function of temperature for most gases over the range of temperatures where perfect gas assumptions are valid. The most commonly used relationship for viscosity is Sutherland's law [48] which is given by

$$\mu = C_1 \frac{T^{\frac{3}{2}}}{T + C_2} \tag{2.14}$$

where C_1 and C_2 are constants for a given gas. The coefficient of thermal conductivity can be appoxmated by the relation given by Worsoe-Schmidt and Leppert [49]

$$k = T^{0.71} \tag{2.15}$$

However, it is often more convenient to determine k from the definition of the Prandtl number Pr

$$Pr = \frac{c_p \mu}{k} \tag{2.16}$$

The Prandtl number is approximately constant for most gases and has a value of 0.72 for air at standard conditions.

The preceding systems of equations define the general time-dependent or unsteady Euler and Navier-Stokes equations for a compressible fluid in conservation law form. This

form of the equations is favored over non-conservative formulations because of its ability to maintain global conservation throughout the fluid space even when discontinuities such as shock waves are present. In practice, the dimensional variables in the governing equations are recast as non-dimensional variables to remove any dependency on a particular set of units of dimension from the equations. For practical calculations of turbulent flows, additional modifications must be made to the system to account for the random fluctuations in the flow variables induced by turbulence. These modifications are described in sections 2.2 and 2.3 of this chapter.

2.2 Non-Dimensionalization of the Governing Equations

Non-dimensional forms of the flow variables in the governing equations are obtained by dividing dimensional quantities such as density and velocity by an appropriate set of reference values. These reference values are normally taken to be the values of the free-stream at infinity. The set of non-dimensional variables used in the baseline code [40] are defined as follows

$$\rho = \frac{\bar{\rho}}{\bar{\rho}_\infty}, u = \sqrt{\gamma}\frac{\bar{u}}{\bar{c}_\infty}, v = \sqrt{\gamma}\frac{\bar{v}}{\bar{c}_\infty}, w = \sqrt{\gamma}\frac{\bar{w}}{\bar{c}_\infty}, E = \frac{\bar{\rho}_\infty \bar{E}}{\bar{p}_\infty} \tag{2.17a}$$

$$p = \frac{\bar{p}}{\bar{p}_\infty}, T = \frac{\bar{T}}{\bar{T}_\infty}, \mu = \frac{\bar{\mu}}{\bar{\mu}_\infty}, k = \frac{\bar{k}}{\bar{k}_\infty}, t = \frac{\bar{t}\bar{c}_\infty}{\bar{L}\sqrt{\gamma}} \tag{2.17b}$$

$$x = \frac{\bar{x}}{\bar{L}}, y = \frac{\bar{y}}{\bar{L}}, z = \frac{\bar{z}}{\bar{L}} \tag{2.17c}$$

where a bar denotes a dimensional quantity. The variable L is a characteristic length such as wing root chord and c_∞ is the free-stream speed of sound given by

$$c_\infty = \sqrt{\frac{\gamma p_\infty}{\rho_\infty}} \tag{2.18}$$

Substitution of the non-dimensional variables in the governing equations leads to two additional non-dimensional quantities, the Reynolds number Re and the Mach number M, which are defined at the free-stream by the relations

$$Re_\infty = \frac{\bar{\rho}_\infty \bar{q}_\infty \bar{L}}{\bar{\mu}_\infty}, \; M_\infty = \frac{\bar{q}_\infty}{\bar{c}_\infty} \tag{2.19}$$

where q_∞ is the magnitude of the freestream velocity vector. For viscous flows, Sutherland's law for molecular viscosity becomes

$$\mu = T^{\frac{3}{2}} \left(\frac{1 + \frac{T_s}{\bar{T}_\infty}}{T + \frac{T_s}{\bar{T}_\infty}} \right) \tag{2.20}$$

where the non-dimensional temperature is given by the non-dimensional equation of state as $T=p/\rho$ and T_s is a reference temperature which has a value of 110.4 ^{0}K for air.

The non-dimensional form of the conserved variables and the inviscid flux components are unchanged from their dimensional forms. The viscous flux components retain their dimensional form with the addition of a scale factor equal to $\sqrt{\gamma}M_\infty / Re_\infty$, i.e. Equation (2.6) become

$$\bar{F}_{vx,y,z} = \frac{\sqrt{\gamma}M_\infty F_{vx,y,z}}{Re_\infty} \tag{2.21}$$

where $\bar{F}$ is the dimensional form of the viscous flux.

2.3 Reynolds Averaged Formulation for Turbulent Flow

Turbulent flow is characterized by random momentum and energy exchanges over a broad spectrum of length and time scales [50]. Therefore, direct simulation of turbulent flows using finite difference, finite volume, or finite element discretizations of the Navier-Stokes equations would require extremely large numbers of computational points and prohibitively small time steps to resolve all of the characteristic scales of the flow. In addition, the random nature of the velocity fluctuations that are inherent in turbulent flows must be treated statistically. Reynolds employed a simple decomposition of the flow variables into time averaged values with a mean component and a fluctuating component with zero mean over time [50]. This decomposition can be written as

$$f = \bar{f} + f' = \frac{1}{\Delta t}\int_{t}^{t=\Delta t} f dt + f' \tag{2.22}$$

where the primes represent the fluctuating values. When this averaging process is applied to the Navier-Stokes equations, new terms appear in the equations that are functions of the fluctuating components of velocity. In particular, momentum transfer terms are generated that can be thought of as stresses. These terms are commonly called the Reynolds's stresses. It is convenient to combine the additional terms generated by the fluctuating values with the flux vectors based on the mean values. For example,

$$F_{vx} = \bar{F}_{vx} + F'_x \tag{2.23}$$

where F'_x can be written in terms of the fluctuating time averaged velocity components as

$$F'_x = \begin{pmatrix} 0 \\ -\overline{\rho u'^2} \\ -\overline{\rho u'v'} \\ -\overline{\rho u'w'} \\ -\frac{1}{2}\overline{\rho u}(\overline{u'^2} + \overline{v'^2} + \overline{w'^2}) \end{pmatrix} \tag{2.24}$$

The Bossinesq assumption [51] is used to relate the Reynolds stresses to the mean strain rates by means of a proportionality factor commonly referred to as the eddy viscosity. For example,

$$\overline{-\rho u'v'} = -2\mu_T\left(\frac{\partial \bar{u}}{\partial y} + \frac{\partial \bar{v}}{\partial x}\right) \tag{2.25}$$

The Bossinesq assumption allows the molecular viscosity and thermal conductivity to be replaced by effective values defined in non-dimensional form as

$$\mu_E = \mu + \mu_T \, , \, k_E = \frac{\mu}{Pr} + \frac{\mu_T}{Pr_T} \tag{2.26}$$

where Pr_T is the turbulent Prandtl number which has an approximate value of 0.9 for air.

Closure of the Reynolds averaged Navier-Stokes equations requires a model for evaluating the eddy viscosity μ_T. Several models of varying degree of accuracy and complexity have been developed over the years to compute the eddy viscosity. The most widely used models are the simple algebraic or zero-equation models of Cebeci and Smith [52] and Baldwin and Lomax [53]. The Baldwin-Lomax model has become the de facto standard model for most engineering calculations. These models give acceptable engineering approximations for attached flows or flows with moderate amounts of separation. Models of this type are based on Prandtl's mixing length theory [51] which relates the eddy viscosity to an appropriate length scale and characteristic turbulent velocity. In the algebraic models, these values are computed from empirically defined relations. Higher order models have been developed that use ordinary or partial differential equations to compute the appropriate quantities. Examples of these are the one equation models of Johnson and King [54] and Spalart and Allmaris [55] and the two equation Jones-Launder k-ε model [56]. The Baldwin-Lomax model has been used exclusively in the research. A detailed description of the model is given in Appendix A.

2.4 The Governing Equations in General Coordinates

For arbitrary geometries, the solution of the Euler or Navier-Stokes equations is normally performed in general curvilinear coordinate systems. This allows irregular boundaries such as the curved surfaces of wings and fuselages to be mapped to align with one or more of the curvilinear coordinates. For the differential forms of the governing equations, a general coordinate transformation must be applied to the equations to map the Cartesian system into the curvilinear system [51]. The partial derivatives in the Cartesian system are recast in terms of partial derivatives in the curvilinear system scaled by appropriate transformation metrics by means of the chain rule for differentiation. In contrast, the integral formulations of Equations (2.1) and (2.3) can be applied directly in a curvilinear system. All that is required is a consistent procedure for defining the cell face areas, cell face unit normal and tangential vectors, and cell volumes. For both the integral and differential forms of the governing equations, the inviscid flux normal to a cell face with area |S| is given by

$$F_n = \begin{pmatrix} \rho(U_n - V_n) \\ \rho u(U_n - V_n) + p n_x \\ \rho v(U_n - V_n) + p n_y \\ \rho w(U_n - V_n) + p n_z \\ E(U_n - V_n) + p U_n \end{pmatrix} |S| \qquad (2.27)$$

where

$$U_n = u n_x + v n_y + w n_z \,, \; V_n = x_t n_x + y_t n_y + z_t n_z \qquad (2.28)$$

This is equivalent to the inviscid flux in a set of general curvilinear coordinates (ξ,η,ζ) when the metric quantities required in the general coordinate transformation are defined geometrically as

$$k_x = \frac{S_{kX}}{V}, k_y = \frac{S_{kY}}{V}, k_z = \frac{S_{kZ}}{V}, k_t = -(x_t k_x + y_t k_y + z_t k_z) \tag{2.29}$$

where k is one of the curvilinear coordinates ξ, η, or ζ, the quantities S_{kX}, S_{kY}, and S_{kZ} are the compenents of the area vector of a surface in the curvilinear system of where k is constant, and V is the cell volume. The parameters k_x, k_y, and k_z are equivalent to the partial derivatives of the curvilinear coordinates with respect to the Cartesian system. Substitution of these metrics into Equations (2.27) yields the inviscid flux in the curvilinear system through a cell face. For example, the inviscid flux through a face where ξ is constant can be written as

$$\hat{F} = \frac{1}{J}\begin{pmatrix} \rho\overline{U} \\ \rho u\overline{U} + p\xi_x \\ \rho v\overline{U} + p\xi_y \\ \rho w\overline{U} + p\xi_z \\ \rho H\overline{U} - p\xi_t \end{pmatrix} \tag{2.30}$$

where $\overline{U} = \xi_x u + \xi_y v + \xi_z w + \xi_t$ is the contravariant velocity, J =1/V is the Jacobian of the general coordinate transformation, and H=(E+p)/ρ is the total enthalpy. Therefore, the

evaluation of the integral form of the governing equations on a cell volume in a curvilinear system is equivalent to the differential form in general coordinates when metric quantites are defined consistently. Similar relationships exit for the other coordinate directions. The viscous flux through ξ can be written

$$\hat{F}_v = \frac{1}{J}\left(F_{vX}\xi_x + F_{vY}\xi_y + F_{vZ}\xi_z\right) \tag{2.31}$$

The differential form of the governing equations in general curvilinear coordinates can then be written as

$$\frac{\partial \hat{Q}}{\partial t} + \frac{\partial(\hat{F} - \hat{F}_v)}{\partial \xi} + \frac{\partial(\hat{G} - \hat{G}_v)}{\partial \eta} + \frac{\partial(\hat{H} - \hat{H}_v)}{\partial \zeta} = 0 \tag{2.32}$$

where $\hat{Q} = Q/J$ and $\hat{G}$, $\hat{H}$, $\hat{G}_v$, and $\hat{H}_v$ are inviscid and viscous flux vectors in the η and ζ analogous to Equations 2.31 and 2.32.

CHAPTER III

NUMERICAL FORMULATIONS OF THE EULER AND NAVIER-STOKES EQUATIONS

This chapter describes the numerical discretization of the governing equations by the finite volume method used in the baseline and distributed flow solvers. The spatial discretization procedures for the Euler and Navier-Stokes equations are presented along with the methods used to compute metric quantities. The artificial dissipation models required to stabilize the numerical schemes are discussed. The implementation of the numerical boundary conditions is described. The explicit and implicit integration methods used to solve the governing equations are presented for both steady and unsteady flow. Finally, the special modifications required for unsteady analyses on moving grid systems are given.

3.1 Finite Volume Spatial Discretization Procedure

In the finite volume method, the computational domain is subdivided into a grid of discrete volumes or cells. The integral forms of the governing equations given by Equations (2.1) and (2.3) are applied locally to each discrete volume. This leads to semi-discrete numerical schemes in which the spatial and temporal terms can be uncoupled in a manner similar to the classical Method of Lines [44] and evaluated by different methods.

Finite volume schemes are classified as "cell-centered" schemes if the flow variables are defined at some point in the cell as a volumetric average of the variables over the cell and "cell-vertex" schemes if the flow variables are defined at the vertices of the cell [57]. In three-dimensional structured grid systems, the volumes are hexahedrons. In unstructured grid systems, the volumes are usually tetrahedral elements similar to those used in finite element formulations [58]. For both classes of finite volume schemes, the vertices of the volume are defined by their Cartesian coordinates. This eliminates the need to specify a global coordinate transformation for curvilinear mesh systems. The connections between the vertices that define the edges of the volume are normally taken to be straight lines. Therefore, metric quantities such as cell volume and the areas and normal vectors of cell faces can be computed using geometric relations [46]. The finite volume scheme used in the baseline code is the structured cell-centered scheme introduced by Jameson, Schmidt, and Turkel [44].

Construction of the finite volume scheme begins with the discretization of the spatial integrals of the flux terms in Equations (2.1) and (2.3). This is done by replacing the integrals with a summation over the six faces of the hexahedron. The flux integral can then be approximated as

$$\int_S \vec{F} \bullet \vec{n} dS \cong \sum_{m=1}^{6} \vec{F}_m \bullet \vec{n}_m \, | \vec{S}_m | \qquad (3.1)$$

where m is one of the cell faces and $| \vec{S}_m |$ is the area of face. Therefore, the inviscid and viscous fluxes must be defined at the cell faces. This requires definitions for the metric quantities.

The volume integrals in Equation (2.1) are evaluated as the product of the average value of the conserved variables in the cell and the cell volume. Therefore, the volume integral at time t_2 for a cell denoted by subscript J becomes

$$\int_{V(t_2)} Q dV = Q_J V_J|_{t_2} \tag{3.2}$$

To be consistent with the procedures used to compute the surface flux, it is more convenient to interpret Q_J as a value defined at some average point in the cell. This removes the exact equality implied by Equation (3.2) and makes the relation approximate. For simplicity, the average point is taken to be the cell center defined by the average of the position vectors of the cell vertices.

The time integral of the flux over each face m in Equation (3.1) can be approximated by

$$\int_{t_1}^{t_2} F_n dt \approx [(1-\theta)F_n|_{t_1} + \theta F_n|_{t_2}]\Delta t \tag{3.3}$$

where $\Delta t = t_2 - t_1$, F_n is the flux normal to the cell face, and θ is a parameter used to define the type of time integration scheme used. Values of θ greater than zero yield implicit integration schemes and a value of zero yields an explicit scheme. For implicit schemes, the value of the flux at t_2 is not known and must be linearized about the previous time level t_1. Replacing t_2 and t_1 with the level indices n and n+1 the discretized form of Equation (2.1) becomes

$$(QV)^{n+1} - (QV)^n + \Delta t \sum_{m=1}^{6} \left((1-\theta) F_n^n + \theta F_n^{n+1} \right)_m S_m = 0 \qquad (3.4)$$

3.1.1 Calculation of Metric Quantities

The discretizations given by Equation (3.1) and (3.2) require definitions for the three components of the cell face area vector of each of the six faces of the cell. The cell volume is also required. The surface area vector on a face can be computed using the cross-product of two of the diagonal vectors of the face [46]. Therefore, for the computational cell shown in Figure 3.1, the area vector $\vec{S}_{4873}$ for the surface with vertices 3, 4, 7, and 8 can be computed as

$$\vec{S}_{4873} = \frac{1}{2} (\vec{r}_{74} \times \vec{r}_{83}) \qquad (3.5)$$

where $\vec{r}$ is a vector drawn from the point defined by the second subscript to the point defined by the first subscript. Equation (3.5) is used for all the other faces of the cell. The coordinate system is assumed to be right-handed. Therefore, the cross-products must be defined such that the outward drawn normals of the area vectors are positive.

The calculation of the cell volume is performed using the computationally efficient procedure proposed by Kordulla and Vinokur [59]. The cell volume is divided in two separate portions. Each of these portions are subdivided into three tetrahedra as shown in Figure 3.2. The total volume of the cell can then be defined as the sum of three vector triple products of the area vectors of three of the cell faces and one of the diagonals of the cell.

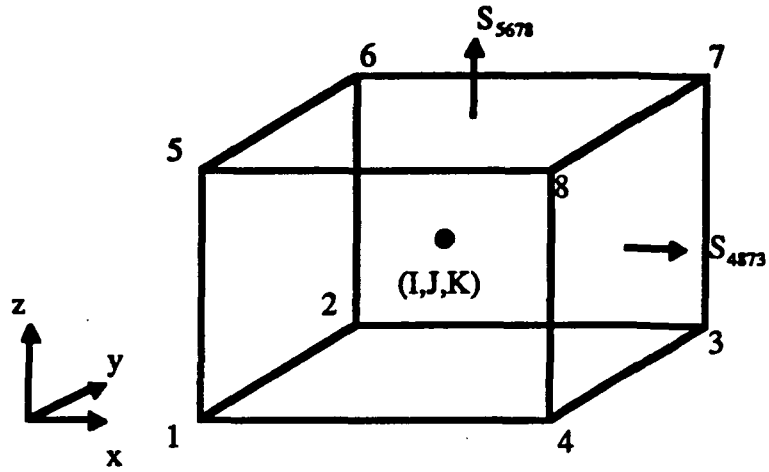


Figure 3.1 Computational Cell

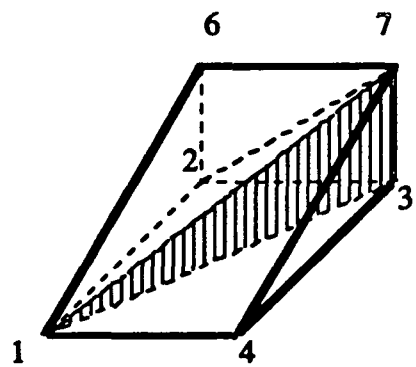


Figure 3.2 Decomposition of Tetrahedron

The volume can be written as

$$V = \frac{1}{3}\vec{r}_{71} \bullet (\vec{S}_{1485} + \vec{S}_{1234} + \vec{S}_{1562}) \tag{3.6}$$

3.1.2 Calculation of Inviscid Fluxes

In the cell-centered scheme, fluxes are approximated at the cell faces using interpolated values of either the flow variables or the flux vector defined at the neighboring cell centers. The two procedures give equivalent results for smooth flows. For flows with shocks, averaging of the flux vector is preferred because it provides the correct shock jump conditions for shock waves aligned with the cell face. For cell centered values defined at indices J and J+1, the inviscid flux at the face J+1/2 is given by the centered approximation

$$F_{J+1/2} = \frac{1}{2}\left(F(Q_L) + F(Q_R)\right) \tag{3.7}$$

where Q_L and Q_R define interpolated states of the conserved variables on either side of the cell face used to construct the flux. A first order approximation to the flux is obtained if Q_L and Q_R are defined as the values of Q at the cell centers J and J+1. Higher order approximations can be obtained by using the Monotone Upstream Centered Conservation Law (MUSCL) interpolation schemes introduced by van Leer [60] to define Q_L and Q_R. Insertion of the first order approximation to the flux into Equation (3.1) produces expressions that are equivalent to a central difference approximation of the spatial derivative of the flux on a uniform mesh. Central difference approximations suffer from a

destabilizing odd-even decoupling of the solution at adjacent grid points. Therefore, an explicit artificial dissipation term is normally added to the inviscid flux to maintain stability.

3.1.3 Calculation of Viscous Fluxes

The viscous fluxes are computed using the same averaging procedure used in the first order approximation to the inviscid fluxes. This requires that both the flow variables and their gradients be defined at cell centers. The definition of the derivatives of the flow variables can be obtained by application of the divergence theorem [40]. For example, the derivative, $\partial u/\partial y$, can be expressed within a volume, V, in terms of the values of u on the surface, S, bounding the volume as

$$\int_V \frac{\partial u}{\partial y} dV = \int_S u dS_y \tag{3.8}$$

where dS_y is the projection of the surface element, dS, in the y direction. This can be discretized at cell centers defined by the subscript L whose neighbors across a face, m, are defined by the subscript M as

$$\left.\frac{\partial u}{\partial y}\right|_J = \frac{1}{V}\sum_{m=1}^{6} \bar{u}_{JM} \, |S_y|_m \tag{3.9}$$

where $|S_y|_m$ is the y component of the area vector on face, m, and

$$\bar{u}_{JM} = \frac{1}{2}(u_J + u_M)_m \tag{3.10}$$

Similar expressions can be obtained for the other derivatives.

A second procedure for computing the viscous flux is to evaluate the derivatives of the flow variables on the cell faces by means of chain-rule differentiation of averaged variables. This procedure exits a second option in the baseline code but was not used in this research. Both procedures yield comparable results. The first approach is faster but requires more memory than the second procedure.

3.2 Artificial Dissipation Models

All spatial discretization schemes for the governing equations introduce error components of different wave lengths into the solution that can destabilize the solution process if allowed to grow unchecked. In addition to the previously mention odd-even decoupling problem inherent in central difference approximations, an aliasing phenomenon can occur in which short wave length error components interact with each other to form destabilizing long wave length components. Therefore, some form of dissipation must be explicitly added to the solution to damp the high frequency waves. This is true for both the Euler and Navier-Stokes equations because the dissipation introduced by the viscous terms in the Navier-Stokes equations is normally not sufficient to damp all of the high-frequency error components.

In addition to errors introduced by the discretization scheme, additional problems can occur in the solution of the Euler equations when shock waves are captured in the solution. The Euler equations contain no dissipative mechanism to enforce the correct entropy condition required by the Second Law of Thermodynamics. Therefore, physically unrealistic solutions such as expansion shocks can occur. Shock capturing problems can also occur in the Navier-Stokes equations if the levels of viscous dissipation are

inadequate. Inside shock waves, the dissipation introduced by molecular viscosity acts to attenuate short scale motion. This means that any numerical dissipation terms introduced in the solution must effectively damp the short scale motion in the region of the shock without degrading the global solution accuracy.

An effective numerical dissipation model for both the Euler and Navier-Stokes equations must be able to damp both the high frequency components and ensure correct shock capturing without seriously degrading the accuracy of the solution. The most accurate models are characteristic based procedures such as Roe's scheme [61] which modify the flux calculation using the characteristic waves entering and leaving a cell interface to provide an appropriate upwind bias in the flux. For Roe's scheme, this is equivalent to solving an approximate Riemann problem at each cell interface. However, characteristic based schemes are computationally expensive. An alternate approach used for central difference schemes is the adaptive dissipation approach introduced by Jameson et. al. [44]. The adaptive approach uses a blended combination of first and third differences of the flow variables to form a dissipative flux at a cell interface that is used to modify the inviscid flux. The modified flux at a cell face J+1/2 can the be written as

$$\hat{F}_{J+1/2} = F_{J+1/2} - d_{J+1/2} \tag{3.11}$$

where $d_{J+1/2}$ is the dissipative flux. Evaluation of Equation (3.1) using the modified flux given by Equation (3.11) yields a net dissipative flux composed of second and fourth differences. The fourth differences are effective in damping the high-frequency errors. The second differences mimic the effect of the viscous terms in the vicinity of shock waves. Scaling factors are applied to the difference components to minimize the effects of the dissipation terms in smooth regions and provide the appropriate levels of each

dissipation term in non-smooth regions. These scaling factors can take the form of scalar coefficients, flux limiters [62], or a matrix coefficient based on characteristic information at the cell face [63]. The baseline code has several different types of numerical dissipation schemes as solution options. Only the four options used in this research will be discussed. In the baseline code, these dissipation models are referred to as the Standard Adaptive Dissipation (SAD) model, the Modified Adaptive Dissipation (MAD) model, the Flux-limited Adaptive Dissipation Model (FAD) model, and the Matrix Based Dissipation (MBD) model.

3.2.1 Standard Adaptive Dissipation

In the standard adaptive dissipation scheme, the dissipative flux, d, in Equation (3.11) is constructed from a combination of first and third differences along each coordinate direction. For example, the dissipative flux along the computational coordinate ξ, can be written in differential form as

$$d_\xi = \rho(A)(\varepsilon_2 \frac{\partial Q}{\partial \xi} - \varepsilon_4 \frac{\partial^3 Q}{\partial \xi^3}) \tag{3.12}$$

where $\rho(A)$ is the spectral radius of the flux Jacobian matrix, $A=\partial F_\xi / \partial Q$, and the parameters ε_2 and ε_4 are weighting parameters used to control the amount of dissipation added by the difference terms. The summation of the modified flux over the cell faces then leads to the required second and fourth difference terms. In the following discussion, the indices I, J, and K identify the position of the center of a cell volume in a three-dimensional computational grid of unit spacing. The position of the cell faces in each of

the coordinate directions is obtained by adding or subtracting 1/2 to the indices of the cell centers. The dissipative flux in the I direction can then be discretized as

$$d_{I+1/2,J,K} = \varepsilon_2 e_{I+1/2,J,K} - \varepsilon_4 (e_{I+3/2,J,K} - 2e_{I+1/2,J,K} + e_{I-1/2,J,K}) \tag{3.13a}$$

where

$$e_{I+1/2,J,K} = \alpha \left(Q_{I+1,J,K} - Q_{I,J,K} \right) \tag{3.13b}$$

$$\alpha = \frac{1}{2} \left(\Lambda_{I+1,J,K} + \Lambda_{I,J,K} \right) \tag{3.13c}$$

$$\Lambda_{I,J,K} = \lambda^I_{I,J,K} + \lambda^J_{I,J,K} + \lambda^K_{I,J,K} \tag{3.13d}$$

The parameters λ^I, λ^J, and λ^K are the spectral radii of the flux Jacobian matrices in each of the coordinates directions at the cell center. For example, the spectral radii in the I direction is defined as

$$\lambda^I = | \overline{U}_{I,J,K} | + c \sqrt{\vec{S}^I_{I,J,K} \bullet \vec{S}^I_{I,J,K}} \tag{3.14a}$$

where

$$\overline{U} = (u - x_t) S^I_x + (v - y_t) S^I_y + (w - z_t) S^I_z \tag{3.14b}$$

$$\vec{S}^I_{I,J,K} = \frac{1}{2} \left(\vec{S}_{I+1/2,J,K} + \vec{S}_{I-1/2,J,K} \right) \tag{3.14c}$$

and S is an area vector defined at either the cell center or the cell face and c is the local speed of sound. Similar expressions are written for the other directions. The factors ε_2 and ε_4 are defined as

$$\varepsilon_2 = \kappa_2 \bar{\upsilon}_{I+1/2,J,K} \, , \varepsilon_4 = \max(0, \kappa_4 - \varepsilon_2) \tag{3.15}$$

where

$$\bar{\upsilon}_{I+1/2,J,K} = \max(\upsilon_{I+2,J,K}, \upsilon_{I+1,J,K}, \upsilon_{I,J,K}, \upsilon_{I-1,J,K}) \tag{3.16}$$

and $\upsilon_{I,J,K}$ is a switching coefficient constructed from the normalized second difference of pressure. This parameter detects the presence of shocks and switches between the first and third differences at shock waves. The pressure switch has the form

$$\upsilon_{I,J,K} = \left| \frac{P_{I+1,J,K} - 2P_{I,J,K} + P_{I-1,J,K}}{P_{I+1,J,K} + 2P_{I,J,K} + P_{I-1,J,K}} \right| \tag{3.17}$$

The parameters κ_2 and κ_4 are specified constants with magnitudes of order one. Standard values of κ_2 and κ_4 are 1/2 and 1/32 respectively. The pressure switch ensures that large amounts of dissipation are introduced near shock waves and at stagnation points. However, this form of the dissipation can be excessively dissipative for viscous flow simulations and lead to erroneous values of skin friction and heat transfer in the boundary layer [64].

3.2.2 Modified Adaptive Dissipation

The Standard Adaptive Dissipation model can be modified to reduce the excessive levels of dissipation introduced in viscous simulations by redefining the parameters scaling the first differences in Equation (3.13b). For example, the scaling in the I direction becomes

$$\alpha = \frac{1}{2}\left(\lambda^{I}_{I+1,J,K} + \lambda^{I}_{I,J,K}\right) \tag{3.18}$$

This leads to different scalings of the dissipation along each of the coordinate directions. However, a directional dependency is introduced that can degrade the convergence rate of the solution on highly stretched computational meshes. Swanson and Turkel [65] have demonstrated that this problem can be alleviated by redefining the spectral radii used in Equation (3.18) to produce an anisotropic scaling. The modified spectral radii have the form

$$\bar{\lambda}^{I} = \lambda^{I}\left[1 + \max\left(\left(\frac{\lambda^{J}}{\lambda^{I}}\right)^{m}, \left(\frac{\lambda^{K}}{\lambda^{I}}\right)^{m}\right)\right] \tag{3.19}$$

where the exponent m has values of order one. In the baseline code, m is set to 1/2. The parameter ϵ_2 is also modified by restricting its maximum value to 1/2.

3.2.3 Flux-Limited Adaptive Dissipation

Both the SAD and MAD formulations exhibit pre- and post-shock oscillations without careful tuning of the κ_2 parameter. Jameson [62] has introduced an alternative formulation of the adaptive dissipation by redefining the coefficients scaling the first and

third differences in terms of flux limiters similar to those employed in MUSCL difference schemes. The flux limiters act to locally switch the central differencing scheme to a first-order upwind scheme at shock waves allowing shocks to be captured without oscillations. In the flux-limited scheme, the dissipative flux at a cell face is defined as

$$d_{I+1/2,J,K} = L(e_{I+3/2,J,K}, e_{I+1,2,J,K}) - 2e_{I+1/2,J,K} + L(e_{I+1/2,J,K}, e_{I-1/2,J,K}) \qquad (3.20)$$

where L is the limiter function and e is the first difference of the solution defined by Equation (3.13b) but scaled by the factor

$$\alpha = \frac{\beta}{2}\left(\lambda^{I}_{I+1,J,K} + \lambda^{I}_{I,J,K}\right) \qquad (3.21)$$

and λ is again the spectral radii. The parameter, β, is defined as

$$\beta = \min\left(\frac{1}{2}, \kappa_2 \bar{\upsilon}_{I+1/2,J,K} + \kappa_4\right) \qquad (3.22)$$

The limiter function , L, is defined for two arguments, a and b, as

$$L(a,b) = [s(a)+s(b)]\ \min(|a|,|b|) \qquad (3.23)$$

where s(a) equals 1/2 when a is greater than or equal to zero and assumes a value of -1/2 when a is less than zero.

3.2.4 Matrix Based Dissipation

The dissipation given by Equation (3.12) applies the same scaling to all the components of the flux. This scale factor is approximately the largest eigenvalue of the Jacobian matrix, A. This can lead to excessive smearing in viscous regions because all the characteristic waves that form the flux are scaled by the same coefficient which is proportional to the fastest wave speed. Swanson and Turkel [63] have proposed an alternate formulation similar to Roe's scheme where the scalar coefficient in Equation (3.12) is replaced by a matrix defined by

$$|A| = T|\Lambda|T^{-1} \tag{3.24}$$

where Λ is a diagonal matrix given the similarity transformation

$$\Lambda = T^{-1}AT \tag{3.25}$$

The transformation matrices, T and T^{-1}, are constructed from the eigenvectors of the flux Jacobians [66]. For the Euler and Navier-Stokes equations, the diagonal elements of Λ are given by the five eigenvalues of the flux Jacobian matrix, A, which can be written for each coordinate direction as

$$\lambda_{1,2,3} = \overline{U}\,,\ \lambda_4 = \overline{U} + c\sqrt{S_x^2 + S_y^2 + S_z^2}\,,\ \lambda_5 = \overline{U} - c\sqrt{S_x^2 + S_y^2 + S_z^2} \tag{3.26}$$

where $\overline{U}$ is the contravariant velocity given by Equation (3.14b) and S is the area vector along any coordinate direction. When the matrix, A, can be diagonalized, |A| can be replaced by a matrix function of the form [67]

$$|A| = |\lambda_1| I + \left(\frac{|\lambda_4| + |\lambda_5|}{2} - |\lambda_1| \right) \left[\frac{\gamma - 1}{c^2} E_1 + \frac{1}{S_x^2 + S_y^2 + S_z^2} E_2 \right] \tag{3.27}$$

$$+ \frac{|\lambda_4| - |\lambda_5|}{2} \left(\frac{1}{c\sqrt{S_x^2 + S_y^2 + S_z^2}} \right) \left[E_3 + (\gamma - 1) E_4 \right]$$

where I is the identity matrix. The matrices E_1, E_2, E_3, and E_4 are written as

$$E_1 = \begin{bmatrix} \phi & -u & -v & -w & 1 \\ u\phi & -u^2 & -vu & -wu & u \\ v\phi & -uv & -v^2 & -wv & v \\ w\phi & -uw & -vw & -w^2 & w \\ H\phi & -uH & -vH & -wH & H \end{bmatrix}, \tag{3.28a}$$

$$E_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ -S_x\overline{U} & S_x^2 & S_xS_y & S_xS_z & 0 \\ -S_y\overline{U} & S_xS_y & S_y^2 & S_zS_y & 0 \\ -S_z\overline{U} & S_xS_z & S_yS_z & S_z^2 & 0 \\ -\overline{U}^2 & \overline{U}S_x & \overline{U}S_y & \overline{U}S_z & 0 \end{bmatrix}, \tag{3.28b}$$

$$E_3 = \begin{bmatrix} -\bar{U} & S_x & S_y & S_z & 0 \\ -u\bar{U} & uS_x & uS_y & uS_z & 0 \\ -v\bar{U} & vS_x & vS_y & vS_z & 0 \\ -w\bar{U} & wS_x & wS_y & wS_z & 0 \\ -H\bar{U} & HS_x & HS_y & HS_z & 0 \end{bmatrix}, \tag{3.28c}$$

$$E_4 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ S_x\phi & -uS_x & -vS_x & -wS_x & S_x \\ S_y\phi & -uS_y & -vS_y & -wS_y & S_y \\ S_z\phi & -uS_z & -vS_z & -wS_z & S_z \\ \bar{U}\phi & -u\bar{U} & -v\bar{U} & -w\bar{U} & \bar{U} \end{bmatrix} \tag{3.28d}$$

where H is the total enthalpy and $\phi = (u^2 + v^2 + w^2)/2$. The special form of |A| given by Equation (3.27) allows an arbitrary vector of to be multiplied by |A| without computing the full matrix. Therefore, terms such as $|A|\varepsilon_2\partial Q/\partial\xi$ can be computed efficiently. The flow variables used in the matrix equation are computed at each cell face by simple arithmetic averages of the variables on either side of the face. Roe's scheme can be recovered by replacing the arithmetic averaged variables with Roe's averaged variables.

As in Roe's scheme, the eigenvalues defined by Equation (3.26) must be modified to maintain sufficient levels of dissipation at stagnation points and sonic lines because λ_1 is zero at stagnation points while λ_4 and λ_5 vanish at sonic lines. Therefore, the values of the eigenvalues are limited as

$$\left|\tilde{\lambda}_1\right| = \max\left(|\lambda_1|, V_l\rho(A)\right) \tag{3.29a}$$

$$\left|\tilde{\lambda}_4\right| = \max\left(|\lambda_4|, V_n\rho(A)\right) \tag{3.29b}$$

$$\left|\tilde{\lambda}_5\right| = \max\left(|\lambda_5|, V_n\rho(A)\right) \tag{3.29c}$$

where V_l and V_n are constants and $\rho(A)$ is the spectral radius of A defined by Equation (3.14a). The constants, V_l and V_n, are determined from numerical experimentation and have typical values of 0.3 and 0.2, respectively. Further details of the formulation and application of the matrix based dissipation are given in References [68] and [69].

3.3 Boundary Conditions

The TEAM code has options for a wide variety of boundary conditions for both external flows such as wings and fuselages and internal flows such as engine inlets and exhaust nozzles. The multi-zone formulation of the baseline code enables the simulation of the flow about complex geometries such as a complete airplane. A patched or composite grid system can be generated in which the computational domain is broken into several grid zones that abut at defined zonal boundaries. In the baseline code, boundary values are defined at the centers of a layer of "ghost" or image cells that surround the boundary surfaces of each zone. The first layer of cells inside the boundary are designated as boundary cells. For external flow simulations, the boundary conditions must be defined at the far-field, body surfaces, and fluid boundaries such as wakes, symmetry planes and zonal interfaces. Each type of boundary condition will be discussed separately.

3.3.1 Far-Field Boundary Conditions

The far-field boundary conditions in the baseline code are obtained using the non-reflecting characteristic boundary conditions introduced in Reference [44]. These boundary conditions define the flow quantities in the boundary cells using the appropriate number of characteristics entering and leaving the boundary. This procedure allows the flow to enter and leave the finite computational domain without generating reflected waves that can destabilize or retard the solution.

For subsonic flows, the boundary conditions are obtained from the local Riemann invariants for a one-dimensional flow normal to the cell faces that make up the boundary surface. The subsonic inflow boundary conditions are obtained from two incoming characteristics and one outgoing characteristic. At the boundary, the Riemann invariants can be defined in terms of the free-stream values at infinity and values extrapolated along the characteristic line from the interior domain as

$$R_\infty = q_{n\infty} - \frac{2}{\gamma - 1} c_\infty \tag{3.30a}$$

$$R_e = q_{ne} + \frac{2}{\lambda - 1} c_e \tag{3.30b}$$

where $q_{n\infty}$ and q_{ne} are magnitudes of normal velocities of the flow variables at the cell face defined using either the free-stream values of the flow variables or the values defined in the boundary cells and the normal vector at the cell face. The variable c is the speed of sound. The Riemann invariants are used to define the magnitude of the normal component of velocity and the speed of sound on the boundary as

$$q_{nb} = \frac{1}{2}(R_+ + R_-), c_b = \frac{\gamma - 1}{4}(R_+ - R_-) \tag{3.31}$$

The values for density, pressure and velocity in the ghost cells can now be defined using the values of q_n and c on the boundary and values of entropy and tangential velocity obtained from either the free-stream or extrapolated flow variables. The sign of the contravariant velocity defined by Equation (3.14b) and obtained from the extrapolated flow variables and the metrics at the cell face is used to distinguish between inflow and outflow boundaries. For inflow boundaries, the flow variables are defined using the following relations

$$s = \rho_-^\gamma / p_-, \rho = \left(sc_b^2/\gamma\right)^{1/(\gamma-1)}, p = \rho c_b^2/\gamma \tag{3.32a}$$

$$\vec{q} = \left(\vec{q}_- - q_{n-}\vec{n}\right) + q_{nb}\vec{n} \tag{3.32b}$$

where s in the entropy and n is the normal vector at the cell face. For outflow boundaries, the flow variables are obtained using Equations (3.32a) and (3.32b) with the values at infinity replaced by the values extrapolated from the interior.

For supersonic flow, all characteristics directions either enter or leave the computational domain. At supersonic inflow boundaries, the flow variables are fixed to their free-stream values. For supersonic outflow boundaries, the flow variables are set to the values in the boundary cells of the outflow boundary. This is equivalent to a zero order extrapolation of the flow variables.

3.3.2 Solid Surface Boundary Conditions

For both the Euler and Navier-Stokes Equations, the solid surface boundary conditions are imposed by setting the mass flux normal to the boundary to zero. In the

inviscid flux, the zero mass flux condition is implemented by setting the contravariant component of velocity normal to the boundary face to zero. This is accomplished automatically for the Navier-Stokes equations by imposing the no-slip condition obtained by setting the velocities u, v, and w to zero for steady flows and equal to the surface velocities, x_t, y_t and z_t, for unsteady flows. For Euler simulations, the surface velocities can be obtained from extrapolated values of the contravariant velocities tangent to the cell face. In Navier-Stokes simulations, the no-slip conditions is imposed by setting the values in the ghost cells such that a simple average of the ghost cell values with the boundary cell values gives the appropriate surface velocity. For steady simulations, this implies that the ghost cell values are obtained by reflection of the three components of velocity in the adjacent boundary cells. For the cell centered finite volume scheme, velocities are not required on the surface to form the inviscid flux. Only the pressure at the cell face is required.

The baseline code has several different procedures of varying degree of accuracy for defining the surface pressure. For steady flow simulations, the simplest and most stable approximation is a zero order extrapolation of the pressure from the boundary cells. This is consistent with the assumption of a zero normal pressure gradient. A more accurate but less stable procedure is to compute the pressure from the normal momentum equation.

For unsteady inviscid flows, the pressure gradient at the surface can obtained from the solution of a normal momentum equation of the form [70]

$$-\rho(\vec{u}_b - \vec{v}_b)\bullet\frac{D\vec{n}}{Dt} + \rho\vec{n}\bullet\frac{D\vec{v}_b}{Dt} = -\vec{n}\bullet\nabla p \qquad (3.33)$$

where u_b and v_b are the flow and surface velocity vectors at the surface cell face, n is the normal vector at the cell face, and ∇p is the gradient of pressure. The operator D /Dt is the material derivative, $\partial /\partial t + (v \bullet \nabla)$. For viscous flows, Equation (3.33) reduces to

$$\vec{n} \bullet \nabla p = -\rho \vec{n} \bullet \frac{D\vec{v}_b}{Dt} \tag{3.34}$$

For inviscid steady flows, Equation (3.33) becomes

$$\vec{n} \bullet \nabla p = \rho \vec{u}_b \bullet (\vec{u}_b \bullet \nabla)\vec{n} \tag{3.35}$$

Equation (3.35) reduces to the zero normal pressure gradient assumption for steady viscous simulations. Solution of the preceding equations requires values of velocity, density, and pressure extrapolated from the interior. Several different approaches for defining the extrapolated variables used in the solution of the normal momentum equation were evaluated in the development of the baseline code. A complete description of the various methods used in the TEAM code is given in Reference [40].

3.3.3 Fluid and Symmetry Plane Boundary Conditions

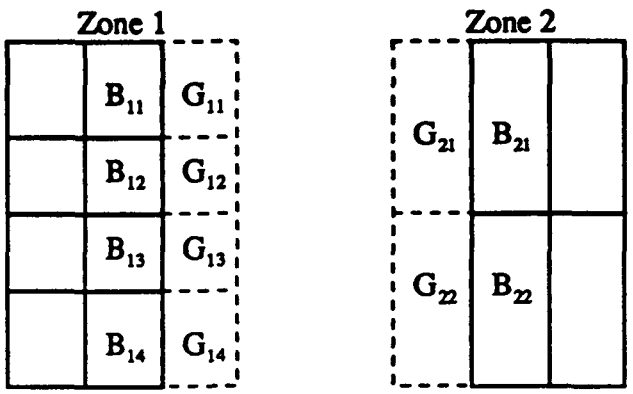
Fluid boundaries occur when two grid zones abut at a common interface or at branch cut where two boundaries or surfaces of the same grid zone coincide. Both types of fluid boundaries are characterized by an interface surface that connects the two zones or grid segments. The baseline code supports three distinct classes of interfaces. The first type of interface is referred to in the TEAM code as a Class 1 interface. For Class 1 interfaces, the grid points on the interface are the same for both zones or grid segments. A Class 2 interface is formed when the interface is composed of a dense grid from one zone and a

coarse grid from the abutting zone that shares common points with the dense grid at some regular mesh spacing on the dense grid. For instance, the coarse grid might consist of every second or third point in the dense grid. In both the Class 1 and Class 2 interfaces, the zones are connected at common grid points. The baseline code allows a third class of interface, Class 3, in which none of the grid points in the two zones are required to coincide. The only restriction on the Class 3 interface is that the interface points must lie on the same level surface.

The accuracy and efficiency of multi-zone discretization procedures depends on the level of global conservation maintained during the solution. It is important that ghost cell values at zonal interfaces are specified in a manner that assures global conservation. For Class 1 boundaries, conservation is maintained by injecting the boundary cell values from the abutting zones into the ghost cells of the adjoining zone. Therefore, the spatial order of accuracy of the flux conservation at the interface is the same as for the interior cells. However, Allmaras [71] has shown that second order accuracy is not possible for Class 2 and Class 3 boundaries if strict conservation is enforced. In addition, first order accuracy can only be obtained using appropriate characteristic extrapolation procedures that are computationally expensive. In the baseline code, a zeroth-order extrapolation procedure proposed by Allmaras is used that maintains conservation across the fluid boundaries at Class 2 and Class 3 interfaces.

As shown in Figure 3.3, the definition of the ghost cells for the coarser of the two grids at a Class 2 or Class 3 interface is defined by an area weighted average of boundary cell data from both the coarse and fine grids. In Figure 3.3, the boundary cells in each zone are designated by B, the ghost cells by the letter G, and S is the corresponding cell face area. The ghost cell values for the dense grid is defined using the values in the boundary cell whose interface surface the corresponding surfaces in the dense grid. For Class 3 interfaces, the number of grid cells and areas used in the extrapolation is set to a

maximum value, usually four, to reduce storage requirements. In addition, each grid surface must be searched to define the appropriate cells and areas to be used in the extrapolation.



$G_{11}=B_{21}$ $\quad G_{21}=C_1B_{11}+C_2B_{12}+C_3B_{21}$

$G_{12}=B_{21}$ $\quad G_{22}=C_4B_{13}+C_5B_{14}+C_6B_{22}$

$G_{13}=B_{22}$ $\quad C_1=S_{11}/S_{21},\ C_2=S_{12}/S_{21},\ C_3=C_1\cdot C_2-1$

$G_{14}=B_{22}$ $\quad C_4=S_{13}/S_{22},\ C_5=S_{14}/S_{22},\ C_6=C_4\cdot C_5-1$

Figure 3.3 Class 2 and Class 3 Fluid Boundary Conditions

The ghost cell values at planes of symmetry are obtained by reflecting the appropriate boundary cell velocity component across the plane and injecting the remaining flow variables from the boundary cells.

3.4 Explicit Solution Procedure

The explicit solution procedure in the baseline code is the multi-stage Runge-Kutta time stepping scheme introduced by Jameson et. al. [44] in the FLO57 code. Because of the restrictions on solution time step inherent in all explicit formulations, three different types of acceleration techniques are employed in the TEAM code for steady flow simulations. For steady Euler simulations, enthalpy damping can be employed to force the solution to the steady state condition of constant total enthalpy. In addition to enthalpy damping, the implicit residual smoothing procedure introduced by Jameson [72] can be used to increase the maximum time step of the multi-stage scheme. Finally, spatially varying time steps can be used for steady flow simulations to reduce the time required to reach a steady state solution. Spatially varying time steps act as a preconditioner on the discretized equations that relaxes stiffness in the solution due to local variations in the flow variables. The accelerated form of the multi-stage time stepping scheme provides a cost effective solution procedure for the steady Euler equations. However, the severe time step restrictions imposed by the explicit formulation make the scheme ill-suited for Navier-Stokes and unsteady flow simulations.

3.4.1 Multi-stage Time Stepping

The discretized form of Equation (2.3) for stationary grids can be written for each cell volume as

$$\frac{d(QV)}{dt} + F_I - F_V - D = 0 \tag{3.36}$$

where F_I is the net inviscid flux, F_V is the net viscous flux, and D is the net dissipative flux through the cell volume. For stationary grids, the volume ,V, is constant. Therefore, Equation (3.36) can be written as an ordinary differential equation of the form

$$\frac{dQ}{dt} + R(Q) = 0 \tag{3.37}$$

where R(Q) is the residual defined by

$$R(Q) = \frac{1}{V}(F_I - F_V - D) \tag{3.38}$$

A multi-stage explicit time stepping scheme based on the Runge-Kutta method is used in the baseline code to integrate Equation (3.37) in time. The m-stage procedure for advancing the solution from a time level n to n+1 can be written as

$$\begin{aligned}
Q^{(0)} &= Q^n \\
Q^{(1)} &= Q^{(0)} - \alpha_1 \Delta t R^{(0)} \\
Q^{(2)} &= Q^{(1)} - \alpha_2 \Delta t R^{(1)} \\
&\vdots \\
Q^{(m-1)} &= Q^{(m-2)} - \alpha_{m-1} \Delta t R^{(m-2)} \\
Q^{(m)} &= Q^{(m-1)} - \alpha_m \Delta t R^{(m-1)} \\
Q^{n+1} &= Q^{(m)}
\end{aligned} \tag{3.39}$$

where Δt is the time step and the parameters, α_m, are constants that vary with the number of stages and residual smoothing. For most simulations, the viscous and dissipative components of the residual are computed only once and held constant for all stages. For the four stage scheme used in this research, the α coefficients have values in ascending order of 1/4, 1/3, 1/2, and 1. In the absence of residual smoothing, the four stage scheme is stable for Courant (CFL) numbers up to $2\sqrt{2}$ for 1-D model problems. Residual smoothing and enthalpy damping can extend the CFL limit of the four stage scheme to a value of 6.

3.4.2 Enthalpy Damping

Enthalpy damping is implemented for steady Euler simulations by adding a forcing term of the form

$$P_E = \begin{bmatrix} \alpha\rho H \\ \alpha\rho uH \\ \alpha\rho vH \\ \alpha\rho wH \\ \alpha\rho H \end{bmatrix} \tag{3.40}$$

to the residual, R. The parameter ,α, is a user specified constant. In addition, the non-dimensional total energy, total enthalpy and pressure are redefined as

$$E = \bar{\rho}_\infty(\bar{E} - \bar{H}_\infty)/\bar{p}_\infty\,,\; H = \bar{\rho}(\bar{H} - H_\infty),\; p = (\gamma-1)\rho\left(E - \frac{1}{2}(\vec{u}\bullet\vec{u}) + H_\infty\right) \tag{3.41}$$

Therefore, the forcing function acts to drive the value of total enthalpy in each cell to the free-stream value as the solution converges.

3.4.3 Residual Smoothing

The concept of residual smoothing was first introduced by Lerat as part of a family of implicit schemes based on Lax-Wendroff type differencing [73]. Jameson applied the procedure to the Runge-Kutta multi-stage scheme [72]. The basis of residual smoothing is to replace the residual defined by Equation (3.38) with an effective value defined by an implicit smoothing or averaging operator. The implicit smoothing acts as a preconditioner by damping out local high- frequency variations in the residual. The smoothing operation can be written in three dimensions as

$$\left(1-\varepsilon(\delta_x^2+\delta_y^2+\delta_z^2)\right)R_s = R \tag{3.42}$$

where δ^2 is the central difference operator for second derivatives, R_s is the smoothed value of the residual and ε is smoothing parameter that can be either constant or vary in space. In practice, Equation (3.42) is solved by approximately factoring it into a sequence of three scalar tridiagonal operators of the form

$$\left(1-\varepsilon_x\delta_x^2\right)\left(1-\varepsilon_y\delta_y^2\right)\left(1-\varepsilon_z\delta_z^2\right)R_s = R \tag{3.43}$$

For constant values of ε, a one dimensional stability analysis shows that the stability of the central difference scheme can be maintained for a given CFL number when the smoothing coefficients satisfies the following condition

$$\varepsilon \geq \frac{1}{4}\left[\left(\frac{CFL}{CFL_U}\right)^2 - 1\right] \tag{3.44}$$

where CFL_U is the Courant number of the unsmoothed scheme. The smoothing operation is normally applied in each stage of multi-stage scheme. Spatially varying forms of the smoothing coefficient have been developed that provide improve stability and convergence characteristics on highly stretched grids [74]. Both constant and spatially varying smoothing coefficients are available as options in the baseline code. The constant coefficients were used in the majority of the simulations performed with the explicit scheme in this research.

3.4.4 Local Time Stepping

For steady inviscid flow simulations, the fixed time step in Equations (3.39) can be replaced by a spatially varying step defined by

$$\Delta t_{I,J,K} = CFL\left(\frac{V_{I,J,K}}{(\lambda^I_{I,J,K} + \lambda^J_{I,J,K} + \lambda^K_{I,J,K})}\right) \tag{3.45}$$

where $\lambda^I_{I,J,K}$, $\lambda^J_{I,J,K}$, and $\lambda^K_{I,J,K}$ are the spectral radii at the cell centers defined by Equation (3.14a), V is the cell volume, and CFL is the user specified Courant number. For viscous simulations, additional restrictions must be imposed on the time step to account for cell Reynolds number effects on the stability criterion of explicit schemes [51]. This can be accomplished by augmenting the convective spectral radii used in Equation (3.45) with a diffusive term based on an approximation to the spectral radii of the Jacobian matrices of

the viscous flux terms [75]. The general form of the diffusive term along each coordinate direction at a cell center can be written as

$$\lambda_{viscous} = \frac{\mu}{3\rho}\left[\left(\frac{S_x^2 + S_y^2 + S_z^2}{V}\right)\max\left(\frac{3\gamma}{Pr}, 4\right) + \frac{\left(|S_xS_y| + |S_xS_z| + |S_yS_z|\right)}{V}\right] \quad (3.46)$$

The time step can then be computed with the spectral radii terms used in Equation (3.45) replaced by the sum of the convective and viscous spectral radii. These values are also used in the construction of the artificial dissipation. The use of local time steps for steady flow simulations converts the time-accurate multi-stage time stepping scheme to a "pseudo-time" stepping scheme. The magnitude of the step is computed to satisfy stability criterion and is not set to resolve discrete time-varying phenomena. The objective is to obtain a steady state in the smallest possible number of steps. The time step is usually updated at the start of each step. For Euler simulations, the time step can be updated at fixed intervals of up to twenty steps without seriously impacting the convergence rate.

3.5 Implicit Integration Procedure

The severe time step restrictions of the explicit multi-stage scheme make them unsuitable for large scale unsteady viscous simulations. Implicit solution schemes are favored for unsteady flow simulations because they allow much larger time steps to be taken without sacrificing the stability of the scheme. However, the computational effort per step for most practical implicit schemes can be several time the effort for explicit formulations. This is due to the fact that implicit formulations are based on a local

linearization of the flux vector at time level n+1 about the previous time level that produces large banded matrix systems of equations. This linearization is obtained from a local Taylor's expansion of the form

$$F^{n+1} = F^n + \frac{\partial F^n}{\partial Q}\frac{\partial Q^{n+1}}{\partial t}\Delta t + O(\Delta t^2) \tag{3.47}$$

where $\partial F^n / \partial Q$ is the flux Jacobian. Replacing $\partial Q^{n+1} / \partial t$ with

$$\frac{\partial Q^{n+1}}{\partial t} = \frac{Q^{n+1} - Q^n}{\Delta t} + O(\Delta t) \tag{3.48}$$

yields the common form of the linearized flux. For $\Delta Q^{n+1} = Q^{n+1} - Q^n$, Equation (3.47) is approximated by

$$F^{n+1} \cong F^n + \frac{\partial F^n}{\partial Q}\Delta Q^{n+1} \tag{3.49}$$

Substitution of Equations (3.48) and Equations (3.49) into the discretized forms of Equations (2.1) and (2.3) yields a general implicit scheme which has the following differential form in the general curvilinear coordinates ξ, η, and ζ:

$$\left[I + \frac{\alpha \Delta t}{V}\left(D_\xi A^n + D_\eta B^n + D_\zeta C^n\right)\right]\Delta Q^{n+1} = -\frac{\Delta t}{V} R(Q^n) \tag{3.50}$$

where D_ξ, D_η, and D_ζ are difference operators, A^n, B^n, and C^n are the flux Jacobian matrices along each coordinate direction, and I is the identity matrix. The scheme is second-order accurate in time when the parameter, α, equals 1/2. The scheme is first order-accurate for all other values of α. R is the residual defined by

$$R(Q^n) = D_\xi(F - F_v) + D_\eta(G - G_v) + D_\zeta(H - H_v) \tag{3.51}$$

where F, G, H, F_v, G_v, H_v are the inviscid and viscous flux's along the three coordinate directions. For a true time-accurate linearization of the complete flux, the flux Jacobian matrices contains contributions from both the inviscid and viscous fluxes. Formation of the inviscid and viscous Jacobians is computationally expensive. Therefore, a common approximation for Navier-Stokes simulations is to neglect the viscous contributions and form the Jacobian matrices from the inviscid flux components. For appropriate levels of dissipation and time steps, this approximation has a minimal impact on solution accuracy and convergence.

3.5.1 Commonly Used Implicit Schemes

Expansion of the left hand side of Equation (3.50) leads to a large banded block matrix system of equations. Solution of this system by direct inversion in three dimensions is impractical for real-world problems. Therefore, considerable research has been devoted to developing computationally efficient indirect solution procedures. These indirect methods fall into two classes: approximate factorization schemes based on the Douglas-Gunn ADI method [76] or the more recently developed relaxation schemes [77,78].

The most widely used ADI schemes were introduced by Briley and McDonald [79] and Beam and Warming [80] in the mid 1970's. Both schemes replace the unfactored implicit operator in Equation (3.50) with a product of three one dimensional operators defined along each coordinate direction. In the three-dimensional Beam-Warming scheme [81], Equation (3.50) is approximated by

$$\left(I+\frac{\alpha\Delta t}{V}D_\xi A^n\right)\left(I+\frac{\alpha\Delta t}{V}D_\eta B^n\right)\left(I+\frac{\alpha\Delta t}{V}D_\zeta C^n\right)\Delta Q^{n+1} = -\frac{\Delta t}{V}R(Q^n) \qquad (3.52)$$

For central differencing, each operator in Equation (3.52) is a block tridiagonal matrix which can be solved by variants of the Thomas algorithm for tridiagonal systems [51]. Although unconditionally stable in two-dimensions, the factorization given by Equation (3.52) can be shown to be unstable in three-dimensions without the inclusion of large amounts of implicit and explicit artificial dissipation. Implicit dissipation is added by including terms similar to Equation (3.12) in the implicit factors. The inclusion of fourth-order dissipation leads to block pentadiagonal factors. Although more efficient than direct simulation, the solution of the block matrix systems is still computationally expensive for three dimensional problems. Another problem with the three factor scheme is that an error term of order $(\Delta t)^3$ is generated by the factorization that can severely impact the convergence rate. Pulliam and Chaussee [66] have introduced a variation of the Beam-Warming algorithm that uses symmetric diagonalization of the Jacobian matrices to produce a stable scheme with scalar tridiagonal or pentadiagonal factors. This biggest drawback of this scheme is that it is at most first order accurate in time.

The large factorization error of the three-factor ADI scheme has lead to the development of two factor schemes which are of order $(\Delta t)^2$ and stable in three

dimensions. Two factor schemes in common use are the partially flux-split scheme of Steger et. al. [82], the hybrid scheme of Sankar [83,84,85], the Lower-Upper (LU) factorization scheme proposed by Jameson and Turkel [86] and Buning and Steger [87] and the Symmetric-Gauss-Seidel variant of the LU scheme (LU-SGS) introduced by Yoon and Jameson [88]. The flux-split scheme was applied by NASA researchers to a wide range of problems [89,90]. The hybrid scheme is effective for solving flows about wings and rotors [84]. The LU was not used extensively until the mid 1980's when Jameson and Yoon [91] applied the scheme to the solution of the Euler equations. Other researchers such as Whitfield [92], Buratynski and Caughey [93] and Yokota and Caughey [94] successfully extended the scheme to solve both the Euler and Navier-Stokes equations. Both the partially flux-split scheme and the hybrid scheme require block tridiagonal inversions in two coordinate directions. The LU scheme requires block diagonal inversions for each factor. The LU-SGS scheme can be constructed in forms that require either block or scalar diagonal inversions.

The scalar diagonal form of the LU-SGS scheme has become the most popular of the two factor schemes because of its low operation count and generally good convergence characteristics. The scheme is especially suited chemically reacting flows because it requires at most a block diagonal inversion of the Jacobian matrices. Therefore, its greatest use has been in the area of chemically reacting and hypersonic flows [95,96,97]. Yoon and Kwack [98] implemented a three-dimensional version of the scheme for solving the compressible Navier-Stokes equations and applied it to standard steady transonic wing cases. This code has been extended using multi-grid to accelerate the convergence to a steady state [99,100]. The scheme has been used successfully by Chen and McCroskey [70,101] and Srinivasan et. al.[102] for unsteady rotor analyses in both hovering and forward flight. Obayashi and Guraswamy [103] implemented a version of the scheme in the ENSAERO code for the solution of coupled aeroelastic problems. An

attractive feature of the scalar form of the LU-SGS scheme is that with careful coding it can achieve computational speeds that are competitive with explicit schemes. In addition, it is a simple algorithm to implement because it requires no special matrix solvers. These factors led to the selection of the scheme for this research.

3.5.2 The LU-SGS Scheme

The LU-SGS scheme has its roots in the LU scheme of Jameson and Turkel. This scheme can be written as

$$LU\Delta Q = -\frac{\Delta t}{V}R \tag{3.53a}$$

where the factors L and U are

$$L = I + \frac{\alpha\Delta t}{V}\left(\nabla_\xi \hat{A}^+ + \nabla_\eta \hat{B}^+ + \nabla_\zeta \hat{C}^+\right) \tag{3.53b}$$

$$U = I + \frac{\alpha\Delta t}{V}\left(\Delta_\xi \hat{A}^- + \Delta_\eta \hat{B}^- + \Delta_\zeta \hat{C}^-\right) \tag{3.53b}$$

and the operators, ∇_ξ, ∇_η, ∇_ζ, Δ_ξ, Δ_η, and Δ_ζ are standard backward and forward differences. The stability of the LU scheme is dependent on each factor being diagonally dominant. This is achieved by replacing the standard flux Jacobians with diagonally dominant forms whose eigenvalues are either all positive (A^+, B^+, and C^+) or all negative (A^-, B^-, and C^-). The LU-SGS scheme is obtained by recasting the factors as Symmetric Gauss Seidel relaxation sweeps [104]. After some algebra, Equations (3.53) become

$$LD^{-1}U = -\frac{\Delta t}{V}R \tag{3.54a}$$

where

$$L = I + \frac{\alpha\Delta t}{V}\left(\nabla_\xi \hat{A}^+ + \nabla_\eta \hat{B}^+ + \nabla_\zeta \hat{C}^+ - \hat{A}^+ - \hat{B}^+ - \hat{C}^+\right) \tag{3.54b}$$

$$D = I + \frac{\alpha\Delta t}{V}\left(\hat{A}^+ - \hat{A}^- + \hat{B}^+ - \hat{B}^- + \hat{C}^+ - \hat{C}^-\right) \tag{3.54c}$$

$$U = I + \frac{\alpha\Delta t}{V}\left(\Delta_\xi \hat{A}^- + \Delta_\eta \hat{B}^- + \Delta_\zeta \hat{C}^- + \hat{A}^- + \hat{B}^- + \hat{C}^-\right) \tag{3.54d}$$

Like the LU scheme, the Jacobian matrices must be diagonally dominant. Yoon and Jameson proposed approximate Jacobians that satisfy this condition. These Jacobians are constructed from the standard Jacobians of the inviscid flux vectors augmented by their corresponding spectral radii. The approximate Jacobians have the forms

$$\hat{A}^\pm = \frac{1}{2}(A \pm \rho(A)I) \tag{3.55a}$$

$$\hat{B}^\pm = \frac{1}{2}(B \pm \rho(B)I) \tag{3.55b}$$

$$\hat{C}^\pm = \frac{1}{2}(C \pm \rho(C)I) \tag{3.55c}$$

A general form of the matrices, A, B, and C is given in Appendix B. The spectral radii such as $\rho(A)$ are defined by

$$\rho(A) = \kappa \max[|\lambda(A)|] \tag{3.56}$$

where $\lambda(A)$ represent the eigenvalues of A and κ is a constant ≥ 1 that is used to control stability and convergence. With these definitions of the diagonally dominant Jacobians, the operator, D, in Equations (3.54) can be written for $\alpha = 1$ as

$$D = \left[1 + \frac{\Delta t}{V}(\rho(A) + \rho(B) + \rho(C))\right] I \tag{3.57}$$

The L and U operators can then be written after expansion of the differences at a cell center defined by the indices I, J, and K as

$$L = D_{I,J,K} - \frac{\Delta t}{V}\left(\hat{A}^{+}_{I-1,J,K} + B^{+}_{I,J-1,K} + C^{+}_{I,J,K-1}\right) \tag{3.58a}$$

$$U = D_{I,J,K} + \frac{\Delta t}{V}\left(\hat{A}^{-}_{I+1,J,K} + B^{-}_{I,J+1,K} + C^{-}_{I,J,K+1}\right) \tag{3.58b}$$

With these operators, the solution of Equation (3.54a) can be obtained by forward and backward relaxation sweeps that proceed from the lower left corner of a zone to the upper right corner and then back. The sweeps are performed in the following order:

Forward sweep:

$$D\Delta Q^{**}_{I,J,K} = \frac{\Delta t}{V}\left[-R + \left(\hat{A}^{+}_{I-1,J,K}\Delta Q^{**}_{I-1,J,K} + \hat{B}^{+}_{I,J-1,K}\Delta Q^{**}_{I,J-1,K} + \hat{C}^{+}_{I,J,K-1}\Delta Q^{**}_{I,J,K-1}\right)\right] \tag{3.59a}$$

Backward sweep:

$$D\Delta Q^{n+1}_{I,J,K} = D\Delta Q^{**}_{I,J,K} - \frac{\Delta t}{V}\left[\hat{A}^{-}_{I+1,J,K}\Delta Q^{n+1}_{I+1,J,K} + \hat{B}^{-}_{I,J+1,K}\Delta Q^{n+1}_{I,J+1,K} + \hat{C}^{-}_{I,J,K+1}\Delta Q^{n+1}_{I,J,K+1}\right] \quad (3.59b)$$

Update Q:

$$Q^{n+1}_{I,J,K} = Q^{n}_{I,J,K} + \Delta Q^{n+1}_{I,J,K} \quad (3.59c)$$

It is important to note that the operator, D, is a scalar diagonal matrix. Therefore, the intermediate and final corrections, ΔQ^{**} and ΔQ^{n+1}, are obtained by simple division. At the boundaries, all off-diagonal values of the intermediate and final corrections are set to zero.

For $\Delta t=\infty$, the LU-SGS scheme can be recast to resemble a quasi-Newton or Newton-like iteration scheme that does not require specification of an optimum CFL number or time step. After multiplying through by $V/\Delta t$ and setting $\Delta t=\infty$, the diagonal operator, D, becomes

$$D = \left[\rho(A) + \rho(B) + \rho(C)\right] I \quad (3.60)$$

and the factor $\Delta t/V$ disappears from the off-diagonal terms in the L and U operators. This scheme cannot achieve the quadratic convergence of a true Newton method due to the approximate Jacobians and the factorization error. However, linear convergence can be demonstrated.

Another advantage of the LU-SGS algorithm is that it is completely vectorizable when the forward and backward sweeps are performed in oblique planes defined by

I+J+K=constant such as the one shown in Figure 3.4. This can be achieved by either reordering the three-dimensional arrays into two dimensions defined by the serial number of the oblique plane and the addresses of the points on the plane or creating mapping arrays that allow the existing three-dimensional array locations to be indexed indirectly. The second approach was used in this research to avoid having to rewrite large sections of the baseline code. The three-dimensional arrays containing the solution, etc. can be taken into the solver routine as dummy arrays and redimensioned as a large one dimensional vector. The location of any point in this vector can be defined by $L = I + (J-1)*MI + (K-1)*MI*MJ$ where MI, MJ, and MK are the maximum dimensions of the corresponding three-dimensional array. The value of L at each point in the plane defined by I+J+K=constant is stored in an integer array along with the number of points in each plane, NP. These arrays provides the correct address for the required values on each oblique plane.

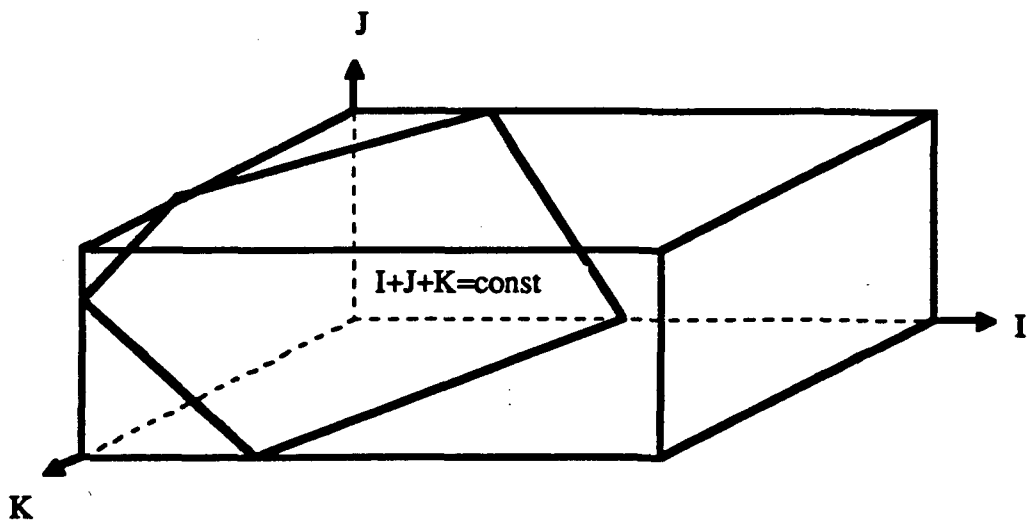


Figure 3.4 Oblique Sweep Planes For LU-SGS Scheme

The solution scheme defined by Equations (3.59) is first order-accurate in time. Higher order accuracy can be obtained by using Newton inner iterations [101,105] at each time step. This procedure also helps minimize the factorization error. Matsuno [106] has introduced the δ^k scheme which is a variant of the inner iteration scheme that can obtain k-th order accuracy in k sub-iterations. Lee and Kim [107] have used the δ^k approach with the LU-SGS scheme to calculate rotor flows in hover and forward flight.

3.6 Moving Grid Procedures

The application of the discretized forms of the governing equations to moving grid systems requires special treatments for calculating the flux, cell volume, and grid speeds. In particular, Thomas and Lombard [108] have demonstrated that an additional geometric constraint called the Geometric Conservation Law must be imposed for moving grid systems to ensure that conservation is maintained for uniform flows. For non-uniform flows, the cell volumes and grid speeds need to be computed in a consistent manner to prevent grid induced errors. The geometric constraint can be expressed for discretizations based on Equation (2.1) as

$$V(t_2) - V(t_1) = \int_{t_1}^{t_2} \oint_{S(t)} \vec{v}_b \bullet \vec{n} dS dt \tag{3.61}$$

Equation (3.61) represents the conservation of volume for a moving cell. Similarly, the integro-differential form of the law can be written as

$$\frac{\partial V}{\partial t} = \oint_{S(t)} \vec{v}_b \bullet \vec{n} dS \tag{3.62}$$

Discretization of Equations (3.61) and (3.62) must performed using the same procedure employed for the flow equations. Both forms of the Geometric Conservation Law require that the grid speeds, v_b, be defined before the change in volume can be computed. Obayashi [109] has presented a procedure for updating the cell volume without prior knowledge of the grid speeds. The right hand side of Equation (3.61) is discretized as

$$\int_{t_1}^{t_2} \oint_{S(t)} \vec{v}_b \bullet \vec{n} dS dt = \sum_{cell} \int_{t_1}^{t_2} \vec{v}_b \bullet \vec{n} S dt \tag{3.63}$$

where S is the area of each cell face. The volume, V_s, swept out by the area, S over the period t_2-t_1 is given by

$$V_s = \int_{t_1}^{t_2} \vec{v}_b \bullet \vec{n} S dt \tag{3.64}$$

When the position of the grid is known at times t_2 and t_1, Equations (3.5) and (3.6) can be used to compute V_s for each cell face. Equation (3.61) can then be computed as

$$V(t_2) - V(t_1) = \sum_{cell} V_s \tag{3.65}$$

The grid speeds normal to each cell face can be computed as

$$\vec{v}_b \bullet \vec{n} = \frac{V_s}{S\Delta t} \tag{3.66}$$

For moving grids, both the implicit and explicit formulations contain an extra term that is added to the residual. For an implicit discretizations with $Q^{n+1}=Q^n+\Delta Q$, $(QV)^{n+1}$ is replaced with $(Q^n+\Delta Q)V^{n+1}$ and

$$(QV)^{n+1} - (QV)^n = \Delta QV^{n+1} + Q^n\left(V^{n+1} - V^n\right) \tag{3.67}$$

The term , $Q^n(V^{n+1}-V^n)$, is added to the residual. This term vanishes for stationary grids since the volume does not change. The change in volume is computed using the Geometric Conservation Law. For many aeroelastic applications, the rate that the volume changes with time is small and the Geometric Conservation Law can be neglected without a severe impact on solution accuracy. Rapidly moving grid systems require the Geometric Conservation Law for accurate solutions.

CHAPTER IV

DISTRIBUTED COMPUTING PROCEDURES AND IMPLEMENTATION

This chapter describes the procedures used to convert the baseline Lockheed/AFOSR multi-zone flow solver into a parallel flow solver for distributed computing systems. The basic approach taken in developing the distributed flow solver was outlined in Chapter 1. The following sections describe the Parallel Virtual Machine (PVM) software interface used for interprocessor communications, the factors that effect communication performance on distributed systems, the communication strategies employed in the solver, load balancing procedures and details of the modifications made to the baseline TEAM code.

4.1 The PVM Message Passing Interface

The Parallel Virtual Machine (PVM) system was developed jointly by the Oak Ridge National Laboratory , the University of Tennessee, and Emory University to provide a standard message passing system for connecting homogenous and heterogeneous systems of UNIX workstations into a large distributed parallel computing system termed a "virtual machine". The virtual machine allows a network of individual workstations to emulate a large dedicated distributed parallel system such as the Intel Paragon. Unlike the Paragon which has specialized internal high speed communications subsystems for interprocessor

communications, PVM communications are usually performed using slower external network hardware systems such as Ethernet.

Since it's inception in 1989, PVM has become the "de-facto" standard for distributed computing on workstations. Recent versions of PVM were extended to support large multiprocessor systems such as the Intel Paragon and the Cray T3D systems. The popularity of PVM can be attributed to its simple but versatile software interface that supports both FORTRAN and C and its ability to run on a wide variety of hardware platforms. The major drawback of the system is relatively poor communications performance [110]. This has led to the development of an alternate interface, the Message Passing Interface (MPI), that is being adopted as an ANSI standard system [111]. However, MPI had not reached the same level of maturity as PVM when this research was initiated. Therefore, PVM was selected for this research.

The PVM system is composed of two components. The first component is a UNIX program called a daemon that runs on each processor in the virtual machine. The daemon functions as a server that handles process control, synchronization, fault detection and message routing between processors. On message passing systems, messages are sent as packets over interprocessor communication channels. The standard PVM system uses both the UDP and TCP/IP UNIX communications protocols [112] to route messages between tasks. The UDP protocol is used for communication between daemons on different machines. The TCP protocol is used for local communication between daemons and tasks on the same machine. Current versions of PVM also allow tasks to bypass the daemon and communicate directly with each other using the TCP protocol. The daemons provide buffers to hold incoming and outgoing messages and ensure that the messages are processed in the correct order. The daemons must be started manually from a list of available hosts supplied by the user. Individual tasks can spawn other tasks on processors belonging to the virtual machine.

The second component of the PVM system is a library of user callable routines in FORTRAN or C for message passing, spawning processes, sequencing tasks, and dynamically reconfiguring the virtual machine. These routines must be linked to the users application. The message passing routines form the core of the programming interface. They allow users to open message buffers that can be packed with data of varying data type and send the data to individual processors with unique message tags. The PVM software performs the data conversions required when binary data is passed between machines of different architectures. This simple but complete library allows users to develop parallel applications with minimal modifications to their existing code.

Three versions of the PVM system were used in this research. The first two version of PVM used were PVM 3.1.4 and 3.2.5 that were released in the public domain by Oak Ridge Laboratories. The final version of PVM used in this research was the IBM PVMe system [113] developed by IBM for their SP1 and SP2 distributed parallel systems. PVMe is based on version 3.2.6 of the public domain PVM. The major difference between the public domain version of PVM and PVMe lies in the communication protocol used for interprocessor communications. PVMe does not use TCP/IP for interprocessor communications. Communication is performed directly between all tasks using the Allnode or High Performance Switch (HPS) hardware subsystems on the SP1 and SP2 and the Communication Subsystem (CSS) communication software interface [113,114].

4.2 Communication Strategies

The distributed flow solver is implemented by using the domain decomposition inherent in the baseline multi-zone solver to map individual grids or sets of grids to separate

processors. The solution on each grid is performed concurrently. Therefore, the performance of the distributed solver is dependent on the efficiency of task sequencing and data communication procedures used in the solution process. The goal is to keep the ratio of communications time to solution time as low as possible. The communications performance is dependent on the speed of the communications channels connecting the processors, the number of messages that must be passed between processors, and the type of network topology used in the distributed system. Therefore, the types of communications strategies adopted for the distributed solver must take into account the network specific factors that effect communication performance.

4.2.1 Communications Performance Factors

The speed of the communications channels in a distributed system is driven by two factors, latency and bandwidth [113]. Latency is defined as the time interval between the instance at which the program institutes a call for a data transmission and the instance at which the actual transfer occurs. This time interval can be thought of as the time required to send a zero length message between nodes. Latency depends upon characteristics of the communications hardware and the layers of software involved in packing and transmitting data. Bandwidth is defined as the total available bit rate of a digital channel and represents the maximum speed of information transfer between nodes. System bandwidth is determined by the speed and type of the network or communications subsystem connecting processors and the overhead incurred by the control data added by the communication protocol during information transfer.

The type network connecting the nodes in a distributed system also plays a significant role in the performance of a distributed flow solver. There are two basic types of communications networks: circuit-switched and packet switched. Circuit switched networks create dedicated paths between nodes and prevent simultaneous traffic over the

paths at communication time. Packet-switching networks segments each message into packets with unique destination tags that are sent over the network. Ethernet is the most widely used example of a packet-switching system.

Networks are also characterized by their physical and electrical topologies [115]. These topologies define whether the network is a shared medium where only a single processor can send messages over the network at a time or a switched network where two or more nodes can communicate with each other simultaneously through a switched circuit. Networks such as Ethernet and FDDI are examples of shared medium networks. Dedicated communications subsystems such as the High Performance Switch on the IBM SP2 are usually switched networks.

The performance of shared medium networks such as Ethernet deteriorates quickly with increasing numbers of nodes and users. This is because Ethernet is a broadcast bus technology that communicates in half-duplex. This means that the end nodes on an Ethernet system cannot simultaneously send and receive messages. All nodes on the network share a single communications channel or bus. Messages are broadcast to each node which is responsible for determining which messages it should receive and which messages to ignore. A message collision can occur when two nodes try to transmit data simultaneously. When a collision occurs the nodes must wait a random amount of time before retransmitting the messages. Therefore, placing several messages on the bus at one time increases the chances of message collision and reduces the system throughput. These factors limit the standard Ethernet to a throughput of 10 megabits per second. Recently, switching technology has been introduced that can raise the performance of Ethernet. Switched systems offer a much higher transfer rate than standard Ethernet. For example, the High Performance Switch system on the IBM SP2 is a bi-directional system with a peak rate of 320 megabits per second in each direction [114].

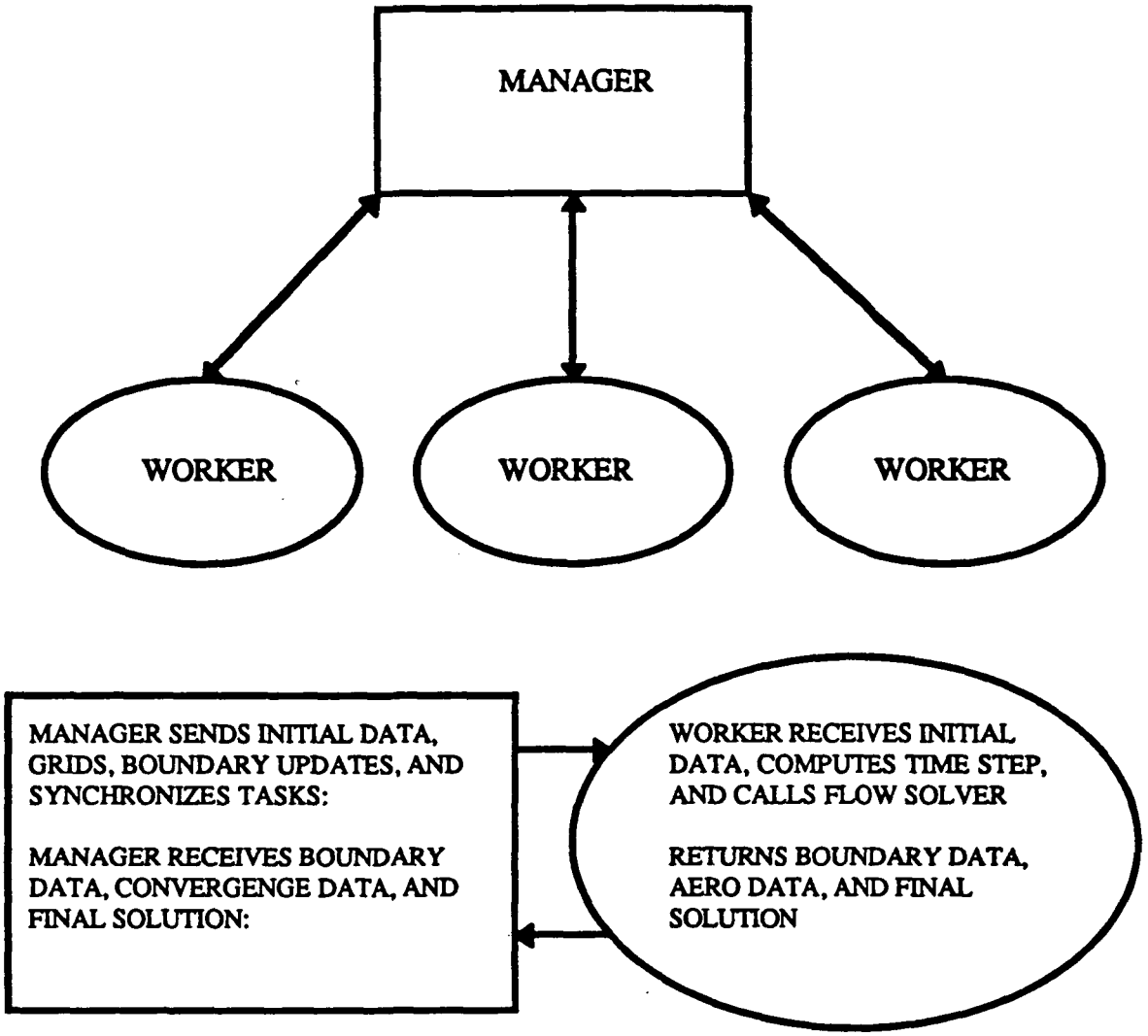


Figure 4.1 The Manager-Worker Strategy

solution steps. The manager is responsible for reading in the solver initialization and control parameters, the computational grids, and the solution arrays required to restart the solution from a previous run. The workers contain the actual Euler and Navier-Stokes solution routines. Prior to the initial solution step, the manager assigns grids to the individual processors and sends the appropriate initialization data and grids to each processor using the PVM interface. The workers compute the initial time steps and initialize the ghost boundary arrays used in the baseline code. At the start of each step,

boundary array data for each processor is sent to the manager that stores the data in a global buffer and then sends each processor the boundary data it requires from grids on other processors. The workers wait until they receive the updated boundary data and then perform a solution step and update their local boundary arrays. Each worker then returns convergence information to the manager, updates the time step, sends the appropriate local boundary arrays back to the manager, and waits to receive the updated boundary data from the manager before continuing with the next solution step. This process is repeated until a maximum number of steps are performed or a specified convergence criterion is reached.

The biggest advantage of the Manager-Worker strategy is the reduced complexity of the control logic required to synchronize the required data exchange between processors. By routing the data exchange through the manager, the number of possible data paths is reduced to the number of worker processes. If the workers are allowed to communicate directly with each other, a maximum of N(N-1)/2 two way paths would be required for N workers if each worker was required to communicate with every other worker in the system. A second advantage of the Manager-Worker approach is that the workers do not have to be the same program as required by the Single Program Multiple Data (SPMD) [9] programming model. This simplifies the addition of other types of modules such as structural analysis routines or modules for design optimization into the solution process. One disadvantage of the Manager-Worker strategy is that a processor must be dedicated to the manager which can be a problem in systems with a small number of available processors. The biggest disadvantage of the Manager-Worker strategy is that routing data back and forth through the manager creates a communications bottleneck with increasing numbers of processors. This bottleneck can lead to excessive amounts of worker idle time as they are forced to wait while the manager accumulates boundary data and then sends it to the appropriate processors. This problem can be acute on dedicated distributed systems

with large numbers of available nodes. Therefore, an optional strategy called Worker-Worker that allows workers to communicate directly with each other was implemented.

4.2.3 The Worker-Worker Strategy

The Worker-Worker strategy was implemented as a modification to the Manager-Worker approach. The manager was maintained to provide the same initialization , I/O, and synchronization functions as the Manager-Worker code. However, the appropriate zonal boundary data required by a worker at the start of each step is obtained directly from the other workers. The manager signals the start of each new step and the workers first send out their updated solution data to appropriate processors and then wait to receive the boundary data they require from the other workers. The workers send convergence data back to the manager at the end of each step. The connections between processors are determined at the start of the solution in the manager from the baseline code's boundary condition data set and predefined grid-to-processor maps.

4.3 Load Balancing

Load balancing is the process of dividing the computational work among individual processors in manner that keeps all processors busy and reduces the idle time spent waiting for data. This idle time is imposed on the processors with the lightest loads by the synchronization required by the zonal boundary update process. The lightest loaded processors might have to wait until the heaviest loaded processor finishes before it gets the boundary data it needs to continue the solution. Therefore, proper load balancing is crucial if the expected speedup of the distributed parallel version of the TEAM code over the serial version is to be obtained. An efficient load balancing procedure must account for both the

total number of grid points mapped to each processor and the computational speed of each processor. These factors are known quantities that remain static during the course of a solution. Additional factors such as system load and communications overhead can also degrade performance and should be included in the load balancing process to obtain the optimum performance. However, these factors cannot be fixed prior to the start of a solution and must be accounted for dynamically. These factors lead to two general types of load balancing: static and dynamic.

Static balancing procedures are effective on dedicated or lightly loaded distributed systems and are the easiest to implement. The load balance is computed at the start of the solution and held fixed to its initial distribution. In dynamic load balancing, the performance of the system is monitored during the course of the solution and the load on each processor is modified at intervals to maintain the optimum performance. Dynamic balancing can be effective on heavily loaded network based systems [37]. However, the solution overhead required for dynamic balancing can be several time that of static balancing. This is particularly true for a structured multi-zone flow solver such as the baseline code which requires an exact definition of the zonal boundaries to form the buffer arrays that hold the required ghost boundary data. In dynamic load balancing system, the dimensions of the buffer arrays would have to be adjusted dynamically or fixed to an inappropriately large number. In addition, the fixed nature of most structured multi-zone grid systems make dynamic load balancing impractical for use with an existing flow solver. The existing zonal interfaces have to be maintained in addition to any new interfaces introduced by the balancing procedure. Unstructured grids are more amenable to dynamic balancing because they treat the grid as a cloud of points surrounding the body surfaces with no zonal boundaries. Therefore, clusters of points can be assigned to processors without the need to maintain an existing zonal interface. An effective dynamic load balancing procedure for unstructured grids has been given by Vidwans et. al. [118] . A

run-time load balancing procedure for structured grids has been given by De Keyser et. al. [119]. This procedure uses a specialized communications and control software system to perform the required load balancing.

The problems inherent in implementing a dynamic load balancing procedure in the distributed flow solver led to the adoption of three different static procedures. The first procedure used was a purely ad hoc approach in which grids were assigned to processors manually. The approach taken was to assign the largest grids to the fastest processors with the most memory. Multiple grids were allowed to reside on a single processor to accommodate the situation where there are more grids than processors. Therefore, two smaller grids could be assigned to a processor to give approximately the same load as a single larger grid on a separate processor.

In addition to the ad hoc approach, two automated static balancing procedures were implemented and evaluated. The first automated procedure used is a version of the Task Queue approach introduced by Johnson [120]. The second automated procedure used in this research is the modified form of the heuristic Crutchfield [121] algorithm introduced by Smith and Palas [13]. The details of the two automated procedures are given in the following sections.

4.3.1 Task Queue Load Balancing Procedure

The Task Queue load balancing scheme is a modified form of the dynamic Pool of Task approach [9] in which a master program creates a pool of tasks that are sent to worker programs as they fall idle. The Task Queue procedure starts by forming a queue of available worker processors sorted with the fastest processors at the top of the queue. A second queue is formed from the available grids with the largest grids assigned in descending order to the fastest processors. When all the processors are assigned a grid, a solution step is performed on each of the first N grids where N is the number of processors

and the order in which the processors finish the first set of grids is stored. The next set grids in the grid queue are assigned to the processors in the order in which the first set of grids finished and a solution step is taken. This procedure is repeated until all the grids in the grid queue have been assigned to assigned processor. The final distribution of grids on processors is held fixed for the remainder of the solution. With the Task Queue approach, the processors that finish first for each set of grids will continue to receive more grids until all the grids are assigned. The advantage of the Task Queue procedure is that the effect of communications overhead and total system load is introduced implicitly in the final load balance. This makes the procedure effective for heavily loaded systems where processors are shared with other users.

4.3.2 The Modified Crutchfield Algorithm

Following Smith et. al. [13,122], the modified Crutchfield algorithm consists of three stages. The first stage is an initialization stage where an initial distribution of grids on processors is defined. The second stage is an optimization stage that minimizes a quantity called the "excess capacity" of each processor. Excess capacity is a measure of the idle time of each processor for a work load defined by the total number of grid points, N(J), on processor J. The excess capacity on each processor is defined for processor speeds, S(J), given as points per second and the maximum of the time required by each processor to compute a given number of points, T_{MAX}, as:

$$C(J)_{EX} = T_{MAX} * S(J) - N(J) \quad : \; J = 1, \text{ No. of Processors} \tag{4.1}$$

where T_{MAX}=MAX[N(J)/S(J)]. Therefore, the excess capacity represents the difference between the total number of grid points on a node that can be processed during the period of time, T_{MAX}, and the actual number of points assigned to the processor. The total excess

capacity is the sum of the excess capacity of each processor. The final stage is a processor elimination step that removes the slowest processor if the elimination results in a faster turnaround. The elimination step is required to determine if an extremely slow processor will produce idle time on the other processors. The steps in each stage are given in References [13] and [122] as:

Initialization:

1. Sort grids by size.
2. Define an average target time:

$$T_{AVG} = \sum_{K=1}^{NGRIDS} NP(K) \Big/ \sum_{J=1}^{NPROC} S(J) \qquad (4.2)$$

where NP(K) is the total number of points in grid K.

3. Define initial excess capacity on each processor as $C_{EX}=T_{AVG}*S(J)$.
4. For each grid, starting with the largest:

   Assign the largest unassigned grid to the processor with the most excess capacity.

   End For

Optimization:

5. Find the processor with the least excess capacity.
6. For each grid K on this processor

For each grid L NOT on this processor

If interchanging K and L decreases the total excess capacity then:

Switch K and L

Go back to Step 5

End If

End For

End For

Elimination:

7. Find the processor with the least excess capacity after Steps 5 and 6
8. If this processor has just one grid then

If eliminating the slowest processor yields the same or improved execution time then:

Eliminate the processor and rerun Steps 5 and 6 if necessary to determine new load balance

Go to Step 7

End If

End If

The Crutchfield algorithm was found to produce an effective load balance when given a reasonable variety of grid sizes and processor speeds. It is particularly suited for heterogeneous distributed systems where a wide variation in processor speeds is possible. In the preceding algorithm, the overhead of communications and total system load are neglected.

4.4 Implementation of Distributed Computing Modifications

Two versions of the Lockheed/AFOSR TEAM code were obtained from the United States Air Force and Lockheed Aeronautical Systems Company. The first version obtained was Version 611. This version was used for the first phase of this research. An improved version of TEAM, Version 713 [123], was used in the final phase of this research. Both versions of the baseline code use the same basic solution algorithm. Version 713 has modifications to improve the reliablity, speed, and accuracy of the code. The following discussion describes the modifications made to Version 713 to produce the present distributed flow solver (PVMTEAM).

4.4.1 Synopsis of Modifications to the Baseline Code

A review of the code structure led to the following decomposition of the baseline code into the Manager and Worker codes. The first three routines in the baseline code (SHELL, TEAM, and SOLVER) and required support routines were copied from the baseline code to form the Manager code. The SHELL routine is the main routine that initializes the size of the main solution arrays using the POINTER/MALLOC procedure for dynamic memory allocation available in the FORTRAN compilers on the majority of computers the baseline code can run on. The ability to do dynamic memory allocation is one of the features of baseline code that made it ideal for converting into a distributed flow solver. The Manager and Worker routines do not have to be recompiled to accommodate changing grid sizes. The subroutine, TEAM, is responsible for reading in the solution initialization and control data, the boundary condition data, the initial grid, and the solution from previous runs for restart cases. Subroutine SOLVER controls the sequencing of the actual solver routine (MENSA) and the calculation and printing of convergence and aerodynamic loads data. For the Manager code, the modifications made in SHELL, TEAM, and SOLVER consisted of

reducing the memory requirements of the Manager, implementing the load balancing algorithms, and inserting the appropriate PVM calls required to send and receive data to and from the Workers. The main solver routines are not called in the Manager. Other baseline code options such as the ability to use a disk based out of core solution technique to reduce the required program size was maintained to enable the use of grid systems that are to large to be help in contiguous memory were maintained.

The Worker code is essentially the same as the baseline code with the exception that no I/O is performed and boundary data required for grids on other processors must be obtained using PVM. The SHELL routine in the Workers sizes all the required solution arrays except the ghost boundary arrays to accommodate only the grids assigned to the processor. The ghost boundary arrays are kept the same dimensions in both the Manager and Worker so that the indexing procedure used for accessing data in the arrays from the baseline code could be maintained. This eliminated an extensive recoding of the boundary update procedure used in the baseline code. In the Manager code, the ghost boundary array acts as a buffer for holding the updated boundary data returned by the Workers. A pseudo code description of both the Manager and Worker codes is given in Appendix C.

Two coding strategies were adopted to simplify program updates and to separate the PVM library calls from the baseline code. One code base is used for both the Manager and Worker. The C preprocessor and UNIX make processor are used to build different version of the code using predefined C preprocessor flags to activate and deactivate code segments during the make process. All calls to the PVM library routines were embedded in "wrapper" routines that act as an interface for passing the data to be communicated between the baseline solver routines and the PVM routines. This will enable the use of a different message passing library such as MPI without significant modifications to the other routines in the Manager and Worker programs.

4.4.2 Boundary Update Procedure

Implementation of the Manger/Worker and Worker/Worker communication strategies required a modification of the zonal boundary update procedure used in baseline code. In the baseline code, zonal boundary arrays are updated with the most recently available data from each zone as the algorithm cycles through the grids. This procedure was modified in the distributed flow solver to perform local updates of the boundary data for only those grids assigned to individual processors. Zones requiring boundary data from grids on other processors must wait until the start of the next complete iteration before receiving updated boundary data. This introduces a lag in the zonal boundary data available to interfacing grids on different processors.

In both the Manager/Worker and Worker/Worker strategies, the exchange of zonal boundary data is performed at the start of each time step. The zonal boundary data from the just completed step is first sent to either the Manager or the appropriate Worker processes. The Worker processes then wait to receive the appropriate updated boundary data from either the Manager or other Workers. Therefore, no processor proceeds with a step until it has received the required boundary data.

CHAPTER V

STEADY FLOW SIMULATIONS ON NETWORK BASED SYSTEMS

This chapter describes the validation and performance testing of the initial version of the distributed flow solver based on Version 611 of the baseline code and a second version of the distributed solver based Version 713 of the baseline code. The initial version was run on two homogeneous distributed systems composed of networks of Hewlett-Packard Co. (HP) PA-RISC and Digital Equipment Company (DEC) ALPHA workstations located at the Georgia Institute of Technology. The explicit Runge-Kutta solution algorithm in the baseline code was used in the initial solver. The LU-SGS implicit algorithm was implemented in the second version of the code along with the original explicit scheme. The second version of the solver was run a cluster of Silicon Graphics (SGI) workstations at the NASA Ames Research Center. The tests with the initial version of the code validated the implementation of the Manager/Worker communications strategy and the Task Queue load balancing scheme. The initial implementation of the solver served as a pathfinder code to determine the level of effort required to convert the baseline code into a distributed flow solver. The knowledge gained in this phase of the research was applied in the development of the second version of the distributed solver. The second version of the solver was used to compare the Task Queue and Crutchfield load balancing schemes and served as the basis for the unsteady flow solver described in Chapter 6.

5.1 Validation and Performance of the Initial Distributed Solver

A series of tests were conducted to validate the modifications made to TEAM to implement a distributed parallel flow solver. Solutions were generated using standard CFD test cases for a body of revolution and two wing alone geometries with the explicit Runge-Kutta solution scheme. Initial tests were performed to validate code modifications and identify logic errors. Additional tests were performed to study performance related issues such as the effect on convergence of lagging the update of boundary data for grids on different processors and the effect of load imbalance on total turnaround time. The effectiveness of the Task Queue load balancing scheme was evaluated by comparison with the ad hoc approach of assigning grids to processors manually.

5.1.1 The MBB Body of Revolution No. 3

The initial code validation test consisted of an Euler solution about the MBB body of revolution [124] shown in Figure 5.1 for a freestream Mach number of 0.8 and an angle of attack of zero degrees. The grid system consisted of two blocks containing 55x21x40 (46200) points and 56x21x40 (47040) points. This system was chosen to provide an approximately equal load balance. The computational grid was obtained by rotating a two-dimensional grid generated in the symmetry plane of the body about the longitudinal axis of the body. The two dimensional grid was generated using the grid generation scheme described in Reference [125]. Both the baseline code and the initial distributed code were run with the same grid system for 500 time steps using spatially varying time stepping and a CFL number of 1. The standard dissipation model (SAD) was used with the input values of the second and fourth order dissipation coefficients in Equation (3.13) set to standard default values of 0.5 for the second order dissipation coefficient and 2.0 for the fourth order dissipation coefficient . These values are divided by 64 inside the code to give the final values used in the dissipative flux calculations.

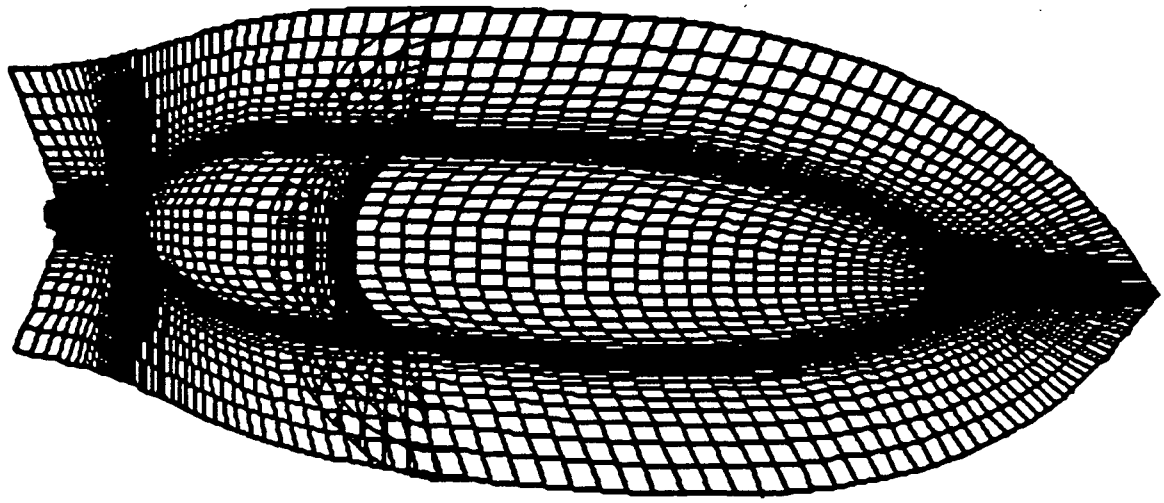


Figure 5.1: MBB Body and Symmetry Plane Grids

The distributed solver was run on a virtual machine consisting of three Hewlett-Packard workstations, a model 730 with 48 megabytes of memory and two model 720 workstations with 36 and 18 megabytes of memory. The baseline code was run on the model 730. For these tests, the Manager and Worker processes ran on different machines with the Manager running on the model 730 machine and the Worker processes running on the two model 720 systems. Each grid block was assigned to separate Worker processes. The effect of the distributed solver modifications on the convergence rate was obtained by comparing the average over the grid of the magnitude of the change in density with time for the two codes. This parameter represents the residual of the continuity equation and is a standard measure of convergence in the baseline code. In addition, the affect on solution accuracy was evaluated by comparing the surface pressure distributions in the boundary cells adjacent to

the plane of symmetry computed by the two codes. The convergence rates for the two codes are compared in Figure 5.2. For the default values of dissipation, distributed and baseline solvers have virtually identical convergence histories.

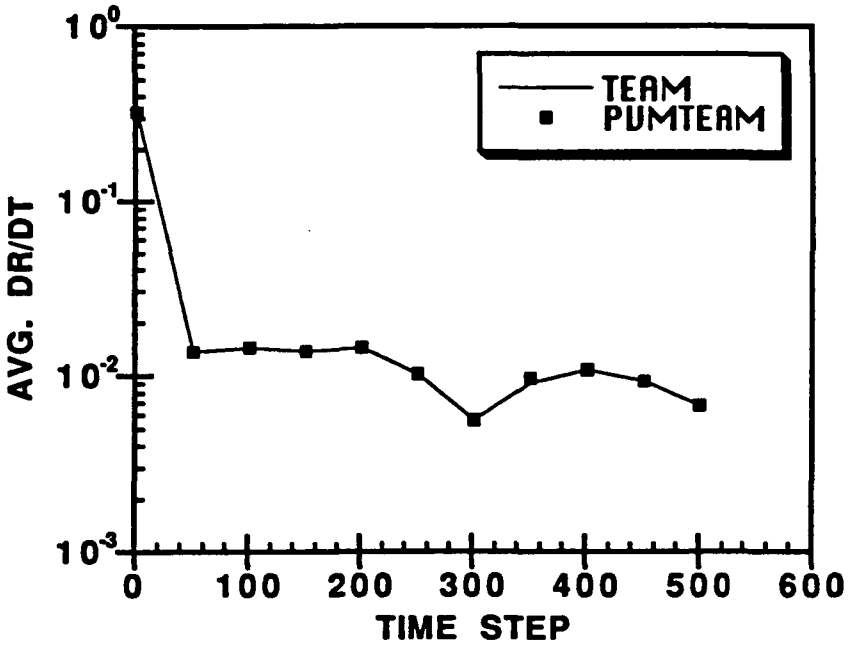


Figure 5.2: Baseline and Distributed Solver Convergence For the MBB Body

The leeward pressure distributions computed by distributed and baseline solvers are compared in Figure 5.3 with experimental data. Both codes produced identical pressure distributions that are in close agreement with the experimental data. These tests validated that the PVM modifications did not effect the accuracy or convergence rate of the code for this case.

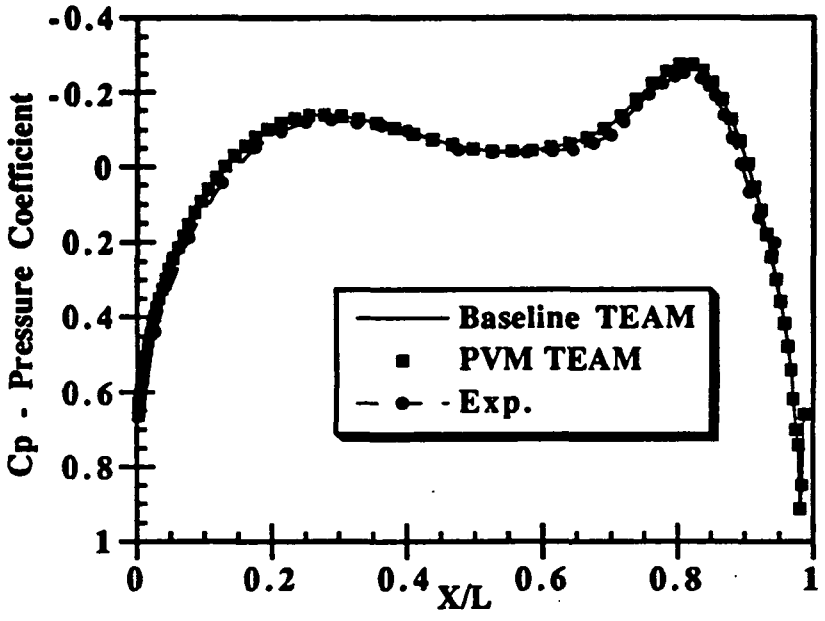


Figure 5.3: Correlation of Computed MBB Body Pressure Distributions With Experiment

The performance of the distributed code on the HP systems was difficult to gauge because of the heavy utilization of the system by other users. Turnaround times for distributed code varied from three to eight hours. A typical turnaround time for the baseline code running on a single machine, the model 730, was three to four hours. This wide range in turnaround performance illustrates one of the problems encountered when a distributed solver is run on machines during periods of heavy usage by other processes.

The MBB case was rerun on a system of identical 133 megahertz DEC ALPHA workstations (model 3000). Solutions were again run for 500 steps. Tests were performed using the two zone grid system used in the validation case with two worker processes and a

three zone system consisting of two blocks of 31,080 points and a third block of 31,920 points assigned to three worker processes. The turnaround performance measured in elapsed wall clock time for the baseline code and the distributed code using the ad hoc load balancing is shown in Figure 5.4.

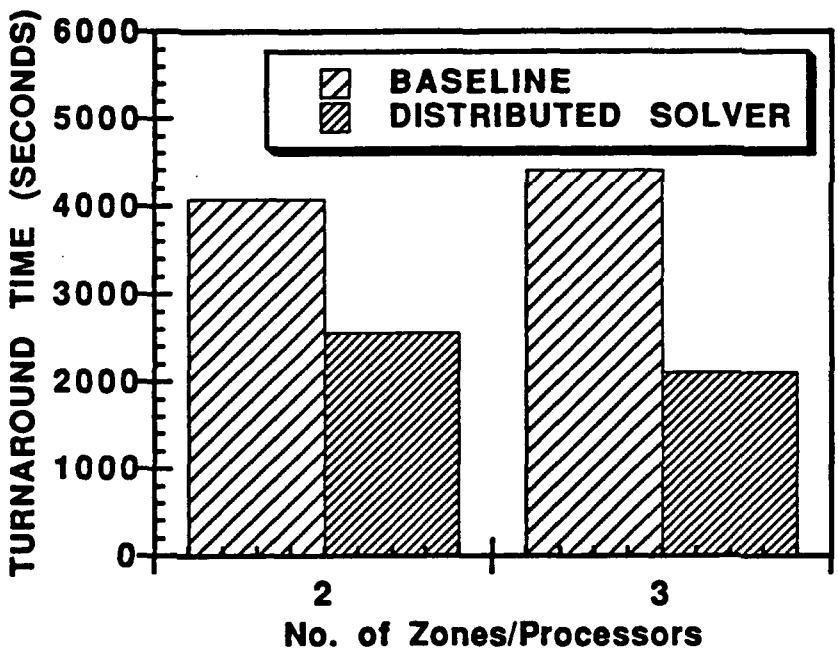


Figure 5.4: MBB Body Turnaround Performance For the Baseline and Distributed Solver

For the two zone grid the minimum execution time was 2560 seconds compared with a baseline solution time of 4080 seconds. This yields a speedup factor of 1.59. The speedup factor is given by

$$S = \frac{T_{SERIAL}}{T_{PARALLEL}} \tag{5.1}$$

where T_{SERIAL} and $T_{PARALLEL}$ are the turnaround times for the baseline and parallel codes. Assuming instantaneous communications, the ideal speedup for the two zone case would be 1.98. The ideal speedup is computed as the ratio of the time required by the baseline code to perform a time step and the maximum of the time required by each of the Workers to perform a time step. For processors of equal speed, the ideal speedup becomes the ratio of the total number of points in the grid system to the number of points on the Worker processor with the largest time per step if the overhead of communications and system load is neglected. For the three zone case, the minimum time was 2112 seconds compared to a baseline time of 4400 seconds. This represents a speedup of 2.08. The ideal speedup for the three zone case would be 2.94. A larger speed up was anticipated for the three block case. The reduced performance for this case was due to a heavy system load on one of the workers and the increased communications cost of the three zone grid. However, these results indicate that with an even load balance and a lightly loaded system effective speedup of code performance can be obtained.

5.1.2 The ONERA M6 Wing

The next validation cases run were for the ONERA M6 wing geometry [126] shown in Figure 5.5. The ONERA M6 wing is has an aspect ratio of 3.8, a taper ratio of 0.562, and a leading edge sweep of 30 degrees. A standard computational grid consisting of five relatively small grid blocks containing 15028, 3680,7820, 1,792, and 1216 points was provided by the Lockheed Aeronautical Systems Company for these tests. This grid system is a small multi-zone system used by Lockheed for validating modifications to the baseline code. The virtual parallel system of Hewlett-Packard workstations used for the body tests

was used for the ONERA M6 tests. Euler solutions were generated for a Mach number of 0.84 and an angle of attack of 3.06 degrees.

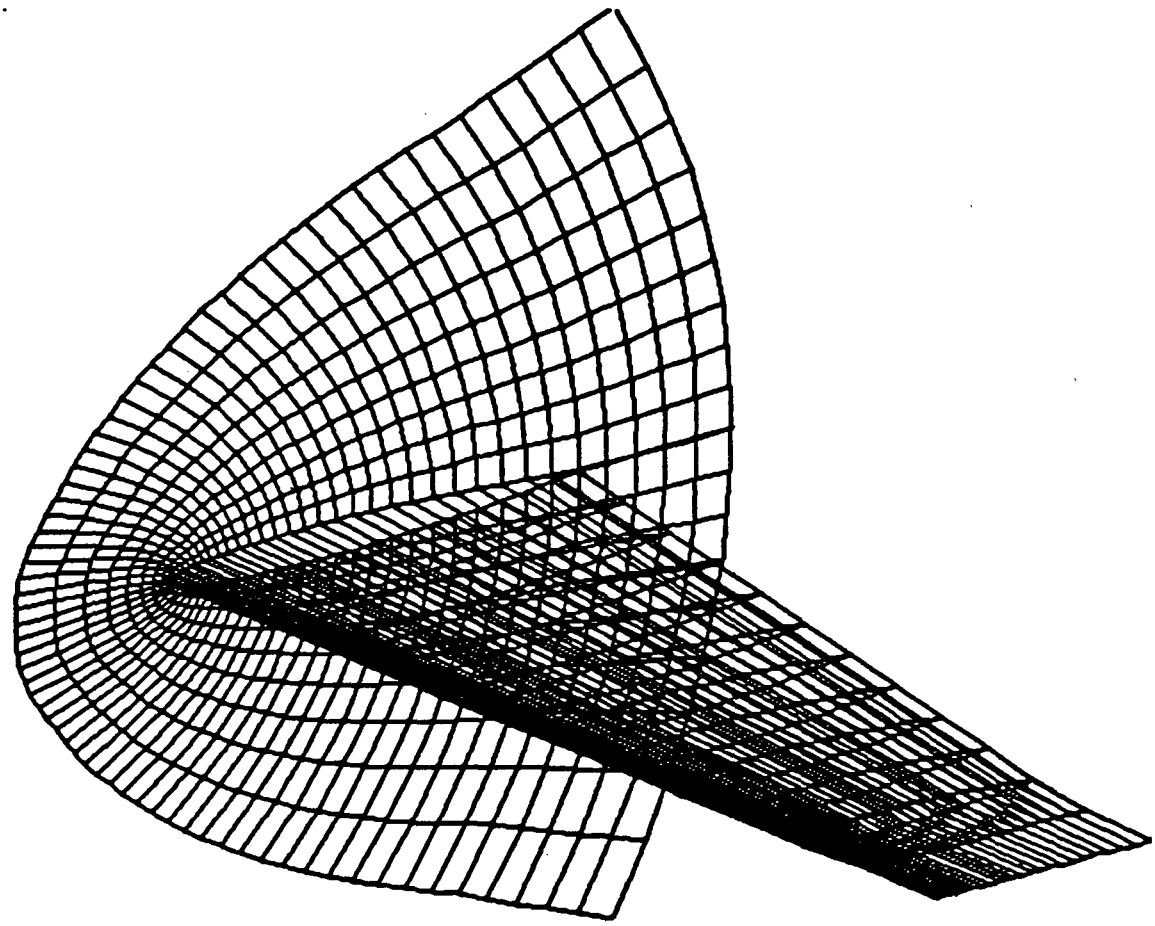


Figure 5.5: ONERA M6 Wing Planform and Symmetry Plane Grids

To maintain a roughly even load balance, the largest grid block was assigned to one Worker process and the remaining four grids were assigned to a second Worker process. This produced a load balance of 15028 points on the first processor and 14508 points on the second processor. Initial solutions were run for 1000 time steps at a CFL number of 6. The initial tests were performed using the modified adaptive dissipation model (MAD) with

the second order parameter dissipation parameter set to 0.1 and the fourth order parameter set to 1.0. These values were chosen to determine the effect of the dissipation model on the convergence rate of the distributed solver. As shown in Figure 5.6, the distributed and baseline solvers yielded drastically different convergence rates with the distributed solver diverging after 400 steps.

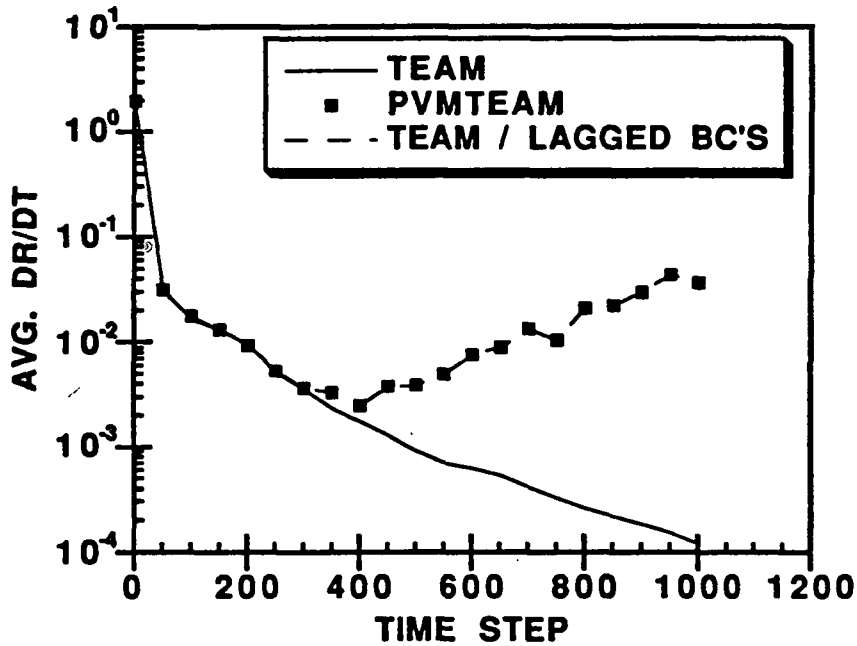


Figure 5.6: Convergence Rates For ONERA M6 Wing With MAD Dissipation

Initially, this divergence was felt to be due to the lag in updating the ghost boundary data shared by grids on the different processors. In the baseline code, zonal boundary arrays are updated with the most recent data before the next grid block is processed. In the distributed solver, only the boundary data for the grids on a single processor are updated with the most recently available data. Grids on other processors must wait for the data to be available from the Manager before the solution can proceed. This results in a lag in some of

the boundary data as grid blocks are processed in parallel. To determine the effect of lagging the boundary data, the baseline code was modified to use previous time step data for the ghost boundary conditions instead of the most recently available data. The convergence rate of the modified baseline code is compared with the rate for the distributed code in Figure 5.6. The convergence rate obtained for the baseline code virtually identical to the convergence rate of the distributed code. Therefore, the levels of dissipation introduced by the MAD formulation for the input second and fourth order parameters was not sufficient to maintain stability when the boundary data is lagged.

The overall slow convergence rates of both the distributed and baseline solvers indicated that the magnitudes of the dissipation coefficients used in the solution were to low. The ONERA M6 runs were repeated using the Modified Adaptive Dissipation model

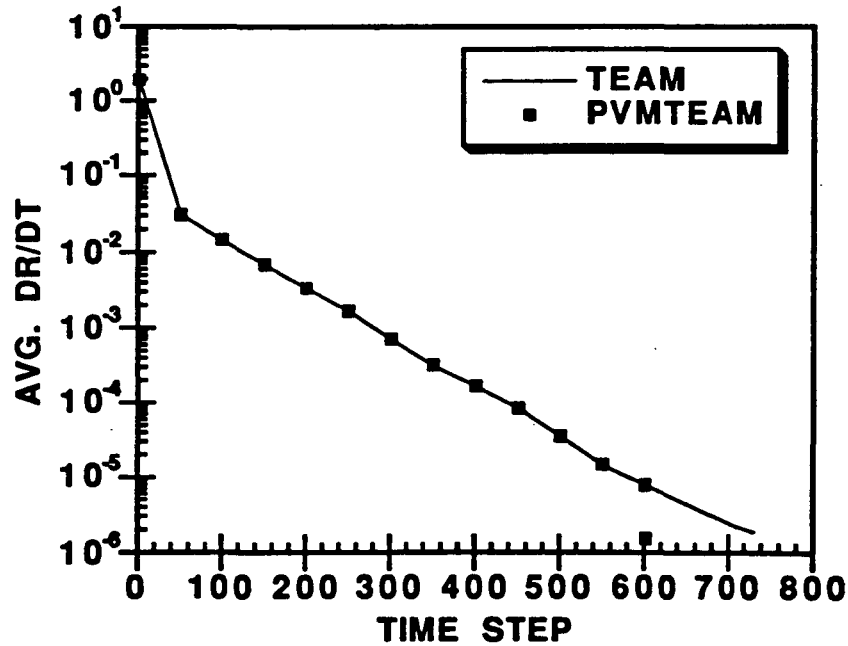


Figure 5.7: ONERA M6 Convergence Rates With Increased MAD Dissipation Coefficients

with the second and fourth order coefficients set to their default values of 0.5 and 2.0. The convergence rates for baseline code and the distributed code with the increased dissipation are shown in Figure 5.7. Both codes have virtually identical convergence rates up until 600 time steps. At that point, the distributed solver reached the automatic cutoff condition of a six order of magnitude drop in the residual. The baseline code continues for another 150 steps.

To examine the effect of changing the dissipation model, another set of tests were made using the more dissipative Standard Adaptive Dissipation Model (SAD).For these cases, tests were first made with the normal boundary update procedures used in the baseline code and distributed codes. The baseline code always uses the most recent boundary data.

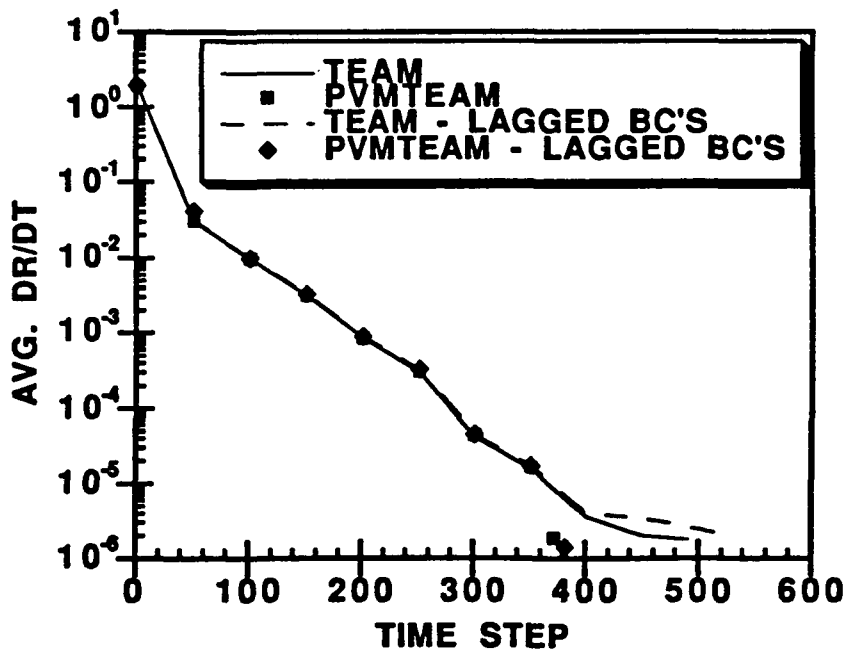


Figure 5.8: The Effect of Lagging Zonal Boundary Updates With SAD Dissipation

The distributed code only uses the most recent data for abutting grids assigned to the same processor. Zonal boundary data for grids on other processors is lagged. A second set of runs were made with the zonal boundary updates in both the baseline and distributed codes lagged to the previous time step. As shown in Figure 5.8, lagging the updates of the boundary data had only a slight effect on the convergence rates with the more dissipative SAD model. It can be concluded from these results that increasing the level of artificial dissipation appears to alleviate the problems associated with lagging boundary data for steady state solutions.

The effect of the distributed solver modifications on solution accuracy were obtained by comparing the computed total force and moment coefficients obtained by the baseline and distributed solvers. In addition, the spanwise pressure distributions at different spanwise stations were compared. The force and moment coefficients computed by the baseline and distributed solvers are compared in Table 5.1. along with the average values of the residual (DR/DT) at convergence and the number of supersonic points (NSUP) for the Modified adaptive dissipation model with the default values of dissipation coefficients. CL is the total wing lift coefficient, CM is the total wing drag coefficient, and CM is the wing pitching moment. The codes are seen to produce virtually identical results at the same levels of convergence when appropriate levels of dissipation are used.

Table 5.1: Comparison of Computed ONERA M6 Loads and Convergence Parameters

VALUE	TEAM	PVMTEAM
DR/DT	1.88E-6	1.548E-6
NSUP	798	798
CL	0.294260	0.294262
CD	0.012855	0.012856
CM	-0.228115	-.228118

The chordwise pressure distributions at two span stations computed by the distributed and baseline solvers are compared with experimental data in Figures 5.9 and 5.10. The poor correlation with experimental data in the shock region is a result of the coarse grid used in these tests. However, the pressure distributions generated by the two flow solver are virtually identical.

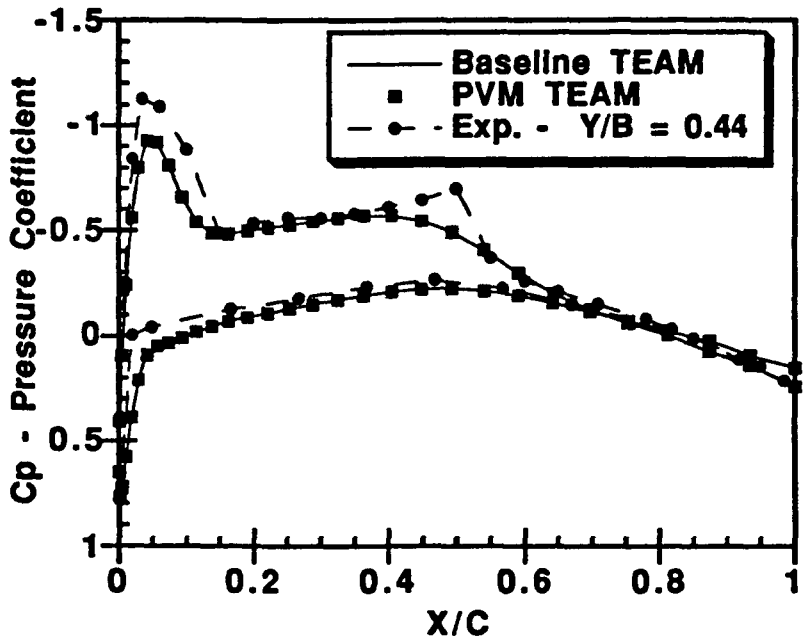


Figure 5.9: ONERA M6 Surface Pressure Distributions at 50% Span

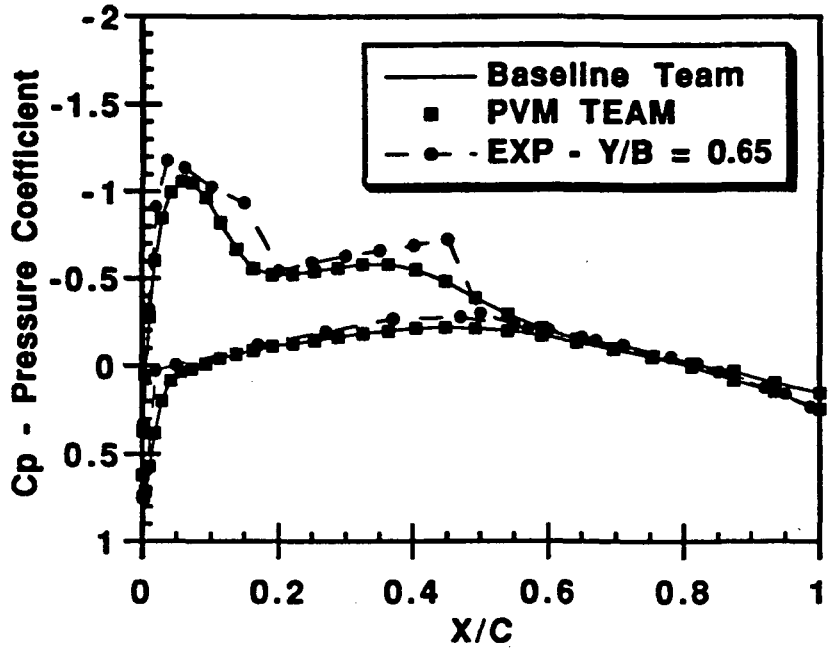


Figure 5.10: ONERA M6 Surface Pressure Distributions at 70% Span

5.1.3 Lockheed/AFOSR Wing C

A third set of Euler analyses were conducted using a seven block grid system for the Lockheed/AFOSR Wing C geometry [127] supplied by the Lockheed Aeronautical Systems Company. Wing C has an aspect ratio of 2.6, a taper ratio of 0.3, and a leading edge sweep angle of 45 degrees. The Wing C geometry is a standard test case used in CFD validation studies [3]. The Lockheed Wing C Euler grid system contains a total of 179309 nodes and 156672 cells. The wing planform and symmetry plane grids are shown in Figures 5.11 and 5.12.

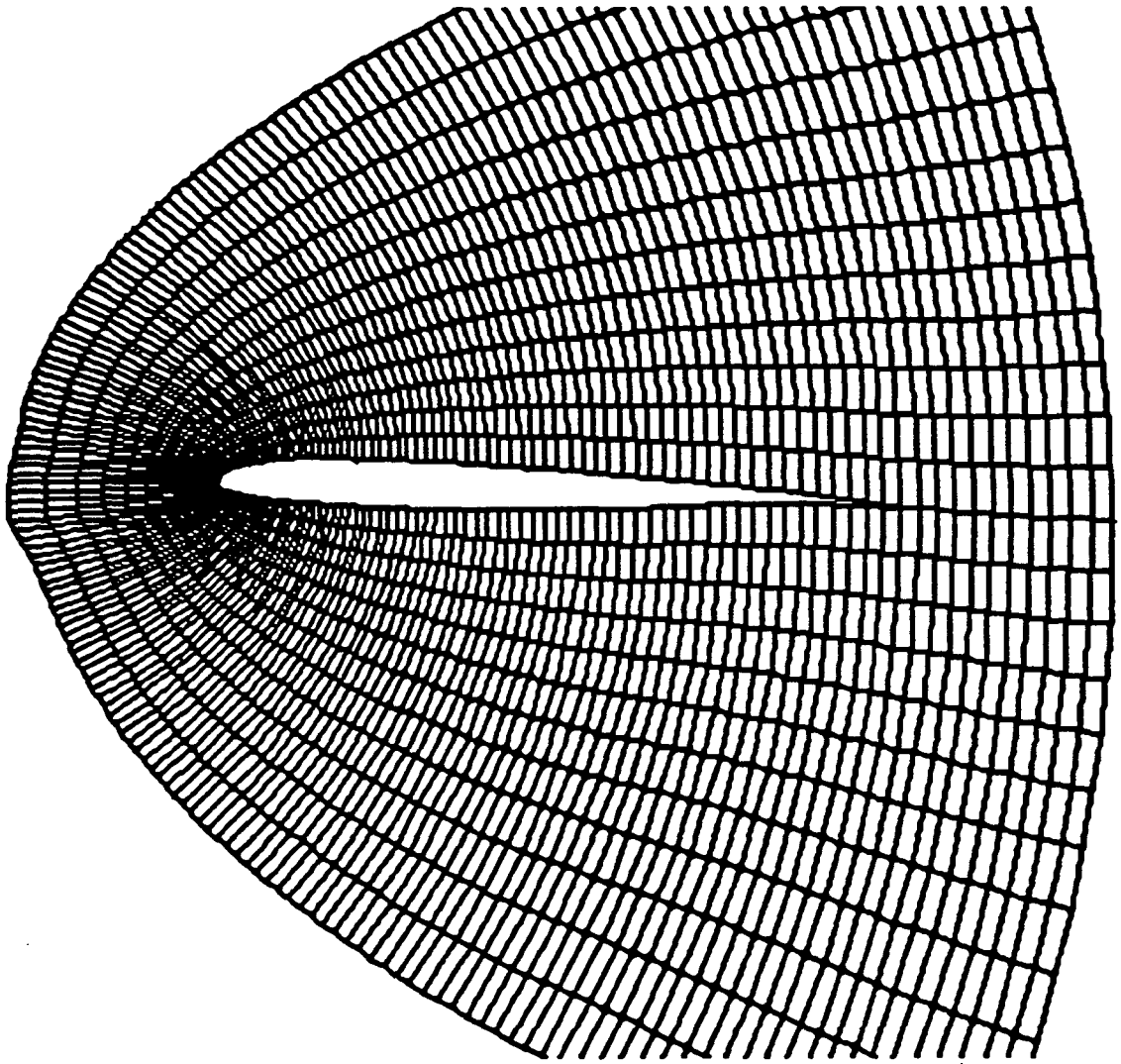


Figure 5.11: Wing C Symmetry Plane Grid

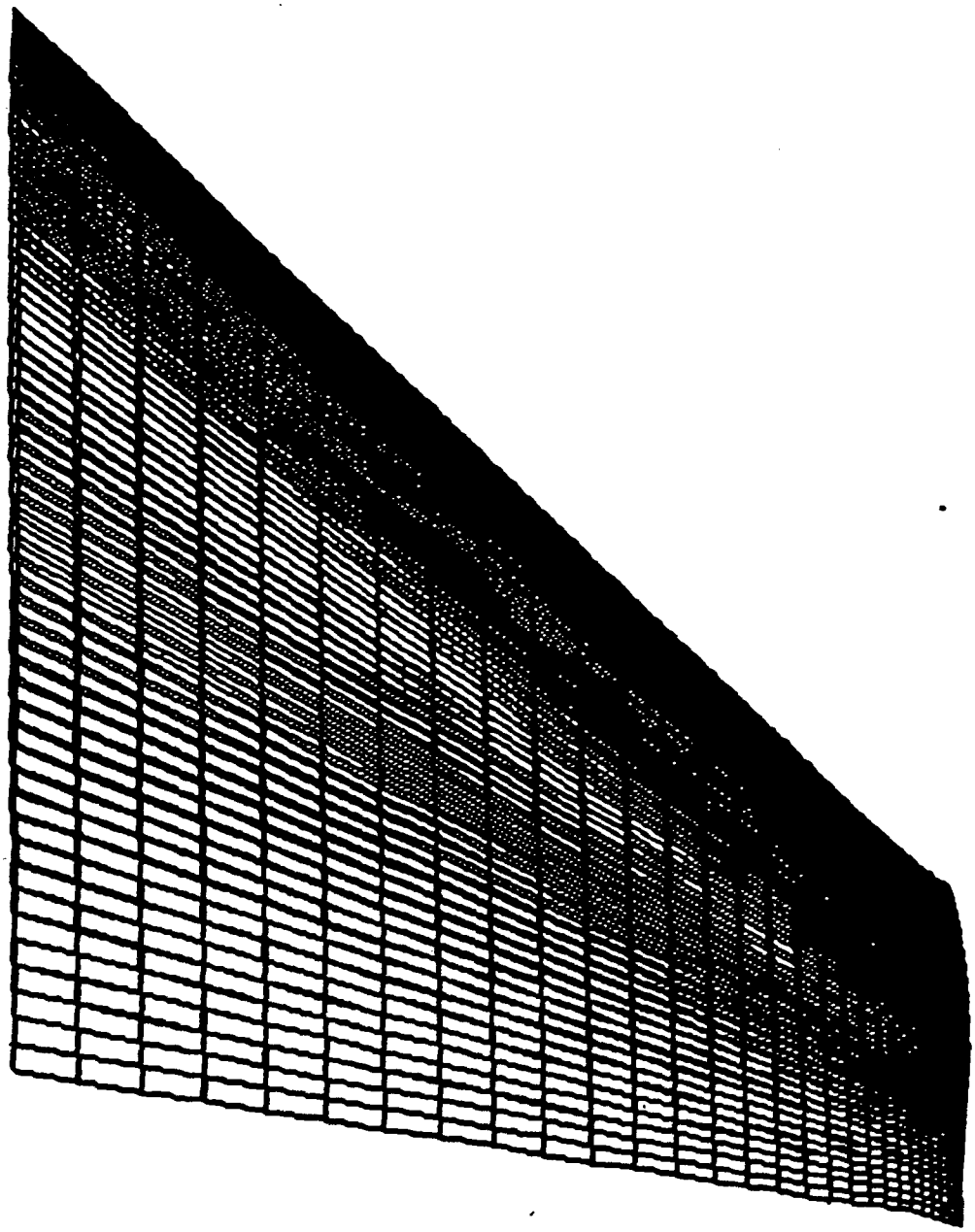


Figure 5.12: Wing C Planform Grid

The grid has a C-O topology. The total number of points in each zone is shown in Figure 5.13. The largest of the seven zones contains 58695 nodes and the smallest zone contains 9867 nodes. Euler solutions were generated on both the HP and DEC systems for a Mach number of 0.89 and an angle of attack of 5 degrees. All cases were run for 1000 steps.

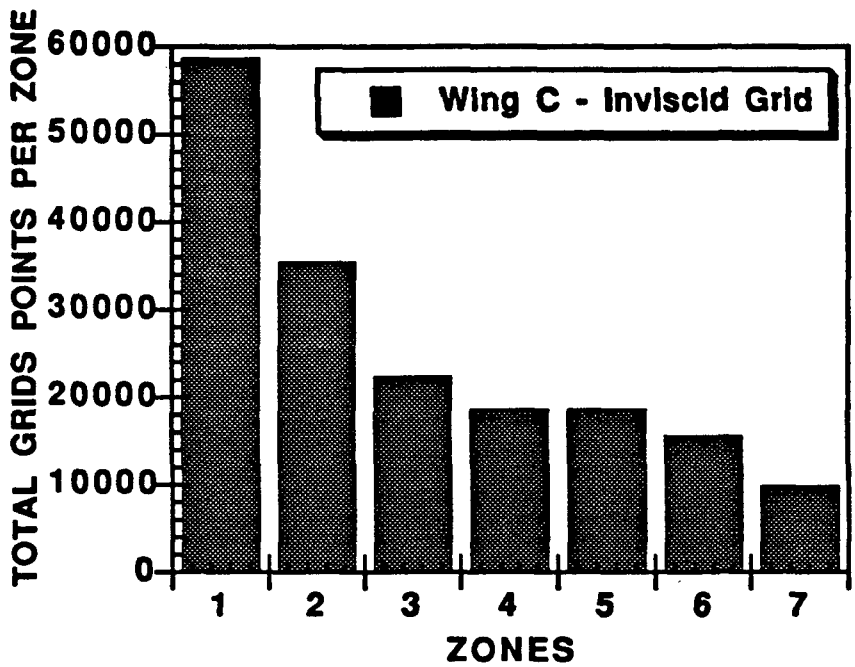


Figure 5.13: *Wing C Euler Grid Zonal Point Distribution*

The HP systems consisted of the three machines used previously along with a faster model 735 system. The Manager and one Worker process ran on the 735 system. The load balance in terms of total grid points per processor was as follows: 29522 points and 38012 points on the Model 720 systems, 54080 points on the Model 730, and 58695 on the model 735. This distribution was chosen to fit each grid into the memory available on each system. Four processors were used on the DEC systems with the Manager and the three Worker processes on separate systems. For the distributed runs on the DEC systems, the

ad hoc load balance on the three Worker processes was 66534, 54080, and 58695 points. All cases were run for 1000 steps. A set of initial runs were made on each system with both the baseline and distributed codes. For these cases, turnaround performance of the distributed code was worse than the baseline code. For example, on the HP system the baseline code required 19620 seconds for 1000 steps and the distributed code required 29942 seconds. This performance degradation was traced to the heavy utilization of the model 735 by another user which lead to excessive job swapping by the operation system. A similar degradation in performance was encountered on the DEC system.

At this point in the research, the Task Queue load balancing procedure was implemented to attempt to minimize the effects of system utilization and load imbalance. Because of the heavy utilization on the HP system, the initial tests of the Task Queue load balancing procedure was performed on the DEC system. A series of runs were made to determine the maximum and minimum performance of the distributed solver on the DEC systems using roughly equal ad hoc load balance and the balance obtained by the Task Queue approach. The maximum and minimum turnaround times in seconds for the baseline code running on a single processor , the ad hoc approach, and the Task Queue approach are given in Figure 5.14. The best time obtained for the baseline code was 21060 seconds. With the ad hoc approach, the minimum time obtained was 14100 seconds and the maximum time obtained was 23430 seconds. The minimum time represents a speedup of 1.49. For the Task Queue, approach the minimum time was 16410 seconds and the maximum time was 27970 seconds. The minimum time for the Task Queue approach represents a speedup of 1.28. The degradation in performance of the Task Queue approach was due to the load balance obtained by the procedure. For the Task queue approach, the distribution obtained on the three Workers was 84149, 54080, and 41080 points. The swapping introduced by the utilization of one of the Workers by another user led to the this load distribution. This indicates that the procedure needs to be refined to optimize the load

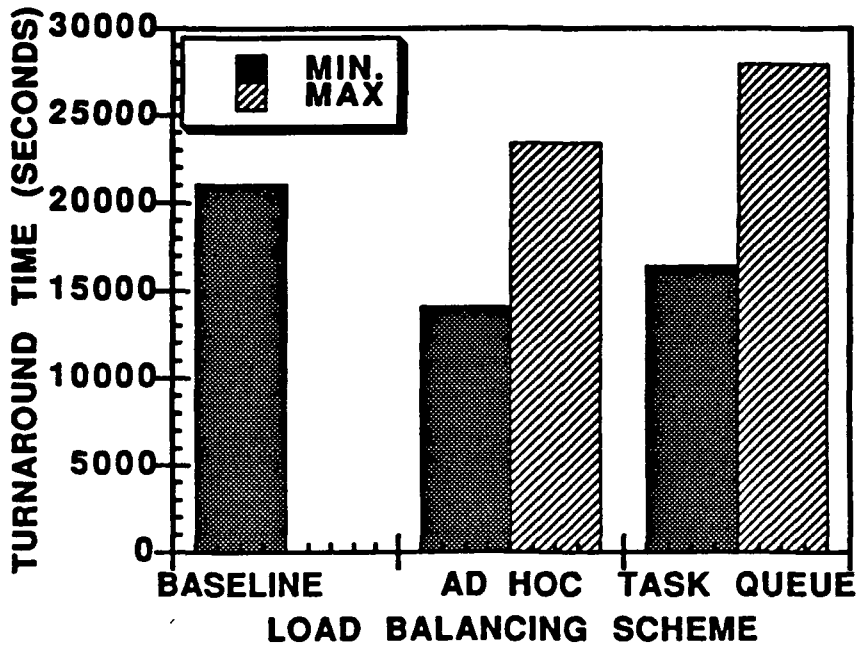


Figure 5.14: Performance of Ad Hoc and Task Queue Load Balancing Schemes

distribution among the processors. The maximum and minimum times achieved with the distributed code for both load balancing approaches again illustrate the sometimes drastic effect that system load has on overall performance on an Ethernet based network system. The minimum times were obtained in one of the few periods that all the worker processors were either idle or very lightly loaded.

The tests with the initial version of the code validated the Manager-Worker strategy used to implement the distributed solver and pointed out the need for an efficient load balancing procedure. At this time, the focus of the research shifted to developing the second version of the code using Version 713 of TEAM as the baseline code. This work was performed at the NASA Ames Research Center on a system of SGI workstations.

5.2 Implementation and Validation of the Second Distributed Solver

Based on the experience gained in the development of the first distributed solver, a different approach was adopted for implementing the PVM modifications into the baseline code to form the distributed solver. All modifications for the distributed solver were implemented into the baseline code using the C language preprocessor and the UNIX *make* utility to turn appropriate code segments on and off. This allowed a single code base to be maintained instead of the two separate sets of source files used in the first solver. In addition, the calls to the PVM library routines were embedded in wrapper subroutines that acted as an interface between the baseline routines and the PVM routines. The second version of the code was used to compare the performance of the Crutchfield static load balancing procedure and as the basis for implementing the implicit LU-SGS algorithm.

The distributed network used for these tests was the RFA cluster of SGI workstations in the NAS facility at the NASA Ames Research Center. The majority of the runs performed in these tests were made using systems of three or five 75 megahertz Indigo 2 systems for the Worker processes and a 50 Megahertz system for the Manager.

5.2.1 Comparison of the Performance of the Load Balancing Procedures

After the second implementation of the distributed solver was debugged, a series of runs were made to compare the performance of the two different load balancing approaches. The Wing C geometry for Euler simulations provided by Lockheed was used for these tests. All solutions were run for 500 time steps using the Flux Adaptive Dissipation (FAD) dissipation model with a second order dissipation coefficient of 0.1 and a fourth order coefficient of 1.5 and the fourth order Runge-Kutta explicit solver. Tests using three Worker processors were run first. These were followed by runs using five

Worker processes. The baseline code was run on a single processor to provide the baseline time used to compute the speed up. The best elapsed run time obtained for the single grid case was 16500 seconds.

Representative speedups obtained by the two load balancing schemes are shown in Figure 5.15. The static balancing approach required 12559 seconds for three Worker processors and 13860 seconds for five Worker processors. The Task Queue approach required 12844 seconds for three Worker processors and 11893 seconds for five Worker processors. The performance degradation for the static balancing five processor case was traced to a combination of the load imbalance obtained by the scheme and the extra systems and communications overhead required by the additional processors.

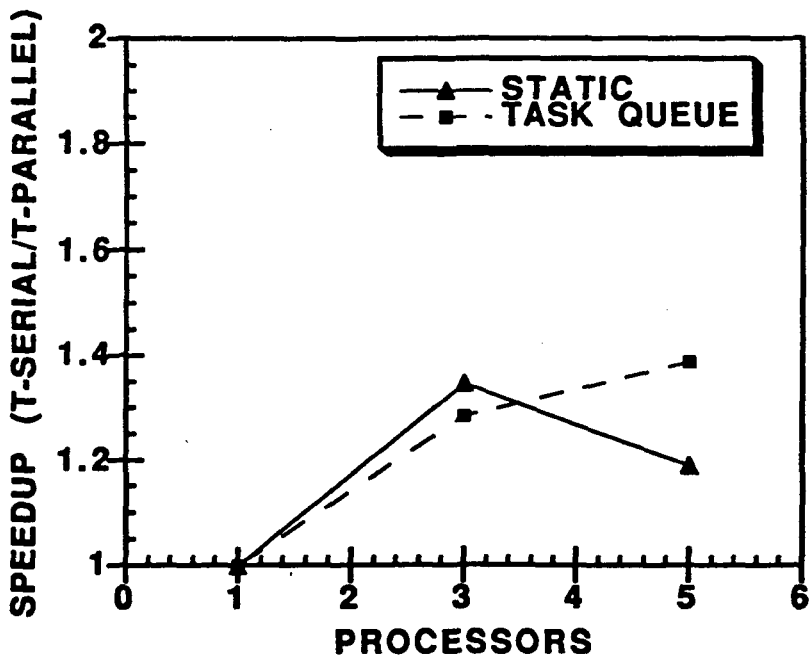


Figure 5.15: Comparison of Speedup for Static and Task Queue Balancing SGI Systems

The load balances obtained by the two schemes are shown in Figures 5.16 and 5.17. The improvement in performance obtained by the Task Queue procedure for the five Worker case is felt to be primarily due to the more favorable balance obtained by the procedure and a reduced level of system overhead. It was concluded from these tests that both schemes perform about the same on lightly loaded systems. The Task Queue approach has a slight advantage for networks with random system loads because it incorporates some of the effects of system and network load into the load balance.

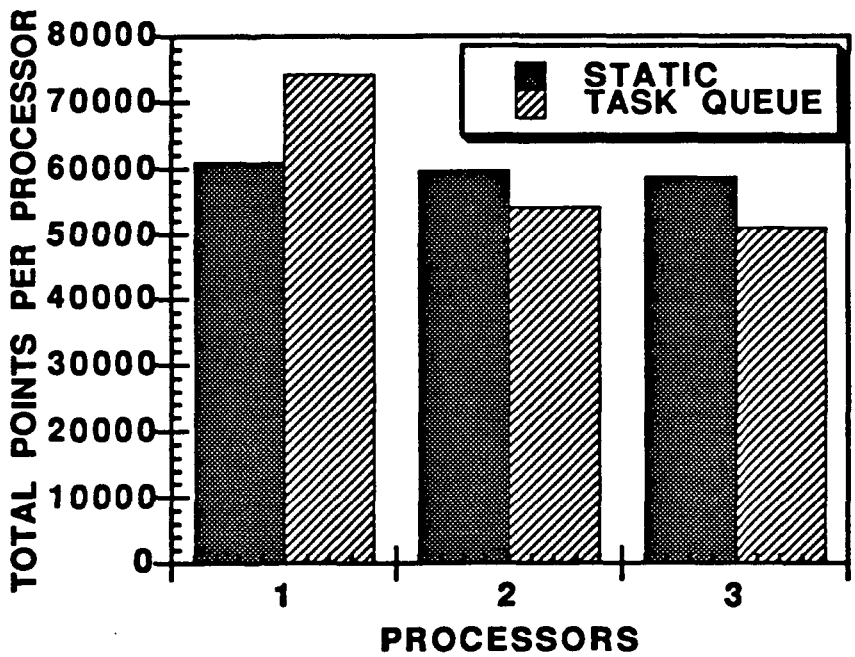


Figure 5.16: Static and Task Queue Load Balance for Three Processors

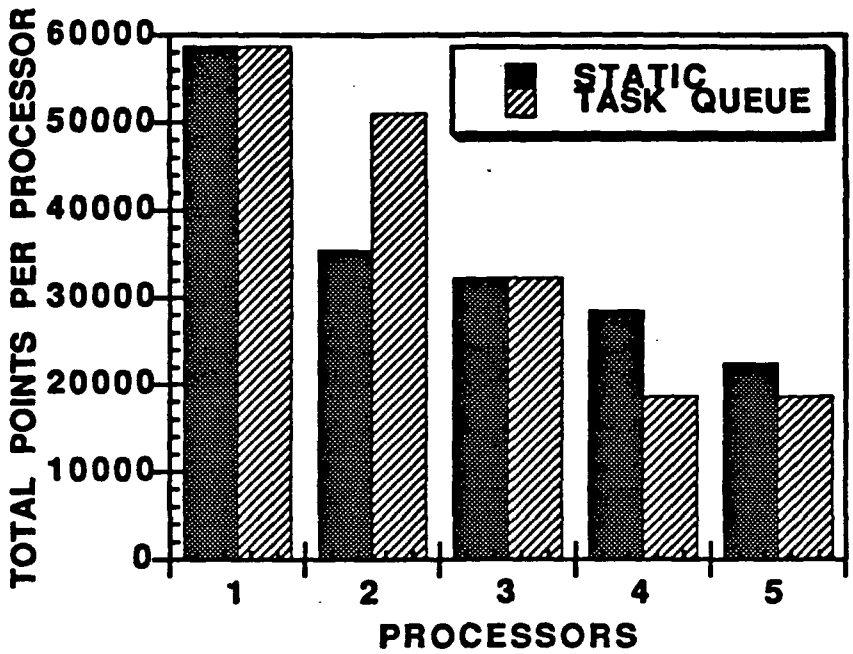


Figure 5.17: Static and Task Queue Load Balances for Five Processors

5.2.2 Implementation of the Implicit Solver

The next phase in the development of the second version of the solver was the implementation of the LU-SGS implicit solver. After a series of tests to debug the coding, a single processor case run using the inviscid Wing C case to obtain the turnaround performance of the baseline code. The best time obtained was 12000 seconds. This represents a substantial improvement over the 16500 seconds required by the Runge-Kutta scheme. This improvement is due to the reduced computational effort of the implicit scheme at each time step. The explicit scheme requires four flux evaluations and extra dissipation evaluations for each step. The implicit scheme only requires one flux evaluation and one dissipation evaluation per step. This offsets the computational effort required to compute the flux Jacobians. The distributed version of the implicit solver was run using the static

load balancing procedure. The best elapsed time for the distributed solver was 6281 seconds using three Workers which represents a speed up of 1.91. The load balance for this case was the same as the three processor Runge-Kutta case. The speed up for the three processor Runge-Kutta case with static balancing was 1.3.

The variation in code performance obtained on the network based systems illustrates the problems that can be encountered when running on Ethernet based systems. In addition to the load balance dictated by the grid system used in the analyses, the systems and communications load were found to play a substantial role in overall code performance. This variation in performance and the limited availability of a sufficient number of workstations led to the development of a third version of the flow solver on the large scale IBM SP2 distributed supercomputing system at NASA Ames Research Center.

CHAPTER VI

STEADY AND UNSTEADY SIMULATIONS ON THE NAS SP2

A series of steady and unsteady simulations were performed on the IBM SP2 system at the NASA Ames Research Center NAS facility. The NAS SP2 system is a large scale distributed system placed at NASA Ames by IBM as part of the High Performance Computing and Communications Program [128]. The SP2 system at the NAS facility is composed of 160 processors or "nodes" connected with a high speed communications subsystem [129]. Each node is an off-the-shelf IBM 590 workstation with a clock speed of 66.7 megahertz and a peak floating point speed of 250 megaflops per second. All the nodes on the NAS SP2 system have at least 128 megabytes of memory. The NAS SP2 system is representative of the current state-of-the art in large scale distributed parallel systems based on workstation technology.

The second version of the distributed flow solver was used for all the simulations performed on the SP2. Initial steady flow simulations were made to validate the implicit version of the code and to compare its performance to the explicit code. These tests were made using the inviscid Wing C case provided by Lockheed. In addition, thin layer Navier-Stokes simulations were performed using a large viscous grid system supplied by Lockheed.

Unsteady flow simulations were performed for the F5 wing geometry [130,131] using the implicit version of the solver. For the unsteady simulations, the modal motion of the wing oscillating in pitch was modeled. A series of tests were performed to determine the

effect of the domain decompositions used with increasing number of processors on the time accuracy of the solution. These tests were also used to compare the performance of the Manager/Worker and Worker/Worker communications strategies. In addition to the unsteady simulations, a series of steady flow simulations were performed with the F5 geometry to determine the effect of the size of the fixed time step on solution accuracy and convergence.

6.1 Wing C Euler Simulations

The previously described seven zone inviscid Wing C geometry and test case was run using the Manager/Worker communications strategy. The principle objectives of these tests were to continue the validation of the implicit algorithm and to compare its accuracy and performance with results from the explicit scheme. As before, the Mach number and angle of attack for these cases were 0.83 and 5 degrees. These runs were made using the Modified Adaptive Dissipation model with the default dissipation coefficients. Load balancing was performed using the static load balancing algorithm. Solutions were run for 400 steps with both the implicit and explicit schemes using systems of three, five and seven SP2 nodes. The load balances for the three and five processor cases were the same as those shown in Figures 5.16 and 5.17 for static balancing. For the seven processor case, each zone was assigned to a different processor. The implicit scheme was run using the infinite time step mode. The explicit scheme was run with a CFL number of 6. The run lengths were limited to 400 steps so that single processor tests could be performed in the time limits set for interactive jobs on the SP2 system that were in place when these cases were run. For these tests, the zonal boundary updates in the implicit scheme are lagged to the previous time step when more than one grid is assigned to a processor. Local updates are used in the explicit solver runs.

The turnaround times for the explicit and implicit versions of the baseline and distributed codes using one, three, five, and seven processors are compared in Figure 6.1.

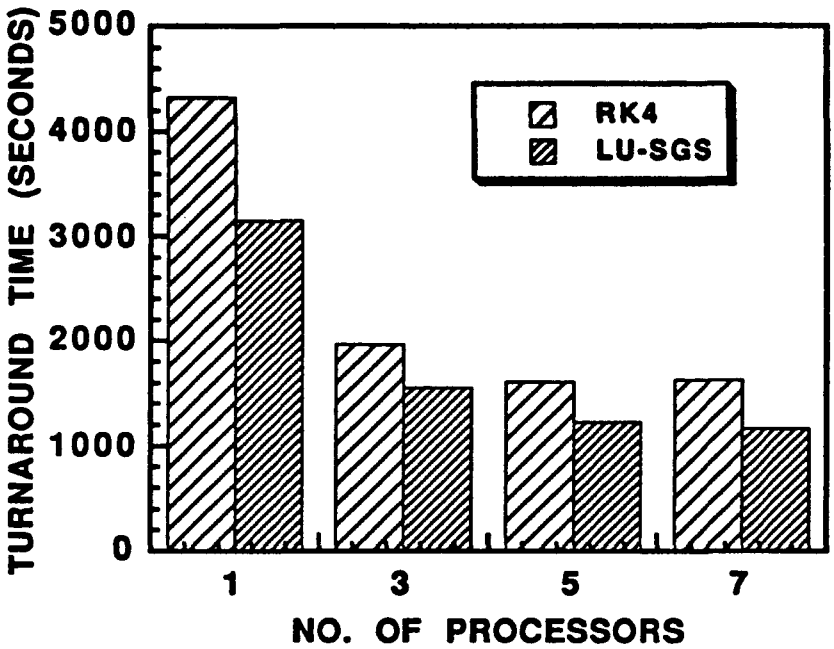


Figure 6.1: Comparison of Turnaround Peformance For The Explicit and Implicit Solvers

The explicit scheme required 4341 seconds for 400 steps on a single processor. The distributed code required 1962 seconds with three processors, 1610 seconds with five processors, and 1627 seconds with seven processors for the explicit scheme. The implicit scheme required 3151 seconds on a single processor, 1555 seconds for three processors, 1227 processors for five processors, and 1167 seconds for seven processors. The improved performance of the implicit scheme is felt to be due to several factors. The primary difference is due to the fact that the implicit scheme is more efficient than the explicit scheme because of the decreased number of flux and dissipation evaluations required during each step. The improvements in performance obtained with increasing

numbers of processors was not expected due to the load imbalance of the five and seven processor cases. This is felt to be due to the reduction in the largest load assigned to a single processor and the smaller message sizes that must be routed through the Manager at each step. This leads to a reduction in the time spent by the PVM routines in setting up and transmitting the messages to each processor and in the idle time spent by the processors with smaller loads as they wait for the most loaded processor to finish. Even though more messages are being sent, the latency period for transmitting and receiving the messages is less. This in turn leads to less idle time on each processor.

The relative speedups for both schemes are shown in Figure 6.2. The maximum speed up of the explicit scheme occurs with five processors and has a value of approximately 2.69. The implicit scheme obtains a maximum speedup of 2.7 with seven processors.

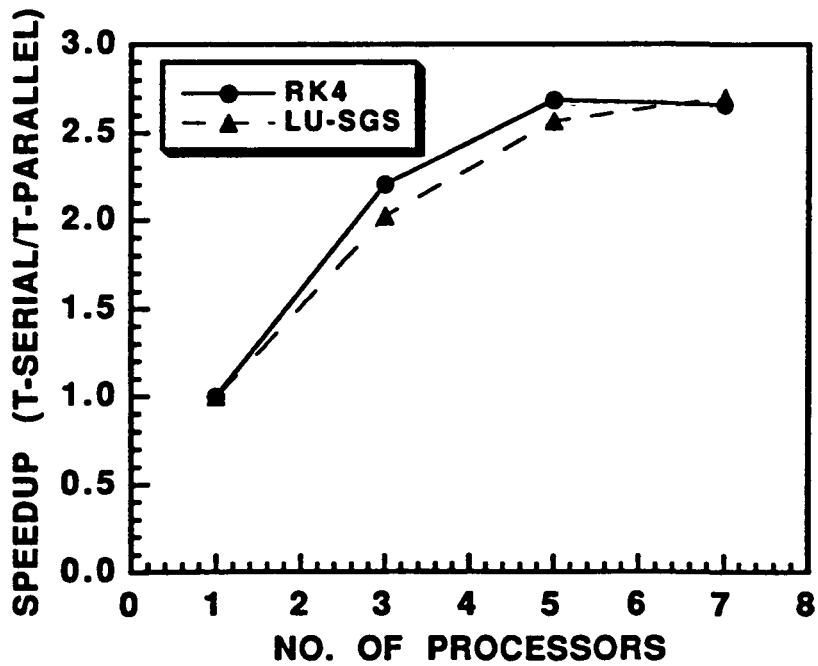


Figure 6.2: Comparison of Explicit and Implicit Scheme Speedups

Figures 6.1 and 6.2 illustrate the effect of load imbalance on speed up with increasing numbers of processors. An equal load on each processor would lead to improved performance. In addition, these results illustrate the advantage of a distributed system with a switched high speed communications subsystem over the Ethernet based distributed systems used in the first phase of the research. The maximum speedups of 2.7 approach the ideal speedup of approximately 3.05 which assumes that all communications are instantaneous.

The total wing lift, drag, and pitching moments (CL, CD, and CM) along with the number of supersonic points and the average change in density with time is shown in Table 6.1 for the implicit and explicit schemes.

Table 6.1: Total Wing Load Coefficients and Convergence Data on Different Processors For the Explicit and Implicit Solvers

VALUE	RK / 1P	RK / 3P	RK / 5P	RK / 7P	LU / 1-7P
CL	0.560777	0.56078	0.560729	0.560745	0.55533
CD	3.9198E-2	3.9197E-2	3.9179E-2	3.9183E-2	3.8764E-2
CM	-0.134185	-0.134817	-0.13475	-0.134765	-0.134475
NSUP	13,011	13,006	13,014	13,012	12,887
DR/DT	1.195E-2	1.202E-2	1.227E-2	1.225E-2	3.4393E-2
L2(1)	2.584E-5	2.553E-5	2.590E-2	2.549E-2	2.425E-6

The results for the explicit solver indicate that the domain decomposition and modified boundary update procedure has only a slight effect on computed loads. The computed values for the implicit scheme do not change with increasing numbers of processors because of the globally lagged boundary updates. No spatially varying time step was used in the implicit calculations. However, a spatially varying step was calculated using a CFL number of 6 to provide a consistent value of time step for computing the average change in

density with time. The code was modified to compute a second convergence parameter, the Root Mean Square (RMS) average norm of the residual of the continuity equation. This parameter is scaled to remove the effects of time step from the right hand side of the discretized equations. The average norm is given by

$$L2(1) = \frac{\sqrt{\sum R(1)^2}}{NC} \tag{6.1}$$

where R(1) is the residual of the continuity equation and NC is the number of interior cells. The values of L2(1) are also given in Table 6.1.

The differences in the loads and number of supersonic points computed by the implicit and explicit schemes indicates that the two schemes are converging at different rates. A second set of runs were made to quantify the effect of levels of convergence on the loads computed by the two schemes. Five processors were used for these tests. The explicit and implicit schemes were first run for 1400 steps. At this point an examination of the change in the number of supersonic points at each step showed that the explicit scheme was almost converged. However, the implicit scheme showed a small but noticeable oscillation in the number of supersonic points at the end of 1400 steps which indicated that a steady state had not been reached. The implicit scheme required another 600 steps to achieve the same level of convergence as the explicit scheme. The force and moment coefficients computed by the explicit and implicit schemes along with the number of supersonic points and the time required for each solution are given in Table 6.2. After 2000 steps, the number of supersonic points computed by the two schemes are the same. The lift coefficients vary by one count where a count is an increment of 0.0001. The drag and pitching moment coefficients are seen to vary by less than a count. These results indicate that the explicit and

implicit schemes will produce the same results for converged solutions. On an elapsed time per step basis, the implicit scheme is more efficient. However, the time required to achieve similar levels of convergence is about the same.

Table 6.2: Computed Load and Convergence Data for the Explicit and Implicit Schemes After 1400 and 2000 Steps

VALUE	RK /1400 STEPS	LU /1400 STEPS	LU / 2000 STEPS
CL	0.558187	0.558180	0.558083
CD	0.039248	0.039198	0.039245
CM	-0.13676	-0.13656	-0.13672
NSUP	12,993	13,012	12,993
TIME(SEC)	5076	3969	5666

The results of the Euler tests indicated that both the explicit and implicit versions of the distributed solver provided acceptable performance and accuracy for inviscid simulations. The next step in the research was to validate the implicit and explicit solvers for viscous simulations.

6.2 Wing C Viscous Simulations

A seven zone viscous grid for the Wing C geometry was obtained from Lockheed. This grid system has a total of 479343 nodes and has a C-O topology. This grid system is representative of the computational grids used in industry to validate Navier-Stokes solvers. The number of points in each zone is shown in Figure 6.3. The largest zone in the grid system contains 111725 points.

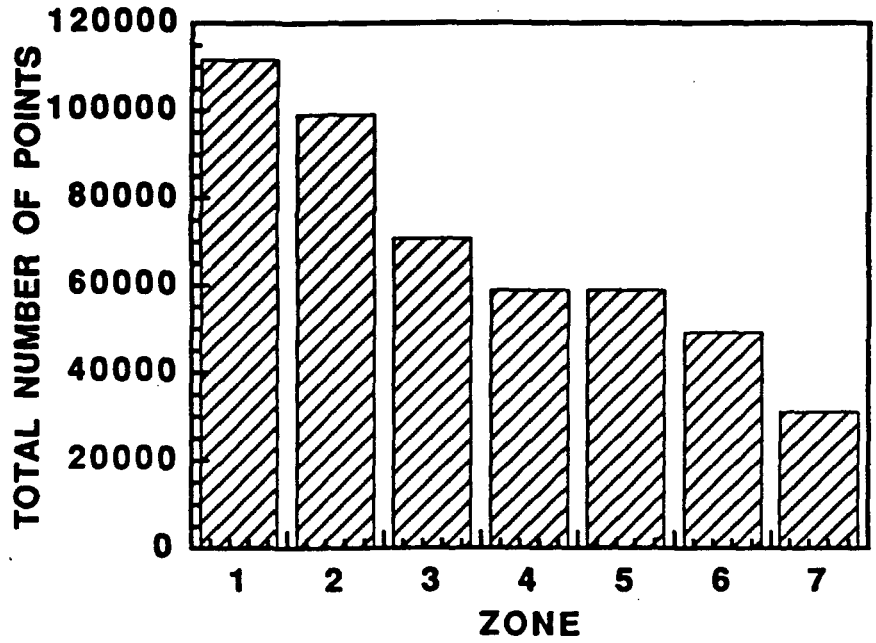


Figure 6.3: Grid Point Distribution for the Viscous Wing C Grid

A standard test case for the viscous grid system was also obtained from Lockheed. As in the inviscid case, the viscous case was run with a freeestream Mach number of 0.85 and an angle-of-attack of 5 degrees. The thin-layer approximation to the Navier-Stokes equations was used to model the viscous stresses. The Baldwin-Lomax turbulence model was used with transition fixed at constant chordwise stations along the span. The distributed viscous Wing C tests were run using 5 processors. The load balance obtained by the static balancing scheme is shown in Figure 6.4.

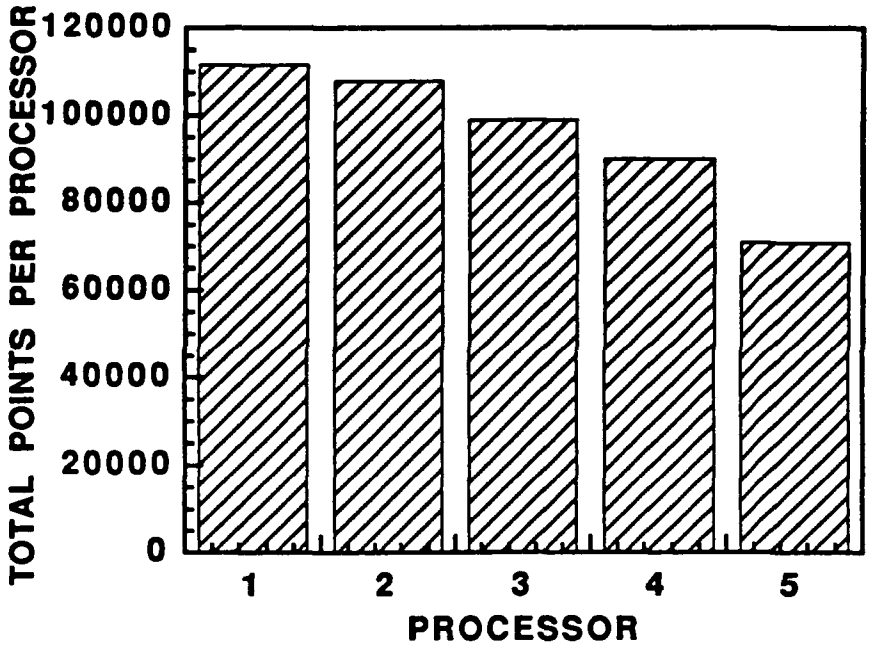


Figure 6.4: Load Balance for Viscous Grid with Five Processors

The test case supplied by Lockheed used the Matrix Based Dissipation model with the second and fourth order dissipation coefficients set to 0.15 and 1.5. The explicit solver used four stages with a CFL number of 4. The dissipation was computed at two stages in the solution in the explicit solver. Residual smoothing was used to accelerate convergence. No enthalpy damping was used since the constant total enthalpy condition is inconsistent with the non-isentropic nature of viscous flows. The implicit solver was run in the infinite time step mode with a CFL number of 4 used to compute a spatially varying time step used in the calculation of the average change in density with time. Attempts were made to run with the single processor versions of the code. However, the memory requirements for in core solutions on a single processor led to excessive paging. This in turn led to extremely slow processing speeds. For instance, ten steps with the single processor version of the explicit solver required 1100 seconds. Therefore, no long single processor tests were

made. However, both the implicit and explicit versions of the distributed code were run for 2000 steps using the Manager/Worker communications strategy. This illustrates another important advantage of distributed systems based on workstation technology. Large problems can be broken into smaller problems that can easily fit in the available memory of each processor.

The initial distributed tests using the values of the dissipation coefficients supplied by Lockheed produced solutions with stalled convergence rates. The average of the change in density with time for the supplied dissipation coefficients are shown in Figure 6.5.

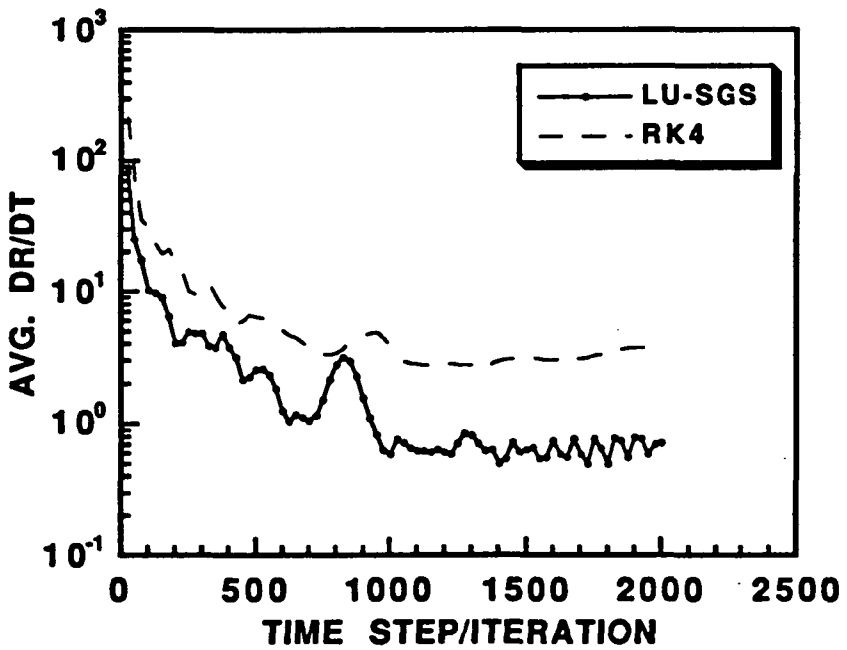


Figure 6.5: Convergence of Explicit and Implicit Schemes With Initial Dissipation Values

A second set of runs were made with the magnitudes of the second and fourth order dissipation coefficients increased to 0.5 and 2.0. The increased levels of dissipation resulted in the improved convergence rates shown in Figure 6.6.

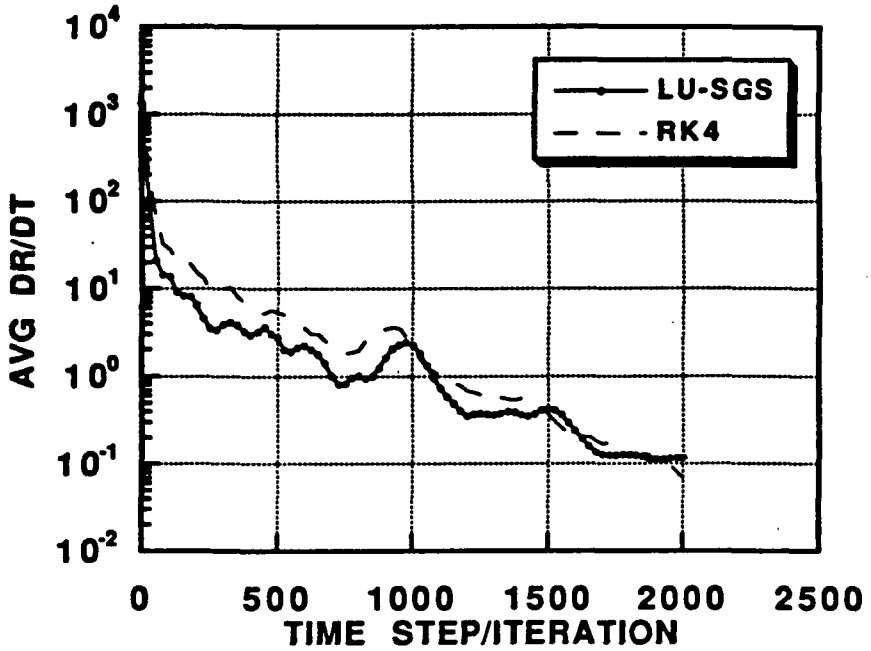


Figure 6.6: Convergence of Explicit and Implicit Schemes With Increased Dissipation

For this viscous case, the LU-SGS scheme required 16940 seconds and the RK4 scheme required 25823 seconds to obtain approximately the same level of convergence. The computed lift and drag coefficients along with the number of supersonic points obtained by the two solvers are given in Table 6.3.

Table 6.3: Comparison of Total Loads and Convergence Data for theViscous Wing C Case Using the Explicit and Implicit Solvers

VALUE	RK4	LU - SGS
CL	0.54650	0.550168
CD	0.045467	0.045796
NSUP	36,885	39,640

The implicit scheme produced a lift coefficient of 0.5502 and a drag coefficient of 0.045796 at the end of 2000 steps. This compares well with a lift coefficient of 0.5465 and a drag coefficient of .045467 produced by the explicit scheme.

The viscous Wing C tests validated the performance of the distributed versions of the explicit and implicit solvers for steady viscous simulations. The next phase of the research examined the performance of the implicit solver for unsteady flow simulations. The explicit solver was not used for these tests because of its increased computational requirements and the prohibitively small time steps that would be required to maintain stability.

6.3 F5 Wing Simulations

A series of simulations were performed using the F5 wing geometry to determine the effectiveness of the implicit version of the distributed flow solver for unsteady flow simulations. The primary goal of these tests was to determine the effect of the boundary update procedure required by the distributed solver on solution accuracy for an oscillating wing. Results were generated for the modal vibration of the F5 wing oscillating in pitch and compared with the experimental results of Tijdeman et. al. [131]. The experiments of Tijdeman et. al. have been used by several authors to validate different computational methods for unsteady flow analysis [132-140]. The performance of the Manager/Worker and the Worker/Worker communication strategies was evaluated for distributed systems of up to 18 processors. In additions, steady simulations were performed to determine the effect of time step size on solution accuracy and performance.

6.3.1 F5 Wing Test Configuration

The test configuration used in the experiments of Tijdeman et. al. is shown in Figure 6.7. In the unsteady experiments, the wing was subjected to a sinusoidal oscillation in pitch about the half chord location of the root section. The unsteady pressures and the modal response of the wing were measured for a range of Mach numbers, mean angles of attacks, and frequencies of oscillations.

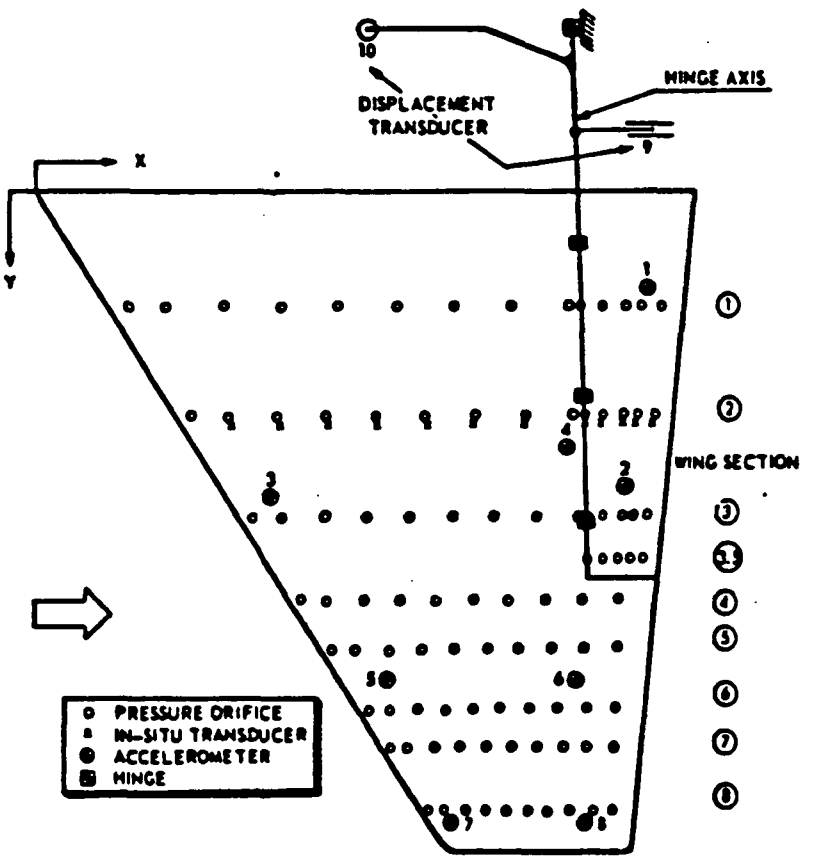


Figure 6.7: F5 Experimental Test Configuration (Reference 130)

The spanwise locations where pressures were measured are given in Table 6.4.

Table 6.4: Spanwise Locations of Experimental Pressure Data for the F5 Wing

Station No.	y/b (%)
1	18.1
2	35.2
3	51.2
4	64.1
5	72.1
6	81.7
7	87.5
8	97.7

The modal response of the wing was obtained by integrating the local accelerations obtained from the accelerometers measured on the wing. An analytical function for the mode shape of the wing was obtained by integrating the accelerometer data twice. This analytical mode function combines both the bending and torsion modes of vibration that occur on swept and tapered wings such as the F5 wing. The experimental test case used in the present research had a freestream Mach number of 0.896, a mean angle of attack of 0 degrees, an amplitude of oscillation of 0.111 degrees, and a frequency of oscillation about the root half chord location of 40 Hertz. The freestream Reynolds number for this test condition was 11 million based on root chord. The F5 wing model used in the experiments has a root chord of 0.6396 meters and a span of 0.6226 meters. The leading edge sweep angle is approximately 32 degrees.

The mode shape function obtained by Tijdeman et. al. for the chosen test case is given in Reference [131] as

$$W(x,y) = -0.329 + 0.977x - 0.088y + 0.244xy - 0.077y^2 - 0.091xy^2 \qquad (6.2)$$

where x is the chordwise ordinate and y is the spanwise ordinate. The procedures used to generate Equation (6.2) assumed there was no chordwise deformation and a parabolic spanwise deformation. In addition, Equation (6.2) is nondimensionalized such that the tangent of the angle of oscillation at the second experimental span station is equal to one.

In the current research, Equation (6.2) is used to define an angular displacement about the chordwise position at each span station at which the angular displacement given by the equation is zero. The spanwise locus of the chordwise positions of zero displacement define the node line for the wing. The modal motion of the wing is assumed to consist of a rigid rotation about the node line. The mean sinusoidal motion of the wing is defined by

$$\alpha(t) = \alpha_0 + \Delta\alpha \sin(\kappa t) \qquad (6.3)$$

where $\alpha(t)$ is the instantaneous angle of attack at time t, α_0 is the mean angle of attack, $\Delta\alpha$ is the amplitude of oscillation, and κ is the reduced frequency defined as

$$\kappa = \frac{2\pi Fc}{U} \qquad (6.4)$$

where F is the frequency in Hertz, c is a reference chord length and U is the freestream velocity. In the experiments of Tijdeman et. al., the reduced frequency is defined such that the value given in the Reference [131] is half the value computed by Equation (6.4). For a frequency of oscillation of 40 Hz and a freestream Mach number of 0.896, the reduced

frequency given by Tijdeman et. al. is 0.275. The value of reduced frequency used in this research is the value given by Equation (6.4) using the root chord as the reference length.

The local angular displacement at a constant span station can be defined in terms of the instantaneous mean sinusoidal motion of the wing given by Equation (6.3 and the mode shape given by Equation (6.2) as

$$\alpha(y,t) = \theta(y)\alpha(t) \tag{6.5}$$

where $\theta(y)$ is the mode shape at each span station based on a rigid rotation about the chordwise location of the local node. In the computational results, Equation (6.5) is applied to the two-dimensional grid planes defined at constant spanwise stations. The rotation of the grid planes are first performed about the node line. These rotations are then transformed to rotations about the root half chord location.

6.3.2 Computational Grid System

The grid system for the computational tests were generated analytically using the procedure used by Guraswamy [138] in the ENSAERO code. Two dimensional grids are generated at each spanwise location by marching outward from the inner boundary defined by the wing and wake surfaces to a fixed outer boundary distance along directions defined by the outward unit normal vectors at each point on the inner boundary. Stretching functions defined in the coordinate direction approximately normal to the inner boundary are used to define the increments in the x and z directions that are added to the coordinates of the previous coordinate surface to define the coordinates on successive surfaces. The y coordinate of each grid plane is held constant.

The grid generation procedure was used to generate a baseline C-H mesh system consisting of 151 streamwise points, 28 spanwise points with 19 or 20 points on the wing surface and 45 normal points. There are total of 190,260 points in the grid. There are 101 points on the body surface at each spanwise station. The region outboard of the wing tip and the wake surfaces aft of the wing trailing edge are modeled as a fluid surfaces of zero thickness. The initial grid normal grid spacing for Euler simulations was set at 0.001 root chords. The normal grid spacing for viscous simulations was set at 0.00002 root chords. Analytical stretching functions were used to cluster the grid in the normal directions and chordwise directions. This grid system was chosen to facilitate comparisons with results reported in References [136] and [137]. The symmetry plane and wing planform grid systems are shown in Figure 6.8.

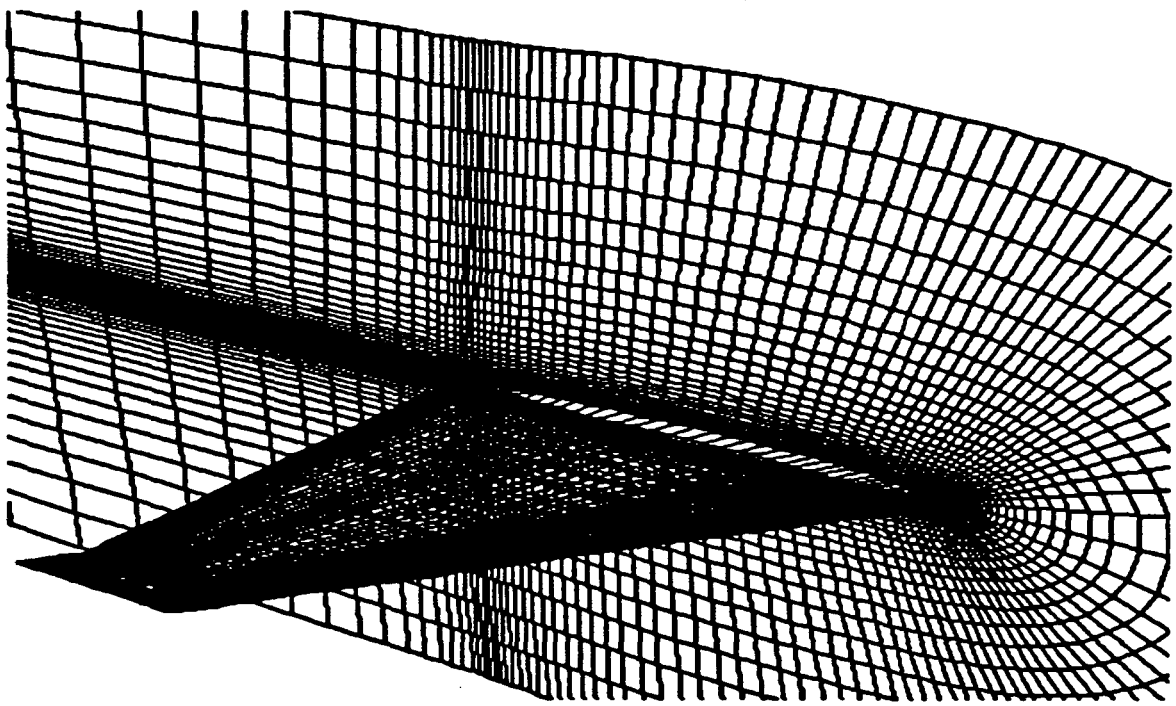


Figure 6.8 F5 Planform and Symmetry Plane Grids

For the distributed simulations, the grid was subdivided into 4, 9, and 18 zones with each zone containing the same number of points to maintain an equal load balance among processors. The 9 and 18 zone cases contained four spanwise stations in each zone and the 4 zone case contained 8 spanwise stations in each zone. The 18 zone system was obtained by splitting the 9 zone system in the normal direction. These grids were used to evaluate the performance of both the Manager/Worker and Worker/Worker communications strategies and to determine the effects of the domain decomposition on solution accuracy.

6.3.3 Results of Unsteady Simulations

A series of Euler and Navier-Stokes analyses were performed using the baseline C-H mesh. Euler simulations using the four zone grid system were performed first. All simulations were run for three cycles of oscillation with 1621 steps per cycle for a total of 4863 steps. This corresponds to a non-dimensional time step of approximately 0.005. The choice of this step size is based on the results of Mello [140] for the same test case. As in Reference [140], the reduced frequency used in the calculation of time step and the motion of the wing was redefined in terms of the freestream Mach number and a reference chord length of one meter to give consistent values for the non-dimensional parameters used in the flow solvers. The Matrix Based Dissipation model was used in these tests. This is equivalent to using an upwind biased scheme.

The unsteady pressure response for the oscillating wing is a complex function that can be broken into real and imaginary components [140]. The real and imaginary pressure coefficients are defined in terms of the instantaneous values of pressure coefficient at each time step by Fourier integrals of the form

$$Cp_{real} = \frac{M_\infty \kappa}{\pi} \int_{t_1}^{t_1+2\pi/M_\infty\kappa} Cp(t)\sin(M_\infty \kappa t)dt \qquad (6.6a)$$

$$Cp_{imag} = \frac{M_\infty \kappa}{\pi} \int_{t_1}^{t_1+2\pi/M_\infty\kappa} Cp(t)\cos(M_\infty \kappa t)dt \qquad (6.6b)$$

where the initial non-dimensional time t_1 is chosen to be large enough such that transients that appear in the initial phases of the solution have disappeared. The integrals are over the time period required for one complete cycle of motion. The real values components of the unsteady coefficients represent the components that are in phase with the prescribed motion. The imaginary components are out of phase with the motion.

The actual evaluation of the integrals in Equations (6.6) is performed by trapezoidal integration. In addition, the final values of the coefficients are scaled by the amplitude of oscillation in radians, $\Delta\alpha$. This scaling was used in the experimental results. The discretized equations for the unsteady coefficients can be written as

$$Cp_{real} = \frac{M_\infty \kappa \Delta t}{\pi \Delta\alpha}\left[\frac{Cp(m_S)+Cp(m_E)}{2} + \sum_{m=m_S+1}^{m_E-1} Cp(m)\sin(M_\infty \kappa m \Delta t)\right] \qquad (6.7a)$$

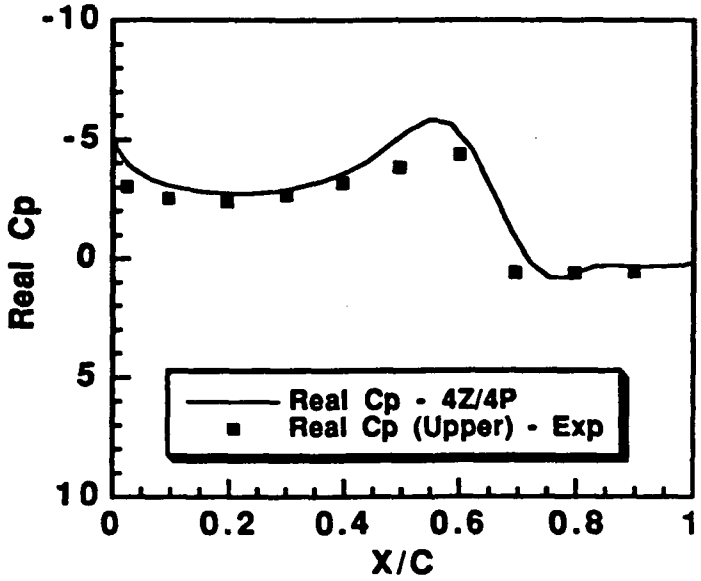
$$Cp_{imag} = \frac{M_\infty \kappa \Delta t}{\pi \Delta\alpha}\left[\frac{Cp(m_S)+Cp(m_E)}{2} + \sum_{m=m_S+1}^{m_E-1} Cp(m)\cos(M_\infty \kappa m \Delta t)\right] \qquad (6.7b)$$

where m_S and m_E are the indices of the time steps at the start and end of the cycle over which the integrals are evaluated. Cp(m) is the pressure coefficient at a point on the wing

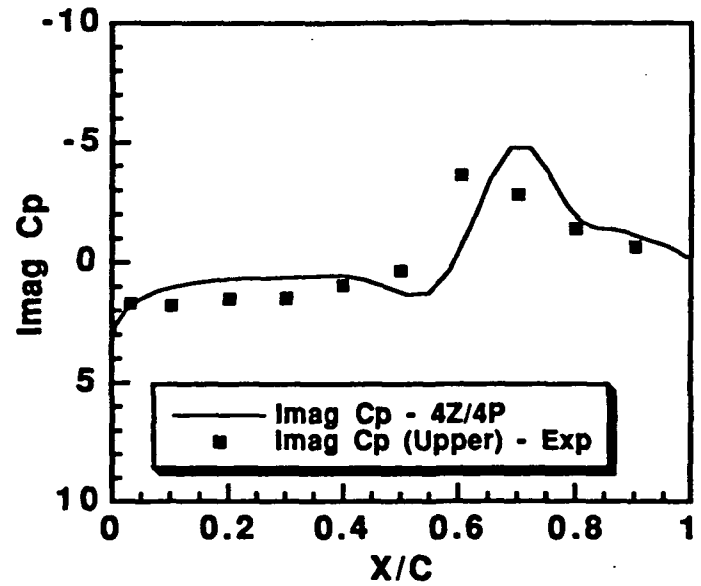
surface for time step m. For the unsteady simulations performed in this research, the real and imaginary pressure coefficients were computed over the third cycle of oscillation.

6.3.3.1 Unsteady Euler Simulations

The computed real and imaginary upper surface pressure coefficients for the four zone/ four processor Euler analysis are compared with digitized experimental data from Reference [131] in Figure 6.9 at spanwise computational grid locations of 35%, 51%, and 88%. The semispan locations of the experimental data are 35.2%, 51.2% , and 87.5%. The computed results presented in Figure 6.9 are in good agreement with both the experimental results and the computational results presented in References [136] and [137]. At the inner wing stations, the correlation of the computed real (in-phase) components of pressure with the experimental data is excellent. The computed shock location in imaginary (out-of-phase) component is slightly downstream of the experimental data. At the outboard wing station, the correlation for the real component shows a shock wave that is upstream of the experimental results and slightly stronger. However, the peak is consistent with the results obtained by Obayashi et. al. [136] using an upwind biased scheme. The correlation of the computed imaginary component of the pressure coefficient with the experiment is very good. On the whole, the computed results are in excellent agreement with the experimental data. It should be noted that the differences in the results are magnified by the pitch amplitude scaling used in both the computed and experimental data. An increment of one in the scaled data corresponds to a change of 0.002 in the actual data. Other differences can be attributed to viscous effects in the outboard region and inconsistencies in the experimental data. These results indicate the implicit scheme provides sufficient time accuracy for the 4 zone / 4 processor Euler simulations for a time step of 0.005. The turnaround performance

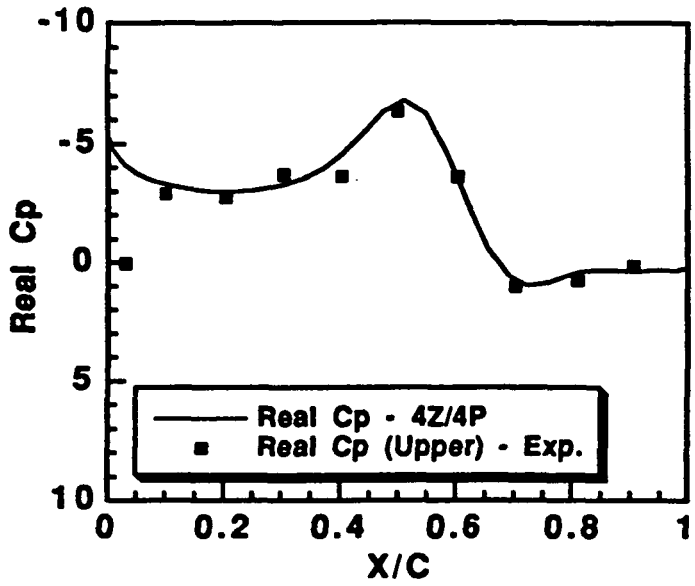


(a) Upper Surface Real Pressure Coefficient at 35% Span

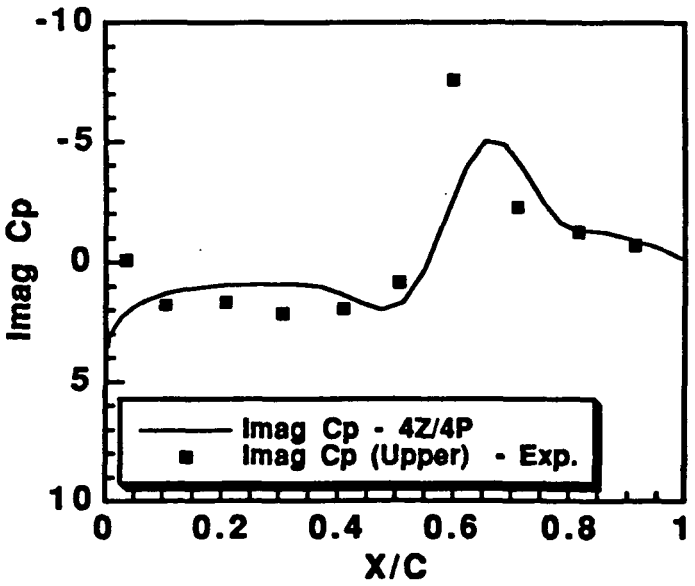


(b) Upper Surface Imaginary Pressure Coefficient at 35% Span

Figure 6.9: Four Processor Real and Imaginary Pressure Distributions for the For the Inviscid F5 Wing Case, M=0.9, F=40Hz.

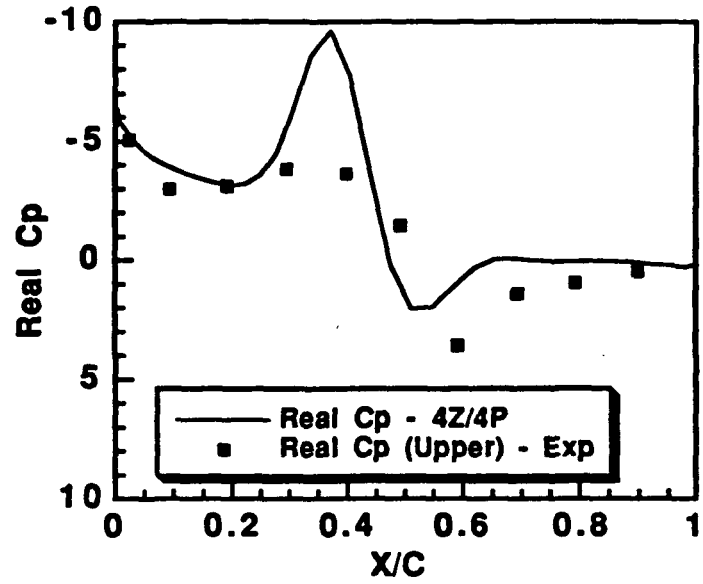


(c) Upper Surface Real Pressure Coefficient at 51% Span

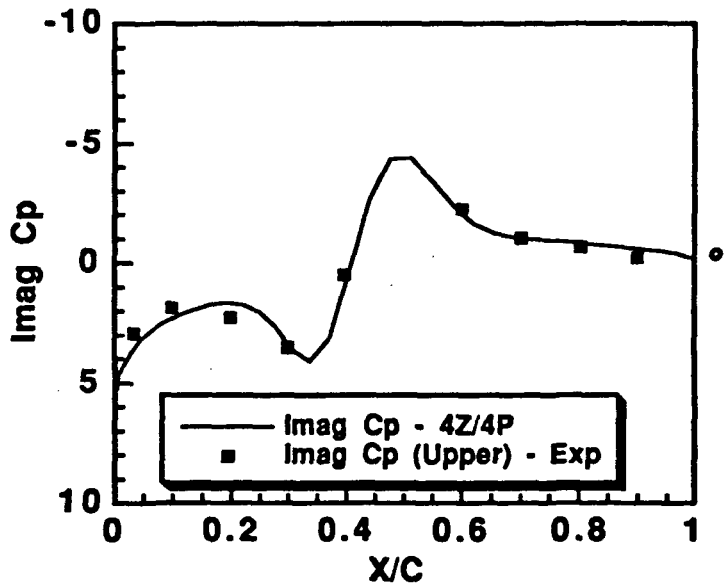


(d) Upper Surface Imaginary Pressure Coefficient at 51% Span

Figure 6.9: Four Processor Real and Imaginary Pressure Distributions for the Inviscid F5 Wing Case, M=0.9, F=40Hz (continued).



(e) Upper Surface Real Pressure Coefficient at 88% Span

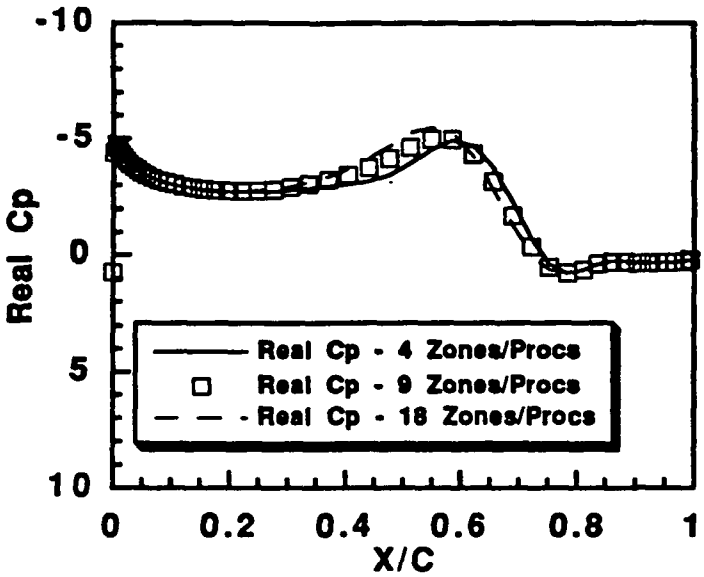


(f) Upper Surface Imaginary Pressure Coefficient at 88% Span

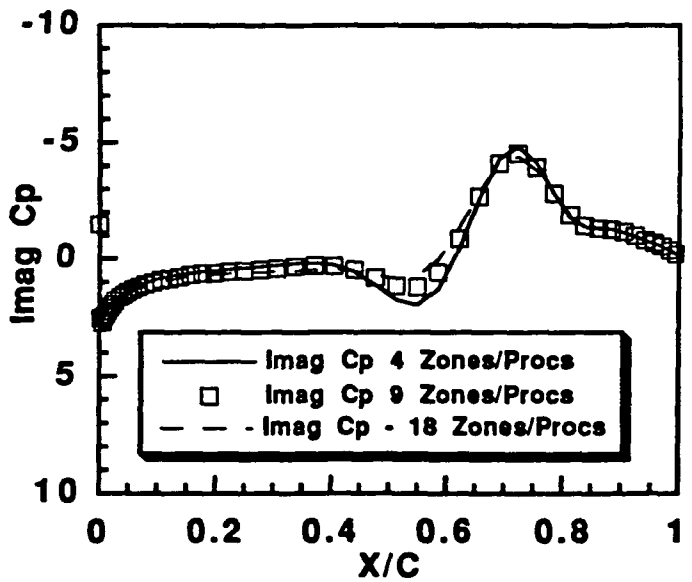
Figure 6.9: Four Processor Real and Imaginary Pressure Distributions for the Inviscid F5 Wing Case, M=0.9, F=40Hz (concluded).

for this case on the SP2 using the Manager/Worker communications strategy was 18174 seconds.

Next, the 9 zone / 9 processor and 18 zone / 18 processor cases were run to determine how turnaround performance scales with increasing number of processors and the effect of the domain decomposition on the accuracy of the implicit scheme. Both the 9 zone and eighteen zone grid systems were purposely limited to a maximum of 4 spanwise grid planes per zone. This makes the solution in each zone almost two-dimensional. In addition, the dissipation can no longer be evaluated at all the interior planes using a five point stencil. The object was to determine the lower bound for the spanwise domain decomposition that would maintain an accurate solution. The real and imaginary upper surface pressures obtained at the 33% and 88.5% of span stations for the 9 and 18 zone cases are compared with the results for the four zone case obtained at the 35% and 88% span stations in Figure 6.10. The difference in the location of the span stations is due to the introduction of an additional grid plane to maintain an even load balance. Reducing the number of spanwise grid planes per block from eight to four is seen to have a slight effect on accuracy at the inboard station. Differences are more pronounced at the outboard station. These differences can be attributed to the loss of implicitness in the solution due to the decreased number of

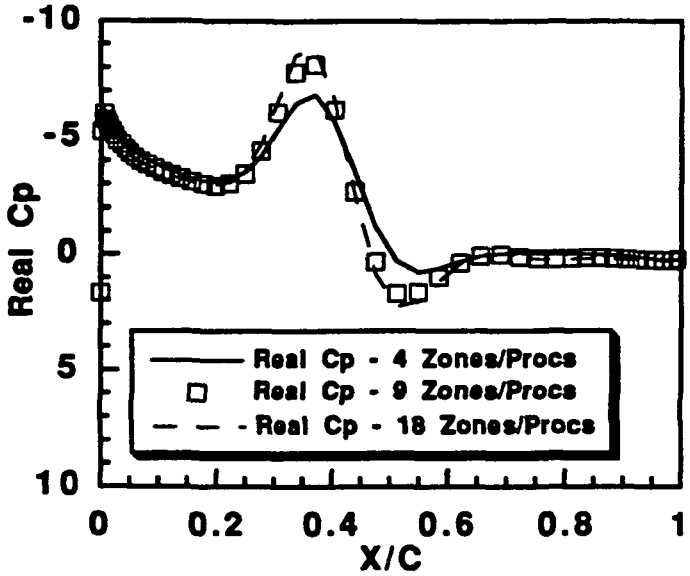


(a) Upper Surface Real Pressure Coefficient at 33% Span

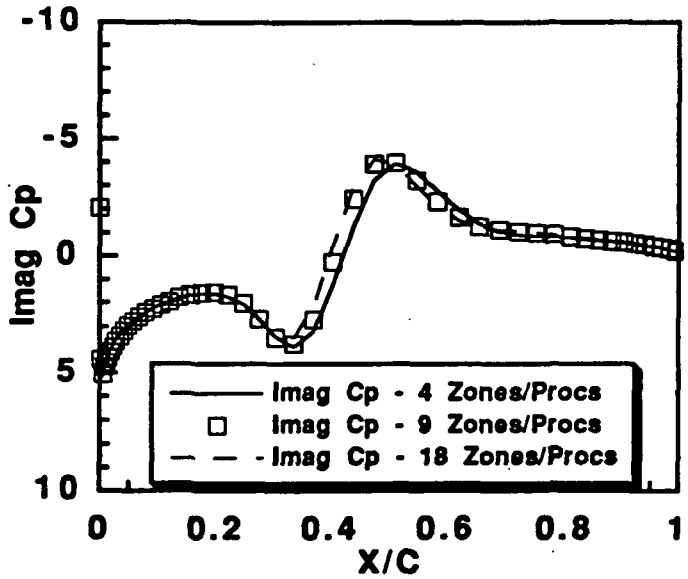


(b) Upper Surface Imaginary Pressure Coefficient at 33% Span

Figure 6.10: Comparison of 4, 9, and 18 Processor Real and Imaginary Pressure Distributions for the Inviscid F5 Wing Case, M=0.9, F=40Hz.



(c) Upper Surface Real Pressure Coeffcients at 88.5% Span



(d) Upper Surface Imaginary Pressure Coefficients at 88.5% Span

Figure 6.10: Comparison of 4, 9, and 18 Processor Real and Imaginary Pressure Distributions for the Inviscid F5 Wing Case (concluded)

spanwise grid planes in each block. In addition, the reduced number of spanwise stations in the transition region between the solid surface and the wake surface outboard of the tip slows the evolution of the flow field around the wing tip. However, these results indicate that acceptable accuracy can be maintained as the computational domain is decomposed into smaller blocks.

It was noticed during the comparisons of the 4, 9, and 18 zone cases that the speedup in turnaround performance using the Manager/Worker strategy deteriorated with increasing numbers of processors. The time required for the 18 zone case to go 4863 steps was 8984 seconds. This is only an improvement in performance of a factor of two over the 4 zone case. This result prompted the implementation of the Worker/Worker communications strategy. The turnaround performance of the Manager/Worker and Worker/Worker strategies for the three grid systems is given in Figure 6.11.

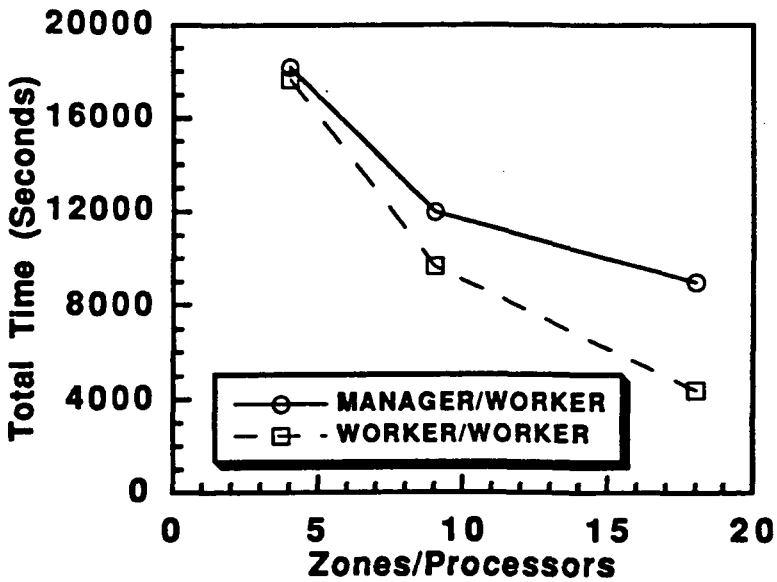


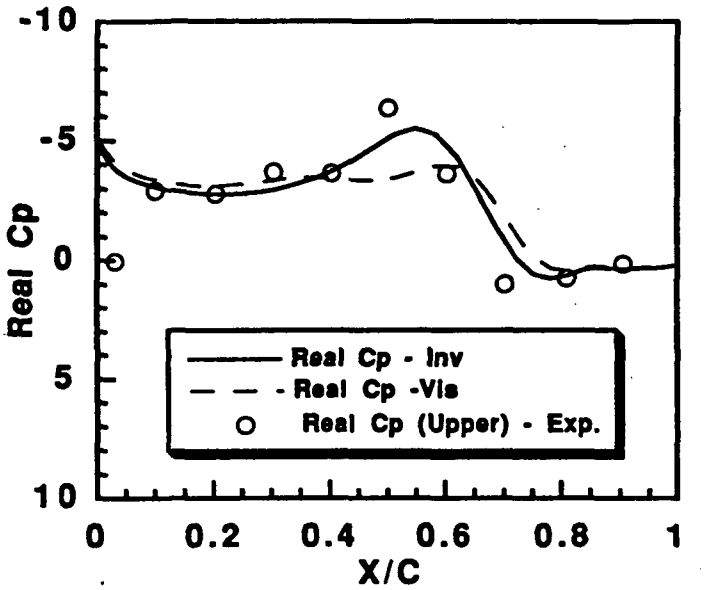
Figure 6.11: Comparison of Manager/Worker and Worker/Worker Performance

For a small number of processors, the Worker/Worker strategy is slightly faster than the Manager/Worker scheme. The turnaround time for the Worker/Worker approach for 4 processors is 17670 seconds. The difference in the performance of the two schemes is seen to increase substantially as the number of processors is increased. For the nine processor case, the Manager/Worker scheme required 11987 seconds for 4863 steps. In contrast, the Worker/Worker scheme required 9671 seconds. The turnaround time for the 18 processor Worker/Worker case is 4410 seconds. This is a relative speedup of a factor of 4 over the 4 processor case. The ideal speedup of the 18 zone case over the 4 zone case would be a factor of 4.5.

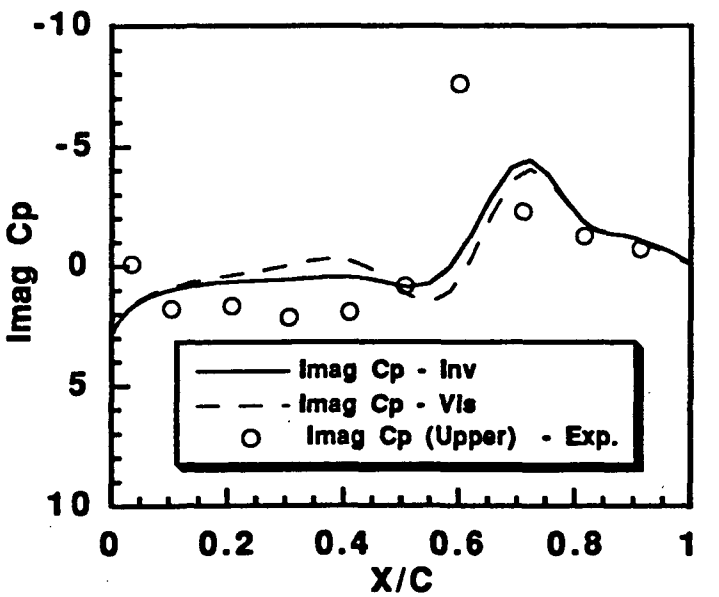
The deterioration of the Manager/Worker strategy is due to the wait time imposed on the Worker processes as the Manager collects and sends the required zonal boundary information to and from each Worker. Because of the equal load balance, all the processors are finishing at approximately the same time. Therefore, a bottleneck is formed as the Manager sends and receives the data to and from each processor. The Worker/Worker strategy removes this bottleneck by allowing the Workers to communicate directly with each other. The time spent waiting to receive data is effectively cut in half as the number of processors increases because the Workers do not have to wait on the Manager. These results demonstrate the superiority of the Worker/Worker scheme for large scale systems with high-speed communications subsystems.

6.3.3.2 Unsteady Viscous Simulations

A turbulent thin-layer Navier-Stokes simulation was performed using the 18 zone grid system and the Matrix Based Dissipation Model. The upper surface real and imaginary pressure coefficients are compared with the inviscid results and the experimental data at the 33% and 88.5% span stations are shown in Figure 6.12. The viscous case was run with a Reynolds number of 9 million. A free transition to turbulence was assumed. These results show little or no improvement due to the incorporation of viscous effects in the correlation

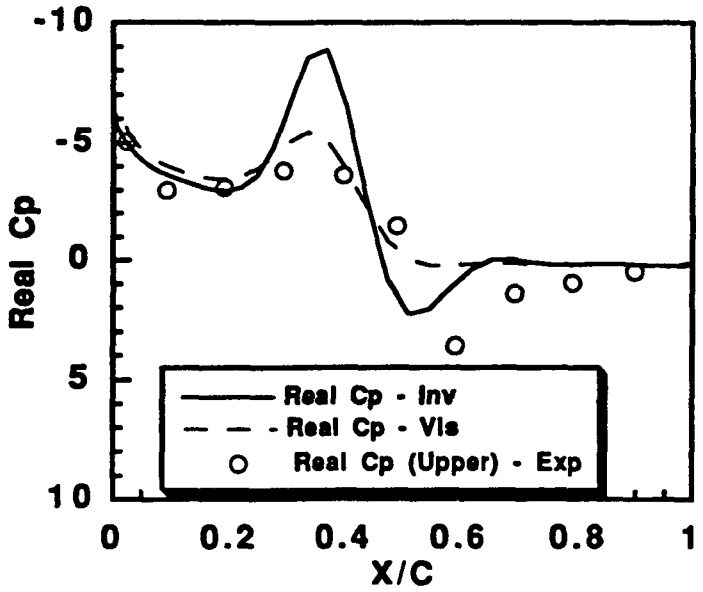


(a) Upper Surface Real Pressure Coefficient at 33% Span

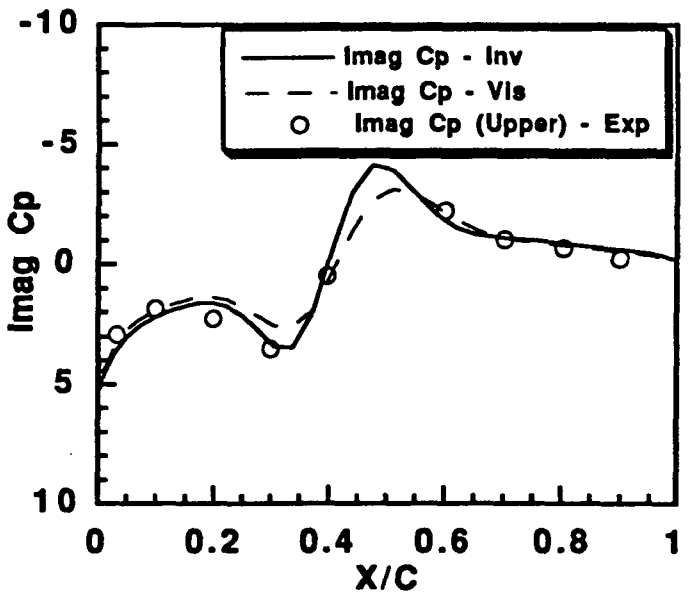


(b) Upper Surface Imaginary Pressure Coefficient at 33% Span

Figure 6.12: Comparison of Viscous and Inviscid Pressure Distributions for the F5 Wing Using Four Processors, M=0.9, F=40Hz.



(c) Upper Real Pressure Coefficient at 88.5% Span



(d) Upper Surface Imaginary Pressure Coefficient at 88.5% Span

Figure 6.12: Comparison of Viscous and Inviscid Pressure Distributions for the F5 Wing Using Four Processors, M=0.9, F=40Hz.(concluded)

with the experimental results. This indicates that a more refined computational grid is required for the viscous simulations. The transition locations should be fixed to the experimental values. These simulations were performed to determine if the inclusion viscous terms would yield a substantially different solution. The viscous case required about 5800 seconds for 4863 steps with the Worker/Worker strategy.

6.3.4 Steady Flow Results

A series of steady thin-layer Navier-Stokes analyses were performed with the unsteady implicit solver using two fixed time steps to determine the effect of the time step size on the solution convergence properties. The analyses were run for the Mach 0.896 case at a Reynolds number of 9 million using the 18 zone grid system. The non-dimensional time steps used were 0.006 and 0.067. The smaller time step was run first for 3600 steps. The large step size was run for 4000 steps. The average change in density with time and the L2(1) average norm of the residual of the continuity equation is given if Figures 6.13 and 6.14 for both time steps. The data in Figure 6.13 was computed with the CFL number set to 1.0. As would be expected, the solution appears to be converging faster for the larger time step size. However, the comparison of the convergence of the number of supersonic points given in Figure 6.15 shows that the large step achieves a steady state only a few hundred steps sooner than the smaller step size.

The computed steady pressure distributions computed with the 0.067 time step are compared with experimental results at the 33%, 53%, and 88.5% span stations in Figure 6.16. The computed lower surface pressure distributions are in excellent agreement with the experimental results. The mismatch in upper surface shock position on the outboard station can be attributed to the differences in the spanwise locations of the computational and the experimental data, the effects of the turbulence model, and an inadequate number of grid points in the normal direction. The time constraints on this research prevented a more

detailed viscous analysis. However, the results from these analyses indicated that the domain decomposition did not have an adverse affect on the steady state solution.

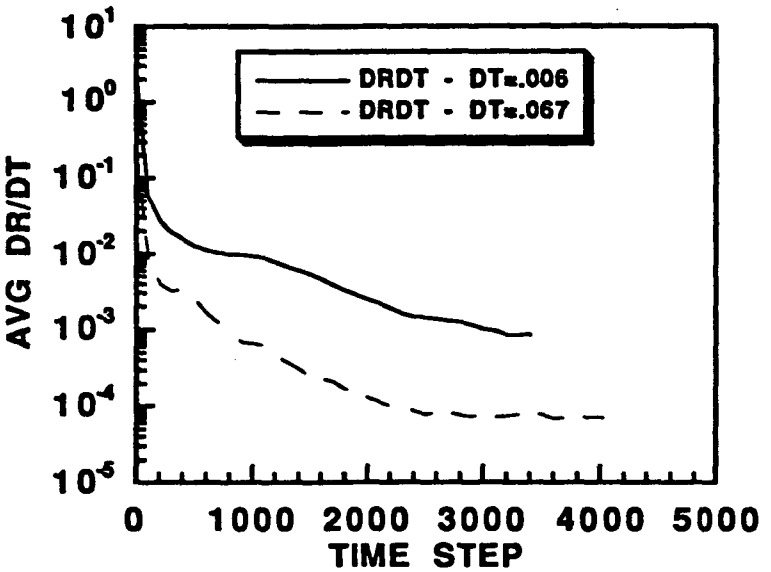


Figure 6.13: The Effect of Time Step Size on the Average Change in Density with Time For the Steady F5 Wing Case Using the Implicit Solver, M=0.9.

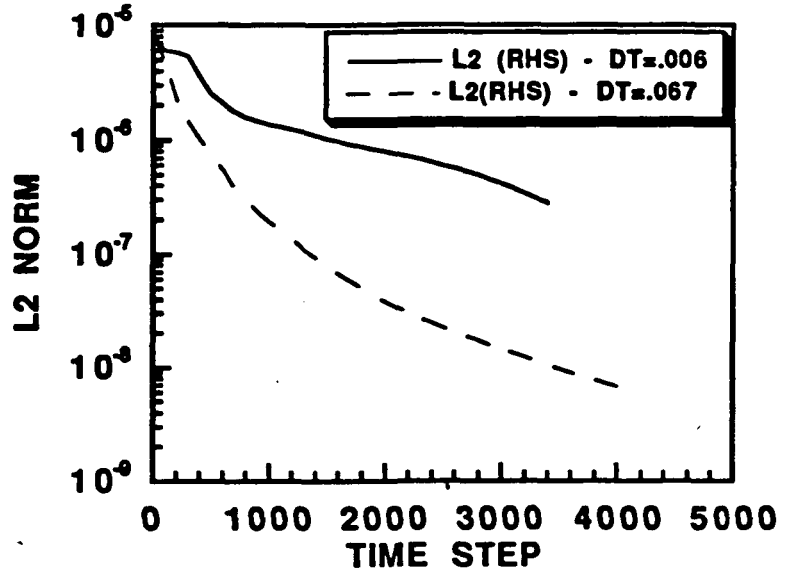


Figure 6.14: The Effect of Time Step Size on the L2 Norm of the Continuity Equation for the Steady F5 Wing Case Using the Implicit Solver, M=0.9.

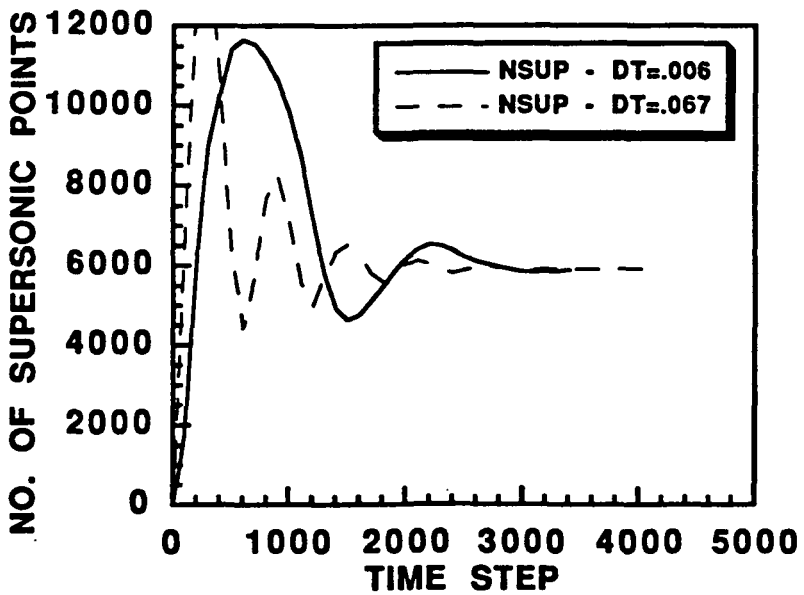
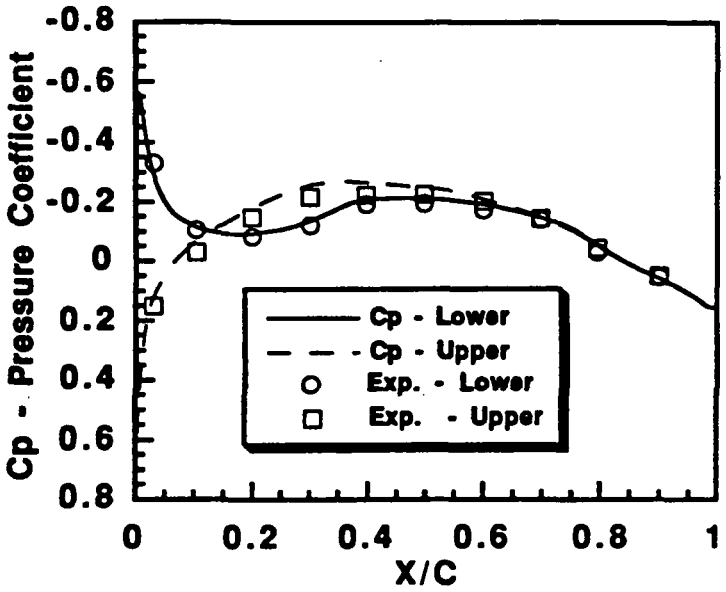
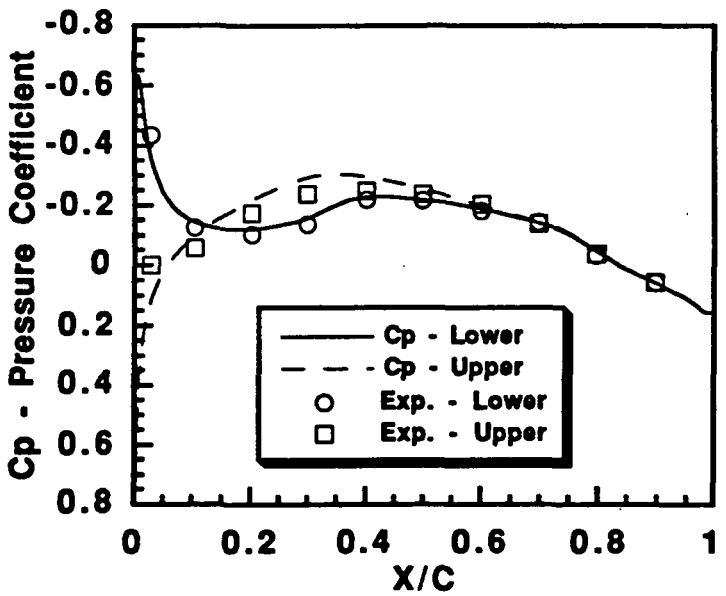


Figure 6.15: The Effect of Time Step Size on the Number of Supersonic Points for the Steady F5 Wing Case Using the Implicit Solver, M=0.9.

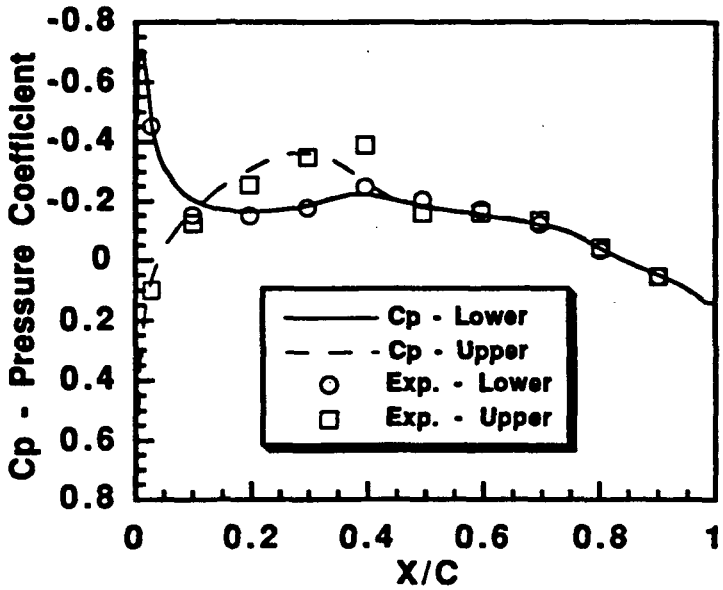


(a) 33% Span



(b) 53% Span

Figure 6.16: Steady Viscous Pressure Distributions for the F5 Wing, M=0.9



(c) 88.5% Span

Figure 6.16: Steady Viscous Pressure Distributions for the F5 Wing, M=0.9 (concluded)

CHAPTER VII

CONCLUSIONS AND RECOMMENDATIONS

In this research, explicit and implicit multi-zone three dimensional flow solvers were successfully implemented to perform as parallel distributed flow solvers on two different types of distributed computing systems using the PVM software interface. The first distributed systems used were composed of small Ethernet based networks of engineering workstations. A series of steady flow analyses on these systems validated the performance of the initial explicit flow solver and demonstrated the need for effective communications and load balancing strategies. Two different static load balancing approaches were evaluated and shown to provide adequate performance on lightly loaded networks. However, it was found that the performance of the network based solver was heavily dependent on system load. The Manager/Worker scheme was found to be an effective communications strategy for small network based systems. The utility of using the PVM software system to turn an existing multi-zone flow solver into a distributed solver was demonstrated. The results from the network based systems demonstrated that small scale Ethernet systems can be used to provide acceptable turnaround for moderate problems.

The second distributed system used in this research was the large scale IBM SP2 distributed system that is part of NASA's Numerical Aerodynamics Simulation facility. The performance of the original explicit solution algorithm and an implicit solver based on the LU-SGS solution scheme of Yoon and Jameson was compared for both steady and unsteady flow simulations. The steady flow simulations demonstrated the accuracy and

performance of the both distributed solvers for both inviscid and viscous simulations of the Lockheed/AFOSR Wing C geometry.

Steady and unsteady flow simulations were performed for the F5 wing with the implicit solver to investigate the effects of domain decomposition on solution accuracy. Unsteady simulations were performed with distributed systems of up to eighteen processors. The Worker/Worker communications strategy was successfully implemented to alleviate the problems of performance degradation encounter with the Manager/Worker scheme for increasing numbers of processors.

The results of the present research to develop efficient computational strategies for three dimensional flow simulations on distributed systems have led to the following conclusions and recommendations for future research.

7.1 Conclusions

The effectiveness of both networks of engineering workstations and large scale distributed systems based on workstation technology for real world Computational Fluid Dynamics simulations was demonstrated. The performance of distributed flow solvers is dependent on the implementation of efficient communications and load balancing strategies. It was found that Ethernet based systems can only be effective when the systems comprising the distributed system are very lightly loaded and network communications traffic is a minimum. A static load balancing procedure such as the Task Queue approach or a fully dynamic balancing scheme that can incorporate some of the effects of system load into the initial load balance is required for optimum performance on loaded systems. For lightly loaded systems, the Crutchfield static balancing algorithm proved to be an effective alternative to the Task Queue scheme.

The type and magnitude of numerical dissipation used in both the implicit and distributed solvers was found to be an important factor in alleviating the adverse effects on solution convergence that are introduced by the lag in the updating of zonal boundary conditions that is inherent in the distributed solvers. Levels of dissipation that were adequate for single processor solutions with the baseline explicit scheme were found to lead to either a stall convergence rate or divergence when used with the distributed solver. However, increasing the levels of fourth order dissipation was found to alleviate the problem. With proper levels of dissipation, the distributed solvers were found to provide the same accuracy as the baseline single processor solvers with greatly improved turnaround time performance.

The distributed implicit solver based on the LU-SGS scheme was found to be a viable alternative to the explicit solver for both inviscid and viscous simulations. For both Euler and Navier-Stokes simulations, the implicit scheme was found to be computationally more efficient on a time required per step basis than the explicit scheme. However, the time required to reach a steady state solution was about the same for the Euler simulations performed for the Wing C configuration.

The viability of an unsteady distributed solver based on the LU-SGS scheme was demonstrated by the unsteady F5 solutions. It was found that the computational domain could be subdivided into relatively small units without a serious degradation in solution accuracy. However, more research is required to quantify the effects of the lagged boundary update procedure on the time accuracy of larger scale problems. The results of the steady F5 analyses indicate that the effect of the domain decomposition on steady state solutions is also minimal.

The Manager/Worker strategy was found to be inadequate for systems such as the SP2 that utilize a switched high-speed communications subsystem. The Worker/Worker strategy was found to deliver superior performance with increasing numbers of processors when

there is an equal load balance. However, no communications strategy can overcome the idle time introduced by an unequal load balance. Therefore, effective procedures for domain decomposition either prior to or during the solution are required to maintain the scaling of turnaround performance that is expected with increasing numbers of processors.

Finally, it is felt that the computational approaches evaluated in this research provide a useful knowledge base for the development of new distributed solvers or the extension of existing multi-zone solvers for use on distributed systems.

7.2 Recommendations

The results and conclusions drawn from the present research have led to the following recommendations for extensions of the distributed flow solvers and for future areas of research:

1) A more detailed analysis needs to be made of the effects of the boundary update procedures on the time accuracy of the solution for viscous simulations. This would require the use of larger viscous grid systems than were used in the present research.

2) An alternate implicit scheme such as the diagonal scheme of Pulliam and Chausee should be compared with the LU-SGS scheme for both steady and unsteady simulations. In addition, the Newton iteration scheme should be implemented to increase the temporal order of accuracy of the implicit schemes and reduce the effect of the boundary update procedure on temporal accuracy. More work is required to quantify if the increased computational effort required by the Newton iteration scheme is justified.

3) The current Manager/Worker code structure should be unified into a single code to provide a Single Program Multiple Data (SPMD) version of the code. This would save a processor on systems with a limited number of available processors.

4) A version of the distributed solver based on the Message Passing Interface (MPI) should be implemented and its performance compared with the PVM versions of the solver.

5) A preprocessor should be developed that can decompose an existing unbalanced multi-zone grid system into a more evenly balanced system. An alternate approach would be to implement a dynamic load balancing algorithm that could generate evenly balanced subdomains from the initial grid system.

Finally, it is hoped that the worked performed in the present research can serve as the basis for the development of a production CFD code that can be a useful tool for aerodynamic analyses and design.

APPENDIX A

THE BALDWIN-LOMAX TURBULENCE MODEL

The Baldwin-Lomax turbulence model is a two-layer algebraic model based on a similar model proposed by Cebeci and Smith [52]. Both models use the Bossinesq approximation to compute a turbulent eddy viscosity, μ_T, that is combined with the molecular viscosity to form an effective value used in the calculation of the viscous terms in the Navier-Stokes equations. For wall bounded flows, the viscous region normal to the wall is assumed to be composed of inner and outer layers with different length scales. Unlike the Cebeci-Smith model which requires an explicit definition of the location of the edge of the boundary layer, the Baldwin-Lomax model defines uses the distribution of vorticity normal to the wall to define the length scales and corresponding eddy viscosities in the inner and outer layers. The value of the eddy viscosity is switched from the inner layer value to the outer layer value at the minimum normal distance from the wall where the inner and outer layer values of eddy viscosity are equal, This point is called the crossover point, $y_{crossover}$.

In the inner layer, the eddy viscosity is defined by the Prandtl-Van Driest formulation

$$\mu_T|_{inner} = \rho \ell^2 |\omega| \tag{A.1}$$

where the length scale, ℓ, is given by

$$\ell = \kappa y\left[1 - e^{\left(-y^+/A^+\right)}\right] \tag{A.2}$$

In Equation (A.2), κ is the von Karman constant which has a value of 0.4 and y is the normal distance from the wall. The term enclosed in brackets is the Van Driest damping factor that goes to zero at the wall. The parameter, y^+, is a non-dimensional length that is used in the law of the wall [48] to define the extent of the viscous sublayer where the molecular viscosity dominates the flow. y^+ is obtained from the relation

$$y^+ = \frac{\sqrt{\rho_w \tau_w}\, y}{\mu_w} \tag{A.3}$$

where ρ_w, τ_w, and μ_w are the density, shear stress and molecular viscosity at the wall. The parameter, A^+, is a constant whose value is normally set to 26 which provides the correct log-law profile near the wall for flow over a flat plate with zero pressure gradient [48] and is consistent with Cebeci's [141] formulations for a constant pressure boundary layer at transonic speeds. The magnitude of the vorticity is defined as

$$|\omega| = \sqrt{\left(\frac{\partial u}{\partial y} - \frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial z} - \frac{\partial w}{\partial y}\right)^2 + \left(\frac{\partial w}{\partial x} - \frac{\partial u}{\partial z}\right)^2} \tag{A.4}$$

In the outer layer, the eddy viscosity is defined as

$$\mu_T|_{outer} = C_{Claus} C_{cp} \rho F_{wake} F_{kleb}(y) \tag{A.5}$$

where C_{claus} is the Clauser constant whose value is 0.0168 and C_{cp} is an empirically derived constant with a standard value of 1.6. F_{wake} is given by

$$F_{wake} = \max\left(y_{max} F_{max} , 0.25 y_{max} U_{diff}^2 / F_{max}\right) \tag{A.6}$$

Values for the parameter F_{max} are obtained from the maximum value of the function

$$F(y) = y|\omega|\left[1 - e^{\left(-y^+ / A^+\right)}\right] \tag{A.7}$$

evaluated normal to the wall. y_{max} is the y location at which F_{max} occurs. The quantity, U_{diff}, is defined by the difference in the magnitudes of the minimum and maximum velocities in the boundary layer profile at a fixed station

$$U_{diff} = \left(\sqrt{u^2 + v^2 + w^2}\right)_{max} - \left(\sqrt{u^2 + v^2 + w^2}\right)_{min} \tag{A.8}$$

For wall bounded flows, the minimum velocity will occur at the solid surface and is zero for steady flows because of the no-slip condition. The function, F_{kleb}, is the Klebanoff intemittency correction that accounts for the intermittent changes in the viscosity at the edge of the boundary layer and is given by the relation

$$F_{kleb}(y) = \left[1 + 5.5\left(\frac{0.3y}{y_{max}}\right)^6\right]^{-1} \qquad \text{(A.9)}$$

In the free wake region behind a body such as a wing or an airfoil, the Baldwin-Lomax model is modified by setting the exponential term in the function F(y) defined by Equation (A.7) to zero. In the wake, the no-slip condition at the solid wall no longer in effect. Therefore, the minimum velocity used in Equation (A.8) must be specified.

Once the inner and outer values of the eddy viscosity are computed, the final values are set using the conditions

$$\mu_T = \begin{cases} \mu_T|_{inner} & \text{for } y \leq y_{crossover} \\ \mu_T|_{outer} & \text{for } y > y_{crossover} \end{cases} \qquad \text{(A.10)}$$

APPENDIX B

THE INVISCID FLUX JACOBIAN MATRICES

The inviscid flux Jacobian matrices A, B, and C used in the implicit schemes can be obtained by expanding the differential of the appropriate flux vector in a given coordinate direction and using the relation

$$dF = \frac{\partial F}{\partial Q} dQ = A dQ \tag{B.1}$$

Along any curvilinear coordinate surface ξ, η, or ζ, the flux Jacobian can be defined by a generic matrix of the form

$$\hat{A} = \begin{bmatrix} k_t & k_x & k_y & k_z & 0 \\ \phi k_x - uU_n & \overline{U} - \beta_2 u k_x & u k_y - \beta v k_x & u k_z - \beta w k_x & \beta k_x \\ \phi k_y - vU_n & v k_x - \beta u k_y & \overline{U} - \beta_2 v k_y & v k_z - \beta w k_y & \beta k_y \\ \phi k_z - wU_n & w k_x - \beta u k_z & w k_y - \beta v k_z & \overline{U} - \beta_2 w k_z & \beta k_z \\ (\phi - H)U_n & H k_x - \beta u U_n & H k_y - \beta v U_n & H k_z - \beta w U_n & \overline{U} + \beta U_n \end{bmatrix} \tag{B.2}$$

where k is one of the general coordinates ξ, η, or ζ. The derivatives k_x, k_y, and k_z can be defined in terms of normalized metric quantities that make Equation (B.2) applicable for

both finite difference and finite volume formulations. For instance, for k=ξ, k_x can be written as

$$k_x = \frac{\xi_x}{\left(\xi_x^2 + \xi_y^2 + \xi_z^2\right)^{1/2}} = \frac{S_x}{\left(S_x^2 + S_y^2 + S_z^2\right)^{1/2}} \tag{B.3}$$

where S_x, S_y, and S_z are the Cartesian components of the area vector normal to surface k. The parameters u, v, and w are the Cartesian components of velocity. H is the total enthalpy per unit volume. The variables ϕ, β, and β_2 are defined as

$$\beta = \gamma - 1 \,,\; \beta_2 = \gamma - 2 \,,\; \phi = \frac{\beta}{2}\left(u^2 + v^2 + w^2\right) \tag{B.4}$$

where γ is the ratio of specific heats. The parameters k_t and U_n are the grid and flow velocities normal to surface k and $\overline{U}$ is the contravariant velocity. These parameters are defined by the relations

$$k_t = -\left(x_t k_x + y_t k_y + z_t k_z\right) ,\; U_n = uk_x + vk_y + wk_z \,,\; \overline{U} = U_n + k_t \tag{B.5}$$

where x_t , y_t and z_t are the Cartesian grid speeds.

APPENDIX C

MANAGER AND WORKER PSEUDO CODES

The functions of the routines that make up the principle communications and control routines of the Manager and Worker distributed codes are described in the following pseudo code. The appropriate PVM functions for sending and receiving data messages are described in Reference [9]. The bulk of the calls to the wrapper routines that provide the interface between the baseline codes and the PVM library routines are embedded in routines SHELL, TEAM, and SOLVER in both the Manager and Worker Codes. The functions of each routine are described separately in the sequence they are called in the codes.

C.1 Manager Code

Main Program SHELL:

1. Enroll in the PVM system
2. Read grid, flow, and boundary data files to compute dynamic memory parameters
3. Allocate required memory
4. Call Subroutine TEAM
5. Exit SHELL

Subroutine TEAM:

1. Read flow, grid, and boundary files for initial case and solution file for restart cases.
2. Input the number desired number of Worker processes and processor names
3. Spawn Worker processes.
4. Compute Load Balance and grid-to-processor maps
5. Send grids, boundary data, control data, etc. to each processor.
6. Call Subroutine SOLVER
7. If solution converged or max. steps reached then

   Stop worker processes

   End if
8. Save solution if flagged to do so
9. Exit TEAM

Subroutine SOLVER:

1. Initialize various flags and initial values
2. For N<=max no. of Steps

   Send start of step flag to workers

   Update step index

   If Manager/Worker strategy Then

   Receive updated boundary data for each zone from Workers

   Send appropriate boundary data to individual Workers

   End If
3. Receive Convergence data from each Worker
4. If Convergence less than or equal to tolerance or N=Max. steps Then

   Receive updated solution from Workers

Receive Aero loads from Workers and compute total loads

Exit For loop (Go to 6)

Endif

5. End For
6. Exit SOLVER

C.2 WORKER CODE

Main Program SHELL:

1. Enroll in PVM
2. Receive Memory initialization parameters from Manager
3. Allocate required memory
4. Call Subroutine TEAM
5. Exit SHELL

Subroutine TEAM:

1. Receive Case parameters from Manager
2. Receive Grids from Manager
3. If not a restart case Then

   Call INIT to generate initial solution

   Else

   Get initial solution from Manager

   End if
4. Get Boundary Condition and other initialization data from Manager

5. Call Subroutine SOLVER
6. Exit TEAM

Subroutine SOLVER:

1. Compute initial metrics and cell volumes
2. Do until stop flag received from Manager
3. If NSTEP = NTIM then

   Compute new time step, spectral radii, etc.

   Update Boundary arrays

   End if
4. If Manager/Worker Then

   Send Manager updated Boundary Array data for grids on processor

   Receive required updated Boundary Array data from Manager

   Else

   Send local Boundary Array data to appropriate Worker processors

   Receive required updated Boundary Array data from other Worker processors

   End if
5. Call MENSA to compute advance solution to next step

   MENSA updates local Boundary Array and computes convergence data
6. Send Convergence data to Manager
7. End do
8. Exit PVM system
9. Exit SOLVER

REFERENCES

1. Melnik, R. , Barber, T., and Verhoff, A., "A Process for Industry Certification of Physical Simulation Codes," AIAA Paper 94-2235, June 1994.

2. Marconi, F., Siclari, M., Chow, R., and Carpenter, G., "Comparison of TLNS3D Computations with Test Data for a Transport Wing/Simple Body Configurations," AIAA Paper 94-2238, June 1994.

3. Raj, P. and Siclari, M., "Toward Certifying CFD Codes Using Wing C and M100 Wing-Body Configurations," AIAA Paper 94-2241, June 1994.

4. Ragsdale, S., ed., *Parallel Programming*, McGraw-Hill Inc., 1991, pp. 3-13.

5. Johnson, G. M., "Parallel Processing in Fluid Dynamics," ISC Report 87003, Institute for Scientific Computing, 1987.

6. Gropp, W. D. and Smith, E. B., " Computational Fluid Dynamics on Parallel Processors," AIAA Paper 88-3793-CP, 1988.

7. Saini, S., "The IBM SP2: Hardware, Software, Porting, and Optimization Overview," NAS Users Seminar, NASA Ames Research Center, July 1994.

8. NAS Distributed Computing Team, "Clustered Workstations and Their Potential Roles As High Speed Processors," NAS Report RNS-94-003, NASA Ames Research Center, April 1994.

9. Geist, G. A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., and Sunderam, V., "PVM 3 Users Guide and Reference Manual," ORNL/TM-12187, September 1994.

10. Dongarra, J., Geist, G., Manchek, R., and Sunderam, V., "Integrated PVM Framework Supports Heterogeneous Network Computing," *Computers in Physics*, Vol. 7. No. 2, March/April 1993.

11. Geist, G.A. and Sunderam, V.S., "Network Based Concurrent Computing on the PVM System," ORNL/TM-11760, June 1991.

12. Beguelin, A., Dongarra, J.S, Geist, G.A., Manchek, R., and Sunderam, V.S., "A Users Guide to PVM (Parallel Virtual Machine)," ORNL/TM-11826, July 1991.

13. Smith, M.H. and Pallas, J.M., "MEDUSA, An Overset Grid Flow Solver for Network-Based Parallel Computer Systems," AIAA Paper 93-3312-CP, July 1993.

14. Swisshelm, J.M., Johnson, G.M., and Kumar, S.P., "Parallel Computations of Euler and Navier-Stokes Flows," Proceedings of the Second Copper Mountain Conference on Multigrid Methods," Copper Mountain, Co., March 31-April 13, 1985.

15. Patel, N. R. Sturek, W. B., and Jordan, H.T., "A Parallelized Solution for Incompressible Flow on a Multiprocessor," AIAA Paper 85-1511, 1985.

16. Wake, B.E., and Eglof, T.A., "Implementation of a Rotary-Wing Three-Dimensional Navier-Stokes Solver on a Massively Parallel Computer," AIAA Paper 89-1939-CP, June 1989.

17. Wake, B.E. and Eglof, T.A., "Application of a Rotary-Wing Viscous Flow Solver on a Massively Parallel Computer," AIAA Paper 90-0334, January 1990.

18. Long, L.N., Khan, M.M., and Sharp, H.T., "Massively Parallel Three-Dimensional Euler/Navier-Stokes Method," *AIAA Journal*, Vol. 29, No. 5., May 1991, pp. 657-666.

19. Agarwal, R., "Development of a Navier-Stokes Code on a Connection Machine," AIAA Paper 89-1938, June 1989.

20. Kallinderis, Y. and Vidwans, A., "Generic Adaptive Grid Navier-Stokes Algorithm," *AIAA Journal*, Vol. 32, No. 1, January 1994, pp. 54-61.

21. Hammond, S. and Barth, T.J., "Efficient Massively Parallel Euler Solver for Two-Dimensional Unstructured Grids," *AIAA Journal*, Vol. 30., No. 4., April 1992, pp. 947-952.

22. Morano, E., and Mavriplis, D., "Implementation of a Parallel Unstructured Euler Solver on the CM-5," AIAA Paper 94-0755, January 1994.

23. Ryan, J.S. and Weeratunga, S. "Parallel Computations of Three-Dimensional Navier-Stokes Flowfields for Supersonic Vehicles," AIAA Paper 93-0064, January 1993.

24. Hixon, D. and Sankar, L.N., "Unsteady Compressible 2-D Flow Calculations on a MIMD Parallel Supercomputer," AIAA Paper 94-077, January 1994.

25. Otto, J., "Parallel Execution of a Three-Dimensional Chemically Reacting Navier-Stokes Code on Distributed Memory Machines," AIAA Paper 93-3307, July 1993.

26. Das, R., Mavriplis, D., Salez, J., Gupta, S., and Ponnusamy, R., "The Design and Implementation of a Parallel Unstructured Euler Solver Using Software Primitives," AIAA Paper 93-0562, January 1992.

27. Venkatakrishnan, V., Simon, H.D., and Barth, T.J., "A MIMD Implementation of a Parallel Solver for Unstructured Grids," *Journal of Supercomputing*, Vol. 6, 1192, pp. 117-137.

28. Venkatakrishnan, V., "Implicit Unstructured Grid Solvers on the iPSC/860," AIAA Paper 94-0759, January 1994.

29. Promano, E., and Weeratunga, S., "Aeroelastic Computations for Wings Through Direct Coupling on Distributed Memory MIMD Parallel Computers," AIAA Paper 94-0095, January 1994.

30. Byun, C. and Guruswamy, G. P., Wing/Body Aeroelasticity Using Finite Difference Fluid/Finite Element Structural Equations on Parallel Computers," AIAA Paper 94-1487, 1994.

31. Imlay, S. and Soetrisno, M., "3-D Navier-Stokes Flow Analysis for Shared and Distributed Memory MIMD Computers," Report P90120.04, AMTEC Engineering, 1991.

32. Fatoohi, R., "Adapting a Navier-Strokes Algorithm for Three Parallel Machines," *Journal of Supercomputing*, Vol. 8, No. 2, June 1994, pp. 91-115.

33. Proceedings of the NASA Workshop on Distributed Computing for Aerosciences Applications, NASA Ames Research Center, October 1993.

34. Hayden, M., Jayasimha, D., and Pillay, S., "Parallel Navier-Stokes Computations on Shared and Distributed Memory Architectures," AIAA Paper 95-0577, January 1995.

35. Deshpande, M., Feng, J., Merkle, C., and Deshpande, A., "Implementation of a Parallel Algorithm on A Distributed Network," AIAA Paper 93-0058, January 1993.

36. Bangalore, A., Latham, R., and Sankar, L.N., "Numerical Simulation of Viscous Flow Over Rotors Using a Distributed Computing Strategy," AIAA Paper 95-0575, January 1995.

37. Bangalore, A., " Computational Fluid Dynamics Studies of High Lift Rotor Systems Using Distributed Computing," Ph. D. Dissertation, School of Aerospace Engineering, Georgia Institute of Technology, May 1995.

38. Weed, R.A. and Sankar, L.N., "Computational Strategies for Three-Dimensional Flow Simulations on Distributed Computer Systems," AIAA Paper 94-2261, June 1994.

39. Weed, R.A. and Sankar, L.N., "Computation Strategies for Three-Dimensional Unsteady Flow Simulations on Distributed Computer Systems," Presented at the Computational Aerosciences Workshop '95, NASA Ames Research Center, March 7-9 1995.

40. Raj, P., Olling, C., Sikora, J., Keen, J., Singer, S., and Breenan, J., "Three-Dimensional Euler/Navier Stokes Aerodynamic Method, Volumes 1-3," AFWAL-TR-87-3074, June 1989.

41. Holst, T., Gundy, K., Flores, J., Chanderjian, N, Kuyak, U., and Thomas, S., "Numerical Solution of Transonic Wing Flows Using an Euler/Navier Stokes Zonal Approach," AIAA Paper 85-1640, July 1985.

42 Vatsa, V.N., Sanetrik, M., Parlette, E., Eiseman, P., and Cheng, Z., "Multi-block Structured Grid Approach for Solving Flows Over Complex Aerodynamic Configurations," AIAA Paper 94-0655, January 1994.

43. Klopfer, G. and Yoon, S., "Multizonal Navier-Stokes Code with the LU-SGS Scheme," AIAA 93-2965, July 1993.

44. Jameson, A., Schmidt, W., and Turkel, E., "Numerical Solution of the Euler Equations for Finite Volume Methods Using Runga-Kutta Time Stepping," AIAA Paper 81-1259, June 1981.

45. Thompson, P.A., *Compressible Fluid Dynamics*, McGraw-Hill Inc., 1972, pp. 15-42.

46. Vinokur, M., "An Analysis of Finite Difference and Finite Volume Formulations of Conservation Laws," *Journal of Computational Physics*, Vol. 81, No. 1, March 1989, pp. 1-52.

47. Gnoffo, P., Gupta, R., and Shinn, J., "Conservation Equations and Physical Models for Hypersonic Air Flows in Thermal and Chemical Non-equilibrium," NASA TP 2867, 1989.

48. White, F.M., *Viscous Fluid Flow, 2nd Ed.*, McGraw-Hill Inc., 1991.

49. Worsoe-Schmidt, P. and Leppert, G., "Heat Transfer and Friction for Laminar Flow of Gas in a Circular Tube at High Heating Rate," *Inter. Journal of Heat and Mass Transfer*, Vol. 8, 1965, p. 1281.

50. Tennekes, H. and Lumley, J.L., *A First Course In Turbulence*, The MIT Press, 1972.

51. Anderson, D., Tannehill, J., and Pletcher, R., *Computational Fluid Mechanics and Heat Transfer*, Hemisphere Publishing Corp., 1984.

52. Cebeci, T. and Smith, A.M.O., *Analysis of Turbulent Boundary Layers*, Academic Press, 1974.

53. Baldwin, B. and Lomax, H., " Thin Layer Approximations and Algebraic Model for Separated Flows," AIAA Paper 78-257, 1978.

54. Johnson, D.A. and King, L., "A Mathematical Simple Turbulence Closure Model for Attached and Separated Turbulent Boundary Layers," *AIAA Journal*, Vol. 23, No. 11, November, 1985, pp. 1684-1692.

55. Spalart, P. and Allmaras, S., "A One Equation Turbulence Model for Aerodynamic Flows," AIAA Paper 92-0439, January 1992.

56. Jones, W.P and Launder, B.E., "The Prediction of Laminarization With a Two-Equation Model of Turbulence," *Inter. Journal of Heat and Mass Transfer*, Vol. 15, 1972.

57. Radespiel, R. and Swanson, R., " An Investigation of Cell Centered and Cell Vertex Multigrid Schemes for the Navier Stokes Equations," AIAA Paper 89-0548, January 1989.

58. Jameson, A., Baker, T.J., and Weatherill, N. P., "Calculation of Inviscid Transonic Flow Over a Complete Aircraft," AIAA Paper 86-0103, January 1986.

59. Kordulla, W. and Vinokur, M., "Efficient Computations of Volume in Flow Predictions," *AIAA Journal*, Vol. 21, No. 6, June 1983, pp. 917-918.

60. van Leer, B., "Towards the Ultimate Conservative Difference Scheme V: A Second Order Sequel to Gudonov's Method," *Journal of Computational Physics*, Vol. 32., 1979, pp. 101-136.

61. Roe, P.L., "Characteristic Based Schemes for the Euler Equations," *Annual Review of Fluid Mechanics*, Vol. 18, 1986, pp. 337-365.

62. Jameson, A., "A Nonoscillatory Shock Capturing Scheme Using Flux-Limited Dissipation," MAE Report 1653, Princeton University, 1984.

63. Swanson, J.C. and Turkel, E., "On Central Difference and Upwind Schemes," *Journal of Computational Physics*, Vol. 101, 1992, pp. 292-306.

64. van Leer, B., Thomas, J., Roe, P, and Newsome, R., "A Comparison of Numerical Flux Formulas for the Eulert and Navier-Stokes Equations," AIAA Paper 87-1104, 1987.

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

66. Pulliam, T.H., and Chausee, D.S., "A Diagonal Form of an Implicit Approximate Factorization Algorithm," *Journal of Computational Physics*, Vol. 39, No. 2, Feb. 1981, pp. 347-363.

67. Turkel, E., and Vatsa, V.N., "Effect of Artificial Viscosity on Three-Dimensional Solutions," *AIAA Journal*, Vol. 32, No. 1, January 1994, pp. 39-45.

68. Jorgenson, P. and Turkel, E., "Central Difference TVD and TVB Schemes for Time Dependent and Steady State Problems," AIAA Paper 92-0053, January 1992.

69. Caughy, D. A., "Implicit Euler Solutions With Symmetric Total-Variation-Diminishing Dissipation," AIAA Paper 93-3356-CP, June 1993.

70. Chen, C.L. and McCroskey, W.J., "Numerical Simulation of Helicopter Multi-Bladed Rotor Flow," AIAA Paper 88-0046, January 1988.

71. Allmaras, S.R.," Embedded Mesh Solutions of the 2-D Euler Equations Using a Cell Centered Finite Volume Scheme," CFDL-TR-85-4, Massachusetts Institute of Technology, August 1985.

72. Jameson, A., and Baker, T.J., "Solution of the Euler Equations for Complex Configurations," AIAA Paper 93-1929, July 1983.

73. Hirsch, C. *Numerical Computation of Internal and External Flows, Volume 2. Computational Methods for Inviscid and Viscous Flows*, John Wiley and Sons, 1990.

74. Martinelli, L., "Calculation of Viscous Flows With Multigrid Methods," Ph. D. Dissertation, MAE Dept., Princeton University, 1987.

75. Vatsa, V.N., Turkel, E., and Abolhassani, J.S., "Extension of Multigrid Methodology to Supersonic/Hypersonic 3-D Viscous Flows," NASA CR 187612, 1991.

76. Douglas, J. and Gunn, J.E., "A General Formulation of Alternating Direction Methods," *Numer. Math.*, Vol. 6, 1964, pp. 428-453.

77. MacCormack, R. W., "Current Status of Numerical Solutions of the Navier-Stokes Equations," AIAA Paper 85-0032, January 1985.

78. Thomas, J.L. and Walters, R.W., "Upwind Relaxation Algorithms for the Navier-Stokes Equations," *AIAA Journal*, Vol. 25, No. 4, April 1987, pp. 527-534.

79. Briley, W.R. and McDonald, H., "Solution of the Multidimensional Compressible Navier-Stokes Equations By a Generalized Implicit Method," *Journal of Computational Physics*, Vol. 24, No. 4, August 1977.

80. Beam, R. and Warming, R.E., "An Implicit Factored Scheme for the Compressible Navier-Stokes Equations," *AIAA Journal*, Vol. 16, No. 4, April 1978, pp. 393-402.

81. Pulliam, T.H. and Steger, J.L., "Implicit Finite Difference Simulations of Three-Dimensional Compressible Flows," *AIAA Journal*, Vol. 18. No. 2, February 1980., pp. 159-167.

82. Steger, J.C. and Warming, R.F., "Flux Vector Splitting of the Inviscid Gasdynamics Equations with Applications to Finite Difference Methods," *Journal of Computational Physics*, Vol. 40, No. 2, 1981, pp. 263-393.

83. Sankar, L.N., Wake, B.E., and Lekoudis, S.G.," Solution of the Unsteady Euler Equations for Fixed and Rotor Wing Configurations," *Journal of Aircraft*, Vol. 23, No. 4, April 1986, pp. 283-289.

84. Wake, B.E. and Sankar, L.N., "Solution of Navier-Stokes Equations for the Flow Over A Rotor Blade," *Journal of the American Helicopter Society*, Vol. 34, No. 2, April 1989, pp. 13-23.

85. Ruo, S.Y. and Sankar, L.N., "Euler Calculations for Wing-Alone Configurations," *Journal of Aircraft*, Vol. 25, No. 5, May 1988, pp. 436-441.

86. Jameson, A. and Turkel, E., "Implicit Schemes and *LU* Decompositions,", *Math. of Computation*, Vol. 37, No. 156, October 1981, pp. 385-397.

87. Buning, P.G. and Steger, J.L., "Solution of the Two-Dimensional Euler Equations with Generalized Coordinate Transformation Using Flux Vector Splitting," AIAA Paper 82-0971, January 1982.

88. Yoon, S. and Jameson, A., "Lower-Upper Symmetric-Gauss-Seidel Method for the Euler and Navier-Stokes Equations," *AIAA Journal*, Vol. 26, No. 9, September 1988, pp. 1025-1026.

89. Ying, S.X., Steger, J.L, Shiff, L.B., and Boganoff, D., "Numerical Simulation of Unsteady Viscous High Angle of Attack Flows Using a Partially Flux Split Algorithm," AIAA Paper 86-2179, August 1986.

90. Edwards, T.A. and Flores, J., "Toward A CFD Nose-to-Tail Capability. Hypersonic Unsteady Navier-Stokes Code Validation," AIAA Paper 89-1672, June 1989.

91. Jameson, A. and Yoon, S., "Lower-Upper Implicit Schemes with Multiple Grids for the Euler Equations," *AIAA Journal*, Vol. 25, No. 7, July 1987, pp. 929-935.

92. Whitfield, D.L., "Implicit Upwind Finite Volume Scheme for the Three-Dimensional Euler Equations," MSSU-EIRS-ASE-85-1, Mississippi State University, September 1985.

93. Buratynski, E.K. and Caughy, D.A., "An Implicit LU Scheme for the Euler Equations Applied to Arbitrary Cascades," AIAA Paper 84-0167, January 1984.

94. Yokota, J.W. and Caughy, D.A., "An LU Implicit Multigrid Algorithm for the Three-Dimensional Euler Equations," AIAA Paper 87-0453, January 1987.

95. Shuen, J.S. and Yoon, S., "Numerical Study of Chemically Reacting Flows Using a Lower-Upper Symmetric Successive Overrelaxation Scheme," *AIAA Journal* , Vol. 27, No. 12, December 1989, pp. 1752-1760.

96. Imlay, S.T. and Eberhardt, S., "Non-equilibrium Thermo-chemical Calculations Using a Diagonal Implicit Scheme," AIAA Paper 91-0468, January 1991.

97. Park, C. and Yoon, S., "Calculation of Real Gas Effects on Blunt Body Trim Angles," *AIAA Journal*, Vol. 30, No. 4, April 1992, pp. 999-1007.

98. Yoon, S. and Kwack, D., "Implicit Navier-Stokes Solver for Three-Dimensional Compressible Flows," *AIAA Journal*, Vol. 30, No. 11, November 1992, pp. 2653-2659.

99. Yoon, S. and Kwack, D., "Multigrid Convergence of an Implicit Symmetric Relaxation Scheme," *AIAA Journal*, Vol. 32, No. 5, May 1994, pp. 950-955.

100. Yoon, S. and Kwack, D., "Multigrid Convergence of an LU Scheme," *Frontiers of CFD 1994*, D.A. Caughey and M.M. Hafez, ed's, John Wiley and Sons, 1994, pp. 319-338.

101. Chen, C., McCroskey, W.J., and Obayashi, S., "Numerical Solutions of Forward-Flight Rotor Flows Using an Upwind Method," AIAA Paper 89-1846, June 1989.

102. Srinivasan, G.R., Baeder, J.D., Obayashi, S., and McCroskey, W.J., "Flowfield of A Lifting Rotor in Hover, A Navier-Stokes Simulation," *AIAA Journal*, Vol. 30, No. 10, October 1992, pp. 2371-2378.

103. Obayashi, S. and Guruswamy, G.P.," Convergence Acceleration of an Aeroelastic Navier-Stokes Solver," AIAA Paper 94-2268, June 1994.

104. Yoon, S. and Jameson, A., "An LU-SSOR Scheme for the Euler and Navier-Stokes Equations," AIAA Paper 87-0600, January 1987.

105. Pulliam, T.H. "Time Accuracy and the Use of Implicit Methods," AIAA Paper 93-3360-CP, July 1993.

106. Matsuno, K. " Higher-Order Time-Accurate Scheme for Unsteady, Three-Dimensional Flows," AIAA Paper 93-3362-CP, July 1993.

107. Lee, D. and Kim, S., "An Efficient Method to Calculate Rotor Flows in Hover and Forward Flight," AIAA Paper 93-3336-CP, July 1993.

108. Thomas, P.D. and Lombard, C.K., "Geometric Conservation Law And Its Application to Flow Calculations with Moving Grids," *AIAA Journal*, Vol. 17, No. 10, pp. 1030-1037.

109. Obayashi, S., "Freestream Capturing for Moving Coordinates in Three-Dimensions," *AIAA Journal*, Vol. 30, No. 4, April 1992, pp. 1125-1127.

110. Saphir, W., "Message Passing on the SP2,", NAS SP2 Users Training Seminar, NASA Ames Research Center, August, 1994.

111. Message Passing Interface Forum, "MPI: A Message Passing Interface Standard," Computer Science Dept. Technical Report CS-94-230, University of Tennessee, 1994.

112. Comer, D., *Internetworking with TCP/IP, Principles, Protocals, and Architectures, 2nd Ed.*, Prentice-Hall, Inc., 1988.

113. IBM Corporation, *IBM AIX PVMe Users Guide and Subroutine Reference, Release 3.0*, 1994.

114. Fatoohi, R., "Performance Evaluation of Communications Networks for Distributed Computing," NAS-95-009, NASA Ames Research Center, March 1995.

115. Cohen, A.M., *A Guide to Networking, 2nd Ed.*, Boyd and Fraser Publishing Co., 1995.

116. Smith, M.H., Private Communication, NASA Ames Research Center, August, 1994.

117. Atwood, C., "Toward Distributed Fluids/Controls Simulations," Presented at the Computational Aerosciences Workshop '95, NASA Ames Research Center, March 7-9 1995.

118. Vidwans, A., Kallinderis, Y., and Venkatakrishnan, V., "A Parallel Dynamic Load Balancing Algorithm for 3-D Adaptive Unstructured Grids," AIAA Paper 93-3313-CP, July 1993.

119. De Keyser, J., Lust, K., and Roose,D., "Run-time Load Balancing Support for A Parallel Multiblock Euler/Navier-Stokes Code with Adaptive Refinement on Distributed Memory Computers," *Parallel Computing*, Vol. 20, August 1994, pp. 1069-1088.

120. Johnson, J., "Distributed Parallel Processing in Computational Fluid Dynamics," Proceedings of NASA Workshop on Distributed Computing in Aerosciences Applications, NASA Ames Research Center, October 1993.

121. Crutchfield, W.Y., "Load Balancing Irregular Algorithms," UCRL-JC-107679, Lawrence Livermore National Laboratory, July 1991.

122. Smith, M.H., "Distributed Parallel Flow Solutions Using Overset Grids," Proceedings of NASA Workshop on Distributed Computing in Aerosciences Applications, NASA Ames Research Center, October 1993.

123. Goble, B.D., Raj, P., and Kinard, T.A., "Three-Dimensional Euler/Navier Stokes Aerodynamic Method (TEAM) Upgrade, Version 713 Users Manual," WL-TR-93-3115, February, 1994.

124. Lorenz-Meyer, W., and Aulehla, T., "MBB Body of Revolution, No. 3," AGARD AR-138, 1979, pp. C1-C31.

125. Noack, R.W. and Anderson, D.A., "Solution Adaptive Grid Generation Using Parabolic Partial Differential Equations," AIAA Paper 88-0315, 1988.

126. Schmitt, V. and Charpin, F., "Pressure Distributions on the ONERA M6 Wing at Transonic Mach Numbers," AGARD AR-138, 1970, pp. B1-B43.

127. Hinson, B.L. and Burdges, K.P., "Acquisition and Application of Transonic Wing and Far-Field Test Data for Three-Dimensional Computational Method Evaluation," AFOSR-TR-80-0421, March 1980.

128. Fieresen, W., "HPCCP Computational Aerosciences Overview," Presented at the NASA Computational Aerosciences Workshop '95, NASA Ames Research Center, March 1995.

129. Saphir, W., "SP2 Hardware and Software Overview," NAS SP2 User Training Seminar, NASA Ames Research Center, August 1994.

130. Tijdeman, H., van Nunen, J.W.G., Kraan, A.N., Persoon, A.J., Poestkoke, R., Roos, R., Schippers, P., and Siebert, C.M., "Transonic Wind Tunnel Tests on an Oscillating Wing with External Stores, Part I: General Description," AFFDL-TR-78-194, Part I, December 1978.

131. Tijdeman, H., van Nunen, J.W.G., Kraan, A.N., Persoon, A.J., Poestkoke, R., Roos, R., Schippers, P., and Siebert, C.M., "Transonic Wind Tunnel Tests on an Oscillating Wing with External Stores, Part II: Clean Wing," AFFDL-TR-78-194, Part II, March 1979.

132. Malone, J.B., Sankar, L.N., and Sotomayer, W.A., "Unsteady Aerodynamics Modeling of a Fighter Wing in Transonic Flow," *Journal of Aircraft*, Vol. 23, No. 8, August 1986, pp. 611-620.

133. Sotomayer, W.A., Sankar, L.N., and Malone, J.B., "A Comparison of Numerical Algorithms for Unsteady Transonic Flows," *Computer Methods in Applied Mechanics and Engineering*, Vol. 64, 1987, pp. 237-265.

134. Guruswamy, G.P. and Goorjian, P.M., "Efficient Algorithm for Unsteady Transonic Aerodynamics of Low Aspect Ratio Wings," *Journal of Aircraft*, Vol. 22, No. 3, March 1985, pp. 193-199.

135. Guruswamy, G.P., "Navier Stokes Computations on Swept-Tapered Wings, Including Flexibility," *Journal of Aircraft*, Vol. 29, No. 4, July-August 1992, pp. 588-597.

136. Obayashi, S., Guruswamy, G.P., and Goorjian, P.M., "Streamwise Upwind Algorithm for Computing Unsteady Transonic Flows Past Oscillating Wings," *AIAA Journal*, Vol. 29, No. 10, October 1991, pp. 1558-1677.

137. Guruswamy, G.P. and Obayashi, S., "Transonic Aeroelastic Computations on Wings Using Navier-Stokes Equations," AGARD CP-507, March 1992, pp. 22-1 - 22-22.

138. Guruswamy, G.P., "Unsteady Aerodynamic and Aeroelastic Calculations on Wings Using Euler Equations," *AIAA Journal*, Vol. 28, No. 3, March 1990, pp. 461-469.

139. Hixon, D.R., "Application of A Generalized Minimal Residual Method to the Calculation of 2D and 3D Unsteady Flows," Ph.D. Thesis, School of Aerospace Engineering, Georgia Institute of Technology, April 1993.

140. Mello, O. A., "An Improved Hybrid Navier-Stokes/Full-Potential Method fro Computation of Unsteady Compressible Viscous Flows," Ph.D. Thesis, School of Aerospace Engineering, Georgia Institute of Technology, November, 1994.

141. Cebeci, T., "Calculation of Compressible Turbulent Boundary Layers with Heat and Mass Transfer," AIAA Paper 70-741, June, 1970.

VITA

Richard Allen Weed was born on October 4, 1952 in Cleveland, Mississippi. He is the son of Mr. and Mrs. Jesse L. Weed of Ruleville, Mississippi. He graduated from Ruleville High School in May, 1970. Mr. Weed received a Bachelor of Science degree in Aerospace Engineering from Mississippi State University in May, 1974. He completed the coarse requirements for a Master of Science degree in Aerospace Engineering at Mississippi State University but left before finishing the thesis requirement to accept a position as a Guidance and Control Engineer in the Software Development Branch of the McDonnell-Douglas Technical Services Company in Houston, Texas in May, 1976.

At McDonnell-Douglas, Mr. Weed rose to the rank of Task Manager for the Ascent Guidance Group before returning to Mississippi State in June, 1979 to complete the requirements for his Master's degree. He received a Master of Science degree in Aerospace Engineering from Mississippi State University in August, 1980. Mr. Weed joined the Advanced Flight Sciences Department of the Lockheed-Georgia Company in Marietta, Georgia as a Scientist, Associate in August 1980. He entered the Ph.D. program in Aerospace Engineering at the Georgia Institute of Technology in September, 1985 as a part-time student while continuing to work at Lockheed.

Mr. Weed accepted a position as a Research Specialist with the Lockheed Missiles and Space Company in Sunnyvale, California in September, 1990. He returned to Georgia Tech as a full time graduate student in January, 1993. He is a senior member of the American Institute of Aeronautics and Astronautics and a member of Sigma Gamma Tau honor society.