Spring 1997


Amidu Olawale Oloso  
Old Dominion University

Follow this and additional works at: https://digitalcommons.odu.edu/mae_etds

Part of the Aerodynamics and Fluid Mechanics Commons, Computer Sciences Commons, Mechanical Engineering Commons, and the Structures and Materials Commons

Recommended Citation
https://digitalcommons.odu.edu/mae_etds/149

This Dissertation is brought to you for free and open access by the Mechanical & Aerospace Engineering at ODU Digital Commons. It has been accepted for inclusion in Mechanical & Aerospace Engineering Theses & Dissertations by an authorized administrator of ODU Digital Commons. For more information, please contact digitalcommons@odu.edu.
THREE-DIMENSIONAL AERODYNAMIC DESIGN
OPTIMIZATION USING DISCRETE SENSITIVITY ANALYSIS AND
PARALLEL COMPUTING

by

Amidu Olawale Oloso
B.Sc., 1986, University of Ibadan, Nigeria
M.Sc., 1988, University of Ibadan, Nigeria

A Dissertation submitted to the Faculty of Old Dominion University
in Partial Fulfillment of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY
MECHANICAL ENGINEERING
OLD DOMINION UNIVERSITY
May 1997

Approved by:

Arthur C. Taylor, III (Director)

Sushil K. Chaturvedi

Gene J.-W. Hou

Robert E. Smith, Jr.

Surendra N. Tiwari

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
ABSTRACT

THREE-DIMENSIONAL AERODYNAMIC DESIGN
OPTIMIZATION USING DISCRETE SENSITIVITY ANALYSIS AND
PARALLEL COMPUTING

Amidu O. Oloso
Old Dominion University, May 1997
Director: Dr. Arthur C. Taylor III

A hybrid automatic differentiation/incremental iterative method was implemented in the general purpose advanced computational fluid dynamics code (CFL3D Version 4.1) to yield a new code (CFL3D.ADII) that is capable of computing consistently discrete first order sensitivity derivatives for complex geometries. With the exception of unsteady problems, the new code retains all the useful features and capabilities of the original CFL3D flow analysis code. The superiority of the new code over a carefully applied method of finite-differences is demonstrated.

A coarse grain, scalable, distributed-memory, parallel version of CFL3D.ADII was developed based on “derivative stripmining”. In this data-parallel approach, an identical copy of CFL3D.ADII is executed on each processor with different derivative input files. The effect of communication overhead on the overall parallel computational efficiency is negligible. However, the fraction of CFL3D.ADII duplicated on all processors has significant impact on the computational efficiency.

To reduce the large execution time associated with the sequential 1-D line search in gradient-based aerodynamic optimization, an alternative parallel approach was developed. The execution time of the new approach was reduced effectively to that of one flow analysis, regardless of the number of function evaluations in the 1-D search. The new approach was found to yield design results that are essentially identical to those obtained from the traditional sequential approach but at much smaller execution time.
The parallel CFL3D ADII and the parallel 1-D line search are demonstrated in shape improvement studies of a realistic High Speed Civil Transport (HSCT) wing/body configuration represented by over 100 design variables and 200,000 grid points in inviscid supersonic flow on the 160-node IBM SP2 parallel computer at the Numerical Aerospace Simulation (NAS) facility, NASA Ames Research Center. In addition to making the handling of such a large problem possible, the use of parallel computation provided significantly reduced overall execution time and turnaround time.
DEDICATION

To Allah (SWT), the Inventor and Master of Aerodynamics, Who says;

In the name of Allah, the Most Gracious, the Most Merciful.

Do they not observe
The birds above them,
Spreading their wings
And folding them in?
None can uphold them
Except (Allah) Most Gracious:
Truly it is He
That watches over all things

Holy Qur’an, Chap. 67, Verse 19

Comments by A. Yusuf ‘Ali:

The flight of birds is one of the most beautiful and wonderful things in nature. The make and arrangement of their feathers and bones, and their streamlined shapes, from beak to tail, are instances of purposive adaptation. They soar with outstretched wings; they dart about with folded wings; their motions upwards and downwards, as well as their stabilization in the air, and when they rest on their feet, have given many ideas to man in the science and art of aeronautics. But who taught or gave to birds this wonderful adaptation? None but Allah, Whose infinite Mercy provides for every creature just those conditions which are best adapted for its life.
ACKNOWLEDGEMENTS

I give all thanks to Allah (SWT) who has made the successful completion of this effort possible.

My most heartfelt and sincere thanks go to my advisor, Dr. Arthur C. Taylor III, whose support and encouragement has had the most positive impact in preparing this dissertation. I would like to thank the other members of my committee Drs. Sushil K. Charturvedi, Gene Hou, Robert Smith and Surendra N. Tiwari for their valuable suggestions in the course of this study. In addition, I would like to express my gratitude to the following people for their useful technical advice and suggestions: Dr. V. M. Korivi of Optimal CAE, Drs. Shreekant Agrawal, Samson Cheung, James Hager, Peter Hartwich and Eric Unger of McDonnell-Douglas Transport Aircraft, Drs. Perry Newman and Larry Green of NASA Langley Research Center, and finally, the people who helped with ADIFOR – Dr. Chris Bischof of Argonne National Laboratory, and Dr. Alan Carle of Rice University. I would like to acknowledge Dr. A. O. Demuren through whose assistance I was recruited into the Ph.D. program and who also first introduced me to parallel computing.

Furthermore, I would like to express my highest love, gratitude and appreciation to my late mother, father, step-mothers, wife, friends, brothers, and other members of my family. I thank Drs. Lamidi Karimu, M. A. Akinlabi and B. Alabi for their support and encouragement.

This work was funded partly by McDonnell-Douglas Corporation. Computations were performed on supercomputers and parallel computers at NASA Langley and NASA Ames Research Centers.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>ABSTRACT</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ii</td>
</tr>
<tr>
<td>DEDICATION</td>
<td>v</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>vi</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>ix</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>xi</td>
</tr>
<tr>
<td>NOMENCLATURE</td>
<td>xiii</td>
</tr>
</tbody>
</table>

Chapter

## I. INTRODUCTION

1.1 Rationale and Motivation ........................................ 1
1.2 Literature Survey .................................................. 2
  1.2.1 Sensitivity Analysis ........................................ 2
  1.2.2 Design Optimization ........................................ 11
1.3 Scope and Objective of the Present Work ..................... 14
1.4 Thesis Outline ..................................................... 17

## II. FLOW FIELD ANALYSIS

2.1 Introduction ....................................................... 18
2.2 Governing Equations .............................................. 19
2.3 Time Integration .................................................. 21
2.4 Space Discretization ............................................. 22
  2.4.1 Residual Discretization .................................... 22
  2.4.2 Implicit Linearization Jacobians ......................... 25
2.5 Initial and Boundary Conditions ............................... 29
2.6 Flow Analysis Results ........................................... 29
  2.6.1 Test Case 1: Flow Over a Generic HSCT Wing/Body 30
  2.6.2 Test Case 2: Flow Over a Proprietary HSCT Wing/Body 39

## III. IMPLEMENTATION OF THE INCREMENTAL ITERATIVE METHOD

IN CFL3D VIA AUTOMATIC DIFFERENTIATION ................. 40

3.1 Introduction ....................................................... 40
3.2 First Order Aerodynamic Sensitivity Equations ............ 40
  3.2.1 Basic Equations ............................................ 41
  3.2.2 The Incremental Iterative Solution Method (IIM) .... 43
3.3 Automatic Differentiation (AD) and Incremental Iterative
  Method (IIM) ...................................................... 47
3.4 Computational Issues ............................................ 51
  3.4.1 Vector Computers ............................................ 51
  3.4.2 Scalar Computers ............................................ 53
3.5 The CFL3D.ADII Shape Sensitivity Analysis Code ............... 56

IV. CODE VALIDATION ................................................................. 59

4.1 Introduction ................................................................. 59
4.2 CFL3D.ADII (Version 1) ...................................................... 59
  4.2.1 Test Case 1: Generic HSCT Wing/Body Configuration .... 59
  4.2.2 Test Case 2: Proprietary HSCT Wing/Body Configuration . 67
  4.2.3 Summary of Results for CFL3D.ADII (Version 1) .......... 74
4.3 CFL3D.ADII (Version 2) ....................................................... 85

V. DISTRIBUTED-MEMORY PARALLEL IMPLEMENTATION OF
   CFL3D.ADII ........................................................................... 83
  5.1 The Challenge .......................................................... 83
  5.2 The NAS IBM-SP2 ......................................................... 84
  5.3 The Approach ............................................................. 86
  5.4 Parallel Sensitivity Analysis Results .............................. 91
  5.5 Efficiency, Speedup and Scalability Analysis .................. 91

VI. AERODYNAMIC SHAPE OPTIMIZATION STUDIES WITH
   PARALLEL 1-D LINE SEARCH ............................................. 102
  6.1 Introduction ............................................................... 102
  6.2 Parallel 1-D Line Search ............................................. 104
  6.3 Design Optimization Studies ........................................ 107
    6.3.1 Design 1: 108 Design Variables ......................... 108
    6.3.2 Design 2: 60 Design Variables, Camber and Twist ... 127

VII. CONCLUSIONS AND RECOMMENDATIONS ..................... 146
  7.1 CFL3D.ADII ............................................................... 146
  7.2 Parallel CFL3D.ADII .................................................... 147
  7.3 Parallel 1-D Line Search ............................................. 149
  7.4 Specific Contributions of this Study .............................. 150

REFERENCES ................................................................. 151

APPENDICES

A. SENSITIVITY DERIVATIVES FROM PARALLEL CFL3D.ADII .... 162
B. FINAL VALUES OF THE DESIGN VARIABLES FOR
  DESIGN 1 AND DESIGN 2 .................................................... 167
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Effects of Machine Precision on Grid Sensitivities from the Finite Difference Method and from an AD-Enhanced Grid Generation Code</td>
<td>55</td>
</tr>
<tr>
<td>4.1(a) Sensitivity Derivatives of $C_L$ (Lift Coefficient), $C_D$ (Drag Coefficient), $C_y$ (Force Coefficient in y-direction) and $CM_x$ (Pitching Moment Coefficient) for Test Case 1 (Generic HSCT Wing/Body Configuration) using the CFL3D.ADI code</td>
<td>63</td>
</tr>
<tr>
<td>4.1 (b) Sensitivity Derivative Ratios ($\frac{\text{CFL3D.ADI}}{\text{Finite Difference}}$)</td>
<td>63</td>
</tr>
<tr>
<td>4.2 (a) Sensitivity Derivatives of $C_L$ (Lift Coefficient), $C_D$ (Drag Coefficient), $C_y$ (Force Coefficient in y-direction) and $CM_x$ (Pitching Moment Coefficient) for Test Case 2 (Proprietary HSCT Wing/Body Configuration) using the CFL3D.ADI code</td>
<td>70</td>
</tr>
<tr>
<td>4.2 (b) Sensitivity Derivative Ratios ($\frac{\text{CFL3D.ADI}}{\text{Forward Finite Difference}}$)</td>
<td>70</td>
</tr>
<tr>
<td>4.2 (c) Sensitivity Derivative Ratios ($\frac{\text{CFL3D.ADI}}{\text{Central Finite Difference}}$)</td>
<td>71</td>
</tr>
<tr>
<td>4.3 (a) CPU Timings for CFL3D, Version 4.1 (Baseline) with Different Code Options for Test Case 2</td>
<td>78</td>
</tr>
<tr>
<td>4.3 (b) CPU Timings for CFL3D.ADI (Versions 1.0 and 2.0) with Different Code Options and 1 Design Variable for Test Case 2</td>
<td>78</td>
</tr>
<tr>
<td>4.3 (c) CPU Timings for CFL3D.ADI (Versions 1.0 and 2.0) with Different Code Options and 5 Design Variables for Test Case 2</td>
<td>78</td>
</tr>
<tr>
<td>4.3 (d) CPU Time Ratios ($\frac{\mu \text{sec}/MGC/\text{GP}/\text{DV}}{\mu \text{sec}/MGC/\text{GP}}$ for CFL3D.ADI) with Different Code Options for Test Case 2</td>
<td>79</td>
</tr>
<tr>
<td>5.1 Sensitivity Derivatives for the Lift Coefficient (CL) from the Parallel Version of CFL3D.ADI for Ten Randomly Selected Design Variables (DVs). Results from the Cray Version for Five of the DVs are included for Comparison</td>
<td>92</td>
</tr>
<tr>
<td>5.2 Sensitivity Derivatives for the Drag Coefficient (CD) from the Parallel Version of CFL3D.ADI for Ten Randomly Selected Design Variables (DVs). Results from the Cray Version for Five of the DVs are included for Comparison</td>
<td>92</td>
</tr>
</tbody>
</table>
5.3 Sensitivity Derivatives for the X-Moment Coefficient (CMx) from the Parallel Version of CFL3D.ADII for Ten Randomly Selected Design Variables (DVs). Results from the Cray Version for Five of the DVs are included for Comparison ........................................ 93

6.1 Design Improvement Summary for Design 1 (108 Design Variables: Camber, Thickness and Twist). Results from Sequential 1-D Search included for Comparison ........................................ 109

6.2 Final Values of 10 Randomly Selected Design Variables for Design 1. Results from Sequential 1-D Search included for Comparison ........................................ 109

6.3 Design Improvement Summary for Design 2 (60 Design Variables: Camber and Twist) ........................................ 129

6.4 Final Values of 10 Randomly Selected Design Variables for Design 2 ........................................ 129


B.1 Final Values of Design Variables for Design 1 (108 Design Variables: Camber, Thickness and Twist) .......... 167

B.2 Final Values of Design Variables for Design 2 (60 Design Variables: Camber and Twist) ........................................ 172

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Three-Factor ADI Scheme; $\xi$ Sweep</td>
<td>27</td>
</tr>
<tr>
<td>2.2 HSCT 24E Filleted Wing/Body Configuration (with Surface Grid in the Wake Region)</td>
<td>31</td>
</tr>
<tr>
<td>2.3 Some of the Surfaces for the HSCT 24E Supersonic Euler Grid</td>
<td>32</td>
</tr>
<tr>
<td>2.4 One of the $x=$constant Grid Planes in the Wake Region. The Enlarged Portion show the Geometric Point Mismatch</td>
<td>34</td>
</tr>
<tr>
<td>2.5 Upper and Lower Grid Blocks for the HSCT 24E Supersonic Euler Grid</td>
<td>35</td>
</tr>
<tr>
<td>2.6 $x=$constant Planes from Upper and Lower Grid Blocks</td>
<td>36</td>
</tr>
<tr>
<td>2.7 Upper Surface (including the Wake Region) Pressure Contours. (a) From Present Study; (b) From the Code used in [35]</td>
<td>37</td>
</tr>
<tr>
<td>2.8 Lower Surface (including the Wake Region) Pressure Contours. (a) From Present Study; (b) From the Code used in [35]</td>
<td>38</td>
</tr>
<tr>
<td>4.1 Wing Planform Design Variables for Test Case 1, HSCT 24E Geometry</td>
<td>61</td>
</tr>
<tr>
<td>4.2 CPU Timing Results for Test Case 1 (2-Level Multigrid, Roe's Diagonalized Scheme, No Limiter, $M_\infty = 2.4$, $\alpha = 1.0$ degrees)</td>
<td>65</td>
</tr>
<tr>
<td>4.3 CPU Timing Results for Test Case 2 (2-Level Multigrid, Roe's Diagonalized Scheme, No Limiter, $M_\infty = 2.4$, $\alpha = 1.9$ degrees)</td>
<td>72</td>
</tr>
<tr>
<td>4.4 Memory Required by CFL3D (Version 4.1) and CFL3D.ADII (Versions 1 and 2) for Test Case 2</td>
<td>76</td>
</tr>
<tr>
<td>5.1 File System Arrangement for the Parallel CFL3D.ADII</td>
<td>90</td>
</tr>
<tr>
<td>5.2 Variation of Parallel Efficiency, $E$ with Number of Nodes, $N$</td>
<td>99</td>
</tr>
<tr>
<td>5.3 Variation of Speedup, $S$ with Number of Nodes, $N$</td>
<td>100</td>
</tr>
<tr>
<td>6.1 Chordwise Variation of the Percentage Change in Mean Camber Line, $Z$ for Design 1 (108 Design Variables, Camber, Thickness &amp; Twist)</td>
<td>111</td>
</tr>
<tr>
<td>6.2 Spanwise Variation of the Percentage Change in Mean Camber Line, $Z$ for Design 1 (108 Design Variables, Camber, Thickness &amp; Twist)</td>
<td>118</td>
</tr>
</tbody>
</table>
6.3 Chordwise Variation of the Percentage Change in Thickness for Design 1 (108 Design Variables, Camber, Thickness & Twist) . . . 119

6.4 Optimization History of the Normalized Objective (\(\)) and Normalized Constraints (\(\&\)) for Design 2 (60 Design Variables, Camber & Twist) .................................................. 128

6.5 Chordwise Variation of the Percentage Change in Mean Camber Line, \(Z\) for Design 2 (60 Design Variables, Camber & Twist) . . . 130

6.6 Spanwise Variation of the Percentage Change in Mean Camber Line, \(Z\) for Design 2 (60 Design Variables, Camber & Twist) . . . 137

6.7 Chordwise Variation of the Percentage Change in Thickness for Design 2 (60 Design Variables, Camber & Twist) ................. 138
NOMENCLATURE

a  speed of sound
b  design variable vector
A, B, C  inviscid Jacobians
C_D  drag coefficient
C_L  lift coefficient
C_{x, y, z}  force coefficients in x, y, z directions
C_{Mx, My, Mz}  moment coefficients in x, y, z directions
CFL  Courant–Friedricks–Lewy number
\hat{F}, \hat{G}, \hat{H}  conserved inviscid fluxes in curvilinear coordinates
e_o  total specific internal energy
I  identity matrix
k  thermal conductivity
p  pressure
Q  field variables
R  residuals
t  time
T  temperature
u, v, w  Cartesian velocity components
U, V, W  contravariant velocity components
x, y, z  Cartesian coordinates

Greek Symbols

a  angle of attack, 1-D line search parameter
\gamma  specific gas constant
\delta_z, \delta_y, \delta_z  finite difference operators for first derivatives
\kappa  spatial accuracy parameter
\lambda  eigenvalue

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
$\Lambda$ diagonal matrix of eigenvalues
$\xi, \eta, \zeta$ body-fitted curvilinear coordinates
$\rho$ density

**Superscripts**
- non-positive eigenvalues
+ non-negative eigenvalues
n time level
m iteration index
* intermediate solutions
T transpose

**Subscripts**
i, j, k nodal points/indices
$\xi, \eta, \zeta$ indicates derivatives relative to these curvilinear directions
$\infty$ free-stream value

**Derivative Quantities**
$F'$ sensitivity of output function $F$
$Q'$ sensitivity of field variables $Q$
$\frac{\partial R}{\partial Q}$ approximate Jacobian operator
$\frac{\partial R}{\partial Q} \frac{\partial R}{\partial \bar{X}}$ Jacobian matrices
$X'$ grid sensitivity

**Miscellaneous**
$\partial$ partial derivative
$\Delta$ incremental change
$\Delta$ backward difference operator
$\nabla$ forward difference operator, gradient or "del" operator
CHAPTER I
INTRODUCTION

1.1 Rationale and Motivation

The field of Computational Fluid Dynamics (CFD) has reached a mature stage where advanced flow codes have been developed to solve engineering fluid-flow problems for a wide variety of flow conditions. Numerical simulation of complex internal and external flows has become a routine. Examples of advanced CFD codes include CFL3D [1, 2], TLNS3D [3], OVERFLOW [4] and PROTEUS [5]. Although CFD has reached an advanced stage, a typical realistic aerodynamic problem can be highly nonlinear and resolving the details of the flow physics, if possible, can be quite computationally intensive. This poses a particularly difficult problem when an advanced CFD code is to be incorporated in gradient-based optimization techniques where several CFD analyses may have to be carried out in a typical design cycle. Using CFD in design optimization has been identified by Jameson [6] as one of the challenges to be met in the area of CFD.

In formal optimization methods with gradient-based approach, gradients or sensitivity derivatives of objective and constraint functions serve as input to the optimizer. In aerodynamic design optimization, computing aerodynamic sensitivity derivatives becomes a major contributor to the overall computational cost. The need to provide CFD codes with sensitivity analysis (SA) capabilities as part of a multidisciplinary environment has been stressed by Sobieski [7]. The National Aeronautics and Space Administration (NASA) has initiated and is supporting several programs committed to multidisciplinary design optimization using SA. Examples include the High-Speed Airframe Integration Research (HiSAIR) project [8] and the Computational Aerosciences (CAS) project of the High Performance Computing and Communications (HPCC) program [9]. The High Speed Civil Transport (HSCT) design activity is under the umbrella of HiSAIR. According to Sobieski

[10], intrinsic to the future success of MDO applications is the maturation of sensitivity analysis–based optimization procedures within the individual engineering disciplines. This work concentrates on one such discipline — aerodynamic design optimization.

From the survey of literature, several earlier studies are noted where aerodynamic design optimization is performed using sensitivity derivatives obtained from CFD. With the exception of the work of Reuther et al. [11] where a continuous adjoint formulation was applied to a complex three–dimensional geometry using the Euler equations and a large number of design variables, the success reported is usually limited to one or more of the following: simplified flow physics, e.g. potential flow, two–dimensional or three–dimensional, inviscid, purely supersonic flows (efficiently solved via space–marching methods for the Euler equations); more sophisticated flow physics but simple geometries, e.g. airfoils and simple nozzles; a small number of design variables; etc. It is observed that the challenges posed by large-scale, industry–level aerodynamic design studies have largely not been met. These challenges include complicated 3–D geometries, large numbers of design variables, and large computational resource requirements that push the envelope of the capability of the currently available supercomputers resulting in large job turnaround times. The present work is motivated by a desire to meet these challenges using the more reliable discrete sensitivity analysis approach.

1.2 Literature Survey

1.2.1 Sensitivity Analysis

The literature on sensitivity analysis and optimization is quite extensive. The pioneering work on sensitivity analysis for MDO started with a plea from Sobieski [7, 12] to the CFD community for extending their present capabilities to include sensitivity analysis of aerodynamic functions.

Sensitivity analysis is based on the principle that approximating the behavior of an unknown function in the neighborhood of a known point can be accomplished efficiently and accurately if the slopes in the neighborhood of a known point are defined, and can be
determined at the known point. Sensitivity analysis can thus be defined as the calculation of slopes or gradients, known as sensitivity coefficients, where the derivatives of the response of a particular system of interest are taken with respect to the design variables of interest. There are many uses of sensitivity coefficients. For example, sensitivity coefficients can be used to assess the sensitivity of a computational model to perturbations in its parameters or initial conditions. This information can then be used to (i) generate a better initial guess for analysis or (ii) in the model validation stage, to verify robustness with respect to empirically determined parameters, that is, to verify that the model behaves as suggested by experimental data [13]. Sensitivity coefficients can also be used in function approximation to predict trends in the response of a system as a consequence of changes in the design variables. Finally sensitivity coefficients can be employed in trade–off design, system identification and design optimization. In particular, design sensitivity analysis helps make iterative aerodynamic design optimization schemes computationally feasible by reducing computational cost compared to direct methods [14]. Due to intensive research done in recent years, design sensitivity analysis has become an important discipline of its own. There are conferences and books [15–17] dedicated to design sensitivity analysis.

The earliest method of computing gradients of aerodynamic functions is the “brute force” finite difference (FD) approximation, which requires performing one (for forward differencing) or two (for central differencing) extra aerodynamic analyses for a perturbed value of each design variable. Thus the FD cost grows linearly with the number of design variables. Since a complete flow field analysis may be computationally expensive to perform, the computational cost of the finite difference approach may become prohibitive, especially if there are many design variables. In addition, it is noted that the FD method may suffer from accuracy problems. Its accuracy deteriorates with increasing step size in nonlinear problems; however, making the step size too small may also be ineffective. That is, the large, computationally costly residual reductions (which are typically required for accuracy with the FD method) may not be able to overcome the numerical noise if the step
size becomes too small. Thus the perturbation range in which the accuracy of the FD method is acceptable becomes problem dependent, and may not even exist [18]. A method known as the finite difference algorithm is outlined in [19] to automatically calculate an optimum step size. This finite difference algorithm was extended in [20] to functions that are governed by matrix equations. This algorithm has not yet been demonstrated for cases in which the functions are calculated iteratively, as in most CFD codes.

Motivated by the preceding shortcomings of the FD approach, several alternative techniques have evolved for computing aerodynamic sensitivity derivatives. These techniques fall under the discipline of sensitivity analysis (SA). Typically with practical engineering problems, analytical closed-form solutions amenable to symbolic differentiation are not available. Therefore, a quasi-analytical (QA) approach based on the Implicit Function Theorem (IFT) is a popular technique in SA, especially within the framework of an emerging field of research known as Multidisciplinary Design Optimization (MDO). Most of the published works in SA are connected to problems in structural mechanics. Among them, of particular interest is the problem of shape design sensitivity analysis, in which the system design parameter is the boundary or the contour of the system [21].

When the IFT is applied to a disciplinary analysis in discretized form, the technique differentiates the discretized governing equations to obtain the companion sensitivity equations that, according to IFT, are always linear, simultaneous algebraic equations in which the sensitivity derivatives (SDs) appear as unknowns [22]. The systems of algebraic aerodynamic sensitivity equations in standard form (i.e. non-incremental or non-delta form), are usually very large. They can be solved by direct methods [23–31], iterative methods [32, 33], or a hybrid direct/iterative method [34]. For a typical realistic three-dimensional aerodynamic design problem, the memory requirements for the direct linear solvers of the sensitivity equations become prohibitive. For such problems, iterative techniques may be the only option. In standard form, the left hand side (LHS) coefficient
matrix cannot be modified without affecting the accuracy of the SDs. Thus, for iterative methods, no approximation that could aid convergence or even make convergence possible can be made on the LHS. This limitations led to the development of the incremental iterative form or delta form for the SA equations. This new formulation allows the use of any iteratively convergent matrix on the LHS. The incremental iterative form is very suitable for CFD codes where the same LHS for flow analysis can easily be used also for sensitivity analysis [35–39]. An excellent review of the recent iterative techniques for SA was given by Taylor et al. [40]. Another approach is to divide the problem domain into small subdomains. Each subdomain is solved separately, and the final SA solution for the entire domain is obtained by iteratively interacting and then finally assembling the solutions from the subdomains [41, 42].

In the above IFT method, differentiation is carried out on the discretized governing equations. This approach is referred to as the discrete approach. An alternative approach is the continuous approach, where differentiation is carried out on the continuous governing equations (i.e. before discretization) using material derivatives [43–46] or generalized calculus of variations [47–53]. With the continuous approach, the resulting sensitivity equations are linear differential equations (for the material derivative approach) or adjoint equations (for the calculus of variations approach); typically these equations are eventually discretized and solved numerically to obtain the required gradient information. The continuous formulation approach offers advantage of flexibility in the sense that the governing equations and the discretization used for the SA can be different from that used for the flow analysis. To ensure accurate and consistent gradients, the discretization used for the SA governing equations as well as the boundary conditions should be consistent with that of the analysis disciplines [45]. The method may nevertheless suffer from significant accuracy problems.

To perform discrete SA, one can differentiate the flow solver code by hand. However, handcoding of derivatives for a large code is a tedious and error-prone process. Moreover for
nonlinear functions, the derivatives are generally more complicated than the function itself. Hence developing a derivative code by hand will likely require a considerable amount of work in comparison with the development of the original code, although it is likely to result in the most efficient code. To defray the cost associated with hand differentiation, an “automatic differentiation” method has recently become available. This method applies a line-by-line symbolic differentiation to an existing code and stores numerical values of the dependent variables for each line. Moving from one line to the next, the algorithm links the derivatives in a chain-differentiation manner as required by the variable dependencies from the beginning to the end of the code. The result is a set of the derivatives of the output with respect to input [22, 54]. The method is implemented in the form of an automatic differentiator code that reads the user’s existing source code and produces a new source code that retains exactly the same capability as the original code but which is enhanced with the ability to compute the sensitivity derivatives. A notable example of an automatic differentiator for FORTRAN is the ADIFOR (Automatic Differentiation of FORTRAN) tool [55–57].

In the discrete SA method, the number of linear systems to be solved to compute the required gradient information is equal to the number of design variables (DVs), since one linear system is solved for each design variable. Moreover, all these linear systems are completely decoupled from each other; thus they can be solved concurrently. This offers a good opportunity for a coarse-grained parallel scheme in which a linear system or a group of linear systems can be solved on a single processor in a multiprocessor environment. The fact that the linear systems are decoupled imply that minimal synchronization is required among processors during parallel computation. The advantage of this possibility cannot be overemphasized. This is because a single run of the sensitivity-enhanced code on a single processor may be forced to compute fewer than the desired number of sensitivities due to memory and/or computational time limitations. These features were exploited in [58] and
[59], and also in this work, to compute gradients in parallel with what is called "derivative stripmining".

Pironneau [60] used the continuous formulation applied to the Navier–Stokes equations to derive sensitivity equations for incompressible low–Reynolds–number flow. Angrand [47] used a similar approach for flow over an airfoil using the irrotational flow (potential flow) approximation. Yates [61] and Yates and Desmarais [62] used a continuous formulation applied to the equations of linear aerodynamic theory and successfully obtained SD's from the integral-equation formulation of these governing equations in two dimensions. Extension of this method to 3–D flow with the Navier–Stokes equations (for flow analysis and to calculate aerodynamic sensitivity derivatives) is possible, in principle. The integral-equation representation of the governing equations has advantages over conventional finite-difference and finite-volume methods, and these advantages carry over to the solution of the resulting sensitivity equations.

Reuther et al. [11], Jameson [63, 64], Jameson and Reuther [65] and Reuther [66] applied control theory to airfoil and wing design. In these works, a continuous formulation together with the adjoint-variable approach was used to obtain the required gradient information. Initially, the method was successfully implemented with conformal mapping for potential flow. Later it was extended to inviscid flow (using Euler equations) in two and three dimensions with a finite-volume discretization. With this method, 2+m flow analysis are required per design cycle, where two analyses are required to solve the flow equations and the adjoint equations (one analysis each) and m is the number of flow analyses required in the line-search procedure. The flow equations and the adjoint equations are solved efficiently by using the multigrid procedure in incremental iterative form. It was not certain whether the continuous adjoint formulations after being discretized maintain consistency with the analysis equations. Some of the continuous adjoint results in [11] and [66] show significant discrepancies in the computed sensitivity derivatives when compared with the finite-difference results. These discrepancies are attributed to lack of consistency.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Frank and Shubin [67], Shubin and Frank [43] and Shubin [68] obtained aerodynamic sensitivity equations using both the discrete and the continuous approaches. These studies indicate that consistent, discrete SD's should be used in aerodynamic design optimization; failure to do so can result in a considerable slowdown or complete failure of the optimization procedure. The continuous method generally does not yield consistent, discrete SD's.

Borgaard and Burns [45, 46] derived aerodynamic sensitivity equations in two dimensions by directly differentiating the continuous Euler equations and the accompanying boundary conditions. With this method, the nonlinear flow equations and the linear flow–sensitivity equations were solved with the same solution procedure. The authors observed in [45] that judicious use of inconsistent, discrete SD's can sometimes result in successful optimization. The approach used in [46] has the advantage that mesh sensitivities need not be computed. However, the authors observed that the derivatives computed are not always consistent. They demonstrated that, for a proper combination of discretization schemes, it was possible to have asymptotic consistency under mesh refinement, which they claim is often sufficient to guarantee convergence of the optimal design algorithm.

Using a continuous formulation, Ibrahim and Baysal [51] derived sensitivity equations in adjoint form and boundary (transversality) equations for the quasi–one dimensional (quasi 1–D) Euler equations. This approach differs from other methods in that a perturbation technique is applied with a variation formulation to find the required gradient information. The resulting adjoint sensitivity equations and flow–analysis equations are solved with the same solution procedure because these equations are similar. The method is applied to the optimization of a quasi–1–D nozzle, that includes a normal shock within the nozzle. Ibrahim [53] and Ibrahim et al. [69] extended this approach to 2–D Euler equations.

Elbanna and Carlson [32] applied the discrete sensitivity approach to calculate aerodynamic sensitivity coefficients in the transonic and supersonic flight regimes, where the governing equations of fluid flow considered are the transonic small–disturbance equations. Later, this approach is applied to the 3–D full–potential equation to compute
aerodynamic sensitivity coefficients for a wing in a transonic flow. In order to avoid the excessive memory of a direct solver approach, they used a conjugate-gradient iterative method to solve the very large system of linear sensitivity equations that is associated with 3-D flow. Elbana and Carlson [70] used a symbolic manipulator, MACSYMA [71], to differentiate various parts of the 3-D full potential flow code and successfully obtain aerodynamic SDs.

Eleshaky and Baysal [41] proposed a domain decomposition technique to solve the discrete sensitivity equations for large 2-D and 3-D problems. This method decomposes the large computational domain into subdomains; the sensitivity equations for the interior cells and the sensitivity equations for the boundary cells that couple the subdomains are iteratively solved with a preconditioned conjugate gradient technique. The feasibility of computing SD's on decomposed computational domains in 2-D was demonstrated on a sample airfoil problem by Lacasse and Baysal [72]; and in 3-D on an axisymmetric nacelle configuration by Eleshaky and Baysal [41].

Chattopadhya and Pagaldipti [73] obtained discrete SDs from the 3-D parabolized Navier–Stokes equations and demonstrated the method for flow over a delta wing. The grid sensitivity terms were calculated using finite differences. In a later study on the same subject [74], the grid sensitivity terms were obtained quasi–analytically. Huddleston et al. [75] calculated consistent, discrete SDs from a 2-D Euler solver using the Gauss–Seidel algorithm with subiterations. The example used in their study was flow over an airfoil at subsonic and transonic flow conditions: they defined the shape of the airfoil with a Bezier–Bernstein parametrization. In their study, they note a discrepancy in the SD's when the quasi–analytical results are compared with the results obtained from finite difference; this discrepancy is attributed to approximation of the derivatives of Roe's flux–difference–splitting scheme.

Korivi et al. [35] and Newman et al. [76] proposed the incremental iterative method (IIM) to solve the sensitivity equation to calculate consistent, discrete SDs. With this
approach, approximations of convenience can be introduced into the coefficient matrix operator without affecting the accuracy of the SD. The IIM enables the same solution strategy that is used to solve the equations of the flow analysis to be used to solve the flow sensitivity equations. This IIM strategy was first implemented in two dimensions for the TLNS equations with both the direct-differentiation and the adjoint-variable approaches; the procedure was demonstrated for two airfoil problems: low-Reynolds-number laminar flow and high-Reynolds-number turbulent flow. In their work, the failure to differentiate the turbulence modeling terms (because of their complexity) resulted in inaccurate discrete SD's. Later, the IIM strategy was implemented in a 3-D marching Euler code to obtain SD's for several geometric and non-geometric design variables [36, 77].

Taylor [37] implemented the IIM strategy of [35] in an advanced, widely used CFD code, CFL3D (Version 4.1) [1]. Efforts were made to retain the full capabilities of the original code during the SD computation process. All necessary differentiations were carried out using ADIFOR [57]. The ADIFOR-enhanced IIM code is capable of computing SD's for multiblock or overlap grids using both multigrid and mesh sequencing techniques (originally for non-linear flow analysis) to accelerate convergence of the linear SD equations. The new code was verified on a model problem: that of single block, transonic flow over an ONERA M6 wing at high Reynolds number using the Baldwin–Lomax turbulence model. Derivatives of the conventional aerodynamic force and moment coefficients were calculated at steady state with respect to a single design variable, that of wing twist at the tip. Derivative results using this new code agree exactly with the derivative results obtained by differentiating CFL3D as a black-box code. Although the IIM code is faster than the black-box code, it is found to be inefficient compared to the original CFL3D code. Several suggestions were made for improvement of the code. Further verifications of the code were carried out by Taylor et al. [38] and in the present work by computing SDs of aerodynamic functions for the wing–body configuration of the HSCT 24E aircraft [36] in supersonic flow. The geometry necessitated the use of a 2–block arrangement for the computational grid; thereby testing the
multiblock capability of the new SD code. The 2-block arrangement required additional information of the sensitivity of interpolation coefficients at the patched interface which is obtained by applying ADIFOR to the supplemental code (RONNIE), which computes these interpolation coefficients.

Bischof et al. [58] computed SD's for a swept transport wing in turbulent, transonic flow both on the Cray Y-MP computer and on the IBM-SP1 parallel computer by coupling an ADIFOR-enhanced wing grid generation program to an ADIFOR-enhanced state-of-the-art 3-D CFD code. On the IBM-SP1, they used coarse-grained parallelism via derivative stripmining to compute SDs, with each SP1 node computing equal (or almost equal) portion of of the total number of SDs. The authors observed that for a small number of design variables, the Cray Y-MP implementation was much faster. However, as the number of design variables grew, the IBM-SP1 became an attractive alternative in terms of computational speed, job turnaround time, and total memory available.

1.2.2 Design Optimization

The most popular design optimization techniques are the gradient-based techniques. There are other less popular non-gradient-based techniques which use genetic algorithms, simulated annealing and neural networks. The review presented here concentrates on the gradient-based techniques which have direct relevance to the present study. Gradient based techniques can be tightly coupled or loosely coupled. A common feature of the tightly coupled formulations is that the function evaluation iterations are concurrent with and embedded within the optimization cycles. Usually in tightly coupled systems, converged function evaluation and converged optimum solution are arrived at simultaneously. In loosely coupled systems, the optimizer is separated from the function evaluation routine. The communication between the two is limited to (i) the optimizer passing the vector of design variables to the function/gradient routines as necessary and (ii) the function/gradient routines returning the required converged function(s) and/or gradient(s) to the optimizer.

Note that in loosely coupled systems, each call to the function/gradient routines demands a
converged flow solution. A gradient–based design may also be a direct design or an inverse design. In direct design, no assumption is made about the optimum solution. The optimizer is left to search for one. In inverse design, a desired targeted feature of the system is specified before hand. The design process is then steered to achieve this target. For example, in aerodynamic shape design studies, an inverse design is a procedure in which typically a target surface–pressure distribution is specified, and the corresponding shape that will best fit this pressure profile is calculated. By the nature of the inverse design, it is apparent that problem formulation must be done carefully since it is possible that the specified pressure distribution, for example, may not be physically realizable; see [78] for a review of inverse design methods.

Some examples of the tightly coupled systems include the simultaneous analysis and design (SAND) optimization formulation of Rizk [79], the “one shot procedure” of Ta’asan et al. [50] and Kuruvila et al. [80], the simultaneous aerodynamic analysis and design (SAADO) formulation of Hou et al. [81], the single– and multi–SAND–SAND approaches of Balling & Sobieski [82], etc. Rizk [83] summarized several CFD applications of the technique in [79]. Hou et al. [81] successfully demonstrated tightly coupled optimization with a discrete adjoint formulation in application to a quasi–1–D nozzle problem. The derivations in [81] are closely related to variational or control theory techniques. Ta’asan et al. [50] and Kuruvila et al. [80] used a continuous adjoint formulation to obtain gradient information and formulated the “one shot procedure” which is a tightly coupled optimization scheme in which a highly efficient multigrid method is used to solve the potential–flow equations and the accompanying adjoint sensitivity equation. With this method, the entire optimization procedure requires only about two to three times the computational cost of a single–flow analysis. Huffman et al. [84] used a continuous adjoint formulation coupled with mesh sequencing to implement a simultaneous analysis and design optimization procedure in the TRANAIR code, which solves the full–potential equations of 3–D fluid
flow. They employed a quasi–Newton type solver to efficiently solve the flow analysis and adjoint sensitivity equations.

Several aerodynamic design studies based on the loosely coupled approach also exist in literature. For 3-D inviscid flow over a wing, Burgreen [33] and Burgreen and Baysal [85, 86] considered both wing-section and planform design variables in their aerodynamic shape–optimization study. Jameson [87] considered wing-section variables only (for a fixed planform) and implemented an optimization technique based on control theory. Chattopadhya and Pagaldipati [73] developed a multidisciplinary, multilevel decomposition procedure for the optimal design of a high-speed transport wing with the parabolized Navier–Stokes equations and quasi–analytical aerodynamic SDs.

Korivi et al. [88] and Korivi [36] used consistent, discrete SD's obtained by the direct–differentiation approach via the incremental iterative method (IIM) with a space–marching algorithm for the Euler equations. Design–improvement studies were accomplished by using grid sensitivities from an automatically differentiated grid–generation code. The HSCT 24E configuration was chosen as the test case for the design–improvement studies. However because of the space–marching algorithm used, only fully supersonic flow can be handled. The scheme is not applicable to general fluid flow problems where characteristic waves are travelling both upstream and downstream.

Burgreen et al. [89] performed aerodynamic shape optimization of 3-D wings by replacing the usual grid point–based approach for surface parametrization with a Bezier–Bernstein polynomial parametrization. It was not clear whether this method will be efficient for realistic 3-D geometries. Other notable schemes include variable complexity design strategies, developed by Hutchinson et al. [90, 91] to combine conceptual and preliminary–design approaches. The strategy has been used to optimize the HSCT wing configuration. Verhoff et al. [92, 93] developed a method for optimal aerodynamic design of wing–sections using analytically computed aerodynamic sensitivities. The scheme also utilizes Chebyshev polynomials together with parametric stretching functions to define
camber and thickness distribution of wing-section. Due to analytical parametrization of the surface, the package produces efficient optimal results.

Oloso and Taylor [59] and the present study developed a scalable massively parallel version of the ADIFOR-IIIM sensitivity-enhanced CFL3D code (CFL3D.ADII) which was used to compute gradients on the IBM-SP2 for a proprietary HSCT wing/body geometry represented by more than 200,000 grid points and using more than 100 design variables concurrently. The configuration was subsequently optimized for drag reduction while the lift and the wing-root bending moment were constrained to their baseline values. Despite the large computational requirements of this problem, design results were obtained at an average execution time of about 17 hours/cycle on the IBM-SP2.

1.3 Scope and Objective of the Present Work

The development of a realistic 3-D aerodynamic design process is still faced with many challenges such as complex geometry, large number of design variables, large CPU time and memory requirements, and long turnaround time. These challenges must be met for aerodynamic analysis/design to be a routine part of a multidisciplinary design optimization strategy. As the review of literature has shown, there are only limited attempts where an existing high-fidelity, well tested, state-of-the-art flow solver is used for flow analysis as well as sensitivity analysis. In these few attempts, only simple 3-D geometries, often parametrized by a few design variables, have been considered. Parametrization with only a few design variables is usually not sufficient for industry-level, practical design optimization of realistic aerodynamic configurations. The incorporation of an advanced flow solver in an aerodynamic design process will provide the opportunity for a good resolution of the flow physics as the geometry under consideration evolves towards the optimum. However, to make the use of such a solver feasible, mechanisms need to be developed and employed to reduce the CPU requirement, computer memory and turnaround time. This was the main focus of this study. The specific objectives of this study are as follows:
(1) To extend the capabilities of a 3-D, general purpose, high-fidelity flow solver to include efficient, automatic computation of consistently discrete SD’s of aerodynamic functions of interest;

(2) To perform computational studies on aerodynamic SD’s of a complex geometry using the AD-enhanced code with provisions for handling advanced features of the original code such as multiple block grids;

(3) To develop a parallel version of the AD-enhanced code for a distributed memory computing platform so that a large number of SD’s can be computed concurrently;

(4) To develop a parallel computing alternative to the sequential one dimensional line search in design optimization;

(5) To perform design optimization studies where parallel computation of SDs and a parallel 1-D line search will be incorporated within each design cycle.

For the first objective, the code selected is a popular, well-tested advanced flow solver known as CFL3D [1]. The code was developed at NASA Langley Research Center. The general features of the code are described in Chap. II of this study. Due to the method of solution used in the code, the code is amenable to being modification for the IIM [35] for computing aerodynamic SDs. The IIM has been demonstrated in [36] to work efficiently within the framework of a space-marching 3-D Euler algorithm for pure supersonic flow. Hence this technique will be implemented in the general purpose CFL3D code so that SDs can be efficiently computed for all flow regimes and boundary conditions. Prior to this work, a preliminary implementation of the IIM in CFL3D has been completed by Taylor [37]. The present work will build on this earlier work.

For the second objective, first order SDs of aerodynamic functions is computed for the wing–body configurations of two HSCT-like aircraft geometries, one generic and the other proprietary. To prove that the advanced features of the pure flow solver extends to the flow/sensitivity analysis solver, one of the geometries is represented by two grid blocks that are connected in a general patched manner. Information on sensitivities of interblock
communication coefficients with respect to geometric design variables is incorporated in the SA calculations for accurate computation of SD's.

For the third objective, the goal is to parallelize the SA code using the derivative stripmining approach of [58]. In this approach, a coarse grain parallel implementation is developed for a distributed-memory environment where a predetermined number of design variables can be assigned to each processor. A successful parallel, distributed-memory implementation will not only reduce turnaround time but will also make aerodynamic SD computation of realistic configurations possible on a massively parallel computer like the IBM-SP2 or even a cluster of workstations where the available memory local to a node or workstation may be limited. The required communication among processors is achieved using the standard Message Passing Interface (MPI) software developed at Argonne National Laboratory (ANL) [94].

The fourth objective is achieved by developing an alternative 1-D line search procedure that is suitable for implementation on distributed-memory computers. The approach is to replace the function evaluation process which requires sequential output (a move parameter, usually denoted by $\alpha$ in the 1-D line search) from the optimizer by a process which anticipates and approximates all possible values of the sequential optimizer's output (i.e. a range that contains all possible move parameters) and computes a priori the required functions in parallel. When the actual move parameters are computed, the optimizer can then "pick" the corresponding functions from the "list" of functions computed a priori. In this manner, the more typical time-consuming sequential function evaluations can be avoided.

Finally, the fifth objective is realized by developing an aerodynamic optimization package making use of all the facilities developed in the first four objectives described above. Design improvement studies are carried out on the wing/body configuration of the proprietary HSCT-like aerovehicle. The goal is to minimize wave drag subject to constraints on lift and wing-root bending moment. Only supersonic flow (Mach No. = 2.4) is considered and the governing flow physics is for the inviscid flow, represented by the Euler equations.
well known and well tested optimizer. Automatic Design Synthesis (ADS) [95] is used to coordinate the optimization process.

1.4 Thesis Outline

The remainder of this study is arranged as follows. Chapter II is concerned with fluid flow analysis where the basic equations governing the inviscid flow physics in CFL3D are presented together with boundary conditions and flow analysis results. Chapter III presents theoretical and computational issues pertaining to the implementation of the IIM in CFL3D on a vector supercomputer such as the Cray YMP and C90 for the evaluation of first order sensitivity derivatives. In Chap. IV, results are presented which show comparison between the IIM and the finite-difference methods for accuracy and computational efficiency. Chapter V is concerned with the development of coarse-grain parallel implementation of the sensitivity-enhanced CFL3D code on a distributed-memory massively parallel computing platform such as the IBM-SP2. In this chapter, results are presented for SDs obtained with respect to more than 100 design variables concurrently for a large 3-D problem of over 200,000 grid points. Chapter VI covers aerodynamic design optimization studies. In this chapter, results are presented for design improvement studies which utilize the SD's obtained from the parallel implementation of the SA code. Also in this chapter, the procedure for parallelizing the 1-D line search is described and results are presented which compare the parallel 1-D search with the sequential one. Chapter VII contains summary and conclusions from this work and also recommendations for future research. Appendix A contains tabulated results for the SDs for all of the design variables used in Chap. V. Finally, Appendix B contains tabulated results for the final vector of design variables from all of the design studies performed in Chap. VI.
CHAPTER II

FLOW FIELD ANALYSIS

2.1 Introduction

An advanced Computational Fluid Dynamics (CFD) code, CFL3D, was chosen to be equipped with Sensitivity Analysis (SA) capability in this study. Before implementing the Incremental Iterative Method (IIM) in this code for SA calculations, it was important to first carry out pure flow field analysis for the intended geometry and flight conditions. This is the focus of this chapter.

The CFL3D code can be used to solve three-dimensional, unsteady/steady, compressible Euler and thin-layer, Reynolds-averaged, Navier-Stokes equations in conservation law form. The governing equations, derived from the basic principles of conservation of mass, momentum and energy, are first written in Cartesian coordinates and then transformed into generalized body-fitted coordinates. The equations are solved by marching in time using a 3-factor, alternating-direction-implicit (ADI) algorithm. There are two options in the code for the upwind numerical approximation of the inviscid fluxes. The first is the Flux-Vector-Splitting (FVS) of van Leer [96] and the second is the Flux-Difference-Splitting (FDS) of Roe [97]. The solution in each sweep direction of the three factor-scheme is either by inversions of 5x5 block-tridiagonal matrices (applicable for FVS/FDS) or scalar tridiagonal matrices (applicable for FDS only). For Roe’s scheme, that is the FDS approach, the inviscid flux Jacobians are only approximate because they are obtained from similarity transformations; furthermore, the block-tridiagonal inversions are simplified using a diagonalizational algorithm resulting in the scalar tridiagonal option for the left hand side (LHS). This latter option for the Roe’s scheme is more suitable for steady-state computations. For time-accurate computations (not addressed in this work), it may be necessary to use subiterations to reduce diagonalization errors. Space-differencing of viscous terms is achieved using the finite-volume equivalent of central differences. Turbulence effects are accounted for using either an algebraic or two-equation eddy
viscosity model. A brief summary of the mathematical formulation follows. In the present study, only inviscid flows will be considered; hence formulations are presented only for the Euler equations. For a more detailed presentation of the governing equations and the discretization process, the reader is directed to the CFL3D Version 4.1 code description and input documentation manual [1].

**2.2 Governing Equations**

The basic governing equations are the three-dimensional, time dependent compressible Euler equations that express the conservation of mass, momentum and energy for an inviscid non-heat-conducting fluid in the absence of external body forces. In generalized curvilinear coordinates, the equations can be written in strong conservation law form using vector notation as:

\[
\frac{\partial \hat{Q}}{\partial t} + \frac{\partial \hat{F}}{\partial \xi} + \frac{\partial \hat{G}}{\partial \eta} + \frac{\partial \hat{H}}{\partial \zeta} = 0
\]

where the conserved state variables of vector \( \hat{Q} \) are defined as:

\[
\hat{Q} = \frac{1}{J} [\rho, \quad \rho u, \quad \rho v, \quad \rho w, \quad \rho e_o]^T
\]

\( u, v, w \) are the Cartesian components of the velocity, \( \rho \) is the density, and \( e_o \) is the specific total energy; i.e., \( e_o = e + \frac{1}{2} (u^2 + v^2 + w^2) \), where \( e \) is the thermodynamic specific internal energy.

The inviscid flux vectors \( \hat{F}, \hat{G} \) and \( \hat{H} \) are

\[
\hat{F} = \frac{1}{J} \begin{bmatrix}
\rho U \\
\rho U u + \xi_x p \\
\rho U v + \xi_y p \\
\rho U w + \xi_z p \\
(\rho e_o + p)U - \xi_p
\end{bmatrix}
\]

\[
\hat{G} = \frac{1}{J} \begin{bmatrix}
\rho U u + \xi_x p \\
\rho U v + \xi_y p \\
\rho U w + \xi_z p \\
\rho e_o + p \xi_p
\end{bmatrix}
\]

\[
\hat{H} = \frac{1}{J} \begin{bmatrix}
\rho U u + \xi_x p \\
\rho U v + \xi_y p \\
\rho U w + \xi_z p \\
(\rho e_o + p)U - \xi_p
\end{bmatrix}
\]
In the above equations, the contravariant velocities $U$, $V$, and $W$ for a fixed grid are given by
\[ U = u\xi_x + v\xi_y + w\xi_z \]
\[ V = u\eta_x + v\eta_y + w\eta_z \]
\[ W = u\zeta_x + v\zeta_y + w\zeta_z \]
and the determinant of the Jacobian matrix of the transformation for Cartesian to generalized coordinates is
\[
J = \frac{\partial(x, y, z)}{\partial(\xi, \eta, \zeta)} \]
\[
= \left[ x_\xi y_\eta \zeta_\zeta + x_\xi y_\zeta \zeta_\eta + x_\eta y_\zeta \zeta_\xi - x_\xi y_\zeta \zeta_\eta - x_\eta y_\zeta \zeta_\xi - x_\zeta y_\eta \zeta_\xi \right]^{-1}
\]
\[ p = (\gamma - 1)\left[\rho e_o - \frac{1}{2}\rho (u^2 + v^2 + w^2)\right] \]  \hspace{1cm} (2.8)

### 2.3 Time Integration

In terms of the steady state residual, Eq. (2.1) can be written as

\[ \frac{\partial \hat{Q}}{\partial t} = - R(Q) \]  \hspace{1cm} (2.9)

Using Euler implicit time integration and linearization, Eq. (2.9) becomes

\[ \left[ \frac{I}{\Delta t} + \frac{\partial R}{\partial Q} \right]^n \Delta Q = - R(Q^n) \]  \hspace{1cm} (2.10)

In terms of difference operators, Eq. (2.10) can be expressed as

\[ \left[ \frac{I}{\Delta t} + \delta_x \frac{\partial \hat{F}}{\partial Q} + \delta_\eta \frac{\partial \hat{G}}{\partial Q} + \delta_z \frac{\partial \hat{H}}{\partial Q} \right]^n \Delta Q = - \left[ \delta_x \hat{F} + \delta_\eta \hat{G} + \delta_z \hat{H} \right] = - R(Q^n) \]  \hspace{1cm} (2.11)

Applying the spatially-split approximate factorization to the above equation yields the following three equations which are solved in the given order to obtain \( \Delta Q \).

\[ \left[ \frac{I}{\Delta t} + \delta_x \frac{\partial \hat{F}}{\partial Q} \right]^n \Delta Q^* = - R^n \]  \hspace{1cm} (2.12)

\[ \left[ \frac{I}{\Delta t} + \delta_\eta \frac{\partial \hat{G}}{\partial Q} \right]^n \Delta Q^{**} = \left( \frac{I}{\Delta t} \right) \Delta Q^* \]  \hspace{1cm} (2.13)

\[ \left[ \frac{I}{\Delta t} + \delta_z \frac{\partial \hat{H}}{\partial Q} \right]^n \Delta Q = \left( \frac{I}{\Delta t} \right) \Delta Q^{**} \]  \hspace{1cm} (2.14)

The solution vector \( Q^{n+1} \) is obtained by the update step, that is

\[ Q^{n+1} = Q^n + \Delta Q \]  \hspace{1cm} (2.15)

The treatment of space-differencing for the Jacobians \( \frac{\partial \hat{F}}{\partial Q}, \frac{\partial \hat{G}}{\partial Q}, \) and \( \frac{\partial \hat{H}}{\partial Q} \) and the terms of the residual \( R \) are discussed in the next section.
2.4 Space Discretization

2.4.1 Residual Discretization

Equation (2.1) can be interpreted as describing the balance of mass, momentum, and energy over an arbitrary control volume. In this regard, the vectors \( \nabla \xi / J, \nabla \eta / J, \nabla \zeta / J \) represent directed areas of cell interfaces normal to the contravariant \( \xi = \text{constant}, \eta = \text{constant}, \text{and} \zeta = \text{constant} \) directions respectively. The Jacobian \( J \) represents the inverse of the cell volume. The advantage of the finite volume approach is that it remains valid in the presence of discontinuities in the flow, such as shocks and contact surfaces. Application of the integral conservation–law form of Eq. (2.1) to a control volume centered at grid point \( i, j, k \) and bounded by lines of constant \( \xi, \eta, \text{and} \zeta \) yields

\[
\begin{align*}
\frac{\partial \rho}{\partial t} & + 
\left( \hat{F}_{i+1/2,j,k} - \hat{F}_{i-1/2,j,k} \right) \\
& + \left( \hat{G}_{i,j+1/2,k} - \hat{G}_{i,j-1/2,k} \right) \\
& + \left( \hat{H}_{i,j,k+1/2} - \hat{H}_{i,j,k-1/2} \right) = 0
\end{align*}
\]  

(2.16)

Thus the residual \( R(Q^n) \) in Eq. (2.11) at time level \( n \) can be written as

\[
R(Q^n) = \left[ \delta_\xi \hat{F} + \delta_\eta \hat{G} + \delta_\zeta \hat{H} \right]^n
\]

\[
= \hat{F}_{i+1/2,j,k} - \hat{F}_{i-1/2,j,k} \\
+ \hat{G}_{i,j+1/2,k} - \hat{G}_{i,j-1/2,k} \\
+ \hat{H}_{i,j,k+1/2} - \hat{H}_{i,j,k-1/2}
\]  

(2.17)

The inviscid flux vectors in the above equation are handled by higher–order upwind differencing using either the flux–vector splitting of van Leer [96] or the flux–difference splitting of Roe [97].

2.4.1.1 van Leer Flux Vector Splitting

Using the van Leer flux–vector splitting in Eq. (2.17), the flux balance in the \( \xi \)–direction across a cell centered at point \((i,j,k)\) can be written as (dropping the \( j \) and \( k \) subscripts)
\[
\hat{F}_{i+1/2} - \hat{F}_{i-1/2} = \delta_{\xi} \hat{F}^+ + \delta_{\xi} \hat{F}^-
\]
\[
= \left[ \hat{F}^+ (Q^-) + \hat{F}^- (Q^+) \right]_{i+1/2} - \left[ \hat{F}^+ (Q^-) + \hat{F}^- (Q^+) \right]_{i-1/2}
\]

Conserved variables \( Q^- \) and \( Q^+ \) are evaluated at cell interfaces by upwind-biased interpolation from cell centers (similar to MUSCL-type differencing; MUSCL stands for Monotone Upstream-Centered Schemes for Conservation Laws). That is
\[
Q^-_{i+1/2} = Q_i + \frac{\phi_{\xi}}{4} \left[ (1 - x_\xi) \nabla_\xi + (1 + x_\xi) \Delta_\xi \right] Q_i
\]
\[
(2.19)
\]
and
\[
Q^+_{i+1/2} = Q_{i+1} - \frac{\phi_{\xi}}{4} \left[ (1 - x_\xi) \Delta_\xi + (1 + x_\xi) \nabla_\xi \right] Q_{i+1}
\]
\[
(2.20)
\]
where
\[
\Delta_\xi Q_i = Q_{i+1} - Q_i, \quad \nabla_\xi Q_i = Q_i - Q_{i-1}
\]
\[
(2.21)
\]
The value of \( \phi \) determines whether extrapolation is first order (\( \phi = 0 \)) or higher-order (\( \phi = 1 \)). Spatial accuracy is determined by the value of \( x \). \( x = -1 \) is second-order accurate fully upwind; \( x = 1/3 \) is third-order accurate upwind-biased (less than third-order accurate for multidimensional computations); and \( x = 1 \) is equivalent to a second-order accurate central differencing.

With the flux-vector splitting of van Leer in generalized coordinates, the forward flux \( \hat{F}^+ \) and the backward flux \( \hat{F}^- \) depend on the contravariant Mach number, \( M_\xi = \frac{\bar{u}}{a} \), in the \( \xi \)-direction. For supersonic flow (\( |M_\xi| \geq 1 \)),
\[
\hat{F}^+ = \hat{F}, \quad \hat{F}^- = 0 \quad (M_\xi \geq +1)
\]
\[
(2.22)
\]
\[
\hat{F}^- = \hat{F}, \quad \hat{F}^+ = 0 \quad (M_\xi \leq -1)
\]
\[
(2.23)
\]
For subsonic flow (\( |M_\xi| < 1 \)),

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
\[
\hat{F}^\pm = \frac{|\nabla(\xi)|}{J} \begin{bmatrix}
    f_{\text{mass}}^\pm \\
    f_{\text{mass}} ^\pm \begin{bmatrix} \hat{\xi}_x(-\bar{u} \pm 2a)/\gamma + u \\ \hat{\xi}_y(-\bar{u} \pm 2a)/\gamma + v \\ \hat{\xi}_z(-\bar{u} \pm 2a)/\gamma + w \end{bmatrix} \\
    f_{\text{energy}}^\pm
\end{bmatrix}
\]  

where

\[
f_{\text{mass}}^\pm = \pm \rho a \left( M_\xi \pm 1 \right)^2/4
\]  

\[
f_{\text{energy}}^\pm = f_{\text{mass}}^\pm \left\{ \begin{array}{l}
\left[-(\gamma - 1)\bar{u}^2 \pm 2(\gamma - 1)\bar{u}a + 2a^2\right]/(\gamma^2 - 1) \\
+ (u^2 + v^2 + w^2)/2
\end{array} \right.
\]  

The direction cosines \( \hat{\xi}_x, \hat{\xi}_y, \) and \( \hat{\xi}_z \) are given by

\[
\hat{\xi}_x = \xi_x/|\nabla \xi| \\
\hat{\xi}_y = \xi_y/|\nabla \xi| \\
\hat{\xi}_z = \xi_z/|\nabla \xi|
\]  

an \( \bar{u} \) is the velocity normal to a \( \xi = \) Constant face, i.e.

\[
\bar{u} = U/|\nabla(\xi)|
\]  

The fluxes in the other two directions are easily formed by interchanging \( \xi \) with \( \eta \) or \( \zeta \), respectively. The van Leer splitting ensures continuous differentiability at sonic and stagnation points; that is, the forward and the backward flux contributions blend smoothly at eigenvalue sign changes. Moreover, this method of handling the fluxes lead to sharp resolution of shocks. However it fails to resolve contact surfaces sharply, that is contacts are smeared [1].
2.4.1.2 Flux Difference Splitting of Roe

Using the flux difference splitting based on the approximate Riemann solver of Roe [97], the flux balance in the \( \xi \) - direction across a cell centered at point \((i, j, k)\) can be written as (dropping the \( j \) and \( k \) subscripts)

\[
\hat{F}_{i+1/2} - \hat{F}_{i-1/2} = \frac{1}{2} \left[ \hat{F}(q^-) + \hat{F}(q^+) - |A|(Q^+ - Q^-) \right]_{i+1/2} - \frac{1}{2} \left[ \hat{F}(q^-) + \hat{F}(q^+) - |A|(Q^+ - Q^-) \right]_{i-1/2}
\]

(2.29)

The definition of conserved state variables \( Q^- \) and \( Q^+ \) are as given in Eqs. (2.19) and (2.20). \( q^- \) and \( q^+ \) represent state variables on cell interfaces determined from upwind-biased interpolations of the primitive variables in a manner similar to Eqs. (2.19) and (2.20) for conserved variables.

For both the van Leer FVS approach and the Roe's FDS approach, the use of higher-order upwind formulas leads to numerical deficiencies, in particular, the generation of oscillations around discontinuities and in the presence of large flow gradients [98]. To resolve this problem so as to maintain monotonicity and eliminate spurious waves, flux limiters are often employed. In the CFL3D code, three types of limiters are provided. These are the smooth (i.e. differentiable) limiter of van Albada [99], the min–mod slope limiting of Chakravarthy and Osher [100] and the smooth limiter tuned to \( \kappa = 1/3 \) for third–order upwind–biased interpolation of state variables, where \( \kappa \) is as defined earlier.

2.4.2 Implicit Linearization Jacobians

In this subsection, the Jacobians used for linearization of the non–linear flow equations for both the van Leer FVS and the Roe's FDS schemes are presented.

2.4.2.1 van Leer Flux–Vector–Splitting

The implicit linearization Jacobians \( \partial \hat{F} / \partial Q \), \( \partial \hat{G} / \partial Q \) and \( \partial \hat{H} / \partial Q \) appearing on the right hand sides of Eqs. (2.12) – (2.14) are handled differently in the two schemes discussed above. For the van Leer approach, the Jacobians are obtained exactly by differentiating
directly the split-fluxes; thus they are split based on the fluxes. In other words, for the 
$\hat{F}^+$ and $\hat{F}^-$ for example, the split Jacobians will be $\hat{A}^+ = \partial \hat{F}^+ / \partial Q^-$ and $\hat{A}^- = \partial \hat{F}^- / \partial Q^+$. In addition, the space discretization of the split Jacobians are based on

the upwind differencing approach. This is made possible because $\hat{A}^+$ has nonnegative
eigenvalues and $\hat{A}^-$ has nonpositive eigenvalues. In addition, both Jacobians have one zero
eigenvalue for subsonic Mach numbers which leads to steady transonic shock structures with
only two transition zones [96]. From the foregoing, the solution from time step $n$ to time step
$n+1$ based on van Leer approach can be written in three sweeps as

$$
\left[ \frac{I}{\Delta t} + \delta^+ \hat{A}^+ + \delta^+ \hat{A}^- \right]^n \Delta Q^* = - R^n \tag{2.30}
$$

$$
\left[ \frac{I}{\Delta t} + \delta^- \hat{B}^+ + \delta^- \hat{B}^- \right]^n \Delta Q^{**} = \left( \frac{I}{\Delta t} \right) \Delta Q^* \tag{2.31}
$$

$$
\left[ \frac{I}{\Delta t} + \delta^+ \hat{C}^+ + \delta^+ \hat{C}^- \right]^n \Delta Q = \left( \frac{I}{\Delta t} \right) \Delta Q^{**} \tag{2.32}
$$

Both the backward difference operator $\delta^-$ and the forward difference operator $\delta^+$
operate on $\Delta Q$. The computational module for the left hand side of Eq. (2.30) is shown in
Fig. 2.1 for the $\xi$ sweep.

From Fig. 2.1, it can be seen that the solution at each point is directly coupled to the two
neighboring points. Hence the scheme requires the solution of a system of block tridiagonal
matrices. Similarly, the other two factors also require block tridiagonal inversions. Since the
tridiagonal systems are decoupled, the entire solution for a given sweep can be vectorized
over the number of lines in a plane times the number of planes taken.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
2.4.2.2 Roe Flux-Difference-Splitting

For the Roe scheme, the Jacobians $\frac{\partial F}{\partial Q}$, $\frac{\partial G}{\partial Q}$ and $\frac{\partial H}{\partial Q}$ are handled in two ways, the first leading to systems of block tridiagonal matrices and the second leading to systems of scalar tridiagonal matrices in each sweep. For both approaches, the Jacobians are approximated by using similarity transformation which is made possible by the mathematical nature of the Euler equations. The inviscid fluxes are homogeneous functions of degree one of the conservative state variable vector $Q$. Hence, for flux $\hat{F}$ for example, one can write

$$\hat{F} = \hat{A}Q \tag{2.33}$$

Therefore, the Jacobian $\hat{A}$ can be written as

$$\hat{A} = \frac{\partial \hat{F}}{\partial Q} = \frac{\partial \hat{F}^+}{\partial Q} + \frac{\partial \hat{F}^-}{\partial Q} = \hat{A}^+ + \hat{A}^- \tag{2.34}$$

Using similarity transformation, $\hat{A}^+$ and $\hat{A}^-$ are given by

---

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
\[
\begin{align*}
\hat{A}^+ &= TA^+T^{-1} \\
\hat{A}^- &= TA^-T^{-1}
\end{align*}
\] (2.35)

where
\[
A^\pm = \frac{A \pm \sqrt{A^2}}{2}
\] (2.36)

are diagonal matrices formed from the eigenvalues of \( \hat{A} = \partial \hat{F}/\partial Q \), that is
\[
A = \text{diag}(\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5)
\] (2.37)

and
\[
\begin{align*}
\lambda_1 &= \lambda_2 = \lambda_3 = \frac{\hat{U}|\nabla \xi|}{f} \\
\lambda_4 &= \frac{(\hat{U} + \hat{a})|\nabla \xi|}{f}, \quad \lambda_5 = \frac{(\hat{U} - \hat{a})|\nabla \xi|}{f}
\end{align*}
\] (2.38)

The contravariant velocity normal to the cell interface is
\[
\hat{U} = \frac{\xi_u u + \xi_v v + \xi_w w}{|\nabla \xi|}
\] (2.39)

and the symbol \( \hat{\cdot} \) indicates a Roe-averaged variable.

Using \( \hat{A}^+ \) and \( \hat{A}^- \) as defined in Eq. (2.35) will lead to block tridiagonal coefficient matrix on the LHS of Eq. (2.30). For the diagonalized algorithm using Roe's scheme, the LHS of Eq. (2.30), that is the \( \xi^- \) sweep of the ADI method, can be written, using Eq. (2.35) as
\[
\begin{align*}
\left[ \frac{I}{J\hat{\Delta}t} + \delta_\xi \frac{\partial \hat{F}}{\partial Q} \right]^n \Delta Q^* &= T \left[ \frac{I}{J\hat{\Delta}t} + \delta_\xi^- A^+ + \delta_\xi^+ A^- \right] T^{-1} \Delta Q^*
\end{align*}
\] (2.40)

The \( \xi^- \) sweep then becomes
\[
\begin{align*}
\left[ \frac{I}{J\hat{\Delta}t} + \delta_\xi^- A^+ + \delta_\xi^+ A^- \right] [T^{-1} \Delta Q^*] &= - T^{-1} R
\end{align*}
\] (2.41)

Due to the repeated eigenvalues ( \( \lambda_1 = \lambda_2 = \lambda_3 \) ), only three scalar tridiagonal LU decompositions are required for each line. The tridiagonal matrix equation can be written as
\[-A^+(M_{i-1/2}, \, Q_{i-1}) \left(T^{-1} \Delta Q^*\right)_{i-1} \]
\[+ \left[ \frac{1}{\Delta t} A^+(M_{i+1/2}, \, Q_{i}) - A^-(M_{i-1/2}, \, Q_{i}) \right] \left(T^{-1} \Delta Q^*\right)_{i} \]
\[+ A^-(M_{i+1/2}, \, Q_{i+1}) \left(T^{-1} \Delta Q^*\right)_{i+1} = - T_i^{-1} R_i \]  

(2.42)

The metric terms, $M$, are evaluated at cell faces. The state variables, $Q$, are evaluated at cell centers. The other two sweeps are handled in a similar manner. For viscous calculations (not addressed in this work), a spectral radius scaling of the viscous Jacobian matrix $H_v$ is employed.

### 2.5 Initial and Boundary Conditions

Since Eqs. (2.12) – (2.14) are solved by numerical time advancement, a set of initial conditions are required to start the time integration process. In this study, initial conditions are set to be the free stream conditions. If there had been a previous run, then initial conditions are read from the restart file.

The CFL3D code supports a wide range of boundary conditions (BCs). These include freestream for supersonic inflow, extrapolation for supersonic outflow, inflow/outflow (based on locally one-dimensional Riemann invariants) for far field subsonic inflow or outflow boundaries, symmetry plane, zonal interface (1–1, patched, chimera or embedded), inviscid surface (tangency), viscous surface (no slip), tunnel flow and singular axis. Other types of boundaries supported include specified pressure ratio outflow boundary with other flow variables obtained by extrapolation from inside the computational domain, specified engine inflow conditions and specified mass-flow coefficient. In this work, the first five types of BCs were used, as needed, for the flow problems of interest.

### 2.6 Flow Analysis Results

Steady inviscid flow was computed for two test cases to initially verify the CFL3D (and some of its advanced capabilities such as patched grid implementation) on the specific flow problems of interest.
2.6.1 Test Case 1: Flow over a Generic HSCT Wing/Body

The first test case comprises flow computation around the wing–body configuration of a generic High Speed Civil Transport (HSCT)–like geometry; i.e., the HSCT 24E developed at NASA Langley Research Center. This geometry, which has been used in previous design optimization studies [36, 88] is represented by simplified numerical descriptions of the configuration components in a wave–drag, or Harris, format. The geometry processing code of [102] obtains data on the components from the numerical description file, and then intersects and fillets them into a blended continuous surface. The grid–generation code of [103] then computes a suitable CFD grid for the continuous surface. Figure 2.2 shows the filleted wing/body configuration together with the surface grid in the wake region. The grid in the wake region has direct significance for the patched grid implementation discussed later in this section. Some of the surfaces of a grid (symmetrical about the x–z plane y=0, each y–z plane at constant x, i.e. streamwise, location) for supersonic Euler computations are shown in Fig. 2.3. The half–space grid size was 37 streamwise × 49 circumferential × 15 normal points. This grid size was very coarse for the geometry under consideration, but was used so that the flow analysis results obtained here could be compared with results obtained from a different code in another study [36] using the same geometry, grid distribution and flow conditions.

For the purpose of computing geometric sensitivity derivatives, it was necessary to provide an adequate parametrization for this geometry. The parametric variables then become the design variables which are used as input in the numerical description of the aerodynamic surfaces. A detailed description of wing planform, thickness, camber and twist used to parametrize the HSCT 24E wing/body is given in [36]. In this work, the planform design variables are used for accuracy validation of the developed CFL3D.ADII sensitivity analysis code. Details are provided in Chap. IV. The flow conditions used for this test case are free stream Mach number, $M_\infty = 2.4$ and angle of attack, $\alpha = 1$ degree.
Fig. 2.2 HSCT 24E Filleted Wing/Body Configuration (with Surface Grid in the Wake Region)
Fig. 2.3 Some of the Surfaces for the HSCT 24E Supersonic Euler Grid

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Figure 2.4 shows one of the grid planes in the wake region of Fig. 2.2. It is apparent that there are geometric point mismatches on the k = 1 surface; hence the more direct 1-1 point boundary condition could not be enforced in this region. To resolve this problem, one of the advanced capabilities of the CFL3D code was employed. The CFL3D code has provision for handling a generalized patched-grid interface between separate grid blocks in a multiple block problem. The algorithm used to obtain the interpolation coefficients for the common patched-grid interface has been described fully in [104]. For Version 4.1 of the CFL3D code used in this study, a separate computer code (RONNIE) based on the above mentioned algorithm has been made available specifically to compute the required interface interpolation coefficients prior to using the actual CFL3D code.

Using the above capability, the computational grid shown in Fig 2.3 is split into two blocks at the circumferential station j=25, where the k=1 surface in the wake region can be treated as a common generalized patched-grid interface for the two blocks. The two new grid blocks (upper and lower) are shown in Figs. 2.5a and 2.5b. Figs. 2.6a and 2.6b shows a typical x = constant station in the wake region of blocks 1 and 2, respectively. The RONNIE code is then used to obtain the required interpolation coefficients for the two blocks.

The two-block arrangement described above is used later in Chap. IV to ascertain whether the newly developed CFL3D.AdII sensitivity code is capable of yielding accurate sensitivity derivatives even for complicated problems requiring multiple block definitions with generalized patched interfaces.

Figures 2.7 and 2.8 present comparison between the results of flow analysis obtained using the CFL3D code and the results from another code that was used in [36]. The "a" part of each figure is from the present study and the "b" part is from the previous study mentioned above. Figure 2.7 shows the upper surface (including the wake region) pressure contours while Fig. 2.8 shows the same result for the lower surface. It can be observed that there is close agreement between the two codes.
Fig. 2.4 One of the x=constant Grid Planes in the Wake Region. The Enlarged Portion shows the Geometric Point Mismatch.
Fig. 2.5 Upper and Lower Grid Blocks for the HSCT 24E Supersonic Euler Grid
Fig. 2.6 $x=$constant Planes from Upper and Lower Grid Blocks
Fig. 2.7 Upper Surface (including the Wake Region) Pressure Contours. (a) From Present Study; (b) From the Code used in [36]

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Fig. 2.8 Lower Surface (including the Wake Region) Pressure Contours. (a) From Present Study; (b) From the Code used in [36]
2.6.2 Test Case 2: Flow over a Proprietary HSCT Wing/Body

At some point in the course of this study, a proprietary HSCT wing/body configuration was provided by the McDonnell-Douglas Corporation for code validation and design studies using the new CFL3D.ADI shape sensitivity analysis code. This geometry is used later in Chaps. V and VI for sensitivity analysis and design optimization studies in a massively parallel computing environment. Also, this geometry serves as the second example used in Chap. IV for accuracy and efficiency studies of the new sensitivity analysis code.

Because the geometry for this second test case is proprietary property, a detailed description of the wing/fuselage configuration and the computational grid can not be provided here. It can only be stated that the geometry is represented by a single-block grid having dimensions $193 \times 33 \times 33$ (a total of 210,177 grid points). Steady inviscid flow was computed at flow conditions Mach number, $M_\infty = 2.4$ and angle of attack, $\alpha = 1.9$. It should be mentioned that this second test case has been used to validate several CFD codes. The test case has also been used to perform aerodynamic shape optimization studies with gradients of aerodynamic functions computed using the finite difference method. The integrated flow quantities $C_L$ and $C_D$ obtained from this study agree with the results from previous proprietary studies which cannot be reported here.
CHAPTER III
IMPLEMENTATION OF THE INCREMENTAL ITERATIVE
METHOD IN CFL3D VIA AUTOMATIC DIFFERENTIATION

3.1 Introduction

In this chapter, the implementation of the Incremental Iterative Method (IIM) for computing first order sensitivity derivatives of aerodynamic functions in the advanced flow solver CFL3D will be described. The sensitivity-enhanced CFL3D code will henceforth be referred to as CFL3D.ADEI. The IIM has been found to yield accurate, reliable consistently discrete sensitivity derivatives within the framework of some other CFD codes [35, 36, 105, 106] in a more efficient manner compared to the finite-difference approach. In Sec. 3.2, the fundamental first order sensitivity equations for discretized three-dimensional Euler equations in incremental iterative form are presented. In Sec. 3.3, the process of judicious utilization of automatic differentiation in the construction of the required differentiated modules of a typical aerodynamic sensitivity analysis code is highlighted. In Sec. 3.4, the computational and accuracy issues pertaining to using sensitivity codes obtained from automatic differentiation on both vector and scalar computers will be discussed. Section 3.5 presents the features of the new CFL3D.ADEI sensitivity code.

3.2 First Order Aerodynamic Sensitivity Equations

A detailed review of recent advances in first order sensitivity analysis for modern, nonlinear CFD software is provided in [40]. A detailed discussion of discrete aerodynamic sensitivity analysis using the incremental iterative method can be found in [35, 36]. Since one emphasis of this work is the implementation of the IIM method in CFL3D, only a brief review of the basic equations are given here. Recalling Eq. (2.9), which is repeated here for convenience, that is

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
At steady state, the flow physics reduce to

\[ R(Q(b), X(b), b) = 0 \]  

(3.2)

where \( R \) is the residual vector, \( Q \) is the vector of field variables, \( X \) is the computational grid, and \( b \) is the vector of design variables. It should be noted that \( R \) depends implicitly on \( b \) through \( Q \) and \( X \), and that \( R \) may also depend explicitly on \( b \). In a similar manner, the vector of aerodynamic output functions \( F \) is dependent on \( Q \), \( X \) and \( b \) as

\[ F = F(Q(b), X(b), b) \]  

(3.3)

All applicable terms in Eqs. (3.2) and (3.3) and all subsequent equations are evaluated at steady state, unless explicitly superscripted with an appropriate iteration index.

### 3.2.1 Basic Equations

The sensitivity derivatives of \( F \) with respect to \( b \) can be obtained either by the direct differentiation method or by the adjoint variable method. In the first approach, Eqs. (3.3) and (3.2) are differentiated directly with respect to \( b \) to yield, respectively, the matrix equations

\[ D F' = \frac{\partial F}{\partial Q} Q' + \frac{\partial F}{\partial X} X' + \frac{\partial F}{\partial b} \]  

(3.4)

\[ R' = \frac{\partial R}{\partial Q} Q' + \frac{\partial R}{\partial X} X' + \frac{\partial R}{\partial b} = 0 \]  

(3.5)

where \( D F' = \frac{dF}{db} \); \( R' = \frac{dR}{db} \); \( Q' = \frac{dQ}{db} \); and \( X' = \frac{dX}{db} \).

The matrix \( D F' \) contains the sensitivity derivatives of interest. Using the notations in [105, 106], the preceding superscript \( D \) denotes that the derivatives are obtained by the direct differentiation method. The matrix \( Q' \) represents the sensitivity derivatives of the field variables. The matrix \( X' \) represents the grid sensitivity terms which can be obtained either by finite difference or by quasi-analytical differentiation of the grid-generation code. The advantage of using the quasi-analytical method will be discussed later in Sec. (3.4.2).
should be noted that Eq. (3.5) is linear in $Q'$. To obtain $D F'$ from Eq. (3.4), it is first required to solve for $Q'$ in Eq. (3.5) and then make the required substitution in Eq. (3.4).

For the adjoint variable method, an alternative to solving for $Q'$ in Eq. (3.5) is employed. In this approach, an adjoint-variable matrix $A$ is introduced to combine Eqs. (3.4) and (3.5). The matrix $A$ is then specified to ensure that the resulting coefficients of $Q'$ vanish. The adjoint-variable method yields

$$
^{A}F' = \frac{\partial F}{\partial X'} + \frac{\partial F}{\partial b} + A^T \left( \frac{\partial R}{\partial X'} + \frac{\partial R}{\partial b} \right) \quad (3.6)
$$

$$
G \equiv \left( \frac{\partial R}{\partial Q} \right)^T A + \left( \frac{\partial F}{\partial Q} \right)^T = 0 \quad (3.7)
$$

The matrix $^{A}F'$ contains the sensitivity derivatives of interest. The superscript $A$ denotes that they are obtained by the adjoint variable method. It should be noted that both $D F'$ and $^{A}F'$ are equivalent; that is, $D F' = ^{A}F' = dF/db$. The very large linear system of Eq. (3.7) is first solved for $A$ which is then used in Eq. (3.6) to calculate the sensitivity derivatives $^{A}F'$.

The dimension of $b$ and, thus, the column dimension of $Q'$ is the number of design variables. The dimension of $F$ and, thus the column dimension of $A$ is the number of output functions. Therefore, if the number of design variables is greater than the number of output functions, then the solution of Eqs. (3.6) and (3.7) is likely to be computationally less expensive than that of Eqs. (3.4) and (3.5). However, for complicated advanced CFD codes like CFL3D applied to large scale problems, it is infeasible, due to memory limitations, to explicitly compute and store the large matrix $\frac{\partial R}{\partial Q}$ (which can then be transposed to obtain $(\frac{\partial R}{\partial Q})^T$) which is required in Eqs. (3.5) and (3.7). For the direct differentiation approach of Eq. (3.5), using an automatic differentiation tool like ADIFOR [55–57], it is fortuitous that the terms $(\frac{\partial R}{\partial Q})Q'$ and $(\frac{\partial R}{\partial X})X'$ are computed without explicit computation and storage of $\frac{\partial R}{\partial Q}$. In fact, application of automatic differentiation to CFD codes rests heavily on avoiding the explicit computation of matrices such as $\frac{\partial R}{\partial Q}$. Due to the nature of the AD
tools however, there is no provision for computing \((\partial R/\partial Q)^T A\) without first computing and storing \((\partial R/\partial Q)^T\), which, as stated, is an impossible task for large-scale problems using modern CFD codes. Hence for such cases, even if the number of design variables is greater than the number of output functions, the direct differentiation approach will nevertheless be employed in computing the required sensitivity derivatives.

Together, the above two standard methods for obtaining the sensitivity derivatives are known as the quasi-analytical methods. It should be noted that to obtain correct and accurate sensitivities, the Jacobian \(\partial R/\partial Q\) as well as \(\partial R/\partial X\) (evaluated at steady state) must include consistent linearization treatment of all boundary conditions. In addition, no approximations can be introduced into any of the terms without simultaneously introducing error into the resulting SDs. Also, given the choice of a higher-order-accurate upwind approximation for the spatial discretization of the flow physics, a consistent, higher-order-accurate, upwind spatial discretization, including a fully consistent treatment of all boundary conditions, is required in the coefficient-matrix operator of the sensitivity equations. The coefficient matrix, either \(\partial R/\partial Q\) or \((\partial R/\partial Q)^T\) of the linear sensitivity equations in standard form is not diagonally dominant [107]; consequently, the computational performance of traditional iterative methods for solving these equations in this standard form is expected to be poor or even fail [108]. A term like the time term \(I/J\Delta T\) in Eq. (2.10) which, in steady state computations primarily serves as a means of improving diagonal dominance of the LHS matrix; cannot be employed. Thus in the standard form, the framework to support the development of iterative methods is rigid and restrictive. All these computational problems have motivated the development of the incremental iterative method described in [35, 36].

3.2.2 The Incremental Iterative Solution Method (IIM)

Recall Eq. (2.10); that is

\[
\left[ \frac{I}{J\Delta t} + \frac{\partial R}{\partial Q} \right] \Delta Q = - R^n
\]  

(3.8)
As observed earlier in Sec. 2.3, Eq. (3.8) represents the fundamental implicit formulation for integrating the Euler equations in time to steady state. The equation is usually called the "delta" form or the "incremental iterative" form. The time-term matrix \( I/J \Delta T \) is diagonal. The large Jacobian matrix \( dR/dQ \) is sparse and has a banded structure. In addition to its use in Eq. (3.8), this important Jacobian matrix plays another central role in the development of the Incremental Iterative Method (IIIM) for sensitivity analysis.

In principle, Eq. (3.8) can be repeatedly solved directly as the solution is advanced to steady state. For very large time steps, the direct method represents Newton's root-finding procedure, where Eq. (3.8) reduces to

\[
\frac{\partial R}{\partial Q} \Delta Q = - R^n
\]  

(3.9)

However, the direct method is not necessarily the most efficient as pointed out in [109], and the large storage requirements of the method make its use not feasible for realistic three-dimensional problems. Therefore, more commonly, an iterative algorithm is selected for use in the repeated solution of Eq. (3.9). Some popular choices of these iterative algorithms include approximate factorization (AF) [1, 2, 5], conventional relaxation algorithms [107, 110], the strongly implicit procedure [111], and the preconditioned conjugate gradient-like methods such as GMRES [33, 112, 113]. For CFL3D used in this study, solution is advanced in time using the spatially-split, three-factor, Alternating Direction Implicit (ADI) AF method.

In many CFD codes, as the case with steady state computations in CFL3D, the \( dR/dQ \) term in the left hand side coefficient matrix of Eq.(3.9) is usually replaced with a convenient iteratively convergent approximation denoted here as \( \hat{dR}/\hat{dQ} \). With this modification, the pure Newton iteration becomes what is sometimes called the quasi-Newton iteration. Thus Eq. (3.9) can be re-written as

\[
\frac{\hat{dR}}{\hat{dQ}} \Delta Q = - R^n
\]  

(3.10)
\[ Q^{n+1} = Q^n + \Delta Q \]  

(3.11)

The left-hand-side coefficient matrix operator \( \partial R / \partial Q \) in Eq. (3.10) is, in many CFD codes, at best only a rough approximation to the exact Jacobian matrix operator that is associated with the true Newton iteration. Thus Eqs. (3.10) and (3.11) are representative of the broad spectrum of iterative algorithms (either implicit or explicit), that are common to CFD software. In CFL3D, such approximations include the use of the ADI algorithm, local time stepping, diagonalized LHS matrix, mesh sequencing, multigrid, etc.

As discussed previously herein and also in [35, 36, 106], there are numerous numerical difficulties associated with solving the sensitivity equations (Eq. (3.5) for direct method or Eq. (3.7) for adjoint variable method) in standard form. Previous studies [35, 36, 76, 77] have shown that these computational difficulties can be overcome, at least in part, by iteratively solving these equations in incremental iterative form. For the direct differentiation method (Eq. (3.5)), the incremental iterative method is cast in a manner similar to Eqs. (3.10) and (3.11) for fluid flow analysis; that is

\[ \frac{\partial R}{\partial Q} \Delta Q' = - R'^m \]  

(3.12)

\[ Q'^{m+1} = Q'^m + \Delta Q' \]  

(3.13)

where

\[ R'^m = \frac{\partial R}{\partial Q} Q'^m + \frac{\partial R}{\partial X} X' + \frac{\partial R}{\partial b} \]  

(3.14)

In Eq. (3.12), the LHS coefficient matrix \( \partial R / \partial Q \) represents any convergent, computationally convenient approximation of the exact Jacobian matrix. In particular, the identical approximate LHS operator and algorithm that are used to solve the nonlinear flow equations, Eq. (3.10), can also be used to solve the linear sensitivity equations, Eq. (3.10). Thus for the CFL3D code considered in this work, the implementation of the IIM is in three sweeps (following the three sweeps of the original non-linear flow analysis), that is (recall Eqs. (2.12), (2.13) and (2.14))
After obtaining $A Q'$, Eq. (3.13) is used to update $Q'$. $R^n$ is obtained using Eq. (3.14).

The similarity between Eqs. (3.15) – (3.17) with Eqs. (2.12) – (2.15) is apparent. The only difference is that $R^n$ at time level $n$ in the non-linear flow equations is replaced with $R'^{m}$ in the linear sensitivity equations. The left hand side (LHS) coefficient matrices remain constant in the sensitivity equations since sensitivity analysis is performed after the flow analysis has reached steady state. As will be discussed subsequently, for large problems, it may be impossible to store the LHS coefficient matrices for reuse during SA iterations due to memory limitations. In such cases, they will have to be recomputed. At convergence, the accuracy of the computed sensitivity derivatives is not compromised because the terms in $R'^{m}$ are evaluated in a manner consistent with the discretization, i.e. the flux and boundary condition treatment, used in CFL3D. The IIM can also be used in the adjoint variable (AV) method to solve Eq. (3.7) as shown below. Note that the LHS operator $\frac{\partial R}{\partial Q}$ must be computed explicitly and transposed. The IIM for the AV method becomes

\[
\begin{align*}
\left[ \frac{I}{J \Delta t} + \delta_{x} \frac{\partial F}{\partial Q} \right] \Delta Q'^{**} &= - R'^{m} \\
\left[ \frac{I}{J \Delta t} + \delta_{\eta} \frac{\partial G}{\partial Q} \right] \Delta Q'^{***} &= \left( \frac{I}{J \Delta t} \right) \Delta Q'^{*} \\
\left[ \frac{I}{J \Delta t} + \delta_{\zeta} \frac{\partial H}{\partial Q} \right] \Delta Q' &= \left( \frac{I}{J \Delta t} \right) \Delta Q'^{**} 
\end{align*}
\]

(3.15)  
(3.16)  
(3.17)

After obtaining $\Delta Q'$, Eq. (3.13) is used to update $Q'$. $R'^{m}$ is obtained using Eq. (3.14).

The similarity between Eqs. (3.15) – (3.17) with Eqs. (2.12) – (2.15) is apparent. The only difference is that $R^n$ at time level $n$ in the non-linear flow equations is replaced with $R'^{m}$ in the linear sensitivity equations. The left hand side (LHS) coefficient matrices remain constant in the sensitivity equations since sensitivity analysis is performed after the flow analysis has reached steady state. As will be discussed subsequently, for large problems, it may be impossible to store the LHS coefficient matrices for reuse during SA iterations due to memory limitations. In such cases, they will have to be recomputed. At convergence, the accuracy of the computed sensitivity derivatives is not compromised because the terms in $R'^{m}$ are evaluated in a manner consistent with the discretization, i.e. the flux and boundary condition treatment, used in CFL3D. The IIM can also be used in the adjoint variable (AV) method to solve Eq. (3.7) as shown below. Note that the LHS operator $\frac{\partial R}{\partial Q}$ must be computed explicitly and transposed. The IIM for the AV method becomes

\[
\begin{align*}
\left( \frac{\partial R}{\partial Q} \right)^T \Delta A &= G^m \\
A^{m+1} &= A^m + \Delta A 
\end{align*}
\]

(3.18)  
(3.19)

where

\[
G^m = \left( \frac{\partial R}{\partial Q} \right)^T A^m + \left( \frac{\partial F}{\partial Q} \right)^T
\]

(3.20)

and the superscript T indicates a matrix transpose.
3.3 Automatic Differentiation (AD) and the Incremental Iterative Method (IIM)

A CFD code can be differentiated in black-box mode using ADIFOR, as described subsequently. ADIFOR stands for Automatic Differentiation of FORtran and its description and usage can be found in [55–57]. When ADIFOR is applied to a code, the resulting code is capable, upon compilation and execution, of yielding, up to machine precision, the numerical value of the derivative of a specified output function with respect to a specified input variable. The new code performs function evaluation as well.

Conceptually, according to [56, 114], the basic CFD flow solution procedure, by combining Eqs. (3.10) and (3.11), can be written as

\[ Q^{n+1}_n = Q^n - P^n R^n; \quad n = 1, 2, 3, \ldots \tag{3.21} \]

where \( P^n = \left( \frac{\partial R^n}{\partial Q} \right)^{-1} \).

Differentiating Eq. (3.21) with respect to \( b \) in black-box mode yields

\[ Q'^{n+1} = Q'^n - P^n R'^n - P'^n R^n; \quad n = 1, 2, 3, \ldots \tag{3.22} \]

Within the framework of steady state sensitivity analysis, the black-box differentiation via ADIFOR of a CFD code has the following disadvantages [37, 105, 106]:

1. At steady state, \( R^n = 0 \). Hence the computation of \( P^n, R^n \) and their product \( P'^n R^n \) in Eq. (3.22) is unnecessary. With the black-box implementation, these unwanted computations are unavoidable and thus a significant amount of CPU time is wasted.

2. At steady state, \( P^n \) is constant. Hence, for the steady state sensitivity analysis, \( P^n \) needs to be computed only once (i.e., after the flow analysis has converged), and then stored in memory. Hence, there is a trade-off between CPU time and memory. If the problem size is small enough for \( P^n \) to fit into the computer memory, then it should be computed once and then stored. Otherwise, it needs to be recomputed at every iteration. If it is possible to "freeze" \( P^n \), CPU time will be saved in two ways. The first
is from $P^n$ itself and second is from $P''^n$. This is because, due to the nature of ADIFOR, $P''^n$ is computed only when $P^n$ is computed. If $P^n$ is computed only once, then $P''^n$ also will be computed only once. The process of reusing $P^n$ already stored in memory is not an uncommon method in CFD. Some CFD codes, in what is called the "frozen Jacobian" method essentially does this by keeping $P^n$ constant for a predetermined number of iterations, especially when the flow analysis solution is close to steady state convergence. With such codes, during sensitivity analysis computation, the "frozen Jacobian" option can be turned on permanently. For codes without this option, some reprogramming can be done to accommodate this. For large three dimensional problems, there usually is not sufficient memory to store $P^n$; hence $P^n$ needs to be recalculated at every iteration. The effect of using the "frozen Jacobian" option on the CPU time required for sensitivity analysis from CFD codes have been documented in [106].

(3) Typically, sensitivity analysis is performed after the flow analysis has converged to steady state. Thus it will be expected that the convergence rate can be accelerated faster than for the flow analysis. But due to the nature of ADIFOR, the differentiated code continues to iterate on the nonlinear flow equations; thus the convergence rate will have to be maintained at a level permitted by the stability restrictions of the flow analysis.

(4) From Eqs. (3.12)-(3.14) for sensitivity analysis, the only term of $R''^m$ that needs to be inside the iteration loop is $(\partial R/\partial Q)(Q''^m)$. However, with black-box differentiation, all of the terms of $R''^m$ are forced to be inside the loop. CPU time is wasted in doing this.

(5) For vector supercomputers, the black-box automatic differentiation may cause the differentiated code to lose some of the vectorization property of the original code. As opposed to the black-box approach depicted by Eq. (3.22), the incremental iterative method (IIM), by combining Eqs. (3.12) and (3.13), can be written as
\[ Q'^{m+1} = Q'^{m} - P R'^{m}, \quad m = 1, 2, 3, \ldots \]  

At steady state where \( R^n = 0 \) and \( P'^{n} R^n \) vanishes, Eqs. (3.22) and (3.23) are identical. However, the computational work for the two equations are different. For an IIM implementation where hand-differentiation is employed, the most efficient code results which avoids the disadvantages of the black-box approach discussed previously. However, for sophisticated and complicated advanced CFD codes, it is impractical to construct a sensitivity analysis (SA) code purely by hand differentiation. That is, a hybrid scheme is sought which exploits the strengths of each method, yet mitigate the weaknesses. The answer lies in a judicious application of ADIFOR as opposed to total black-box application. This approach forms the backbone of the studies in [37, 38, 105, 106, 115] and also the present study.

The approach involves using ADIFOR to differentiate only the right hand side (RHS) of Eq. (3.10) which is the residual that governs the flow physics. Thus automatic differentiation is now employed to differentiate the RHS of Eq. (3.10); the results are then assembled to create the RHS of Eq. (3.14). Doing this will eliminate most of the drawbacks of the black-box approach, previously described; in particular, the unwanted continuous computation of \( P'^{n} \) at every iteration is eliminated. However, due to the nature of ADIFOR, the continuous repeated computation of \( R^n \), unfortunately, can not be eliminated. During the construction of \( R'^{m} \) with ADIFOR, care should be taken to ensure that only the term \( (\partial R/\partial Q)(Q'^{m}) \) is placed inside the sensitivity analysis iteration loop. All other terms should be computed outside this loop. Also, care should be taken to include all necessary terms such as boundary condition routines and interior-cell residual routines in the construction of \( R'^{m} \), in order to obtain accurate sensitivity derivatives. While assembling the overall hybrid ADIFOR/IIM (ADII) code, a vectorization study should be performed to identify and correct loops that vectorize in the original code but do not vectorize in the new code. The result of the above efforts will be a new sensitivity analysis code that is as efficient as possible, though not as efficient as a hand-differentiation/IIM code.
It was noted previously that if the number of design variables is larger than the number of
the output functions, the adjoint variable formulation of Eqs. (3.6) and (3.7) (standard form)
or Eqs. (3.18) and (3.19) (incremental iterative form) may be a better strategy compared to
the direct–differentiation approach of Eqs. (3.4) and (3.5) or Eqs. (3.10) and (3.11).
However, for complicated advanced CFD codes, the direct–differentiation formulation may
always be employed even if the number of design variables is larger than the number of
output functions. This is because, to be able to use the adjoint variable formulation, the very
large transposed Jacobian matrix $(\partial R/\partial Q)^T$ needs to be explicitly computed and
postmultiplied by the adjoint variable matrix $A$ to obtain the term $(\partial R/\partial Q)^T A$ in Eq. (3.7) or
Eq. (3.18). For modern CFD codes, it is infeasible to explicitly compute and store $\partial R/\partial Q$ and
then transpose it to obtain $(\partial R/\partial Q)^T$ because of the extremely large memory required.
Unfortunately, due to the nature of ADIFOR, the term $(\partial R/\partial Q)^T A$ cannot presently be
constructed via automatic differentiation. The reason is because ADIFOR operates only in
the forward mode and $\partial R/\partial Q$ and also $\partial R/\partial X$ are never explicitly constructed. In the
direct–differentiation approach of Eq. (3.5) or (3.14), the terms $(\partial R/\partial Q)Q'$ and $(\partial R/\partial X)X'$
are constructed without the explicit calculation of the very large Jacobian matrices $\partial R/\partial Q$
and $\partial R/\partial X$, respectively, and without explicit postmultiplication by the matrices $Q'$ or $X'$,
respectively. Of course, the AD–enhanced code, which can evaluate these complete
expressions, will require increased memory over that of the original code. However, this
increase is approximately equal only to the memory of the original code times the column
dimension of $Q'$ or $X'$. For the present application, this is NDV, which is the dimension of $b$
(or the dimension of that fraction of $b$ for which SDs are to be concurrently calculated via the
ADII method). The final result is an extremely fortuitous conservation of computer memory.
Without this conservation of memory, given the overwhelming size of $\partial R/\partial Q$ and $\partial R/\partial X$,
the application of ADIFOR to advanced CFD codes would be infeasible. Despite these
positive features with respect to computer memory, one should note that the CPU time
associated with each repeated evaluation of \( \frac{\partial R}{\partial Q}Q'^m \) via AD-generated code will be significantly larger than that which could be achieved (in principle) via hand differentiation and an efficient, hand-coded procedure for evaluation of these same terms.

### 3.4 Computational Issues

In this section, the computational issues relevant to the usage of sensitivity analysis codes obtained from ADIFOR on vector and scalar computers are discussed.

#### 3.4.1 Vector Computers

Prior to the compilation and execution of any AD-enhanced FORTRAN source code, a parameter \( g_{p} \) is specified within the code. For each execution of the code, this parameter determines the number of independent (design) variables with respect to which derivatives are concurrently computed. Thus, the user has the following options [105, 106]:

1. Compute all required derivatives by executing the AD-enhanced code once for each independent variable (i.e., NDV code executions with \( g_{p} = 1 \)).

2. Compute all required derivatives by executing the AD-enhanced code only once (i.e., one code execution, with \( g_{p} = \text{NDV} \)).

3. Set \( g_{p} \) such that \( 1 \leq g_{p} \leq \text{NDV} \); this requires multiple executions (less than NDV) of the AD-enhanced code, where subgroups of \( g_{p} \) derivatives are concurrently computed for each code execution.

The specified value of \( g_{p} \) has a significant impact on computational requirements in several critical ways. With respect to memory, for example, recall that the memory increase of the AD-enhanced code is approximately equal to \( g_{p} \) times the memory of the original code. Thus, if this parameter is too large, the memory requirements of the code could be excessive.

The AD-enhanced code retains all do-loops and function evaluations of the original code. Within each original do-loop is inserted one or more new innermost do-loops. The length of each new do-loop is \( g_{p} \) (e.g., \( \text{DO 10 I = 1, g_{p}} \)). Inside these new loops, derivative calculations are made. The presence of these new innermost do-loops has a
profound impact (frequently negative) on the vectorization characteristics for performance on vector computers, e.g. Cray supercomputers.

(1) The do-loops of the original code, which previously vectorized, will no longer be vectorized in the AD-enhanced version. An exception to this is when g_p_ \leq 5; the "aggressive" Cray compiler option will automatically "unwind" the new innermost loops and may restore the vectorization of the original loops, complete with the derivative calculations.

(2) For g_p_ \geq 6, vectorization of the original loops is not recovered, however, the new innermost loops are vectorized. Nevertheless, overall code performance remains poor on Cray computers unless g_p_ is large enough that the vector lengths become sufficiently long for efficient execution on these machines. At the same time, however, for large g_p_ the computer memory requirements of the AD-enhanced CFD software can become excessively large.

Apart from the vectorization considerations discussed above, the number of arithmetic operations per concurrently computed derivative is always decreased as g_p_ increases. This happens because, for each execution of an AD-enhanced code, part of the derivative calculations occur outside of the innermost loops, and the results are reused for all derivative calculations within the innermost loops. Furthermore, the complete function evaluations of the original code are performed only once (but, as needed, are thereafter used for derivative calculations within the innermost loops).

In [105, 106], the consequences discussed above have been demonstrated with several examples. In these references, it was observed that g_p_ = 5 produces the highest computational efficiency per design variable, and this efficiency is progressively reduced as g_p_ is reduced to 1. An exception to this is when g_p_ is much larger than 5 where the innermost DO-LOOPS are now long enough for useful vectorization. A particularly inefficient case is that of g_p_ = 6 (thereafter efficiency gradually increases as g_p_ increases). In the case with NDV = 6, rather than perform one code execution with g_p_ = 6,
two code executions, each with $g_p < 6$ (e.g., the first execution with $g_p = 5$ and the second execution with $g_p = 1$), were significantly more efficient.

### 3.4.2 Scalar Computers

For scalar computers, usually workstations computing in 32–bit single precision mode or 64–bit double precision mode, the issues associated with vectorization discussed above for vector computers are non-existent. However, the memory issues are also applicable to the scalar computers. The implication of this is that there is no restriction on the value of $g_p$ that can be set on scalar machines; that is, as many derivatives as the memory on the machine will allow can be computed concurrently.

During the present study, as will be shown later in Chap. IV, the finite–difference method is used to check the accuracy of the new AD–enhanced CFL3D (CFL3D.ADII) code. It was interesting to note that, in addition to the difficulty of choosing the right step size, the finite difference method can be extremely sensitive to machine precision. On the other hand, the AD–enhanced code appears to suffer no significant loss of accuracy with respect to the precision of the machine used for computation. This phenomenon is illustrated in the following example where grid sensitivities (i.e. the $X'$ term in Eq. (3.14)) are computed by (i) using the finite difference method and (ii) from a grid generation code that has been differentiated using ADIFOR. Comparisons are made between 32–bit single–precision and 64–bit double–precision computations on a node of the IBM–SP2 (RS6000). It should be noted here that the RS6000 architecture of the IBM–SP2 is internally designed to compute in 64–bit precision, but if the double precision compiler option is not enforced, results are returned in 32–bit single precision mode. There is some CPU time overhead on the RS6000 if computations are done in single precision, but there is a saving of 50% in memory. Grid sensitivities are important in this study because only geometric shape design variables are considered in the design optimization studies that will be discussed later in Chap. VI.

The grid sensitivity results presented here were obtained from the AD–enhanced grid generator code for the proprietary HSCT–like wing–body configuration of Test Case 2...
which was described in Chap. II. A more detailed presentation of the parametrization used to
describe this geometry will be discussed later in Chap. IV. The purpose in this section is just
to illustrate how sensitive the finite difference method can be to machine precision. Results
are presented in Table 3.1 for grid sensitivities with respect to a single design variable for two
randomly selected grid points. The design variable is a multiplier for a mathematical
function that represents the wing twist at various wing spanwise stations. The step-size used
in the finite difference process is 0.01. This step-size is within the range used previously in
proprietary design improvement studies on a Cray machine.

From Table 3.1, it can be observed that there is no significant difference between the
single-precision results and the double-precision results for the AD-enhanced grid code
(rows 5 and 6). Also both agree very well with the finite difference results from the
double-precision calculations, that is row 1 for the first-order forward difference method
and row 3 for the second-order central difference method. In addition, for the step size and
the design variable considered, the forward finite difference method yields results that are as
good as the results from the central difference method in double-precision mode. However,
the finite difference results from single-precision calculations (rows 2 and 4) differ both in
sign and magnitude compared to results from the AD-enhanced code.

The first implication of the above realization is that it is quite possible that, for the grid
generation code used here, if care is not taken, the derivatives obtained using the finite
difference procedure may be completely wrong, especially if computations are performed in
32-bit precision as opposed to 64-bit precision. Moreover, the observation also places
greater emphasis on the difficulty that may be encountered in choosing the right-step size for
the finite difference method. The AD-enhanced code suffers none of these shortcomings.

Another implication is that the AD-enhanced code requires twice as much memory
using 64-bit precision compared to the memory required for 32-bit precision computations
without any improvement in accuracy. Thus a 32-bit machine will handle twice the problem
Table 3.1 Effects of Machine Precision on Grid Sensitivities from the Finite Difference Method and from an AD-Enhanced Grid Generation Code

<table>
<thead>
<tr>
<th>Grid Sensitivity</th>
<th>Grid Point (56,20,20)</th>
<th>Grid Point (95,15,15)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\frac{\partial x}{\partial DV}$</td>
<td>$\frac{\partial y}{\partial DV}$</td>
</tr>
<tr>
<td>Forward Finite Difference</td>
<td>DP*</td>
<td>-3.022E-4</td>
</tr>
<tr>
<td></td>
<td>SP**</td>
<td>4.883E-2</td>
</tr>
<tr>
<td>Central Finite Difference</td>
<td>DP</td>
<td>-3.021E-4</td>
</tr>
<tr>
<td></td>
<td>SP</td>
<td>6.104E-2</td>
</tr>
<tr>
<td>AD-enhanced Code</td>
<td>DP</td>
<td>-3.021E-4</td>
</tr>
<tr>
<td></td>
<td>SP</td>
<td>-3.022E-4</td>
</tr>
</tbody>
</table>

* DP = Double Precision
* * SP = Single Precision
size that a 64-bit machine will handle with the same memory requirement. This is particularly a big advantage on a machine like IBM-SP2 which stores results in 32-bit precision, even though computations are performed in 64-bit precision. For example, in this work, as will be demonstrated later, each node of the IBM-SP2 (with 128 MB of memory) was able to contain a problem size of over 200,000 grid points using the new CFL3D.ADII code where aerodynamic functions and their derivatives (for one design variable) were computed (for the 3-D Euler equations). If computations were to be performed while enforcing the 64-bit precision mode (i.e. double precision), it would not have been possible to fit this problem size on an IBM-SP2 node.

3.5 The CFL3D.ADII Shape Sensitivity Analysis Code

The IIM theory, the use of ADIFOR and the associated computational issues that have been discussed in the previous sections of these chapter have all been combined and implemented in the state-of-the-art, well known, general purpose CFL3D flow-analysis code. The resulting code is an efficient general-purpose code for geometric-shape flow sensitivity analysis. This code has since been referred to herein as CFL3D.ADII.

CFL3D.ADII can be used to obtain accurate gradient information for subsequent use in aerodynamic shape design optimization. To date, two versions of this new sensitivity analysis code have emerged. The first version, henceforth referred to as Version 1, was used for the validation studies presented later in Sec. 4.2 of Chap. IV. Prior to the present work, Version 1 has been found to yield accurate sensitivity derivatives for single grid block problems with computational efficiency significantly greater than that obtained via a black box AD of CFL3D [37, 115]. However, through further tests in this study which will be shown in Sec. 4.2, this first version, even though accurate, had disappointing computational efficiency compared to the performance of the original CFL3D flow analysis code, particularly on Cray computers where the vectorization of the new sensitivity code had some severe "bottlenecks".
The present work builds on the progress made in the first version of CFL3D.ADI. In this work, based on suggestions in [37], the second version has been developed with the following improvements:

(1) A careful study of Version 1 of CFL3D.ADI revealed that some critical subroutines that vectorized in the original CFL3D flow analysis code failed to vectorize in the CFL3D.ADI sensitivity analysis code. The reason for these was because during the process of applying ADIFOR to the affected subroutines, calls to some external exception error handling routines destroyed the vectorization of some of the critical DO-loops. This problem was corrected by replacing the calls to the external routines with suitable FORTRAN statements that will do the same work as the external routines without compromising the vectorization property of the affected loops.

(2) The second improvement attempt involves recoding the part of the code where the field variables $Q$ are being updated such that if the non-linear flow analysis is already well converged prior to sensitivity analysis, there will be no need to continue updating on $Q$, hence the entire $Q$ updating process can be bypassed. Doing this can reduce the computational work for the linear sensitivity analysis process in the following ways [37]:

(a) The computational cost associated with the update part of the nonlinear flow equations at each multigrid cycle will be saved. It should be noted, unfortunately, that due to the nature of the ADIFOR system of differentiation, the full cost of an iteration on the nonlinear equations cannot be saved. Only the iterative algorithmic operations on the nonlinear flow residuals can be deleted; the unwanted, repeated calculation of the nonlinear flow residuals will continue at each multigrid cycle.

(b) With the deletion of the update on the nonlinear flow equations, a significantly larger "time step" can be used to advance the solution of the linear sensitivity equations which should result in increased convergence rate. The above statement is based on the argument that the time step which is optimum for solving the linear sensitivity equations will most probably be too large when applied to the nonlinear flow equations due to more
stringent stability restrictions. However, based on observations in this work, it was not yet conclusive if the convergence of the CFL3D.ADII sensitivity analysis code can be accelerated with a time step larger than the final time step used for solving the nonlinear flow equations.

It should be mentioned that CFL3D.ADII (Version 2) is fully compatible with the original CFL3D (Version 4.1) flow code. All features and capabilities of the CFL3D code have been included in the CFL3D.ADII code; the only exception is the unsteady flow capabilities. That is, unsteady aerodynamic sensitivity derivatives cannot yet be calculated. However, all of the remaining powerful, user friendly features of the original CFL3D code have been preserved. In particular, aerodynamic shape SD's can be accurately calculated for very complex geometries using the multiblock capabilities of this code, including its general patched–grid capability or its overlapped/embedded–grid capability. Furthermore, the new code maintains the same algorithmic capabilities for efficiently solving the linear sensitivity equations that are featured in the original CFL3D code for solving the nonlinear flow equations. These algorithm features include the three–factor, spatially–split, approximate factorization procedure with a choice of either the efficient Roe–diagonalized scheme or the block–tridiagonal inversion scheme. In addition, multigrid and/or mesh sequencing is retained for significantly accelerated solution of the flow sensitivity equations.
CHAPTER IV
CODE VALIDATION

4.1 Introduction

In this chapter, results are presented for sensitivity derivatives (SD's) computed with both Versions 1 and 2 of the new CFL3D.ADEI sensitivity analysis code. The accuracy of the two versions is validated with the conventional finite difference (FD) approach. Computational performance is compared relative to the FD method and also, perhaps more significant, relative to the performance of the highly efficient, highly vectorized CFL3D flow analysis code. Sec. 4.2 is concerned with the results obtained from Version 1 while Sec. 4.3 reports results from the newer Version 2. The results presented in this chapter and the subsequent chapters are obtained for supersonic inviscid flow conditions governed by the Euler equations.

4.2 CFL3D.ADII (Version 1)

This section is broken into two parts. The first part presents results from Test Case 1, which is the generic HSCT wing/body geometry while the second part presents results from Test Case 2, which is the proprietary HSCT wing/body configuration.

4.2.1 Test Case 1: Generic HSCT Wing/Body Configuration

The first test case considered for code validation is the same geometry used for the first test case of Sec. 2.6 in Chap. II. It should be recalled that the flow conditions for this test case are free stream Mach number, \( M_\infty = 2.4 \) and angle of attack, \( \alpha = 1 \) degree. These flow conditions were the same as those used in [36]. The grid generation code for this geometry is front-ended with a parametrized input format which links the desired design variables to the grid generator. The grid sensitivities \( X' \) are obtained by applying ADIFOR [55–57] to the grid generator. The CPU time and memory required for computing the grid and its sensitivities are negligible compared to the cost and memory required by the CFL3D.ADEI
code. The wing planform design variables used for validation studies are shown in Fig. 4.1. They include the root chord, the break chord, the tip chord, the inboard span, and the outboard span.

As indicated in Sec. 2.6.1, there are geometric point mismatches in the wake region of the surface grid for this geometry. Hence the geometry is represented by two grid blocks (see Fig. 2.5 of Chap. II) and the required interpolation coefficients for communication between these grid blocks have been obtained using the specialized code RONNIE [1]. For the purpose of computing sensitivity derivatives, it was first necessary to obtain the derivatives of the interface interpolation coefficients with respect to the design variables of interest. This is because, for the two block arrangement, the flow physics residual and the boundary conditions now depend, in addition to other variables, on the interpolation coefficient vector $W$. Thus, during the sensitivity analysis process, these functions need to be differentiated implicitly with respect to vector of design variables $b$ through $W$. For example, the residual $R$ at steady state can now be written as

$$R(Q(b), X(b), W(b), b) = 0$$

(4.10)

Note the introduction of the new variable $W$. Thus $R'$ now becomes

$$R' = \frac{\partial R}{\partial Q} Q' + \frac{\partial R}{\partial X} X' + \frac{\partial R}{\partial W} W' + \frac{\partial R}{\partial b} = 0$$

(4.10)

To obtain $W'$, it was necessary to apply ADIFOR to RONNIE. (Note: $W' = \frac{dW}{db}$ = $\frac{\partial W}{\partial X} X'$ where recall $X' = \frac{dX}{db}$.) Like $X$ and $X'$, the computational cost of obtaining $W$ and $W'$ is insignificant compared to the cost of the actual sensitivity analysis.
Figure 4.1 Wing Planform Design Variables for Test Case 1, HSCT 24E Geometry
Table 4.1(a) shows the sensitivity derivatives of some of the typical aerodynamic coefficients with respect to the wing planform design variables shown in Fig. 4.1. The aerodynamic coefficients for which derivatives were obtained include the lift coefficient $C_L$, the drag coefficient $C_D$, the force coefficient in the y direction $C_y$, and the pitching moment coefficient $C_{M_y}$. In Table 4.1(b), the derivatives are compared in form of ratios with those obtained via the forward finite difference method. The derivatives from both methods agree as indicated by the shown ratios, all of which are unity to four significant figures. This test case clearly shows the CFL3D.ADII code to be accurate, even for problems represented by multiple grid blocks with general patched interfaces. For the finite difference method, the step (or perturbation) size used was of the order of $10^{-5}$ for all five design variables and to ensure adequate accuracy, the flow analysis residual was converged to an average of $10^{-10}$. The finite difference method was employed in the most efficient manner in which the restart file from the well converged solution for the baseline configuration was used to initiate computations for the perturbed configurations for all design variables. The impact of using the restart file on CPU time is discussed in the next paragraph. For the CFL3D.ADII method, the sensitivity analysis iteration process was converged to an average residual level of the order of $10^{-5}$. It is remarkable that the sensitivity analysis results with this less stringent residual condition compares very favorably with the results from the finite difference method.

Since the first-order-accurate, one-sided, forward differencing was employed in the finite difference--method for this test case, it required only a total of five non-linear flow analyses (in addition to the baseline non-linear flow analysis) to obtain sensitivity derivatives for the five planform design variables. For the CFL3D.ADII code, the SD's are computed for the five DV's concurrently. This is the most efficient mode for the CFL3D.ADII code [106] (on Cray vector computers, as explained previously). It should be noted that the computational cost (CPU time) for the baseline flow analysis is common to both the finite difference method and the CFL3D.ADII method since both are started from a
Table 4.1(a) Sensitivity Derivatives of $C_L$ (Lift Coefficient), $C_D$ (Drag Coefficient), $C_y$ (Force Coefficient in $y$-direction) and $C_{M_y}$ (Pitching Moment Coefficient) for Test Case 1 (Generic HSCT Wing/Body Configuration) using the CFL3D.ADI code

<table>
<thead>
<tr>
<th>Design Variable</th>
<th>$\nabla C_L$</th>
<th>$\nabla C_D$</th>
<th>$\nabla C_y$</th>
<th>$\nabla C_{M_y}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Root Chord</td>
<td>2.957E-2</td>
<td>3.471E-3</td>
<td>8.954E-3</td>
<td>-1.226E-2</td>
</tr>
<tr>
<td>2. Break Chord</td>
<td>6.511E-4</td>
<td>-1.131E-4</td>
<td>-4.537E-4</td>
<td>-1.366E-4</td>
</tr>
<tr>
<td>3. Tip Chord</td>
<td>1.036E-5</td>
<td>9.064E-6</td>
<td>-3.098E-5</td>
<td>9.656E-6</td>
</tr>
<tr>
<td>4. Inboard Span</td>
<td>5.670E-3</td>
<td>7.204E-4</td>
<td>-4.932E-4</td>
<td>-3.359E-4</td>
</tr>
<tr>
<td>5. Outboard Span</td>
<td>-2.919E-3</td>
<td>1.682E-4</td>
<td>-1.062E-3</td>
<td>1.198E-3</td>
</tr>
</tbody>
</table>

Table 4.1(b) Sensitivity Derivative Ratios $\left( \frac{\text{CFL3D.ADI}}{\text{Finite Difference}} \right)$

<table>
<thead>
<tr>
<th>Design Variable</th>
<th>$\frac{\nabla C_{L,\text{ADI}}}{\nabla C_{L,\text{FD}}}$</th>
<th>$\frac{\nabla C_{D,\text{ADI}}}{\nabla C_{D,\text{FD}}}$</th>
<th>$\frac{\nabla C_{y,\text{ADI}}}{\nabla C_{y,\text{FD}}}$</th>
<th>$\frac{\nabla C_{M_y,\text{ADI}}}{\nabla C_{M_y,\text{FD}}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Root Chord</td>
<td>1.0000</td>
<td>0.9997</td>
<td>1.0002</td>
<td>1.0000</td>
</tr>
<tr>
<td>2. Break Chord</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>3. Tip Chord</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>4. Inboard Span</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>5. Outboard Span</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
</tbody>
</table>

Nomenclature

$\nabla (\ ) \equiv \frac{\partial (\ )}{\partial DV}$ = Sensitivity Derivative with respect to Design Variable DV

$FD$ = From the Finite Difference Method

$ADI$ = From the CFL3D.ADI Code

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
converged baseline flow analysis solution. Figure 4.2(a) shows the number of multigrid cycles and CPU time for the baseline solution, the total number of multigrid cycles and the corresponding total CPU time for the five non-linear flow analyses required for the forward finite difference method (using a restart file from the converged baseline solution for each perturbed configuration), the CPU time required per multigrid cycle per grid point for the pure flow analysis and finally, the average residual level to which the flow analyses were converged. As said earlier, the FD solutions are obtained efficiently by using the restart file from the converged solution of the baseline geometry. If all of the five flow analyses required for the FD method were started from the free stream initial conditions, it would have taken an estimated 1360 CPU seconds to obtain the required level of convergence, as opposed to the actual 565 CPU seconds. Thus using the baseline restart file led to a saving of more than 50% in CPU time for the FD method. Figure 4.2(b) shows the number of multigrid cycles, total CPU time and CPU time per multigrid cycle per grid point per design variable obtained from the CFL3D.ADII code. Five DV's are considered concurrently, hence, five linear sensitivity analysis systems were solved concurrently. The figure also shows the average residual level to which the CFL3D.ADII code was converged. Figure 4.2(c) shows the comparison, as a ratio, between the total CPU time for the finite difference method and the total CPU time for the CFL3D.ADII code. Figure 4.2(d) shows the comparison, as a ratio, between the CPU time, in μsec per multigrid cycle per grid point, of the original CFL3D code and the CPU time, in μsec per multigrid cycle per grid point per design variable, of the CFL3D.ADII code. All non-linear flow analyses and linear flow sensitivity analyses were carried out using two-level multigrid and the highly efficient Roe's diagonalized scheme with no limiters.

From Fig. 4.2(c), it is apparent that version 1 of the CFL3D.ADII is slower than the finite difference (FD) method by about 50% for this test case. This difference in performance between the two methods can be attributed partly to the nature of this test case (accurate gradients can be obtained from forward FDs, as opposed to two-sided central FDs) and partly to the poor computational efficiency of CFL3D.ADII, Version 1.
(a): CFL3D Non-Linear Flow Analysis

(i) Baseline (1 non-linear)

Number of MGC = 339

CPU Time (sec) = 272

(ii) Forward FD (5 non-linear)

Number of MGC = 750

CPU Time (sec) = 565

(iii) μsec/MGC/GP from Baseline Solution = \((272 \times 10^6) / (339 \times 27195)\)

= 29.50

(iv) μsec/MGC/GP from FD Solution = \((565 \times 10^6) / (750 \times 27195)\)

= 27.70

(v) Average Residual Level = 10^{-10}

NOTE: (iii) and (iv) are close, as expected.

(b): CFL3D.ADI Linear Flow Sensitivity Analysis (5 Linear Systems)

Number of MGC = 98

CPU Time (sec) = 1055

μsec/MGC/GP/DV = \((1055 \times 10^6) / (98 \times 27195 \times 5)\)

= 79.17

Average Residual Level = 10^{-5}

(c): Comparison of CPU Time between CFL3D.ADI and FD

\[
\text{Total CPU Time for CFL3D.ADI} = \frac{\text{CPU Time of 5 Linear Systems} + \text{CPU Time of Baseline}}{\text{CPU Time of 5 Nonlinear Systems} + \text{CPU Time of Baseline}}
\]

= \frac{1055 + 272}{595 + 272} = 1.53

Fig. 4.2 CPU Timing Results (Cray-YMP) for Test Case 1 (Generic HSCT Wing/Body Configuration, 2-Level Multigrid, Roe's Diagonalized Scheme, No Limiter, \(M_\infty = 2.4\), \(\alpha = 1.0\) degrees)
(d): Comparison of CPU Time between CFL3D:ADI and CFL3D

\[
\frac{\text{CFL3D:ADI (\mu sec/MGC/GP/DV)}}{\text{CFL3D (\mu sec/MGC/GP)}} = \frac{79.17}{29.50} = 2.68
\]

Abbreviations used in this figure

FD \equiv \text{Finite Difference}

GP \equiv \text{Grid Point}

DV \equiv \text{Design Variable}

MGC \equiv \text{Multigrid Cycle}

Fig. 4.2 Continued
Although the CFL3D.ADII code is free from the accuracy problems associated with the FD method, for it to be really competitive from a CPU time standpoint, the relative inefficiency noted above should be resolved. From Fig. 4.2(d), the CPU time for a multigrid cycle per grid point for one design variable using CFL3D.ADII (based on solving all 5 linear analysis concurrently) costs about 2.6 times as much as the CPU time for a multigrid cycle per grid point of the non-linear flow analysis using the original CFL3D code. It was also necessary to reduce this performance gap between CFL3D.ADII and CFL3D.

4.2.2 Test Case 2: Proprietary HSCT Wing/Body Configuration

The second test case used for code validation of the CFL3D.ADII (Version 1) is the proprietary wing/body configuration of Test Case 2 in Sec. 2.6.2 of Chap. II. As stated earlier, because of its proprietary nature, a full description of this geometry cannot be provided. However, the procedure upon which the parametrization of the geometry is based will be discussed. The geometry is represented by a single grid block of size 199x33x33. To obtain the required grid sensitivities $X'$, the grid generation code (QGRID) was differentiated using ADIFOR. Unlike the first test case represented by two grid blocks, there is no need for any patched interface interpolation coefficients or their derivatives. The design variables for the geometry are defined in the following general parametrization relation:

$$ \text{perturbed geometry} = \text{original geometry} + \sum \nu \phi $$

In the above relation, $\nu$ and $\phi$ are further defined as

$$ \nu = \nu(V) $$

$$ \phi = \phi(\text{original geometry, other variables}) $$

The variable $V$ is the design variable of interest. Usually, $\nu$ is a linear function of $V$. For a given $V$, a range of wing-span stations is defined as the region of influence. The quantity $\nu$ varies linearly from zero at the first wing-span station of the region of influence to a maximum of $V$ at a designated span station in the middle of the region and again linearly back to zero at the last span station of the region. The term ‘other variables’ in the expression for
φ include, among others, variables used to control thickness and camber effects and also variables to indicate the streamwise location where the design variable will be effective. For a particular region of influence (i.e., a range of wing span stations), an arbitrary number of design variables can be defined, each design variable having its own value, V, specified at a middle-span station different from others. Also, the definition of region of influence is arbitrary. A typical wing may be divided into as many as ten regions or more. Each region is in turn represented by several design variables. In addition, some regions may be defined for the fuselage also. The above parametrization provides extreme flexibility in the choice of design variables for a given geometry. However, for adequate and practical aerodynamic geometric representation of a typical wing/body configuration of the HSCT aircraft, a large number of design variables will be needed. Thus, for a comprehensive design optimization process using advanced CFD codes coupled with gradient–based optimization techniques, it becomes a challenge to compute the required sensitivity derivatives because of the large number of design variables. This challenging issue is addressed in Chaps. V and VI of this dissertation, where the parallel computing approach is presented as a viable solution. The objective of this section was just to validate the CFL3D.ADII code for accuracy and efficiency; hence only a few design variables are considered.

Like the first test case, sensitivity derivatives are computed for five randomly chosen design variables. Two of the design variables are the V values corresponding to the wing thickness orthonormal functions for two arbitrarily selected regions of influence; another two are the V values corresponding to the wing twist functions at some other two arbitrarily chosen regions, and the last one is a V value corresponding to the wing section camber for yet another arbitrarily chosen region. The step–size used for the FD methods is 0.01. This step size is within the range of typical step sizes used for the same problem in previous design studies. Table 4.2(a) shows the sensitivity derivatives of \( C_L \), \( C_D \), \( C_y \) and \( C_{My} \) with respect to the five design variables from the CFL3D.ADII code. In Tables 4.2(b) and 4.2(c), the derivatives are compared in form of ratios with those obtained via the forward FD and central
FD methods respectively. The ratios in Table 4.2(b) shows that there are significant discrepancies (up to 35% in $\partial C_L/\partial D V$ for the fifth design variable) between the forward FD method and the CFL3D.ADII code. This may imply that the step size used is too big for the first-order-accurate (i.e. $O(\Delta V)$) forward FD method to be accurate. An attempt to use a much smaller step size for possible better accuracy led to overshoots in the computed grid sensitivity and produced meaningless results. This was probably because the effect of the very small step size was drowned in the numerical noise generated by round-off errors. From Table 4.2(c), the second-order-accurate (i.e. $O(\Delta V)^2$) central FD method gave results that are much closer to those of the CFL3D.ADII code than the forward FD method. The maximum difference in this case is less than 5%.

The timing studies presented next are restricted to the central FD and the CFL3D.ADII methods due to good accuracy agreement between the two; the forward FD is excluded. The central FD method required a total of ten non-linear flow analyses, in addition to the baseline non-linear flow analysis, to obtain sensitivity derivatives for the five planform design variables. For the CFL3D.ADII code, the SD's are computed for the five DV's concurrently. As in the first test case, all nonlinear flow analyses for the FD method as well as the linear analysis of the CFL3D.ADII were restarted from the converged solution of the baseline configuration. The presentation of timing studies is also quite similar to that of test case 1, as can be seen in Fig. 4.3. For the FD method, all nonlinear flow solutions were converged to an average residual of $10^{-10}$ while for the CFL3D.ADII code, the linear problems were converged to an average residual of only about $10^{-4}$. Figures 4.3(a) and 4.3(b) show the total CPU time for the FD method and CFL3D.ADII code. Note that if the FD solutions were not started with the baseline restart, the total CPU would have been about 44000 CPU seconds instead of the actual 20080 CPU seconds recorded. Thus, as in the first test case, the use of the baseline restart produced a saving of more than 50% in the CPU time for the FD method.
Table 4.2(a) Sensitivity Derivatives of $C_L$ (Lift Coefficient), $C_D$ (Drag Coefficient), $C_Y$ (Force Coefficient in y–direction) and $C_{M_Y}$ (Pitching Moment Coefficient) for Test Case 2 (Proprietary HSCT Wing/Body Configuration) using the CFL3D.ADII code

<table>
<thead>
<tr>
<th>Design Variable</th>
<th>$\nabla C_L$</th>
<th>$\nabla C_D$</th>
<th>$\nabla C_Y$</th>
<th>$\nabla C_{M_Y}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Twist</td>
<td>1.220E-4</td>
<td>1.019E-5</td>
<td>-3.300E-5</td>
<td>-7.747E-6</td>
</tr>
<tr>
<td>Twist</td>
<td>3.936E-5</td>
<td>4.505E-6</td>
<td>7.931E-6</td>
<td>-3.583E-5</td>
</tr>
<tr>
<td>Camber</td>
<td>2.057E-3</td>
<td>2.376E-4</td>
<td>2.208E-4</td>
<td>-4.157E-4</td>
</tr>
<tr>
<td>Thickness</td>
<td>9.057E-4</td>
<td>4.753E-4</td>
<td>6.177E-4</td>
<td>-2.381E-4</td>
</tr>
<tr>
<td>Thickness</td>
<td>-2.074E-5</td>
<td>1.608E-4</td>
<td>1.612E-4</td>
<td>-7.688E-5</td>
</tr>
</tbody>
</table>

Table 4.2(b) Sensitivity Derivative Ratios

<table>
<thead>
<tr>
<th>Design Variable</th>
<th>$\frac{\nabla C_{L,ADII}}{\nabla C_{L,FD}}$</th>
<th>$\frac{\nabla C_{D,ADII}}{\nabla C_{D,FD}}$</th>
<th>$\frac{\nabla C_{Y,ADII}}{\nabla C_{Y,FD}}$</th>
<th>$\frac{\nabla C_{M_Y,ADII}}{\nabla C_{M_Y,FD}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Twist</td>
<td>1.0058</td>
<td>1.0261</td>
<td>1.0033</td>
<td>0.9119</td>
</tr>
<tr>
<td>Twist</td>
<td>1.0000</td>
<td>0.9996</td>
<td>0.997</td>
<td>1.0000</td>
</tr>
<tr>
<td>Camber</td>
<td>0.9961</td>
<td>0.9815</td>
<td>0.9986</td>
<td>1.0292</td>
</tr>
<tr>
<td>Thickness</td>
<td>1.0011</td>
<td>0.9446</td>
<td>0.9615</td>
<td>0.9811</td>
</tr>
<tr>
<td>Thickness</td>
<td>0.7407</td>
<td>0.8617</td>
<td>0.8916</td>
<td>0.7838</td>
</tr>
</tbody>
</table>

* See Nomenclature in Table 4.1

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Table 4.2(c) Sensitivity Derivative Ratios

\[
\left( \frac{CFL3D \cdot ADII}{Central \ Finite \ Difference} \right)
\]

<table>
<thead>
<tr>
<th>Design Variable</th>
<th>(\frac{\nabla CL_{ADI}^*}{\nabla CL_{FD}})</th>
<th>(\frac{\nabla CD_{ADI}}{\nabla CD_{FD}})</th>
<th>(\frac{\nabla C_{Y,ADI}}{\nabla C_{Y,FD}})</th>
<th>(\frac{\nabla C_{M,Y,ADI}}{\nabla C_{M,Y,FD}})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Twist</td>
<td>1.0008</td>
<td>1.0119</td>
<td>1.0004</td>
<td>0.9531</td>
</tr>
<tr>
<td>Twist</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>Camber</td>
<td>0.9928</td>
<td>0.9896</td>
<td>0.9866</td>
<td>1.0031</td>
</tr>
<tr>
<td>Thickness</td>
<td>1.0023</td>
<td>1.0002</td>
<td>0.9998</td>
<td>0.9954</td>
</tr>
<tr>
<td>Thickness</td>
<td>0.9990</td>
<td>1.0025</td>
<td>1.0012</td>
<td>1.0022</td>
</tr>
</tbody>
</table>

* See Nomenclature in Table 4.1

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
(a): CFL3D Non–Linear Flow Analysis

(i) Baseline (1 non–linear)
  Number of MGC = 1200
  CPU Time (sec) = 4400

(ii) Central FD (10 non–linear)
  Number of MGC = 5600
  CPU Time (sec) = 20080

(iii) $\mu$sec/MGC/GP from Baseline Solution = \( \frac{4400 \times 10^6}{1200 \times 210177} \)
      = 17.45

(iv) $\mu$sec/MGC/GP from FD Solution = \( \frac{20080 \times 10^6}{5600 \times 210177} \)
     = 17.06

(v) Average Residual Level = $10^{-10}$

NOTE: (iii) and (iv) are very close, as expected.

(b): CFL3D.ADI Linear Flow Sensitivity Analysis (5 Linear Systems)

Number of MGC = 252
CPU Time (sec) = 20250

$\mu$sec/MGC/GP/DV = \( \frac{20250 \times 10^6}{252 \times 210177 \times 5} \)
= 76.47

Average Residual Level = $10^{-4}$

(c): Comparison of CPU Time between CFL3D.ADI and FD

\[
\begin{align*}
\text{Total CPU Time for CFL3D.ADI} & = \frac{\text{CPU Time of 5 Linear Systems} + \text{CPU Time of Baseline}}{	ext{CPU Time of 10 Nonlinear Systems} + \text{CPU Time of Baseline}} \\
& = \frac{20250 + 4400}{20080 + 4400} = 1.007
\end{align*}
\]

Fig. 4.3 CPU Timing Results (Cray–YMP) for Test Case 2 (Proprietary Wing/Body Configuration, 2–Level Multigrid, Roe’s Diagonalized Scheme, No Limiter, $M\infty = 2.4$, $\alpha = 1.9$ degrees)
(d): Comparison of CPU Time between CFL3D.ADII and CFL3D

\[
\frac{\text{CFL3D . ADII (\mu sec/MGC/GP/DV)}}{\text{CFL3D (\mu sec /MGC/GP)}} = \frac{76.47}{17.45} = 4.38
\]

(e): Abbreviations used in this figure

FD ≡ Finite Difference
GP ≡ Grid Point
DV ≡ Design Variable
MGC ≡ Multigrid Cycle

Fig. 4.3 continued
Also shown in Fig. 4.3(a) is the CPU time (in μsec) required for one multigrid cycle (MGC) per grid point (GP) for the CFL3D analysis code. A similar quantity, this time in μsec/MGC/GP/design variable, for the CFL3D.ADII code is shown in Fig. 4.3(b). Figure 4.3(c) shows the comparison, as a ratio, between the total CPU time for the finite difference method and the total CPU time for the CFL3D.ADII code. Figure 4.3(d) shows the comparison, as a ratio, between the CPU time, in μsec/MGC/GP, of the original CFL3D code and the CPU time, in μsec/MGC/GP/DV, of the CFL3D.ADII code.

From Fig. 4.3(c), it can be seen that version 1 of the CFL3D.ADII, despite its possible relative computational inefficiency performs as well as the FD method from the CPU time standpoint. It should be recalled that this test case is actually a more realistic problem than the first. Thus for realistic problems characterized by large number of grid points and which may require central FD for accurate gradient computation, even the first version of the CFL3D.ADII code appears to compare favorably with the FD method, in addition to being more reliable. From Fig. 4.3(d), the CPU time for a multigrid cycle per grid point for one design variable using CFL3D.ADII (based on solving all 5 linear analysis concurrently) costs about four times as much as the CPU time for a multigrid cycle per grid point of the non-linear flow analysis using the original CFL3D code. Thus the poor performance of the first version of the CFL3D.ADII code relative to the original CFL3D code is again confirmed by this test case.

4.2.3 Summary of Results for CFL3D.ADII (Version 1)

(1) The CFL3D.ADII code computes sensitivity derivatives that are essentially the same as those obtained from carefully implemented finite difference methods. This is true even for test case 1 that is represented by two grid blocks with a general patched interface. Thus the accuracy of the CFL3D.ADII code is verified.

(2) Depending on the nature of the problem, the total CPU time required by the CFL3D.ADII code (Version 1) may or may not be larger than the total CPU time required by the finite-difference method. For instance, for test case 1 where accurate gradients could be
obtained from the first-order-accurate forward FD method, the CFL3D.ADII code is slower by about 50%. However, for test case 2 which required the second-order-accurate central FD method for accurate gradients, the total CPU time for the CFL3D.ADII code is essentially the same as that of the FD method. It is desirable that the total CPU time for the CFL3D.ADII be always lower than that of the FD method, whether it is forward FD or central FD.

(3) For the first test case, the ratio of the CPU time for one multigrid cycle per design variable of the CFL3D.ADII code to the CPU time for one multigrid cycle of the CFL3D code is about 2.7. For the second test case, this ratio is about 4.7. To bring the total CPU time down for the CFL3D.ADII, it is necessary to bring this ratio down as much as possible. This can be achieved by ensuring that the CFL3D.ADII code is as efficient as possible. The steps discussed in Sec. 3.5 were carried out toward this purpose. The result is the new, improved Version 2 of CFL3D.ADII, the performance of which is discussed in the next section.

4.3 CFL3D.ADII (Version 2)

The second version of the CFL3D.ADII code, henceforth referred to as CFL3D.ADII (Version 2), has been developed using CFL3D.ADII (Version 1) as the starting point. The changes made to Version 1 that resulted in the new improved Version 2 have been detailed in Section 3.5. In this section, the focus is to compare the computational performance between both versions. The results presented here have been documented in [39]. The test case used for performance comparison is Test Case 2 of the previous Sec. 4.2, that is the proprietary HSCT wing/body configuration with a total of 210,177 grid points.

From a memory requirement standpoint, there is no significant difference between the two versions of the CFL3D.ADII code. This fact is illustrated in Fig. 4.4, where the memory requirements are shown for the test case under consideration. For the pure fluid flow analysis (i.e., function evaluation) using CFL3D Version 4.1, the required memory is about 10MW (Megawords). For sensitivity analysis where derivatives are computed for a single design
CFL3D (Version 4.1): 9.975 MWords

CFL3D.ADI (Version 1.0) with 1 DV: 19.657 MWords

CFL3D.ADI (Version 2.0) with 1 DV: 20.160 MWords

CFL3D.ADI (Version 1.0) with 5 DV: 58.282 MWords

CFL3D.ADI (Version 2.0) with 5 DV: 58.784 MWords

DV: Design Variable

MWords: MegaWords

Fig. 4.4 Memory Required by CFL3D (Version 4.1) and CFL3D.ADI (Versions 1 and 2) for Test Case 2
variable, both versions of CFL3D.ADII requires about 20MW each, i.e., 10MW for the function and 10MW for the gradient. For sensitivity analysis where derivatives are computed for five design variables concurrently, both versions of the CFL3D.ADII code requires about 60MW each, i.e., 10MW for the function and 10MW for each of the design variables (i.e. $10 + 5 \times 10 = 60MW$). From the foregoing, it can be concluded that for either versions of the CFL3D.ADII, the memory required for gradient evaluation varies linearly with the number of design variables. Thus as the number of design variables grows, the total memory can become quite large and can easily exceed the limit of the currently available vector supercomputers such as the Cray C90. If the number of design variables is large, an alternative for this kind of computers will be to break the problem into groups of design variables, each group having five design variables. With this option, a job can then be submitted for each group. The problem with this option is that the turnaround time required to obtain the sensitivity derivatives from all of the groups will be correspondingly large since a job submission for each group of design variables will wait its turn on the usually overloaded and long queues of these supercomputers. For a gradient–based design optimization process which usually requires many gradient evaluations, the poor turnaround time becomes a serious limitation. All these problems have motivated the search for a viable alternative via parallel computing, which is another objective of this work. The realization of this objective is detailed in Chaps. V and VI of this study. The results presented in this section have significance for problems with few design variables that can easily fit into the memory of the available vector supercomputers, for example, Cray YMP and C90. All computations in this section were carried out on a Cray YMP, as in the previous section.

The CPU timing results are presented in Tables 4.3(a)–(d). Four code options are considered. These options are derived by combining the choice of an algorithm, e.g. the Roe's diagonalized scheme with the choice of a flux limiter, e.g. the min–mod limiter. The four code options are (1) the Roe's Diagonalized scheme with No Limiter (RD.NL), (2) the
Table 4.3(a) CPU Timings* for CFL3D, Version 4.1 (Baseline) with Different Code Options for Test Case 2

<table>
<thead>
<tr>
<th>Code Option</th>
<th>CPU Time (μsec/MGC/GP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RD.NL</td>
<td>17.7</td>
</tr>
<tr>
<td>RD.MM</td>
<td>18.3</td>
</tr>
<tr>
<td>VL.NL</td>
<td>41.5</td>
</tr>
<tr>
<td>VL.MM</td>
<td>42.1</td>
</tr>
</tbody>
</table>

Table 4.3(b) CPU Timings* for CFL3D.ADII (Versions 1.0 and 2.0) with Different Code Options and 1 Design Variable for Test Case 2

<table>
<thead>
<tr>
<th>Code Option</th>
<th>CPU Time (μsec/MGC/GP)</th>
<th>Speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Version 1.0</td>
<td>Version 2.0</td>
</tr>
<tr>
<td>RD.NL</td>
<td>397.8</td>
<td>48.4</td>
</tr>
<tr>
<td>RD.MM</td>
<td>803.1</td>
<td>54.1</td>
</tr>
<tr>
<td>VL.NL</td>
<td>644.2</td>
<td>75.6</td>
</tr>
<tr>
<td>VL.MM</td>
<td>1046.7</td>
<td>81.2</td>
</tr>
</tbody>
</table>

Table 4.3(c) CPU Timings* for CFL3D.ADII (Versions 1.0 and 2.0) with Different Code Options and 5 Design Variables for Test Case 2

<table>
<thead>
<tr>
<th>Code Option</th>
<th>CPU Time</th>
<th>Speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>μsec/MGC/GP</td>
<td>μsec/MGC/GP/DV</td>
</tr>
<tr>
<td></td>
<td>Version 1.0</td>
<td>Version 2.0</td>
</tr>
<tr>
<td>RD.NL</td>
<td>384.5</td>
<td>180.1</td>
</tr>
<tr>
<td>RD.MM</td>
<td>718.9</td>
<td>188.5</td>
</tr>
<tr>
<td>VL.NL</td>
<td>714.5</td>
<td>182.1</td>
</tr>
<tr>
<td>VL.MM</td>
<td>1048.8</td>
<td>190.4</td>
</tr>
</tbody>
</table>

*CPU Time for Cray–YMP (Sabre)
Table 4.3(d) CPU Time Ratios \( \frac{\mu \text{sec / MGC/GP/DV for CFL3D ADII}}{\mu \text{sec / MGC/GP for CFL3D}} \) with

Different Code Options for Test Case 2

<table>
<thead>
<tr>
<th>Code Option</th>
<th>With 1 DV</th>
<th></th>
<th>With 5 DV (per DV)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Version 1.0</td>
<td>Version 2.0</td>
<td>Version 1.0</td>
<td>Version 2.0</td>
</tr>
<tr>
<td>RD.NL</td>
<td>22.39</td>
<td>2.73</td>
<td>4.34</td>
<td>2.04</td>
</tr>
<tr>
<td>RD.MM</td>
<td>43.81</td>
<td>2.96</td>
<td>7.86</td>
<td>2.06</td>
</tr>
<tr>
<td>VL.NL</td>
<td>15.47</td>
<td>1.82</td>
<td>3.44</td>
<td>0.878</td>
</tr>
<tr>
<td>VL.MM</td>
<td>24.90</td>
<td>1.92</td>
<td>4.98</td>
<td>0.905</td>
</tr>
</tbody>
</table>

**Abbreviations used in Tables 4.3(a)–(d)**

GP = Grid Point

DV = Design Variable

MGC = Multigrid Cycle

RD = Roe’s diagonalized upwind scheme

VL = van Leer’s upwind scheme

NL = No limiter

MM = min – mod flux limiter

\( \mu \text{sec} \) = micro – second (of CPU time)
Roe's Diagonalized scheme with Min–Mod limiter (RD.MM), (3) the Van Leer's scheme with No Limiter (VL.NL) and (4) the Van Leer's scheme with Min–Mod limiter (VL.MM). Only the first code option, that is RD.NL, was considered in the previous Sec. 4.2.2.

Table 4.3(a) shows the timings for the original CFL3D (Version 4.1) code. Results are presented in microseconds per multigrid cycle per grid point (μsec/MGC/GP). It is evident from this table that the Roe's scheme requires about half the time required by the van Leer's scheme. In Table 4.3(b), timing results are shown for both Versions 1 and 2 of CFL3D.ADII where gradients of aerodynamic functions are computed for just one design variable. This case is henceforth referred to as the 1DV mode. Again, as in Table 4.3(a), CPU timings are in μsec/MGC/GP. As can be seen in Table 4.3(b), there is tremendous speed up in Version 2 as compared to Version 1. Speed up factors, representing the ratio of the CPU time for Version 1 to that of Version 2, range from 8.2 to 14.8, depending on the code option used. In Table 4.3(c), timing results are shown for both versions of CFL3D.ADII as in Table 4.3(b), except that sensitivity derivatives are computed concurrently for five design variables. This case is henceforth referred to as the 5DV mode. In Table 4.3(c), results are presented first in μsec/MGC/GP, that is the CPU time for five design variables. In the same table, the time required for each design variable in this 5DV mode are shown as μsec/MGC/GP/DV. (Note: μsec/MGC/GP/DV = μsec/MGC/GP divided by 5.) The improved performance of version 2 over version 1 is evident from the speed–up factors which are also shown in the table. The improvement is very significant, with speed–up factors ranging from 2.13 to 5.51, (but not as dramatic as in Table 4.3(b)).

Finally, Table 4.3(d) compares directly (in form of ratios) the cost of computing sensitivity derivatives using both versions of CFL3D.ADII with the cost of the pure flow analysis using the original CFL3D Version 4.1. Comparisons are made for both the 1DV and 5DV modes. It is extremely significant to note that for Version 2 of CFL3D.ADII, some of the ratios in Table 4.3(d) are less than unity. This indicates that for some of the code options, the cost of one multigrid cycle per design variable using the improved CFL3D.ADII (Version
2) is lower than that of one multigrid cycle of CFL3D (Version 4.1). This is a major breakthrough for this methodology, especially when compared with the traditional finite difference (FD) method. (Recall that the FD method requires one extra function evaluation per design variable if it is forward FD or two extra function evaluations if it is central FD.) Each MGC of the function evaluation requires the same CPU time as a MGC of the baseline. Also, the total number of iterations that is required for adequate convergence in the FD method is by far larger than the number of iterations required by the CFL3D ADII. This fact is illustrated in Figs. 4.2 and 4.3 of Sec. 4.2. In Fig. 4.3 for example, which was for the RD.NL code option and the same test case used in this section, the central FD mode requires a total of 5600 iterations while the CFL3D ADII requires only about 252 iterations. It should be noted that apart from being more efficient, CFL3D ADII (Version 2) has the same level of accuracy as the first version. Therefore, for the same test case, both versions will require the same number of multigrid cycles (MGC) to give identical SD results, except that Version 2 will achieve this at a much smaller CPU time. The number of MGC in Figs. 4.2 and 4.3 on Version 1 (from the previous section) can subsequently be used to estimate the total CPU time required by the second version. Although the total number of MGC results shown in Fig. 4.3 are for the code option RD.NL, the number of MGC cycles that will be required by the other code options for the same level of accuracy is unlikely to be significantly different. Thus for the code options with less than unity ratios in Table 4.3(d), there will be substantial savings in the total CPU time using the new version of CFL3D ADII. Even for ratios larger than unity, the fact that CFL3D ADII provides accurate results with far fewer MGC will make the total CPU time significantly less than for the FD method. For example, for our test case which requires 252 MGC with the RD.NL code option, the total CPU time for 5 design variables using the new version of the CFL3D ADII in the 1DV mode will be
\[ 4400 + 252 \times 5 \times 48 \times 210177/10^4 = 17217.43 \text{ seconds.} \]
In the 5DV mode, the CPU time will be
\[ 4400 + 252 \times 5 \times 36 \times 0.02 \times 210177/10^4 = 13938.25 \text{ seconds.} \]
From Fig. 4.3(d), the FD method requires 24480 CPU seconds. Therefore, the 1DV mode of
CFL3D.ADI (Version 2.0) provides a net saving of about 30% in CPU time while the 5DV
mode provides a net saving of about 43%.

The improved performance of version 2 of CFL3D.ADI over version 1 demonstrated
above is due largely to improved vectorization of Version 2. Another improvement step
mentioned in Sec. 3.5, that is, eliminating the update on the converged flow field solution $Q$
so that possible larger time steps can be taken for the SA calculations, merits a more thorough
investigation.
CHAPTER V
DISTRIBUTED–MEMORY PARALLEL IMPLEMENTATION OF CFL3D.ADII

5.1 The Challenge

For a meaningful design optimization study of the proprietary HSCT wing/body configuration (Test Case 2 of Chaps. II and IV), a large number of design variables (usually on the order of hundreds) needs to be considered concurrently. Based on the studies from Sec. 4.3, each design variable requires a memory of about 10MW for this test case. Thus for 100 design variables for example, about 1GW (GigaWord) will be needed. This exceeds the capacity of most of the currently available supercomputers. As observed in section 4.3, breaking the problem into smaller units that can fit into the available memory will require many batch job submissions which implies large turnaround time for the complete set of gradients to be computed. This represents a bottleneck in a design process, where these gradients will have to be evaluated a number of times. To overcome these problems, recourse was made to the use of parallel computation. Thus the next stage of this work was to modify the CFL3D.ADII so that it can be implemented on a parallel computing platform. The platform of choice is the distributed–memory parallel computing paradigm where interprocessor communication is achieved by message–passing. A specific example of such a platform is the 160–node IBM–SP2 at the NAS (Numerical Aerospace Simulation) facility located at NASA Ames research center. In this work, a version of CFL3D.ADII was developed for coarse grain implementation on the SP2 as well as any other platform similar to the SP2, for example, a cluster of workstations. The rest of this chapter concentrates on the development of the parallel version of CFL3D.ADII. The use of this parallel version in the design optimization studies of the proprietary HSCT wing/body configuration (Test Case 2) will be discussed in the next chapter.
Section 5.2 provides a brief description of the IBM–SP2 parallel computer at NAS. Section 5.3 presents a description of the approach used for coarse grain parallel implementation of CFL3D.ADII. Section 5.4 shows the sensitivity derivatives computed with the parallelized CFL3D.ADII. More than 100 design variables are considered concurrently. The computational efficiency, speedup and scalability issues are discussed in section 5.5.

5.2 The NAS IBM–SP2

Most of the information contained in this section is obtained from the IBM homepage and the NAS homepage on the internet.

The basic architecture of the IBM–SP2 parallel computer is a distributed memory, message passing parallel processor. Each node is essentially an IBM RS6000/390 or RS6000/590 workstation based on the POWER2 multichip RISC processor, the proven leader in microprocessing technology. The nodes are connected by a fast network, with some software to make them look like a single parallel computer. The SP2 connects with the outside world via open communication standards such as Ethernet, FDDI, FCS, ATM, etc. Two different types of SP2 nodes are available; thin nodes and wide nodes. The thin nodes are basically IBM RS6000/390 workstations and the wide nodes are IBM RS6000/590 workstations. Compared to the thin nodes, the wide nodes have more expansion capability and two or four times bigger memory bandwidth, leading to twice the floating point performance for some codes.

The NAS IBM–SP2 has a total of 160 nodes, all of which are the RS6000/590 wide nodes. Each node has at least 128 MB of main memory and 2 GB of disk space. Some nodes have bigger memory (up to 512 MB) and bigger disk space (up to 8 GB). The NAS IBM–SP2 also has an external file system accessible by all nodes. The full 160-node SP2 has 23.9 GB of main memory, 458 GB of disk space, 342 GB/sec main memory bandwidth and 42.8 GFlops peak performance. Like any other 590 workstation, the NAS SP2 nodes have a clock rate of 66.7 MHz, a data cache of 256 KB, two integer computation units and two
floating-point computation units. Each floating-point unit can finish two 64-bit operations (a multiply and an add) each clock period. This gives a peak performance of a little more than 250 MFlops (4 operations per clock period) for the POWER2 processor. This is roughly half the performance of a Cray–YMP processor. The cache-to-processor bandwidth is four 64-bit words per clock period, which is adequate to feed the floating-point units. The main memory-to-cache bandwidth is the same as the POWER2 cache-to-processor bandwidth. This allows the 590 to run at much closer to peak performance than machines with poor memory bandwidth. There is still a latency penalty for using main memory, and transfers from main memory are still done a cache line (256 bytes) at a time.

The high performance switch that connects the SP2 nodes as a network is a multi-stage, omega, buffered-wormhole routing packet-switch. The flow control is token-based. The switch can theoretically transfer data between SP2 nodes at 1 μsec latency and 40 MB/sec bidirectional bandwidth. However with software, the latency is approximately 45 μsec and the bandwidth is about 34 MB/sec. The switch operates synchronously, that is, the network is driven by a global clock, although individual devices may be out of phase.

Each node of the SP2 runs a full version of the AIX operating system which is the IBM’s implementation of Unix. AIX is augmented by tools for system management (AIX Parallel System Support programs), job management and scheduling (LoadLeveler) and the development and execution of message passing applications (AIX Parallel Environment). Instead of the LoadLeveler, NAS uses its locally developed package PBS (Portable Batch System). The NAS SP2 supports only message passing programs. Programs can be written in FORTRAN 77, FORTRAN 90, C, C++ and HPF (high Performance FORTRAN). The available message passing libraries include the standard Message Passing Interface (MPI), the IBM’s proprietary Message Passing Library (MPL) and the Parallel Virtual Machine (PVM).

For many applications, the disk space in the user’s home directory is not sufficient. There are a number of alternative sources of disk space. The first and the most convenient is
the scratch space system. The scratch is a 16GB file system mounted to all the nodes via NFS. This file system is not backed up and files more than 3 days old may be deleted. The second alternative is the /tmp file system. This is available on and local to each node. However, using the /tmp file system is rather tricky. Also, the /tmp is available only for the current job. As soon as the job finishes, /tmp is purged. Hence this file system should only be used for temporary files needed during a single job.

The third alternative is to use the NAS mass storage systems. There are routines that can be called within a program for copying files from the mass storage system to each node and vice versa. This alternative can be used to move a file created on the /tmp of a node to permanent storage before control is returned from the current job. In this study, the scratch system was mostly employed.

5.3 The Approach

The approach employed for the coarse-grain parallel implementation of the CFL3D.ADII is similar to the derivative stripmining method of [58]. In this technique, an identical copy of CFL3D.ADII is run on each active node of the SP2, but each node with different grid sensitivity data. The grid sensitivity data for each node (which is local to the node) may be for a single design variable or a group of design variables. In other words, each node is dedicated to a process. A process in this sense is defined as an execution of a function (CFL3D.ADII) with a given input (grid sensitivity data). Since all nodes execute the same function but with different data sets, the method falls under the classification of data parallelism. The implementation is coarse-grain because the function, that is CFL3D.ADII is an agglomeration of very large number of different tasks performed on the entire geometrical domain of the problem, i.e., the grid/grid sensitivity data for the whole wing/body configuration. The communication among processors (or among nodes of the SP2 in this case) is achieved via message passing. The particular message passing software is the Message Passing Interface (MPI) standard. Like most message passing systems, the task for each node (CFL3D.ADII) is identical, as stated earlier, and is defined at program startup.
No tasks are allowed to be created or destroyed during execution. Thus the implementation is also a Single Program Multiple Data (SPMD) model.

The target problem was to obtain sensitivity derivatives for at least 100 design variables concurrently for the proprietary wing/body configuration of test case 2. To be able to use the derivative stripmining approach, it was necessary that a node of SP2 has enough memory to contain CFL3D.ADII for at least one design variable. Because this was possible, then each node was assigned to a design variable. For larger problems, this may not be possible due to the memory limitation of each node. In this case, a finer grain implementation would have to be considered, such that the problem can be partitioned into smaller units that will easily fit into the available memory. An example of such a partition would be to divide the geometrical domain into subdomains such that sensitivity derivatives can be obtained for each subdomain. The results from the subdomains will then be combined to give the results for the entire domain. The larger the number of subdomains, the higher the degree of parallelism, but also the higher the communication cost. The overall objective is to partition the problem domain into a suitable number of subdomains and then assign the subdomains to processors such that the competing goals of maximizing processor utilization and minimizing communication costs can be satisfied. It is also important that the partition meets design requirements on the target parallel computer, for example allowable execution time and memory limitation (10 CPU hours and 128 MB for most of the NAS SP2 nodes). The execution time is not so much of a constraint in this case as is memory. If a converged solution is not obtained in the first run, new solution can always be started using the restart file generated from the previous run. Thus the first important step is to establish if CFL3D.ADII for 1DV can fit into the memory of an SP2 node.

It was stated earlier that on Cray–YMP, CFL3D.ADII for 1DV requires about 20MWords of memory for the problem being considered. This is equivalent to 160MBytes. The Cray, by default, computes in 64-bit precision. This is the same precision for the IBM RS6000 processor if computations were performed in double precision. Thus in double
precision mode, CFL3D.ADII with 1DV will obviously not fit into most of the NAS SP2 nodes with only 128 MB of memory per node. In single precision mode, the required memory will be halved to only 80MB. Thus if the required accuracy can be obtained with single precision computation, CFL3D.ADII with 1DV will fit comfortably into the memory of each node. For the CFL3D.ADEI code, there are two major aspects to worry about. The first is the grid sensitivity. As illustrated in Sec. 3.4.2 of Chap. III, if the grid sensitivity is obtained by quasi-analytical differentiation of the grid generation code via ADIFOR, accuracy is not impaired, even when computations are performed in the single-precision mode on a node of the SP2. Note that this is not true for the finite difference method, as discussed in that earlier section. The second aspect to worry about is the flow physics itself. For instance, if the physics involves turbulent flow with highly stretched grids close to the geometry, it may be necessary to perform computations in double precision. However, the flow physics in this case is governed by the Euler equations, and sufficient accuracy can be obtained in single precision. To be sure that this is so, a few of the derivatives from the SP2 are compared with equivalent ones from the Cray-YMP. Details are provided in the next section, Sec. 5.4. Having ascertained that the desired accuracy can be obtained in single precision, the parallel code was designed in this mode. Hence it was possible to fit CFL3D.ADII with 1DV per node. Altogether, a total of 108 design variables are considered concurrently on 108 nodes of the SP2. The nodes are identified as node 0 to node 107. Node 0 is the coordinating node.

Having determined how the problem will be partitioned, the next stage is to design the internodal communication pattern. To reduce CPU time overhead associated with communication through massive data movement, the parallel CFL3D.ADII code was designed to ensure data locality for each node at the start of each run. Because of the size of the problem, it became necessary to make use of the scratch file system mentioned earlier in Sec. 5.2. This file system is cross-mounted on all the SP2 nodes and it is accessible only through the PBS. All large data files such as the grid file, the grid sensitivity files, the
PLOT3D grid and flow field variables files, flow analysis restart file and lastly, the sensitivity analysis restart files are all located in the scratch directory. The entire file system for the parallel code is divided into three groups. The first group comprises the input files to pure flow analysis, i.e. the usual CFL3D input file, the grid file, the flow restart file, the patched-grid coefficient file (if applicable) and the overlapped-grid coefficient file (if applicable). The second group comprises the output files from the pure fluid flow analysis, i.e. all ASCII output files from CFL3D and the PLOT3D files. The third group comprises sensitivity-analysis-related files such as the grid sensitivity file, the sensitivity restart file, the patched-grid coefficient sensitivity file (if applicable) and the overlapped-grid coefficient sensitivity file (if applicable). The files in the first group are accessible by all participating nodes. The files in the second group are accessible only by the coordinating node (usually node 0). The files in the third group are unique to each node. In other words, each node has its own unique grid sensitivity file and sensitivity restart file accessible only by that node. At the end of all computations, all nodes from node 1 to node 107 send their respective sensitivity derivatives and sensitivity analysis convergence history data to node 0. Node 0 gathers all the results and outputs them into a single file. It should be noted that this is the only major communication required among the nodes and it takes place only after control is returned from the main computational kernel of CFL3D-ADI2. Compared to the computational cost, the overhead introduced by this minimal communication is negligible. The flow chart in Fig. 5.1 illustrates the arrangement of the file system described above.
Fig. 5.1 File System Arrangement for the Parallel CFL3D.ADI2
5.4 Parallel Sensitivity Analysis Results

The sensitivity derivatives (SD) for aerodynamic functions such as the lift coefficient $C_L$, the drag coefficient $C_D$, the $x$-, $y$-, and $z$-force and moment coefficients ($C_x$, $C_y$, $C_z$, $CM_x$, $CM_y$, and $CM_z$) with respect to 108 geometric design variables were computed concurrently. Each node of the SP2 is dedicated to 1 design variable. The 108 design variables are composed of 12 wing twist functions, 48 camber functions and 48 thickness functions. Tables 5.1, 5.2 and 5.3 show the SD values for $C_L$, $C_D$ and $CM_z$ respectively for ten randomly selected design variables. The SD results for the entire 108 design variables for these three aerodynamic coefficients are shown in Table A.1 of Appendix A. In Tables 5.1 – 5.3, SD's for five out of the ten randomly selected DV's computed with the Cray version of the CFL3D.ADII are also presented. The residual level for the SP2 results was about $10^{-3}$ while for the Cray results, it was about $10^{-4}$. The SP2 residual stagnated at a higher level due to the combined effect of using a flux limiter (because of the complexity of the problem considered) together with single-precision arithmetic. The SP2 results, nevertheless, are correct. This was confirmed by comparison between the Cray and the SP2 results, which shows that there is no loss of accuracy by using the single-precision arithmetic on the SP2 for this test case.

5.5 Efficiency, Speedup and Scalability Analysis

When developing an application for a multiprocessor parallel computer, an important principle is to concentrate a computational resource (or a combination of resources) on problems that ordinarily can not be solved or will take too long to solve on conventional single processor computers. The computational resource can be processor speed, main memory, or even input/output (I/O) bandwidth. For the parallel CFL3D.ADII presented in this chapter, the most important issue is to have sufficient memory for a large number of design variables. This condition is met in the IBM–SP2. In addition, the execution time for the entire 108 design variables on 108 nodes is effectively equal to the execution time for 1 design variable on 1 node, since communication time and idle time are negligible. However,
Table 5.1 Sensitivity Derivatives for the Lift Coefficient ($C_L$) from the Parallel Version of CFL3D.ADII for Ten Randomly Selected Design Variables (DVs). Results from the Cray Version for Five of the DVs are included for Comparison

<table>
<thead>
<tr>
<th>DV</th>
<th>$\frac{\partial C_L}{\partial DV_{SP2}}$</th>
<th>$\frac{\partial C_L}{\partial DV_{Cray}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.34696E-03</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.18439E-03</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0.39360E-04</td>
<td>0.39360E-04</td>
</tr>
<tr>
<td>13</td>
<td>0.46031E-02</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>-0.20718E-03</td>
<td>-0.20718E-03</td>
</tr>
<tr>
<td>60</td>
<td>-0.50851E-04</td>
<td>-0.50851E-04</td>
</tr>
<tr>
<td>61</td>
<td>0.38792E-03</td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>-0.91131E-03</td>
<td></td>
</tr>
<tr>
<td>84</td>
<td>0.11413E-03</td>
<td>0.11413E-03</td>
</tr>
<tr>
<td>108</td>
<td>-0.30570E-05</td>
<td>-0.30570E-05</td>
</tr>
</tbody>
</table>

Table 5.2 Sensitivity Derivatives for the Drag Coefficient ($C_D$) from the Parallel Version of CFL3D.ADII for Ten Randomly Selected Design Variables (DVs). Results from the Cray Version for Five of the DVs are included for Comparison

<table>
<thead>
<tr>
<th>DV</th>
<th>$\frac{\partial C_D}{\partial DV_{SP2}}$</th>
<th>$\frac{\partial C_D}{\partial DV_{Cray}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.32437E-04</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.19762E-04</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0.45047E-05</td>
<td>0.45047E-05</td>
</tr>
<tr>
<td>13</td>
<td>0.62859E-03</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>-0.53622E-04</td>
<td>-0.53622E-04</td>
</tr>
<tr>
<td>60</td>
<td>-0.12042E-04</td>
<td>-0.12042E-04</td>
</tr>
<tr>
<td>61</td>
<td>0.27860E-03</td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>0.13697E-03</td>
<td></td>
</tr>
<tr>
<td>84</td>
<td>0.14228E-03</td>
<td>0.14228E-03</td>
</tr>
<tr>
<td>108</td>
<td>0.56120E-04</td>
<td>0.56120E-04</td>
</tr>
</tbody>
</table>
Table 5.3 Sensitivity Derivatives for the X-Moment Coefficient ($C_{Mx}$) from the Parallel Version of CFL3D.ADI for Ten Randomly Selected Design Variables (DV's). Results from the Cray Version for Five of the DV's are included for Comparison

<table>
<thead>
<tr>
<th>DV</th>
<th>$\frac{\delta C_{Mx}}{\delta DV \text{ SP2}}$</th>
<th>$\frac{\delta C_{Mx}}{\delta DV \text{ Cray}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-0.10300E-03</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>-0.75024E-04</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>-0.32818E-04</td>
<td>-0.32818E-04</td>
</tr>
<tr>
<td>13</td>
<td>-0.89476E-03</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>0.16085E-03</td>
<td>0.16085E-03</td>
</tr>
<tr>
<td>60</td>
<td>0.59000E-04</td>
<td>0.59000E-04</td>
</tr>
<tr>
<td>61</td>
<td>-0.65748E-04</td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>0.83599E-03</td>
<td></td>
</tr>
<tr>
<td>84</td>
<td>-0.38303E-04</td>
<td>-0.38303E-04</td>
</tr>
<tr>
<td>108</td>
<td>0.22820E-04</td>
<td>0.22820E-04</td>
</tr>
</tbody>
</table>
this does not imply that the computational efficiency, $E$, is 100% nor that the actual speedup, in terms of the overall CPU time in a uniprocessor implementation, is ideal. The reason for this is due to the presence of unwanted redundant calculations which occur in the present parallel implementation, as will be explained shortly. Since $E$ is not 100%, it is necessary to assess how $E$ is affected by the redundant calculations and whether $E$ will decrease without bound as the problem size or number of design variables increases, in which case the parallel implementation will not be scalable. The outcome of this investigation, especially from the standpoint of computational efficiency, should not be taken as an absolute figure of merit for the parallel implementation. This is because the computational efficiency is measured against a uniprocessor implementation which actually is not feasible due to memory limitation. As will be shown later, the CPU time for the uniprocessor implementation is only an estimate, since it was not possible to fit all 108 design variables on one node at once to measure the actual CPU time. It was not even possible to fit more than a single design variable on one node, with the present problem size.

From preliminary studies on the IBM-SP2, it was found that about 35% of the overall computations in CFL3D.ADI (for the Roe Diagonalized, No Limiter code option, RD.NL) is performed outside the DO-loops for derivative calculations. This out-of-the loop part of the computation is executed once, regardless of the number of design variables for which derivatives are computed concurrently. This cost is due to the fact that ADIFOR-generated derivative code always computes the function evaluations of the original code, as well as the derivative calculations. However, if CFL3D.ADI is executed more than once for the same problem, the out-of-the loop computation is replicated as many times as the number of executions. For example, in the case considered here, this portion of the overall computations is replicated 108 times, since each of the 108 nodes executes an identical copy of CFL3D.ADI. An alternative to the unwanted redundant calculations will be to perform this computation once and for all on one node and make the resulting data available to all nodes. This however, due to the extremely fine granularity (complex and interwoven
computation) of this approach, will imply thousands of calls to the message passing routines that involve movement of large volumes of data across the network. The result of this is that the total communication time will become a dominating contribution to the overall execution time. In addition, the idle time may become significant because it is now possible that some nodes may have to wait for data. The overall computational efficiency will drop sharply, even though the level of extractable parallelism is higher. Thus performing the redundant calculations locally on each node was deemed to be a better alternative which also was much easier to implement in parallel.

For the code option used here, with the associated 35% redundant calculations, the estimated overall execution time if all design variables were considered concurrently on just one processor (an impossible task due to excessive memory requirements) will be \( T_{\text{all}} = (N - 0.35(N - 1))T_C \) where \( N \) is the number of design variables and \( T_C \) is the computation time for 1 design variable in the serial mode, i.e., in the absence of parallel computation. The parallel execution time, \( T_E \) is given by \( T_E = T_C + T_{CM} + T_I \) where \( T_C \) is the computation time (or the execution time for 1 DV on a node that is not operating in parallel mode), \( T_{CM} \) is the communication time and \( T_I \) is the idle time. As stated earlier in this section, \( T_{CM} \) and \( T_I \) are negligible. \( T_I \) is negligible because no node waits for data from another node. \( T_{CM} \) is negligible based on the following analysis. The total volume of data communicated (in 4-byte or 32-bit words), based on the implementation and file arrangement discussed in Sec. 5.3, is given by

\[
\text{Total Data Volume} = N \times NAF1 \times NITER + N \times NAF2 \tag{5.1}
\]

where \( N \) is the number of design variables, \( NAF1 \) is the number of aerodynamic functions for which the sensitivity residual history information will be output, \( NITER \) is the total number of multigrid cycles (MGC) for the sensitivity analysis computations and \( NAF2 \) is the number of aerodynamic functions for which gradient information will be output. For the first part of Eq. (5.1), there is a total of \( N \times NAF1 \) message startups. For the second part, there is a total of \( N \) message startups. Therefore, the total CPU time due to communication is, using Eq. (5.1)
\[ \text{Communication Time} = N(NAF_1 + 1)T_s + N(NAF_1 \times \text{NITER} + NAF_2)T_w \] \hspace{1cm} (5.2)

\(T_s\) is the message startup cost and \(T_w\) is the cost/word. One word is 4 bytes. For the high performance switch of the IBM-SP2, \(T_s\) is measured to be about 42 \(\mu\) sec and \(T_w\) about 0.13 \(\mu\) sec. These values agree with the values quoted for the IBM-SP2 in [116]. For the parallel CFL3D.ADII as implemented here, \(NAF_1 = 4\), \(NAF_2 = 3\), and \(\text{NITER} = 200\). That is the sensitivity residual history information is output for four aerodynamic functions, namely \(C_L\), \(C_D\), \(C_y\) and \(C_{My}\). The sensitivity residual history output is comparable to that of the original CFL3D code. The gradient information is output for three functions, namely \(C_L\), \(C_D\), and \(C_{My}\). These three functions are the ones which will be used in Chap. VI for design optimization studies. Using Eq. (5.2) with \(N = 108\) and the data provided above, the total communication time \(T_{CM} = 108 \times (4 + 1) \times 42 + 108 \times (4 \times 200 + 3) \times 0.13 = 33,954.12 \mu\) sec \(= 0.03395\) sec. The average execution time to compute gradients with respect to one design variable using CFL3D.ADII on a node of the SP2 in serial mode is about 13 CPU hours. It is apparent that the communication time is completely negligible. Even when \(N\) becomes extremely large, say, a million, \(T_{CM}\) will just be about 234 seconds, which is still negligible compared to the overall execution time of about 13 hours. Thus for simplicity, \(T_{CM}\) is ignored in later discussions. For other message-passing parallel computing platform where \(T_s\) and \(T_w\) are much larger than for the homogeneous IBM-SP2 network, it may not be safe to ignore \(T_{CM}\) completely. For example, for workstations on the Ethernet, which is the worst case documented in [116], \(T_s\) is about 1500 \(\mu\) sec and \(T_w\) about 5.0 \(\mu\) sec. With the problem considered here, \(T_{CM}\) becomes \(108 \times (4 + 1) \times 1500 + 108 \times (4 \times 200 + 3) \times 5.0 = 1243620 \mu\) sec \(= 1.2436\) sec. This is still very small compared to the total execution time. However, with a very large number of nodes, e.g. one million, the communication time is more than three hours. Obviously, the effect of communication can no longer be neglected. For this kind of network with very large message startup cost and relatively large cost/word, there are a number of steps that can be taken for the parallel CFL3D.ADII as implemented in this study, in order to remain scalable. The first
step will be to divide the task of writing the sensitivity residual history output among a certain number of processors, instead of Processor 0 doing all the output. If this still does not work satisfactorily, then a last resort will be to have each processor writing its own output, in the same manner that the very large data files are typically managed, though this will imply a large number of output files that may be difficult to handle. Finally, the startup cost part of Eq. (5.2) can be reduced substantially by reducing the number of message startups to the minimum, which is $N$. This can be achieved by packing all the messages to be sent from each node into a single array, and then sending them at once. This is possible, since all the required data are available.

From the foregoing, for the implementation of the parallel CFL3D ADII on the SP2, $T_E$ is effectively equal to $T_C$. Thus the parallel computational efficiency is given by

$$E = \frac{T_{all}}{NT_E} = \frac{(N - 0.35(N - 1))T_C}{NT_E}$$

$$\approx \frac{(N - 0.35(N - 1))T_C}{NT_C}$$

$$= \frac{N - 0.35(N - 1)}{N}$$

(5.3)

For the case considered here with $N = 108$, $E$ is about 65.3%. Eq. (5.3) can be rewritten as follows:

$$E = \frac{N(1 - 0.35)}{N} + \frac{0.35}{N}$$

$$= 0.65 + \frac{0.35}{N}$$

(5.4)

From Eq. (5.4), when $N$ becomes large, the term $0.35/N$ becomes very small and hence $E$ asymptotically approaches a constant value equal to 0.65. Thus $E$ is bounded from below by 0.65. At $N = 108$, $E$ has practically reached the asymptotic limit. Fig. 5.2(a) shows a plot $E$ vs. $N$, up to $N = 108$ while Fig. 5.2(b) shows a similar plot but for $N = 108$ to 1000. The purpose of Fig. 5.2(a) is to show the behavior of $E$ at low values of $N$ while the purpose of Fig. 5.2(b) is to show the behavior of $E$ as $N$ becomes large. From both figures, it can be seen that for low
values of \( N \), up to \( N \approx 15 \), \( E \) decreases rapidly as \( N \) increases. The rate of decrease in \( E \) also decreases rapidly such that at \( N \approx 15 \), \( E \) has practically stopped decreasing and the curve levels off asymptotically to a value of \( E \approx 0.65 \), as observed before. The relative speedup \( S \) is defined as \( N E \), that is the product of number of processors and the computational efficiency. For \( N = 108 \), the speedup is equal to \( 108 \times 0.653 = 70.52 \). Fig. 5.3 shows a plot of \( S \) vs. \( N \), up to \( N = 108 \). From this figure, it can be seen that the speedup curve is linear with a slope of 0.65 and an intercept of 0.35 on the speedup axis. This is expected since \( S = NE \) and using Eq. 5.4, \( S \) reduces to \( 0.65N + 0.35 \). It should be noted that the slope of the speedup curve is actually equal to the asymptotic efficiency while the intercept is equal to the fraction representing the aforementioned unwanted redundant calculations. The ideal speedup curve, that is, \( S = N \), is also shown in Fig. 5.3 for comparison.

Scalability is usually a study of how \( T_E \) and \( E \) vary with increase in \( N \) for a fixed size problem size. However, with the application developed here, a more interesting scalability study is to consider what happens if, for instance, more SP2 nodes are made available so that more design variables can be included in a design study. This kind of situation will fall under what has been referred to as scaled problem analysis in [116]. In this analysis, the issue is to ensure that the amount of computation performed scales with \( N \) so that \( E \) is kept constant. In order words, the uniprocessor time must increase at the same rate as total parallel time or, equivalently, the amount of productive calculations required must increase at the same rate as the overhead attributed to redundant calculations, communication, and idle time. This implies that as \( N \) increases, \( E \) must remain constant. By simply examining the efficiency studies presented earlier, it is obvious that the parallel CFL3D ADI, as implemented here, is truly scalable, especially in the desired region of large \( N \). This is because once \( E \) reaches the asymptotic value of 65\%, it remains constant there regardless of how large \( N \) is.
Fig. 5.2 Variation of Parallel Efficiency, $E$ with Number of Nodes, $N$
Fig. 5.3 Variation of Speedup, $S$ with Number of Nodes, $N$
The overall execution time for the parallel CFL3D.ADI with 108 design variables (on 108 nodes) is measured to be about 13 CPU hours on the SP2. As mentioned earlier, this is essentially the same as the execution time obtained when CFL3D.ADI was executed for one design variable on one node. This confirms that the overhead due to communication is negligible. In summary, sensitivity derivatives of aerodynamic functions for a 3-D realistic problem with respect to more than 100 design variables were obtained on the SP2 for an execution time of slightly more than half a day. The CPU time limit on the SP2 per job submission is 10 hours. Thus the gradient computation was achieved in two job submissions. Due to the load scheduling policy and extremely heavy work load of the IBM-SP2, the turn around time for the two job submissions totalled about one week.
CHAPTER VI
AERODYNAMIC SHAPE OPTIMIZATION STUDIES WITH PARALLEL 1-D LINE SEARCH

6.1 Introduction

This chapter deals with the development of an overall shape optimization package for realistic three-dimensional aerodynamic geometries. For the gradient-based optimization techniques employed in this work, a new design $X^{r+1}$ at design iteration $r+1$ is obtained from the old design $X^r$ by

$$X^{r+1} = X^r + a^r S^r$$  \hspace{1cm} (6.1)

where $X^r$ and $X^{r+1}$ are the current and the new vectors of shape design variables respectively, and $a^r > 0$ is the move parameter and $S^r$ is a vector of search direction. The design process is initialized at $r = 1$ with $X^1 = X_o = Initial Shape$. From Eq. (6.1), it is apparent that, for a new design, only $a^r$ and $S^r$ need to be calculated, since $X^r$ is known.

Typically, a design optimization process comprises the following steps:

1. Provide $X^r$, a vector of design variables. Initially at $r = 1$, $X^1 = X_o = Initial Shape$.
2. Compute the objective and the constraint functions as well as their gradients.
3. Check for convergence. If satisfied, terminate. Otherwise, continue.
4. Compute a search direction vector $S^r$ in the usable-feasible sector of the design space.
5. Perform a one-dimensional (1-D) line search along $S^r$ to estimate the move parameter $a^r$ which reduces the objective as much as possible without violating any of the constraints.
6. Obtain a new design from Eq. (6.1) and return to step 1.

In the above process, steps 2 to 5 are very significant. For aerodynamic design studies, steps 3 and 4 can be accomplished with any of the existing efficient general-purpose
optimization packages, for example, Automatic Design Synthesis, ADS [95] and Design
Optimization Control/Design Optimization Tools, DOC/DOT [117]. However, steps 2 and 5
demand careful attention. Step 2 usually is both CPU time and memory intensive. In Chaps.
III and IV of this work, an efficient aerodynamic software tool (CFL3D.ADII) was
developed specifically to address step 2. It was demonstrated in Ch. 5 that a parallel version
of this software is capable of yielding the required gradients accurately for a large number of
design variables for a complex 3–D aerodynamic shape (Test Case 2 geometry) within a
reasonable amount of turnaround time. Step 5 also can be quite computationally intensive
because it involves many function evaluations while searching for the optimum \( \alpha' \).
Conventionally, the objective and the constraint functions are evaluated for several proposed
values of \( \alpha \), and a numerical interpolation scheme is then used to determine
\( \alpha' \) which
provides the minimum of the objective in the \( S \) direction without violating any of the
constraints. Usually in practice, an initial \( \alpha \) is proposed and a function evaluation is
performed. If the design is improved, \( \alpha \) is incremented and a new design is obtained. This
process is continued until the optimum design for that particular search direction is trapped
within some required tolerance of \( \alpha' \) (as in the golden–section method), or until a range of \( \alpha \)
is obtained which contains \( \alpha' \) (in which case a polynomial interpolation is then used to obtain
\( \alpha' \)). Either way, the process of locating \( \alpha' \) is by nature highly sequential, since the current \( \alpha \) is
estimated based on the previous one. In design studies using CFD, this implies a series of
CFD solutions computed one after the other, leading to a large amount of execution time.

In the literature on aerodynamic design studies using CFD, a number of approaches have
been employed to mitigate the computational cost associated with steps 2 and 5. The simplest
of these approaches is to select a priori, among the options offered by the optimization
package of choice, a method for computing \( S \) and \( \alpha' \) that will require the smallest number of
gradient and function evaluations to achieve the optimum solution [42]. Another approach is
to use the so called flow prediction or approximate analysis method where a new solution at
the next design point within the 1–D line search part is estimated, at a cheaper cost, by a
truncated first-order-accurate Taylor series expansion of the solution algorithm about the
current known design point [30, 42]. This approach is similar to the reanalysis technique
usually employed in structural design optimization [118]. The approach is made possible
because during the 1-D line search, the changes in $x$ are sometimes small and the new
solution is expected to be close to the current "nearby" solution. The major problem with this
approach is that, after using the approximate analysis for a number of iterations in the 1-D
search loop, the predicted flow field solution begins to deteriorate, leading to very crude
estimates for the values of the objective and the constraint functions, and also possibly poor
convergence rates for the 1-D search process. This problem was overcome in [42] by
performing an exact CFD analysis to update the flow field solution and to provide a new
baseline solution if the 1-D search procedure is not completed after a predetermined number
of approximate flow field analysis. Another problem is that the execution time for the
approximate analysis can become quite high for large, 3-D problems.

6.2 Parallel 1-D Line Search

An alternative strategy designed to reduce the execution time for the 1-D line search
process is presented. The approach exploits the multiprocessor environment offered by the
IBM-SP2 or any similar architecture. The idea is to replace the highly sequential function
evaluations in the 1-D line search with an equivalent parallel one. The strategy proceeds as
follows: The search direction $S$ is computed in the same way as in the sequential approach.
However, after computing $S$, before the optimizer starts the 1-D search process, a number,
say $N$, of CFD solutions are computed. Each CFD solution $p$, where $p$ goes from 1 to $N$,
is computed with a different estimated vector of design variable $pX_{\text{estimate}}$ obtained from

$$pX_{\text{estimate}} = x' + p \alpha_{\text{estimate}}S'$$

(6.2)

where $x'$ is the current design and $p\alpha_{\text{estimate}}S'$ is the step size for solution $p$. How $p\alpha_{\text{estimate}}$
is determined will be discussed later. It is apparent that the $N$ CFD solutions are completely
decoupled and hence can be computed concurrently. This concurrency is exploited on the
IBM-SP2 where each solution $p$ is computed on each node. Thus the $N$ solutions are

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
computed on \( N \) nodes in parallel. There are a total of \( N \) different values of the objective function and \( N \) different values for each of the constraint functions. Thereafter, a suitable polynomial is used to interpolate the values of the objective and the constraints as functions of \( a \), one polynomial for each of the objective and the constraints. In this work, the cubic spline is employed. Once the cubic splines are generated, the optimizer now uses them for the required function evaluations during the 1-D line search. The computational cost of evaluating the splines as well as the cost of obtaining the search direction \( S \) is negligible compared to the cost of a full CFD solution. The result is that the execution time for the 1-D line search is essentially reduced to the execution time for just one CFD analysis. The total CPU time, however, is equal to \( N \times CPU_{analysis} \) where \( CPU_{analysis} \) is the CPU time for one CFD analysis. The target advantage is the substantial reduction in execution time which is gained. The total CPU time may be greater or smaller than that of the sequential approach depending on whether the number of function evaluations is greater or smaller than \( N \) and also on how effective the use of the restart file is in the sequential approach.

For a proper and accurate implementation of the parallel 1-D line search described above, it is important to provide an appropriate \( \alpha_{estimate} \) for the \( p^{th} \) CFD solution. Two issues are involved. The first is finding the maximum value of \( \alpha \), that is \( \alpha_{max} \), that will admit all possible \( \alpha \)'s which the optimizer may propose during the 1-D line search of a design cycle. Using the upper bound \( X^u \), lower bound \( X^l \), current value \( X_i \) and the component \( S_i \) of the search direction for design variable \( i \), the maximum alpha, \( \alpha_{max} \), that will drive this variable to its upper or lower bound for the current design cycle is given by

\[
\alpha_{max}^i = \frac{X^u_i - X_i}{S_i} \quad \text{if} \quad S_i > 0
\]

\[
\alpha_{max}^i = \frac{X^l_i - X_i}{S_i} \quad \text{if} \quad S_i < 0
\]
Considering all design variables, the chosen $a_{\text{max}}$ is the smallest of all $a_{\text{max}}^i (i = 1 \text{ to NDV})$, where NDV is the number of design variables) which, based on Eq. (6.3), will drive some variable to either its upper or lower bound; i.e.,

$$a_{\text{max}} = \min\left[ a_{\text{max}}^i \right], \ i = 1 \text{ to NDV} \quad (6.4)$$

Having determined $a_{\text{max}}$, the second issue is to determine how many nodes (i.e. $N$) need to be used. $a_{\text{estimate}}$ corresponds to a fraction of $a_{\text{max}}$ which node $p$ will utilize. For equally spaced steps, $a_{\text{estimate}} = (\rho - 1)\Delta a$, where $\Delta a = a_{\text{max}}/(N - 1)$. Because of the possible highly nonlinear physics governing the function evaluation, it is required that $\Delta a$ be made small enough such that the polynomial interpolation will adequately represent the actual physics. This is very important because if the objective and the constraints are not evaluated accurately during the 1-D line search, larger number of design cycles, and hence more gradient evaluations, may be required before the optimum can be reached. This is highly undesirable. $\Delta a$ can be made small enough by using a large number of nodes. However, this is not recommended because it may be difficult to procure the required number of nodes. Even if enough nodes are available, the total CPU time will be increased unnecessarily without any further decrease in execution time. Therefore, instead of increasing the number of nodes, the overall optimization problem should be set up so that the upper and lower bounds of the design variables are scaled to small values. If this is done, then with proper scaling of the search direction vector $S$ by the optimizer, it is guaranteed from Eq. (6.3) that $a_{\text{max}}$ will be correspondingly small. Thus for accurate polynomial interpolation, only a few CFD solutions (hence only a few IBM–SP2 nodes) will be needed since $\Delta a = a_{\text{max}}/(N - 1)$ will be small enough to produce the required accuracy even when $N$ is small. For the design studies performed here, $a_{\text{max}}$ is of $O(1)$, $N$ is of $O(10)$ and hence $\Delta a$ is of $O(0.1)$.

The parallel 1-D line search approach just described is incorporated into an overall design package that includes the parallel CFL3D·ADII of Chap. V for flow and flow–gradient computation, and the general purpose optimization software ADS [95] for the
overall coordination of the design process. Section 6.3 presents details of the design studies subsequently performed with the new software.

6.3 Design Optimization Studies

To demonstrate the capabilities of the developed software, two design studies were carried out on the same geometry. The geometry is the proprietary wing/body configuration (Test Case 2) whose parametrization has been described in Sec. 4.2.2 of Chap. IV. The flow regime considered is inviscid supersonic cruise at Mach number $M_{\infty} = 2.4$ and angle of attack $\alpha = 1.9^\circ$. The grid size is 210,177 points. The design optimization problem for both studies is formulated as follows

$$\text{minimize} \quad \frac{C_D}{C_{D_b}}$$

subject to

$$\frac{C_L}{C_{L_b}} \geq 1$$

and

$$\frac{|CM_{X_i}|}{|CM_{X_b}|} \leq 1$$

Side Constraints on design variables $-0.125 \leq X_i \leq 0.125 \quad i = 1, NDV$

The subscript B implies "baseline" values. In order words, the objective is to minimize drag while constraining the lift coefficient and the magnitude of the wing-root bending moment to their baseline values. Also, side constraints are imposed to specify the lower and the upper bounds for the vector of design variables. For both design studies, the search direction was obtained from ADS using the sequential quadratic programming as the strategy and the modified method of feasible directions for constrained optimization as the optimizer. For the 1-D line search (with the cubic spline polynomials representing the CFD analysis code), the option chosen is polynomial interpolation after bounds have first been established for the aerodynamic functions. This combination of strategy, optimizer and 1-D search is recommended if the analysis for computing functions is iterative and if function and
gradient evaluations are expensive [95]. The theoretical framework for ADS can mostly be found in [119].

### 6.3.1 Design 1: 108 Design Variables

For the first design study, hereafter referred to as design 1, the same vector of 108 design variables used for the code validation of the parallel CFL3D.ADII (see Chap. V) are considered. It will be recalled that this vector of design variables are made up of 12 twist functions at the wing trailing edge, 48 wing section camber functions and 48 wing section thickness functions. These variables are effective over 28 wing sections (out of a total of 32) located at spanwise stations 5–32. The first 4 wing sections that are excluded from the design process are those closest to the fuselage because an appropriate parametrization that will allow smooth blending between the fuselage and the wing root region during the design process was not available at the time of this work. All design variables have initial values equal to zero. For all design cycles, 108 IBM–SP2 nodes were employed to compute the required gradients for the 108 design variables concurrently, 1 node/design variable.

The result of the optimization study for design 1 after two optimization cycles are shown in Table 6.1. The initial and final values of the the drag coefficient $C_D$, the lift coefficient $C_L$ and the wing-root bending moment $CM_x$ are presented. For the parallel 1-D search, 10 SP2 nodes are employed to compute ten CFD analyses, each analysis with its own vector of design variables obtained from Eq. (6.2). The ten objective and constraint functions that are now available are then interpolated with natural cubic splines, one spline polynomial for each aerodynamic coefficient of interest. These spline functions then replace the usual CFD analysis during the 1-D line search. Table 6.1 also contains results obtained using the traditional sequential 1-D line search for comparison. Table 6.2 shows the final values of 10 randomly selected design variables from the parallel and the sequential 1-D searches. The result for the entire 108 design variables are given in Table B.1 of Appendix B. From Tables 6.1, 6.2 and B.1, it is apparent that the parallel and the sequential 1-D line search methods...
Table 6.1 Design Improvement Summary for Design 1 (108 Design Variables: Camber, Thickness and Twist). Results from Sequential 1-D Search included for Comparison. Results Normalized with Baseline Values

<table>
<thead>
<tr>
<th></th>
<th>Initial</th>
<th>Final Parallel</th>
<th>Final Sequential</th>
<th>% Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective ($C_D$)</td>
<td>1.00000</td>
<td>0.89424</td>
<td>0.89424</td>
<td>-10.6*</td>
</tr>
<tr>
<td>Constraint 1 ($C_L$)</td>
<td>1.00000</td>
<td>1.00200</td>
<td>1.00200</td>
<td>+0.2</td>
</tr>
<tr>
<td>Constraint 2 ($IC_{Mx}$)</td>
<td>1.00000</td>
<td>0.99780</td>
<td>0.99780</td>
<td>-0.2</td>
</tr>
</tbody>
</table>

* % Reduction in drag too optimistic because wing is structurally unacceptable

Table 6.2 Final Values of 10 Randomly Selected Design Variables for Design 1. Results from Sequential 1-D Search included for Comparison

<table>
<thead>
<tr>
<th>DV</th>
<th>$Value_{sequential}$</th>
<th>$Value_{parallel}$</th>
<th>$\frac{Value_{parallel}}{Value_{sequential}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.12422E+00</td>
<td>0.12422E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>6</td>
<td>-0.12073E+00</td>
<td>-0.12073E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>12</td>
<td>-0.12079E+00</td>
<td>-0.12079E+00</td>
<td>0.99998E+00</td>
</tr>
<tr>
<td>13</td>
<td>0.28155E-01</td>
<td>0.28154E-01</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>36</td>
<td>-0.40791E-01</td>
<td>-0.40794E-01</td>
<td>0.10001E+01</td>
</tr>
<tr>
<td>60</td>
<td>-0.42492E-01</td>
<td>-0.42495E-01</td>
<td>0.10001E+01</td>
</tr>
<tr>
<td>61</td>
<td>-0.12422E+00</td>
<td>-0.12422E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>82</td>
<td>-0.11811E+00</td>
<td>-0.11811E+00</td>
<td>0.10000E+01</td>
</tr>
<tr>
<td>84</td>
<td>-0.11830E+00</td>
<td>-0.11830E+00</td>
<td>0.99996E+00</td>
</tr>
</tbody>
</table>
gave essentially identical results. Thus for this problem, there is no loss of accuracy in using the parallel 1-D search approach.

After the second design cycle, the drag has been reduced by about 10.7% (7.2 counts) while the lift and the root bending moment coefficients are within 0.2% of their baseline values. Further progress could not be made because, as will be explained later, the wing geometry became physically unreasonable. Figure 6.1 shows the chordwise variation of the percentage change in mean camber line (Z) for the affected wing airfoil sections 5–32. It can be observed that the largest changes (up to 7%) occur at the wing sections in the inboard half of the wing. The change reduces gradually until it becomes very small (about −0.5% to 0.5%) as the wing tip is approached. This observation is reinforced in Fig. 6.2 which shows the spanwise variation of Z at 25%, 50%, and 75% chord locations. These results suggest that, if the four sections at the wing root have not been constrained, they most probably will experience the largest change. To be able to do this, the fuselage camber will need to be included in the optimization process.

Figure 6.3 shows the chordwise variation of percentage change in the thickness for the affected sections. It is apparent that over the entire range, there is generally substantial reductions in thickness, up to almost 90% in the trailing edge region of some of the sections e.g. Fig. 6.3 c–f. The reduction in drag reported earlier (see Table 6.1) was largely a result of the thickness reduction, although there was some contribution from the shifts in the mean camber lines of the airfoil sections. Significant drag reduction when wing thickness is reduced, as observed here, is actually expected for inviscid supersonic cruise design. The substantial changes in thickness is due to lack of proper constraint on the wing volume. The outcome of this design is a wing that is not structurally acceptable. Thus the drag reduction obtained is simply too optimistic.
Fig. 6.1 Chordwise Variation of the Percentage Change in Mean Camber Line, $Z$ for Design 1 (108 Design Variables, Camber, Thickness & Twist)
Fig. 6.1 % Change in Z for Design 1 Continued
Fig. 6.1 % Change in Z for Design 1 Continued
Fig. 6.1 % Change in Z for Design 1 Continued
Fig. 6.1 % Change in Z for Design 1 Continued
Fig. 6.1 % Change in Z for Design 1 Continued
Fig. 6.1 % Change in Z for Design 1 Continued
Fig. 6.2 Spanwise Variation of the Percentage Change in Mean Camber Line, Z for Design 1 (108 Design Variables, Camber, Thickness & Twist)
Fig. 6.3 Chordwise Variation of the Percentage Change in Thickness for Design 1 (108 Design Variables, Camber, Thickness & Twist)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Fig. 6.3 % Change in Thickness for Design 1 Continued
Fig. 6.3 % Change in Thickness for Design 1 Continued
Fig. 6.3 % Change in Thickness for Design 1 Continued
Fig. 6.3 % Change in Thickness for Design 1 Continued
Fig. 6.3 % Change in Thickness for Design 1 Continued
Fig. 6.3 % Change in Thickness for Design 1 Continued
The overall execution time for both design cycles on the SP2 is about 34 hours, that is, an average of 17 hours per cycle. The gradient computation takes about 13 hours while the parallel 1-D line search takes about 4 hours. The gradient computation was achieved in two job submissions on the SP2 since the maximum available CPU time per submission is 10 hours. The execution time for computing the search direction, checking convergence and evaluating the cubic splines are negligible. For the 1-D line search method used, the optimizer makes five calls to the cubic spline polynomials during each design cycle. This implies five function evaluations per cycle. This is confirmed by the sequential 1-D search on the Cray--YMP which also goes through five function evaluations (CFD analyses) per cycle. The five function evaluations of the sequential 1-D search, with the use of the restart file, take a total of about 420 multigrid cycles and a CPU time of about 2 Cray--YMP hours. With an average of 81 CPU seconds per cycle on a node of the IBM--SP2, the sequential 1-D search would have taken about 10 hours. Compared to 4 hours for the parallel 1-D search method, the latter provides a speedup of about 2.5 for the 1-D search part alone. The parallel 1-D line search will be even more useful for the 1-D search options where there is a large number of function evaluations per design cycle (e.g., the popular golden section method). This is because, regardless of the 1-D search method, the execution time will remain approximately constant (since the spline polynomials cost virtually nothing to evaluate) for the parallel 1-D search method. On the other hand, the execution time for the sequential 1-D search method increases with the number of function evaluations.

The turnaround time for the two design cycles on the IBM--SP2 was about 2 weeks, even though the execution time was only about 34 hours. The reason for these was because the gradient evaluation required access to 108 nodes at the same time. These many nodes are currently available for a single user usually only on weekends due to the load scheduling policy and heavy work load of the NAS IBM--SP2. Thus it takes a weekend to complete a design cycle. Even then, this turn around time is good considering the size of the problem. The final result is that the possibility of performing aerodynamic design optimization for
3-D complex geometries using CFD/discrete sensitivity analysis with a large number of design variables has been demonstrated. This was the primary objective of this work.

6.3.2 Design 2: 60 Design Variables, Camber and Twist

The second design study, henceforth referred to as design 2, is the same as design 1 except that only the first 60 design variables are considered. Thus the thickness effect which is associated with the last 48 design variables of design 1 is eliminated. The goal was to perform optimization studies that will give realistic geometric results. The 60 design variables affect only the wing twist and the wing sections camber. The lower and upper bounds for all design variables were -0.1 and 0.1 respectively. A total of five design cycles were completed. A plot of $C_D$, $C_L$ and $CM_x$, normalized with their baseline values, versus the number of optimization cycles is shown in Fig. 6.4. As can be seen from this figure, it is not economical from a computational point-of-view to go beyond the fifth design cycle, because the relative change in objective function is just about 0.1% from the fourth to the fifth cycle. Table 6.3 shows the initial and final values of the objective and the constraints. There was a drag reduction of only about 1.8% (1.2 counts). $C_L$ is constrained to within 0.02% and $CM_x$ to within 0.2%. Table 6.4 shows the final values of ten of the design variables. The results for the entire 60 design variables are given in Table B.2 of Appendix B. The drag reduction shown in Table 6.3 is brought about primarily by the changes in the mean camber line, $Z$, of the wing airfoil sections. This is illustrated in Fig. 6.5, where the percentage change in $Z$ is plotted against normalized chord for all of the affected airfoil sections 5–32. Like in the previous design, the largest changes in $Z$ (up to about 4.5%) occur in the inboard part of the wing. $Z$ then gradually reduces as the wing tip is approached. Close to the wing tip, $Z$ changes by as little as about 0.5%. Fig. 6.6 which shows the spanwise variation of $Z$ at the 25%, 50% and 75% chord locations confirms the observation just discussed. Also from Fig. 6.5, it can be observed that camber perturbations are the largest in the aft regions of the inboard and mid-span wing sections. For the locations in the outboard region close to the wing tip, the wing sections tend to translate upward. The observed camber
Fig. 6.4 Optimization History of the Normalized Objective \((C_D/C_{D*})\) and Normalized Constraints \((C_L/C_{L*} \& CM_X/CM_{X*})\) for Design 2

(60 Design Variables, Camber & Twist)
Table 6.3 Design Improvement Summary for Design 2 (60 Design Variables: Camber and Twist). Results Normalized with Baseline Values

<table>
<thead>
<tr>
<th></th>
<th>Initial</th>
<th>Final</th>
<th>% Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective ($C_D$)</td>
<td>1.00000</td>
<td>0.98224</td>
<td>1.78</td>
</tr>
<tr>
<td>Constraint 1 ($C_L$)</td>
<td>1.00000</td>
<td>0.99980</td>
<td>-0.02</td>
</tr>
<tr>
<td>Constraint 2 ($ICM_{x1}$)</td>
<td>1.00000</td>
<td>0.99776</td>
<td>-0.22</td>
</tr>
</tbody>
</table>

Table 6.4 Final Values of 10 Randomly Selected Design Variables for Design 2

<table>
<thead>
<tr>
<th>Design Variable</th>
<th>Final Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.8103700E−01</td>
</tr>
<tr>
<td>6</td>
<td>0.7317600E−01</td>
</tr>
<tr>
<td>12</td>
<td>0.4154100E−01</td>
</tr>
<tr>
<td>18</td>
<td>0.8472000E−01</td>
</tr>
<tr>
<td>24</td>
<td>0.1590600E−01</td>
</tr>
<tr>
<td>30</td>
<td>0.8255700E−01</td>
</tr>
<tr>
<td>36</td>
<td>0.5036600E−01</td>
</tr>
<tr>
<td>42</td>
<td>-0.7051700E−01</td>
</tr>
<tr>
<td>48</td>
<td>-0.1681500E−01</td>
</tr>
<tr>
<td>60</td>
<td>0.6695300E−01</td>
</tr>
</tbody>
</table>
Fig. 6.5 Chordwise Variation of the Percentage Change in Mean Camber Line, $Z$ for Design 2 (60 Design Variables, Camber & Twist)
Fig. 6.5 % Change in Z for Design 2 Continued
Fig. 6.5 % Change in Z for Design 2 Continued
Fig. 6.5 % Change in Z for Design 2 Continued
Fig. 6.5 % Change in Z for Design 2 Continued
Fig. 6.5 % Change in $Z$ for Design 2 Continued
Fig. 6.5 % Change in Z for Design 2 Continued
Fig. 6.6 Spanwise Variation of the Percentage Change in Mean Camber Line, Z for Design 2 (60 Design Variables, Camber & Twist)
changes for all the airfoil sections indicate that the constraint at the wing trailing edge needs to be somewhat relaxed to allow possible upward movement, as the optimizer is suggesting. The reduction in drag is not as dramatic as in the first design because the thickness of the airfoil section is constrained to their baseline values and thus the wing airfoil sections are not allowed to become thinner. This fact is illustrated in Fig. 6.7, which shows that plots of the percentage change in thickness versus the normalized chord for all the sections are essentially flat (at a value of zero).

The average execution time for each design cycle is roughly the same as that of design case 1; i.e., about 17 hours. Thus the total execution time for the five design cycles is about 85 hours. Note that the execution time per design cycle stays about the same regardless of the number of design variables. This is because the execution time for the bulk part of the computation (that is parallel gradient computation and parallel 1-D search) is essentially independent of the number of design variables, as discussed in Chap. V. Therefore, varying the number of design variables has no significant effect on the execution time. The number of design variables however has significant impact on the turnaround time. For example, because the number of design variables is smaller (60 compared to 108), it became much easier to secure enough IBM-SP2 nodes for the gradient computations, even during the week days. The result was that the turn around time went down to only about two days per design cycle. Thus the entire five cycles were completed in about 10 days, even on the busy NAS IBM-SP2. This is remarkable for a problem as big as the one considered here, where an advanced CFD code was used for 3-D aerodynamic analysis as well as discrete sensitivity analysis.
Fig. 6.7 Chordwise Variation of the Percentage Change in Thickness for Design 2 (60 Design Variables, Camber & Twist)
Fig. 6.7 % Change in Thickness for Design 2 Continued
Fig. 6.7 % Change in Thickness for Design 2 Continued
Fig. 6.7 % Change in Thickness for Design 2 Continued
Fig. 6.7 % Change in Thickness for Design 2 Continued

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Fig. 6.7 % Change in Thickness for Design 2 Continued
Fig. 6.7 % Change in Thickness for Design 2 Continued
CHAPTER VII

CONCLUSIONS AND RECOMMENDATIONS

7.1 CFL3D.ADII

A hybrid automatic differentiation/incremental iterative method was implemented in the general purpose advanced computational fluid dynamics code, CFL3D Version 4.1. The automatic differentiation tool employed is ADIFOR. The resulting code, referred to as CFL3D.ADII, was found to yield accurate, consistently discrete, first-order sensitivity derivatives for complex geometries. The code retains all the useful features and capabilities of the original CFL3D flow analysis code. The only exception is the unsteady capability; that is, unsteady flow sensitivity derivatives cannot yet be evaluated. In other words, CFL3D.ADII is capable of accurately computing the steady state, geometric sensitivity derivatives for very complex geometries represented by multiblock, general patched grids, overlapped grids and even embedded grids. Furthermore, the new code maintains the same algorithm capabilities for efficiently solving the linear sensitivity equations that are featured in the original CFL3D code for solving the nonlinear flow equations. These algorithm features include the three-factor, spatially-split, approximate-factorization procedure with a choice of either the efficient diagonalized scheme of Roe or the block-tridiagonal inversion scheme. In addition, multigrid and/or mesh sequencing is retained for significantly accelerated solution of the flow-sensitivity equations.

A first version of CFL3D.ADII was found to be computationally inefficient, especially on vector supercomputers, when compared with the original highly vectorized CFL3D code. Despite these inefficiency, the first version performed as well as a carefully applied method of finite differences from the CPU time standpoint for a realistic problem, apart from being more reliable.

Based on the experience with the first version, a new version was developed which has improved vectorization. The result was a version that was significantly, and for some code options, dramatically better than the first version, especially on vector computers. Compared
to the finite difference method, the second version was about 40% faster for a realistic 3-D problem.

Although the second version of the CFL3D.ADII was significantly better than the first, there are still several issues that require more thorough investigation. An example is the conjecture proposed by Taylor [37] which says that, since the sensitivity analysis is performed starting with converged nonlinear flow residuals, the update on the flow solution can be terminated. This should allow the sensitivity analysis to proceed with a larger CFL number, and hence improved convergence rate. This issue was examined in CFL3D.ADII only briefly and no conclusions could be drawn yet. However, because of its promise for efficiency improvement, it should be investigated further.

Another issue is turbulent-flow sensitivity analysis. So far, most of the fine tuning done on CFL3D.ADII concerned only inviscid flow governed by the Euler equations. Although accurate sensitivities have been obtained for turbulent flow with CFL3D.ADII [37], most of the turbulent-flow sensitivity analysis subroutines are yet to be fully optimized. This is recommended for future work.

Finally, the CFL3D.ADII can currently compute only steady-state sensitivity derivatives with respect to only geometric design variables. In some applications, it is necessary to perform sensitivity analysis with non-geometric design variables and for unsteady problems. Future improvement of CFL3D.ADII should include these capabilities.

### 7.2 Parallel CFL3D.ADII

A parallel version of the CFL3D.ADII was developed based on the “derivative stripmining” approach of [58]. In this approach, a copy of CFL3D.ADII operates on a predetermined subset of the design variables, that is the approach is data parallel. The parallel version is suitable for implementation on distributed-memory parallel computers like the IBM-SP2 or even a cluster of workstations. The interprocessor communication was achieved via the Message Passing Interface (MPI) standard. Because of the size of the
problem considered, only one design variable could be assigned to a node of the IBM-SP2 (which was the parallel computer employed).

To avoid excessive communication overhead, the large data files required by each computer node are made local to each node rather than by message passing. In addition, the implementation directly "mimics" the single node application, where all large data files are accessed by open/read/write operations. For the test case considered, the parallel CFL3D.ADII computing sensitivity derivatives for \( N \) design variables using \( N \) SP2 nodes has the same execution time as the serial CFL3D.ADII computing sensitivity derivatives for 1 design variable on 1 node. This was because the communication time was negligible. The only communication among the nodes is limited to gathering to node 0 the computed sensitivity derivatives and the residual history data for all design variables. The overall execution time for the test case considered was about 13 hours on the SP2. This execution time remained constant, regardless of the number of design variables, as long as the same number of design variables are assigned to each of the SP2 nodes.

In CFL3D.ADII, a fraction of the overall computation is duplicated on all the nodes. With this fraction taken into consideration, the parallel CFL3D.ADII has an asymptotic computational efficiency \( E \) that is equal to \( 1 - C_{\text{rep}} \) for large \( N (N \geq 15) \), where \( C_{\text{rep}} \) is the fraction of replicated computation and \( N \) is the number of design variables. Note that \( E \) is independent of \( N \). For the test case and code option considered, \( C_{\text{rep}} \) was about 0.35. Thus \( E \) was 0.65. Because \( E \) is bounded from below asymptotically for reasonably large \( N \), the parallel CFL3D.ADII is scalable and has a linear speedup curve with a slope that is equal to \( E \) and an intercept that is equal to \( C_{\text{rep}} \).

The parallel implementation of CFL3D.ADII is coarse grain. The problem is partitioned based on design variables, but not on the geometrical domain. Future efforts for larger problems should involve modifying the CFL3D.ADII so that partitioning can be done based on design variables as well as the geometrical domain. This will increase the percentage of extractable parallelism with a corresponding reduction in \( C_{\text{rep}} \), though with extra

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
communication overhead. This is a subject of proposed future extensions of the present work.

7.3 Parallel 1–D Line Search

To mitigate the large execution time associated with the sequential 1–D line search in gradient–based aerodynamic design optimization, an alternative parallel approach suitable for implementation on a distributed–memory multiprocessor platform was developed. The execution time of the new approach was reduced to just about that of one flow analysis, regardless of the number of function evaluations in the 1–D search. The new approach was found to yield design results that are essentially identical to those obtained from the traditional sequential approach. For the problem considered and the choice of 1–D search option (polynomial interpolation after bounds have been established for the aerodynamic functions), the new parallel approach gave a speedup of about 2.5 over the sequential approach. The potential saving in execution time could be much higher for 1–D search options that usually involve a large number of function evaluations, like the golden–section method, since the wall–clock time of the parallel approach is independent of the number of function evaluations; in contrast, the cost and turnaround time of the sequential approach grows with the number of function evaluations.

A design optimization package which incorporates the parallel CFL3D.ADII for gradient evaluations and the parallel 1–D line search required about 17 hours execution time per design cycle on the IBM–SP2 for the test case considered. Provided that there are as many computer nodes as the design variables, the execution time per design cycle using this optimization package is effectively independent of the number of design variables. However, the overall turnaround time, at least on the NAS IBM–SP2, is affected by the number of design variables, and hence the number of nodes demanded by a problem. The larger the number of design variables, the longer it takes to secure enough nodes and the larger the turnaround time. For a design case with 108 design variables, the turnaround time
was about one week per design cycle. For a second case with only 60 design variables, the turnaround time went down to only about two days per design cycle.

The parallel 1-D search approach proceeds (after the search direction has been calculated) by first computing the maximum value of the move parameter \( a \) that will cover the entire range of the move parameters which the optimizer may propose during the 1-D search process. The maximum move parameter is then divided into a predetermined number of steps, each step is used to compute a different vector of design variables. The flow analysis code is then used to perform function evaluation for each vector of design variables. Since the function evaluations for the vectors of design variables are decoupled, all function evaluations can be computed concurrently on a parallel computer. The evaluated functions are then fitted with natural cubic splines which serve as a substitute for the CFD flow analysis code during the 1-D search process. When an actual move parameter is computed, the optimizer calls the splines to perform the required function evaluation. In this manner, the more typical time-consuming sequential function evaluation is avoided.

7.4 Specific Contributions of this Study

The central objective of this work was to formulate an efficient procedure which is suitable for aerodynamic shape optimization of complex 3-D geometries represented by a large number of grid points and a large number of design variables using an advanced sensitivity-enhanced CFD code for both flow analysis and discrete shape flow sensitivity analysis. As detailed above, this objective was achieved through (i) the development of the CFL3D.ADII discrete shape sensitivity analysis code and its subsequent implementation on a distributed-memory parallel computer, and (ii) the development of a parallel 1-D line search to replace the typical time-consuming sequential 1-D line search in design optimization. The developed procedure was subsequently demonstrated in the design improvement studies of a realistic proprietary 3-D High Speed Civil Transport (HSCT) wing/body geometry represented by over 200,000 grid points and over 100 design variables. The flow physics was inviscid represented by the 3-D Euler equations.
REFERENCES


APPENDIX A

SENSITIVITY DERIVATIVES FROM PARALLEL CFL3D.ADII

This appendix presents the sensitivity derivatives of the lift coefficient, the drag coefficient and x-moment (root bending moment) coefficient computed with the parallel CFL3D.ADII for the full 108 design variables of the proprietary HSCT wing/body configuration used in Chap. V.

Table A.1 Sensitivity Derivatives from the Parallel Version of CFL3D.ADII for 108 Design Variables (DV). $C_L \equiv$ Lift Coefficient, $C_D \equiv$ Drag Coefficient, $C_{Mx} \equiv X$-Moment (or Root Bending Moment) Coefficient

<table>
<thead>
<tr>
<th>DV</th>
<th>$\frac{\partial C_L}{\partial DV}$</th>
<th>$\frac{\partial C_D}{\partial DV}$</th>
<th>$\frac{\partial C_{Mx}}{\partial DV}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.34696E-03</td>
<td>0.32437E-04</td>
<td>-0.10300E-03</td>
</tr>
<tr>
<td>2</td>
<td>0.26484E-03</td>
<td>0.27295E-04</td>
<td>-0.74137E-04</td>
</tr>
<tr>
<td>3</td>
<td>0.24403E-03</td>
<td>0.26207E-04</td>
<td>-0.73619E-04</td>
</tr>
<tr>
<td>4</td>
<td>0.23184E-03</td>
<td>0.24145E-04</td>
<td>-0.75095E-04</td>
</tr>
<tr>
<td>5</td>
<td>0.19142E-03</td>
<td>0.19196E-04</td>
<td>-0.68293E-04</td>
</tr>
<tr>
<td>6</td>
<td>0.18439E-03</td>
<td>0.19762E-04</td>
<td>-0.75024E-04</td>
</tr>
<tr>
<td>7</td>
<td>0.13071E-03</td>
<td>0.15368E-04</td>
<td>-0.62343E-04</td>
</tr>
<tr>
<td>8</td>
<td>0.10576E-03</td>
<td>0.12749E-04</td>
<td>-0.58964E-04</td>
</tr>
<tr>
<td>9</td>
<td>0.10128E-03</td>
<td>0.12028E-04</td>
<td>-0.64664E-04</td>
</tr>
<tr>
<td>10</td>
<td>0.96899E-04</td>
<td>0.11181E-04</td>
<td>-0.70060E-04</td>
</tr>
<tr>
<td>11</td>
<td>0.10811E-03</td>
<td>0.11854E-04</td>
<td>-0.87892E-04</td>
</tr>
<tr>
<td>12</td>
<td>0.39360E-04</td>
<td>0.45047E-05</td>
<td>-0.32818E-04</td>
</tr>
<tr>
<td>13</td>
<td>0.46031E-02</td>
<td>0.62859E-03</td>
<td>-0.89476E-03</td>
</tr>
<tr>
<td>14</td>
<td>0.23521E-02</td>
<td>0.34973E-04</td>
<td>-0.30179E-03</td>
</tr>
<tr>
<td>15</td>
<td>0.98533E-04</td>
<td>-0.16508E-03</td>
<td>-0.88524E-05</td>
</tr>
<tr>
<td>16</td>
<td>-0.99955E-03</td>
<td>-0.28808E-04</td>
<td>0.45762E-03</td>
</tr>
</tbody>
</table>

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Table A.1 Continued

<table>
<thead>
<tr>
<th>DV</th>
<th>$\frac{\partial C_L}{\partial DV}$</th>
<th>$\frac{\partial C_D}{\partial DV}$</th>
<th>$\frac{\partial C_{M_x}}{\partial DV}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>0.32306E-02</td>
<td>0.45909E-03</td>
<td>-0.69031E-03</td>
</tr>
<tr>
<td>18</td>
<td>0.20668E-02</td>
<td>0.32721E-04</td>
<td>-0.57990E-03</td>
</tr>
<tr>
<td>19</td>
<td>-0.33399E-03</td>
<td>-0.14383E-03</td>
<td>0.40168E-03</td>
</tr>
<tr>
<td>20</td>
<td>-0.82114E-03</td>
<td>-0.52851E-04</td>
<td>0.37464E-03</td>
</tr>
<tr>
<td>21</td>
<td>0.31765E-02</td>
<td>0.45322E-03</td>
<td>-0.85339E-03</td>
</tr>
<tr>
<td>22</td>
<td>0.20608E-02</td>
<td>0.63137E-04</td>
<td>-0.73525E-03</td>
</tr>
<tr>
<td>23</td>
<td>-0.66982E-03</td>
<td>-0.14542E-03</td>
<td>0.54203E-03</td>
</tr>
<tr>
<td>24</td>
<td>-0.58057E-03</td>
<td>-0.60691E-04</td>
<td>0.23386E-03</td>
</tr>
<tr>
<td>25</td>
<td>0.32041E-02</td>
<td>0.44589E-03</td>
<td>-0.95777E-03</td>
</tr>
<tr>
<td>26</td>
<td>0.16489E-02</td>
<td>0.63332E-04</td>
<td>-0.66430E-03</td>
</tr>
<tr>
<td>27</td>
<td>-0.74809E-03</td>
<td>-0.15521E-03</td>
<td>0.44512E-03</td>
</tr>
<tr>
<td>28</td>
<td>-0.37953E-03</td>
<td>-0.63960E-04</td>
<td>0.19373E-03</td>
</tr>
<tr>
<td>29</td>
<td>0.25451E-02</td>
<td>0.36447E-03</td>
<td>-0.83358E-03</td>
</tr>
<tr>
<td>30</td>
<td>0.97191E-03</td>
<td>0.39709E-04</td>
<td>-0.48632E-03</td>
</tr>
<tr>
<td>31</td>
<td>-0.54514E-03</td>
<td>-0.14741E-03</td>
<td>0.27890E-03</td>
</tr>
<tr>
<td>32</td>
<td>-0.23453E-03</td>
<td>-0.51351E-04</td>
<td>0.15701E-03</td>
</tr>
<tr>
<td>33</td>
<td>0.22748E-02</td>
<td>0.37265E-03</td>
<td>-0.87154E-03</td>
</tr>
<tr>
<td>34</td>
<td>0.71285E-03</td>
<td>0.36644E-04</td>
<td>-0.43288E-03</td>
</tr>
<tr>
<td>35</td>
<td>-0.43357E-03</td>
<td>-0.17798E-03</td>
<td>0.20824E-03</td>
</tr>
<tr>
<td>36</td>
<td>-0.20718E-03</td>
<td>-0.53622E-04</td>
<td>0.16085E-03</td>
</tr>
<tr>
<td>37</td>
<td>0.15684E-02</td>
<td>0.28468E-03</td>
<td>-0.71980E-03</td>
</tr>
<tr>
<td>38</td>
<td>0.40623E-03</td>
<td>0.37890E-04</td>
<td>-0.22884E-03</td>
</tr>
<tr>
<td>39</td>
<td>-0.34450E-03</td>
<td>-0.14204E-03</td>
<td>0.18394E-03</td>
</tr>
<tr>
<td>40</td>
<td>-0.13098E-03</td>
<td>-0.58660E-04</td>
<td>0.74314E-04</td>
</tr>
<tr>
<td>41</td>
<td>0.12133E-02</td>
<td>0.22399E-03</td>
<td>-0.64264E-03</td>
</tr>
<tr>
<td>42</td>
<td>0.24712E-03</td>
<td>0.38210E-04</td>
<td>-0.11965E-03</td>
</tr>
<tr>
<td>DV</td>
<td>$\frac{\partial C_L}{\partial DV}$</td>
<td>$\frac{\partial C_D}{\partial DV}$</td>
<td>$\frac{\partial C_M}{\partial DV}$</td>
</tr>
<tr>
<td>-----</td>
<td>----------------------------------</td>
<td>----------------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>43</td>
<td>-0.33663E-03</td>
<td>-0.91003E-04</td>
<td>0.23189E-03</td>
</tr>
<tr>
<td>44</td>
<td>-0.10685E-03</td>
<td>-0.45165E-04</td>
<td>0.60973E-04</td>
</tr>
<tr>
<td>45</td>
<td>0.93610E-03</td>
<td>0.16967E-03</td>
<td>-0.56652E-03</td>
</tr>
<tr>
<td>46</td>
<td>0.16233E-03</td>
<td>0.30138E-04</td>
<td>-0.75670E-04</td>
</tr>
<tr>
<td>47</td>
<td>-0.30576E-03</td>
<td>-0.58628E-04</td>
<td>0.24527E-03</td>
</tr>
<tr>
<td>48</td>
<td>-0.12260E-03</td>
<td>-0.31035E-04</td>
<td>0.92613E-04</td>
</tr>
<tr>
<td>49</td>
<td>0.73538E-03</td>
<td>0.47754E-04</td>
<td>-0.59624E-03</td>
</tr>
<tr>
<td>50</td>
<td>0.15155E-03</td>
<td>0.86204E-05</td>
<td>-0.92908E-04</td>
</tr>
<tr>
<td>51</td>
<td>-0.25897E-03</td>
<td>-0.36439E-04</td>
<td>0.24492E-03</td>
</tr>
<tr>
<td>52</td>
<td>-0.15732E-03</td>
<td>-0.27447E-04</td>
<td>0.13602E-03</td>
</tr>
<tr>
<td>53</td>
<td>0.94186E-03</td>
<td>-0.35943E-04</td>
<td>-0.93589E-03</td>
</tr>
<tr>
<td>54</td>
<td>0.33724E-03</td>
<td>-0.20054E-04</td>
<td>-0.32244E-03</td>
</tr>
<tr>
<td>55</td>
<td>-0.19476E-03</td>
<td>-0.36048E-04</td>
<td>0.19667E-03</td>
</tr>
<tr>
<td>56</td>
<td>-0.19971E-03</td>
<td>-0.27734E-04</td>
<td>0.18942E-03</td>
</tr>
<tr>
<td>57</td>
<td>0.24763E-03</td>
<td>-0.41872E-04</td>
<td>-0.28563E-03</td>
</tr>
<tr>
<td>58</td>
<td>0.13098E-03</td>
<td>-0.24897E-04</td>
<td>-0.13290E-03</td>
</tr>
<tr>
<td>59</td>
<td>0.10131E-04</td>
<td>-0.17964E-04</td>
<td>0.56340E-05</td>
</tr>
<tr>
<td>60</td>
<td>-0.50851E-04</td>
<td>-0.12042E-04</td>
<td>0.59000E-04</td>
</tr>
<tr>
<td>61</td>
<td>0.38792E-03</td>
<td>0.27860E-03</td>
<td>-0.65748E-04</td>
</tr>
<tr>
<td>62</td>
<td>-0.21361E-03</td>
<td>0.29032E-05</td>
<td>-0.10456E-04</td>
</tr>
<tr>
<td>63</td>
<td>0.81145E-03</td>
<td>0.51966E-03</td>
<td>-0.77639E-03</td>
</tr>
<tr>
<td>64</td>
<td>-0.15861E-02</td>
<td>0.21678E-03</td>
<td>0.27859E-03</td>
</tr>
<tr>
<td>65</td>
<td>0.60940E-03</td>
<td>0.22800E-03</td>
<td>-0.53481E-04</td>
</tr>
<tr>
<td>66</td>
<td>-0.35366E-03</td>
<td>0.29734E-04</td>
<td>0.17819E-03</td>
</tr>
<tr>
<td>67</td>
<td>-0.36665E-03</td>
<td>0.42161E-03</td>
<td>-0.24925E-03</td>
</tr>
<tr>
<td>68</td>
<td>-0.63135E-03</td>
<td>0.17710E-03</td>
<td>0.67566E-04</td>
</tr>
<tr>
<td>DV</td>
<td>( \frac{\partial C_L}{\partial DV} )</td>
<td>( \frac{\partial C_D}{\partial DV} )</td>
<td>( \frac{\partial C_M}{\partial DV} )</td>
</tr>
<tr>
<td>-----</td>
<td>---------------------------------</td>
<td>---------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>69</td>
<td>0.48808E-03</td>
<td>0.23472E-03</td>
<td>0.24213E-03</td>
</tr>
<tr>
<td>70</td>
<td>-0.98252E-03</td>
<td>0.44645E-04</td>
<td>0.49365E-03</td>
</tr>
<tr>
<td>71</td>
<td>-0.50568E-03</td>
<td>0.39484E-03</td>
<td>-0.10619E-03</td>
</tr>
<tr>
<td>72</td>
<td>-0.10713E-03</td>
<td>0.17492E-03</td>
<td>-0.15703E-04</td>
</tr>
<tr>
<td>73</td>
<td>0.22572E-03</td>
<td>0.25322E-03</td>
<td>0.41973E-03</td>
</tr>
<tr>
<td>74</td>
<td>-0.13153E-02</td>
<td>0.70679E-04</td>
<td>0.69135E-03</td>
</tr>
<tr>
<td>75</td>
<td>-0.16241E-03</td>
<td>0.36119E-03</td>
<td>-0.99350E-04</td>
</tr>
<tr>
<td>76</td>
<td>0.26582E-04</td>
<td>0.16210E-03</td>
<td>0.23870E-05</td>
</tr>
<tr>
<td>77</td>
<td>0.41013E-04</td>
<td>0.23748E-03</td>
<td>0.40745E-03</td>
</tr>
<tr>
<td>78</td>
<td>-0.10029E-02</td>
<td>0.88526E-04</td>
<td>0.66683E-03</td>
</tr>
<tr>
<td>79</td>
<td>0.11033E-03</td>
<td>0.28614E-03</td>
<td>-0.98681E-04</td>
</tr>
<tr>
<td>80</td>
<td>0.52491E-04</td>
<td>0.13383E-03</td>
<td>-0.21115E-05</td>
</tr>
<tr>
<td>81</td>
<td>-0.20222E-04</td>
<td>0.27571E-03</td>
<td>0.52619E-03</td>
</tr>
<tr>
<td>82</td>
<td>-0.91131E-03</td>
<td>0.13697E-03</td>
<td>0.83599E-03</td>
</tr>
<tr>
<td>83</td>
<td>0.24754E-03</td>
<td>0.29023E-03</td>
<td>-0.14464E-03</td>
</tr>
<tr>
<td>84</td>
<td>0.11413E-03</td>
<td>0.14228E-03</td>
<td>-0.38303E-04</td>
</tr>
<tr>
<td>85</td>
<td>-0.91055E-04</td>
<td>0.22985E-03</td>
<td>0.50872E-03</td>
</tr>
<tr>
<td>86</td>
<td>-0.72183E-03</td>
<td>0.15018E-03</td>
<td>0.79412E-03</td>
</tr>
<tr>
<td>87</td>
<td>0.16036E-03</td>
<td>0.22644E-03</td>
<td>-0.57512E-04</td>
</tr>
<tr>
<td>88</td>
<td>0.13066E-03</td>
<td>0.94236E-04</td>
<td>-0.78602E-04</td>
</tr>
<tr>
<td>89</td>
<td>-0.14240E-03</td>
<td>0.18906E-03</td>
<td>0.45621E-03</td>
</tr>
<tr>
<td>90</td>
<td>-0.55757E-03</td>
<td>0.15112E-03</td>
<td>0.68603E-03</td>
</tr>
<tr>
<td>91</td>
<td>0.68163E-04</td>
<td>0.19832E-03</td>
<td>0.58793E-04</td>
</tr>
<tr>
<td>92</td>
<td>0.87272E-04</td>
<td>0.75436E-04</td>
<td>-0.29804E-04</td>
</tr>
<tr>
<td>93</td>
<td>-0.26341E-03</td>
<td>0.15037E-03</td>
<td>0.48562E-03</td>
</tr>
<tr>
<td>94</td>
<td>-0.45171E-03</td>
<td>0.14972E-03</td>
<td>0.60448E-03</td>
</tr>
<tr>
<td>DV</td>
<td>$\frac{\partial C_L}{\partial DV}$</td>
<td>$\frac{\partial C_D}{\partial DV}$</td>
<td>$\frac{\partial C_{\text{Mx}}}{\partial DV}$</td>
</tr>
<tr>
<td>-----</td>
<td>-----------------------------------</td>
<td>-----------------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>95</td>
<td>0.35021E-04</td>
<td>0.18321E-03</td>
<td>0.87140E-04</td>
</tr>
<tr>
<td>96</td>
<td>0.41240E-04</td>
<td>0.80042E-04</td>
<td>0.15823E-04</td>
</tr>
<tr>
<td>97</td>
<td>-0.38186E-03</td>
<td>0.85864E-04</td>
<td>0.49232E-03</td>
</tr>
<tr>
<td>98</td>
<td>-0.41055E-03</td>
<td>0.13897E-03</td>
<td>0.55270E-03</td>
</tr>
<tr>
<td>99</td>
<td>-0.10178E-04</td>
<td>0.16466E-03</td>
<td>0.11292E-03</td>
</tr>
<tr>
<td>100</td>
<td>0.22434E-04</td>
<td>0.94406E-04</td>
<td>0.30456E-04</td>
</tr>
<tr>
<td>101</td>
<td>-0.26587E-03</td>
<td>0.96665E-04</td>
<td>0.36103E-03</td>
</tr>
<tr>
<td>102</td>
<td>-0.37939E-03</td>
<td>0.15884E-03</td>
<td>0.52837E-03</td>
</tr>
<tr>
<td>103</td>
<td>-0.85020E-04</td>
<td>0.19806E-03</td>
<td>0.19906E-03</td>
</tr>
<tr>
<td>104</td>
<td>0.11782E-04</td>
<td>0.12300E-03</td>
<td>0.46124E-04</td>
</tr>
<tr>
<td>105</td>
<td>-0.83950E-04</td>
<td>0.56975E-04</td>
<td>0.86178E-04</td>
</tr>
<tr>
<td>106</td>
<td>-0.37371E-04</td>
<td>0.71284E-04</td>
<td>0.61648E-04</td>
</tr>
<tr>
<td>107</td>
<td>-0.33410E-04</td>
<td>0.88268E-04</td>
<td>0.69698E-04</td>
</tr>
<tr>
<td>108</td>
<td>-0.30570E-05</td>
<td>0.56120E-04</td>
<td>0.22820E-04</td>
</tr>
</tbody>
</table>
APPENDIX B

FINAL VALUES OF THE DESIGN VARIABLES FOR DESIGN 1 AND DESIGN 2

Table B.1 shows the final values for the vector of design variables obtained for the first design optimization case of Chap. VI. There was a total of 108 variables for this case. Column 2 contains the results obtained when the traditional sequential approach was employed for the 1-D line search part of the optimization process. Column 3 contains the results using the new parallel 1-D line search procedure detailed in Chap. VI. The last column contains the comparison, in form of ratios, between the two approaches. All ratios are essentially unity, indicating good agreement between the two methods.

Table B.1 Final Values of Design Variables for Design 1 (108 Design Variables: Camber, Thickness and Twist)

<table>
<thead>
<tr>
<th>DV</th>
<th>Value_{sequential}</th>
<th>Value_{parallel}</th>
<th>Value_{parallel} / Value_{sequential}</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.12422E+00</td>
<td>0.12422E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>2</td>
<td>0.12175E+00</td>
<td>0.12175E+00</td>
<td>0.10000E+01</td>
</tr>
<tr>
<td>3</td>
<td>-0.40972E-01</td>
<td>-0.40962E-01</td>
<td>0.99976E+00</td>
</tr>
<tr>
<td>4</td>
<td>-0.40776E-01</td>
<td>-0.40778E-01</td>
<td>0.10001E+01</td>
</tr>
<tr>
<td>5</td>
<td>-0.83115E-01</td>
<td>-0.83114E-01</td>
<td>0.99999E+00</td>
</tr>
<tr>
<td>6</td>
<td>-0.12073E+00</td>
<td>-0.12073E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>7</td>
<td>-0.12422E+00</td>
<td>-0.12422E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>8</td>
<td>-0.11901E+00</td>
<td>-0.11900E+00</td>
<td>0.99995E+00</td>
</tr>
<tr>
<td>9</td>
<td>-0.11923E+00</td>
<td>-0.11923E+00</td>
<td>0.10000E+01</td>
</tr>
<tr>
<td>10</td>
<td>-0.12139E+00</td>
<td>-0.12139E+00</td>
<td>0.10000E+01</td>
</tr>
<tr>
<td>11</td>
<td>-0.12158E+00</td>
<td>-0.12158E+00</td>
<td>0.10000E+01</td>
</tr>
<tr>
<td>12</td>
<td>-0.12079E+00</td>
<td>-0.12079E+00</td>
<td>0.99998E+00</td>
</tr>
<tr>
<td>13</td>
<td>0.28155E-01</td>
<td>0.28154E-01</td>
<td>0.99997E+00</td>
</tr>
</tbody>
</table>

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Table B.1 Continued

<table>
<thead>
<tr>
<th>DV</th>
<th>(\text{Value}_{\text{sequential}})</th>
<th>(\text{Value}_{\text{parallel}})</th>
<th>(\text{Value}<em>{\text{parallel}} / \text{Value}</em>{\text{sequential}})</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>0.12422E+00</td>
<td>0.12422E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>15</td>
<td>0.12422E+00</td>
<td>0.12422E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>16</td>
<td>0.40804E-01</td>
<td>0.40806E-01</td>
<td>0.10001E+01</td>
</tr>
<tr>
<td>17</td>
<td>-0.71499E-02</td>
<td>-0.71495E-02</td>
<td>0.99995E+00</td>
</tr>
<tr>
<td>18</td>
<td>0.12422E+00</td>
<td>0.12422E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>19</td>
<td>0.12415E+00</td>
<td>0.12414E+00</td>
<td>0.99995E+00</td>
</tr>
<tr>
<td>20</td>
<td>0.79968E-01</td>
<td>0.79968E-01</td>
<td>0.10000E+01</td>
</tr>
<tr>
<td>21</td>
<td>-0.68135E-01</td>
<td>-0.68134E-01</td>
<td>0.99998E+00</td>
</tr>
<tr>
<td>22</td>
<td>0.12422E+00</td>
<td>0.12422E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>23</td>
<td>0.12422E+00</td>
<td>0.12422E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>24</td>
<td>-0.41937E-01</td>
<td>-0.41940E-01</td>
<td>0.10001E+01</td>
</tr>
<tr>
<td>25</td>
<td>-0.93704E-01</td>
<td>-0.93704E-01</td>
<td>0.99999E+00</td>
</tr>
<tr>
<td>26</td>
<td>0.12422E+00</td>
<td>0.12422E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>27</td>
<td>0.12422E+00</td>
<td>0.12422E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>28</td>
<td>-0.41461E-01</td>
<td>-0.41464E-01</td>
<td>0.10001E+01</td>
</tr>
<tr>
<td>29</td>
<td>-0.10537E+00</td>
<td>-0.10537E+00</td>
<td>0.99998E+00</td>
</tr>
<tr>
<td>30</td>
<td>0.64083E-01</td>
<td>0.64080E-01</td>
<td>0.99995E+00</td>
</tr>
<tr>
<td>31</td>
<td>0.12422E+00</td>
<td>0.12422E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>32</td>
<td>-0.42957E-01</td>
<td>-0.42959E-01</td>
<td>0.10001E+01</td>
</tr>
<tr>
<td>33</td>
<td>-0.12407E+00</td>
<td>-0.12407E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>34</td>
<td>-0.12384E+00</td>
<td>-0.12384E+00</td>
<td>0.10000E+01</td>
</tr>
<tr>
<td>35</td>
<td>0.12422E+00</td>
<td>0.12422E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>36</td>
<td>-0.40791E-01</td>
<td>-0.40794E-01</td>
<td>0.10001E+01</td>
</tr>
<tr>
<td>37</td>
<td>-0.12422E+00</td>
<td>-0.12422E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>38</td>
<td>-0.39065E-01</td>
<td>-0.39058E-01</td>
<td>0.99982E+00</td>
</tr>
<tr>
<td>39</td>
<td>0.12422E+00</td>
<td>0.12422E+00</td>
<td>0.99997E+00</td>
</tr>
</tbody>
</table>
Table B.1 Continued

<table>
<thead>
<tr>
<th>DV</th>
<th>Value\textsubscript{sequential}</th>
<th>Value\textsubscript{parallel}</th>
<th>Value\textsubscript{parallel} / Value\textsubscript{sequential}</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>-0.40776E-01</td>
<td>-0.40774E-01</td>
<td>0.99996E+00</td>
</tr>
<tr>
<td>41</td>
<td>-0.12401E+00</td>
<td>-0.12401E+00</td>
<td>0.10000E+01</td>
</tr>
<tr>
<td>42</td>
<td>-0.11539E+00</td>
<td>-0.11538E+00</td>
<td>0.99993E+00</td>
</tr>
<tr>
<td>43</td>
<td>0.12422E+00</td>
<td>0.12422E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>44</td>
<td>-0.46062E-01</td>
<td>-0.46065E-01</td>
<td>0.10001E+01</td>
</tr>
<tr>
<td>45</td>
<td>-0.12357E+00</td>
<td>-0.12357E+00</td>
<td>0.99999E+00</td>
</tr>
<tr>
<td>46</td>
<td>-0.39087E-01</td>
<td>-0.39069E-01</td>
<td>0.99953E+00</td>
</tr>
<tr>
<td>47</td>
<td>0.12337E+00</td>
<td>0.12337E+00</td>
<td>0.99998E+00</td>
</tr>
<tr>
<td>48</td>
<td>-0.40776E-01</td>
<td>-0.40778E-01</td>
<td>0.10001E+01</td>
</tr>
<tr>
<td>49</td>
<td>-0.12422E+00</td>
<td>-0.12422E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>50</td>
<td>0.42267E-01</td>
<td>0.42270E-01</td>
<td>0.10001E+01</td>
</tr>
<tr>
<td>51</td>
<td>0.12422E+00</td>
<td>0.12422E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>52</td>
<td>-0.41750E-01</td>
<td>-0.41753E-01</td>
<td>0.10001E+01</td>
</tr>
<tr>
<td>53</td>
<td>-0.12416E+00</td>
<td>-0.12415E+00</td>
<td>0.99993E+00</td>
</tr>
<tr>
<td>54</td>
<td>-0.72766E-01</td>
<td>-0.72764E-01</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>55</td>
<td>0.12422E+00</td>
<td>0.12422E+00</td>
<td>0.9997E+00</td>
</tr>
<tr>
<td>56</td>
<td>-0.41890E-01</td>
<td>-0.41893E-01</td>
<td>0.10001E+01</td>
</tr>
<tr>
<td>57</td>
<td>0.53108E-01</td>
<td>0.53105E-01</td>
<td>0.9994E+00</td>
</tr>
<tr>
<td>58</td>
<td>0.12246E+00</td>
<td>0.12246E+00</td>
<td>0.9998E+00</td>
</tr>
<tr>
<td>59</td>
<td>0.12422E+00</td>
<td>0.12422E+00</td>
<td>0.9997E+00</td>
</tr>
<tr>
<td>60</td>
<td>-0.42492E-01</td>
<td>-0.42495E-01</td>
<td>0.10001E+01</td>
</tr>
<tr>
<td>61</td>
<td>-0.12422E+00</td>
<td>-0.12422E+00</td>
<td>0.9997E+00</td>
</tr>
<tr>
<td>62</td>
<td>-0.12422E+00</td>
<td>-0.12422E+00</td>
<td>0.9997E+00</td>
</tr>
<tr>
<td>63</td>
<td>-0.12422E+00</td>
<td>-0.12422E+00</td>
<td>0.9997E+00</td>
</tr>
<tr>
<td>64</td>
<td>-0.12422E+00</td>
<td>-0.12422E+00</td>
<td>0.9997E+00</td>
</tr>
<tr>
<td>65</td>
<td>-0.12313E+00</td>
<td>-0.12313E+00</td>
<td>0.10000E+01</td>
</tr>
<tr>
<td>DV</td>
<td>Value\textsubscript{sequential}</td>
<td>Value\textsubscript{parallel}</td>
<td>Value\textsubscript{parallel} / Value\textsubscript{sequential}</td>
</tr>
<tr>
<td>-----</td>
<td>---------------------------------</td>
<td>-------------------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>66</td>
<td>-0.12422E+00</td>
<td>-0.12422E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>67</td>
<td>-0.12422E+00</td>
<td>-0.12422E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>68</td>
<td>-0.12422E+00</td>
<td>-0.12422E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>69</td>
<td>-0.12385E+00</td>
<td>-0.12385E+00</td>
<td>0.10000E+01</td>
</tr>
<tr>
<td>70</td>
<td>-0.12422E+00</td>
<td>-0.12422E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>71</td>
<td>-0.12422E+00</td>
<td>-0.12422E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>72</td>
<td>-0.11704E+00</td>
<td>-0.11703E+00</td>
<td>0.99994E+00</td>
</tr>
<tr>
<td>73</td>
<td>-0.12422E+00</td>
<td>-0.12422E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>74</td>
<td>-0.12422E+00</td>
<td>-0.12422E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>75</td>
<td>-0.12092E+00</td>
<td>-0.12092E+00</td>
<td>0.99998E+00</td>
</tr>
<tr>
<td>76</td>
<td>-0.97039E-01</td>
<td>-0.97024E-01</td>
<td>0.99985E+00</td>
</tr>
<tr>
<td>77</td>
<td>-0.12422E+00</td>
<td>-0.12422E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>78</td>
<td>-0.12413E+00</td>
<td>-0.12413E+00</td>
<td>0.99996E+00</td>
</tr>
<tr>
<td>79</td>
<td>-0.12050E+00</td>
<td>-0.12049E+00</td>
<td>0.99991E+00</td>
</tr>
<tr>
<td>80</td>
<td>-0.10969E+00</td>
<td>-0.10969E+00</td>
<td>0.10000E+01</td>
</tr>
<tr>
<td>81</td>
<td>-0.12422E+00</td>
<td>-0.12422E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>82</td>
<td>-0.11811E+00</td>
<td>-0.11811E+00</td>
<td>0.10000E+01</td>
</tr>
<tr>
<td>83</td>
<td>-0.12422E+00</td>
<td>-0.12422E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>84</td>
<td>-0.11830E+00</td>
<td>-0.11830E+00</td>
<td>0.99996E+00</td>
</tr>
<tr>
<td>85</td>
<td>-0.12422E+00</td>
<td>-0.12422E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>86</td>
<td>-0.11295E+00</td>
<td>-0.11295E+00</td>
<td>0.10000E+01</td>
</tr>
<tr>
<td>87</td>
<td>-0.12001E+00</td>
<td>-0.12000E+00</td>
<td>0.99995E+00</td>
</tr>
<tr>
<td>88</td>
<td>0.40888E-01</td>
<td>0.40891E-01</td>
<td>0.10001E+01</td>
</tr>
<tr>
<td>89</td>
<td>-0.12419E+00</td>
<td>-0.12418E+00</td>
<td>0.99994E+00</td>
</tr>
<tr>
<td>90</td>
<td>-0.12422E+00</td>
<td>-0.12422E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>91</td>
<td>-0.11822E+00</td>
<td>-0.11822E+00</td>
<td>0.99996E+00</td>
</tr>
</tbody>
</table>
Table B.1 Continued

<table>
<thead>
<tr>
<th>DV</th>
<th>$V_{\text{sequential}}$</th>
<th>$V_{\text{parallel}}$</th>
<th>$\frac{V_{\text{parallel}}}{V_{\text{sequential}}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>92</td>
<td>-0.39175E-01</td>
<td>-0.39043E-01</td>
<td>0.99664E+00</td>
</tr>
<tr>
<td>93</td>
<td>-0.71497E-01</td>
<td>-0.71484E-01</td>
<td>0.99982E+00</td>
</tr>
<tr>
<td>94</td>
<td>-0.12422E+00</td>
<td>-0.12422E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>95</td>
<td>-0.11955E+00</td>
<td>-0.11955E+00</td>
<td>0.99999E+00</td>
</tr>
<tr>
<td>96</td>
<td>-0.10949E+00</td>
<td>-0.10949E+00</td>
<td>0.10000E+01</td>
</tr>
<tr>
<td>97</td>
<td>0.79129E-01</td>
<td>0.79129E-01</td>
<td>0.99999E+00</td>
</tr>
<tr>
<td>98</td>
<td>-0.12422E+00</td>
<td>-0.12422E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>99</td>
<td>-0.12244E+00</td>
<td>-0.12244E+00</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>100</td>
<td>-0.11222E+00</td>
<td>-0.11221E+00</td>
<td>0.99994E+00</td>
</tr>
<tr>
<td>101</td>
<td>-0.49690E-01</td>
<td>-0.49689E-01</td>
<td>0.99999E+00</td>
</tr>
<tr>
<td>102</td>
<td>-0.49389E-01</td>
<td>-0.49388E-01</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>103</td>
<td>-0.49034E-01</td>
<td>-0.49033E-01</td>
<td>0.99999E+00</td>
</tr>
<tr>
<td>104</td>
<td>-0.41109E-01</td>
<td>-0.41108E-01</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>105</td>
<td>-0.45342E-01</td>
<td>-0.45341E-01</td>
<td>0.99998E+00</td>
</tr>
<tr>
<td>106</td>
<td>-0.43373E-01</td>
<td>-0.43371E-01</td>
<td>0.99996E+00</td>
</tr>
<tr>
<td>107</td>
<td>-0.43973E-01</td>
<td>-0.43971E-01</td>
<td>0.99997E+00</td>
</tr>
<tr>
<td>108</td>
<td>-0.33000E-01</td>
<td>-0.33000E-01</td>
<td>0.10000E+01</td>
</tr>
</tbody>
</table>
Table B.2 shows the final values for the vector of design variables obtained for the second design optimization case of Chap. VI. There was a total of 60 variables for this case.

<table>
<thead>
<tr>
<th>Design Variable</th>
<th>Final Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.8103700E-01</td>
</tr>
<tr>
<td>2</td>
<td>0.7722300E-01</td>
</tr>
<tr>
<td>3</td>
<td>0.7560200E-01</td>
</tr>
<tr>
<td>4</td>
<td>0.7569500E-01</td>
</tr>
<tr>
<td>5</td>
<td>0.7608400E-01</td>
</tr>
<tr>
<td>6</td>
<td>0.7317600E-01</td>
</tr>
<tr>
<td>7</td>
<td>0.4645000E-01</td>
</tr>
<tr>
<td>8</td>
<td>0.4413400E-01</td>
</tr>
<tr>
<td>9</td>
<td>0.5067800E-01</td>
</tr>
<tr>
<td>10</td>
<td>0.6040100E-01</td>
</tr>
<tr>
<td>11</td>
<td>0.6958500E-01</td>
</tr>
<tr>
<td>12</td>
<td>0.4154100E-01</td>
</tr>
<tr>
<td>13</td>
<td>-0.1540000E-01</td>
</tr>
<tr>
<td>14</td>
<td>0.8595600E-01</td>
</tr>
<tr>
<td>15</td>
<td>0.7654200E-01</td>
</tr>
<tr>
<td>16</td>
<td>-0.3549500E-01</td>
</tr>
<tr>
<td>17</td>
<td>-0.3136800E-01</td>
</tr>
<tr>
<td>18</td>
<td>0.8472000E-01</td>
</tr>
<tr>
<td>19</td>
<td>0.8588800E-01</td>
</tr>
<tr>
<td>20</td>
<td>0.1040700E-01</td>
</tr>
<tr>
<td>21</td>
<td>-0.2839500E-01</td>
</tr>
<tr>
<td>22</td>
<td>0.8303700E-01</td>
</tr>
<tr>
<td>23</td>
<td>0.3149400E-01</td>
</tr>
</tbody>
</table>

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
<table>
<thead>
<tr>
<th>Design Variable</th>
<th>Final Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>0.1590600E-01</td>
</tr>
<tr>
<td>25</td>
<td>-0.2173000E-01</td>
</tr>
<tr>
<td>26</td>
<td>0.8294600E-01</td>
</tr>
<tr>
<td>27</td>
<td>0.3373700E-01</td>
</tr>
<tr>
<td>28</td>
<td>0.3291700E-01</td>
</tr>
<tr>
<td>29</td>
<td>-0.3497200E-01</td>
</tr>
<tr>
<td>30</td>
<td>0.8255700E-01</td>
</tr>
<tr>
<td>31</td>
<td>0.8045500E-01</td>
</tr>
<tr>
<td>32</td>
<td>0.3497100E-01</td>
</tr>
<tr>
<td>33</td>
<td>-0.7099500E-01</td>
</tr>
<tr>
<td>34</td>
<td>0.8237900E-01</td>
</tr>
<tr>
<td>35</td>
<td>0.8685300E-01</td>
</tr>
<tr>
<td>36</td>
<td>0.5036600E-01</td>
</tr>
<tr>
<td>37</td>
<td>-0.8281500E-01</td>
</tr>
<tr>
<td>38</td>
<td>0.5914600E-01</td>
</tr>
<tr>
<td>39</td>
<td>0.8602200E-01</td>
</tr>
<tr>
<td>40</td>
<td>0.6905500E-01</td>
</tr>
<tr>
<td>41</td>
<td>-0.8303900E-01</td>
</tr>
<tr>
<td>42</td>
<td>-0.7051700E-01</td>
</tr>
<tr>
<td>43</td>
<td>0.5138400E-01</td>
</tr>
<tr>
<td>44</td>
<td>0.6494800E-01</td>
</tr>
<tr>
<td>45</td>
<td>-0.8165000E-01</td>
</tr>
<tr>
<td>46</td>
<td>-0.6905500E-01</td>
</tr>
<tr>
<td>47</td>
<td>0.3228300E-01</td>
</tr>
<tr>
<td>48</td>
<td>-0.1681500E-01</td>
</tr>
<tr>
<td>49</td>
<td>0.8040700E-01</td>
</tr>
<tr>
<td>50</td>
<td>0.2201400E-01</td>
</tr>
</tbody>
</table>
Table B.2 Continued

<table>
<thead>
<tr>
<th>Design Variable</th>
<th>Final Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>51</td>
<td>-0.4372700E-01</td>
</tr>
<tr>
<td>52</td>
<td>0.3328200E-01</td>
</tr>
<tr>
<td>53</td>
<td>0.8114600E-01</td>
</tr>
<tr>
<td>54</td>
<td>0.8620800E-01</td>
</tr>
<tr>
<td>55</td>
<td>0.3349800E-01</td>
</tr>
<tr>
<td>56</td>
<td>0.2620700E-01</td>
</tr>
<tr>
<td>57</td>
<td>0.8373200E-01</td>
</tr>
<tr>
<td>58</td>
<td>0.8483100E-01</td>
</tr>
<tr>
<td>59</td>
<td>0.4709200E-01</td>
</tr>
<tr>
<td>60</td>
<td>0.6695300E-01</td>
</tr>
</tbody>
</table>
VITA

Amidu Olawale Oloso was born in Ibadan, Nigeria on March 17, 1964 to the family of Alhaj Sirajudeen and Mrs. Bashirat Oloso. He attended the University of Ibadan where he graduated with Bachelor of Science (First Class Honors) in Agricultural Engineering in June of 1986. He underwent a one-year National Youth Service Corp program (compulsory for all fresh graduates in Nigeria) from 1986 to 1987 at a government farm in Sokoto, Nigeria. Mr. Oloso returned back to the same university, University of Ibadan, in 1987 for graduate studies. He graduated with Master of Science in Agricultural Engineering in August of 1988. Between June of 1989 and December of 1990, Mr. Oloso worked as a Lecturer II (Assistant Professor) in his department, Agricultural Engineering at the University of Ibadan where he taught several engineering courses and supervised final year projects. Mr. Oloso spent the whole of the year 1991 in residence at the Silsoe Campus of the Cranfield Institute of Technology, England, where he conducted research in engineering properties of Agricultural materials. Since January of 1992, Mr. Oloso has pursued a doctoral degree program in Mechanical Engineering at Old Dominion University in Norfolk, Virginia in the area of Aerodynamic Design Optimization and Parallel Computation.