NASA Technical Memorandum 102188

Development of a Navier-Stokes Algorithm for Parallel-Processing Supercomputers

Julie M. Swisshelm

May 1989

NASA
National Aeronautics and Space Administration

NASA Technical Memorandum 102188

Development of a Navier-Stokes Algorithm for Parallel-Processing Supercomputers

Julie M. Swisshelm, Ames Research Center, Moffett Field, California

May 1989

NASA

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California 94035

Table of Contents

DEVELOPMENT OF A NAVIER-STOKES ALGORITHM FOR PARALLEL-PROCESSING SUPERCOMPUTERS

ABSTRACT

An explicit flow solver, applicable to the hierarchy of model equations ranging from Euler to full Navier-Stokes, is combined with several techniques designed to reduce computational expense. The computational domain consists of local grid refinements embedded in a global coarse mesh, where the locations of these refinements are defined by the physics of the flow. Flow characteristics are also used to determine which set of model equations is appropriate for solution in each region, thereby reducing not only the number of grid points at which the solution must be obtained, but also the computational effort required to get that solution. Acceleration to steady-state is achieved by applying multigrid on each of the subgrids, regardless of the particular model equations being solved. Since each of these components is explicit, advantage can readily be taken of the vector- and parallel-processing capabilities of machines such as the Cray X-MP and Cray-2.

1. Introduction

1.1 Motivation

The objective of this research is to develop and explore an algorithmic strategy for efficient simulation of complex aerodynamic flows on parallel-processing supercomputers. Minimizing flow field solution time can be accomplished by increasing both the efficiency of numerical algorithms and their rate of computation. Although several researchers have computed flows over complete aircraft configurations [1-3], the routine use of this capability as a design tool is beyond the reach of available processing power. Likewise, progress is being made in the simulation of internal flows, such as the flow through turbomachine stages [4]. However, simulation of an entire engine in which all parts are functioning would require orders-of-magnitude increases in computer speed and memory size. On a much different physical scale, calculation of turbulent flows using large eddy simulation with the Navier-Stokes equations requires such high resolution that computations on appropriate grids can currently only be done for very simple geometries. To apply the full Navier-Stokes equations to turbulent flows over complex geometries with no modeling of turbulence is, again, far beyond present capabilities. Figure 1.1 [5] summarizes these observations.

There are several ingredients key to attacking these currently-intractable fluid dynamics problems. The first is the development of robust algorithms which require a minimal number of operations to obtain solutions. Secondly,

judicious distribution of grid points in the physical domain of the problem to optimize resolution and minimize cost is important. Finally, attaining the maximum execution rate (and hence, efficiency) possible of the fastest computers currently available, while paying due consideration to architectural design evolution, is crucial.

The state-of-the-art in algorithm technology for compressible flow simulation includes both implicit and explicit solution procedures for the Reynolds-averaged Navier-Stokes equations; examples are the Beam-Warming implicit scheme or the explicit MacCormack or Runge-Kutta schemes with multigrid convergence acceleration. Additionally, multiblock or zonal methods are gaining in popularity, and more sophisticated artificial dissipation models, upwind schemes using Riemann solvers and total variation diminishing schemes are being increasingly used for higher accuracy in complex flow simulation. Emphasis is beginning to shift toward time-accurate three-dimensional simulations as computing power and storage capacity grows.

Closely coupled to algorithm strategies are advances in grid generation techniques. The use of unstructured meshes, automated or adaptive grid refinements, localized grid-embedding schemes, and interactive grid generation procedures contribute to a reduction in both computational costs and human effort required to devise discretizations of complex flow field domains. Expert systems may even become the common means of grid definition in the future [6,7].

Achieving the maximum performance of the most powerful computers available enjoins the use of multiple processing units focused on a single solution. The need for using more than one processor for improvement in computational

capability is driven by the fact that the speed of the single CPU is approaching its technological limit [8]. The actual implementation of parallelism in computer architecture takes many forms, as will be discussed subsequently; whether many weak processors are better suited for CFD algorithms than are a few powerful ones is a subject of contemporary research. The level of maturity of software (such as parallel languages, autotasking compilers, and tools for multitasking analysis) has significant impact on the pace at which parallel algorithms are developed. The techniques of vectorization, after a decade of use, are fairly well-understood, while methods for multiprocessing are relatively undeveloped. Until mature tools to facilitate parallel code development become widely available, significant attention must be paid to the details of programming advanced computers.

The challenge of solving complex flow problems in a parallel computing environment forms the impetus for this work. With the ultimate goal of being able to numerically solve the Navier-Stokes equations for realistic three-dimensional configurations, the approach taken here entails the application of a robust flow solver, the creation of embedded grid refinements, the use of convergence acceleration, and adaptation of the solution procedure to a parallel-processing supercomputer. Optimization of each element in the solution strategy must be done with consideration of its effect on the others; that is, the absolutely best scalar algorithm may perform quite poorly on a particular advanced-architecture machine, and as such might not be the method of choice. Parallel computing requires a "detailed understanding of the interactions among large-scale applications, parallel algorithms, and the architectures of parallel-processing devices" [9]. Therefore, a synergistic approach is taken.

Due to the high level of contemporary interest in this approach, a review of recent developments is useful. The reader may note that much of the research reviewed here has been carried out in the not-to-distant past; this fact attests to the increasing importance of computational resources in aerodynamic simulation.

1.2 Literature Review

A survey of the modern literature in computational fluid dynamics, with particular attention to the aforementioned subject areas, helps to form a sound basis for devising an efficient and robust solution procedure. A review of three-dimensional Euler and Navier-Stokes computations reveals a number of innovative, cost-conscious methods, as researchers strive to make the most of available resources. A representative subset of these important investigations is presented in section 1.2.1, together with a brief history of computational fluid dynamics methods. The convergence acceleration procedure employed in the present work, multigrid, which receives much attention from applied mathematicians as well as computational fluid dynamicists, is discussed in section 1.2.2. The contributions of the mathematicians to the understanding of and improvements to multigrid complement its more intuitive usage in CFD. Zonal schemes and grid embeddings are a subset of the grid techniques used to further reduce simulation costs and improve the accuracy of flow results. Recent work in this area is detailed in section 1.2.3. Parallelism in computer architectures has provided the stimulus for a plethora of research activities in parallel algorithms and in mapping algorithms to machines and is the focus of the fourth section of the present chapter. Finally, several of the issues addressed in the current work have also been considered by other investigators, and are mentioned where appropriate.

1.2.1 Three-Dimensional Euler and Navier-Stokes Computations

With the increasing availability of computational resources such as the Cray-2, with its massive memory, and the X-MP, with its SSD secondary storage capability, a limited number of simulations of three-dimensional Navier-Stokes flows about complex geometries have been performed. The first viscous flow simulation about a full aircraft configuration (a delta-wing experimental aircraft concept, the X24C-10D), using the MacCormack method, was reported by Shang and Scherr in 1985 [1]. Jameson, Baker, and Weatherill [2] shortly thereafter computed the inviscid flow about a transport configuration that included engine nacelles. Other complete aircraft computations have been carried out. Flores, Reznick, Holst, and Gundy [3] and Reznick and Flores [10] have used a zonal grid scheme to compute separated flow about an F-16 fighter-type aircraft. Yu, Kusunose, Chen, and Sommerfield [11] performed inviscid transonic simulations for a commercial transport. Chaussee, Rizk, and Buning [12] solved the parabolized Navier-Stokes equations for the flow around the space shuttle, and, more recently, compared the numerically generated flow field about the shuttle in ascent configuration with experimental data. Volpe, Siclari, and Jameson [13] have applied a new multigrid method to inviscid fighter-type configurations.

Holst [14] has surveyed recent progress in three-dimensional applications. Work which lays the foundation for computations about complete aircraft configurations includes the investigation of flows about three-dimensional bodies. This is often the culmination of new algorithm development in two dimensions which is extended to three-dimensional simulations. Holst [14] presents estimated memory and execution time requirements for solving the Navier-Stokes equations at various levels of simplification for airfoils, wings, and complete aircraft

configurations using a computer capable of one gigaflop, or one billion floating point operations per second. While the inviscid flow over a complete aircraft would be possible to simulate in minutes on such a system, solutions to the Reynolds-averaged form of the equations would consume hours or even days of CPU time. Large eddy simulations are projected to take years, and the direct simulation of all scales of turbulence with the Navier-Stokes equations in this gigaflop machine would be virtually impractical (i.e., off the scale). It is clear from this that even orders of magnitude (versus merely incremental) increases in speed will not sate the appetite of the computational fluid dynamicist.

Shankar and Chakravarthy [15] have presented results of inviscid cases for multiple three-dimensional aerodynamic bodies. Benek, Buning, and Steger [16] have solved the inviscid flow about a wing/body/tail configuration using a Chimera, or overset, grid scheme. Benek, Donegan, and Suhs [17] then extended this method to viscous flows for a three-dimensional multiple-body configuration. Holst, Gundy, Flores, Chaderjian, Kaynak, and Thomas [18] applied a zonal procedure to the viscous flow over a wing with good results. In Ref. 19, Flores examines and enhances the convergence properties of the zonal scheme.

Complex flow phenomena, such as those which develop over wings at high angles of attack, provide a strong test of any three-dimensional method. Obayashi and Fujii [20] have calculated viscous transonic flow over a swept wing using an LU factored scheme, and Fujii, van Dalsem, Schiff, and Steger [21] have subsequently used this scheme to compute vortex breakdown for a strake-delta wing. Other vortical flow calculations include that of a planar delta wing at high angle of attack [22], unsteady flow about a hemisphere-cylinder [23], the horseshoe vortex around a blunt edge wing on a flat plate [24,25], an inviscid flat

plate [26], and an inviscid cropped delta wing [27]. A survey of high angle of attack, vortex-dominated flow computations may be found in Newsome and Kandil [28].

A slightly more challenging CFD problem, which seems to attract fewer attempts at solution, is three-dimensional transonic internal flow simulation, especially with applications to turbomachinery and turboprop engines. Geometric constraints and the relative motion of blade rows increase the difficulty of numerical solution. An approach taken by Rai, using patched grids in relative motion, has yielded impressive results for rotor-stator interaction [4,29]. Chima [30] has solved a quasi-three-dimensional form of the governing equations with an explicit multigrid scheme for various applications including centrifugal impellers, radial diffusers, and other axial turbomachinery components. Adamczyk and Graham [31] have derived the "averaged-passage" form of the Navier-Stokes equations to facilitate simulation of the flow through a multistage turboprop.

Time-dependent three-dimensional Navier-Stokes flows about oscillating airfoils have been simulated by Nakamichi [32]. Van Dalsem [33] has simulated a jet in ground effect with a crossflow, representative of V/STOL operation, using a fortified Navier-Stokes scheme.

Rotorcraft computations, by their nature, require solution of the unsteady form of the equations of motion. The forces resulting from the interaction of helicopter blades with vortices, in both hover and forward-flight conditions, and the aeroacoustic characteristics of the flow are features desired from a computation. Chen, McCroskey, and Ying [34] have simulated inviscid multiblade rotor flows

using patched grids. Srinivasan and McCroskey [35] investigated tip vortex rollup with a Navier-Stokes solution in quasi-steady conditions. Strawn, Purcell, and Caradonna [36] couple the full potential equations with an integral acoustics code to perform helicopter sound studies.

An area which has lately seen renewed interest is hypersonics. The two primary applications in this flow regime are hypersonic transport vehicles (such as the National AeroSpace Plane) and reentry vehicles. A hypersonic vehicle design cannot be tested at flight conditions using current wind tunnel capabilities; it must, therefore, rely heavily on computational aerodynamics. Li [37] has computed the hypersonic flow about conical aerobrake bodies, Bardina and Lombard [38] have simulated the flow about a reentry vehicle, and Stoufflet, Periaux, Fezoui, and Dervieux [39] have done computations for the Hermes space vehicle using finite elements. Dwoyer and Kumar [40] present an overview and summary of recent progress in the computational analysis of hypersonic airbreathing aircraft flow fields. White and Drummond [41] focus on scramjet applications of CFD. Shang and McMaster [42,43] have investigated the interaction of jets with hypersonic crossflows. Additional recent hypersonic computations are reported in Refs. 44, 45, and 46.

A discussion of Euler and Navier-Stokes computations is incomplete without some mention of the numerical methods for CFD in an historical perspective. The evolution of computational fluid dynamics has been driven by the need to solve flow problems more complicated than those for which analytical solutions can be found. Numerical approximations to the partial differential equations which govern fluid motion have a much wider range of application than do exact solutions of simplified subsets of these equations.

In the early 1900s, Richardson [47] used finite difference operators to solve elliptic equations iteratively. Courant, Friedrichs, and Lewy [48] made a major contribution to the theoretical development of finite difference methods in 1928 when they used finite differences to prove existence and uniqueness theorems for PDEs. In a paper describing the work, they state what is popularly referred to as the CFL condition for stability: that the finite difference domain of dependence must contain the continuum domain of dependence.

In the 1940s, Southwell [49] developed a relaxation method which was performed by hand calculation. Emmons [50-52] obtained the first solutions with embedded shock waves by applying hand-relaxation procedures to transonic problems.

With the introduction of the first electronic computers, interest in the solution of transient problems (those governed by parabolic or hyperbolic PDEs) supplanted the original emphasis on elliptic problems. Von Neumann [53] contributed to the stability analysis introduced by Courant, Friedrichs, and Lewy by establishing stability criteria, and he and Richtmyer [53,54] successfully used dissipative techniques to capture shocks.

In 1954, Lax [55] examined the use of the conservation form of the governing partial differential equations. Controversy developed in the late 1950s with Morawetz's claim [56-58] that shock-free flows about 2-D profiles are in general isolated, thus negatively impacting airfoil design. The consensus was, however, that in a sufficiently large neighborhood of these design conditions, wave drag for shock-free airfoils was negligible.

In 1970, Magnus and Yoshihara [59], following ideas put forth by Crocco [60], obtained solutions to the inviscid steady flow problem by solving the unsteady equations with steady boundary conditions (a purely hyperbolic problem), thereby avoiding the difficulties which arise in solving the mixed elliptic-hyperbolic equations governing steady flow. An alternative approach to this problem was taken by Murman and Cole [61], who used type-dependent differencing with an iterative solution procedure to treat the dual character of the steady equations.

In 1969, MacCormack [62] presented a two-step explicit Lax-Wendroff scheme to solve the time-dependent Navier-Stokes equations, which is now quite popular due to the fact that it has a low operations count and maps well onto many parallel and vector computer architectures.

The Beam-Warming [63] implicit algorithm, introduced in the mid-1970s, eliminated the need to adhere to the CFL stability limitation, as is the case with explicit schemes. This and other implicit methods form the basis for many efficient implementations on scalar computers as well as vector machines.

Around 1980, approximate Riemann solvers [64] began to be popularized as highly-accurate techniques for solving hyperbolic systems of equations. Concurrently, upwind schemes were developed for fluid dynamics applications, and the use of these two related approaches for the solution of flows with strong shocks has since steadily increased [65].

Total variation diminishing (TVD) schemes, developed by Harten [66] and Chakravarthy and Osher [67] in the early 1980s, generate second order accurate

solutions with sharp nonoscillatory approximations to shocks and contact discontinuities for hyperbolic conservation-law systems.

Design applications generally lag research by about ten years. In the 1960s designers used panel methods for solving the linear inviscid equations for shock-free applications. The nonlinear potential equations were implemented as design tools in the 1970s, and presently the Euler formulations are gaining acceptance with the availability of high-speed computers.

Fundamentals for the numerical solution of partial differential equations may be found in Richtmyer and Morton [68], Richtmyer [69], and Mitchell [70], while the applications of such methods to the equations governing fluid dynamics are emphasized in Roache [71], Anderson, Tannehill, and Pletcher [72], and Ames [73], among others.

1.2.2 Multigrid Convergence Acceleration

The use of multigrid (or multiple-grid) schemes to accelerate the convergence of numerical methods for solution of the time-dependent and steady-state forms of the Euler and Navier-Stokes equations began with Ni's work in 1982 [74]. He used a series of coarse grids on which a distribution scheme was applied to accelerate a one-step Lax-Wendroff algorithm for steady-state inviscid problems. Johnson extended this scheme to the entire class of Lax-Wendroff algorithms [75] and to the Navier-Stokes equations as well [76]. Jespersen [77,78] and Jameson [79,80] also investigated the application of multigrid to the solution of the Euler equations. Jameson implemented multigrid using a multistage time-stepping scheme on all grids, with careful use of dissipation operators to ensure

convergence. Stubbs [81] applied Ni's scheme to implicit methods, while Jameson and Yoon [82,83] extended the more classical form of multigrid [84] in the same way. Johnson [85] further modified Ni's scheme to eliminate the computation and storage of the Jacobians with the flux-based scheme, Koeck and Chattot [86] extended multiple grids to inviscid three-dimensional flows, and Johnson and Swisshelm [87] applied multigrid to three-dimensional viscous flows. Jespersen [88] showed that multigrid can be used to accelerate time-dependent solutions. More general information about multigrid may be found in Refs. 85 and 89-93.

Hemker and Spekreijse [94,95] apply multigrid to the solution of the steady Euler equations. This approach differs from Ni's scheme in that the multigrid is used as a fast solver for matrix inversions instead of as an accelerator of a time-stepping solution procedure. Mulder [96] and Shaw and Wesseling [97] have used similar techniques.

Hall [98] has used a cell-vertex multigrid method for acceleration to the steady-state solution of the Euler equations using a Lax-Wendroff scheme, and Salmond [99] has extended this method to the three-dimensional case of a wing geometry.

Perez et al. [100] and Lallemand and Dervieux [101] have applied multigrid to finite element methods in the solution of the two-dimensional Euler equations. Jameson and Mavriplis [102] and Mavriplis [103] have implemented multigrid with an unstructured mesh finite volume scheme with good results.

Parallel multigrid schemes were introduced by Johnson, Swisshelm, and Kumar [104] for solving the Euler and Navier-Stokes equations, and by

Greenbaum [105] for application of classical multigrid. Both techniques are based on decoupling the coarse grids and distributing their computation across multiple processors. Frederickson and McBryan [106] have proposed a multigrid scheme suitable for massively-parallel SIMD systems.

Several multigrid methods have been developed for use in embedded grid refinement solution procedures. Bassi, Grasso, and Savini [107] and Johnson, Swisshelm, Pryor and Ziebarth [108] used this technique to solve two-dimensional Navier-Stokes flows. Johnson and Swisshelm [109], and Swisshelm and Johnson [110] have extended this idea to three-dimensional inviscid and viscous flow computation.

Uses of multigrid in adaptive grid schemes will be further discussed in the next section, including the work of Berger and Jameson [111], Thompson and Ferziger [112], Woodward and Colella [113], and Dannenhoffer and Baron [114].

1.2.3 Embedded Grids and Zonal Schemes

Grid generation for complete configurations is one of the major pacing items in computational fluid dynamics [115]. One means of overcoming the difficulties presented by complex geometries is to partition a computational or physical domain into some reasonable collection of individually-gridded subregions. These types of techniques fall into two main categories: patched grid schemes and overset or overlaid grid schemes. "Patched" implies some kind of alignment between subregions at their shared boundaries, while "overlaid" grids are generally topologically dissimilar with minimal commonality of grid points between subregions.

Patched grid systems may consist of local refinements to some global coarse grid, or two grids which are aligned at their shared boundary and move relative to one another. Usab [116] used a local mesh embedding technique to achieve two-dimensional Euler solutions with a scheme based on Ni's multigrid convergence acceleration method. Holst and coworkers applied grid refinements in the solution of viscous transonic wing flows [18] and simulations for fighter-like configurations [3]. Swisshelm, Johnson, Pryor, and Ziebarth implemented a multigrid embedded scheme for internal flow applications for two dimensions [108] and Swisshelm and Johnson extended it to three dimensions [110].

A variation on the aforementioned scheme is one in which the grid regions are allowed to move relative to one another. Rai has applied this idea with good results for two-dimensional rotor-stator interactions [117] and has extended it to the three-dimensional case including the hub and tip effects common in turbomachinery [4].

Local refinements may also be used in an adaptive sense; that is, as physical flow features evolve, grid points are added during the computation to enhance resolution in selected areas (for example, to resolve viscous effects or to sharpen discontinuities). Berger and Oliger [118] have applied an adaptive method to the solution of hyperbolic partial differential equations, and Berger and Jameson [111] have applied it to the Euler equations. Bell, Colella, Trangenstein, and Welcome [119] have solved a blast wave problem with this scheme. Thompson and Ferziger [112] presented another adaptive multigrid refinement scheme for the incompressible Navier-Stokes equations. Kallinderis and Baron [120,121] have extended Dannenhoffer and Baron's work [114] to include automatic embeddings for viscous and inviscid flows in two dimensions. Bassi, Grasso, and Savini

[107] calculate two-dimensional transonic Navier-Stokes flows using an adaptive method which couples a fine-grid Runge-Kutta scheme with multigrid.

Nakahashi [122] presented a novel approach to the use of patched grid systems in 1986 by using disjoint rectilinear finite-difference meshes around bodies which were connected with finite-element-type point distributions (i.e., triangulations). This enabled the resolution of viscous terms near surfaces for two-dimensional flows while providing the flexibility of a totally unstructured mesh. Nakahashi and Obayashi extended this composite grid scheme to three-dimensional problems including wing/nacelles [123] and multiple bodies [124].

Nakahashi and Deiwert developed a unique approach to grid adaptation based on a spring analogy [125,126], and adaptive finite element grids have been presented in Morgan [127] and Lohner et al. [128,129].

The most common implementation of overset meshes found in the literature is called the Chimera grid scheme [130], used to resolve the flow about multiple-body configurations. Benek, Steger, and Dougherty [131] used such a scheme for the two-dimensional Euler equations applied to a multielement airfoil, and Eberhardt and Baganoff [132] improved the scheme by treating the intergrid boundaries with the method of characteristics. Dougherty, Benek, and Steger [133] applied it to multiple bodies in relative motion (the store separation problem), and Benek, Buning, and Steger [16] extended it to three-dimensional wing/body/tail and multiple-body cases, while Benek, Donegan, and Suhs [134] have computed 3-D viscous flows.

General information on grid generation for CFD may be found in Thompson [135,136].

1.2.4 Parallel Processing

Increased computational capability, in addition to improved algorithm efficiency, is necessary for complex flow simulations. Very large memories and fast processing speeds are required. As the speed of component technology approaches physical limitations [8], advanced architectures become more important in the evolution of computers.

In this section, recent work on those aspects of parallel processing which are relevant to CFD is reviewed. A comprehensive survey of all research in this area, although perhaps interesting to the reader, is beyond the scope of this thesis. For further information, one is referred to the following books. A good overview of parallel-processing concepts is found in Hockney and Jesshope [137]. The material they present is augmented by illustration of the principles as applied to specific architectures. A more hardware-oriented discourse is contained in Stone [138]. Here, the fundamentals of interconnection schemes and memory hierarchies are discussed. The proceedings of any of the numerous annual conferences on this subject, such as the International Conference on Parallel Processing, may be enlightening. Dongarra and Duff [139] have complied a bibliography of advanced-architecture super- and superminicomputers, with summaries of important characteristics and performance.

The main factor affecting multiprocessing, the amount of non-parallelizable work present in an algorithm, was first addressed in equation form by Amdahl

[140], with what has become commonly referred to as Amdahl's law. Worlton [141], in a thoughtful treatise on the subject of parallel processing, has taken Amdahl's relationship and expanded it to take into account inefficiencies that arise in multiprocessing due to synchronization overhead, load balancing, and task overhead. He further contributes to the reader's enlightenment by examining several limiting cases, thus drawing some useful conclusions about the realistic amount of performance to be gained by parallel processing.

Gustafson [142], on the other hand, puts forth the hypothesis that instead of viewing parallel processing as limited by Amdahl's law, the observation can be made that problem size is generally scaled up with increasing numbers of processors. This "scaled speedup" concept assumes that the amount of parallel work in a problem will increase linearly with problem size, while the amount of sequential work remains more or less fixed. Hence, he claims that parallel efficiency should not suffer as much as Amdahl's law would imply.

Martin and Mueller-Wichards [143] present an overview of current practice in supercomputer performance evaluation and propose a set of metrics for the characterization of algorithms and architectures and their interactions, although they do not actually apply these principles. Entire conferences (for example, SIGMETRICS '88 [144]) are currently devoted to the topic of performance evaluation for parallel computer systems.

The first use of advanced architectures for CFD came in the mid-1970s with the ILLIAC IV parallel processor [145] and the STAR-100 vector processor [146]. The ILLIAC IV was an array of 64 processing elements, each with 2048 words of memory and a 240 nanosecond cycle time. The STAR-100, precursor to the

Cyber 205, had a peak speed of 100 Mflops, and high memory bandwidth which was achieved through eight-way interleaving of the memory banks and pipelined transmission of data to the vector processor.

Various researchers have adapted fluid dynamics applications to parallel-processing computers. Those using Cray's multitasking library include Patel, Sturek, and Hiromoto [147], for a viscous axisymmetric ramjet problem, and Swisshelm and Johnson [110] and Swisshelm [148] for a three-dimensional Navier-Stokes zonal multigrid code. Mulac, Celestina, Adamczyk, Misegades, and Booth [149] implemented microtasking on a Cray X-MP to parallelize a turboprop flow solver. Misegades, Krause, and Booth [150] likewise microtask three applications codes: an unsteady viscous nozzle, a three-dimensional viscous wing code, and a three-dimensional Euler solver for a fighter configuration. Greenbaum [105], McCormick and Quilan [151], and Johnson and Swisshelm [109] have investigated the implementation of multigrid methods on multiprocessors.

Very impressive performance results on a massively-parallel distributed system have been obtained by Gustafson, Montry, and Benner [152] for a set of applications, including a flux-corrected-transport algorithm, on a 1024-processor NCube. Measured speedups of over 500, compared to uniprocessor execution times, have resulted in greater optimism regarding the use of massively-parallel systems for real codes. Included in this work is a thorough analysis of important parallel-processing issues.

Others have also successfully used distributed-memory parallel systems for CFD applications. Catherasoo [153] has implemented the vortex method and Bassett and Catherasoo [154] have implemented a finite-volume Runga-Kutta

scheme on an Ametek 2010 hypercube. Erlebacher, Bokhari, and Hussaini [155] simulate compressible transition using a Flex/32. Fatoohi and Grosch use both a Goodyear MPP and the Flex/32 to solve the Cauchey-Riemann equations [156], to test a parallel alternating-direction implicit (ADI) scheme [157], and to simulate two-dimensional incompressible Navier-Stokes flow [158]. Naik and Ta'asan [159,160] also compute two-dimensional incompressible flow using multigrid on a hypercube.

Murman, Modiano, Haimes, and Giles [161] and Modiano [162] have examined the performance of isolated CFD code components on several machines, including the Alliant and the Intel hypercube. Asserting the belief that it is important to optimize the most common CFD loops, they parallelize the pressure computation, the flux summation, and the tridiagonal inversions used for implicit smoothing in convergence acceleration.

Recently, Levit and Jespersen [163] have implemented a two-dimensional compressible Navier-Stokes code (for shear flow computations) on a Connection Machine. This massively-parallel system consists of more than sixteen thousand bit-serial processors connected in a hypercube topology. The small amount of local memory forces grid points to be distributed such that each node contains only a few points. Their implementation of ARC2D requires replacement of the Thomas algorithm for a tridiagonal matrix inversion with odd-even elimination (cyclic reduction) to attain efficient parallel execution. Frederickson and McBryan [106,164] have also achieved high processing rates for a multigrid solver on the Connection Machine.

Several surveys of recent progress in algorithms and applications codes for scientific computing have been written. Ortega and Voight [165] simply collect all vector and parallel algorithm work in bibliographic form. Johnson [9,166] has gathered actual performance data for parallel scientific applications and has attempted to filter it and categorize it in terms of shared and distributed memory systems. He too discusses fundamental concepts relating to Amdahl's law.

Some software tools have been developed to aid parallel algorithm and code development. These include Fortran language extensions such as CoFortran [167] and SCHEDULE [168]. Graphical analysis tools to facilitate debugging multitasked code have been created by Dongarra and Sorensen [168], Seager et al. [169] and others [170].

Research into parallel languages is being conducted. Some examples discussed at a recent conference include Ada, Haskell, LGDF-2 (large grain data flow), Sisal, and Unity [171].

1.3 Scope

The present research entails testing the feasibility of several computational approaches and the suitability of their combinations. Though not an exhaustive investigation, a path was chosen that, in the author's experience, would yield the most useful and promising results. Since explicit finite-difference schemes are quite easy to parallelize, methods of this type are used almost exclusively. However, while efficient multiprocessing is a key consideration in the derivation of a method, algorithmic efficiency is no less important. Thus, the very robust, yet

slowly converging, basic scheme used here is made much faster with the application of multigrid. Awareness of the need to conserve both operations and storage is manifested in the introduction of the zonal equation and embedded refinement aspects of the algorithm. The need for an accurate, stable scheme is self-evident. Hence, the intent is to demonstrate the feasibility of a three-dimensional solution procedure which will continue to evolve - as enhancements are made to algorithms (e.g., TVD schemes), multigrid techniques, and adaptive grid schemes, and as parallel computer architectures are refined and become larger and more powerful.

A set of model problems for two- and three-dimensional testing was chosen to provide some flow complexity (internal problems) without requiring extensive geometry definition or gridding expertise. The emphasis is not fluid phenomena; instead it is on the algorithm - architecture interaction for a three-dimensional flow simulation procedure which is applicable to complex configurations.

Preliminary testing of each component of the algorithm is carried out in 2D, extended to 3D, and then integrated into the 3D solver. Flow results are presented, but the main focus is on performance, i.e., the goal is that the flow results remain invariant while the simulation time is reduced by various techniques employed in the procedure.

This work is presented in the following chapters. First, the equations of motion, which represent the mathematical problem to be solved, will be discussed. The solution scheme, including the basic flow solver, multigrid convergence acceleration, embedded grid refinements, and the zonal application of the equations, and the synthesis of these elements into a unified algorithm is then

described. Next, topics associated with parallel processing and supercomputing are mentioned, with both algorithmic or software considerations and hardware architectures analyzed. Both aerodynamic flow results for the model problem and performance of the algorithm and its parallel implementation are given. Conclusions are then drawn, and, finally, a discussion of future directions for this research is presented.

2. Equations of Motion

While solution to the full viscous form of the equations of motion is ultimately desired, computational capabilities limit the scale of resolution currently possible for three-dimensional flow. The equations used to model aerodynamic flows of interest are often simplified relative to the full Navier-Stokes equations which contain all scales of turbulence. An examination of the actual physics in different regions of a solution domain can reveal that solution to the most complex model equations is not always necessary to capture the essential flow features. This motivates the development of the current flow solver which incorporates several levels of modeling for a single simulation.

Three forms of the equations of motion governing fluid flow in the continuum regime are presented in this chapter. The Reynolds-averaged form of the unsteady Navier-Stokes equations, their thin-layer version, and the Euler equations are each defined and discussed.

2.1 Euler Equations

The Euler equations define the motion of a inviscid, nonconducting fluid, and may be written in conservation form and Cartesian coordinates as

$$q_t + f_x + g_y + h_z = 0$$

where the vector of conserved quantities, q, and the flux vectors, f, g, and h, are defined as:

$$q = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ E \end{bmatrix} \quad f = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ (E+p)u \end{bmatrix} \quad g = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho wv \\ (E+p)v \end{bmatrix} \quad h = \begin{bmatrix} \rho w \\ \rho uw \\ \rho vw \\ \rho w^2 + p \\ (E+p)w \end{bmatrix}$$

The subscripts t, x, y, and z represent partial derivatives in time and space, and the variables ρ, u, v, w, p and E are density, velocity in the x-, y-, and z-directions, pressure, and total energy per unit volume, respectively. It has been shown by Viviand [172] that these equations maintain their strong conservation form under arbitrary transformation of coordinates. Their generalized coordinate form, which is used in the present research, is presented in Appendix A.

The Euler system consists of five equations which represent conservation of mass, momentum, and energy for inviscid flow. The mass conservation, or continuity, equation represents a balance between the change in mass contained in a volume, fixed in space and time, with the mass flux into and out of that volume. Likewise, the three momentum equations represent the same relationship for x-, y-, and z-momentum. The change in total energy in a fixed volume plus the total energy flux must similarly be balanced by the work done by external forces, internal stresses, and heat flux into a volume [72].

The Euler equations in their unsteady compressible form are classified as hyperbolic. Hyperbolic systems may be numerically solved by time marching, or evolution, methods. In their steady form, these equations are classified elliptic-hyperbolic for subsonic flow and hyperbolic for supersonic flow. The system in

this form is difficult to solve, because elliptic equations require a different numerical approach than hyperbolic equations. Hence much of the success in computing steady solutions to subsonic and transonic flow problems has come from solution of the unsteady system. For incompressible flow, the unsteady equations are elliptic-hyperbolic, while in steady form they are elliptic. In contrast with the compressible equations, the steady incompressible system is relatively easy to solve with an iterative (or possibly semi-direct) elliptic solver, while the mixed-type unsteady system is often attacked by using an evolution scheme which solves an elliptic system at each time step, i.e., embedded in the hyperbolic solver.

Additional equations are necessary for closure of the Euler system of equations. Specific internal energy, e, may be related to the pressure and density by assuming a calorically-perfect gas

$$p = (\gamma - 1)\rho e$$

where γ denotes the ratio of specific heats and the total energy per unit volume is given by

$$E = \rho\left(e + \frac{1}{2}(u^2 + v^2 + w^2)\right)$$

to complete the set of equations.

Auxiliary conditions are necessary to complete the specification of an Euler problem in the presence of flow field boundaries. At surfaces, the flow is required to be tangent to the boundary. At arbitrary flow boundaries, such as inlets, exits, or at far-field boundaries, the mathematical properties of the partial differential equations are examined to determine the number of conditions to be specified. The eigenvalues of the coefficient matrices of the partial differential equations are used to determine the characteristic directions. At arbitrary flow boundaries, one condition must be specified corresponding to each incoming characteristic. For the two-dimensional Euler equations, the characteristics are v/u, v/u, $u + c$, and $u - c$. At a flow inlet, subsonic flow (u less than c) requires three conditions and supersonic flow (u greater than c) requires four. At an exit or outflow boundary, one condition for subsonic flow is needed, while zero conditions for supersonic flow are necessary for a mathematically correct problem. As the initial condition, the five independent variables need to be specified over the entire flow domain.

2.2 Navier-Stokes Equations

The three-dimensional Reynolds-averaged Navier-Stokes equations may be written in conservation-law form as

$$q_t + F_x + G_y + H_z = 0$$

where subscripts t, x, y, and z again represent partial derivatives, F, G, and H are given by

$$F = f - Re^{-1}d, \qquad G = g - Re^{-1}r, \qquad H = h - Re^{-1}s,$$

and the vectors q, f, g, h are as previously defined, while d, r, and s are defined as:

$$d = \begin{bmatrix} 0 \\ \tau_{xx} \\ \tau_{yx} \\ \tau_{zx} \\ \beta_x \end{bmatrix} \qquad r = \begin{bmatrix} 0 \\ \tau_{xy} \\ \tau_{yy} \\ \tau_{zy} \\ \beta_y \end{bmatrix} \qquad s = \begin{bmatrix} 0 \\ \tau_{xz} \\ \tau_{yz} \\ \tau_{zz} \\ \beta_z \end{bmatrix}$$

The terms in the viscous flux vectors represent

$$\tau_{xx} = \lambda(u_x + v_y + w_z) + 2\mu u_x \qquad \tau_{xy} = \tau_{yx} = \mu(u_y + v_x)$$

$$\tau_{yy} = \lambda(u_x + v_y + w_z) + 2\mu v_y \qquad \tau_{xz} = \tau_{zx} = \mu(u_z + w_x)$$

$$\tau_{zz} = \lambda(u_x + v_y + w_z) + 2\mu w_z \qquad \tau_{yz} = \tau_{zy} = \mu(v_z + w_y)$$

$$\beta_x = \gamma\kappa Pr^{-1}e_x + u\tau_{xx} + v\tau_{xy} + w\tau_{xz}$$

$$\beta_y = \gamma\kappa Pr^{-1}e_y + u\tau_{yx} + v\tau_{yy} + w\tau_{yz}$$

$$\beta_z = \gamma\kappa Pr^{-1}e_z + u\tau_{zx} + v\tau_{zy} + w\tau_{zz}$$

Note that the subscripts to τ do not represent partial derivatives. The total energy per unit volume is defined as before, as is the specific internal energy, e, from the perfect gas law. The coefficient of thermal conductivity, κ, and the viscosity coefficients, λ and μ, are assumed to be functions only of temperature. Furthermore, λ is expressed in terms of the dynamic viscosity, μ, by invoking

Stokes' assumption of zero bulk viscosity

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

Re and *Pr* denote the Reynolds and Prandtl numbers, respectively.

The following assumptions are made in the derivation of the Navier-Stokes equations: 1) the fluid is a continuum, 2) the fluid is Newtonian (stresses are dependent only on first derivatives of velocity and the fluid is isotropic), 3) the fluid has zero bulk viscosity, and 4) heat conduction behaves according to the Fourier law, i.e., is directly proportional to the first derivative of the temperature.

Additionally, the Navier-Stokes equations are used here in Reynolds-averaged form. By averaging the equations over some time interval, which is long relative to turbulent eddy fluctuations yet short relative to the macroscopic changes in the flow field, additional terms often referred to as Reynolds' stresses, result. The original quantities in the partial differential equation system are transformed to become the mean flow variables. The turbulent shear stresses, representing in some sense a time-averaged transport of momentum and energy [173], must be modeled when the Reynolds-averaged Navier-Stokes equations are used to predict turbulent flow. Following the Boussinesq [174] assumption of an apparent eddy viscosity, the present research uses the two-layer algebraic eddy viscosity turbulence model of Baldwin and Lomax [175], which is patterned after that of Cebeci [176]. No additional partial differential equations are introduced into the system by this model. Information from the mean flow variables is used to algebraically compute μ_T and κ_T, which together with μ and κ form the turbulent viscosity and heat conduction coefficients.

The Reynolds-averaged Navier-Stokes equations may be used to simulate flows with separation, and to determine buffeting and total drag with a sufficiently sophisticated turbulence model [173]. A closer approximation to the full Navier-Stokes equations is achieved with an approach known as large eddy simulation, where large-scale turbulent motion is solved numerically from first principles and only the small, dissipative eddies are modeled [72]. Large eddy simulation requires several orders of magnitude more computing power than the simulation of the Reynolds-averaged Navier-Stokes equations and still must have a sub-grid-scale turbulence model. This model can be eliminated by performing direct simulation of all scales of turbulent motion.

Computational requirements (in terms of memory and speed) for these levels of simulation are presented in Figure 1.1 [5]. The bold line represents what resources are currently available. A performance improvement of 10^3 times will be necessary for large eddy simulations of realistic three-dimensional problems, and a 10^6 -fold increase will be required for simulations which directly resolve all turbulence scales.

The Navier-Stokes equations may be mathematically classified as follows: In their compressible form, the unsteady equations are hyperbolic-parabolic and the steady equations are elliptic-hyperbolic. The incompressible equations are elliptic-parabolic in unsteady form and elliptic in steady form. As previously noted, partial differential equations which change type along *a priori* unknown boundaries in the flow field are more difficult to solve because of the need for type-dependent differencing. The unsteady compressible Navier-Stokes equations are solvable by the same methods as the time-dependent compressible Euler equations.

For a solution to these equations to be properly specified for some bounded domain, additional mathematical conditions must be prescribed. Initial values of the conserved variables (of which there are five in 3-D) must be specified at all points in the field. At flow boundaries, the number of conditions specified is determined by analysis of the characteristics at those boundaries (see Section 2.1). At solid boundaries, heat flux or temperature is fixed, and velocity normal and tangent to the boundary is specified to be zero - the impermeable and no-slip wall conditions. At arbitrary flow boundaries, the number of incoming characteristics determines the number of variables to be fixed. As for the Euler equations, subsonic flow requires specification of three variables at the inlet and one at the outlet, while supersonic inflow and outflow require four at the inlet and zero at the exit.

The thin-layer form of the Navier-Stokes equations can be written in Cartesian coordinates as

$$q_t + f_x + g_y + h_z = Re^{-1} S_z$$

where q, f, g, and h are defined as for the Euler and full Navier-Stokes equations, and S is

$$S = \begin{bmatrix} 0 \\ \mu u_z \\ \mu v_z \\ (\lambda+2\mu) w_z \\ \gamma\kappa Pr^{-1} e_z + (\lambda+2\mu) w w_z \end{bmatrix}$$

The generalized coordinate form of S is found in Appendix A.

In the thin-layer equations, the viscous terms with derivatives parallel to body surfaces are neglected, while all other terms are retained. This is justified by an examination of what is really being computed on a grid that is highly stretched normal to a surface [175]. Gradients of flow quantities are rapidly changing in the boundary layer in the direction normal to the wall. A highly-clustered grid spacing can be sufficient to resolve these viscous terms. Coarse spacing between points in lines or planes parallel to a solid boundary will not resolve the flow gradients in those directions which are generally of small magnitude and thus may be safely neglected. The thin-layer equations have the same mathematical type as the full Navier-Stokes equations, and are therefore solvable by the same methods.

The systems of partial differential equations presented in this chapter may be approximated by a variety of methods; in this work, finite difference schemes applied on a structured, rectilinear grid (or hexahedral in 3-D) are used. The Euler, Reynolds-averaged Navier-Stokes, and thin-layer forms of these equations of motion are solved where appropriate for the physics of a particular flow region in the computational domain. In the next chapter, the details of the numerical flow solver are presented.

3. Numerical Solution Procedure

In order to numerically solve the three-dimensional Navier-Stokes equations efficiently for complex flows, algorithms must be developed that minimize computational work, yet take maximum advantage of the architectural features of currently available and future generation supercomputers. In light of the increasing reliance on multiple processors to gain computational speed, these algorithms should, therefore, be highly parallelizable. With this in mind, the scheme described in this chapter has been created from a combination of numerical solution techniques which are explicit, but that, when used in conjunction with each other, result in a fast and efficient means of obtaining solutions to the equations of motion.

The building blocks of the algorithm are presented in the following sections. The basic fine grid flow solver, an explicit, two-step predictor - corrector scheme, is described, with details such as flow boundary treatment and the implementation of artificial dissipation. A multigrid method is applied to accelerate the convergence to a steady-state solution. The problem of generating appropriate grids for complex flows is addressed through the application of embedded grid refinements. Computational work is further reduced by creating a zonal scheme in which the different levels of governing equations, in the hierarchy from Euler to Navier-Stokes, are applied to different regions of the field. Also discussed are the implementation of intergrid boundary conditions. The assemblage of these elements into a unified solution algorithm concludes the chapter.

3.1 Fine-Grid Flow Solver

MacCormack's explicit predictor - corrector scheme [62] was chosen as the basic flow solver for the present study. It is a widely-used, well-understood, simple and robust two-step Lax-Wendroff algorithm. The intention is not to restrict the solution procedure unnecessarily by this choice, however, and it is conjectured that other methods, such as Runge-Kutta [177] or Beam-Warming [63], could be substituted in its place. Stubbs [81] has shown that multigrid may be applied to an implicit scheme, and Jespersen [88] has used it to accelerate time-dependent calculations. Note that either an implicit or an explicit method could be used as the basic solver, but that explicit schemes are generally more amenable to parallel-processing.

The finite-difference equations, second order accurate in both space and time, may be written as

$$
\begin{aligned}
\Delta q_{i,j,k} = & -\frac{\Delta t}{\Delta x}\left(F^n_{i+1,j,k} - F^n_{i,j,k}\right) - \frac{\Delta t}{\Delta y}\left(G^n_{i,j+1,k} - G^n_{i,j,k}\right) \\
& -\frac{\Delta t}{\Delta z}\left(H^n_{i,j,k+1} - H^n_{i,j,k}\right) \\
\delta q_{i,j,k} = & -\frac{\Delta t}{2\Delta x}\left[\left(F^n_{i+1,j,k} - F^n_{i,j,k}\right) + \left(F'_{i,j,k} - F'_{i-1,j,k}\right)\right] \\
& -\frac{\Delta t}{2\Delta y}\left[\left(G^n_{i,j+1,k} - G^n_{i,j,k}\right) + \left(G'_{i,j,k} - G'_{i,j-1,k}\right)\right] \\
& -\frac{\Delta t}{2\Delta z}\left[\left(H^n_{i,j,k+1} - H^n_{i,j,k}\right) + \left(H'_{i,j,k} - H'_{i,j,k-1}\right)\right]
\end{aligned}
\tag{3.1}
$$

where:

$$\delta q_{i,j,k} = \left[q(t + \Delta t) - q(t) \right]_{i,j,k}$$

$$q'_{i,j,k} = q^n_{i,j,k} + \Delta q_{i,j,k}$$

$$F'_{i,j,k} = F'(q_{i,j,k}) \ , \quad G'_{i,j,k} = G'(q_{i,j,k}) \ , \quad H'_{i,j,k} = H'(q_{i,j,k})$$

The vectors q, F, G, and H have been defined in Chapter 2. Here, t, x, y, and z represent time and the three spatial dimensions, respectively. The superscript n denotes the discretized time level, where $t = n\Delta t$. Likewise, the subscripts i, j, and k determine location in the discretized domain according to $x = i\Delta x$, $y = j\Delta y$, and $z = k\Delta z$, where Δx, Δy, and Δz are the grid spacings in the three coordinate directions. Illustrations of the two- and three-dimensional stencils formed by the scheme are shown in Figure 3.1.

The MacCormack algorithm as represented by Eq. 3.1 is in "forward predictor - backward corrector" form. This means that the spatial derivatives of the flux vectors F, G, and H are forward-differenced in the predictor step and backward-differenced in the corrector step. The order of these two steps, as well as the direction of the discretization of the three spatial derivatives, may be alternated to prevent asymmetry arising from one-sided differencing, although this was not found to be necessary for the present applications. In order to maintain second order accuracy, the derivatives of the viscous terms in F, G, and H must be carefully discretized [72]. The x-derivatives contained in F are differenced in the opposite direction of F_x (i.e., backward in the predictor and forward in the corrector in the current implementation), while the y- and z-derivatives in F are approximated with central differencing. Likewise, y- and z-derivatives contained

in G and H, respectively, are differenced opposite to G_y and H_z, while central differencing is applied to the cross-derivative terms.

The MacCormack scheme can be derived from the Taylor series expansion

$$q(t + \Delta t) = q(t) + \Delta t q_t + \frac{\Delta t^2}{2} q_{tt} + \cdots$$

keeping the appropriate terms to insure second order accuracy in time. Spatial second order accuracy is attained with the appropriate differencing of the derivatives in x, y, and z. The complete derivation is contained in Appendix B.

The flow solver is stable only for time steps (Δt) less than the limit imposed by the Courant-Friedrichs-Lewy (CFL) condition [178]. The ratio of time step size to mesh spacing is restricted in such a way that the domain of dependence of a point in the difference scheme must contain all points of its domain of dependence in the differential equation [179]. For the three-dimensional case, this limit may be expressed as

$$\Delta t = \left[\frac{|u|}{\Delta x} + \frac{|v|}{\Delta y} + \frac{|w|}{\Delta z} + a \left[\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2} + \frac{1}{\Delta z^2} \right]^{-1/2} \right]^{-1}$$

The derivation and generalized coordinate form of this equation are contained in Appendix C. The variable a represents the local speed of sound, and u, v, and w are the three components of velocity. The viscous time step limit is approximated by neglecting the convection terms in the Navier-Stokes equations and performing a von Neumann stability analysis on the linearized equations [180]. This limit may be written as

$$\Delta t_v = \left[\frac{1}{2} Pr\ \mathrm{Re}\right]\left[\frac{\gamma\,\mu}{\rho}\left(\frac{1}{\Delta x^2}+\frac{1}{\Delta y^2}+\frac{1}{\Delta z^2}\right)\right]^{-1}$$

The variables μ, γ, and ρ represent dynamic viscosity, the ratio of specific heats, and density.

Once the time step limit has been determined at each grid point as the minimum result from the two equations above, the following means of advancing the solution are available. If maintaining time accuracy at each iteration is desired, a global time step equal to the minimum Δt in the field is used at all points. When a steady state solution to the unsteady equations is needed, the requirement for time accuracy is relaxed, and the local time step at each grid location may be used. This second technique removes the stiffness in the system introduced by the stretched mesh and accelerates the convergence to a steady state solution because larger time steps may be taken in areas of the mesh where grid spacing is large. Local time stepping is used to obtain all flow results presented in this work.

For transonic flow computations, artificial damping is added as a fractional step to the fine-grid solver so that shock discontinuities may be properly captured. The additional terms introduced into the equations are of second order globally but are locally first order where the density gradient is large. Numerically, they have the form:

$$\Delta q_D = \Delta t \nu_D \left[\Delta x^2(|\rho_x|q_x)_x + \Delta y^2(|\rho_y|q_y)_y + \Delta z^2(|\rho_z|q_z)_z\right]$$

The partial differential equation system is thus modified to be

$$q_t + F_x + G_y + H_z + \nu_D \left[\Delta x^2 (|\rho_x| q_x)_x + \Delta y^2 (|\rho_y| q_y)_y + \Delta z^2 (|\rho_z| q_z)_z \right] = 0$$

The coefficient ν_D is order 1. This equation is only applied during the computation of supercritical flows. Its generalized coordinate form is presented in Appendix D.

On very fine grids, such as those generated for the three-dimensional viscous flow applications, additional dissipation is necessary to stabilize MacCormack's method. The very fine mesh spacing admits very high frequency error modes which are not adequately damped by the simple artificial viscosity terms presented above. Terms representing fourth partial derivatives of the conservation variables in each direction are therefore added to the solution. This has the effect of modifying the partial differential equation to have the form

$$q_t + F_x + G_y + H_z + \mu \left[\Delta x^3 q_{xxxx} + \Delta y^3 q_{yyyy} + \Delta z^3 q_{zzzz} \right] = 0$$

The cubes of Δx, Δy, and Δz give the artificial terms proper dimensionality. Through a von Neumann stability analysis [181], it has been shown that $\mu = 1/16$ gives zero amplification of the highest frequencies in the solution for a representative one-dimensional, linearized model problem.

The above expression for the artificial viscosity transformed into generalized coordinates may be found in Appendix D. It is applied as another fractional step after the MacCormack scheme, sweeping in each of the three spatial directions. The coefficient μ often needs to be tuned for grids with different mesh spacing,

since the model problem analysis gives an optimum value of μ only for a very restricted, one-dimensional case, and does not provide a good general guideline for multidimensional cases. This tuning is achieved through computational experimentation in the present research.

As noted in Chapter 2, along viscous solid wall boundaries the no-slip condition is applied, temperature or heat flux specified, and pressure determined from the normal momentum relation. This last equation, formed by combining the three transformed momentum equations, is written in generalized coordinate form as

$$
\begin{aligned}
&- \rho U(\zeta_x u_\xi + \zeta_y v_\xi + \zeta_z w_\xi) - \rho V(\zeta_x u_\eta + \zeta_y v_\eta + \zeta_z w_\eta) \\
&= (\xi_x\zeta_x + \xi_y\zeta_y + \xi_z\zeta_z)p_\xi + (\eta_x\zeta_x + \eta_y\zeta_y + \eta_z\zeta_z)p_\eta + (\zeta_x^2 + \zeta_y^2 + \zeta_z^2)p_\zeta
\end{aligned}
$$

and is solved implicitly for pressure in the streamwise (ξ) direction along each wall. Any other values not specified by the mathematical boundary conditions are computed with an explicit method, that is, extrapolation from the interior of the domain.

The surface pressure boundary condition, as shown above, is the only non-explicit calculation in the current algorithm. By solving the pressure relation implicitly, symmetry of the solution (for a symmetric flow) is preserved along the boundary. This condition can, in fact, be replaced by an equivalent explicit discretization of the partial differential equation, thus removing the recursive implicit computation, which inhibits the vectorization of this boundary condition. The two-dimensional code has been thus modified and tested, and is discussed in Chapter 7.

3.2 Multigrid Convergence Acceleration

Although MacCormack's method is efficient and quite robust for solving the time-dependent equations of motion, its convergence to steady-state solutions is slow due to its explicitness and CFL-limited time-step size. One means of increasing the rate of convergence is to apply multigrid in the manner first introduced by Ni [74] and modified and extended by Johnson [75,76]. This multigrid algorithm may be used for Euler, thin-layer, and full Navier-Stokes computations. By solving Lax-Wendroff-type equations for δq (the change in the solution at each time step) on a collection of grids coarser than the base grid, and communicating the results back to the base grid in the form of an additional update to the MacCormack solution, convergence of the fine-grid scheme is accelerated.

The actual mechanism by which multigrid improves the convergence to steady state of Euler and Navier-Stokes solution procedures has been explained in two ways. It is postulated by some [75,76] that the use of coarse grids facilitates the propagation of transients out of the computational domain. Others [182] have shown that the use of multigrid serves to damp out lower frequency errors in the solution by virtue of the fact that the coarse grids see these modes more easily than the fine grids. One could also conjecture that some combination of these two theories are the basis of the success of the multigrid technique, and investigation on this subject is ongoing [183].

3.2.1 Coarse-Grid Schemes

The multigrid method was originally devised as a way to lower the number of operations (or in other words, the cost) needed to compute a steady-state solution

on a machine with serial, von Neumann-type architecture. The sequential algorithm uses a series of successively coarser grids, created from some base grid by deleting every other point in each coordinate direction, to propagate information from the fine-grid solver throughout the computational domain. The number of points in each direction on the base grid is required to be expressible as $n(2^p) + 1$ for p a natural number and n an integer greater than or equal to 2, where p is the number of grid coarsenings, and n is the number of coarsest-grid intervals [184].

The coarse grid scheme is derived from a one-step Lax-Wendroff method written in a coarse grid setting. Such a scheme may be expressed as

$$\delta q_{coarse} = \Delta t q_t + \frac{\Delta t^2}{2} q_{tt}$$

Recalling that $q_t = -(F_x + G_y + H_z)$, this may be rewritten as

$$\delta q_{coarse} = -\Delta t(F_x + G_y + H_z) - \frac{\Delta t^2}{2}(F_x + G_y + H_z)_t$$

Observing that

$$\Delta q = -\Delta t(F_x + G_y + H_z)$$

we obtain the equation

$$\delta q_{coarse} = \Delta q - \frac{\Delta t^2}{2}(F_x + G_y + H_z)_t \tag{3.2}$$

Various coarse-grid schemes may now be derived according to the way in which the second term on the right hand side of Eq. 3.2 is treated. If we let it be approximated as

$$-(F_x + G_y + H_z)_t \cong \Big[A(F_x + G_y + H_z)\Big]_x + \Big[B(F_x + G_y + H_z)\Big]_y + \Big[C(F_x + G_y + H_z)\Big]_z$$

where A, B, and C are the Jacobian matrices,

$$A = \frac{\partial F}{\partial q}, \quad B = \frac{\partial G}{\partial q}, \quad C = \frac{\partial H}{\partial q},$$

we obtain the class of Jacobian-based acceleration schemes, of which the method due to Ni [74] is a member.

For example, a simple discretized Jacobian-based scheme for a coarse grid having double the spacing of some fine grid may be written as:

$$\delta q_{i,j,k} = \frac{1}{8}\left[(I + \frac{\Delta t}{\Delta x}A + \frac{\Delta t}{\Delta y}B + \frac{\Delta t}{\Delta z}C)\Delta q\right]_{i-1,j-1,k-1} \tag{3.3}$$

$$+ \left[(I + \frac{\Delta t}{\Delta x}A - \frac{\Delta t}{\Delta y}B + \frac{\Delta t}{\Delta z}C)\Delta q\right]_{i-1,j+1,k-1}$$

$$+ \left[(I + \frac{\Delta t}{\Delta x}A - \frac{\Delta t}{\Delta y}B - \frac{\Delta t}{\Delta z}C)\Delta q\right]_{i-1,j+1,k+1}$$

$$+ \left[(I + \frac{\Delta t}{\Delta x}A + \frac{\Delta t}{\Delta y}B - \frac{\Delta t}{\Delta z}C)\Delta q\right]_{i-1,j-1,k+1}$$

$$+ \left[(I - \frac{\Delta t}{\Delta x}A + \frac{\Delta t}{\Delta y}B + \frac{\Delta t}{\Delta z}C)\Delta q\right]_{i+1,j-1,k-1}$$

$$+ \left[(I - \frac{\Delta t}{\Delta x}A + \frac{\Delta t}{\Delta y}B - \frac{\Delta t}{\Delta z}C)\Delta q\right]_{i+1,j-1,k+1}$$

$$+ \left[(I - \frac{\Delta t}{\Delta x}A - \frac{\Delta t}{\Delta y}B - \frac{\Delta t}{\Delta z}C)\Delta q\right]_{i+1,j+1,k+1}$$

$$+ \left[(I - \frac{\Delta t}{\Delta x}A - \frac{\Delta t}{\Delta y}B + \frac{\Delta t}{\Delta z}C)\Delta q\right]_{i+1,j+1,k-1}$$

If this represented a single-grid Lax-Wendroff scheme, Δq would be computed from

$$\Delta q_{i+1,j+1,k+1} = -\Delta t(F_x + G_y + H_z)_{i+1,j+1,k+1}$$

For multigrid, instead of solving this equation by discretizing the flux balance on the coarse grid, we approximate Δq with a restriction, R, of the most recently computed fine-grid correction

$$\Delta q_{coarse} \cong R(\delta q_{fine})$$

The effect of this restricted fine-grid correction is then distributed according to Eq. 3.3 to obtain a coarse-grid correction. This correction, in turn, is prolonged to the fine grid, to become an additional update to the fine-grid solution. In two dimensions, Figure 3.2 contrasts the one-step scheme written on a fine grid with the coarse-grid scheme written on a grid with double the spacing.

We observe that in the basic integration scheme a correction at one grid point affects only its nearest neighbors, while in a k-level multigrid scheme the same correction affects all points up to $2^{(k-1)}$ mesh spacings distant. Furthermore, since the coarse-grid scheme simply propagates the effects of the fine-grid corrections, the fine-grid solution accuracy is maintained.

The Jacobians, A, B, and C, are five-by-five solution-dependent matrices needed at each grid point and each time step and are, therefore, costly to compute and store. To increase the efficiency of the multigrid scheme, Johnson [85]

has developed a flux-based coarse-grid scheme which does not use the Jacobian matrices. If we let the right hand side of Eq. 3.2 be approximated by

$$(F_x + G_y + H_z)_t \cong \frac{1}{\Delta t}\left[(F_x + G_y + H_z)^{n+1} - (F_x + G_y + H_z)^n\right]$$

thus yielding the equation

$$\delta q_{coarse} = \Delta q - \frac{\Delta t}{2}\left[(F_x + G_y + H_z)^{n+1} - (F_x + G_y + H_z)^n\right]$$

where

$$F^n = F(q^n)\,, \qquad G^n = G(q^n)\,, \qquad H^n = H(q^n)$$

$$F^{n+1} = F(q^n + \Delta q)\,, \quad G^{n+1} = G(q^n + \Delta q)\,, \quad H^{n+1} = H(q^n + \Delta q)$$

we obtain a class of flux-based coarse-grid schemes. The quantity Δq is again approximated by a restriction of the fine-grid value of δq and second-order accurate spatial differencing is used. The computation and storage of the five-by-five Jacobians has been replaced by spatial differences of the flux vectors, thus reducing the cost of the multigrid scheme.

3.2.2 Information Flow

The information flow of the sequential multigrid scheme is depicted in Figure 3.3. First, a fine-grid correction, δq_1, is computed. Then δq_1 is restricted to the next-coarser grid, where δq_2 is computed. The δq_2 correction is both restricted

to grid *3* and prolonged to grid *1*, where it provides an additional update to the fine-grid solution. On grids *3* through *N-1* the procedure is analogous to that on grid *2*. When δq_N has been computed and prolonged to grid *1* to provide the N^{th} update to the fine-grid solution, the next multiple-grid cycle is ready to begin. Injection is used as the restriction operator for the sequential coarse-grid scheme, and linear interpolation is used as the prolongation operator.

The parallel coarse-grid algorithm [104,184], illustrated by Figure 3.4, removes the dependence of grids *3* through *N* upon their immediate predecessors, and creates a scheme which is particularly suitable for implementation on MIMD-type computer architectures. In contrast to the sequential multigrid algorithm, δq_1 is now restricted to each of grids *2* through *N*. All of the coarse grids may then be updated simultaneously and independently of each other. Averaging is used in the restriction operator for grids *3* through *N*, while the prolongation operator remains linear interpolation. Testing has demonstrated that the convergence rate of the parallel coarse-grid algorithm is nominally the same as that of the sequential scheme.

3.2.3 Implementation Details

Dissipative effects have a local character and their influence need not be taken into account in the construction of coarse-grid schemes. Rather, it is the convective terms, with their global character, that are the key element in coarse-grid propagation. Hence, coarse-grid schemes for viscous flow computations may be formulated on the basis of the inviscid equations of motion [76]. Such a scheme leads to a multiple-grid convergence acceleration procedure which is independent of the nature of the dissipative terms retained in the viscous model

equations. In other words, a coarse-grid scheme based on the Euler equations may be employed, without modification, to accelerate the convergence of viscous flow computations based on the Navier-Stokes equations, the thin-layer equations, or any other viscous model equations which contain the full inviscid Euler equations. The flux-based coarse-grid scheme may be written in convective form as follows:

$$\delta q_{coarse} = \Delta q - \frac{\Delta t}{2}\left[(f_x + g_y + h_z)^{n+1} - (f_x + g_y + h_z)^n\right]$$

This version of the coarse-grid scheme is used in the present computational algorithm.

Boundary conditions are enforced only on the fine grid. This has the advantage of decoupling the coarse-grid scheme from both the physical and numerical nature of these boundary conditions. That is to say: the coarse-grid scheme always sees a Dirichlet problem. Likewise, any numerical damping terms which may be necessary are also only applied on the fine grid. This enhances the modularity of the coarse-grid scheme.

3.3 Embedded Grid Refinements

In order to properly capture the physics of the flow field with a discretized solution procedure and a limited number of grid points, these points must be clustered in certain regions. Although the Euler equations, which include only inviscid terms, may be sufficiently resolved on a fairly coarse mesh, the full

Navier-Stokes equations require very fine mesh spacing to resolve the viscous terms (which are gradients of velocity gradients) near surfaces.

One way to cluster grid points in specific regions of the flow is to use stretching. Applying this technique to distribute grid points may be as simple as algebraic or geometric calculations or as complicated as solving elliptic or hyperbolic partial differential equations. Although grid stretching is used to some extent in the present work, it has limitations and is not always sufficient for applications with complex geometries. Meshes which are highly stretched in one coordinate direction and relatively coarse in another may have cells with high aspect ratios, which can have a destabilizing effect on a numerical solution algorithm. Clustering is difficult to achieve in the presence of complex domain boundaries, and concentrating enough points for high resolution in one area may unnecessarily add points elsewhere in the field.

Alternative methods for optimizing the grid point distribution in a complex flow field include overlaid or Chimera grids [133], zonal or patched grids [18,185], and the use of unstructured meshes [103]. A Chimera scheme has local meshes, each tailored to conform to one of the bodies in a multibody configuration, overlaid on some global mesh. This technique has been used successfully to resolve the flow about multi-component airfoils and to compute time-dependent store separation problems in two dimensions [133]. Three-dimensional applications include flow simulation about wing/tail/body and multiple-body configurations [16]. Zonal or patched grids cover a computational domain with only minimal overlap at interfaces between regions, such that the information transfer between grid zones is less complicated than with the Chimera scheme. Moving patched

grids have been used extensively by Rai [185] to simulate time-dependent rotor - stator interaction in two and three dimensions. Holst et al. [18] have used a zonal mesh arrangement, similar in some respects to the technique of this work, for an internal flow computation. Unstructured grids have been applied to successfully compute the inviscid flow about a commercial transport [2], and their accuracy as compared to that of structured meshes is currently being studied [186, 187].

We have chosen to explore the feasibility of introducing embedded refinements into a global (and relatively coarse) mesh, in the same manner as presented by Usab [116] or Kallinderis and Baron [120], as one method of creating acceptable grids for complex geometries. The locations of the embeddings are defined *a priori*, drawing upon our experience with similar flow solutions. This process, of course, could be automated, and also could be used in an adaptive grid scheme (see, for instance, Berger [188]). In the current algorithm, each grid embedding is required to contain as a subset any underlying coarser grid points. In other words, refinements are formed by successive halving the grid spacing in each coordinate direction. This is illustrated in Figure 3.5 for a simple two-dimensional domain.

Extensive testing has been performed on the two-dimensional test case [108] using the grid arrangement depicted in Figure 3.5. The model problem consists of a sting-mounted blade in a channel. A global coarse grid was defined, and the first embedding placed to facilitate shock capture in supercritical flow, while a further refinement was positioned along the viscous wall boundary for high resolution of the boundary layer.

A simple embedding arrangement has been created to test the zonal grid refinement scheme in three dimensions. A single refinement region is defined, which covers the area above the airfoil to the exit, and spans the entire transverse direction. In this problem, the intergrid boundary conditions, work reduction by grid point elimination, and the zonal application of the Euler/Navier-Stokes solver can be studied without the additional complexity of managing many refinement regions. This 3-D model problem is illustrated in Figure 3.6.

The three-dimensional application which has motivated the development of this algorithm, a model of the geometry of a turbomachinery blade row, is illustrated in Figure 3.7. Highest resolution of the flow solution is required along the blade surface and in the corners created by the juncture of the endwalls and the blade. Here, three-dimensional viscous effects dominate the flow, and the viscous terms in the equations of motion require high resolution and therefore a very fine grid (one which would be equivalent to a 129x33x33 global mesh). Figure 3.8 shows the positioning of the embedded refinements for a typical complex 3-D computation. Figure 3.8*a* depicts the global coarse grid (33x9x9), and 3.8*b* and *c* show the two subgrids, each refinements of the global grid by factors of two and four in all dimensions, respectively. Figure 3.9 presents the assemblage of this set of grids. Further details about the test problems are contained in Chapter 5.

The boundaries between the grid regions require particular attention. Injection (i.e., merely using the underlying fine-grid information without any weighting) is used to transfer information from the fine grid to the coarser regions. However, for communication in the direction from coarse to fine, a more complicated transfer is necessary. Interpolation is the simplest way to fill in the fine-

grid boundary points. To maintain conservation at these intergrid boundaries, however, the coarse-grid interior points which lie on the fine-grid boundary must be corrected by performing a flux balance of the surrounding cells. This technique has been used with some success in two dimensions [116], but a derivation of the corresponding 3-D implementation shows that it requires an inordinate amount of additional floating point operations, and, although applied only at a small percentage of the total number of grid points, does not appear to be cost effective. A third possible treatment of these points is to overlap the fine and coarse regions by more than one row or plane, so that the actual boundaries lie farther in the interior of neighboring regions. In the present work, interpolation has been used to transfer information in the coarse to fine direction.

3.4 Zonal Application of Flow Solver

Given a set of grid embeddings such as described in the previous section, a procedure has been developed by which the appropriate flow equations are solved in each region of the computational domain. The flow physics and the coarseness of the grid in each zone provide the guidelines for determining which set of equations are appropriate.

The full Reynolds-averaged Navier-Stokes equations are needed to resolve three-dimensional viscous effects, such as those arising in corners where the blade and endwall meet. The thin-layer equations suffice in regions where diffusion processes parallel to a body surface may be neglected (assuming that the grid is body-conforming). In grid zones far from viscous boundaries, where the flow is inviscid, the Euler equations are applied. Computational costs decrease (because

fewer terms need to be computed) as one moves through this hierarchy from full viscous to inviscid. Additionally, convective terms can be sufficiently resolved on a coarser discretization than, for instance, diffusive terms in the boundary layer. These two considerations are combined to reduce computational expense substantially.

MacCormack's method is applicable to all zones of the flow field. Since the convective coarse-grid implementation is used, the multigrid scheme may be applied without modification to both viscous and inviscid flow zones.

A natural coupling of the zonal scheme to the use of embedded fine grids exists. Since high resolution is required for the full Navier-Stokes equations, it makes sense to solve them only in the finest grid regions. Thin-layer equations, applicable to viscous wall boundary regions, are solved on the appropriately-refined grids near surfaces. The inviscid Euler equations are well-suited for regions of the flow field away from surface boundaries where no refinements have been made to the grid.

3.5 Algorithm Assembly

The flow field updating begins with mesh *1*, the finest embedding. After one time step on mesh *1*, mesh *2* is updated exterior to mesh *1* while convergence acceleration is applied at the mesh-*2* points interior to mesh *1*. Next, mesh *3* (the global coarse grid) is updated exterior to mesh *2* while convergence acceleration is applied at the mesh-*3* points interior to mesh *2*. Updating proceeds in this fashion until the global mesh has been advanced one time step. Then,

convergence acceleration is applied to coarsenings of the global grid. The appropriate equations are solved in each region, as described in section 3.4, while convergence acceleration uses only the inviscid terms. This cycle is repeated until the desired measure of convergence is satisfied.

Figure 3.10 shows an example, for a typical 2-D implementation, of the computational steps and grid regions involved in one cycle through the algorithm. Part *a* shows the grid regions. Where grid points/lines are shown, either the MacCormack scheme or multigrid is applied: MacCormack only in Part *a*; MacCormack interior to the bold line (intergrid boundary) and multigrid exterior to the bold line in Parts *b* and *c*; and multigrid only in Part *d*.

The numerical solution procedure described in this chapter combines an explicit scheme with several execution- and convergence-acceleration techniques to yield a robust solver with wide applicability to viscous and inviscid internal flows. It is particularly well-suited to parallel-processing supercomputers. Extension to flows with more complicated physics and geometries is expected to further demonstrate the efficiency and flexibility of the procedure. Automated adaptive gridding, better zonal interface boundary treatment, and further investigation of artificial dissipation operators would also contribute to improving the scheme, and some of these additions and modifications are planned. The results of computational experimentations using this algorithm are presented in Chapter 5. Next, parallel processing considerations are discussed.

4. Parallel Processing

In an effort to build faster, more powerful computational engines, computer architects are increasingly incorporating parallelism in their designs. This is necessitated by the current development stage of scalar or von Neumann-type computers, in which components are approaching their technological performance limits [8]. To effectively use multiprocessing devices for the numerical simulation of aerodynamic flows, the scientist must understand both the architecture of these machines and the methods by which their parallelism is optimally exploited. Unfortunately, the development of software tools such as compilers and languages lags the pace of hardware innovations, so that elegant, portable expression of parallel algorithms is not yet possible.

In this chapter, parallel computer hardware and software will be analyzed, and the impact of each on algorithmic considerations for concurrent computing will be illustrated. Details about current-generation supercomputers and the programming techniques employed to exercise their full capabilities, such as vectorization and multitasking, will be discussed. Typical formulae for evaluating parallel-processing performance will be presented. In concluding this chapter, we will look to the future by examining both next-generation supercomputers as well as some present-day advanced-architecture superminicomputers, which may determine the direction large-scale computing will take in the next decade.

4.1 General Considerations

Parallelism can be achieved at many levels within a computer system, some of which are transparent to the user while others require a nontrivial programming effort for their effective use. With a sound understanding of architectural issues, available software tools can be intelligently implemented, and algorithm strategies can be devised to maximize the performance of the supercomputer.

4.1.1 Parallel Computer Architectures

Computers may be classified architecturally according to a taxonomy developed by Flynn [189]. A purely scalar computer, in which instructions are issued sequentially and operate on single data elements, is a single-instruction stream, single-data stream (SISD) machine. Vector and array processors, which apply single instructions to vectors or arrays of data, are classified single-instruction, multiple-data (SIMD) computers. Systems consisting of multiple processors which may operate independently of one another are referred to as multiple-instruction, multiple-data (MIMD) machines. SIMD computers either use vector instructions to operate on large data sets in a manner similar to an assembly line (or pipeline), as on a Cyber 205, or they distribute data across a large array of less-powerful processors which then execute instructions in lock-step, as on an MPP or Connection Machine. On MIMD computers, separate processors may execute different instructions simultaneously.

Parallelism in computer architectures may be present at a number of levels [137,190], which complicates the seemingly straightforward classification scheme set forth by Flynn. At the highest level in a multiprocessor system, different jobs

(i.e., entire programs) may be executing at the same time on the various processors. In a multitasking environment, individual programs may use more than one processor at once. At a lower level, instructions may be partitioned over several functional units, or the functional units themselves may have parallel or pipelined steps. At the lowest level, bit manipulation can be performed in parallel. Computer designs generally incorporate some subset of these ideas in hardware. In this work, we are primarily concerned with taking advantage of parallelism at the individual program level (i.e., the level which the user sees), referred to as multitasking.

Several categories of parallel processing computers, including both SIMD and MIMD types, are in evidence among currently-available hardware. Key characteristics of these systems are: the number and power of central processing units (CPUs), the location of memory, and the nature of the connection scheme between the CPUs and memory [166]. Computers generally have either a few powerful processors or many weak ones, since the extreme cost of building a system with many powerful processors is prohibitive. An example of the former type of system is the Cray X-MP/48, with four vectorizing CPUs, each of which is more powerful than Cray's previous supercomputer, the Cray-1. A machine such as the Connection Machine is representative of the "many weak" type, having as many as 2^{16} single-bit processors which all execute the same instruction.

The main (or primary) memory in a multiprocessor computer may either be distributed among processors (a loosely-coupled system) or shared, with equal accessibility, by all processors (a tightly-coupled system). With either type of memory implementation, additional levels of storage are generally present in the machine. Registers and cache memory are relatively small in size, closely

associated with individual processors, and have the highest access speeds. Typically cache memory, with an access time on the order of one clock cycle, is 4 to 20 times faster than main memory [138]. A shared main memory may be partitioned into banks to improve the rate of access to data. Retrieving a string of data from many different banks can be done faster than accessing it all from a single bank (where memory cycle or refresh time would limit the frequency at which each datum can be obtained). The largest main memories currently range in size up to two gigabytes. Slower, larger storage devices such as an SSD (Solid-State Storage Device) are used to increase the memory capacity of the machine by as much as four additional gigabytes. These various levels of memory hierarchy are illustrated in Figure 4.1.

Interprocessor communication is an important feature of any multiprocessor system. Some connection scheme possibilities include busses and switching networks such as full crossbars and hypercubes [138]. A bus has the simplest topology for linking processors to memory but has the highest potential for becoming a system bottleneck. Insufficient bandwidth can result in contention for the bus, reducing system performance. A full crossbar switch, on the other hand, minimizes the possibilities for contention but becomes increasingly complex as the number of processors grows. The crossbar directly connects every processor with every memory module, so that contentions for the interconnect are essentially nonexistent. For large numbers of processors the full crossbar switch is prohibitively expensive. While busses and crossbars lie at opposite ends of the spectrum of interconnection schemes, switching networks, such as the butterfly or perfect shuffle, and multiprocessors linked together in a hypercube lie somewhere in between. Processors in a hypercube are directly connected only to a subset of the other processors, and access the rest via message-passing (or other forms of

routing) through intermediate processors. Figure 4.2 shows three of the most common interconnection schemes: the bus, the full crossbar switch, and the hypercube.

Although the cost of multiprocessor systems grows linearly with the number of CPUs (assuming the relative cost of the interconnect to be negligible, a very conservative assumption for this comparison), the performance grows more slowly. For a massively-parallel system to be cost effective, problem and program parallelism must be fully exploited and efficiently mapped onto the architecture of the machine. The software tools that will enable such implementations are still in early stages of development and will be discussed in sections 4.2 and 4.3.

4.1.2 Adapting Algorithms to Parallel Architectures

In a discussion of parallel processing applied at the program level, the term process or task refers to a unit of computation that may be executed in parallel with or independently of one or more other sections of code. In general, these separately-running processes need to exchange information among themselves, which leads to synchronization concerns, communication management, and overhead costs. The effects of load balancing, granularity, overhead, and Amdahl's law are all important factors affecting parallel-processing performance.

To analyze the performance of a computer having two distinct speeds of execution, Amdahl [140] developed the model

$$R(f) = \frac{1}{(1 - f) / R_H + f / R_L}$$

where the resulting system rate, *R(f)*, is determined by the fraction of results *(1-f)* generated at the high rate, R_H, and the remaining fraction, *f*, generated at the low rate, R_L. For a multiple-processor system, the high rate is simply the number of processors, *P*, multiplied by the low rate. Amdahl's law is then written

$$R(f) = \frac{R_L}{(1 - f) / P + f}$$

If the execution time, *T*, is defined as the number of results generated at a given execution rate, and speedup (S_P) is defined as the ratio of execution time on a single processor (T_1) to that on the multiprocessor system (T_P), then

$$S_P = \frac{T_1}{T_P} = \frac{R(f)}{R_L} = \frac{1}{(1 - f) / P + f}$$

From this relation, it can be seen that the fraction of residual sequential code, *f*, limits possible speedup as *P* gets large. This equation is illustrated graphically in Figure 4.3.

Worlton [141] uses Amdahl's law to examine the effects of overhead, granularity, and synchronization requirements. In his interpretation of the model, *N*, *t*, t_s, and t_o represent, respectively, the number of tasks between synchronization points, average task execution time (granularity), and task overhead time due to parallel execution. The single-processor execution time is then

$$T_1 = Nt$$

Assume, for simplicity, that the number of tasks is evenly divisible by the number of processors and the tasks are all equal in length (i.e., assuming a perfect load balance), so that each task takes $(t + t_o)$ time to execute, and t_s time is required for synchronization. The total parallel execution time is then

$$T_P = t_s + \frac{N(t + t_o)}{P}$$

The speedup for this case is given by

$$S_P = \frac{T_1}{T_P} = \frac{Nt}{t_s + N(t + t_o) / P}$$

The best possible speedup on P processors is obviously equal to P. (This excludes the case of superlinear speedup, which, while plausible for certain algorithms such as tree searching, does not appear to have applicability to the numerical solution of partial differential equations in fluid dynamics.)

Efficiency is defined as

$$E_P = \frac{S_P}{P} = \frac{1}{(t_s P / Nt) + (1 + t_o / t)}$$

so that one hundred percent efficiency is equivalent to the maximum speedup, P, attainable on P processors.

If we neglect synchronization time to examine the relative effects of granularity and overhead size, the equation for efficiency can be rewritten

$$E_P = \frac{1}{(1 + t_o / t)}$$

so that the ratio of overhead to granularity must be very small for good efficiency, or, in other words, task granularity must be much larger than task overhead.

If, on the other hand, we examine the effect of synchronization time while neglecting the task overhead, the equation becomes

$$E_P = \frac{1}{(Pt_s / Nt + 1)}$$

so that synchronization overhead is minimized by increasing task length.

Several conclusions can be drawn from the above model. The performance of a parallel-processing computer is constrained by the fraction of the work that must be done in sequential mode, which limits the attainable speedup as the number of processors is increased. This fraction of sequential code can result from the mathematical model or the algorithm [142]. Synchronization and task overhead effects can be reduced by increasing the granularity of tasks, while better load balancing is achieved by having the number of tasks be equally divisible by the number of processors and all of equal length.

There are several ways in which an algorithm may be partitioned. Functional decomposition of a problem is appropriate when physical processes are evolving on different length or time scales. An example of this is the parallel coarse-grid scheme, in which the various grid levels have been decoupled so that they may be

computed concurrently. Another functional decomposition can be achieved by distributing each equation of a system of partial differential equations to a different processor. Spatial decomposition is appropriate for flow solvers in which groups of points can be viewed as being independent of one another on a given time level. Line implicit and explicit schemes for solving partial differential equations fall under this category. It may even be possible to formulate temporal or phase decomposition schemes where, as the solver finishes a time step on one part of the grid, the next time step could be started there, while the computation of the previous time step is being completed elsewhere in the field. These techniques may be combined to create an optimal partitioning scheme for a particular parallel-processing architecture.

Exploitation of the parallel-processing capabilities of a system requires either extensions to an existing language such as Fortran or development of an entirely new language. Merely relying on compiler sophistication is insufficient to harness the full power of the hardware for most applications. Although language extensions are necessary for the short term (and very common among computer vendors), their diversity on various systems makes it impossible to write transportable parallel code. No consensus has been reached thus far regarding the form multitasking (or, for that matter, vectorization) should take. This is evident in some of the examples in the following two sections.

4.2 Vectorization

Vector computers achieve higher performance than scalar or sequential computers by pipelining identical, yet independent, DO loop level operations on large

data sets. These data sets, when stored in some regular arrangement in memory, are referred to as vectors, and the pipelining technique is called vectorization. All current-generation supercomputers used in the present work have this capability.

Vectorization has as its goal the production of at least one result per CPU clock cycle per vector pipeline. Since most operations take some multiple of clock cycles to complete, functional units are divided into single-cycle stages and operations performed in an assembly-line fashion to achieve this end. This also implies that data must be fetched from memory at the rate of at least one or two operands per clock cycle and be stored at the same rate. This is often accomplished by partitioning the memory into phased banks, such that successive banks may be accessed one cycle apart for vector operations.

Vectorization is performed at the DO loop level in Fortran. Loops with a regular (i.e., constant) stride are most amenable to this technique. Some barriers to the vectorization include [137]: conditional and branch statements, recursive computations, nonlinear and indirect addressing, and subroutine calls within loops. In order for a compiler to automatically recognize vectorizable loops, moderate to extensive code restructuring is usually required, ranging from simple reordering of array indices or lines of Fortran, to implementation of explicit gather/scatter instructions to create the vectors, to algorithm modification to eliminate recursive computations.

Vectorizing Fortran code for the Cyber 205 involves the following. First, array indices are ordered such that the dimension of the leftmost index is largest and the rightmost is smallest. Likewise, DO loops are arranged such that the innermost corresponds to the first (leftmost) index, etc. These loops can then be

vectorized by the 205 compiler if the data are stored contiguously, the loops have stride one, and none of the aforementioned barriers is present in the loop. For cases where the data are not stored contiguously, or the loops have a nonunit stride, explicit vector Fortran syntax must be employed to achieve vectorization [191]. (At this point the code loses its portability.) For loops computing values at a high percentage of the locations in a vector or array, the WHERE block construct is appropriate. A bit vector mask with length equal to that of the operand vectors is created, containing ones denoting locations where new values are desired and zeroes which mask out other locations. Computations are carried out for the entire vector, and then results are stored only in locations corresponding to the one bits in the masking vector. Vector intrinsic functions may be used to gather or compress data into contiguous memory locations when relatively few of the vector or array values are to be modified. After the vector operation, the results are either scattered or expanded and merged into appropriate storage locations.

The techniques used to facilitate automatic vectorization on the Cyber 205 are also applicable on Cray computers. Additionally, the Cray compiler is capable of vectorizing constant, nonunit stride DO loops. Directives may be inserted in the code, without destroying its portability, to instruct the compiler to ignore apparent dependencies in order to vectorize computations with complicated array indices.

The following are examples of vectorization techniques and implementation taken directly from the three-dimensional code used for the present research.

The basic flow solver, MacCormack's explicit predictor - corrector scheme, has as its computational kernel a set of unit stride nested DO loops, the easiest and most efficiently vectorized construct for the Cyber 205. Note the ordering of indices and loops.

```
      DO 800  KE = 1, 5
      DO 800  JY = 2, NM
      DO 800  KZ = 2, LM
      DO 800  IX = 2, MM
      DQ(IX,KZ,JY,KE) = 0.5E0 * ( -DTAU(IX,KZ,JY,1) *
     *         ( (FCQ(IX,KZ,JY,KE) - FCQ(IX-1,KZ,JY,KE) )
     *         + (GCQ(IX,KZ,JY,KE) - GCQ(IX,KZ,JY-1,KE) )
     *         + (HCQ(IX,KZ,JY,KE) - HCQ(IX,KZ-1,JY,KE) )
     *         - (SSQ(IX,KZ,JY,KE) - SSQ(IX,KZ-1,JY,KE) ) )
     *         + DELQ(IX,KZ,JY,KE) )
      QQ(IX,KZ,JY,KE) = QQ(IX,KZ,JY,KE) + DQ(IX,KZ,JY,KE)
  800 CONTINUE
```

Since the above equations are applied only to the interior of the computational domain, thus creating gaps in an otherwise contiguous storage arrangement (i.e., at the boundary locations), only the innermost loop vectorizes automatically. In order to create a computation where the vector length is on the order of the number of points in the domain (i.e., m^n) instead of just the dimension in one direction (order (m)), the WHERE block control structure provided on the Cyber 205 is employed. A bit mask vector is used to control the updating of data, limiting it to the interior, while the computation was carried out for all points in the domain. The syntax follows.

```
      DESCRIPTOR BCORD
      BIT BCORD
       (assign BCORD)
      DO 800  KE = 1, 5
      WHERE ( BCORD )
         DQ(2,2,2,KE; ILC) = 0.5 * ( -DTAU(2,2,2,1; ILC) *
     *        ( (FCQ(2,2,2,KE; ILC) - FCQ(1,2,2,KE; ILC) )
     *        + (GCQ(2,2,2,KE; ILC) - GCQ(2,2,1,KE; ILC) )
     *        + (HCQ(2,2,2,KE; ILC) - HCQ(2,1,2,KE; ILC) )
     *        - (SSQ(2,2,2,KE; ILC) - SSQ(2,1,2,KE; ILC) ) )
     *        + DELQ(2,2,2,KE; ILC) )
         QQ(2,2,2,KE; ILC) = QQ(2,2,2,KE; ILC) + DQ(2,2,2,KE; ILC)
      ENDWHERE
  800 CONTINUE
```

The WHERE block construct is too inefficient for the multigrid computations, which are performed at less than 25 percent of the grid points in the three-dimensional case. Gather and scatter instructions are more appropriate for such a sparse computation. Their use is shown here.

```
      DO 800  KE = 1, 5
      VQQ((KE-1)*IP1+1; IP1) = Q8VCMPRS (QQ(1,1,1,KE; IMLN),
     *        BCG1((IGD-2)*IMLN+1; IMLN); VQQ((KE-1)*IP1+1; IP1))
  800 CONTINUE
       (multigrid computations with VQQ)
      DO 900  KE = 1, 5
      VDQ2((KE-1)*IMLN+1; IMLN) = Q8VXPND (VDQ2((KE-1)*IP1+1; IP2),
     *        BCG2((IGD-2)*IMLN+1; IMLN); VDQ2((KE-1)*IMLN+1; IMLN))
  900 CONTINUE
      DO 910  KE = 1, 5
      DQ(1,1,1,KE; IMLN) = Q8VCTRL (VDQ2((KE-1)*IMLN+1; IMLN),
     *        BCG2((IGD-2)*IMLN+1; IMLN); DQ(1,1,1,KE; IMLN))
  910 CONTINUE
```

On the Cray, the prolongation step of the multigrid algorithm contains no barriers to vectorization, but has too many complicated variable indices for the compiler to decode and analyze for dependencies, so it fails to vectorize. By inserting a directive [192] the programmer can force the compiler to vectorize the code in spite of the apparent vector dependencies.

```
      INTL = IGD - 1
      DO 720  KZ = KKB, KKE, IGS
      DO 720  I = IIB, IIE, IGS
      IIVL = IGS
      IGJ = 2 * IGS
      DO 720  III = 1, INTL
      IIVL = IIVL/2
      IGJ = IGJ/2
      JEE = N - IGJ
      DO 720  J = 1, JEE, IGJ
CDIR$ IVDEP
      DO 720  K = 1, 5
      DQ(I,KZ,J+IIVL,K) = 0.50 * (DQ(I,KZ,J,K) + DQ(I,KZ,J+IGJ,K))
  720 CONTINUE
```

The parallel coarse-grid algorithm has been vectorized in a similar fashion, with the additional feature of combining the points to be updated on grids 2 through N into one long vector. This minimizes the vector startup overhead and thus further improves the speed of the algorithm on an SIMD computer. The performance of the vectorized code is presented in Chapter 5 for both two- and three-dimensional test cases.

4.3 Multitasking

As stated previously, parallel-processing involves a collection of separately-running processes, called tasks, running on multiple CPUs. Multitasking, a term popularized by Cray, describes the technique by which software tools are used to manage multiple processors operating on a single program. Creation, synchronization, and termination of processes, and the communication among them are the primary functions needed to create and execute parallel programs. These capabilities can be provided either by extensions to the Fortran language or by the inclusion of precompiler directives on the Cray systems used in the present research.

To properly multitask an applications code, an analysis must be performed to determine data dependencies, parallel structures, and shared variables present in the solution procedure. Data dependencies arise in situations where one process reads shared data that another process writes. The dependency is satisfied when the latter process has completed writing the data.

A task graph (also called a data dependency graph) can provide a way of visualizing the task and data interdependencies. It is constructed by defining parallel structures or processes and representing them as a graphical objects, and then connecting these objects with lines depicting the data flow in the program. A sample task graph, describing the three-dimensional multigrid MacCormack scheme, is presented in Figure 4.4. Large-granularity tasks have been constructed to minimize the overhead incurred by library calls used in multitasking. The most important task groups to note in the figure are those corresponding to the basic fine grid scheme and the parallel multigrid scheme, examples of spatial and functional decomposition, respectively, and the data dependencies.

Extensions to programming languages which allow the creation and termination of processes, synchronization of processes, and communication among them have been developed for multiprocessing on Cray supercomputers. Both the Cray-2 and the Cray X-MP have software libraries of multitasking tools. Two variations of multitasking are available on the X-MP, namely, macrotasking [193] and microtasking [150]. Macrotasking is intended for application to large-grained problem partitioning, while microtasking, by virtue of its very low overhead, may also be used efficiently at a fine-grained level. Microtasking has only been recently developed for the Cray-2. A careful examination of the problem is necessary in order to define large code structures suitable for parallel execution when

using macrotasking. Microtasking, on the other hand, is easily implemented in a code which has been optimized for vectorization as well as a code which has been previously macrotasked. Neither of these tools results in code which can be run in parallel mode on any other system, though. For this reason, other extensions to Fortran for use on parallel machines have been developed. The SCHEDULE package [168] and CoFortran [167] language extensions have been written to enable the execution of parallel algorithms on a variety of shared-memory MIMD systems without code modification. The library routine calls used by each are included, along with the Cray multitasking tools, in Appendix E.

The sequential multigrid algorithm contains many opportunities for creating small granularity parallelism but relatively few opportunities for the sort of large granularity necessary to produce good multitasking speedup in the face of non-trivial multitasking overhead. This observation, together with the desirability of non-sequential multigrid schemes for reasons of algorithm flexibility, led to the construction of the two-dimensional parallel multigrid algorithm described in Chapter 3. In this algorithm, grids which are independent of one another may be updated simultaneously on separate processors. In fact, such a simple strategy may result in a poor load balance across processors because of the different amounts of work inherent in updating grids of different coarseness. However, more refined strategies are possible. Grids may, for example, be grouped together into tasks of approximately equal work, or they may be melded into tasks with other large-grained multitaskable code segments in order to equilibrate processor loading. Notice further that, by multitasking large-grained structures, the vectorization potential of code within these structures remains intact.

Given that MacCormack's method and the multigrid scheme are both explicit, parallel processing may also be employed without drastic algorithmic modifications. In the three-dimensional case, spatial decomposition is the method by which parallelizable tasks are created. The computational domain, a grid with dimensions 129x33x33, is partitioned by planar slices in the lengthwise channel direction, as shown in Figure 4.5. This direction is chosen so as to preserve vectorization potential in the inner DO loops, which execute along lines parallel to the partition boundaries (the x-direction). Some overhead is introduced to enable flexible task management through variably-sized partitioning. Redundancy is also added to the computational scheme along the boundaries between tasks to eliminate the need to protect shared data from being updated more than once per time step in those regions.

Detailed results of performance studies using macrotasking and microtasking are presented in Chapter 5.

4.4 Performance Evaluation

The absence of a good general means of quantifying parallel-processing supercomputer performance is an area of concern for the computational scientist. The peak megaflop ratings of a system are of little worth unless one is interested solely in matrix multiplications (or whatever the peak-speed operation of a particular machine may be). However, there are a few basic measurements that can be made to determine at least the relative improvement in execution time and efficiency obtained by vectorization and multitasking. For single-processor vector computers, speedup is the comparison between CPU time for scalar versus vector

modes. For computers with multiple CPUs, speedup quantifies the relative wall clock time for uniprocessor versus multiprocessor execution on a dedicated machine. A general equation for speedup is then

$$S_P = T_1 / T_P$$

where T represents time, and the subscript P refers to the number of processors.

Given this definition, we can also find the efficiency resulting from multiple processing, simply by dividing the speedup by the number of processors

$$E_P = S_P / P$$

The efficiency is not so clearly definable on vector processors as it is a function of startup time, vector length, and vector pipe length.

If the percentage of sequential code is known, the theoretical speedup can be calculated from Amdahl's law and compared to the actual speedup measured on the system. This will quantify the amount of overhead due to task startup and synchronization costs.

Another criterion for performance evaluation is the number of floating-point operations per second (usually expressed in megaflops/second) executed by the computer on a given application. The peak Mflop rate, seldom attained by the user, is not a particularly useful quantity. Achieving peak speed requires keeping all functional units on all available processors busy without pausing for any synchronization or communication.

The desire to determine a more realistic (i.e., sustainable) Mflop rating leads to the use of benchmarks, or suites of programs which are meant to (hopefully) represent a typical workload on a computer. Benchmarking codes range from simple matrix manipulations, to compute-intensive kernels extracted from applications codes, to full-blown numerical simulations. Widely-used benchmark suites include LINPACK [194], the Livermore Loops [195], and the NAS Kernels [196]. When benchmarking a parallel computer, these programs must often be recoded to include parallel Fortran extensions or rewritten in an entirely different language in order to be able to measure the multiprocessing system performance.

The study of parallel-processing efficiency raises a peripheral issue which is sometimes used as an argument against applying multiple processors to a single application. Multitasking reduces system throughput by introducing additional overhead and synchronization costs (since, in general, parallelization results in less than 100% efficiency) which are not incurred by a uniprocessor run. However, in order to advance the frontiers of computational science and to solve problems that are not tractable on single-CPU systems because of prohibitively long execution times, parallel-processing must be exercised [197]. A study by Bieterman [198] has actually shown that throughput can even be improved by multitasking on a system that is constrained by memory size.

4.5 Current-Generation Supercomputers

The following are classified as current-generation supercomputers: Cyber 205, Cray X-MP, Cray-2, Fujitsu VP-100/200/400, Hitachi S810/820, and NEC SX2/A2. A supercomputer is defined as one of the fastest computers presently

available, a floating definition. The characteristics of the Japanese supercomputers [199] (the Fujitsu, Hitachi, and NEC machines) are summarized in Table 4.1, and are included in this section for completeness, although no calculations were performed on any of these systems. All are vector SIMD computers.

The three machines to be discussed in detail in this section, the Cyber 205, the Cray X-MP, and Cray-2, have all been used extensively in carrying out the current research. Their important features are recorded in Table 4.2.

4.5.1 Cyber 205

The Control Data Cyber 205 [200] is a single CPU vector-processing supercomputer. It was first commercially marketed in 1981, and can be configured with one, two, or four vector pipes and one to 32 million 64-bit words of main memory. The 205 has a CPU clock cycle time of 20 nanoseconds and memory access cycle time of 80 nanoseconds. Vector operations are performed by fetching data directly from memory and returning the result back to memory, without the use of any intermediate registers. These operations require that vector elements be stored in contiguous locations. The virtual memory of the machine allows paging of blocks of memory to and from a secondary storage level, which allows programs that use more than the available main memory of the machine to be executed without explicit user intervention.

In order to be able to fully exploit the vector-processing capability of the Cyber 205, CDC has provided enhancements to the Fortran language in the form of explicit vector syntax [191]. Also furnished are language extensions that permit the efficient management of the paging that arises in the use of virtual memory.

Some of the advantages of the 205 relative to other supercomputers include its bit manipulation capabilities, use of virtual memory, and good performance for applications using long vector computations. However, the 205 is relatively inefficient for short vector operations, and the need to manage virtual memory to prevent thrashing can become a nontrivial task. Additionally, creating long vectors with elements in contiguous memory locations generally requires the use of nonstandard Fortran syntax which adds significant effort to the development of a code. Results of vectorized two- and three-dimensional versions of the multigrid algorithm, presented in Chapter 5, illustrate the relative merits of relying on the compiler versus investing a great deal of individual effort into enhancing performance.

4.5.2 Cray X-MP

The Cray X-MP [201] can be configured with one, two, or four vector-processing CPUs and between two and sixteen million 64-bit words of shared main memory. The four-processor X-MP/4 can have four to sixteen Mwords of ECL (emitter-coupled logic) bipolar memory, with an access cycle time of 34 nanoseconds, while the main memories of the X-MP/1 and 2 systems use MOS (metal oxide semiconductor) technology, with an access cycle time of 68 nanoseconds. The X-MP/1 and 2 were introduced in 1982 with a CPU clock cycle time of 9.5 nanoseconds, and were followed in 1984 by a four-processor system. An 8.5 nanosecond clock machine has also been developed, and lately an extended-architecture (EA) version of the X-MP, which allows the use of a larger main memory, has been introduced.

To vectorize computations, the Cray moves data from main memory into registers before performing the calculations, in comparison with the direct memory-to-memory mode for vector operations on the Cyber 205. Each X-MP CPU has four parallel memory ports: two for vector fetches, one for vector stores, and one for independent I/O operations [201] to provide high bandwidth between memory and processors. A cluster of registers in the system is dedicated to the sole purpose of efficient interprocessor communication and control.

Vectorization of constant-stride DO loops is done automatically by the compiler, a process which can be either aided or inhibited by user intervention in the form of compiler directives. Parallel-processing is achieved through macrotasking or microtasking as is discussed in section 4.3. In the former, extensions to the Fortran language in the form of multitasking library calls are used, while the latter is accomplished through the use of preprocessor directives.

The X-MP has the following advantages over other supercomputers. Its optimal vector length is 64, so that relatively short vectors perform just as well as long ones. It has up to four processors which gives it potentially four times the single-CPU performance. Microtasking is provided as a low overhead, easily-implemented tool for parallel processing. The requirement of contiguous storage of vector values is overcome by the use of vector registers. On the other hand, main memory size is not exceedable, as there is no virtual memory, which restricts problem size. The use of an SSD (Solid-State Storage Device) somewhat alleviates this restriction, although its use is not transparent to the user. The macrotasking version of multitasking has relatively high overhead and requires fairly extensive code restructuring for its efficient use.

4.5.3 Cray-2

The Cray-2 [202], introduced in 1985, has the largest shared main memory of any supercomputer (268 million 64-bit words) and the fastest CPU clock speed (4.1 nanoseconds). It is a four-processor system, cooled by immersion in an inert liquid. Each processor has a peak speed of 488 Mflops, with the total system rated at 1.9 Gflops. The processors share one memory port, and main storage is configured in 128 banks.

The large primary memory size of the Cray-2 is its main advantage over other current-generation supercomputers. It also possesses a small cache memory which has much faster access than main memory. The UniCos operating system is also considered by many to be a favorable characteristic. Although it possesses very fast CPUs, instructions are issued only once every two clock cycles, and for most applications its computational speed is equivalent to that of the Cray X-MP on a per-CPU basis. Multitasking is currently less efficient as implemented on the Cray-2 than on the X-MP due to the different hardware designs of the machines, the most notable of which is the single path from CPUs to main memory on the Cray-2. The large memory has a fairly slow access cycle time, on the order of 80-120 nanoseconds, which generally leads to a higher incidence of memory bank conflicts in many computer codes, impacting both single- and multiprocessing execution rates.

4.6 Next-Generation Supercomputers

The important (relevant) characteristics of the supercomputers which comprise the "next generation," that is, those having a significant performance increase over currently-available machines, are summarized in Table 4.3 [203].

Cray has recently delivered its first Y-MP [204], a follow-on to the X-MP line. The Y-MP initially has a maximum of eight processors with 6 nanosecond clock cycles, with one instruction issue per clock cycle. Peak speed is expected to be 2.5 gigaflops, with capability of one gigaflop sustained. It will have 32 million 64-bit words of main memory with 30 nanosecond access time, plus up to 512 megawords of auxiliary memory in the form of an SSD. Main memory size may increase in later models as higher density chips become available. In preliminary benchmark tests on the Y-MP, the NAS kernels [196] have attained twice the speed of a single CPU of the X-MP.

The Cray-3, successor to the Cray-2, will have 16 processors, each of which will be two to three times faster than a single Cray-2 processor [205]. This high performance will be partially attributable to gallium arsenide technology used in the processor chips to achieve a 2 nanosecond CPU clock cycle time. Peak speed is purported to reach 16 gigaflops. The Cray-3 will have 512 million 64-bit words of fast static RAM as main memory. Initial deliveries are expected as early as 1990.

The ETA-10, successor to the CDC Cyber 205, is available with as many as eight processors, each up to 2.5 times as fast as the 205 [206]. Several systems have been delivered. The fastest eight-processor system is purported to have a

CPU clock cycle of 7 nanoseconds, with liquid-nitrogen cooling, and with four megawords of memory local to each processor and a 256-megaword shared secondary memory. Peak speed of the full-blown system in 64-bit mode may reach four gigaflops.

The Japanese supercomputer vendors also have plans for next-generation supercomputers. Hitachi will have a single-processor two gigaflop system and NEC is planning a multiprocessor to attain peak speeds in excess of twenty gigaflops for 1989 [203].

4.7 Advanced-Architecture Machines

While supercomputer designers are cautiously moving toward more highly-parallel systems, many of the superminicomputer vendors are already supplying machines with massive parallelism. Although individual processors of these systems are not nearly as powerful as, for instance, a single CPU of the Cray-2, the fact that they can be focused in large numbers on a single computation has great potential. Their usefulness in large-scale scientific computing depends on the success with which applications and algorithms can be efficiently partitioned and parallelized.

Three types of advanced-architecture minisupercomputers are briefly described here. The first example is a distributed-memory, SIMD system. The Connection Machine 2 [207] consists of a massive array of bit serial processors (up to 2^{16}) connected in a hypercube topology. Each processor has its own memory (on the order of 32Kbits). Instructions are issued from a front end

computer such that all processors operate in lockstep. This computer has the potential for speeds on the order of several gigaflops, and promising results for a CFD application have already been obtained [163].

A second class of architecture is the shared-memory, MIMD system. The Alliant FX/8 falls under this category [208]. Although the X-MP and Cray-2 are also of this type, the less-powerful minisupercomputer is distinguished by its more sophisticated software environment and lower cost. The Alliant's Fortran compiler automatically multitasks code in much the same way as the Crays vectorize, with minimal interaction from the programmer taking the form of compiler directives. Its performance on selected loops extracted from CFD applications ranges from 15 to 23 Mflops on an eight-processor system [161].

The majority of distributed-memory MIMD computers (the third example of a parallel computer architecture) can be categorized as "hypercubes." The hypercube class of multiprocessors originated with the Cosmic Cube project at Caltech [209]. Current commercial systems include the Intel iPSC/2 [210], the Ametek 2010 [211], and the NCube [212]. Bassett and Catherasoo [154] have demonstrated reasonable performance of the Ametek system on one of Jameson's "FLO" codes [177], and a CFD package has developed for the iPSC [210].

The most notable work to date for a massively-parallel computer, however, has been carried out by Gustafson, Montry, and Benner [152] on the Ncube. Using a 1024-processor configuration, they were able to attain measured speedups greater than 400 with three applications codes, including a flux-corrected transport CFD algorithm.

Also proposed in Ref. 152 is an alternative performance measure for parallel, distributed-memory computers. Determining the actual speedup is restricted by the limited memory available on any one processor of a massively parallel system. That is, the largest problem for which performance can actually be measured (by comparing T_1 and T_P) is determined by the size of problem that fits on a single node. Gustafson et al. claim that as more processors are applied to a computation, problem size should grow correspondingly (i.e., the computational scientist will want to use all available resources). They postulate that the serial portions of an algorithm remain fixed in size under such a scenario, and hence the effect of residual sequential code on speedup is not as strong as Amdahl's law would predict.

As more and more computational fluid dynamics applications are restructured and implemented on advanced-architecture superminicomputers, a variety of the above-mentioned classes of parallel-processors may emerge as the most appropriate for certain classes of simulations. Investigations in this area can help determine the most promising directions for future supercomputer design. The many factors contributing to the performance of these machines must be carefully considered and weighed against one another in order to provide the maximum possible computing resource to the scientist. In the following chapter, the results of the current investigation implementing a complex CFD algorithm on shared-memory SIMD and MIMD systems are presented.

5. Computational Results

Given the solution procedure and parallel-processing considerations described herein, a set of model problems has been generated to test the efficiency and robustness of the algorithm. They are representative of internal flow problems, such as those which arise in turbomachinery blade rows and wind tunnels.

5.1 Problem Specification

For two-dimensional computational experiments, the basic geometry consists of a cascade of bicircular arc blades at zero angle of attack and no stagger. The three-dimensional model contains a bicircular arc blade mounted between endwalls at variable sweep angle. These problems allow the investigation of accuracy, stability, and performance of the algorithm without presenting unnecessarily complex grid generation requirements.

These particular geometries have been selected because internal flow problems often present more of a challenge to the computational fluid dynamicist than external flow problems. This is evidenced by the lag in the development of internal flow codes compared to those for external applications and is due in part to the more restrictive geometries and boundary conditions. The transients in an external flow problem are typically allowed to radiate to infinity in

all or almost all directions, while the transients in an internal flow problem are constrained to radiate out small inlets and exits. Gridding is also more difficult for internal flows because the stretching is more complex, due to the need to resolve flow near a number of surfaces and also to have good resolution in the interior of the domain.

The two-dimensional computational domains for both viscous and inviscid problems are illustrated in Figure 5.1. A bicircular arc blade is located one chord length from both the inlet and the exit. Channel height is also equal to one chord length. Symmetry permits limiting the computational domain to that shown by the dashed lines in the figure.

Physical boundary conditions for two-dimensional inviscid flow cases are also included in Figure 5.1. At the inlet, total temperature, total pressure, and flow direction are specified. The flow tangency condition is enforced at the upper and lower boundaries. At the exit, static pressure is fixed.

For two-dimensional viscous flow computations, the geometry of the domain is the same as that used for the inviscid cases. The problem of flow past a sting-mounted bicircular-arc airfoil is constructed by imposing the appropriate viscous boundary conditions. Inlet, exit, and upper wall boundaries have the same conditions as the inviscid case. Along the lower boundary, however, the airfoil and sting are treated as viscous walls at which the no-slip condition and temperature or temperature gradient are specified.

The computational domains used for the three-dimensional model problem are presented in Figure 5.2. As in 2-D, the domain is intended to represent the

geometry of a turbomachinery blade row or cascade. The sweep angle of the blade is permitted to vary from 0 to 26 degrees relative to the normal of the flow direction. The blade, at 0 degrees sweep (i.e., perpendicular to the incoming flow direction), is positioned one chord downstream from the inlet with its trailing edge one and a half chords upstream of the exit. This positioning ensures that, as the sweep angle is increased to 26 degrees, the blade will remain at least one chord upstream of the exit. The channel height and width are each equal to one chord length.

Boundary conditions for the three-dimensional test problems are also shown in Figure 5.2. For inviscid flow, the inlet total temperature, total pressure, and the angle of the incoming flow are fixed. At the exit, static pressure is fixed. Along the front, back, top, and bottom walls, the tangency condition is imposed.

For the viscous problem, the inlet and exit conditions remain the same, while the front and back boundaries are treated as viscous walls (representing the endwalls to which the blade is attached in the cascade), and, as such, the no-slip condition and temperature or temperature gradient are specified there. Along the upper boundary flow tangency is imposed. Along the blade surface and downstream along the lower boundary to the exit, the viscous wall conditions of no-slip and the temperature or temperature gradient are specified.

The Reynolds number of the flows computed span the range from 8.4×10^3 to 2.0×10^5 based on cascade gap and critical speed. Airfoil thickness is varied from 10% of the channel height to 0% (which represents a flat plate). The angle of attack in all cases is zero.

5.2 Grid Definition

A fairly coarse global mesh is used as the base grid, and all grid regions are supersets of it. It is generated algebraically. The two-dimensional mesh contains lateral grid lines which are smoothly stretched away from the lower wall boundary in a geometric progression. Transverse grid lines are equispaced over the blade and then stretched away from the leading and trailing edges to the inlet and exit, respectively. Typical 2-D grids are illustrated in Figure 5.3. Although the figure shows either 65x17 or 129x33 points on the grids, computations have also been performed on grids dimensioned with 65x33 points. Initial spacing in the normal direction near walls is varied as appropriate for viscous and inviscid wall boundary conditions and flow cases.

The three-dimensional grids are constructed by simply stacking the two-dimensional grid planes. These planes are clustered near the front and back walls of the computational domain. Figure 5.4 shows the inlet, lower- and rear-wall grids for typical three-dimensional inviscid (airfoil) and viscous (flat plate) problems. Calculations have been carried out on meshes dimensioned 65x9x17 (the largest that would fit in main memory on the Cyber 205) and 65x17x17 and 129x33x33 (on the Cray X-MP and Cray-2). The latter two are displayed in Figure 5.4. Using this notation, the first dimension represents the number of points in the lengthwise or x-direction, the second, the transverse or z-direction, and the third, the vertical or y-direction.

5.3 Algorithm Performance

Numerous computational experiments have been carried out for both two- and three-dimensional test cases to measure the robustness and the speed of the various components of the algorithm presented here. The data obtained enable an examination of accuracy, stability, convergence behavior, and vectorization and multitasking efficiency. In addition to testing the performance of each element of the solution procedure individually, the fully-integrated multitasked, vectorized, embedded zonal multigrid three-dimensional scheme has been tested on the Cray-2. Results are presented here in the form of flow visualizations, discussions of accuracy and stability, convergence comparisons, and tabulated speedup and efficiency measurements.

5.3.1 Flow Results

Since the primary goal of this research is to improve the speed at which a given three-dimensional flow solution can be computed, one measure of success is that all flow results should be independent of the variant of algorithm being applied. Any discrepancies must be resolved. With this in mind, typical results for two- and three-dimensional cases are presented as a baseline.

Figures 5.5 and 5.6 show isomach contours on a 129x33 grid for the following two-dimensional flow cases: inviscid subcritical flow over a 10% thick circular arc blade at freestream Mach number 0.5, and inviscid shocked supercritical flow over the same blade with a freestream Mach number of 0.675. Figures 5.7 and 5.8 show velocity profiles from a 65x33 grid for viscous separated laminar flow over a 5% thick circular arc blade and viscous separated turbulent flow

over the same blade. We note here that these converged steady-state flow results are identical (to the precision of the computer being used) for both unaccelerated MacCormack and MacCormack with multigrid solution procedures. Likewise, the use of vectorization and multitasking do not alter the flow results in any of these cases.

Figures 5.9 and 5.10 depict corresponding inviscid results obtained with the two-dimensional embedded grid scheme. These cases are computed with two levels of grid refinement and with only the Euler equations solved in all regions. The global coarse grid is dimensioned 33x9; the refined region near the airfoil has spacing identical to a 129x33 grid. Comparison of the baseline results (Figures 5.5 and 5.6) with embedded grid results shows that using simple grid embeddings with interpolations at the intergrid boundaries yields qualitatively acceptable flow solutions. The oscillations near the shock (due to lack of a good damping term for the very fine grid) are even duplicated.

Three-dimensional results have been obtained for inviscid subcritical and supercritical and viscous laminar flows over swept and unswept blades. Figure 5.11 shows surface Mach number distributions along selected planes for the 65x17x17 subcritical and supercritical inviscid flow case with the blade at a sweep angle of 26 degrees. Figure 5.12 contains surface isomach plots comparing the inviscid supercritical flow over unswept and swept 10% thick blades calculated on a 65x17x17 mesh. Figure 5.13 presents velocity profiles at 50% span for flow at Reynolds number 3.4×10^4 and sweep angle of 26 degrees. Again, results are identical for MacCormack and multigridded MacCormack.

5.3.2 Accuracy and Stability

The MacCormack method yields a solution that is second order accurate in both space and time [62]. For very fine three-dimensional grids, artificial viscosity (in the form of a fourth-order term described in Chapter 3) is added to the scheme to maintain stability. The effect of this additional damping is to enable convergence while maintaining the capability for high resolution of the interesting features of the flow.

Multigrid, when applied to the explicit predictor-corrector scheme, or to any other Lax-Wendroff scheme, for that matter, has been shown to yield the same solution to within the accuracy of the fine-grid scheme [76]. All results obtained here substantiate this assertion.

The embedded grid scheme, however, can affect the flow results in certain cases if care is not taken. Proper placing of the grid refinements is crucial to obtaining a correct flow solution, especially when different equation levels are being modeled in adjacent regions. Treatment of the intergrid boundaries in a non-conservative manner also impacts accuracy. Obviously, coarser grids have larger truncation error and can only be used in regions of the flow where the physical quantities are well-behaved or smooth.

The stability of the scheme is dependent on the treatment of the domain boundary conditions, the intergrid boundary conditions, and the amount of artificial viscosity introduced to stabilize the fine-grid solver. In all computations the time-step method was implemented using local time stepping and was

run at 90% of the prescribed CFL limit. Tests in which the CFL limit was intentionally exceeded resulted in divergence of the solution. This implies that the observed region of stability of the scheme agrees with the theoretical prediction.

5.3.3 Convergence Behavior

Several components of the algorithm serve to accelerate the convergence to a steady-state solution. Multigrid uses a series of coarser grids underlying a global fine grid to enhance performance of a basic fine-grid scheme, while the embedded zonal grid scheme reduces computational cost by the elimination of unnecessary grid points and some of the viscous term calculations.

Multigrid, applied to Lax-Wendroff-type methods for viscous and inviscid flow problems, has been tested extensively in two dimensions by Johnson [75,76,85]. In Table 5.1, a short summary of Johnson's results for the baseline flow problems used in the present work is recorded. Speedup for these cases is measured as the work to obtain a converged solution for the fine-grid scheme compared to that for the multigridded scheme. In Ref. 75, the term "work reduction factor" is used analogously. A more complete presentation of the two-dimensional multigrid results may be found in any of the aforementioned references.

The two-dimensional parallel coarse-grid algorithm maintains essentially the same convergence behavior as the sequential algorithm. Consequently, the multiple-grid speedups obtained with it are nearly identical to those of the sequential algorithm.

The multigrid method has been extended to three dimensions [87]. Used in conjunction with the explicit MacCormack method, speedup (as defined above) of nearly five times has been demonstrated for the subcritical test cases. The inviscid supercritical flow problem has a higher fine-grid convergence rate and therefore shows less improvement due to multigrid. These results are tabulated in Table 5.2. The basic fine grid for these test cases has 65x9x17 points, and three grid levels are used. The viscous test case yields a speedup factor of 4.4.

Tables 5.3 and 5.4 contain the results of the embedded zonal multigrid algorithm performance studies. The solution procedure was first implemented in two dimensions [108]. Numerical experiments were performed to determine the optimal positioning of the grid refinement levels. The inviscid cases do not take advantage of the zonal-equation part of the solver, since the Euler equations are solved everywhere in the domain. They therefore represent a measure of the work reduction achieved be merely deleting grid points in areas of the flow field where gradients are expected to be small. Speedup for these cases is defined as the work to a converged solution on a globally fine grid (equivalent to the finest embedding level) compared to the work to convergence using the embedded multigrid scheme. Two levels of grid refinements were used (see Figure 3.5) and resulting speedups are presented in Table 5.3. For the viscous case the thin-layer Navier-Stokes equations were solved on the finest embedding and the Euler equations defined the flow elsewhere. The speedup of 30 is a measure of the combined effect of grid point elimination and simplification of the governing equations via the zonal solver. These very encouraging results prompted further investigation into the viability of the scheme for three-dimensional problems.

The three-dimensional implementation of the scheme has demonstrated the performance characteristics contained in Table 5.4. The test grid has one refinement level corresponding to a global fine grid of 65x17x17 points (see Figure 3.6). For these inviscid cases, although just the Euler equations are solved everywhere, the artificial viscosity that has been added to the three-dimensional code is only computed and added to the solution at the finest mesh level. Thus the 3-D inviscid cases no longer provide a measure of the improvement in performance due solely to grid refinement alone, since the damping computation has been eliminated from use on the coarser grids. In the viscous case, the full Navier-Stokes equations are solved in the refined mesh region and the inviscid equations are applied elsewhere. The performance improvements demonstrated by these simple test cases with only one embedding lead to the speculation that even better results are possible for more complex problems.

Results have been obtained using the embedded multigrid for a case with the finest embedding equivalent to a global grid dimensioned 129x33x33, but, due to the difficulty of computing the corresponding baseline case (in terms of amount of CPU time required), no quantitative comparison can be made. However, the ability of the embedded multigrid scheme to obtain such a solution in a more reasonable amount of time attests to its potential.

5.4 Multitasking and Vectorization Performance

In addition to the investigation into various algorithmic components detailed above, a second, equally important study has been conducted to examine and improve the performance of the scheme on advanced-architecture

supercomputers, in order to further minimize the "time to solution" for large-scale flow problems. Both vectorization and multitasking on Cray and CDC supercomputers have been employed in this effort.

5.4.1 Vectorization

Vectorization of the two-dimensional parallel coarse-grid algorithm on a Cyber 205 yields speedups of 2.7 to 3.0 for the test cases considered. The vectorization results are summarized in Table 5.5. The overall speedups for the explicitly vectorized parallel coarse-grid scheme are recorded and contrasted with both the scalar and explicitly vectorized versions of the sequential coarse-grid scheme. Although the explicitly vectorized sequential scheme is highly efficient, a noticeable performance improvement results from use of the parallel scheme. The table also compares the performance of the parallel coarse-grid code segment with the analogous segment in the sequential version. By this measure, the performance improvement over the explicitly vectorized sequential code is about 8%. One should note that a performance gain of this size is significant since the potential for further vectorization of the code is quite low.

The three-dimensional multigrid MacCormack code was optimized to take full advantage of the automatic vectorization performed by the Cyber 205 compiler. Speedups of 4.2, 3.1, and 2.6 over the scalar code were obtained for selected inviscid subcritical, inviscid supercritical and turbulent viscous flows, respectively (see Table 5.6).

Further vector speedup was obtained by implementing Cyber 205 explicit vector Fortran. The speedups produced by this hand-vectorized code are

contained in Table 5.6, and show an average improvement of 45% over the automatically-vectorized code.

5.4.2 Multitasking

Extensive investigation into parallel-processing and multitasking has been carried out. Various problem decomposition strategies and multitasking utilities have been implemented, and results of these investigations are presented here.

Performance of parallel computers may be evaluated by comparing wall clock time for both uniprocessed and multiprocessed runs on a dedicated machine. This ratio is called the speedup. Dividing the speedup by the number of processors gives the efficiency of processor utilization, which provides a measure of load balancing and overhead costs.

A two-dimensional version of the code makes extensive use of parallelism inherent in the physical problem to obtain a good load balance when using the Cray-2 and X-MP macrotasking software. As the macrotasking approach requires the use of calls to a subroutine library, it has rather high overhead and thus yields best results for large-grained code segments. The problem has been decomposed both functionally and spatially. For the fine-grid scheme, the computational domain is partitioned into p strips (where p is the number of processors available), in such a way as to preserve vectorizability of the innermost *DO* loops. The parallel coarse-grid implementation of the multigrid scheme yields large-grained tasks, consisting of entire grid levels, to facilitate multitasking. Table 5.7 shows the resulting speedups of the coarse grid computations alone, in

the range of 77% to 89% efficiency. Table 5.8 presents the fine-grid macrotasking results and contrasts them with the same decomposition strategy implemented using microtasking. The microtasking results are only marginally better than the macrotasking ones because the algorithm has been restructured to maximize task granularity, with both getting good performance at over 90% efficiency.

Table 5.9 presents the two-dimensional macrotasking results from both the Cray X-MP/48 and two Cray-2s (with differing memory-access speeds) for the basic solver with multigrid. X-MP performance shows that the algorithm has been efficiently parallelized, while the poorer performance on the Cray-2s can be attributed to several factors. Macrotasking software is less mature on the Cray-2. By examining the difference in performance between the earlier Cray-2 (with 120 nanosecond memory-access cycle time) and the newer model (with 80 nanosecond access time), some of the decreased efficiency appears to be attributable to memory bank conflicts. The single path to memory may also impact multiprocessor efficiency. This is further discussed below.

Three-dimensional microtasking results for a small-grained partitioning of the multigrid algorithm, due to Pryor [213], are shown in Table 5.10.

The three-dimensional embedded zonal grid algorithm has been multitasked and tested on two Cray-2s with different memory-access speeds. Macrotasking has been implemented in the form of TSKSTARTs and TSKWAITs in the code, partitioned as described above. All performance data were obtained during dedicated time on both machines, and all runs were repeated at least three times to check the consistency of the timing data. To distinguish between the

two Cray-2 computers, they will be referred to here as Navier (serial number 2013), with a faster (80 nanosecond) memory-access time, and Stokes (serial number 2002), with the slower (120 nanosecond) memory access.

Table 5.11 shows the speedups and efficiency of processor utilization for two-, three-, and four-processor 100-time-cycle test cases of the three-dimensional code on both Navier and Stokes. A speedup of 2.67 on four processors, corresponding to 67% efficiency, is attained on Navier. On two and three processors, efficiencies of 86% and 74% were achieved, respectively. Comparing these results with the results obtained on the slower-memory Stokes indicates that some of the multitasking inefficiencies may be due to bank conflicts, that is, contention between processors for access to the 256-Megaword shared memory. A 2.5% to 8% increase in efficiency is observed when using the faster memory Cray-2. However, these improvements do not reflect the 15-20% improvement expected from merely increasing the memory-access speed. This implies that the other factors, possibly including the single path to memory, also serve to deteriorate multitasking performance.

Table 5.12 contains information about the best and worst performance of individual task groups, which were created by the algorithm partitioning. Each set of tasks may be classified (loosely) as one of the following: 1) loading one array into another (no operations); 2) simple computation (add and/or multiply) at all grid points; 3) many computations with unit stride at all points; 4) many computations with nonunit stride; and 5) heterogeneous tasks. Both the best- and worst-performing tasks in this study fall under category 4, indicating that task classification, at least by this scheme and for a complicated CFD problem, is not necessarily a good predictor of multitasking performance. If the

C-2

tasks are ranked according to unitasked wall clock execution time (a measure of granularity, on a dedicated machine), the better speedups generally correlate with the longer task segments, an indicator that task overhead (or the ratio of overhead to granularity) is slowing down the computation. The tasks remain too complicated to sufficiently detect the presence and extent of bank conflicts, so that any conclusions about the effect of memory contention must be drawn from the above comparison of performance on the two Cray-2s. Additional insight might be gained from experiments consisting of identical tasks with different array sizes, which would vary the memory access patterns.

6. Conclusions

The objective of this research has been to develop a robust and efficient computational algorithm for the solution of two- and three-dimensional aerodynamic flows which are governed by the Reynolds-averaged Navier-Stokes equations. A major consideration in the development of this methodology has been to try to fully exploit the vector- and parallel-processing capabilities of current-generation supercomputers.

This goal has been achieved with the following solution procedure. With an explicit flow solver as the basic ingredient, embedded refinements to the grid are introduced in regions of the flow field requiring high resolution (or, in other words, grid points are deleted in areas where high resolution is not needed). The basic scheme is then applied in a zonal implementation, whereby only the appropriate equations in the hierarchy from Euler to full Navier-Stokes are numerically solved depending on the physical characteristics of the flow, as well as the grid resolution, in various regions. Multigrid is applied to this embedded zonal method to accelerate convergence when steady-state solutions are desired. Vectorization and multitasking are used extensively to obtain the best performance on the most powerful computers available today.

Each component has been tested in two and three dimensions, and then integrated into the full three-dimensional scheme. Performance statistics for the

various pieces of the algorithm have been gathered in an attempt to predict the capabilities of the complete scheme.

A set of test cases representative of the internal flow through a turbomachinery blade row have been compiled. Results have been presented for a set of two- and three-dimensional model problems, both inviscid and viscous, subcritical and supercritical, and laminar and turbulent flows.

In two dimensions, multigrid alone yields speedups ranging from 2.1 to 8.2 over the fine-grid MacCormack scheme (as measured in work to convergence). For the three-dimensional problem, multigrid accelerates the computation by a factor of between 2.5 to 4.7.

The embedded zonal multigrid solver in two dimensions shows improvement over the fine-grid MacCormack scheme by factors of 6.1 to 30.2 with two levels of grid embeddings. In three dimensions, with only one level of grid embedding, speedups as much as 16.0 have been obtained.

Vectorization of the multigrid code in three dimensions on the Cyber 205 resulted in performance increases by a factor of nearly six over the scalar code. Introduction of the parallel coarse grid scheme gained an additional 8% in computational speed.

Much investigation has been carried out concerning the efficient use of parallel computers, and this topic has evolved into a focal point for the present research. Individual components and then the full scheme have been partitioned and multitasked. Using the Cray X-MP, two versions of multitasking were

implemented: macrotasking, a library-based utility which works best with large-grained tasks, and microtasking, a loop-level parallelization mechanism that can be efficiently used to multitask fine-grained code segments. Macrotasking was also employed on two Cray-2s having different memory access speeds.

The efficiency of processor use through macrotasking the two-dimensional multigrid code ranged from 83% to 94% on four and two processors, respectively, on the X-MP/48, and 56% to 84% on the Cray-2 for the same cases. Microtasking on the X-MP improved four-processor performance to 95% efficiency on the basic MacCormack fine-grid scheme, and the faster-memory Cray-2 improved performance to 90% efficiency for the 2-D multigrid code. As another data point, the three-dimensional multigrid code was microtasked on the X-MP and yielded speedups of 1.96 and 3.55 on two and four processors, respectively.

The fully integrated embedded zonal multigrid algorithm has been macrotasked on the Cray-2. The best multitasking performance speedups attained (over the unitasked analog) are 1.7, 2.2, and 2.7 on two, three, and four processors. A study of the performance of isolated components of the three-dimensional Navier-Stokes computation has also been carried out.

The three-dimensional multitasked embedded zonal multigrid code produces flow solutions on parallel-processing supercomputers that, if computed with a unitasked MacCormack fine-grid flow solver, would take an unreasonable amount of CPU time. Hence, it presents a means of investigating flow fields which are currently intractable. The viability of the solution algorithm, detailed herein, has been demonstrated by flow results and performance measurements.

7. Future Research

The algorithm and parallel-processing research carried out in this work has resulted in almost as many questions being raised as answered. In this chapter, some of the outstanding issues are noted and categorized, and the direction of possible future work is discussed.

7.0 Refinements

The treatment of the intergrid boundary conditions in the embedded grid scheme by simple bilinear interpolation, while sufficient to demonstrate the algorithm's viability, deserves further study. Various approaches applied by others in similar methods, including Usab [116], Rai [185], and Holst et al. [18], merit thorough examination in both two- and three-dimensional cases. As alluded to in Chapter 3, a conservative intergrid boundary scheme would presumably produce better results while increasing the computational work. Minimizing the cost of a conservative interface treatment would be a serious concern in such a research effort.

7.1 Extensions

A generalization of the grid embedding scheme to allow arbitrary numbers of grid levels and embedding regions should be implemented. The present

algorithm poses no barriers to such a generalization; an improved data management scheme (such as Usab's pointer system) might also enhance efficiency, in addition to providing the obvious benefit of making the scheme more flexible.

The generation of grid refinements could then be automated, as determined by the physics of the flow. The method of implementation would be based on previous work, such as Thompson and Ferziger [112], Berger [188], or Kallinderis and Baron [120,121], where either the magnitude of the pressure gradients or the solution residual is used to as a feature detection scheme to define refinement regions, which must be constrained to being regular. Berger's work in this area is of particular interest because it also incorporates adaptive refinements, yet another possible enhancement to the present scheme. This would enable the grid refinements to adapt to the physically evolving flow structures and automatically redefine themselves wherever or whenever necessary. Such a capability is particularly attractive for application to time-dependent flows.

7.2 Algorithms

As mentioned previously, all components of the solution procedure are explicit except for the surface pressure boundary condition, which consists of solving a normal momentum equation implicitly along x gridlines. The solution of this partial differential equation could also be computed with an explicit method, so as to remove the recursive computation so that the boundary condition would be vectorizable and parallelizable. Preliminary testing in two dimensions has shown that this yields results of the same order of accuracy as the implicit method and does, indeed, improve performance by a small percentage. Future work would include improving the two-dimensional implementation and then putting it into

the three-dimensional code. The advantages of using an explicit computation over an implicit one are improved code performance in both vector and parallel mode and reduction of the number of operations (and therefore the CPU time) required to compute the boundary conditions.

The use of artificial viscosity to stabilize flow computations is another area needing further attention. Much research has been conducted by others into this topic, including Pulliam [214] and Jameson et al. [82,215]. The approach used here, i.e., computational experimentation to determine the damping coefficients, is not feasible for very large problems due to the high cost it would entail. Total variation diminishing (TVD) schemes, such as ENO [216,217], are used to properly capture strong shocks and have been studied in detail by Yee and others [218-220].

The basic flow solver is not required to be the MacCormack method. If the explicitness of the solver is the most attractive feature, then the Runge-Kutta scheme of Jameson, Schmidt and Turkel [177] would be a viable alternative. If implicit schemes can be shown to have good performance on parallel computers, then one of the driving forces to maintain the explicitness of the basic flow solver is removed. Experimental work could be done with Beam-Warming [63] or another implicit method as the basic solver.

A parallel coarse-grid acceleration scheme has been developed and tested in two dimensions with good results, and should be extended for the three-dimensional case. This would be a straightforward task. Another multigrid variation, the fully parallel scheme, was attempted in two dimensions, but required underrelaxation and did not exhibit good convergence properties. It had the

advantage, however, of decoupling the fine and coarse grids from each other for each time step of the computation. Such asynchronous computation would be desirable for load balancing purposes. Further attention is perhaps merited.

Residual averaging was also briefly studied during the course of this work. Both implicit and explicit schemes were implemented in two dimensions but not optimized and not included in the results here. Due to the good results obtained by others using such acceleration techniques, further investigation is warranted, followed by extension to three dimensions.

7.3 Parallel-Processing

A number of topics for research have arisen as a result of the multitasking investigation carried out here. Distributed-memory multiprocessors, parallel software tools, alternative partitioning strategies, and the development of new parallel algorithms are discussed in the following.

The implementation of a CFD code on shared-memory parallel computers has been the focus of the current research. Some work is being performed to adapt such schemes to distributed-memory multiprocessors such as the Hypercube (MIMD) (by Bassett and Catherasoo [154], for example) and the Connection Machine (SIMD) (by Jespersen and Levit [163]). Early results show some promise for such architecture/algorithm combinations. An interesting task would be porting the three-dimensional embedded zonal multigrid scheme to such a machine. The spatial decomposition techniques used in the method are quite applicable to distributed architectures, and the explicit method allows an arbitrary partitioning of the domain, enabling optimization of communications and

redistribution of the workload without algorithmic modifications. On systems with hypercube interconnection topologies, binary-reflected gray-code mapping has been demonstrated as a good means of determining the most efficient communications paths.

The software tools provided for Fortran multitasking on the Cray-2 and X-MP are the macrotasking library subroutine calls and the microtasking preprocessor directives. Other extensions to Fortran for use on parallel machines have been developed, including the SCHEDULE package by Dongarra and Sorensen [168] and CoFortran by Weeks [167]. (See Appendix E for a brief description of the key routines of each.)

The SCHEDULE package [168] is intended to be a short-term solution to the problem of the inavailability of portable multitasking software. Each parallel machine currently has its own language or language extensions, so that code developed for parallel execution on one computer must be modified to run on another. SCHEDULE is meant to be usable on any shared-memory multiprocessing system (provided that an interface has been written for that machine). The package has been implemented in the two-dimensional version of the parallel multigrid algorithm, which had already been macrotasked on the Cray (both the X-MP and Stokes). Rewriting the parallel-processing management was relatively straightforward except for one major drawback. SCHEDULE has no efficient means of repeating the same set of processes (or the same "task graph") over and over, as is done in an iterative method and in most CFD solvers, so that it is not yet sophisticated enough to be useful to the computational aerodynamicist.

CoFortran is another attempt at creating a portable parallel programming language [167]. It consists of a combination of Fortran and Fortran-like concurrent language constructs. Like SCHEDULE, it is intended to be machine-independent, provided that an interface exists on the system to be used. It has been developed to provide the researcher with tools for the parallel solution of computational fluid dynamics problems, and is currently available on the X-MP and may soon be available on the Cray-2. It has not been tested with the algorithm of this work.

Microtasking shows the most promise for improving the multitasking capabilities of the Cray-2, based on experience with it on the X-MP, and its incorporation into the three-dimensional algorithm is called for. A code which has been macrotasked is easily converted to microtasking and should show improved performance, as has been demonstrated with the two-dimensional version of the multigrid algorithm on the Cray X-MP/48 [184].

As mentioned in the previous section, other schemes could be substituted for MacCormack to enhance the robustness and convergence properties of the solution procedure. The parallel performance of these other schemes would also be investigated. Although explicit schemes are very straightforward to parallelize, they involve many iterations of small operations count. Implicit schemes, on the other hand, take fewer, high-operations-count iterations, and may hold greater opportunity, or at least another alternative, for creating large-grained tasks for parallelization.

Another approach to developing parallel CFD codes, proposed by Shieh [221], is temporal decomposition. It is described as follows.

> Considering a 3D computation using an explicit or semi-implicit finite difference scheme, when a CPU is computing the jth grid plane for the nth time step, it is possible that one could proceed to compute the next time step, (n + 1)th, for (j - 2)th grid plane with another CPU. Thus, for a model with m grid points in j direction excluding boundary points, one could utilize (m + 1)/2 processors to computing in a pipe-line fashion... Each subtask processes the whole computational domain. Hence, the overhead for establishing parallel tasks and data transfer from one task to another is minimal.

7.4 Applications

Another obvious direction for future research to take, since the method has been developed for complex flow problems, is to apply it to a realistic configuration. A new grid generator would be desirable and probably necessary for such an endeavor - perhaps Sorensen's GRAPE code [222] or one of Thompson's codes [136]. Possibilities for the application include a full 3D configuration, an 3D inlet or nozzle geometry, a shuttle configuration, or a 3D staggered blade row for a turbomachine.

A different track to take would be to add species equations for reacting or dissociating gasses to the Navier-Stokes equations, resulting in a system of equations with different behavior or stiffness characteristics, one for which the embedded multigrid scheme may be appropriate.

References

1. Shang, J.S., and Scherr, S.J.: Navier-Stokes Solution of the Flow Field Around a Complete Aircraft. AIAA Paper 85-1509, July 1985.

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

3. Flores, J., Reznick, S.G., Holst, T.L., and Gundy, K.: Transonic Navier-Stokes Solutions for a Fighter-Like Configuration. AIAA 87-0032, January 1987.

4. Rai, M.M.: Unsteady Three-Dimensional Navier-Stokes Simulations of Turbine Rotor-Stator Interaction including Tip Effects. AIAA Paper 87-2058, 1987.

5. Peterson, V.L., and Arnold, J.O.: The Impact of Supercomputers on Experimentation: A View from a National Laboratory. ASEE Annual Conference Proceedings, 1985.

6. Andrews, A.E.: Progress and Challenges in the Application of Artificial Intelligence to Computational Fluid Dynamics. AIAA Paper 87-0593, January 1987.

7. Andrews, A.E.: A Knowledge-Based Approach to Automated Flow Field Zoning for Computational Fluid Dynamics (CFD). NASA Ames Research Center, May 1988.

8. Kutler, P., Steger, J.L., and Bailey, F.R.: Status of Computational Fluid Dynamics in the United States. AIAA Paper 87-1135 CP, 1987.

9. Johnson, G.M.: Requirements for Parallel Processing in Scientific Computation. Presented at the Third International Conference on Supercomputing, Boston, May 1988.

10. Reznick, S.G., and Flores, J.: Strake-Generated Vortex Interactions for a Fighter-Like Configuration. AIAA 87-0589, January 1987.

11. Yu, N.J., Kusunose, K., Chen, H.C., and Sommerfield, D.M.: Flow Simulations for a Complex Airplane Configuration Using the Euler Equations. AIAA Paper 87-0454, January 1987.

12. Chaussee, D.S., Rizk, Y.M., and Buning, P.G.: Viscous Computation of a Space Shuttle Flow Field. *Lecture Notes in Physics*, Vol. 218, Springer-Verlag, Berlin, 1985, pp. 148-153.

13. Volpe, G., Siclari, M.J., and Jameson, A.: A New Multigrid Euler Method for Fighter-Type Configurations. AIAA 87-1160, June 1987.

14. Holst, T.L.: Numerical Solution of the Navier-Stokes Equations about Three-Dimensional Configurations - A Survey. *Supercomputing in Aerospace*, NASA CP 2454, March 1987, pp. 281-298.

15. Shankar, V., and Chakravarthy, S.: Development and Application of Unified Algorithms for Problems in Computational Science. *Supercomputing in Aerospace*, NASA CP 2454, 1987.

16. Benek, J.A., Buning, P.G., and Steger, J.L.: A 3-D Chimera Grid Embedding Technique. AIAA Paper 85-1523, July 1985.

17. Benek, J.A., Donegan, T.L., and Suhs, N.E.: Extended Chimera Grid Embedding Scheme with Application to Viscous Flows. AIAA Paper 87-1126, June 1987.

18. Holst, T.L., Gundy, K.L., Flores, J., Chaderjian, N.M., Kaynak, U., and Thomas, S.D.: Numerical Solution of Transonic Wing Flows Using an Euler/Navier-Stokes Zonal Approach. AIAA Paper 85-1640, 1985.

19. Flores, J.: Convergence Acceleration for a Three-Dimensional Euler/Navier-Stokes Zonal Approach. AIAA Paper 85-1495, July 1985.

20. Obayashi, S., and Fujii, K.: Computation of Three-Dimensional Viscous Transonic Flows with the LU Factored Scheme. AIAA Paper 85-1510, July 1985.

21. Fujii, K., van Dalsem, W.R., Schiff, L.B., and Steger, J.L.: Numerical Simulation over a Strake-Delta Wing. AIAA Paper 87-1229, June 1987.

22. Thomas, J.L., Taylor, S.L., and Anderson, W.K.: Navier-Stokes Computations of Vortical Flows over Low Aspect Ratio Wings. AIAA Paper 87-0207, January 1987.

23. Ying, S.X., Steger, J.L., and Schiff, L.B.: Numerical Simulation of Unsteady, Viscous, High-Angle-of-Attack Flows Using a Partially Flux-Split Algorithm. AIAA Paper 86-2179, 1986.

24. Zhang, H.X., and Zheng, M.: Supersonic Viscous Flow Past a Blunt Edge Wing on a Flat Plate. CARDC Report, 1986.

25. Briley, W.R., Buggeln, R.C., and McDonald, H.: Solution of the Three-Dimensional Navier-Stokes Equations for a Steady Laminar Horseshoe Vortex Flow. AIAA Paper 85-1520, July 1985.

26. Bruneau, C.H., Chattot, J.J., Laminie, J., and Temam, R.: Computation of Vortex Flow past a Flat Plate at High Angle of Attack. *Lecture Notes in Physics*, Vol. 264, Springer-Verlag, Berlin, 1986, pp. 134-140.

27. Murman, E.M., and Goodsell, A.M.: Transonic Vortical Flow. IUTAM Symposium Transsonicum III, Gottingen, Germany, May 1988.

28. Newsome, R.W., and Kandil, O.A.: Vortex Flow Aerodynamics - Physical Aspects and Numerical Simulation. AIAA Paper 87-0205, January 1987.

29. Rai, M.M., and Madavan, N.K.: Multi-Airfoil Navier-Stokes Simulations of Turbine Rotor-Stator Interaction, AIAA Paper 88-0361, January 1988.

30. Chima, R.V.: Development of an Explicit Multi-Grid Algorithm for Quasi-Three-Dimensional Viscous Flows in Turbo Machinery. NASA TM-87128, 1986.

31. Adamczyk, J.J., and Graham, R.W.: Numerical Solution of Multiblade Row Turbomachinery Flows. *Cray Channels*, July 1986.

32. Nakamichi, J.: Calculations of Unsteady Navier-Stokes Equations around an Oscillating Three-Dimensional Wing Using a Moving Grid System. AIAA Paper 87-1158, June 1987.

33. Van Dalsem, W.R.: Study of V/STOL Flows Using the Fortified Navier-Stokes Scheme. *Computational Fluid Dynamics* (de Vahl Davis, G., and Fletcher, C., eds.), North-Holland, Amsterdam, 1988, pp. 725-735.

34. Chen, C.L., McCroskey, W.J., and Ying, S.X.: Euler Solution of Multi-Blade Rotor Flow. 13th European Rotorcraft Forum, Paper No. 2-2, September 1987.

35. Srinivasan, G.R., and McCroskey, W.J.: Navier-Stokes Calculations of Hovering Rotor Flowfields. AIAA Atmospheric and Flight Mechanics Conference, AIAA Paper 87-2629-CP, August 1987.

36. Strawn, Purcell, and Caradonna: Geometric Acoustics and Transonic Helicopter Sound. Presented at the AIAA Aeroacoustics Conference, Sunnyvale, 1987.

37. Li, C.P.: Chemical Nonequilibrium and Viscous Flow Computation for Conic Aerobrake Bodies. *Computational Fluid Dynamics*, (de Vahl Davis, G., and Fletcher, C., eds.) North-Holland, Amsterdam, 1988, pp. 461-471.

38. Bardina, J., and Lombard, C.K.: Three Dimensional Hypersonic Flow Simulations with the CSCM Implicit Upwind Navier-Stokes Method. AIAA Paper 87-1114, June 1987.

39. Stoufflet, B., Periaux, J., Fezoui, F., and Dervieux, A.: Numerical Simulation of Three-Dimensional Hypersonic Euler Flows around Space Vehicles Using Finite Elements. AIAA Paper 87-0560, January 1987.

40. Dwoyer, D.L., and Kumar, A.: Computational Analysis of Hypersonic Airbreathing Aircraft Flow Fields. *Supercomputing in Aerospace*, NASA CP 2454, March 1987, pp. 239-255.

41. White, M.E., Drummond, J.P., and Kumar, A.: Evolution and Status of CFD Techniques for Scramjet Applications. AIAA Paper 86-0160, January 1986.

42. Shang, J.S., and McMaster, D.L.: Interaction of Jets in Hypersonic Cross Stream. AIAA Paper 87-0057, January 1987.

43. Shang, J.S., and McMaster, D.L.: Supersonic Transverse Jet from a Rotating Ogive Cylinder into a Hypersonic Flow. AIAA Paper 87-1441, 1987.

44. Rieger, H.: Solution of Some Three-Dimensional Viscous and Inviscid Supersonic Flow Problems by Finite-Volume Time- and Space-Marching Schemes. *Aerodynamics of Hypersonic Lifting Vehicles*, AGARD-CP428, 1987.

45. Chandler, G.V., and MacCormack, R.W.: Hypersonic Flow Past 3-D Configurations. AIAA Paper 87-0480, January 1987.

46. Wake, B.E., Sankar, L.N., Wu, J., and Ruo, S.Y.: An Efficient Procedure for the Numerical Solution of Three-Dimensional Viscous Flows. AIAA Paper 87-1159, June 1987.

47. Richardson, L.F.: The Approximate Arithmetic Solution by Finite Differences of Physical Problems involving Differential Equations, with an Application to the Stresses in a Masonry Dam. *Transactions of the Royal Society of London*, Series A, Vol. 210, 1910, pp. 307-357.

48. Courant, R., Friedrichs, K., and Lewy, H.: Uber die patiellen Differenzengleichungen der Mathematischen Physik. *Mathematische Annalen*, Vol. 100, 1928, pp. 32-74.

49. Southwell, R.V.: On Relaxation Methods: A Mathematics for Engineering Science. *Proceedings of the Royal Society of London*, Series A, Vol. 184, 1945, pp. 253-288.

50. Emmons, H.W.: The Numerical Solution of Compressible Fluid Flow Problems. NACA TN-932, 1944.

51. Emmons, H.W.: The Theoretical Flow of a Frictionless, Adiabatic, Perfect Gas inside of a Two-Dimensional Hyperbolic Nozzle. NACA TN-1003, 1946.

52. Emmons, H.W.: Flow of a Compressible Fluid Past a Symmetrical Airfoil in a Wind Tunnel and in Free Air. NACA TN-1746, 1948.

53. von Neumann, J.: Proposal and Analysis of a New Numerical Method for the Treatment of Hydrodynamical Shock Problems. Applied Mathematics Panel, National Defense Research Committee, Report 108.1R, 1944.

54. von Neumann, J., and Richtmyer, R.D.: A Method for the Numerical Calculation of Hydrodynamic Shocks. *J. Applied Physics*, Vol. 21, 1950, pp. 232-237.

55. Lax, P.D.: Weak Solutions of Nonlinear Hyperbolic Equations and Their Numerical Computation. *Communications on Pure and Applied Mathematics*, Vol. 7, 1954, pp. 159-193.

56. Morawetz, C.S.: On the Non-Existence of Continuous Transonic Flows Past Profiles I. *Communications on Pure and Applied Mathematics*, Vol. 9, 1956, pp. 45-68.

57. Morawetz, C.S.: On the Non-Existence of Continuous Transonic Flows Past Profiles II. *Communications on Pure and Applied Mathematics*, Vol. 10, 1957, pp. 107-131.

58. Morawetz, C.S.: On the Non-Existence of Continuous Transonic Flows Past Profiles III. *Communications on Pure and Applied Mathematics*, Vol. 11, 1958, pp. 129-144.

59. Magnus, R., and Yoshihara, H.: Inviscid Transonic Flow over Airfoils. *AIAA Journal*, Vol. 8, 1970, pp. 2157-2162.

60. Crocco, L.: A Suggestion for the Numerical Solution of the Steady Navier-Stokes Equations. *AIAA Journal*, Vol. 3, 1965, pp. 1824-1832.

61. Murman, E.M., and Cole, J.D.: Calculation of Plane Steady Transonic Flows. *AIAA Journal*, Vol. 9, 1971, pp. 114-121.

62. MacCormack, R.W.: The Effect of Viscosity on Hypervelocity Impact Cratering. AIAA Paper 69-354, 1969.

63. Beam, R.W., and Warming, R.F.: An Implicit Finite Difference Algorithm for Hyperbolic Systems in Conservation-Law Form. *Journal of Computational Physics*, Vol. 22, 1976, pp. 87-110.

64. Roe, P.L.: Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes. *Journal of Computational Physics*, Vol. 43, 1981, pp. 357-372.

65. Roe, P.: A Survey of Upwind Difference Techniques. Presented at the 11th International Conference on Numerical Methods in Fluid Dynamics, Williamsburg, Virginia, June 1988.

66. Harten, A.: High Resolution Schemes for Hyperbolic Conservation Laws. *Journal of Computational Physics*, Vol. 49, 1983, pp. 357-393.

67. Chakravarthy, S., and Osher, S.: Applications of a New Class of High Accuracy TVD Schemes to the Navier-Stokes Equations. AIAA Paper 85-0165, January 1985.

68. Richtmyer, R.D., and Morton, K.W.: *Difference Methods for Initial-Value Problems.* Wiley, New York, 1967.

69. Richtmyer, R.D.: *Difference Methods for Initial-Value Problems.* 2nd ed., Interscience Publishers, New York, 1967.

70. Mitchell, A.R.: *Computational Methods in Partial Differential Equations.* Wiley, London, 1969.

71. Roache, P.J.: *Computational Fluid Dynamics.* Hermosa Publishers, Albuquerque, rev. 1976 (orig. 1972).

72. Anderson, D.A., Tannehill, J.C., and Pletcher, R.H.: *Computational Fluid Mechanics and Heat Transfer.* MacGraw-Hill, 1984.

73. Ames, W.F.: *Numerical Methods for Partial Differential Equations* (2nd edition). Academic Press, New York, 1977.

74. Ni, R.-H.: A Multiple-Grid Scheme for Solving the Euler Equations. *AIAA Journal*, Vol. 20, 1982, pp. 1565-1571.

75. Johnson, G.M.: Multiple-Grid Acceleration of Lax-Wendroff Algorithms. NASA TM-82843, 1982.

76. Johnson, G.M.: Multiple-Grid Convergence Acceleration of Viscous and Inviscid Flow Computations. *Journal of Applied Mathematics and Computation*, Vol. 13, Nos. 3-4, November 1983, pp. 375-398.

77. Jespersen, D.C.: A Multigrid Method for the Euler Equations. AIAA Paper 83-0124, January 1983.

78. Jespersen, D.C.: Design and Implementation of a Multigrid Code for the Euler Equations. Proceedings of the International Multigrid Conference, Copper Mountain, Colorado, 6-8 April 1983.

79. Jameson, A.: A Multigrid Euler Solver. Proceedings of the International Multigrid Conference, Copper Mountain, Colorado, 6-8 April 1983.

80. Jameson, A.: Solution of the Euler Equations for Two-Dimensional Transonic Flow by a Multigrid Method. MAE Report No. 1613, Princeton University, June 1983.

81. Stubbs, R.M.: A Multiple-Gridding of the Euler Equations with an Implicit Scheme. AIAA Paper 83-1945, 1983.

82. Jameson, A., and Yoon, S.: Multigrid Solution of the Euler Equations Using Implicit Schemes. AIAA Paper 85-0293, 1985.

83. Jameson, A., and Yoon, S.: LU Implicit Schemes with Multiple Grids for the Euler Equations. AIAA Paper 86-0105, 1986.

84. Brandt, A.: Multilevel Adaptive Solutions to Boundary Value Problems. *Mathematics of Computation*, Vol. 31, 1977, pp. 333-390.

85. Johnson, G.M.: Flux-Based Acceleration of the Euler Equations. NASA TM-83453, 1983.

86. Koeck, C., and Chattot, J.J.: Computation of Three-Dimensional Vortex Flows Past Wings Using the Euler Equations and a Multiple-Grid Scheme. *Lecture Notes in Physics*, Vol. 218, Springer-Verlag, Berlin, 1985, pp. 308-313.

87. Johnson, G.M., and Swisshelm, J.M.: Multiple-Grid Solution of the Three-Dimensional Euler and Navier-Stokes Equations. *Lecture Notes in Physics*, Vol. 218, Springer-Verlag, Berlin, 1985, pp. 286-290.

88. Jespersen, D.C.: A Time-Accurate Multiple-Grid Algorithm. AIAA Paper 85-1493, July 1985.

89. Briggs, W.L.: *A Multigrid Tutorial.* Society for Industrial and Applied Mathematics, Philadelphia, 1987.

90. Brand, K., Lemke, M., and Linden, J.: Multigrid Bibliography, Edition 4. Gesellschaft fur Mathemtik und Datenverarbeitung, May 1986.

91. Jespersen, D.C.: Recent Developments in Multigrid Methods for Steady Equations. Computational Fluid Dynamics Vol. 2, Von Karman Institute, 1984.

92. Hackbusch, W., and Trottenberg, U.: *Multigrid Methods.* Springer-Verlag, Berlin, 1982.

93. McCormick, S.: *Multigrid Methods.* SIAM Frontiers Series, Vol. 3, Society for Industrial and Applied Mathematics, Philadelphia, 1987.

94. Hemker, P.W. and Spekreijse, S.P.: Multiple Grid and Osher's Scheme for the Efficient Solution of the Steady Euler Equations. NMR 8507, Centrum voor Wiskunde en Informatica, Stichting Mathematisch Centrum, Amsterdam, 1985.

95. Hemker, P.W., Koren, B., and Spekreijse, S.P.: A Nonlinear Multigrid Method for the Efficient Solution of the Steady Euler Equations. *Lecture Notes in Physics,* Vol. 264, Springer-Verlag, Berlin, 1986, pp. 308-313.

96. Mulder, W.: Multigrid Relaxation for the Euler Equations. *Lecture Notes in Physics,* Vol. 218, Springer-Verlag, Berlin, 1985, pp. 417-421.

97. Shaw, G., and Wesseling, P.: Multigrid Solution of the Compressible Navier-Stokes Equations on a Vector Computer. *Lecture Notes in Physics,* Vol. 264, Springer-Verlag, Berlin, 1986, pp. 566-571.

98. Hall, M.G.: Cell-Vertex Multigrid Schemes for Solution of the Euler Equations. Proceedings of Conference on Numerical Methods for Fluid Dynamics, University of Reading, UK, 4-10 April 1985.

99. Salmond, D.J.: A Cell-Vertex Multigrid Scheme for the Solution of the Euler Equations for Transonic Flow Past a Wing. *Lecture Notes in Physics,* Vol. 264, Springer-Verlag, Berlin, 1986, pp. 549-553.

100. Perez, E., Periaux, J., Rosenblum, J.P., Stoufflet, B., Dervieux, A., and Lallemand, M.H.: Adaptive Full-Multigrid Finite Element Methods for Solving the Two-Dimensional Euler Equations. *Lecture Notes in Physics,* Vol. 264, Springer-Verlag, Berlin, 1986, pp. 523-527.

101. Lallemand, M.H., and Dervieux, A.: A Multigrid Finite Element Method for Solving the Two Dimensional Euler Equations. Proceedings of the Third Copper Mountain Conference on Multigrid Methods, Copper Mountain, Colorado, 6-10 April 1987.

102. Jameson, A., and Mavriplis, D.: Multigrid Solution of the Euler Equations on Unstructured and Adaptive Meshes. Proceedings of the Third Copper Mountain Conference on Multigrid Methods, Copper Mountain, Colorado, 6-10 April 1987.

103. Mavriplis, D.J.: Accurate Multigrid Solution of the Euler Equations on Unstructured and Adaptive Meshes. AIAA Paper 88-3706-CP, July 1988.

104. Swisshelm, J.M., Johnson, G.M., and Kumar, S.P.: Parallel Computation of Euler and Navier-Stokes Flows. *Journal of Applied Mathematics and Computation*, Vol. 19, Nos. 1-4, July 1986, pp. 321-331.

105. Greenbaum, A.: A Multigrid Method for Multiprocessors. UCRL-92211, presented at the Second Copper Mountain Conference on Multigrid Methods, Copper Mountain, Colorado, 1-3 April 1985.

106. Frederickson, P.O., and McBryan, O.: Superconvergent Multigrid Methods. Cornell Theory Center preprint, May 1987.

107. Bassi, F., Grasso, F., and Savini, M.: Solution of the Compressible Navier Stokes Equations by Using Embedded Adaptive Meshes. *Lecture Notes in Physics*, Vol. 264, Springer-Verlag, Berlin, 1986, pp. 113-119.

108. Johnson, G.M., Swisshelm, J.M., Pryor, D.V., and Ziebarth, J.P.: Multitasked Embedded Multigrid for Three-Dimensional Flow Simulation. *Lecture Notes in Physics*, Vol. 264, Springer-Verlag, Berlin, 1986, pp. 350-356.

109. Johnson, G.M., and Swisshelm, J.M.: Multigrid for Parallel-Processing Supercomputers. Proceedings of the Third Copper Mountain Conference on Multigrid Methods, Copper Mountain, Colorado, 6-10 April 1987.

110. Swisshelm, J.M., and Johnson, G.M.: Development of a Aerodynamics Algorithm for Parallel-Processing Supercomputers. *Computational Fluid Dynamics* (de Vahl Davis, G., and Fletcher, C., eds.), North-Holland, Amsterdam, 1988, pp. 689-698.

111. Berger, M.J., and Jameson, A.: An Adaptive Multigrid Method for the Euler Equations. *Lecture Notes in Physics*, Vol. 218, Springer-Verlag, Berlin, 1985, pp. 92-97.

112. Thompson, M.C., and Ferziger, J.H.: An Adaptive Multigrid Approach to the Steady-State Incompressible Navier-Stokes Equations. *Computational Fluid Dynamics* (de Vahl Davis, G., and Fletcher, C., eds.), North-Holland, Amsterdam, 1988, pp. 715-724.

113. Woodward, P., and Colella, P.: The Numerical Simulation of Two-Dimensional Fluid Flow with Strong Shocks. *Journal of Computational Physics*, Vol. 54, 1984, pp. 115.

114. Dannenhoffer, J.F. III, and Baron, J.R.: Adaptive Procedure for Steady State Solution of Hyperbolic Equations. AIAA Paper 84-0005, 1984.

115. Kutler, P.: A Perspective of Theoretical and Applied Computational Fluid Dynamics. AIAA Paper 83-0037, January 1983.

116. Usab, W.J.: Embedded Mesh Solutions to the Euler Equation Using a Multiple-Grid Method. Ph.D. Thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, December 1983.

117. Rai, M.M.: Navier-Stokes Simulations of Rotor-Stator Interactions Using Patched and Overlaid Grids. AIAA Paper 85-1519, July 1985.

118. Berger, M.J., and Oliger, J.: Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations. *Journal of Computational Physics*, Vol. 53, No. 3, March 1984, pp. 484-512.

119. Bell, J.B., Colella, P. Trangenstein, J.A., and Welcome, M.: Adaptive Methods for High Mach Number Reacting Flow. AIAA Paper 87-1168 CP, 1987.

120. Kallinderis, J.G., and Baron, J.R.: Adaptation Methods for a New Navier-Stokes Algorithm. AIAA Paper 87-1167 CP, 1987.

121. Kallinderis, J.G., and Baron, J.R.: Unsteady and Turbulent Flow using Adaptation Methods. Presented at the 11th International Conference on Numerical Methods in Fluid Dynamics, Williamsburg, Virginia, June 1988.

122. Nakahashi, K.: FDM-FEM Zonal Approach for Computations of Compressible Viscous Flows. *Lecture Notes in Physics*, Vol. 264, Springer-Verlag, Berlin, 1986, pp. 494-498.

123. Nakahashi, K., and Obayashi, S.: Viscous Flow Computations Using a Composite Grid. AIAA Paper 87-1128 CP, 1987.

124. Nakahashi, K., and Obayashi, S.: FDM-FEM Zonal Approach for Viscous Flow Computations over Multiple Bodies. AIAA Paper 87-0604, January 1987.

125. Nakahashi, K., and Deiwert, G.: A Practical Adaptive-Grid Method for Complex Fluid-Flow Problems. *Lecture Notes in Physics*, Vol. 218, Springer-Verlag, Berlin, 1985, pp. 422-426.

126. Nakahashi, K., and Deiwert, G.S.: A Self-Adaptive-Grid Method with Application to Airfoil Flow. AIAA Paper 85-1525, July 1985.

127. Morgan, K., Peraire, J., Theraja, R.R., and Stewart, J.R.: An Adaptive Finite Element Scheme for the Euler and Navier-Stokes Equations. AIAA Paper 87-1172, June 1987.

128. Lohner, R., and Patnaik, G.: Applications of the Method of Flux-Corrected Transport to Generalized Meshes. *Lecture Notes in Physics*, Vol. 264, Springer-Verlag, Berlin, 1986, pp. 428-434.

129. Lohner, R., Morgan, K., and Zienkiewicz, O.C.: An Adaptive Finite Element Method for High Speed Compressible Flow. *Lecture Notes in Physics*, Vol. 218, Springer-Verlag, Berlin, 1985, pp. 388-392.

130. Benek, J.A., Steger, J.L., Dougherty, F.C., and Buning, P.G.: Chimera: A Grid-Embedding Technique. NASA TM-89246, April 1986.

131. Benek, J.A., Steger, J.L., and Dougherty, F.C.: A Flexible Grid Embedding Technique with Application to the Euler Equations. AIAA Paper 83-1944, July 1983.

132. Eberhardt, S., and Baganoff, D.: Overset Grids in Compressible Flow. AIAA Paper 85-1524, July 1985.

133. Dougherty, F.C., Benek, J.A., and Steger, J.L.: On Applications of Chimera Grid Schemes to Store Separation. NASA TM-88193, 1985.

134. Benek, J.A., Donegan, T.L., and Suhs, N.E.: Experience with 3-D Composite Grids. *Supercomputing in Aerospace*, NASA CP 2454, March 1987, pp. 271-277.

135. Thompson, J.F., Warsi, Z.U.A., and Mastin, C.W.: *Numerical Grid Generation: Formulations and Applications.* North-Holland, New York, 1983.

136. Thompson, J.F.: A Survey of Composite Grid Generation for General Three-Dimensional Regions. *Numerical Methods for Engine-Airframe Integration,* AIAA, New York, 1986, pp. 52-85.

137. Hockney, R.W. and Jesshope, C.R.: *Parallel Computers.* Adam Hilger, Bristol, 1983.

138. Stone, H.S.: *High-Performance Computer Architecture.* Addison-Wesley, Reading, 1987.

139. Dongarra, J.J., and Duff, I.S.: Advanced Architecture Computers. Argonne National Laboratory TM 57 (Revision 1), January 1987.

140. Amdahl, G.M.: Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. *Proceedings of the American Federation of Information Processing Societies,* Vol. 30, Thompson, Washington, DC, 1967, pp. 483-485.

141. Worlton, J.: Toward a Science of Parallel Computation. *Computational Mechanics - Advances and Trends.* AMD - Vol. 75 (A.K. Noor, ed.), ASME, New York, 1987, pp. 23-35.

142. Gustafson, J.L.: Reevaluating Amdahl's Law. *Communications of the ACM,* Vol. 31, No. 5, May 1988, pp. 532-533.

143. Martin, J.L., and Mueller-Wichards, D.: Supercomputer Performance Evaluation: Status and Directions. *The Journal of Supercomputing,* Vol. 1, 1987, pp. 87-104.

144. Proceedings of 1988 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems. Special Issue of *Performance Evaluation Review,* ACM Press, Vol. 16, No. 1, May 1988.

145. Knapp, M.A., Ackins, G.M., and Thomas, J.: Applications of the ILLIAC IV. *Parallel Processor Systems, Technologies, and Applications,* Spartan Books, New York, 1970, pp. 34-35.

146. Ibbett, R.N.: *The Architecture of High Performance Computers.* Springer-Verlag New York, Inc., New York, 1982.

147. Patel, N.R, Sturek, W.B., and Hiromoto, R.E.: A Parallel Compressible Flow Algorithm for Multiprocessors. *Applications of Parallel Processing in Fluid Dynamics,* FED - Vol. 47 (O. Baysal, ed.), ASME, New York, 1987, pp. 49-64.

148. Swisshelm, J.M.: Multitasking a Three-Dimensional Navier-Stokes Algorithm for the Cray-2. To be presented at Supercomputing '88, Orlando, 14-18 November 1988.

149. Mulac, R.A., Celestina, M.L., Adamczyk, J.J., Misegades, K.P., and Dawson, J.M.: The Utilization of Parallel Processing in Solving the Inviscid Form for the Average-Passage Equation System for Multistage Turbo Machinery. AIAA Paper 87-1108, June 1987.

150. Misegades, K., Krause, L., and Booth, M.: Microtasking Fluid Dynamics Codes on the Cray X-MP. *Applications of Parallel Processing in Fluid Dynamics,* FED - Vol. 47 (O. Baysal, ed.), ASME, New York, 1987, pp. 49-64.

151. McCormick, S. and Quilan, D.: Multilevel Load Balancing for Multiprocessors. Proceedings of the Third Copper Mountain Conference on Multigrid Methods, Copper Mountain, Colorado, 6-10 April 1987.

152. Gustafson, J.L., Montry, G.R., and Benner, R.E.: Development of Parallel Methods for a 1024-Processor Hypercube. *SIAM Journal on Scientific and Statistical Computing,* Vol. 9, No. 4, July 1988.

153. Catherasoo, C.J.: Separated Flow Simulations Using the Vortex Method on a Hypercube. AIAA Paper 87-1109, June 1987.

154. Bassett, M.E., and Catherasoo, C.J.: Simulation of Transonic Flow on a Hypercube. AMETEK TR-87-012, May 1987.

155. Erlebacher, G., Bokhari, S.H., and Hussaini, M.Y.: An Efficient Parallel Algorithm for the Simulation of Three-Dimensional Compressible Transition on a 20 Processor Flex/32 Multicomputer. NASA CR 178340, July 1987.

156. Fatoohi, R., and Grosch, C.E.: Solving the Cauchey-Riemann Equations on Parallel Computers. NASA CR 178307, May 1987.

157. Fatoohi, R., and Grosch, C.E.: Implementation of an ADI Method on Parallel Computers. *Journal of Scientific Computing,* Vol. 2, No. 2, June 1987, pp. 175-193.

158. Fatoohi, R., and Grosch, C.E.: Implementation and Analysis of a Navier-Stokes Algorithm on Parallel Computers. NASA CR 181614, January 1988.

159. Naik, V.K., and Ta'asan, S.: Performance Studies of the Multigrid Algorithms Implemented on Hypercube Multiprocessor Systems. NASA CR 178237, January 1987.

160. Naik, V.K., and Ta'asan, S.: Implementation of Multigrid Methods for Solving Navier-Stokes Equations on a Multiprocessor System. NASA CR 178319, June 1987.

161. Murman, E.M., Modiano, D., Haimes, R., and Giles, M.: Performance of Several CFD Loops on Parallel Processors. Report CFDL-TR-88-3, Massachusetts Institute of Technology, March 1988.

162. Modiano, D.: Performance of a Common CFD Loop on Two Parallel Architectures. Report CFDL-TR-87-11, Massachusetts Institute of Technology, November 1987.

163. Levit, C., and Jespersen, D.: Explicit and Implicit Solution of the Navier-Stokes Equations on a Massively Parallel Computer. To be presented at Symposium on Advances and Trends in Structural Mechanics and Fluid Dynamics, Washington, DC, 17-19 October 1988.

164. Frederickson, P.O., and McBryan, O.: Parallel Superconvergent Multigrid. Cornell Theory Center TR 12, 1987.

165. Ortega, J.M., and Voight, R.G.: A Bibliography on Parallel and Vector Numerical Algorithms. NASA CR 178335, July 1987.

166. Johnson, G.M.: Parallel Processing in Fluid Dynamics. *Applications of Parallel Processing in Fluid Mechanics*, FED - Vol. 47 (O. Baysal, ed.), ASME, New York, 1987, pp. 1-8.

167. Weeks, C.L.: Concurrent Extensions to the Fortran Language. Presented at the First World Congress on Computational Mechanics, 22-26 September 1986.

168. Dongarra, J.J., and Sorensen, D.C.: A Portable Environment for Developing Parallel FORTRAN Programs. *Parallel Computing*, Vol. 5, 1987, pp. 175-186.

169. Seager, M., Campbell, S, Sikora, S., Strout, R., and Zosel, M.: Graphical Multiprocessing Analysis Tool (GMAT). User Documentation, Lawrence Livermore National Laboratory, 1987.

170. Balasundaram, V., Baumgartner, D., Callahan, D., Kennedy, K., and Subhlok, J.: PTOOL: A System for Static Analysis of Parallelism in Programs. Rice University COMP TR88-71, June 1988.

171. Bailey, D.H.: private communication, 1988.

172. Viviand, H.: Conservative Forms of Gas Dynamic Equations. *La Recherche Aerospatiale*, No. 1, January-February 1974, pp. 65-68.

173. Peterson, V.L.: Impact of Computers on Aerodynamics Research and Development. *Proceedings of the IEEE*, Vol. 72, No. 1, January 1984, pp. 68-79.

174. Boussinesq, J.: Essai Sur La Theorie Des Eaux Courantes. *Mem. Presentes Acad. Sci.*, Vol. 23, Paris, p. 46.

175. Baldwin, B.S., and Lomax, H.: Thin Layer Approximation and Algebraic Model for Separated Turbulent Flows. AIAA Paper 78-257, 1978.

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

177. Jameson, A., Schmidt, W., and Turkel, E.: Numerical Solution of the Euler Equations by Finite Volume Methods Using Runge Kutta Time Stepping Schemes. AIAA Paper 81-1259, 1981.

178. Courant, R., Fredrichs, K., and Lewy, H.: On Partial Difference Equations of Mathematical Physics, *IBM Journal of Research and Development*, Vol. 11, No. 2, 1967, pp. 215-234. (English translation of the original work of 1928.)

179. Lax, P.D.: Hyperbolic Difference Equations: A Review of the Courant-Friedrichs-Lewy Paper in Light of Recent Developments. *IBM Journal of Research and Development*, Vol. 11, No. 2, 1967, pp. 235-238.

180. Carter, J.E.: Numerical Solutions of the Navier-Stokes Equations for the Supersonic Laminar Flow over a Two-Dimensional Compression Corner. NASA TR R-385, July 1972.

181. Chima, R.V.: private communication, 1986.

182. Jameson, A. and Baker, T.J.: Multigrid Solution of the Euler Equations for Aircraft Configurations. AIAA Paper 84-0093, January 1984.

183. Chima, R.V., Turkel, E., and Schaffer, S.: Comparison of Three Explicit Multigrid Methods for the Euler and Navier-Stokes Equations. NASA TM-88878, 1987.

184. Johnson, G.M., Swisshelm, J.M., and Kumar, S.P: Concurrent-Processing Adaptations of a Multiple-Grid Algorithm. AIAA Paper 85-1508, July 1985.

185. Rai, M.M.: An Implicit, Conservative, Zonal-Boundary Scheme for Euler Equation Calculations. AIAA Paper 85-0488, 1985.

186. Giles, M.: Accuracy of Node-Based Solutions on Irregular Meshes. Presented at the 11th International Conference on Numerical Methods in Fluid Dynamics, Williamsburg, Virginia, June 1988.

187. Lindquist, D., and Giles, M.: A Comparison of Numerical Schemes on Triangular and Quadrilateral Meshes. Presented at the 11th International Conference on Numerical Methods in Fluid Dynamics, Williamsburg, Virginia, June 1988.

188. Berger, M.: Adaptive Mesh Refinement for Parallel Processors. Presented at the Third SIAM Conference on Parallel Processing for Scientific Computing, Los Angeles, December 1987.

189. Flynn, M.J.: Some Computer Organizations and Their Effectiveness. *IEEE Transactions on Computers*, Vol. 21, No. 9, September 1972, pp. 948-960.

190. Baysal, O.: Supercomputing of Supersonic Flows Using Upwind Relaxation and MacCormack Schemes.. *Applications of Parallel Processing in Fluid Mechanics,* FED - Vol. 47, O. Baysal, ed., ASME, New York, 1987, pp. 1-8.

191. Cyber 205 Tutorial. Institute for Computational Studies, Fort Collins, CO, 1984.

192. Cray-2 Fortran (CFT2) Reference Manual. SR-2007, Cray Research, Inc., Mendota Heights, MN, revised October 1986.

193. Cray-2 Multitasking Programmer's Manual. SN-2026, Cray Research, Inc., Mendota Heights, MN, revised June 1987.

194. Dongarra, J., Bunch, J., Moler, C., and Stewart, G.: LINPACK Users' Guide. Society for Industrial and Applied Mathematics, Philadelphia, 1976.

195. McMahon, F.H.: The Livermore Fortran Kernels: A Computer Test of Numerical Performance Range. UCRL-53745, Lawrence Livermore National Laboratory, December 1986.

196. Bailey, D.H., and Barton, J.T.: The NAS Kernel Benchmark Program. NASA Ames Research Center, March 1985.

197. Wilson, K.G.: Grand Challenges to Computational Science. Cornell Center for Theory and Simulation in Science and Engineering, Ithaca, New York, October 1986.

198. Bieterman, M.: The Impact of Microtasked Applications in a Multiprogramming Environment. ETA-TR-69, Boeing Computer Services, December 1987.

199. Worlton, J.: An Update on Competition in the Supercomputer Industry: Japan vs. USA. Report to the Federal Coordinating Council on Science, Engineering, and Technology (FCCSET), Washington, DC, February 1987.

200. Cyber 205 Workshop. Institute for Computational Studies, Fort Collins, CO, 1984.

201. Chen, S.S.: Advanced Large-Scale and High-Speed Multiprocessor System for Scientific Applications: Cray X-MP-4 Series. AIAA 7th Computational Fluid Dynamics Conference, Cincinnati, OH, July 1985.

202. The Cray-2 Computer System. Cray Research, Inc., Minneapolis, MN, 1985.

203. IEEE Scientific Supercomputer Subcommittee: U.S. Supercomputer Vulnerability. Draft, IEEE, July 1988.

204. The Cray Y-MP Computer System. Cray Research, Inc., Minneapolis, MN, February 1988.

205. Cray-3 press release, Cray Research, Inc., Minneapolis, MN, 1988.

206. ETA10 System Overview: ETA System V. Draft 3, ETA Systems PUB-1232, St. Paul, MN, April 1988.

207. Connection Machine Model CM-2 Technical Summary. Thinking Machines Technical Report HA87-4, April 1987.

208. Introduction to the Alliant. Short Course, NASA Ames Research Center, 1987.

209. Seitz, C.L.: The Cosmic Cube. *Communications of the ACM,* Vol. 28, No. 1, January 1985, pp. 22-23.

210. Intel iPSC/2 presentation. NASA Ames Research Center, January 1988.

211. AMETEK Computer Research Division Series 2010. AMETEK Computer Research Division, Monrovia, CA, 1987.

212. Hayes, J.P., Mudge, T., Stout, Q., Colley, S., and Palmer, J.: A Microprocessor-Based Hypercube Supercomputer. *IEEE Micro,* October 1986, pp. 6-17.

213. Pryor, D.V., Swisshelm, J.M., and Johnson, G.M.: Parallel Processing Applied to Navier-Stokes Simulation. Presented at SIAM 1986 National Meeting, Boston, 21-25 July 1986.

214. Pulliam, T.: Artificial Dissipation Models for the Euler Equations. *AIAA Journal,* Vol. 24, No. 12, December 1986, pp. 1931-1940.

215. Jameson, A., and Mavriplis, D.: Finite Volume Solutions of the Two-Dimensional Euler Equations on a Regular Triangular Mesh. AIAA Paper 85-0435, January 1985.

216. Harten, A., and Osher, S.: Uniform High-Order Accurate Nonoscillatory Schemes I. *SIAM Journal of Numerical Analysis,* Vol. 24, 1987, pp. 279-309.

217. Harten, A., Enquist, B., Osher, S., and Chakravarthy, S.: Uniform High-Order Accurate Nonoscillatory Schemes III. *Journal of Computational Physics,* Vol. 71, 1987, pp. 231-303.

218. Yee, H.C., and Harten, A.: Implicit TVD Schemes for Hyperbolic Conservation Laws in Curvilinear Coordinates. AIAA Paper 85-1513, July 1985.

219. Yee, H.C., Warming, R.F., and Harten, A.: Implicit Total Variation Diminishing (TVD) Schemes for Steady-State Calculations. *Journal of Computational Physics,* Vol. 57, 1985, pp. 327-360.

220. Yee. H.C.: Numerical Experiments with a Symmetric High-Resolution Shock Capturing Scheme. NASA TM-88325, June 1986.

221. Shieh, L.J.: private communication, June 1988.

222. Sorensen, R.L.: GRAPE: Two-Dimensional Grids about Airfoils and Other Shapes by the Use of Poisson's Equation. ARC-11379, 1980.

223. James, R.C., and James, G.: *Mathematics Dictionary.* 4th ed., Van Nostrand Reinhold, New York, 1976.

Tables

Table 4.1 Characteristics of Japanese Supercomputers

Vendor: Machine: (Year):	Fujitsu VP200 (1984)	Hitachi S810/20 (1983)	NEC SX-2 (1985)
CPUs	1	1	1
Peak Performance Rate (MFLOPS)	533	630	1300
Clock period (nanoseconds)	14/7 *	14	6
Main memory (Megawords)	8-32	4-32	16-32
Secondary memory (Megawords)	-	32-128	16-256

* vector/scalar speed

Table 4.2 Characteristics of Supercomputers Used in Present Research

Vendor: Machine: (Year):	Control Data Cyber 205 (1981)	Cray X-MP/1 or 2 (1982)	Cray X-MP/4 (1984)	Cray Cray-2 (1985)
CPUs	1	1 or 2	4	4
Peak Performance Rate (MFLOPS)	200/400	233/466	932	1952
Clock period (nanoseconds)	20.0	9.5/8.5		4.1
Main memory (Megawords)	4-16	1-16		256
Secondary memory (Megawords)	-	32-512		-
Memory access	80nsecs	4cycles		120/80nsecs
Memory banks		64		128

Table 4.3 Performance Characteristics of Next-Generation Supercomputers

Machine	Number of CPUs	Peak Speed GFLOPS
Cray Y-MP	8	2.3
Cray-3	16	16.0
ETA-10/G	8	5.1
NEC SX-3	4	20.0
Hitachi S-820/80	1	2.0
Fujitsu VP400	1	1.1

Table 5.1 Two-Dimensional Multigrid Speedups

Test Case	Speedup
Inviscid Subcritical	5.1
Inviscid Supercritical	2.1
Laminar Viscous	7.3
Turbulent Viscous	8.2

Table 5.2 Three-Dimensional Multigrid Speedups

Test Case	Speedup
Inviscid Subcritical	4.7
Inviscid Supercritical	2.5
Turbulent Viscous	4.4

Table 5.3 Two-Dimensional Embedding Speedups
(129 x 33 finest grid, 2 embeddings)

Test Case	Speedup
Inviscid Subcritical	16.4
Inviscid Supercritical	6.1
Turbulent Viscous	30.2

Table 5.4 Three-Dimensional Embedding Speedups
(65 x 17 x 17 finest grid, 1 embedding)

Test Case	Speedup
Inviscid Subcritical	7.0
Inviscid Supercritical	4.6
Turbulent Viscous	16.0

Table 5.5 Two-Dimensional Vectorization Speedups

Test Case	Complete Scheme		Coarse-Grid Scheme	
	Sequential	Parallel	Sequential	Parallel
Inviscid Subcritical	2.99	3.18	1.67	1.80
Inviscid Supercritical	2.89	3.04	1.73	1.87
Turbulent Viscous	2.72	1.86	1.73	1.86

Table 5.6 Three-Dimensional Vectorization Speedups

Test Case	Automatic	Explicit
Inviscid Subcritical	4.2	5.7
Inviscid Supercritical	3.1	4.9
Turbulent Viscous	2.6	3.6

Table 5.7 Two-Dimensional Macrotasked Parallel Coarse-Grid Scheme Performance

Machine	2 Processors		4 Processors	
	Speedup	Efficiency	Speedup	Efficiency
Cray X-MP	1.78	0.89	3.06	0.77

Table 5.8 Two-Dimensional Multitasked Basic Solver Performance

Machine	2 Processors		4 Processors	
	Speedup	Efficiency	Speedup	Efficiency
Cray X-MP with macrotasking	1.91	0.96	3.58	0.90
Cray X-MP with microtasking	1.93	0.97	3.78	0.95

Table 5.9 Two-Dimensional Macrotasked Multigridded Scheme Performance

Machine	2 Processors		4 Processors	
	Speedup	Efficiency	Speedup	Efficiency
Cray X-MP	1.87	0.94	3.30	0.83
Cray 2 (NAS) (2002)	1.69	0.84	2.23	0.56
Cray 2 (AFWL) (2014)	1.80	0.90	2.58	0.65

Table 5.10 Three-Dimensional Microtasked Multigridded Scheme Performance

	2 Processors		3 Processors		4 Processors	
Machine	Speedup	Efficiency	Speedup	Efficiency	Speedup	Efficiency
Cray X-MP	1.96	0.98	2.83	0.94	3.55	0.89

Table 5.11 Three-Dimensional Macrotasked Embedded Multigrid Scheme Performance

	2 Processors		3 Processors		4 Processors	
Machine	Speedup	Efficiency	Speedup	Efficiency	Speedup	Efficiency
Cray-2 (Navier)	1.73	0.86	2.22	0.74	2.67	0.67
Cray-2 (Stokes)	1.69	0.84	2.11	0.70	2.47	0.62

Table 5.12 Performance of Individual Task Groups, Three-Dimensional Macrotasked Multigridded Scheme (Sp = speedup, Ef = efficiency)

Machine	Length (secs)	2 Processors		3 Processors		4 Processors	
		Sp	Ef	Sp	Ef	Sp	Ef
(Best)							
Navier	5.82	1.90	0.95	2.69	0.90	3.54	0.88
Stokes	5.95	1.88	0.94	2.63	0.88	3.40	0.85
(Worst)							
Navier	0.28	1.50	0.75	1.41	0.47	1.88	0.47
Stokes	0.29	1.48	0.74	1.36	0.45	1.79	0.45

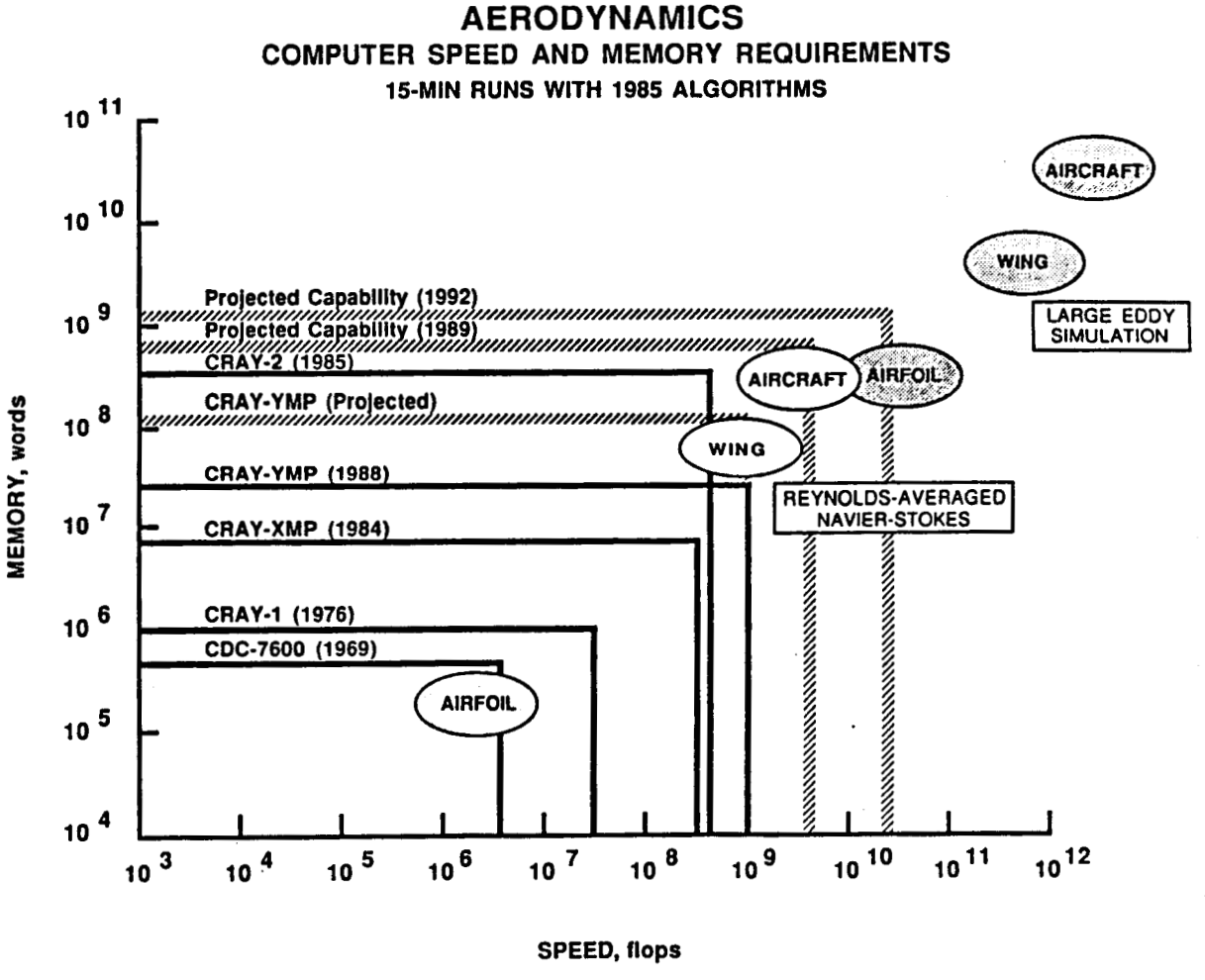
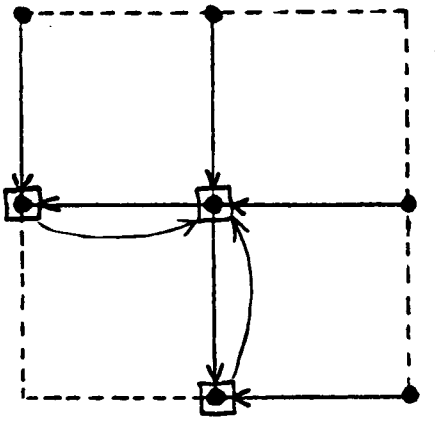


Figure 1.1. - Requirements for Computational Aerodynamics.

2-D



● - point used in predictor step

□ - predicted point

⟵ - predictor step

↷ - corrector step

(center point is being updated)

3-D

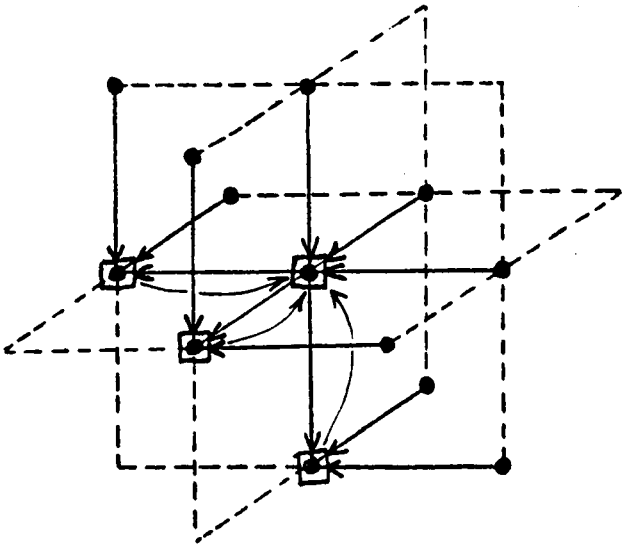
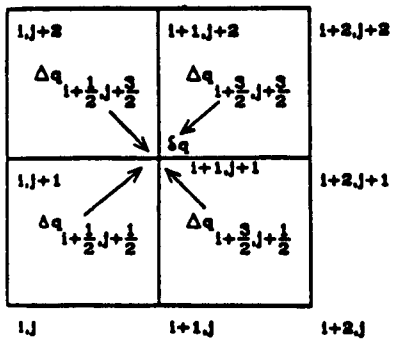
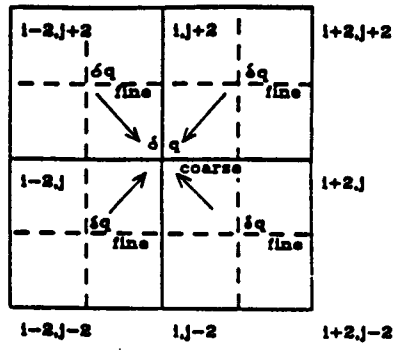


Figure 3.1 - MacCormack Finite-Difference Stencils

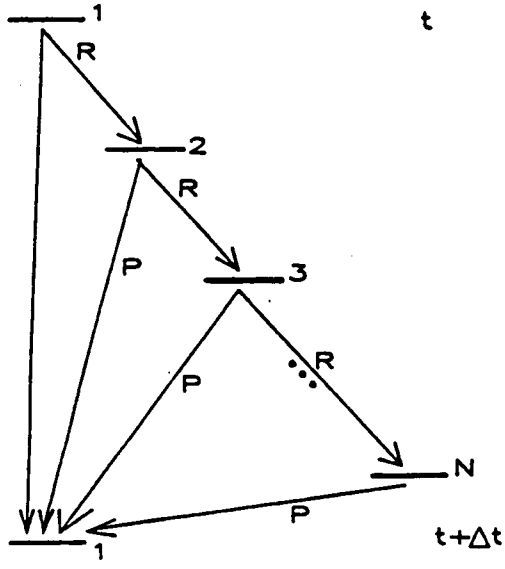


One-Step Lax-Wendroff Scheme on Fine Grid



Coarse-Grid Scheme

Figure 3.2 - Comparison of Fine- and Coarse-Grid Schemes



R = Restriction from Fine to Coarse Grid
P = Prolongation to Fine Grid

Figure 3.3 - Sequential Multigrid Information Flow

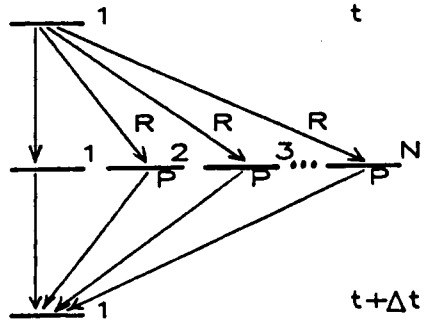


Figure 3.4 - Parallel Coarse-Grid Scheme Information Flow

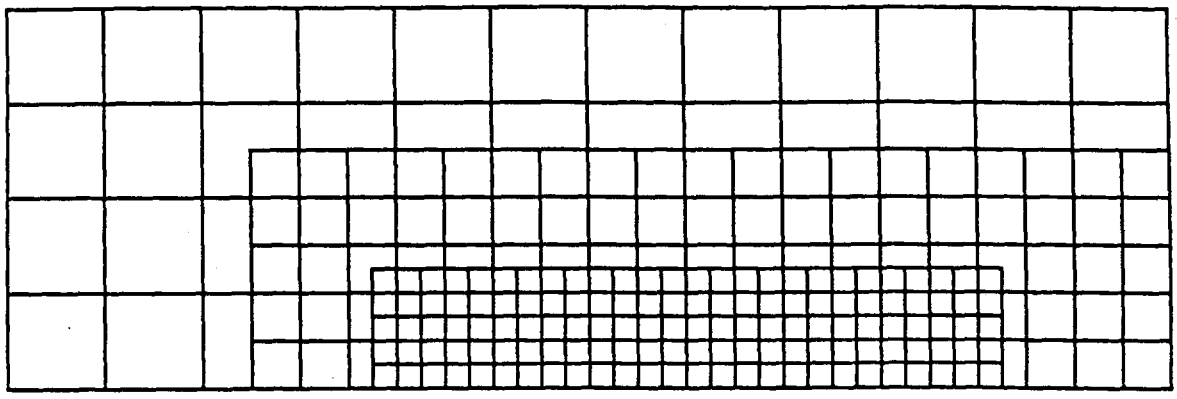


Figure 3.5. - Example of Grid Embedding for a Two-Dimensional Problem

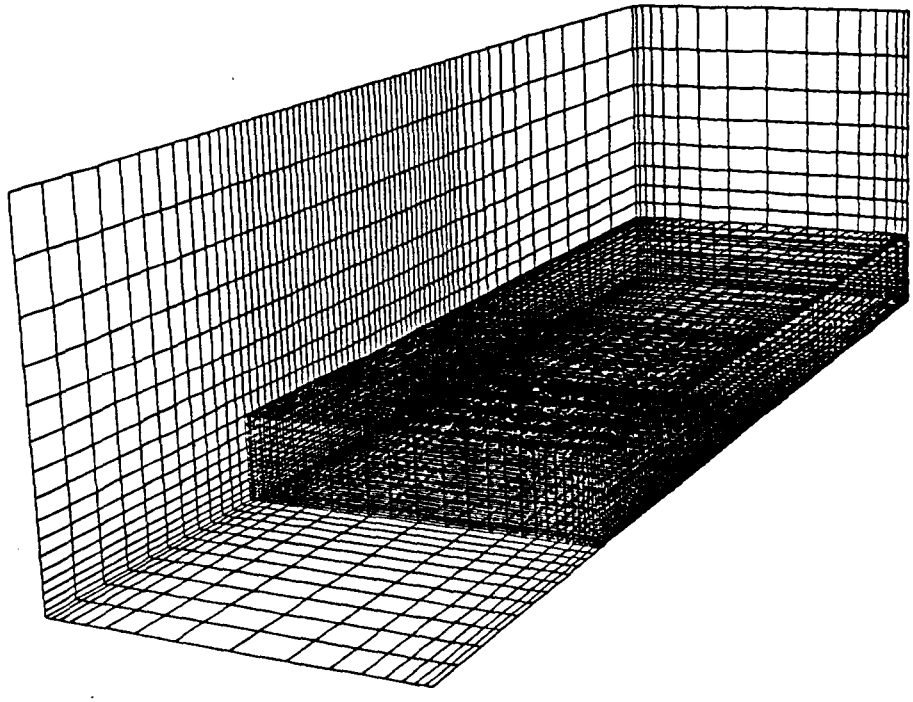


Figure 3.6. - Grid Embedding for the Three-Dimensional Model Problem

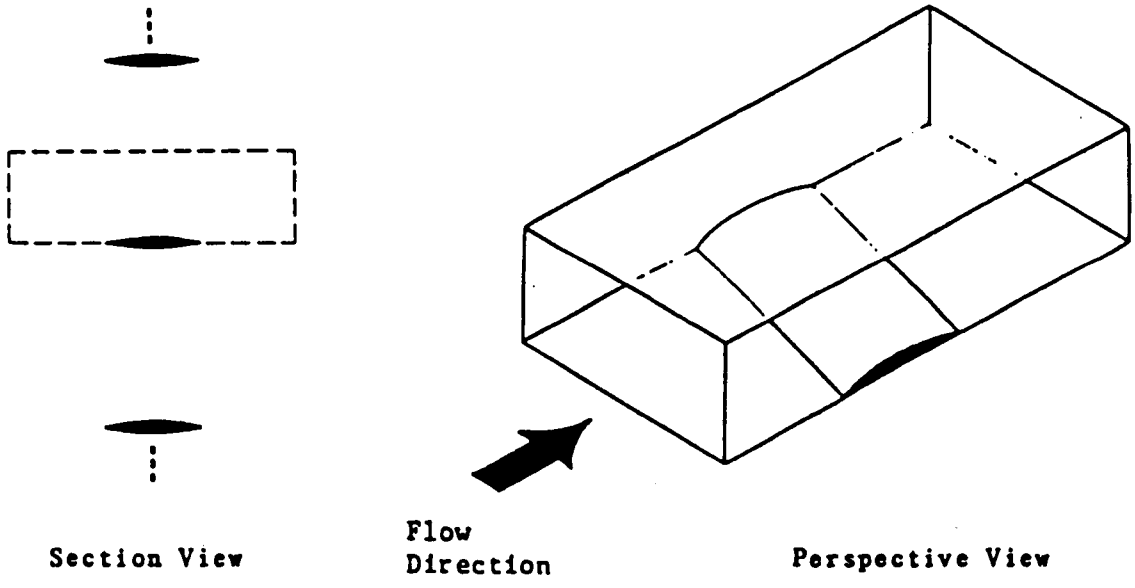


Figure 3.7 - Three-Dimensional Cascade

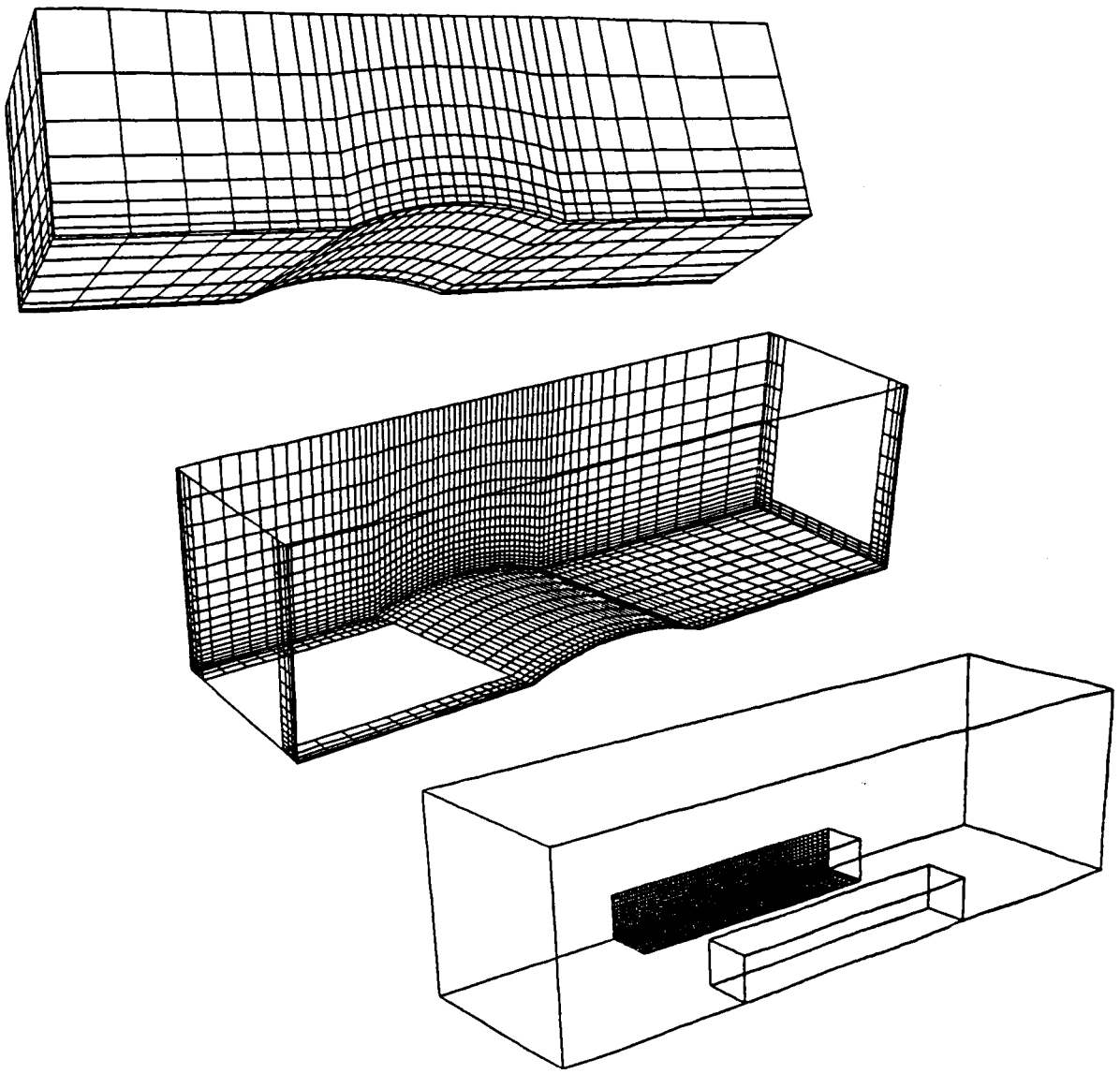


Figure 3.8 - Three-Dimensional Refinements: a. global coarse grid; b. first refinement; c. second refinement.

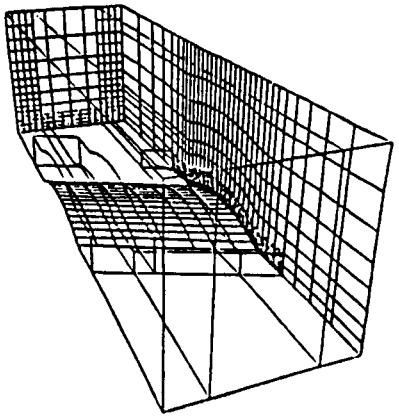
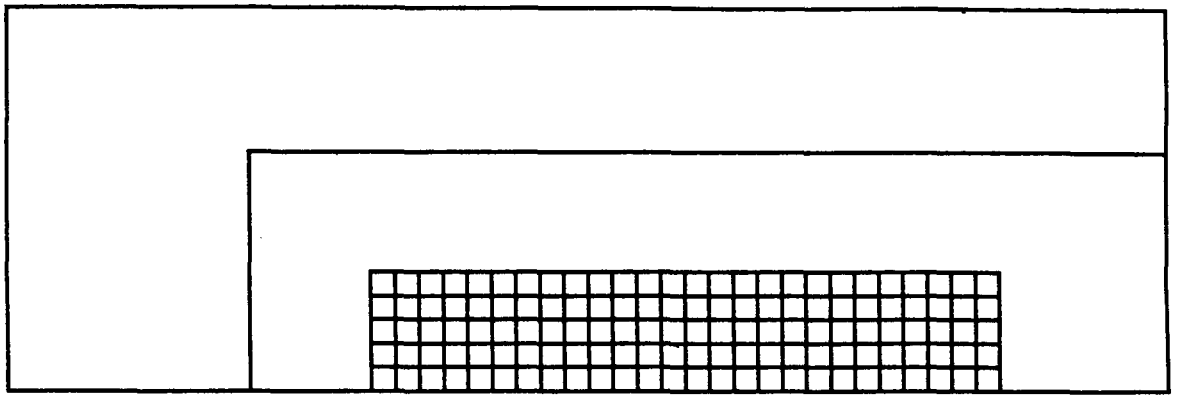
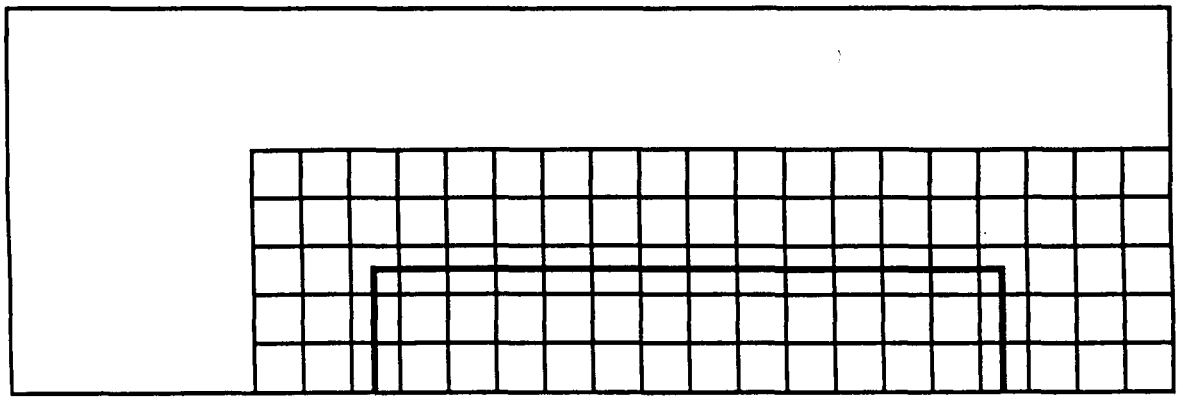


Figure 3.9 - Three-Dimensional Grid Refinements

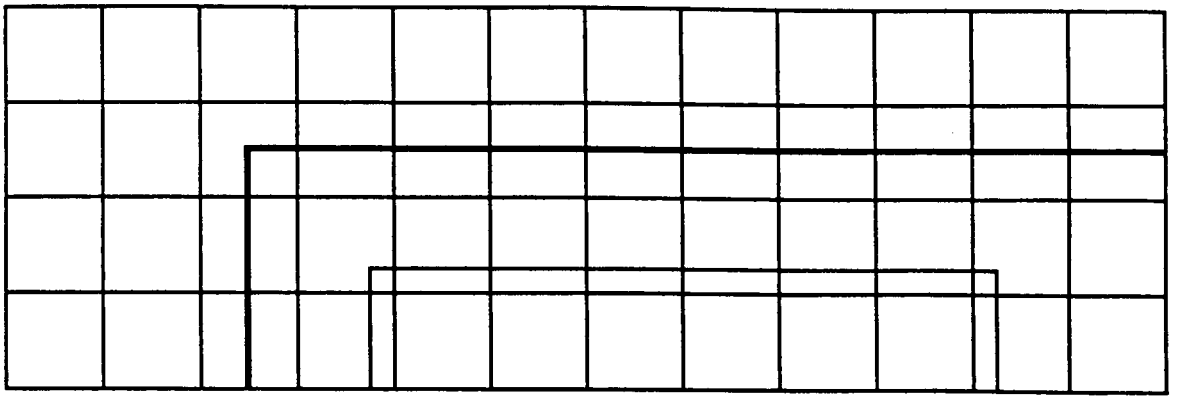


a. MacCormack Time Step

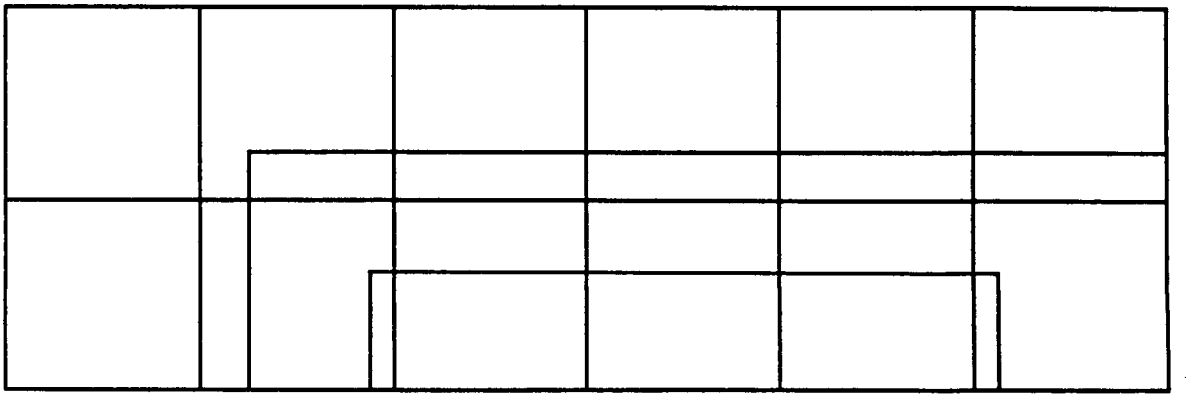


b. MacCormack Time Step at Grid Points Exterior to Bold Line
Multigrid Time Step at Points Interior to Bold LIne

Figure 3.10 - Implementation of Computational Scheme



c. MacCormack Time Step at Grid Points Exterior to Bold Line
Multigrid Time Step at Points Interior to Bold LIne



d. Multigrid Time Step at All Points Shown

Figure 3.10 - Implementation of Computational Scheme (cont'd)

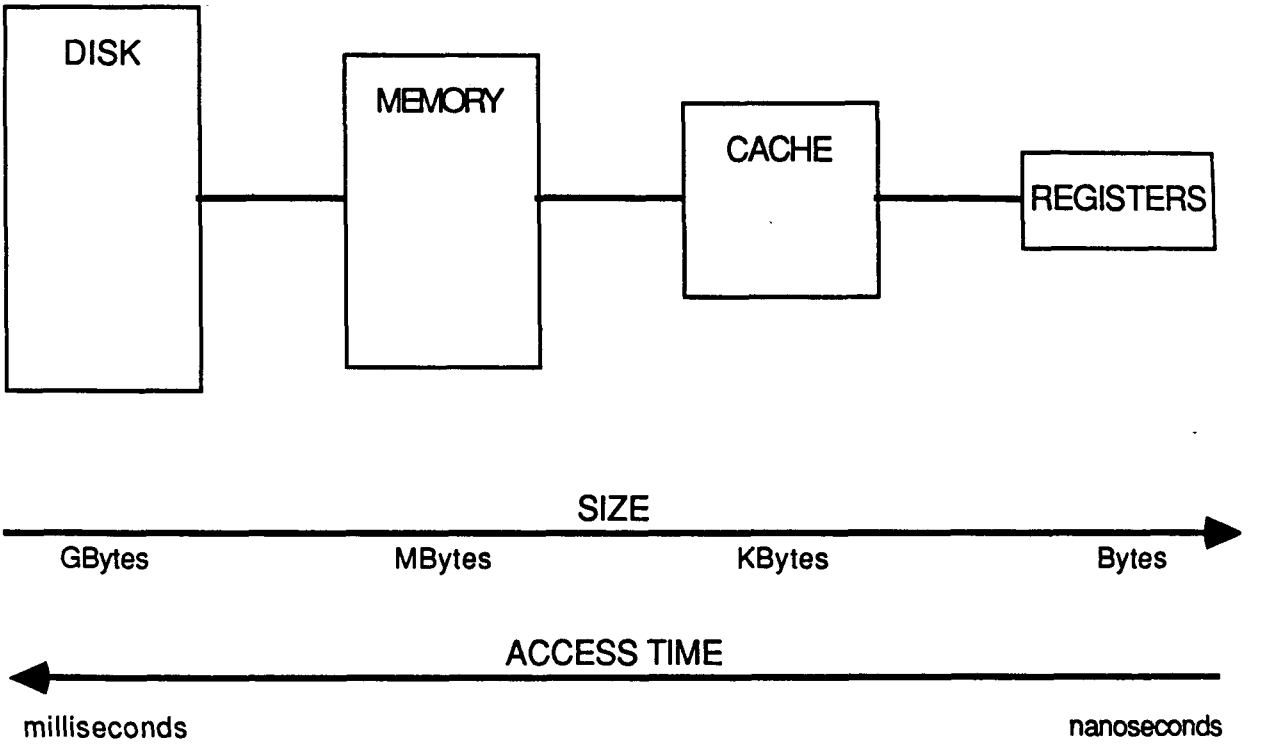
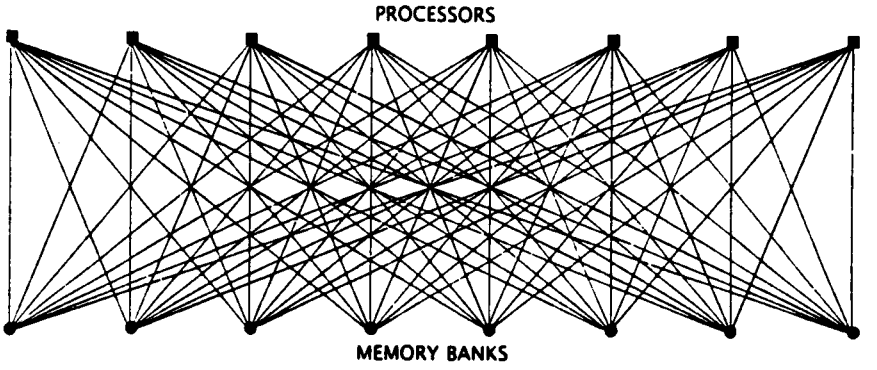
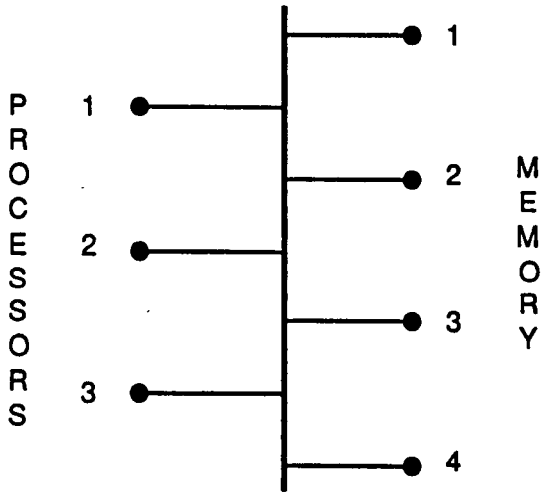


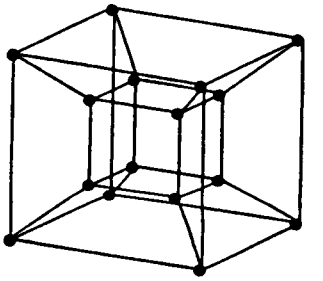
Figure 4.1 - Memory Level Hierarchy



Full Crossbar Switch



Bus



Hypercube (4-dimensional)

Figure 4.2 - Common Multiprocessor Interconnection Schemes

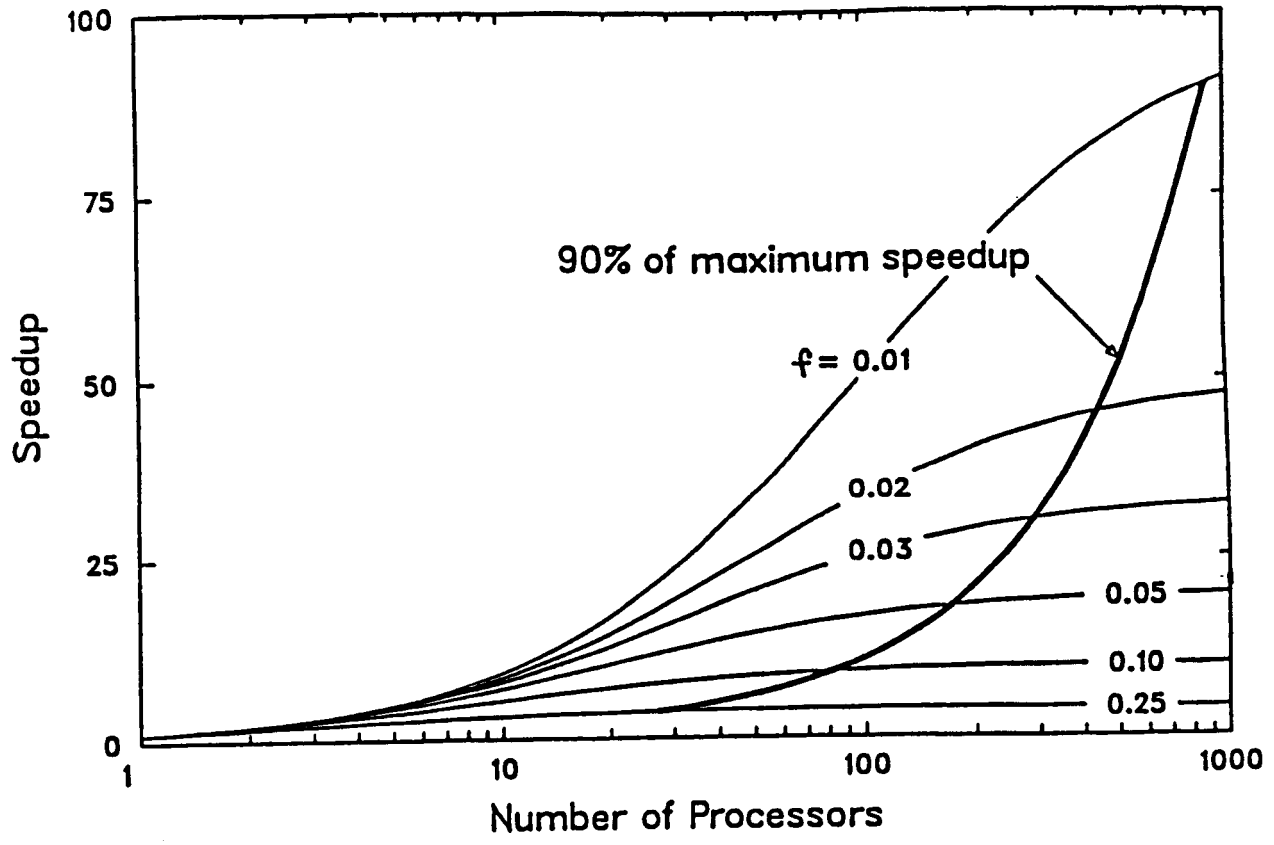


Figure 4.3 - Amdahl's Law (f = fraction of residual sequential code)

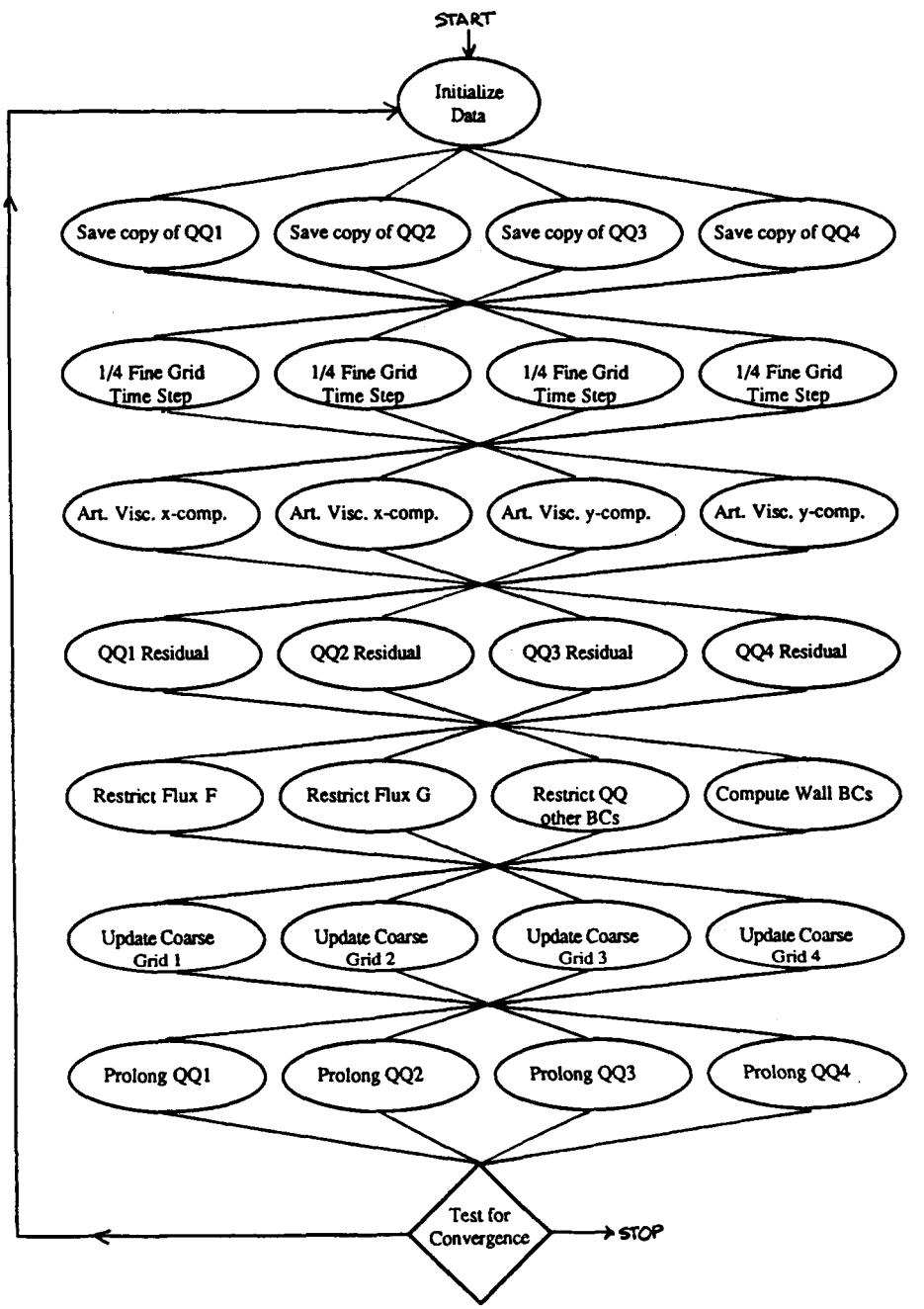


Figure 4.4 - Task Graph for Two-Dimensional Multigridded MacCormack Scheme

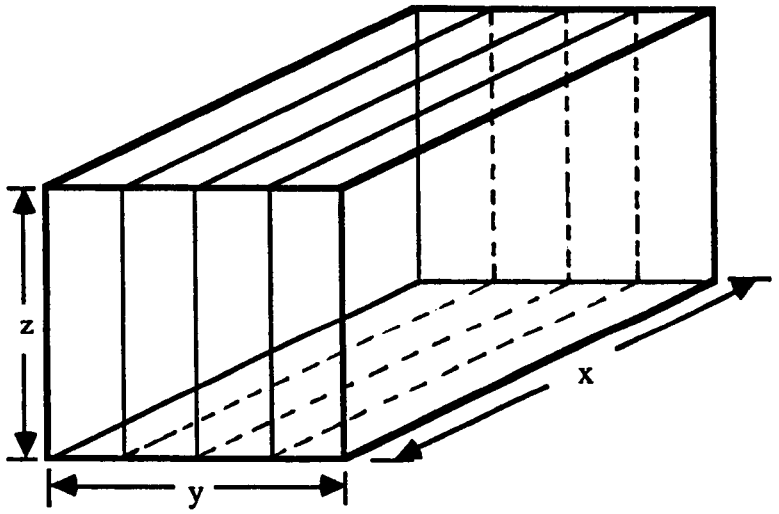


Figure 4.5 - Computational Domain with Planar Slices in the Lengthwise Direction

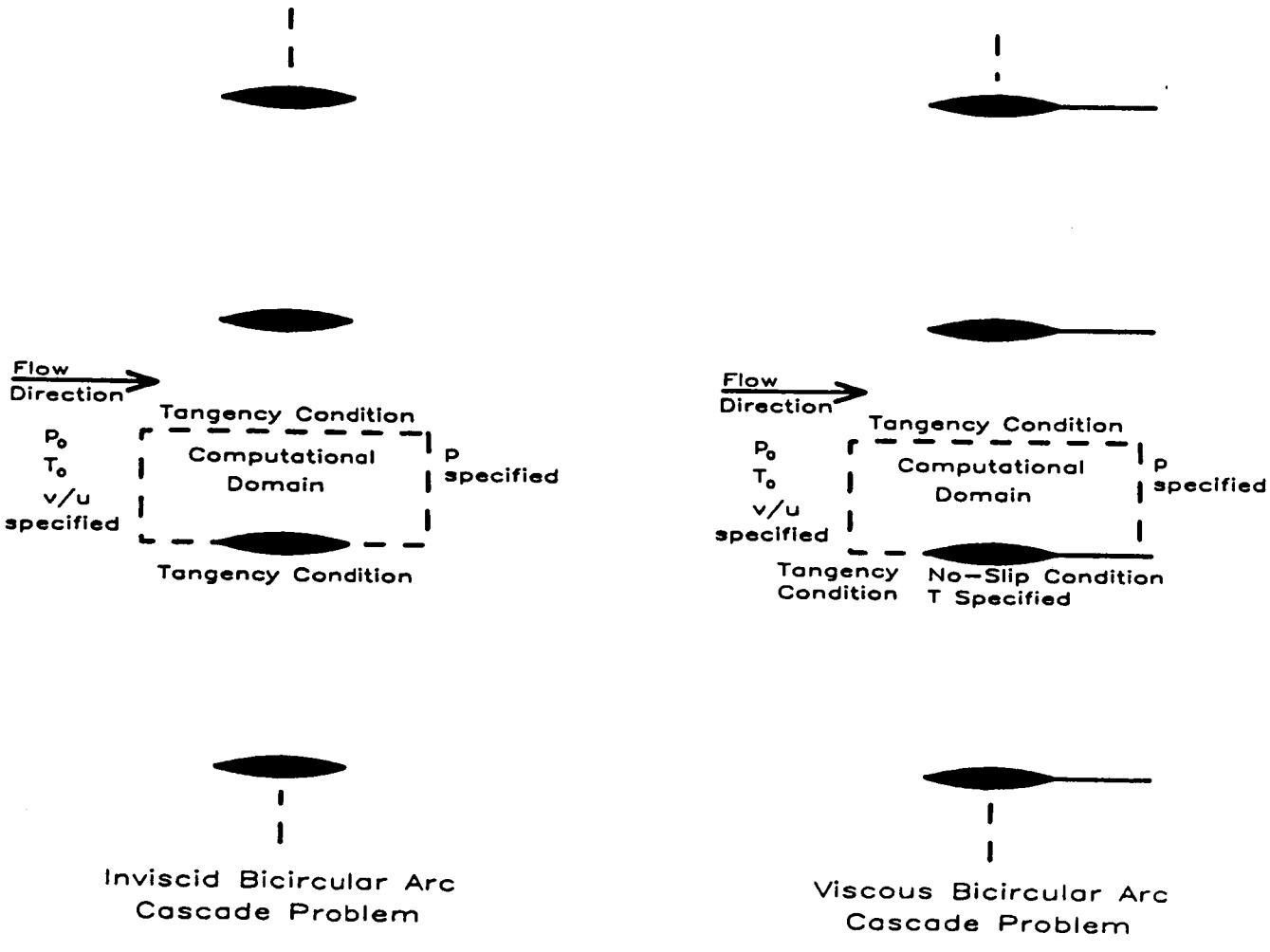


Figure 5.1. - Two-Dimensional Model Problems.

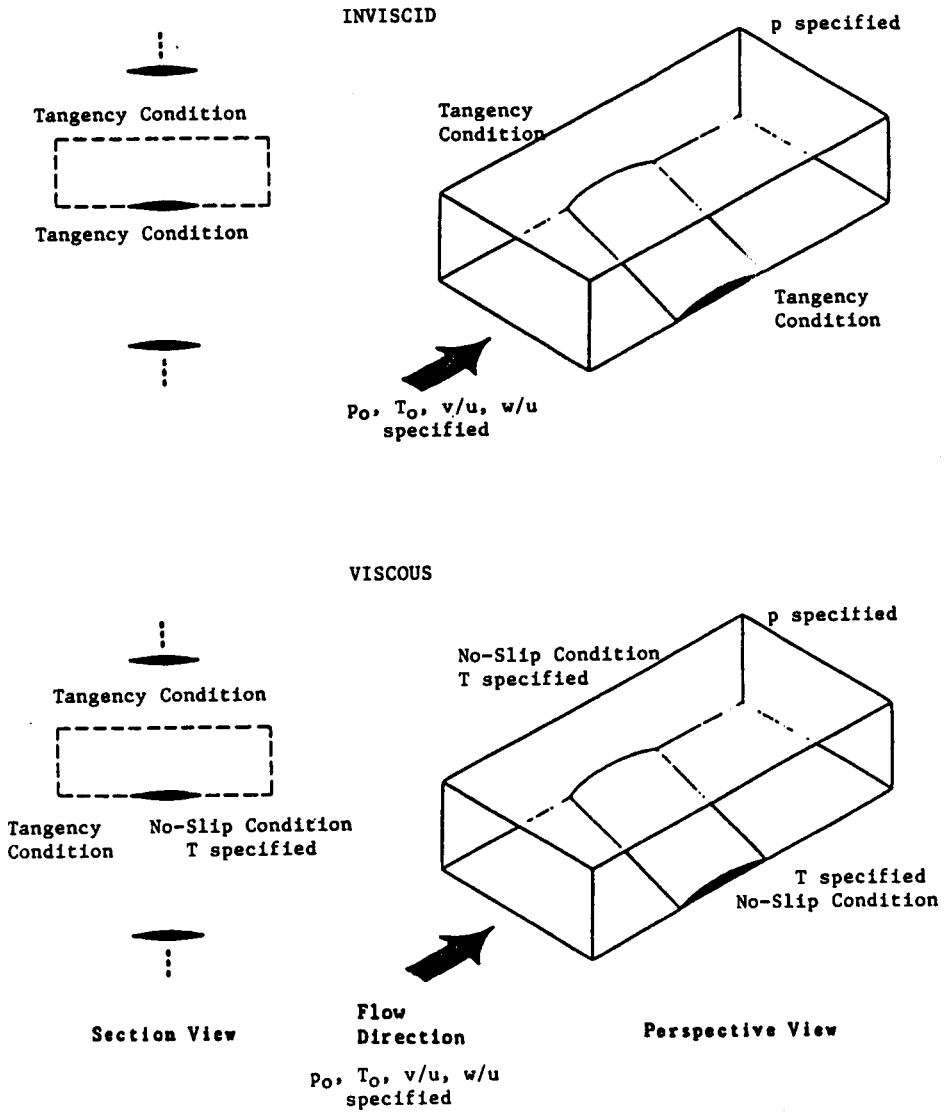


Figure 5.2. - Three-Dimensional Model Problems.

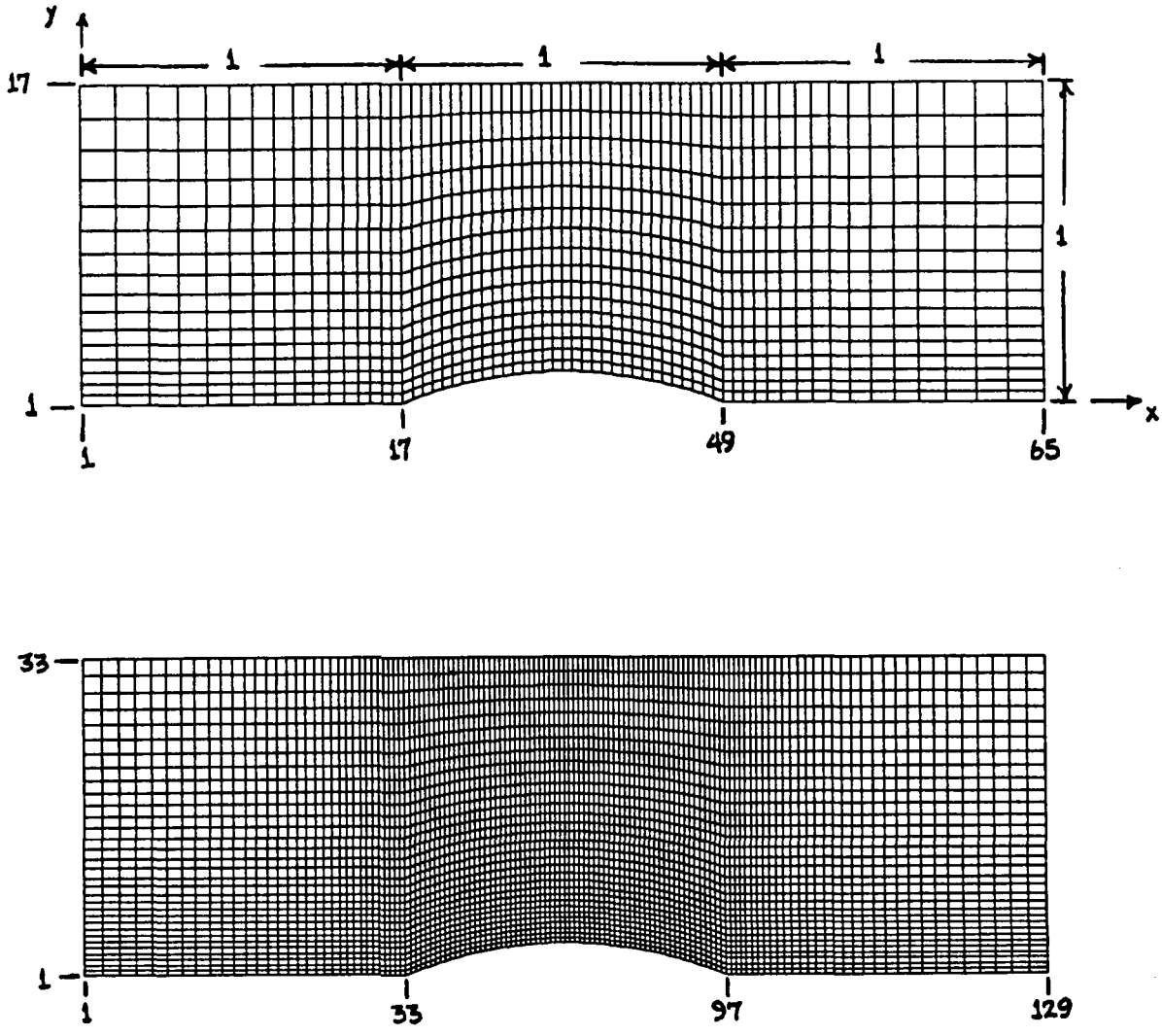


Figure 5.3. - Typical Two-Dimensional Grids.

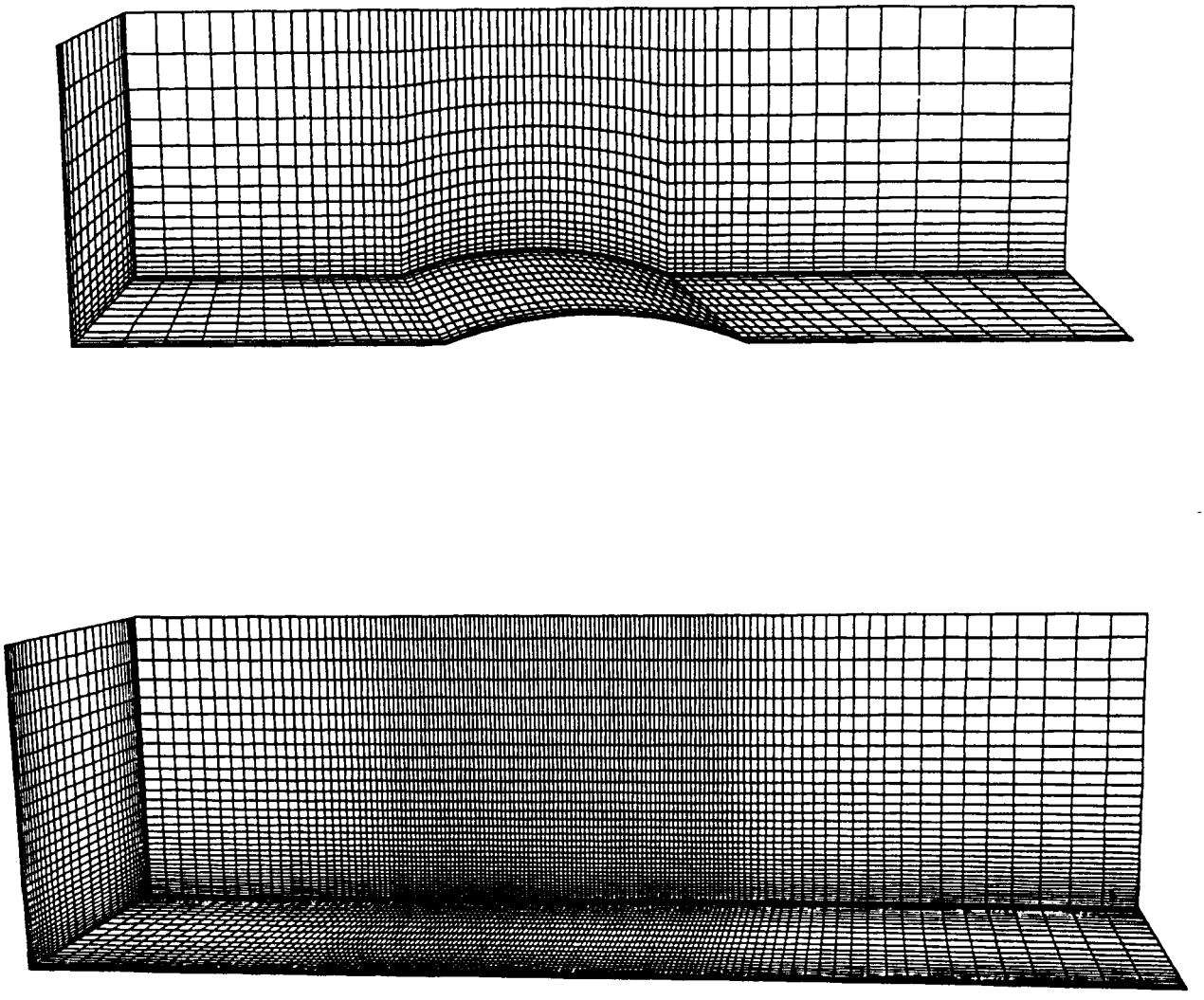


Figure 5.4. - Sample Three-Dimensional Grids for Inviscid and Viscous Cases.

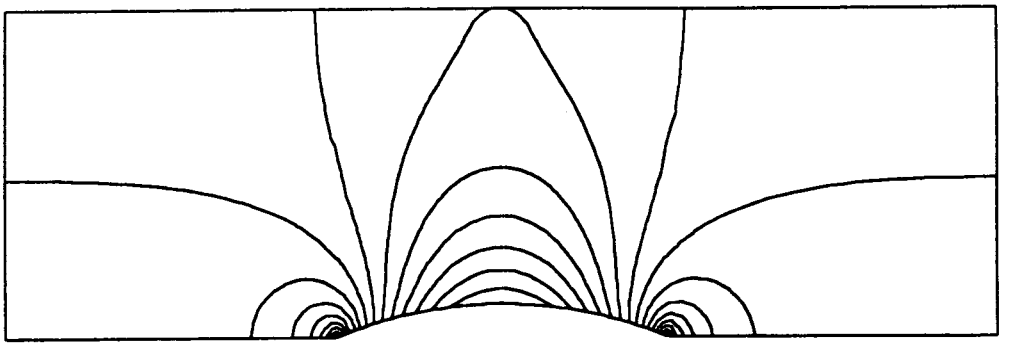


Figure 5.5. - Two-Dimensional Inviscid Subcritical Flow.
Isomach Contours.

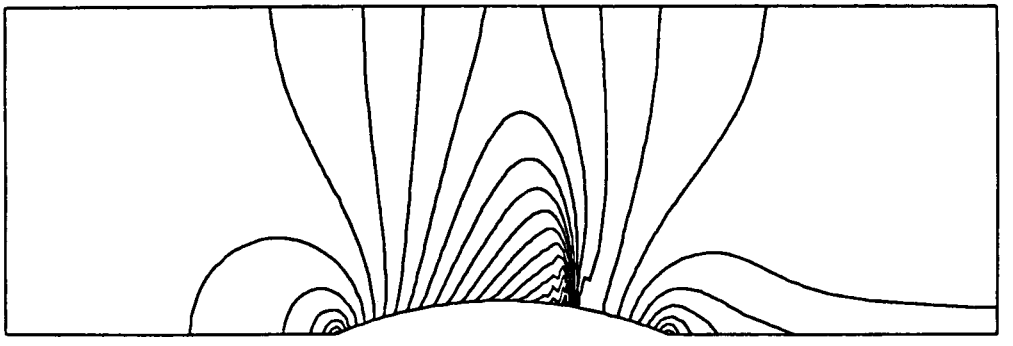


Figure 5.6. - Two-Dimensional Inviscid Supercritical Flow.
Isomach Contours.

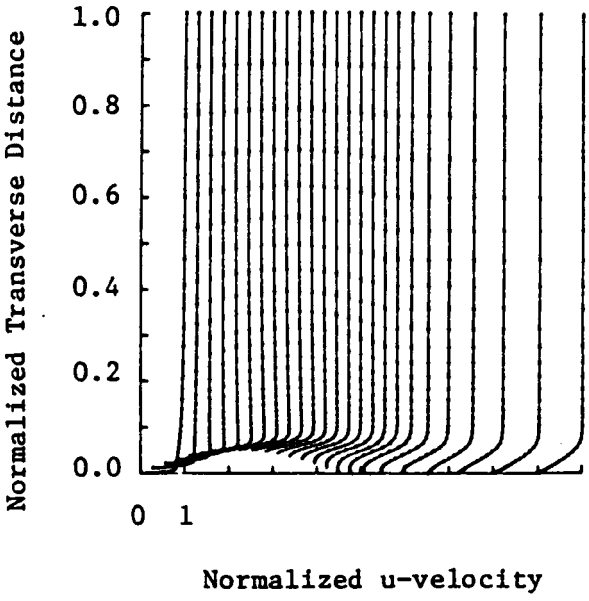


Figure 5.7. - Two-Dimensional Viscous Laminar Flow.
Velocity Profiles, Re = 3.4 x 10^4.

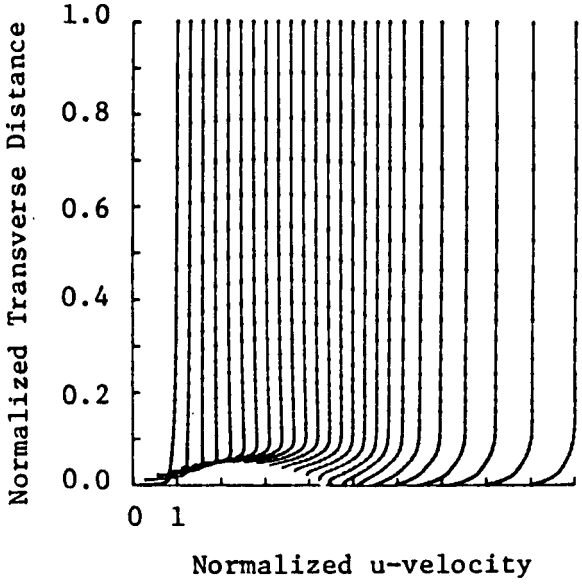


Figure 5.8. - Two-Dimensional Viscous Turbulent Flow.
Velocity Profiles, Re = 3.4 x 10^4.

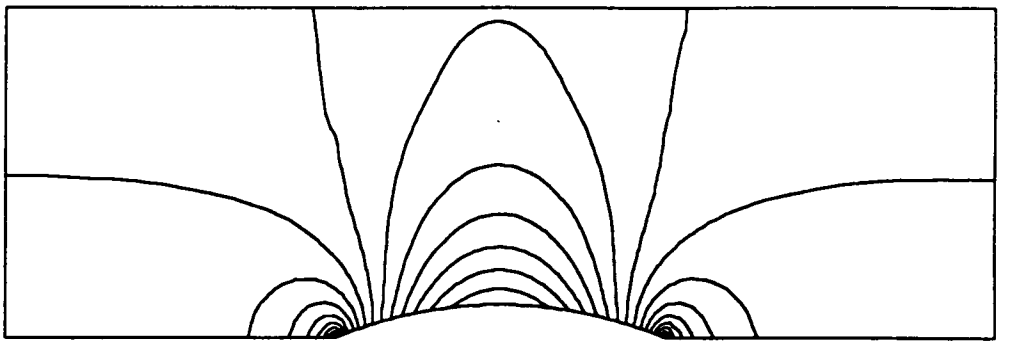


Figure 5.9. - Two-Dimensional Inviscid Subcritical Flow.
Isomach Contours. (Using Embedded Grids)

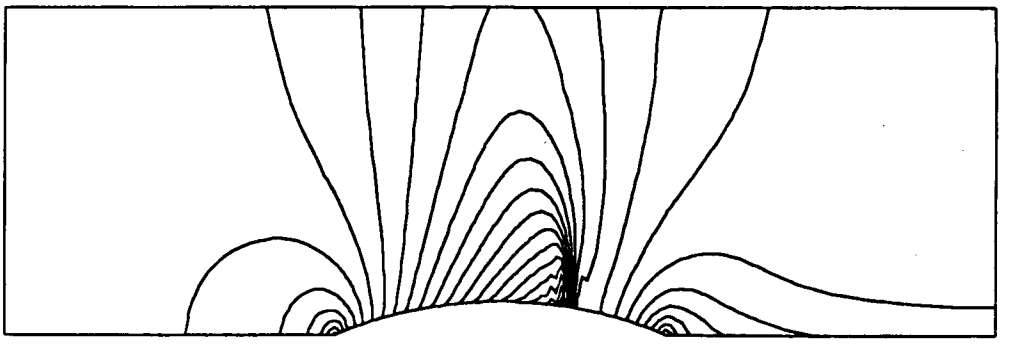


Figure 5.10. - Two-Dimensional Inviscid Supercritical Flow.
Isomach Contours. (Using Embedded Grids)

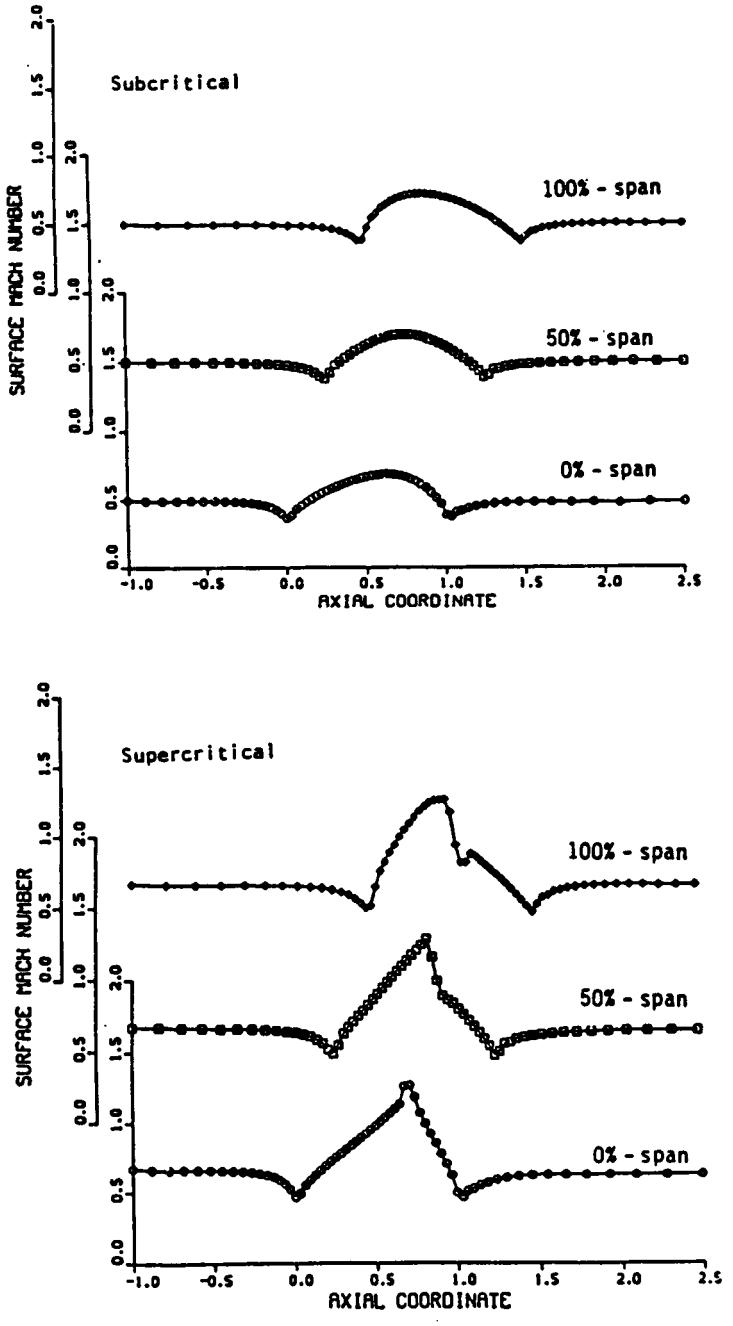


Figure 5.11. - Inviscid Three-Dimensional Flow. Sweep Angle = 26 Degrees.

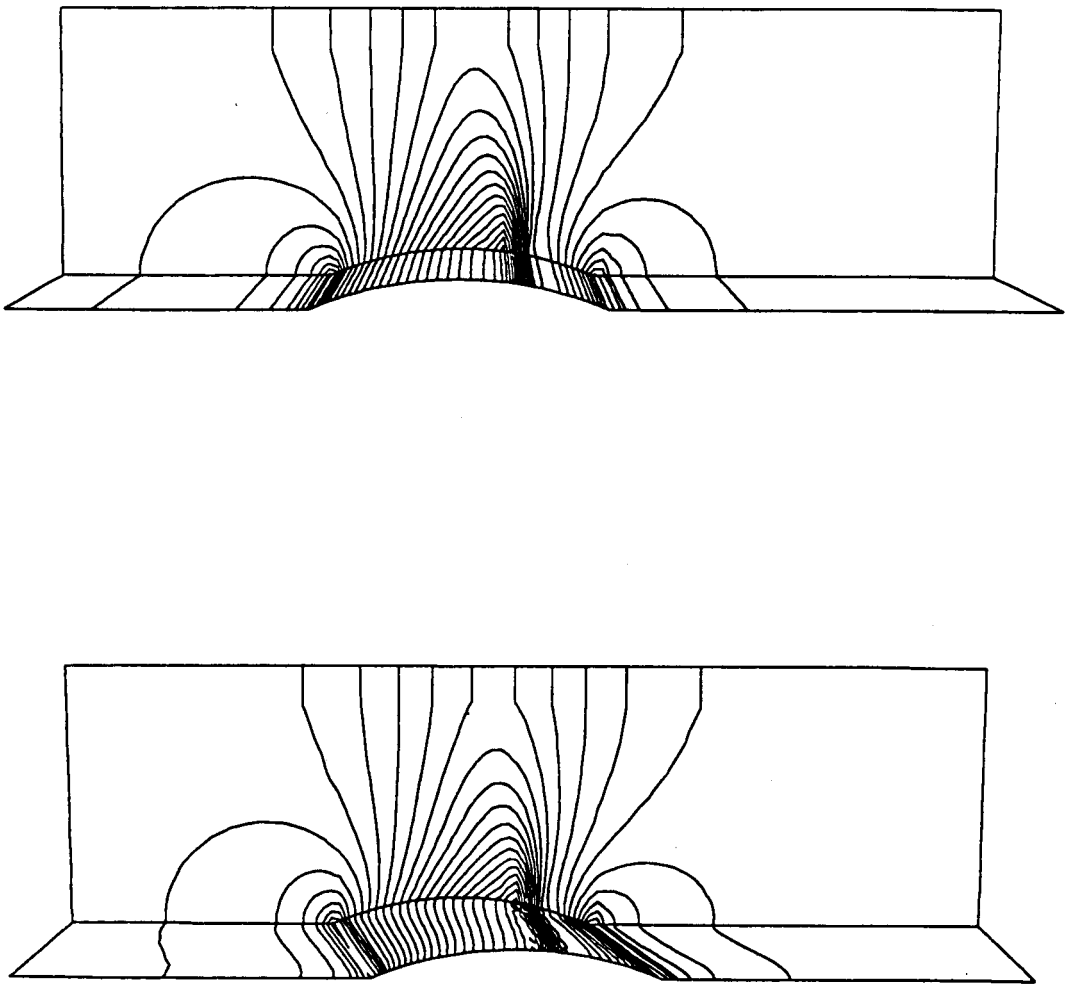
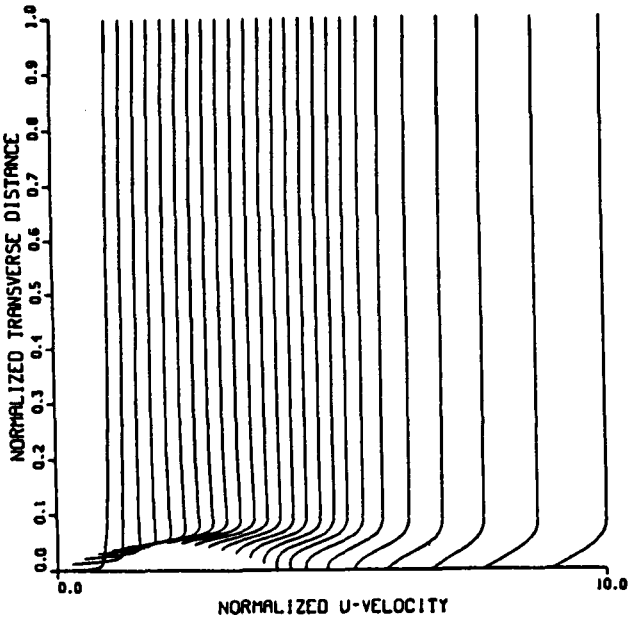
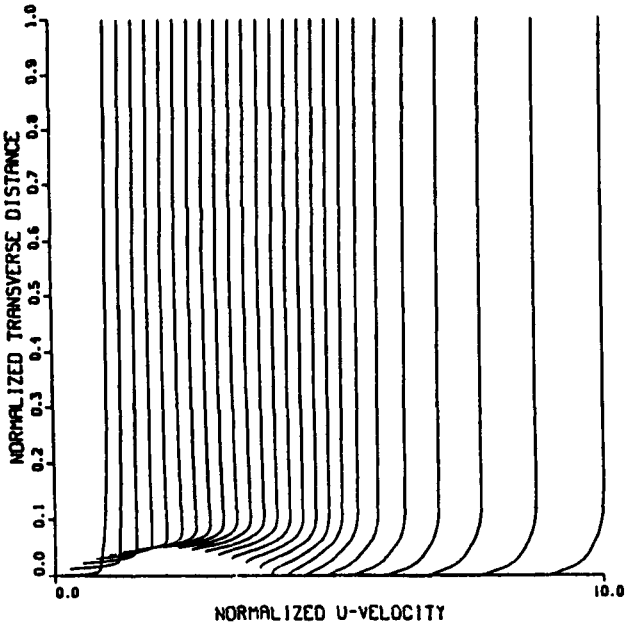


Figure 5.12. - Inviscid Supercritical Three-Dimensional Flow.
Surface Isomach Contours, Unswept and Swept Blade.



Laminar Flow.



Turbulent Flow.

Figure 5.13. - Viscous Three-Dimensional Flow. Velocity Profiles at 50% Span, $Re = 3.4 \times 10^4$, Sweep Angle = 26 Degrees.

APPENDICES

APPENDIX A. Generalised Coordinate Form, Equations of Motion

The equations of motion may be written in generalized coordinate form as

$$q'_\tau + F'_\xi + G'_\eta + H'_\zeta = 0$$

where, for the Euler equations,

$$F' = f', \qquad G' = g', \qquad H' = h'$$

for the thin-layer equations,

$$F' = f', \qquad G' = g', \qquad H' = h' - Re^{-1}S',$$

and for the full Reynolds-averaged Navier-Stokes equations,

$$F' = f' - Re^{-1}d', \qquad G' = g' - Re^{-1}r', \qquad H' = h' - Re^{-1}s',$$

The transformed variables (denoted by primes) represent the following vectors

$$q' = J^{-1}\begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ E \end{bmatrix} \qquad f' = J^{-1}\begin{bmatrix} \rho U \\ \rho uU+\xi_x p \\ \rho vU+\xi_y p \\ \rho wU+\xi_z p \\ (E+p)U-\xi_t p \end{bmatrix}$$

$$g' = J^{-1}\begin{bmatrix} \rho V \\ \rho uV+\eta_x p \\ \rho vV+\eta_y p \\ \rho wV+\eta_z p \\ (E+p)V-\eta_t p \end{bmatrix} \qquad h' = J^{-1}\begin{bmatrix} \rho W \\ \rho uW+\zeta_x p \\ \rho vW+\zeta_y p \\ \rho wW+\zeta_z p \\ (E+p)W-\zeta_t p \end{bmatrix}$$

where the contravariant velocities are defined as

$$U = \xi_t + \xi_x u + \xi_y v + \xi_z w$$

$$V = \eta_t + \eta_x u + \eta_y v + \eta_z w$$

$$W = \zeta_t + \zeta_x u + \zeta_y v + \zeta_z w$$

and the viscous terms are

$$d' = J^{-1}\begin{bmatrix} 0 \\ \xi_x\tau_{xx} + \xi_y\tau_{xy} + \xi_z\tau_{xz} \\ \xi_x\tau_{yx} + \xi_y\tau_{yy} + \xi_z\tau_{yz} \\ \xi_x\tau_{zx} + \xi_y\tau_{zy} + \xi_z\tau_{zz} \\ \xi_x\beta_x + \xi_y\beta_y + \xi_z\beta_z \end{bmatrix} \qquad r' = J^{-1}\begin{bmatrix} 0 \\ \eta_x\tau_{xx} + \eta_y\tau_{xy} + \eta_z\tau_{xz} \\ \eta_x\tau_{yx} + \eta_y\tau_{yy} + \eta_z\tau_{yz} \\ \eta_x\tau_{zx} + \eta_y\tau_{zy} + \eta_z\tau_{zz} \\ \eta_x\beta_x + \eta_y\beta_y + \eta_z\beta_z \end{bmatrix}$$

$$s' = J^{-1}\begin{bmatrix} 0 \\ \zeta_x\tau_{xx} + \zeta_y\tau_{xy} + \zeta_z\tau_{xz} \\ \zeta_x\tau_{yx} + \zeta_y\tau_{yy} + \zeta_z\tau_{yz} \\ \zeta_x\tau_{zx} + \zeta_y\tau_{zy} + \zeta_z\tau_{zz} \\ \zeta_x\beta_x + \zeta_y\beta_y + \zeta_z\beta_z \end{bmatrix}$$

Viscous terms for the thin-layer form of the equations, contained in S', are

$$S' = J^{-1}\begin{bmatrix} 0 \\ \mu(\zeta_x^2 + \zeta_y^2 + \zeta_z^2)u_\zeta + (\mu/3)(\zeta_x u_\zeta + \zeta_y v_\zeta + \zeta_z w_\zeta)\zeta_x \\ \mu(\zeta_x^2 + \zeta_y^2 + \zeta_z^2)v_\zeta + (\mu/3)(\zeta_x u_\zeta + \zeta_y v_\zeta + \zeta_z w_\zeta)\zeta_y \\ \mu(\zeta_x^2 + \zeta_y^2 + \zeta_z^2)w_\zeta + (\mu/3)(\zeta_x u_\zeta + \zeta_y v_\zeta + \zeta_z w_\zeta)\zeta_z \\ \left[(\zeta_x^2 + \zeta_y^2 + \zeta_z^2)\left[0.5\mu(u^2 + v^2 + w^2)_\zeta + \kappa Pr^{-1}(\gamma - 1)^{-1}(a^2)_\zeta\right]\right. \\ \left. \quad + \ (\mu/3)(\zeta_x u + \zeta_y v + \zeta_z w)(\zeta_x u_\zeta + \zeta_y v_\zeta + \zeta_z w_\zeta) \right] \end{bmatrix}$$

The terms τ and β in the viscous flux vectors are defined as in Chapter 2, with any derivatives previously written in Cartesian space now expanded in transformed space by the chain rule. For example, u_x is expanded as

$$u_x = \xi_x u_\xi + \eta_x u_\eta + \zeta_x u_\zeta$$

The Jacobian of the transformation may be written as

$$J^{-1} = x_\xi y_\eta z_\zeta + x_\zeta y_\xi z_\eta + x_\eta y_\zeta z_\xi - x_\xi y_\zeta z_\eta - x_\eta y_\xi z_\zeta - x_\zeta y_\eta z_\xi$$

and the metric terms are

$$\xi_x = J(y_\eta z_\zeta - y_\zeta z_\eta) \qquad \eta_x = J(z_\xi y_\zeta - y_\xi z_\zeta)$$

$$\xi_y = J(z_\eta x_\zeta - x_\eta z_\zeta) \qquad \eta_y = J(x_\xi z_\zeta - x_\zeta z_\xi)$$

$$\xi_z = J(x_\eta y_\zeta - y_\eta x_\zeta) \qquad \eta_z = J(y_\xi x_\zeta - x_\xi y_\zeta)$$

$$\zeta_x = J(y_\xi z_\eta - z_\xi y_\eta) \qquad \xi_t = - x_\tau \xi_x - y_\tau \xi_y - z_\tau \xi_z$$

$$\zeta_y = J(x_\eta z_\xi - x_\xi z_\eta) \qquad \eta_t = - x_\tau \eta_x - y_\tau \eta_y - z_\tau \eta_z$$

$$\zeta_z = J(x_\xi y_\eta - y_\xi x_\eta) \qquad \zeta_t = - x_\tau \zeta_x - y_\tau \zeta_y - z_\tau \zeta_z$$

For a temporally invariant grid (like the one used in the present algorithm), $t = \tau$, and the time-dependent derivatives in the metric terms vanish.

APPENDIX B. Derivation of the MacCormack Scheme from Taylor Series Expansion (in 1-D)

The MacCormack scheme can be derived from the Taylor series expansion

$$q(t + \Delta t) = q(t) + \Delta t q_t + \frac{\Delta t^2}{2} q_{tt} + \cdots \qquad (B.1)$$

Recalling that the equations of motion in one dimension may be represented by $q_t = - f_x$, defining $A = \frac{\delta f}{\delta q}$, and applying Euler's theorem on homogeneous functions [223] (which gives $f = \frac{\delta f}{\delta q} q = Aq$), it can then be shown that

$$q_{tt} = A f_{xx}$$

Substituting the equations for q_t and q_{tt} into (B.1) and truncating the expansion after the second order term yields the following:

$$q_i^{n+1} = q_i^n - \Delta t f_x + \frac{\Delta t^2}{2} A f_{xx} \qquad (B.2)$$

Discretizing the spatial derivatives in (B.2),

$$q_i^{n+1} = q_i^n - \Delta t \left[\frac{f_{i+1}^n - f_{i-1}^n}{2\Delta x} \right] + \frac{\Delta t^2}{2} \left[\frac{A \left(f_{i+1}^n - 2 f_i^n + f_{i-1}^n \right)}{\Delta x^2} \right]$$

Since $f_i^n = A q_i^n$ and $f_{i-1}^n = A q_{i-1}^n$, we can write

$$q_i^{n+1} = q_i^n - \frac{\Delta t}{2\Delta x}\left[(f_{i+1}^n - f_i^n) + \left[A\,(q_i^n - \frac{\Delta t}{\Delta x}(f_{i+1}^n - f_i^n))\right.\right. \qquad (B.3)$$

$$\left.\left. - A\,(q_{i-1}^n - \frac{\Delta t}{\Delta x}(f_i^n - f_{i-1}^n))\right]\right]$$

Defining

$$\bar{q}_i = q_i^n - \frac{\Delta t}{\Delta x}(f_{i+1}^n - f_i^n), \quad \bar{f}_i = f(\bar{q}_i), \quad \text{and} \quad \bar{f}_{i-1} = f(\bar{q}_{i-1}) \quad (B.4)$$

then from (B.4) and Euler's theorem

$$\bar{f}_i = A\bar{q}_i = A\left[q_i^n - \frac{\Delta t}{\Delta x}(f_{i+1}^n - f_i^n)\right] \qquad (B.5)$$

$$\bar{f}_{i-1} = A\bar{q}_{i-1} = A\left[q_{i-1}^n - \frac{\Delta t}{\Delta x}(f_i^n - f_{i-1}^n)\right] \qquad (B.6)$$

Substituting (B.5) and (B.6) into (B.3) yields

$$q_i^{n+1} = q_i^n - \frac{\Delta t}{2\Delta x}\left[(f_{i+1}^n - f_i^n) + (\bar{f}_i - \bar{f}_{i-1})\right]$$

Thus, the two-step MacCormack scheme, derived from a Taylor series expansion, can be written in forward predictor - backward corrector form as

$$\bar{q}_i = q_i^n - \frac{\Delta t}{\Delta x}(f_{i+1}^n - f_i^n)$$

$$q_i^{n+1} = q_i^n - \frac{\Delta t}{2\Delta x}\left[(f_{i+1}^n - f_i^n) + (\bar{f}_i - \bar{f}_{i-1})\right]$$

APPENDIX C. CFL Stability Limit Analysis (in 2-D)

The inviscid stability limit for the equations of motion may be derived from nonconservative form of the Euler equations,

$$q_t + Aq_x + Bq_y = 0 ,$$

where

$$q = \begin{bmatrix} \rho \\ u \\ v \\ p \end{bmatrix} \qquad A = \begin{bmatrix} u & \rho & 0 & 0 \\ 0 & u & 0 & 1/\rho \\ 0 & 0 & u & 0 \\ 0 & \rho c^2 & 0 & u \end{bmatrix} \qquad B = \begin{bmatrix} v & 0 & \rho & 0 \\ 0 & v & 0 & 0 \\ 0 & 0 & v & 1/\rho \\ 0 & 0 & \rho c^2 & v \end{bmatrix}$$

If we define C and θ such that

$$C = A\sin\alpha + B\sin\beta ,$$

$$\cos\theta = \frac{\sin\alpha}{\sqrt{\sin^2\alpha + \sin^2\beta}} , \qquad \text{and} \qquad \sin\theta = \frac{\cos\alpha}{\sqrt{\sin^2\alpha + \sin^2\beta}} ,$$

then C can be rewritten as

$$C = \sqrt{\sin^2\alpha + \sin^2\beta} \begin{bmatrix} u' & \rho\cos\theta & \rho\sin\theta & 0 \\ 0 & u' & 0 & (1/\rho)\cos\theta \\ 0 & 0 & u' & (1/\rho)\sin\theta \\ 0 & \rho c^2\cos\theta & \rho c^2\sin\theta & u' \end{bmatrix}$$

where $u' = u\cos\theta + v\cos\theta$ and c is the local speed of sound. Transforming the Euler equations to generalized coordinate form,

$$q'_t + Aq'_\xi + Bq'_\eta = 0 ,$$

the transformed matrix, $\bar{C}$, is then defined as

$$\bar{C} = \Delta t \sqrt{(\sin\alpha/\Delta\xi)^2 + (\sin\beta/\Delta\eta)^2} \begin{bmatrix} \bar{u}' & \rho k_1 & \rho k_2 & 0 \\ 0 & \bar{u}' & 0 & (1/\rho)\, k_1 \\ 0 & 0 & \bar{u}' & (1/\rho)\, k_2 \\ 0 & \rho c^2 k_1 & \rho c^2 k_2 & \bar{u}' \end{bmatrix}$$

where $\bar{u}'$ is a function of the contravariant velocities (which are defined in Appendix B)

$$\bar{u}' = U\cos\theta + V\sin\theta$$

and

$$k_1 = \xi_x \cos\theta + \eta_x \sin\theta$$

$$k_2 = \xi_y \cos\theta + \eta_y \sin\theta$$

The eigenvalues of the matrix contained in the definition of $\bar{C}$ are

$$\mu_1 = \bar{u}', \quad \mu_2 = \bar{u}', \quad \mu_3 = \bar{u}' + c\sqrt{k_1^2 + k_2^2}, \quad \mu_4 = \bar{u}' - c\sqrt{k_1^2 + k_2^2}$$

so that, for $\Delta\xi = \Delta\eta = 1$, the four eigenvalues of $\bar{C}$ can be written as

$$\lambda = \Delta t \sqrt{\sin^2\alpha + \sin^2\beta} \begin{cases} \bar{u}' \\ \bar{u}' \\ \bar{u}' + c\sqrt{k_1^2 + k_2^2} \\ \bar{u}' - c\sqrt{k_1^2 + k_2^2} \end{cases}$$

The stability condition requires that the spectral radius λ be less than 1, or that

$$\Delta t \leq \left[|U| + |V| + c\sqrt{(\xi_x^2 + \xi_y^2) + 2(\xi_x\eta_x + \xi_y\eta_y) + (\eta_x^2 + \eta_y^2)} \right]^{-1}$$

The three-dimensional generalized-coordinate form of the stability limit, by a similar derivation, may be written

$$\Delta t \leq \Bigg[|U| + |V| + |W| + c\Big[(\xi_x^2 + \xi_y^2 + \xi_z^2) + (\eta_x^2 + \eta_y^2 + \eta_z^2) + (\zeta_x^2 + \zeta_y^2 + \zeta_z^2) + 2(\xi_x(\eta_x + \zeta_x) + \eta_x\zeta_x + \xi_y(\eta_x + \zeta_y) + \eta_y\zeta_y + \xi_z(\eta_z\zeta_z) + \eta_z\zeta_z)\Big]^{1/2} \Bigg]^{-1}$$

APPENDIX D. Generalized Coordinate Form, Artificial Dissipation

The artificial damping used for supercritical flows may be expressed in generalized coordinates as

$$\Delta q_\nu = \frac{\Delta t \nu}{J}\left[(|\xi_x \rho_\xi| q_\xi)_\xi + (|\eta_y \rho_\eta| q_\eta)_\eta + (|\zeta_z \rho_\zeta| q_\zeta)_\zeta\right]$$

The artificial dissipation term for very fine grids in three-dimensional computations may be expressed in generalized coordinates as

$$\Delta q_\mu = \Delta t \mu \left[C_\xi q_{\xi\xi\xi\xi} + C_\eta q_{\eta\eta\eta\eta} + C_\zeta q_{\zeta\zeta\zeta\zeta}\right] \qquad (D.1)$$

We note that the terms C_ξ, C_η, and C_ζ in (D.1) may be interpreted as corresponding to arclengths along which the artificial dissipation acts. These lengths are defined to be

$$C_\xi = \sqrt{x_\xi^2 + y_\xi^2 + z_\xi^2}$$

$$C_\eta = \sqrt{x_\eta^2 + y_\eta^2 + z_\eta^2}$$

$$C_\zeta = \sqrt{x_\zeta^2 + y_\zeta^2 + z_\zeta^2}$$

APPENDIX E. SCHEDULE, CoFortran, Microtasking, and Macrotasking

SCHEDULE: Fortran extensions [171]

Static Dependency Graph

CALL DEP - assigns data dependencies for each task
CALL PUTQ - puts each task in the queue for execution

Dynamic Spawning

CALL NXTAG - automatically assigns data dependencies and forces parent process to wait for task completion
CALL SPAWN - puts the process in the execution queue

Low-Level Synchronization

CALL LOCKON - protects code section from concurrent execution
CALL LOCKOFF - marks the end of protected code

CoFortran: Fortran extensions [170]

Concurrency Statements

CoProcess - first statement of concurrent subroutine
CoBegin - begin parallel block of code
CoEnd - end parallel block of code

Synchronization Statements

CoInit - creates CoProcess from the main program
CoStop - signals end of parallel code management in main program
CoStart - starts parallel execution of CoProcess
CoWait - synchronization point in parallel execution

Data Passing

SHARE - share a common block among processes
RELEASE - unshare the common block

Microtasking: Preprocessor directives [152]

CMIC$ MICRO	- denotes subroutine to be microtasked
CMIC$ CONTINUE	- continue parallel execution at next subroutine level
CMIC$ DO GLOBAL	- starts a parallel task for each pass through the following DO-loop
CMIC$ PROCESS	- begins control structure (with ALSO and END PROCESS), starting a parallel task
CMIC$ ALSO PROCESS	- starts next parallel task (ends previous task)
CMIC$ END PROCESS	- end of parallel task control structure
CMIC$ STOP ALL PROCESS	- exit PROCESS/DO GLOBAL

Macrotasking: Fortran extensions [195]

CALL TSKTUNE - enables tuning of processor handling

Task Creation

CALL TSKSTART - starts parallel tasks
CALL TSKWAIT - synchronizes parallel tasks

Message Passing

CALL EVPOST - posts an event
CALL EVWAIT - waits for an event to be posted
CALL EVCLEAR - clears (or unposts) an event

Barrier Synchronizaton

CALL BARASGN - creates a barrier for tasks
CALL BARSYNC - registers arrival of task at barrier

Low-Level Synchronization

CALL LOCKON - protects critical region
CALL LOCKOFF - end protection of critical region

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM-102188		

4. Title and Subtitle

Development of a Navier-Stokes Algorithm for Parallel-Processing Supercomputers

5. Report Date

May 1989

6. Performing Organization Code

7. Author(s)

Julie M. Swisshelm

8. Performing Organization Report No.

A-89121

10. Work Unit No.

536-01-11

9. Performing Organization Name and Address

Ames Research Center
Moffett Field, CA 94035

11. Contract or Grant No.

13. Type of Report and Period Covered

Technical Memorandum

12. Sponsoring Agency Name and Address

National Aeronautics and Space Administration
Washington, DC 20546-0001

14. Sponsoring Agency Code

15. Supplementary Notes

This report was originally prepared as a Ph.D. dissertation.

Point of Contact: Julie M. Swisshelm, Ames Research Center, M/S 258-5
Moffett Field, CA 94035 (415) 694-4430 or FTS 464-4430

16. Abstract

An explicit flow solver, applicable to the hierarchy of model equations ranging from Euler to full Navier-Stokes, is combined with several techniques designed to reduce computational expense. The computational domain consists of local grid refinements embedded in a global coarse mesh, where the locations of these refinements are defined by the physics of the flow. Flow characteristics are also used to determine which set of model equations is appropriate for solution in each region, thereby reducing not only the number of grid points at which the solution must be obtained, but also the computational effort required to get that solution. Acceleration to steady-state is achieved by applying multigrid on each of the subgrids, regardless of the particular model equations being solved. Since each of these components is explicit, advantage can readily be taken of the vector- and parallel-processing capabilities of machines such as the Cray X-MP and Cray-2.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Navier-Stokes equations Parallel processing Computational fluid dynamics	Unclassified-Unlimited Subject Category – 02

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
Unclassified	Unclassified	172	A08