NASA Technical Memorandum 86675

Multiple Grid Problems on Concurrent Processing Computers

D. Scott Eberhardt and Donald Baganoff

February 1986

NASA
National Aeronautics and
Space Administration

NASA Technical Memorandum 86675

Multiple Grid Problems on Concurrent Processing Computers

D. Scott Eberhardt, Ames Research Center, Moffett Field, California
Donald Baganoff, Stanford University, Stanford, California

February 1986

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California 94035

TABLE OF CONTENTS

MULTIPLE GRID PROBLEMS ON CONCURRENT-PROCESSING COMPUTERS

D. Scott Eberhardt and Donald Baganoff*

Ames Research Center

SUMMARY

Three computer codes have been studied which make use of concurrent-processing computer architectures in computational fluid dynamics (CFD). The purpose of the study is to gain experience using a multiple-instruction/multiple-data (MIMD) computer for CFD applications.

MIMD architectures are being suggested as the most likely candidate for the next generation of supercomputers. In order to make efficient use of the multiple-processor architectures, the user will have to learn the skills of parallel programming. The three parallel codes written for this study, and tested on a two-processor MIMD facility at NASA Ames Research Center, are suggested for efficient parallel computations.

The first code studied is a well-known program which makes use of the Beam and Warming, implicit, approximate-factored algorithm. This study demonstrates the parallelism found in a well-known scheme, which achieved speedups exceeding 1.9 on the two-processor MIMD test facility.

The second code studied made use of an embedded grid scheme which is used to solve problems having complex geometries, The particular application for this section considered an airfoil/flap geometry in an incompressible flow. The scheme eliminates some of the inherent difficulties found in adapting approximate factorization techniques onto MIMD machines and allows the use of chaotic relaxation and asynchronous iteration techniques.

The third code studied is an application of overset grids to a supersonic blunt-body problem. The code addresses the difficulties encountered when embedded grids are used on a compressible, and therefore nonlinear, problem. The complex numerical boundary system associated with overset grids is discussed and several boundary schemes are suggested. A boundary scheme based on the method of characteristics achieved the best results.

A brief review of computer architectures is also presented and the particular machine used in this study is described.

*Department of Aeronautics and Astronautics, Stanford University, Stanford, CA 94305.

1.0 INTRODUCTION

Computer technology has progressed dramatically in the last few decades. Much of this progress has been achieved through increased parallelism within the computer system. The parallelism results from allowing functional units and modules to operate simultaneously. The organization of the functional units, modules, and interconnections of a computer system is described as the computer's architecture. All current machines, ranging from the Intel 8086 microprocessor to the CRAY-1 and CYBER 205, have some amount of parallelism designed into their architecture.

Recent designs for large-scale scientific computers use parallel architectures with multiple processing elements which allow independent and simultaneous execution of program tasks. These machines fall into the general class of multiple-instruction/multiple-data (MIMD) architectures, which means that each processing element contains its own instruction stream and data stream.

New developments in computer architectures are expected to require new computer algorithms to fully exploit the new features. The goal of this study is to gain experience in adapting known computational fluid dynamics (CFD) algorithms to parallel, or concurrent, machines and to develop or suggest new concurrent codes based on this experience. The study will focus on three algorithms in CFD which were tested on a two-processor MIMD facility at NASA Ames Research Center.

The first code studied was AIR3D, an implicit, approximate-factored algorithm that has practically achieved benchmark status. This research provided practical experience in taking a well-known scheme, following a particular train of thought, and implementing it on an MIMD machine.

Following the discussion of AIR3D will be a presentation of a state-of-the-art code, the "Chimera Grid Scheme" of Steger, Dougherty, and Benek, using a grid scheme being developed to solve problems having complex geometries (ref. 1). The code makes use of overset grids for an airfoil/flap geometry and incompressible flow. The scheme has certain features which make it ideally suited for parallel-processing, including the elimination of some of the inherent difficulties found in adapting approximate-factorization techniques onto MIMD machines. The implementation of this scheme on the NASA Ames MIMD test facility will be discussed, and a concept called chaotic relaxation will be presented.

The third code studied is an application of overset grids to a problem in compressible flow, which, in this case, is a supersonic blunt-body problem. The code was developed to study the nonlinear effects of compressible flows on overset grids, which were not thoroughly addressed in a previous application of transonic airfoil flow (ref. 2). The discussion will center around the development of a method of handling the grid-interface boundary conditions associated with overset grids. The code was created by the author to run on the NASA Ames test facility, which will be described later.

Since this research bridges two fields of study, the discussion of the codes will involve concepts drawn from both CFD and computer science. An introduction follows, which will provide background information in CFD and computer architecture. The review of CFD will cover only the information that applies to this study. A brief review of computer architectures will be presented, with a major emphasis on MIMD architectures and parallelism. The particular computer used in this study will also be described. One objective of this study is to show the many advantages that result from pooling knowledge from these two fields.

The vast number of MIMD architectures and CFD algorithms make it impossible to investigate all of the possible implementations of the various algorithms on MIMD computers. In this limited survey, many basic properties can be identified which yield significant improvements in concurrent solution procedures for fluid mechanics. Once these properties are identified, users can take advantage of MIMD architectures.

We wish to thank all the people who were involved, in one way or another, in the research and preparation of this material. We would like to thank Carol F. Dougherty for the use of her Chimera code and some of her figures, and Thomas H. Pulliam for the use of his AIR3D and ARC2D codes. We would also like to thank Joseph L. Steger for his help on Centaur, and for introducing us to the embedded grid technique.

2.0 BACKGROUND

The following discussion introduces some basic concepts that we hope will show how CFD and computer science may be merged in an advantageous way. The section reviewing CFD develops the motivation for investigating the particular codes used in this study and discusses current research efforts in this area. The review of computer science describes basic architectures, specifically outlining MIMD architectures; presents some important terminology that will be used throughout this study; and provides a sampling of some MIMD research efforts.

2.1 Computational Fluid Dynamics

2.11 Some current limitations- In the last two decades, the capability in CFD has progressed from the solution of simple problems requiring modest computational power to the solution of full, two-dimensional Navier-Stokes algorithms. However, even with today's computing power, solutions of three-dimensional problems are severely limited. Current aerodynamic codes generally make use of finite-difference techniques, finite-element techniques, or panel methods. Finite-differencing is the simplest to implement for nonlinear applications, but it is difficult to adapt to complex geometries. Finite-element techniques may be more adaptable to complex geometries, but are generally more cumbersome and inefficient to implement. Panel methods are restricted to linear theory. All of the codes used in this study employ

finite-difference methods, so only the difficulties associated with finite-differencing will be discussed. The three codes used in this research use methods which were introduced to overcome these difficulties.

One difficulty in solving large, practical CFD problems arises when one attempts to create a simple computational grid. Finite-differencing generally uses a single rectangular mesh, which, if represented in the computational domain, is monotonic, i.e., grid lines don't cross or coalesce into new grid lines. This requirement is difficult to meet if one is dealing with complex geometries such as airfoils with flaps or wings with engine pylons. In fact, some geometries which may be considered simple, such as a channel with a step, can present major difficulties when one attempts to create a simple rectangular mesh in computational space, as is shown in figure 2.1. Until the grid problem is solved, it is doubtful that the flow field of an entire aircraft can be calculated.

A second restriction to solving large CFD problems is the difficulty of resolving regions of high gradient, such as shocks or shear layers. Resolving high gradients demands that the grid be dense in those regions, which requires some advance knowledge of the solution. For example, to resolve a shock wave on a transonic airfoil, the shock position must be known at grid-generation time if grid points are to be clustered properly. Some solution-adaptive grid algorithms are available which cluster grid points in numbers proportional to the gradients, but require frequent calculation of grid and metric terms. A simple grid technique is needed which is versatile and efficient in terms of covering high-gradient regions with a dense grid, while leaving known, low-gradient regions coarse.

A third restriction on solving large CFD programs is the computer memory requirement. The Euler equations require storage of 13 quantities per grid point, for a scheme in two time levels and three dimensions, which is common for most algorithms. Since a typical grid may require more than a million grid points, most supercomputers cannot store the variables in their core memory. The variables are then put into secondary storage, which may result in large time penalties. The storage requirement, along with the basic computation time of a large problem, is the primary reason why three-dimensional solutions are not routinely practical. Unfortunately, the obvious solution of adding more computer core memory is too expensive to be considered.

Other major problems also exist when one applies finite-difference techniques. Stability and accuracy are a major concern. Improper modeling of the governing equations, or use of too large an iteration step in a solution process, can render a code useless. Other areas, such as turbulence modeling and real gas effects, are of importance, but they lead to research areas which were not considered in this study.

2.1.2 Current CFD research topics- The difficulties mentioned above are the focus of many current studies. It should be reemphasized that the problems and difficulties that were discussed, and the research topics that will be reviewed, are all related to finite-difference techniques. Other techniques avoid some of these problems but have their own pitfalls.

Several approaches have been suggested to solve the problem of generating a grid for a complex geometry. The first is "grid patching" which allows the flow field to be broken into several regions, each of which can be solved by a rectangular, monotonic, computational grid. This technique is demonstrated in figure 2.2. Physical space is divided into regions which are each fitted with their own grid. In two dimensions, grid boundaries are curves, and the grids do not overlap. The grid boundaries, unfortunately, introduce discontinuities in the grid in the general application. Examples of this type of scheme can be found in Rai (ref. 3), and Hessenius and Pulliam (ref. 4).

Another approach is to use overset grids. In this technique, each minor body element is contained within its own simple, minor grid, which is then overlayed onto a major grid. This approach has been used by Steger, Dougherty, and Benek (refs. 1 and 2) and Vadyak and Atta (ref. 5) (fig. 2.3). Both grid patching and overset grids require special handling of grid interfaces. The grid interface of overset grids will be extensively discussed later, since it is highly relevant to the MIMD implementations of this research.

The technique of adaptive gridding is being researched as an approach to solving the grid generation problems about complex flow fields. By adapting the grid to the solution as the solution develops, one can eliminate unnecessary grid points introduced by conservative guesswork. The method consists of generating a new grid every few iterations as the solution progresses, using the gradients of the solution to guide the grid generation process. The result is the generation of efficient grids at each point in the solution procedure. For an example of adaptive gridding, see Dwyer (ref. 6).

Several schemes have been introduced that attempt to circumvent the large memory requirements. One popular scheme is the method of "pencils" (ref. 7). This technique breaks up the large computational grid into pencils, or smaller blocks, which are each solved and then replaced in core memory by the next block. A carefully implemented scheme may reduce the number of pencil loads to two loads per two iterations. However, this appears to be a cumbersome technique that, hopefully, will not be required on future machines that have larger core memories.

Many competing finite-difference schemes have been used with varying degrees of success in terms of stability, accuracy, and speed. This study uses a well-tested, implicit scheme, which was chosen for its superior stability characteristics.

2.2 Computer Design

2.2.1 Computer architectures- A simple introduction to computer architectures will now be presented with examples of each type of machine, a discussion of MIMD characteristics, and a discussion of some past MIMD research projects.

The first computers were von Neumann machines. These computers remain the most common type in use today, from mainframes to microprocessor-driven personal computers. A von Neumann machine has s central processing unit (CPU) and a separate

memory system. Instructions are fetched from the memory system and interpreted by the CPU. Only one instruction is executed at a time, and only one operation is performed each time. Both data and instructions are stored in the same memory system. This is a simple, well-known architecture. However, the von Neumann architecture has inherent physical limitations, such as signal propagation delays between the processing element and memory. If the ultimate limit of propagation speed is achieved (the speed of light), then a signal will take 3 nsec to travel 1 m. For comparison, the CRAY-1s has a clock period of 12.5 nsec, which is representative for the fastest machines. The signal propagation limit is the primary reason why the von Neumann architecture is not suitable for future, large-scale scientific computing.

During a period from 1950 to 1975, computer performance improved by a factor of 10^5. Approximately 10^3 of this improvement has resulted from improvements in electronic components. The remainder of the improvement resulted from architectural improvements which were principally due to increased parallelism (ref. 3). Parallelism has been introduced by overlapping unrelated functions. For example, data required for the next instruction may be read by the processor while the arithmetic units are performing their computations. A tremendous performance improvement was achieved by separating the input/output (I/O) functions from the main processor and providing a separate computer, or I/O channel, to perform data transfers to slow printers and other peripheral devices in parallel. Some machines overlap steps in the instruction-decoding procedure by pipelining several instructions into a decoding unit. Another improvement results from pipelining the computations as is done in vector-pipelined machines.

Vector-pipelined machines are a faster, more powerful class of machines which are considered, by Flynn (ref. 9), to be a subset of the single-instruction/multiple-data (SIMD) classification. One of the best known vector-pipeline machines is the CRAY-1. This machine has processing elements that can be described as assembly-line units, which can execute steps in parallel. An instruction, for example a two-register "add" instruction, will take several clock cycles to complete on a von Neumann machine. A machine such as the CRAY-1 can add two vectors by shifting the two operands into the "pipe" at a rate of one from each vector per clock cycle. Imagine an automobile assembly line which is gearing up to produce a hundred cars. A von Neumann machine would take one car at a time and send it down the assembly line, which is extremely inefficient. The vector-pipeline machine shifts a car onto the line as soon as the previous car has completed the first step and is moving to the next step. Thus, the first car will take an equal amount of time to reach the end of the assembly line, but from then on, a car will be shifted out at each time cycle. The cars, in this example, represent the elements of a vector and the assembly line represents the pipeline.

The vector-pipeline machine has a single processor. (The CRAY-1 has multiple arithmetic logical units (ALU).) Processing elements such as these can be linked into a multiple-processor system, thereby increasing the parallelism and creating more computer horsepower from a given processor technology. Alternatively, many simple processing elements, that can be produced inexpensively as a result of

integrated circuit technology, can be linked to exceed CRAY speeds for large-scale computation. These processors can be distributed physically within the memory system, thus reducing the signal-propagation penalty of memory accesses. Multiple-processor architectures have tremendous potential for increasing parallelism, and will be the primary innovation in supercomputer designs for the next decade. The following types of machines contain multiple processing elements, each capable of performing operations simultaneously.

The first class of multiprocessor machines are the processor array computers, another subset of SIMD architectures. The processor array machine is a parallel machine which performs identical operations simultaneously on an array of processing elements. A single command unit interprets the instruction and commands the processing elements to perform the task in lockstep. Thus, it will take the same time to add two vectors together as it would take to add two scalars together. The ILLIAC IV, housed at NASA Ames Research Center from 1972 to 1981, was a processor array computer.

The next form of multiple-processor architecture is the MIMD stream architecture. These machines are being developed commercially for both business mainframes and scientific supercomputers. An MIMD machine is a multiple processor machine that runs tasks concurrently. The number of processors in an MIMD machine can range from two, as is the case with the test facility of this study, to thousands. The processors are neither running lockstep, nor are they generally executing the same operation. It is entirely possible to have each processor executing a different users' program. In the implementation of this research, the processors will be told to cooperate and perform tasks common to a single problem.

MIMD and SIMD architectures each have advantages and disadvantages. For array calculations, where the parallel computation can be accomplished by performing identical computations on each element of the array, the SIMD machine is the most efficient. This is due to the lockstep nature of the machine. An MIMD machine, on the other hand, would require a forced synchronization since each processor runs independently, not in lockstep. This results in small synchronization penalties. The study of the AIR3D code (section 3) gives some indication of the overhead required for synchronizing a two-processor system running a conventional CFD code, which is also a suitable system for SIMD computation. For nonarray calculations, or array calculations with conditional branches, SIMD architectures are inefficient. MIMD architectures allow for each processor to follow its own, unique branch or a separate task. MIMD machines therefore are more flexible than SIMD machines.

MIMD machines also offer a clear advantage over SIMD machines in their modes of operation. SIMD machines allow only one user at a time, since each processing element is controlled by a single control unit. MIMD machines offer three operational modes: single user, time-shared with one processor per user, and time-shared with a multiprocessor, multitask, queue procedure. These modes will be described in the following section.

A measure of the performance of multiprocessor systems is necessary to evaluate the computation speed. For SIMD machines, two performance parameters have been

established for comparison. A maximum-performance parameter gives the maximum computational rate, usually given in megaflops (Mflops--million floating point operations per second), and assumes an infinite vector length and full use of the computer's parallelism. A second parameter is the half-performance length, which is the vector length that is required to achieve half of the maximum performance (ref. 7). These measurements are based on vector, or array, calculations, so they cannot completely describe the performance of an MIMD machine, which does not necessarily require vectors to achieve parallelism. MIMD machines are generally parameterized by their maximum computation rate of selected benchmark programs. This introduces a dependence on the particular benchmark codes which may not use the unique architectural features of the machine.

The performance data obtained for the NASA Ames MIMD test facility does not separate the parallelism due to the algorithm from the parallelism due to the machine. The measurement used is a speedup factor which is the ratio of the execution times for the code running on a single processor to the same code running on both processors in the test system. A simple estimate of the machine performance for the two-processor test facility would be a speedup factor of two. However, machine hardware is not ideal, so the hardware performance cannot reach a speedup of two. An even more important consideration is the code being tested. Most codes have parts that are inherently serial, and cannot be speeded up through parallelism. In this study, the goal of the computational fluid dynamicist is to produce algorithms and codes which can efficiently use concurrency.

2.2.2 MIMD architectures- MIMD machines can vary widely in their architecture. Some machines share their entire memory among the processors, whereas some share only small blocks, leaving a larger local memory to be used by one processor only. The use of different memory systems can have major effects on programming. The memory systems for two MIMD research facilities--the NASA Ames MIMD test facility and the CRAY X-MP--will be compared. Note that both of these machines have only two processors in their current configuration, so they represent only a small class of MIMD architectures. Following this discussion, some operating system differences will be outlined. Some concurrent programming terminology, which will be used throughout the paper, will also be presented.

The CRAY X-MP shares all of its memory and combines two CRAY-1s into a single system. It has two processors that connect to a single memory system so that all data in the memory are shared between the two processors. A machine such as the CRAY X-MP, therefore, will be able to handle large amounts of shared data. The NASA Ames MIMD test facility consists of two VAX 11/780s connected via a 1/4 Mbyte MA780 dual-ported memory. Thus, this system has a limited amount of shared memory and is not able to run programs that require large amounts of shared data. Each VAX 11/780 has its own large local memory in which it can store large amounts of unshared data. The architectures of these two facilities are sketched in figure 2.4.

There are some practical problems that occur when using shared memory. One of the most significant is the memory contention problem. A single bank of memory can be accessed only once during each memory cycle. Therefore, other processors must wait until the memory is freed by the processor currently accessing it. The small

number of processors in the two research systems at NASA Ames will not cause noticeable problems. However, the overall effects of memory contention on hardware performance can be large on large-scale, multiprocessor systems. Another practical problem, that of memory protection, is usually associated with the operating system. The operating system is responsible for protecting blocks of data from processors that are not supposed to access them. On a system such as the VAX MIMD facility at NASA Ames, the solution is to store data to be protected from other processors into the proper local memory so that the other processors cannot access it. The different methods of protecting memory result in major differences in how the computer can be used.

A program contains three types of data groups. The first type is local data, which are data that are local to a single block, or subroutine, of the program. In FORTRAN, local data represents the data local to a subroutine. The second type of data group is the task-global. This is equivalent to a common block in FORTRAN, which allows all subroutines that use that common block to access it. The last data type is the shared-global. This data group resembles a FORTRAN common block except that different tasks, or programs, can share the data. Currently, the CRAY X-MP does not have a task-global data group. Therefore, any data that are placed in a common block are automatically shared among tasks, and cannot be protected from the other tasks. If two tasks are running concurrently and each requires a task-global data block, the block must be duplicated so that each task will not "grab" the other's data. This in itself is not too restrictive, unless the code is transferred to a machine with extensive amounts of task-global memory and little shared-global memory such as the VAX MIMD facility. Extensive modifications are then required.

MIMD machines offer three modes of operation (environments). The first, used in the study of AIR3D, was a single-user mode. This mode allows a single user access to all processors. The second mode of operation (i.e., multiuser) allocates each processor to a separate user. In a manner analogous to timesharing on a single processor, where each user is allocated a time block, the users are each allocated a processor. Computers using this mode have been built, but are not considered to be true MIMD machines, since the processors do not share data.

The third mode of operation is a multiprocessor, multitasking environment where tasks are queued and submitted to the first available processor. This is the most efficient mode of operation, since no processor will be idle regardless of the size or number of the tasks queued. This mode is becoming the most widely used on MIMD machines. A practical approach to solving large-scale CFD problems in this environment is to segment the problem into several smaller tasks. Each segment can be submitted to the circulating processor queue to be executed by the next available processor. The overset grid programs of sections 4 and 5 have been routinely executed in a variation of this mode of operation (the NASA Ames MIMD test facility has two job queues, one for each processor). The overset grid programs are ideally suited to this mode of operation and the concept of chaotic relaxation becomes important in this environment.

Terminology which is often used to describe the concurrent flow of a program will now be introduced. Flow of a concurrent code is managed by operations that

occur at the junctions and branches of a typical flow chart (fig. 2.5). When a single task is divided into multiple concurrent tasks this is called a "Fork." When two or more tasks combine into a single task this is called a "Join." Another operation, "Sync," is used when multiple tasks must synchronize at some point before they continue. A Join or Sync point requires some method to record which tasks have reached the Join or Sync point. This is accomplished by having the tasks record an "Event." When all of the required Events have been recorded for a Join or Sync, the program continues. Figure 2.6 summarizes the operations. Although the operating system of an MIMD computer will know how to handle these operations, they must be explicitly placed in the program and, unfortunately, no universal high-level FORTRAN commands exist at this time. Therefore, programming can be difficult, and codes will not be transportable from one machine to another until high-level FORTRAN commands are incorporated directly into the programming language. Other languages such as concurrent PASCAL and ADA have concurrent constructs within the language. However, these languages are not popular with engineers, so they will not be considered. Since no FORTRAN commands exist, the above names (i.e., Fork, Join, Sync, Event) will be used throughout.

2.2.3 Sample MIMD research- It is the responsibility of the programmer to extract parallelism from an algorithm. New schemes will be required which exploit the characteristics of multiprocessor machines, just as new methods were developed for vector processors and other architectures. For example, the half-performance length for the four-pipe CYBER-205, which is about 400, suggests the use of long vectors compared to the CRAY-1, which has a half-performance length of about 10 to 20. Each new multiple-processor design will have some unique features which will favor a particular choice of algorithm. The choice will be left to the engineer until sufficient artificial-intelligence capabilities are available.

Parallel algorithms have been a popular research area in computer science for some time. Many of these are matrix-inversion algorithms. In particular, numerous tridiagonal schemes have been developed. Lambiotte and Voigt (ref. 10) have developed parallel algorithms for the CDC STAR-100 computer; Stone (ref. 11) has designed parallel tridiagonal solvers for machines such as the STAR-100 and the ILLIAC IV; and Barlow and Evans (ref. 12) have gained experience with explicit methods using the MIMD computer at Loughborough University. These algorithms have been very successful on MIMD testbeds and all seem promising. Despite the fact that tridiagonal solvers are commonly used in CFD, the complicated MIMD tridiagonal scheme was not the preferred choice for this research. In CFD, the matrix routines are typically executed thousands of times per iteration, making it easier to solve several matrix routines concurrently.

Another type of algorithm that has been studied are the chaotic algorithms. Chazan and Miranker (ref. 13) developed the framework for an algorithm called chaotic relaxation and Baudet (ref. 14) has gained experience on the Carnegie-Mellon C. mmp machine. These chaotic algorithms are a unique product of multiple-processor computers. Since the processors do not operate in lockstep, the chaotic algorithms have been developed to remove the synchronizations that are required in most serial programs. Synchronizations are implicit in the coding procedure familiar to most

users. In a chaotic algorithm, one must forget the notion of sweeping across the data in a "do loop" fashion. Instead, regions are solved as they are allocated CPU time, not in any specific order or time. This will have important implications for concurrent multi-user machines operating in the queuing mode described above. The subject of chaotic relaxation is rather difficult to follow at first, so it will not be discussed at this point; but will be reintroduced in greater detail as needed.

2.3 Focus and Approach

For this research, three computer codes have been studied. Two are adaptations of working schemes, and one was developed specifically for this research. Each code represents the cutting edge of CFD research for various applications. The code AIR3D is a three-dimensional, implicit, approximate-factored algorithm that has been widely used. Its large size and the fact that many CFD researchers are familiar with it make it an ideal candidate to test on a MIMD machine. This test will determine whether approximate-factored algorithms can be implemented on MIMD facilities to some advantage. It will also help develop some insight into more efficient use of MIMD machines in CFD.

The next code is essentially the Dougherty, Steger, and Benek chimera grid scheme applied to an incompressible, two-dimensional airfoil/flap configuration. This code was chosen to test the concept of using multiple processors to solve multiple-grid problems. The practicality of using chaotic relaxation on overset grid schemes was also studied.

The final code, Centaur, was developed to investigate difficulties of using overset grids on a compressible problem. In these situations, the Euler equations are used for the solution of a supersonic blunt body. The overset grid is used to resolve the bow shock near the front of the cylindrical-wedge blunt body studied. The central purpose of this code was to serve as a test vehicle for studying boundary schemes for the overset grid boundaries, in particular when the shock wave crosses the boundary.

All three codes were developed and tested on the NASA Ames MIMD test facility. This facility consists of two VAX 11/780 machines with an MA780 dual-ported, shared memory. The MA780 contains 256 Kbyte (or 1/4 Mbyte) of memory. Each VAX 11/780 is a self-contained unit and each serves a role in the NASA Ames VAX network. Since the facility was not exclusively an MIMD facility, most work was done on a single machine run in a time-shared mode. Unfortunately, this means that not only did the two tasks have to compete for CPU time, they also had to compete, at times, with more than 40 interactive users on the system. Also, because of the limited computation speed of the VAX, the codes were generally kept small. When important timing data were required, the facility was shut down to all other users so that an accurate measurement could be made.

NASA Ames acquired a CRAY X-MP in September 1983. However, the version of the operating system available for this research did not contain the elements required for concurrent processing of a single code. AIR3D is currently being adapted to the

CRAY X-MP, by NASA Ames personnel, and tested at a Cray Research facility in Mendota, MN.

Before each code and its results are detailed, the central purpose of this study should be restated. The purpose of this study is to discover whether parallelism found in CFD can take advantage of the benefits of a sample MIMD machine. The first test will be the implementation of a popular CFD algorithm to determine whether it is well suited for MIMD architecture. An overset grid scheme will then be studied to investigate its concurrent properties in an incompressible and a compressible application.

3.0 APPROXIMATE FACTORIZATION AND MIMD ARCHITECTURE

The first section in this study covers the implementation of an approximate factored algorithm onto a representative MIMD computer. The purpose of this study is to determine whether approximate factorization has properties suitable for parallel processing on MIMD machines. Possible applications include using the newly acquired CRAY X-MP at NASA Ames to solve some popular CFD algorithms. Also, gaining a complete understanding of the mapping of the algorithm to the architecture will help to attain a long-term goal, which is to determine how the choice of MIMD architecture influences the efficiency of solving approximate factored algorithms. The particular code that will be used for this study is the Pulliam and Steger AIR3D code (ref. 15), commonly known as ARC3D (Ames Research Center 3D).

The code AIR3D is a three-dimensional, implicit, approximate-factored algorithm which solves either the Euler equations or the Navier-Stokes equations with a thin-layer approximation. Options include either a laminar or turbulent boundary layer, for the viscous case, and a default grid about an axisymmetric, hemispherical-nosed projectile.

A detailed discussion of AIR3D will first be presented covering the equation set solved and the particular form of the approximate factorization algorithm used. General properties of approximate factorization will be included in this discussion. The required steps to implement the code on a generic MIMD machine will then be presented, along with a discussion of how the memory system will influence the procedure used to adapt the code to a particular machine. This will be followed by the specific application on the NASA Ames research machine, which attempts to simulate a possible CRAY X-MP implementation. (At the time of this writing, the CRAY X-MP was not available for testing concurrent processing.) Results for this study will be in the form of the speedup performance parameter described earlier, in which the execution time for the concurrent implementation is compared to the execution time of a serial implementation. These execution times will give an indication of processor synchronization overhead and data communication overhead. However, the dominant factor in the speedup measurement will be the concurrency within the algorithm, so the emphasis will be placed on the algorithm in this research.

The mode of operation used for the study is the single-user, dedicated system mode. A multiuser, time-sharing system would not be practical for this study because of the required synchronizations, which will be shown later. An assumption, that all data can be made available to both processors in a sharable memory, is made which allows the system to mimic the CRAY X-MP memory architecture. A conceptual machine design which has properties specially suited for a dedicated, approximate-factored algorithm will be presented.

The study of AIR3D is part of a larger research effort in the Computational Research and Technology Branch at NASA Ames Research Center. As part of this effort, two additional codes have been adapted to the NASA Ames MIMD test facility by NASA researchers. The two parallel studies, which also use well-known CFD algorithms, are "TWING," a three-dimensional potential algorithm, and Rogallo's "LES" (Large Eddy Simulation) code using spectral methods (refs. 16 and 17).

3.1 Equation Set

The code AIR3D is an implicit finite-difference program for time-accurate, three-dimensional, flow calculations. It is capable of handling viscous effects and incorporates an algebraic turbulence model as a selected option. The code is also capable of handling arbitrary geometries through the use of a general coordinate transformation. A more complete description of the code can be found in a paper by Pulliam and Steger (ref. 15), which will be summarized here.

The three-dimensional, nonsteady, Navier-Stokes equations can be transformed and written for an arbitrary, curvilinear space, while retaining the strong conservation-law form, without increased complexity of the governing set. The following form shows the resulting equations when transformed from x, y, z, and t space to ξ, η, ζ, and τ space.

$$\frac{\partial}{\partial \tau} q + \frac{\partial}{\partial \xi} (E + E_v) + \frac{\partial}{\partial \eta} (F + F_v) + \frac{\partial}{\partial \zeta} (G + G_v) = 0 \qquad (3.1)$$

where

$$q = J^{-1}\begin{bmatrix}\rho\\ \rho u\\ \rho v\\ \rho w\\ e\end{bmatrix}, \qquad E = J^{-1}\begin{bmatrix}\rho U\\ \rho Uu + \xi_x p\\ \rho Uv + \xi_y p\\ \rho Uw + \xi_z p\\ U(e + p) - \xi_t p\end{bmatrix}$$

$$F = J^{-1}\begin{bmatrix}\rho V\\ \rho Vu + \eta_x p\\ \rho Vv + \eta_y p\\ \rho Vw + \eta_z p\\ V(e + p) - \eta t p\end{bmatrix}, \qquad G = J^{-1}\begin{bmatrix}\rho W\\ \rho Wu + \zeta_x p\\ \rho Wv + \zeta_y p\\ \rho Ww + \zeta_z p\\ W(e + p) - \zeta t p\end{bmatrix} \tag{3.2}$$

and

$$\begin{aligned} U &= \xi_t + \xi_x u + \xi_y v + \xi_z w\\ V &= \eta_t + \eta_x u + \eta_y v + \eta_z w\\ W &= \zeta_t + \zeta_x u + \zeta_y v + \zeta_z w \end{aligned} \tag{3.3}$$

$$J^{-1} = x_\xi y_\eta z_\zeta + x_\zeta y_\xi z_\eta + x_\eta y_\zeta z_\xi - x_\xi y_\zeta z_\eta - x_\eta y_\xi z_\zeta - x_\zeta y_\eta z_\xi \tag{3.4}$$

The quantities U, V, and W are the contravariant velocities written without metric normalization. This general transformation includes the possibility of a moving grid. The viscous terms, E_v, F_v, and G_v, will not be presented here but can be found in many papers on the subject (refs. 15,18,19). In this formulation, the Cartesian velocity components u, v, w are nondimensionalized with respect to the free-stream speed of sound, a_∞, and the density, ρ, is normalized with respect to ρ_∞. The total energy per unit volume, e, where $e = \rho[\varepsilon + (1/2)V^2]$, ε is the internal energy, and V is the velocity magnitude, is normalized with respect to $\rho_\infty a_\infty$. Pressure is given in terms of these variables by

$$p = (\gamma - 1)\left[e - \frac{1}{2}\rho(u^2 + v^2 + w^2)\right] \tag{3.5}$$

The metric terms themselves are defined in detail in reference 15.

This program makes use of a thin layer approximation throughout, resulting in fewer grid points and fewer computations. The thin-layer approximation uses coordinates similar to boundary-layer coordinates and ignores viscous terms associated with small velocity gradients. Therefore, if the ξ and η coordinates are chosen to lie parallel to the body surface, only the ζ viscous terms will be included. This is similar to a boundary-layer model in which streamwise viscous terms are ignored. Thus, this approximation requires grid refinement in only the ζ, or perpendicular, direction. The new set of equations simplifies to

$$\frac{\partial}{\partial \tau} q + \frac{\partial}{\partial \xi} E + \frac{\partial}{\partial \eta} F + \frac{\partial}{\partial \zeta} G = Re^{-1} \frac{\partial}{\partial \zeta} S \tag{3.6}$$

where

$$S = J^{-1} \begin{bmatrix} 0 \\ \mu(\xi_x^2 + \xi_y^2 + \xi_x^2)u_\zeta + (\mu/3)(\zeta_x u_\zeta + \zeta_y v_\zeta + \zeta_z w_\zeta)\zeta_x \\ \mu(\xi_x^2 + \xi_y^2 + \xi_z^2)v_\zeta + (\mu/3)(\zeta_x u_\zeta + \zeta_y v_\zeta + \zeta_z w_\zeta)\zeta_y \\ \mu(\xi_x^2 + \xi_y^2 + \xi_z^2)w_z + (\mu/3)(\zeta_x u_\zeta + \zeta_y v_\zeta + \zeta_z w_\zeta)\zeta_z \\ (\zeta_x^2 + \zeta_y^2 + \zeta_z^2)(0.5\mu(u^2 + v^2 + w^2)_\zeta + \kappa Pr^{-1}(\gamma - 1)(a^2)_\zeta) \\ \qquad + (\mu/3)(\zeta_x u + \zeta_y v + \zeta_z w)(\zeta_x u_\zeta + \zeta_y v_\zeta + \zeta_z w_\zeta) \end{bmatrix} \tag{3.7}$$

and Re is the Reynolds number, based on the nose radius, Pr is the Prandtl number, and κ and μ are the coefficient of thermal conductivity and the dynamic viscosity, respectively. An algebraic turbulence model is also incorporated in AIR3D which makes use of the method of Baldwin and Lomax (ref. 20).

A supersonic projectile geometry is used as a test problem. The projectile geometry chosen has a hemispherical nose and a cylindrical afterbody, and is in a Mach 1.2 flow at a 19° angle of attack. This configuration allows for the use of simple boundary conditions, such as supersonic inflow at upstream boundaries and supersonic outflow conditions for supersonic outflow boundaries. The physical space can be cut in half by taking advantage of the symmetry of the projectile, which also results in simple boundary conditions along the plane of symmetry. Solid-body conditions are used on the body, i.e., no normal velocity for inviscid flow and no-slip for viscous flow.

3.2 Algorithm Used

The approximate-factorization algorithm to be used has its origins in the Alternating Direction, Implicit (ADI) algorithm. The ADI algorithm was introduced for various applications by Peaceman and Rachford (ref. 21), Douglas (ref. 22), and Douglas and Rachford (ref. 23) for scalar elliptic equations. The method was extended to hyperbolic equations by Douglas and Gunn (ref. 24) and Beam and Warming (ref. 19). For a hyperbolic set of equations such as the Euler equations, which we represent by the general form

$$\frac{\partial}{\partial t} q + \frac{\partial}{\partial x} E + \frac{\partial}{\partial y} F + \frac{\partial}{\partial z} G = 0 \tag{3.8}$$

where q is a vector, E = E(q), F = F(q), and G = G(q), the corresponding finite-difference equations can be expressed in operator notation and written as follows:

$$\mathscr{L}_{xyz}\,\Delta q^n = \mathscr{F}_{xyz} q^n \tag{3.9}$$

In this form, the left-hand side is the implicit part and the right-hand side is the explicit part of the algorithm; and the equation is expressed in delta form where $\Delta q^n = q^{n+1} - q^n$. The operators in equation (3.9) are general operators which result from the finite-differencing and the local time-linearization of the terms containing the E, F, and G differentials, which will be discussed later. The approximate factorization is introduced as a less computationally costly means of inverting the operator on the left-hand side. The operator is first factored into three separate operators that are spatially independent, as follows:

$$\mathscr{L}_x\mathscr{L}_y\mathscr{L}_z\,\Delta q^n = \mathscr{L}_{xyz}\,\Delta q^n + o(\Delta t^2) \tag{3.10}$$

and the approximation is introduced by ignoring the second-order terms in equation (3.10). This allows the introduction of a three-step solution process in which each step inverts an independent spatial operator. Intermediate variables are encountered in this way, but they do not add to the storage requirements since they may overwrite the previous level. The three-step solution is given by

$$\begin{aligned} q^* &= \mathscr{L}_y\mathscr{L}_z\,\Delta q^n = \mathscr{L}_x^{-1}\mathscr{F}_{xyz} q^n \\ q^{**} &= \mathscr{L}_y^{-1} q^* \\ q^{n+1} &= q^n + \mathscr{L}_z^{-1} q^{**} \end{aligned} \tag{3.11}$$

The particular finite-difference scheme used in AIR3D is the implicit, approximate-factorization algorithm of Beam and Warming (ref. 19). The scheme was chosen to be implicit to avoid the restrictive stability bounds of explicit methods when applied to small grid spacings. The delta form of the algorithm, in which incremental changes in quantities are calculated, is used to allow approximate factorization without explicitly subtracting factorization errors and, in addition, is a convenient choice for steady-state solutions.

Central differencing is used for all three directions. The finite-difference equations are spatially split so that three separate one-dimensional problems are solved at each time step. The central differencing yields block tridiagonal matrices which are inverted in each spatial coordinate. This decoupling of the operators for each spatial coordinate, which results from the approximate factorization, provides the principal motivation for considering parallel processing as a means to carry out the calculations. The method used to invert the operators will be detailed at the end of this section.

The approximate factorization of the finite-difference algorithm results in the following set of finite-difference equations:

$$(I + h\delta_\xi \tilde{A}^n - \varepsilon_i J^{-1}\nabla_\xi \Delta_\xi J)(I + h\delta_\eta \tilde{B}^n - \varepsilon_i J^{-1}\nabla_\eta \Delta_\eta J)$$
$$\times (I + h\delta_\zeta \tilde{C}^n - hRe^{-1}\delta_\zeta \tilde{M}^n - \varepsilon_i J^{-1}\nabla_\zeta \Delta_\zeta J)\Delta q^n$$
$$= -\Delta t(\delta_\xi E^n + \delta_\eta F^n + \delta_\zeta G^n - Re^{-1}\delta_\zeta S^n) - \varepsilon_e J^{-1}[(\nabla_\xi \Delta_\xi)^2 + (\nabla_\eta \Delta_\eta)^2 + (\nabla_\zeta \Delta_\zeta)^2]Jq^n \tag{3.12}$$

in which $\Delta q^n = q^{n+1} - q^n$ and $h = \Delta t$ for first-order Euler time differencing, or $h = \Delta t/2$ for second-order trapezoidal time differencing. The finite-difference operators δ, ∇, and Δ are defined as

$$\delta_x f = \frac{1}{2\Delta x}(f_{j+1} - f_{j-1})$$
$$\nabla_x f = \frac{1}{\Delta x}(f_j - f_{j-1}) \tag{3.13}$$
$$\Delta_x f = \frac{1}{\Delta x}(f_{j+1} - f_j)$$

Implicit and explicit smoothing terms have been added to help damp out the high-frequency oscillations. In the implicit operator, the numerical damping terms use second-order differencing to maintain the block tridiagonal nature of the implicit part of the code, while in the explicit right-hand side they use fourth-order differencing. The smoothing coefficients are ε_i and ε_e for the implicit and explicit smoothing, respectively. The matrices $\tilde{A}^n$, $\tilde{B}^n$, and $\tilde{C}^n$ are local linearizations of E^{n+1}, F^{n+1}, and G^{n+1}, respectively, and are obtained using a Taylor series expansion on Δq. The coefficient matrix $\tilde{M}^n$ is obtained by a Taylor series expansion of the viscous vector S^{n+1}. These matrices will not be presented here, but can be found in the paper by Pulliam and Steger (ref. 15).

In terms of operator notation, each operator becomes a block tridiagonal matrix with the following structure.

$$\mathscr{L}_x = (I + h\delta_\xi \tilde{A}^n - \varepsilon_i J^{-1}\nabla_\xi \Delta_\xi J)$$
$$\mathscr{L}_y = (I + h\delta_\eta \tilde{B}^n - \varepsilon_i J^{-1}\nabla_\eta \Delta_\eta J) \tag{3.14}$$
$$\mathscr{L}_z = (I + h\delta_\zeta \tilde{C}^n - hRe^{-1}\delta_\zeta \tilde{M}^n - \varepsilon_i J^{-1}\nabla_\zeta \Delta_\zeta J)$$

The block tridiagonal operators $\mathscr{L}_x$, $\mathscr{L}_y$, and $\mathscr{L}_z$ are spatially decoupled, and so can be inverted independently. This spatial decoupling is the property that allows

concurrent processing. In the past, this property has been used to support vectorization, on vector processors, by solving each operator as an array. Here, this property has been exploited to adapt the code to MIMD architectures. The spatial decoupling helps in the following way. Since each operator contains derivatives in only one spatial direction, all lines of data in that coordinate can be solved independently; for example, each line of j, where j is the x index, can be solved independently on every point in the k,ℓ plane, where k and ℓ are the indices of y and z, respectively. Thus, an MIMD machine could, in principle, use as many processors as there are points in each plane, assuming there is no other restriction. The explicit operator, $\mathscr{E}_{xyz}$, can be handled in any convenient manner, since it is completely explicit, and all required data are available at each time step. Without the spatial decoupling, the implicit algorithm would have no inherent parallelism, so adapting the code to MIMD machines would be a difficult programming task. The recognition of this inherent parallelism was the primary factor in pursuing the study of approximate-factorization algorithms.

It should be noted that this decoupling is a feature of the approximate factorization, and is not associated with the central differencing used in AIR3D. Thus, any approximate-factorization algorithm exhibiting this kind of spatial decoupling should allow the use of the same sort of parallelism.

3.3 Memory Allocations and MIMD Implementation

The presence of parallelism in the approximate factorization algorithm suggests a simple approach for adapting AIR3D onto a MIMD facility. However, the memory structure of the MIMD facility will make an enormous difference in the actual implementation. As mentioned in the introduction, MIMD machines have different types of memory systems. At one extreme, all of the memory in the multiprocessor machine is shared among the processors. At the other extreme, the machine has a very small shareable memory, whereas each processor may have its own large local memory. As the adaptation of AIR3D is discussed, particular attention will be made to these architectural differences and how they affect the particular implementation.

The simplest implementation of an approximate-factored algorithm is on a facility with a large common memory. All of the data can be stored in the shared memory and, therefore, can be accessed by all of the processors. The CRAY X-MP is a machine with a moderately large 2-megaword (MWord) shared memory so the large, common-memory implementation presented here is the approach that is considered in this study. For this discussion it will be assumed that the amount of shared memory is infinite, and memory-access conflicts among processors do not occur. This is a reasonable assumption for the CRAY X-MP because it has only two processors, and memory-access conflicts are expected to be minimal. In fact, one of the objectives of this study is to obtain a measure of this conflict for a practical problem in CFD. This ideal approach will now be presented.

3.3.1 Ideal implementation- An ideal MIMD implementation of the spatially decoupled procedure begins as follows. The first step is to compute the right-hand

side of equation (3.12). Because it is explicit and all data at the current time step are available (in the shareable memory), the data can be divided into several groups. A convenient split is to divide the data evenly along a particular direction (x) by the number of processors available. If j is the index representing the x direction and J_{max} is the total number of x grid points, then for a two-processor system, one processor can be assigned the points 1 to $J_{max}/2$ and the other $J_{max}/2 + 1$ to J_{max}. Note that no overlapping is necessary, since all data are present and accessible by both processors in shared memory. The split is represented ideally by a Fork as shown in figure 3.1. All processors must be synchronized at the completion of this step before continuing to the implicit integration. This synchronization, a Join in figure 3.1 (following the explicit right-hand side calculation) is the first of four such synchronizations.

After all tasks have joined, verifying completion of their respective segments of the right-hand side, the inversion of $\mathscr{L}_x$ may begin. A single line of j can be inverted at any point in the k,ℓ plane, independent of all other lines of j. Thus, the workload can be distributed among the several processors and directed to any desired division of the k,ℓ plane. Again, this split is represented by a Fork in figure 3.1. At the completion of the x sweep, a second synchronization, or Join, must occur before proceeding to the y sweep.

For the y sweep, the database can be split anywhere in the j,ℓ plane to separate the decoupled k lines for concurrent processing. Upon completion of this sweep, the processors must be joined a third time before continuing the z sweep. The z sweep can be split anywhere in the j,k plane, and computation proceeds as before. A fourth and final Join is required at the end of the z sweep to complete a single iteration loop. Note that if the y and z sweeps were processed before the x sweep, then two Joins can be eliminated by allowing each processor to solve the $\mathscr{F}_{xyz}$, $\mathscr{L}_y$, and $\mathscr{L}_z$ before the data must be repartitioned. This iteration loop may have to be repeated 200-600 times before a converged solution is obtained with typical CFD applications.

An example of a simple two-dimensional problem requiring a two-step solution procedure with three synchronizations is shown in figure 3.2. The figure outlines the process for the implicit operator on a four-processor system.

3.3.2 Architectural restrictions- Practical architectural restrictions will now be introduced. First, the shareable memory will never be infinite. Therefore, the problem may be limited to the available common memory. If we assume that the available shared memory is sufficient, then we arrive at the next restriction. This restriction is memory bus bandwidth. Take the example previously discussed (fig. 3.2). Let us suppose the MIMD facility has enough processors to assign one processor per point in each plane so that an operator could be inverted by a single sweep of each processor. Then a situation would exist where more than one processor would try to access the shared memory at any given time. However, only one processor can access a memory block at a time, so there would be memory conflicts, and the other processors would have to wait. As the number of processors grow, the number of conflicts grow and the waiting time increases. An industry rule of thumb is that the memory bus becomes saturated with as few as four processors. A simple solution

to this memory-bus bandwidth problem is to introduce separate memory banks with private access. However, the networking of such a system to allow a certain amount of data transfer is a difficult problem, and represents a major field of research in computer science. The CRAY X-MP, with only two processors, does not suffer noticeably from memory-bus saturation.

At the other end of the hardware spectrum are machines that have limited shared memory. The NASA Ames MIMD test facility falls into this category. However, the goal of this particular study was to develop procedures for implementing the algorithm on the CRAY X-MP. Therefore, instead of using the procedure described in the discussion that follows, the VAX test facility was used to simulate the CRAY X-MP's architecture. This was accomplished by modifying the problem, which will be detailed later.

The procedure that would normally be used in these machines uses the concept of "pencils" in breaking up the data into blocks that fit into the common memory. A pencil is a specific block of data which includes all data along a grid line. This procedure is commonly used for solving large-scale problems on small-memory systems. On a single-processor system with limited memory, the method is to read a pencil from an outside source (such as a disk drive) into memory, and perform the required operation (such as the inversion of the $\mathscr{L}_x$ operator). When the operation is complete, the result is exchanged for a new pencil, and the task is repeated until the entire operator for one spatial plane is inverted. This process must be repeated for each spatial plane of data. For example, the data blocks shown in figure 3.2 can be viewed as pencils and the entire process is analogous to the MIMD procedure previously described. Each read-and-write of a pencil requires that the data be restructured, since the actual procedure is to take a three-dimensional array and divide it into several new, three-dimensional arrays. Reading and writing pencils into memory takes considerable time, so an effort is generally made to fit the entire problem into the available memory. To minimize the time, only two such reads-and-writes of pencils in two iterations are done. This technique, which is detailed in Lomax and Pulliam (ref. 7), basically requires that the order of operation for the operators $\mathscr{L}_x$, $\mathscr{L}_y$, and $\mathscr{L}_z$ be rotated with each iteration, allowing pencils to be oriented properly for a longer period.

An MIMD implementation of this procedure occurs as follows. First, it must be assumed that one processor has access to all of the data. This processor is responsible for managing the pencils for all of the other processors. Its function will be to load the pencils into a memory that can be accessed by the other processors. The implementation, on a machine with a single, shared-memory unit for all processors, would be to take the largest pencil which can fit into the memory block and process it. Each processor would then take a portion of the pencil and operate on it. On the other hand, if each processor has a semiprivate memory bank shared with a processor which coordinates pencil loads, then each memory bank could be loaded with a different pencil, so that each processor operates on separate data. This hardware implementation is a specific network implementation of the memory bank scheme described above. Again, as described in reference 7, only two loading cycles of pencils are required during each two iterations.

On observing the memory-bandwidth problem of the ideal MIMD implementation and the technique of pencils for the limited memory application, one can deduce a simple hardware construction for the algorithm (fig. 3.2). This construction is based on the "dance hall" model, which draws an analogy between processors and dancers. Each processor is given a block of memory and dances around with it until it eventually passes its partner to another processor in exchange for a new partner. For the two-dimensional problem of figure 3.2, the computational domain is divided by the four processors and the two sweeps into 16 blocks, as in figure 3.3. However, if one determines which processor accesses which block, one sees immediately that the diagonal blocks are accessed only by a single processor. In fact, the blocks in the computational domain can be associated with matrix elements A_{jk}, in which j and k identify the processors which access a block in the two sweeps. Figure 3.3 shows a four-processor implementation using this notation. Thus, for a dedicated approximate factorization multiprocessor system, the memory configuration may be chosen so that a block is accessed by a single processor if $j = k$ or by two processors if $j \neq k$. Since each block is accessed by one processor at a time, the implementation could make use of hardware switches which may be reset by the software on each sweep.

The primary motivation for this approach is to eliminate the problem of memory-bus bandwidth. The memory implementation discussed here would circumvent this problem, provided the required switching can be suitably implemented in hardware. Also, in principle, there would be no restriction on how many elements, A_{jk}, are used. Although software may be required to align the database for each A_{jk} access, which would introduce some additional complexity, this memory implementation would be an interesting possibility for CFD usage.

The problems associated with different MIMD architectures manifest themselves in either memory bus saturation or data-reformatting overhead. The primary objective of the individual codes is number-crunching, so any penalties resulting from these problems are undesirable. Since the MIMD research facility at NASA Ames is only a two-processor system, the penalties were minimal.

3.4 Implementation on the VAX Test Facility

The implementation of AIR3D on the MIMD test facility at NASA Ames will now be described. The specific memory architecture assumed for this study mimics that of the CRAY X-MP. In other words, it is assumed that the processors have access to all data through a sufficiently large common memory. A task flow chart is presented in figure 3.4. This chart is quite different from the ideal case of figure 3.1, because of practical considerations which will be described later.

The initial segment of the code (fig. 3.4) includes the input routines, initialization routines, and the grid-generation routines. This part of the code is serial. These serial operations require an almost insignificant amount of CPU time, so no effort was made to make them parallel. The input routines set the angle of attack, Mach number, and other important flow variables. Software switches are also

set which specify the grid option and initialization option used and determine whether viscous effects are to be included and if they are laminar or turbulent. The initialization routines allow the choice of an impulsively started solution or a startup from a previous solution which is obtained from a file. The grid-generation routine allows the selection of either a grid stored in a file or the default grid on a hemispherical nose with a cylindrical afterbody, which it calculates. At this point, the MIMD code must create the concurrent tasks.

The practical considerations which account for the differences in figures 3.1 and 3.4 are due to the VAX operating-system overhead required to initiate tasks in a time-shared system. In order to reduce the overhead, the code was implemented in such a way that the concurrent tasks are initiated only once. Each Fork in figure 3.1 would have required initialization of a new task in the operating system, which is costly. The method chosen was to use an initial Fork and then use Syncs instead of Joins , so that the concurrent tasks are always resident in the operating system. The difference between figures 3.1 and 3.4 is that the Forks and Joins are all replaced by Syncs, and the dashed lines in figure 3.4 represent the tasks in hibernation. Since this modification resulted from the accounting overhead of a time-shared system, it should be considered important when transferring a code to an MIMD machine that uses time-shared accounting.

After the initial segment is run, the program enters the main iteration loop. The main iteration loop contains the code segment, which updates the solution by one iteration step. This segment begins by calculating the boundary conditions explicitly. Currently, the boundary conditions are calculated serially since extensive effort would be required to decompose it into concurrent tasks. When the program has completed the boundary condition calculation, it records an EVENT which signals the right-hand side tasks to begin ("wake up from hibernation"). The right-hand-side operator is then calculated, followed by the explicit smoothing. This part of the code is executed concurrently. When the concurrent tasks are completed EVENTs are recorded for both, which is a signal for the main task to continue. (Recall that this would be a Join in the ideal implementation.) The concurrent tasks will be returned to the beginning of the loop and will hibernate until the required EVENT from the main task is recorded at the next iteration.

At this point the residual operator is available (at steady state, the right-hand side becomes zero), so convergence is tested by calculating the L_2 norm, defined by

$$||R||_2 = \left(\frac{1}{J \times K \times L} \sum_{j=1}^{J} \sum_{k=1}^{K} \sum_{\ell=1}^{L} |R_{j,k,\ell}|^2 \right)^{1/2} \qquad (3.15)$$

in which $R_{j,k,\ell}$ is the right-hand side of equation (3.12) at j,k,ℓ. Optional output routines give diagnostic information, such as a pressure distribution, when requested. In all applications, this section of code is serial. The final step in the main iteration loop is the implicit integration, which requires three sweeps through the data base. The MIMD codes execute each sweep concurrently with Syncs

between each sweep. Again, ideally, Forks and Joins would be used, but Syncs are implemented as in the explicit, concurrent tasks. This main iteration loop constitutes the majority of the CPU time since it is repeated 200-600 times for typical solutions and it is computationally intensive.

The final portion of the code contains the output routines and is entirely serial. This portion places the output data into output files for future data processing.

The basic flow of the program has been presented, but the problem of data protection must also be addressed. Data protection is the responsibility of the operating system and differs from machine to machine. Data protection can prevent the second processor from overwriting information generated by the first processor. The first type of data blocks are the local data blocks. These local data blocks are the most protected form of data since they are local to the unique subroutine which accesses them. In general, local data information is lost when the particular subroutine is exited. Protection of these blocks does not change from the serial implementation to the MIMD implementation. The next type of data block is the shared data block. An example of shared data in AIR3D is the main-solution data block, the vector q, which is addressed and required by all concurrent tasks. This data block must be stored in a shareable memory, or partitioned into pencils as discussed earlier. Other data blocks that must be stored in a shareable data group include the scratch space for the previous time-step solution, the right-hand-side operator, and intermediate variables of equation (3.11), the grid data, and all of the "bookkeeping" variables.

The task global data is the final data type that must be considered. Many temporary scratch arrays are used during execution. For example, the grid metric terms are calculated in scratch space when required. If both processors use the scratch space simultaneously, they will generate erroneous results. The VAX facility has an abundance of task global memory, so this is no problem. All that is required to prevent the other processor from accessing the scratch space is to not place the scratch space in shared memory. The CRAY X-MP, on the other hand, must have duplicate sets of scratch space to accommodate the lack of task global memory. The simplest procedure is to add a new dimension to all of the scratch arrays which represents the processors which will access it.

The NASA Ames MIMD test facility and the CRAY X-MP can be compared. The CRAY X-MP has the advantage that the two processors are coupled by the same operating system. Therefore, the main task can create all concurrent tasks and allocate them to separate processors. The VAX facility is much more loosely coupled. Each processor is guided by its own operating system and will not allow a task to be created by another processor. The result is the necessity to run a slave task on the second processor, whose sole function is to create the concurrent tasks for the second processor and to manage local processor data. Another difference is that the CRAY X-MP allows tasks to be contained within a single source code, whereas the VAX MIMD facility requires that each concurrent task be compiled and linked as a separate program. The CRAY X-MP requires only one program to run concurrently. The MIMD test facility, however, requires six programs--one for the main task and two for

concurrent tasks (one for the right-hand side operator, one for the left-hand side operator)--for each processor, which is one unfortunate limitation of the VAX MIMD test facility.

Another difference between the two NASA MIMD facilities is the memory system. As mentioned previously, the CRAY X-MP shares all of its memory, whereas the MA780 dual-ported memory in the MIMD test facility has only 1/4 Mbyte. There is no difficulty in putting the required shared data onto the CRAY X-MP, but all of the required shared data cannot be stored in the MA780 of the VAX facility. The resolution of this difficulty will be discussed later. Also, recall that the CRAY X-MP has no task global memory protection. The CRAY X-MP operating system assumes all nonlocal data are shareable data, and declares them as such. The VAX facility assumes all data are task-global unless explicitly stated as a shared memory block. This is a situation in which two opposing philosophies have greatly affected the structure of the data blocks.

3.5 Difficulties of Implementation on the VAX Facility

Several tradeoffs and compromises were made in the concurrent implementation of AIR3D for the VAX facility. First, because the code is computationally intensive, solutions could not be carried out to convergence on the two VAXs. The total dedication of the MIMD test facility at NASA Ames would have been required for an unacceptable period of time in order to reach a properly converged solution. Since this amount of run time was not considered, the test cases were run for only 10 to 20 iterations to get sample timings.

Another rather severe restriction encountered was the limited size of the MA780 dual-ported memory. Because the code is three-dimensional and each grid point is associated with 14 variables, which will quickly use up the shared memory, two steps had to be taken to tailor the problem to the limited memory. First, the grid metrics were removed from shared memory and a copy was placed in each processor's local memory. This eliminated 3 of the 14 variables required. It must be noted that in an unsteady problem, where the grid metrics change dynamically, this procedure would not be allowed. In the present problem, the passing of the grid metrics from one processor to the other occurs only during the initialization routines.

The second step taken was to reduce the grid density. The normal default case for the hemispherical-nosed, cylindrical-afterbody geometry was an array of 30×18×30 points. The VAX tests in this study used a 20×10×20 array, which results in 75% fewer grid points. At this level of coarseness, the code became unstable after a large number of iterations, and no attempt was made to seek a converged solution. This was not a serious limitation because the objective was to obtain a run-time comparison between a serial and an MIMD configuration; for this comparison convergence is not a necessary condition.

Another approach could be taken which uses pencils. This approach places the pencil containing only the data that is required by the second processor in the shared memory, which means that only half of the data has to be in shared memory at

one time. However, this also means that the data have to be reformatted and shifted at least twice for each set of two iterations, which adds an unfair burden to the timings. The approach of using pencils was not tested since the immediate objective of this research was to develop the procedure for the CRAY X-MP. The mode used for the VAX facility test suited this objective, so it mimics the approach used for the CRAY facility. However, one clear advantage that was discovered while using pencils is that the slave processor requires only a generic solver for all three sweeps of the implicit integration. This can result in a compact code and, on certain vector machines, could speed up computation time of the slave processor considerably.

In the original formulation of the program, metric derivatives were calculated as needed to avoid the extra memory space required to store them. The code was written to calculate all of the metric derivatives along a particular grid line when the subroutine was called. This presented no problem with the implicit part of the code, since all lines of data were decoupled. However, for the explicit part of the code, when the metric derivatives crossing the division between the two data halves were calculated, extra work was necessary to include some overlapping data in the calculation. The amount of overlap required was two points, for the fourth-order finite-differencing used in this code, since each half of the explicit calculation must "see" into the other half of the data for a distance of two points.

3.6 Results

The performance data of the NASA Ames MIMD test facility were obtained while running the system as a single-user system. This allowed the use of all the memory with only the operating system competing for CPU time. Execution times of the code were obtained by the computer system clock, and were used to determine the speedup performance. The two options tested were the Euler equations and the Navier-Stokes equations with the turbulent, thin-layer approximation.

Three timing measurements are presented for the two flow-solvers, which include progressively more hardware/operating system penalties. The measurements were made to demonstrate the variations in timing data that are found for different computer environments. The three measurements are for CPU task timings, total CPU timings, and real-time (stopwatch) timings. Speedup as used here is defined by

$$\text{Speedup} = \frac{t_{\text{serial}}}{t_{\text{concurrent}}} \tag{3.16}$$

The first set of timings, the CPU task timings, were made by recording the CPU time for each element, or task, of the program. The tasks are defined in a manner consistent with the previous discussion of the code. They include the setup, which is serial, both the serial and concurrent parts of the explicit calculation and the implicit integration; the boundary conditions and residual calculation; and the output routines. The serial portions of the explicit calculation and the implicit integration are primarily overhead required for parallel processing. This timing

procedure makes it easy to separate the serial and concurrent timings for extrapolating speedup for a larger number of iterations. Tables 1 and 2 compare the serial timings to the concurrent timings for the Euler and Navier-Stokes solvers, respectively. The data presented are representative of all the iterations since the task timings for each iteration were found to be very close, although not identical. From these timings it is clear that, within the main iteration loop, a signifiant improvement in computation time is achieved. The tasks which calculate the right-hand-side operator and invert the left-hand-side operator show speedups of nearly 2.0, with very little overhead. This result demonstrates the negligible effect of memory conflicts and synchronization times for the two-processor facility. The main iteration loop showed a speedup of 1.905 for the Euler solution and 1.914 for the Navier-Stokes solution. These results demonstrate that the overall speedup attained for this implementation is quite good for a single iteration cycle, and represents a respectable asymptotic limit for typical numbers of iterations required in CFD applications which include initialization and output routines. Curves of speedup versus number of iterations are shown in figures 3.5 and 3.6. These curves were computed using the following formula:

$$\text{Speedup} = \frac{(t_{setup} + n(t_{BC} + t_{RHS} + t_{LHS}) + t_{output})_{serial}}{(t_{setup} + n(t_{BC} + t_{RHS} + t_{LHS}) + t_{output})_{concurrent}} \qquad (3.17)$$

The two processors yielded different timings for the same concurrent task which were found to vary by as much as 5%. However, each processor was consistent with its own timings. As a result, it was decided to use the task timings for the extrapolated MIMD performance curves from the same processor that executed the serial code. Also, tasks were not penalized for certain overhead steps. For example, no task was charged for the CPU time required to "wake" a process or cause it to "hibernate." In view of this, steps were taken to ensure that most parallel processing overhead was properly charged.

The next two sets of timings represent the total time spent in executing the code (including all overhead) but excludes penalties for work done by the operating system in job management, etc. (see tables 3 and 4). Each time is the sum of the CPU times for the main process and each of the subprocesses. These results would be representative of non-time-shared machine, where no job-management interruptions are allowed, and all processors operate at the same speed. Again, for this implementation, where each processor has a slightly different speed, the speedup was measured by using the data for the same processor that was used for the serial code. These results are slightly lower than the previous results, as seen in figures 3.5 and 3.6, since all program-related overhead was properly charged.

The last set of timings are "stopwatch-style" timings (see tables 5 and 6). The measured time represents the total elapsed time from initial startup of the job to its conclusion, which gives the actual speedup in turnaround time one can expect for this system. This number includes all job-accounting overhead from the operating system and the difference in speed of the two processors. These timings are very machine-dependent and, given the MIMD testbed used, they can be assumed to

represent the low end of speedup (figs. 3.5 and 3.6). The difference between this timing and the timings of the first method represents the improvement that can be made within a given computer environment.

All three timings show that the Navier-Stokes solution, with the turbulent, thin-layer approximation, yielded a greater speedup than the Euler solution. This is a consequence of the larger number of calculations needed for the Navier-Stokes solution that appear in the parallel portions of the code. An even greater speedup of both solvers could be achieved if the boundary conditions and residual calculations were made parallel. These two portions of code are located in the main iteration loop and are therefore a significant cause of inefficiency. The initial setup and final output routines represent such a small fraction of the total CPU time, for the realistic case of a large number of iterations, that it is not useful to extract any parallelism they may contain. The timings reported here represent speedups that can be obtained using standard programming techniques.

3.7 Conclusions

A significant amount of parallel code has been identified in a standard benchmark CFD code. The code uses an approximate factored algorithm which, in a very straightforward manner, can be run on a concurrent-processing computer. This implicit algorithm was shown to achieve a speedup of greater than 1.9 on a two-processor system for representative solutions without undue effort. The general approximate factored algorithm is a good choice to run on the new generation of MIMD computers.

On processor systems with more than two processors, the code can be implemented using the method presented here. However, the memory-bus bandwidth problem will have to be overcome before this procedure can be implemented efficiently. Unless this problem is resolved, it is unlikely that approximate factorization will be able to take advantage of large-scale parallelism in the hardware without using new solution techniques. However, this memory-bus saturation problem is based on experience with other studies, and therefore should not be used to predict how many processors will cause bus saturation with the approximate-factorization-algorithm solution derived here.

Working with the computer environment used for this section brought to light some significant features. The two processors which were used required different execution times. Although this would not be significant on a machine such as the CRAY X-MP, it led to significant differences on the NASA Ames MIMD test facility. The conclusion was that concurrent algorithms should strive to be asynchronous, which would eliminate overhead from the different processor speeds. The concurrent implementation of the approximate factored algorithm presented in this section cannot be run asynchronously, so a more complex scheme of balancing processors would be necessary.

Some stumbling blocks at the beginning of the study showed the importance of memory management. Local and shared memories must be separate and protected. This

requirement led to a significant amount of interaction between the programmer and the operating system/computer architecture, which ideally should be eliminated with an operational computer. The need for the programmer to access the operating system for memory management must be eliminated. One approach would be to design additional language constructs available to the programmer that would help control memory protection. A sophisticated operating system and compiler that could handle these high-level language constructs would be required.

This study used a well-known CFD scheme which was not originally designed for MIMD machines. The largest penalty on the speedup parameter was due to the algorithm itself because of the serial modules embedded in the main iteration loop. The codes in the following sections avoid the use of serial code modules. They also have features which may eliminate memory conflicts, and they may allow asynchronous data transfers, thus avoiding the synchronization overhead.

4.0 OVERSET GRIDS--INCOMPRESSIBLE FLOW

The second problem studied on the NASA Ames MIMD test facility was the Steger, Dougherty, and Benek, Chimera Grid Scheme in an incompressible application (ref. 1). This application was chosen as an ideal candidate for MIMD study for several reasons. The primary motivation is its reduced shared-memory requirement, which results from the use of overset grids. Also, it allows the study of chaotic relaxation, a method which removes synchronization overhead. The overset-grid scheme is introduced as a method for solving problems with complex geometries. The name Chimera is derived from the mythological beast that has the head of a lion, the body of a goat, and the tail of a snake, and is used in analogy to complex geometries in CFD. The basic technique used is to generate a separate grid, which is simple and monotonic, for each part of a more complex geometry. The application used in reference 1 is an airfoil with a trailing-edge flap. Several applications of overset grids have been tested. Besides the Steger, Dougherty, and Benek stream-function solution which will be detailed later, an airfoil/flap configuration solving the Euler equations has been tested by Benek, Steger, and Dougherty (ref. 2), and a wing/nacelle potential solution has been implemented by Atta and Vadyak (ref. 5).

The problem of solving flow fields about complex geometries has proved to be difficult. Controlling grid point distribution and clustering on a single, global grid that is rectangular and monotonic in computational space is not always possible. The embedded grid technique offers a possible solution.

The basic idea behind overset, or embedded, grids is that the complex geometry is divided into several elementary pieces. For example, the airfoil and flap configuration in figure 4.1 can be divided into two simple grid problems, with the flap grid embedded in the airfoil grid as in figure 4.2. Each grid can then have the desired densities, clustering, and skewness. In addition, simple grid-generation

routines can be employed to create the respective grids. The only added difficulty is in the handling of information that crosses the grid boundaries.

A flow-field solver must be chosen that retains the simplicity of single grid problems. Also, the modified overset-grid solver must allow the use of implicit schemes for stability considerations. For the Chimera grid, Steger et al. propose a method which greatly simplifies any grid-overlapping difficulties, and results in minor coding changes (ref. 1). The scheme can be used on most implicit methods, and will be detailed later. Atta and Vadyak have devised a similar method for their problems. They freeze the grid boundaries for a period of 40 to 50 iterations, and then interpolate for new, updated boundaries, a cycle which repeats four to five times for the global solution.

A convenient property of the overset grid scheme is that the communication between the grids can be isolated from the main flow-solver. This property allows the grid communication procedure to be removed from the main flow-solver and thus be contained as a boundary-condition statement. The numerical procedure for handling the grid boundaries can then be studied as a numerical boundary scheme. An additional benefit of overset grids is that the required data for communication is less than 5% of the total data. This minimal shared-data requirement allows large overset-grid problems to be solved on MIMD machines with minimal shared memory.

The apparent parallel structure of the multiple grids and the minimal shared-memory requirements allows this approach to fit nicely onto an MIMD machine, such as the VAX MIMD facility. This section focuses on the details and results of implementing the Steger, Dougherty, and Benek stream function code on the NASA Ames MIMD test facility.

4.1 Solution of Chimera Grid

Although the specifics of the Chimera grid technique are described in references 1 and 2, they will be summarized here. The problem that is solved in this study is the airfoil/flap configuration of figure 4.1. The airfoil uses a 150×47 "O" grid, and the flap uses a 70×11 "O" grid that is laid on top of the airfoil grid (fig. 4.2). The overset flap grid is a simple grid that can be solved as any single grid problem, except that its outer boundary variables are supplied by interpolation from the major airfoil grid. The airfoil grid, however, must have a hole cut out of it to exclude points that lie within the flap surface (fig. 4.3). This hole also covers the flow region around the flap to minimize the effect of locally high gradients around the flap.

The modification to account for the hole is to choose a contour within the flap grid, and flag all airfoil grid points that lie within that boundary. This flagging is accomplished in an array called IBLANK. If the point is inside the hole, IBLANK is set to zero; otherwise it is unity. Figure 4.4 shows the blanked-out points as "x"s. Once these points have been flagged, all fringe points are flagged; the fringe points represent the points that will be interpolated from the minor grid. In figure 4.4, the points marked "O" are the fringe points. This procedure is

entirely automated, and the IBLANK array now becomes the only required modification, other than the interpolated boundary points, that is required in a general flow-solver. The scheme used in this section of the study will be presented, and examples of the application of the IBLANK array will be given.

The equation set that is applied to this problem is the stream function equation

$$\nabla^2\psi = 0 \tag{4.1}$$

Therefore this problem is incompressible, and contains no discontinuities which could introduce added complexity, as will be pointed out later. Since the problem is elliptic, central differencing is used. The finite-difference scheme is thus

$$\delta_{xx}\psi^n_{j,k} + \delta_{yy}\psi^n_{j,k} = 0 \tag{4.2}$$

where

$$\delta_{xx}\psi_j = (\psi_{j+1} - 2\psi_j + \psi_{j-1})/(\delta x)^2 \tag{4.3a}$$

$$\delta_{yy}\psi_k = (\psi_{k+1} - 2\psi_k + \psi_{k-1})/(\delta y)^2 \tag{4.3b}$$

The solution algorithm used is a standard ADI scheme. If the finite-difference form is recast in delta form with the relaxation parameter, in which the relaxation parameter is used for time-like differencing, the equation becomes

$$(I - \omega\delta_{xx})(I - \omega\delta_{yy})(\psi^{n+1} - \psi^n) = \omega(\delta_{xx} + \delta_{yy})\psi^n \tag{4.4}$$

This approximate-factored equation can be solved in a two-step solution procedure similar to AIR3D's three-step solution procedure. This procedure is

$$\begin{aligned}(I - \omega\delta_{xx})\delta\psi^* &= \omega(\delta_{xx} + \delta_{yy})\psi^n \\ (I - \omega\delta_{yy})(\psi^{n+1} - \psi^n) &= \delta\psi^*\end{aligned} \tag{4.5}$$

This scheme can be solved without modification to run on the minor grid, with appropriate handling of the boundary conditions. On the major grid, the array IBLANK must be used to keep this general form.

The array IBLANK is used in the following manner. At each inversion stage of equation (4.5), the basic algorithm yields a tridiagonal system which, for an interior grid of only seven points, would be

$$\begin{bmatrix} b & c & & & & & \\ a & b & c & & & & \\ & a & b & c & & & \\ & & a & b & c & & \\ & & & a & b & c & \\ & & & & a & b & c \\ & & & & & a & b \end{bmatrix} \begin{bmatrix} \delta\psi_1 \\ \delta\psi_2 \\ \delta\psi_3 \\ \delta\psi_4 \\ \delta\psi_5 \\ \delta\psi_6 \\ \delta\psi_7 \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \\ r_7 \end{bmatrix} \tag{4.6}$$

where $\delta\psi = (\psi^{n+1} - \psi^n)$. Now suppose that the fourth and fifth points are blanked out by IBLANK. The result would be to replace the elements a, c, r_4, and r_5 by zero and change the element b to one. The new system becomes

$$\begin{bmatrix} b & c & & & & & \\ a & b & c & & & & \\ & a & b & c & & & \\ & & & 1 & & & \\ & & & & 1 & & \\ & & & & a & b & c \\ & & & & & a & b \end{bmatrix} \begin{bmatrix} \delta\psi_1 \\ \delta\psi_2 \\ \delta\psi_3 \\ \delta\psi_4 \\ \delta\psi_5 \\ \delta\psi_6 \\ \delta\psi_7 \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ 0 \\ 0 \\ r_6 \\ r_7 \end{bmatrix} \tag{4.7}$$

The IBLANK array conveniently takes care of the bookkeeping for this procedure by replacing or not replacing the elements, depending upon the value of IBLANK, since IBLANK is either one or zero, depending upon the point. This procedure is identical in concept to resetting ω, the relaxation parameter in equation (4.5), to zero for the flagged points. The simplicity of this procedure and its adaptability to current algorithms is what makes it so useful.

The system in equation (4.7) is used to calculate the values of $\delta\psi$ at all grid points. However, $\delta\psi$ is set to zero at the blanked-out points within the hole (flagged) boundary. These points must be updated in some manner to maintain a consistent solution across the hole. This is done by interpolating values from the overset grid.

The interpolation of data from one grid to the other is an integral part of the overset grid scheme. All overlapping-grid boundary data are obtained in this manner. The minor grid gets its outer, k_{max}, boundary data from the major grid by

interpolation. The major grid gets its hole, or flagged, points from the minor grid by interpolation. In the following section, several improved methods of handling these overset-grid boundary points will be discussed. With these methods, the interpolation used is a second-order interpolation or

$$\psi_{\ell} = \psi_m + \delta x\, \psi_x\big|_m + \delta y\, \psi_y\big|_m + \frac{\delta x^2}{2} \psi_{xx}\big|_m + \delta x \delta y\, \psi_{xy}\big|_m + \frac{\delta y^2}{2} \psi_{yy}\big|_m \qquad (4.8)$$

where ψ_{ℓ} represents the point that needs to be updated and the terms $(\ )_m$ are calculated from the other grid at the nearest point, m. Note that this formula is given in x,y coordinates and must be transformed into the computational coordinates ξ,η.

The appropriate conditions for the boundaries not associated with the overset grids are the conventional boundary conditions. Far-field conditions on the major grid are velocity, or Neumann, conditions. Tangential velocity and a Kutta condition are required to specify ψ on the solid body surfaces of the airfoil in the major grid and the flap in the minor grid. The final boundaries, in computational space, are easily implemented by using an "O" grid, which makes the scheme periodic. These conditions are coded in the standard method as though no overset grid scheme existed.

4.2 Implementation of Two Grids on Two Processors

The basic method for implementing the stream function airfoil/flap problem has been presented. This scheme was tested by Steger et al. and implemented in such a way that overset grid boundary data were updated by interpolation explicitly (ref. 1). The procedure used in that program was to complete an iteration on the major grid, then complete an iteration on the minor grid, and finally interpolate for flagged points. With this explicit coupling of the grids, the two grids are solved independently of each other during each iteration. This loose coupling is the primary motivation for using an MIMD machine. The MIMD version of the code is essentially the same implementation as the serial version except that the iterations performed on the major and minor grids are done concurrently.

The basic features of the concurrent code will be reviewed in the context of MIMD architectures. The code was tested on the NASA Ames MIMD test facility described earlier. The particular characteristics of this test facility were used to dictate the approach taken to implement the concurrent code.

The code begins with an initialization which includes the grid package. The grid is generated externally and is introduced as a data file. The grid package flags out the hole region, finds nearest neighbors for interpolation, and sets up IBLANK. This portion of the code is serial, but represents a small part of the overall time required for execution. Once the initialization has been completed, two tasks are spawned. Each task represents one grid-solution procedure. The tasks

are identical and use a grid number (ID parameter) to determine which grid will be used. There is a Sync point at the beginning of each iteration loop within the tasks which allows the solvers to synchronize before they receive the interpolated data. The main program is essentially idle for the rest of the computation. Figure 4.5 is a diagram showing the Fork of the spawning and the Sync points.

The iteration loop is straightforward, and is not too different from the serial version. The Sync point forces the two grids to start each iteration together. The interpolated points are then updated explicitly from the previous time step so that the solution is identical to the serial version. Here, however, this procedure is being executed simultaneously in the two concurrent tasks. The ADI solver is then employed in each task, and an iteration is completed. The final step is to interpolate for the flagged points in the other grid before a new iteration begins. Note that this is the actual computation of overset boundary data, which is stored in an intermediate buffer. The updating procedure at the beginning of each iteration loop is only for overwriting the flagged points with the contents of the intermediate buffer. The separation of the two interpolation tasks, the computation and updating, is essential to allow for the required synchronization between these two tasks. This completes an iteration loop, so a synchronization occurs as a new iteration begins.

In this particular application, the two grids are greatly mismatched in the total number of grid points. Since the number of operations is roughly proportional to the number of grid points, it is apparent that there will be approximately a 10:1 ratio of computations required in the major grid task compared to the minor grid task. Thus it would be expected that the CPU time required for the minor grid is one-tenth that required for the major grid. This imbalance is an obvious flaw since one processor, in an MIMD machine, would be idle 90% of the time. In fact, this idle time was observed. No attempt was made to balance the two processors in this code since the purpose of this problem was to develop and explore a concept rather than to develop a production scheme. For a production scheme, an attempt would have to be made to balance the work, which could most easily be achieved by balancing the number of grid points in the two grids.

Despite the idle time of one processor, there are two basic advantages of the overset grid scheme over codes such as the concurrent version of AIR3D discussed previously. One advantage is that, at most, one synchronization per iteration is necessary. This eliminates synchronization overhead, thus speeding up the computation. If the overset grids were balanced with the major grid, this would be a very efficient use of the machine in terms of operating-system costs.

The second, and most significant, advantage of this scheme is its memory requirement. The tasks can be easily divided so that the majority of the data is not shared. Ideally, the tasks corresponding to each grid would be contained locally with each processor. The only data that must be shared is the small amount of boundary data required for the overset grid and hole boundaries, which is less than 5% of the total data. Thus, a machine with a very small shared memory can be used for this program. In fact, the shared-memory requirements are so small that a network of computers with a high-speed, data-transfer link could be used. Memory-

contention problems would be nearly eliminated since the amount of shared-memory access is small compared to the overall execution time per iteration. Unfortunately, no machine capable of testing this idea is available since existing machines either have many small processors that are too small to handle a complete grid, or they have only a few processors and consequently do not have noticeable memory-contention problems.

The use of local data, task-global data, and shared data can be explored further. As mentioned previously, some machines have no memory protection for task-global data. The concurrent scheme presented here is based almost entirely upon task-global memory. The NASA Ames MIMD facility has an abundance of task-global memory and very little shared memory. Thus the development of the code followed the theory that limited shared memory would be the more restrictive case, and any code designed to run on such a machine would automatically fit onto machines with larger shared memories. However, the CRAY X-MP has all of its memory shared, but it has no task-global data. A simple solution to this problem would be to protect all common blocks by duplicating them, which is accomplished by adding an extra subscript to the task-global data. The modified code, however, would then not be transportable to small shared-memory machines. If a task-global data block were used on the CRAY X-MP and similar machines, then this code and others like it would be transportable to all MIMD machines. Cray Research has recently announced plans to implement task-global memory, as a result of pressure from users (which exemplifies the need for communication between computer users and computer designers).

4.3 Results

Results of this concurrent scheme are numerically equal to the results obtained by Steger et al. (ref. 1). The incompressible case shows no streamline alterations as they cross grid boundaries (see fig. 4.6). The pressure coefficient is shown in figure 4.7, and the effect of the flap on the airfoil is definitely noticeable. Without the flap, the airfoil C_p plot would be symmetric on the top and bottom surfaces. However, the two surfaces are distinctly different, which verifies that the influence of the flap is taken into account.

Timing results for the computation were not important for this study. The main purpose of this study was to verify that acceptable solutions can be obtained using overset grid schemes on MIMD computers. Although the particular problem chosen was not ideal for MIMD machines, it demonstrates that overset grid schemes are well suited for MIMD machines.

Two conclusions can be drawn from this section of the study. First, the tasks must be properly balanced for an overset grid scheme to be efficient on an MIMD machine. Secondly, because of operating-system considerations, synchronizations should be removed from the scheme, giving an asynchronous scheme.

Asynchronous methods need to be developed to suit the time-shared, multitasking queue procedure. The multiprocessor, multitasking queue procedure was introduced as the most likely choice for the mode of operation of a MIMD system. If the

concurrent version of AIR3D were run in such an environment, the data blocks (pencils) would be executed in an arbitrary fashion dictated by the operating system's queuing algorithm. With a two-processor system and two data blocks, the second block could remain in the queue until the processing of the first block is completed. The program would then have to wait until the second block is completed before it could submit the next block into the queue for the next sweep. An asynchronous scheme would not be restricted by such a queuing scheme. The individual blocks could be processed and immediately resubmitted to the queue asynchronously, without regard to the state of the other concurrent blocks. Thus, an asynchronous scheme would not depend upon the queuing algorithm to achieve optimal performance. In the section that follows, it is assumed that a completely random queuing procedure is used.

A less important motivation for developing asynchronous schemes is to remove synchronization overhead. Two primary factors contribute to the synchronization penalty--the CPU execution time for the synchronization and the waiting time of a processor. The study of AIR3D demonstrated that the first of these penalties, the execution time, is negligible, but the waiting period could reach 5% of the total time. An asynchronous scheme would avoid this waiting period and thus achieve a possible 5% improvement in speed. However, this additional speed may not be enough to justify additional work to create an asynchronous scheme since far superior speed may come from new developments.

The problem of balancing processor work could also be accomplished by balancing the number of grid points for each overset grid task. Another approach is to use a nonuniform time step. In problems in which a steady-state solution is the desired result, a uniform time step or relaxation step is not required. A solution can converge to the steady-state solution faster if an optimum time step or relaxation step is chosen for each grid point. This can be applied point by point or region by region. Therefore, it is not unreasonable to fractionally time-step one region or relax a region for several iterations while another region is updated only once. So, in principle, the overset grid can be relaxed in this application for 10 iterations while the major grid is relaxed for a single iteration. This would balance the utilization of the two processors. It should be noted that for the sample problem of this section, a 10:1 ratio of relaxation steps would not be very efficient.

A simple approach following the above procedure would be to synchronize the two grids at the beginning of each iteration on the major grid with the beginning of every tenth iteration on the minor grid. The following discussion will show that this synchronization is not required. Although on this particular problem, this approach may not be efficient, there are problems for which an asynchronous calculation would be of advantage. An example is a problem in which the time step in a time-accurate calculation is mismatched because of stability problems. In this example, the time step for each grid can be adjusted with the processing speed to balance with other grids.

4.4 Asynchronous Iteration

The chaotic relaxation scheme will be presented to justify this attempt at an asynchronous solution to an overset grid problem. The chaotic relaxation scheme is a new approach to solving MIMD problems, and this paper discusses the first known application of such a scheme to CFD or problems with multiple grid regions. An application of chaotic relaxation to this problem will also be presented.

Asynchronous iteration is unique to MIMD implementations of computer codes. In MIMD architectures, asynchronous procedures can occur which add to the randomness or chaoticness of a scheme. In a chaotic scheme, each task will proceed randomly without any information on the states of the other tasks. There has been some past interest in such random schemes, but very few applications. The research has primarily focused on developing theorems which will prove whether a scheme will or will not converge, and with which limitations. Pioneering work in this area was done by Chazan and Miranker in 1969 (ref. 13). More recently Baudet (refs. 14,25) (1978) has generalized some of Chazan and Miranker's results.

Baudet's theorem will now be discussed, and some general definitions of asynchronous schemes will be presented. The theorem will be presented for a linear system of equations in which we assume that a fixed-point solution to the problem exists. The model problem we will investigate is the fixed-point problem for an operator F with a vector x which satisfies the equation

$$x = F(x) \tag{4.9}$$

The following definition for asynchronous iteration is a copy of the definition given by Baudet (ref. 14).

> If F is a vector operator as in equation (4.9) and x° is an initial vector then an asynchronous iteration corresponding to operator F is defined recursively as
>
> $$x_j^n = \begin{cases} x_j^{n-1} & \text{if } j \notin J_n \\ f_j\left(x_1^{s_1^n}, \ldots, x_m^{s_m^n}\right) & \text{if } j \in J_n \end{cases}$$
>
> where x_j^n is a sequence of vectors, $J = \{J_j \mid j_n = 1,2,\ldots\}$ is a sequence of nonempty subsets of $\{1,2,\ldots,m\}$ and $S = \{s_1^n, \ldots, s_m^n\}$ is a sequence of elements from the set of natural numbers. The following restrictions on j and S must also hold for $j = 1,2,\ldots$:
>
> i) $s_j^n \leq n - 1; \quad j = 1,2,\ldots;$
>
> ii) s_j^n tends to infinity as n tends to infinity;
>
> iii) j occurs infinitely often in the sets $J_n, \quad j = 1,2,\ldots.$

This generalized definition was expanded from Chazan and Miranker's definition, where condition ii) is replaced by the more restrictive condition:

ii') a fixed integer, s, exists such that $n - s_j^n \leq s$ for $n = 1,2,\ldots$.

This more restricted condition was used by Chazan and Miranker to define chaotic relaxation. One can imply that a scheme that is a chaotic relaxation scheme is an asynchronous, iterative scheme. The scheme studied in this research clearly falls into the general category of asynchronous iterations. Since each iteration increases the iteration difference, $n - s_j^n$, the condition ii') does not necessarily hold.

It may be easier to understand this definition by considering a system of processors solving a set of explicitly coupled equations. Suppose, for example, there are m processors and $2m$ equations, or elements, to solve; only half of the system can be updated at a time. The other half, chosen by a random queuing procedure, is held back at the previous iteration level. The elements chosen by the queuing algorithm to be updated are the elements in the sets J_n, where n denotes the cycle number. For the next cycle, the processors are again given new elements to work on, and in some cases the previously updated solution is again updated and the nonupdated solutions are again ignored. Thus, in the solution of each element, data from the first iteration level and the second iteration level are mixed. This mixing is assumed to occur completely at random, and the difference in iteration levels between elements will be a random function of the iteration level. The iteration level of each element is defined in the set S, where s_j^n represents the iteration level of the j element and n is the number of cycles the processors have executed. The three conditions imposed to classify the scheme as an asynchronous iteration are that no implicit dependence of points in the solution procedure exist; that the iteration level, s_j^n, for the least-updated element tends toward infinity as the number of processing cycles tends toward infinity; and that every element must be chosen by the queuing algorithm infinitely many times in an infinite computation.

Baudet's theorem for asynchronous iterations requires the existence of a contracting operator. A contracting operator is a specific case of a Lipchitzian operator, which is defined as follows: For every x and y the operator F is a Lipshitzian operator if there exists a nonnegative $n \times n$ matrix A such that

$$|F(x) - F(y)| \leq A|x - y| \tag{4.10}$$

and the inequality holds for each component. The matrix A is called a Lipchitzian matrix. A contracting operator adds the constraint that the corresponding Lipchitzian matrix A must satisfy $\rho(A) < 1$, where $\rho(A)$ is the spectral radius of the matrix A. For the case of a linear operator, such as the particular finite-difference operator used in this study, where $F(x) = Ax + b$, this definition can also be stated: F is a contracting operator if and only if $\rho(|A|) < 1$.

Baudet's theorem for asynchronous iterations is stated as follows:

> If F is a contracting operator (on a closed subset D of $\mathscr{R}^n$ with $F(D) \subset D$), then any asynchronous iteration (defined by the sets j and S) corresponding to F with initial conditions x^0 converges to the unique fixed point of F.

This theorem can be useful for determining the success of solving schemes asynchronously. However, developing a model problem which uses overset grids to test this theorem has proven difficult. The interpolation matrix, which couples the grids, adds complexity to the contracting operator; so the spectral radius of $|A|$, $\rho(|A|)$, must be computed numerically. In this application the asynchronous method was tested by experimentation so convergence implies that $\rho(|A|) < 1$.

A practical argument why an asynchronous procedure might work follows. If the overset grid boundaries are updated asynchronously, the procedure is to read the boundary data that is available in the intermediate buffer as it is required. Since the other grid will be at any point in its execution, there is no knowledge of which iteration level the intermediate buffer represents. In fact, portions of the overset boundary can be at a different iteration level when the information is required from the other grid. But as the solution progresses, the difference in the boundary data at progressive iteration levels should drop to zero, since the problem has a steady-state solution and it is expressed in delta form; so the overall effect of receiving mixed data will not make any difference to the final steady-state solution. The transient solution, however, may have discontinuous data at the boundaries, which can cause undesirable waves to develop. It is conceivable that such a scheme would be unstable and would never converge. Also, the asynchronous effect is totally random in nature and can vary from run to run. However, provided that the scheme uses a contracting finite-difference operator, the scheme will have sufficient damping to damp out any unstable waves that develop.

The concept of asynchronous iterations has been applied to the incompressible airfoil/flap problem and is presented in the following section. The scheme falls into the more general category of asynchronous iterations and not the category of chaotic relaxation schemes since, as will be shown later, the condition ii) does not hold. Unfortunately, as mentioned previously (see note), the details of the finite-difference technique are too complex to allow the calculation of $\rho(|A|)$. Therefore, numerical experimentation was used to test the convergence.

4.5 Implementation and Results of Asynchronous Iteration

The application of chaotic relaxation to the overset grid problem is achieved by removing the synchronization at the beginning of the iteration loop. The resulting code has a single Fork when the tasks are spawned, and an Event which signals a converged solution. Results obtained from this procedure are indistinguishable from the synchronous results. Figures 4.8 and 4.9 show the results from the asynchronous solution. When figures 4.8 and 4.9 were overlayed on figures 4.6 and 4.7,

respectively, the asynchronous and synchronous solutions were found to be identical. The timings are of interest. Since each processor was operating at 100% utility, the minor grid completed approximately 10 times the number of iterations that the major grid completed. The convergence is based upon the major grid, which took the same number of iterations as its synchronous counterpart. Thus the solution converged in 483 iterations of the major grid, and, in this time, 4789 iterations were completed on the minor grid for one particular run. This mismatch in the iteration number occurs because of the mismatch in grid sizes and demonstrates that grid balancing can be useful.

A simple way to interpret the results with such a mismatched iteration level is to look at it as fractional time-stepping. The minor grid receives a single set of outer boundary values for a period of about 10 iterations. The solution, therefore, is given an opportunity to relax to a solution based on that set of boundary data. This boundary scheme is stable, since the ADI algorithm is stable with Dirichlet boundary data on the outer boundary. Since the major grid does not appear to benefit from the refined minor-grid solution, the extra iterations are unhelpful and unnecessary. However, this procedure has shown that overlapping grids do not need to match iteration times exactly, but can be offset or skewed. The example presented here is an extreme case in which the iteration skewness is about 10:1. The synchronous case is a case in which the iteration skewness is fixed at exactly 1:1. A steady-state solution with a time-dependent transient solution will be presented in the next section with a skewness close to but not exactly, 1:1.

4.6 Conclusions

An overset grid scheme has been implemented on an MIMD facility, producing results which successfully recreate results obtained on a serial machine. The most noteworthy conclusion from this portion of the study is that overset grid schemes which are implemented on multiprocessor computers have small shared-memory requirements. This is significant since it implies that memory-system networks can be designed with little worry of memory-contention problems. An understanding of the memory contention problem, which will be important when implementing this code onto a CRAY X-MP, has also been achieved. It is apparent from a comparison of the concurrent version of AIR3D and the overset grid problem that the architecture of the memory system can be an important factor when choosing an algorithm.

The incompressible calculations have also demonstrated the feasibility of asynchronous and chaotic methods for linear problems. Currently, this scheme is the only known application of an asynchronous iteration technique used to solve a practical CFD problem.

This section has dealt only with incompressible flow, which does not have the nonlinearities common to many CFD applications. The success of the overset grid scheme and its asynchronous implementation can partly be attributed to this lack of nonlinearities. Therefore, it is necessary to study overset grids on a compressible-flow problem to investigate its usefulness on general CFD problems. The study

of the nonlinear effects on the overset grid boundaries is the focus of the following section.

5.0 OVERSET GRIDS--COMPRESSIBLE FLOW

The third and final section of this study represents another application of the Chimera grid scheme. The name Centaur was chosen for the scheme after the mythological beast which is half man, half horse, following the lead of the name Chimera. This application of the Chimera grid scheme addresses complex flow fields in compressible flows and in particular studies shock waves that cross the overset grid boundary. A blunt body, with a cylindrical-nosed wedge and a 6.5° afterbody, in a free-stream Mach number of 2.0, was chosen to serve as a convenient test vehicle to study overset grid problems.

Many aerodynamic flow fields of interest contain shock waves or shear layers which must be properly resolved. For a scheme to be useful in aerodynamic calculations, it must resolve and locate these shock waves properly. Therefore, the Chimera grid scheme, when applied to transonic flows must accurately define shock waves without adversely affecting the solution in other ways. Shocks may cross overset grid boundaries in many practical aerodynamic applications. This section focuses on the problems that arise when a shock wave crosses an overset-grid boundary.

Benek, Steger, and Dougherty applied the Chimera grid scheme to the solution of the Euler equations about an airfoil/flap configuration in the transonic flight regime (ref. 2). Their single trial resulted in an ill-defined shock wave at the grid boundaries (fig. 5.1) and exhibited poor convergence. The goal of this portion of the study was to understand and correct the adverse effects caused by shock waves interacting with the overset grid boundary. The blunt-body geometry was chosen as a practical test problem which allowed the boundary regions to be studied conveniently. A scheme which can successfully solve nonlinear problems, calculate the flow field about complex configurations, use multiprocessor computers, and compete favorably with popular serial techniques will be of interest to the CFD community.

5.1 Blunt-Body Grid--Centaur

The motivation for choosing the blunt-body grid is that it contains a shock wave and its grid requirements are conveniently small. The supersonic case studied needs only a 25×21 r-θ major grid to resolve the shock wave (fig. 5.2). The major grid calculation can also make use of simple numerical-boundary procedures because of its supersonic nature. These boundary procedures use Dirichlet boundary data on the inflow boundary; symmetry conditions, which can be included directly in the finite-difference scheme, on the centerline; extrapolation on the outflow boundary, resulting in a wedge-afterbody geometry; and solid-body conditions on the body

surface. Simplifying the boundary conditions allows the investigator to concentrate on the overset grid boundaries.

The overset grid has been designed so that it represents a practical test for the Chimera grid scheme. A hole is not required in the major grid because a secondary body is not present, but one was added so that the problem would simulate a more general application. Simplicity was desired, so the symmetry plane was used as one boundary of the overset grid (fig. 5.3). Another boundary is aligned with the body so that solid-body conditions can be applied. The remaining boundaries are situated so that one boundary is perpendicular to the flow direction and the other is a constant θ ray of 47.5°. When the minor grid is overset on the major grid (fig. 5.3), the mesh-line skewness at the leading boundary is apparent.

The advantage of this particular overset grid is in its relationship to the flow direction and shock direction. The front plane is perpendicular to the incoming, free-stream flow, and by simply adjusting its horizontal position the size of the minor grid can be scaled (figs. 5.3 and 5.4). The shock wave is parallel to the front edge of the minor grid, and the interaction of the shock with the grid boundary can be studied. The shock wave crosses the upper plane in the normal direction and represents the problem of most interest to this study. The interaction between these two boundaries and the shock wave can be isolated and studied separately.

The size of the hole in the major grid can also be easily adjusted, with a fixed number of overlapping points, regardless of the minor (overset) grid size. This will allow the study of hole-shock interactions when the shock and hole boundaries are close. Figures 5.3 and 5.4 show the two grids and the blanked-out points. In one part of the investigation, no hole was used, so the interactions between the two grids could be studied by allowing the major grid to be isolated from the overset grid.

5.2 Transonic Flow Solver--ARC2D

The blunt-body problem studied is for a steady, two-dimensional, compressible flow. The Euler equations for this case become

$$\frac{\partial q}{\partial \tau} + \frac{\partial F}{\partial \xi} + \frac{\partial G}{\partial \eta} = 0 \tag{5.1}$$

where the vectors q, F, and G are defined by

$$q = J^{-1}\begin{bmatrix}\rho\\ \rho u\\ \rho v\\ e\end{bmatrix}, \quad F = J^{-1}\begin{bmatrix}\rho U\\ \rho uU + \xi_x p\\ \rho vU + \xi_y p\\ U(e + p)\end{bmatrix}, \quad G = J^{-1}\begin{bmatrix}\rho V\\ \rho uV + \eta_x p\\ \rho vV + \eta_y p\\ V(e + p)\end{bmatrix} \tag{5.2}$$

and where

$$p = (\gamma - 1)\left[e - \frac{1}{2}\rho(u^2 + v^2)\right] \tag{5.3}$$

$$U = \xi_x u + \xi_y v \tag{5.4a}$$

$$V = \eta_x u + \eta_y v \tag{5.4b}$$

and

$$J^{-1} = x_\xi y_\eta - x_\eta y_\xi \tag{5.5}$$

Note that this is the same equation set used for the inviscid part of AIR3D, except that the third dimension has been eliminated and the possibility of a moving grid has been removed. Again, these equations are expressed in terms of a general, curvilinear, coordinate transformation. The algorithm used to solve all points, except the boundary points of each grid, is the Beam and Warming approximate-factorization algorithm (ref. 19). Their implicit algorithm, written for two dimensions, is

$$(I + h\delta_\xi A^n)(I + h\delta_\eta B^n)\Delta q^n = -\Delta t(\delta_\xi F^n + \delta_\eta G^n) = R^n \tag{5.6}$$

where R^n is the residual operator, which tends to zero as the solution approaches steady state, and $\Delta q^n = q^{n+1} - q^n$. The matrices A^n and B^n are local linearizations of F^{n+1} and G^{n+1}, that is

$$F^{n+1} = F^n + A^n(q^{n+1} - q^n) + o(\Delta t^2) \tag{5.7a}$$

$$G^{n+1} = G^n + B^n(q^{n+1} - q^n) + o(\Delta t^2) \tag{5.7b}$$

where $A^n = (\partial F/\partial q)^n$ and $B^n = (\partial G/\partial q)^n$.

As described previously, the major grid will have one or more holes in the general application. This hole is marked by the IBLANK array, as in the previous section, and results in a simple modification of the overset-grid solver. The Beam and Warming algorithm is modified as follows,

$$(I - \text{IBLANK}h\delta_\xi A^n)(I - \text{IBLANK}h\delta_\eta B^n)\Delta q^n = -\text{IBLANK}\Delta t(\delta_\xi F^n + \delta_\eta G^n) \tag{5.8}$$

where $h = \Delta t/2$ for this study. The matrices which result from equation (5.8) have a similar structure to the example of equation (4.7), except that the elements of the tridiagonal system of equation (5.8) are 4×4 blocks. With this procedure, all grid points in both grids are updated, except the boundary points and points lying within the hole region of the major grid. The outer boundary of the overset grid and the major grid hole points are updated by means of an interpolation procedure, which is the mechanism that couples the two grids' solutions. The interpolation procedure applied to the outer boundary of the overset grid was found to be critical and was the primary focus of this study. The major grid outer boundary, which was not critical for this study, used the boundary conditions described above.

5.3 Overset Grid Boundary Schemes and Results

The different strategies for updating the outer boundary of the overset grid will not be presented. The solution procedure follows that of the previous section. At the completion of an iteration cycle, each grid task performs the calculation to update the other grid's boundary. These boundary data are stored in a temporary data block in shared memory. As the next iteration begins, the temporary data is used as input to a numerical boundary scheme which updates the overset grid boundary and the hole boundary. All of the schemes studied start with the same interpolated data in shared memory, but the data are treated differently in the numerical boundary scheme.

5.3.1 Direct interpolation- The method used by Benek et al. was repeated first (ref. 2). The interpolation uses simple second-order, nine-point interpolation. The interpolation in transformed coordinates is

$$
\begin{aligned}
q^* = q_m &+ \frac{1}{2}\xi_x\left[(\xi_y)_\xi \frac{\partial q}{\partial \xi} + (\eta_y)_\xi \frac{\partial q}{\partial \eta}\right]\Bigg|_m + \frac{1}{2}\eta_x\left[(\xi_y)_\eta \frac{\partial q}{\partial \xi} + (\eta_y)_\eta \frac{\partial q}{\partial \eta}\right]\Bigg|_m \\
&+ \frac{1}{2}\xi_y\left[(\xi_x)_\xi \frac{\partial q}{\partial \xi} + (\eta_x)_\xi \frac{\partial q}{\partial \eta}\right]\Bigg|_m + \frac{1}{2}\eta_y\left[(\xi_x)_\eta \frac{\partial q}{\partial \xi} + (\eta_x)_\eta \frac{\partial q}{\partial \eta}\right]\Bigg|_m \\
&+ \xi_x\xi_y \frac{\partial^2 q}{\partial \xi^2}\Bigg|_m + (\xi_y\eta_x + \xi_x\eta_y)\frac{\partial^2 q}{\partial \xi \partial \eta}\Bigg|_m + \eta_x\eta_y \frac{\partial^2 q}{\partial \eta^2}\Bigg|_m
\end{aligned}
\tag{5.9}
$$

where q^* are the interpolated values and the quantities $(\ )_m$ are obtained from the other grid. The numerical boundary scheme employed by Benek et al. was to use these interpolated values as Dirichlet data to update both the overset grid boundary and the hole boundary (ref. 2).

Two overset grid configurations were tested with the direct-interpolation boundary scheme. The first used a small overset grid so that the front of the grid

was positioned approximately one-half radius in front of the steady-state shock position (fig. 5.3). The major grid hole was designed so that four unflagged points overlapped the minor grid, which positioned the hole boundary in the shock region. Results for this test show a poor solution at the edge of the overset grid, where the shock crosses, and in the overlap region of the major grid. The convergence history, given in figure 5.5, shows that the solution did not converge after 1000 iterations, or $t = 400$. After 500 iterations, when the shock reached its steady-state position, a 40-iteration cycle of the residual operator began. The shock bounced between several grid points and never settled. The oscillations around the shock, which are common in central-differenced schemes, result in very large peaks and islands in the Mach and pressure-contour plots of figures 5.6 through 5.9. For comparison, the convergence history and contour plots for a solution obtained on a single grid is shown in figures 5.10 through 5.12. The convergence behavior is nearly linear, and the oscillations near the shock are gone. This first test verifies that the adverse effects found by Benek et al. (ref. 2) can be repeated by using a direct-interpolation boundary scheme.

The second grid tested had an overset grid with the upstream boundary a full radius away from the steady-state shock. The overlap between the overset grid boundary and hole was increased to eight points, as is shown in figure 5.13, thus moving the hole boundary away from the steady shock position. This test was also run for 1000 iterations and the convergence history in figure 5.14 shows that it is convergent. Although the convergence rate is only about half that of the single grid rate, the results appear to be better than Benek's test suggested. The solution also demonstrates an improvement in the shock region (figs. 5.15 through 5.18). The shock is still smeared as it crosses the overset grid boundary, but the peaks and islands have diminished. The major grid also has a better qualitative solution.

A third grid was tested to study the problem of a shock moving tangentially across an overset grid boundary. This problem arises when the shock wave propagates beyond the upstream boundary of the overset grid as it moves to its steady-state position. A small overset grid which was contained entirely within the subsonic flow was used. The shock crosses the hole boundary of the major grid successfully, but causes nonphysical pressures as it crosses the overset grid boundary. These tests point out the importance of shock location with respect to the grids.

5.3.2 Frozen boundaries- A natural followup test is to determine whether the solutions on the two grids are compatible. It is possible that the unsteady boundary data will never become steady without adding damping to the interpolation. In other words, the oscillations in the convergence history of the previous section may result from a forcing function derived from the interpolation from the other grid. The grid of figure 5.3 was used. The solution was run for 500 iterations, or $t = 200$, allowing enough time for the shock to reach its steady-state position, and then the boundary values were frozen. The solutions continued for another 500 iterations, and the residual operator was monitored. During these 500 iterations, the major-grid residual operator dropped to machine accuracy within the first 200 iterations. The minor-grid residual operator dropped about two and a half orders. The

residual plots are shown in figure 5.19. Note, however, that the minor grid has a sawtooth shape in its residual plot, which is not uncommon for single-grid problems. Solutions for this run are indistinguishable from the continuous interpolation (see figs. 5.20 through 5.23). The convergence suggests that the two grid solutions are not incompatible but are inconsistent in their time relaxations.

5.3.3 Boundary data averaging- Additional overset boundary schemes were attempted which showed no noticeable improvement over direct interpolation. One scheme averaged the interpolated data with data obtained by a prediction based on the grid requiring the boundary data. The predicted value was obtained using the second-order approximation

$$(q^{n+1}_{j_{max}} - q^{n}_{j_{max}}) = (q^{n+1}_{j_{max}-1} - q^{n}_{j_{max}-1}) \qquad (5.10)$$

The hole boundary incorporated a similar prediction using its nearest, unflagged neighbor. This approximation is built into the implicit, block tridiagonal solver, which helps the overall convergence rate. The benefit results from a predictor/corrector-like approach in which the predictor is implicit and therefore is not restricted to explicit stability bounds.

The purpose of the boundary-averaging scheme was to add damping to the time-dependent system and to remove the 40-iteration cycle experienced in the previous methods. The result was to decrease the cycle time but not affect the convergence (fig. 5.24). The solution was degraded with peak overshoots that exceeded those of direct interpolation (figs. 5.25 through 5.28). The tests were run on the grid system of figure 5.3, which was the most restrictive case.

The direct interpolation scheme presented in the previous section uses the predicted boundary technique for its stability characteristics, but the predicted values are overwritten by the interpolated data.

5.3.4 No-hole schemes- A series of steps were taken to isolate boundaries and study them independently. The two grids were decoupled so that the transmission of data from one grid to the other went in only one direction. This was accomplished by removing the hole region of the major grid. All boundary data for the major grid was then supplied from numerical boundary schemes that were independent of the overset, or minor, grid solution. Thus, the transmission of data followed a path from the major grid to the minor grid, but not in the other direction. Several numerical boundary schemes were attempted with the hope that if a favorable scheme was found for the overset grid boundary in this decoupled mode, it should transfer directly to a favorable scheme when the hole is reintroduced.

The numerical boundary schemes tested on the overset grid boundary included direct interpolation; averaging; a delta interpolation, in which Δq^n becomes the interpolated variable; and a fixed upstream boundary with delta interpolation on the outflow boundary. All of these schemes were unstable. This result is somewhat surprising because, in principle, the boundary data are supplied from an accurate

solution (from the major grid). One would expect that the interpolated boundary data would be consistent with the overset grid solution, and the scheme would be stable. However, Dirichlet data applied on the outflow leads to overspecification. The overspecification can be understood by considering the method of characteristics on the outflow boundary. This method will be described in the following section. When the grids are decoupled, the solution is unstable, implying inconsistent boundary data, but when coupled the solution is stable. This situation demonstrates the complexity of passing boundary data among multiple grids.

5.3.5 Characteristic boundaries- Physics dictates how information travels in a fluid and these considerations should be incorporated into a numerical scheme. The method of characteristics uses information propagation to determine the solution for hyperbolic equations.

A coupled system of equations, such as the one-dimensional Euler equations, can be written, in vector form, as

$$Q_t + E_x = 0 \tag{5.11}$$

If we introduce the matrix A, where $A = (\partial E/\partial Q)$, or

$$A = \begin{bmatrix} 0 & 1 & 0 \\ (\gamma - 3)(u^2/2) & -(\gamma - 3)u & \gamma - 1 \\ (\gamma - 1)u^3 - (\gamma eu/\rho) & (\gamma e/\rho) - [3(\gamma - 1)u^2/2] & \gamma u \end{bmatrix} \tag{5.12}$$

with c, the local speed of sound, and then write the equation in nonconservative form, for illustration, we have

$$Q_t + AQ_x = 0 \tag{5.13}$$

The matrix A has a complete set of eigenvalues and eigenvectors and therefore can be diagonalized

$$\Lambda = T^{-1}At \tag{5.14}$$

where

$$\Lambda = \begin{bmatrix} u & 0 & 0 \\ 0 & u + c & 0 \\ 0 & 0 & u - c \end{bmatrix} \tag{5.15}$$

and

$$T = \begin{bmatrix} 1 & (\rho/\sqrt{2}c) \\ u & (\rho/\sqrt{2}c)(u + c) \\ u^2/over2 & (\rho/\sqrt{2}c)\{(u^2/2) + uc + [c^2/(\gamma - 1)]\} \end{bmatrix}$$

$$\begin{bmatrix} (\rho/\sqrt{2}c) \\ (\rho/\sqrt{2}c)(u - c) \\ (\rho/\sqrt{2}c)\{(u^2/2) - uc + [c^2/(\gamma - 1)]\} \end{bmatrix} \qquad (5.16a)$$

$$T^{-1} = \begin{bmatrix} 1 - (\gamma - 1)(u^2/2c^2) & (\gamma - 1)(u/c^2) & -(\gamma - 1)(1/c^2) \\ (1/\sqrt{2}\rho c)[(\gamma - 1)(u^2/2) - uc] & (1/\sqrt{2}\rho c)[c - (\gamma - 1)u] & (1/\sqrt{2}\rho c)(\gamma - 1) \\ (1/\sqrt{2}\rho c)[(\gamma - 1)(u^2/2) + uc] & -(1/\sqrt{2}\rho c)[c + (\gamma - 1)u] & (1/\sqrt{2}\rho c)(\gamma - 1) \end{bmatrix}$$

(5.16b)

If the matrix T is frozen, then equation 5.13 can be transformed using this diagonalization to

$$T^{-1}Q_t + T^{-1}ATT^{-1}Q_x = W_t + \Lambda W_x = 0 \qquad (5.17)$$

so

$$(w_i)_t + \lambda_i(w_i)_x = 0 \qquad (5.18)$$

This new set of decoupled equations can now be discussed in terms of their characteristics or eigenvalues.

Suppose the flow is supersonic; then the three eigenvalues, u, $u + c$, and $u - c$ are all positive. Since the equations are hyperbolic, a one-dimensional problem will have Dirichlet data imposed upstream for all three equations. The outflow boundary will have no data imposed, and so a numerical boundary scheme must be used. This numerical boundary scheme should mimic the physical flow of information. Therefore, it should use data which are propagated from the interior and are used to specify three variables at the outflow boundary. Figure 5.29 illustrates this flow of information. If the flow is subsonic, then the eigenvalues u and $u + c$ are positive, but the eigenvalue $u - c$ is negative. Figure 5.30 shows the flow of information in this case. Two conditions are thus given on an inflow boundary, and one on an outflow boundary for subsonic flow. The remaining conditions

must be obtained from a numerical boundary scheme as before. If more conditions are specified than the physics of the problem suggests, the problem is overspecified and inconsistencies can arise which will cause the scheme to be unstable. A detailed discussion of characteristic boundaries can be found in many references, such as Yee (ref. 26) or Pulliam (ref. 27).

The method of characteristics implies the use of certain boundary conditions for the overset grid boundaries. In the cases tested, the upstream boundary of the overset grid is in the purely supersonic region of the flow. From the characteristic point of view, all data are specified by the physics. Therefore, interpolated data from the major grid on the upstream boundary are consistent with the characteristics. The outflow boundary must be treated differently. For an outflow boundary, Dirichlet data from the interpolation lead to overspecification. A consistent numerical boundary scheme must be used in the outflow region. For example, for subsonic outflow only one characteristic variable may come from the interpolation.

Implementation of characteristic boundaries is simple if certain approximations are allowed. The characteristic implementation discussed here follows the approach of Moretti (ref. 28), and is nonconservative, which can cause errors in the results. However, interpolation is also nonconservative, so the boundary scheme is already nonconservative. In the scheme used here, the Euler equations are written in nonconservative form. The equations are decoupled into normal and tangential coordinates in such a way as to yield six equations for the Riemann variables, two of which are redundant. These equations are derived in a manner similar to the diagonalization procedure presented earlier and are of the form

$$R_{it} + \Lambda_i R_{in} = 0 \tag{5.19}$$

The spatial direction n corresponds to one of the generalized coordinate directions. The Riemann variables and the λ's are given by

$$R = \left\{ \frac{2c}{\gamma - 1} - U,\ \frac{2c}{\gamma - 1} + U,\ V,\ \frac{2c}{\gamma - 1} - V,\ \frac{2c}{\gamma - 1} + V,\ U \right\}^t \tag{5.20a}$$

and

$$\Lambda = \{U - c,\ U + c,\ U,\ V - c,\ V + c,\ V\}^t \tag{5.20b}$$

The redundant equations are dropped. The use of the Riemann variables in the boundary scheme depends upon the characteristic directions. If the lambda value, λ_i, is positive on an outflow boundary, then the corresponding Riemann variable is calculated from interior data by the predictor method described earlier (eq. (5.10)). If λ_i is negative, then the corresponding Riemann variable is calculated from exterior data (i.e., interpolated data for the overset grid, or free stream for the major grid). This portion of the study will focus only on the characteristic boundaries applied to the overset-grid outflow boundary.

In the implementation for this portion of the study, two complete sets of boundary data are first calculated. The first set results from interpolation. The second set results from the implicit predictor scheme of equation (5.10). The outflow boundary is then tested to find the local Mach number. If the point is supersonic, implying supersonic outflow, then the boundary data are overwritten with the values found from the interior in the predictor scheme. This is a result of having all positive λ_i's. If the point is subsonic, then the two sets of boundary data are converted into Riemann variables to obtain the correct data. The value of R_1, where in this implementation $R_1 = U + [2c/(\gamma - 1)]$, is calculated from the predicted data of the minor grid. Two other variables are calculated from the predicted data; they are tangential velocity, or V, and the entropy. The final variable is calculated using the interpolated data. This corresponds to R_2, where $R_2 = U - [2c/(\gamma - 1)]$. These new variables are now combined to form q, where $q = q(R_1,R_2,V,s)$ and s is the entropy, which overwrites the boundary data for the outflow boundary. A similar approach can be used on the upstream boundary, but as mentioned previously, this results in overwriting the boundary entirely with the interpolated data.

In the first attempt, the local Mach number was calculated from the major grid, since it was decided that the major grid solution was more reliable than the predicted solution from the minor grid. This method was unstable, which implies that the Mach number used was not consistent with the minor grid solution. It was found that, in the neighborhood of the sonic line, the two slightly different solutions predicted supersonic outflow and subsonic outflow, so at one point the two grids suggested the use of different conditions at the same boundary. This difference at one point eventually led to growing errors and eventually rendered the scheme unstable. It was concluded that the signs of the eigenvalues, for the minor-grid outflow, must be calculated using data from the interior of the minor grid and not from the major grid.

When the Mach number calculation was modified so that the signs of the eigenvalues agreed with the minor grid solution, the scheme was stable. Convergence was linear (fig. 5.31). The solution was also qualitatively better than the direct interpolation procedure on both grids. The major grid, in fact, is identical to a single grid solution, and the minor grid shows none of the poor shock characteristics that were found in the previous solutions (figs. 5.32 and 5.33).

In summary, when the two-grid system was decoupled in such a way that information traveled from the major grid to the minor grid, only one boundary scheme for the overset grid was stable. A scheme using characteristic boundaries, with the eigenvalues predicted from the overset grid itself, resulted in linear convergence and a qualitatively "good" solution. Schemes such as direct interpolation, averaging, delta interpolation, and characteristics with the eigenvalues calculated from the major grid were all unstable.

5.3.6 Characteristics on coupled grids- With the success of the characteristic approach of the preceding section, it appears that equal success would be achieved with the coupled-grid system. The first attempt used direct interpolation to update the hole boundary. Three grids were tested with this approach. The first grid was

identical to the second grid tested for the direct interpolation of section 5.3.1 (fig. 5.13), where the upstream overset-grid boundary and hole boundary region are removed some distance from the shock region. The convergence history, shown in figure 5.34, shows that the solution is convergent with a convergence rate that is not much different from the single-grid rate. This promising result demonstrates the effectiveness of this improved boundary scheme. The L_2 norm of the error after 1000 iterations is approximately one order of magnitude less than the direct interpolation for the major grid. Figure 5.35 compares these two convergence histories. Despite the better overall convergence rate, however, it is apparent that the convergence rate is not linear. It appears that the peaks and valleys in figure 5.34 are due to shifts of points from subsonic to supersonic, and vice versa, on the minor-grid outflow boundary. A comparison of the minor-grid convergence histories is shown in figure 5.36.

The solution on the grid of figure 5.13 has the nice features demonstrated with the uncoupled grids of section 5.3.5. The minor grid has a smooth shock crossing the outflow boundary, and the major grid has none of the kinks that have been found in all the other cases. Figures 5.37 through 5.40 are the corresponding Mach number and pressure contours for this run.

The second grid tested used the overset grid of figure 5.4, in which the upstream boundary was moved to within one half of a radius of the shock position. However, in this run the hole boundary allowed for an overlap of eight grid points between the overset grid boundary and the hole boundary. Thus, the hole boundary was kept away from the shock to avoid interference. The convergence history for this run (fig. 5.41) shows that moving the upstream boundary of the overset grid closer to the shock had a negative effect on the convergence rate. The third and last grid tested used the same overset grid, but allowed for only four overlapping points in determining the hole region. This put the hole boundary in the steady-state shock region. The convergence history for this run (fig. 5.42) demonstrates the same convergence history experienced by direct interpolation on the same grid. Thus, despite the improved convergence caused by the characteristic approach for the overset boundary scheme, the location of the overset grid boundary and hole boundary is important in determining the overall convergence of the scheme.

These tests have demonstrated favorable properties of the characteristic boundary approach. However, they have also shown that if either the hole boundary or the overset-grid outer boundary are positioned where they can interfere with the shock, the convergence will be adversely affected.

A scheme was tested using characteristic boundaries for the hole boundary in addition to the overset grid boundary. This scheme was unsuccessful as the shock tried to cross the hole boundary, because the data predicted for the boundary by the major grid always suggested pure supersonic flow. Therefore, there was no vehicle for communicating with the minor grid to allow the shock to propagate. This conceptual problem should be noted if characteristic boundary schemes are to be used in other overset grid applications.

5.4 Comparison with Other Data

Merely improving the convergence rate of a scheme does not make it useful. The solution should compare favorably with known correct results. The two schemes that were judged were the direct interpolation approach of section 5.3.1 and the characteristic approach of section 5.3.6, both using the grid system of figure 5.13. While interpreting the data, one should keep in mind that the grid used was fairly coarse. Therefore, with shock smearing, the shock appears much thicker than a physical shock. However, it is still possible to locate the approximate shock position in order to get an estimate of the quality of the solution.

The direct interpolation test is compared with the results of Rai (ref. 3) and Lyubimov and Rusanov (ref. 29) in figure 5.43. These results show that the shock location is in better agreement with that of Rai. This result is forward of the more accepted shock position predicted by Lyubimov and Rusanov. Rai used a first-order accurate scheme and suggested that this was the cause for his erroneous shock location. The nonconservative interpolation at the overset grid boundary is the most likely culprit in the example presented here.

The characteristic boundary approach shows more favorable agreement with Lyubimov and Rusanov's predictions. Figure 5.44 demonstrates that the characteristic approach positions the shock in the immediate neighborhood of the accepted shock position and downstream of Rai's shock position. It appears that in addition to the advantages of the characteristic boundary scheme described in section 5.3.6, this approach gives solutions which are in agreement with accepted results.

A comparison of the results of direct interpolation in figure 5.43 with the results using the characteristic approach in figure 5.44 shows a significant movement in the shock location. Although this result suggests that the characteristic approach is superior, it points to a strong sensitivity of the results on the overset-boundary scheme.

5.5 MIMD Notes

One major aim of the Centaur code was to study the numerical problems of overset grids. The code was implemented on the NASA Ames MIMD test facility to also study concurrent processing. The procedure used is similar to the previous study of the airfoil/flap problem so the implementation will only be reviewed here.

The primary difference between Centaur and Chimera is that the Chimera grid package is resident in both tasks so that the tasks are completely decoupled except for the overset-grid boundary data and the synchronizations. In other words, whatever constants are required by both tasks are duplicated in the task-global data blocks. With this minimal amount of shared data, the two processors may conceivably be remote nodes in a major computer network. Since the overhead of the duplication is small, it makes little difference in the overall time required to run 1000 iterations. An additional feature of this implementation is that the two tasks could be

implemented using an identical source. This feature has the advantage of requiring only one program image to manage.

The basic implementation of the code is as follows. The same code is submitted for execution on both machines. The first operation each machine performs is to find out which task it is--the major or minor grid task. This is accomplished by a first-come/first-serve algorithm. The first processor to claim a task gets the major grid task. The other processor finds the task-flag set and, therefore, will choose the minor grid task. This simple technique can be applied to any number of processors and tasks and does not require any special startup procedure. Once the tasks have been chosen, initialization is duplicated by both tasks. A Sync point occurs at the beginning of the first iteration loop. This is the first synchronization in the execution. At this point, each task solves its own independent grid, and each outputs to its own output files, independent of the other task. All of the results presented previously used a synchronous solution procedure in which the tasks were synchronized before the beginning of each iteration.

An asynchronous implementation of Centaur is somewhat more complex than the stream-function program of the preceding section. Centaur is a nonlinear, nonsteady code which allows a shock to propagate away from the wall before it reaches its final, steady location. This code is more complex because the coefficient matrix, when expressed in the fixed-operator notation of section 4.4, is nonlinear and may develop into a noncontracting matrix. This problem uses balanced grids to avoid wide deviation in iteration levels, which would cause the coefficient matrix to cease to be a contracting matrix. Another factor which determines the contracting properties of the coefficient matrix is the time step. The time step in a time-accurate scheme should be lowered whenever chaotic relaxation is employed. An asynchronous solution procedure does not fall under the general category of asynchronous iterative methods described in section 4.4, but can be classified with the more specific chaotic relaxation methods. A direct-interpolation approach was tested asynchronously, with no apparent difference in the solution to the synchronous results. However, at this time there is still too little understanding of overset grid implementations in unsteady, transonic flow to justify studying chaotic techniques. As more becomes known about the asynchronous solutions, the chaotic solutions will warrant further study.

5.6 Conclusions

The Centaur code was introduced as a test problem to investigate some of the properties of overset grids in transonic flow. The Chimera grid scheme was used and the Euler equations were solved numerically. The study focused on the information exchange between the major and minor overset grids. The primary goal was to understand some of the factors that contributed to the poor solution obtained by Benek et al. (ref. 2).

A series of overset grid boundary schemes were investigated. The first test was identical to the method used by Benek et al. (ref. 2). When the hole region of

the major grid was removed, the only convergent boundary scheme was the characteristic scheme. This result showed linear convergence and removed the smearing where the shock crossed the boundary. It was discovered that the scheme is affected by the manner in which signs of the eigenvalues are chosen. When this scheme was applied to the coupled grid system with the hole being updated by direct interpolation, the results agreed with the accepted results of Lyubimov and Rusanov. The results also showed that the shock position is sensitive to the overset-boundary scheme chosen and that the hole boundary and overset grid boundary can destroy the convergence rate if it is allowed to interfere with the shock.

Current research at NASA Ames is focusing on conservative interpolation schemes. Suggested approaches have been to interpolate the conservative variables, apply a flux balance equation on the boundary, or apply a shock correction procedure. It is hoped that the results obtained in this study will be of some use to large-scale, multiple-geometry problems.

6.0 CONCLUDING REMARKS

Parallelism in computer architectures is increasing with each new generation of computers. Multiple-processor architectures will probably be the primary innovation in the next decade (ref. 8). The goal of this study was to gain knowledge in the application of state-of-the-art algorithms on multiple-processor computers.

A method for implementing approximate factorization algorithms onto MIMD computers has been suggested. This method followed the approach that is used on vector machines. The decoupled operators of each spatial sweep were solved concurrently on the multiple processors. This technique, applied to a well-known CFD code, AIR3D, on the NASA Ames MIMD test facility, showed a optimal speedup of 1.905 for two processors. This study has shown the feasibility and the benefit of using multiple processors for solving approximate factored algorithms and has also introduced ideas for creating algorithms better suited for concurrent processing.

The study of AIR3D and reviewing of certain current CFD research trends led to the study of multiple grid problems. Multiple grid solution procedures lend themselves nicely to concurrent processing since they consist of loosely coupled tasks. The airfoil/flap problem of Steger, Dougherty, and Benek was studied on the Ames MIMD test facility, and their results were reproduced. An implementation using asynchronous iterations was also tested, and is the first known trial of this chaotic procedure on a CFD application. Asynchronous iterations with minimal data communication can be an advantage when implemented on a MIMD facility with a small shared memory.

The overset grid scheme had some poor qualities, which were improved in this study. The Chimera grid scheme was applied to a blunt-body problem, in a code called Centaur, which served as a convenient test vehicle. The poor qualities that have been attributed to overset grids are a poor convergence rate, and a smeared shock wave where the shock crosses a grid boundary. These problems have been blamed

on the handling of the overset grid boundaries. A successful boundary scheme was implemented that uses characteristics to allow proper information transfer. The interaction between the shock wave and the overset grid boundaries demonstrates the need to move boundaries that run parallel to the shock as far from the shock as possible.

This study represents only a small selection of codes that can be implemented on MIMD machines, but the basic knowledge that has been developed may be used to design algorithms in the future. Concurrent processing represents the future in large-scale scientific computing, so the achievement of an understanding of machine-algorithm interactions is necessary.

REFERENCES

1. Steger, J. L.; Dougherty, F. C.; and Benek, J. A.: A Chimera Grid Scheme. Advances in Grid Generation. FED, vol. 5, the American Society of Mechanical Engineers, New York, 1983.

2. Benek, J. A.; Steger, J. L.; and Dougherty, F. C.: A Flexible Grid Embedding Technique with Application to the Euler Equations. Proceedings of the 6th AIAA Computational Fluid Dynamics Conference, Danvers, MA, 1983.

3. Rai, M. M.: A Conservative Treatment of Zonal Boundaries for Euler Equation Calculations. AIAA Paper 84-0164, Reno, NV, 1984.

4. Hessenius, K. A.; and Pulliam, T. H.: A Zonal Approach to Solution of the Euler Equations. AIAA Paper 82-0969, St. Louis, MO, 1982.

5. Atta, E. H.; and Vadyak, J.: A Grid Interfacing Zonal Algorithm for Three Dimensional Transonic Flows About Aircraft Configurations. AIAA/ASME 3rd Joint Thermophysics, Fluids, Plasma and Heat Transfer Conference, St. Louis, MO, 1982.

6. Dwyer, H. A.: A Discussion of Some Criteria for the Use of Adaptive Gridding. Proceedings of the 6th AIAA Computational Fluid Dynamics Conference, Danvers, MA, 1983.

7. Lomax, H.; and Pulliam, T. H.: A Fully Implicit Factored Code for Computing Three Dimensional Flows on the ILLIAC IV. Parallel Computations, G. Rodrigue, ed., Academic Press, New York, 1982.

8. Hockney, R. W.; and Jesshop, C. R.: Parallel Computers. Adam Hilger Ltd., Bristol, 1983.

9. Flynn, M. J.: Very High-Speed Computing Systems. Proceedings of the IEEE, vol. 54, no. 12, Dec. 1966.

10. Lambiotte, J. J.; and Voigt, R. G.: The Solution of Tridiagonal Linear Systems on the CDC STAR-100 Computer. ACM Transactions on Mathematical Software, vol. 1, no. 4, Dec. 1975.

11. Stone, H. S.: Parallel Tridiagonal Equation Solvers. ACM Transactions on Mathematical Software, vol. 1, no. 4, Dec. 1975.

12. Barlow, R. H.; and Evans, D. J.: Parallel Algorithms for the Iterative Solution to Linear Systems. The Computer Journal, vol. 25, no. 1, 1982.

13. Chazan, D., and Miranker, W.: Chaotic Relaxation. Linear Algebra and Its Applications, vol. 2, 1969.

14. Baudet, G. M.: Asynchronous Iterative Methods for Multiprocessors. Journal of the Association for Computing Machinery, vol. 25, no. 2, Apr. 1978.

15. Pulliam, T. H.; and Steger, J. L.: Implicit Finite-Difference Simulations of Three Dimensional Compressible Flow. AIAA Journal, vol. 18, no. 2, Feb. 1980.

16. Holst, L. L.; and Thomas, S. D.: Numerical Solution of Transonic Wing Flowfields. AIAA Journal, vol. 21, no. 6, June 1983.

17. Rogallo, R. S.: Numerical Experiments in Homogeneous Turbulence. NASA TM-81315.

18. Steger, J. L.: Implicit Finite-Difference Simulation of Flow about Arbitrary Two-Dimensional Geometries. Journal of Computational Physics, vol. 16, 1978.

19. Beam, R. M.; and Warming, R. F.: An Implicit Factored Scheme for the Compressible Navier-Stokes Equations. AIAA Journal, vol. 16, no. 4, Apr. 1978.

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

21. Peaceman, D. W.; and Rachford, H. H.: The Numerical Solution of Parabolic and Elliptic Differential Equations. Journal of the Society of Industrial and Applied Mathematics, vol. 3, 1955.

22. Douglas, J.: On the Numerical Integration of $\partial^2 u/\partial x^2 + \partial^2 u/\partial y^2 = \partial u/\partial t$ by Implicit Methods. Journal of the Society of Industrial and Applied Mathematics, vol. 3, 1955.

23. Douglas, J.; and Rachford, H. H.: On the Numerical Solution of the Heat Conduction Problems in Two and Three Space Variables. Trans. Am. Math. Soc., vol. 82, 1956.

24. Douglas, J; and Gunn, J. E.: A General Formulation of Alternating Direction Method--Part I. Parabolic and Hyperbolic Problems. Numerische Mathematik, vol. 6, 1964.

25. Baudet, G. M.: The Design and Analysis of Algorithms for Asynchronous Multiprocessors. Ph.D. Thesis, Carnegie-Mellon University, April 1978.

26. Yee, H. C.: Numerical Approximation of Boundary Conditions with Applications to Inviscid Equations of Gas Dynbamics. NASA TM-81265, March 1981.

27. Pulliam, T. H.: Characteristic Boundary Conditions for the Euler Equations. Numerical Boundary Condition Procedures. NASA CP-2201, 1981.

28. Moretti, G.: A Physical Approach to the Numerical Treatment of Boundaries in Gas Dynamics. Numerical Boundary Condition Procedures. NASA CP-2201, 1981.

29. Lyubimov, A. N.; and Rusanov, V. V.: Gas Flows Past Blunt Bodies. NASA TT-F-715, 1973.

TABLE 1.- TASK TIMINGS--EULER EQUATIONS--PROGRAM BREAKDOWN

Task	Time-MIMD Code		Time-Serial Code	Speedup
	Serial	Concurrent		
Setup	6.58		6.03	0.916
RHS	.02	4.02	7.78	1.926
BC + Resid.	.74		.74	1.000
LHS	.09	13.64	26.76	1.949
Output	(1.64)			
(optional)		not measured	assume	1.000
Time/Iteration	(.85)	(17.66)	(35.26)	1.905
Output routines	3.24		3.29	1.015

$$\text{Speedup} = \frac{t_{setup} + n(t_{bc} + t_{rhs} + t_{lhs}) + t_{output}}{t'_{setup} + n(t'_{bc} + t'_{rhs} + t'_{lhs}) + t'_{output}}$$

n - iterations
t - serial
t' - concurrent

Examples

Iterations	Speedup	Iterations	Speedup
1	1.574	15	1.872
2	1.705	50	1.895
5	1.813	100	1.900
10	1.857	400	1.904

TABLE 2.- TASK TIMINGS--NAVIER-STOKES EQUATIONS--
PROGRAM BREAKDOWN

Task	Time-MIMD Code		Time-Serial Code	Speedup
	Serial	Concurrent		
Setup	6.57		6.07	0.924
RHS	.02	7.84	15.15	1.927
BC + Resid.	.84		.84	1.000
LHS	.08	15.47	30.50	1.972
Output	(1.64)			
(optional)		(not measured)	assume	1.000
Time/Iteration	(.94)	(23.31)	(46.42)	1.914
Output routines	3.24		3.22	.994

Examples			
Iterations	Speedup	Iterations	Speedup
1	1.635	50	1.906
2	1.751	100	1.910
5	1.842	400	1.913
10	1.876		
15	1.889		
25	1.899		

TABLE 3.- TOTAL CPU TIMINGS FOR THE EULER EQUATIONS
EULER CPU TIMES

Iterations	t_{MAIN}	t_{RHS}	t_{LHS}	t_{MIMD}	t_{Serial}	Speedup
1	11.60	4.21	13.67	29.48		1.537
1	11.79	4.23	13.74	29.76	45.32	1.523
2	14.25	8.41	27.36	50.02	82.70	1.653
5	16.10	20.98	67.95	105.03	187.36	1.784
10	22.60	42.04	132.38	197.02	365.60	1.856
15	28.03	63.03	204.58	295.64	544.49	1.842
25	33.45	104.79	338.04	476.28	897.78	1.885

TABLE 4.- TOTAL CPU TIMINGS FOR THE NAVIER-STOKES EQUATIONS
NAVIER-STOKES CPU TIMES

Iterations	t_{MAIN}	t_{RHS}	t_{LHS}	t_{MIMD}	t_{Serial}	Speedup
1	11.62	7.84	15.47	34.93	56.70	1.623
2	13.93	15.69	30.87	60.49	105.07	1.737
5	16.19	39.15	77.15	132.49	244.24	1.843
10	20.58	78.17	155.13	253.88	478.13	1.883
25	33.36	196.72	386.95	617.03	1173.25	1.901

TABLE 5.- STOPWATCH TIMINGS FOR THE EULER EQUATIONS--EULER CLOCK TIMES

Iterations	t_{MAIN}	t_{Serial}	Speedup
1	44.61		1.034
1	36.98	46.14	1.248
2	58.10	82.70	1.423
5	114.68	188.32	1.642
10	213.66	366.98	1.718
15	311.86	545.98	1.751
25	499.44	899.53	1.801

TABLE 6.- STOPWATCH TIMINGS FOR THE NAVIER-STOKES EQUATIONS--NAVIER-STOKES CLOCK TIMES

Iterations	t_{MAIN}	t_{Serial}	Speedup
1	43.73	58.39	1.335
2	68.58	106.75	1.557
5	151.50	246.11	1.642
10	269.69	479.39	1.778
25	647.78	1175.21	1.814

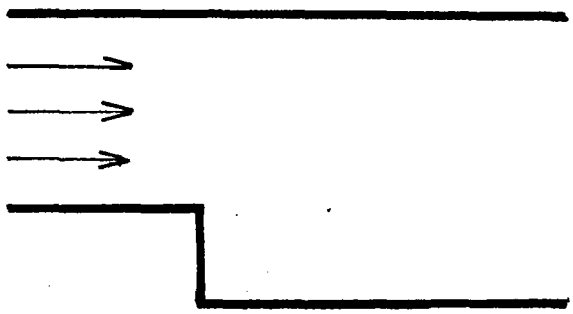

Figure 2.1a.- Channel with step.

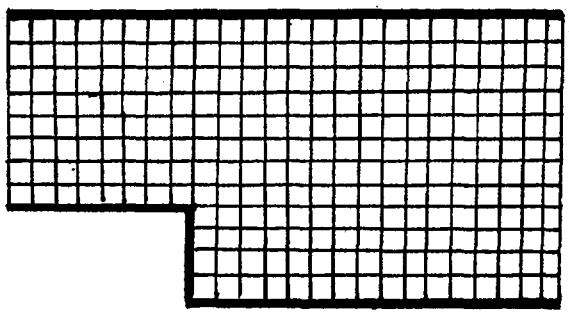

Figure 2.1b.- Non-rectangular grid in computational space.

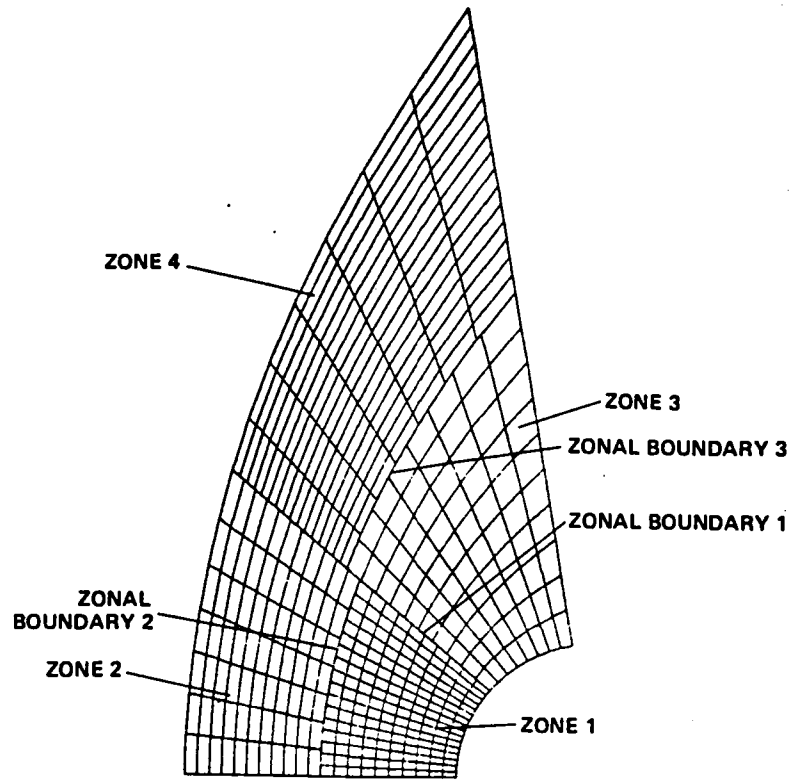

Figure 2.2- Zonal grid applied to blunt body.

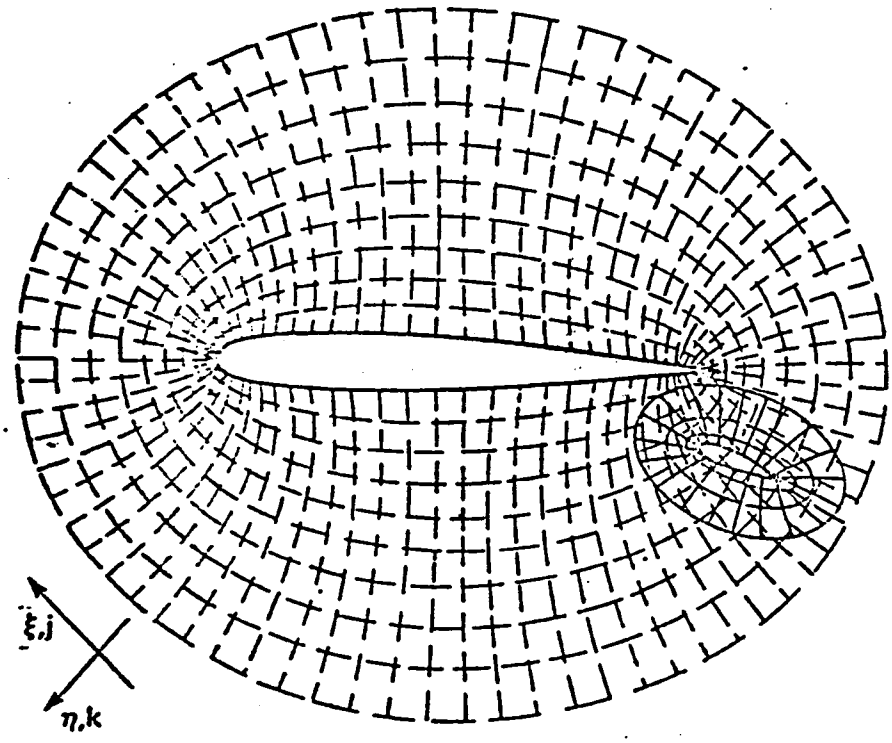

Figure 2.3.- Airfoil/flap geometry using overset grids.

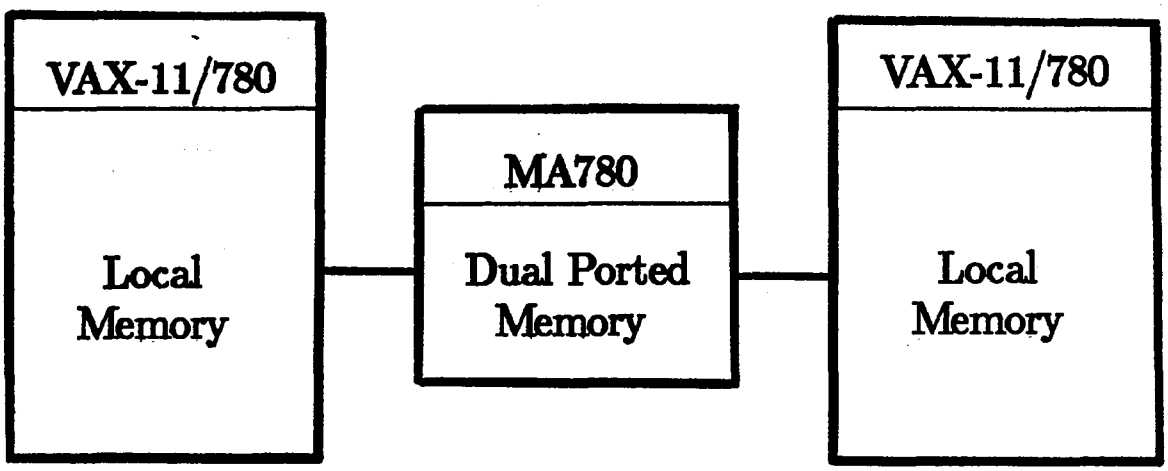

Figure 2.4a.- NASA Ames MIMD test facility.

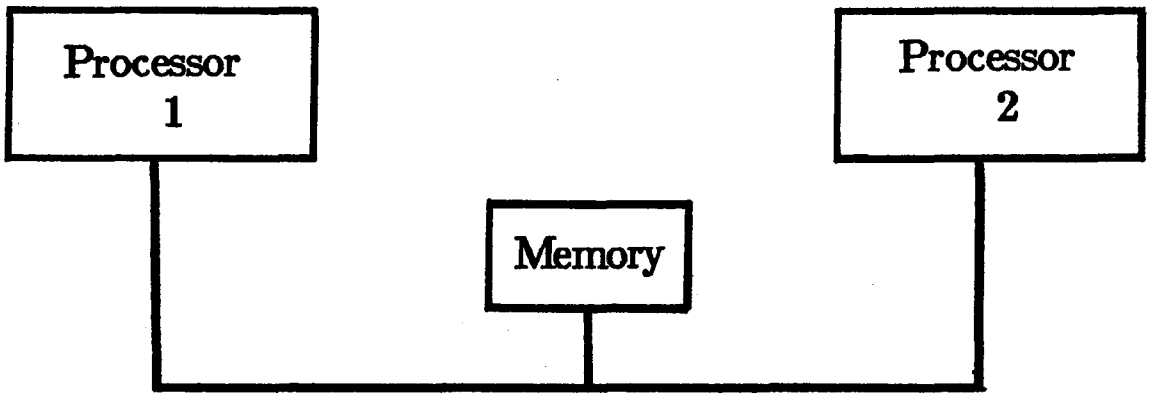

Figure 2.4b.- CRAY X-MP.

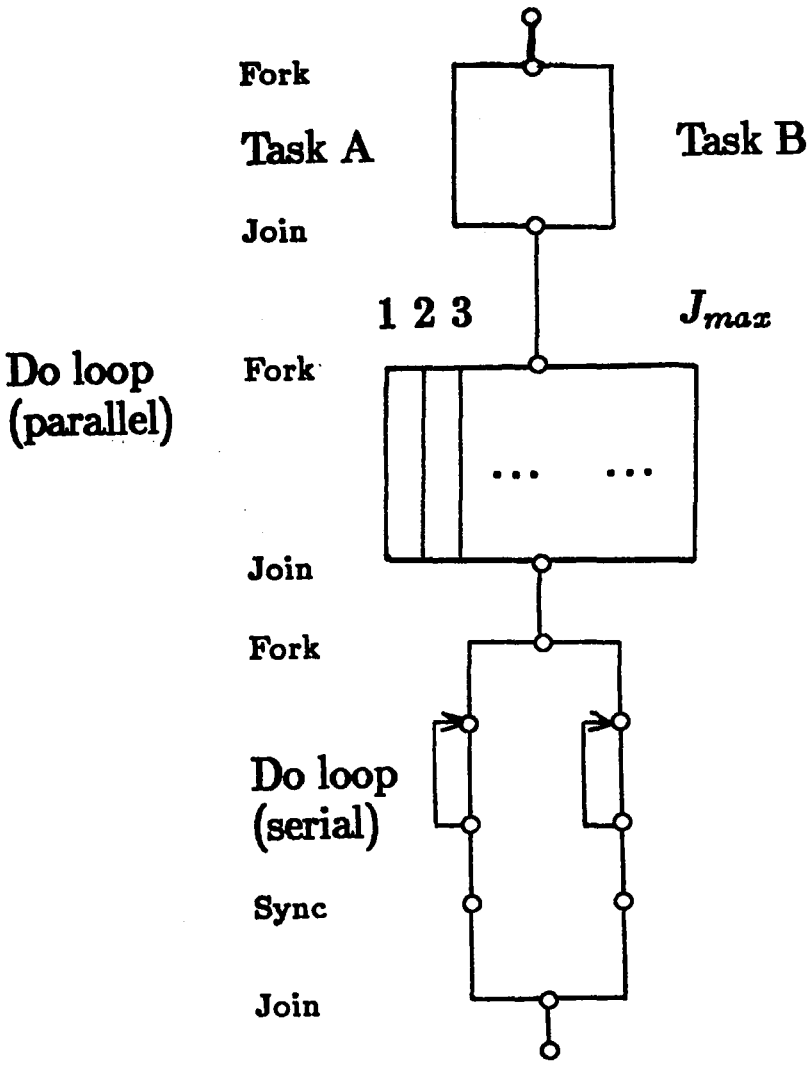

Figure 2.5.- Sample program flow chart.

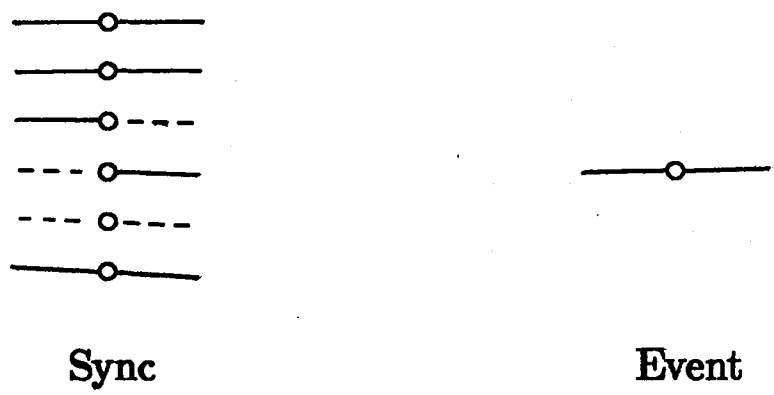

Figure 2.6.- Summary of MIMD operations.

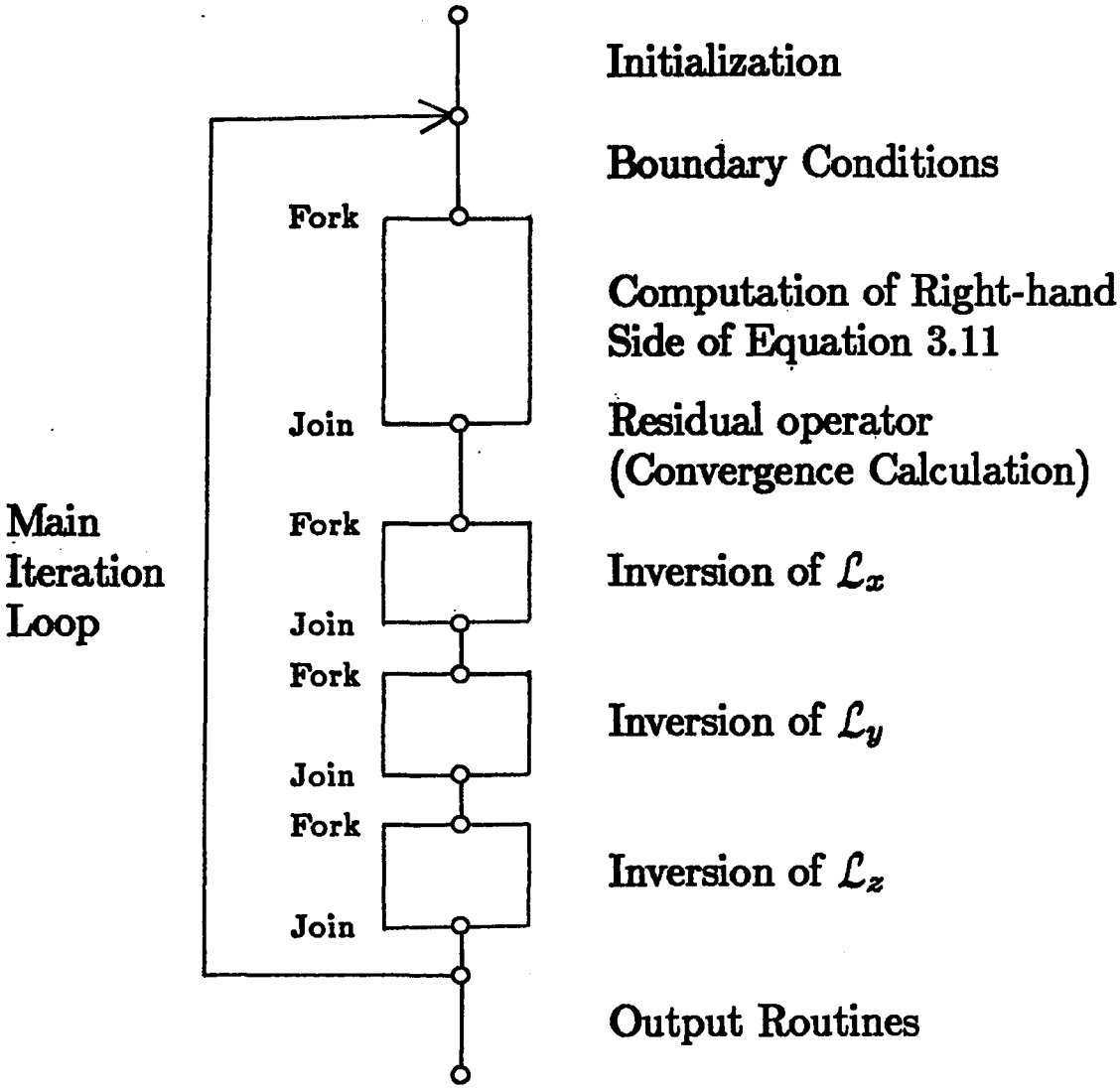

Figure 3.1.- Flow chart of AIR3D for ideal two-processor MIMD implementation.

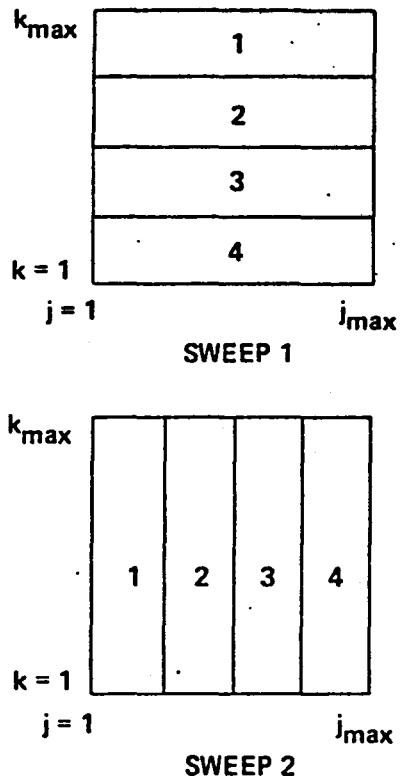

Figure 3.2.- Implementation of a two-dimensional problem on a four-processor system.

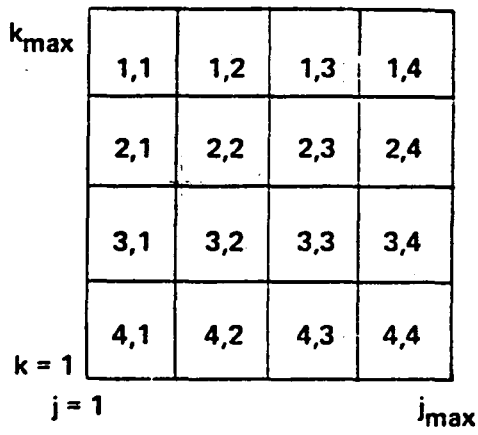

Figure 3.3.- Memory partition for a four-processor system set up for a two-dimensional problem.

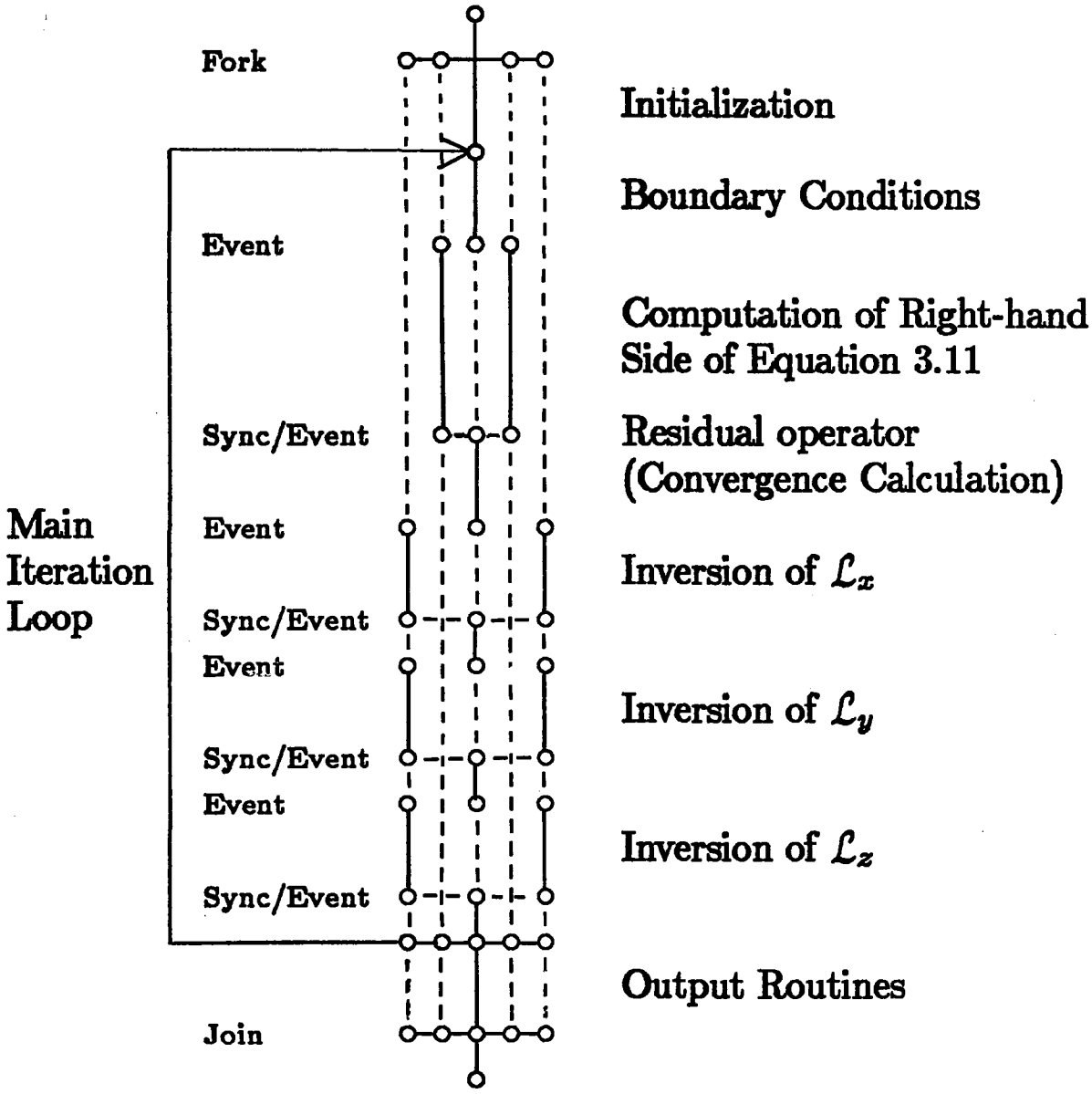

Figure 3.4.- Actual flow chart of AIR3D for this study.

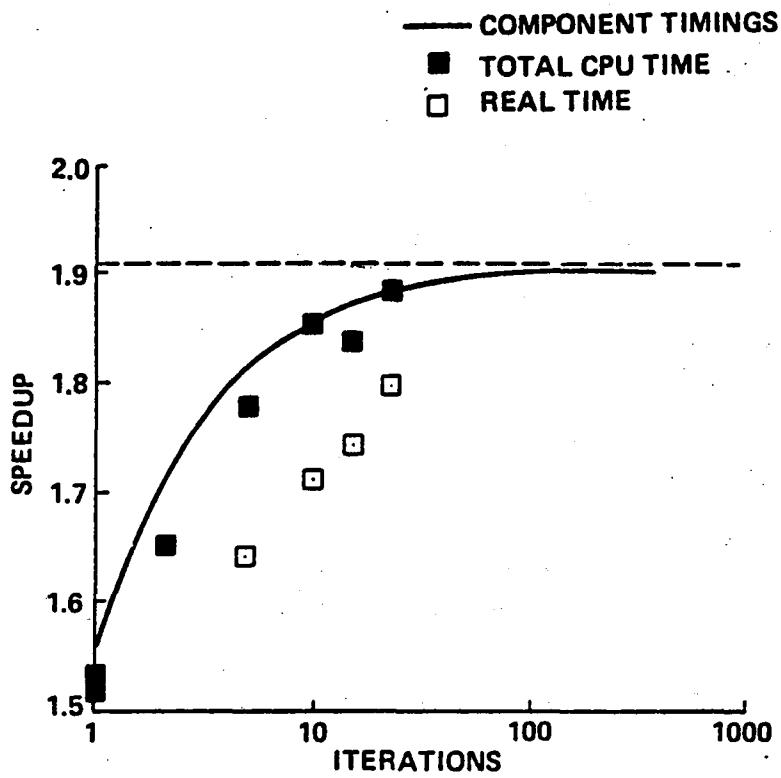

Figure 3.5.- Speedup of the Pulliam-Steger AIR3D code using two processors to solve the Euler equations.

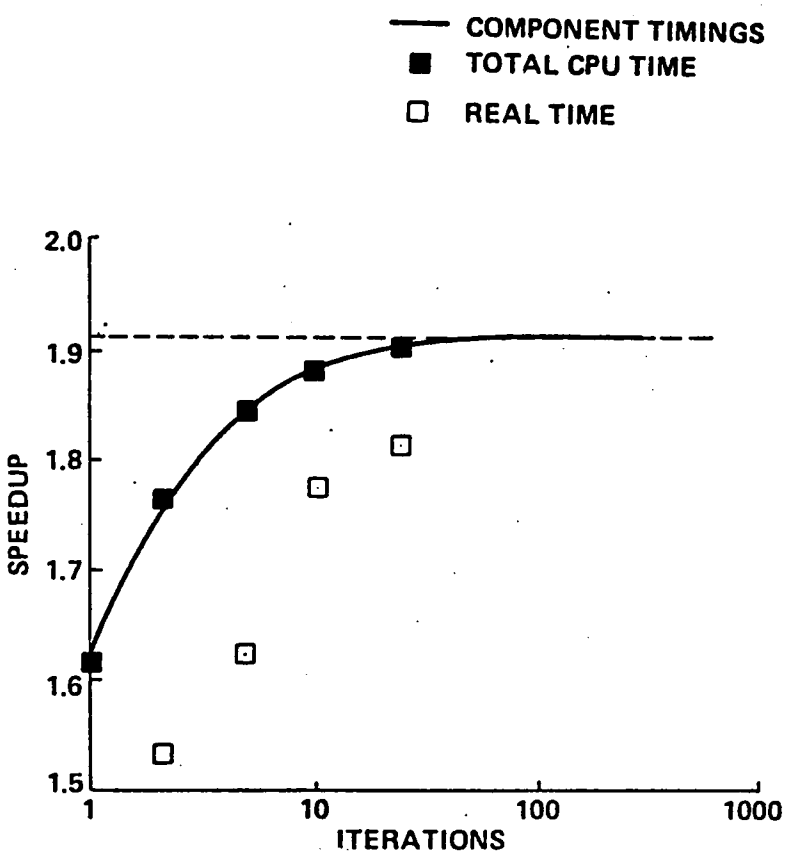

Figure 3.6.- Speedup of the Pulliam-Steger AIR3D code using two processors to solve the Navier-Stokes equations with a thin-layer approximation and an algebraic turbulence model.

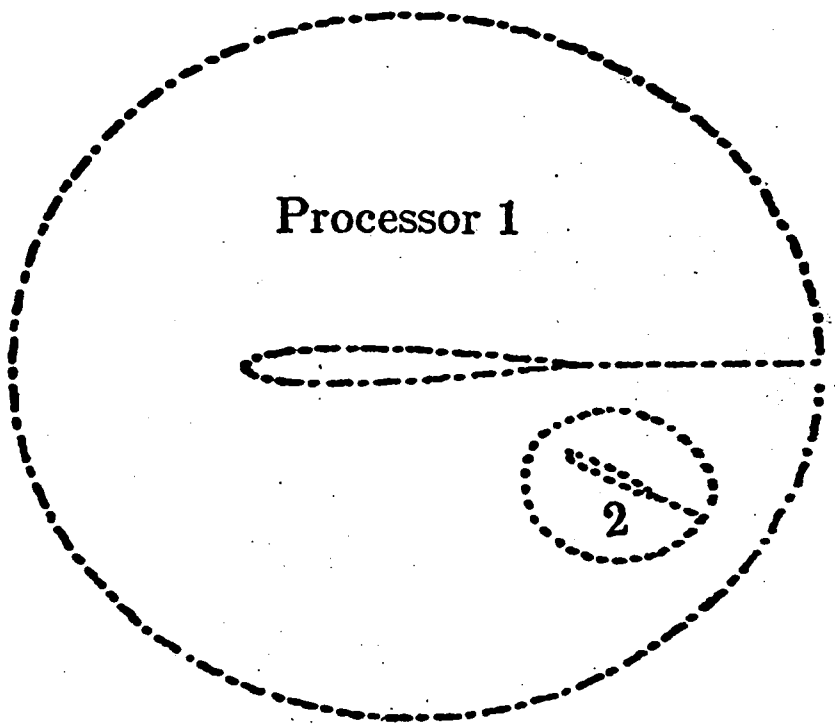

Figure 4.1.- Multiple grid application using multiple processors.

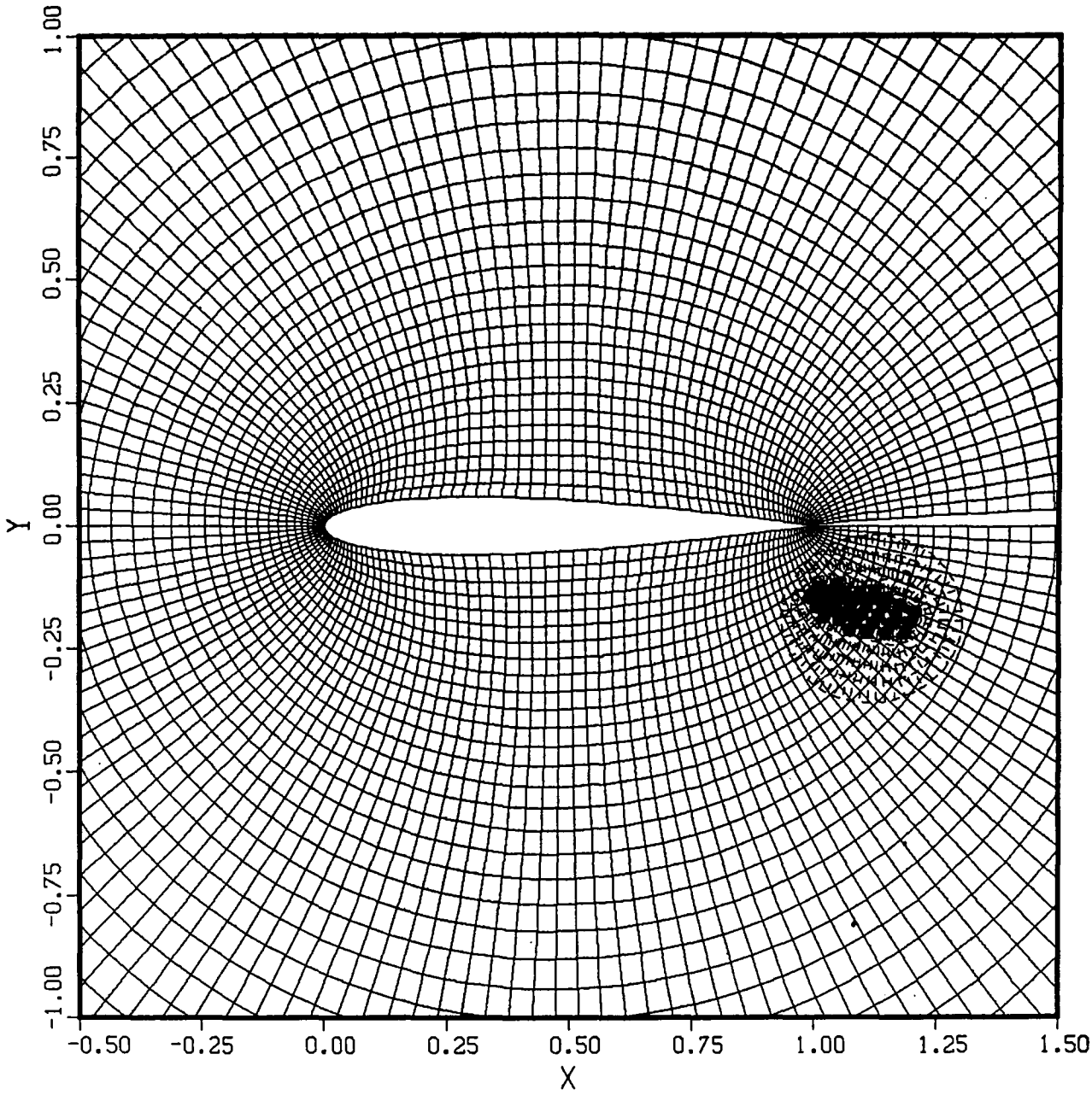

Figure 4.2.- Airfoil/flap configuration with overset grid (ref. 1).

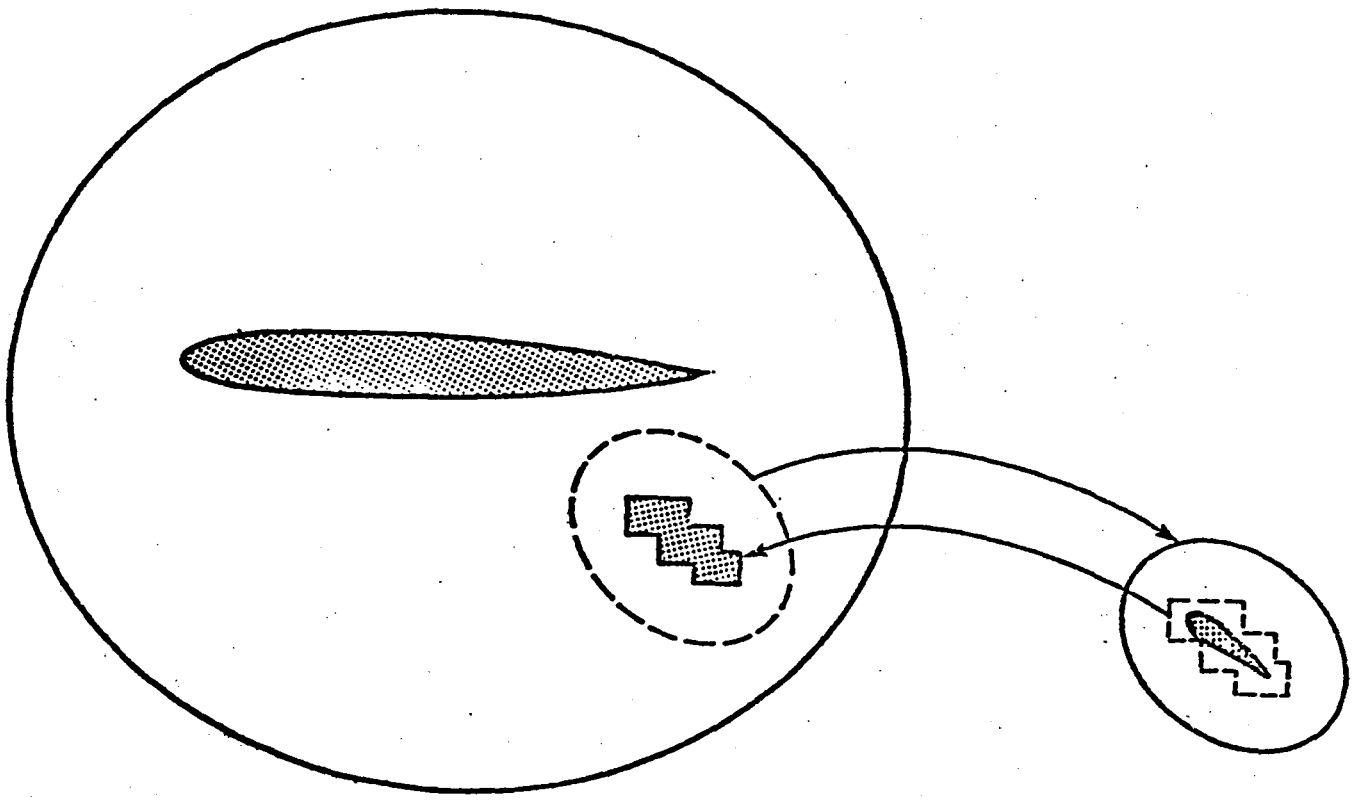

Figure 4.3.- Transfer of information between grids (ref. 1).

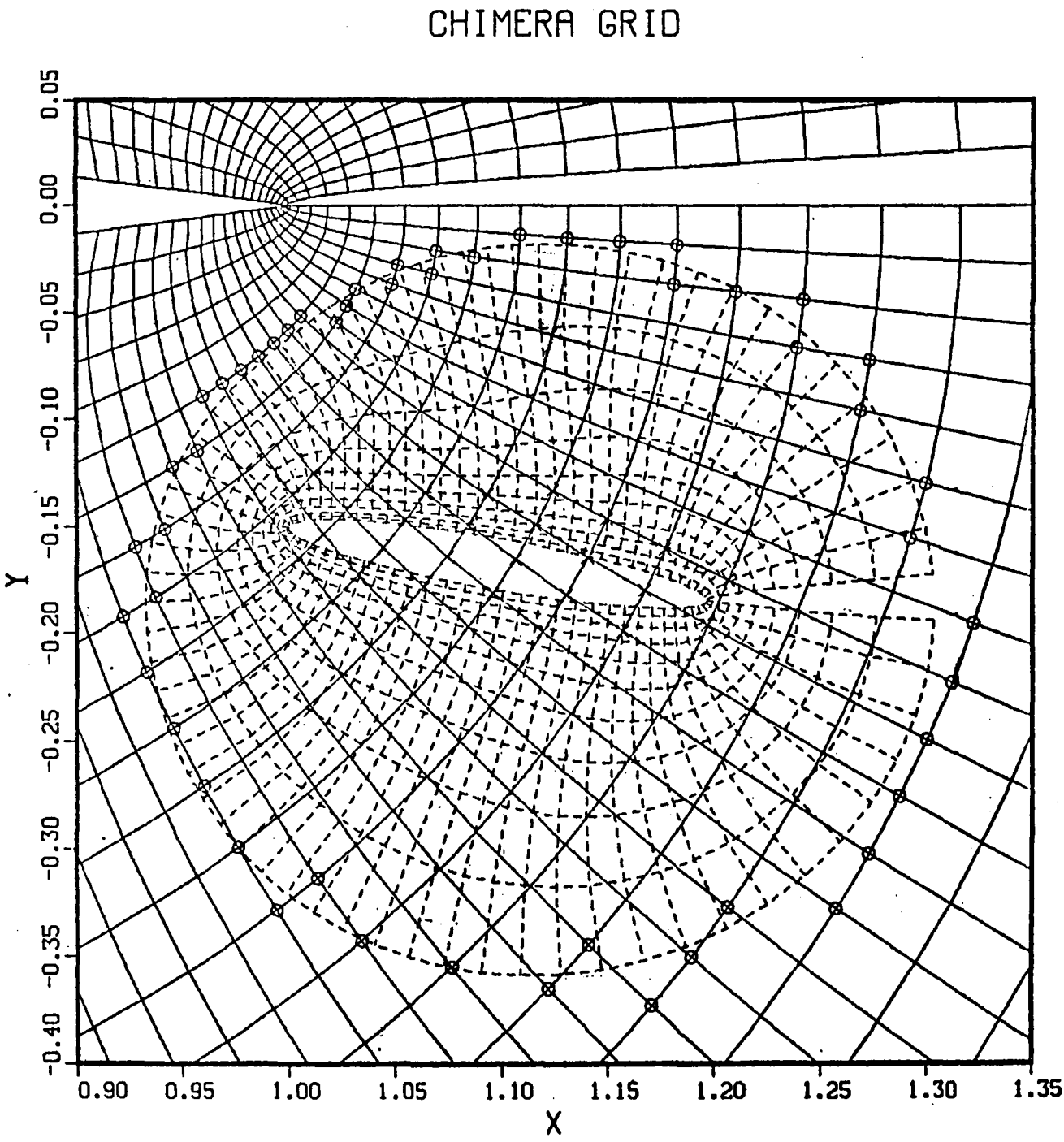

Figure 4.4.- Flap grid showing hole and fringe points.

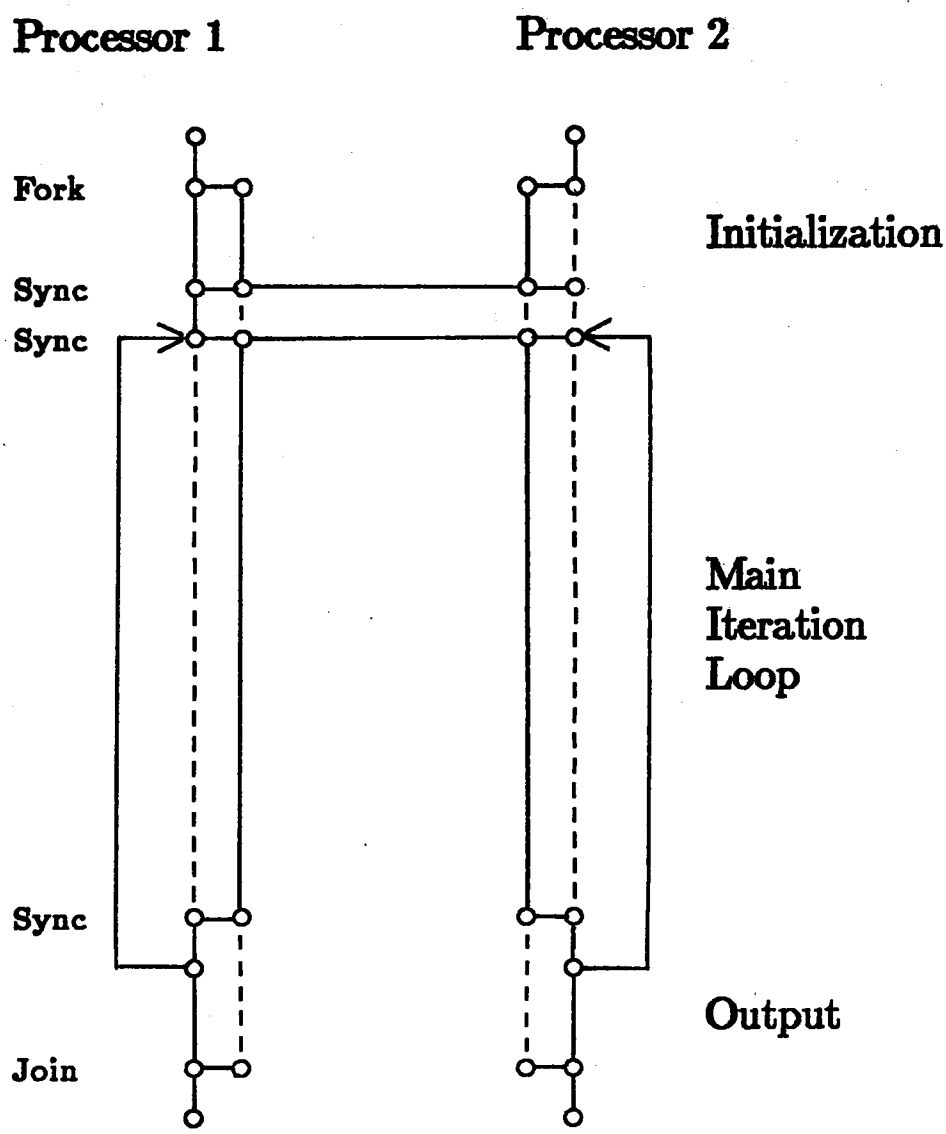

Figure 4.5.- Flow chart for concurrent application of incompressible problem using overset grids.

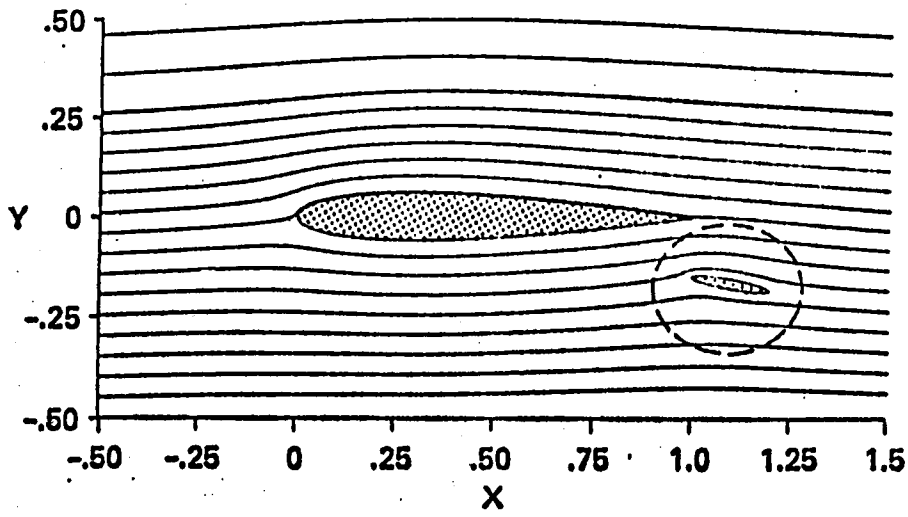

Figure 4.6.- Streamlines. Results of airfoil/flap configuration--synchronous.

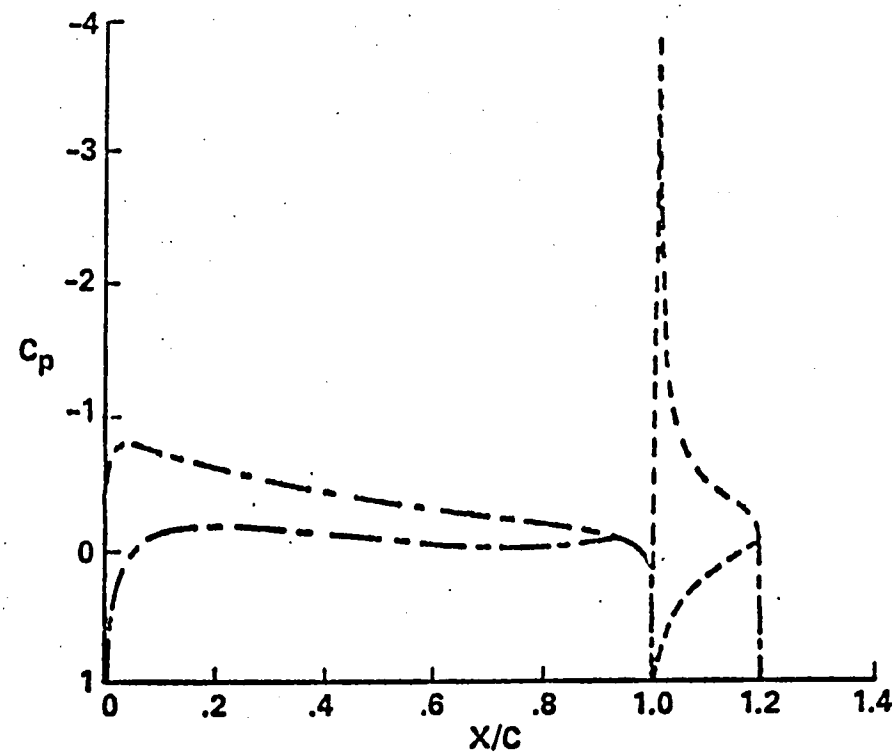

Figure 4.7.- Incompressible pressure distribution. Results of airfoil/flap configuration--synchronous.

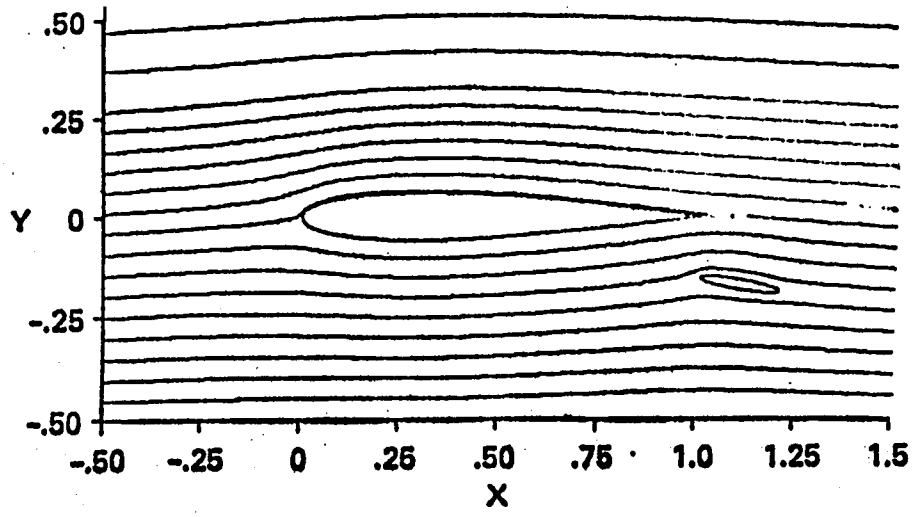

Figure 4.8.- Streamlines. Results of airfoil/flap configuration--asynchronous.

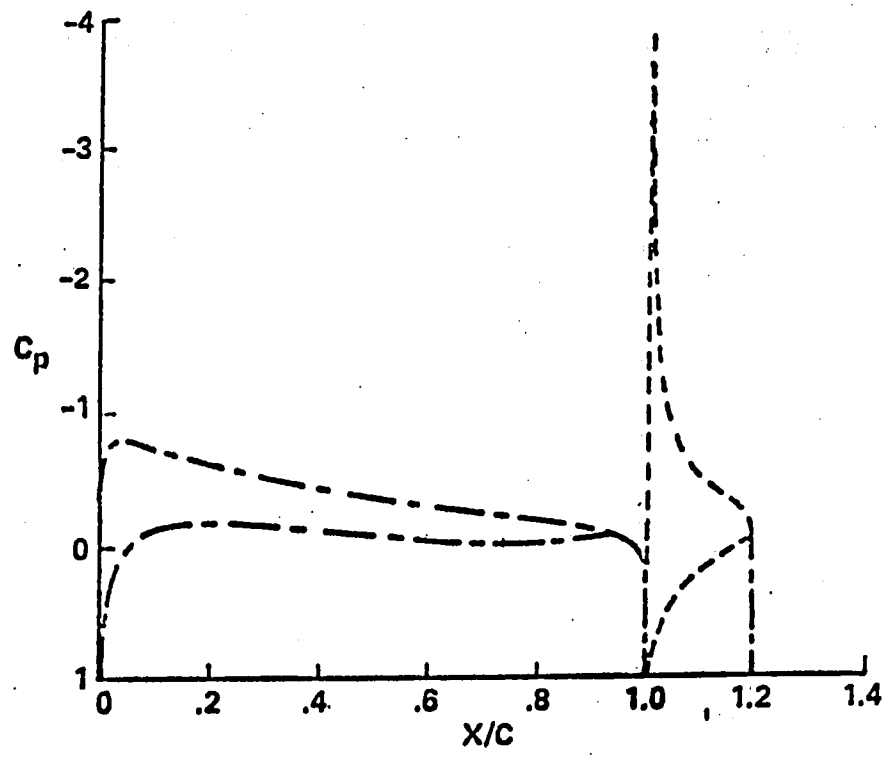

Figure 4.9.- Incompressible pressure distribution. Results of airfoil/flap configuration--asynchronous.

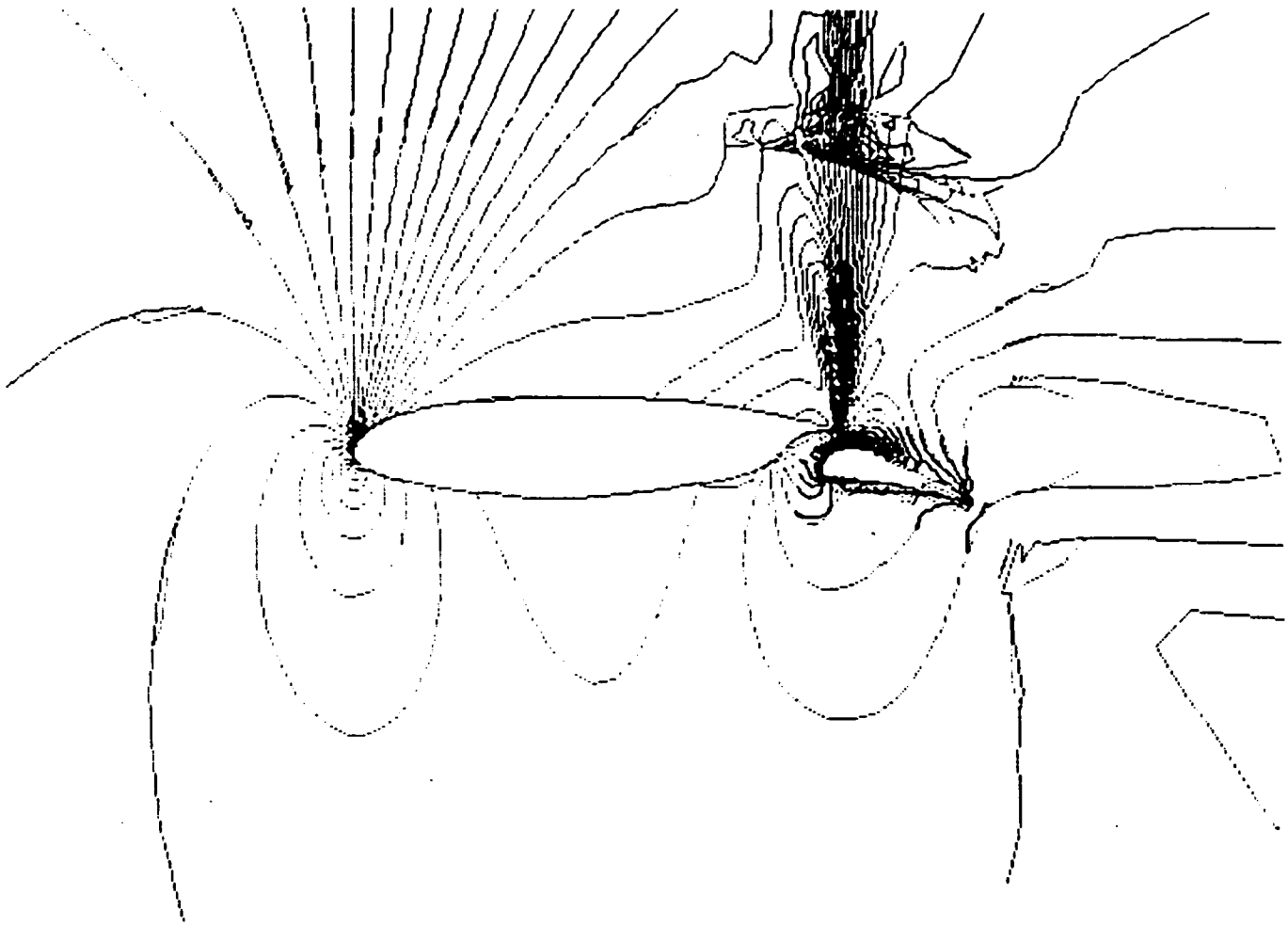

Figure 5.1.- Solution of transonic airfoil/flap configuration (ref. 2).

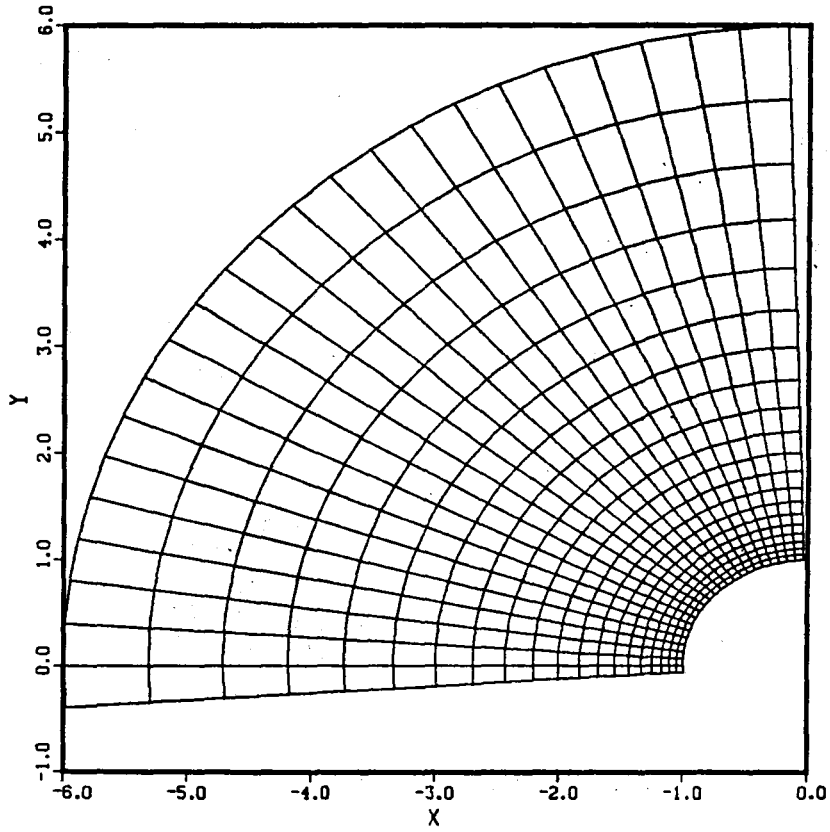

Figure 5.2.- Major grid for blunt-body application.

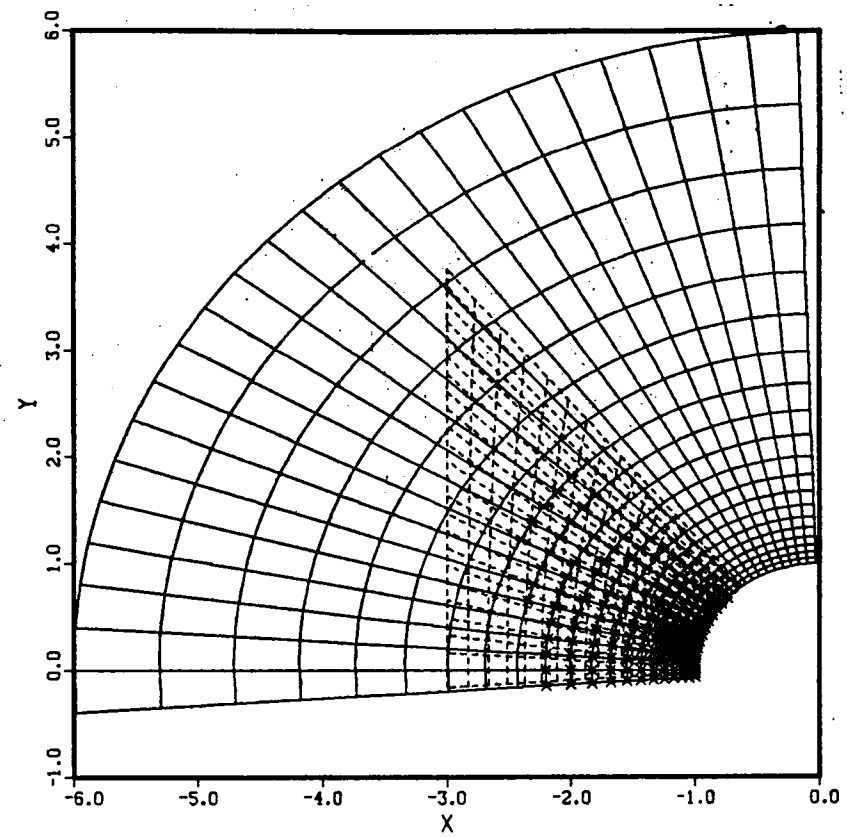

Figure 5.3.- Minor grid overset on major grid.

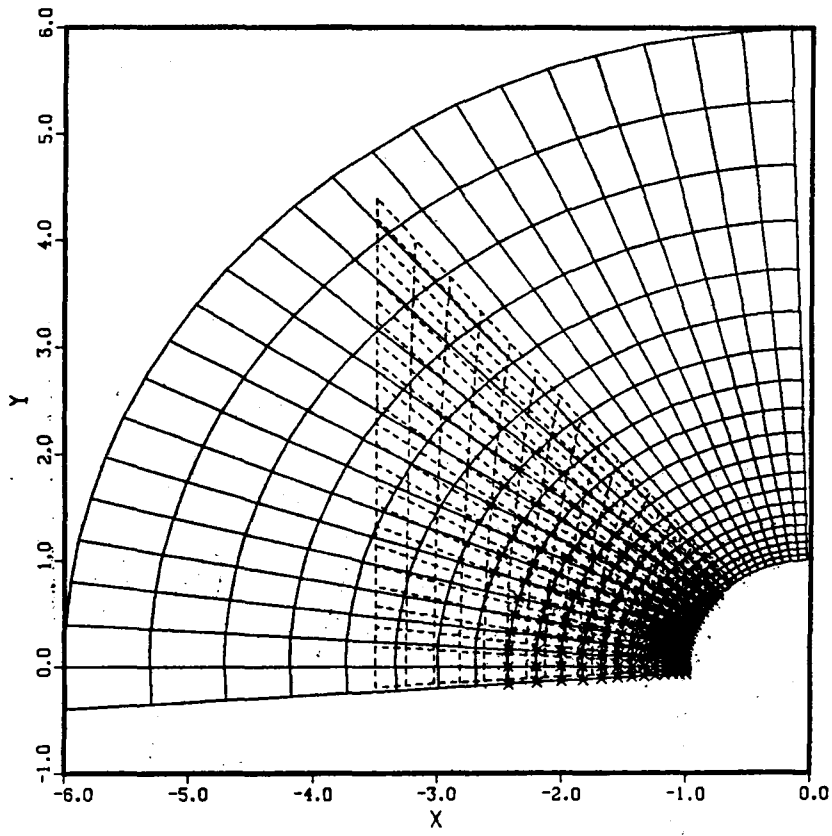

Figure 5.4.- Larger minor-grid overset on major grid.

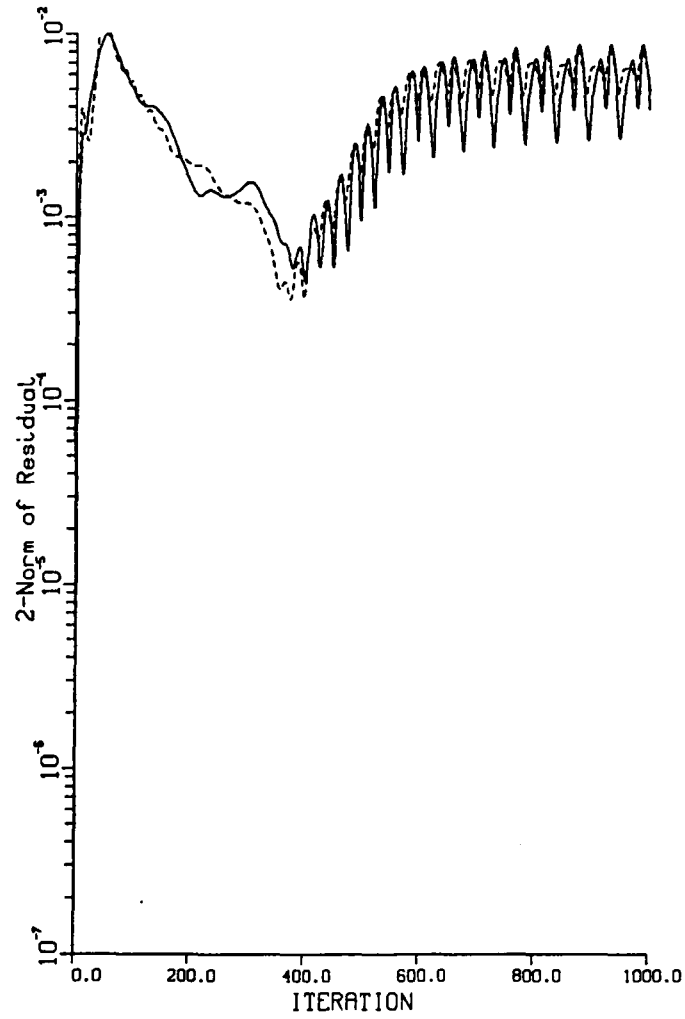

Figure 5.5.- Convergence history--direct interpolation, small minor grid.

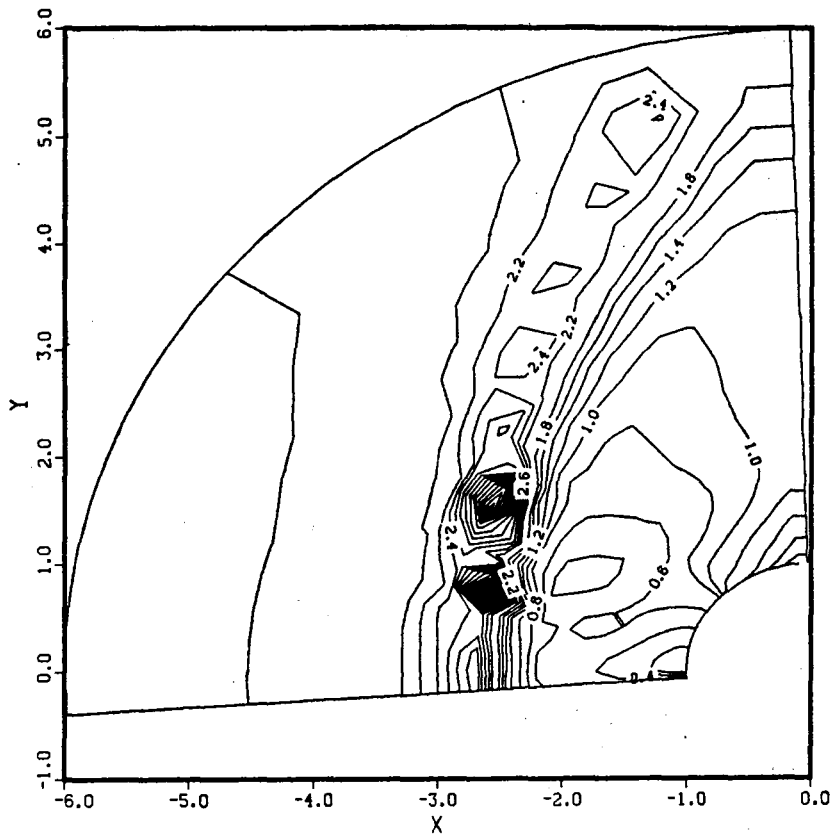

Figure 5.6.- Mach number contours--major grid using direct interpolation boundary scheme and small overset grid.

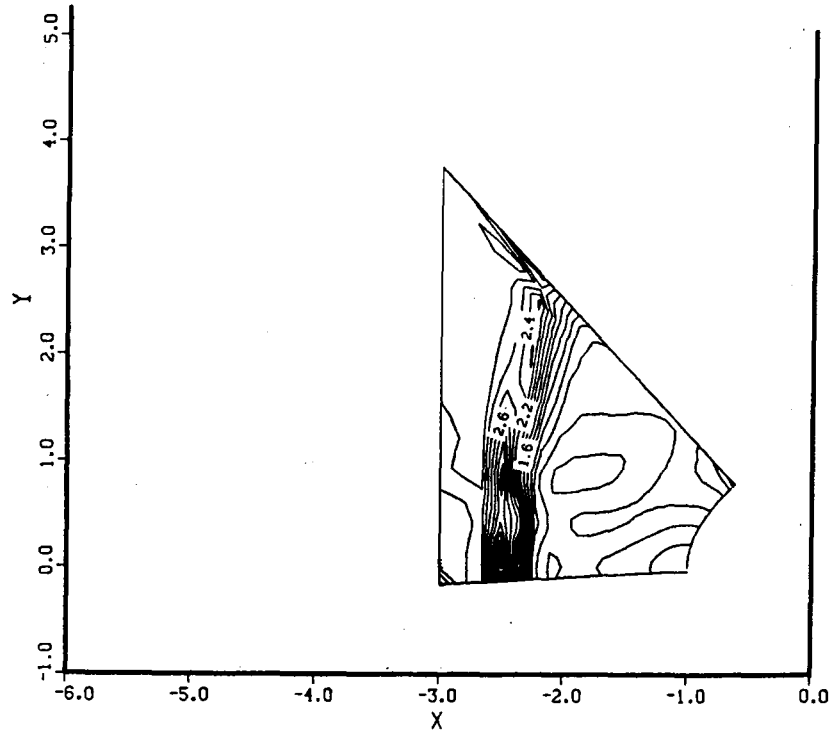

Figure 5.7.- Mach number contours--minor grid using direct interpolation boundary scheme.

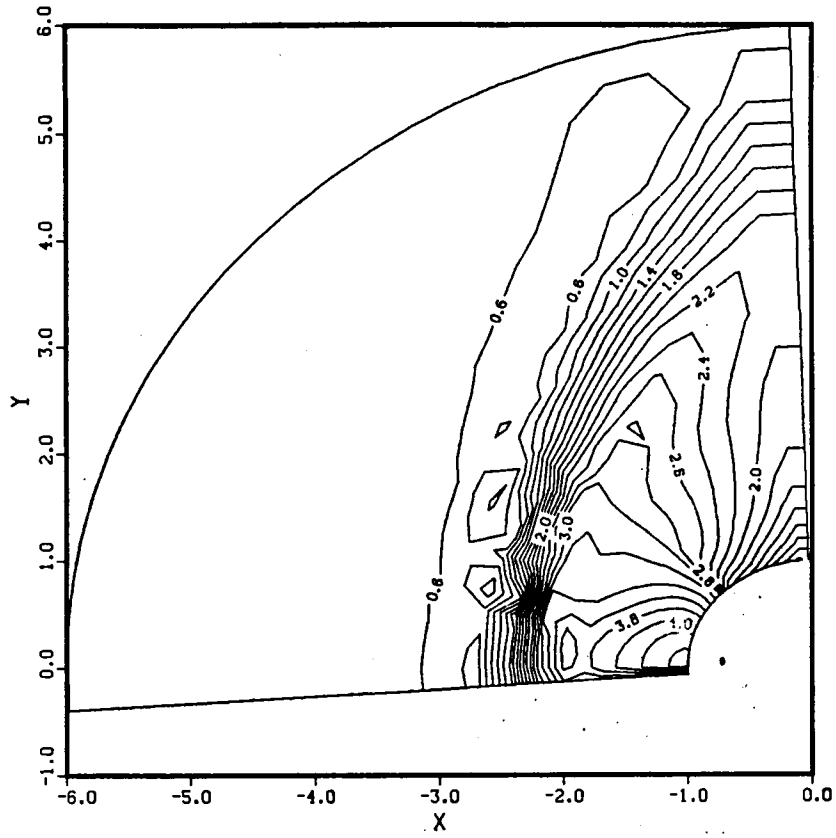

Figure 5.8.- Pressure contours--major grid using direct interpolation boundary scheme and small overset grid.

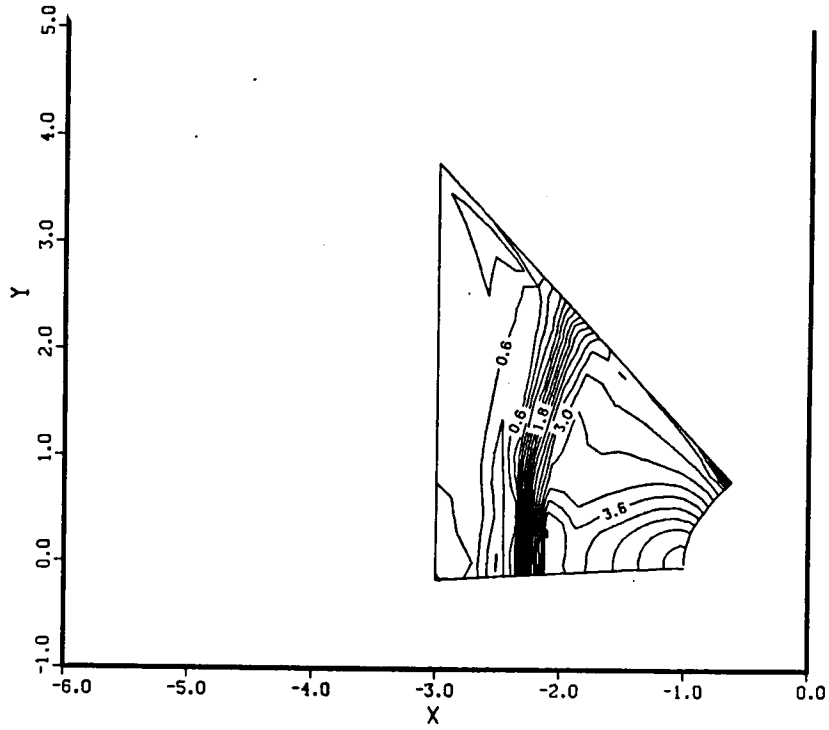

Figure 5.9.- Pressure contours--minor grid using direct interpolation boundary scheme.

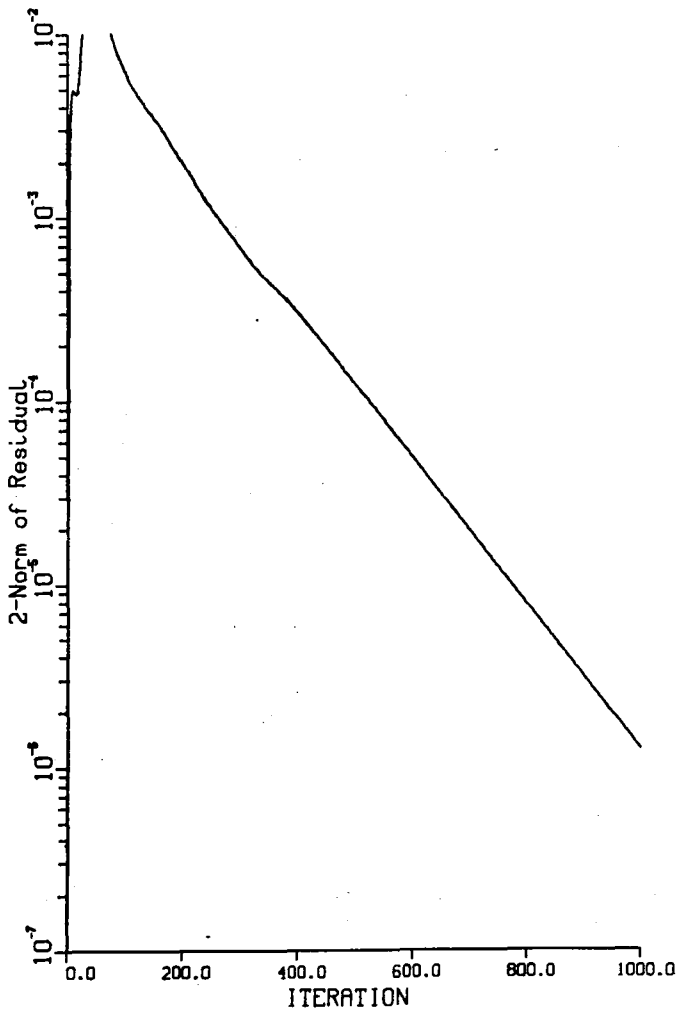

Figure 5.10.- Convergence history--single grid.

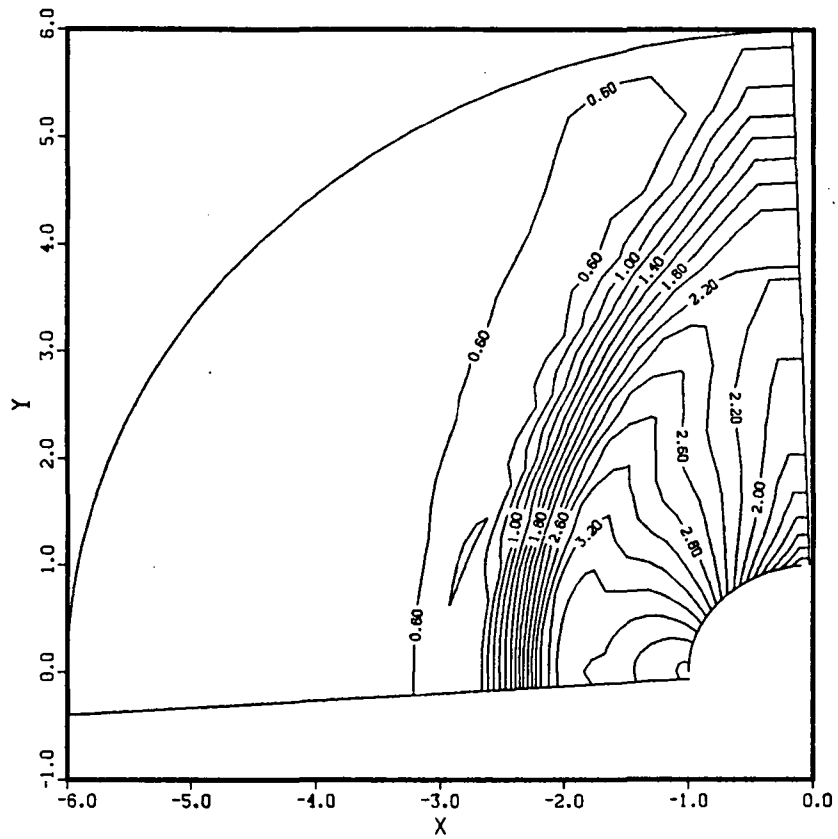

Figure 5.11.- Mach number contours--single grid solution.

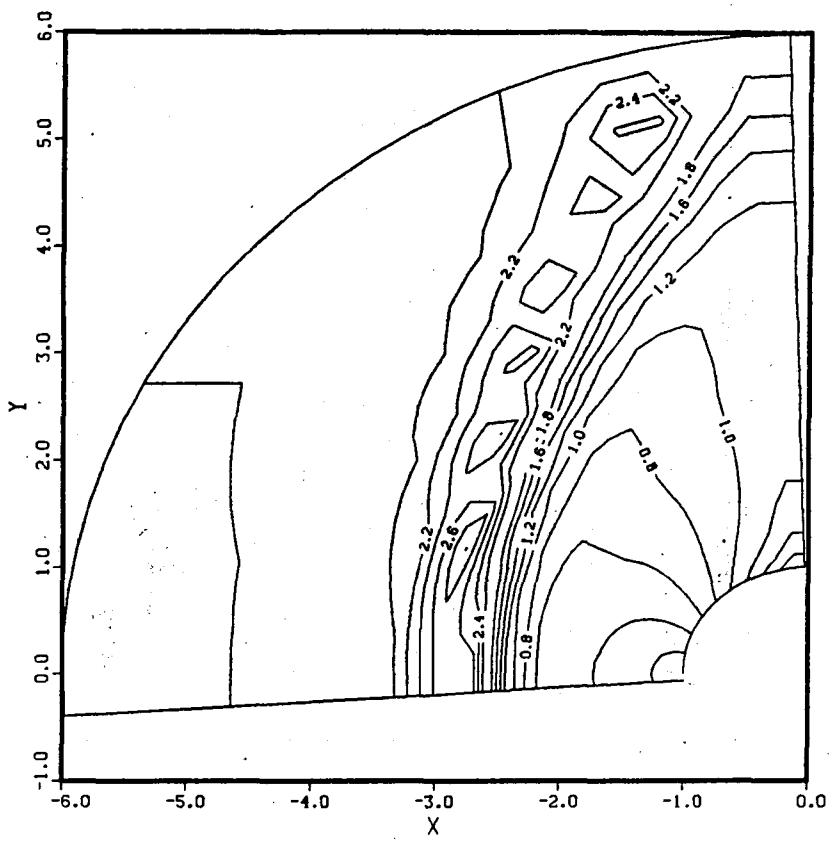

Figure 5.12.- Pressure contours--single grid solution.

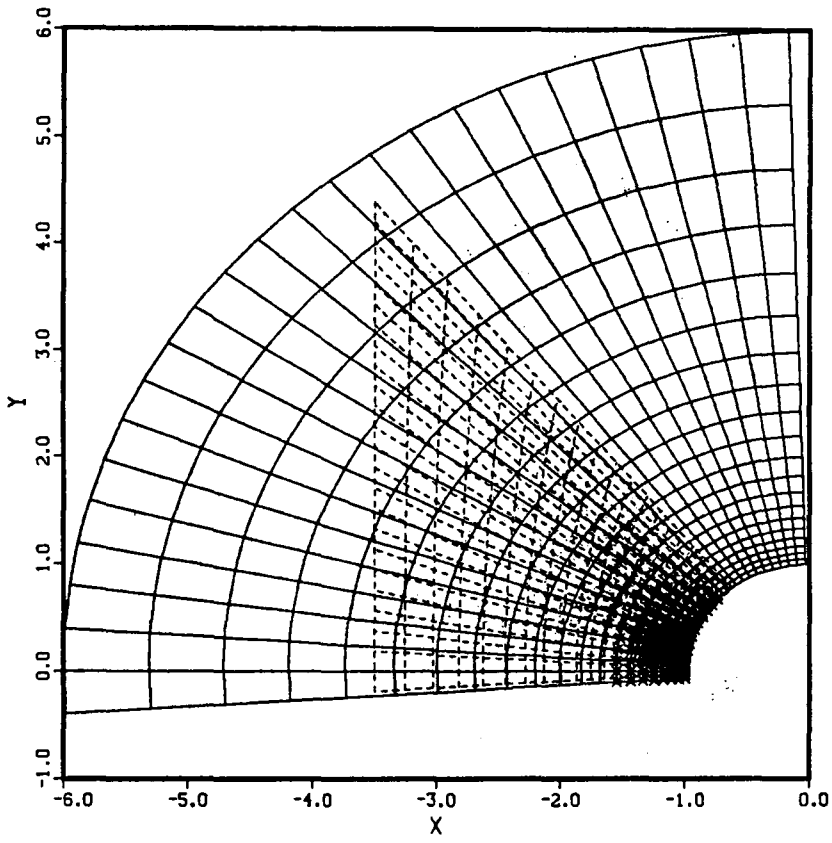

Figure 5.13.- Larger minor grid overset on major grid with smaller hole.

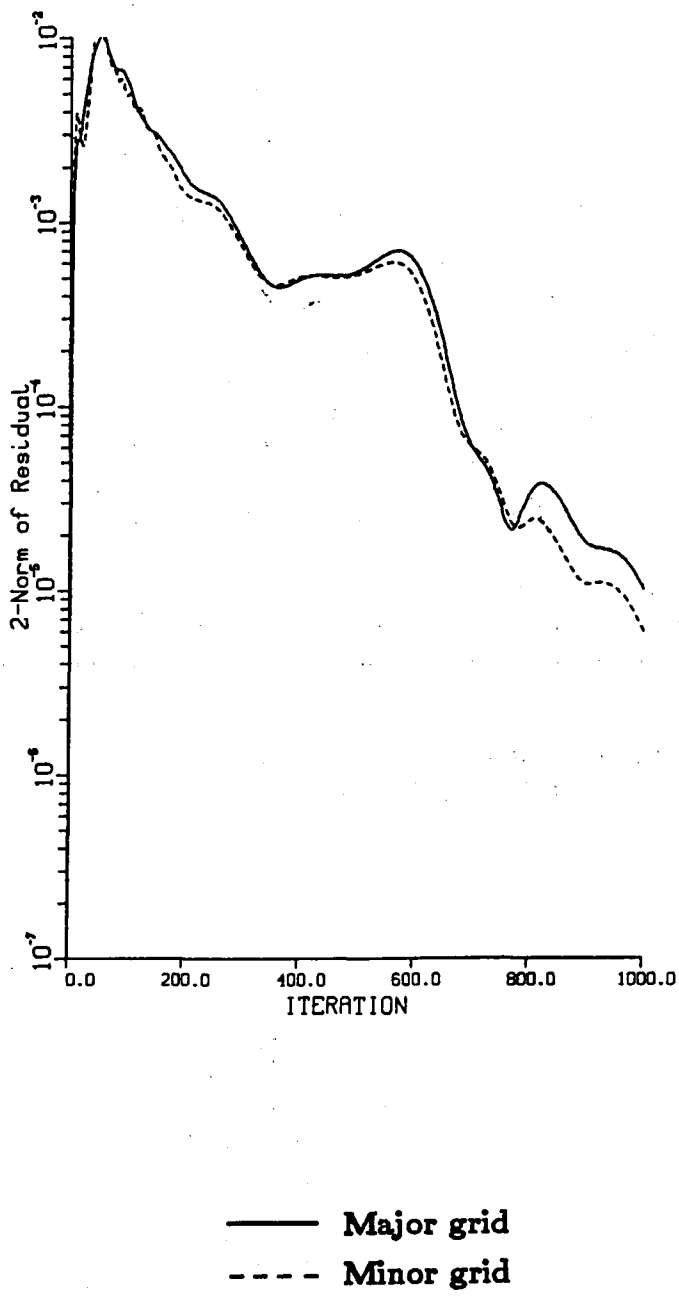

Figure 5.14.- Convergence history--direct interpolation, large minor grid.

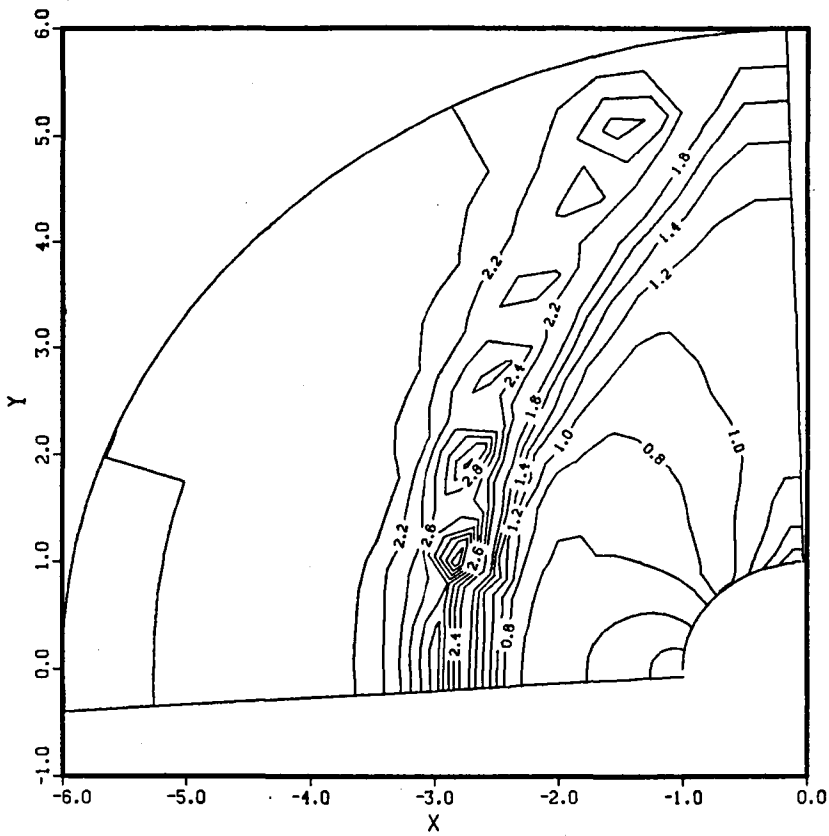

Figure 5.15.- Mach number contours--major grid using direct interpolation boundary scheme and large overset grid.

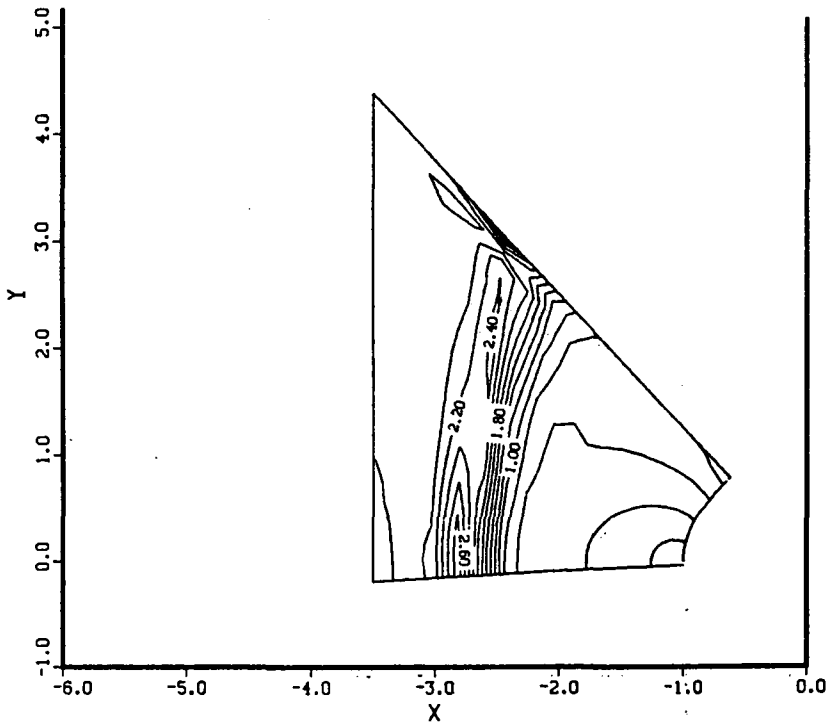

Figure 5.16.- Mach number contours--minor grid using direct interpolation boundary scheme.

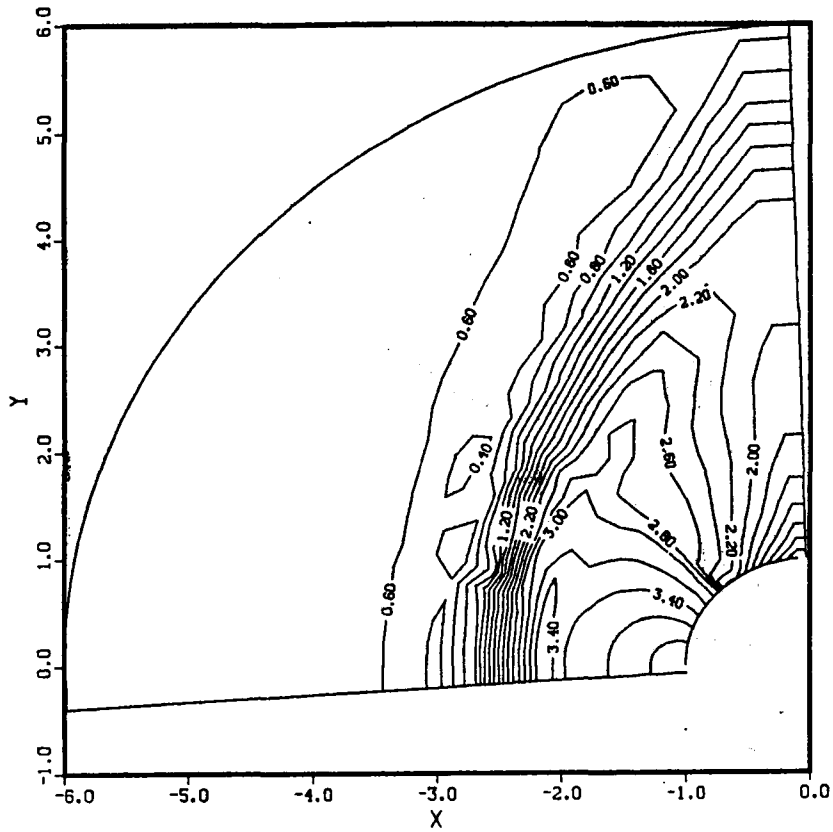

Figure 5.17.- Pressure contours--major grid using direct interpolation boundary scheme and large overset grid.

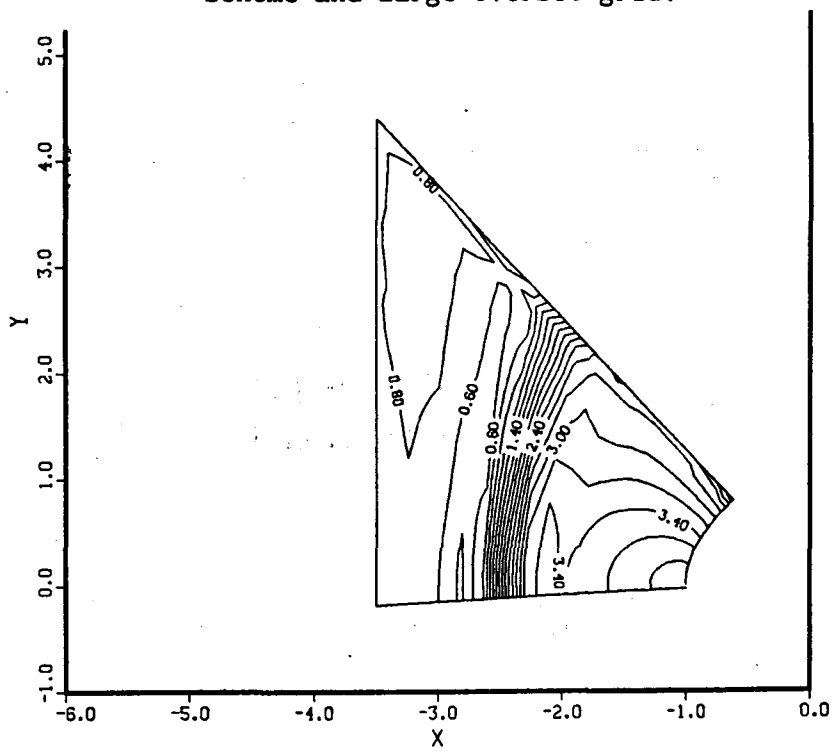

Figure 5.18.- Pressure contours--minor grid using direct interpolation boundary scheme.

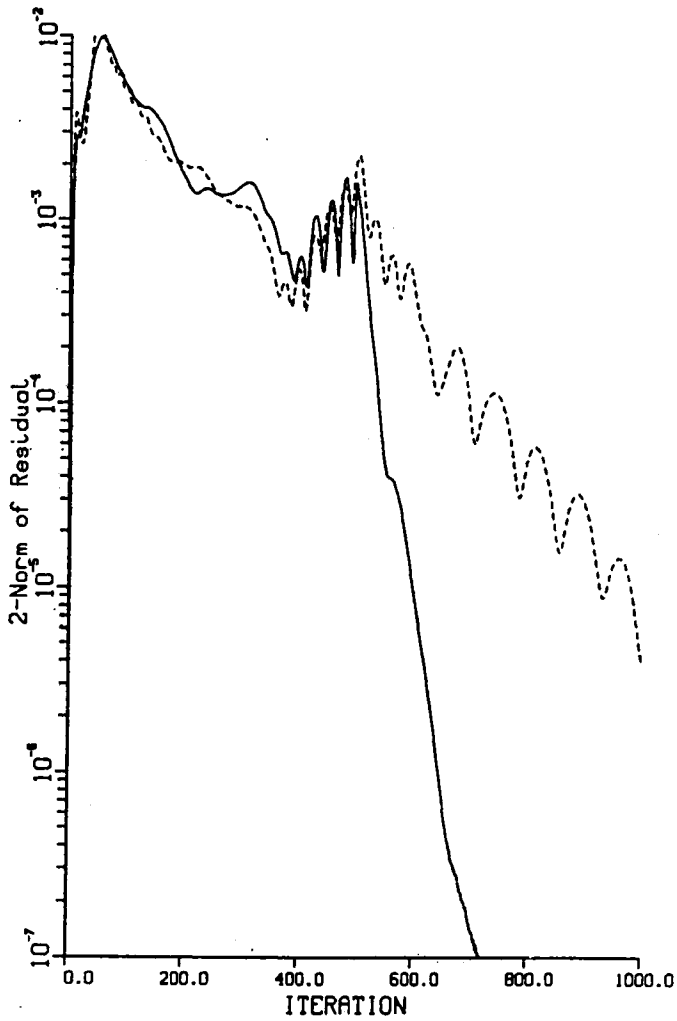

Figure 5.19.- Convergence history--direct interpolation--500 iterations boundary data frozen--after 500 iterations.

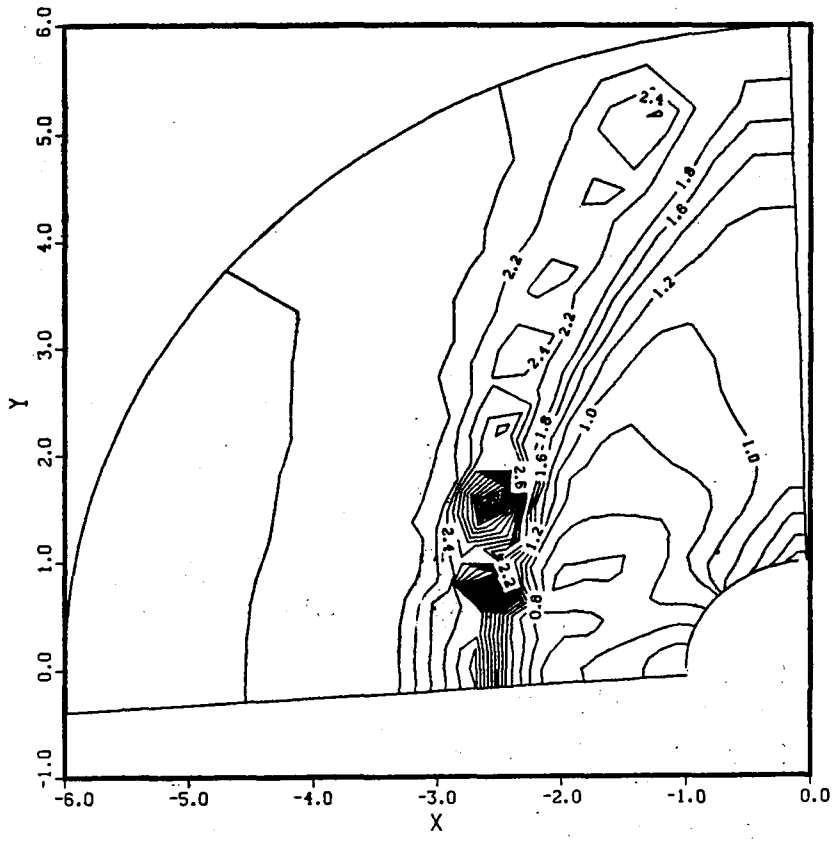

Figure 5.20.- Mach number contours--major grid using direct interpolation boundary scheme for first 500 iterations, then boundary frozen.

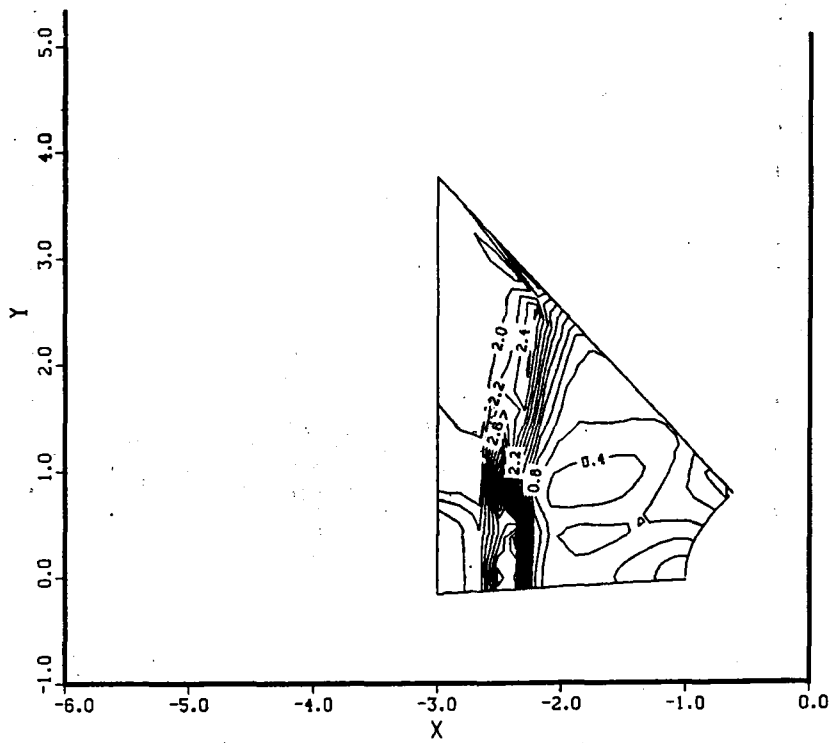

Figure 5.21.- Mach number contours--minor grid using direct interpolation boundary scheme for first 500 iterations, then boundary frozen.

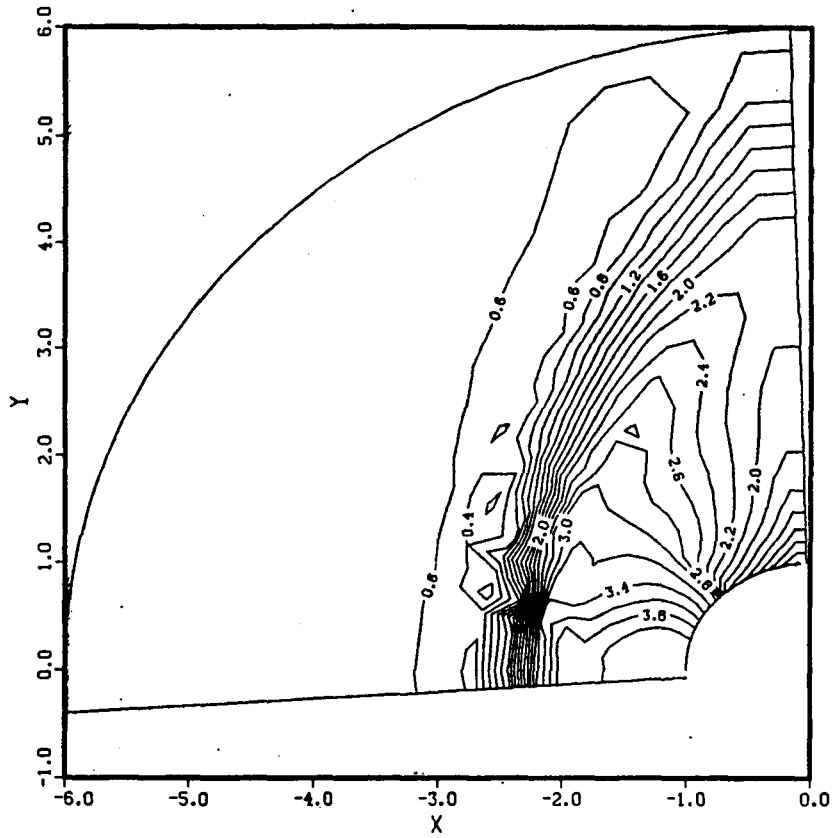

Figure 5.22.- Pressure contours--major grid using direct interpolation boundary scheme for first 500 iterations, then boundary frozen.

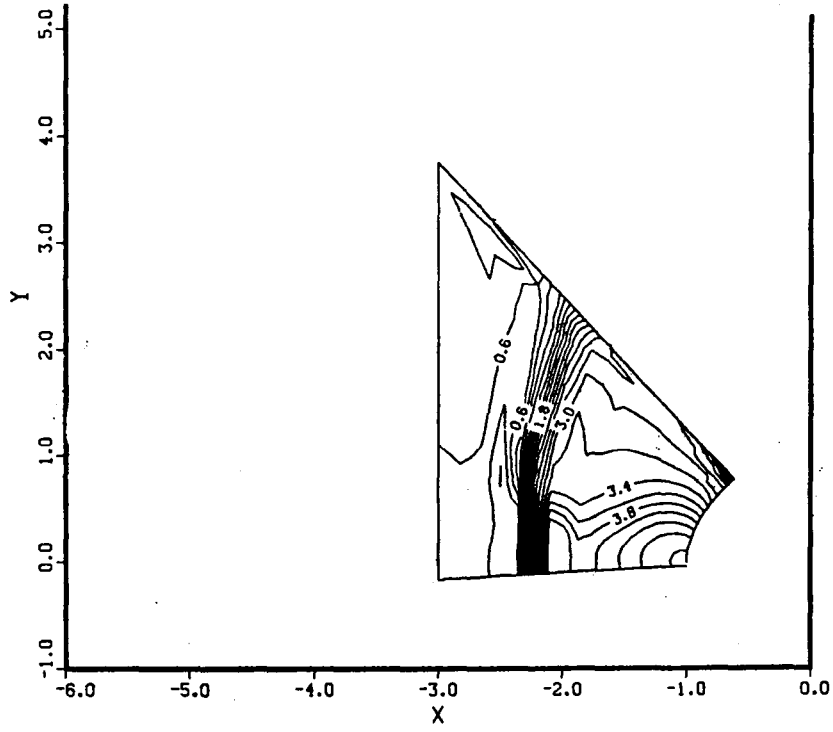

Figure 5.23.- Pressure contours--minor grid using direct interpolation boundary scheme for first 500 iterations, then boundary frozen.

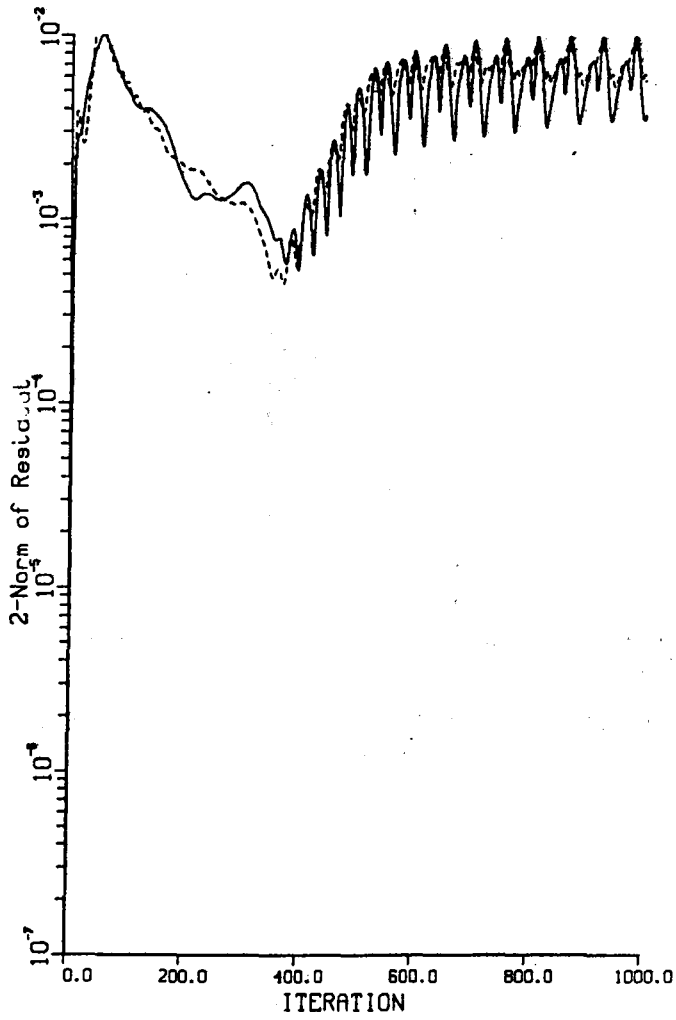

Figure 5.24.- Convergence history--boundary data averaging.

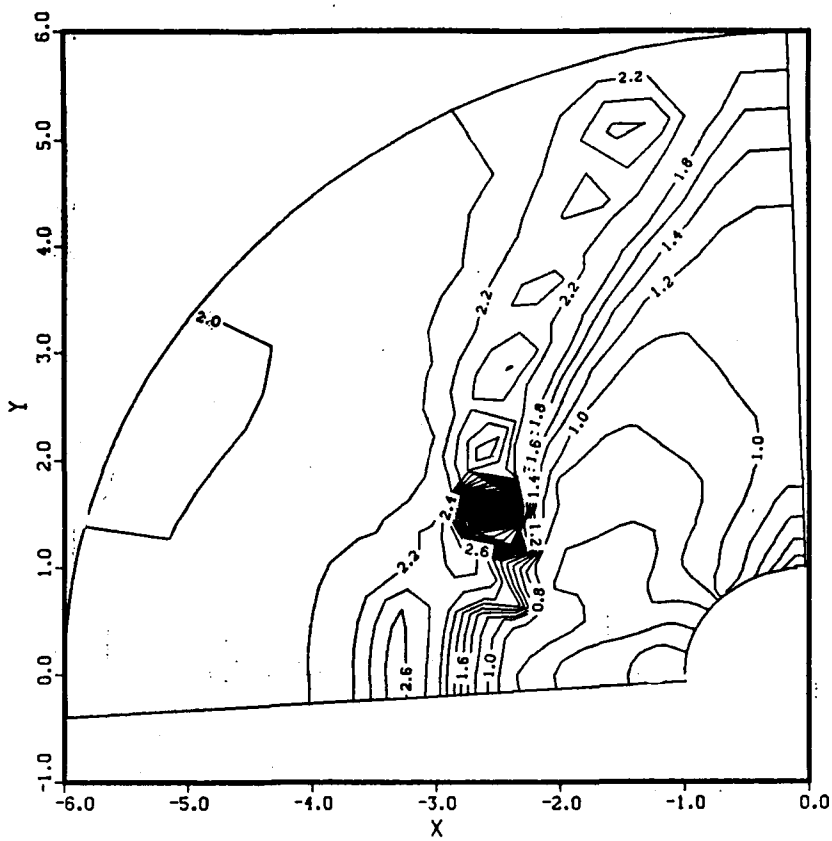

Figure 5.25.- Mach number contours--major grid using boundary-data-averaging boundary scheme.

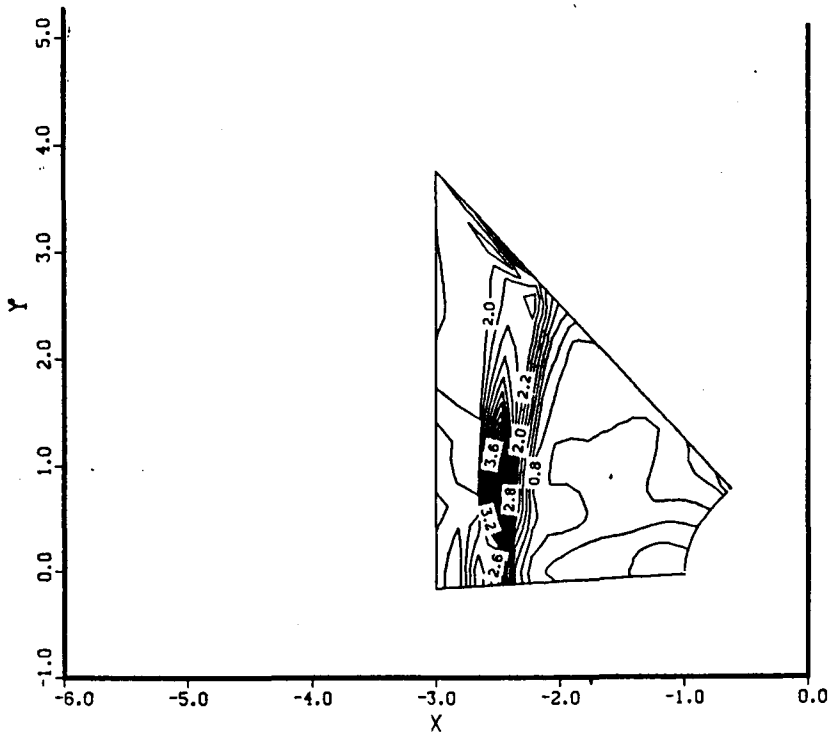

Figure 5.26.- Mach number contours--minor grid using boundary-data-averaging boundary scheme.

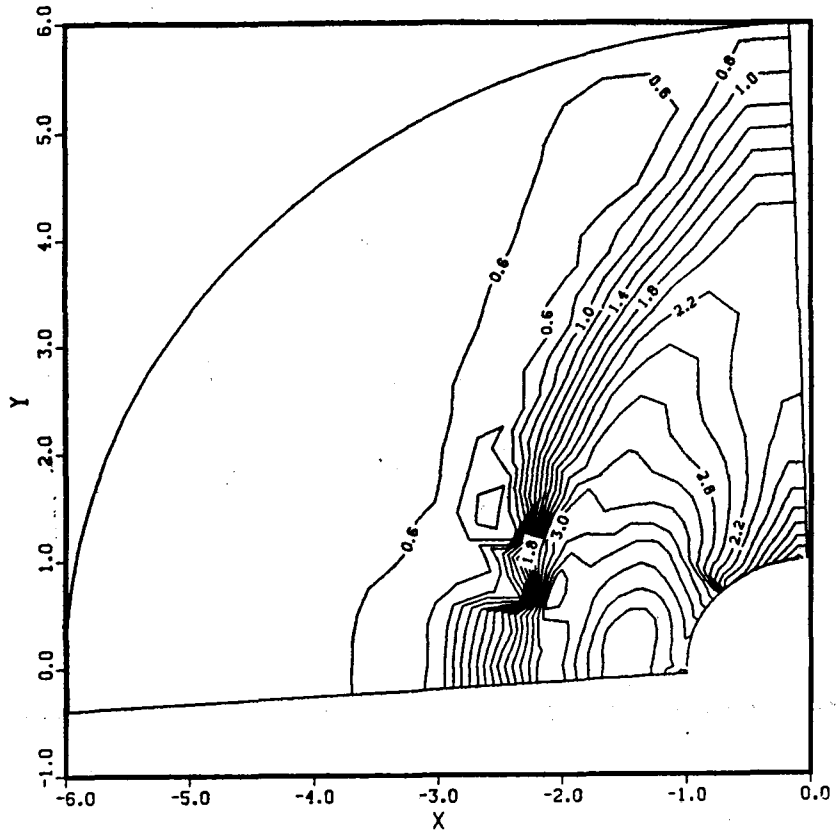

Figure 5.27.- Pressure contours--major grid using boundary-data-averaging boundary scheme.

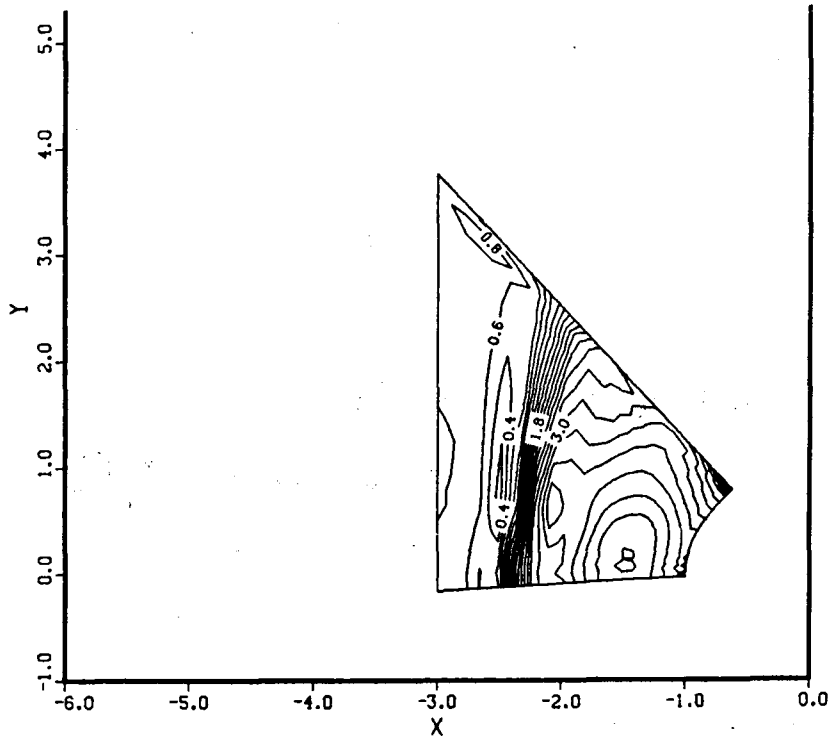

Figure 5.28.- Pressure contours--minor grid using boundary-data-averaging boundary scheme.

$$M > 1$$

$$\begin{aligned} \mathbf{u} - \mathbf{a} &\Rightarrow u + \tfrac{\gamma-1}{2} a \\ \mathbf{u} + \mathbf{a} &\Rightarrow u - \tfrac{\gamma-1}{2} a \\ \mathbf{u} &\Rightarrow s \end{aligned}$$

Figure 5.29.- Information flow through supersonic boundary.

$$M < 1$$

$$\begin{aligned} u + \tfrac{\gamma-1}{2} a &\Leftarrow \mathbf{u} - \mathbf{a} \\ \mathbf{u} + \mathbf{a} &\Rightarrow u - \tfrac{\gamma-1}{2} a \\ \mathbf{u} &\Rightarrow s \end{aligned}$$

Figure 5.30.- Information flow through subsonic boundary.

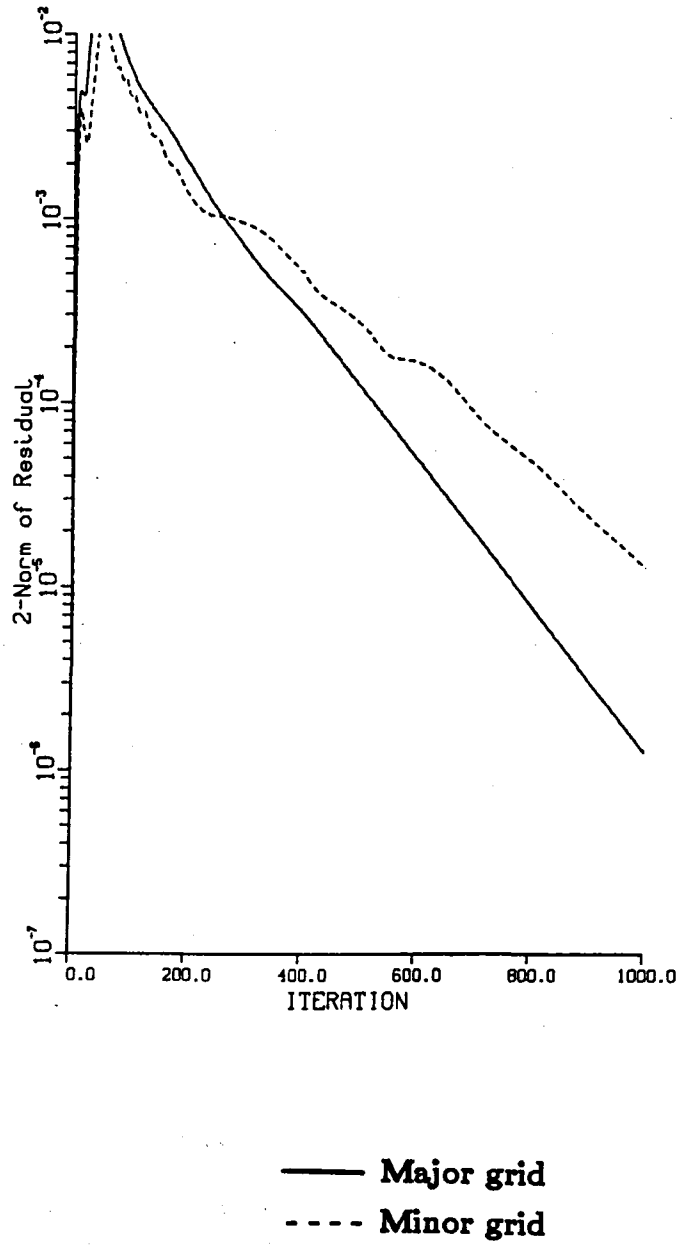

Figure 5.31.- Convergence history--no hole in major grid, characteristic boundaries on minor grid.

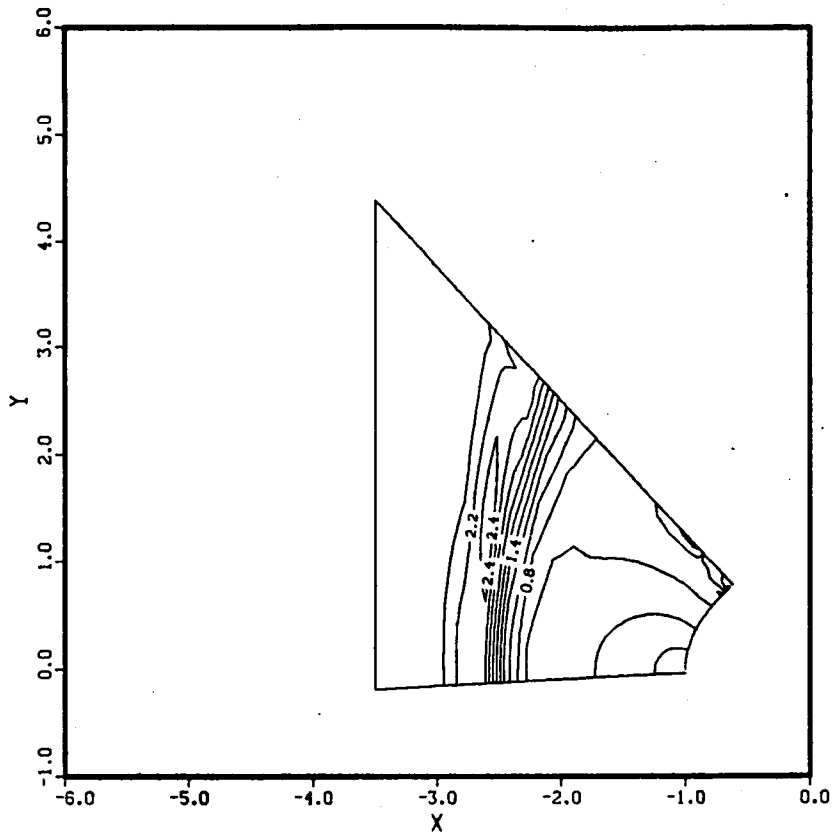

Figure 5.32.- Mach number contours--minor grid using characteristic boundary scheme and no hole in major grid.

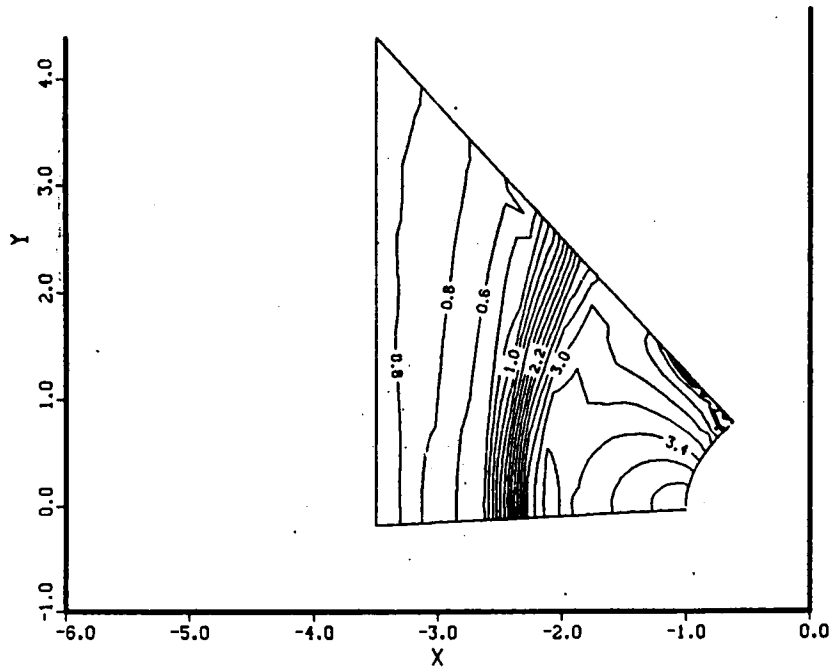

Figure 5.33.- Pressure contours--minor grid using characteristic boundary scheme and no hole in major grid.

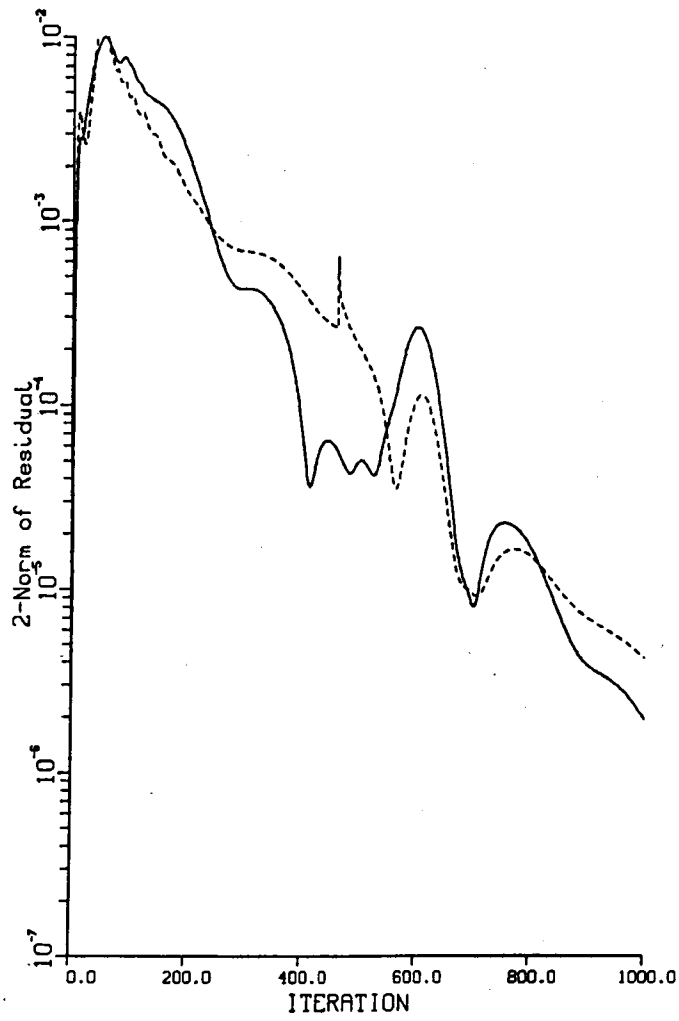

Figure 5.34.- Convergence history--characteristic boundaries on minor grid.

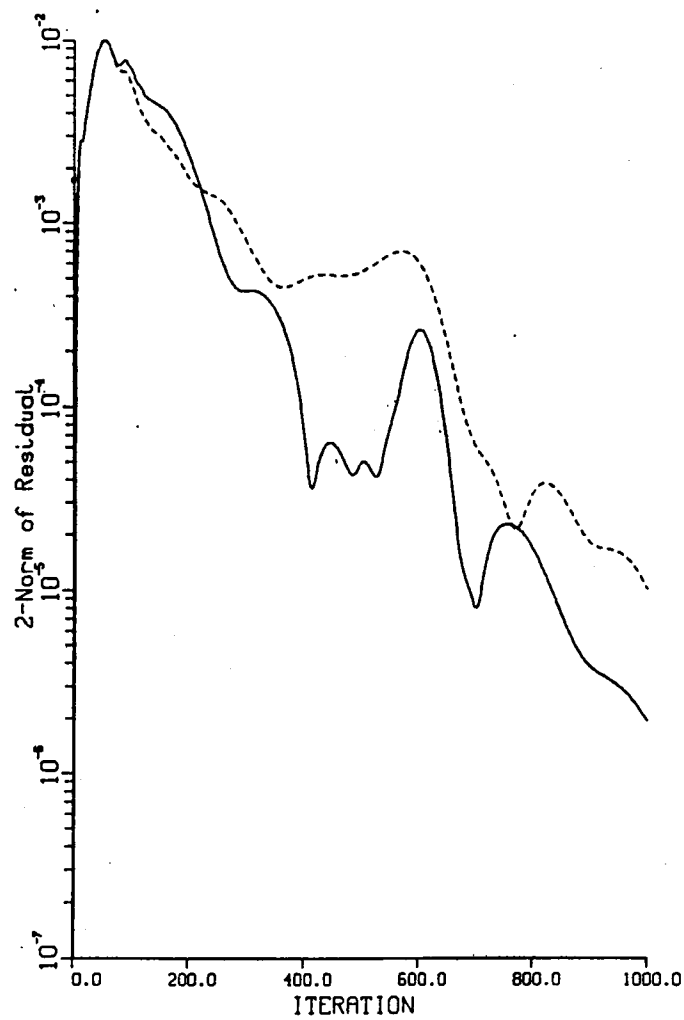

Figure 5.35.- Convergence history--characteristic boundary scheme vs. direct interpolation--major grid.

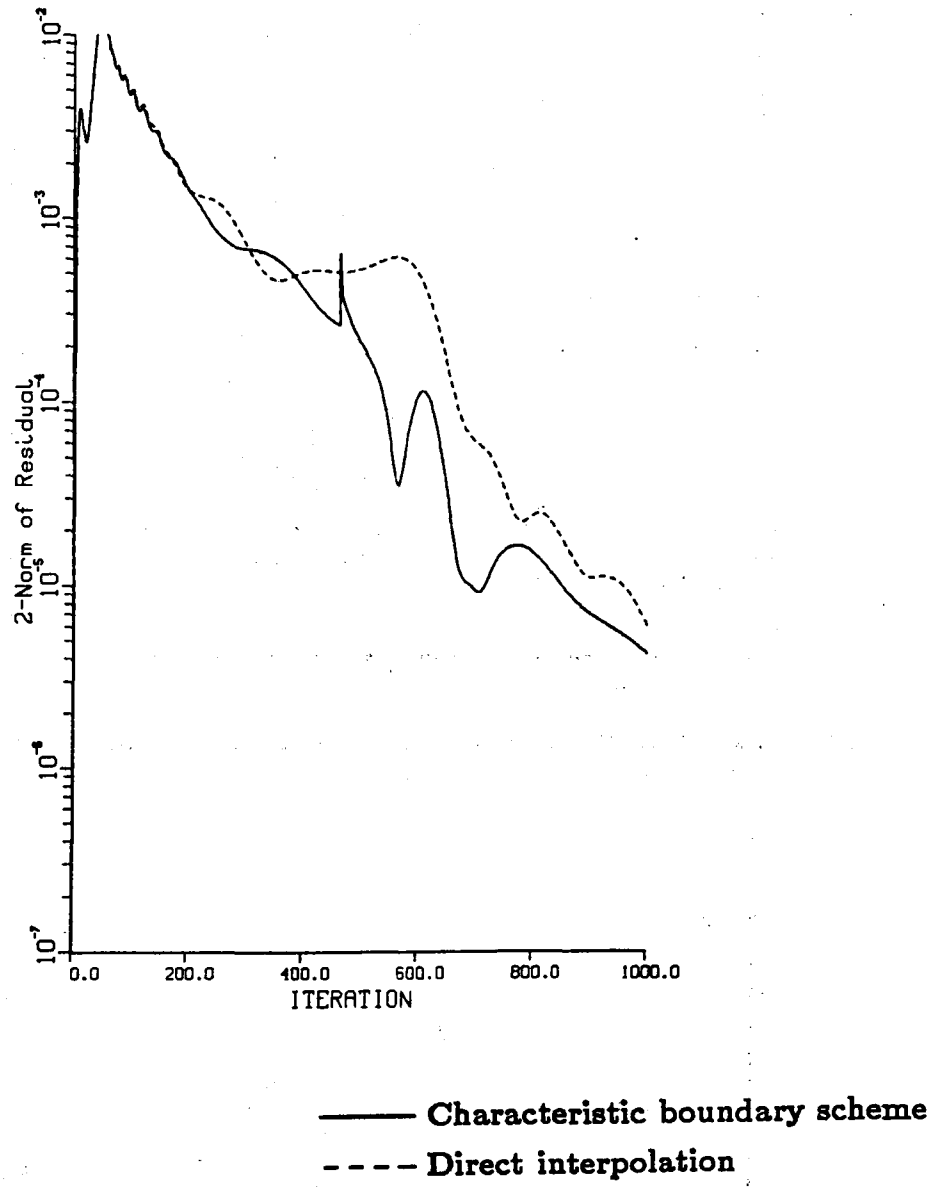

Figure 5.36.- Convergence history--characteristic boundary scheme vs. direct interpolation--minor grid.

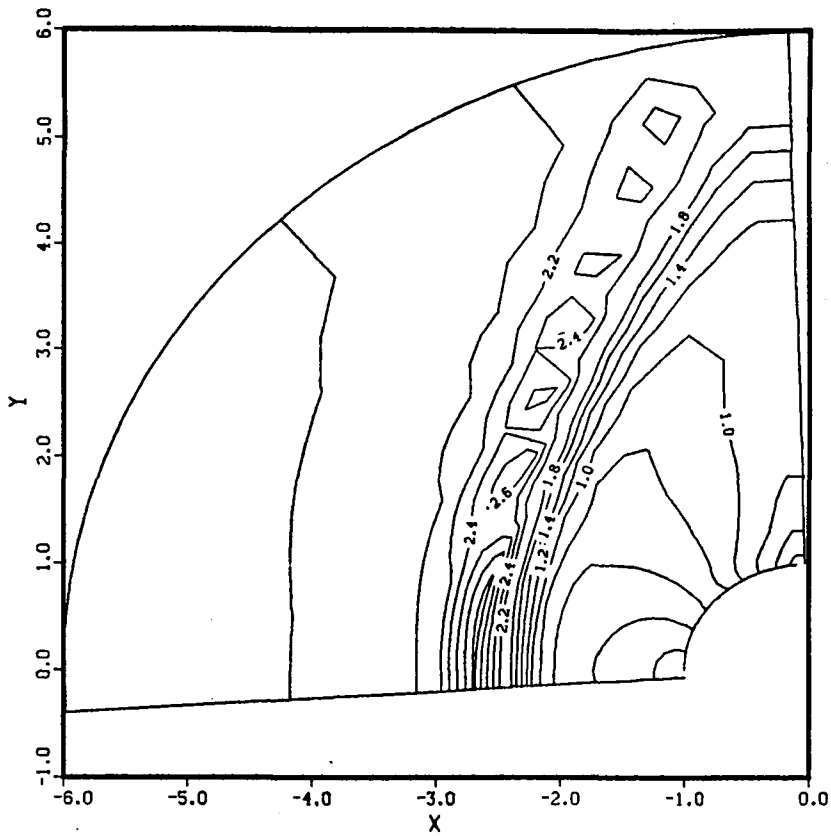

Figure 5.37.- Mach number contours--major grid using characteristic boundary scheme.

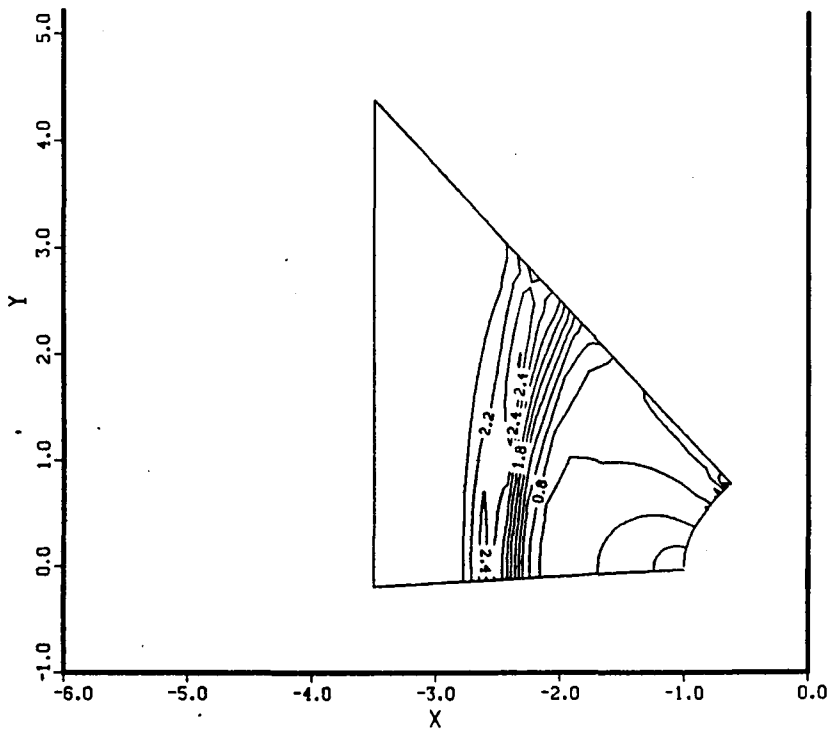

Figure 5.38.- Mach number contours--minor grid using characteristic boundary scheme.

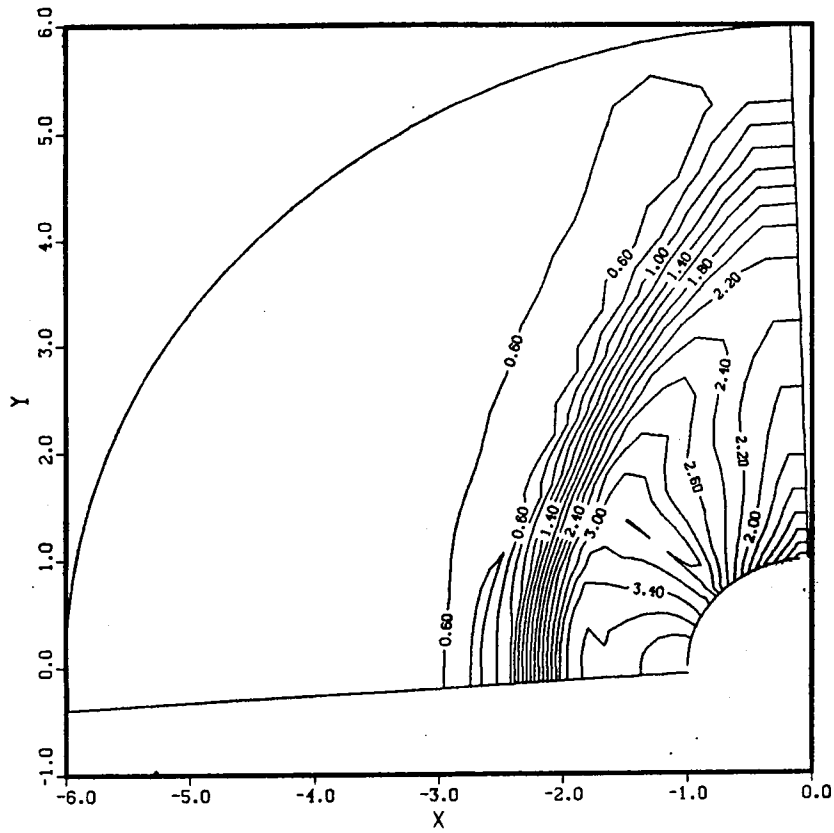

Figure 5.39.- Pressure contours--major grid using characteristic boundary scheme.

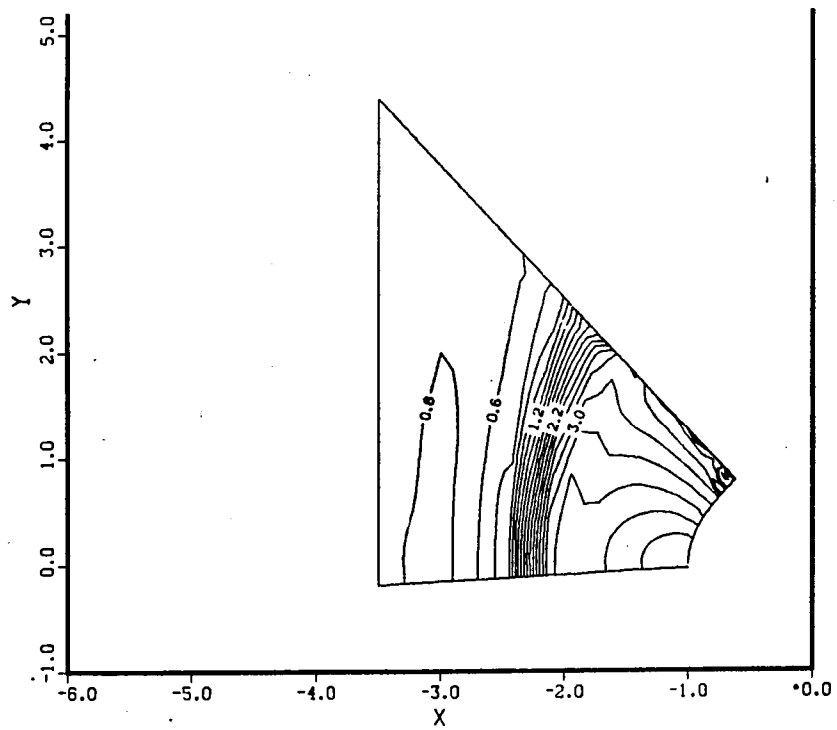

Figure 5.40.- Pressure contours--minor grid using characteristic boundary scheme.

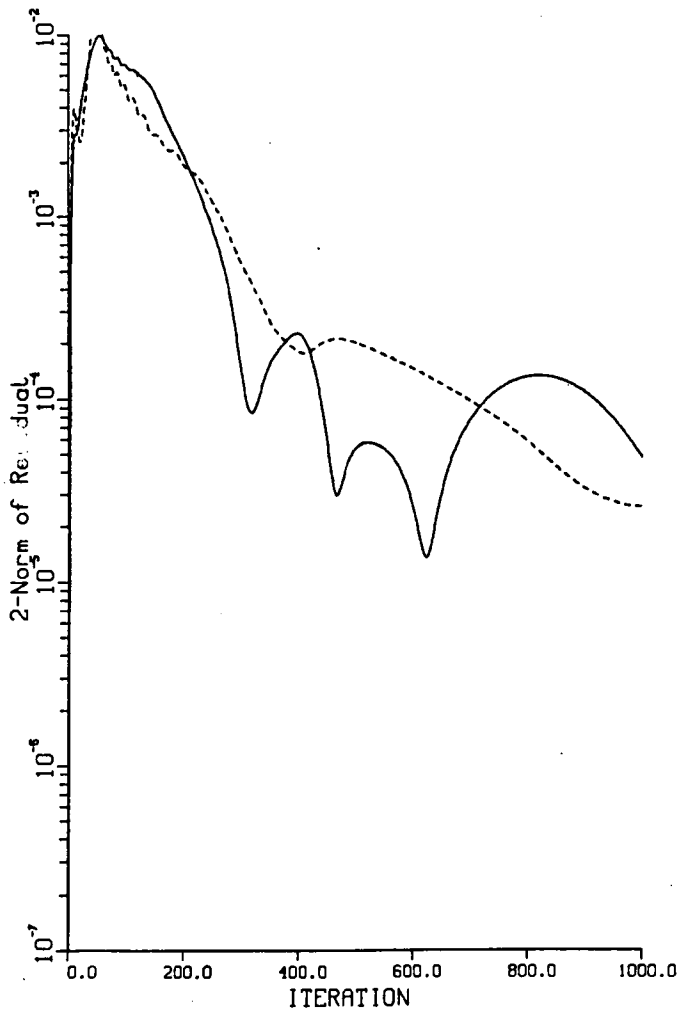

Figure 5.41.- Convergence history--characteristic boundary scheme using grids of figure 5.4.

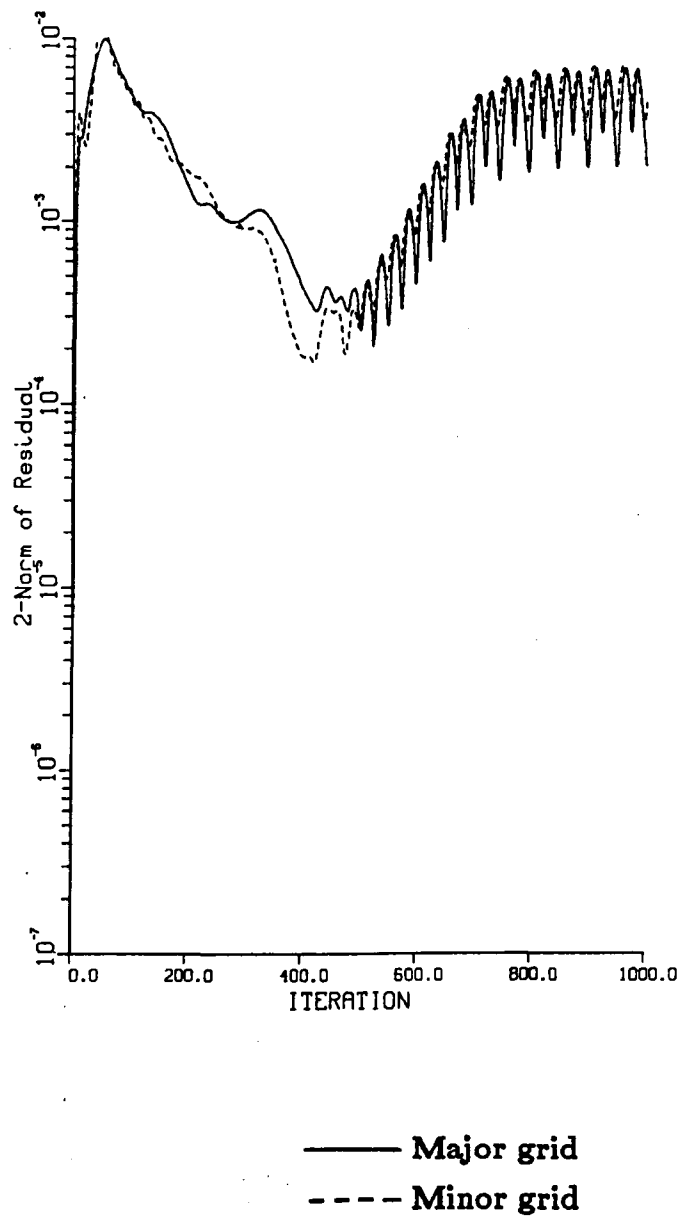

Figure 5.42.- Convergence history--characteristic boundary scheme using grids of figure 5.3.

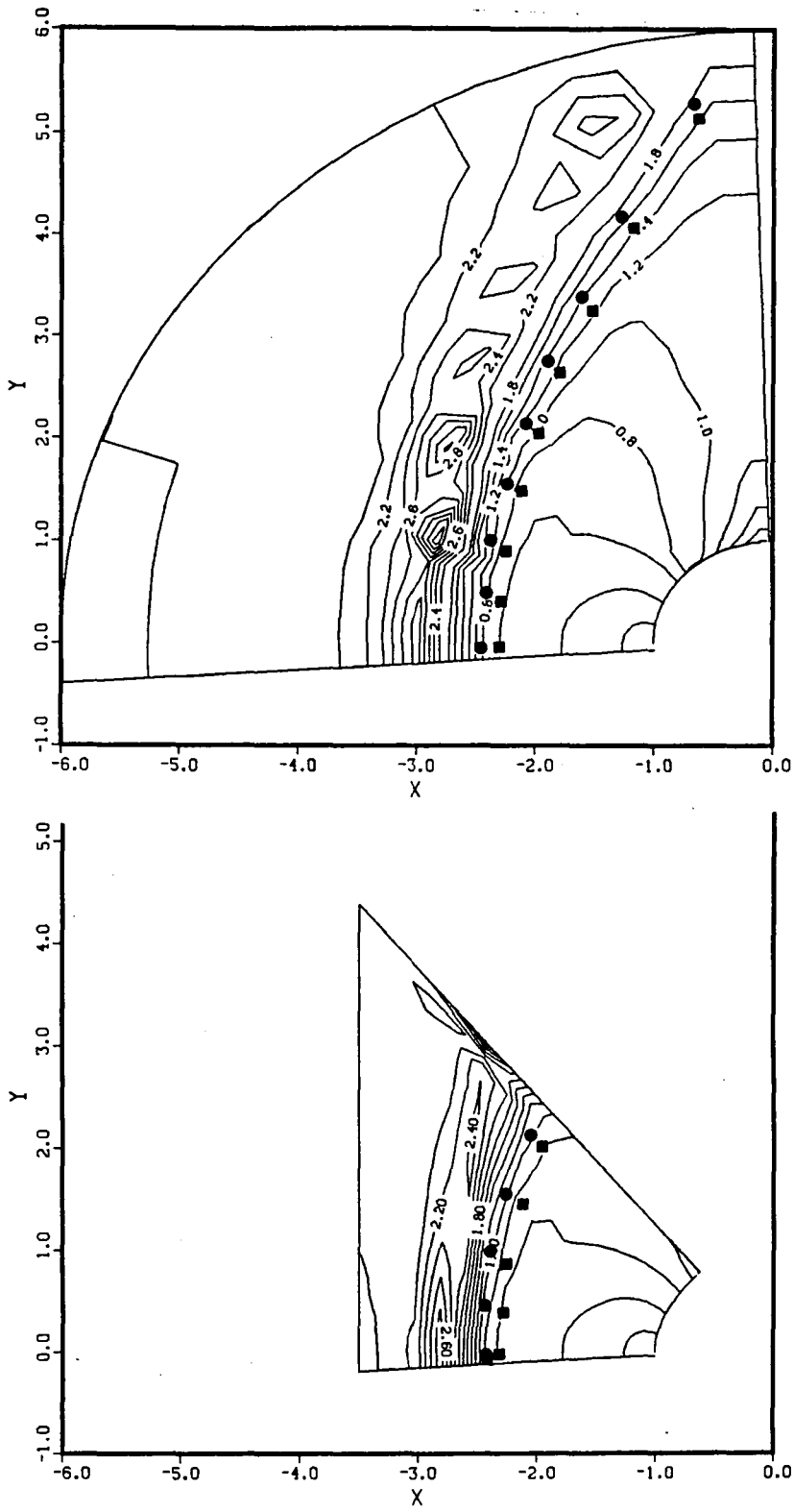

Figure 5.43.- Mach number contours (M = 2.0)--direct interpolation boundary scheme.

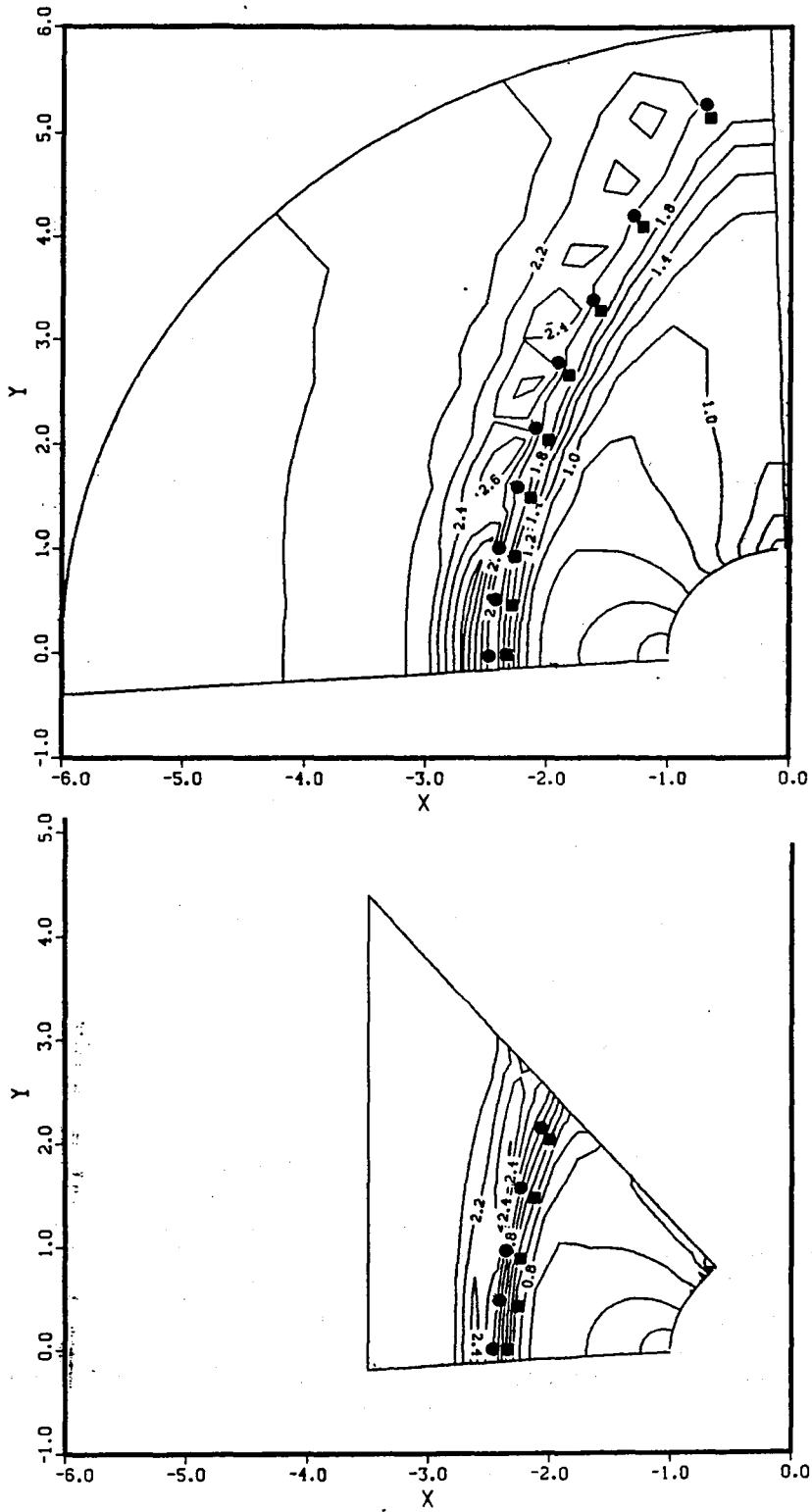

Figure 5.44.- Mach number contours (M = 2.0)--characteristic boundary scheme.

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM 86675		

4. Title and Subtitle	5. Report Date
MULTIPLE GRID PROBLEMS ON CONCURRENT-PROCESSING COMPUTERS	February 1986
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
D. Scott Eberhardt and Donald Baganoff (Department of Aeronautics and Astronautics, Stanford University, Stanford, CA 94305)	A-85103
	10. Work Unit No.

9. Performing Organization Name and Address	11. Contract or Grant No.
Ames Research Center Moffett Field, CA 94035	
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address	Technical Memorandum
National Aeronautics and Space Administration Washington, DC 20546	14. Sponsoring Agency Code 505-37-01

15. Supplementary Notes

Point of Contact: D. Scott Eberhardt, M/S 229-4, Ames Research Center, Moffett Field, CA 94035, (415)694-5235 or FTS 464-5235

16. Abstract

Three computer codes have been studied which make use of concurrent processing computer architectures in computational fluid dynamics (CFD). The three parallel codes were tested on a two processor MIMD facility at NASA Ames Research Center, and are suggested for efficient parallel computations.

The first code studied is a well-known program which makes use of the Beam and Warming, implicit, approximate factored algorithm. This study demonstrates the parallelism found in a well-known scheme and it achieved speedups exceeding 1.9 on the two processor MIMD test facility.

The second code studied made use of an embedded grid scheme which is used to solve problems having complex geometries. The particular application for this study considered an airfoil/flap geometry in an incompressible flow. The scheme eliminates some of the inherent difficulties found in adapting approximate factorization techniques onto MIMD machines and allows the use of chaotic relaxation and asynchronous iteration techniques.

The third code studied is an application of overset grids to a supersonic blunt body problem. The code addresses the difficulties encountered when using embedded grids on a compressible, and therefore nonlinear, problem. The complex numerical boundary system associated with overset grids is discussed and several boundary schemes are suggested. A boundary scheme based on the method of characteristics achieved the best results.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Concurrent processing Parallel processing Comp. fluid dynamics Overset grids	Unlimited Subject category - 059

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price*
Unclassified	Unclassified	110	A06

*For sale by the National Technical Information Service, Springfield, Virginia 22161

End of Document