Spring 2000

Orthogonal grid generation, non-reflecting boundary condition, and parallel computation for fluid flow

Shaopeng Sun

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®

Orthogonal Grid Generation, Non-Reflecting Boundary Condition, and Parallel Computation for Fluid Flow

by

Shaopeng Sun

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

COLLEGE OF ENGINEERING AND SCIENCE
LOUISIANA TECH UNIVERSITY

May 2000

LOUISIANA TECH UNIVERSITY

THE GRADUATE SCHOOL

05/01/2000
Date

We hereby recommend that the dissertation prepared under our supervision by Shaopeng Sun entitled Orthogonal Grid Generation, Non-reflecting Boundary Condition and Parallel Computation for Fluid Flow be accepted in partial fulfillment of the requirements for the Degree of Ph.D/ACAM

Supervisor of Dissertation Research

Head of Department
ACAM
Department

Recommendation concurred in:

Advisory Committee

Approved:
Director of Graduate Studies

Approved:
Director of the Graduate School

Dean of the College

GS Form 13
2/97

ABSTRACT

The purpose of this study is to investigate one of the most interesting areas in computational fluid dynamics. The content of this paper is divided into three parts. The first part is the orthogonal grid generation. The second part is the non-reflecting boundary condition in curvilinear coordinates. The third part is parallel computation by Message Passing Interface.

The grid generation method is presented by solving elliptic partial difference equations. The elliptic grid generation method is based on the use of composite mapping, which consists of a non-linear algebraic transformation and an elliptic transformation. The elliptic transformation is based on the Laplace equations for the domains or on the Laplace-Beltrami equations for surfaces. The algebraic transformation maps the computational space one-to-one onto a parameter space, and the elliptic transformation maps the parameter space one-to-one onto the domain or the surfaces. The composition of these two mappings is a differentiable and one-to-one, which has a non-vanish Jacobian. Finally, some complicated test examples are given. Computation results show that the grids generated by these methods are smooth and orthogonal. The grid quality meets our requirement for high accuracy numerical simulation.

Numerical methods for time-dependent hyperbolic systems require time-dependent boundary conditions when the system is solved in a finite domain. The "correct" boundary conditions are crucial in solving such a system. The non-reflecting

boundary conditions based on the Navier-Stokes equations have been derived for curvilinear coordinates. High order scheme is used to discretize the non-reflecting boundary conditions. Several examples of non-reflecting boundary conditions are tested. The results are compared with the reference method based on extrapolation or Rieman invariants. It is found that this non-reflecting boundary condition is much more accurate and effective than the tradition methods used to impose boundary conditions.

A parallel spatial direct numerical simulation code is developed to simulate the spatial evolving disturbances associated with the laminar-to-turbulent transition in a compressible boundary layer. MPI (Message Passing Interface) is employed to parallelize all processes for a distributed memory parallel computer. Explicit time-stepping is used in the DNS code on IBM/SP2 to simulate the flow transition. The machine-dependent phenomenon, which is always being considered as a problem for parallel computation, is successfully avoided. A fundamental breakdown on a flat plate boundary layer transition at Mach 0.5 is then studied using this code. The results demonstrate the optimistic future of MPI to direct numerical simulation.

APPROVAL FOR SCHOLARLY DISSEMINATION

The author grants to the Prescott Memorial Library of Louisiana Tech University the right to reproduce, by appropriate methods, upon request, any or all portions of this Dissertation. It is understood that "proper request" consists of the agreement, on the part of the requesting party, that said reproduction is for his personal use and that subsequent reproduction will not occur without written approval of the author of this Dissertation. Further, any portions of the Dissertation used in books, papers, and other works must be appropriately referenced to this Dissertation.

Finally, the author of this Dissertation reserves the right to publish freely, in the literature, at any time, any or all portions of this Dissertation.

Author Xiaopeng Sun

Date 05/01/00

GS Form 14
2/97

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to take this opportunity to thank Professor Chaoqun Liu for his guidance during my graduate studies at Louisiana Tech University. I thank him for his financial support which provided me the opportunity to study at Louisiana Tech University. Many thanks for his office advice, patience, and effort in correcting the manuscript of this thesis.

I would like to express my gratitude and sincere appreciation to Dr. Richard Greechie at Louisiana Tech University and Professor Zuosheng Yang at Nanjing University of Aeronautics and Astronautics (P.R. China) for their encouragement and guidance through my graduate studies.

I would also like to thank Dr. Zhining Liu, Dr. Hua Shan, and Dr. Li Jiang for their great help and contribution to my development. I significantly benefited from their precious advice and suggestions.

Thanks also goes to my committee members, Dr. Alley Butler, Dr. Raja Nassar, and Dr. Ben Choi, for their support and suggestions.

I also like to take this chance to express my special thanks to AFOSR for their support by allowing me to use their CRAY and IBM/SP2 computers.

Finally, I dedicate this thesis to my wife Min Xu and my family. Without their love and support, this thesis could not have been completed.

CHAPTER 1

Orthogonal Grid Generation

Abstract

The present elliptic grid generation method is based on the use of composite mapping which consists of a non-linear algebraic transformation and an elliptic transformation. The elliptic transformation is based on solving the Laplace equations for the domains or the Laplace-Beltrami equations for surfaces. The algebraic transformation maps the computational space one-to-one onto a parameter space; then the elliptic transformation maps the parameter space one-to-one onto the domain or the surfaces. The composition of these two mappings is a differentiable and one-to-one and has a non-vanishing Jacobian. Finally, some complicated test examples are given. Computation results show that the grids generated by these methods are smooth and orthogonal. The grid quality meets our requirement for high accuracy numerical simulation.

1.1. Introduction

In past decades, numerical generation of curvilinear coordinate systems has provided the key to the development of finite difference solutions of partial differential equations on regions with arbitrarily shaped boundaries. This system is an essential part of most numerical solutions of partial differential equations on regions with general boundaries. The system is classified into three basic classes: algebraic systems, conformal

transformation, and partial differential equation systems in which the coordinates are the solutions of these equations.

Algebraic grid generation is the fastest procedure among these classes. It generates the grid by propagating the boundary toward the interior domain and specifies the point distribution in the interior of the domain. The algebraic procedures allow the explicit control of the grid point distribution which is basically an interpolation among boundaries and intermediate surfaces in the field. This method is particularly attractive for use with interactive graphics because grids can be produced quickly. But, this method is suitable only for a simple geometry. For a complex geometry, it will be very difficult for this method to generate a smooth and non-folding grid.

Conformal mapping satisfies Laplace equations with boundary conditions from the Cauchy-Riemann conditions. It uses elementary transformation functions in the complex plane and then generates the coordinate systems about special boundary curves that are contours of the mapping. The Joukowski and Karman airfoils are examples of boundaries that can be treated in this manner. This method can generate an orthogonal grid in the domain. However, conformal mapping has little control over the coordinate system. So, it may not be possible to produce a system that is well adapted to the physical solution to be performed. The general 2-D configuration can be treated very well. But, the point distribution on the boundaries cannot be specified. Moreover, the internal structure cannot be controlled. The only real benefit from conformal mapping is the simplicity of the transformed partial differential equations. Meanwhile, the possibility of 3-D grid generation by conformal mapping is still questionable.

For partial differential equation systems, the equations can be either elliptic or

hyperbolic. Among them, elliptic grid generation system is the most popular method. The pioneer work is given by J. Thompson. His system is known as the system of second-order elliptic partial grid generation equations. The elliptic system produces the smoothest coordinates for general boundary point distributions. This method takes advantage of the smoothing tendencies inherited from elliptic operators that produce a relatively smooth coordinate system regardless of the boundary. Because the elliptic generating systems are based on partial differential equations, the application requires only the writing of the equations in the physical domain. With the introduction of composite mapping and the source term, there is great flexibility in this method to control the interior grid point distribution and the orthogonality of the grid. Practical problems, whether 2-D or 3-D, can be treated by this method very well.

The computational grid is an integral part of any numerical scheme; the grid can affect both the accuracy and the convergence rate of the numerical solution. Several decades of experience have led researchers in computational fluid dynamics to regard certain grid properties as favorable. The properties are generally associated with high-quality grids including orthogonal grid cells, smooth grid lines, smooth variation of grid cell volumes, and low-aspect-ratio grid cells. In addition, the grid should be clustered in regions where large numerical errors are expected. A grid that does not possess one or all of these properties may yield an unsatisfactory solution.

In our work, composite mapping was used to generate the 2-D, surface and the 3-D grid. This composite mapping is one-to-one and differentiable. Two steps are used for composite mapping: from the computation space to the parameter space by an algebraic transformation and from the parameter space to the physical space by solving the elliptic

partial differential equations. By the property of elliptic equations systems, good orthogonality and smoothness are obtained. In this chapter, our work is divided into three parts: 1) 2-D grid generation; 2) surface grid generation; 3) 3-D grid generation. Finally, results generated by these methods are shown and analyzed.

1.2. Two-dimensional grid generation

We first consider a simply connected bounded domain D in a 2-D space with Cartesian coordinates $\bar{x} = (x, y)^T$. Suppose that D is bounded by four edges E_1, E_2, E_3, E_4. Let (E_1, E_2) and (E_3, E_4) be the two pairs of opposite edges as show in Fig.1.1

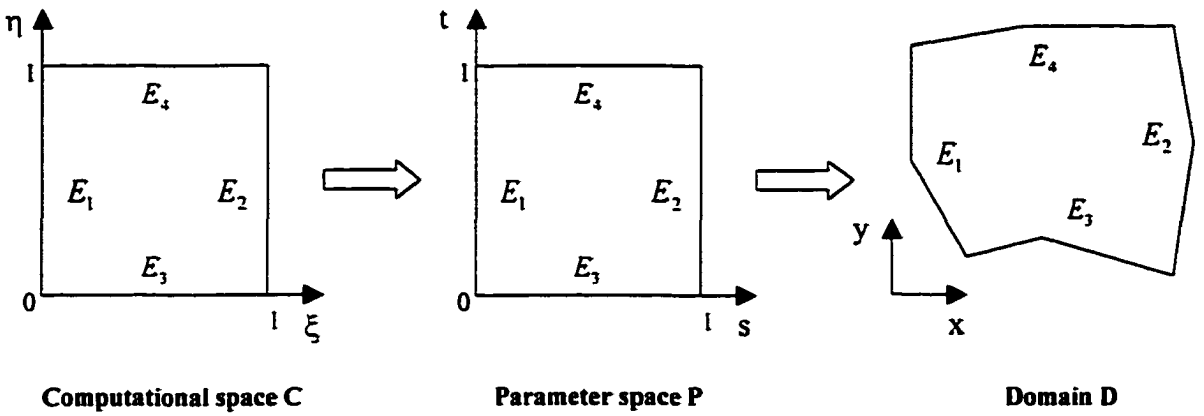


Fig.1.1 Transformation from the computational space C to the physical domain D

We define the computational space C as the unit square in 2-D space with Cartesian coordinates $\bar{\xi} = (\xi, \eta)^T$. Assume that a mapping $\bar{x} : \partial C \mapsto \partial D$ is prescribed which maps the boundary of C one-to-one onto the boundary of D. This mapping defines the boundary grid point distribution. Assume that

- ξ=0 at edge E_1 and ξ=1 at edge E_2,
- η=0 at edge E_3 and η=1 at edge E_4.

We hope to construct a mapping $\bar{x}: \partial C \mapsto \partial D$ which satisfies the boundary conditions and is a differentiable one-to-one mapping. Furthermore, we require that the interior grid point distribution is a good distributation.

A natural mapping $\bar{x}: \partial C \mapsto \partial D$ exists which meets these requirements. This mapping will be the composition of an algebraic transformation and an elliptic transformation based on the Laplace equations. The algebraic transformation is a differentiable one-to-one mapping from the computational space C onto a parameter space P. The parameter space is also a unit square. We will see below that the algebraic transformation will only depend on the prescribed boundary grid point distribution at the four boundaries and is a differentiable one-to-one mapping from the computational space C onto a parameter space P. The elliptic transformation is used to transform from the parameter space P onto the domain D. Because of the elliptic transformation, the transformation is still one-to-one mapping. The transformation will depend only on the shape of the domain D and is independent of the prescribed boundary grid point distribution. The elliptic transformation may be considered as a property of the domain D. The composition of these two mappings defines the interior grid point distribution and is a differentiable one-to-one mapping from the computational domain C onto the physical domain D.

The parameter space P can be described with Cartesian coordinates $\bar{s} = (s,t)^T$. In 2-D space, P is a unit square. The parameters s and t should have the following

properties:

- s=0 at edge E_1 and s=1 at edge E_2,
- s is the normalized arc-length along edge E_3 and E_4.
- t=0 at edge E_3 and t=1 at edge E_4.
- t is the normalized arc-length along edges E_1 and E_2.

The transformation $\vec{s}:\partial D \mapsto \partial P$ is defined based on these requirements. In the interior of D, t and s are required to be harmonic functions of x and y. Therefore, they satisfy the Laplace equations:

$$\Delta s = \frac{\partial^2 s}{\partial x^2} + \frac{\partial^2 s}{\partial y^2} = 0 \tag{1.2.1}$$

$$\Delta t = \frac{\partial^2 t}{\partial x^2} + \frac{\partial^2 t}{\partial y^2} = 0 \tag{1.2.2}$$

The above two Laplace equations, together with the above specified boundary conditions, define the mapping $\vec{s}:\partial D \mapsto \partial P$. It should be noted that this mapping depends only on the shape of the domain D and is independent of the prescribed boundary grid point distribution. By interchanging the dependent and independent variables, a non-linear elliptic partial differential equation can be derived for $\vec{x}:\partial P \mapsto \partial D$. In this way, we have to solve the non-linear elliptic boundary value problem in P to define this mapping. This mapping defines the elliptic transformation. This mapping is also differentiable and one-to-one mapping.

The algebraic transformation should be a differentiable one-to-one mapping from the computational space C onto the parameter space P. Because $\vec{x}:\partial C \mapsto \partial D$ is prescribed and $\vec{x}:\partial P \mapsto \partial D$ is defined as above, it follows that $\vec{s}:\partial C \mapsto \partial P$ is also

defined.

From the requirements we mentioned, the mapping between computational space (ξ,η) and parameter space (s,t) should satisfy

$$s(0,\eta) = 0,\ s(1,\eta) = 1,$$

$$s(\xi,0) = s_{E_1}(\xi),\ s(\xi,1) = s_{E_4}(\xi) \tag{1.2.3}$$

and

$$t(\xi,0) = 0,\ t(\xi,1) = 1,$$

$$t(0,\eta) = t_{E_1}(\xi),\ t(1,\eta) = t_{E_2}(\xi) \tag{1.2.4}$$

where the function $s_{E_1}(\xi), s_{E_4}(\xi)$ and $t_{E_1}(\xi), t_{E_2}(\xi)$ are monotonically increasing. They are defined by the boundary grid point distributation.

Define the two algebraic equations for the mapping $\bar{s} : \partial C \mapsto \partial P$ as

$$s = s_{E_1}(\xi)(1-t) + s_{E_4}(\xi)t \tag{1.2.5}$$

$$t = t_{E_1}(\eta)(1-s) + s_{E_2}(\xi)s \tag{1.2.6}$$

From the above equations, we can see that a coordinate line ξ=const is mapped to the parameter space P as a straight line: s is a linear function of t. In the same way, the line η=const is mapped to P as a straight line: t is a linear function of s. For given values of ξ and η, the corresponding s and t values are found to be the intersection point of the two straight lines. This is the reason why the above mapping is called "algebraic straight line transformation" because it uses straight lines in the parameter space P. This mapping is also a differentiable one-to-one mapping since it has the positiveness of the Jacobian: $s_\xi t_\eta - s_\eta t_\xi > 0$.

The algebraic transformation $\bar{s} : \partial C \mapsto \partial P$ and the elliptic transformation

$\bar{x}: \partial P \mapsto \partial D$ are differentiable and one-to-one. Therefore, the composite mapping $\bar{x}: \partial C \mapsto \partial D$ which is defined as $x(\xi) = x(s(\xi))$ is also differentiable and one-to-one. Actually, because of the properties of the basic mapping, we expect the interior grid point distributation to be a good reflection of the boundary grid point distribution.

The composite mapping satisfies the elliptic system of Poisson equations. This can be seen from Warsi and Thompson[1982, 1984]. The problem here is that equations (1.2.1) and (1.2.2) are not useful because they contain control functions which depend on the derivatives of the inverse mapping $\xi: \partial P \mapsto \partial C$. Therefore, we need to do extra work to obtain the expressions of these control functions which depend only on the derivatives of the mapping $\bar{s}: \partial C \mapsto \partial P$.

We define two covariant base vectors as follows: $\bar{a}_1 = \bar{x}_\xi$ and $\bar{a}_2 = \bar{x}_\eta$.

The two contravariant base vectors are $\bar{a}^1$ and $\bar{a}^2$. They are defined according to the rules

$$(\bar{a}^i, \bar{a}_j) = \delta^i_j \tag{1.2.7}$$

where i, j =1,2. δ^i_j is the kronecker symbol.

The determinant J^2 of the covariant metric tensor is $J^2 = a_{11}a_{22} - a_{12}^2$.

Now, consider an arbitrary function $\phi = \phi(\xi, \eta)$. Then ϕ is also defined in domain D and the Laplace of ϕ is expressed as follows:

$$\Delta\phi = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = \frac{1}{J}\left\{\left(Ja^{11}\phi_\xi + Ja^{12}\phi_\eta\right)_\xi + \left(Ja^{12}\phi_\xi + Ja^{22}\phi_\eta\right)_\eta\right\} \tag{1.2.8}$$

Considering the special cases $\phi = \xi$ and $\phi = \eta$. The above equation yields

$$\Delta\xi = \frac{1}{J}\left\{\left(Ja^{11}\right)_{\xi} + \left(Ja^{12}\right)_{\eta}\right\}$$

$$\Delta\eta = \frac{1}{J}\left\{\left(Ja^{12}\right)_{\xi} + \left(Ja^{22}\right)_{\eta}\right\} \tag{1.2.9}$$

Hence, the Laplace of ϕ can be expressed as

$$\Delta\phi = a^{11}\phi_{\xi\xi} + 2a^{12}\phi_{\xi\eta} + a^{22}\phi_{\eta\eta} + \Delta\xi\phi_{\xi} + \Delta\eta\phi_{\eta} \tag{1.2.10}$$

Substituting $\phi = s$ and $\phi = t$, respectively, in the above equation yields:

$$\Delta s = a^{11}s_{\xi\xi} + 2a^{12}s_{\xi\eta} + a^{22}s_{\eta\eta} + \Delta\xi s_{\xi} + \Delta\eta s_{\eta} \tag{1.2.11}$$

$$\Delta t = a^{11}t_{\xi\xi} + 2a^{12}t_{\xi\eta} + a^{22}t_{\eta\eta} + \Delta\xi t_{\xi} + \Delta\eta t_{\eta} \tag{1.2.12}$$

Because of the requirement that s and t are harmonic in the domain D, we have $\Delta s = 0$ and $\Delta t = 0$. Using these equations, we obtain the equations for the Laplacian of ξ and η,

$$\begin{pmatrix}\Delta\xi\\ \Delta\eta\end{pmatrix} = a^{11}P_{11} + 2a^{12}P_{12} + a^{22}P_{22} \tag{1.2.13}$$

where

$$P_{11} = -T^{-1}\begin{pmatrix}s_{\xi\xi}\\ t_{\xi\xi}\end{pmatrix},\ P_{12} = -T^{-1}\begin{pmatrix}s_{\xi\eta}\\ t_{\xi\eta}\end{pmatrix},\ P_{22} = -T^{-1}\begin{pmatrix}s_{\eta\eta}\\ t_{\eta\eta}\end{pmatrix} \tag{1.2.14}$$

and the matrix T is defined as

$$T = \begin{pmatrix}s_{\xi} & s_{\eta}\\ t_{\xi} & t_{\eta}\end{pmatrix} \tag{1.2.15}$$

The coefficients of the vectors $P_{11} = \left(P_{11}^{1}, P_{11}^{2}\right)^{T}$, $P_{12} = \left(P_{12}^{1}, P_{12}^{2}\right)^{T}$, $P_{22} = \left(P_{22}^{1}, P_{22}^{2}\right)^{T}$ are the control functions. The six control functions are completely defined and easily computed for a given algebraic transformation, $s = s(\xi)$.

Finally, substitution of $\phi = \bar{x}$ in equation (1.2.10) yields

$$\Delta\bar{x} = a^{11}\bar{x}_{\xi\xi} + 2a^{12}\bar{x}_{\xi\eta} + a^{22}\bar{x}_{\eta\eta} + \Delta\xi\bar{x}_{\xi} + \Delta\eta\bar{x}_{\eta} \tag{1.2.16}$$

Substituting equation (1.2.16) into equation (1.2.13) and using the fact that $\Delta\bar{x} = 0$, we obtain the following Poisson grid generation system.

$$\begin{aligned} &a^{11}\bar{x}_{\xi\xi} + 2a^{12}\bar{x}_{\xi\eta} + a^{22}\bar{x}_{\eta\eta} \\ &+\left(a^{11}P_{11}^{1} + 2a^{12}P_{12}^{1} + a^{22}P_{22}^{1}\right)\bar{x}_{\xi} + \left(a^{11}P_{11}^{2} + 2a^{12}P_{12}^{2} + a^{22}P_{22}^{2}\right)\bar{x}_{\eta} = 0 \end{aligned} \tag{1.2.17}$$

Using the well-known expression for the contravariant metric tensor components, we have the following:

$$\begin{aligned} J^2a^{11} &= a_{22} = \left(\bar{x}_{\eta}, \bar{x}_{\eta}\right) \\ J^2a^{12} &= -a_{12} = -\left(\bar{x}_{\xi}, \bar{x}_{\eta}\right) \\ J^2a^{22} &= a_{11} = \left(\bar{x}_{\xi}, \bar{x}_{\xi}\right) \end{aligned} \tag{1.2.18}$$

The Possion grid generation system defined by equation (1.2.17) can be simplified by multiplication with J^2. Therefore, we obtain

$$\begin{aligned} &\theta^{11}\bar{x}_{\xi\xi} + 2\theta^{12}\bar{x}_{\xi\eta} + \theta^{22}\bar{x}_{\eta\eta} \\ &+\left(\theta^{11}P_{11}^{1} + 2\theta^{12}P_{12}^{1} + \theta^{22}P_{22}^{1}\right)\bar{x}_{\xi} + \left(\theta^{11}P_{11}^{2} + 2\theta^{12}P_{12}^{2} + \theta^{22}P_{22}^{2}\right)\bar{x}_{\eta} = 0 \end{aligned} \tag{1.2.19}$$

with

$$\theta^{11} = \left(\bar{x}_{\eta}, \bar{x}_{\eta}\right),\ \theta^{12} = -\left(\bar{x}_{\xi}, \bar{x}_{\eta}\right),\ \theta^{22} = \left(\bar{x}_{\xi}, \bar{x}_{\xi}\right) \tag{1.2.20}$$

These equations, together with the expressions for the control functions P_{ij}^{k} given by equation (1.2.15), form the 2-D grid generation system.

1.3. Surface grid generation on minimal surface

Grid generation on a minimal surface is, in fact, a straightforward extension of the

grid generation in a domain of 2-D space. In 3-D space, four connected curved edges define a surface. A minimal surface is defined as a surface bounded by these edges with zero mean curvature. Therefore, the shape of the minimal surface is a soap film bounded by four curved edges.

As in the 2-D case, the situation is now in a 3-D physical space R^3 with Cartesian coordinates $\bar{x} = (x, y, z)^T$. Again, we define four curved edges in R^3 as E_1, E_2, E_3, E_4. Let (E_1, E_2) and (E_3, E_4) be the two pairs of opposite edges as shows in Fig.1.2.

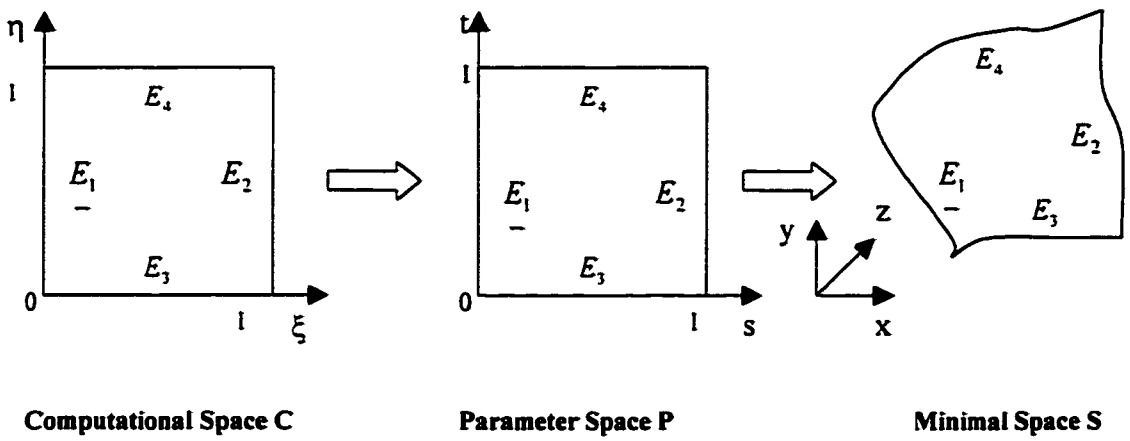


Fig.1.2 Transformation from the computational space to a minimal surface

The parameter space P is the unit square in 2-D space with Cartesian coordinates $\bar{s} = (s, t)^T$. s and t are required to satisfy the following properties:

- s=0 at edge E_1 and s=1 at edge E_2,
- s is the normalized arc-length along the edge E_3 and E_4.
- T=0 at edge E_3 and t=1 at edge E_4.
- T is the normalized arc-length along edges E_1 and E_2.

Furthermore, we require that

$$\Delta s = 0 \tag{1.3.1}$$

$$\Delta t = 0 \tag{1.3.2}$$

$$H = 0 \tag{1.3.3}$$

where Δ is the Laplace-Beltrami operator for the surface and H is the mean curvature.

These three requirements, together with the specified boundary conditions, define a unique mapping $\bar{x} : \partial P \mapsto \partial R^3$. By this definition, the mapping is a differentiable one-to-one mapping. The shape of the surface as defined by this mapping is a minimal surface because of the requirement that the mean curvature H is zero. The parametrization of the surface is defined by equations (1.3.1) and (1.3.2). The surface defined by this minimal space is independent of the specified boundary grid point distribution at the four edges as shown later in the results. For the mapping from the computational space to the parameter space, we use the same algebraic transformation as used in the 2-D case. So, the mapping $\bar{s} : \partial C \mapsto \partial P$ is defined by a straight line transformation with equations (1.2.5) and (1.2.6).

Now, we defined all transformations. They are $\bar{s} : \partial C \mapsto \partial P$ which is defined by equations (1.2.5)--(1.2.6), and $\bar{x} : \partial P \mapsto \partial R^3$ by equations (1.3.1) -- (1.3.3). The composite mapping $\bar{x} : \partial C \mapsto \partial R^3$ is defined as $\bar{x} = \bar{x}(\bar{s}(\xi))$ and describes the interior grid point distribution on the minimal surface. This composite mapping is still differentiable and one-to-one.

The Laplace-Beltrami operator applied on $\bar{x}$ satisfies the famous relation as

$$\Delta \bar{x} = 2H\bar{n} \tag{1.3.4}$$

where the mean curvature H is defined as

$$H = \frac{1}{2}(a^{11}\bar{x}_{\xi\xi} + 2a^{12}\bar{x}_{\xi\eta} + a^{22}\bar{x}_{\eta\eta}, \bar{n}) \tag{1.3.5}$$

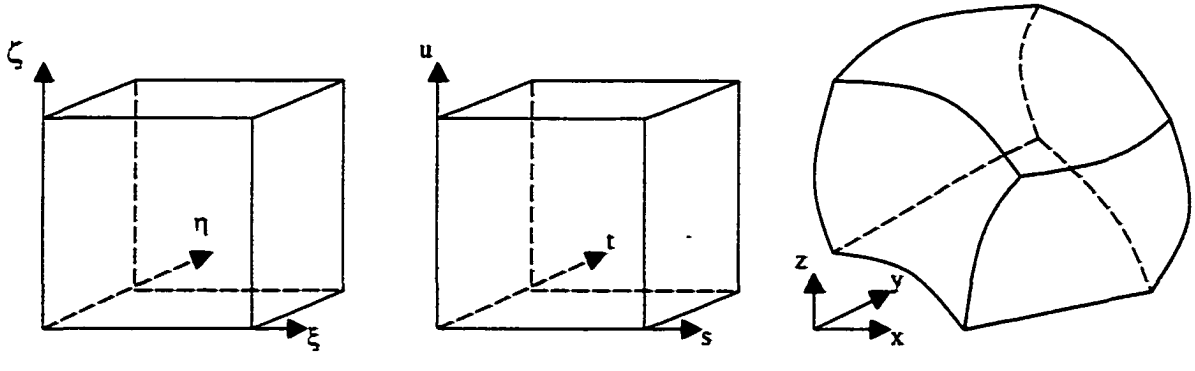
Because of using the minimal surface, the mean curvature is zero. Then we have

$$\Delta\bar{x} = 0 \tag{1.3.6}$$

Following the same derivation as described in the last section, we obtain exactly the same non-linear system of elliptic partial differential equations as the 2-D grid generation system. The only difference compared to the 2-D case is now $\bar{x} = (x, y, z)^T$ instead of $\bar{x} = (x, y)^T$.

1.4. Three-dimensional grid generation

The 2-D grid generation method can be extended to 3-D grid generation. In the 3-D space with Cartesian coordinates $\bar{x} = (x, y, z)^T$, domain D is bounded by six faces F_1, F_2, F_3, F_4, F_5, F_6. (F_1, F_2), (F_3,F_4), and (F_5,F_6) are the three pairs of opposite faces. On these faces, there are twelve edges labeled as { E_i, i=1...12}. The positions of the face and edge are shown in Fig.1.3



Computational space C Parameter space P Domain D

Fig.1.3 Transformation from the computational space to the physical domain D

Let the computational space C be the unit cube in 3-D space with Cartesian

coordinate $\bar{\xi} = (\xi, \eta, \zeta)^T$. Assume that a mapping $x : \partial C \mapsto \partial D$ is prescribed which maps the boundary of C one-to-one onto the boundary of D. This mapping defines the boundary grid point distributation. It meets the following requirements:

- ξ=0 on face F1 and ξ=1 on face F2;
- η=0 on face F3 and η=1 on face F4;
- ζ=0 on face F5 and ζ=1 on face F6.

As in the 2-D case, we introduce the parameter space P as the unit cube 3-D space with Cartesian coordinates $\bar{s} = (s, t, u)^T$. We also require that the parameters satisfy the following properties:

- s=0 on face F1 and s=1 on face F2;
- t=0 on face F3 and t=1 on face F4;
- u=0 on face F5 and u=1 on face F6
- s is the normalized arc-length at edge E1, E2, E3, E4;
- t is the normalized arc-length at edge E5, E6, E7, E8;
- u is the normalized arc-length at edge E9, E10, E11, E12.

The coordinates (s, t, u) are defined at all twelve edges of the domain D. The twelve edge functions $s_{E_1} \ldots s_{E_{12}}$ are monotonically increasing and defined by the specified boundary point distribution at all twelve edges.

As in the 2-D case, the algebraic mapping from the computational space to the parameter space $s : C \mapsto P$ is defined as

$$s = s_{E_1}(\xi)(1-t)(1-u) + s_{E_2}(\xi)t(1-u) + s_{E_3}(\xi)(1-t)u + s_{E_4}(\xi)tu \qquad (1.4.1)$$

$$t = t_{E_5}(\eta)(1-s)(1-u) + t_{E_6}(\eta)s(1-u) + t_{E_7}(\eta)(1-s)u + t_{E_8}(\eta)su \qquad (1.4.2)$$

$$u = u_{E_9}(\zeta)(1-s)(1-t) + u_{E_{10}}(\zeta)s(1-t) + u_{E_{11}}(\zeta)(1-s)t + u_{E_{12}}(\zeta)st \qquad (1.4.3)$$

this mapping depends only on the boundary grid point distribution at these twelve edges of the domain D. From the above definition, we can see that the algebraic mapping is differentiable and one-to-one.

Because $x:\partial C \mapsto \partial D$ is prescribed and $s:\partial C \mapsto \partial P$ is defined by the algebraic bilinear transformation, (s,t,u) coordinates are specified at all boundaries of the domain D, including the interior of the six faces F1,...,F6. Require that (s,t,u) are harmonic functions in the interior of the domain D; then

$$\Delta s = \frac{\partial^2 s}{\partial x^2} + \frac{\partial^2 s}{\partial y^2} + \frac{\partial^2 s}{\partial z^2} = 0$$

$$\Delta t = \frac{\partial^2 t}{\partial x^2} + \frac{\partial^2 t}{\partial y^2} + \frac{\partial^2 t}{\partial z^2} = 0 \qquad (1.4.4)$$

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = 0$$

Therefore, a linear elliptic boundary value problem defines the mapping $s:\partial D \mapsto \partial P$. The mapping $s:\partial D \mapsto \partial P$ is one-to-one, and the inverse mapping $s:\partial P \mapsto \partial D$ exists and is differentiable.

The algebraic transformation $s:\partial C \mapsto \partial P$ and the elliptic transformation $x:\partial P \mapsto \partial D$ are one-to-one mappings. Then, the composite mapping $x:\partial C \mapsto \partial D$, defined as x=x(s(ξ)), is differentiable and one-to-one. The composite mapping $x:\partial C \mapsto \partial D$ satisfies the elliptic Poisson system with control functions defined by the algebraic mapping $s:\partial C \mapsto \partial P$. Actually, this 3-D grid generation system is a straightforward extension of the 2-D system. Therefore, we follow the 2-D grid

generation method to derive the 3-D grid generation method.

Define the three covariant base vectors as

$$\bar{a}_1 = \bar{x}_\xi,\ \bar{a}_2 = \bar{x}_\eta,\ \bar{a}_3 = \bar{x}_\zeta \tag{1.4.5}$$

The covariant metric tensor components are the following:

$$a_{ij} = (a_i, a_j), i = \{1,2,3\}, j = \{1,2,3\} \tag{1.4.6}$$

The three contravariant base vectors are defined to be the following:

$$(a^i, a_j) = \delta^i_j, i = \{1,2,3\}, j = \{1,2,3\} \tag{1.4.7}$$

Let J^2 be the determinant of the covariant metric tensor. For the arbitrary function $\phi = \phi(\xi,\eta,\zeta)$, the Laplace operator for ϕ can be written as

$$\Delta\phi = \frac{1}{J}\begin{Bmatrix}\left(Ja^{11}\phi_\xi + Ja^{12}\phi_\eta + Ja^{13}\phi_\zeta\right)_\xi + \left(Ja^{12}\phi_\xi + Ja^{22}\phi_\eta + Ja^{23}\phi_\zeta\right)_\eta \\ + \left(Ja^{13}\phi_\xi + Ja^{23}\phi_\eta + Ja^{33}\phi_\zeta\right)_\zeta\end{Bmatrix}$$

Substituting $\phi = \xi$, $\phi = \eta$, $\phi = \zeta$, we obtain the expression for $\Delta\xi$, $\Delta\eta$, $\Delta\zeta$ as

$$\begin{aligned}\Delta\xi &= \frac{1}{J}\left\{\left(Ja^{11}\right)_\xi + \left(Ja^{12}\right)_\eta + \left(Ja^{13}\right)_\zeta\right\} \\ \Delta\eta &= \frac{1}{J}\left\{\left(Ja^{12}\right)_\xi + \left(Ja^{22}\right)_\eta + \left(Ja^{23}\right)_\zeta\right\} \\ \Delta\zeta &= \frac{1}{J}\left\{\left(Ja^{13}\right)_\xi + \left(Ja^{23}\right)_\eta + \left(Ja^{33}\right)_\zeta\right\}\end{aligned} \tag{1.4.8}$$

Substituting (1.4.8) by (1.4.7), we have

$$\Delta\phi = a^{11}\phi_{\xi\xi} + 2a^{12}\phi_{\xi\eta} + 2a^{12}\phi_{\xi\zeta} + a^{22}\phi_{\eta\eta} + 2a^{23}\phi_{\eta\zeta} + a^{22}\phi_{\zeta\zeta} + \Delta\xi\phi_\xi + \Delta\eta\phi_\eta + \Delta\zeta\phi_\zeta \tag{1.4.9}$$

Letting $\phi = \phi(s,t,u)$ in the above equation and using the requirement that s, t, and u are harmonic in domain D, i.e., $\Delta s = 0, \Delta t = 0, \Delta u = 0$, we obtain the following

equations:

$$\begin{pmatrix}\Delta\xi\\ \Delta\eta\\ \Delta\zeta\end{pmatrix} = a^{11}P_{11} + 2a^{12}P_{12} + 2a^{13}P_{13} + a^{22}P_{22} + 2a^{23}P_{23} + a^{33}P_{33} \qquad (1.4.10)$$

where

$$P_{11} = -T^{-1}\begin{pmatrix}s_{\xi\xi}\\ t_{\xi\xi}\\ u_{\xi\xi}\end{pmatrix},\ P_{12} = -T^{-1}\begin{pmatrix}s_{\xi\eta}\\ t_{\xi\eta}\\ u_{\xi\eta}\end{pmatrix},\ P_{13} = -T^{-1}\begin{pmatrix}s_{\xi\zeta}\\ t_{\xi\zeta}\\ u_{\xi\zeta}\end{pmatrix}$$

$$P_{22} = -T^{-1}\begin{pmatrix}s_{\eta\eta}\\ t_{\eta\eta}\\ u_{\eta\eta}\end{pmatrix},\ P_{23} = -T^{-1}\begin{pmatrix}s_{\eta\zeta}\\ t_{\eta\zeta}\\ u_{\eta\zeta}\end{pmatrix},\ P_{33} = -T^{-1}\begin{pmatrix}s_{\zeta\zeta}\\ t_{\zeta\zeta}\\ u_{\zeta\zeta}\end{pmatrix} \qquad (1.4.11)$$

and the matrix T is defined as

$$T = \begin{pmatrix}s_\xi & s_\eta & s_\zeta\\ t_\xi & t_\eta & t_\zeta\\ u_\xi & u_\eta & u_\zeta\end{pmatrix} \qquad (1.4.12)$$

The vectors P11, P12, P13, P22, P23, P33 are the control functions. The coefficients of the control functions are completely defined and can be easily computed for a given algebraic transformation mapping s=s(ξ).

Finally, let $\phi = \bar{x}$ and using the equation $\Delta\bar{x} = 0$, we have

$$\begin{aligned} &a^{11}\bar{x}_{\xi\xi} + 2a^{12}\bar{x}_{\xi\eta} + 2a^{12}\bar{x}_{\xi\zeta} + a^{22}\bar{x}_{\eta\eta} + 2a^{23}\bar{x}_{\eta\zeta} + a^{33}\bar{x}_{\zeta\zeta}\\ &+ \Delta\xi\bar{x}_\xi + \Delta\eta\bar{x}_\eta + \Delta\zeta\bar{x}_\zeta = 0 \end{aligned} \qquad (1.4.13)$$

The final form of the grid generation system can be written as

$$\begin{aligned}
&\theta^{11}\bar{x}_{\xi\xi}+2\theta^{12}\bar{x}_{\xi\eta}+2\theta^{12}\bar{x}_{\xi\zeta}+\theta^{22}\bar{x}_{\eta\eta}+2\theta^{23}\bar{x}_{\eta\zeta}+\theta^{33}\bar{x}_{\zeta\zeta}\\
&+\left(\theta^{11}P_{11}^{1}+2\theta^{12}P_{12}^{1}+\theta^{13}P_{13}^{1}+\theta^{22}P_{22}^{1}+2\theta^{23}P_{23}^{1}+\theta^{33}P_{33}^{1}\right)\bar{x}_{\xi}\\
&+\left(\theta^{11}P_{11}^{2}+2\theta^{12}P_{12}^{2}+\theta^{13}P_{13}^{2}+\theta^{22}P_{22}^{2}+2\theta^{23}P_{23}^{2}+\theta^{33}P_{33}^{2}\right)\bar{x}_{\eta}\\
&+\left(\theta^{11}P_{11}^{3}+2\theta^{12}P_{12}^{3}+\theta^{13}P_{13}^{3}+\theta^{22}P_{22}^{3}+2\theta^{23}P_{23}^{3}+\theta^{33}P_{33}^{3}\right)\bar{x}_{\zeta}=0
\end{aligned} \tag{1.4.14}$$

where

$$\begin{aligned}
&\theta^{11}=a_{22}a_{33}-a_{23}^{2},\ \theta^{12}=a_{13}a_{23}-a_{12}a_{33}\\
&\theta^{13}=a_{12}a_{23}-a_{13}a_{22},\theta^{22}=a_{11}a_{33}-a_{13}^{2}\\
&\theta^{23}=a_{13}a_{12}-a_{11}a_{23},\theta^{33}=a_{11}a_{22}-a_{12}^{2}
\end{aligned} \tag{1.4.15}$$

and

$$\begin{aligned}
&a_{11}=\left(\bar{x}_{\xi},\bar{x}_{\xi}\right),\ a_{12}=\left(\bar{x}_{\xi},\bar{x}_{\eta}\right),\ a_{13}=\left(\bar{x}_{\xi},\bar{x}_{\zeta}\right)\\
&a_{22}=\left(\bar{x}_{\eta},\bar{x}_{\eta}\right),a_{23}=\left(\bar{x}_{\eta},\bar{x}_{\zeta}\right),\ a_{33}=\left(\bar{x}_{\zeta},\bar{x}_{\zeta}\right)
\end{aligned} \tag{1.4.16}$$

3-D grid is generated by solving the elliptic partial differential equations (1.4.14). All related control functions are well defined above.

1.5. Results and analysis

Results of grids in the 2-D domain are shown in Fig. 2-9. All grids are grid-folding free, and the interior grid point distribution is a good reflection of the prescribed boundary grid point distribution. An initial grid is required as the starting solution for the non-linear elliptic Poisson system. The final grid is independent of the initial grid, the quality of the initial grid is unimportant, and severe grid-folding of the initial grid is allowed. In our work, the algebraic method is used to generate this initial grid.

Fig.1.4 shows a region about an ellipse with the long axis located horizonally. The computational accuracy is set to 10^{-6}. In this figure, we can see that the grid in the interior of the domain is smooth. Usually, in computational fluid dynamics, the solution is very sensitive to the grid quality near the object. Therefore, in our test, we specify the orthogonality on the surface of the ellipse. Fig.1.5 shows the detailed grid near the surface of the ellipse. It can be seen clearly that the grid at the elliptic boundary is orthogonal.

In order to test the correctiness of our program, the same ellipse configuration with the long axis located vertically is generated in Fig.1.6. The grid is also very smooth. Because we again specify the orthognality near the ellipse's surface, we drew the detailed grid in Fig. 1.7. We can see that the orthogonality near the surface of ellipse is being well kept.

In the 2-D case, we choose a complicated configuration for test purposes as shown in Fig.1.8. The configuration is a section of the aircraft body. The boundary points are chosen from the reference paper, so it may not be smooth everywhere. But generally, it can be used to show the capibility of this method. This configuration is very difficult for the algebraic method. Our initial grid has severe grid-folding near the boundary of the object which is obtained by using the algebraic grid generation method. After applying this method, we got a very good grid with no grid-folding. Especially at the corner of the boundary, we see that good orthogonality has been well kept. Fig.1.8 shows the grid which is generated by this method. The grid is smooth, and the grid points are nicely distributed in the domain according to the boundary grid distribution. Fig.1.9 shows the details of the concave part of this configuration. Also, as we expected, it reflected boundary point distribution very well and shows that the orthogonality has been well

kept.

Figures 1.10-1.13 show some examples for surface grid generation,. First, the configuration of the flat plate is given to verify the method. The four boundaries are given on a plane. Figure. 1.10 shows that the gererated grid is still a plane and that the grid distribution is a good reflection of the boundary condition.

An example of a grid on a characteristic minimal surface is shown in Fig.1.11. This is the so-called square Scherck surface. The initial algebraic grid can be generated by the algebraic method. The four boundaries are specified. Using the method introduced in Section 1.3, Fig.1.11 shows the generated grid. Fig.1.12 illustrates what happens when the prescribed boundary grid point distribution is changed. This figure shows clearly that the shape of the minimal surface is independent of the prescribed boundary grid point distribution.

Results of 3-D grid generation are shown in Fig.1.13-1.15. The wing configuration of an aircraft is chosen as our test case. The grid is a combination of both C-type and H-type grids. In the y direction, the H-type grid is used, and, in the (x,z) plane, the C-type grid is generated. The whole grid was generated by two blocks. One block is from wing root to wing tip. Another block is from the wing tip to the far boundary in the y direction. The initial grid was generated by the algebraic method. Actually, the initial grid does not affect the final grid. Fig.1.13 shows the boundary of the grid. In order to get a correct result near the boundary layer, the grid points close to the wing are very dense. At the far field, the grid points are distributed sparsely. The surface of the wing and the symmetric plane (x,z) can be seen in detail in Fig.1.14. In Fig.1.15, the plane section (x,z) of the grid is shown. Shown at the root of the wing is the symmetry plane, and in the

middle is the interface between two grid blocks.

The grid is grid-folding free, and the interior grid point distribution is a good reflection of the prescribed boundary grid point distribution at the boundaries.

Generally, the elliptic grid generation method produces very excellent grids in the sense of smoothness, grid point distribution, and regularity. The elliptic grid generation method is based on the composition of an algebraic and elliptic transformation. The elliptic transformation is based on the Laplace equations for domain and the Laplace-Beltrami equations for surfaces. The composite mapping satisfies the familiar grid generation systems of the Poisson equations with control functions specified by the algebraic transformation. Results in 2-D, surface, and 3-D problems show that this grid generation system is very successful in high accuracy computation fluid dynamics.

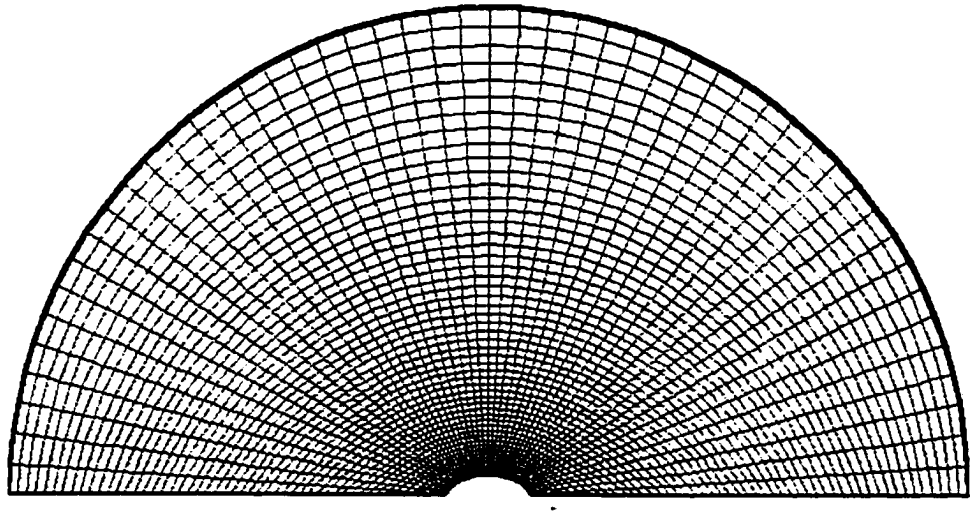


Figure 1.4 2-D grid generation of a half-ellipse

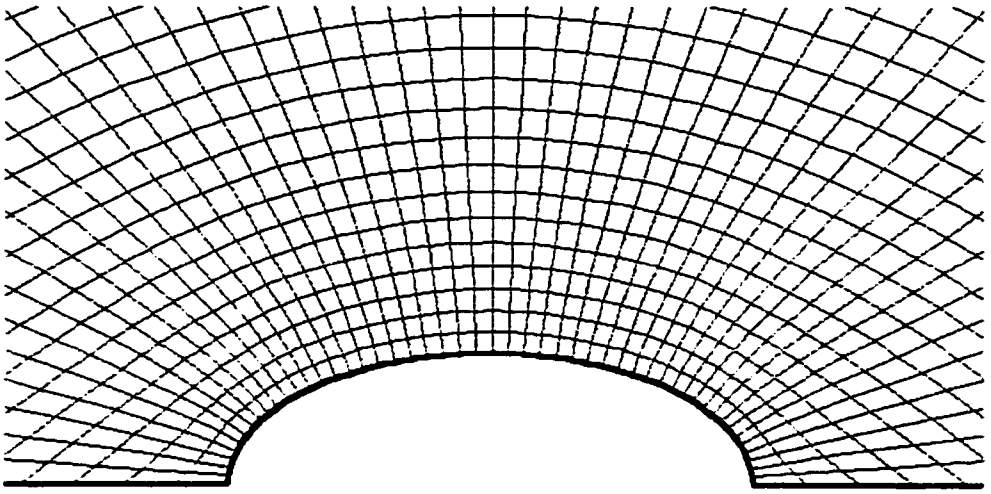


Figure 1.5 Detail of 2-D grid generation of a half-ellipse

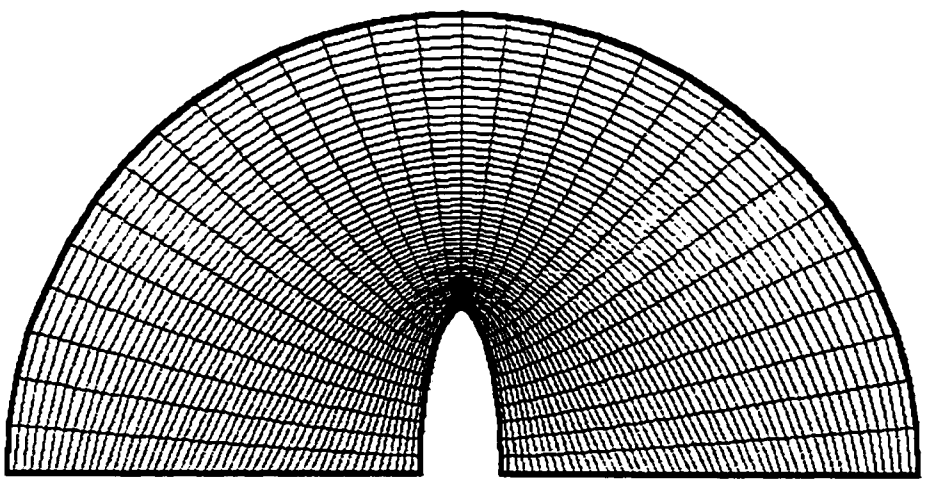


Figure 1.6 2-D grid generation of a half-ellipse

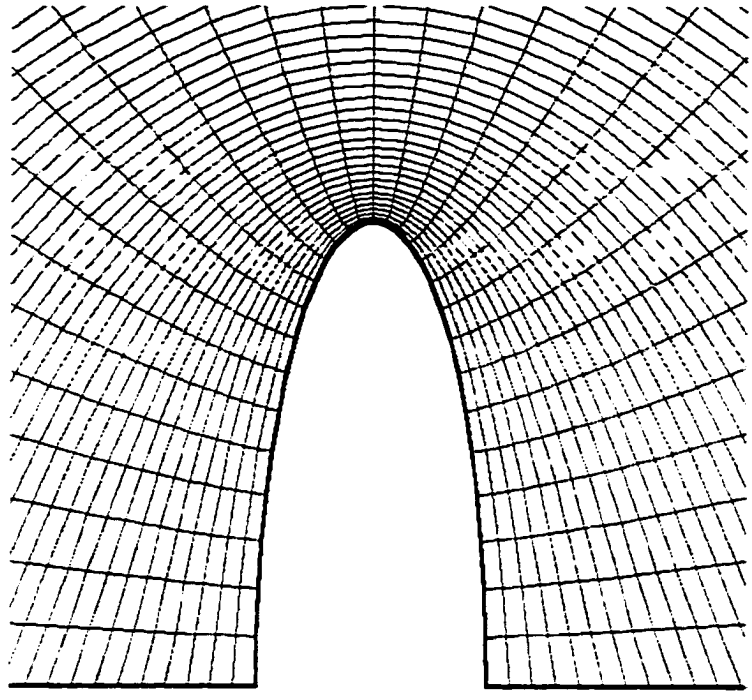


Figure 1.7 Detail of 2-D grid generation of a half-ellipse

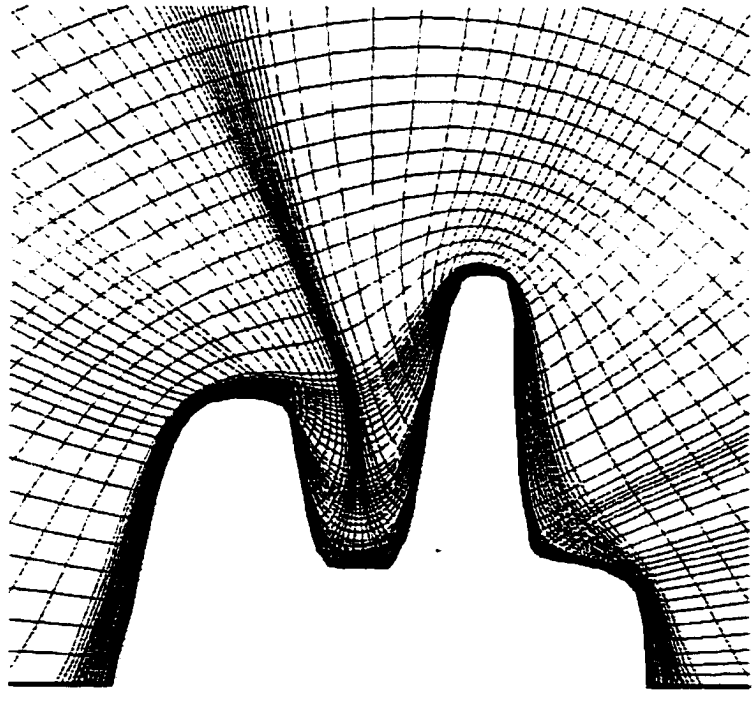


Figure 1.8 2-D grid generation for a complex boundary

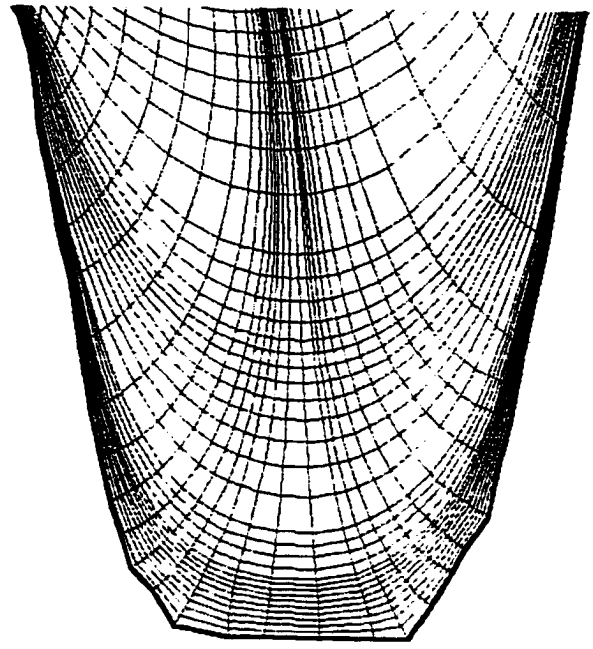


Figure 1.9 Grid generation of a concave boundary

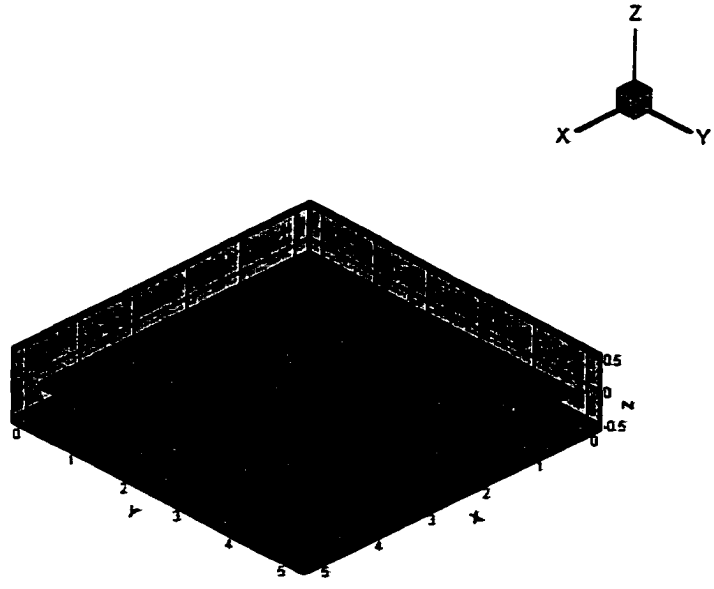


Figure 1.10 Surface grid generation for a flat plate

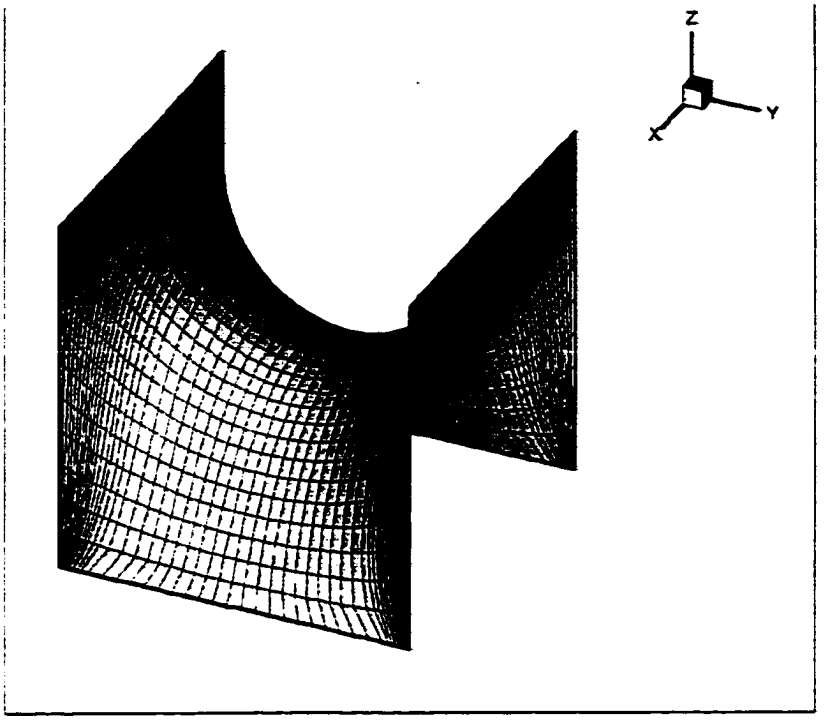


Figure 1.11 Minimal surface of a square scherk surface (1)

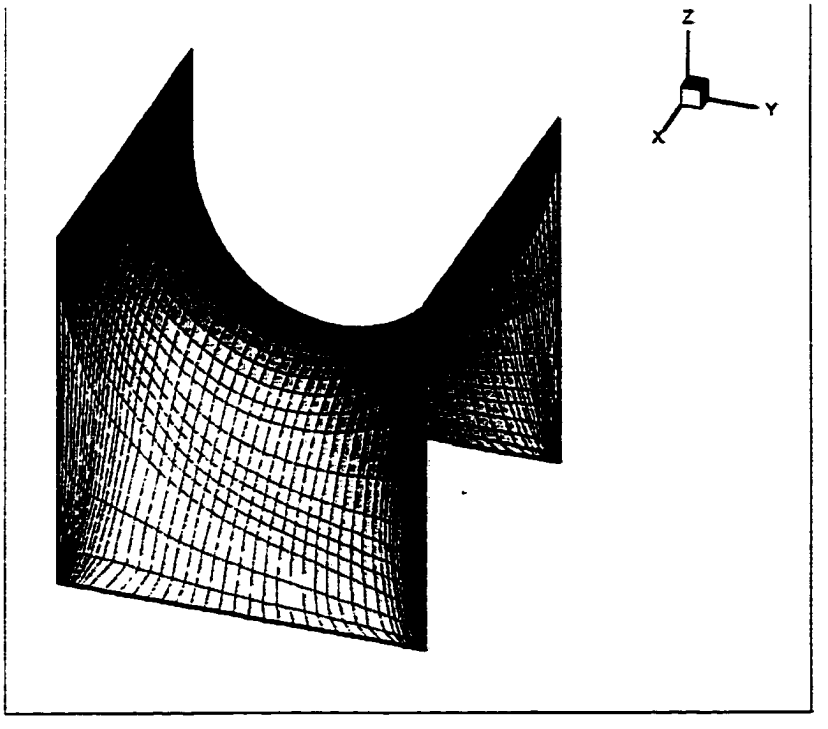


Figure 1.12 Minimal surface of a square scherk surface (2)

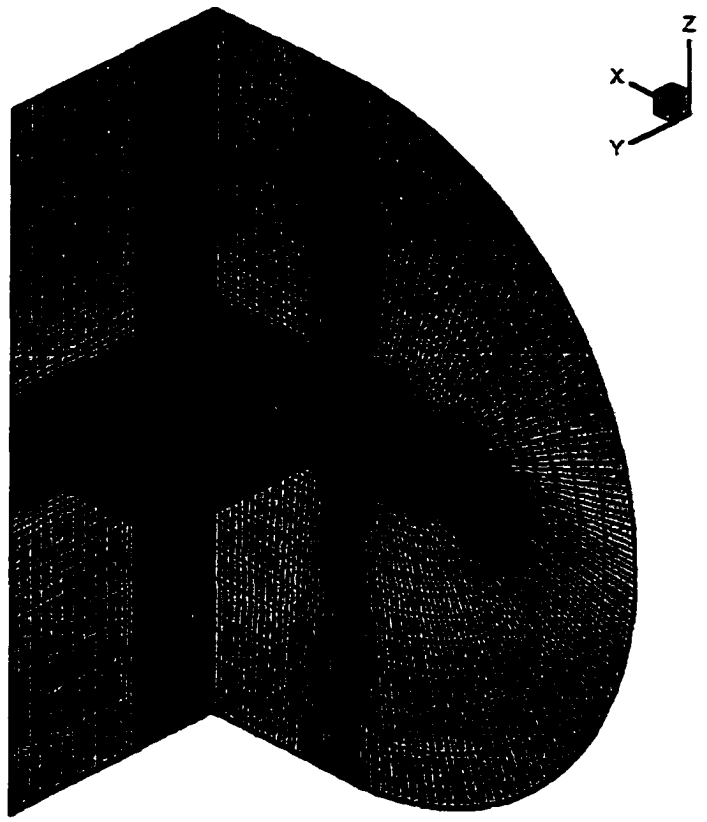


Figure 1.13 3-D grid generation of a wing configuration

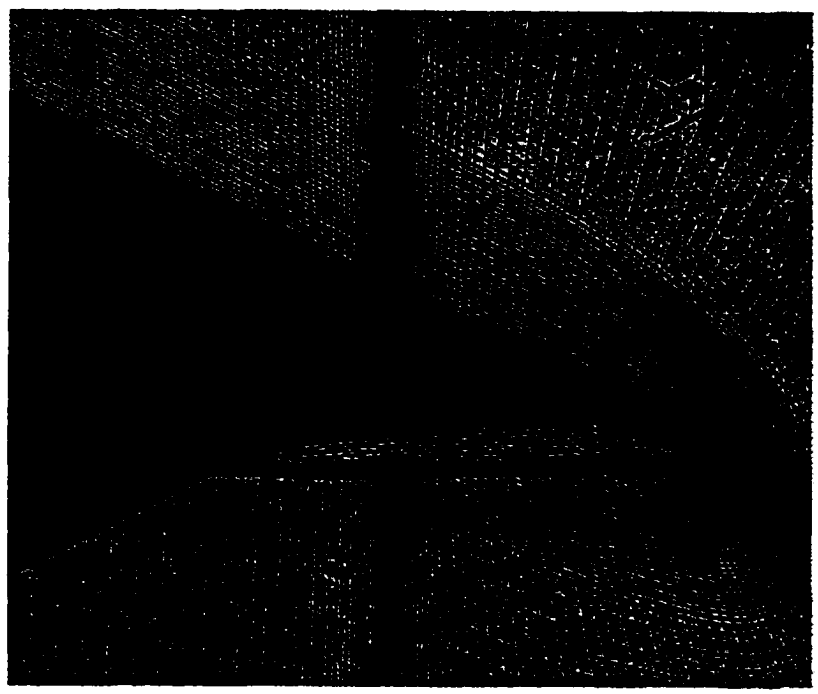


Figure 1.14 Detail of 3-D grid generation of a wing configuration

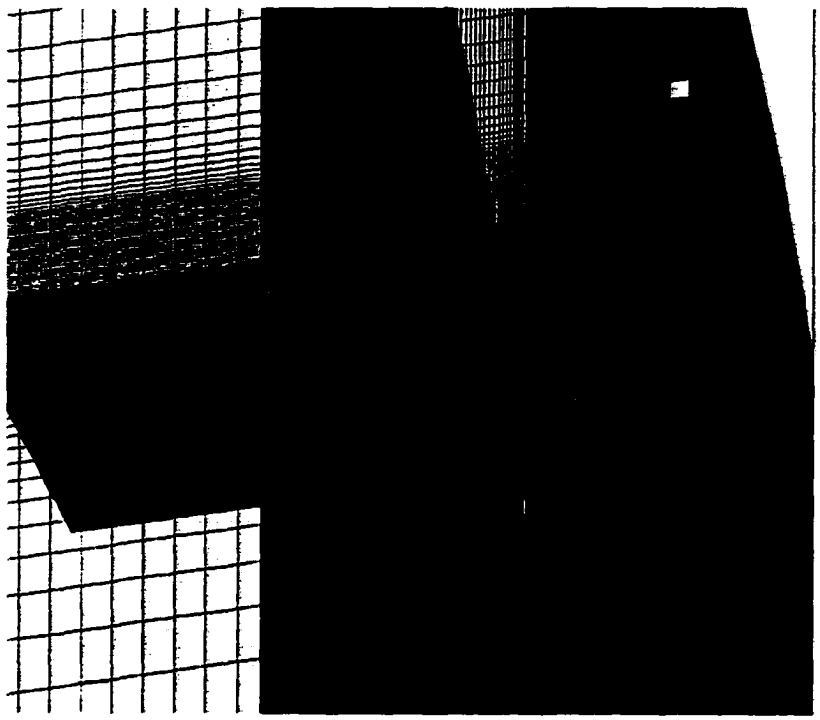


Figure 1.15 Detail of 3-D grid generation of a wing configuration

CHAPTER 2

Non-Reflecting Boundary Condition in Curvilinear Coordinates

Abstract

Time-dependent numerical methods for hyperbolic systems require time-dependent boundary conditions when the system is solved in a finite domain. The "correct" boundary is a crucial problem in solving such a system. In this chapter, non-reflecting boundary conditions based on the Navier-Stokes equations have been derived for curvilinear coordinates. A high-order scheme is used to discretize the non-reflecting boundary conditions. Several examples of non-reflecting boundary conditions are tested. The results are compared with the reference method based on extrapolation or Rieman invariants. It is found that the non-reflecting boundary condition is much better than the traditional methods of imposing boundary conditions.

2.1. Introduction

Numerical solutions to hyperbolic systems of differential equations, such as fluid dynamic equations, are usually obtained over a finite region. The time evolution of the system is governed not only by the state in the interior of the region, but also by waves which enter the region from outside its boundary.

The basic idea of non-reflecting boundary condition is based on the characteristic analysis. As we know, for hyperbolic equations, the waves propagate along the characteristic line. Only the domain in the downstream region can be affected. Based on this phenomenon, the boundary condition must be specified carefully. An improper boundary condition will distort the results and corrupt the numerical solution.

Direct numerical simulation of the Navier-Stokes equations has been the focus of many recent studies. In the field of finite difference methods, a modern algorithm based on high-order schemes can provide spectral-like resolution and very low numerical dissipation (Thompson, 1987 and 1990). The precision and the potential applications of these schemes, however, are constrained by the boundary conditions which have to be included in the final numerical models. Most direct numerical simulations are performed with periodic boundary conditions. In these configurations, the reference frame moves at the mean flow speed, and flow periodicity is assumed. A periodic geometry is the only type of geometry for which the problem can be closed exactly at the boundary. The computation domain is folded onto itself, and no boundary conditions are actually required. The periodicity assumption considerably limits the possible applications of these simulations. Simulations in which no periodicity is assumed and the flow inlets and outlets must be treated are much more practical. Indeed, these simulations are strongly dependent on boundary conditions and their treatment. General boundary conditions for direct numerical simulation of compressible flows are needed. The new constraints imposed on boundary condition formulations by these unsteady computations performed with high-order numerical methods in non-periodic domains are the following.

1. Direct numerical simulation of compressible flows requires an accurate control of

wave reflections from the boundaries of the computational domain. This is not the case when Navier-Stokes codes are used only to compute steady states. In this case, one is not interested in the behavior of the boundaries as long as a final steady state can be obtained. It is worth noting that the mechanisms by which waves are eliminated in many codes is somewhat unclear and very often due to numerical dissipation. As direct numerical simulation algorithms strive to minimize numerical viscosity, acoustic waves have to be eliminated by another mechanism such as non-reflecting or absorbing boundary conditions.

2. A large amount of experimental evidence suggests that acoustic waves are strongly coupled to many instability as well as acoustic waves (Bechert and Stahl, 1988). This interaction may even lead to large flow instabilities as, for example, in the case of the edgestone experiment (Ho and Nosseir, 1981). In the field of reacting flows, combustion instabilities provide numerous examples of interactions between turbulent combustion and acoustic waves. The simulation of these phenomena requires an accurate control of the behavior of the computation boundaries. Many studies have been concerned with direct numerical simulation of combustion instabilities. But the identification of the acoustical behavior of boundaries is not explicit, and its effects on the results are unclear. The problem of the downstream boundary is often removed by considering acoustic outlets where all variables are obtained by extrapolation. Even in cases where physical waves are not able to propagate upstream from the outlet, numerical waves may do so and interact with the flow. For example, recent studies show that strong numerical coupling mechanisms between inlet and outlet boundaries can lead to

non-physical oscillations for the 1-D advection equation (Vishnevetsky and Pariser, 1986).

3. Discretization and implementation of boundary conditions require more than the knowledge of the conditions ensuring well-posedness of the original Navier-Stokes equations. Other conditions have to be added to the original set of boundary conditions to solve for variables which are not specified by the boundary conditions. These additional conditions are often called "numerical" boundary conditions although they should be viewed only as compatibility relations required by the numerical method and not as boundary conditions. The computational results depend not only on the original equations and the boundary conditions but also on the numerical scheme and on the numerical conditions used at the boundaries.

The purpose of our work is to construct a systematic method for specifying these boundary conditions for the Navier-Stokes equations. The method presented here is an extension of methods developed for hyperbolic equations (Thompson, 1990). It allows control of the different waves which cross the boundaries.

The final results should meet the following requirements:

1. When the viscous term vanishes, the boundary conditions reduce to the Euler type boundary condition.
2. The method does not use any extrapolation procedure, thereby suppressing the arbitrariness in the construction of the boundary condition.
3. The number of boundary conditions specified for the Navier-Stokes equations is obtained through analysis.

In this chapter, the theory and the derivation of a non-reflecting boundary condition in a curvilinear coordinate system are given in detail. Then, according to the problem we are interested in, some typical boundary conditions are given. Finally, the non-reflecting boundary condition was used in some practical cases to test the correctness and its superiority over the conventional boundary treatment.

2.2. Theory and derivation of non-reflecting boundary conditions

We will call a boundary condition a physical boundary condition when it specifies the known physical behavior of one or more of the dependent variables at the boundary. For example, specification of the inlet longitudinal velocity on a boundary is a physical boundary condition. These conditions are independent of the numerical method used to solve the relevant equations. We expect the number of necessary and sufficient physical boundary conditions to be that suggested by theoretical analysis.

When the number of physical boundary conditions is less than the number of primitive variables, we should find some numerical methods to specify the boundary conditions.

An appealing technique for specifying boundary conditions for a hyperbolic system is to use relations based on characteristic lines, i.e., on the analysis of the different waves crossing the boundary. This method has been extensively studied by many authors. Our object here is to construct such a method for the Navier-Stokes equations. Such a method (Poinsot, and Lele, 1992) is called Navier-Stokes characteristic boundary conditions (NSCBC). This method is not only used for the Navier-Stokes equations, it can also be used in the Euler equations with a zero viscosity term.

The key point of NSCBC is that hyperbolic systems of equations represent the propagating of waves. Some of the waves are propagating into the computational domain while others are propagating out of it. The outward propagating waves have their behavior defined entirely by the solution at and within the boundary, and no boundary conditions are needed or so-called overspecified. The inward propagating waves depend on the solution exterior to the computational domain and therefore require boundary conditions to complete the specification of their behavior. Here, we discuss how to decompose the Navier-Stokes equations into wave modes and how to specify the boundary conditions.

First, suppose that all derivations are under the assumptions that the grid is static and the object is rigid. Therefore, $\frac{\partial}{\partial t}\left(\frac{1}{J}\right)=0$, $\xi_t=0$, $\eta_t=0$ and $\zeta_t=0$.

The conservation form of the Navier-Stokes equations is

$$\begin{aligned}&\frac{\partial}{\partial t}\left(\frac{Q}{J}\right)+\frac{\partial}{\partial \xi}\left(\frac{F}{J}\right)+\frac{\partial}{\partial \eta}\left(\frac{G}{J}\right)+\frac{\partial}{\partial \zeta}\left(\frac{H}{J}\right)\\&=\frac{1}{\mathrm{Re}}\left\{\frac{\partial}{\partial \xi}\left(\frac{F_v}{J}\right)+\frac{\partial}{\partial \eta}\left(\frac{G_v}{J}\right)+\frac{\partial}{\partial \zeta}\left(\frac{H_v}{J}\right)\right\}\end{aligned} \tag{2.2.1}$$

These equations can be written in a non-conservative form as

$$\frac{\partial}{\partial t}\left(\frac{Q}{J}\right)+A\frac{\partial}{\partial \xi}(Q)+B\frac{\partial}{\partial \eta}(Q)+C\frac{\partial}{\partial \zeta}(Q)=Vis \tag{2.2.2}$$

where the term "Vis" represents all viscous terms. Let $\overline{U}=u\xi_x+v\xi_y+w\xi_z$, $\overline{V}=u\eta_x+v\eta_y+w\eta_z$, and $\overline{W}=u\zeta_x+v\zeta_y+w\zeta_z$ be the controvariables along ξ, η, ζ direction. Let $\phi=\frac{\gamma-1}{2}\left(u^2+v^2+w^2\right)$. From the definition of F, G, and H, we

And A, B and C are

$$A=\frac{1}{J}\begin{bmatrix} 0 & \xi_x & \xi_y & \xi_z & 0 \\ \xi_x\phi-u\bar{U} & (2-\gamma)u\xi_x+\bar{U} & \xi_y u-\xi_x(\gamma-1)v & \xi_z u-\xi_x(\gamma-1)w & \xi_x(\gamma-1) \\ \xi_y\phi-v\bar{U} & \xi_x v-\xi_y(\gamma-1)u & (2-\gamma)v\xi_y+\bar{U} & \xi_z v-\xi_y(\gamma-1)w & \xi_y(\gamma-1) \\ \xi_z\phi-w\bar{U} & \xi_x w-\xi_z(\gamma-1)u & \xi_y w-\xi_z(\gamma-1)v & (2-\gamma)w\xi_z+\bar{U} & \xi_z(\gamma-1) \\ -\bar{U}\left(\frac{a^2}{\gamma-1}-\frac{\gamma-2}{\gamma-1}\phi\right) & \frac{a^2+\phi}{\gamma-1}\xi_x-(\gamma-1)u\bar{U} & \frac{a^2+\phi}{\gamma-1}\xi_y-(\gamma-1)v\bar{U} & \frac{a^2+\phi}{\gamma-1}\xi_z-(\gamma-1)w\bar{U} & \gamma\bar{U} \end{bmatrix} \tag{2.2.3a}$$

$$B=\frac{1}{J}\begin{bmatrix} 0 & \eta_x & \eta_y & \eta_z & 0 \\ \eta_x-u\bar{V} & (2-\gamma)u\eta_x+\bar{V} & \eta_y u-\eta_x(\gamma-1)v & \eta_z u-\eta_x(\gamma-1)w & \eta_x(\gamma-1) \\ \eta_y-v\bar{V} & \eta_x v-\eta_y(\gamma-1)u & (2-\gamma)v\eta_y+\bar{V} & \eta_z v-\eta_y(\gamma-1)w & \eta_y(\gamma-1) \\ \eta_z-w\bar{V} & \eta_x w-\eta_z(\gamma-1)u & \eta_y w-\eta_z(\gamma-1)v & (2-\gamma)w\eta_z+\bar{V} & \eta_z(\gamma-1) \\ -\bar{V}\left(\frac{a^2}{\gamma-1}-\frac{\gamma-2}{\gamma-1}\phi\right) & \frac{a^2+\phi}{\gamma-1}\eta_x-(\gamma-1)u\bar{V} & \frac{a^2+\phi}{\gamma-1}\eta_y-(\gamma-1)v\bar{V} & \frac{a^2+\phi}{\gamma-1}\eta_z-(\gamma-1)w\bar{V} & \gamma\bar{V} \end{bmatrix} \tag{2.2.3b}$$

$$C=\frac{1}{J}\begin{bmatrix} 0 & \zeta_x & \zeta_y & \zeta_z & 0 \\ \zeta_x-u\bar{W} & (2-\gamma)u\zeta_x+\bar{W} & \zeta_y u-\zeta_x(\gamma-1)v & \zeta_z u-\zeta_x(\gamma-1)w & \zeta_x(\gamma-1) \\ \zeta_y-v\bar{W} & \zeta_x v-\zeta_y(\gamma-1)u & (2-\gamma)v\zeta_y+\bar{W} & \zeta_z v-\zeta_y(\gamma-1)w & \zeta_y(\gamma-1) \\ \zeta_z-w\bar{W} & \zeta_x w-\zeta_z(\gamma-1)u & \zeta_y w-\zeta_z(\gamma-1)v & (2-\gamma)w\zeta_z+\bar{W} & \zeta_z(\gamma-1) \\ -\bar{W}\left(\frac{a^2}{\gamma-1}-\frac{\gamma-2}{\gamma-1}\phi\right) & \frac{a^2+\phi}{\gamma-1}\zeta_x-(\gamma-1)u\bar{W} & \frac{a^2+\phi}{\gamma-1}\zeta_y-(\gamma-1)v\bar{W} & \frac{a^2+\phi}{\gamma-1}\zeta_z-(\gamma-1)w\bar{W} & \gamma\bar{W} \end{bmatrix} \tag{2.2.3c}$$

Let q be the primary variables: $q=[\rho,u,v,w,p]^T$. Hence, we have the following:

$$\frac{\partial Q}{\partial q}=P=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ u & \rho & 0 & 0 & 0 \\ v & 0 & \rho & 0 & 0 \\ w & 0 & 0 & \rho & 0 \\ \frac{1}{2}\left(u^2+v^2+w^2\right) & \rho u & \rho v & \rho w & \frac{1}{\gamma-1} \end{bmatrix} \tag{2.2.4}$$

From P, we can calculate the inverse of P.

$$P^{-1}=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -\frac{u}{\rho} & \frac{1}{\rho} & 0 & 0 & 0 \\ -\frac{v}{\rho} & 0 & \frac{1}{\rho} & 0 & 0 \\ -\frac{w}{\rho} & 0 & 0 & \frac{1}{\rho} & 0 \\ \frac{(\gamma-1)}{2}\left(u^2+v^2+w^2\right) & -(\gamma-1)u & -(\gamma-1)v & -(\gamma-1)w & \gamma-1 \end{bmatrix} \tag{2.2.5}$$

Rewriting the Navier-Stokes equations in a non-conversation form,

$$\frac{P}{J}\frac{\partial}{\partial t}(q)+AP\frac{\partial}{\partial \xi}(q)+BP\frac{\partial}{\partial \eta}(q)+CP\frac{\partial}{\partial \zeta}(q)=Vis \tag{2.2.6}$$

$$\frac{\partial}{\partial t}(q)+JP^{-1}AP\frac{\partial}{\partial \xi}(q)+JP^{-1}BP\frac{\partial}{\partial \eta}(q)+JP^{-1}CP\frac{\partial}{\partial \zeta}(q)=JP^{-1}Vis \tag{3.2.7}$$

And calculating the following terms,

$$P^{-1}A=\frac{1}{J}\begin{bmatrix} 0 & \xi_x & \xi_y & \xi_z & 0 \\ \frac{1}{\rho}\left[\phi\xi_x-u\bar{U}\right] & \frac{1}{\rho}\left[\xi_x(1-\gamma)u+\bar{U}\right] & (1-\gamma)\frac{v}{\rho}\xi_x & (1-\gamma)\frac{w}{\rho}\xi_x & (1-\gamma)\frac{1}{\rho}\xi_x \\ \frac{1}{\rho}\left[\phi\xi_y-v\bar{U}\right] & (1-\gamma)\frac{u}{\rho}\xi_y & \frac{1}{\rho}\left[\xi_y(1-\gamma)v+\bar{U}\right] & (1-\gamma)\frac{w}{\rho}\xi_y & (1-\gamma)\frac{1}{\rho}\xi_y \\ \frac{1}{\rho}\left[\phi\xi_z-w\bar{U}\right] & (1-\gamma)\frac{u}{\rho}\xi_z & (1-\gamma)\frac{v}{\rho}\xi_z & \frac{1}{\rho}\left[\xi_z(1-\gamma)w+\bar{U}\right] & (1-\gamma)\frac{1}{\rho}\xi_z \\ \bar{U}\left[\phi-\gamma\frac{p}{\rho}\right] & (1-\gamma)u\bar{U}+\xi_x\frac{\gamma P}{\rho} & (1-\gamma)v\bar{U}+\xi_y\frac{\gamma P}{\rho} & (1-\gamma)w\bar{U}+\xi_z\frac{\gamma P}{\rho} & (\gamma-1)\bar{U} \end{bmatrix}$$

(2.2.8a)

$$P^{-1}B=\frac{1}{J}\begin{bmatrix} 0 & \eta_x & \eta_y & \eta_z & 0 \\ \frac{1}{\rho}\left[\phi\eta_x-u\bar{U}\right] & \frac{1}{\rho}\left[\eta_x(1-\gamma)u+\bar{U}\right] & (1-\gamma)\frac{v}{\rho}\eta_x & (1-\gamma)\frac{w}{\rho}\eta_x & (1-\gamma)\frac{1}{\rho}\eta_x \\ \frac{1}{\rho}\left[\phi\eta_y-v\bar{U}\right] & (1-\gamma)\frac{u}{\rho}\eta_y & \frac{1}{\rho}\left[\eta_y(1-\gamma)v+\bar{U}\right] & (1-\gamma)\frac{w}{\rho}\eta_y & (1-\gamma)\frac{1}{\rho}\eta_y \\ \frac{1}{\rho}\left[\phi\eta_z-w\bar{U}\right] & (1-\gamma)\frac{u}{\rho}\eta_z & (1-\gamma)\frac{v}{\rho}\eta_z & \frac{1}{\rho}\left[\eta_z(1-\gamma)w+\bar{U}\right] & (1-\gamma)\frac{1}{\rho}\eta_z \\ \bar{U}\left[\phi-\gamma\frac{p}{\rho}\right] & (1-\gamma)u\bar{U}+\eta_x\frac{\gamma P}{\rho} & (1-\gamma)v\bar{U}+\eta_y\frac{\gamma P}{\rho} & (1-\gamma)w\bar{U}+\eta_z\frac{\gamma P}{\rho} & (\gamma-1)\bar{U} \end{bmatrix}$$

(2.2.8b)

$$P^{-1}C=\frac{1}{J}\begin{bmatrix} 0 & \zeta_x & \zeta_y & \zeta_z & 0 \\ \frac{1}{\rho}\left[\phi\zeta_x-u\bar{U}\right] & \frac{1}{\rho}\left[\zeta_x(1-\gamma)u+\bar{U}\right] & (1-\gamma)\frac{v}{\rho}\zeta_x & (1-\gamma)\frac{w}{\rho}\zeta_x & (1-\gamma)\frac{1}{\rho}\zeta_x \\ \frac{1}{\rho}\left[\phi\zeta_y-v\bar{U}\right] & (1-\gamma)\frac{u}{\rho}\zeta_y & \frac{1}{\rho}\left[\zeta_y(1-\gamma)v+\bar{U}\right] & (1-\gamma)\frac{w}{\rho}\zeta_y & (1-\gamma)\frac{1}{\rho}\zeta_y \\ \frac{1}{\rho}\left[\phi\zeta_z-w\bar{U}\right] & (1-\gamma)\frac{u}{\rho}\zeta_z & (1-\gamma)\frac{v}{\rho}\zeta_z & \frac{1}{\rho}\left[\zeta_z(1-\gamma)w+\bar{U}\right] & (1-\gamma)\frac{1}{\rho}\zeta_z \\ \bar{U}\left[\phi-\gamma\frac{p}{\rho}\right] & (1-\gamma)u\bar{U}+\zeta_x\frac{\gamma P}{\rho} & (1-\gamma)v\bar{U}+\zeta_y\frac{\gamma P}{\rho} & (1-\gamma)w\bar{U}+\zeta_z\frac{\gamma P}{\rho} & (\gamma-1)\bar{U} \end{bmatrix} \tag{2.2.8c}$$

We obtain the following:

$$JP^{-1}AP=\begin{bmatrix} \bar{U} & \rho\xi_x & \rho\xi_y & \rho\xi_z & 0 \\ 0 & \bar{U} & 0 & 0 & \frac{1}{\rho}\xi_x \\ 0 & 0 & \bar{U} & 0 & \frac{1}{\rho}\xi_y \\ 0 & 0 & 0 & \bar{U} & \frac{1}{\rho}\xi_z \\ 0 & \gamma p\xi_x & \gamma p\xi_y & \gamma p\xi_z & \bar{U} \end{bmatrix} \tag{2.2.9a}$$

$$JP^{-1}BP=\begin{bmatrix} \bar{V} & \rho\eta_x & \rho\eta_y & \rho\eta_z & 0 \\ 0 & \bar{V} & 0 & 0 & \frac{1}{\rho}\eta_x \\ 0 & 0 & \bar{V} & 0 & \frac{1}{\rho}\eta_y \\ 0 & 0 & 0 & \bar{V} & \frac{1}{\rho}\eta_z \\ 0 & \gamma p\eta_x & \gamma p\eta_y & \gamma p\eta_z & \bar{V} \end{bmatrix} \tag{2.2.9b}$$

$$JP^{-1}CP=\begin{bmatrix} \bar{W} & \rho\zeta_x & \rho\zeta_y & \rho\zeta_z & 0 \\ 0 & \bar{W} & 0 & 0 & \frac{1}{\rho}\zeta_x \\ 0 & 0 & \bar{W} & 0 & \frac{1}{\rho}\zeta_y \\ 0 & 0 & 0 & \bar{W} & \frac{1}{\rho}\zeta_z \\ 0 & \gamma p\zeta_x & \gamma p\zeta_y & \gamma p\zeta_z & \bar{W} \end{bmatrix} \tag{2.2.9c}$$

Calculating the characteric value of above matrixs, let $\left|JP^{-1}AP-\lambda I\right|=0$, $\left|JP^{-1}BP-\lambda I\right|=0$, $\left|JP^{-1}CP-\lambda I\right|=0$. In the ξ direction, we have

$$\begin{bmatrix} \overline{U}-\lambda & \rho\xi_x & \rho\xi_y & \rho\xi_z & 0 \\ 0 & \overline{U}-\lambda & 0 & 0 & \frac{1}{\rho}\xi_x \\ 0 & 0 & \overline{U}-\lambda & 0 & \frac{1}{\rho}\xi_y \\ 0 & 0 & 0 & \overline{U}-\lambda & \frac{1}{\rho}\xi_z \\ 0 & \gamma p\xi_x & \gamma p\xi_y & \gamma p\xi_z & \overline{U}-\lambda \end{bmatrix}=0 \qquad (2.2.10)$$

$$\left(\overline{U}-\lambda\right)^2\begin{vmatrix} \overline{U}-\lambda & 0 & \frac{1}{\rho}\xi \\ 0 & \overline{U}-\lambda & \frac{1}{\rho}\xi \\ \gamma p\xi_y & \gamma p\xi_z & \overline{U}-\lambda \end{vmatrix}-\left(\overline{U}-\lambda\right)\frac{\xi_x}{\rho}\begin{vmatrix} 0 & \overline{U}-\lambda & 0 \\ 0 & 0 & \overline{U}-\lambda \\ \gamma p\xi_x & \gamma p\xi_y & \gamma p\xi_z \end{vmatrix}=0$$

$$\left(\overline{U}-\lambda\right)^3\left\{\left(\overline{U}-\lambda\right)^2-a^2\left(\xi_x^2+\xi_y^2+\xi_z^2\right)\right\}=0$$

Solving the above equation, we get the 5 eigenvalues of matrix $JP^{-1}AP$:

$$\lambda_1=\overline{U}-a\sqrt{\xi_x^2+\xi_y^2+\xi_z^2}$$

$$\lambda_{2,3,4}=\overline{U}$$

$$\lambda_5=\overline{U}+a\sqrt{\xi_x^2+\xi_y^2+\xi_z^2}$$

With the eigenvalue in matrix form as

$$\Lambda=\begin{bmatrix} \overline{U}-a\left|\nabla\xi\right| & 0 & 0 & 0 & 0 \\ 0 & \overline{U} & 0 & 0 & 0 \\ 0 & 0 & \overline{U} & 0 & 0 \\ 0 & 0 & 0 & \overline{U} & 0 \\ 0 & 0 & 0 & 0 & \overline{U}+a\left|\nabla\xi\right| \end{bmatrix} \qquad (2.2.11)$$

From the eigenvalue we obtained above, we can get the left and right eignvectors of the matrix $JP^{-1}AP$ as

$$S=\begin{bmatrix} \frac{1}{2a^2} & \frac{1}{a^2} & 0 & 0 & \frac{1}{2a^2} \\ -\frac{\xi_x}{2|\nabla\xi|\rho a} & 0 & -\frac{\xi_y}{|\nabla\xi|^2} & -\frac{\xi_z}{|\nabla\xi|^2} & \frac{\xi_x}{2|\nabla\xi|\rho a}_x \\ -\frac{\xi_y}{2|\nabla\xi|\rho a} & 0 & \frac{\xi_x^2+\xi_z^2}{|\nabla\xi|^2\xi_x} & -\frac{\xi_z\xi_y}{|\nabla\xi|^2\xi_x} & \frac{\xi_y}{2|\nabla\xi|\rho a}_y \\ -\frac{\xi_z}{2|\nabla\xi|\rho a} & 0 & -\frac{\xi_y\xi_z}{|\nabla\xi|^2\xi_x} & \frac{\xi_y^2+\xi_x^2}{|\nabla\xi|^2\xi_x} & \frac{\xi_z}{2|\nabla\xi|\rho a}_z \\ \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} \end{bmatrix} \tag{2.2.12}$$

$$S^{-1}=\begin{bmatrix} 0 & -\frac{\gamma p\xi_x}{a|\nabla\xi|} & -\frac{\gamma p\xi_y}{a|\nabla\xi|} & -\frac{\gamma p\xi_z}{a|\nabla\xi|} & 1 \\ a^2 & 0 & 0 & 0 & -1 \\ 0 & -\xi_y & \xi_x & 0 & 0 \\ 0 & -\xi_z & 0 & \xi_x & 0 \\ 0 & \frac{\gamma p\xi_x}{a|\nabla\xi|} & \frac{\gamma p\xi_y}{a|\nabla\xi|} & \frac{\gamma p\xi_z}{a|\nabla\xi|} & 1 \end{bmatrix} \tag{2.2.13}$$

Other eigenvalues of $JP^{-1}BP$, $JP^{-1}CP$ can be obtained similiarly. The only difference is that they are in different directions (η, ζ).

The non-conservative form of the Navier-Stokes equations is

$$\frac{\partial}{\partial t}(q)+S\Lambda S^{-1}\frac{\partial}{\partial\xi}(q)=RM=\left(JP^{-1}Vis-JP^{-1}BP\frac{\partial}{\partial\eta}(q)-JP^{-1}CP\frac{\partial}{\partial\zeta}(q)\right) \tag{2.2.14}$$

Let $\bar{L}=\Lambda S^{-1}\frac{\partial q}{\partial\xi}$ be a vector; therefore, we have

$$S^{-1}\frac{\partial}{\partial t}(q)+\Lambda S^{-1}\frac{\partial}{\partial\xi}(q)=S^{-1}RM \tag{2.2.15}$$

which is equivalent to

$$S^{-1}\frac{\partial}{\partial t}(q)+\bar{L}=S^{-1}RM \tag{2.2.16}$$

Now, we rewrite the Navier-Stokes equations in another form which contains the quantities L, which we will use to specify the boundary conditions. The only term which has relations in the ξ direction is L. The other terms are related to the transverse coordinates.

The purpose of specifying boundary conditions is to apply whatever information is needed at the boundaries of the computational domain in order to complete the definition of the behavior of the system. The number of boundary conditions which may be imposed depends on the physics of the problem and may not be specified arbitrarily. This is the wave nature of hyperbolic equations.

Each eignvalue λ_i obtained above represents the characteristic velocity at which a particular wave mode propagates (such as advaction waves, sound waves). At a point on some coordinate, some characteristic velocities describe outgoing waves, while others describe incoming waves. The behavior of the outgoing waves is completely determined by data contained within and on the computational domain, while the behavior of the incoming waves is specified by data external to and on the boundary of a computational domain. The number of boundary conditions which must be specified at a point on the boundary is equal to the number of incoming waves at that point. We will specify the boundary conditions by determing the values of L for incoming waves, and compute the the other components of L from the interior of the domain.

The component of L can be written as

$$L=\begin{bmatrix} 0 & -\lambda_1\frac{\gamma p\xi_x}{a|\nabla\xi|} & -\lambda_1\frac{\gamma p\xi_y}{a|\nabla\xi|} & -\lambda_1\frac{\gamma p\xi_z}{a|\nabla\xi|} & 1 \\ \lambda_2 a^2 & 0 & 0 & 0 & -\lambda_2 \\ 0 & -\lambda_3\xi_y & \lambda_3\xi_x & 0 & 0 \\ 0 & -\lambda_4\xi_z & 0 & \lambda_4\xi_x & 0 \\ 0 & \lambda_5\frac{\gamma p\xi_x}{a|\nabla\xi|} & \lambda_5\frac{\gamma p\xi_y}{a|\nabla\xi|} & \lambda_5\frac{\gamma p\xi_z}{a|\nabla\xi|} & \lambda_5 \end{bmatrix}\begin{bmatrix} \frac{\partial\rho}{\partial\xi} \\ \frac{\partial u}{\partial\xi} \\ \frac{\partial v}{\partial\xi} \\ \frac{\partial w}{\partial\xi} \\ \frac{\partial p}{\partial\xi} \end{bmatrix} \tag{2.2.17}$$

$$L_1=\lambda_1\left[-\frac{\gamma p\xi_x}{a|\nabla\xi|}\frac{\partial u}{\partial x}-\frac{\gamma p\xi_y}{a|\nabla\xi|}\frac{\partial v}{\partial x}-\frac{\gamma p\xi_z}{a|\nabla\xi|}\frac{\partial w}{\partial x}+\frac{\partial p}{\partial\xi}\right] \tag{2.2.18a}$$

$$L_2=\lambda_2\left[a^2\frac{\partial\rho}{\partial x}-\frac{\partial p}{\partial x}\right] \tag{2.2.18b}$$

$$L_3=\lambda_3\left[-\xi_y\frac{\partial u}{\partial x}+\xi_x\frac{\partial v}{\partial x}\right] \tag{2.2.18c}$$

$$L_4=\lambda_4\left[-\xi_z\frac{\partial u}{\partial x}+\xi_x\frac{\partial w}{\partial x}\right] \tag{2.2.18d}$$

$$L_5=\lambda_5\left[\frac{\gamma p\xi_x}{a|\nabla\xi|}\frac{\partial u}{\partial x}+\frac{\gamma p\xi_y}{a|\nabla\xi|}\frac{\partial v}{\partial x}+\frac{\gamma p\xi_z}{a|\nabla\xi|}\frac{\partial w}{\partial x}+\frac{\partial p}{\partial\xi}\right] \tag{2.2.18e}$$

The element of L_i i=1...5 can be determined as follows: when the characteristic velocity λ_i points out of the computational domain, compute the corresponding L_i from its definition by using one-sided derivative approximations. When λ_i points into the computational domain, specify the value of L_i from the boundary condition. Serveral useful boundary conditions will be discussed later in this chapter.

After we obtain the vector L, from the definition of L, we have $\frac{\partial q}{\partial\xi}=\Lambda S^{-1}\overline{L}$,

which can be written as:

$$\frac{\partial \rho}{\partial \xi} = \frac{1}{a^2}\left[\frac{1}{2}\left(\frac{L_1}{\overline{U} - a|\nabla\xi|} + \frac{L_5}{\overline{U} + a|\nabla\xi|}\right) + \frac{L_2}{\overline{U}}\right] \tag{2.2.19a}$$

$$\frac{\partial u}{\partial \xi} = \frac{\xi_x}{2\rho a|\nabla\xi|}\left[-\frac{L_1}{\overline{U} - a|\nabla\xi|} + \frac{L_5}{\overline{U} + a|\nabla\xi|}\right] - \frac{1}{\overline{U}|\nabla\xi|^2}\left(\xi_y L_3 + \xi_z L_4\right)$$

(2.2.19b)

$$\frac{\partial v}{\partial \xi} = \frac{\xi_y}{2\rho a|\nabla\xi|}\left[-\frac{L_1}{\overline{U} - a|\nabla\xi|} + \frac{L_5}{\overline{U} + a|\nabla\xi|}\right] + \frac{1}{\xi_x \overline{U}|\nabla\xi|^2}\left(\left(\xi_x^2 + \xi_y^2\right)L_3 - \xi_y \xi_z L_4\right)$$

(2.2.19c)

$$\frac{\partial w}{\partial \xi} = \frac{\xi_z}{2\rho a|\nabla\xi|}\left[-\frac{L_1}{\overline{U} - a|\nabla\xi|} + \frac{L_5}{\overline{U} + a|\nabla\xi|}\right] - \frac{1}{\xi_x \overline{U}|\nabla\xi|^2}\left(\xi_y \xi_z L_3 - \left(\xi_x^2 + \xi_y^2\right)L_4\right)$$

(2.2.19d)

$$\frac{\partial p}{\partial \xi} = \frac{1}{2}\left(\frac{L_1}{\overline{U} - a|\nabla\xi|} + \frac{L_5}{\overline{U} + a|\nabla\xi|}\right) \tag{2.2.19e}$$

Now, the non-conservative Navier-Stokes equation is the following:

$$\frac{\partial}{\partial t}(q) + SL = \left(JP^{-1}Vis - JP^{-1}BP\frac{\partial}{\partial \eta}(q) - JP^{-1}CP\frac{\partial}{\partial \zeta}(q)\right) \tag{2.2.20}$$

Because matrixes S and L are already known, the term SL can be calculated from the above description. The right-hand term of the equation has no relation to the ξ direction and contains only the transverse direction. Therefore, the right-hand term can be calaulated by using just conventional difference schemes.

Let $\overline{d} = S\overline{L}$, then

$$S\bar{L} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \end{bmatrix} = \begin{bmatrix} \frac{1}{a^2}\left[\frac{1}{2}(L_1 + L_5) + L_2\right] \\ \frac{\xi_x}{2\rho a|\nabla\xi|}(L_5 - L_1) - \frac{1}{|\nabla\xi_x|^2}(\xi_y L_3 + \xi_z L_4) \\ \frac{\xi_x}{2\rho a|\nabla\xi|}(L_5 - L_1) + \frac{1}{|\nabla\xi_x|^2\xi_x}\left[(\xi_x^2 + \xi_y^2)L_3 - \xi_z\xi_y L_4\right] \\ \frac{\xi_x}{2\rho a|\nabla\xi|}(L_5 - L_1) - \frac{1}{|\nabla\xi_x|^2\xi_x}\left[(\xi_x^2 + \xi_y^2)L_3 - \xi_z\xi_y L_4\right] \\ \frac{1}{2}(L_1 + L_5) \end{bmatrix}$$

(2.2.21)

If we omit the viscous term, the non-conservation form of the Navier-Stokes equation become

$$\frac{\partial\rho}{\partial t} + d_1 + \bar{V}\frac{\partial\rho}{\partial\eta} + \rho\left(\eta_x\frac{\partial u}{\partial\eta} + \eta_y\frac{\partial v}{\partial\eta} + \eta_z\frac{\partial w}{\partial\eta}\right) + \bar{W}\frac{\partial\rho}{\partial\zeta} + \rho\left(\zeta_x\frac{\partial u}{\partial\zeta} + \zeta_y\frac{\partial v}{\partial\zeta} + \zeta_z\frac{\partial w}{\partial\zeta}\right) = 0$$

(2.2.22a)

$$\frac{\partial u}{\partial t} + d_2 + \bar{V}\frac{\partial u}{\partial\eta} + \frac{1}{\rho}\eta_x\frac{\partial p}{\partial\eta} + \bar{W}\frac{\partial u}{\partial\zeta} + \frac{1}{\rho}\zeta_x\frac{\partial p}{\partial\zeta} = 0 \qquad (2.2.22b)$$

$$\frac{\partial v}{\partial t} + d_3 + \bar{V}\frac{\partial v}{\partial\eta} + \frac{1}{\rho}\eta_y\frac{\partial p}{\partial\eta} + \bar{W}\frac{\partial v}{\partial\zeta} + \frac{1}{\rho}\zeta_y\frac{\partial p}{\partial\zeta} = 0 \qquad (2.2.22c)$$

$$\frac{\partial u}{\partial t} + d_4 + \bar{V}\frac{\partial w}{\partial\eta} + \frac{1}{\rho}\eta_z\frac{\partial p}{\partial\eta} + \bar{W}\frac{\partial w}{\partial\zeta} + \frac{1}{\rho}\zeta_z\frac{\partial p}{\partial\zeta} = 0 \qquad (2.2.22d)$$

$$\frac{\partial p}{\partial t} + d_5 + \gamma p\left(\eta_x\frac{\partial u}{\partial\eta} + \eta_y\frac{\partial v}{\partial\eta} + \eta_z\frac{\partial w}{\partial\eta}\right) + \bar{V}\frac{\partial p}{\partial\eta} + \gamma p\left(\zeta_x\frac{\partial u}{\partial\zeta} + \zeta_y\frac{\partial v}{\partial\zeta} + \zeta_z\frac{\partial w}{\partial\zeta}\right) + \bar{W}\frac{\partial p}{\partial\zeta} = 0$$

(2.2.22e)

From Eqs.(2.2.22a) to (2.2.22e), we have

$$\gamma M^2\left(\frac{1}{\rho}\frac{\partial p}{\partial t}-\frac{p}{\rho^2}\frac{\partial \rho}{\partial t}\right)+\gamma M^2\left(\frac{1}{\rho}d_5-\frac{p}{\rho^2}d_1\right)+\gamma M^2\left[\frac{(\gamma-1)p}{\rho}\left(\eta_x\frac{\partial u}{\partial \eta}+\eta_y\frac{\partial v}{\partial \eta}+\eta_z\frac{\partial w}{\partial \eta}\right)\right]$$
$$+\overline{V}\frac{\partial T}{\partial \eta}+\gamma M^2\left[\frac{(\gamma-1)p}{\rho}\left(\zeta_x\frac{\partial u}{\partial \zeta}+\zeta_y\frac{\partial v}{\partial \zeta}+\zeta_z\frac{\partial w}{\partial \zeta}\right)\right]+\overline{W}\frac{\partial T}{\partial \zeta}=0 \tag{2.2.23}$$

Note that $T=\gamma M^2\frac{P}{\rho}$ and $\frac{\partial T}{\partial \xi}=\frac{\gamma M^2}{\rho}\frac{\partial p}{\partial \xi}-\frac{\gamma M^2 p}{\rho^2}\frac{\partial \rho}{\partial \xi}$. Then we have the similar expression can be obtained for the temperature T:

$$\frac{\partial T}{\partial t}+\frac{M^2}{\rho}\left[\frac{1}{2}(\gamma-1)(L_1+L_5)-L_2\right]+\gamma M^2\left[\frac{(\gamma-1)p}{\rho}\left(\eta_x\frac{\partial u}{\partial \eta}+\eta_y\frac{\partial v}{\partial \eta}+\eta_z\frac{\partial w}{\partial \eta}\right)\right]$$
$$+\overline{V}\frac{\partial T}{\partial \eta}+\gamma M^2\left[\frac{(\gamma-1)p}{\rho}\left(\zeta_x\frac{\partial u}{\partial \zeta}+\zeta_y\frac{\partial v}{\partial \zeta}+\zeta_z\frac{\partial w}{\partial \zeta}\right)\right]+\overline{W}\frac{\partial T}{\partial \zeta}=0 \tag{2.2.24}$$

2.3. Some typical non-reflection boundary conditions

For computational fluid dynamics, the boundaries of a physical domain are of interest. Typically, there are several types of boundary conditions. The most useful boundary conditions (Jiang and Shan, 1999) we are interested are described below:

1. No-slip boundary condition

On a no-slip wall, not only is the normal velocity component zero, but also friction causes the transverse velocity component to be zero. The initial data must have all zero velocity components at the wall. Here, we specify that T=const. Therefore, we have

$$u=v=w=0 \text{ then } \overline{U}=\overline{V}=\overline{W}=0$$

Assume ξ is the normal direction. From equations (2.2.22b) to (2.2.22d), we have

$$\frac{\xi_x}{2\rho a|\nabla\xi|}(L_5-L_1)-\frac{1}{|\nabla\xi_x|^2}(\xi_y L_3+\xi_z L_4)=-\frac{1}{\rho}\eta_x\frac{\partial p}{\partial\eta}-\frac{1}{\rho}\zeta_x\frac{\partial p}{\partial\zeta} \tag{2.3.1}$$

$$\frac{\xi_x}{2\rho a|\nabla\xi|}(L_5-L_1)+\frac{1}{|\nabla\xi_x|^2\xi_x}\left[(\xi_x^2+\xi_y^2)L_3-\xi_z\xi_y L_4\right]=-\frac{1}{\rho}\eta_y\frac{\partial p}{\partial\eta}-\frac{1}{\rho}\zeta_y\frac{\partial p}{\partial\zeta} \tag{2.3.2}$$

$$\frac{\xi_z}{2\rho a|\nabla\xi|}(L_5-L_1)-\frac{1}{|\nabla\xi_x|^2\xi_x}\left[(\xi_x^2+\xi_y^2)L_3-\xi_z\xi_y L_4\right]=-\frac{1}{\rho}\eta_z\frac{\partial p}{\partial\eta}-\frac{1}{\rho}\zeta_z\frac{\partial p}{\partial\zeta} \tag{2.3.3}$$

Solving equations, we obtain

$$L_3=\frac{1}{\rho}\left[(\eta_x\xi_y-\eta_y\xi_x)\frac{\partial p}{\partial\eta}+(\zeta_x\xi_y-\zeta_y\xi_x)\frac{\partial p}{\partial\zeta}\right] \tag{2.3.4}$$

$$L_4=\frac{1}{\rho}\left[(\eta_x\xi_z-\eta_z\xi_x)\frac{\partial p}{\partial\eta}+(\zeta_x\xi_z-\zeta_z\xi_x)\frac{\partial p}{\partial\zeta}\right] \tag{2.3.5}$$

At the wall surface, $\lambda_1=\bar{U}-a|\nabla\xi|<0$, $\lambda_{2,3,4}=\bar{U}=0$, $\lambda_5=\bar{U}+a|\nabla\xi|>0$. Therefore, corresponding to λ_1, L_1 is determined by the interior point by using one-sided difference. While L_5, which corresponds to λ_5, must be specified by the boundary condition. From (2.3.1), we have

$$L_5=L_1+\frac{2\rho a|\nabla\xi|}{\xi_x}\left[\frac{1}{|\nabla\xi_x|^2}(\xi_y L_3+\xi_z L_4)-\frac{1}{\rho}\left(\eta_x\frac{\partial p}{\partial\eta}-\zeta_x\frac{\partial p}{\partial\zeta}\right)\right] \tag{2.3.6}$$

Using the equation for the temperature T, then

$$L_2=\frac{1}{2}\bar{U}(\gamma-1)\left[\frac{L_1}{\bar{U}-a|\nabla\xi|}+\frac{L_5}{\bar{U}+a|\nabla\xi|}\right] \tag{2.3.7}$$

2. Subsonic inflow

Assume the fluid flow along the ξ direction as shown in Fig.2.1, with the characteristic values

$$\lambda_1 = \overline{U} - a|\nabla\xi| < 0, \quad \lambda_{2,3,4} = \overline{U}, \quad \lambda_5 = \overline{U} + a|\nabla\xi| > 0$$

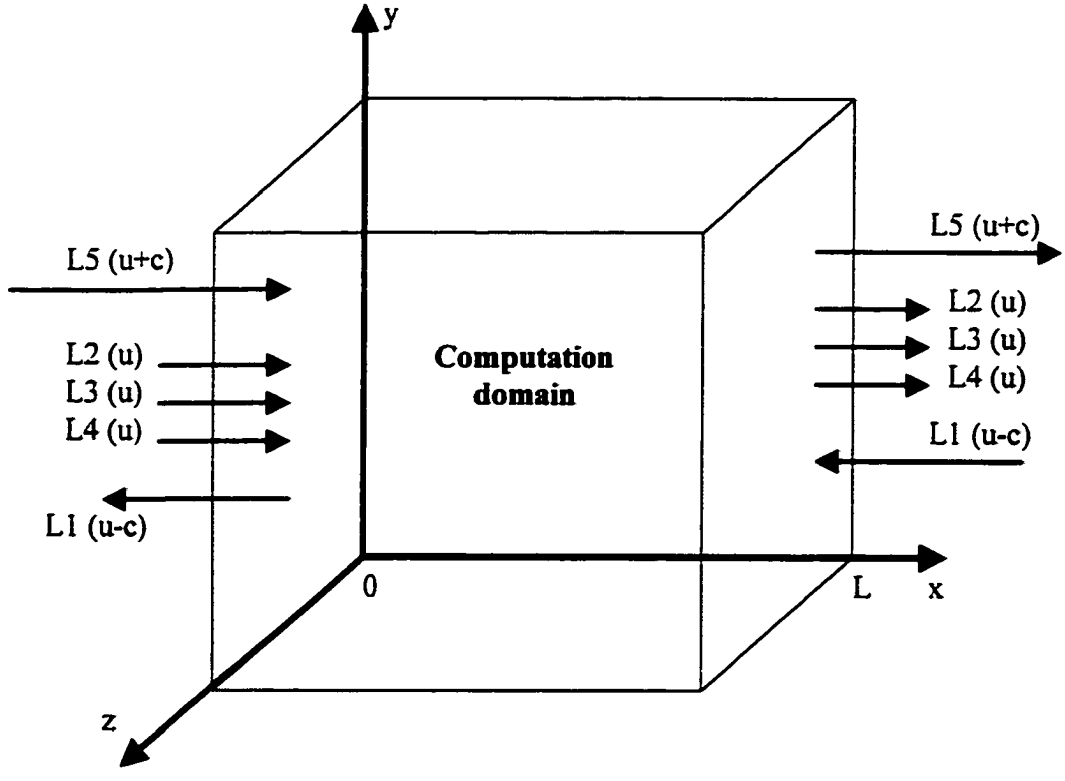


Fig 2.1 Waves leaving and entering the computational domain

Because $\lambda_1 < 0$, L_1 can be obtained from the interior computational domain by one-sided difference. The other four characteristic values are greater than zero, so they must be specified at the inflow boundary. The formula of L_i corresponding to each λ_i can be found from the above considerations. They are

$$L_3 = \xi_y \frac{\partial u}{\partial t} - \xi_x \frac{\partial v}{\partial t} \tag{2.3.8}$$

$$L_4 = \xi_y \frac{\partial u}{\partial t} - \xi_x \frac{\partial v}{\partial t} \tag{2.3.9}$$

$$L_5 = L_1 + \frac{2\rho a|\nabla\xi|}{\xi_x}\left[\frac{1}{|\nabla\xi|^2}(\xi_y L_3 + \xi_x L_4) - \frac{\partial u}{\partial t}\right] \tag{2.3.10}$$

$$L_2 = \frac{\rho}{M^2}\frac{\partial T}{\partial t} + \frac{1}{2}(L_1 + L_5) \tag{2.3.11}$$

3. Subsonic outflow boundary condition

Assume the fluid flow along the ξ direction. For the subsonic outflow, the static pressure p_∞ is given. At the boundary, characteristic waves L_2, L_3, L_4, L_5 are going outward from the computational domain, while L_1 is entering the computational domain. Therefore, L_2, L_3, L_4, L_5 are calculated from the interior points of the computational domain by one-sided difference, and L_1 is given by Poinsot and Lele, 1992.

$$L_1 = K(p - p_\infty) \tag{2.3.12}$$

Meanwhile, $K = \delta(1 - M^2)a/l$, and δ is a constant.

4. Far field boundary condition

Assume that ξ is the normal direction of the far field. The direction of the characteristic waves are defined automatically by the local field values. The outgoing waves L_i are calculated from the interior points of the computational domain, while the inward waves L_i are set to zero.

$$L_i = \begin{cases} L_i, \lambda_i > 0 \\ 0, \lambda_i \le 0 \end{cases} \tag{2.3.13}$$

2.4. Results and analysis

Using the non-reflecting boundary condition, we calculate some test cases. The first one is the simulation of a subsonic steady laminar boundary layer over a flat plat. This is a typical test problem because it has a theoretical solution. The Navier-Stokes equations are solved. because it is a time-dependent problem, the local time step technique is used to get the maximum time step at each grid point. Although the function value at local time step is not accurate, it will approach a steady state. The computational domain is constructed as a 3-D rectangular box. Because the flow is symmetric in the direction vertical to the paper, the periodic boundary conditions are used in this direction.

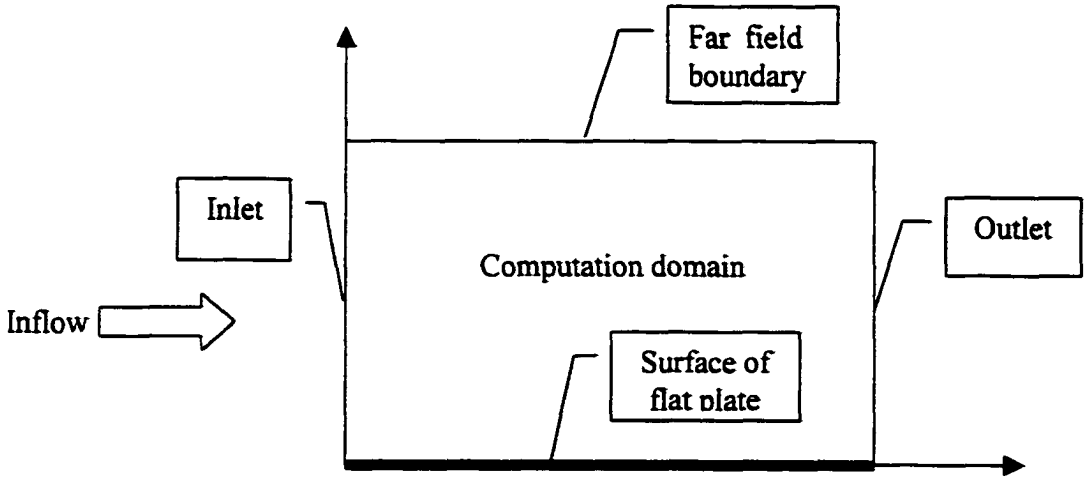


Fig 2.2 Illustration of computational domain and boundary of a flat plate.

As shown in Fig. 2.2, the computational domain is bounded by boundaries. Four non-reflecting boundary conditions are used here. At the inlet of the computational domain, the incoming boundary condition is specified by using (2.3.8) to (2.3.11). At the outlet of the computational domain, the outflow boundary condition is specified by using

(2.3.12) and the method described in last section. At the far boundary, equation (2.3.13) is used to obtain the non-reflection boundary condition. At the surface of a flat plate, the velocities in three directions are specified as zero. Here the boundary conditions are obtained by using (2.3.4) to (2.3.7). The Mach number of the incoming flow is $M_\infty = 0.1$ and Reynolds number is 1.0e+6. The incident angle of the incoming flow is zero. Fig. 2.3 shows the computation grids for the flat plate. The grid size is 65x97. The flow is laminar flow over the flat plate. We choose two very sensitive variables (velocities om the x and y directions) as our comparison parameters. Fig 2.4 and Fig 2.6 compared u by showing the u contour. We can see there are some waves entering the domain from the boundary. It is the same for the v contour. These are the unexpected physical waves entering the computation domain.

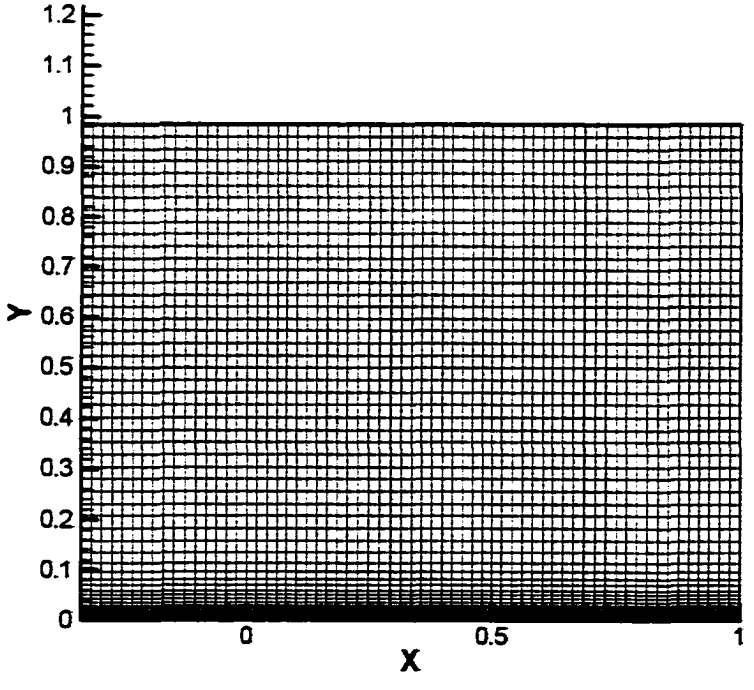


Fig. 2.3 Computation grid

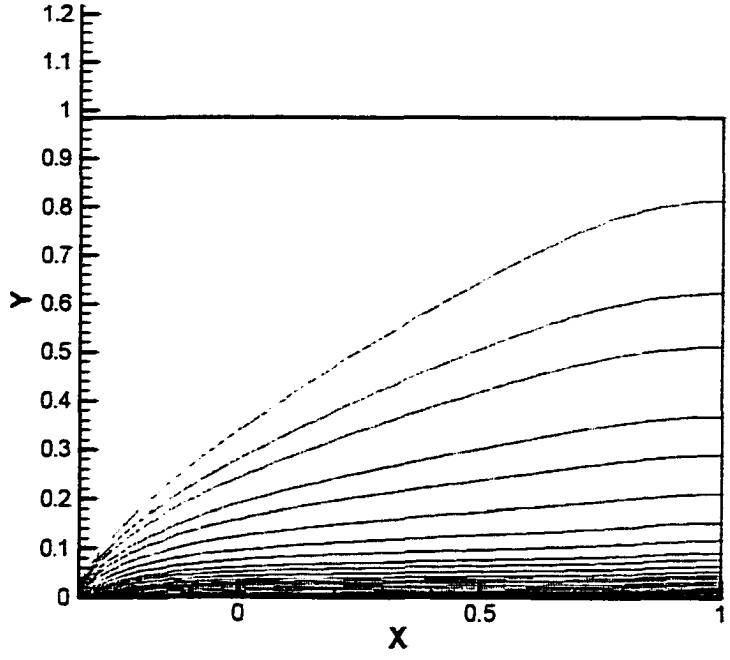


Fig 2.4 Contour of velocity along the x direction for a flat plate.

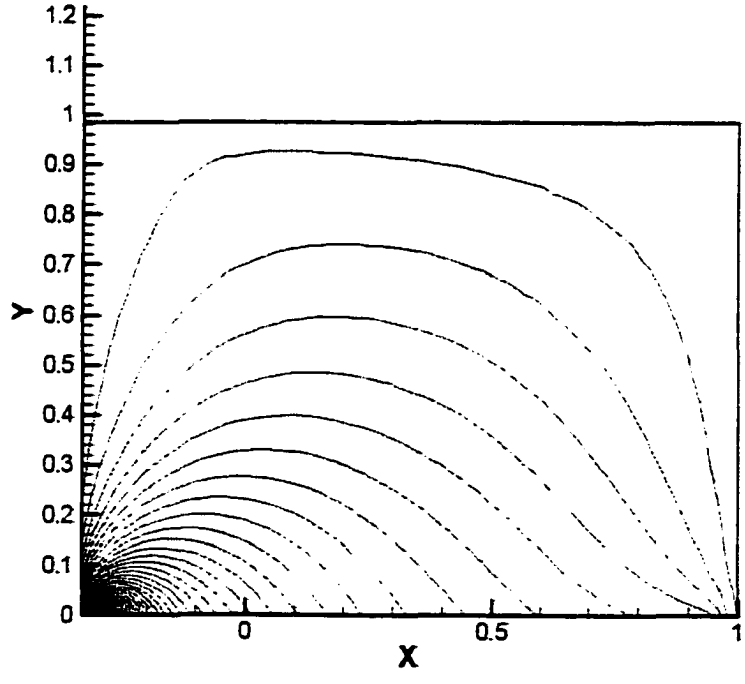


Fig 2.5 Contour of velocity along the y direction for a flat plate.

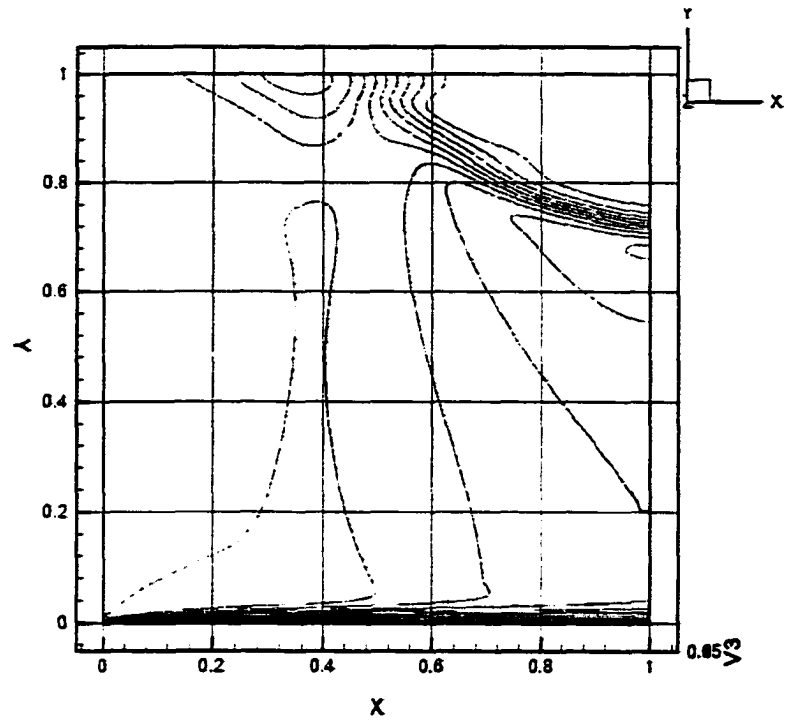


Fig 2.6 Contour of velocity along the x direction for a flat plate using conventional boundary conditions

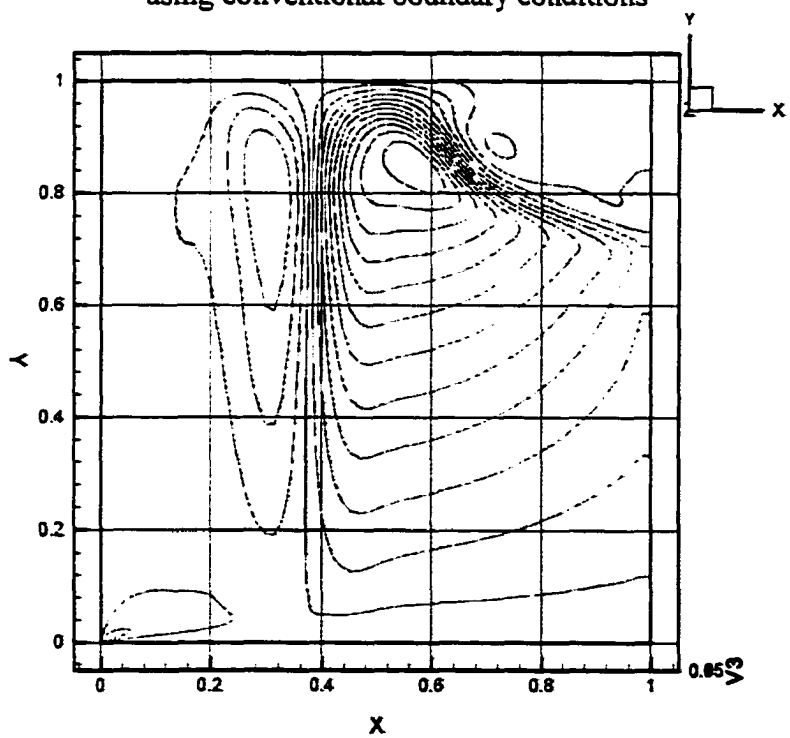


Fig 2.7 Contour of velocity along the y direction for a flat plate using conventional boundary conditions

The second example was tested for a cylinder with infinite length in the span-wise direction. As mentioned in the first case, the periodic boundary condition was used in this direction. The grid size for this problem is 151x101x11. The grid in the (x,y) plane is shown in Fig.2.8. Usually, in computational fluid dynamics, the far field boundary is about 5 to 10 times the characteristic length from the object surface. Here, in order to show the advantage of the non-reflecting boundary condition, we choose the far field boundary which is quite close to the object. As we see in Fig.2.8, the characteristic length of this configuration is the radius of the cylinder (characteristic length=1). The far boundary is just about one characteristic length away from the object surface.

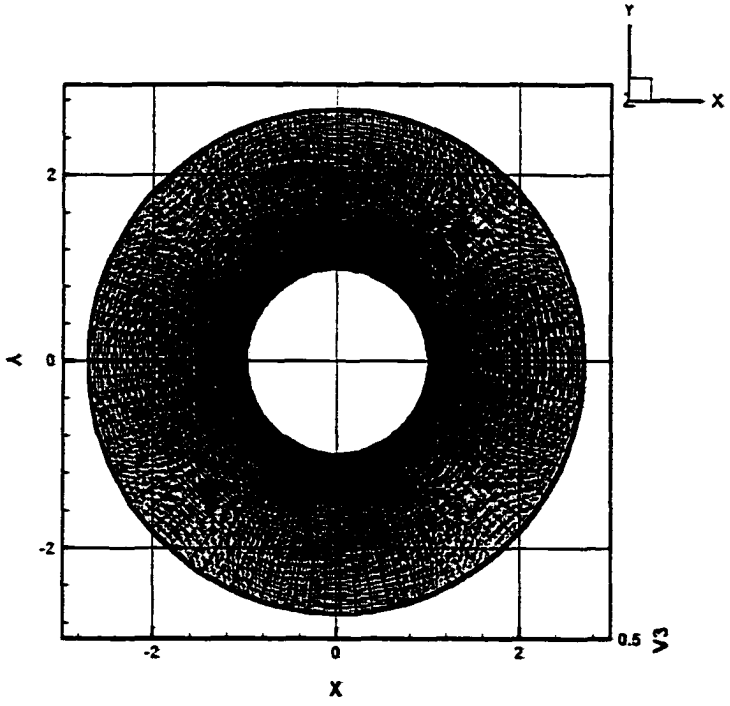


Fig 2.8 Section of grid for a cylinder of the (x,y) plane

First, the conventional boundary condition which uses 1-D Rieman analysis to specify the boundary condition is used at the far field boundary. In order to compare the results, we use the non-reflecting boundary condition at the far field boundary. On the

surface, because the flow speed at the solid wall surface is zero, we use the non-slip non-reflecting boundary condition as we derived in the last section. The effect of the boundary condition can be seen in Fig.2.9 to Fig 2.12. Fig 2.11 is the calculation result by using 1-D Rieman analysis as boundary condition, and Fig 2.9 is the calculation result by the non-reflecting boundary condition. Both of these figures show the velocity contour in the x direction. In Figure 2.11, it is clear that at the far boundary there are some waves entering the computational domain. But Figure 2.9 shows the results of the non-reflecting boundary condition, which have no reflection of waves because the flow field is very clean. It is obvious that the result using a non-reflecting boundary condition is much better than the one using a conventional boundary condition. Fig 2.10 and Fig 2.12 show a similar comparison. The only difference is that they are velocity contours in the Y direction. From these two figures, we can get the same conclusion.

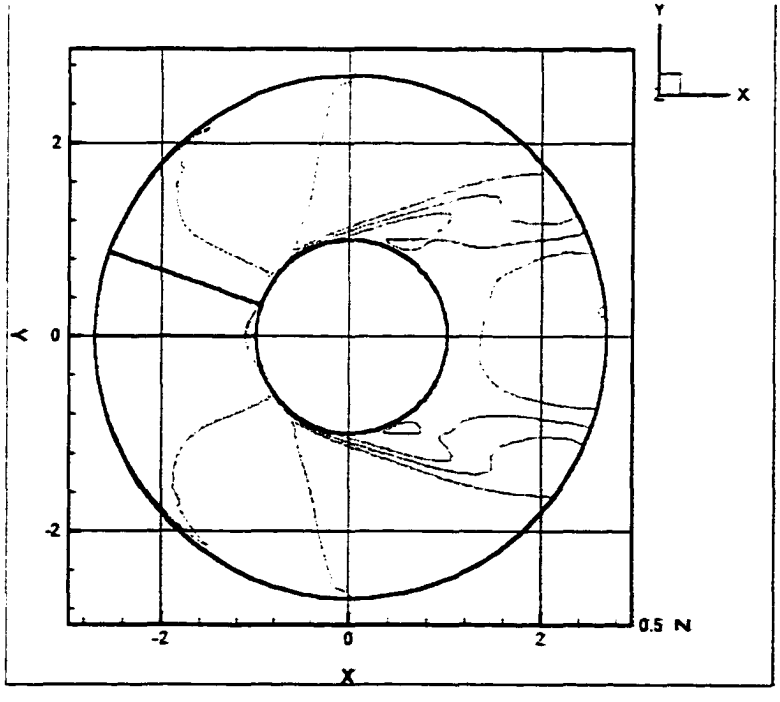


Fig 2.9 Velocity contour (X direction)
using a non-reflecting boundary condition

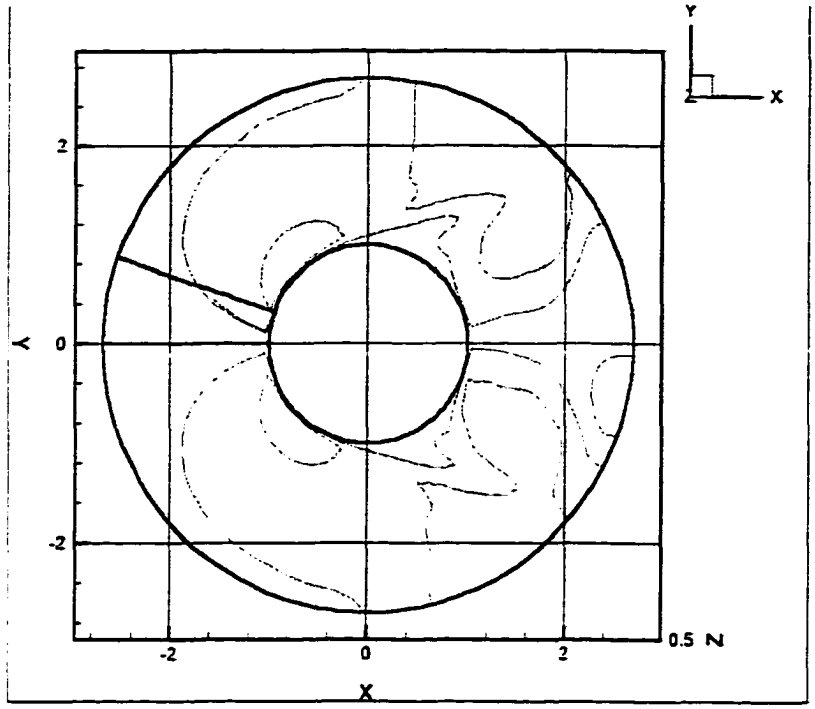


Fig 2.10 Velocity contour (Y direction)
using a non-reflecting boundary condition

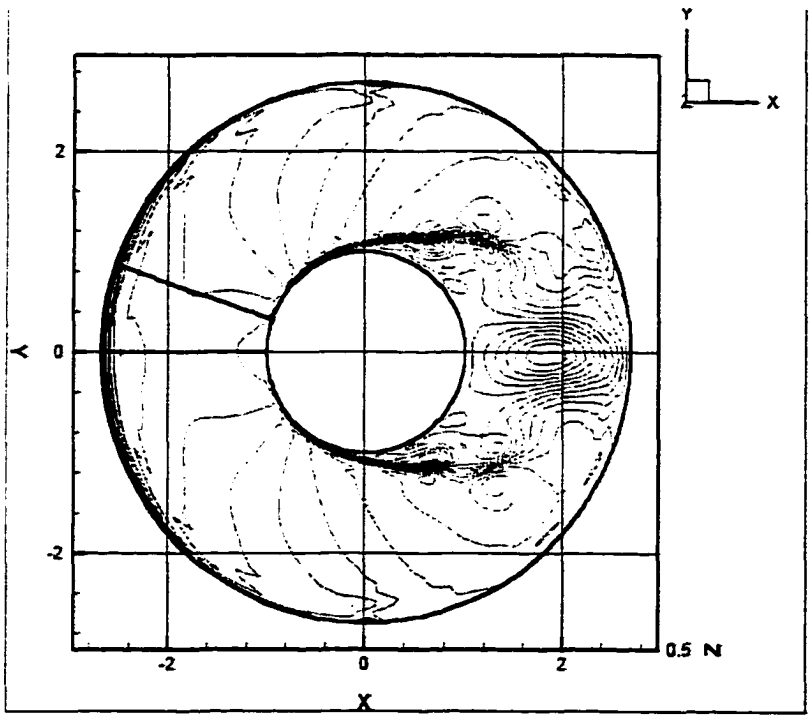


Fig 2.11 Velocity contour (X direction)
using 1-D Rieman analysis at the boundary

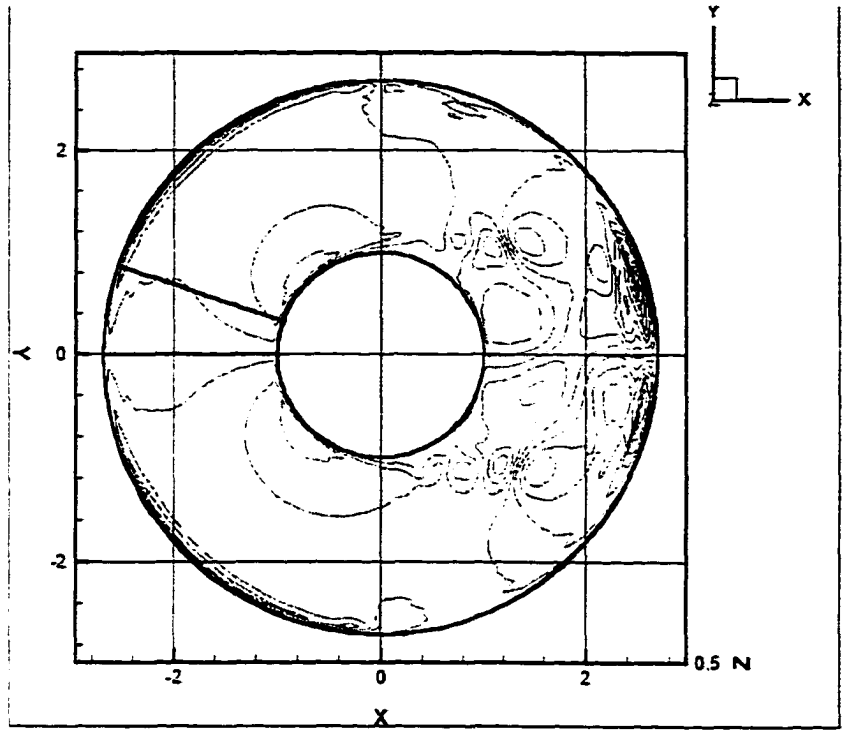


Fig 2.12 Velocity contour (Y direction)
using 1-D Rieman analysis at the boundary

From the above tests, it is obvious that the non-reflecting boundary condition is superior to the conventional boundary conditions, especially for high accuracy numerical simulation like the DNS.

2.5. Appendix: compution of derivatives

The calculation of time derivatives in the interior consists simply of approximating the spatial derivatives in the above equations (2.2.18a)--(2.2.18e) with suitable numerical schemes. There are many approximation methods available, but we choose finite difference methods here. The accuracy of the finite difference method is described by orders. The higher the order, the more accurate the approximation.

A globally 4-th order accurate scheme may be achieved by using the 4-th order

approximations

$$\left.\frac{\partial f}{\partial \xi}\right|_{i,j,k} = \frac{1}{12\Delta\xi}\left[8\left(f_{i+1,j,k} - f_{i-1,j,k}\right) - \left(f_{i+2,j,k} - f_{i-2,j,k}\right)\right],\ i = 2,\ldots,N-2 \quad (2.5.1)$$

$$\left.\frac{\partial f}{\partial \xi}\right|_{0,j,k} = \frac{1}{6\Delta\xi}\left[18\left(f_{1,j,k} - f_{0,j,k}\right) - 9\left(f_{2,j,k} - f_{0,j,k}\right) + 2\left(f_{3,j,k} - f_{0,j,k}\right)\right] \quad (2.5.2)$$

$$\left.\frac{\partial f}{\partial \xi}\right|_{1,j,k} = \frac{1}{6\Delta\xi}\left[2\left(f_{1,j,k} - f_{0,j,k}\right) + 6\left(f_{2,j,k} - f_{1,j,k}\right) - 3\left(f_{3,j,k} - f_{1,j,k}\right)\right] \quad (2.5.3)$$

$$\left.\frac{\partial f}{\partial \xi}\right|_{N-1,j,k} = \frac{1}{6\Delta\xi}\left[2\left(f_{N,j,k} - f_{N-1,j,k}\right) + 6\left(f_{N-1,j,k} - f_{N-2,j,k}\right) - \left(f_{N-1,j,k} - f_{N-3,j,k}\right)\right]$$

(2.5.4)

$$\left.\frac{\partial f}{\partial \xi}\right|_{N,j,k} = \frac{1}{6\Delta\xi}\left[18\left(f_{N,j,k} - f_{N-1,j,k}\right) - 9\left(f_{N-1,j,k} - f_{N-2,j,k}\right) + 2\left(f_{N,j,k} - f_{N-3,j,k}\right)\right]$$

(2.5.5)

Similar expression can be obtained in η, ζ directions.

At the boundary, the normal derivatives are subsumed into the definitions of the L_i quantities. Those L_i, for which the corresponding characteristic velocities λ_i are directed out of the computational domain, are evaluated from their deinfition using one-sided approximation which is shown above. The remain L_i values are determined from the boundary condition fomulas in the last section.

CHAPTER 3

Parallel Computation by MPI

Abstract

A parallel spatial direct numerical simulation code is developed to simulate the spatial evolving disturbances associated with the laminar-to-turbulent transition in a compressible boundary layer. MPI (Message Passing Interface) is employed to parallelize all processes for a distributed memory parallel computer. Explicit time stepping is used in the DNS code on IBM/SP2 to simulate the flow transition. The machine-dependent phenomenon, which is always considered a problem for parallel computation, is successfully avoided. A fundamental breakdown on a flat plate boundary layer transition at Mach 0.5 is then studied using this code. The results demonstrate the optimistic future of MPI to direct numerical simulation.

3.1. Introduction

The demands of both the scientific/engineering and the commercial communities forever increasing computing power have led to dramatic improvements in computer architecture. Initial efforts were concentrated on achieving high performance on a single processor, but the more recent past has been witness to attempts to harness multiple processors, with massive power being obtained through replication.

For the most part, users of parallel computing systems tend to be those with large mathematical problems to solve, with the demand for power reflecting a desire to obtain results faster and/or more accurately. Unfortunately, the existing numerical algorithms, on which we have come to rely were developed with a single processor in mind, and the transition from a serial to a parallel environment is therefore not straightforward.

3.1.1 Parallel computer

Until relatively recently, the standard architecture model for most digital computers was introduced by von Neumann. The von Neumann model assumes that the programs and data are held in the store of the machine and that a central processing unit (CPU) fetches instructions from the store and executes them. The instructions result in either the store being manipulated or information being put or output. Machines based on this model are entirely sequential in operation -- one instruction is executed in each time interval.

The first general-purpose electronic digital computer, called the ENVIC, was developed at the University of Pennsylvania in 1946. The past five decades have been witness to dramatic improvements in computer technology. The improvements were partly the result of advances in semiconductor technology, but also arose from an evolution of the original von Neumann model in which the requirement of purely serial processing was abandoned.

The development of parallel computers has been at two levels, both of which have attempted to enhance the performance of a particular class of machine. Early efforts centered on the development of high performance supercomputers to solve very large

scientific problems, such as fluid dynamic simulation, weather forecasting and so on. The most popular supercomputer was the Cray series. The eight-processor Cray Y-MP and the four-processor Cray-2 of the late 1980s are capable of Gflops (gigaflops, or thousands of Mflops) performance. These machines exhibit limited parallelism with just a small number of powerful processing units operating in parallel.

The second strand of parallel computer development has centered around the desire to produce machines that are capable of performance approaching that of a supercomputer, but at a considerably reduced cost. These machines achieve their performance either by using vector processing capabilities, or by including a number of parallel processing units, or both. Thus, parallelism in computing is not only present in supercomputers, but also increasingly common in less powerful machines.

It is instructive to relate these developments in computer architecture to the computer solution of numerical problems. Throughout the decades 1950--1990, the architectural development was, for the most part, invisible to the user. Programs which worked optimally on one machine were likely also to work well on other systems. The only hardware feature of which the programmer might need to be aware of was the available memory. Virtual memory aided portability; programs simply had to minimize the amount of data transfer between main and secondary storage. The development of units possessing vector processing capabilities affected the performance of an implementation. By making the vector structure of the algorithm visible to the compiler, significant improvements could be obtained over a corresponding code for which this structure was not apparent. The impact of the computer on the user is considerably greater in the case of a multiprocessor system and is compounded if its memory is distributed

over the individual processors. Such architectural considerations crucially affect the choice of algorithm and the way that an algorithm is expressed [Freeman, 1992]. It is this aspect of parallel processing that we take as our principal theme.

The demand for increased performance may result from a desire

- to decrease the execution time of certain programs so that results can be obtained in a reasonable time, or
- to run programs for the same amount of time but obtain higher accuracy or more accurate modeling of the underlying physical problem by increasing the problem size in some way, or
- to solve a problem previously considered too large for existing architectures.

One of the standard ways of classifying computer systems is that proposed by Flynn in 1968 according to the number of instruction streams and the number of data streams. The classical von Neumann machine has a single instrument and single data stream and is identified as **single-instruction single-data (SISD)** machine. At the opposite extreme is the **multiple-instruction multiple-data (MIMD)** system, in which a collection of autonomous processors operates on their own data streams.

The classical von Neumann machine is divided into a CPU and main memory. But no matter how fast the CPU is, the speed of execution of programs is limited by the rate at which we can transfer the sequence of instructions and data between memory and the CPU.

Multiprocessor systems consist of a number of interconnected processors. Each of them is capable of performing complex tasks independently of the others. An individual processor, or node, may be a scalar or vector processor, or even a multiprocessor. Fig-3.1

is one of MIMD systems in which each processor has its own memory and can be regarded as a single machine. They are called the distributed-memory MIMD. In particular, this system is asynchronous. There is often no global clock. The processors are specifically programmed to synchronize with each other if processor needs to be synchronous. Processors are connected by the interconnect network. Through the network, the processors can communicate with each other. It can be seen directly that the speed for a processor which needs only local memory data is much faster than the speed of a processor which needs data stored in an other processor's memory. Since the data must be gotten from an interconnect network, the communication will slow down the process speed of the processor. Therefore, it is a good strategy to use as much local memory as possible.

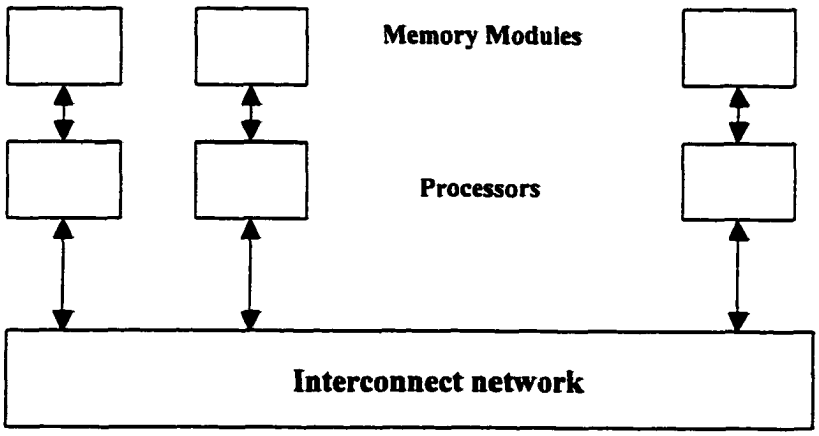


Figure 3.1 MIMD Structure

Because the inter-connected network is much slower than the processor itself, scientists spend a lot of time studying the structure of the network. Physical connection between processor and memory may take the form of a common bus, or the processors themselves may have communication capabilities which allow them to be directly connected together. Alternatively, some sort of a switching mechanism may be employed.

The most commonly used networks are cross bar, clos connection, hypercubes, and fat tree. They are used depending on the purpose of the computer via a cost and speed factor. Switch is one that is able to connect a number of inputs with a number of outputs according to specified permitted combinations.

3.1.2. Software support

As numerical problem solvers, we have become accustomed to the provision on a uni-processor of a number of sophisticated tools to aid program development. In a multiprocessor environment we expect to have these and more. At the very least, we anticipate

- an operating system,
- high-level programming languages supporting parallelism,
- compilers which generate efficient code from programs written in these language, and
- libraries of numerical software.

From the operating system, we might expect the capability to allocate processes to processors dynamically. From the programming languages, we must be able to express the parallelism for our problems which will help to identify parallelism inherent in existing sequential codes. If a numerical library is provided which itself exploits parallelism, then we may be able to develop a parallel code by writing a sequential main program which simply makes calls to appropriate parallel routines. In our work, we choose this one as our tools to do the parallel computation for DNS.

For any parallel language designed for use of a local memory system, we expect

facilities for process creation, destruction, and identification. In addition, we require the ability explicitly to communicate between processors. Specifically, we may wish to

- send a message from one processor to another, and
- receive a message in one processor from another

where a message involves some transfer of data. For example, we might wish to accumulate a sum of values, each of which is determined by a separate process. Assume that a single process is to form the sum; then all other processors must forward their values (pass messages) to that process. Besides these one-to-one message-passing primitives, it is likely that we will also require facilities to

- send a message from one process to all other processes (broadcast or scatter)
- receive a message in one process from all others (gather).

We carefully distinguish two types of communications, synchronous communications and asynchronous communication. By synchronous communications, the sending process is held until the corresponding receiving process is ready. Similarly, the receiving process is held until the corresponding sending process is ready. By asynchronous communication, the sending process is not held waiting for the receiving process. If the receiving process is not ready, then the data to be transferred are put into a buffer until the message transfer can be completed.

Associated with the creation of processes is the placement of each process on a given processor. In a local memory environment, this placement is usually undertaken by the programmer, rather than left to the system software. Hence, additionally, we might expect facilities to

- associate a process with a specified processor, and, possibly,

- associate an inter-processor communication link with physical inter-processor link.

This process-to-process allocation implicitly maps data requirements to local memory.

When developing parallel programs within a message-passing environment, the physical structure of the computer system on which the code is to run may need to be borne in mind, although it is desirable to minimize the dependence of the code on the structure's topology. Communication between processes resident on adjacent processors is straightforward and takes place along the link between the processes running intermediate processors to pass the message on. Ideally, we would like the need for such throughrouting to be hidden from the programmer. With throughrouting communication software, the programmer can nominate a process to receive the message and then leave it to the system to ensure that the message arrives at the correct destination.

3.2 Message-passing style parallel programming with MPI

MPI stands for Message Passing Interface. MPI uses different approaches to achieve parallel computation. Unlike High Performance Fortran, MPI does not develop any new languages. It specifies a library of functions that can be called from a C or Fortran program. The foundation of this library is a small group of functions that can be used to achieve parallelism by message passing. A message-passing function is simply a function that explicitly transmits data from one process to another. Message passing is a powerful and very general method of expressing parallelism. A message-passing program can be used to create extremely efficient parallel programs, and message-passing is

currently the most widely used method for programming in many types of parallel computers. Its principal drawback is that it is very difficult to design and develop programs using message-passing. Indeed, it has been called the "assembly language of parallel computing" because it forces the programmer to deal with so many details. In spite of this, the history of parallel computing suggests that it is sufficiently deliberate. It does not take an undue effort to design extremely sophisticated programs. Furthermore, as more and more software is developed that uses MPI, more and more sophisticated algorithms will be encapsulated in portable MPI libraries. The inclusion of these algorithms into a program will be simply a matter of calling a function.

The reason we choose MPI as our parallel tool is because it has the following characteristics:

- a standard library for message-passing (communication) among multiple processors,
- portability, which supported by almost all parallel computers (including network of PCs and workstation),
- Efficiency, which is more efficient than a shared-memory style programming,
- Program re-usability, where a significant portion of an existing program needs to be modified.

3.3. Basic operation of MPI

MPI is the most commonly used method for programming distributed-memory MIMD systems. But a characteristic which should be noted here is that MPI is a kind of single-program multiple-data program (SPMD). It means source code written on different processors is the same. But when they are executed, the programs will be executed in

different ways. The effect of running different programs on different processors is obtained by the use of conditional branches within the source code. This is the most common approach to program MIMD systems.

In basic message-passing, the processes coordinate their activities by explicitly sending and receiving messages. The process must specify the communication processes when it wants to communicate with them. The communication between processes can be either in blocking type or non-blocking type. For example, when process 1 calls function "receive" if the message for process 1 is still not available, process 1 will remain idle until the message is available. This is the block type communication. An alternative communication type is non-blocking communication, that is, the process returns immediately after the call. The system would be responsible for the remained work. The use of non-block communication can provide dramatic improvements in the performance of message-passing programs. If a node of a parallel system has the ability to compute and communicate simultaneously, the overhead caused by communication can be substantially reduced. For example, if each node of the system has a communication coprocessor, then we can start a non-block communication and perform computations that do not depend on the result of the communication, and when the computations are completed, finish the non-block communication. While the computations are being carried out, the communication co-processor can do most of the work required by the non-block operation. Since communication is very expensive relative to computation, overlapping communication and computation can result in tremendous performance gain.

The most commonly used MPI functions are listed in the appendix.

3.4. Finite differences solver for DNS

The mathematical model for the DNS problem is to solve the Navier-Stokes equation. The 3-D, compressible, time-dependent Navier-Stokes equations under the general curvilinear coordinate system can be written as below:

$$\frac{\partial q}{\partial t}+\frac{\partial F}{\partial x}+\frac{\partial G}{\partial y}+\frac{\partial H}{\partial z}=\frac{1}{\mathrm{Re}}\left(\frac{\partial F_v}{\partial x}+\frac{\partial G_v}{\partial y}+\frac{\partial H_v}{\partial z}\right)$$

Because of the sensitivity of physics, DNS code must meet the requirement of both high accuracy and strong numerical stability. High-order discretizations have been developed by several researchers (Orszag, 1971; Lele, 1992; Rai and Moin, 1993). Here, we use the standard 6th-order central difference for convective terms and 4th-order central difference for viscous terms. The 4th and 6th order of the difference formula are as follows.

$$\frac{\partial f}{\partial x}=\frac{1}{12\Delta x}\left(-f(x_{i+2})+8f(x_{i+1})-8f(x_{i-1})+f(x_{i-2})\right)$$

$$\frac{\partial f}{\partial x}=\frac{1}{60\Delta x}\left(f(x_{i+3})-9f(x_{i+2})+75f(x_{i+1})-75f(x_{i-1})+9f(x_{i-2})-f(x_{i-3})\right)$$

where f can be any of the flux vectors and x can be one of three coordinates.

For time integration, we adopt the classical 4th-order Runge-Kutta scheme. We can rewrite the semi-discrete Navier-Stokes system as

$$\frac{d}{dt}U_{i,j,k}+R(U_{i,j,k})=0$$

R is the convection and diffusion term. By the Runge-Kutta method, the Navier-Stokes equations can be expressed as follows, where Δt is the time interval, and the upper

index n is the time at nΔt:

$$U^{(0)} = U^{(n)}$$

$$U^{(1)} = U^{(0)} - \frac{\Delta t}{2} * R^{(0)}$$

$$U^{(2)} = U^{(0)} - \frac{\Delta t}{2} * R^{(1)}$$

$$U^{(3)} = U^{(0)} - \Delta t * R^{(2)}$$

$$U^{(4)} = U^{(0)} - \frac{\Delta t}{2} * (R^{(0)} + 2R^{(1)} + 2R^{(2)} + R^{(3)})$$

$$U^{(n+1)} = U^{(4)}$$

It is obvious that the system is parabolic. When we want to calculate variable values at time step n+1, we need only variable values at time step n. Therefore, this method can be efficient in memory utilization by a careful arrangement of the array of the code. Also, by this parabolic property, we can merge the solver with the parallel method smoothly.

The above system can be solved with proper specifications of boundary conditions. In general, all flow quantities can be specified at the inflow boundary. The prescription of the inflow boundary condition depends on the ways that the disturbances are introduced into the base flow. Because we use a flat plate as our calculation example, all the boundary conditions are specified for this case.

At the wall, we give the no-slip and isothermal boundary condition. At some position from the inflow boundary, the blowing/suction condition is enforced to simulate the flow transition. At inflow, outflow, and far field boundary, sponge lay is used to eliminate the wave reflection. In the span-wise direction, the periodic boundary condition is imposed. This boundary condition can be specified during solving the system by the Runge-Kutta method.

3.5 Parallel implementation of flow solver

There are essentially two approaches to design parallel programs. In the first, called the data-parallel approach, we partition the data among the processors, and each processor executes more or less the same set of commands on its data. In the second, called the control-parallel approach, we partition the tasks we wish to carry out among the processes, and each processor executes commands that are essentially different from some or all of the other processors.

Data-parallel programming is more common. Perhaps the most important thing is that data-parallel is scalability, that is, this approach can be used to solve larger problems with more processes. Here, we choose this approach to do parallel computation.

Because of the nature of the explicit flow solver, variables at a grid point can be solved independently. The parameter it needs is the variables' value at the last time step. It can be seen from the algorithm of the forth-order Runge-Kutta method we used. Even it has 4 stages, but each stage uses at most the last time step/stage's variables' value. Therefore, for each grid point, its variables can be solved individually. Since the method is explicit, for a fixed time step (n+1), we let IDM, JDM, KDM be the grid point numbers in the ξ, η and ζ direction, respectively. The approximate solution values U(n+1, i,j,k) , i=1,...,IDM; j=1,...,JDM; k=1,...,KDM are independent of each other, and may be computed concurrently. A natural work-sharing strategy is to employ some forms of domain decomposition in the space dimension. Typically, we would organize the grid points into blocks. The beginning of the computation at each time step imposes a synchronization point which, in a local memory environment, requires process intercommunication of the approximate solution values at the edges of the space

discretization blocks.

Based on the above analysis, we perform a domain decomposition. This means assigning grid points to different processors. Each processor is responsible for the calculation of some grid point's variables' value. In order to maintain a good load balance among multi-processors, the amount of computation processed by each processor should be as equal as possible. Here, because the computation is related to each point of the grid, how to distribute these points among processor, or how to partition the computational domain is a very important aspect in our parallelism. By the finite difference method which we adopted, the 3-D computational domain can be considered as a box in the (ξ, η, ζ) space. The physical coordinates at each corresponding point are consistent during the calculation. That means the problem can be distributed among processors by simply partitioning the entire domain and assigning the sub-domain to each process. Each process performs all the calculation that is almost the same as that of the original serial program. The only difference is that the computation in each processor will be in a small scale -- a subset of the global domain.

A 3-D grid in the computational (ξ, η, ζ) space is shown in Fig.3.1. In the flat plate problem, because we use less grid point in span-wise (J-direction) than the other two (I, K) directions, so we do not partition data in the J-direction.

Domain decomposition can be done in several ways (C.F., Vidwans, and Kallinders, 1993). In the present work, we considered the following two instances of grid partition.

1. "Strip" partition (Fig.3.2(a)):

In this kind of partition, the global domain is divided along one of the coordinates. Data in different sub-domains are distributed to different processors. Data are not partitioned along the other two directions. The cutting planes are all along one of (I,J,K) directions.

2. "All-round" partition (Fig.3.2(b)):

The global domain is divided along I and K directions into smaller sub-domains. Each sub-domain contains part of the original data and is assigned to different processes.

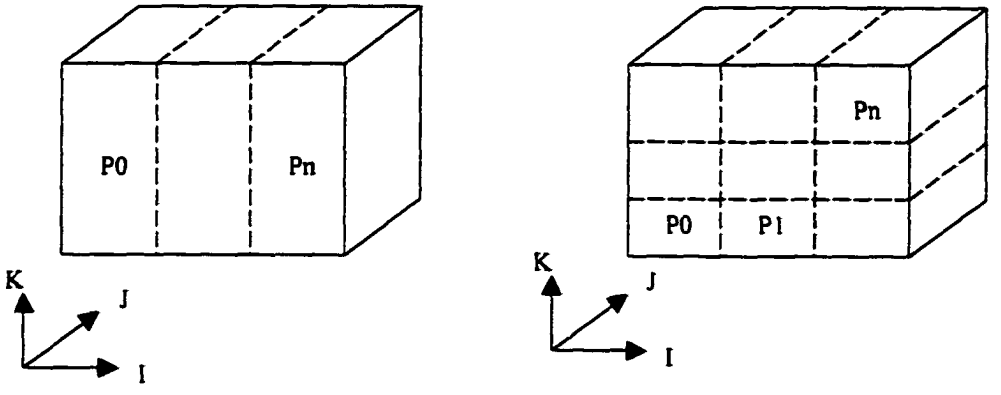


Fig.3.2 Domain partition: (a) strip partition, (b) all-round partition.
P0 ... Pn represent for the process

By defining a suitable number of processes in each axis of the coordinates, we can achieve a good load balance. Both kinds of partition can be obtained without significant problems. Definitely, the first partition is easy to achieve good load balance, while the second one needs some considerations before domain partition. Load balance can be achieved in the same way. The only difference is that the interface of a sub-domain may have a variable number of grid points. In the following sections, we will see that the area of interface is related to the amount of communication. Therefore, this area is critical for

efficient computation. Each process does the same calculation as the serial program, but it does the calculation on the sub-domain, a smaller region than the original domain. That means that each process will spend less time to get the results in the sub-domain. Because all processes do the computation in parallel, the computation time will be significantly saved compared to a serial program running on one process.

3.6. Communication and load balance

If we wish to optimize the efficiency of a particular applications program, then it is essential to ensure that all processors are doing useful work for as much time as possible. This means that, as far as possible, we must avoid processors being held at synchronization points waiting for information from other processors before they are able to be processed. Clearly, in this sense, a sequential program running on a single processor is 100% efficient. The aim is to implement an algorithm in such a way as to employ as many processors as possible while at the same time ensuring that all the processors are sufficiently usefully active. We refer to this action as load balancing.

Communication is a big issue in parallel computation. Compared to the serial program, a parallel program has three main sources of overheads:

1. communication;
2. idle time;
3. extra computation.

Idle time and extra computation can be minimized by the arrangement of domain decomposition. But communication cannot be avoided. Among the three overheads, the communication is the main contribution of overheads, especially in large scientific

computation when communication is repeated over and over. So we must carefully design our parallel algorithm before solving the problem. Because we use a high-order difference scheme to do DNS, which has a larger stencil as the order of numerical scheme increases, the need for data communication will also increase because of the expanding of the stencils. By using the explicit scheme, only the information at neighbor processors at a previous step is needed. Because of the characteristics of an MIMD machine, each processor has its own local memory. At the end of each time step, values at certain grid points will need to be transferred to an adjacent processor. A common situation is that each processor holds (m*JDM*n) grid points. Here, m, n are dimensions of a sub-domain in I, K directions. We denote by "internal boundary values" for those values that are needed by other processors at the next time step and must be transmitted. This is illustrated in Fig. 3.2.

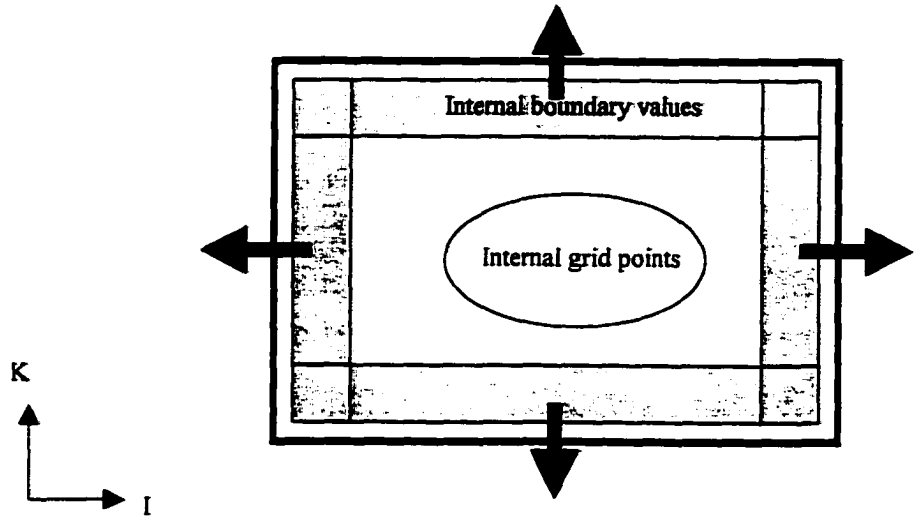


Fig.3.3 Communication of internal boundary

The shaded areas are these internal boundary values which need to be transferred to adjacent processors. The arrows indicate the transfer direction. For the internal grid

points in Fig. 3.3, one still can use a serial algorithm to obtain the values. But, for the grid points in the above shaded areas, when one calculates the value at these points, one also needs the neighbor points' values which are stored in the adjacent processors. Therefore, for these points, there are two actions that must be done: 1) sending their values to their adjacent processors, 2) receiving values from their adjacent processors. After these two steps are finished, their values at a new time step can be obtained as a normal difference algorithm implementation. In our program, because a 6th-order or a 4th-order difference scheme was used, more grid points' values need to be transferred. We can see this in Fig.3.4

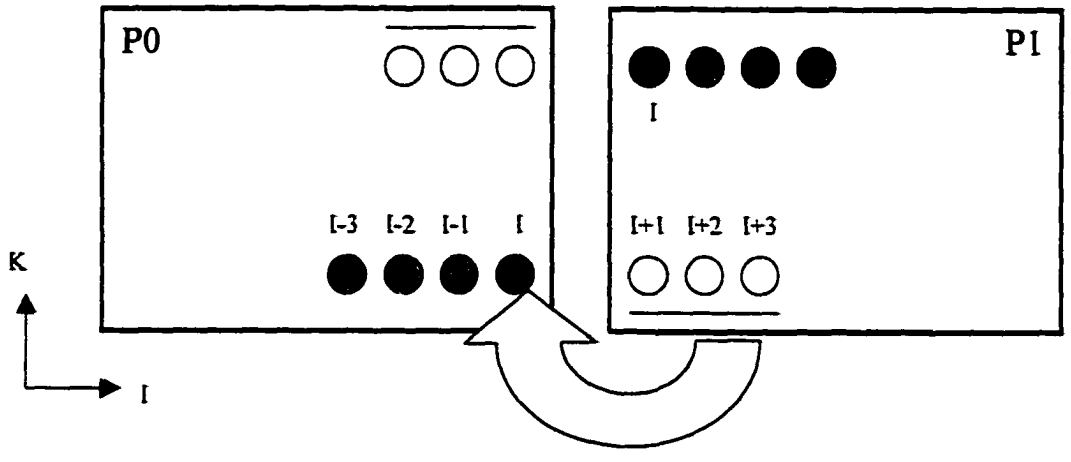


Fig 3.4 Communication between adjacent processors

P0 and P1 are adjacent processors. If we want to calculate the value of grid point (cross with a circle in Fig.3.3) near the boundary of the sub-domain, seven points at the last time step are needed. We label this point with "I" and the other points with I-3, I-2, I-1, I+1, I+2, and I+3. It can be seen that, if we want to calculate the "I" point's value at processor P0, I+1, I+2 and I+3 points are in the processor P1. Therefore, they must be

transferred to P0. The problem is similar for the point I at processor P1. Since the communication takes a lot of time, it will be very time consuming if we transfer data point by point. Here, we use data type in MPI. The internal boundary values for an adjacent processor are packed together as a data type. Usually, we define two data types in MPI to send and receive several rows or columns of data. In this way, data transfer between processors will be performed just by sending or receiving the defined data type only once. Typically, one sub-domain has north, south, east, and west neighbors. For each processor, at least four communications must be made in order to calculate the spatial derivatives in the I and K directions.

As we have seen, the data amount for communication is proportional to the intersection area between neighbor processors. For strip partitioning, the area between processors is always the same. This means that no matter how many processors we use, the communications between two adjacent processors are the same. Therefore, efficiency is low and will become the bottleneck for our problem as the number of processors increases. For all-round partitioning, as the number of processors increases, the sub-domain become smaller and smaller. Thus its interface area will become smaller and smaller. The data communication between this processor and other adjacent processors will decrease as the number of processors increases. Therefore, the total overhead time will be reduced. Hence, based on the above analysis, we use this kind of partitioning in our present work.

3.7. Parallel efficency and performance

The efficiency of a parallel code is a measurement to verify how well the code utilizes the available processors to solve the problem. The best way is to make all processors work in exactly the same size in order to avoid the situation that some processors are busy while the others are idle. One way to consider the efficiency is to hold the global problem size and let the number of processors increase. The wall-clock time for solving the problem is measured as the number of processor increases. Another way to measure the efficiency is to hold the size of the problem and check the total run time.

Let T_p be the execution time for a parallel program on p processors. We define the following terms:

1. S_p, the algorithm speed-up ratio on p processors, is given by

$$S_p = T_1 / T_p$$

This quantity measures the speed-up to be gained by the parallelization of a given algorithm. It thus directly measures the effects of synchronization and communication delays on the performance of a parallel algorithm. Ideally, we would like S_p to grow linearly with p. Unfortunately, even for a good parallel algorithm we can only, at best, expect the speed-up initially to grow at a close to linear rate. Eventually, it will decline because the overhead will increase as the number of p increases.

2. ε, the efficiency on p processors, is given by

$$\varepsilon = T_1 / (p * T_p)$$

Note that efficency is compared by dividing time consumed on one processor with the total time consumed on p processors. In practice, the efficiency should decrease as p

increases. Because, in parallel computation, there are some overheads as compared to computation on one processor. The more processors, the more overheads occur. However even though there are some overheads for parallel computation, the total computation time can be decreased significantly.

To test the performance of our MPI parallel code, several combinations of processors and grid sizes are used. Fig. 3.4 shows the efficiency versus the number of processors for the grid size of 481*32*81. Very good performance is achieved when the number of processors increases to as many as 64.

The timing results with different grid sizes are depicted in Fig. 3.5, from which we can see that the acceleration is almost proportional to the number of processors.

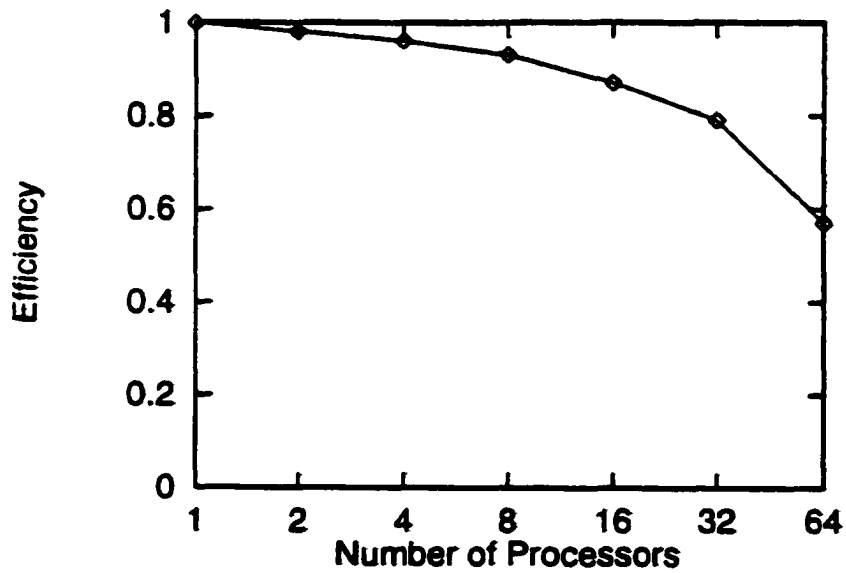


Fig. 3.5 Computing effiency

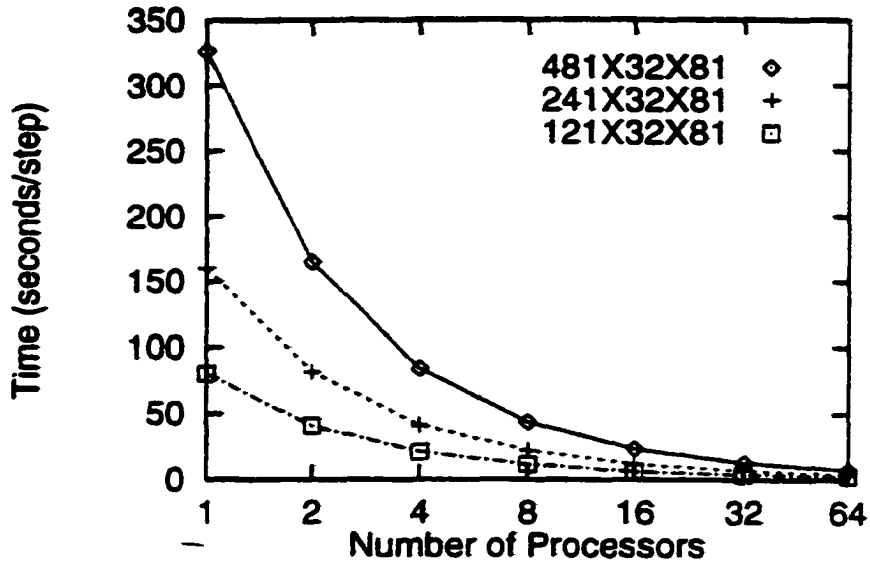


Fig. 3.6 Computing Time

Another test is to use different grid sizes with fixed number of processors. Fig. 3.6 describes the performance of the current MPI code under the IBM/SP2. Here, the basic grid size N_{g0} is set to 61*32*41. A series of multiple N_{g0} grid points, N_g=61*32*41, 121*32*41, 121*32*81, 241*32*81, 481*32*81, are assigned. Three curves with 4, 16, and 64 processors respectively, are plotted in Fig. 3.7. Again, the acceleration is found to be almost proportional to the number of processors.

The performance of fixed per-processor grid size is also tested. Though it is expected that only a slight decrement in the performance occurs, it is difficult to get all the processors dedicated when the number of processors increases to a certain number, which might be the major reason of slowing down, as shown in Fig. 3.8

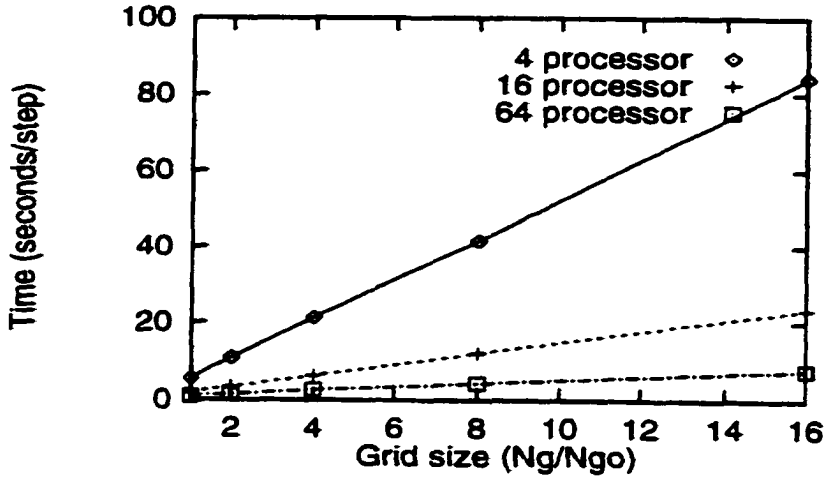


Fig. 3.7 Computing time with a fixed processors

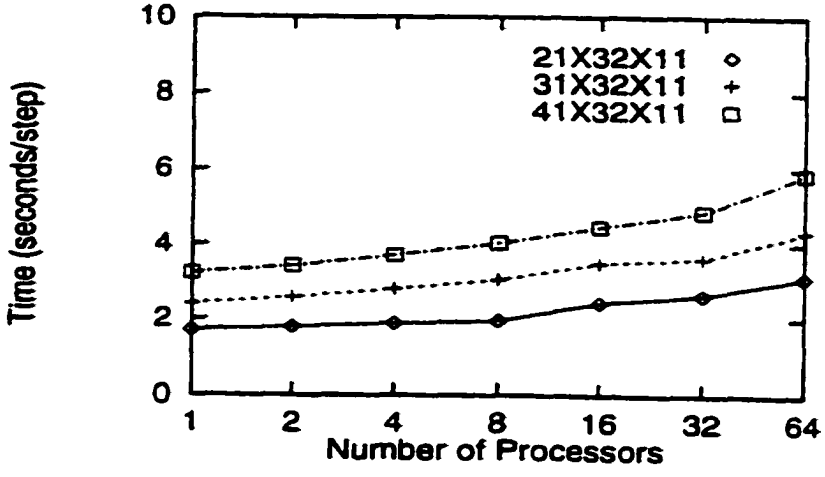


Fig. 3.8 Computing time with a fixed per-processor grid size

Besides the above tests, a Sun Enterprise 3000 with shared memory and six processors is also used to test the portability of our MPI code. A similar performance is observed.

3.8 Numerical Results

The MPI code is first validated with the Mach number 0.5 for the flat plate boundary layer transition. A sketch of the computational domain is given in Fig. 3.9. The

base flow is obtained by solving the compressible similarity system (Stewartson, 1964) using shooting method. The solid wall is assumed to be adiabatic for the base flow and isothermal for the perturbation. The Reynolds number is set to Re = 875 (based on δ), and $\omega_{2d} = 0.1$. Compressible linear stability theory (LST) provides an eigenvalue $\alpha = 0.2636 - i0.005623$.

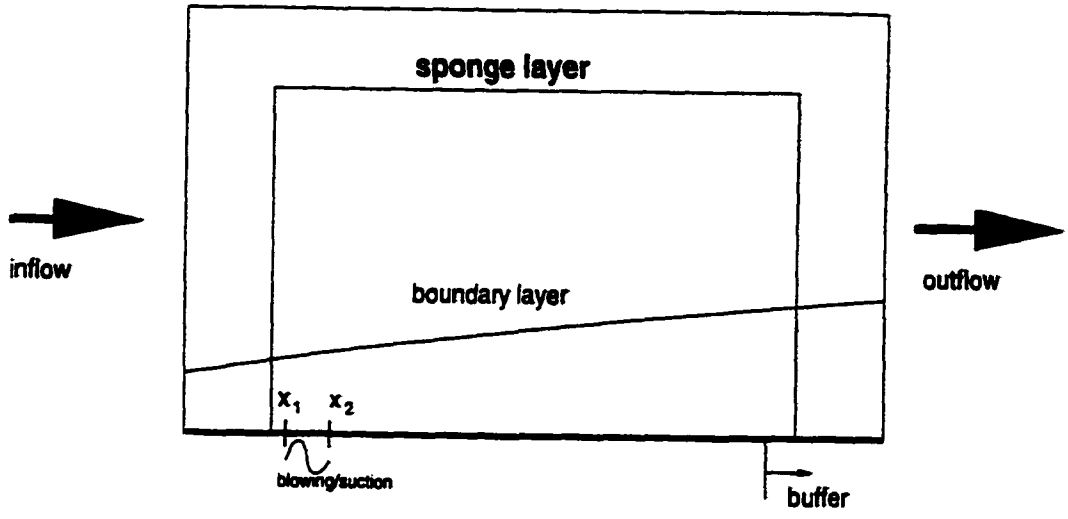


Fig. 3.9 Computational domain of a boundary-layer transition problem

The whole computational domain is set to 15 T-S wavelengths to ensure the development of the least stable mode. The grid is 16/wave*51*1, with the last wavelength used as the buffer domain (sponge layer). Also, one T-S period is divided into 1000 time steps. The base flow is assumed to be parallel. Fig. 3.9 depicts the disturbance amplitude of u' and v', showing that the least stable mode is picked up very well and grows as LST predicted. Also, we can see that the sponge layers successfully eliminate the reflecting waves in all directions to keep the physical domain very clean. Eigenfunctions of this case are also compared with LST and found to agree very well (Fig. 3.10).

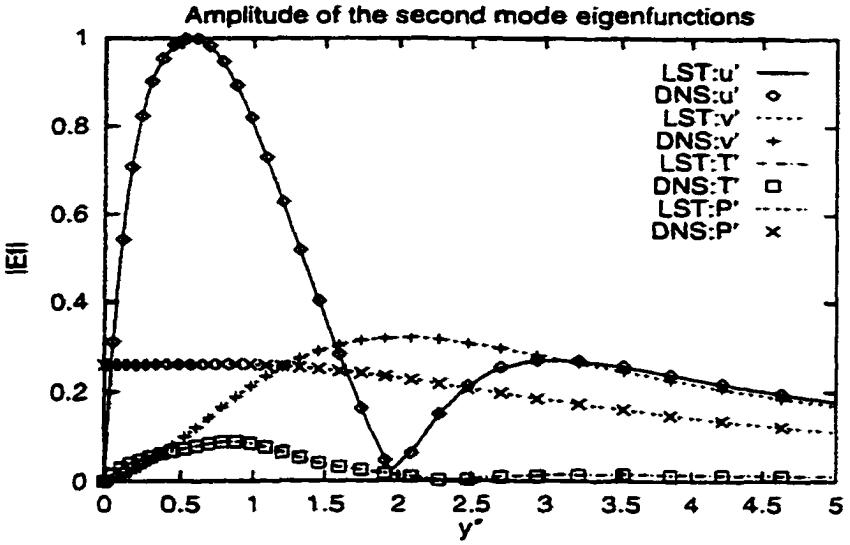


Fig. 3.10 Comparison between DNS and LST of the distriburbance amplitude of u' and v'

A K-type transition in a 2-D subsonic flat plate boundary layer is then simulated by using the same code. A 353*51*32 grid, which includes an inflow sponge (first 16 streamwise grid points), a suction/blowing slot (near 24 streamwise grid points), an outflow buffer domain (the last 16 streamwise grid points), and an approximately 9 T-S wavelengths physical domain, is used. The height of the computational domain is 30 (based on the δ at the end point of suction/blowing slow, x2), the T_∞=300K. The amplitude of disturbance is set to $\omega_{2d} = 0.005$, $\omega_{3d} = 0.001$, the spanwise wave number is $\beta = 0.2$, and the angular frequency $\omega = 0.1$.

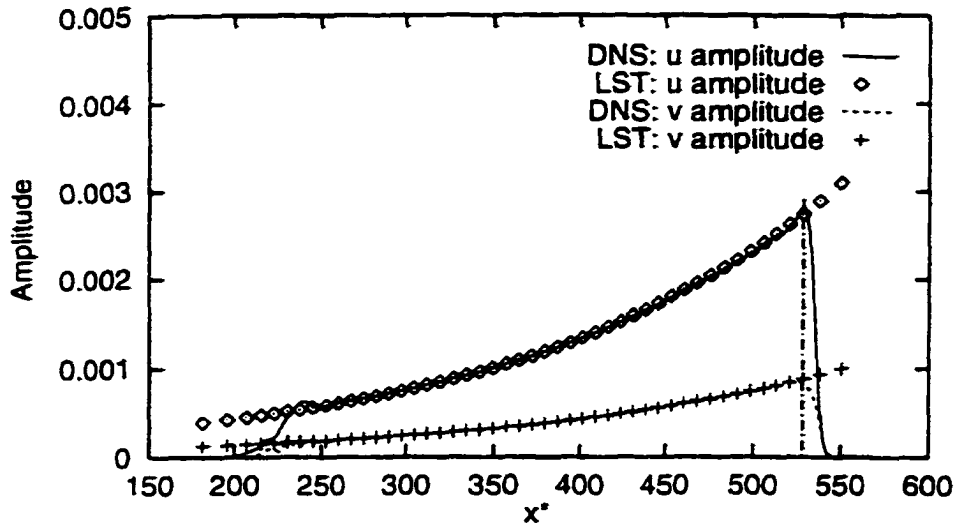


Fig. 3.11 Comparison of DNS and LST results
for the amplitude eignfunctions of subsonic flat plate boundary layer

One forcing period is divided into 1000 time steps. By using a 3*2 partition, it takes about 4.7 seconds per time-step, or 8.16 μsec per grid point per time-step.

Fig. 3.11 depicts the instantaneous contour plots of perturbation velocity

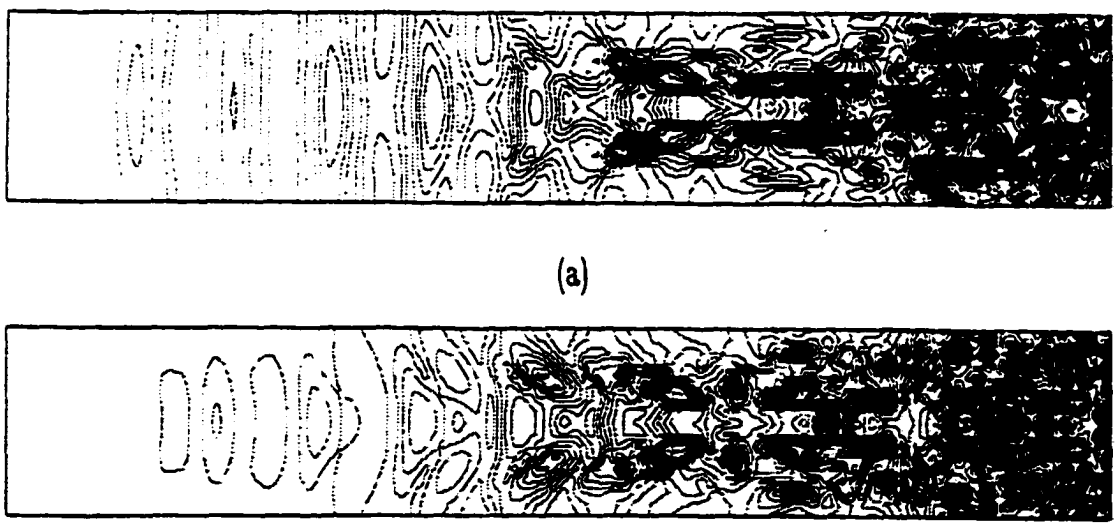


Fig.3.12 Instantaneous contour plots of perturbation (a) velocity amplitude and (b) vorticity emplitude. for the K-type breakdown of a Mach 0.5 flat boundary layer

Amplitude and pertubation vorticity magnitude on the y=0.3727(x, z) planes after 9 forcing periods. It shows that the transition process is very similiar to that of incompressible flow, except that the "let" of the so-called lambda waves are longer. Also, even after the lambda waves break into smaller scale eddies, the major splitting appears in the spanwise direction, where the length scale in the streamwise direction is relatively longer.

3.9. Conclusion

MPI is applied to perform parallel computations for DNS. By using the parallelized code, the current approach shows the ability of DNS to simulate the whole process of compressible flow transition with medium Reynolds number. The MPI code shows the flexibility of using different parallel machines, which was a problem encountered by researchers for a long time.

From the performance we obtained, we found that the computing time can be reduced substantially. The current code obtained nearly ideal linear speedup up to 64 processors. The advantage of a parallel machine in CPU and memory storage makes it very attractive for large-scale computation.

3.10 Appendix

1. Point to point communication functions

MPI_Recv(void* message, int count, MPI_Datatype datatype, int source, int tag,
MPI_Comm comm, MPI_Status* status)

MPI_Send(void* message, int count, MPI_Datatype datatype, int dest, int tag,
MPI_Comm comm)

MPI_Irecv(void* message, int count, MPI_Datatype datatype, int source, int tag,
MPI_Comm comm, MPI_Request* request)

MPI_Isend(void* message, int count, MPI_Datatype datatype, int dest, int tag,
MPI_Comm comm, MPI_Request* request)

Int MPI_Wait(MPI_Request* request, MPI_Status status)

2. Derived datatype functions

int MPI_Type_commit(MPI_Datatype* datatype)

int MPI_Type_contiguous(int count, MPI_Datatype oldtype, MPI_Datatype newtype)

int MPI_Type_stuct(int count, int blocklength[], MPI_Aint displacement[],
MPI_Datatype types[], MPI_Datatype* newtype)

Int MPI_Type_Vector(int count, int blocklength, int stride, MPI_Datatype oldtype,
MPI_Datatype* newtype)

Int MPI_Pack(void inbuf, int incount, MPI_Datatype datatype, void* pack_buf,
Int pack_buf_size, int* position, MPI_Comm comm)

Int MPI_Unpack(void* pack_buf, int pack_buf_size, int* position, void* outbuf,

Int outcount, MPI_Datatype datatype, MPI_Comm comm)

3. Collective communication functions

int MPI_Bcast(void buffer, int counter, MPI_Datatype datatype, int root,

MPI_Comm comm)

Int MPI_Gather(void* sendbuf, int sendcount, MPI_Datatype sendtype, void* recvbuf,

int recvcount, MPI_datatype recvtype, int root, MPI_Comm comm)

Int MPI_Scatter(void* sendbuf, int sendcount, MPI_Datatype sendtype, void* recvbuf,

int recvcount, MPI_datatype recvtype, int root, MPI_Comm comm)

int MPI_Reduce(void* operand, void* result, int count, MPI_Datatype datatype,

MPI_Op operator, int root, MPI_Comm comm)

4. Environmental management

int MPI_Comm_create(MPI_Comm comm, MPI_Group new_group,

MPI_Comm* new_comm)

Int MPI_Comm_rank(MPI_Comm comm, int* rank)

Int MPI_Comm_Size(MPI_Comm comm, int* size)

int MPI_Init(int* argc_ptr, char** argc_ptr[])

int MPI_Finalize(void)