The Pennsylvania State University

The Graduate School

Department of Aerospace Engineering

UNSTEADY SEPARATED FLOW SIMULATIONS USING A CLUSTER OF WORKSTATIONS

A Thesis in

Aerospace Engineering

by

Anirudh Modi

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 1999

We approve the thesis of Anirudh Modi.

Date of Signature

Lyle N. Long
Professor of Aerospace Engineering
Thesis Adviser

Philip J. Morris
Boeing/A.D. Welliver Professor of Aerospace Engineering

Dennis K. McLaughlin
Professor of Aerospace Engineering
Head of the Department of Aerospace Engineering

ABSTRACT

The possibility of predicting the full three-dimensional unsteady separated flow around complex ship and helicopter geometries is explored using unstructured grids with a parallel flow solver. The flow solver used is a modified version of the *Parallel Unstructured Maritime Aerodynamics* (PUMA) software, which was written by Dr. Christopher Bruner as part of his doctoral thesis at the Virginia Polytechnic Institute and State University. The efficiency and accuracy of PUMA at resolving several steady state solutions and a fully three-dimensional unsteady separated flow around a sphere were studied in order to determine if it was a suitable platform to base future work on. The *COst effective COmputing Array* (COCOA), a powerful 50-processor Beowulf cluster, was also built and tested as a part of the effort to make all this possible at a very economic cost. Unstructured grids were utilized in order to maximize the number of cells in the area of interest, while minimizing cells in the far field. A high level of clustering is required to solve viscous unsteady problems, and unstructured grids offer the least expensive method to ensure this. NASA's VGRID package was used to generate the unstructured grids.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I am thankful to my advisor **Dr. Lyle Long**, who amidst his busy schedule, spared time to guide me through the project. Without his support, motivation and useful suggestions, this project would not have taken its present shape. I also thank him for the great computing facilities extended and the freedom allowed to me throughout, which made working with him a pleasure. I am also grateful to Dr. Christopher Bruner for all his timely help with PUMA.

Thanks to Steven Schweitzer and Robert Hansen for their grid generation expertise and to NASA Langley for providing VGRID, the unstructured grid generation software. Thanks also to Obika Nwobi, whose lively presence in the office has been a constant source of inspiration and whose confidence in me was enthralling. I also owe my gratitude to Jingmei Liu, who was always there to help me whenever I was in doubt. I am also very grateful to Robert Hansen and Anurag Agarwal for the several enlightening discussions during the writing of this thesis.

I would like to thank the Rotorcraft Center of Excellence (RCOE) here at Penn State, for providing the financial support for this work. I would also like to thank the NSF for an MRI equipment grant (CTS 97-24639) to build COCOA.

Above all, I am highly indebted to my parents and family, and my fiancé Preeti, without whose constant love and support, all this would not have been possible.

Chapter 1

Motivation and Background

The prediction of unsteady separated, low Mach number flows over complex configurations (like ships and helicopter fuselages) is known to be a very difficult problem. The possibility of predicting these types of flows with the aid of inexpensive parallel computers is explored in this work. A parallel, finite volume flow solver was used and efforts were made to expedite the entire solution process.

The increasing use of helicopters in conjunction with ships poses major problems. In the presence of high winds and rough seas, excessive ship motions and turbulent separated flow from sharp-edged, box-like ship super-structures make landing a helicopter on ships a very hazardous operation. The strong unsteady flows can cause severe rotor blade deformations. Statistically, a helicopter can safely land on a frigate in the North Sea only 10 percent of the time in winter. Recent research on ship airwakes has been conducted from several different approaches[1]. One of the sources of relevant research is *building aerodynamics*, which shows the general features of flow around blunt bodies of different aspect ratios, and about clusters of buildings. The most likely model of a ship, but rather crude, is a sharp edged blunt body called the General Ship Shape (GSS) [2, 3]. More geometrically precise studies have been carried out in wind tunnels [4, 5, 6] and full scale tests have been conducted by the US Navy [7], which provide some important information on real ship airwakes. There have been other attempts at numerically simulating ship airwakes, e.g. use of a steady-state flow solver based on the 3D multi-zone, thin-layer Navier-Stokes (TLNS) method [8], and use of an unsteady, inviscid low-order method solver [9]. No method to-date has been entirely satisfactory for predicting these flow fields.

The prediction of separated flow around helicopters has also been an equally important issue. In 1990, Chaffin and Berry [10] utilized the well known CFL3D flow solver for their investigation into separated flow around helicopter fuselages. CFL3D predicted the flow sufficiently well for the study of viscous properties near the fuselage, but it did not capture the separated flow in the detail needed for a study of rotor-fuselage interactions. Also, the solution was computationally expensive as CFL3D is a serial code that utilizes structured grids. Duque et al [11] have used the OVERFLOW flow solver to analyze the flow around the United States Army's RAH-66 Comanche helicopter. The OVERFLOW flow solver uses the TLNS equations for its predictions. OVERFLOW uses the Parallel Virtual Machine (PVM) language to distribute the computational load in parallel. PVM is not the standard for parallel communication and is generally not well supported. The viscous solution over the isolated fuselage compared well with the experimental data, but the computational solution with a modeled main rotor did not compare favorably. This clearly demonstrated the need for better rotor and rotor wake simulations.

Another class of problems that are very similar to those above are the flow over spheres and cylinders. Spheres and cylinders are considered as prototype examples from the class of flows past axisymmetric bluff bodies. Over the decades, a lot of work has gone into the study of unsteady separated flow over spheres and cylinders at various Reynolds numbers. Since these are simple geometric shapes and are easily reproducible and thus tested, they have enjoyed a lot of importance in the study and validation of numerical flow solvers designed to deal with such complex flows. Extensive experimental data is readily available for several different flow conditions for both the flow over a cylinder [12, 13, 14, 15, 16, 17, 18] and over a sphere [19, 20, 21, 22, 23, 24, 25]. Zdravkovich [26] has devoted a set of books which attempt to bring out the importance of the study of flow over a cylinder, and summarize all the work that has gone into it over the century. Flow over a sphere is a classic problem which has intrigued experts over the years. With the advent of modern supercomputers, the study has seen renewed interest. Recently, Tomboulides [27] has carried on a complete *Direct Numerical Simulation* (DNS) and *Large Eddy Simulation* (LES) of flow over the sphere at various Reynolds numbers ranging from 50 to $20,000$. While flow over a cylinder can often be modeled as a 2D problem, that over a sphere cannot, which makes it much more complicated and computationally expensive.

Recently, Morris and Long [28] have proposed a new method, Non-Linear Disturbance Equations (NLDE), which is used to solve for the unsteady fluctuations in a flow field given the steady state solution as the input. This method is fourth-order accurate in time and space and promises very good results. NLDE is a very good candidate for solving such unsteady separated flow problems, given a reasonably accurate steady or near-steady state solution for the flow-field using any solver.

1.1 Flow Solvers

Numerical solutions of problems in fluid dynamics are usually formulated using one of two methods: the finite difference method (FDM) or the finite volume method (FVM). In the finite difference approach, a finite difference approximation of the differential equation is solved. The equation is first transformed from the physical domain to a uniform computational domain, and the differential form of the equation is solved at the node points. By contrast, the finite volume approach solves the integral form of the equation. In the finite volume formulation, the integral expressions for conservation of mass, momentum, and energy over an arbitrary control volume are solved directly rather than first being transformed to differential form. Since these expressions are valid for any control volume, considerable flexibility is permitted in the definition of the control volume's shape. Therefore, the main advantage of the finite volume approach over the finite difference approach, is that, it is more suitable for complex geometry, as the equations can be discretized directly in the "physical space." This was the main motivation behind using PUMA for this work as opposed to CFL3D. The detailed comparison between the two solvers will be discussed in Section 3.5.

The finite volume formulation is independent of the shape of the cell in the grid, and can work with unstructured grids consisting of polyhedral elements (usually tetrahedra). Structured grids can be thought of as a special case of unstructured grids consisting of only rectangles (2D) or bricks (3D). Unstructured finite volume methods are computationally more expensive per cell than the finite difference methods, but this difference is often compensated by the savings obtained in the number of cells required to model the geometry by the use of unstructured grids.

Another requirement for the solver is the maximum size of the problem it can handle, and the speed it can handle it with. Since serial programs are limited by both physical memory and speed of the serial computer, a parallel solver was necessary to be able to solve large problems like the flow over a helicopter fuselage or ship.

During 1993-96, Christopher W.S. Bruner [29, 30] worked towards an efficient and portable parallel finite volume flow solver called PUMA (Parallel Unstructured Maritime Aerodynamics). PUMA implemented several different time-integration algorithms for the Euler and Navier-Stokes equations using an explicit message-passing paradigm. Although PUMA was designed to be portable, it was written for high communications bandwidth supercomputers like the IBM SP, SGI Power Challenge and Intel Paragon, and its performance was virtually untested on the new breed of inexpensive but powerful computers like the Beowulf clusters, which have relatively higher latency. PUMA was extensively validated for several standard 2D and 3D shock problems like the RAE2822 airfoil, the M6 ONERA wing, and the supersonic ramp problem, for which extensive experimental data were available. While PUMA was designed to also run time-accurate, little testing was done to this effect. Most of the cases tested were either 2D or not very complex 3D shapes (e.g., M6 ONERA wing), only under transonic or supersonic steady regime.

1.2 Hardware and Software Issues

In order to speed up the investigations, a standard method was initiated in software to bring complex geometries easily from paper to the final flow solution. ProEngineer was utilized as the design package to get geometries into a digital format. ProEngineer is a widely used Computer Aided Design (CAD) package that is simple to use and allows a user to create complex geometries from simple 2D drawings. Since the CAD package used is dependent on personal preference, the file transfer to a grid generator should be generic. The NASA Langley unstructured grid generator (VGRID) is designed to import files in the *Initial Graphics Exchange Specification* (IGES) which is the United States national standard for the exchange of data between dissimilar CAD systems. The procedure from the IGES file to the final post-processed solution is depicted by the flowchart in figure 1.1.

An initiative to build a cheap but powerful **Beowulf cluster**[1] was undertaken. The cluster, COCOA (COst effective COmputing Array), aimed at the study of complex fluid dynamics problems using computational techniques without depending on expensive external supercomputing resources. It was thoroughly tested to study its feasibility for the flow solver applications. Several benchmarks were conducted to measure its overall effectiveness.

Several major and minor modifications were incorporated into PUMA to achieve this objective. All the communication routines in PUMA were checked thoroughly to make sure that they worked well on high-latency clusters. Those that did not perform well were modified to achieve significant speed-ups. Several pre-processing and post-processing utilities were written, and a general real-time visualization system was implemented to facilitate the study of the flow characteristics without any delay, as and when the solution was being computed.

[1] A cluster of commodity personal computers running the `LINUX` operating system.

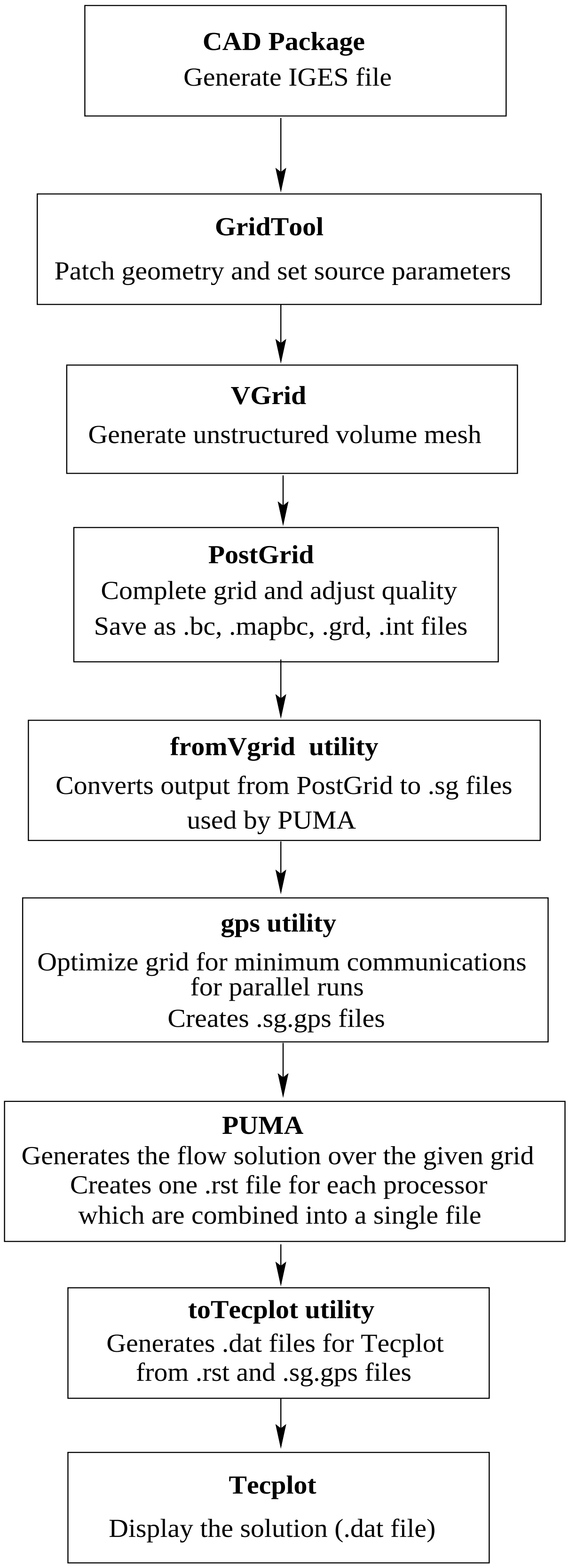

Figure 1.1. Solution Methodology

1.3 Thesis scope

The purpose of this work was to study the efficiency and accuracy of PUMA at resolving several steady state solutions and fully three-dimensional unsteady separated flows around complex geometries, in order to determine if it was a suitable platform to base future work on. Results obtained from PUMA for the GSS geometry and sphere were compared to available experimental data in order to achieve this.

A central differencing operator was also introduced into PUMA to facilitate the implementation of k-exact method and Non-Linear Disturbance Equations (NLDE) in the future. The possibility of using PUMA as a steady state solver acting as a pre-requisite to the NLDE [28] method to be used in order to predict the unsteady separated flow associated with helicopter fuselages was explored. This basic work is the beginning of a project to apply NLDE to a helicopter fuselage with a full rotor system. This would allow an accurate prediction of the flow field including all fuselage-rotor interactions.

A reasonably large Beowulf cluster was also built and its cost-effectiveness and performance were studied for our CFD related applications.

Chapter 2

Grid Generation

Before any numerical solution can be computed, the physical domain must be filled with a computational grid. The grid must be constructed in a way to accurately preserve the geometry of interest while providing the proper resolution for the algorithm to be applied. The two major categories of grid construction are structured grids and unstructured grids. Each type of grid has its own particular advantages and disadvantages. Structured grids are easier to handle computationally because their connectivity information is stored block to block. Structured grids are however more difficult to construct and tend to waste memory with unnecessary cells in the far field. Unstructured grids are more difficult to handle computationally because their connectivity is stored for each node. Unstructured grids, however, tend to be easier to construct and do not waste memory in far field cell resolution. Unstructured solvers often result in simpler computer codes too, which means they are easier to maintain and modify.

Figure 2.1 shows an NACA 0012 airfoil with an unstructured grid. This grid was created to capture vortices at 5 degrees angle of attack. This is shown by the concentration of cells in the known vortex area. This clearly demonstrates how unstructured grids can concentrate cells in areas of interest, thus allowing for large computational problems to be examined. Given the complex nature of helicopter flow problems, the unstructured approach offers the best opportunity to quickly, accurately and effectively obtain flow solutions. Figures 2.2 and 2.3 show structured and unstructured surface grids for the General Ship Shape (GSS) geometry, respectively. As can be seen clearly, the unstructured grid has been utilized to concentrate the cells around the deck region (region of interest) without having too many cells in the other regions. (Note: The number of cells for the unstructured grid shown here is much larger than for the structured grid, simply because the domain for the unstructured grid is also much larger.)

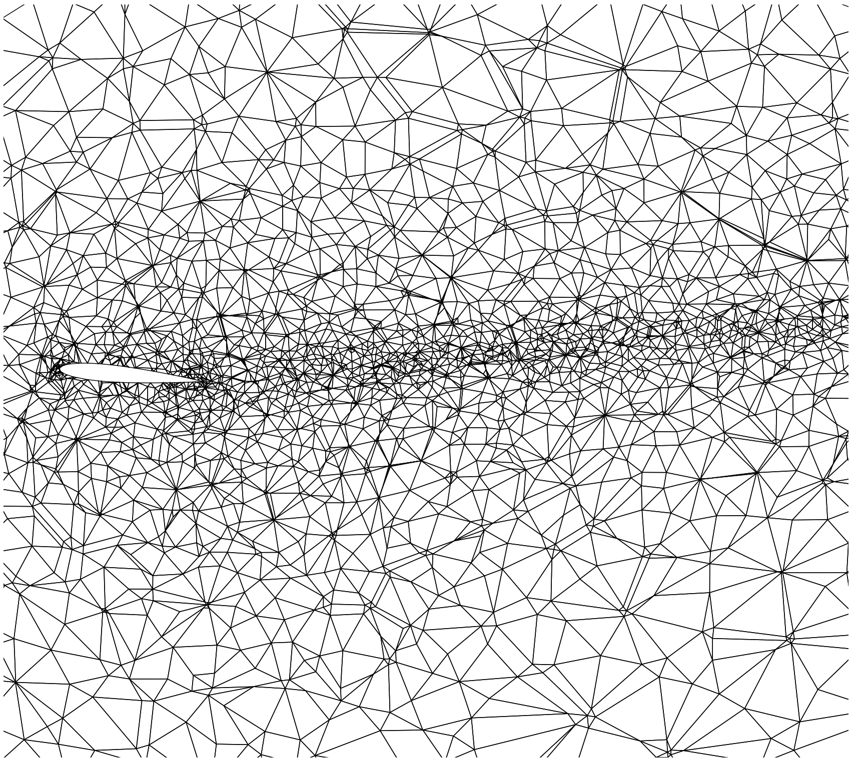

Figure 2.1. Sectional view of unstructured grid for a NACA 0012 wing

2.1 VGRID: Unstructured Grid Generator

The unstructured grids created around the geometries studied in this research, were generated using a grid generator based on the advancing front method (AFM), namely VGRID. VGRID [31] was developed by *ViGYAN, Inc.* in association with the NASA Langley Research Center, as a method of quickly and easily generating grids around complex objects. VGRID is a fully functional, user-oriented unstructured grid generator. GridTool is the program that acts as a bridge between CAD packages and grid generation in VGRID. The typical process starts with generating a drawing for the geometry of interest in a CAD package (e.g., ProEngineer, AutoCAD, IDEAS). This geometry is then exported from the CAD package to an *Initial Graphics Exchange Specification* (IGES) format. GridTool can then prepare the geometry for grid generation by providing VGRID with a complete and accurate definition of the geometry. This is accomplished by specifying curves along the geometry and then turning these curves into unique surface patches that define the geometry. Source terms are then added to the computational domain, which provide VGRID with the starting information for the grids in the AFM. The source terms can be freely placed anywhere in the

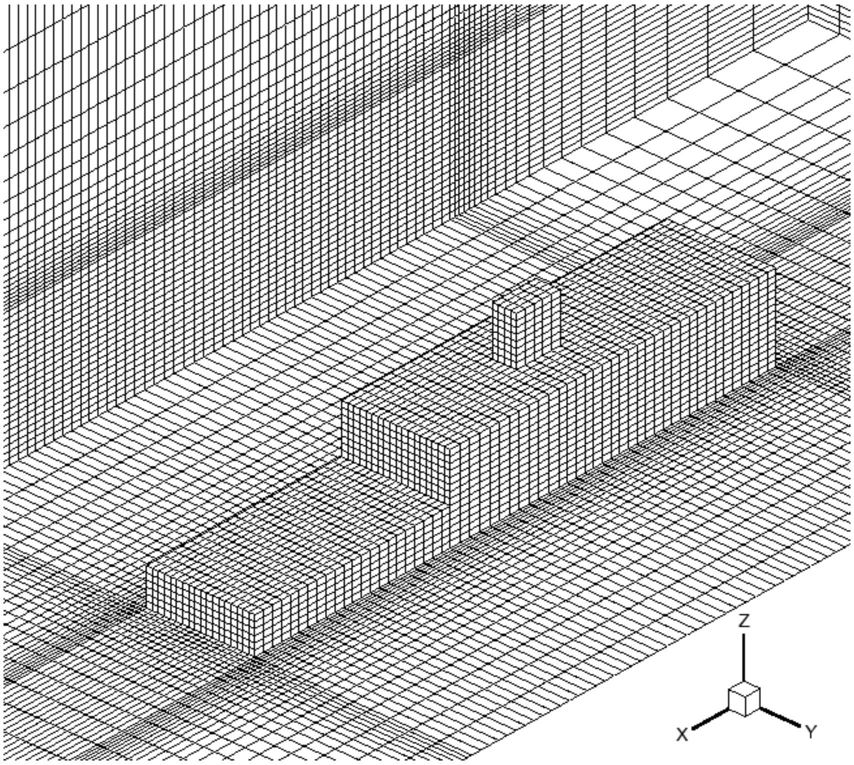

Figure 2.2. Structured surface grid for GSS geometry (694556 faces, 227120 cells)

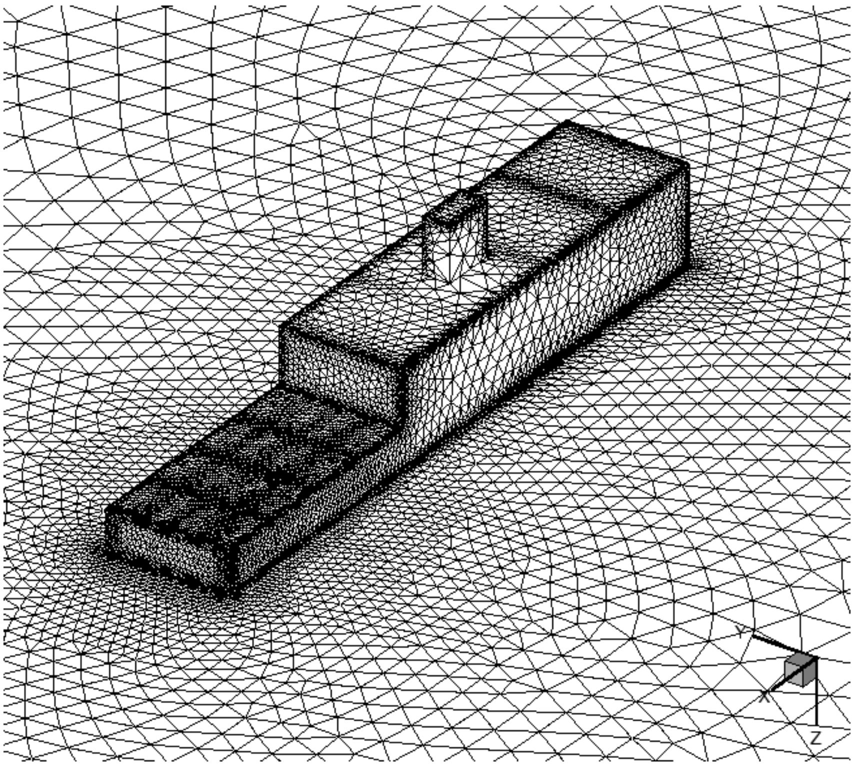

Figure 2.3. Unstructured surface grid for GSS geometry (984024 faces, 483565 cells)

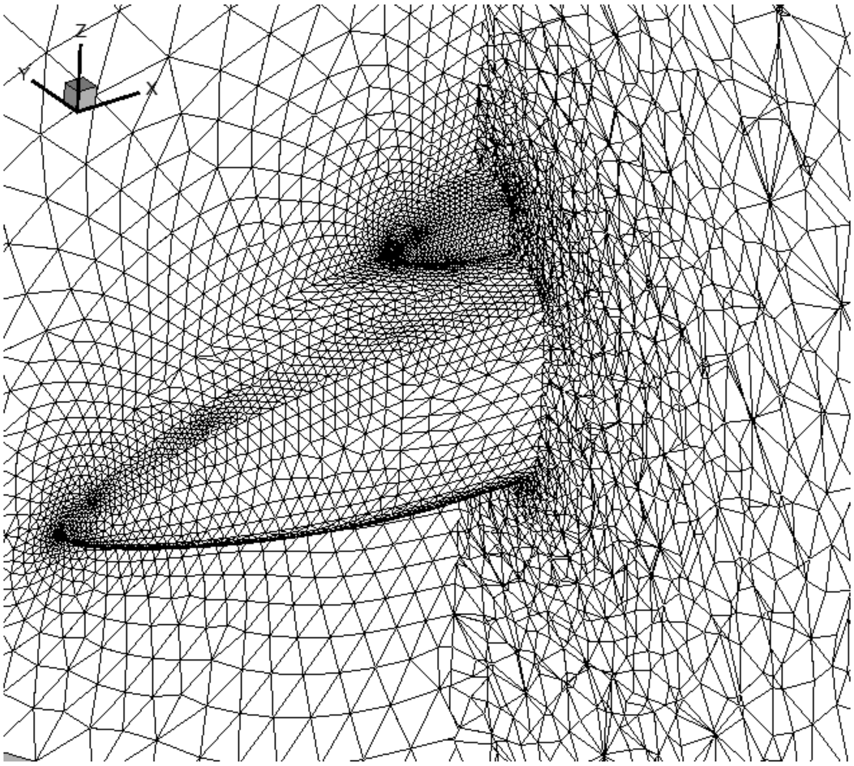

Figure 2.4. Volume grid generation over ROBIN helicopter geometry (532492 faces, 260858 cells)

domain and act as a control mechanism for clustering. Several examples of grids generated using GridTool and VGRID can be seen in figures 2.3, 2.4, 2.5, 2.6 and 2.7. This Apache helicopter geometry (figure 2.6) was obtained from H.E. Jones of the US Army.

VGRID takes the output generated from GridTool and uses it as the basis for the unstructured advancing front grid growth. VGRID's first step is to create a surface mesh based on the provided surface patches and source terms. The source terms specified in GridTool can be considered simple heat sources. The spatial variation in the grid is then determined by computing the heat diffusion from the discrete heat sources. The heat equation is solved using a Gauss-Seidel iterative scheme with an applied successive over-relaxation. The advancing front grows from the smallest source terms outward. The heat diffusion is calculated, and then one layer of the grid front is grown. This iterative process is continued until the entire domain is filled.

VGRID is capable of generating an automatic viscous layer for Navier-Stokes algorithmic solutions. This option is turned on with a simple flag in GridTool. When the viscous flag is active, VGRID creates small layers of prisms all along the surface of the given geometry. These prisms completely and uniformly cover all surfaces making this an extremely easy

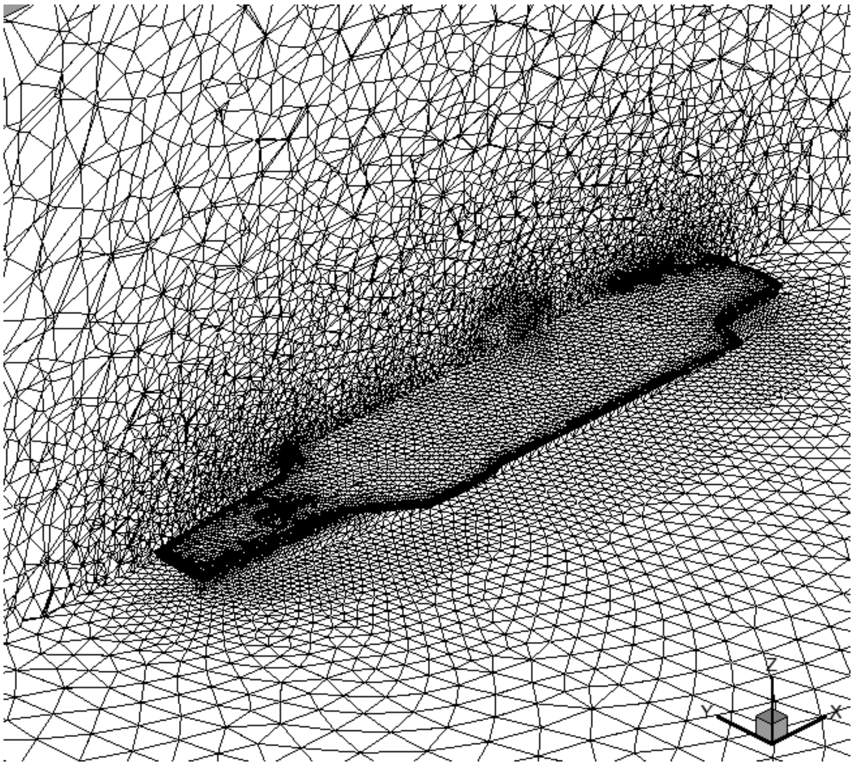

Figure 2.5. Volume grid generation over aircraft carrier CVN-75 (974150 faces, 478506 cells)

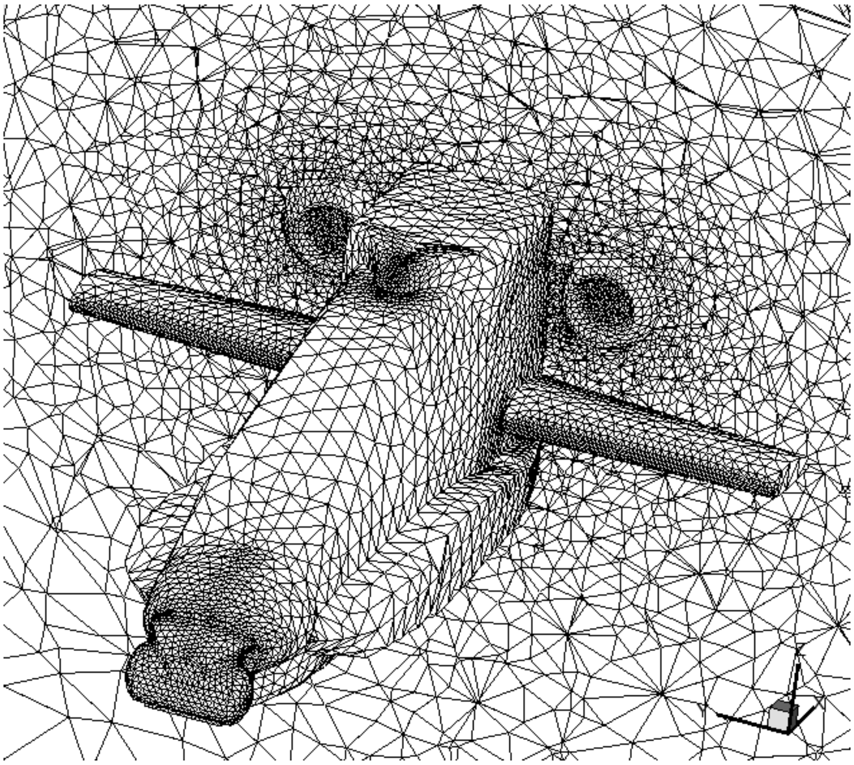

Figure 2.6. Volume grid generation over Apache helicopter geometry (1125596 faces, 555772 cells)

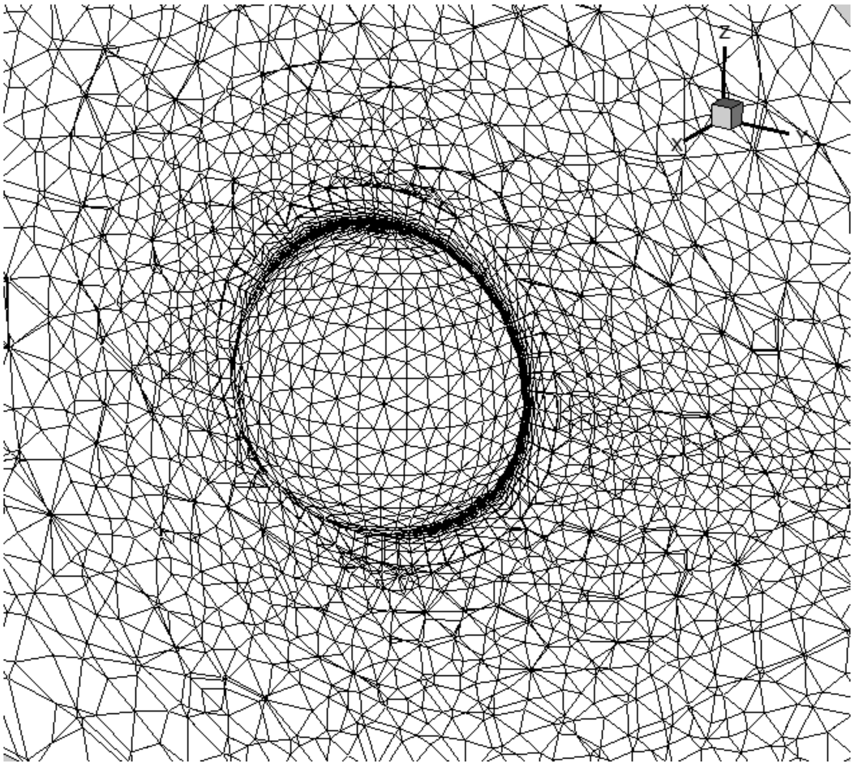

Figure 2.7. Viscous volume grid generation over a sphere (617665 faces, 306596 cells)

way to generate viscous grids. VGRID also automatically determines the number of viscous prism layers from the complexity of the shape. More information on using VGRID can be found in Master's thesis of Steven Schweitzer [32].

Chapter 3

PUMA

3.1 Introduction

Parallel **U**nstructured **M**aritime **A**erodynamics (**PUMA**), is a computer program for the analysis of internal and external non-reacting compressible flows over arbitrarily complex 3D geometries. It was written by Dr. Christopher W.S. Bruner as a part of his doctoral thesis [29]. It is written entirely in ANSI C using MPI (Message Passing Interface) libraries for message passing, and hence is highly portable giving good performance. It is based on the Finite Volume Method (FVM) that solves the full three-dimensional Reynolds-Averaged Navier-Stokes (RANS) equations, and supports mixed topology unstructured grids composed of tetrahedra, wedges, pyramids and hexahedra (bricks). PUMA may be run so as to preserve time accuracy or as a pseudo-unsteady formulation to enhance convergence to steady-state. It uses dynamic memory allocation, thus problem size is limited only by the amount of memory available on the machine. It needs approximately 582 bytes/cell and 634 bytes/face using double precision variables (not including message passing overhead). The development of PUMA was highly motivated by the lack of work that compares the performance of several different algorithms on the same problem. Hence PUMA implements a range of time-integration schemes like *Runge-Kutta*, *Jacobi* and various *Successive Over-relaxation Schemes (SOR)*, as well as both *Roe* and *Van Leer* numerical flux schemes. It also implements various monotone limiters used in second-order computations (*Venkatakrishnan, Barth, Van Albada, Superbee*). PUMA requires anywhere between $25000 - 30000$ floating point operations/iter/cell, depending very much on the nature of the grid and cell connectivity, for a second order accurate computation running in double precision using SSOR time-integration and Roe fluxes. Several modifications and extensions have been incorporated into PUMA during the course of this work, and several others are planned for the near future.

3.2 Domain Decomposition

Parallel computing paradigms also fall into two broad classes: *functional decomposition* and *domain decomposition*. These classes reflect how a problem is allocated to processes. Functional decomposition divides a problem into several distinct tasks that may be executed in parallel; one field where this is popular today is that of Multidisciplinary Design Optimization. On the other hand, domain decomposition distributes data across processes, with each process performing more or less the same operations on the data. Domain decomposition is to be distinguished from the Single Instruction Multiple Data (SIMD) model in Flynn's taxonomy of computer architectures [33]: in the SIMD model, each process executes the same instructions at the machine level, i.e., all processes proceed in lockstep, doing exactly the same operations on different data. This kind of parallelism (sometimes called fine-grain parallelism) is usually found at the loop level, whereas domain decomposition is applied at the problem level. PUMA uses the *Single Program Multiple Data* (SPMD) parallelism, i.e., the same code is replicated to each process. One process acts as the master and collects the results from all other processes, thus doing slightly more work than the other processes.

Domain decomposition is generally done with the aim to minimize communications cost (i.e. time), by distributing the cells in the domain in some optimal way.

$$\text{communication time} = \text{latency} + \frac{\text{message size}}{\text{communications bandwidth}}$$

Following are the two commonly used approaches for domain decomposition:

1. **Minimization of inter-domain boundaries**: This approach is ideal for slow networks (lower communications bandwidth) or whenever many small messages are passed. It neglects the latency in the above expression. In this approach, the total length of inter-domain boundaries is minimized (which is length of all shared data). In other words, it minimizes the shared part of perimeter/surface area of the domain (Figure 3.2). It requires a-priori knowledge of the number of domains (compute nodes) and hence has to be done every time the problem is run on a different number of nodes. Popular techniques to implement this approach need the eigenvalues of the connectivity matrix, and thus can be expensive for large problems. However, this approach produces the best grids for large problems.

2. **Minimization of communications**: This approach is ideal for fast networks or

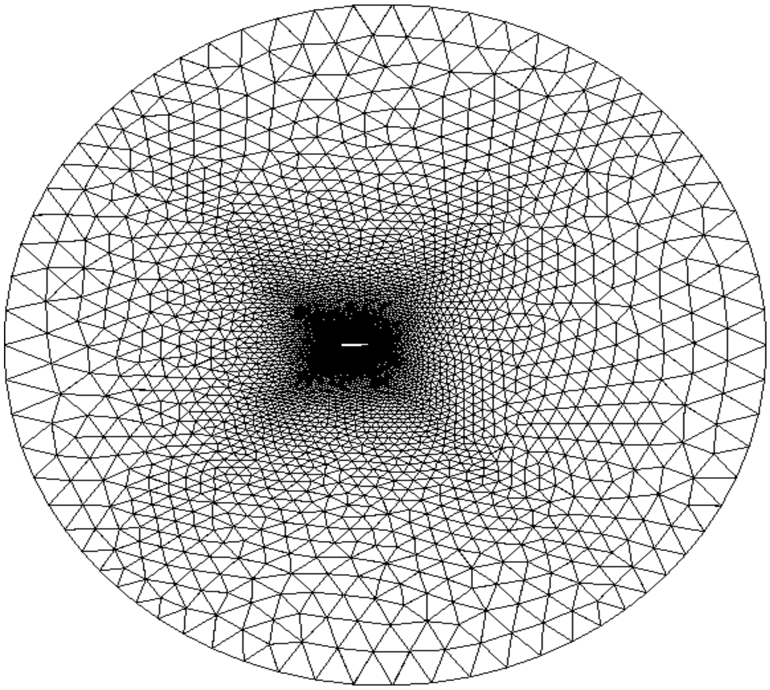

Figure 3.1. Unstructured grid for RAE2822 airfoil

whenever few big messages are passed. It neglects the second term in the above expression as communication bandwidth is assumed large. This approach minimizes the total number of messages exchanged (Figure 3.3). Since the above minimization corresponds to minimization of bandwidth of the cell connectivity matrix, this approach is independent of the number of domains (compute nodes). A large knowledge base exists in this area and is readily accessible so implementation is sound and easy.

These domain decomposition techniques are absolutely essential since we are dealing with unstructured grids in which the ordering of grid cells is generally arbitrary (especially, if say, Advancing Front Method was used to create the 2D/3D grids or Delaunay Triangulation for the 2D grids). Either approach can be used with PUMA, as domain decomposition is done by a separate program at the pre-processing stage. However, the latter approach is preferred as it is aimed at machines with high communications bandwidth, and also since it is independent of the number of processors. Currently, the Gibbs-Poole-Stockmeyer (GPS) [34] algorithm is used to achieve this. For the first approach, a publicly available graph partitioning software package called METIS [35], which is being actively developed by the Department of Computer Science at University of Minnesota, can be used. A comparison of the partitioning generated by the two approaches can be seen by applying it to the RAE2822 grid (figure 3.1) for an

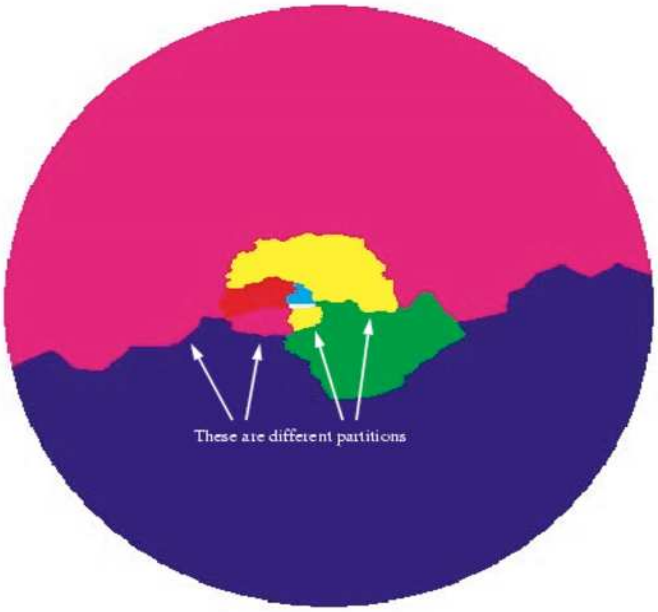

Figure 3.2. Typical 8-way partitioning of the RAE2822 grid (figure 3.1) using a graph partitioning algorithm. The colors correspond to different computational domains (or partitions). (Courtesy Bruner, C.W.S. [29])

Figure 3.3. 8-way partitioning of the RAE2822 grid (figure 3.1) from the Gibbs-Poole-Stockmeyer reordering (Courtesy Bruner, C.W.S. [29])

8-processor case to give figures 3.2 and 3.3, respectively.

3.3 Parallelization in PUMA

PUMA implements explicit and implicit time integration schemes on distributed memory computers [29]. The solution procedure for the parallel implementation can be divided into six major steps:

1. Each compute node reads its own portion of the grid file at startup.
2. Cells are divided among the active compute nodes at runtime based on cell IDs, and only faces associated with local cells are read.
3. Faces on the interface surface between adjacent computational domains are duplicated in both domains. Fluxes through these faces are computed in both domains.
4. Solution variables are communicated between domains at every timestep which ensures that the computed solution is independent of the number of compute nodes.
5. Communication of the solution across domains is all that is required for first-order spatial accuracy, since Q_L and Q_R are simply cell averages to the first order (Q being the conservative set of flow variables $\{\rho, \rho u, \rho v, \rho w, \rho E\}$).
6. If the left and right states are computed to higher-order, then Q_L and Q_R are shared explicitly with all adjacent domains. The fluxes through each face are then computed in each domain to obtain the residual for each local cell.

3.4 Boundary Condition Implementation

Implicit boundary conditions are used in PUMA when any of the implicit time-integration schemes are used (i.e. the boundary condition is evaluated at the next timestep as opposed to the current timestep). The boundary condition Jacobian ($\frac{\partial Q_{BC}}{\partial Q}$) is obtained numerically using central differences, which is completely general and does much to improve the maintainability of the code. There are several types of BCs implemented in PUMA, and more can be added without much difficulty as and when needed. The existing list of boundary conditions is given in table 3.1 [36]. A sample input file for PUMA is present in Appendix B.

Type	Description	Variable values specified
0	Do nothing	
1	Fixed at freestream	
2	Fixed at given values	ρ,u,v,w,p
3	First order extrapolation from the interior	
4	Second order extrapolation from the interior	
5	Inflow/outflow (Riemann)	
6	Specified p_0, T_0	p_0, T_0
7	Fixed back pressure pressure	p
8	Tangency	
9	No slip adiabatic wall	
10	No slip fixedtemperature wall T_{wall}	T_{wall}
11	Zero u velocity	
12	Zero v velocity	
13	Zero w velocity	

Table 3.1. PUMA boundary conditions

3.5 CFL3D vs PUMA

To begin an earnest effort to accurately model the unsteady separated flow around complex 3D geometries, a flow solver had to be selected. Chaffin and Berry [10] utilized the extremely popular CFL3D flow solver for their investigation into separated flow around helicopter fuselages. Currently, the primary developers of CFL3D are Dr. Christopher L. Rumsey and Dr. Robert T. Biedron of the Aerodynamics and Acoustics Branch at NASA Langley. CFL3D predicted the fuselage flow sufficiently well for the study of viscous properties near the fuselage. CFL3D however did not capture the separated flow in the detail needed for a study of rotor fuselage interactions. PUMA was untested at solving such large problems around complex shapes, but PUMA offers many key advantages over CFL3D and other flow solvers. Table 3.2 compares the features of CFL3D [37] and PUMA [29]. CFL3D is a serial code that utilizes structured grids while PUMA is a parallel code that uses unstructured grids. The parallelization of PUMA, and the fact that it uses unstructured grids, are the two most important advantages that PUMA has over CFL3D. A parallel code offers many advantages over a serial code. In a parallel code, the domain is decomposed and distributed across several processors which allows it to handle much larger problems than its serial counterpart. In a large computational domain, the grid points are distributed amongst the various processors allowing meshes with several million grid points. Since the computational load is also distributed, parallel codes also tend to run much faster than the serial codes. This, of course, is only true if communication times between the processors are kept to a minimum. Currently programmers at NASA Langley are working on parallelizing CFL3D, but at this time it is not available. PUMA uses an unstructured mesh while CFL3D utilizes a structured mesh. Modern unstructured mesh generators like VGRID, which have already been described earlier, are extremely easy to use and nearly automatic. The use of unstructured meshes, therefore, greatly decreases the amount of time spent generating appropriate meshes. Unstructured meshes unlike structured meshes also eliminate needless grid points in the far field. By easily concentrating cells around appropriate areas and not elsewhere, extremely complex shapes can be meshed quickly, easily and with the fewest possible nodes.

Both PUMA and CFL3D are 2^{nd} order accurate although neither of them currently implement a k-exact method to improve their order of accuracy (more will be discussed on this in Chapter 8). PUMA offers a wide variety of time integration schemes, both explicit and implicit, while CFL3D only uses *Alternating Direction Implicit* (ADI) schemes. This makes PUMA much more versatile, since ADI schemes are slow to converge on certain problems and are not well suited for unsteady problems. PUMA was originally written to conduct a comparison of various time integration schemes on parallel computers [29]. Dr. Christopher Bruner concluded that the best performing algorithm in PUMA is dependent upon the communication network utilized by the computer system. The presence of several commonly used algorithms in the PUMA code allows individual researchers to determine the algorithm that is most appropriate for their particular needs. In this way, PUMA, unlike CFL3D offers a large amount of flexibility for different problems and circumstances.

On the downside, PUMA does not have the robust turbulence modeling that is present in CFL3D. However, the lack of good turbulence modeling is not a big issue, since it can be added to PUMA without much difficulty owing to the relatively small size of the PUMA code. PUMA is around $10,000$ lines of code while CFL3D is $100,000$ lines of code. This order of magnitude difference makes it much easier for anyone to learn, understand and modify PUMA. Also of importance is the fact that PUMA is written in C, while CFL3D is written in Fortran 77 (F77). With the F90 standard implemented and F95 on the way, it will not be long before CFL3D will need to be drastically updated to work with modern compilers. While F77 remains the standard due to a large portion of legacy flow solver, the momentum is gradually shifting toward lightweight, highly portable C codes. This brief comparison of PUMA and CFL3D makes it clear that PUMA offers a much better starting point for capturing unsteady flows. There are many other flow solvers that were not considered due to their lack of portability, limited usefulness in the area of separated flow, or their extreme complication. PUMA's simplistic, parallel, unstructured approach is ideal for quick modifications for almost any external flow solution need.

	CFL3D	**PUMA**
Method	Finite Difference	Finite Volume
Parallel	N	Y
Multigrid	Y	N
Unstructured Grids	N	Y
Structured Block Grids	Y	N
Order of Accuracy	2^{nd}	2^{nd}
Time Integration Schemes		
Runge-Kutta	N	Y
Gauss Seidel	N	Y
Jacobi	N	Y
RSOR	N	Y
FSOR	N	Y
SSOR	N	Y
ADI	Y	N
Turbulence Models		
Baldwin-Lomax	Y	N
Baldwin-Barth	Y	N
Spalart-Allmaras	Y	Y
Wilcox $k-\omega$	Y	Y
Menter $k-\omega$	Y	N
Speziale-Gatski $k-\omega$	Y	N
Speziale-Gatski $k-\epsilon$	Y	N
Abid $k-\epsilon$	Y	N
Girimaji $k-\epsilon$	Y	N
Upwind	Y	Y
Compressible	Y	Y
Lines of Code	100,000	10,000
Programming Language	Fortran 77	C and MPI
Portable	Y	Y

Table 3.2. CFL3D vs PUMA (Courtesy Steven Schweitzer [32])

3.6 Modifications to PUMA

During the course of the project, several modifications and improvements were made to PUMA. Some of these were major whereas some were not, but all of them contributed towards making PUMA easier to understand and work with. Some of the more important modifications are highlighted below:

1. The entire source code was parsed manually and was indented and beautified to make it more readable and understandable. Several redundancies in the code were removed by replacing them with sensible functions (`PUMA_MALLOC()`, `print_malloc()`, `free_mem()`, `myprintf()`, `fatal()`, `warning()` in the file `err.c`), which resulted in about 5% savings in terms of length of the code. All the warnings given by the compiler were removed which made the code more robust and easier to debug. In particular, several functions were added to manage the memory in the code more cleanly, and to display the output from each processor in a very organized manner. A memory counter was implemented to automatically keep track of the amount of memory allocated and deallocated by each variable on each processor, thus making the debugging of memory related problems very easy. Currently, significantly more debugging information is displayed by PUMA than it did previously.

2. A simple versioning system was added to PUMA to keep track of the various modifications made to the code. Since extensive development of PUMA by several people simultaneously, is proposed in the near future, this is a much needed feature.

3. The original version of PUMA required the nodes defining the faces to be given in a specific order only (routine `gridGeometry()` in file `gridgeom.c`). This often differed from the output generated by VGRID, which made the grid file unusable by PUMA. This also caused the volume of the cells to be calculated as negative, and the face normals in the opposite direction, which was unacceptable by PUMA. Several minor modifications were made to this routine to make the code lenient in handling the order of nodes, thus making it independent of the order in which the nodes were specified for a given face. This change essentially does not affect the validity check for the grid if all the faces in the grids are triangles (which is what is dominantly used). However, for grids containing faces with greater than 3 sides (i.e. quadrilaterals), the validity check is compromised and invalid grids may be accepted by PUMA, thus giving incorrect solutions.

4. There were several MPI related problems encountered while running PUMA that made it extremely slow on Beowulf clusters. Since PUMA was written with very high communications bandwidth, low-latency parallel computers like SGI Power Challenge, IBM

SP, Cray T3E and Intel Paragon in mind, its communication routines did not work well on high-latency Beowulf clusters. This was observed clearly in the initialization of the code, which often took anywhere between 30 minutes to several hours on 8 processor runs on one of our Beowulf clusters (using moderately sized grids consisting of about $250,000$ cells). On debugging the code thoroughly, this was found to be due to the fact that the master processor in PUMA was reading the grid file one line at a time, and broadcasting contents of the line to every processor separately (subroutine `getGrid()` in file `io.c`). This was causing considerable latency overhead which slowed down the entire process. After modifying the code to read several hundred lines at a time and broadcasting the combined data to every processor using a reasonably sized buffer, a speedup of almost 10 was achieved. Currently, initialization of typical cases does not exceed 5 to 10 minutes with maximum time observed to be 25 minutes for a grid containing about 1.5 million cells.

5. Another serious communication related problem was observed with PUMA while running it on COCOA with option for second-order spatial accuracy turned on. In this case, each iteration for any large problem (with cells typically greater than $100,000$), took several times longer than the corresponding run on the PSU IBM-SP2. For larger problems containing more that $500,000$ cells, the 16 processor run slowed down dramatically, being even slower than the 2 processor runs on the same cluster. The cause for this behavior was traced to a set of `MPI_Isend/MPI_Irecv` calls, which were used to extrapolate the state variables from the boundaries of the neighboring processors to achieve second-order spatial accuracy (routine `varExtrap()` in `fluxes.c`). In this case too, thousands of very small messages ($<$ 100 bytes) were being exchanged between the processors continuously, which caused the communication time to increase tremendously due to the high-latency of the network. This can also be clearly explained after looking at the figures 4.14 and 4.15 in chapter 4, which show that network performance is extremely bad when packet sizes are less than 1000 bytes. However, the solution here was not as trivial as for the above case, since the algorithm and the data structure in use did not allow the data to be packed into one large message easily. After some modification to the routine, a message buffer was implemented which combined several such small messages into one before starting the communication. The results were amazing, and a speed-up of anywhere from 10 to 50 was observed depending on the problem size. A comparison of the performance before and after implementation

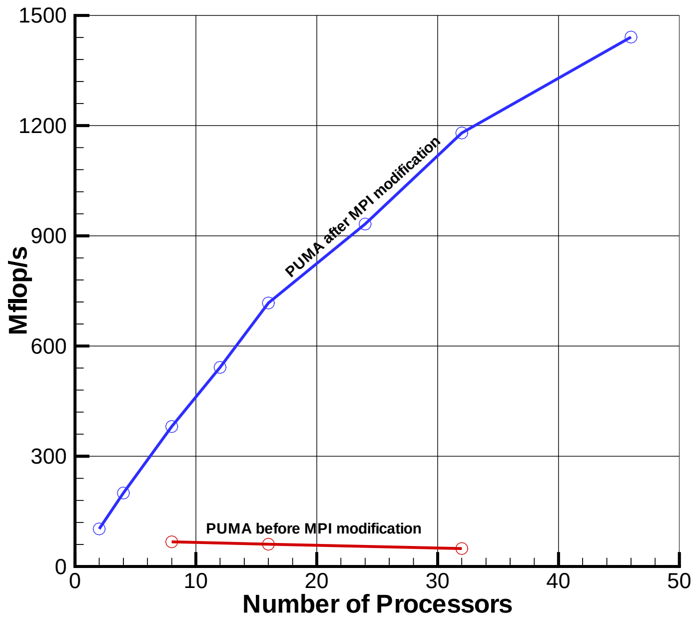

Figure 3.4. Improvement in PUMA performance after combining several small MPI messages into one (for unstructured GSS case with $483,565$ cells described in Section 6.1.1)

of this buffer can be seen in figure 3.4.

6. Several minor modifications were made to PUMA in order to make the implementation of the "Live CFD-cam" (section 5.2) easier. This primarily consisted of modifying PUMA to generate an output file (named `restart.txt`) containing the name of the last completely written restart file.

7. Added artificial dissipation terms to Runge-Kutta time-integration scheme for the implementation of the central differencing flux operator. More details are provided on this in the next section.

3.7 Artificial Dissipation terms for Central Differencing Operator

In principle, the physical viscous terms of the Navier-Stokes equations are capable of providing the numerical scheme with the dissipative property necessary for stability and capturing of discontinuities. However, for high Reynolds number flows, this can only be achieved by resorting to extremely small mesh spacings throughout the domain. Thus, in practice, it is necessary to introduce artificial dissipative terms to maintain stability in the essentially inviscid portions of the flow field, and to efficiently capture the discontinuities. These additional dissipative terms must be carefully constructed to ensure that the accuracy of the scheme is preserved both in the inviscid region of the flow field where the convective terms dominate, as well as in the boundary-layer and wake regions where the artificial dissipation terms must be much smaller than the physical viscous terms [38, 39]. The original PUMA code used upwind methods that have implicit dissipation, and did not require any additional explicit dissipation.

The artificial dissipation operator on unstretched unstructured grids has previously been constructed as a blend of an undivided Laplacian and biharmonic operator in the flow field. The formulation here has been extended for 3D unstructured grids from the 2D unstructured formulation proposed by Mavripilis and Jameson [39]. Since the biharmonic operator may be viewed as a Laplacian of a Laplacian, the dissipation operator may be reformulated as a global undivided Laplacian operating on a blend of the flow variables and their second differences:

$$D(q_i) = \alpha_i \nabla^2 u_i = \alpha_i \sum_{k=1}^{n} [u_k - u_i] \tag{3.1}$$

where

$$u_i = (k_2^{'})_i q_i - k_4 \nabla^2 q_i = (k_2^{'})_i q_i - k_4 \sum_{k=1}^{n} [q_k - q_i] \tag{3.2}$$

and

$$q = \begin{bmatrix} \rho \\ u \\ v \\ w \\ p \end{bmatrix} \tag{3.3}$$

In the above equations, $\nabla^2 q$ denotes the undivided Laplacian of q. The overall scaling factor α, is taken proportional to the maximum eigenvalue of the Euler equations for the inviscid flow calculations:

$$\alpha_i = A_i(U_i + c) \tag{3.4}$$

where A_i is the average area of all the faces forming the cell i. The first term in the above equation constitutes a relatively strong second-order dissipation term which is necessary to prevent unphysical conditions in the vicinity of a shock or stagnation point. To preserve the second-order accuracy of the scheme, this term must be turned off in regions of smooth flow.

This is accompanied by evaluating $k_2^{'}$ at cell i as:

$$(k_2^{'})_i = k_2 \frac{\sum_{k=1}^{n}[p_k - p_i]}{\sum_{k=1}^{n}[p_k + p_i]} \tag{3.5}$$

Hence $k_2^{'}$ is proportional to an undivided Laplacian of the pressure, which is constructed as a summation of the pressure differences along all faces constituting cell i. This construction has the required property of being of order unity near a shock and small elsewhere. k_2 is an empirically determined coefficient which is taken as 0 for subcritical flows (i.e., $M < 0.85$) and 1/2 for transonic and supersonic flows (i.e., $M \geq 0.85$). The value of k_4, the fourth-order artificial dissipation coefficient, depends on the specific application, but is typically in the range 1/256 to 1/64.

Finally, the artificial dissipation term $D(q_i)$ is calculated for every cell in the domain and is subtracted from the existing residual value for that cell to give the new residual. This residual is then used in the Jameson-style Runge-Kutta timestepping scheme.

Although this term for the central differencing operator has been implemented in PUMA, its success has not yet been determined. Initial testing with the RAE2822 and the M6 ONERA wing shock problems have had mixed results. Current work is under progress to test this and also compare it with upwinding schemes. Once the central differencing operator is working to satisfaction, implementation of NLDE and k-exact will commence for PUMA.

Chapter 4

Parallel Computers

4.1 COst effective COmputing Array (COCOA)

4.1.1 Introduction

The COst effective COmputing Array (COCOA) is a Penn State Department of Aerospace Engineering initiative to bring low cost parallel computing to the departmental level. COCOA is a 50 processor cluster of off the shelf PCs connected via fast-ethernet (100 Mbit/sec). The PCs are running RedHat Linux with MPI for parallel programming and DQS for queueing the jobs. Each node of COCOA consists of Dual 400 MHz Intel Pentium II Processors in SMP configuration and 512 MB of memory. A single Baynetworks 24-port fast-ethernet switch with a backplane bandwidth of 2.5 Gbps was used for the networking. The whole system cost approximately $ 100,000 (1998 US dollars). Detailed information on how COCOA was built can be obtained from Appendix A [40]. COCOA was built to enable the study of complex fluid dynamics problems using CFD without depending on expensive external supercomputing resources. Most of the runs for this work were done on COCOA.

4.1.2 Benchmarks

Some of the benchmarks conducted on COCOA are described here. While they are by no means comprehensive, they give a good idea about COCOA's performance for the kind of problems that are proposed to be tackled.

Since the cluster was primarily intended for fluid dynamics related applications, PUMA was chosen as one of the main benchmarks. Figures 4.1, 4.2, 4.3 and 4.4 show the performance obtained from an inviscid run on a general ship shape (GSS) geometry on different number of processors using PUMA. For this case, an unstructured tetrahedral grid with 483565 cells and 984024 faces was used and the run consumed 1.2 GB of physical memory. The benchmark clearly shows that COCOA was almost twice as fast as the Penn State IBM SP (older RS/6000-370 nodes) for our applications.

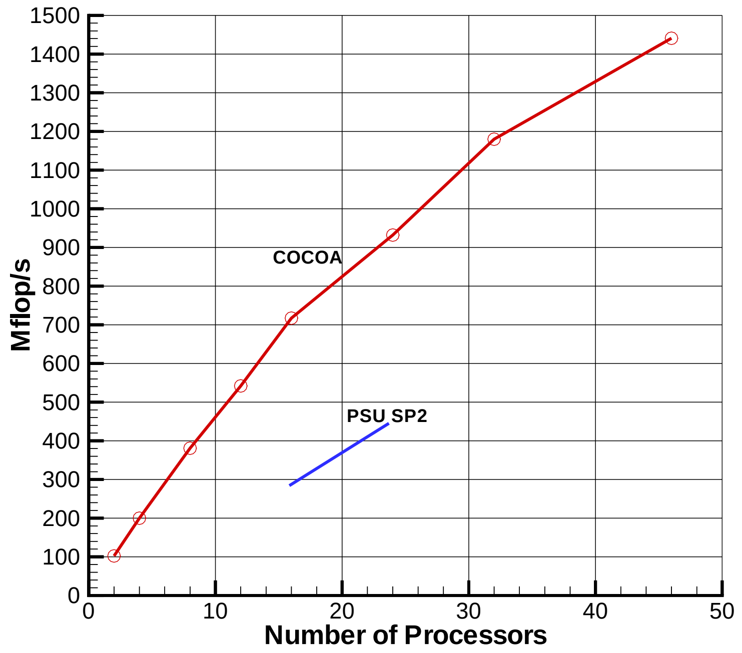

Figure 4.1. Total Mflops vs Number of Processors on COCOA for PUMA test case

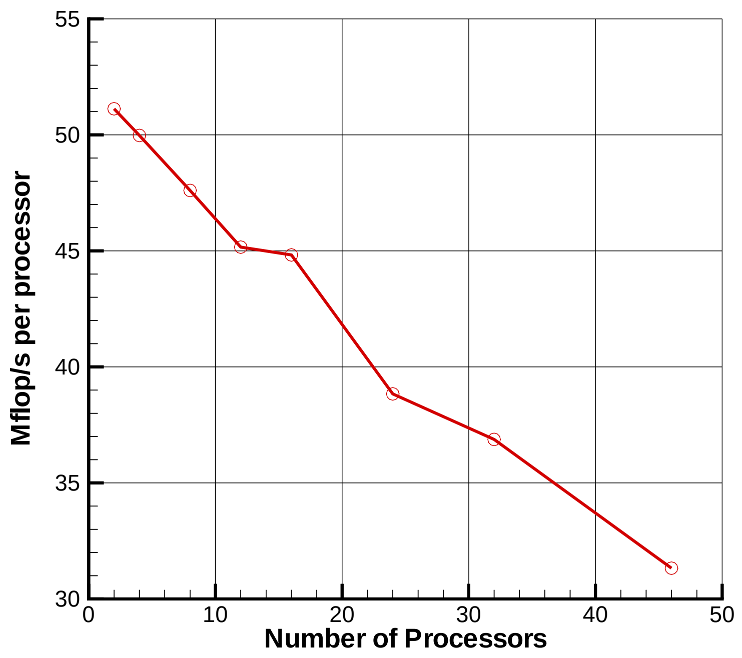

Figure 4.2. Mflops per processor vs Number of Processors on COCOA for PUMA test case

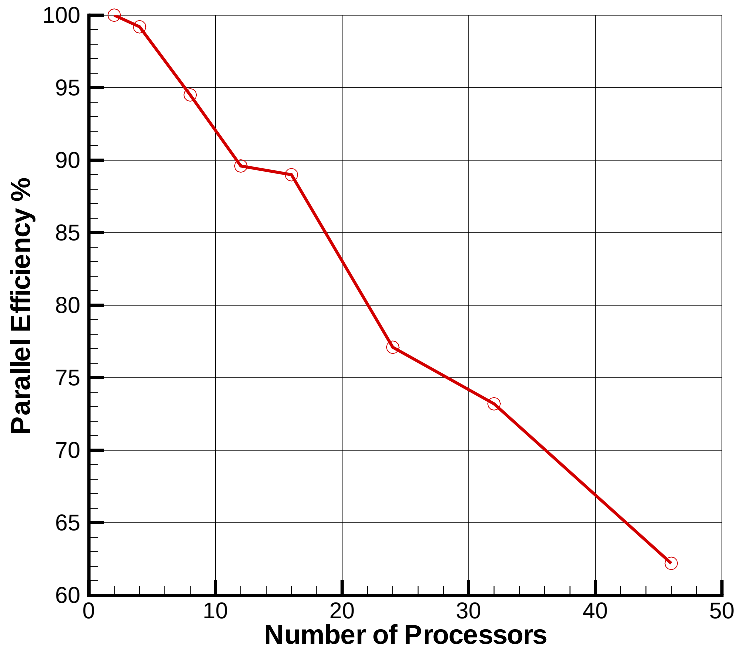

Figure 4.3. Parallel Efficiency vs Number of Processors on COCOA for PUMA test case

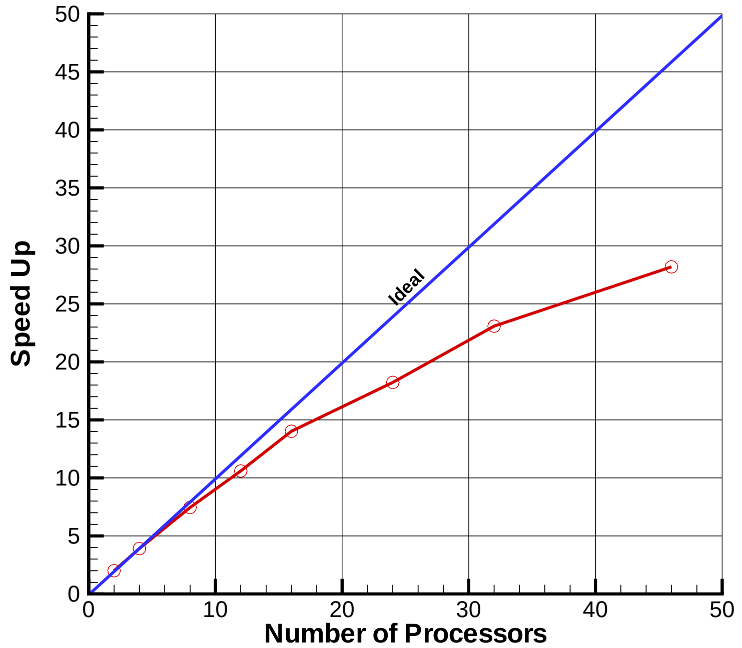

Figure 4.4. Speed-up vs Number of Processors on COCOA for PUMA test case

A set of well-known parallel benchmarks related to CFD were also conducted on COCOA using the publicly available "NAS[1] Parallel Benchmarks (NPB) suite v2.3" [41] written in Fortran 77. Of the eight tests contained in the suite, five were *kernel benchmarks* (EP, MG, CG, FT, and IS) and the other three were *Simulated CFD Application Benchmarks* (LU, SP, and BT). A very brief description on each test is given below:

1. **The Embarrassingly Parallel (EP) Benchmark**: The first of the five kernel benchmarks is an *embarrassingly parallel* problem. In this benchmark, two-dimensional statistics are accumulated from a large number of Gaussian pseudo-random numbers, which are generated according to a particular scheme that is well-suited for parallel computation. This problem is typical of many *Monte Carlo* applications. Since it requires almost no communication, in some sense this benchmark provides an estimate of the upper achievable limits for floating-point performance on a particular system. Results for this benchmark on COCOA are shown in figure 4.5.

2. **Multigrid (MG) Benchmark**: The second kernel benchmark is a simplified multigrid kernel, which solves a 3D Poisson PDE. This problem is simplified in the sense that it has constant rather than variable coefficients as in a more realistic application. This code is a good test of both short and long distance highly structured communication. The Class B problem uses the same size grid as of Class A but a greater number of inner loop iterations. Results for this benchmark on COCOA are shown in figure 4.6.

3. **Conjugate Gradient (CG) Benchmark**: In this benchmark, a conjugate gradient method is used to compute an approximation to the smallest eigenvalue of a large, sparse, symmetric positive definite matrix. This kernel is typical of unstructured grid computations in that it tests irregular long-distance communication and employs sparse matrix-vector multiplication. Results for this benchmark on COCOA are shown in figure 4.7.

4. **3D FFT PDE (FT) Benchmark**: In this benchmark a 3-D partial differential equation is solved using Fast Fourier Transforms (FFTs). This kernel performs the essence of many spectral methods. It is a good test of long-distance communication performance. The rules of the NPB specify that assembly-coded, library routines may be

[1]NAS stands for *Numerical Aerospace Simulation* facility, and is a part of NASA Ames Research Center

used to perform matrix multiplication and one-dimensional, two-dimensional, or three-dimensional FFTs. Thus this benchmark is somewhat unique in that computational library routines may be legally employed. Results for this benchmark on COCOA are shown in figure 4.8.

5. **Integer Sort (IS) Benchmark**: This benchmark tests a sorting operation that is important in particle method codes. This type of application is similar to particle-in-cell applications of physics, wherein particles are assigned to cells and may drift out. The sorting operation is used to reassign particles to the appropriate cells. This benchmark tests both integer computation speed and communication performance. This problem is unique in that floating point arithmetic is not involved. Significant data communication, however, is required. Results for this benchmark on COCOA are shown in figure 4.9.

6. **LU Simulated CFD Application (LU) Benchmark**: The first of these is the so-called the lower-upper diagonal (LU) benchmark. It does not perform a LU factorization but instead employs a symmetric successive over-relaxation (SSOR) numerical scheme to solve a regular-sparse, block 5×5 lower and upper triangular system. This problem represents the computations associated with a newer class of implicit CFD algorithms, typified at NASA Ames by the code INS3D-LU. This problem exhibits a somewhat limited amount of parallelism compared to the next two benchmarks. A complete solution of the LU benchmark requires 250 iterations. Results for this benchmark on COCOA are shown in figure 4.10.

7. **SP Simulated CFD Application (SP) Benchmark**: The second simulated CFD application is called the scalar pentadiagonal (SP) benchmark. In this benchmark, multiple independent systems of non-diagonally dominant, scalar pentadiagonal equations are solved. A complete solution of the SP benchmark requires 400 iteration. Results for this benchmark on COCOA are shown in figure 4.11.

8. **BT Simulated CFD Application (BT) Benchmark**: The third simulated CFD application is called the block tridiagonal (BT) benchmark. In this benchmark, multiple independent systems of non-diagonally dominant, block tridiagonal equations with a 5×5 block size are solved. SP and BT are representative of computations associated with the implicit operators of CFD codes such as ARC3D at NASA Ames. SP and BT are similar in many respects, but there is a fundamental difference with respect

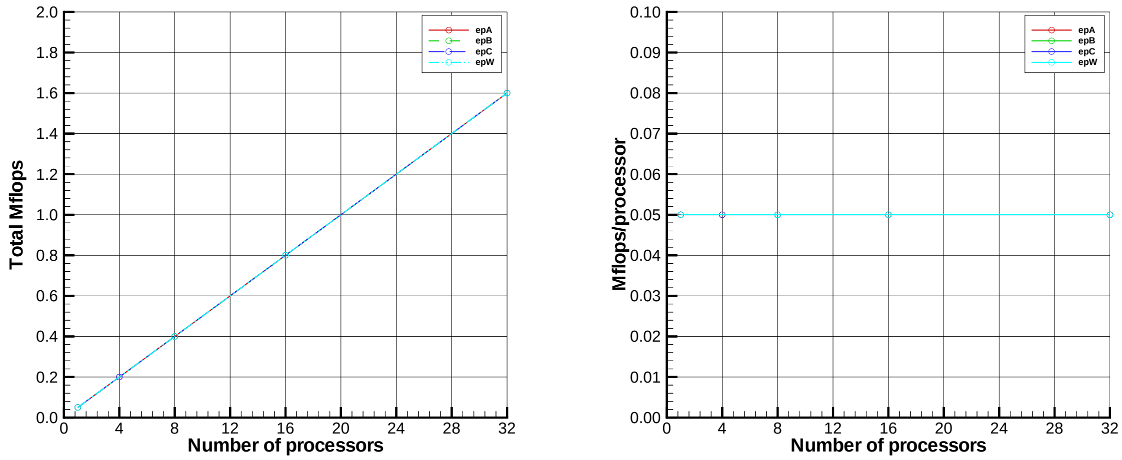

Figure 4.5. NAS Parallel Benchmark on COCOA: "Embarrassingly Parallel" (EP) test

to the communication to computation ratio. Timings are cited as complete run times, in seconds, as with the other benchmarks. For the BT benchmark, 200 iterations are required. Results for this benchmark on COCOA are shown in figure 4.12.

There were four different problem sizes for each test: Class "W", "A", "B" and "C". While class "W" was the workstation-size test (smallest), size "C" was supercomputer-size test (biggest). The problem size for the classes "A", "B" and "C" for each of the eight tests is given in table 4.1.

Benchmark Code	**Class A**	**Class B**	**Class C**
Embarrassingly parallel (EP)	2^{28}	2^{30}	2^{32}
Multigrid (MG)	256^3	256^3	512^3
Conjugate gradient (CG)	1.4×10^4	7.5×10^4	1.5×10^5
3D FFT PDE (FT)	$256^2 \times 128$	512×256^2	512^3
Integer sort (IS)	2^{23}	2^{25}	2^{27}
LU solver (LU)	64^3	102^3	162^3
Pentadiagonal solver (SP)	64^3	102^3	162^3
Block tridiagonal solver (BT)	64^3	102^3	162^3

Table 4.1. NAS Parallel Benchmarks Problem Size

Figure 4.13 shows the results from the class "C" LU test. These tests (class "C") are specifically designed for supercomputers with large problem solving capacity. The LU benchmark solves a finite difference discretization of the 3D compressible Navier-Stokes equations through a block-lower-triangular and block-upper-triangular approximate factorization of

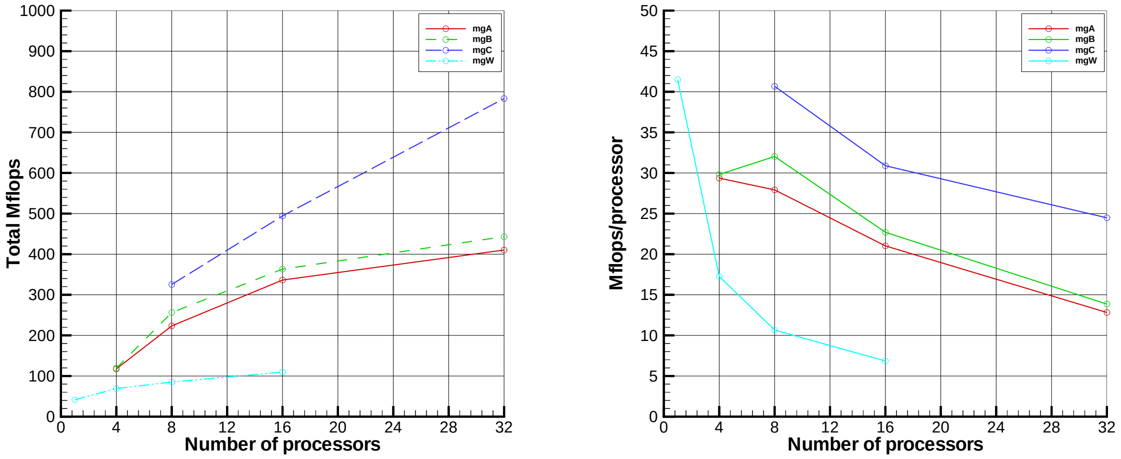

Figure 4.6. NAS Parallel Benchmark on COCOA: “Multigrid” (MG) test

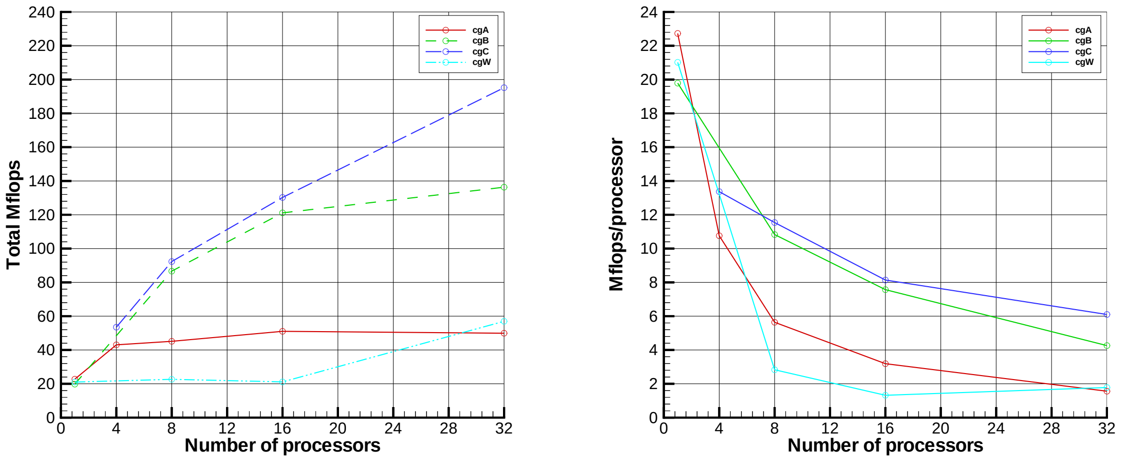

Figure 4.7. NAS Parallel Benchmark on COCOA: “Conjugate Gradient” (CG) test

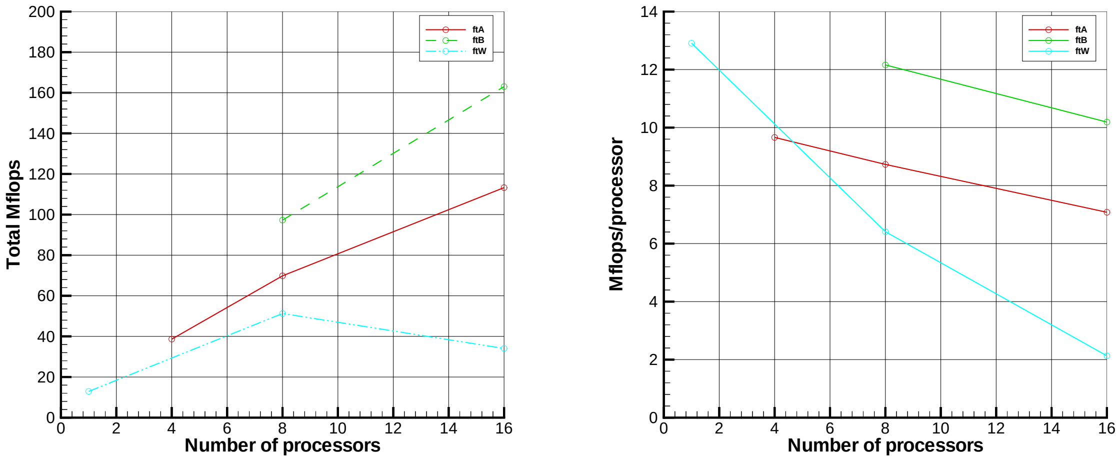

Figure 4.8. NAS Parallel Benchmark on COCOA: “3D FFT PDE” (FT) test

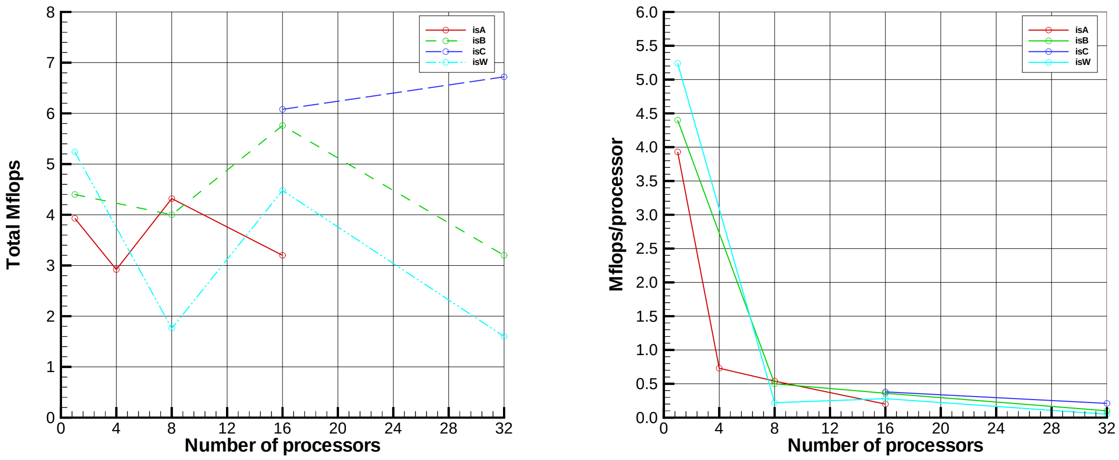

Figure 4.9. NAS Parallel Benchmark on COCOA: “Integer Sort” (IS) test

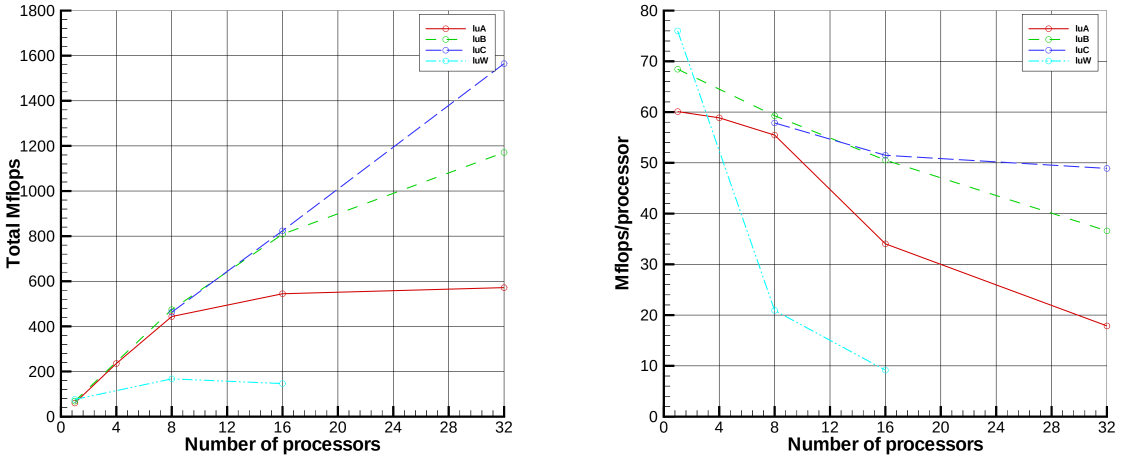

Figure 4.10. NAS Parallel Benchmark on COCOA: “Lower-Upper diagonal solver” (LU) test

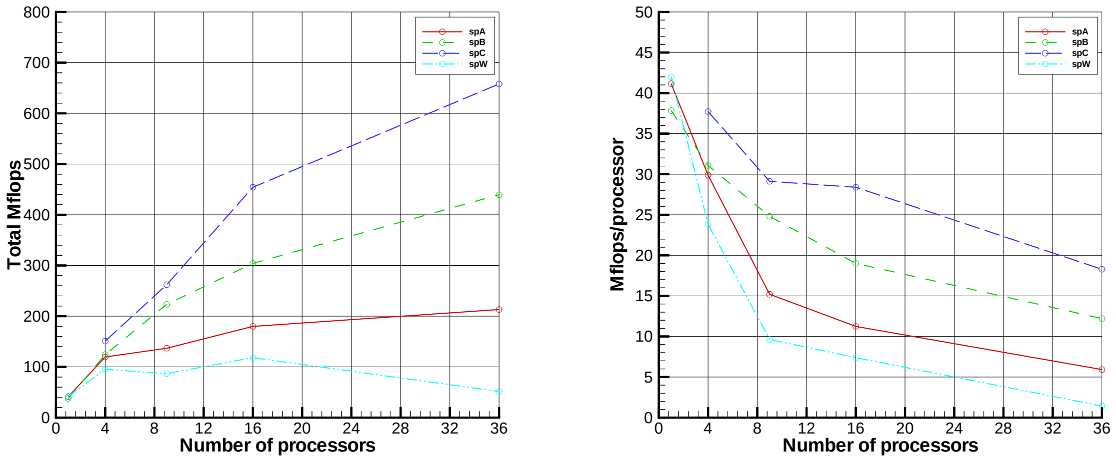

Figure 4.11. NAS Parallel Benchmark on COCOA: “Scalar Pentadiagonal solver” (SP) test

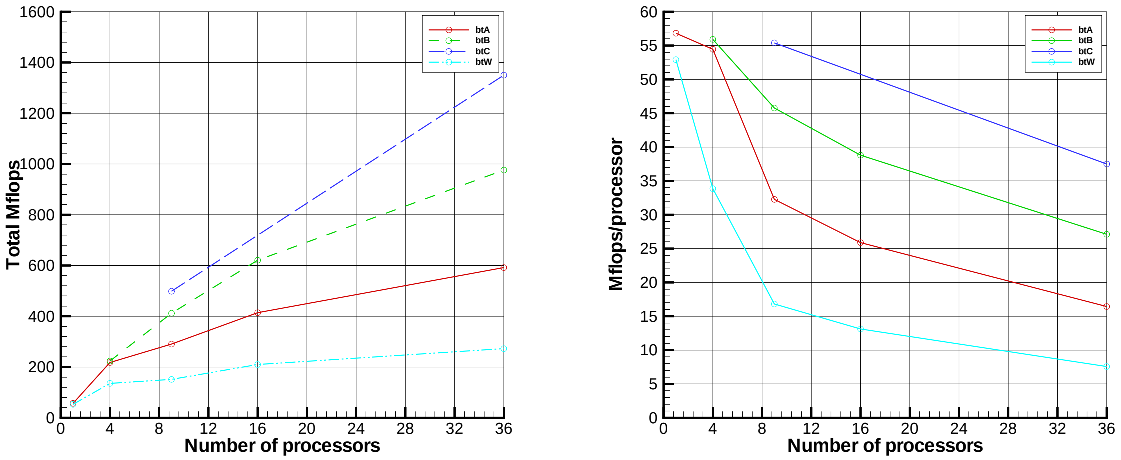

Figure 4.12. NAS Parallel Benchmark on COCOA: "Block Tridiagonal solver" (BT) test

the original difference scheme. The LU factored form is cast as a relaxation, and is solved by SSOR. This benchmark shows that although COCOA is the slowest among the supercomputers compared, it still offers comparable performance at a fraction of the cost.

Figure 4.14 shows the results obtained from the `netperf` test (done using: `netperf -t UDP_STREAM -l 60 -H <target-machine> - -s 65535 -m <packet-size>`). This is indicative of the communication speed between any two nodes. It is seen that almost 96% of the peak communication speed of 100 Mbit/sec is achieved between every two nodes for packet sizes above 1000 bytes. The valley seen around 1500 bytes is due to the fact that the *Maximum Transfer Unit* (MTU) of the ethernet driver is set to 1500 bytes. Thus, when packet size goes just above 1472 bytes (since header information takes 28 bytes), the packet has to be broken into more than one *User Datagram Protocol* (UDP) packet causing the communication speed to decrease slightly. To see how MPI performs on COCOA, an `MPI_Send/Recv` test was done with several number of processors and different packet sizes. As seen from figure 4.15, it is clear that communication speed is best for packet sizes greater than 10 Kbytes (in the range 60 to 90 Mbits/sec between any two nodes). The effect of the SMP nodes can also be clearly seen here, as the communication speed goes down significantly from the 24 processor case to the 32 processor case. This is due to the fact that all our 46 processors are housed in 23 different nodes, each having only one ethernet card. Thus, when more than 23 nodes are requested on COCOA, some nodes have more than one processor active, thus dividing the available communication bandwidth among the single ethernet card

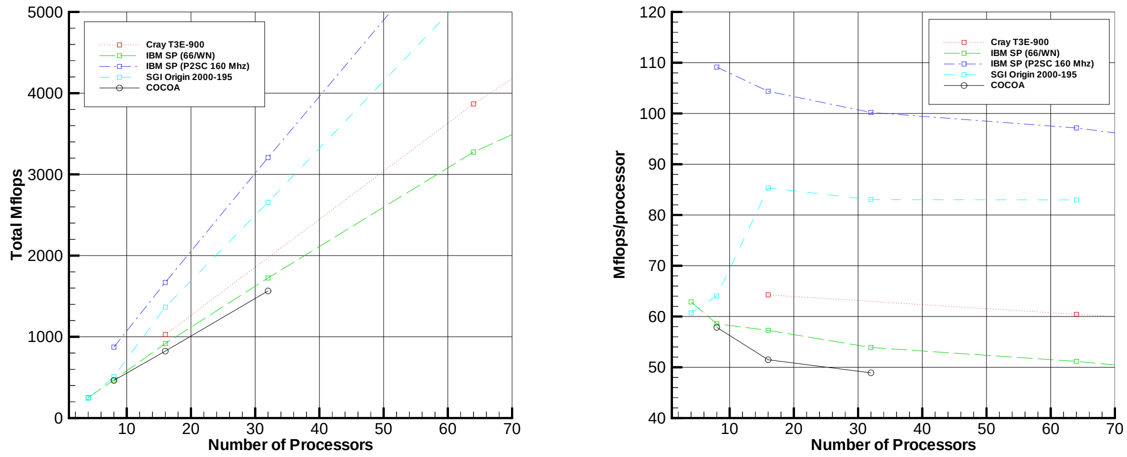

Figure 4.13. NAS Parallel Benchmark on COCOA: comparison with other machines for Class “C” LU test

and slowing down the effective communication speed.

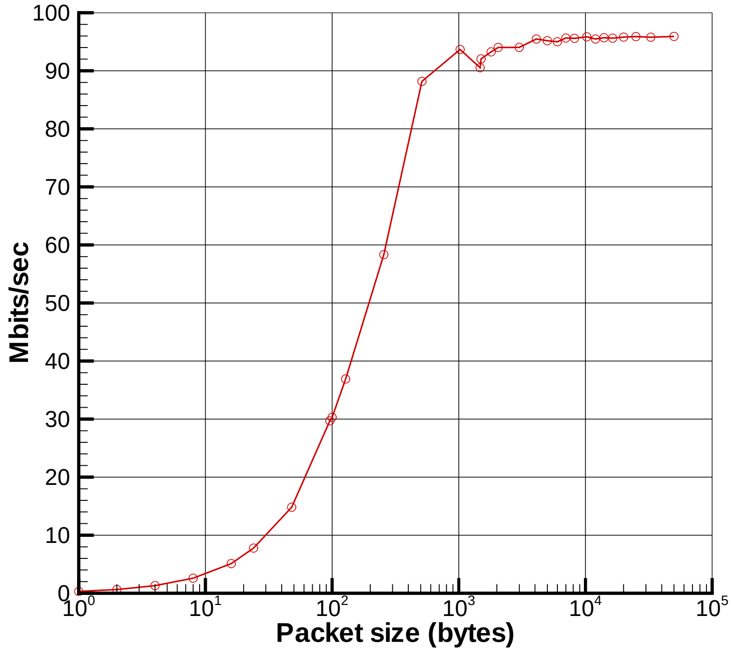

Figure 4.14. Mbits/sec vs Packet size on COCOA for netperf test

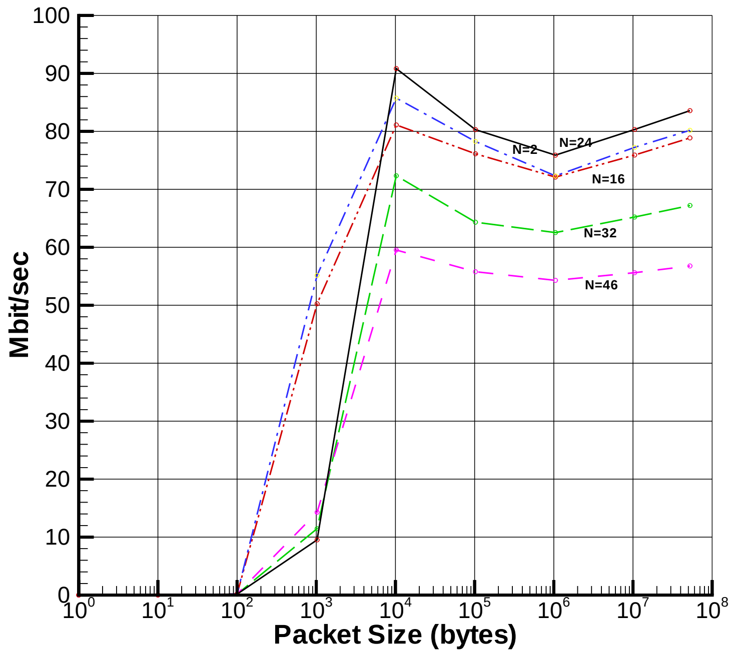

Figure 4.15. Mbits/sec vs Packet size on COCOA for MPI_Send/Recv test

Chapter 5

Post-processing and Visualization

5.1 Post-processing output from PUMA

Post-processing the output data from PUMA in order to visualize the solution was a very important part of all the runs. A lot of effort was put into this to make the whole process fast and automated.

Since TECPLOT was the primary visualization software used for all our work, a utility `toTecplot` was written in C to convert the restart data (in binary format) from PUMA to a TECPLOT output file. This was necessary as PUMA computes the solution data at the cell centers, whereas TECPLOT requires it at the nodes. Also, the restart file contained no information about the grid geometry, and hence the corresponding grid file had to be read by `toTecplot` to map the solution to the geometry correctly. The `toTecplot` utility also had several features which enabled only specific surfaces to be displayed instead of the entire volume data. Functions to calculate *vorticity* ($\nabla \times V$) and *dilatation* ($\nabla . V$) were added to the utility to facilitate in the visualization of unsteady phenomena like vortex shedding and wake propagation. These functions were non-trivial as they had to operate on unstructured grids.

5.2 Live CFD cam

One of the most annoying problems with running any reasonably complex CFD simulation is to determine the correct solver parameters (e.g., CFL number, number of inner iterations, tolerance). These are often done by experimenting with these numbers in conjunction with looking at the solution. All this requires post-processing of the entire data repetitively which consumes a lot of time and effort. Another important requirement for any good numerical simulation package is that the solution be visualized frequently so that one can see the changes in the flow solution as they occur. This is especially important in the case of a time-accurate run for an unsteady flow simulation. One can also come to know in advance if the solution seems to be diverging and take corrective action without wasting a lot of expensive computational resources. To facilitate all this, the entire post-processing and visualization phase were automated to create what is referred to as the "Live CFD-cam".

5.2.1 Implementation

Several scripts had to be written in order to implement the idea of the "Live CFD-cam". The main steps are listed below:

1. Some minor changes were made to the PUMA source to generate an output file containing the name of the last completely written restart file. This was necessary since the restart files were written in several parts with each node writing its own portion, all of which were combined to give the final file. Also, consecutive restart files could have different names based on the iteration number or the time elapsed. If the script updating the data was to be run in real-time, it needed to know the name of the last restart file generated immediately, as well as needed the guarantee that all the nodes have completely written their portions of the restart files so that the post-processing could begin. This was best achieved by modifying PUMA to update a certain file as soon as it got the green signal from every node saying that their part of the restart file had been completely written.

2. A server side shell script - `server_cfd` (Appendix C.1), was written, which detected the creation of a new restart file and acted accordingly to process the restart file to generate a corresponding image file. The main steps involved in this were:

    (a) Reading an initialization file `SERVER.conf` (Appendix C.2) containing important information regarding the location of the grid file, the names of the TECPLOT layout files, size of the output images to be generated, and the final destination of these files. This was added for generality and flexibility, so that several CFD-cams could function simultaneously without conflicting, each displaying totally different cases.

    (b) Detection for the creation of a new restart file.

    (c) Combining the portions of restart files generated by each node into one restart file which was ready for further processing.

    (d) Using `toTecplot` to read the restart file and the corresponding grid geometry file to generate a TECPLOT output file.

(e) Another useful utility `tec2gif` (Appendix C.3) was written in shell script which converted any TECPLOT data file into a GIF image file using a given layout file as an input. This was done in *batch mode*, i.e. without invoking any graphical interface and by simply utilizing the batch and macro feature of TECPLOT and several public domain packages like `ghostview` and `ImageMagick`. `tec2gif` was then used to generate a single GIF image file from the above TECPLOT file corresponding to each layout file supplied, displaying a specific flow region or variable of interest.

(f) Creating an animated GIF from the above GIF files using the utilities from the `ImageMagick` software package.

(g) Reading the current residual history file and generating the plot of convergence as a GIF image file.

(h) Sending all the above mentioned GIF image files with relevant and updated convergence data over the network to the machine hosting the *Live CFD-cam* web-page. This was again done in *batch mode* without the need of passwords using the `secure-shell (ssh)` program.

3. A client side (web-server side) shell script - `client_cfd` (Appendix C.4), was written, which detected the receipt of new data files from the server script and updated the web-page accordingly. The main steps involved in this were:

   (a) Detection for new set of data/images sent by the server script every fixed interval of time.

   (b) If updated files are found, the program reads in a given set of template HTML files, and generates corresponding new HTML files which contain the correct information with links for the images and data.

   (c) All the above HTML files have a `REFRESH` tag which enable the pages to automatically reload itself every specified interval thus giving an impression of a Live update.

The above implementation helped in seamlessly automating the entire process of post-processing and visualizing the complex flow data in real-time. A sample snapshot of the "Live CFD-cam" can be seen in figures 5.1 and 5.2.

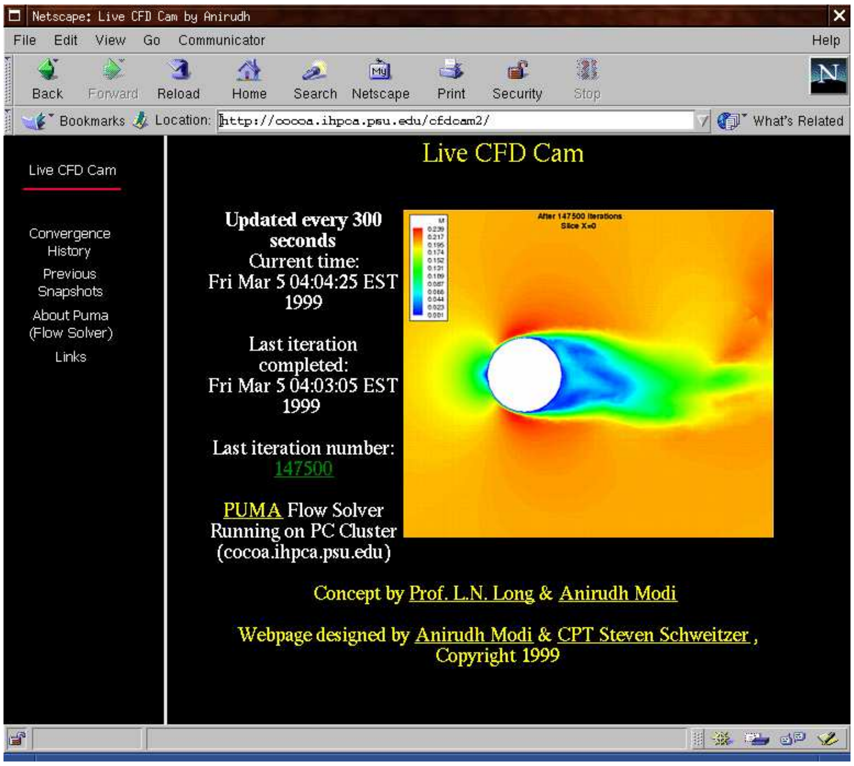

Figure 5.1. Live CFD-cam on COCOA showing the contour snapshots for the sphere run

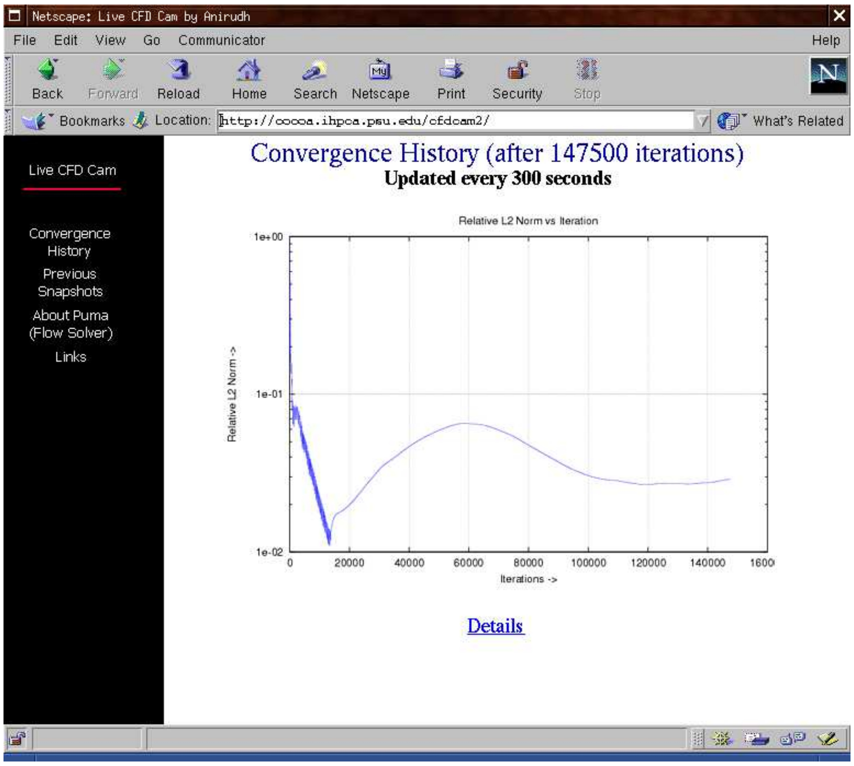

Figure 5.2. Live CFD-cam on COCOA showing the convergence history for the sphere run

Chapter 6

Results

6.1 Various Ship Configurations

6.1.1 General Ship Shape (GSS)

Increasing use of helicopters in conjunction with ships poses many major problems. In the presence of high winds and rough sea, excessive ship motions and turbulent, separated flow from sharp-edged box-like ship super-structures make landing a helicopter on ships a very hazardous operation. The strong unsteady flows can cause severe rotor blade deformations. There have been numerous incidences where the helicopter blades have actually impacted the helicopter fuselage, which is called a 'tunnel strike'. Statistically, a helicopter can safely land on a frigate in the North Sea only 10 percent of the time in winter. In addition, flight simulators also have no adequate models for airwake. In order to avoid this and other engage/disengage problems, determining safe operating envelopes is very costly and time consuming. On the other hand, many numerical simulation attempts of this flow field for minimizing cost have not been successful due to the inherently unsteady nature of flow and the low-speed character of the flow (which may cause numerical stiffness).

Recent research on ship airwakes has been conducted from several different approaches[1]. One of the sources of relevant research is building aerodynamics which shows the general features of flow about blunt bodies of different aspect ratios, and about clusters of buildings. The most likely model of a ship, but rather crude, is a sharp edged blunt body called the General Ship Shape (GSS). The GSS is a standard shape set forth by the United States Navy as a geometry to be used to compare flow solvers. The GSS represents a typical United States Navy frigate with the area of interest being the flow field on the aft deck. The aft deck primarily acts as a landing area for helicopters. The turbulent, separated flow caused by the ship structure can make landing a helicopter extremely difficult. The strong unsteady flows can also cause severe blade deformation during start up and shut down of a helicopter. This GSS geometry unstructured grid and solution was completed in conjunction with the Pennsylvania State University ship airwake studies [3]. More geometrically precise studies have been carried out in wind tunnels [4, 5, 6] and full scale tests have been conducted by the US Navy [7], which provide some important information on real ship airwakes. All these experimental tests are crucial for validating numerical models. Wind tunnel tests can be quite costly, and the flow measurements on real Naval ships is very difficult and costly to obtain.

It would be very useful to have numerical methods that could simulate ship airwakes. There have been other attempts at numerically simulating ship airwakes, e.g. a steady-state flow solver based on the 3D multi-zone, thin-layer Navier-Stokes (TLNS) method [8] and an unsteady, inviscid low-order method solver [9]. No method to-date has been entirely satisfactory for predicting these flow fields.

This was the first complex case which was used to validate PUMA. The flow conditions were taken to be $M_\infty = 0.065$ and the yaw angle $\beta = 30°$. Both inviscid and viscous ($Re = 1.15 \times 10^8$) cases were run [2]. For the inviscid run, a structured grid with 227120 cells and 694556 faces was used and the run consumed 1.2 GB of memory. For the viscous run, an unstructured tetrahedral grid with 483565 cells and 984024 faces was used and the run consumed 1.1 GB of memory. The convergence history for the viscous run is shown in in figure 6.1. It shows a plot of the relative residual[1] against the number of iterations/timesteps. The surface velocity on the GSS geometry as computed by PUMA is shown in figure 6.2. Qualitative comparisons were then made with the available experimental data. As seen in figures 6.4, 6.7 and 6.10, which compare both the inviscid and viscous solutions with the experiments, it is very clear that PUMA gives very good solution for this case. The viscous solution is seen to be clearly better in the region of flow separation.

The results although extremely good, can be improved by implementing the atmospheric boundary layer for the water surface, which has been currently taken as a friction-less surface.

Apart from the above runs, 14 other inviscid cases were run using the unstructured grid. These cases were for yaw angles of 0°, 30°, 60°, 90°, 120°, 150° and 180°, with two different flow speeds of 40 and 50 knots, respectively. The surface U/U_∞ contours for all these cases can be seen in figures 6.12, 6.13 and 6.14. These cases were run to study the effect of various flow conditions on the engage/disengage operations of ethe helicopter rotor. The region of interest for this study was the front deck of the ship (figure 6.15). More detail on this can be found in work by Keller and Smith [42].

[1] Ratio of the current residual to the initial residual

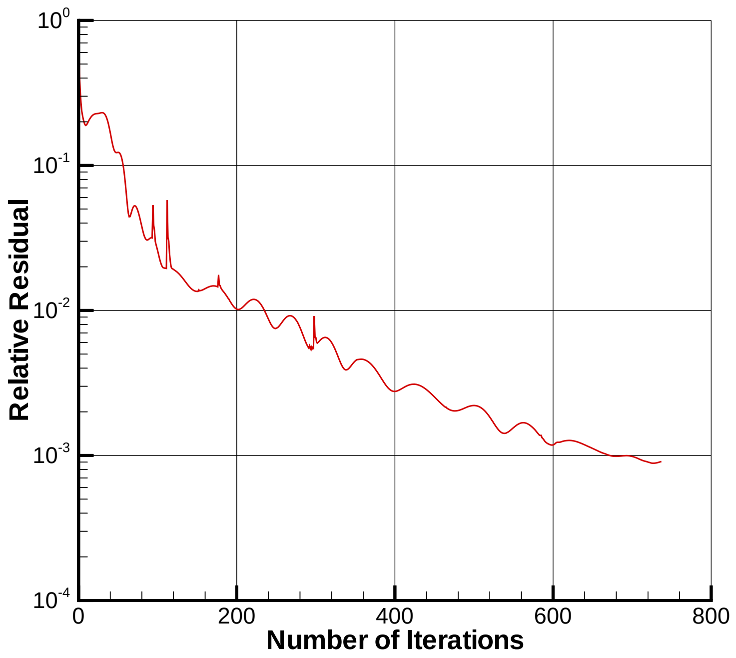

Figure 6.1. Convergence history for GSS viscous run for 30° yaw case

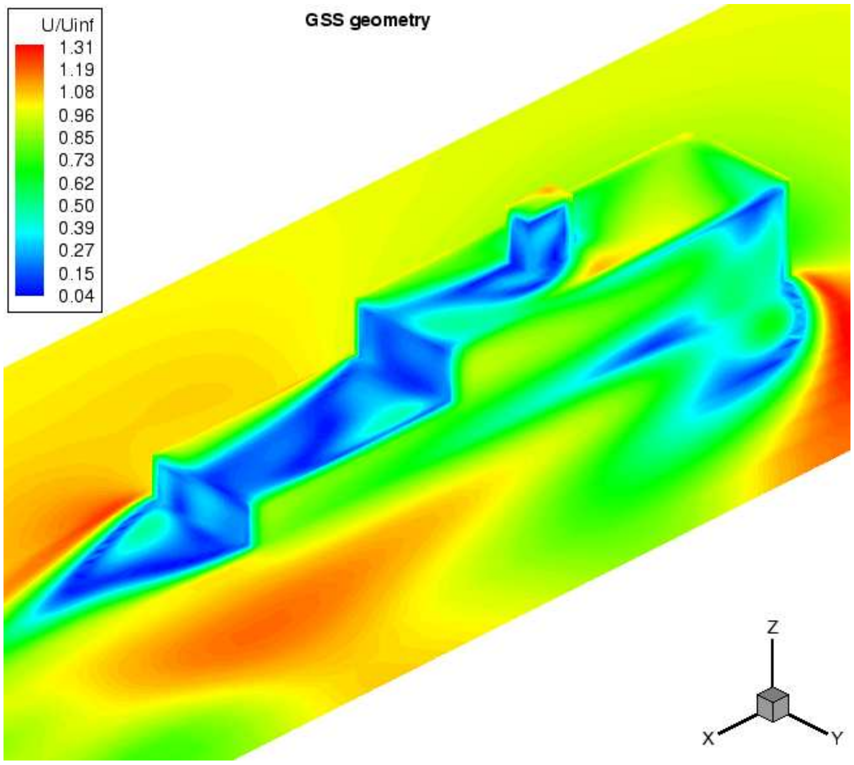

Figure 6.2. Surface Mach contours for GSS geometry for 30° yaw case

Figure 6.3. Oil flow pattern on front part of bridge deck of GSS for 30° yaw case

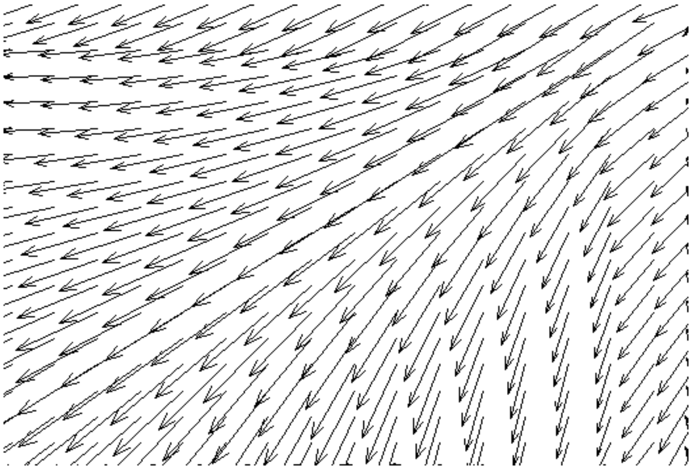

Figure 6.4. Vector plot on front part of bridge deck of GSS for 30° yaw case (inviscid case)

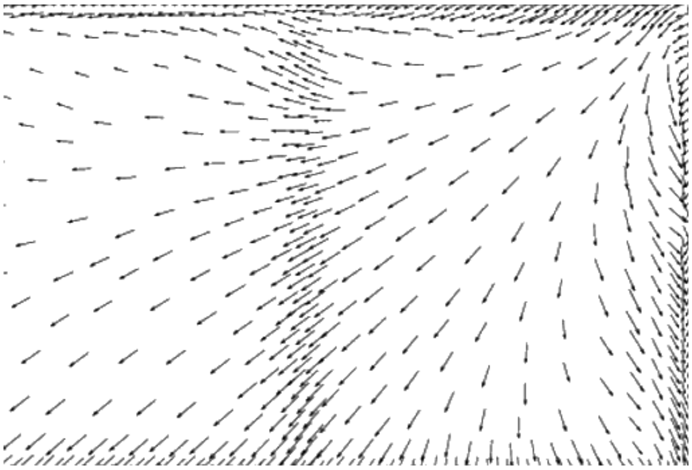

Figure 6.5. Vector plot on front part of bridge deck of GSS for 30° yaw case (viscous case)

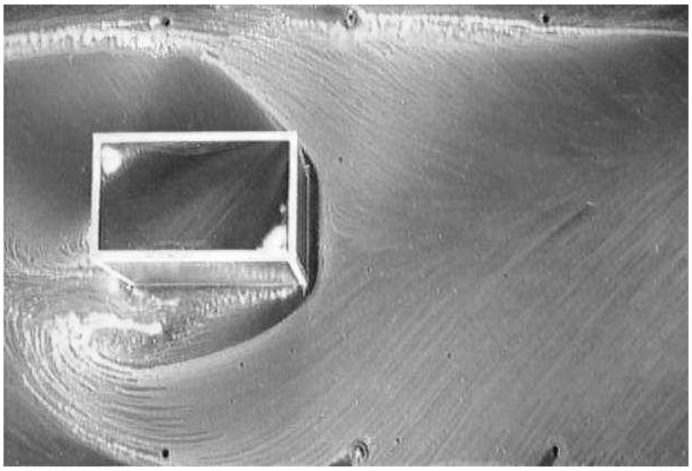

Figure 6.6. Oil flow pattern on middle part of bridge deck of GSS for 30° yaw case

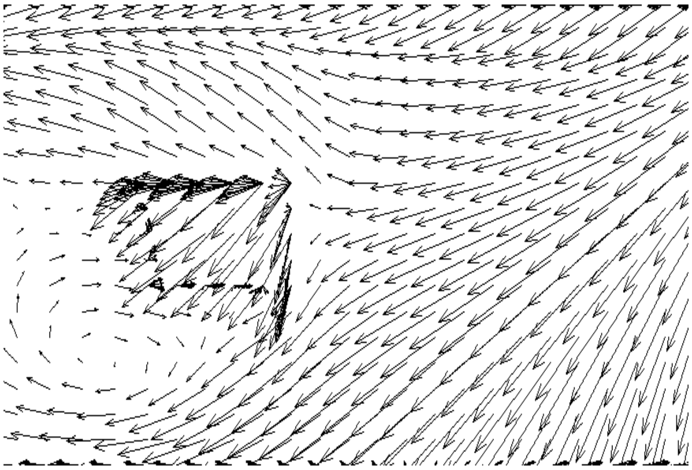

Figure 6.7. Vector plot on middle part of bridge deck of GSS for 30° yaw case (inviscid case)

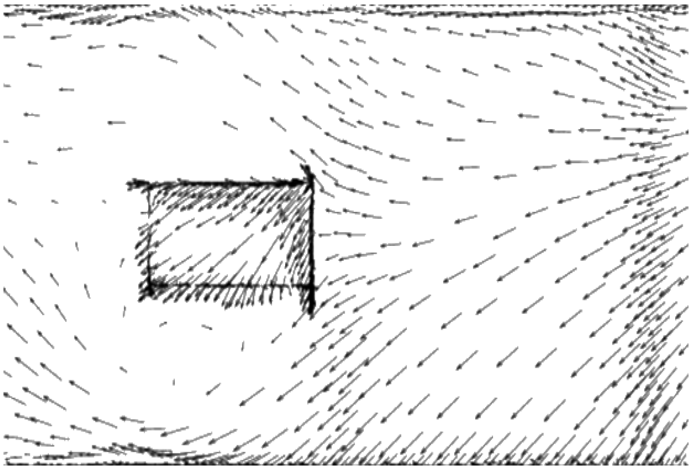

Figure 6.8. Vector plot on middle part of bridge deck of GSS for 30° yaw case (viscous case)

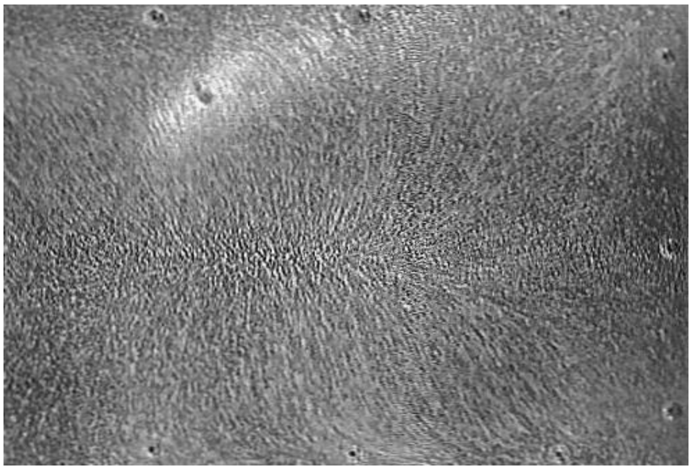

Figure 6.9. Oil flow pattern on front deck of GSS for 30° yaw case

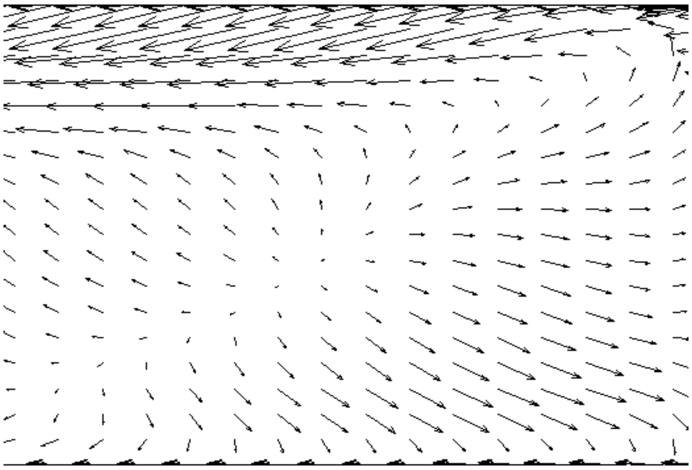

Figure 6.10. Vector plot on front deck of GSS for 30° yaw case (inviscid case)

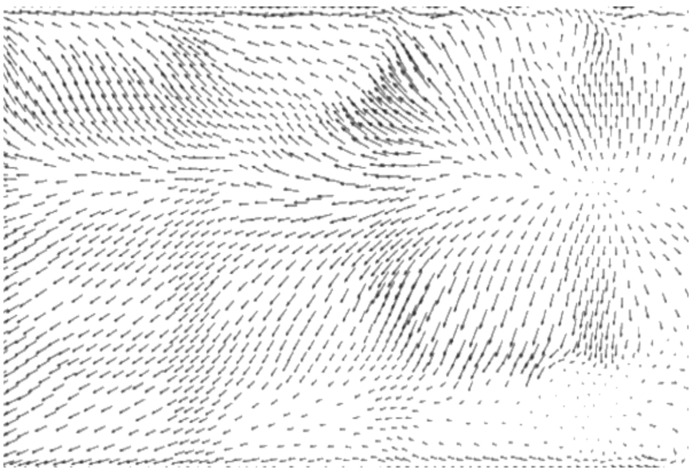

Figure 6.11. Vector plot on front deck of GSS for 30° yaw case (viscous case)

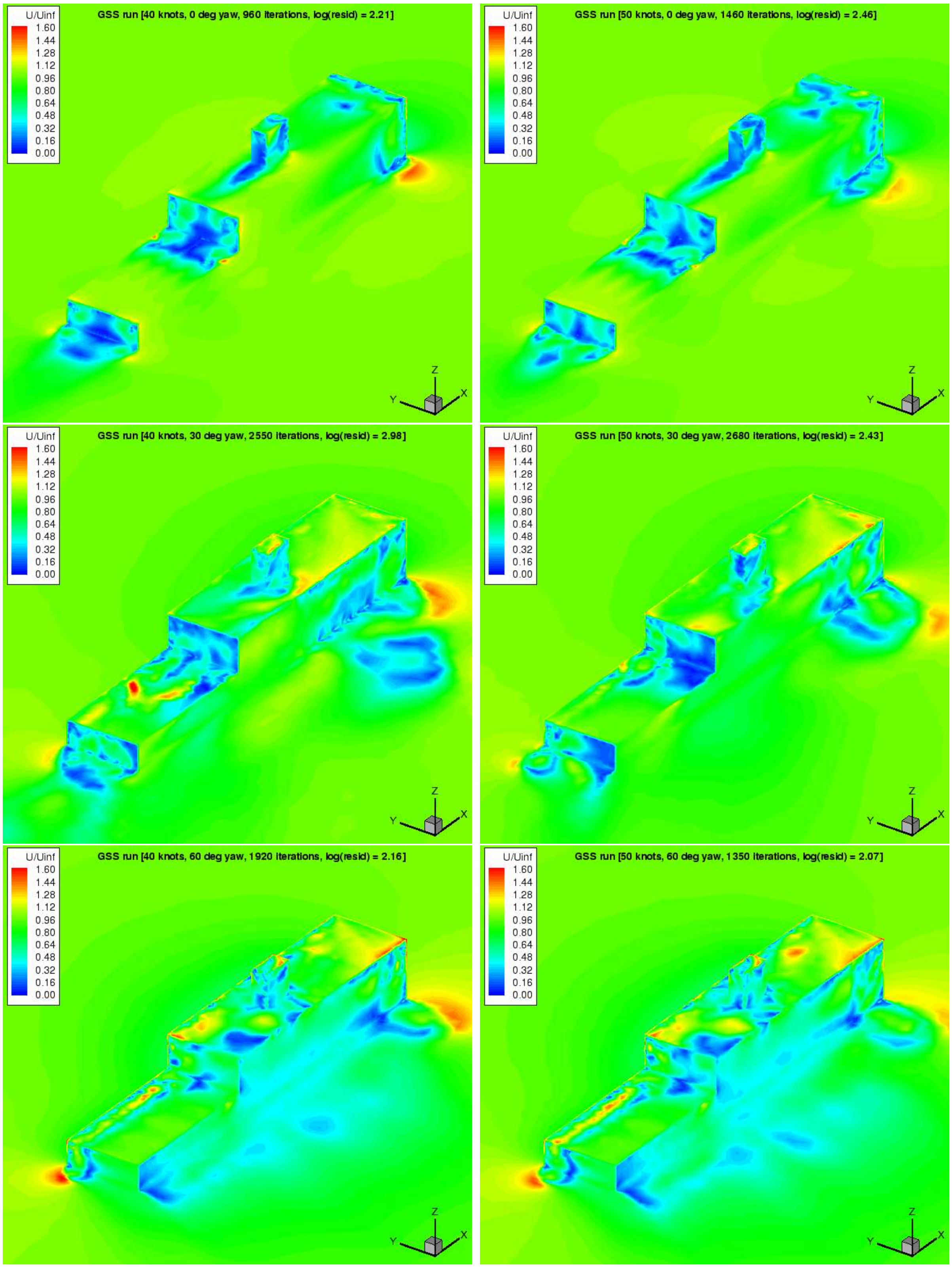

Figure 6.12. Surface U/U_{∞} contours for GSS geometry for flow speeds of 40 and 50 knots, with yaw angles of 0°, 30° and 60°, respectively

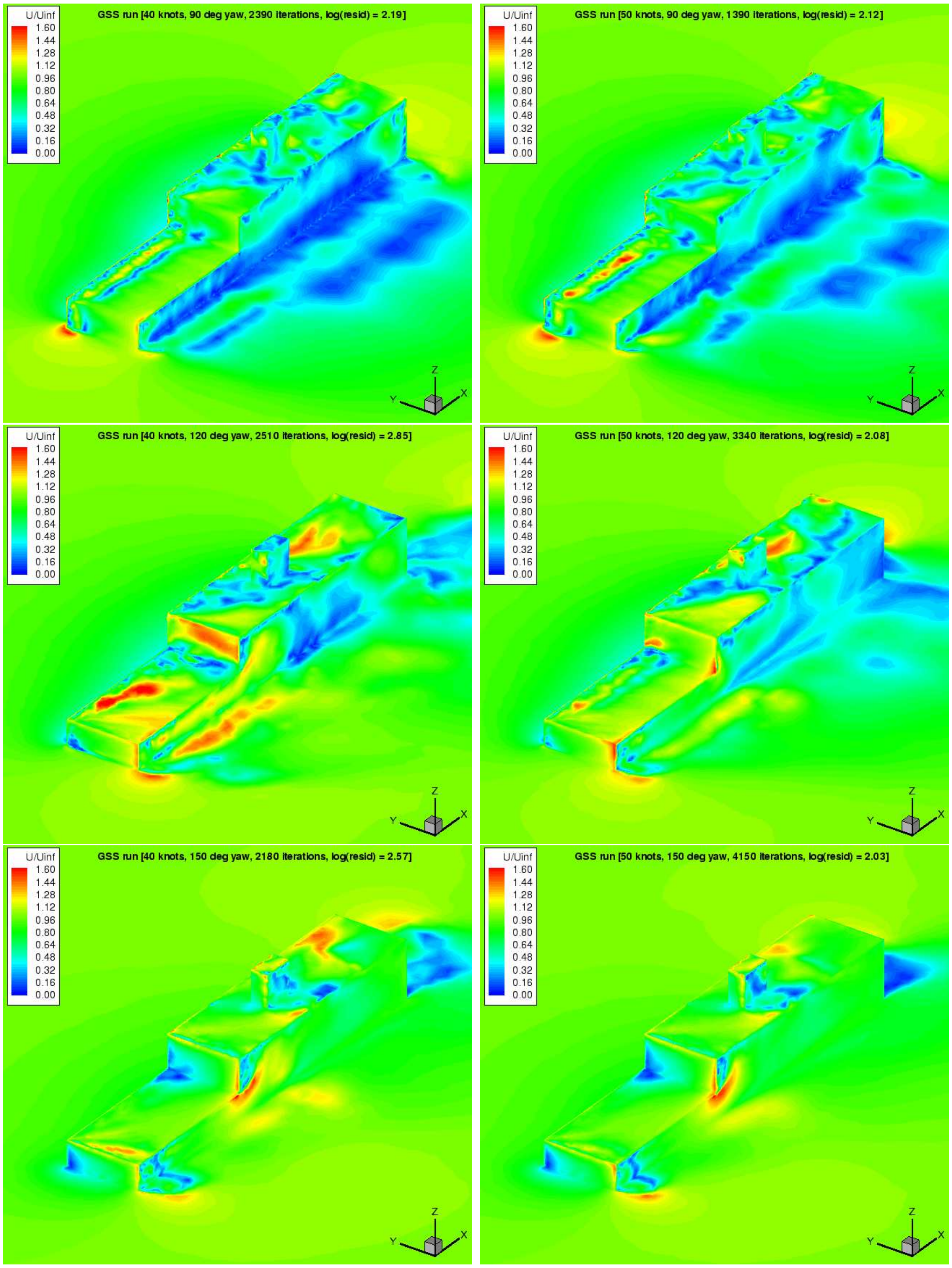

Figure 6.13. Surface U/U_∞ contours for GSS geometry for flow speeds of 40 and 50 knots, with yaw angles of 90°, 120° and 150°, respectively

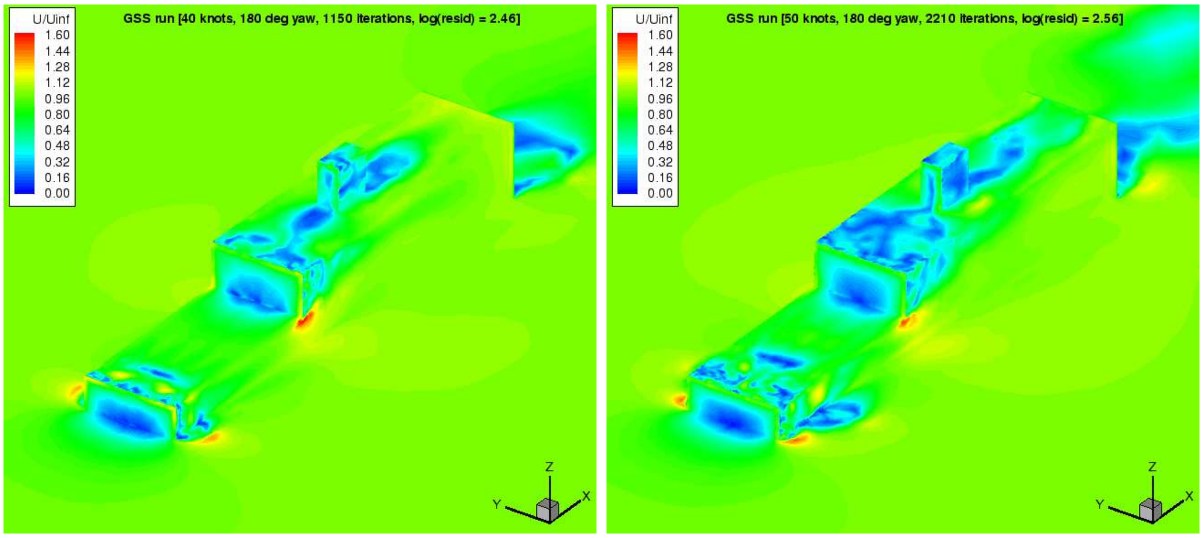

Figure 6.14. Surface U/U_∞ contours for GSS geometry for flow speeds of 40 and 50 knots, at yaw angle of 180°

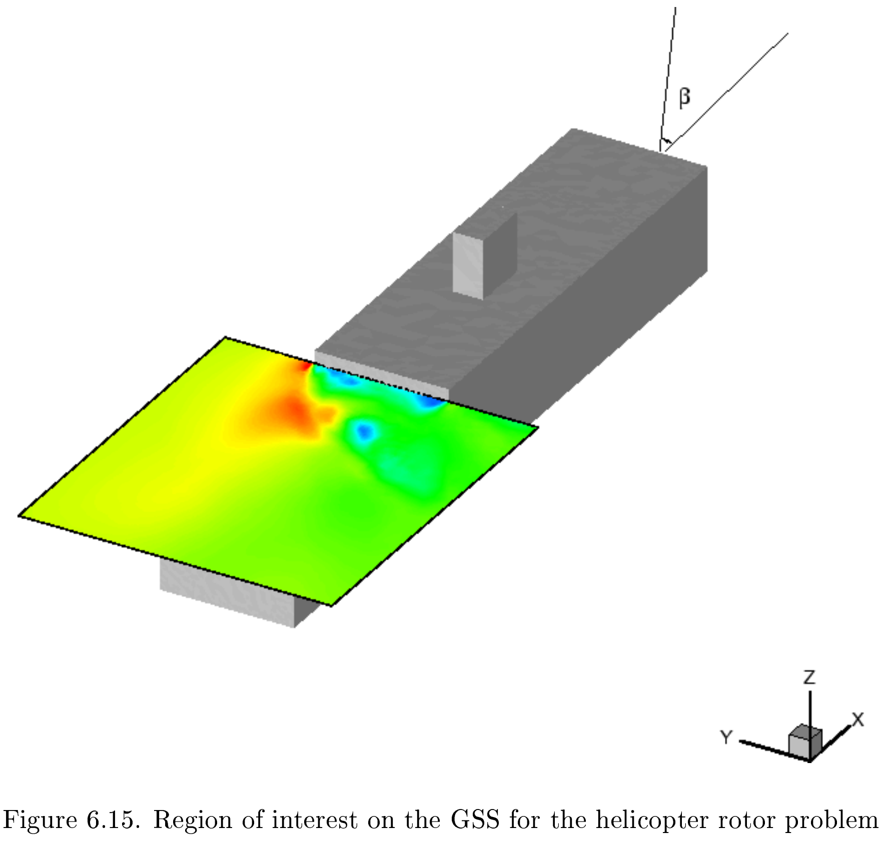

Figure 6.15. Region of interest on the GSS for the helicopter rotor problem

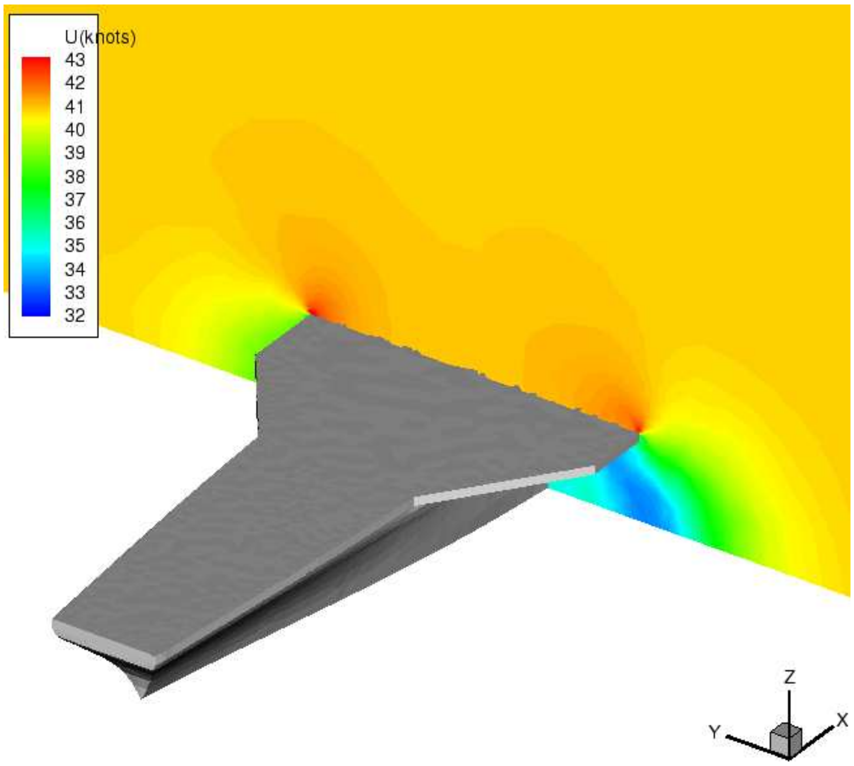

Figure 6.16. Total velocity contours for a slice for CVN-75 run (0° yaw, $U = 40$ knots)

6.1.2 Aircraft Carrier (CVN-75)

The CVN-75 is one of most modern and powerful ships in the US Navy. The purpose of this run was solely to demonstrate PUMA's capability in dealing with flows over such complex and practical geometries with equal ease. While there are no experimental data available for this geometry, visualization of the flow solution obtained showed it to be very realistic.

Two inviscid cases were run for the aircraft carrier geometry (CVN-75), one for 0° yaw and the other for 30° yaw. The grid contained 478506 cells and 974150 faces, and the run consumed about 1.1 GB of memory. The velocity contour for a section cutting the aircraft carrier vertically is shown in figure 6.16. The run took around 8 hours on a 8-processor Pentium II-266 cluster (245 seconds/timestep).

6.1.3 Landing Helicopter Aide (LHA)

This case was run to study a specific spot on the LHA where a lot of problems have been noticed in the event of a helicopter landing. This spot was to the rear-left of the island

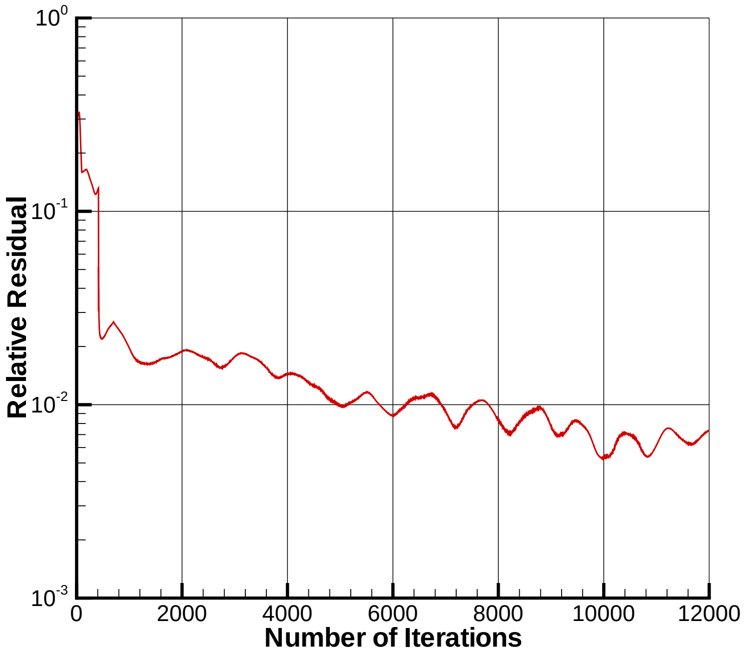

Figure 6.17. Convergence history for the LHA run ($M_\infty = 0.0375$)

on the LHA (e.g., U.S.S. Saipan), where vortex shedding and the wake due to the presence of the island were possibly causing flow complications. The flow conditions were 25 knots (12.7 m/s), i.e. $M_\infty = 0.0375$ at 5° yaw. An inviscid grid was generated which was highly clustered in this region of interest around the island. The final grid consisted of 1216709 cells and 2460303 faces and the run consumed 2.3 GB of memory. The convergence history for the run is shown in figure 6.17. The initial 2000 timesteps were traversed using SSOR and the remaining 8000 using the less expensive 2-stage Jameson-style Runge-Kutta scheme. The entire run took 39 hours on 32 processors of COCOA.

The U/U_∞ and C_p contours for LHA surface are shown in figures 6.18 and 6.19, respectively. Figure 6.20 shows the horizontal plane surveys on the LHA ranging from 5 ft above the deck to 30 ft above the deck, and depicting the C_p contours overlaid with the streamlines. The figure clearly shows the presence of a vortex immediately behind the island.

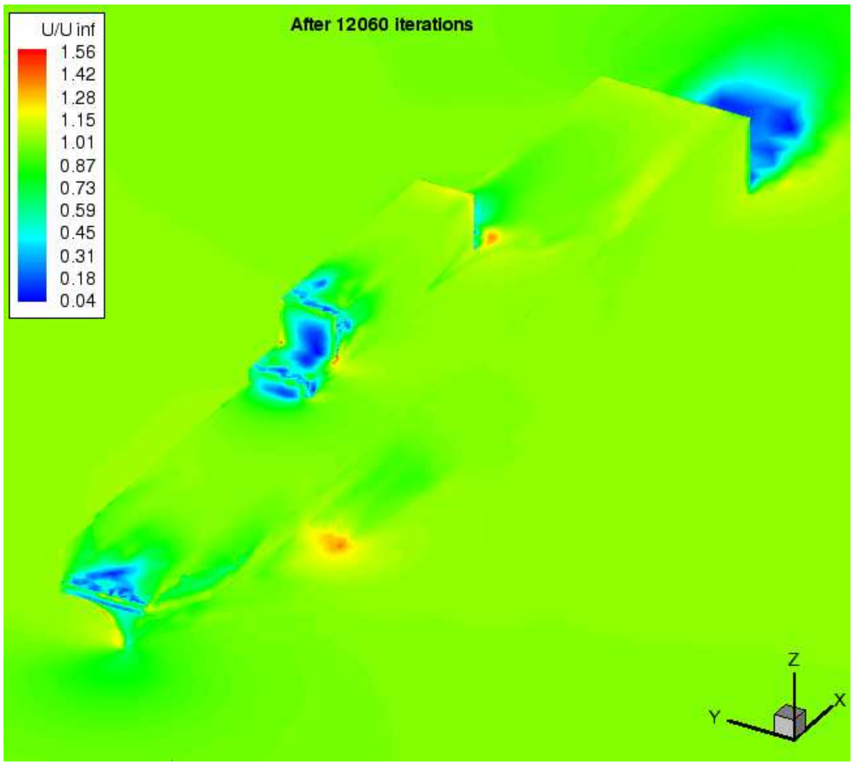

Figure 6.18. Surface U/U_∞ contours for LHA geometry ($\beta = 5°$, $U_\infty = 25$ knots)

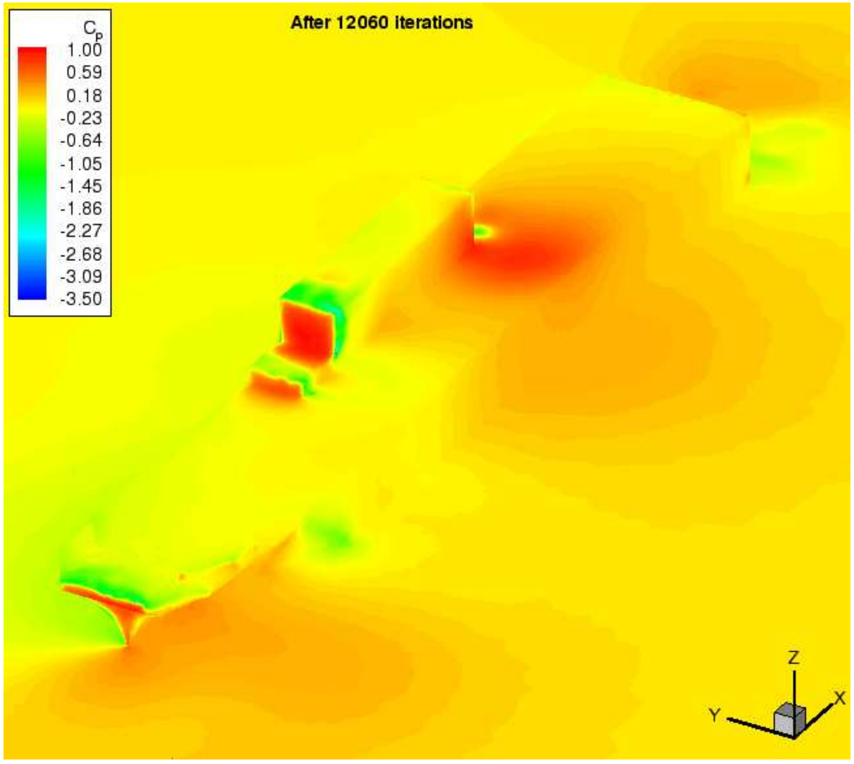

Figure 6.19. Surface C_p contours for LHA geometry ($\beta = 5°$, $U_\infty = 25$ knots)

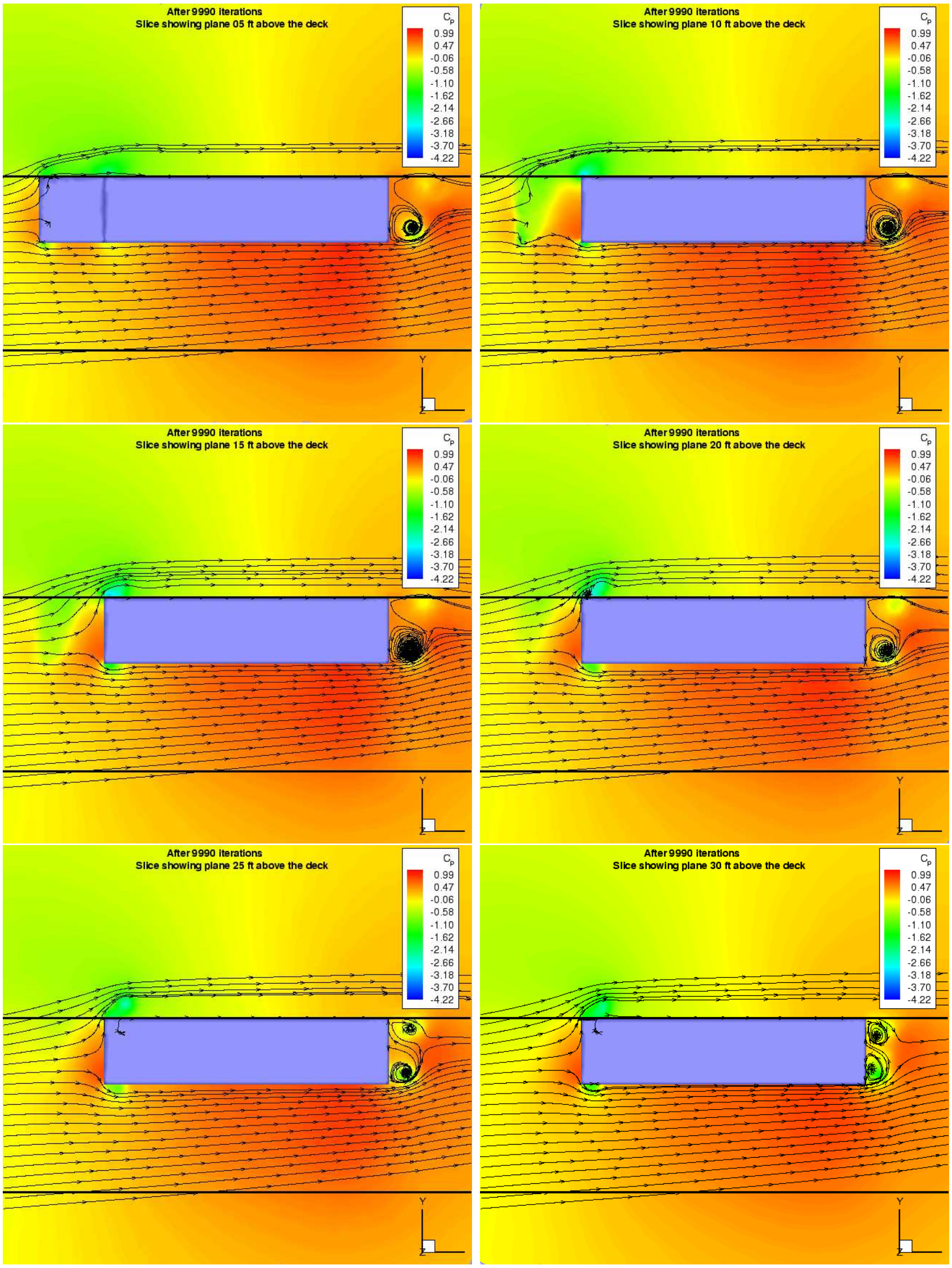

Figure 6.20. Horizontal Plane Surveys for LHA solution ($\beta = 5°$, $U_\infty = 25$ knots)

6.2 Various Helicopter Fuselage Configurations

6.2.1 ROtor Body INteraction Fuselage (ROBIN)

The *ROtor Body INteraction fuselage* (ROBIN) is a generic rotorcraft fuselage shape that has been extensively tested in wind tunnels [43]. It's shape is mathematically derived from a set of super-ellipse equations, hence it is easy to reproduce and grid. A large amount of experimental data exists for the ROBIN shape. The ROBIN shape has also had several computational studies completed on it with a variety of grids and flow solvers.

Since the given ROBIN fuselage geometry is symmetrical about one of its planes, only one-half of it has been modeled utilizing the symmetry plane. This reduces the number of cells in the grid to half, and hence the computational time for the flow solution. The final inviscid grid consisted of 260858 cells and 532492 faces, and the run consumed around 550 MB of memory. Three different cases were run with flow parameters of $U_\infty = 58$ m/s $= 114$ knots ($M_\infty = 0.18$) and angle of attack of $0°$, $-5°$ and $-10°$ (i.e., nose down), respectively. The convergence history for the 8 processor run for $-5°$ case is shown in figure 6.21, and the surface C_p and Mach contours (overlaid with streamlines) are shown in figures 6.22 and 6.23. Figure 6.24 shows the U/U_∞ and C_p contours for all the three cases.

Comparison with various experimental data is available for this case, which can be found in the Master's thesis of Steven Schweitzer [32]. The static pressures from PUMA compare extremely well with experimental data except for areas around the tail boom. This result was expected given that the tail boom area is an area of highly unsteady flow. PUMA's solutions also compared favorably with solutions from CFL3D presented in Chaffin et al [10]. The small discrepancies between PUMA and CFL3D can be accounted for by the fact that the CFL3D solution was viscous while the PUMA solution was inviscid. Using the RANS algorithms in PUMA will allow a viscous solution to be obtained, but it is not likely to fully capture the unsteady flow over the tail boom area.

6.2.2 Boeing Generic Helicopter Fuselage

Dr. Hormoz Tadghighi of Boeing Mesa has provided Penn State University with their generic helicopter shape. Boeing Mesa has collected a large amount of experimental data on this generic helicopter shape. This Boeing generic helicopter shape is not as streamlined as the ROBIN fuselage, and will offer a chance to study a more realistic helicopter fuselage and

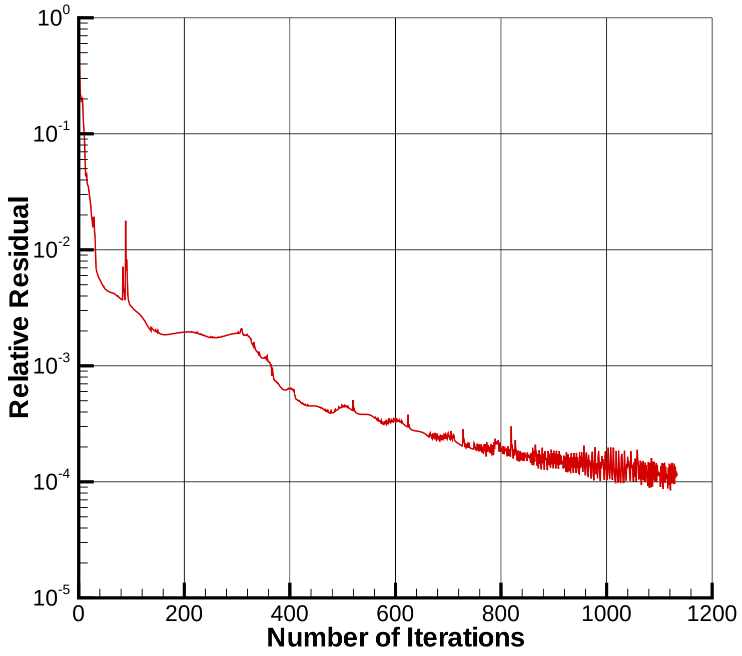

Figure 6.21. Convergence history for ROBIN inviscid run for $\alpha = -5°$ case

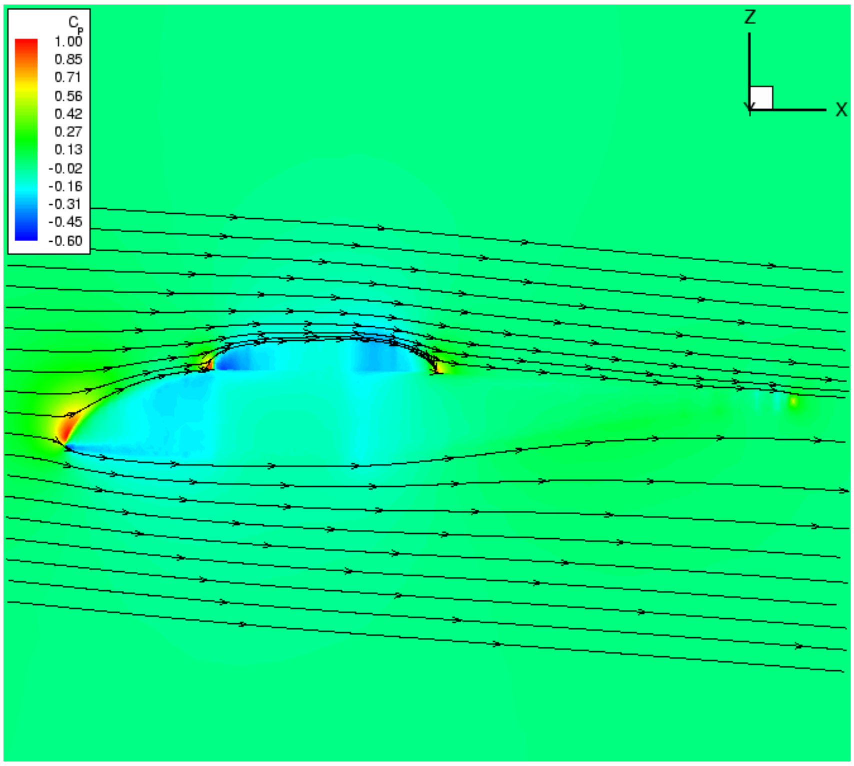

Figure 6.22. Surface C_p contours for ROBIN fuselage overlaid with streamlines ($\alpha = -5°$)

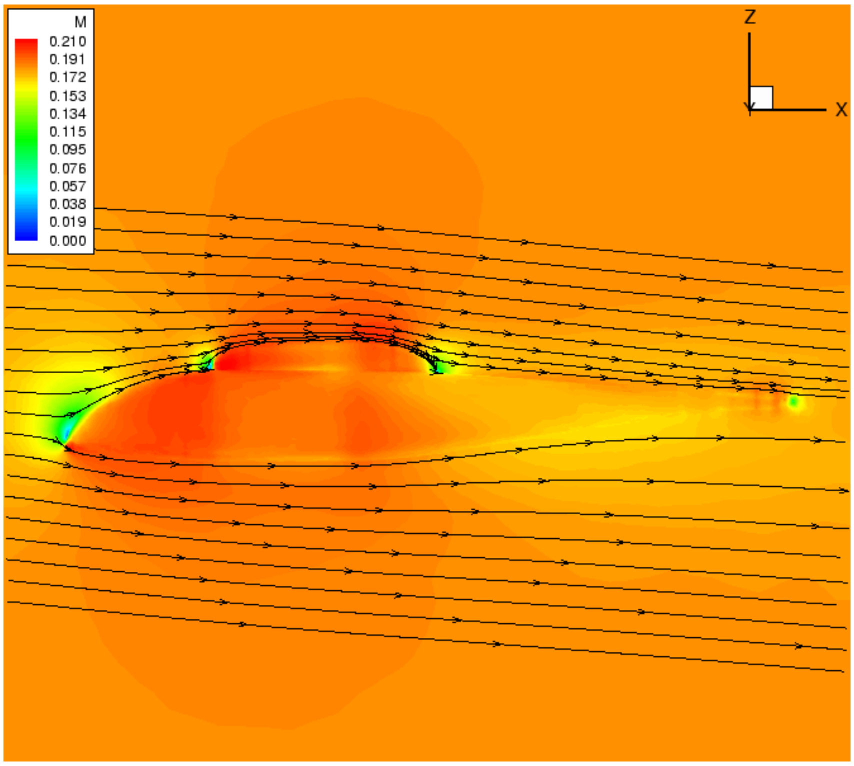

Figure 6.23. Surface Mach contours for ROBIN fuselage overlaid with streamlines ($\alpha = -5°$)

compare the results with experimental data. This generic shape has been proposed to be used for all further analysis of PUMA and all future modifications.

Like the ROBIN geometry, the Boeing generic helicopter fuselage geometry too was symmetrical and was thus modeled using a symmetry plane. The final inviscid grid consisted of 380089 cells and 769240 faces, and the run consumed around 810 MB of memory. The flow conditions were $U_\infty = 58$ m/s $= 114$ knots ($M_\infty = 0.18$) with no angle of attack ($\alpha = 0°$). The convergence history for the 6 processor run is shown in figure 6.25. Surface C_p contours for the 0° pitch case are shown in figures 6.26 and 6.27, while the surface Mach contours are shown in figures 6.29 and 6.28. These only offer a cursory look into the flow solution for this geometry, as more work is yet to be done, including comparing the above solution with the available experimental data.

6.2.3 Apache AH-64 Helicopter Fuselage

The flow over an Apache AH-64 was also simulated in order to study more complex helicopter fuselage geometries. For the Apache helicopter fuselage geometry, the inviscid grid consisted

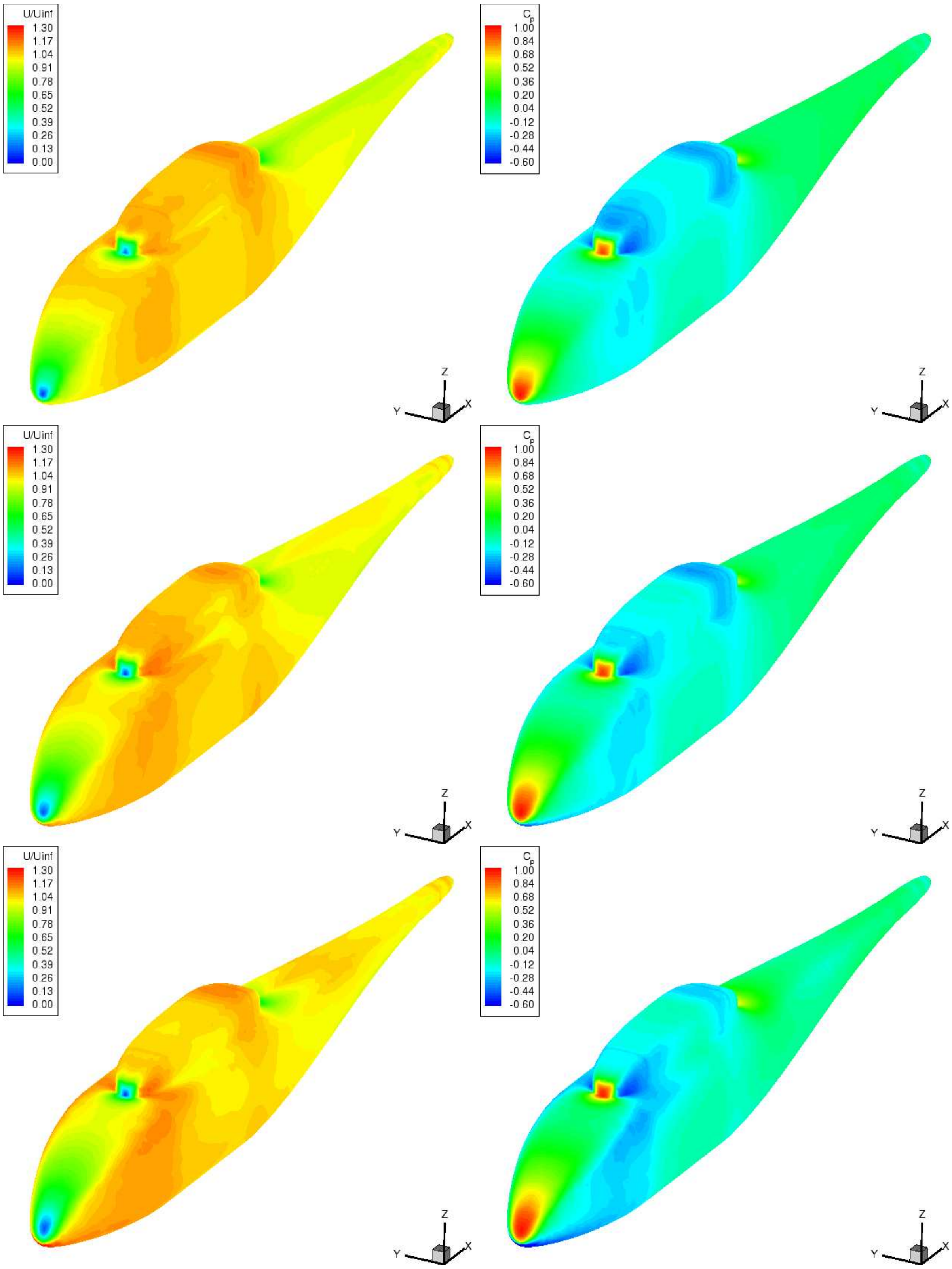

Figure 6.24. Surface U/U_∞ and C_p contours for ROBIN fuselage with $\alpha = 0°$, $-5°$ and $-10°$ (i.e., nose down), respectively

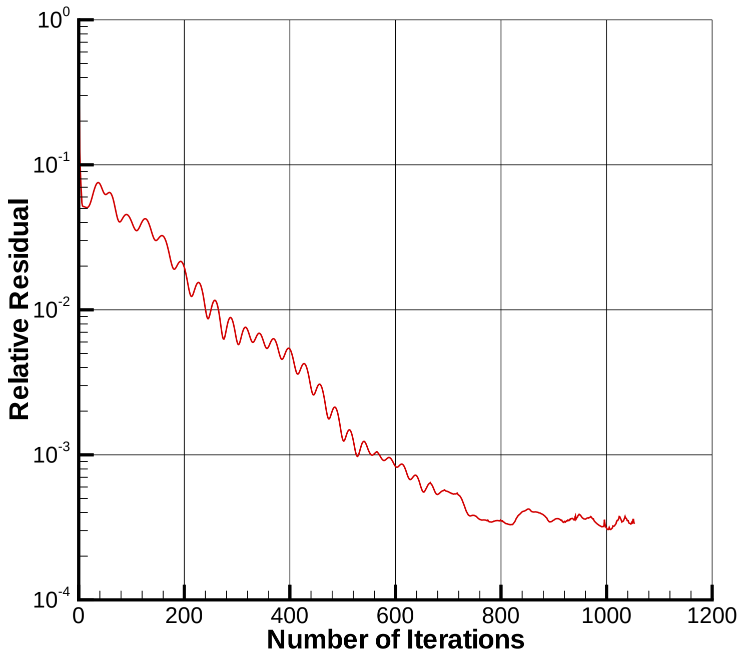

Figure 6.25. Convergence history for generic helicopter fuselage run for $\alpha = 0°$ case

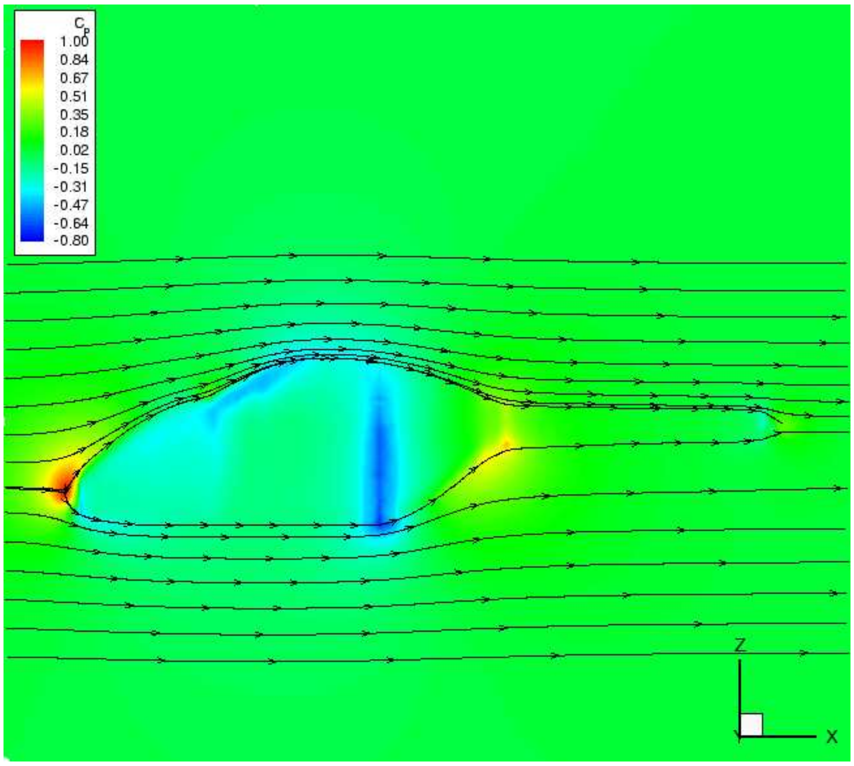

Figure 6.26. Surface C_p contours for generic helicopter fuselage overlaid with streamlines ($\alpha = 0°$)

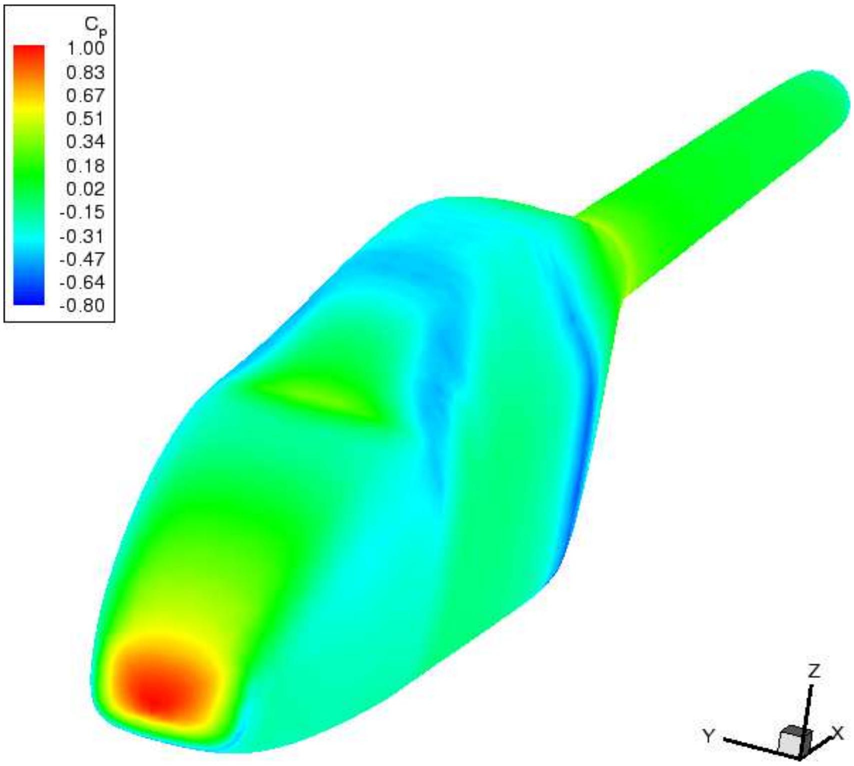

Figure 6.27. Surface C_p contours for generic helicopter fuselage ($\alpha = 0°$)

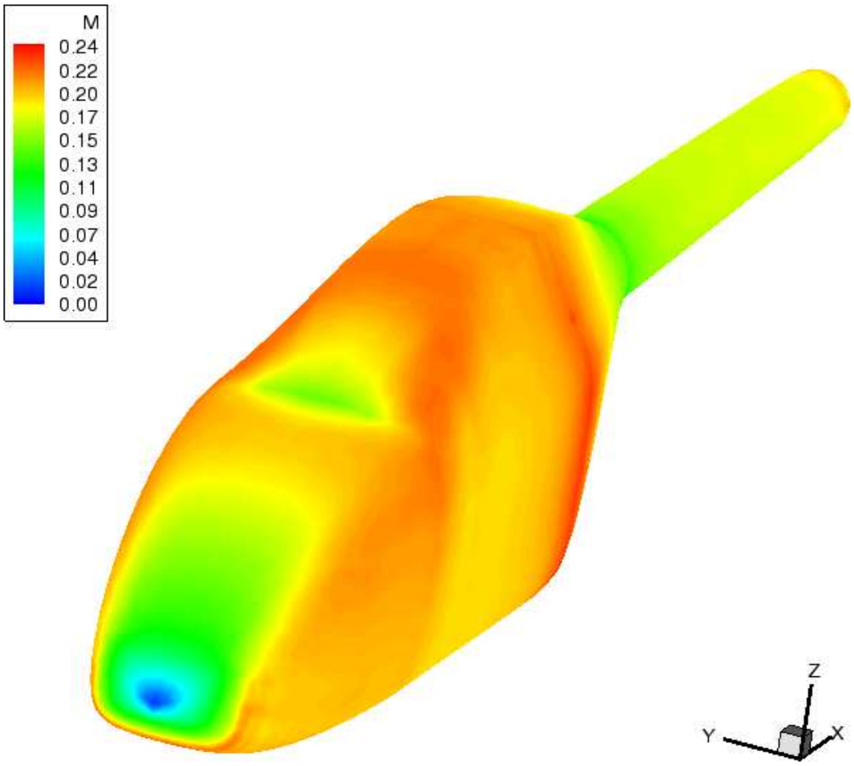

Figure 6.28. Surface Mach contours for generic helicopter fuselage ($\alpha = 0°$)

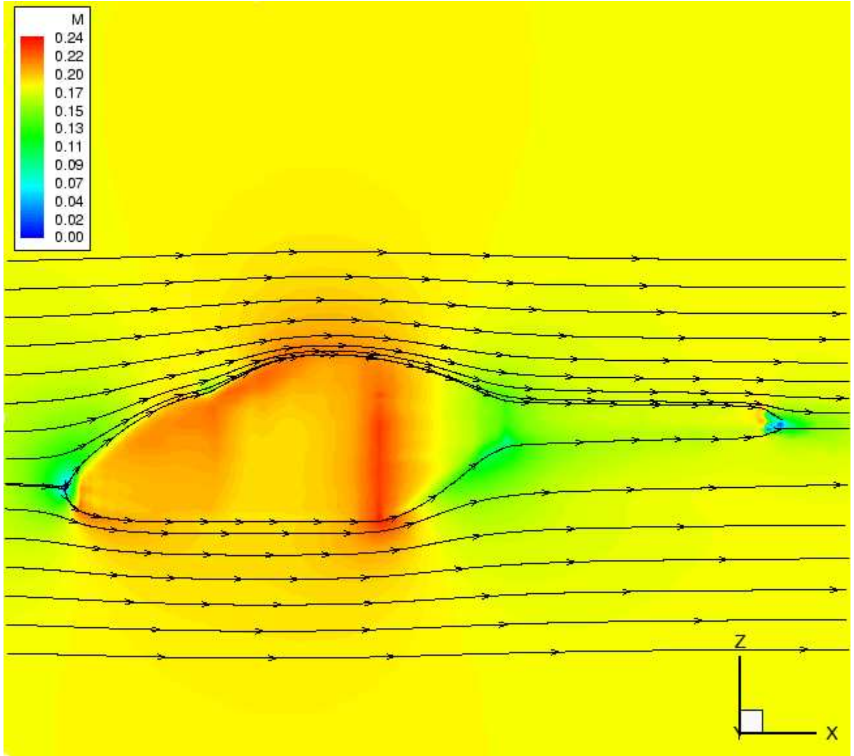

Figure 6.29. Surface Mach contours for generic helicopter fuselage overlaid with streamlines ($\alpha = 0°$)

of 555772 cells and 1125596 faces and the run consumed 1.9 GB of memory. The convergence history for the 8 processor run is shown in figure 6.30 and the surface Mach and Pressure contours are shown in figures 6.31 and 6.32 respectively. The flow conditions were $U_\infty = 114$ knots which corresponds to a Mach number of 0.181. Another contour plot of the Mach number across the mid-section of the helicopter (plane $Y = 0$) is shown in figure 6.33. Although the solution obtained looks very realistic, no experimental data was available to corroborate with.

6.3 Axisymmetric Bluff Bodies

6.3.1 Viscous 3D Cylinder ($Re = 1000$)

This is the first fully viscous and unsteady flow solution attempted using PUMA.

The flow domain for this case was box shaped and is shown in figure 6.34. The domain extended 10 diameters in the front of the cylinder, 20 diameters vertically above and below the cylinder, and 40 diameters in rear of the cylinder (region of interest). The length of

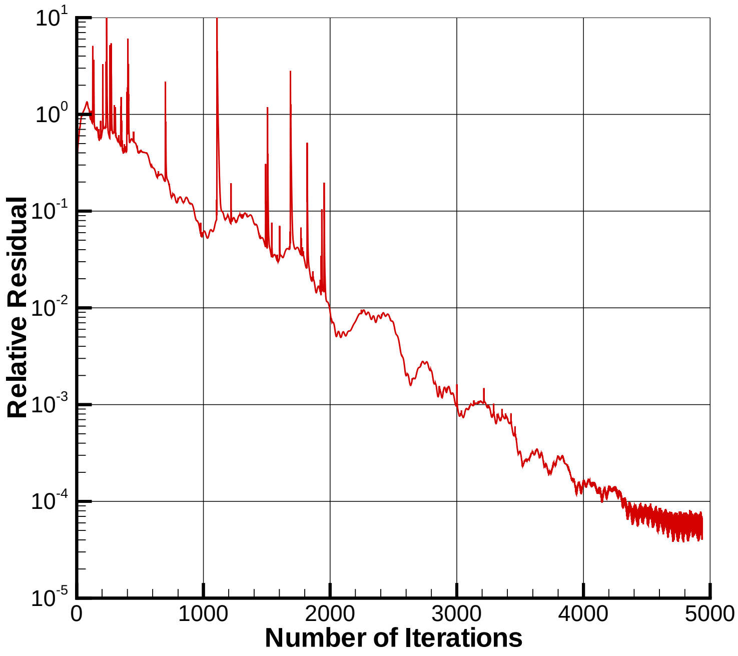

Figure 6.30. Convergence history for the Apache Helicopter run

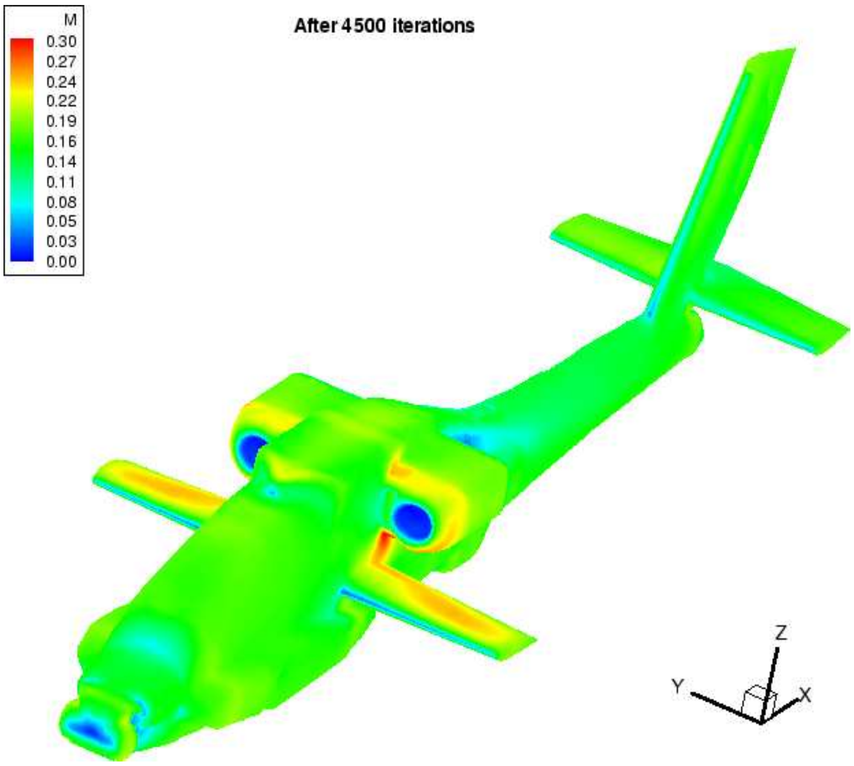

Figure 6.31. Surface Mach contours for Apache helicopter fuselage ($U_\infty = 114$ knots)

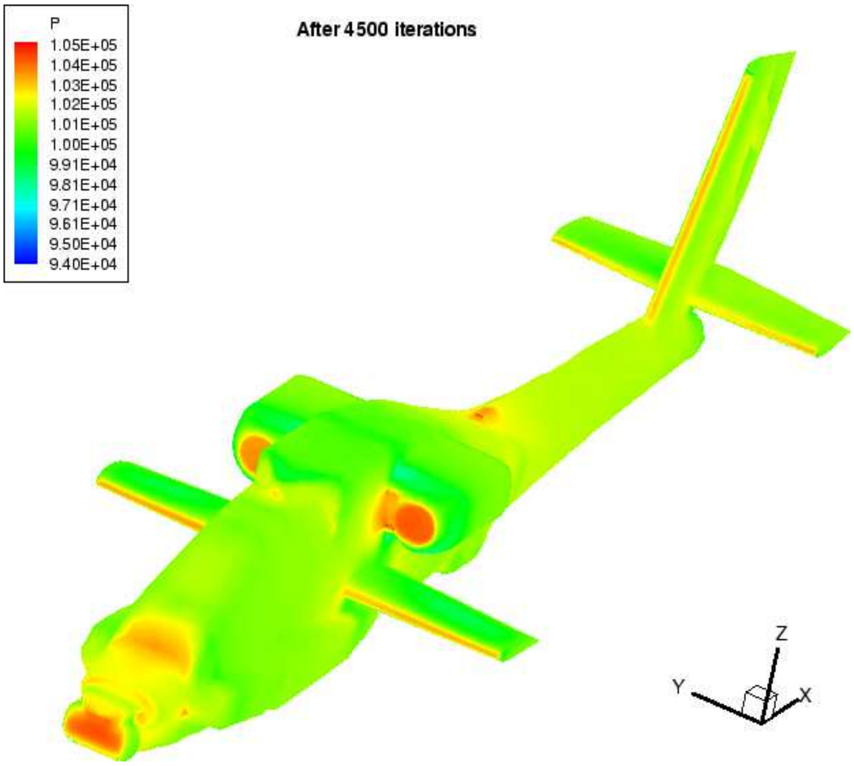

Figure 6.32. Surface Pressure contours for Apache helicopter fuselage ($U_\infty = 114$ knots)

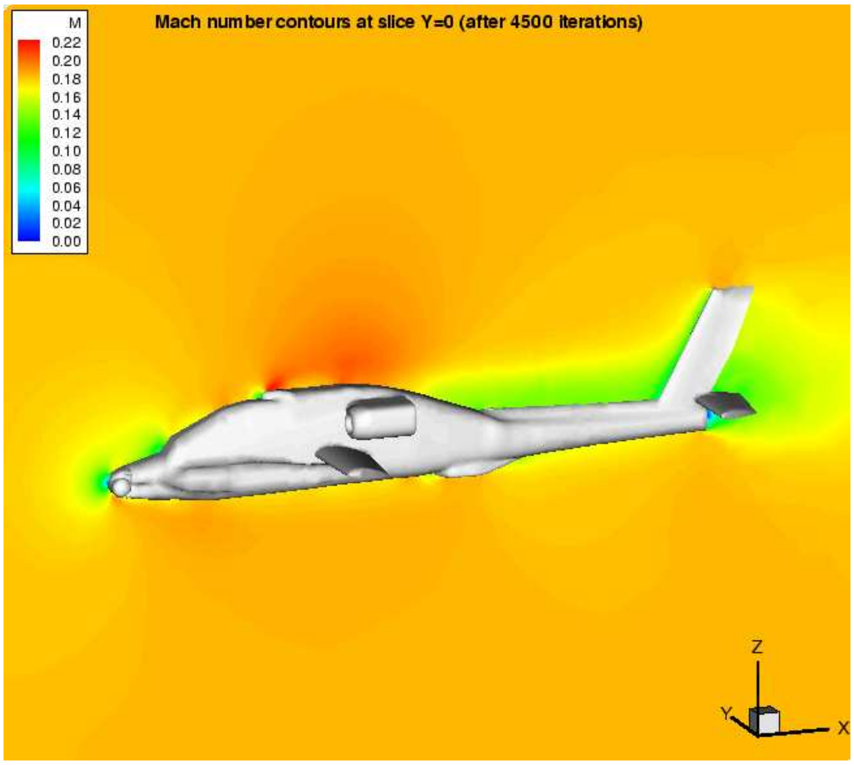

Figure 6.33. Mach contours for flow around Apache helicopter fuselage at slice $Y = 0$ ($U_\infty = 114$ knots)

the cylinder was 10 diameters and the domain extended 5 diameters from either side of the end-plates of the cylinder. Further, the grid was clustered in the expected region of vortex shedding downstream of the cylinder. The boundary layer thickness was calculated from the laminar boundary layer theory [44] with 10 layers of prismatic cells placed within the boundary layer. The final viscous grid case consisted of 806668 cells and 1620576 faces. The steady state run consumed 2.4 GB of memory on 32 nodes of COCOA (using SSOR), whereas the time-accurate run consumed 1.1 GB of memory on 32 nodes of COCOA (using 4-stage Jameson-style Runge-Kutta). The flow conditions were $U_\infty = 41$ knots (corresponding to $M_\infty = 0.061$) and $Re = 1000$. The convergence history for the run is shown in figure 6.35 and the Mach contours (overlaid with streamlines) along the mid-section of the 3D cylinder is shown in figure 6.36. The case was run using SSOR to get a somewhat steady state solution (with residual reduction of about 2 orders in magnitude), and then switched over to 2-stage time-accurate Runge-Kutta scheme. The time-accurate run went on for $20,000$ timesteps (corresponding to physical time of $t = 0.014$ seconds). The timestep for this run was $\Delta t = 7.14064 \times 10^{-7}$ sec $= 7.436 \times 10^{-6}$ diameter units. The times used henceforth, are mentioned in diameter units, i.e. the time required by the flow to convect one diameter in the flowfield. Since the computational cost for running the time-accurate case over several periods of vortex shedding was anticipated to be extremely large for such a fine grid, the run was abandoned at this point. The initial results (figure 6.36) compared very well with some qualitative sketches of the vortex shedding available.

6.3.2 Viscous Sphere ($Re = 1000$)

A complete unsteady separated flow solution over a sphere in uniform flow has been simulated using PUMA. Since flow over a sphere is considered as a prototype example from the class of flows past axisymmetric bluff bodies, and because extensive experimental data is available for it, this was considered as an ideal case to use for the validation of PUMA.

The flow domain for this case was box shaped and is shown in figure 6.37. The grid extended for 5 diameters in the front, the top and the bottom of the sphere, and 20 diameters in rear of it. The blockage ratio corresponding to these dimensions is 0.65%. Since the boundary layer on the sphere does not become turbulent until a Reynolds number of the order of $300,000$ is reached, estimates of the boundary layer thickness were derived from the laminar boundary layer solution for axisymmetric bodies [44]. As for the cylinder grid, 10 layer of prismatic cells were placed within the boundary layer. The final viscous grid obtained consisted of 306596

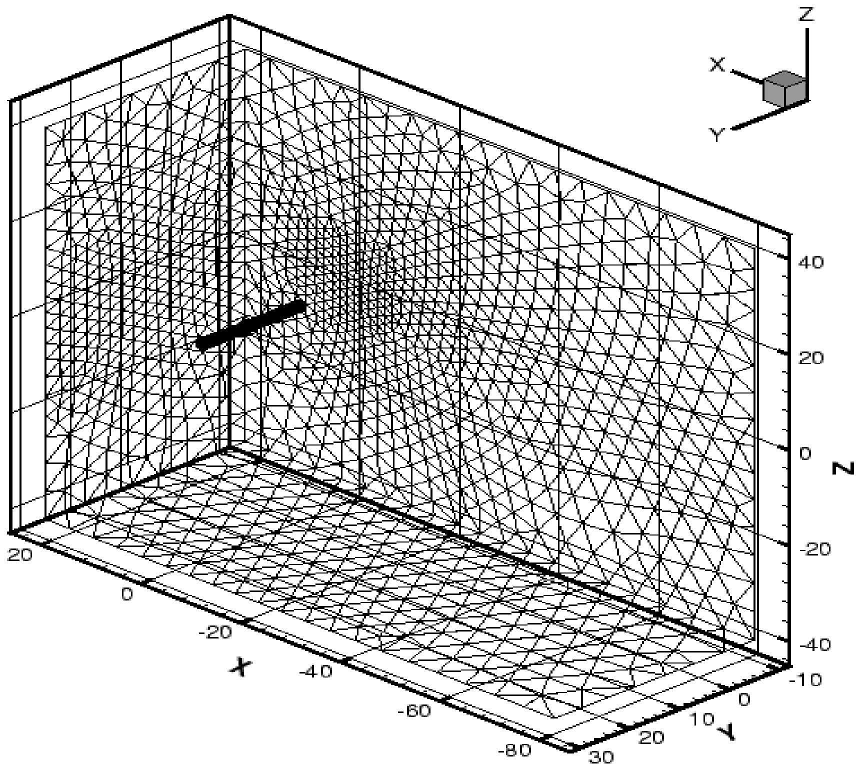

Figure 6.34. Domain for the viscous cylinder grid

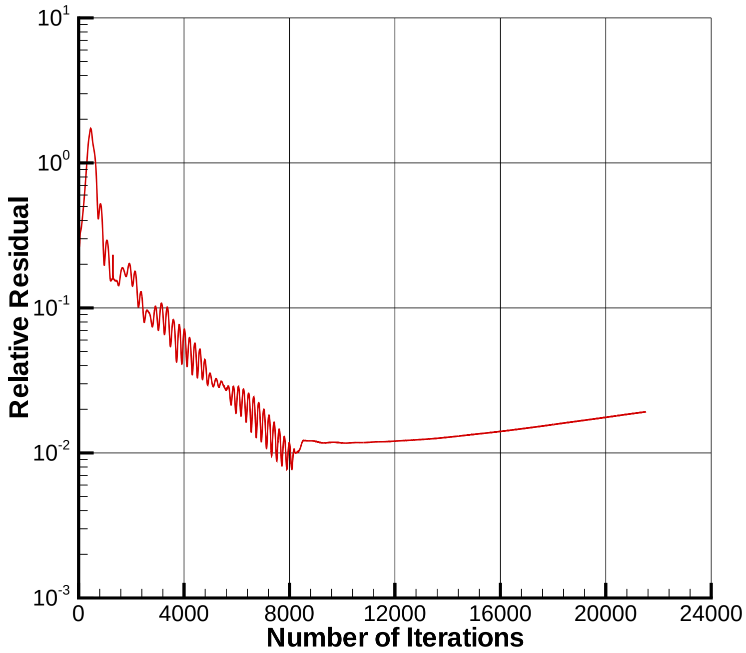

Figure 6.35. Convergence history for viscous cylinder run ($M_\infty = 0.061$, $Re = 1000$)

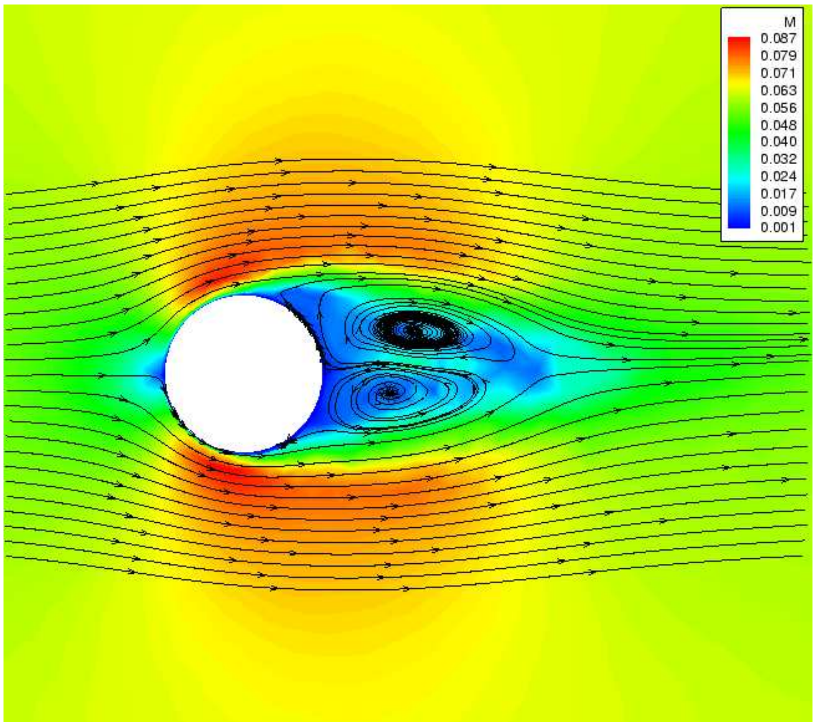

Figure 6.36. Mach contours with streamlines at mid-section for viscous cylinder run ($M_\infty = 0.061$, $Re = 1000$)

cells and 617665 faces, and the run consumed 600 MB of memory on 32 nodes of COCOA (running 2-stage Runge-Kutta). The flow conditions were $M_\infty = 0.2$ ($U_\infty = 68$ m/s) and $Re = 1000$ based on the diameter. As for the cylinder run, this case too was initially run using SSOR to get a somewhat steady state solution (with residual reduction of about 2 orders in magnitude), and then switched over to a 2-stage time-accurate Runge-Kutta scheme. The time-accurate run commenced after 14,000 iterations, but all data before iteration 50,000 was discarded to avoid the transients. Therefore, $t = 0$ corresponds to iteration 50,000. The convergence history for the run is shown in figure 6.38, and the C_p and Mach contours along the planes $X = 0$ and $Z = 0$ are shown in figures 6.39 and 6.40. For the time-accurate run, each timestep corresponds to $\Delta t = 6.45911 \times 10^{-7}$ sec $= 2.197 \times 10^{-5}$ diameter units[2] . From the available experimental data, for $Re = 1000$, the Strouhal number is expected to be very close to 0.2, thus giving a vortex shedding period of approximately $T = 0.15$ sec $= 5$ units, for this simulation.

In spite of our running for just two cycles of the vortex shedding period ($t = 10 = 2T$), the

[2]time required by the flow to convect one diameter in the flowfield

time-averaged C_p results (figure 6.41) compared quite well with the experimental data from Modi and Akutsu [20]. The correlation between the surface C_p values was almost perfect in the range $0° \leq \theta < 110°$, but seemed to vary by approximately a constant amount for the wake region ($110° \leq \theta \leq 180°$). Axial velocity data along the center line trailing the center of the sphere are compared to both the DNS simulation by Tomboulides [27], and the experimental data from Wu and Faeth [25] in figures 6.45 and 6.46. The comparison is not very impressive, but the nature of the results look very similar. The difference can be attributed to the fact that the simulation described here ran for just two cycles of vortex shedding (and 10 time units), unlike the DNS data from Tomboulides [27] which was averaged over more than 20 shedding cycles. Due to the extremely small number of samples, our time-averaged results were very sensitive to the starting and ending points of the sampled data. Figure 6.42 shows the time history plot of the non-dimensional axial velocity (v/U_∞) of several points in the wake of the sphere. Axial velocity profiles for different planes in the wake of the sphere are also shown in figure 6.43. The nature of the flow again matches with the experiments here, as reverse flow is noticed only until about 1.8 diameters downstream of the sphere (measuring from the center of the sphere). Figure 6.44 shows the variation of the lift (C_l), drag (C_d) and side force (C_m) coefficients over time of the simulation.

Figure 6.47 shows the iso-surface at which axial velocity is 90% of U_∞ for the instant $t = 9.34$. This gives the approximate shape of the wake. Figure 6.48 attempts to show the three-dimensional nature of the wake region for the same instant. Figure 6.49 depicts the flow along the slice $X = 0$ at different instants of time. Although the streamlines shown do not signify the actual unsteady streamlines, they do give the general idea about the tendency of the flow. Streaklines are what are really needed to be shown, but they are very tedious and difficult to obtain. The figure clearly shows the alternating nature of the vortex shedding for $t = 0$ and $t = 2.75$. However, the same observation cannot be made for the latter two figures, maybe because the vortex shedding has not yet stabilized and thus cannot be considered periodic. Figure 6.50 gives the far-field view for the same flow field at $t = 9.34$.

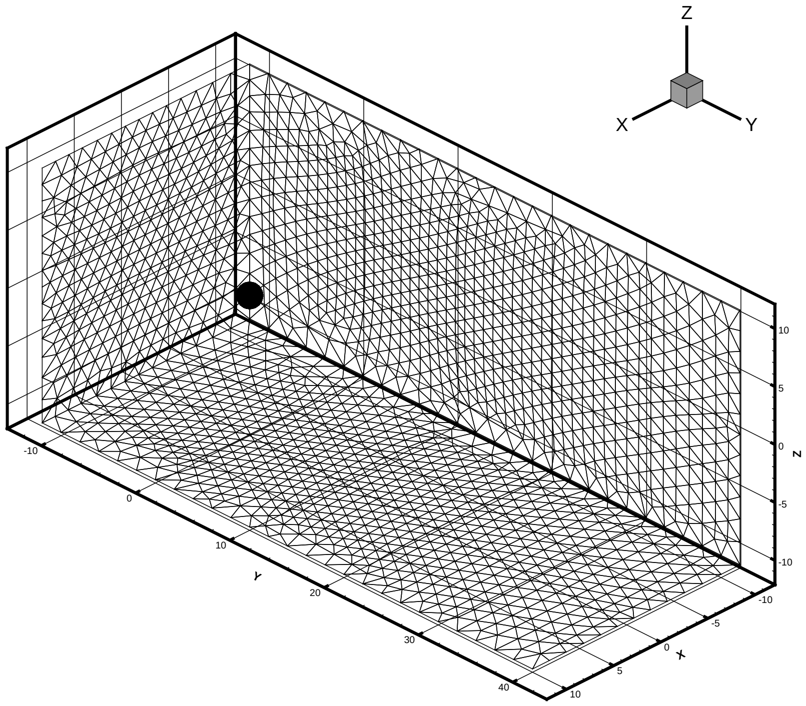

Figure 6.37. Domain for the viscous sphere grid

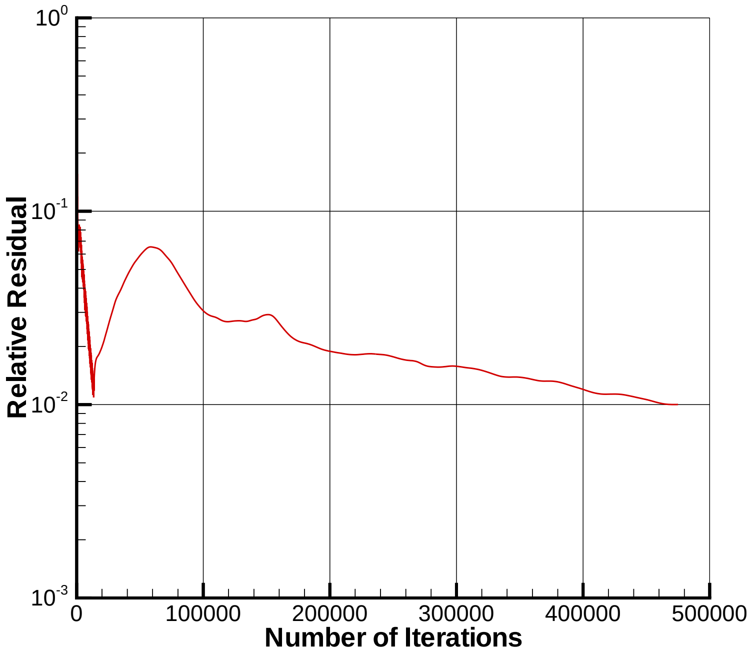

Figure 6.38. Convergence history for viscous sphere run ($M_\infty = 0.2$, $Re = 1000$)

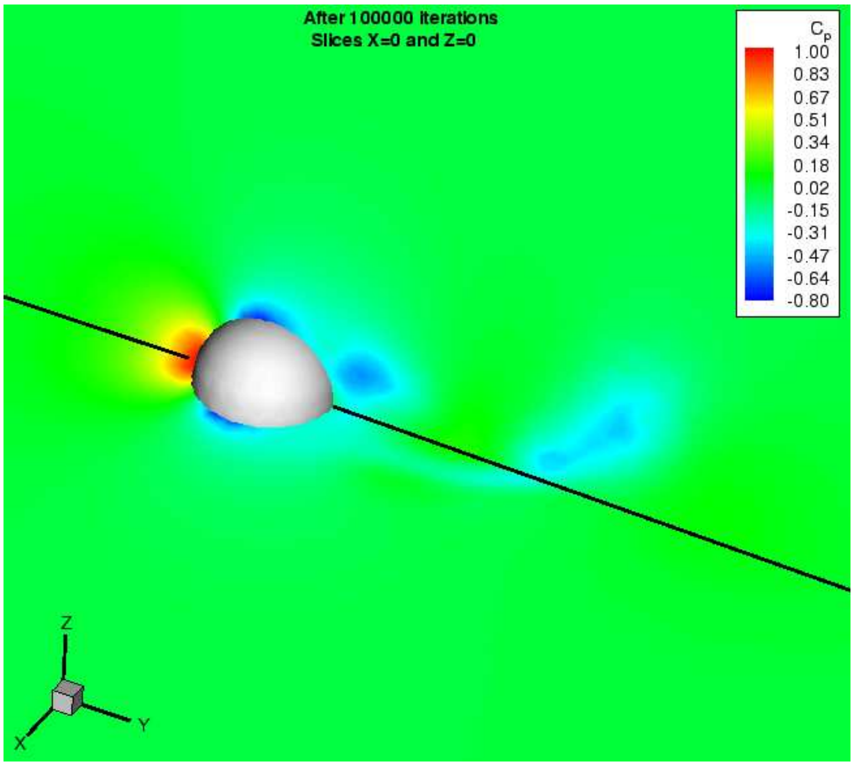

Figure 6.39. C_p contours for viscous sphere run ($M_\infty = 0.2$, $Re = 1000$)

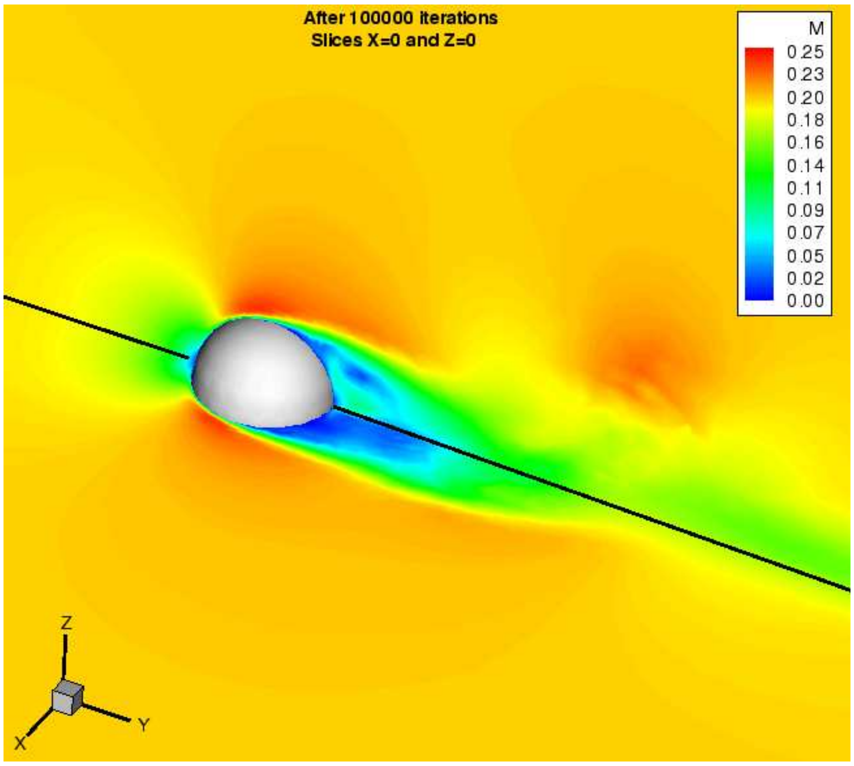

Figure 6.40. Mach contours for viscous sphere run ($M_\infty = 0.2$, $Re = 1000$)

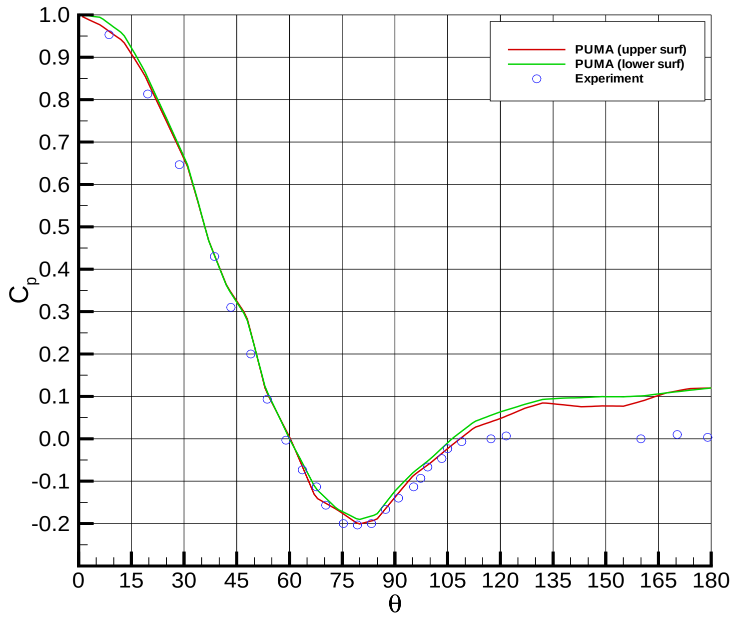

Figure 6.41. Time averaged plot for C_p vs θ for upper surface compared with experiments [20], for viscous sphere run ($M_\infty = 0.2$, $Re = 1000$) [Note: Here, C_p is defined as $(P_\theta - P_{60^\circ})/(P_{0^\circ} - P_{60^\circ})$]

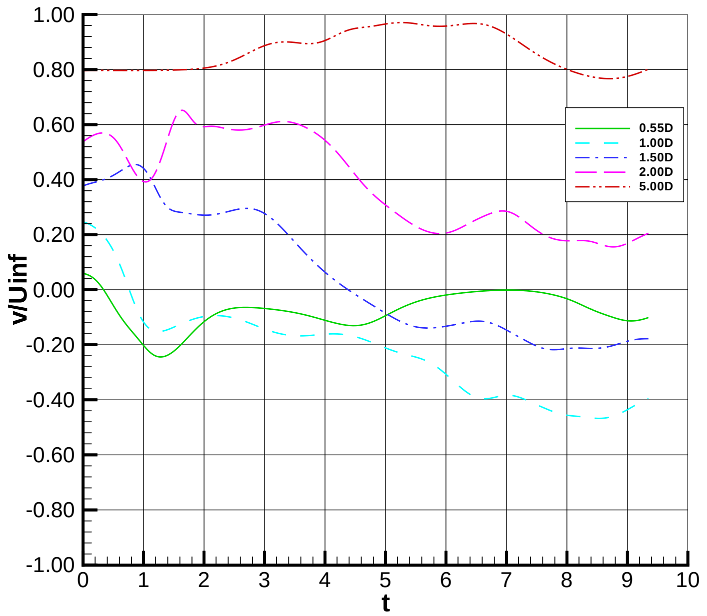

Figure 6.42. Time history depicting axial velocity for points at fixed distances behind the center of the sphere along the y-axis

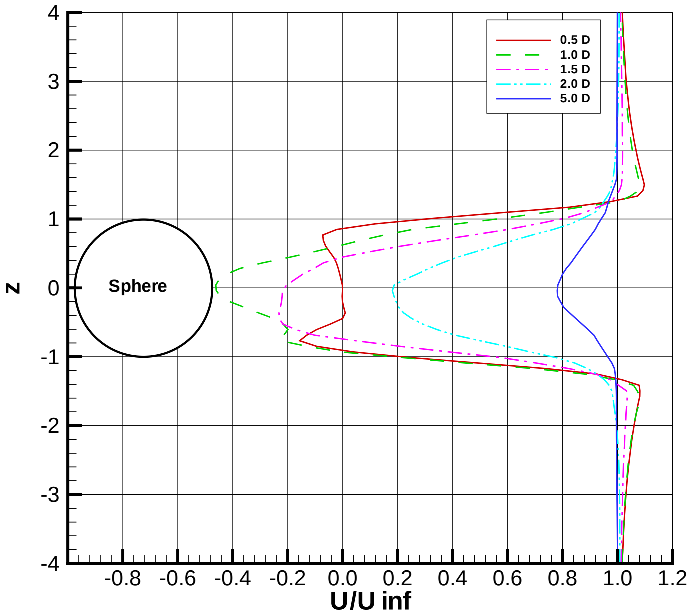

Figure 6.43. Instantaneous velocity profile ($t = 8.79$) showing wake region for viscous sphere run ($M_\infty = 0.2$, $Re = 1000$)

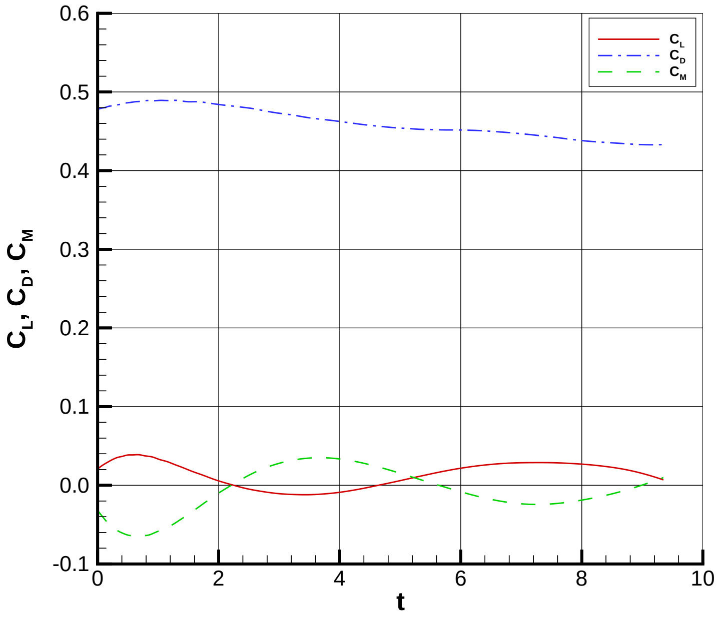

Figure 6.44. Variation of total C_l, C_d and C_m coefficients with time for viscous sphere run ($M_\infty = 0.2$, $Re = 1000$)

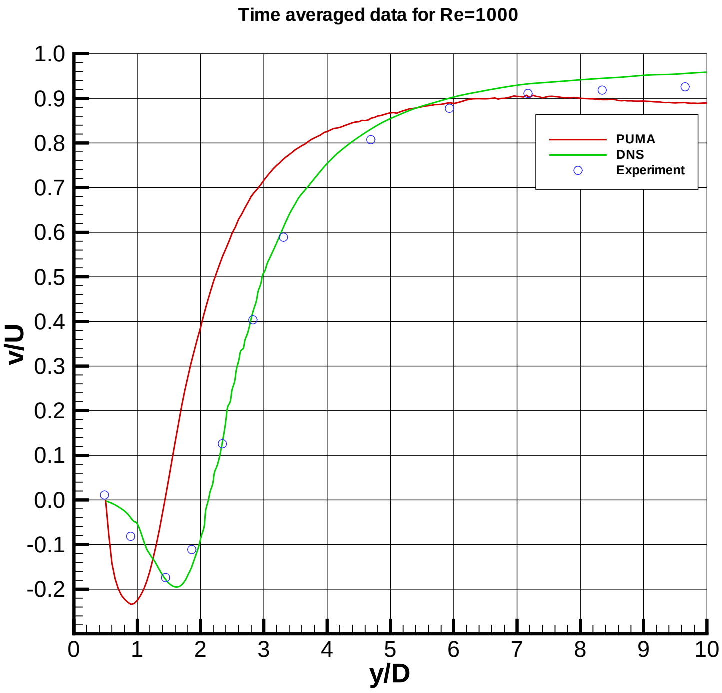

Figure 6.45. Time averaged axial velocity plot (v/U_∞) for viscous sphere run ($M_\infty = 0.2$, $Re = 1000$)

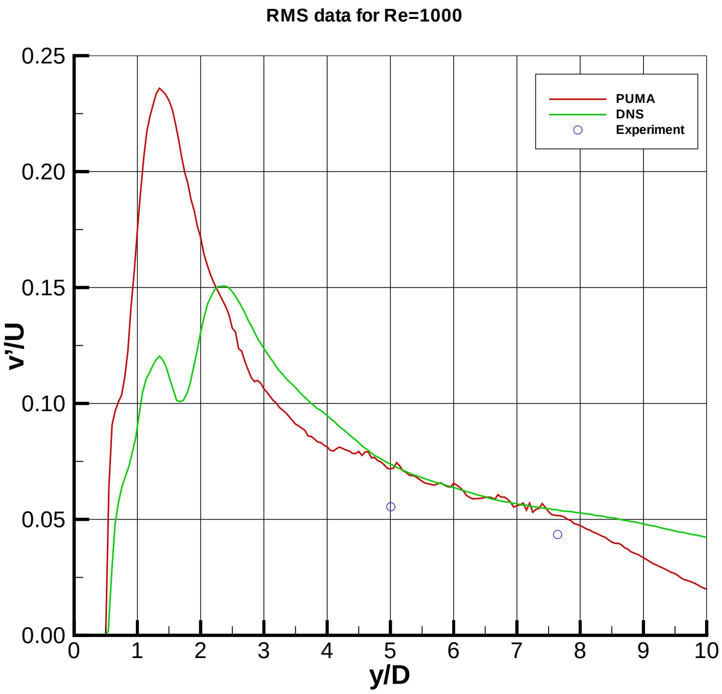

Figure 6.46. Time averaged RMS velocity plot (v'/U_∞) for viscous sphere run ($M_\infty = 0.2$, $Re = 1000$)

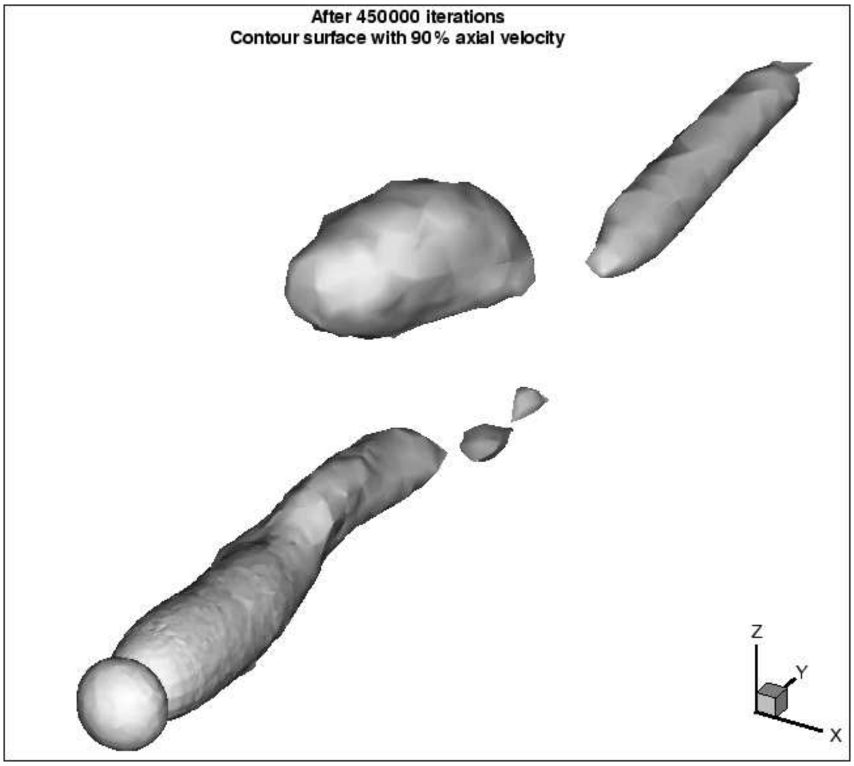

Figure 6.47. Iso-surface at which axial velocity is 90% of U_∞ ($t = 9.34$)

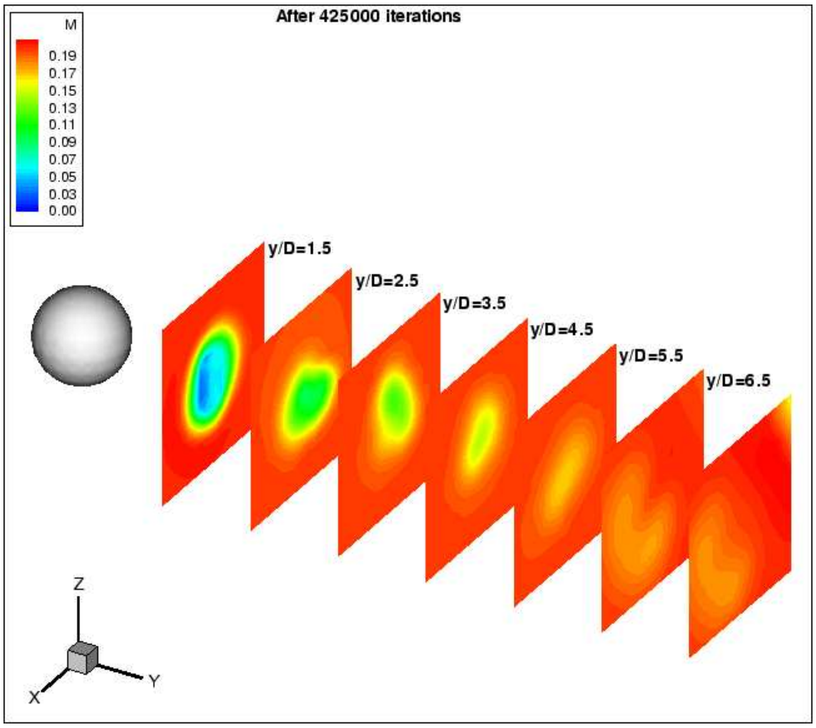

Figure 6.48. Mach contour slices every 2 diameters in the wake region for the viscous sphere run ($M_\infty = 0.2$, $Re = 1000$)

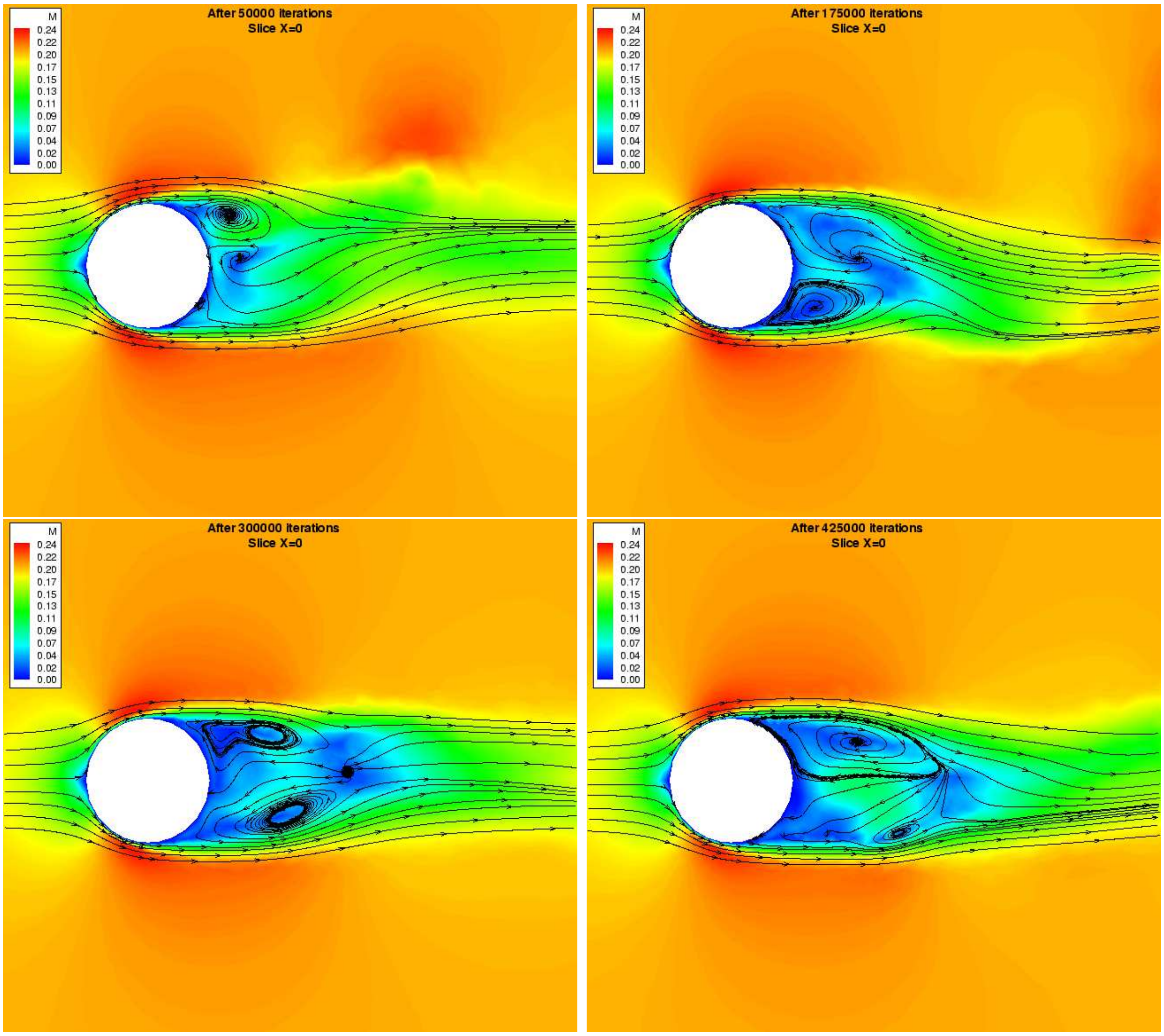

Figure 6.49. Mach contours and streamlines (at $t = 0.0, 2.75, 5.50, 8.25$) for viscous sphere run ($M_\infty = 0.2$, $Re = 1000$)

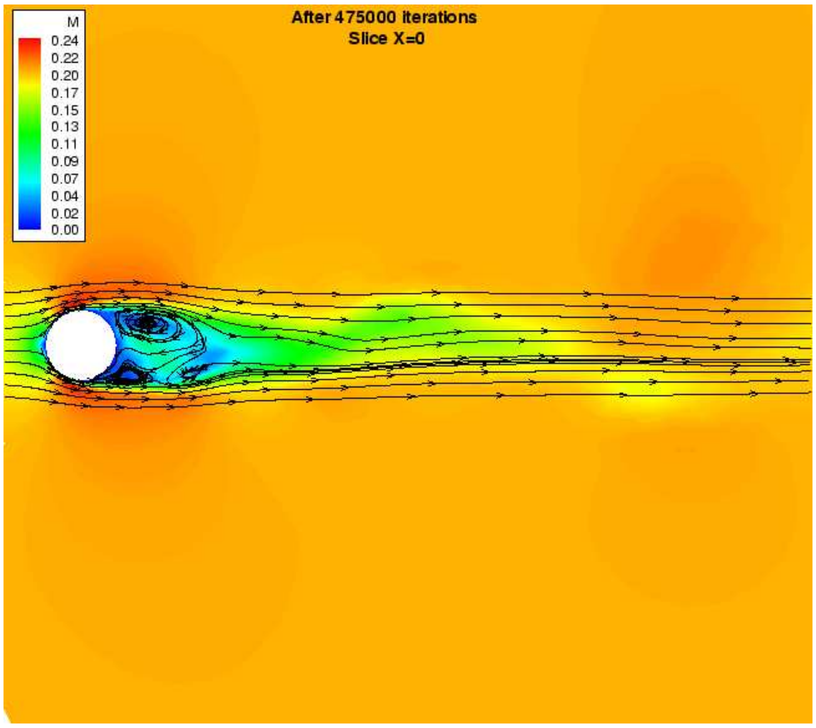

Figure 6.50. Mach contours and streamlines (at $t = 9.34$) for viscous sphere run ($M_\infty = 0.2$, $Re = 1000$)

Chapter 7

Conclusions

A complete, fast and efficient unstructured grid based flow solution around several complex geometries has been demonstrated. The objective to achieve this at a very affordable cost using inexpensive departmental level supercomputing resources like COCOA, has been fulfilled. The several modifications made to PUMA, and the implementation of the "Live CFD-cam", have immensely enhanced the debugging and post-processing capabilities of our code. It has also served as an innovative and extremely useful tool for the communication and discussion of such complex problems among far-flung people on the internet, and is being looked upon as a means to implement *Computational Steering* of PUMA over the internet. Combining the power of rapid grid generation tools like GridTool and VGRID, along with the fast inexpensive Beowulf cluster, the flow solver PUMA, and the "Live CFD-cam", incredible turn-around times for several large problems involving complex geometries, right from the geometry description to the visualization of the post-processed solution, of under a day, have been achieved.

The flow solver PUMA has been verified with experimental data from inviscid and viscous solutions around the GSS geometry. Although the experimental data was just qualitative in nature, the excellent correlation between the two for such a complex flow was extremely encouraging. Both the inviscid and viscous solutions compared very well, with the viscous solution being better in the region of flow separation. Recent work by Steven Schweitzer [32] has also compared inviscid solutions from PUMA around the ROBIN body with experiments, and the outcome was very favorable. Several other large problems for flow around different ship configurations (LHA, aircraft carrier CVN-75) and helicopter fuselages (AH-64 Apache, Boeing generic fuselage) have been run, but lack of any experimental data has prevented the measure of PUMA's success for these cases. Viscous, unsteady runs for $Re = 1000$ around a finite cylinder and sphere have also been conducted. For the sphere case, in spite of our running for just two cycles of the vortex shedding period, the time-averaged C_p results compared quite well with the experimental data. The correlation between the surface C_p values was almost perfect in the range $0° \leq \theta < 110°$, but seemed to vary by approximately a constant amount for the wake region ($110° \leq \theta \leq 180°$). The code also was able to predict the trends in the wake axial velocity, although there were some discrepancies. This can be attributed to the fact that the simulation described here ran for just two cycles of vortex shedding (10 time units), which made the data statistically stiff, as compared to the DNS data which was averaged over more than 20 cycles. Based on the validation and the experience from the above runs, PUMA is seen to be a very good starting point for modeling such complex unsteady separated flows.

The idea of building COCOA was a huge success. COCOA was found to be extremely suitable for the current class of numerical simulations dealt with at Penn State. For most of our production codes including PUMA, COCOA proved to be almost twice as fast compared to the Penn State IBM-SP supercomputer, given the same number of processors, while being built at a fraction of the cost. COCOA was also found to have good scalability with most of the MPI applications used. Although Beowulf clusters have very high latency as compared to the conventional supercomputers, this did not really affect the applications, as most of the codes used contained only a few large messages to be communicated at every timestep. PUMA did not at first perform well on COCOA, but after modifications to pack the small messages into bigger messages, dramatic increase in performance was obtained. For those codes that have high communication to computation ratios, COCOA was not found to be a suitable platform because of the high latency. With several more enhancements planned for COCOA in the near future, like addition of several more nodes and increasing of the networking bandwidth by addition of more fast-ethernet cards to every node, the performance is expected to go up almost linearly.

Chapter 8

Future Work

8.1 k-exact Reconstruction

It has been shown that higher order solvers can be much more efficient than lower order solvers. A 4th order accurate method can be $50 - 100$ percent more efficient than a 2nd order method. Due to the nature of unstructured grids, unstructured flow solvers tend to be 2nd order accurate in both time and space. Accuracy in flow solvers that use unstructured grids can be greatly improved by implementing a local refinement scheme.

PUMA is currently a 2nd order accurate finite volume flow solver that could be significantly improved by implementing a higher order accurate approximation. This can be implemented through a reconstruction technique know as k-exact [45].

Reconstruction consists of determining the point-wise variation of the flow field variables from the cell averages with the restriction that integrating the point-wise function recovers the cell averages. Since the flow variables are known at the center of each tetrahedron a control volume of X cell centers can be constructed. The flow variables in the control volume can then be approximated by a polynomial of degree k:

$$P^k(x,y,z) = \sum_{i=0}^{k}\sum_{j=0}^{k-i}\sum_{l=0}^{k-(i+j)} C_{i,j,l}x^i y^j z^l \tag{8.1}$$

This polynomial satisfies a criteria call k-exactness. k-exactness means that a polynomial of degree k recovers the cell average correctly. In 3D, the polynomial contains $\frac{(k+1)(k+2)(k+3)}{6}$ coefficients and in 1D and 2D, it contains $k+1$ and $\frac{(k+1)(k+2)}{2}$ coefficients, respectively. e.g. $P^1(x,y,z) = C_0 + C_1 x + C_2 y + C_3 z$. As a final task to complete the reconstruction, a support stencil of number equal to the number of coefficients is selected, and the resulting linear system is solved. e.g for P^1 above

$$\begin{bmatrix} 1 & \overline{x}_1 & \overline{y}_1 & \overline{z}_1 \\ 1 & \overline{x}_2 & \overline{y}_2 & \overline{z}_2 \\ 1 & \overline{x}_3 & \overline{y}_3 & \overline{z}_3 \\ 1 & \overline{x}_4 & \overline{y}_4 & \overline{z}_4 \end{bmatrix} \begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix} = \begin{pmatrix} \overline{Q}_1 \\ \overline{Q}_2 \\ \overline{Q}_3 \\ \overline{Q}_4 \end{pmatrix}$$

PUMA should be modified to include a k-exact reconstruction. To approximate 4th order accuracy at least 35 cells will be required in each control volume. A detailed comparison of the accuracy improvement versus the computational costs should be completed on a variety of geometries, from simple to complex.

8.2 Preconditioning

Most of the PUMA test cases were conducted at a relatively high Mach number. Low speed flow can cause stiffness in the governing equations. This stiffness is caused when the ratio of the convective speed to the speed of sound is quite small. This can be best understood by examining the eigenvalues of the Euler equations $[u, u+c, u-c]$. This will lead to large errors or slow convergence in the flow solution when applying an iterative method. This stiffness problem can be solved by the inclusion of a preconditioner. This simply consists of the addition of a transformation matrix to put the coefficient matrix into a more favorable invertible matrix form [46]. If $Ax = b$ is the stiff system of equations a preconditioning matrix Q can be easily applied. The new system of equations would then become $Q^{-1}Ax = Q^{-1}b$. This new system of equations would be easier to solve, converge faster and produce more accurate results at low Mach numbers.

REFERENCES

[1] Healey, J. V. The prospects for simulating the helicopter/ship interface. *Naval Engineers Journal*, pp. 45–63, March 1987.

[2] Long, L. N., Liu, J., and Modi, A. Higher Order Accurate Solutions of Ship Airwake Flow Fields Using Parallel Computers. *RTO AVT Symposium on Fluid Dynamics Problems of Vehicles Operating near or in the Air-Sea Interface*, October 1998.

[3] Liu, J., and Long, L. N. Higher Order Accurate Ship Airwake Predictions for the Helicopter/Ship Interface Problem. *54th Annual American Helicopter Society Forum*, 1998.

[4] Rhoades, M. M., and Healey, J. V. Flight deck aerodynamics of a non-aviation ship. *Journal of Aircraft*, **29(4)**, pp. 619–626, 1992.

[5] Hurst, D. W., and Newman, S. Wind tunnel measurements of ship induced turbulence and the prediction of helicopter rotor blade response. *Vertica*, **12(3)**, pp. 267–278, 1988.

[6] Healey, J. V. Establishing a database for flight in the wake of structures. *Journal of Aircraft*, **29(4)**, pp. 559–564, 1992.

[7] Williams, S., and Long, K. R. Dynamic interface testing and the pilots rating scale. *The 53rd AHS Forum*, April 1997.

[8] Tai, T. C., and Cario, D. Simulation of DD-963 ship airwake by Navier-Stokes method. *AIAA Paper 93-3002*, 1993.

[9] Landsberg, A. M., Young, T. R., J., and Boris, J. P. An efficient, parallel method for solving flows in complex three-dimensional geometries. *AIAA Paper 94-0413*, 1994.

[10] Chaffin, M. S., and Berry, J. D. Navier-stokes and potential theory solutions for a helicopter fuselage and comparison with experiment. *NASA TM 4566*, 1990.

[11] Duque, E., and Berry, J. A Comparision of Computed and Experimental Flowfields of the RAH-66 Helicopter. *AHS Aeromechanics Specialist Meeting*, 1995.

[12] Bevilaqua, P. M., and Lykoudis, P. S. Mechanism of Entrainment in Turbulent Wakes. *AIAA Journal*, **9**, pp. 1657–1659, August 1971.

[13] Bevilaqua, P. M., and Lykoudis, P. S. Some Observations on the Mechanism of Entrainment. *AIAA Journal*, **15**, pp. 1194–1196, August 1977.

[14] Papailiou, D. D., and Lykoudis, P. S. Turbulent vortex streets and the entrainment mechanism of the turbulent wake. *Journal of Fluid Mechanics*, **62**, pp. 11–31, 1974.

[15] Cantwell, B., and Coles, D. An experimental study of entrainment and transport in the turbulent near wake of a circular cylinder. *Journal of Fluid Mechanics*, **136**, pp. 321–374, 1983.

[16] Bevilaqua, P. M., and Lykoudis, P. S. Turbulence memory in self-preserving wakes. *Journal of Fluid Mechanics*, **89**, pp. 589–606, 1978.

[17] Roshko, A. Experiments on the flow past a circular cylinder at very high Reynolds number. *Journal of Fluid Mechanics*, **10**, pp. 345–356, 1961.

[18] Roshko, A. On the Wake and Drag of Bluff Bodies. *Journal of Aeronautical Sciences*, **22**, pp. 124–132, 1955.

[19] Sakamoto, H., and Haniu, H. The formation mechanism and shedding frequency of vortices from a sphere in unifrom shear flow. *Journal of Fluid Mechanics*, **287**, pp. 151–171, 1995.

[20] Modi, V. J., and Akutsu, T. Wall Confinement Effects for Spheres in the Reynolds Number Range of 30-2000. *Trans. of ASME I: Journal of Fluids Engg*, **106**, pp. 66–73, March 1984.

[21] Sakamoto, H., and Haniu, H. A Study on Vortex Shedding From Spheres in a Uniform Flow. *Trans. of ASME I: Journal of Fluids Engg*, **112**, pp. 386–392, December 1990.

[22] Taneda, S. Visual observations of the flow past a sphere at Reynolds numbers between 10^4 and 10^6. *Journal of Fluid Mechanics*, **85**, pp. 187–192, 1978.

[23] Achenbach, E. Vortex shedding from spheres. *Journal of Fluid Mechanics*, **62**, pp. 209–221, 1974.

[24] Lauchle, G. C., Jones, A. R., Dreyer, J. J., and Wang, J. Flow-induced lift forces on a towed sphere. *Proceedings of the ASME Noise Control and Acoustics Division*, **25**, pp. 103–111, 1998.

[25] Wu, J., and Faeth, G. Sphere wakes in still surroundings at intermediate reynolds numbers. *AIAA Journal*, 1993.

[26] Zdravkovich, M. M. *Flow Around Circular Cylinders, Vol 1: Fundamentals.* Oxford Science Publications, 1997.

[27] Tomboulides, A. G. Direct and large-eddy simulation of wake flows: Flow past a sphere. PhD Dissertation, Princeton University, Department of Mechanical and Aerospace Engineering, June 1993.

[28] Morris, P. J., Long, L. N., Bangalore, A., and Wang, Q. A Parallel Three-Dimensional Computational Aeroacoustic Method Using Non-Linear Disturbance Equations. *Journal of Computational Physics*, **133**, pp. 56–74, 1997.

[29] Bruner, C. W. S. Parallelization of the Euler equations on unstructured grids. PhD Dissertation, Virginia Polytechnic Institute and State University, Department of Aerospace Engineering, http://scholar.lib.vt.edu/theses/public/, May 1996.

[30] Bruner, C. W. S., and Walters, R. W. Parallelization of the Euler equations on unstructured grids. *AIAA Paper 97-1894*, January 1997.

[31] Pirzadeh, S. Progress Toward a User-Oriented Unstructured Viscous Grid Generator. *AIAA Paper 96-0031*, January 1996.

[32] Schweitzer, S. Computational Simulation of Flow around Helicopter Fuselages. Master's Thesis, The Pennsylvania State University, Department of Aerospace Engineering, http://family.schweitzer.net/steve/thesis.pdf, January 1999.

[33] Flynn, M. J. Some computer organizations and their effectiveness. *IEEE Transactions in Computers*, **C-21**, pp. 948–960, 1972.

[34] Duff, I. S., Erisman, A. M., and Reid, J. K. *Direct methods for sparse matrices.* Oxford University Press, 1986.

[35] Karypis, G. Family of Multilevel Partitioning Algorithms. http://www-users.cs.umn.edu/ karypis/metis/, 1997.

[36] Bruner, C. W. S. Parallel Unstructured Maritime Aerodynamics (PUMA) manual. http://cocoa.ihpca.psu.edu/puma/manual.pdf, 1996.

[37] Krist, S. L., Beidron, R. T., and Rumsey, C. L. CFL3D User Manual. *Aerodynamic and Acoustic Methods Branch of the NASA Langley Research Center*, 1996.

[38] Jameson, A., Baker, T. J., and Weatherhill, N. P. Calculation of Inviscid Transonic Flow over a Complete Aircraft. *AIAA Paper 86-0103*, January 1986.

[39] Mavriplis, D. J., Jameson, A., and Martinelli, L. Multigrid Solution of the Navier-Stokes Equations on Triangular Meshes. *AIAA Paper 89-0120*, January 1989.

[40] Modi, A. COst effective COmputing Array. http://cocoa.ihpca.psu.edu, 1998.

[41] Woo, A. NAS Parallel Benchmarks. http://science.nas.nasa.gov/Software/NPB/, 1997.

[42] Keller, J., and Smith, E. Analysis and Control of the Transient Shipboard Engagement Behavior of Rotor Systems. *Proceedings of the 55th Annual National Forum of the American Helicopter Society (Montreal, Canada)*, May 1999.

[43] Freeman, C. E., and Mineck, R. E. Fuselage Measurements of a Helicopter Wind-Tunnel Model with a 3.15-meter Diameter Single Rotor. *NASA TM 80051*, 1979.

[44] Schlichting, H. *Boundary-Layer Theory.* McGraw Hill Co., 7th edition, 1979.

[45] Barth, T. J. Recent Developments in High Order k-exact Reconstruction on Unstructured Meshes. *AIAA Paper 93-0668*, January 1993.

[46] Turkel, E., Vatsa, V. N., and Radespiel, R. Preconditioning methods for low-speed flows. *NASA CR-201605 ICASE Report No. 96-57*, October 1996.

Appendix A

COCOA HOWTO?

What is COCOA?

COCOA stands for **CO**st effective **CO**mputing **A**rray. It is a Beowulf class super-computer. Beowulf is a multi computer architecture which can be used for parallel computations. It is a system which usually consists of one server node, and one or more client nodes connected together via Ethernet or some other fast network. It is a system built using commodity hardware components, like any office desktop PC with standard Ethernet adapters, and switches. It does not contain any custom hardware components and is trivially reproducible. `<http://cocoa.ihpca.psu.edu/>`.

What hardware was used to build COCOA?

26 WS-410 workstations from Dell `<http://www.dell.com>`, each consisting of:

1. Dual 400 MHz Intel Pentium II Processors w/512K L2 cache
2. 512 MB SDRAM
3. 4 GB UW-SCSI2 Disk
4. 3COM 3c509B Fast Ethernet adapter (100 Mbits/sec)
5. 32x SCSI CD-ROM drive
6. 1.44 MB floppy drive
7. Cables

In addition, the following were also used:

1. One Baynetworks 450T 24-way 100 Mbits/sec switch
2. Two 12-way Monitor/keyboard/mouse switches
3. Four 500 kVa Uninterruptible Power Supplies from APC.
4. One monitor, keyboard, mouse and 54 GB of extra UW-SCSI2 hard disk space for one PC which was used as the server.

What is the operating system on COCOA?

Linux! In specific, RedHat Linux 5.1 distribution `<http://www.redhat.com>`.

Linux is a free version of the Unix operating system, and it runs on all PC/i386 compatible computers (and now also on PowerPCs, Alphas, Sparcs, Mips, Ataris, and Amigas). The Linux kernel is written by Linus Torvalds <torvalds@transmeta.com> and other volunteers. Most of the programs running under Linux are generic Unix freeware, many of them from the GNU project.

What software is installed on COCOA?

On the server, the following software is installed:

1. Base packages from RedHat Linux 5.1 distribution `<http://www.redhat.com>`
2. Freeware GNU C/C++ compiler as well as Pentium optimized GNU C/C++ compiler (*gcc, pgcc*)
3. Fortran 77/90 compiler and debugger by Portland Group
4. Freeware **M**essage **P**assing **I**nterface (MPI) libraries for parallel programming in C/C++/Fortran 77/Fortran 90.
5. Scientific Visualization Software TECPLOT from Amtec Corporation `<http://www.amtec.com>`

How much did COCOA cost?

Approximately *$100,000*!

A.1 Details on how COCOA was built

A.1.1 Setting up the hardware

Setting up the hardware was fairly straight-forward. Here are the main steps:

1. Unpacked the machines, mounted them on the rack and numbered them.
2. Set up the 24-port network switch and connected one of the 100 Mbit ports to the second ethernet adapter of the server which was meant for the private network. The rest of the 23 ports were connected to the ethernet adapters of the clients. Then

an expansion card with 2 additional ports was added on the switch to connect the remaining 2 clients.

3. Stacked the two 16-way monitor/keyboard switches and connected the video-out and the keyboard cables of each of the 25 machines and the server to it. A single monitor and keyboard were then hooked to the switch which controlled the entire cluster.

4. Connected the power cords to the four UPS.

A.1.2 Setting up the software

Well, this is where the real effort came in! Here are the main steps:

1. The server was the first to be set up. RedHat Linux 5.1 was installed on it using the bundled CD-ROM. Most of the hardware was automatically detected (including the network card), so the main focus was on partitioning the drive and choosing the relevant packages to be installed. A 3 GB growable root partition was created for the system files and the packages to be installed. Two 128 MB swap partitions were also created and the rest of the space (50 GB) was used for various user partitions. It was later realised that a separate `/tmp` partition of about 1 GB was a good idea.

2. The latest stable Linux kernel (then `#2.0.36`) was downloaded and compiled with SMP support using the Pentium GNU CC compiler `pgcc` `<http://www.goof.com/pcg/>` (which generates highly optimised code specifically for the Pentium II chipset) with only the relevant options required for the available hardware. The following optimisation options were used: `pgcc -mpentiumpro -O6 -fno-inline-functions`. Turning on SMP support was just a matter of clicking on a button in the *Processor type and features* menu of the kernel configurator (started by running `make xconfig`).

3. The new kernel-space NFS server for linux (*knfsd*) `<http://www.csua.berkeley.edu/~gam3/knfsd/>` was installed to replace the earlier user-space NFS server to obtain improved NFS performance. For quick and hassle-free installation, a RedHat RPM package was obtained from `<http://rufus.w3.org/linux/RPM/>`, a popular RPM repository. The default options were used.

4. `ssh` was downloaded from `<http://www.cs.hut.fi/ssh/>`, compiled and installed for secure access from the outside world. `ssh-1.2.26` was preferred over the newer

ssh-2.0.11 as ssh v2.x was much slower as well as backward incompatible. sshd daemon was started in runlevel 3 under /etc/rc.d/rc3.d. Recently, RedHat RPMs for ssh have started appearing in <http://rufus.w3.org/linux/RPM/> and several other RPM repositories, which make it much easier to install.

5. Both the 3c905B ethernet adapters were then configured; one that connected to the outside world (eth1) with the real IP address 128.118.170.11, and the other which connected to the private network (eth0) using a dummy IP address 10.0.0.1. Latest drivers for the 3COM 3c905B adapters written by Donald Becker (3c59x.c v0.99H <http://cesdis.gsfc.nasa.gov/linux/drivers/vortex.html>) were compiled into the kernel to ensure 100 Mbit/sec Full-duplex connectivity. This was checked using the vortex-diag utility <http://cesdis.gsfc.nasa.gov/linux/diag/vortex-diag.c>. For the configuration, the following files were modified: /etc/sysconfig/network, /etc/sysconfig/network-scripts/ifcfg-eth0 and /etc/sysconfig/network-scripts/ifcfg-eth1. Here is how they looked for me after modification: /etc/sysconfig/network:

```
NETWORKING=yes
FORWARD_IPV4=no
HOSTNAME=cocoa.ihpca.psu.edu
DOMAINNAME=ihpca.psu.edu
GATEWAY=128.118.170.1
GATEWAYDEV=eth1
NISDOMAIN=ihpca.psu.edu
```

/etc/sysconfig/network-scripts/ifcfg-eth0:

```
DEVICE=eth0
IPADDR=10.0.0.1
NETMASK=255.255.255.0
NETWORK=10.0.0.0
BROADCAST=10.0.0.255
ONBOOT=yes
BOOTPROTO=none
USERCTL=no
```

/etc/sysconfig/network-scripts/ifcfg-eth1:

```
DEVICE=eth1
IPADDR=128.118.170.11
NETMASK=255.255.255.0
NETWORK=128.118.170.0
BROADCAST=128.118.170.255
ONBOOT=yes
BOOTPROTO=none
USERCTL=no
```

6. For easy and automated install, I decided to boot each of the PCs from the network using the BOOT protocol. The BOOTP server was enabled by uncommenting the following line in `/etc/inetd.conf`:

```
bootps  dgram   udp     wait    root    /usr/sbin/tcpd  bootpd
```

   A linux boot floppy was prepared with the kernel support for 3c905B network adapter which was used to boot each of the client nodes to note down their unique 96-bit network hardware address (e.g., 00C04F6BC052). Using these address, the `/etc/bootptab` was edited to look like:

```
.default:\
        :hd=/boot:bf=install.ks:\
        :vm=auto:\
        :dn=hpc.ihpca.psu.edu:\
        :gw=10.0.0.1:\
        :rp=/boot/client/root:

node1:ht=ethernet:ha=00C04F6BC0B8:ip=10.0.0.2:tc=.default
node2:ht=ethernet:ha=00C04F79AD76:ip=10.0.0.3:tc=.default
node3:ht=ethernet:ha=00C04F79B5DC:ip=10.0.0.4:tc=.default
.
.
.
node25:ht=ethernet:ha=00C04F79B30E:ip=10.0.0.26:tc=.default
```

7. The `/etc/hosts` file was edited to look like:

```
127.0.0.1       localhost       localhost.localdomain
```

```
# Server [COCOA]
128.118.170.11 cocoa.ihpca.psu.edu cocoa.aero.psu.edu cocoa

# IP address <--> NAME mappings for the individual nodes of the cluster
10.0.0.1        node0.hpc.ihpca.psu.edu node0           # Server itself!
10.0.0.2        node1.hpc.ihpca.psu.edu node1
10.0.0.3        node2.hpc.ihpca.psu.edu node2
.
.
.
10.0.0.26       node25.hpc.ihpca.psu.edu node25
```

The `/etc/host.conf` was modified to contain the line:

```
order hosts,bind
```

This was to force the lookup of the IP address in the `/etc/hosts` file before requesting information from the DNS server.

8. The filesystems to be exported were added to `/etc/exports` file which looked like:

```
/boot           node*.hpc.ihpca.psu.edu (ro,link_absolute)
/mnt/cdrom      node*.hpc.ihpca.psu.edu (ro,link_absolute)
/usr/local      node*.hpc.ihpca.psu.edu (rw,no_all_squash,no_root_squash)
/home1          node*.hpc.ihpca.psu.edu (rw,no_all_squash,no_root_squash)
/home2          node*.hpc.ihpca.psu.edu (rw,no_all_squash,no_root_squash)
/home3          node*.hpc.ihpca.psu.edu (rw,no_all_squash,no_root_squash)
/home4          node*.hpc.ihpca.psu.edu (rw,no_all_squash,no_root_squash)
```

9. For rapid, uniform and unattended installation on each of the client nodes, RedHat 5.1 KickStart installation was ideal. Here is how my kickstart file `/boot/install.ks` looked like:

```
lang en
network --bootproto bootp
nfs --server 10.0.0.1 --dir /mnt/cdrom
keyboard us
zerombr yes
clearpart --all
```

```
part / --size 1600
part /local --size 2048
part /tmp --size 400 --grow
part swap --size 127
install
mouse ps/2
timezone --utc US/Eastern
rootpw --iscrypted kQvti0Ysw4r1c
lilo --append "mem=512M" --location mbr
%packages
@ Networked Workstation
%post
rpm -i ftp://10.0.0.1/pub/CLUSTER/RPMS/wget-1.5.0-2.i386.rpm
rpm -i ftp://10.0.0.1/pub/CLUSTER/RPMS/xntp3-5.93-2.i386.rpm
/usr/bin/wget ftp://10.0.0.1/pub/CLUSTER/kernel/vmlinuz -O/boot/vmlinuz
/usr/bin/wget ftp://10.0.0.1/pub/CLUSTER/conf/lilo.conf -O/etc/lilo.conf
/sbin/lilo
/usr/bin/wget ftp://10.0.0.1/pub/CLUSTER/conf/hosts.equiv -O/etc/hosts.equiv
sed "s/required\(.*securetty\)/optional\1/g" /etc/pam.d/rlogin > /tmp/rlogin
mv /tmp/rlogin /etc/pam.d/rlogin
```

For more info on RedHat KickStart installation, look at: `<http://wwwcache.ja.net/dev/kickstart/KickStart-HOWTO.html>`. In one of the post installation commands above, the first line of the `/etc/pam.d/rlogin` file is modified to contain:

```
auth        optional     /lib/security/pam_securetty.so
```

This is to required enable `rlogin/rsh` access from the server to the client without password which is very useful for the software maintenance of the clients. Also, the `/etc/hosts.equiv` file mentioned above looks like this:

```
node0
node1
node2
node3
.
.
```

```
.
node25
```

The RedHat Linux 5.1 CD-ROM was then mounted as `/mnt/cdrom` on the server which was NFS exported to the client nodes. A new kernel with SMP support was compiled for the client nodes in very much the same way as for the server and was used to replace the existing kernel in the RedHat book diskette. This kernel however had lesser options compiled in as it was only meant to act as a client. Additionally, option for "kernel level autoconfiguration using BOOTP" was enabled in the *Networking options* menu of the kernel configurator. This was required in order for the node to automatically get its IP address from the server at boot time. Support for The configuration file of the boot diskette was modified so as to boot directly in the KickStart mode. All that was needed to configure each client now was to insert the boot diskette, power-up the workstation and wait until the automatic installation was completed. Simple, eh ?!

10. As soon as all the clients were rebooted after installation, the cluster was up and running! Some useful utilities like `brsh` (`<http://www.beowulf.org/software/RPMS/beobase-2.0-1.i386.rpm>`) were installed to enable `rsh` a single identical command to each of the client nodes. This was then used to make any fine changes to the installation. NIS could have been installed to manage the user logins on every client node, but instead a simple shell script was written to distribute a common `/etc/passwd`, `/etc/shadow` and `/etc/group` file from the server.

11. Most of the services were disabled in `/etc/inetd.conf` for each of the client nodes as they were unnecessary. The stripped down `/etc/inetd.conf` for the client nodes finally looked like:

```
shell   stream  tcp     nowait  root    /usr/sbin/tcpd  in.rshd
auth    stream  tcp     nowait  nobody  /usr/sbin/in.identd in.identd -l -e -o
```

12. *automount* package was installed on each of the nodes to automatically mount the various user partitions on demand. Although this gave slightly improved NFS performance, it was found to be buggy and unstable. Finally, it was decided that *automount* for Linux was not yet ready for prime-time and was removed in favor of conventional NFS mounts.

13. The Portland Group Fortran 77/90 and HPF compilers (commercial) were then installed on the server.

14. Source code for freeware implementation of MPI library, MPI-CH was downloaded from `<http://www.mcs.anl.gov/mpi/>` and compiled using `pgcc`. Installing it on the server on the `/usr/local` partition was quite straight-forward with no major hassles. The `mpif77` script was modified to suit our needs and a similar `mpif90` was created. The `/usr/local/mpi/util/machines/machines.LINUX` was then modified to add two entries for each client node (as they were dual-processor SMP nodes). Jobs could now be run on the cluster using interactive `mpirun` commands!

15. A queueing system, DQS v3.0 was downloaded from `<http://www.scri.fsu.edu/~pasko/dqs.html>`, compiled and installed as `/usr/local/DQS/` making it available to all the client nodes through NFS. Appropriate server and client changes were then made to get it functional (i.e. adding the relevant services in `/etc/services`, starting `qmaster` on the server and `dqs_execd` on the clients) , although a few minor irritants were encountered. These were mainly owing to the bad documentation for DQS. It took a long time for me to figure out exactly how to configure the DQS to recognize a slave node, but once it was done, setting up the same for rest of the nodes was trivial. Wrapper shell scripts were then written by me for `qsub`, `qstat` and `qdel` which not only beautified the original DQS output (which was *ugh* to begin with!), but also added a few enhancements. For example, `qstat` was modified to show the number of nodes requested by each pending job in the queue. Also, three additional shell scripts `qinfo`, `qload` and `qmem` were written to give some useful load data for the nodes and the cluster resource utilization.

16. COCOA was now fully-functional, up and running and ready for benchmarking and serious parallel jobs! As with the kernel, use of `pgcc` compiler was recommended for all the C/C++ codes. In particular, using `pgcc` with options "`-mpentiumpro -O6 -funroll-all-loops`" for typical FPU intensive number crunching codes resulted in *30 %* increase in execution speed over the conventional `gcc` compiler.

Appendix B

Sample input file for PUMA

```
Gamma      R          rhoRef    Vref      muRef
1.4        287.04     1.3813    320.46    1.831e-5
rho_inf     u_inf      v_inf      w_inf      p_inf         fsMult    xyzMult
1.3813      58.1152    0.0        0.0        101325.0      1.0       1.0
mu_inf   T_Suth  PrLam   PrTurb
1.831e-5 110.0   0.72    0.9
numIts      maxMinutes      wrRestart      wrResid      relResid      absResid
150         50000           5a             1            1.0e-6        1.0e-16
CFLconst      CFLslope
26            15
localStepping  spatial_order    inviscid_flux     integ_scheme    limiter
1              2                "roe"             "ssor"          "none"
innerIters    innerTol      omega     stages    K         commScheme
50            1.0e-8        1.0       2         5.0e+2    "delta q"
gridName                   restartFrom      restartTo        residName
"grids/apache.sg.gps"      ""               "apache.rst"     "apache.rsd"
implicitBCs    viscousModel      artDiss      VIS-2   VIS-4   M_CR
1              "inviscid"        0            1.00    1.0     0.85
Num_surfaces
7
surface BC_type rho u v w p
1       8
2       5
3       5
4       5
5       5
6       5
7       5
```

Appendix C

Scripts for the Live CFD-cam

C.1 Shell script for the server_cfd utility

```
#!/bin/sh
#----------------------------------------------------------------------------
#
# Usage: server_cfd <CFD-cam ID> <boolean arg to keep/delete generated files>
#
#----------------------------------------------------------------------------
read_conf_file () {
if [ $# != 1 ] ; then
      echo "Usage: read_conf_file {conf-file-name}"
      exit 1
fi
grid=`awk '/GridFile/ {print $3}' $1`
opt=`awk '/toTecplot_Options/ {print $3,$4}' $1`
layout=`awk '/Tecplot_Layout_File/ {print $3,$4,$5,$6,$7,$8,$9,$10,$11, \
               $12,$13,$14,$15,$16,$17,$18,$19,$20}' $1`
host=`awk '/Destination_Machine/ {print $3}' $1`
ddir=`awk '/Destination_Directory/ {print $3}' $1`
fn=`awk '/Destination_File_Name/ {print $3}' $1`
size=`awk '/ImageSize/ {print $3}' $1`
curfile=`awk '/Remote_Flag_File/ {print $3}' $1`
rsdfile=`awk '/Residual_File/ {print $3}' $1`
}
#----------------------------------------------------------------------------
kill_program () {
if [ $# != 1 ] ; then
      echo "Usage: kill_program {program-name}"
      exit 1
fi
for f in `myps | grep "$1" | grep -v grep | grep -v "vim " | \
      grep -v $$ | awk '{print $2}' 2>/dev/null`;
do
      kill -9 $f
done
}
#----------------------------------------------------------------------------
server_init () {
if [ $# != 1 ] ; then
      echo "Usage: server_init {restart-file-name}"
      exit 1
fi
while test ! -f $1
```

```
do
      sleep 2
done

while [ `cat $1 | wc -c` = 0 ]
do
      sleep 2
done
}
#---------------------------------------------------------------------------
does_file_exist () {
if [ $# != 1 ] ; then
      echo "Usage: does_file_exist {restart-file-name}"
      exit 1
fi
file=`tail -1 $1`
base=`basename $file .rst`
existence=0
secondflag=`echo $file | wc -w`
if [ $secondflag = 0 ]; then
      return $existence
fi
for f in $file*; do
      if test -f $f
      then
            existence=1
      fi
done
return $existence
}
#---------------------------------------------------------------------------
consolidate_file () {
if test ! -f $base.rst
then
      echo "Making $file"
      consolidate $file
      rm -f $file.q.* $file.r.*
      ln -s -f $file last.rst
fi
}
#---------------------------------------------------------------------------
make_plt_file () {
inum=`echo $file | sed 's/i//g' | awk '{print $1}' OFMT="%04d" FS="_"`
if test ! -f $base.plt
then
      make_tecplot_vort $grid $file $opt > /dev/null 2> /dev/null
fi
}
#---------------------------------------------------------------------------
make_gifs () {
num=0
```

```
for f in $layout; do
      num=`echo $num | awk '{printf "%02d", $1+1}'`
      sed "s/IterNUM/$inum/g" $f > /tmp/lay.$$
      myslice_template $base.plt /tmp/lay.$$ $size
      mv ${base}.gif ${base}_${num}.gif
      rm -f /tmp/lay.$$
done
rm -f ${base}.gif
convert -delay 300 -loop 100000 ${base}_*.gif $base.gif
rm -f ${base}_*.gif
cp $base.gif LIVE.gif
}
#----------------------------------------------------------------------------
get_target_gif_name () {
if [ `echo $fn | grep -c ITER` = 0 ]
then
      fname=$fn
else
      fname=$base.gif
fi
}
#----------------------------------------------------------------------------
plot_convergence_history () {
mkplot $rsdfile 4 convergence.gif $convsize "Relative L2 Norm vs Iteration" \
        "Iterations ->" "Relative L2 Norm ->" \
        "set logscale y" "set format y \"%5.0e\""
}
#----------------------------------------------------------------------------
copy_to_webpage () {
printf "%s" "Copying $fname to Client...."
machine=`hostname`
echo HOST = `nslookup $machine | awk '/Name:/ {print $2}'` > /tmp/CURRENT.$$
echo TIME = `date` >> /tmp/CURRENT.$$
echo DIRECTORY = $ddir >> /tmp/CURRENT.$$
echo FILENAME = $fname >> /tmp/CURRENT.$$
scp -p LIVE.gif $host/$ddir/$fname
scp -p convergence.gif $host/$ddir/convergence/
scp -p err.out $host/$ddir/convergence/
scp -p /tmp/CURRENT.$$ $curfile
echo "done [`date`]"
rm -f /tmp/CURRENT.$$
}
#----------------------------------------------------------------------------
delete_last () {
rm -f LIVE.gif
mv $base.rst last.rst
mv $base.gif last.gif
mv $base.plt last.plt
}
#----------------------------------------------------------------------------
# Size for convergence graph (%)
```

```
convsize=70

kill_program "server_cfd $1"
read_conf_file SERVER.conf
rm -f FOREVER
server_init restart.txt

echo "Starting:"
echo
cp $HOME/bin/TEMPLATE.gif LIVE.gif
#display -update 2 -geometry $size% LIVE.gif &
while test ! -f FOREVER
do

      does_file_exist restart.txt

      if [ $existence = 1 ] ; then
            if test ! -f $base.gif
            then
                  consolidate_file
                  read_conf_file SERVER.conf
                  make_plt_file
                  make_gifs
                  get_target_gif_name
                  plot_convergence_history
                  copy_to_webpage
                  if [ $# = 2 ] ; then
                        delete_last
                  fi
            else
                  sleep 2
            fi
      else
            sleep 2
      fi
done
#----------------------------------------------------------------------------
```

C.2 Sample initialization file `SERVER.conf` for the `server_cfd` utility

```
GridFile = grids/apache.sg.gps
ImageSize = 60
toTecplot_Options =  1 remove_surf.inp
Tecplot_Layout_Files = apache_M_nomesh.lay apache_CP_nomesh.lay
Destination_Machine = anirudh@cocoa.ihpca.psu.edu:~/public_html/cfdcam6
Destination_Directory = Apache
Destination_File_Name = ITER
Remote_Flag_File = anirudh@cocoa.ihpca.psu.edu:~/public_html/cfdcam6/CURRENT
Residual_File = apache.rsd
```

C.3 Shell script for the tec2gif utility

```
#!/bin/sh
#
if ! test $# -eq 3
  then
     b=`basename $0 .e`
     echo "Usage: $b <tecplot data file> <tecplot layout file>" \
          "<size percentage>"
     exit 1
fi
size=$3
cur=current$$
exp=export$$

for f in $1; do      # MAIN LOOP: for every .plt file specified

iter=`echo $f | sed 's/i//g' | awk '{print $1*1}' OFMT="%04d" FS="_"`
base=`basename $f .plt`
sed "s/\(VarSet.*'\).*'/\1$f'/g" $2 | sed "s/IterNUM/$iter/g" > $cur.lay
printf "%s" "Creating $base.gif......"
rm -f export.eps
printf "%s" "0"
cat $HOME/bin/Tecplot/slice.mcr | sed "s/current/$cur/g" | \
     sed "s/export/$exp/g" > /tmp/slice$$.mcr
tecplot -b -p /tmp/slice$$.mcr > /dev/null
printf "%s" "1"
touch $exp.eps
while test ! -f $exp.eps
     do
          sleep 1
     done
sleep 2
touch $exp.eps
mv $exp.eps $base.eps
sleep 2
touch ${base}.eps
printf "%s" "2"
mogrify -geometry ${size}% -format gif ${base}.eps
printf "%s" "3"
mv ${base}.eps /tmp/export.eps
printf "%s" "4...."
echo "done"

done          # for every .plt file specified
# Clean up....
rm -f tecplot.phy
rm -f tp??????
rm -f current$$.lay batch.log /tmp/slice$$.mcr
```

C.4 Shell script for the client_cfd utility

```
#!/bin/sh
#
# Usage: client_cfd <ID of the CFD-cam> <refresh interval in seconds>
#
cfddir=$HOME/public_html/cfdcam$1
curfile=CURRENT

for f in `myps | grep "client_cfd $1" | grep -v grep | grep -v vim |\
            grep -v $$ | awk '{print $2}' 2>/dev/null`;
do
      echo Killed process $f
      kill -9 $f
done

cd $cfddir
rm -f FOREVER
if test ! -f $curfile
then
      cp defaults/CURRENT.template $curfile
      cp defaults/CURRENT.gif template/i0000_template.gif
      cp defaults/CURRENT.gif template/convergence/convergence.gif
      mogrify -geometry 400x300 template/convergence/convergence.gif
fi

if [ `echo $2 | awk '{print $1*1}'` = 0 ]
then
      interval=10
else
      interval=$2
fi

while test ! -f FOREVER
do

host=`cat $curfile | grep HOST | awk '{print $3}'`
time=`cat $curfile | grep TIME | awk '{print $3,$4,$5,$6,$7,$8,$9}'`
file=`cat $curfile | grep FILENAME | awk '{print $3}'`
ddir=`cat $curfile | grep DIRECTORY | awk '{print $3}'`
iter=`echo $file | sed 's/i//g' | awk '{print $1*1}' OFMT="%d" FS="_"`
curtime=`date`
mogrify -geometry 400x300 $ddir/$file
echo Made $ddir/$file [`date`]
cat index_template.html |\
      sed "s/INTERVAL/$interval/g" |\
      sed "s/MACHINE_NAME/$host/g" |\
      sed "s/DIRECTORY/$ddir/g" |\
      sed "s/IMAGE\.GIF/$file/g" |\
      sed "s/LAST_TIME/$time/g" |\
      sed "s/CUR_TIME/$curtime/g" |\
```

```
      sed "s/LAST_ITER/$iter/g" \
      > index.html
cat main_template.html |\
      sed "s/INTERVAL/$interval/g" |\
      sed "s/MACHINE_NAME/$host/g" |\
      sed "s/DIRECTORY/$ddir/g" |\
      sed "s/IMAGE\.GIF/$file/g" |\
      sed "s/LAST_TIME/$time/g" |\
      sed "s/CUR_TIME/$curtime/g" |\
      sed "s/LAST_ITER/$iter/g" \
      > main.html
cat sidebar_template.html |\
      sed "s/INTERVAL/$interval/g" |\
      sed "s/MACHINE_NAME/$host/g" |\
      sed "s/DIRECTORY/$ddir/g" |\
      sed "s/IMAGE\.GIF/$file/g" |\
      sed "s/LAST_TIME/$time/g" |\
      sed "s/CUR_TIME/$curtime/g" |\
      sed "s/LAST_ITER/$iter/g" \
      > sidebar.html
chmod a+r sidebar.html $ddir/$file
find $ddir/convergence -type l -name "*" -exec rm -f {} \; # Remove links
cd $ddir/convergence/
ln -s convergence.gif $iter.gif
echo "<html><body bgcolor=white><pre>" > conv.html
cat err.out >> conv.html
echo "</pre></body></html>" >> conv.html
ln -s conv.html $iter.html
cd ../../
cfile=$iter.gif
cat convergence_template.html |\
      sed "s/INTERVAL/$interval/g" |\
      sed "s/MACHINE_NAME/$host/g" |\
      sed "s/DIRECTORY/$ddir/g" |\
      sed "s/IMAGE\.GIF/$cfile/g" |\
      sed "s/LAST_TIME/$time/g" |\
      sed "s/CUR_TIME/$curtime/g" |\
      sed "s/LAST_ITER/$iter/g" \
      > $ddir/convergence/index.html
sleep $interval
done
```