MODIFIED NODAL INTEGRAL METHOD FOR NAVIER-STOKES EQUATIONS INCORPORATED WITH GENERIC QUADRILATERAL ELEMENTS, AND GPU-BASED PARALLEL COMPUTING

BY

KAI HUANG

DISSIDERTATION

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Nuclear Engineering in the Graduate College of the University of Illinois at Urbana-Champaign, 2011

Urbana, Illinois

Doctroal Committee:

Professor Rizwan-uddin, Chair
Professor Roy A. Axford
Professor Barclay G. Jones
Assistant Professor Carlos A. Pantano-Rubino
Abstract

This dissertation can be broadly divided into two connected parts: development and testing of a new numerical scheme for time-dependent, incompressible Navier-Stokes (N-S) equations in non-rectangular domains; and, implementation and solution of the modified nodal integral method (MNIM), developed earlier for rectangular domains, on graphics processing units (GPUs).

Nodal methods have become the backbone and workhorse of the core design production codes used in the nuclear industry for decades. As a variation of the coarse mesh nodal methods, the modified nodal integral method can accurately solve the time-dependent, incompressible N-S equations using less computation time, hence provides more efficient solution to the fluid flow problems than many other conventional schemes that rely on fine meshes. However, the transverse integration procedure (TIP) required in the formulation of the MNIM limits the scheme to be only applicable to rectangular elements. In order to remove this limitation and extend the MNIM to non-rectangular computational meshes/domains, a modified nodal integral method incorporated with generic quadrilateral elements is developed using a simple isoparametric geometry mapping. The mapping is used to transform: 1) the irregular four-node quadrilateral elements into square elements; 2) the original set of N-S equations into a set of transformed equations valid over the transformed computational domain. Then the new nodal scheme is formulated for the transformed equations. The numerical scheme developed is applied to several test problems of increasing complexity. Results show that the scheme works very well with quadrilateral elements of different shapes and degrees of distortion, maintaining the high accuracy.
and efficiency of the MNIM; and that the new scheme has inherent upwinding.

To further enhance the computational capabilities, one needs to exploit the latest developments in computing hardware. Realizing that graphics processing units can provide superior computational power over conventional CPUs, the cutting-edge GPU-computing and the highly efficient nodal scheme are married in a double precision GPU implementation of the MNIM for the 3D, incompressible N-S equations in the second part of this dissertation. The GPU implementation is applied to simulate the lid-driven cavity flows in a unit cube and a prism with aspect ratio of two, and is validated. A performance analysis indicates that the MNIM on GPU can be an order of magnitude faster than on a CPU.
To Father and Mother.
Acknowledgments

This dissertation work would not have been completed without the support of many people. My sincerest appreciation goes to my adviser (and mentor), Professor Rizwanuddin, for his continuous guidance and encouragement in my study. I wish to thank him especially for his tremendous patience and support during the most difficult time in my life. I would also like to thank Professor Roy Axford, Professor Barclay Jones, and Professor Carlos Pantano-Rubino for serving on my doctoral final examination committee. Special thanks are due to Professor S. Pratap Vanka for serving on my preliminary examination committee.

I am indebted to Meimei Ng (Weiwei Wu), whose encouragement and support were indispensable in the early years of my Ph.D. study. I am fortunate to have met and received help from all the friends: Nan Niu, Junxian Lin, Dr. Jianwei Hu, Rajat Saksena, Jia Xing, Yi Wang, Dong Ding, Xiang Chen, Qi Lou, Yizhou Yan, Dr. Bei Ye, Dr. Geng Fu, and Yonghui Li, during my study at the University of Illinois at Urbana-Champaign. Also, I would like to thank Wei Zhang, for her incomparable encouragement that not only made this work possible, but also made me mature.

Finally, I wish to express my appreciation and love to my father Xiaoping Huang and my mother Qi Zhou, for their unconditional support and love.
# Table of Contents

List of Tables ................................................................. viii

List of Figures ................................................................. ix

List of Abbreviations ......................................................... xii

List of Symbols ............................................................... xiii

1 Introduction ................................................................. 1
  1.1 Motivation ............................................................... 1
  1.2 Structure of the Dissertation ......................................... 2

2 A Nodal Scheme for Quadrilateral Elements ......................... 3
  2.1 Background ............................................................. 3
  2.2 Limitation of the Nodal Integral Method for Domains with Complex
      Geometries .............................................................. 5
  2.3 A Nodal Scheme for Quadrilateral Elements ....................... 8

3 Modified Nodal Integral Method Incorporated with Quadrilateral
   Elements for Navier-Stokes Equations—Formalism .................... 12
  3.1 Isoparametric Mapping and Coordinate Transform .................. 12
  3.2 Primitive Form of the Navier-Stokes Equations .................... 15
  3.3 Transformed Navier-Stokes Equations ................................ 18
  3.4 Transverse Integration Procedure .................................... 22
  3.5 Solutions to the Transverse-Integrated Equations ................... 28
  3.6 Continuity of the Unknowns and their Directional Derivatives .... 31
  3.7 Constraint Conditions for the Elimination of Pseudo-Source Terms 36
  3.8 Final Set of Discrete Equations ...................................... 40
  3.9 Summary .................................................................. 45

4 Modified Nodal Integral Method Incorporated with Quadrilateral
   Elements for Navier-Stokes Equations—Numerical Results .......... 46
  4.1 Introduction ............................................................. 46
  4.2 Poiseuille Flow Between Parallel Plates ............................ 46
  4.3 Modified Lid-Driven Cavity Problem .................................. 55
  4.4 Tangential Annular Flow .............................................. 67
  4.5 Time-Dependent Periodic Flow ....................................... 73
  4.6 Summary .................................................................. 82
List of Tables

3.1 Correspondence of $F$ coefficients ........................................... 43

4.1 Numerical values of corresponding $F$ coefficients for element $(5, 5)$ in
the square domain of the modified lid-driven cavity problem. .............. 59

4.2 RMS errors for the modified lid-driven cavity problem. ................. 61

4.3 RMS errors and CPU time for the tangential annular flow problem. ... 73

4.4 $l$ coefficients in equation (4.21) at $t = 2\pi$. ............................ 82

4.5 $l$ coefficients in equation (4.21) at $t = 3\pi$. ............................ 82

5.1 Characteristics of the Tesla C1060 and Quadro FX5600 GPUs. ....... 101

5.2 Single precision performance of GPU and CPU implementations for
the Burgers solver. ................................................................. 101

5.3 Double precision performance of GPU and CPU implementations for
the Burgers solver. ................................................................. 101

6.1 Performance comparison of GPU and CPU implementations using the
MNIM to simulate the lid-driven cavity flow in a cube. ....................... 131
# List of Figures

2.1 Locations of the discrete variables in element \((i,j)\) ............................... 6  
2.2 Discretization in a quarter annulus ......................................................... 9  
2.3 Global mapping ......................................................................................... 10  
2.4 2D natural coordinate system ................................................................. 11  
3.1 Mapping of the quadrilateral element to a \((2 \times 2)\) square element .... 13  
3.2 Continuity of the unknown \(\pi^d\) between adjacent elements .................. 29  
3.3 Compatibility criterion between adjacent elements ............................... 32  
3.4 Computational mesh for the transverse-integrated unknowns ............... 42  
4.1 Schematic diagram for the Poiseuille flow between parallel plates ....... 48  
4.2 Computational domains for the Poiseuille flow problem ....................... 49  
4.3 Comparison of numerical and exact \(u\) velocity along \(E'F'\) in the rotated square domain \(A'B'C'D'\) for the Poiseuille flow problem .......... 50  
4.4 RMS error of the \(u\) velocity in the square domain \(ABCD\) and \(A'B'C'D'\) for the Poiseuille flow problem ......................................................... 51  
4.5 Comparison of numerical and exact \(u\) velocity along \(MN\) in the trapezoidal domain \(IJKL\) for the Poiseuille flow problem .......... 53  
4.6 RMS error of the \(u\) velocity in the trapezoidal domain \(IJKL\) for the Poiseuille flow problem ......................................................... 54  
4.7 Schematic diagram and computational mesh for the modified lid-driven cavity problem ......................................................... 56  
4.8 Numerical results for the velocity vectors of the modified lid-driven cavity problem on a \(20 \times 20\) uniform square mesh for \(Re = 1\) ............ 62  
4.9 Comparison of exact and numerical solutions for \(\pi^{eq}\) and \(\pi^d\) over the parallelogram \(ABCD\) for the modified lid-driven cavity problem \((8 \times 8\) mesh, \(\theta = \frac{\pi}{4}\)) ......................................................... 63  
4.10 Comparison of exact and numerical solutions along line \(EF\) for the modified lid-driven cavity problem ......................................................... 64  
4.11 Comparison of exact and numerical solutions of the edge-averaged \(u\) velocities along lines \(EF\) and \(GH\) for the modified lid-driven cavity problem ......................................................... 65  
4.12 Numerical results for the velocity vectors of the modified lid-driven cavity problem over the \(ABCD\) domain for \(8 \times 8\) mesh cases ............ 66  
4.13 Schematic diagram and computational mesh for the tangential annular flow problem ......................................................... 67
4.14 Flow map of the tangential annular flow problem for 15 × 10 mesh. 70
4.15 Flow map of the tangential annular flow problem for 30 × 20 mesh. 71
4.16 Percentage error of element-averaged velocity for elements along the center line in the tangential annular flow problem. 72
4.17 Velocity and pressure fields for the time-dependent periodic flow problem at $t = 0$. 75
4.18 Schematic diagram of the computational mesh for the time-dependent periodic flow problem. 76
4.19 Schematic velocity vector plot of the time-dependent periodic flow problem at $t = 2k\pi$ ($k = 0, 1, 2 \cdots$). 76
4.20 Flow map of the time-dependent periodic flow problem for 160 × 15 mesh at $t = \frac{3}{4}\pi$. 77
4.21 Flow map of the time-dependent periodic flow problem for 180 × 20 mesh at $t = \frac{3}{4}\pi$. 78
4.22 Comparison of numerical and exact velocity results of the time-dependent periodic flow problem at $t = 2\pi$. 79

5.1 Current mainstream multi-core CPU and many-core GPU. 85
5.2 CPU and GPU performance in floating-point operations. 86
5.3 Comparison of CPU and GPU architectures. 89
5.4 NVIDIA Tesla C1060 GPU architecture. 90
5.5 Modeling of thread and memory in CUDA. 93
5.6 Schematic diagram for memory transfer between host and device. 95
5.7 Flow chart of the CUDA Burgers solver. 102
5.8 Snapshots showing $u$ and $v$ velocity results of the Burgers equation. 103
5.9 Difference of steady-state $u$ velocity results of the Burgers equation calculated using double and single precision on a 32 × 32 mesh. 104
5.10 Difference of steady-state $v$ velocity results of the Burgers equation calculated using double and single precision on a 32 × 32 mesh. 105

6.1 Relationship between computational element, GPU thread/block. 110
6.2 Red-Black ordering of Cartesian grid points. 112
6.3 Configuration and coordinate system of the lid-driven cavity flow in a cube. 117
6.4 $u$ velocity profile for $Re = 100$ along the vertical centerline of the cubic cavity. 119
6.5 $v$ velocity profile for $Re = 100$ along the horizontal centerline of the cubic cavity. 119
6.6 $u$ velocity profile for $Re = 400$ along the vertical centerline of the cubic cavity. 120
6.7 $v$ velocity profile for $Re = 400$ along the horizontal centerline of the cubic cavity. 120
6.8 $u$ velocity profile for $Re = 1000$ along the vertical centerline of the cubic cavity. 121
6.9  $v$ velocity profile for $Re = 1000$ along the horizontal centerline of the cubic cavity. .................................................. 121
6.10  Velocity field for $Re = 100$ in the $x = 0.5$ plane of the cubic cavity. .................................................. 122
6.11  Velocity field for $Re = 400$ in the $x = 0.5$ plane of the cubic cavity. .................................................. 123
6.12  Velocity field for $Re = 1000$ in the $x = 0.5$ plane of the cubic cavity. .................................................. 124
6.13  Configuration and coordinate system of the lid-driven cavity flow in a prism. .................................................. 125
6.14  $u$ velocity profile for $Re = 100$ along the vertical centerline of the prismatic cavity with aspect ratio of 2. .............................. 127
6.15  Velocity field for $Re = 100$ in the $z = 0.5$ plane of the prismatic cavity with aspect ratio of 2. .............................. 127
6.16  $u$ velocity profile for $Re = 400$ along the vertical centerline of the prismatic cavity with aspect ratio of 2. .............................. 128
6.17  Velocity field for $Re = 400$ in the $z = 0.5$ plane of the prismatic cavity with aspect ratio of 2. .............................. 128
6.18  $u$ velocity profile for $Re = 1000$ along the vertical centerline of the prismatic cavity with aspect ratio of 2. .............................. 129
6.19  Velocity field for $Re = 1000$ in the $z = 0.5$ plane of the prismatic cavity with aspect ratio of 2. .............................. 129
6.20  Percentages of GPU time for different kernels and operations. .................................................. 130
# List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>Arithmetic Logic Unit</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>CUDA</td>
<td>Compute Unified Device Architecture</td>
</tr>
<tr>
<td>FPU</td>
<td>Floating-Point Unit</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphics Processing Unit</td>
</tr>
<tr>
<td>LHS/RHS</td>
<td>Left Hand Side/Right Hand Side</td>
</tr>
<tr>
<td>MIMD</td>
<td>Multiple Instruction, Multiple Data</td>
</tr>
<tr>
<td>MNIM</td>
<td>Modified Nodal Integral Method</td>
</tr>
<tr>
<td>NGFM</td>
<td>Nodal Green’s Function Method</td>
</tr>
<tr>
<td>NGTM</td>
<td>Nodal Green’s Tensor Method</td>
</tr>
<tr>
<td>NIM</td>
<td>Nodal Integral Method</td>
</tr>
<tr>
<td>N-S</td>
<td>Navier-Stokes</td>
</tr>
<tr>
<td>PCBM</td>
<td>Partial Current Balance Method</td>
</tr>
<tr>
<td>PDE</td>
<td>Partial Differential Equation</td>
</tr>
<tr>
<td>RMS</td>
<td>Root Mean Square</td>
</tr>
<tr>
<td>SFU</td>
<td>Special Function Unit</td>
</tr>
<tr>
<td>SIMD</td>
<td>Single Instruction, Multiple Data</td>
</tr>
<tr>
<td>SM</td>
<td>Streaming Multiprocessor</td>
</tr>
<tr>
<td>SP</td>
<td>Streaming Processor</td>
</tr>
<tr>
<td>TIP</td>
<td>Transverse Integration Procedure</td>
</tr>
</tbody>
</table>
List of Symbols

- \( g \): Body force term
- \( H \): Height
- \( m \): Normal vector
- \( p \): Pressure
- \( R \): Radius
- \( \text{Re} \): Reynolds number
- \( t \): Time
- \( u, v, w \): Cartesian velocity components
- \( \mathbf{v} \): 3D velocity vector
- \( \mathbf{w} \): 2D velocity vector
- \( \theta \): Rotation angle
- \( \mu \): Dynamic viscosity
- \( \nu \): Kinematic viscosity
- \( \rho \): Fluid density
- \( \omega \): Angular velocity
- \( \Omega \): Domain
1 Introduction

1.1 Motivation

Initially studied by researchers centuries ago, the fluid flow problem remains one of the most challenging problems in science and engineering today due to its extraordinary complexity. Realizing the impossibility of finding analytic solutions to describe flows beyond simple cases, scientists and engineers have turned to numerical methods. By developing increasingly better (efficient) numerical schemes, researchers have been able to solve increasingly more complex fluid flow problems. On the other hand, efforts have also continued to take advantage of continued development in computing power.

The Navier-Stokes (N-S) equations are used to study viscous flows. This dissertation presents a research work aimed at finding an efficient numerical solution to the time-dependent, incompressible N-S equations. The work comprises two parts representing the two major approaches to achieve this aim: advanced numerical schemes and advanced computing resources. In the first part, a nodal integral method (a class of coarse-mesh methods) for the time-dependent, incompressible N-S equations is developed for generic quadrilateral elements, eliminating the existing restrictions on a class of nodal scheme that make it applicable only to rectangular meshes/domains. In the second part, to take advantage of the computational power of the cutting-edge GPU-computing technology, a new implementation of the Modified Nodal Integral Method (MNIM) for the 3D incompressible N-S equations is realized on Graphics Processing Units (GPUs).
1.2 Structure of the Dissertation

The remainder of this dissertation is organized in the following manner. Chapter 2 first provides a review of the coarse mesh nodal methods, next discusses the existing inability of the nodal integral method to treat non-rectangular grid elements, and consequently proposes a new nodal approach using algebraic mapping technique to remove the limitation. Based on the proposed approach, the Modified Nodal Integral Method incorporated with generic quadrilateral elements is developed in chapter 3 for the time-dependent incompressible Navier-Stokes equations. The developed method is applied to several benchmark fluid flow problems for testing purposes in chapter 4. Chapter 5 discusses the GPU-based parallel computing technique, including the re-view of general-purpose computing on GPUs, the modern GPU architecture, and the Compute Unified Device Architecture (CUDA) which is the most popular programming model of the GPU-computing. A Burgers equation solver implemented using CUDA is also presented as a case study in chapter 5, with emphasis on computation performance and accuracy. Chapter 6 presents the new implementation of the MNIM on GPUs for 3D incompressible N-S equations. The GPU implementation is tested using two cases. Chapter 7 summarises and concludes the dissertation.
A Nodal Scheme for Quadrilateral Elements

2.1 Background

Many problems in engineering and physical sciences are usually modelled and described in the form of partial differential equations (PDE). There are generally three conventional categories of numerical schemes that are most commonly used in seeking solutions of partial differential equations: finite difference method, finite volume method, and finite element method. The idea of finite difference method is to discretize the computational domain into finite number of grid points, and approximate the derivatives in terms of the value of the dependent variables of neighboring points by truncating Taylor expansion of the unknown at a desired order of accuracy [1]. Unlike the finite difference method, finite volume method requires the problem domain to be divided into control volumes. Conservation laws are applied on the control volumes to obtain a set of corresponding algebraic equations [2]. The finite element method, first developed for the structural mechanics problems, decomposes the computational domain into so-called elements. This method reduces the original partial differential equation(s) to a set of algebraic equations mainly by minimizing a functional defined over each element [1].

The three methods mentioned above have been widely used to solve partial differential equations in science and engineering applications, including the Navier-Stokes equations. However, large-scale, multi-physics simulation of engineering devices such as nuclear reactors, if treated with these more conventional numerical methods, requires tremendous machine hours due to the large number of unknowns from fine
meshes necessary for a desirable level of accuracy. Numerical methods that can sustain accuracy over coarse mesh size in order to significantly reduce machine hours are therefore desirable.

Coarse mesh methods, originally developed in nuclear engineering field for neutronics problems, have been evolving over the last few decades. Proposed by Burns and Dorning, Partial Current Balance Method (PCBM) was among the first coarse mesh methods to treat multi-group neutron diffusion equations [3,4]. PCBM applies a *locally defined* two-dimensional Green’s function to convert the neutron diffusion equation into a local integral equation, involving only the nearest neighbor-coupling. Later, Nodal Green’s Function Method (NGFM) [5,6] was developed after the emergence of the *transverse integration* technique, which results in a scheme with less complexity and smaller number of unknowns per node, compared with PCBM. Around the beginning of 1980s, two variations of the nodal methods were proposed to deal with neutronics and fluid flow problems: Nodal Green’s Tensor Method (NGTM) [7,8] and Nodal Integral Method (NIM) [9–11]. These two methods are proved to be mathematically equivalent to each other. NGTM inherits the idea to apply the locally defined Green’s functions from PCBM, while NIM extends the idea of the *transverse integration* technique. Since NIM does not rely on the local Green’s functions, it is simpler to implement than NGTM. NIM has been developed over the years to solve the Navier-Stokes equations with different levels of success [11–14]. NIM was also developed to solve the convection-diffusion equation [15]. The NIM, compared with other schemes such as LECUSSO [16] for the convection-diffusion equation, is found to be more efficient [15].

More recently, Modified Nodal Integral Method (MNIM)—a variation of NIM—was developed for the 2D and 3D, time-dependent, Navier-Stokes equations [17]. The MNIM is distinguished from the original NIM in that the former introduces an idea to partially evaluate the convective terms in the Navier-Stokes equations from the
velocity information obtained at the previous time step, while the latter dumps the convective terms wholly into the so-called “pseudo source” term. The idea introduced is similar to the well-known concept of “delayed coefficients” [18]. With the merit in better capturing and treating the nonlinear convective terms, the MNIM shows high accuracy and efficiency even on relatively coarse meshes, which in turn makes the scheme a strong candidate for modeling complex flows such as turbulence [19]. However, the local transverse integration procedure required to obtain the set of ordinary differential equations for each element limits the MNIM to fluid flow fields that, in 2D, can be decomposed into rectangular elements based on a Cartesian grid. As a result, the efficiency achieved by using coarse meshes is adversely impacted due to the need to use smaller rectangular elements for problems on domains with complex geometries. This restriction of the NIM/MNIM to model flows in irregular-shaped domains will become more clear in the next section.

2.2 Limitation of the Nodal Integral Method for Domains with Complex Geometries

The computational domain in the MNIM is discretized in brick-like elements/nodes with element-centered local coordinates. In 2D cases, for example, the entire domain \((X,Y,T)\) is divided into rectangular space-time elements \((i,j,n)\) of size \((2a_i \times 2b_j \times 2\tau_n)\) with local coordinates \((x,y,t; -a_i \leq x \leq a_i, -b_j \leq y \leq b_j, -\tau_n \leq t \leq \tau_n)\). While most other numerical methods seek directly to solve for the point values or the volume averaged values of the dependent variables in the original governing PDE’s, the NIM nevertheless takes the edge (in 2D) or surface (in 3D) transverse-integrated values as dependent variables to be solved. These variables are obtained after the Transverse Integration Procedure (TIP), a NIM-featured procedure in which the PDE is integrated with respect to all independent variables except one, leading to an
ODE. The TIP is repeated for the whole set of governing equations in all space-time
directions, thus reducing each PDE to a set of ODE’s with the transverse-averaged
variables as unknowns. For instance, assume $\phi_{i,j,n}(x, y, t)$ is a function varying over
the element indexed by $(i, j, n)$ in the domain. One of the function’s corresponding
transverse-integrated variables, denoted by $\overline{\phi}_{i,j,n}^{yt}(x)$, indicates that $\phi_{i,j,n}(x, y, t)$ has
been locally integrated within the element in the $y$ and $t$ directions. $\overline{\phi}_{i,j,n}^{yt}(x)$ is
formally defined as

$$
\overline{\phi}_{i,j,n}^{yt}(x) = \frac{1}{4b_j \tau_n} \int_{-b_j}^{b_j} \phi_{i,j,n}(x, y, t) \, dy \, dt.
$$

Following the conventional notation in the NIM, the over-bar in equation (2.1) il-
lustrates that the barred variable has a sense of averaging, and the letters trailing
the bar point out the directions over which the TIP has been carried out. The other
transverse-integrated variables are similarly defined and symbolized. Figure 2.1 shows

![Figure 2.1: Locations of the discrete variables in element $(i, j)$.
](image)

the locations of the discrete counterparts to the above-defined variables within ele-


ment \((i, j)\) of the computational grid. The triangle, square and circle points in the
figure represent the discrete variables \(\overline{\phi}^{yt}, \overline{\phi}^{xt}\) and \(\overline{\phi}^{xy}\), respectively. Note that unlike
the more traditional numerical methods that use discrete variables at the grid points,
the *discrete* unknowns in the NIM are the transverse-averaged variables (or to say,
functions) evaluated on the surfaces of the space-time elements. For example, the
discrete variable \(\overline{\phi}^{yt}_{i,j,n}\) is designated as

\[
\overline{\phi}^{yt}_{i,j,n} \equiv \overline{\phi}^{yt}_{i,j,n}(x = a_i),
\]

which is obtained by evaluating the transverse-averaged variable \(\overline{\phi}^{yt}_{i,j,n}(x)\) at \(x = a_i\).
The NIM essentially develops a set of algebraic equations for these discrete variables
based on the continuity property of the unknowns and their flux across the element
interfaces. Reference [11] can be referred to for a comprehensive description of the
derivation of the NIM.

An important fact about the NIM becomes now obvious, that is, the TIP can only
be applied in the directions aligned along the axes of the local orthogonal coordinates.
This fact further implies that the mesh of the NIM has to be composed of rectangular
elements (the elements may be uniform or non-uniform).

In theory, any domain can be fitted by a cluster of rectangular elements, and the
requirement of a certain accuracy can easily be achieved by controlling the size of the
elements. In many applications, the computational domain has a simple rectangular
shape that can be perfectly filled by the union of rectangular elements. But such
elements become less effective for problems defined on domains with irregular shape,
and thus alternative approaches are desirable. Some of the more general approaches
to adapt Cartesian grid based numerical methods (other than the NIM) to complex
geometries include Cartesian cut-element method [20], Immersed Boundary Method
(IBM) [21–23], and the ghost fluid method [23–26]. As a coarse mesh method, the
NIM however, is even more adversely impacted than other schemes when forced to use rather fine mesh in the area near the boundary. To overcome this limitation, a hybrid approach combining the nodal integral method and other numerical schemes was reported earlier for both fluid flow problems [27] and neutron diffusion equation [28]. The hybrid approach tackles the interior of the computational domain in the context of the conventional NIM, yet employs triangular elements in the regions along the curved boundaries, and then applies the finite element method or finite analytic method on those triangular elements. The hybrid approach is successful in that the high accuracy and efficiency of the nodal method are maintained. It nevertheless is a scheme based on the conventional NIM, not taking full advantage of the progress made in the more recent study of MNIM. Also, the hybrid approach relies on two types of elements respectively in the interior area and around the area near the curved boundary. Moreover, the coupling of the NIM and the finite element or finite analytic method in the hybrid approach increases complexity in computation—particularly, two different sets of discrete equations have to be derived for the two types of elements, which further adds heterogeneity to the method. This dissertation, instead, proposes a nodal scheme based on the more advanced MNIM for generic quadrilateral elements. To remove the aforementioned limitation of MNIM in an integral and compact sense, the new scheme will develop a single set of discrete equations to govern every element, without having to resort to other types of elements or numerical methods. The motivation and idea of the proposed scheme are discussed in the next section.

2.3 A Nodal Scheme for Quadrilateral Elements

To numerically solve a problem over a domain with complex geometries, meshing practice has demonstrated that less restrictions on the shape of elements used often bring about higher efficiency and more flexibility. Figure 2.2 serves as a good example
illustrating the advantage in applying a non-rectangular grid over a rectangular grid in an annulus area. The number of elements is shown to significantly decrease if

(a) Discretization with rectangular grid  (b) Discretization with non-rectangular grid

Figure 2.2: Discretization in a quarter annulus.

the annulus is discretized by generic quadrilateral elements instead of brick-like ones required by the conventional NIM, implying that a non-rectangular grid should fit the NIM better. The NIM calculations however can be done on rectangular elements only, due to the TIP-induced limitation that has been discussed above. Hence a mapping from the non-rectangular grid to the rectangular grid is needed.

In fact, non-rectangular grid constituted by arbitrary quadrilateral elements has long and widely been applied to the more traditional numerical schemes (such as the finite difference method). Specifying a generalized coordinate system, the non-rectangular grid in the physical space can be globally mapped to a rectangular counterpart in the computational space [29] (see figure 2.3). Calculations are next done on the mapped uniform grid. In the meantime the governing equations also need be transformed according to the mapping, leading to the transformed equations. Fair enough, the cost of attaining ease regarding geometry lies in the obligation to deal with an often very complicated set of transformed equations. And this will become
Figure 2.3: Global mapping from the physical domain to the computational domain. Notice the point \((x_i, y_j)\) in \((x, y)\) space is mapped to the point \((\xi_i, \eta_j)\) in \((\xi, \eta)\) space.
more evident in the next chapter.

![2D natural coordinate system](image)

Figure 2.4: 2D natural coordinate system.

The mapping technique described above can in a straightforward manner be incorporated into the NIM/MNIM as an approach to circumvent its limitation, except for one distinction. Due to the local analytical solution in the development of the NIM/MNIM, the new approach must apply, instead of a global mapping as discussed in the previous paragraph, a local mapping that transforms each single quadrilateral element in the Cartesian coordinate system to a square element in a so-called natural coordinate system with coordinates ranging between $-1$ and $1$ [30] (see figure 2.4). Playing a key role in dealing with elements with curved boundaries in the Finite Element Method [31], the approach has been successfully implemented to solve the Poisson equation and the convection-diffusion equation in irregular-shaped domains using the NIM [32]. The results showed that accuracy of the nodal integral scheme can be maintained for non-rectangular elements in the context of the Poisson and convection-diffusion equations. In the present dissertation work, this approach is further applied to implement an MNIM-based method to solve the much more challenging Navier-Stokes equations. The development of the new method is elaborated in the next chapter.
3 Modified Nodal Integral Method Incorporated with Quadrilateral Elements for Navier-Stokes Equations—Formalism

3.1 Isoparametric Mapping and Coordinate Transform

Consider a generic quadrilateral element specified in spatial domain $\hat{\Omega}$, and a square element with origin at the centroid in domain $\Omega$, as shown in figure 3.1. Notice that $\hat{\Omega}$ is characterized by global coordinates $x$ and $y$, while $\Omega$ by local natural coordinates $\xi$ and $\eta$ with $-1 \leq \xi, \eta \leq 1$. Assume an isoparametric mapping from domain $\hat{\Omega}$ to $\Omega$, such that $x(\xi, \eta) = [L(\xi, \eta)]x$ and $y(\xi, \eta) = [L(\xi, \eta)]y$ where $x$ and $y$ denote column vectors containing nodal values of the $x$ and $y$ coordinates, and that the matrix $[L(\xi, \eta)] = [l_i(\xi, \eta); i = 1, 2, 3, 4]$ has bilinear Lagrange interpolation functions (see appendix A) as its four components. The objective is to transform the global coordinate $(x, y)$ to the local coordinate $(\xi, \eta)$. The explicit form of this algebraic mapping is given as

$$
\begin{align*}
x &= \frac{(1+\xi)(1+\eta)}{4}x_1 + \frac{(1-\xi)(1+\eta)}{4}x_2 + \frac{(1-\xi)(1-\eta)}{4}x_3 + \frac{(1+\xi)(1-\eta)}{4}x_4, \\
y &= \frac{(1+\xi)(1+\eta)}{4}y_1 + \frac{(1-\xi)(1+\eta)}{4}y_2 + \frac{(1-\xi)(1-\eta)}{4}y_3 + \frac{(1+\xi)(1-\eta)}{4}y_4,
\end{align*}
$$

(3.1)

where $(x_i, y_i)$ indicates the $i$th node’s coordinates in global domain $\hat{\Omega}$.

Let $C^0(\hat{\Omega})$ and $C^0(\Omega)$ denote the spaces of continuous functions defined over $\hat{\Omega}$ and $\Omega$, respectively. Then for any continuous function $\hat{F}(x, y) \in C^0(\hat{\Omega})$, by substituting
Figure 3.1: Mapping of the quadrilateral element to a $(2 \times 2)$ square element.
for \(x\) and \(y\) from equation (3.1), there exists a corresponding continuous function \(F(\xi, \eta) \in C^0(\Omega)\) such that

\[
F(\xi, \eta) = \hat{F}(x(\xi, \eta), y(\xi, \eta)).
\]  

(3.2)

Further, note that the mapping indicated by equation (3.1) is one-to-one and onto, which guarantees the existence of an inverse function \(\hat{F}(x, y) \in C^0(\hat{\Omega})\) such that

\[
\hat{F}(x, y) = F(x(\xi, \eta), \eta(x, y)).
\]  

(3.3)

Moreover, if the function \(\hat{F}(x, y)\) is differentiable, then its first and second partial derivatives\(^\dagger\) with respect to \(x\) and \(y\) are easily obtained from the chain rule as

\[
\begin{align*}
\hat{F}_x &= F_{,\xi} \xi_x + F_{,\eta} \eta_x \\
\hat{F}_y &= F_{,\xi} \xi_y + F_{,\eta} \eta_y 
\end{align*}
\]  

(3.4)

and

\[
\begin{align*}
\hat{F}_{xx} &= F_{,\xi} \xi_{xx} + F_{,\eta} \eta_{xx} + \xi_x (F_{,\xi} \xi_x + F_{,\eta} \eta_x) + \eta_x (F_{,\eta} \xi_x + F_{,\eta} \eta_x) \\
\hat{F}_{xy} &= F_{,\xi} \xi_{xy} + F_{,\eta} \eta_{xy} + \xi_x (F_{,\xi} \xi_y + F_{,\eta} \eta_y) + \eta_x (F_{,\eta} \xi_y + F_{,\eta} \eta_y) \\
\hat{F}_{yy} &= F_{,\xi} \xi_{yy} + F_{,\eta} \eta_{yy} + \xi_y (F_{,\xi} \xi_y + F_{,\eta} \eta_y) + \eta_y (F_{,\eta} \xi_y + F_{,\eta} \eta_y) 
\end{align*}
\]  

(3.5)

Thus, a partial differential equation whose spatial variables are defined over a generic quadrilateral element in global domain \(\hat{\Omega}\) can be transformed into a target equation having spatial variables defined over a square element in local domain \(\Omega\) by transforming the original equation term by term using the above equations (3.3), (3.4) and (3.5), as long as each term represents a continuous function. It can be foreseen that the transformed equations will be more complicated than their original counterparts,

\(^\dagger\)For convenience, the dissertation uses, except for the regular fractional notation, comma notation for partial derivatives. Explanation will be given when both kinds of notations appear in the same equation.
implied especially by the complex expression for second derivatives on the right hand side of equation (3.5). Geometric coefficients such as $\xi_{,x}$ will need to be calculated to complete the transformation$^1$.

### 3.2 Primitive Form of the Navier-Stokes Equations

The two-dimensional, time-dependent, incompressible, isothermal Navier-Stokes equations in primitive variables are

\[
\frac{\partial \hat{u}}{\partial t} + \hat{u} \frac{\partial \hat{u}}{\partial x} + \hat{v} \frac{\partial \hat{u}}{\partial y} - \nu \left( \frac{\partial^2 \hat{u}}{\partial x^2} + \frac{\partial^2 \hat{u}}{\partial y^2} \right) + \frac{1}{\rho} \frac{\partial \hat{p}}{\partial x} + \hat{b}_x = 0,
\]

\[
\frac{\partial \hat{v}}{\partial t} + \hat{u} \frac{\partial \hat{v}}{\partial x} + \hat{v} \frac{\partial \hat{v}}{\partial y} - \nu \left( \frac{\partial^2 \hat{v}}{\partial x^2} + \frac{\partial^2 \hat{v}}{\partial y^2} \right) + \frac{1}{\rho} \frac{\partial \hat{p}}{\partial y} + \hat{b}_y = 0,
\]

in which $x$, $y$ and $t$ denote independent spatial variables and time variable, respectively, while $\hat{b}_x$ and $\hat{b}_y$ represent body force terms such as gravity. The three dependent variables $\hat{u}$, $\hat{v}$ and $\hat{p}$ are unknown functions to be solved, with hat indicating that they are defined over the global domain $\hat{\Omega}$. Now that there are three unknowns and the same number of equations, it is desirable to have each equation solved for just one unknown. A natural and convenient idea is to assign the velocity components $u$ and $v$ to be solved using their corresponding momentum equations. The continuity equation (equation (3.6)) is, as a consequence, left for the pressure variable $p$. Note that there is no explicit pressure term in (3.6). The conventional way to resolve this

$^1$See appendix B.
difficulty is to derive a Poisson equation for pressure by mathematically manipulating the governing equations [33]. The derivation of the pressure Poisson equation is given below.

The equations

\[
\frac{\partial}{\partial t} \left( \frac{\partial \hat{u}}{\partial x} \right) + \frac{\partial}{\partial x} \left( \hat{u} \frac{\partial \hat{u}}{\partial x} + \hat{v} \frac{\partial \hat{u}}{\partial y} - \nu \frac{\partial^2 \hat{u}}{\partial x^2} - \nu \frac{\partial^2 \hat{u}}{\partial y^2} + \hat{b}_x \right) + \frac{1}{\rho} \frac{\partial^2 \hat{p}}{\partial x^2} = 0 \tag{3.9}
\]

and

\[
\frac{\partial}{\partial t} \left( \frac{\partial \hat{v}}{\partial y} \right) + \frac{\partial}{\partial y} \left( \hat{u} \frac{\partial \hat{v}}{\partial x} + \hat{v} \frac{\partial \hat{v}}{\partial y} - \nu \frac{\partial^2 \hat{v}}{\partial x^2} - \nu \frac{\partial^2 \hat{v}}{\partial y^2} + \hat{b}_y \right) + \frac{1}{\rho} \frac{\partial^2 \hat{p}}{\partial y^2} = 0 \tag{3.10}
\]

are obtained by differentiating (3.7) and (3.8) with respect to \(x\) and \(y\), respectively. The addition of equations (3.9) and (3.10) gives

\[
\frac{\partial}{\partial t} \left( \frac{\partial \hat{u}}{\partial x} + \frac{\partial \hat{v}}{\partial y} \right) - \nu \frac{\partial^2 \hat{u}}{\partial x^2} \left( \frac{\partial \hat{u}}{\partial x} + \frac{\partial \hat{v}}{\partial y} \right) - \nu \frac{\partial^2 \hat{v}}{\partial y^2} \left( \frac{\partial \hat{u}}{\partial x} + \frac{\partial \hat{v}}{\partial y} \right) + \frac{1}{\rho} \left( \frac{\partial^2 \hat{p}}{\partial x^2} + \frac{\partial^2 \hat{p}}{\partial y^2} \right) + \frac{\partial}{\partial x} \left( \hat{u} \frac{\partial \hat{u}}{\partial x} + \hat{v} \frac{\partial \hat{u}}{\partial y} + \hat{b}_x \right) + \frac{\partial}{\partial y} \left( \hat{u} \frac{\partial \hat{v}}{\partial x} + \hat{v} \frac{\partial \hat{v}}{\partial y} + \hat{b}_y \right) = 0. \tag{3.11}
\]

Recall the definition of divergence of the velocity \(\hat{u} \equiv (\hat{u}, \hat{v})^T\):

\[
\nabla \cdot \hat{u} = \frac{\partial \hat{u}}{\partial x} + \frac{\partial \hat{v}}{\partial y}. \tag{3.12}
\]

So the terms \(\frac{\partial \hat{u}}{\partial x} + \frac{\partial \hat{v}}{\partial y}\) in (3.11) can be grouped to be a dilatation term representing the divergence of the velocity field. Re-organising (3.11) in the form of a Poisson equation for \(\hat{p}\) yields

\[
\frac{\partial^2 \hat{p}}{\partial x^2} + \frac{\partial^2 \hat{p}}{\partial y^2} = -\rho \left( \frac{\partial \hat{u}}{\partial x} \right)^2 - 2\rho \frac{\partial \hat{u}}{\partial y} \frac{\partial \hat{v}}{\partial x} - \rho \left( \frac{\partial \hat{v}}{\partial y} \right)^2 - \rho \frac{\partial \hat{b}_x}{\partial x} - \rho \frac{\partial \hat{b}_y}{\partial y} - \rho \left[ \frac{\partial \hat{D}}{\partial t} + \hat{u} \frac{\partial \hat{D}}{\partial x} + \hat{v} \frac{\partial \hat{D}}{\partial y} - \nu \left( \frac{\partial^2 \hat{D}}{\partial x^2} + \frac{\partial^2 \hat{D}}{\partial y^2} \right) \right]. \tag{3.13}
\]
where the dilatation term $\hat{D}$ is given by

$$\hat{D} \equiv \frac{\partial \hat{u}}{\partial x} + \frac{\partial \hat{v}}{\partial y}.$$  \hfill (3.14)

Equation (3.13) is derived from (thus not independent of) the two momentum equations, hence the information provided by the continuity equation must be coupled into the pressure Poisson equation so as to make it independent. Indicated by (3.6), the divergence of the velocity field is zero for incompressible flows. Therefore all the terms containing $\hat{D}$ vanish in equation (3.13). It has been reported [34] that numerical instability may arise in the developed scheme if the dilatation term is set identically to zero. However, such numerical instabilities are not observed in the present work. Setting $\hat{D}$ to zero in (3.13) yields

$$\frac{\partial^2 \hat{p}}{\partial x^2} + \frac{\partial^2 \hat{p}}{\partial y^2} = -\rho \left( \frac{\partial \hat{v}}{\partial x} \right)^2 - 2\rho \frac{\partial \hat{u}}{\partial y} \frac{\partial \hat{v}}{\partial x} - \rho \left( \frac{\partial \hat{v}}{\partial y} \right)^2 - \rho \frac{\partial \hat{b}_x}{\partial x} - \rho \frac{\partial \hat{b}_y}{\partial y}. \hfill (3.15)$$

The introduction of the Poisson-type pressure equation allows each governing equation to solve for one unknown while maintaining the symmetry of the original set of the N-S equations. The idea nevertheless contributes little to alleviate one of the major challenges with respect to solving the Navier-Stokes equations: the treatment of the nonlinear, convective terms in the momentum equations. Early Nodal Integral Method for the Navier-Stokes equations merged all terms except for the diffusion term into the pseudo-source terms during the so-called Transverse Integration Procedure [11,12]. This radical treatment of the nonlinearity renders a scheme that is not very satisfying in terms of accuracy. With the intention to retain more information from the convective terms by using a concept similar to the idea of “delayed-coefficients” [18], in the Modified Nodal Integral Method a set of equations obtained from adding linearized convective terms (partially evaluated from element-averaged velocities at
the previous time step) to both sides of the original momentum equations is solved [17, 18]. The consequent advantage of this treatment is detailed in reference [35]. Following reference [35], the two momentum equations (3.7) and (3.8) can be re-written as

$$\frac{\partial \hat{u}}{\partial t} + \hat{u}_p \frac{\partial \hat{u}}{\partial x} + \hat{v}_p \frac{\partial \hat{u}}{\partial y} - \nu \left( \frac{\partial^2 \hat{u}}{\partial x^2} + \frac{\partial^2 \hat{u}}{\partial y^2} \right) = -\frac{1}{\rho} \frac{\partial \hat{p}}{\partial x} - \hat{b}_x + (\hat{u}_p - \hat{u}) \frac{\partial \hat{u}}{\partial x} + (\hat{v}_p - \hat{v}) \frac{\partial \hat{u}}{\partial y},$$

(3.16)

and

$$\frac{\partial \hat{v}}{\partial t} + \hat{u}_p \frac{\partial \hat{v}}{\partial x} + \hat{v}_p \frac{\partial \hat{v}}{\partial y} - \nu \left( \frac{\partial^2 \hat{v}}{\partial x^2} + \frac{\partial^2 \hat{v}}{\partial y^2} \right) = -\frac{1}{\rho} \frac{\partial \hat{p}}{\partial y} + \hat{b}_y + (\hat{u}_p - \hat{u}) \frac{\partial \hat{v}}{\partial x} + (\hat{v}_p - \hat{v}) \frac{\partial \hat{v}}{\partial y},$$

(3.17)

where $\hat{u}_p$ and $\hat{v}_p$ represent the element-averaged $u$ and $v$ velocities from the previous time step, respectively. It should be noted that the equations (3.15), (3.16) and (3.17) are mathematically equivalent to the original set of Navier-Stokes equations (3.6), (3.7) and (3.8), and that the former are solved numerically in this dissertation work.

### 3.3 Transformed Navier-Stokes Equations

For a two-dimensional space decomposed into a union of generic quadrilateral elements in the global domain $\hat{\Omega}$, the equations (3.15), (3.16) and (3.17) can be mapped by applying the algebraic transformation technique (as discussed in section 3.1) within each element to a set of transformed governing equations described by local natural coordinates $(\xi, \eta)$. During the transformation, a trivial mapping is also implied in the time-dimension. The transformed equations for element $(i, j, k)^\dagger$ in the space-time

\[\text{\textsuperscript{\dagger}\text{In this chapter, } i \text{ and } j \text{ refer to spacial indices and } k \text{ refers to the time index.}}\]
and domain are:

\[
\frac{\partial u}{\partial t} + \left[ (\xi_x u_p + \xi_y v_p) - \nu (\xi_{xx} + \xi_{yy}) \right] \frac{\partial u}{\partial \xi} + \xi_x (u - u_p) \frac{\partial u}{\partial \xi} \\
+ \xi_y (v - v_p) \frac{\partial u}{\partial \xi} + \left[ (\eta_x u_p + \eta_y v_p) - \nu (\eta_{xx} + \eta_{yy}) \right] \frac{\partial u}{\partial \eta} + \eta_x (u - u_p) \frac{\partial u}{\partial \eta} \\
+ \eta_y (v - v_p) \frac{\partial u}{\partial \eta} = 0,
\]

(3.18)

\[
\frac{\partial v}{\partial t} + \left[ (\xi_x u_p + \xi_y v_p) - \nu (\xi_{xx} + \xi_{yy}) \right] \frac{\partial v}{\partial \xi} + \xi_x (u - u_p) \frac{\partial v}{\partial \xi} \\
+ \xi_y (v - v_p) \frac{\partial v}{\partial \xi} + \left[ (\eta_x u_p + \eta_y v_p) - \nu (\eta_{xx} + \eta_{yy}) \right] \frac{\partial v}{\partial \eta} + \eta_x (u - u_p) \frac{\partial v}{\partial \eta} \\
+ \eta_y (v - v_p) \frac{\partial v}{\partial \eta} = 0,
\]

(3.19)

and

\[
(\xi_x^2 + \xi_y^2) \frac{\partial^2 p}{\partial \xi^2} + (\eta_x^2 + \eta_y^2) \frac{\partial^2 p}{\partial \eta^2} + (\xi_{xx} + \xi_{yy}) \frac{\partial p}{\partial \xi} + (\eta_{xx} + \eta_{yy}) \frac{\partial p}{\partial \eta} \\
+ 2 (\xi_x \eta_x + \xi_y \eta_y) \frac{\partial^2 p}{\partial \xi \partial \eta} + \rho \left( \xi_x \frac{\partial u}{\partial \xi} + \eta_x \frac{\partial u}{\partial \eta} \right)^2 + \rho \left( \xi_y \frac{\partial v}{\partial \xi} + \eta_y \frac{\partial v}{\partial \eta} \right)^2 \\
+ 2 \rho \left( \xi_y \frac{\partial u}{\partial \xi} + \eta_y \frac{\partial u}{\partial \eta} \right) \left( \xi_x \frac{\partial v}{\partial \xi} + \eta_x \frac{\partial v}{\partial \eta} \right) + \rho (b_1 \xi_x + b_1 \eta_y) \\
+ \rho (b_2 \xi_x + b_2 \eta_y) = 0.
\]

(3.20)

Attention should be paid to the fact that the transformed equations are valid over each and every element. And the element-specific subscripts \((i, j, k)\) on all parameters and dependent variables are omitted for brevity.\(^1\) Also note that both notations for the partial derivatives are used in the above equations in order to differentiate unknown

\(^1\)Unless a different subscript is used, a subscript \((i, j, k)\) is implied for all parameters and dependent variables in the equations in the rest of this chapter. Also, if one or more of the three components in the subscript are omitted, then the omitted components are implied in the corresponding positions. For example, the shrinked subscript \((i, j - 1)\) implies \((i, j - 1, k)\); and the subscript \((k + 1)\) is a compact form of \((i, j, k + 1)\).
derivative terms from the known terms.†

The transformed set of equations are significantly more complicated than the original set. The transformed momentum equations involve additional nonlinear convective terms, e.g., \((\xi, y v \frac{\partial u}{\partial \xi})\) in equation (3.18) and \((\eta, x u \frac{\partial u}{\partial \eta})\) in equation (3.19); linear convective terms, e.g., \((\xi, x u_p \frac{\partial u}{\partial \xi})\) in equation (3.18) and \((\xi, y v_p \frac{\partial u}{\partial \xi})\) in equation (3.19); linear combination of pressure gradients in the two directions, e.g., \((\xi, x \frac{\partial p}{\partial \xi} + \eta, x \frac{\partial p}{\partial \eta})\) in equation (3.18) and \((\xi, y \frac{\partial p}{\partial \xi} + \eta, y \frac{\partial p}{\partial \eta})\) in equation (3.19); as well as mixed derivative terms similar to the diffusion term, e.g., \((\frac{\partial^2 u}{\partial \xi \partial \eta})\) in equation (3.18) and \((\frac{\partial^2 u}{\partial \xi \partial \eta})\) in equation (3.19). The transformed pressure equation involves additional, convection-like first derivative terms with constant coefficients, e.g., \((\xi, xx \frac{\partial p}{\partial \xi})\) and \((\eta, xx \frac{\partial p}{\partial \eta})\) in (3.20). The appearance of the additional terms are due to the transport process in the normal direction of an inclined surface between adjacent quadrilateral elements. These terms represent the components (in the axes directions) from the orthogonal decomposition of the convection and diffusion taking place along the surface normal. The transformed equations result from pure mathematical manipulations, hence no approximations are introduced in the transformation of the element and the governing equations. The following parameters are defined to simplify the subsequent development of the numerical method based on (3.18)–(3.20):

\[
K_1(\xi, \eta) = \eta, x u_p + \eta, y v_p - (\eta, xx + \eta, yy) \nu, \tag{3.21}
\]

\[
K_2(\xi, \eta) = (\eta, x^2 + \eta, y^2) \nu, \tag{3.22}
\]

\[
K_3(\xi, \eta) = \eta, xx + \eta, yy, \tag{3.23}
\]

\[
K_4(\xi, \eta) = - (\eta, x^2 + \eta, y^2), \tag{3.24}
\]

†The usual fractional notation is used for the unknown partial derivatives for emphasis, while the compact comma notation is used for partial derivatives that are known.
\( K_5(\xi, \eta) = \xi_x u_p + \xi_y v_p - (\xi_{xx} + \xi_{yy}) \nu, \) \hspace{1cm} (3.25)

\( K_6(\xi, \eta) = (\xi_{xx}^2 + \xi_{yy}^2) \nu, \) \hspace{1cm} (3.26)

\( K_7(\xi, \eta) = \xi_{xx} + \xi_{yy}, \) \hspace{1cm} (3.27)

\( K_8(\xi, \eta) = - (\xi_{xx}^2 + \xi_{yy}^2). \) \hspace{1cm} (3.28)

Applying (3.21)–(3.28), equations (3.18)–(3.20) are re-written in terms of \( K_n \) \((n = 1, \ldots, 8)\) as

\[
\frac{\partial u}{\partial t} + K_5 \frac{\partial u}{\partial \xi} + \xi_x (u - u_p) \frac{\partial u}{\partial \xi} + \xi_y (v - v_p) \frac{\partial u}{\partial \eta} + K_1 \frac{\partial u}{\partial \eta} + \eta_x (u - u_p) \frac{\partial u}{\partial \eta} + \xi_{xx} u_p + \xi_{yy} v_p - (\xi_{xx} + \xi_{yy}) \nu, \tag{3.29}
\]

\[
\frac{\partial v}{\partial t} + K_5 \frac{\partial v}{\partial \xi} + \xi_x (u - u_p) \frac{\partial v}{\partial \xi} + \xi_y (v - v_p) \frac{\partial v}{\partial \eta} + K_1 \frac{\partial v}{\partial \eta} + \eta_x (u - u_p) \frac{\partial v}{\partial \eta} + \xi_{xx} u_p + \xi_{yy} v_p - (\xi_{xx} + \xi_{yy}) \nu, \tag{3.30}
\]

\[
- K_8 \frac{\partial^2 p}{\partial \xi^2} = K_4 \frac{\partial^2 p}{\partial \eta^2} + K_7 \frac{\partial p}{\partial \eta} + K_3 \frac{\partial p}{\partial \eta} + 2 (\xi_{xx} \eta_x + \xi_{yy} \eta_y) \frac{\partial^2 p}{\partial \xi \partial \eta}, \tag{3.31}
\]

Generally, generic quadrilateral elements filling a 2D global domain will vary in shape. Each quadrilateral element is mapped into a square element in the computational space. Thus different mappings may result for different elements, depending on the shape of the element. As a consequence, the values of parameters \( K_n \) may also vary over the global domain, accordingly. Another consequence of the mapping is that the
resulting equations may take different forms for diverse elements. In the following sections, the modified nodal integral scheme is applied to the transformed Navier-Stokes equations (3.29)–(3.31) that are valid over local square elements.

3.4 Transverse Integration Procedure

Applying the transverse-integration operator†

\[
\frac{1}{4t_0} \int_{-t_0}^{t_0} \int_{-1}^{1} d\xi \, dt
\]

to equations (3.29), (3.30) and (3.31) respectively yields

\[
K_{10} \frac{d\pi^{\xi t}(\eta)}{d\eta} - K_{20} \frac{d^2\pi^{\xi t}(\eta)}{d\eta^2} = \overline{S_1}^{\xi t}(\eta),
\]

(3.32)

\[
K_{10} \frac{d\pi^{\xi t}(\eta)}{d\eta} - K_{20} \frac{d^2\pi^{\xi t}(\eta)}{d\eta^2} = \overline{S_2}^{\xi t}(\eta),
\]

(3.33)

\[
K_{30} \frac{d\pi^{\xi t}(\eta)}{d\eta} - K_{40} \frac{d^2\pi^{\xi t}(\eta)}{d\eta^2} = \overline{S_3}^{\xi t}(\eta),
\]

(3.34)

where the transverse-integrated unknowns are defined as

\[
\overline{\phi}^{\xi t}(\eta) = \frac{1}{4t_0} \int_{-t_0}^{t_0} \int_{-1}^{1} \phi(\xi, \eta, t) \, d\xi \, dt, \quad \phi = u, v, p,
\]

(3.35)

and \(K_{n0} (n = 1, \ldots, 4)\) are defined as the element-averaged \(K_n (n = 1, \ldots, 4)\), i.e.,

\[
K_{n0} = \frac{1}{4} \int_{-1}^{1} \int_{-1}^{1} K_n(\xi, \eta) \, d\xi \, d\eta, \quad n = 1, \ldots, 4.
\]

(3.36)

The resulting equations (3.32)–(3.34) are inhomogeneous, linear, second order, ordinary differential equations. The right hand sides of (3.32)–(3.34) are called pseudo-

†The parameter \(2t_0\) denotes the size of time step.
source terms in the MNIM. RHS’s are composed of the terms (mostly the ones for which the transverse-integration and partial differential operators do not commute) not explicit in the resulting equations. The pseudo-source terms are given by

\[
\overline{S}_1^{\xi t}(\eta) \equiv -\frac{1}{4t_0} \int_{-t_0}^{t_0} \int_{1}^{1} d\xi \, dt \left[ \frac{\partial u}{\partial t} + K_5 \frac{\partial u}{\partial \xi} + \xi_x (u - u_p) \frac{\partial u}{\partial \xi} + \xi_y (v - v_p) \frac{\partial u}{\partial \xi} \\
+ \eta_x (u - u_p) \frac{\partial u}{\partial \eta} + \eta_y (v - v_p) \frac{\partial u}{\partial \eta} - 2 \left( \xi_x \eta_x + \xi_y \eta_y \right) \nu \frac{\partial^2 u}{\partial \xi \partial \eta} \right] (3.37)
\]

\[
\overline{S}_2^{\xi t}(\eta) \equiv -\frac{1}{4t_0} \int_{-t_0}^{t_0} \int_{1}^{1} d\xi \, dt \left[ \frac{\partial v}{\partial t} + K_5 \frac{\partial v}{\partial \xi} + \xi_x (u - u_p) \frac{\partial v}{\partial \xi} + \xi_y (v - v_p) \frac{\partial v}{\partial \xi} \\
+ \eta_x (u - u_p) \frac{\partial v}{\partial \eta} + \eta_y (v - v_p) \frac{\partial v}{\partial \eta} - 2 \left( \xi_x \eta_x + \xi_y \eta_y \right) \nu \frac{\partial^2 v}{\partial \xi \partial \eta} \right] (3.38)
\]

and

\[
\overline{S}_3^{\xi t}(\eta) \equiv -\frac{1}{4t_0} \int_{-t_0}^{t_0} \int_{1}^{1} d\xi \, dt \left[ \frac{\partial^2 p}{\partial \xi^2} + K_7 \frac{\partial p}{\partial \xi} + 2 \left( \xi_x \eta_x + \xi_y \eta_y \right) \frac{\partial^2 p}{\partial \xi \partial \eta} \right. \\
\left. + \rho \left( \xi_x \frac{\partial u}{\partial \xi} + \eta_x \frac{\partial u}{\partial \eta} \right)^2 + 2 \rho \left( \xi_y \frac{\partial u}{\partial \xi} + \eta_y \frac{\partial u}{\partial \eta} \right) \left( \xi_x \frac{\partial v}{\partial \xi} + \eta_x \frac{\partial v}{\partial \eta} \right) \right] (3.39)
\]

It should be pointed out that the following two successive approximations commonly used in the NIM/MNIM [10, 15, 17, 36] have been used to arrive at equations (3.32)–(3.34). The first approximation is approximating the average of the product by product of the averages, namely, for two functions \( \Psi(\xi, \eta, t) \) and \( \Lambda(\xi, \eta, t) \),

\[
\frac{1}{4t_0} \int_{-t_0}^{t_0} \int_{1}^{1} \Psi(\xi, \eta, t) \Lambda(\xi, \eta, t) \, d\xi \, dt \approx \frac{1}{4t_0} \int_{-t_0}^{t_0} \int_{1}^{1} \Psi(\xi, \eta, t) \, d\xi \, dt \frac{1}{4t_0} \int_{-t_0}^{t_0} \int_{1}^{1} \Lambda(\xi, \eta, t) \, d\eta \, dt. \quad (3.40)
\]
The second approximation is

\[
\frac{1}{4t_0} \int_{-t_0}^{t_0} \int_{-1}^{1} K_n(\xi, \eta) \, d\xi \, dt \approx \frac{1}{4} \int_{-1}^{1} \int_{-1}^{1} K_n(\xi, \eta) \, d\xi \, d\eta = K_{n0}, \quad n = 1, \ldots, 4. \tag{3.41}
\]

It has already been shown that the above approximations are consistent with the order of the numerical scheme for NIM [10, 17, 32].

Similarly, applying the transverse-integration operator

\[
\frac{1}{4t_0} \int_{-t_0}^{t_0} \int_{-1}^{1} d\eta \, dt
\]

to equations (3.29), (3.30) and (3.31) respectively yields

\[
K_{50} \frac{d\Pi^{nt}(\xi)}{d\xi} - K_{60} \frac{d^2\Pi^{nt}(\xi)}{d\xi^2} = S_1^{nt}(\xi), \tag{3.42}
\]

\[
K_{50} \frac{d\Pi^{vt}(\xi)}{d\xi} - K_{60} \frac{d^2\Pi^{vt}(\xi)}{d\xi^2} = S_2^{nt}(\xi), \tag{3.43}
\]

and

\[
K_{70} \frac{d\Pi^{pt}(\xi)}{d\xi} - K_{80} \frac{d^2\Pi^{pt}(\xi)}{d\xi^2} = S_3^{nt}(\xi), \tag{3.44}
\]

where the transverse-integrated unknowns are defined as

\[
\bar{\phi}^{nt}(\xi) = \frac{1}{4t_0} \int_{-t_0}^{t_0} \int_{-1}^{1} \phi(\xi, \eta, t) \, d\eta \, dt, \quad \phi = u, v, p, \tag{3.45}
\]

and \(K_{n0} \ (n = 5, \ldots, 8)\) are defined as the element-averaged \(K_n \ (n = 5, \ldots, 8)\), i.e.,

\[
K_{n0} = \frac{1}{4} \int_{-1}^{1} \int_{-1}^{1} K_n(\xi, \eta) \, d\xi \, d\eta, \quad n = 5, \ldots, 8. \tag{3.46}
\]
The pseudo-source terms in (3.42)–(3.44) are given by

\[
S_{1}^{\eta t}(\xi) \equiv -\frac{1}{4t_{0}} \int_{t_{0}}^{t_{0}} \int_{-1}^{1} d\eta dt \left[ \frac{\partial u}{\partial t} + \xi_{,x} (u - u_{p}) \frac{\partial u}{\partial \xi} + \xi_{,y} (v - v_{p}) \frac{\partial u}{\partial \xi} + K_{1} \frac{\partial u}{\partial \eta} 
+ \eta_{,x} (u - u_{p}) \frac{\partial u}{\partial \eta} + \eta_{,y} (v - v_{p}) \frac{\partial u}{\partial \eta} - 2 (\xi_{,x} \eta_{,x} + \xi_{,y} \eta_{,y}) \nu \frac{\partial^{2} u}{\partial \xi \partial \eta} 
- K_{2} \frac{\partial^{2} u}{\partial \eta^{2}} + \frac{1}{\rho} \left( \xi_{,x} \frac{\partial u}{\partial \xi} + \eta_{,x} \frac{\partial u}{\partial \eta} \right) + b_{1}(\xi, \eta, t) \right],
\]

(3.47)

\[
S_{2}^{\eta t}(\xi) \equiv -\frac{1}{4t_{0}} \int_{t_{0}}^{t_{0}} \int_{-1}^{1} d\eta dt \left[ \frac{\partial v}{\partial t} + \xi_{,x} (u - u_{p}) \frac{\partial v}{\partial \xi} + \xi_{,y} (v - v_{p}) \frac{\partial v}{\partial \xi} + K_{1} \frac{\partial v}{\partial \eta} 
+ \eta_{,x} (u - u_{p}) \frac{\partial v}{\partial \eta} + \eta_{,y} (v - v_{p}) \frac{\partial v}{\partial \eta} - 2 (\xi_{,x} \eta_{,x} + \xi_{,y} \eta_{,y}) \nu \frac{\partial^{2} v}{\partial \xi \partial \eta} 
- K_{2} \frac{\partial^{2} v}{\partial \eta^{2}} + \frac{1}{\rho} \left( \xi_{,y} \frac{\partial v}{\partial \xi} + \eta_{,y} \frac{\partial v}{\partial \eta} \right) + b_{2}(\xi, \eta, t) \right],
\]

(3.48)

and

\[
S_{3}^{\eta t}(\xi) \equiv -\frac{1}{4t_{0}} \int_{t_{0}}^{t_{0}} \int_{-1}^{1} d\eta dt \left[ -K_{4} \frac{\partial^{2} p}{\partial \eta^{2}} + K_{3} \frac{\partial p}{\partial \eta} + 2 (\xi_{,x} \eta_{,x} + \xi_{,y} \eta_{,y}) \frac{\partial^{2} p}{\partial \xi \partial \eta} 
+ \rho \left( \xi_{,x} \frac{\partial u}{\partial \xi} + \eta_{,x} \frac{\partial u}{\partial \eta} \right)^{2} + 2 \rho \left( \xi_{,y} \frac{\partial u}{\partial \xi} + \eta_{,y} \frac{\partial u}{\partial \eta} \right) \left( \xi_{,x} \frac{\partial v}{\partial \xi} + \eta_{,x} \frac{\partial v}{\partial \eta} \right) 
+ \rho \left( \xi_{,y} \frac{\partial v}{\partial \xi} + \eta_{,y} \frac{\partial v}{\partial \eta} \right)^{2} + \rho (b_{1,1} \xi_{,x} + b_{1,1} \eta_{,x} + b_{2,1} \xi_{,y} + b_{2,1} \eta_{,y}) \right].
\]

(3.49)

Again, the approximation in equation (3.40) is used to arrive at equations (3.42)–(3.44). The other approximation (similar to (3.41)) used here is

\[
\frac{1}{4t_{0}} \int_{-t_{0}}^{t_{0}} \int_{-1}^{1} K_{n}(\xi, \eta) d\eta dt \approx \frac{1}{4} \int_{-1}^{1} \int_{-1}^{1} K_{n}(\xi, \eta) d\xi d\eta = K_{n0}, \quad n = 5, \ldots, 8. \quad (3.50)
\]

Next, applying the third (and last) transverse-integration operator

\[
\frac{1}{4} \int_{-1}^{1} \int_{-1}^{1} d\xi d\eta
\]

25
to equations (3.29) and (3.30) respectively yields

\[
\frac{d \overline{\phi}^\xi(t)}{dt} = \overline{S}_1^\xi(t),
\]
(3.51)

\[
\frac{d \overline{\phi}^\eta(t)}{dt} = \overline{S}_2^\eta(t),
\]
(3.52)

where the transverse-integrated unknowns are defined as

\[
\overline{\phi}^\xi(t) = \frac{1}{4} \int_{-1}^{1} \int_{-1}^{1} \phi(\xi, \eta, t) \, d\xi \, d\eta, \quad \phi = u, v,
\]
(3.53)

and the pseudo-source terms on the right hand sides are given by

\[
\overline{S}_1^\eta(t) \equiv -\frac{1}{4} \int_{-1}^{1} \int_{-1}^{1} d\xi d\eta \left[ K_5 \frac{\partial u}{\partial \xi} + \xi, \frac{\partial u}{\partial \xi} + \xi, \frac{\partial v}{\partial \eta} + \xi, \frac{\partial v}{\partial \eta} + 2 \left( \xi, \frac{\partial p}{\partial \xi} + \eta, \frac{\partial p}{\partial \eta} \right) + b_1(\xi, \eta, t) \right],
\]
(3.54)

\[
\overline{S}_2^\eta(t) \equiv -\frac{1}{4} \int_{-1}^{1} \int_{-1}^{1} d\xi d\eta \left[ K_5 \frac{\partial v}{\partial \xi} + \xi, \frac{\partial v}{\partial \xi} + \xi, \frac{\partial u}{\partial \eta} + \xi, \frac{\partial u}{\partial \eta} + 2 \left( \xi, \frac{\partial p}{\partial \xi} + \eta, \frac{\partial p}{\partial \eta} \right) + b_2(\xi, \eta, t) \right].
\]
(3.55)

No approximation is introduced arriving at equations (3.51) and (3.52). Unlike the previous two transverse-integration operators, applying the operator

\[
\frac{1}{4} \int_{-1}^{1} \int_{-1}^{1} d\xi d\eta
\]

to the pressure equation (3.31) does not generate transverse-integrated ordinary differential equations, due to the absence of time derivative terms in (3.31) [35]. As a
result, a total set of eight ordinary differential equations—(3.32)–(3.34), (3.42)–(3.44), and (3.51)–(3.52)—are obtained after the TIP; all with inhomogeneous pseudo-source terms. The pseudo-source terms not only are nonlinear, they also couple the set of ODE’s. In the NIM/MNIM, however, these pseudo-source terms are dealt with by first expanding them in Legendre polynomials and then truncating the expansions at the zeroth order†, which leads the NIM/MNIM to be a second order scheme for rectangular elements [10, 35]. Benefits of this approach are two-fold: simplifying the pseudo-source terms; decoupling the set of ODE’s. The eight transverse-integrated ODE’s thereafter become:

\[
K_{10} \frac{d\tilde{p}^{\xi t}(\eta)}{d\eta} - K_{20} \frac{d^2\tilde{p}^{\xi t}(\eta)}{d\eta^2} = S_{1}^{\xi t}, 
\]

\[
K_{10} \frac{d\tilde{p}^{\xi t}(\eta)}{d\eta} - K_{20} \frac{d^2\tilde{p}^{\xi t}(\eta)}{d\eta^2} = S_{2}^{\xi t}, 
\]

\[
K_{30} \frac{d\tilde{p}^{\xi t}(\eta)}{d\eta} - K_{40} \frac{d^2\tilde{p}^{\xi t}(\eta)}{d\eta^2} = S_{3}^{\xi t}, 
\]

\[
K_{50} \frac{d\tilde{p}^{\eta t}(\xi)}{d\xi} - K_{60} \frac{d^2\tilde{p}^{\eta t}(\xi)}{d\xi^2} = S_{1}^{\eta t}, 
\]

\[
K_{50} \frac{d\tilde{p}^{\eta t}(\xi)}{d\xi} - K_{60} \frac{d^2\tilde{p}^{\eta t}(\xi)}{d\xi^2} = S_{2}^{\eta t}, 
\]

\[
K_{70} \frac{d\tilde{p}^{\eta t}(\xi)}{d\xi} - K_{80} \frac{d^2\tilde{p}^{\eta t}(\xi)}{d\xi^2} = S_{3}^{\eta t}, 
\]

\[
\frac{dU^{\xi \eta}(t)}{dt} = S_{1}^{\xi \eta}, 
\]

\[
\frac{dU^{\xi \eta}(t)}{dt} = S_{2}^{\xi \eta}. 
\]

†More generally, truncation of the Legendre expansions of pseudo-source terms at a higher order may help elevate the order of the nodal methods; but that requires additional approximations [37,38].
Notice that the right hand sides in equations (3.56)–(3.63) represent constants—the zeroth order Legendre expansion of the corresponding pseudo-source terms in equations (3.32)–(3.34), (3.42)–(3.44), and (3.51)–(3.52). Similar to MNIM for regular elements, complete symmetry is observed between $u$ and $v$ velocities as well as the $\xi$ and $\eta$ directions in the formulation [35].

### 3.5 Solutions to the Transverse-Integrated Equations

Equations (3.56)–(3.61) and (3.62)–(3.63) are groups of second and first order, linear, ordinary differential equations, respectively, with constant inhomogeneous terms. The equations are all analytically solvable within each square element in the domain $\Omega$. The local solutions of equations (3.56)–(3.61) are either quadratic or of constant+linear+exponential form [17], depending on the geometry of the irregular quadrilateral element specified in domain $\hat{\Omega}$; while equations (3.62) and (3.63) have simple linear solutions.

For example, the solution of equation (3.56) is

$$w^{\xi t}(\eta) = -\frac{S_{1}^{\xi t}}{2K_{20}} \eta^2 + C_{1}\eta + C_{2} \tag{3.64}$$

if $K_{10} = 0$, and

$$w^{\xi t}(\eta) = C_{3} e^{\left(\frac{K_{10}}{K_{20}}\right)\eta} + \frac{S_{1}^{\xi t}}{K_{10}} \eta + C_{4} \tag{3.65}$$

if $K_{10} \neq 0$. The solution of equation (3.62) is

$$w^{\xi \eta}(t) = S_{1}^{\xi \eta} t + C_{5}. \tag{3.66}$$
It deserves emphasis that equation (3.64) is an asymptotic case of (3.65), meaning
the former can be recovered by taking the limit of the latter as $K_{10}$ approaches zero†.
The analytic formula (3.65) is however not suitable for calculating $\pi^{\xi t}(\eta)$ numerically
when $K_{10}$ becomes very close or identical to zero. Therefore equation (3.64) is also
retained.

In all, eight solutions are obtained for the eight transverse-integrated ODE’s. Note
$C_n (n = 1, \ldots, 5)$ in the above solutions refer to arbitrary constants that need to be
eliminated. Evaluating the solutions on the surfaces (boundaries) of each element
leads to expressions (in terms of the discrete transverse-integrated unknowns) for the
arbitrary constants, which are eliminated subsequently. For example, the following

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure3.2.png}
\caption{Continuity of the unknown $\pi^{\xi t}$ between adjacent elements.}
\end{figure}

equations hold for the transverse-integrated variable $\pi^{\xi t}(\eta)$ within element $(i, j)$ (see
† A Taylor series expansion of the right hand side of (3.65) verifies this statement.

29
If $K_{10_{i,j}} = 0$, then evaluating the left hand sides of equations (3.67) and (3.68) using (3.64) yields

$$
\frac{-S_{1_{i,j}}^{\xi t}(1)^2}{2K_{20_{i,j}}} + C_1(1) + C_2 = \bar{u}_{i,j}^{\xi t}
$$

and

$$
\frac{-S_{1_{i,j}}^{\xi t}(-1)^2}{2K_{20_{i,j}}} + C_1(-1) + C_2 = \bar{u}_{i,j-1}^{\xi t},
$$

respectively. Algebraic equations (3.69) and (3.70) can be easily used to solve for $C_1$ and $C_2$ as follows:

$$
C_1 = \frac{\bar{u}_{i,j}^{\xi t} - \bar{u}_{i,j-1}^{\xi t}}{2},
$$

$$
C_2 = \frac{\bar{u}_{i,j}^{\xi t} + \bar{u}_{i,j-1}^{\xi t}}{2} + \frac{S_{1_{i,j}}^{\xi t}}{2K_{20_{i,j}}},
$$

Then substituting (3.71) and (3.72) into equation (3.64) results in the elimination of $C_1$ and $C_2$. All the other arbitrary constants are eliminated in the same manner, and hence local solutions to the eight transverse-integrated unknowns ($\bar{u}^{\xi t}$, $\bar{u}^{\eta t}$, $\bar{u}^{\xi \eta}$, $\bar{v}^{\xi t}$, $\bar{v}^{\eta t}$, $\bar{v}^{\xi \eta}$, $\bar{p}^{\xi t}$, $\bar{p}^{\eta t}$) in the ODE’s (3.56)–(3.63) can be written in expressions free of arbitrary constants. These expressions are next used to formulate a set of discrete equations by imposing continuity conditions.
3.6 Continuity of the Unknowns and their Directional Derivatives

From the conservation law, continuity of the transverse-integrated unknowns must be ensured on surfaces between adjacent elements. For the two unknowns $\bar{u}^{\xi \eta}$ and $\bar{v}^{\xi \eta}$ in the first order ODE’s (equations (3.62) and (3.63)), continuity condition is easily imposed by evaluating each of the local solutions at $t = t_0$, resulting in the following two discrete equations:

$$
\bar{u}^{\xi \eta}_{i,j} - \bar{u}^{\xi \eta}_{i,j,k} - 2 t_0 \bar{S}^{\xi \eta}_{1i,j} = 0, \quad (3.73)
$$

$$
\bar{v}^{\xi \eta}_{i,j} - \bar{v}^{\xi \eta}_{i,j,k} - 2 t_0 \bar{S}^{\xi \eta}_{2i,j} = 0. \quad (3.74)
$$

However, for the other six unknowns ($\bar{u}^{\xi t}$, $\bar{v}^{\xi t}$, $\bar{v}^{\eta t}$, $\bar{p}^{\xi t}$ and $\bar{p}^{\eta t}$) in the second order ODE’s (3.56)–(3.61), two continuity conditions need to be satisfied. The first is continuity of the unknown functions, and this has already been implicitly applied during the process of eliminating the arbitrary constants in section 3.5. The second is continuity of unknown function’s directional derivative in the direction normal to the common surface of neighbouring elements, which is elaborated below.

Consider two adjacent quadrilateral elements $(i, j)$ and $(i + 1, j)$ in the global $(x, y)$–domain as shown in figure 3.3(a). The continuity of the directional derivative of the unknown $\phi$ ($\phi = u, v, p$), in the direction normal to $AD$, is enforced by the following equation:

$$
\mathbf{m}_1 \cdot \nabla \hat{\phi}_{i,j}(x_{10}, y_{10}) = \mathbf{m}_1 \cdot \nabla \hat{\phi}_{i+1,j}(x_{10}, y_{10}), \quad (3.75)
$$

where $\mathbf{m}_1 = (m_{1x}, m_{1y})$ denotes to the normal of the edge $AD$, $(x_{10}, y_{10})$ denotes to
Figure 3.3: Compatibility criterion between adjacent elements.
the midpoint of \( AD \), and

\[
\nabla \hat{\varphi}_{i,j} = \left( \frac{\partial \hat{\varphi}_{i,j}}{\partial x}, \frac{\partial \hat{\varphi}_{i,j}}{\partial y} \right). \tag{3.76}
\]

Note that the following equation is required to relate the gradient in the global domain to the gradient in the local \((\xi, \eta)\)-domain:

\[
\nabla \hat{\varphi}_{i,j} = \nabla \varphi_{i,j} \begin{pmatrix} \xi, \xi, \\ \eta, \eta \end{pmatrix} = \left( \frac{\partial \varphi_{i,j}}{\partial \xi}, \frac{\partial \varphi_{i,j}}{\partial \eta} \right) \begin{pmatrix} \xi, \xi, \\ \eta, \eta \end{pmatrix}. \tag{3.77}
\]

Using (3.77), equation (3.75) becomes

\[
\left( \xi_{m1,i,j} \frac{\partial \varphi_{i,j}}{\partial \xi} + \eta_{m1,i,j} \frac{\partial \varphi_{i,j}}{\partial \eta} \right) \bigg|_{(\xi_{10}, \eta_{10})} = \left( \xi_{m1,i,j} \frac{\partial \varphi_{i+1,j}}{\partial \xi} + \eta_{m1,i,j} \frac{\partial \varphi_{i+1,j}}{\partial \eta} \right) \bigg|_{(\xi_{10}, \eta_{10})}, \tag{3.78}
\]

where the two coefficients \( \xi_{m1} \) and \( \eta_{m1} \) are defined by

\[
\xi_{m1} = m_{1x} \xi + m_{1y} \xi, \tag{3.79}
\]

\[
\eta_{m1} = m_{1x} \eta + m_{1y} \eta, \tag{3.80}
\]

and \((\xi_{10}, \eta_{10})\) is the midpoint of edge \( A'D' \) (shown in figure 3.3(b)), i.e., \((\xi_{10}, \eta_{10}) = (1, 0)\). The terms \( \frac{\partial \varphi_{i,j}}{\partial \xi} \) and \( \frac{\partial \varphi_{i,j}}{\partial \eta} \) (actually resulting from the two components of the gradient \( \nabla \varphi_{i,j} \)) in equation (3.78) can be approximated by the transverse-averaged values at the midpoint of each edge as:

\[
\frac{\partial \varphi_{i,j}}{\partial \xi} (\xi, \eta) \bigg|_{\xi=1, \eta=0} = \frac{d \varphi_{i,j}^{xl}}{d \xi} (\xi) \bigg|_{\xi=1}, \tag{3.81}
\]

\[
\frac{\partial \varphi_{i,j}}{\partial \eta} (\xi, \eta) \bigg|_{\xi=1, \eta=0} = \frac{d \varphi_{i,j}^{xt}}{d \eta} (\eta) \bigg|_{\eta=0}. \tag{3.82}
\]
Applying the approximation, equation (3.78) is re-written in terms of the transverse-integrated unknowns as:

\[
\left[ \xi_{m1,i,j} \left( \frac{d\overline{\phi}_{t,i,j}(\xi)}{d\xi} - d\overline{\phi}_{t,i+1,j}(\xi) \right) + \eta_{m1,i,j} \left( \frac{d\overline{\phi}_{t,i,j}(\eta)}{d\eta} - d\overline{\phi}_{t,i+1,j}(\eta) \right) \right] \bigg|_{\xi=1,\eta=0} = 0. \quad (3.83)
\]

The above process can be similarly repeated at the interface \(AB\) of the elements \((i,j)\) and \((i,j+1)\) to yield

\[
\left[ \xi_{m2,i,j} \left( \frac{d\overline{\phi}_{t,i,j}(\xi)}{d\xi} - d\overline{\phi}_{t,i+1,j}(\xi) \right) + \eta_{m2,i,j} \left( \frac{d\overline{\phi}_{t,i,j}(\eta)}{d\eta} - d\overline{\phi}_{t,i+1,j}(\eta) \right) \right] \bigg|_{\xi=0,\eta=1} = 0. \quad (3.84)
\]

In equation (3.84), the two coefficients \(\xi_{m2}\) and \(\eta_{m2}\) are defined by

\[
\xi_{m2} = m_{2x} \xi, x + m_{2y} \xi, y, \quad (3.85)
\]

\[
\eta_{m2} = m_{2x} \eta, x + m_{2y} \eta, y, \quad (3.86)
\]

where \(m_{2x}\) and \(m_{2y}\) are components of the normal vector \(m_2\) of the edge \(AB\). Note that the symbol \(\phi\) can stand for \(u, v\) and \(p\). Therefore replacing \(\phi\) by \(u, v\) and \(p\) in (3.83) and (3.84) yields six equations, in which the transverse-integrated unknowns are successively substituted by the analytical local solutions obtained in the previous section. After the above steps, six discrete algebraic equations are generated for unknowns \(\overline{u}^t, \overline{v}^t, \overline{\tau}^t, \overline{\nu}^t, \overline{p}^t\) and \(\overline{\varphi}^t\); and they are listed together with equations (3.73) and (3.74) as follows:

\[
A_{11}(\overline{u}_{t,i,j} - \overline{u}_{t,i-1,j}) + A_{12}(\overline{u}_{t,i+1,j-1} - \overline{u}_{t,i+1,j}) - A_{13} \overline{u}_{t,i-1,j} + A_{14} \overline{u}_{t,i,j} + A_{15} \overline{u}_{t,i+1,j} \\
- A_{16} \overline{S}_{1i,j}^t + A_{17} \overline{S}_{1i+1,j}^t + A_{18} \overline{S}_{1i,j}^{\xi_t} + A_{19} \overline{S}_{1i+1,j}^{\xi_t} = 0, \quad (3.87)
\]

\[
A_{11}(\overline{v}_{t,i,j} - \overline{v}_{t,i-1,j}) + A_{12}(\overline{v}_{t,i+1,j-1} - \overline{v}_{t,i+1,j}) - A_{13} \overline{v}_{t,i-1,j} + A_{14} \overline{v}_{t,i,j} + A_{15} \overline{v}_{t,i+1,j} \\
- A_{16} \overline{S}_{2i,j}^t + A_{17} \overline{S}_{2i+1,j}^t + A_{18} \overline{S}_{2i,j}^{\xi_t} + A_{19} \overline{S}_{2i+1,j}^{\xi_t} = 0, \quad (3.88)
\]
\[ A_{31}(p_{ij}^{\xi} - p_{i,j-1}^{\xi}) + A_{32}(p_{i+1,j}^{\xi} - p_{i,j}^{\xi}) - A_{33} p_{i-1,j}^{\eta} + A_{34} p_{i,j}^{\eta} + A_{35} p_{i+1,j}^{\eta} = 0, \]  
\[ A_{36} S_{1i,j}^{\eta} + A_{37} S_{1i+1,j}^{\eta} + A_{38} S_{3i,j}^{\eta} + A_{39} S_{3i+1,j}^{\eta} = 0, \]  
\[ - A_{41} u_{ij}^{\xi} + A_{42} u_{ij}^{\eta} + A_{43} u_{i,j+1}^{\xi} + A_{44}(u_{i-1,j}^{\eta} - u_{i,j+1}^{\eta}) + A_{45}(u_{i,j}^{\eta} - u_{i-1,j}^{\eta}) = 0, \]  
\[ + A_{46} S_{1i,j}^{\eta} + A_{47} S_{1i+1,j}^{\eta} - A_{48} S_{2i,j}^{\eta} + A_{49} S_{2i+1,j}^{\eta} = 0, \]  
\[ - A_{51} v_{ij}^{\xi} + A_{52} v_{ij}^{\eta} + A_{53} v_{i,j+1}^{\xi} + A_{54}(v_{i-1,j}^{\eta} - v_{i,j+1}^{\eta}) + A_{55}(v_{i,j}^{\eta} - v_{i-1,j}^{\eta}) = 0, \]  
\[ + A_{56} S_{2i,j}^{\eta} + A_{57} S_{2i+1,j}^{\eta} - A_{58} S_{3i,j}^{\eta} + A_{59} S_{3i+1,j}^{\eta} = 0, \]  
\[ - A_{61} p_{ij}^{\xi} + A_{62} p_{ij}^{\eta} + A_{63} p_{i,j+1}^{\xi} + A_{64}(p_{i-1,j}^{\eta} - p_{i,j+1}^{\eta}) + A_{65}(p_{i,j}^{\eta} - p_{i-1,j}^{\eta}) = 0, \]  
\[ + A_{66} S_{3i,j}^{\eta} + A_{67} S_{3i+1,j}^{\eta} - A_{68} S_{4i,j}^{\eta} + A_{69} S_{4i+1,j}^{\eta} = 0, \]  
\[ \bar{w}_i^{\xi} - \bar{w}_{i,j,k-1}^{\xi} - 2t_0 \bar{S}_{1i,j}^{\eta} = 0, \]  
\[ \bar{v}_i^{\xi} - \bar{v}_{i,j,k-1}^{\xi} - 2t_0 \bar{S}_{2i,j}^{\eta} = 0, \]  

in which the coefficients \( A \)'s are functions of the intermediate parameter \( K \)'s, as well as geometric coefficients \( \xi_{m1}, \xi_{m2}, \eta_{m1}, \) and \( \eta_{m2} \). The \( A \)'s are defined (see appendix C) to simplify the expressions in the development of the scheme. For example,

\[ A_{11} = \frac{\eta_{m1} \cosh \left( \frac{K_{i+1}}{K_2} \right) K_1}{2K_2}; \quad A_{12} = \frac{\eta_{m1} \cosh \left( \frac{K_{i+1}}{K_{2i+1}} \right) K_{1i+1}}{2K_{2i+1}}. \]  

So far, continuity of eight unknowns \((\bar{u}^{\xi}, \bar{u}^{\eta}, \bar{u}^{\xi}, \bar{v}^{\xi}, \bar{v}^{\eta}, \bar{v}^{\xi}, \bar{p}^{\xi}, \bar{p}^{\eta})\) and their directional derivatives on interface of neighboring elements have been imposed to obtain a set of discrete algebraic equations. Note that the eight discrete equations obtained above involve, except for the eight transverse-integrated unknowns, the same number of unknown pseudo-source terms. To eliminate the pseudo-source terms, eight more discrete equations are arrived at in the next section by imposing constraint.
3.7 Constraint Conditions for the Elimination of Pseudo-Source Terms

The idea of constraint conditions comes from the intrinsic requirements of MNIM that: 1) the element-averaged continuity and momentum equations be satisfied in the MNIM; 2) each transverse-integrated unknown be unique, regardless of the order of multiple integration in the transverse-integration procedure. Following NIM [10] and MNIM [17], these two constraint conditions are applied to eliminate the pseudo-source terms.

From the first set of constraint conditions, i.e., ensuring that equations (3.18)–(3.20) are satisfied within each element in an integral sense, three equations are derived by first applying the element-averaging operator

$$\frac{1}{8t_0} \int^{t_0}_{-t_0} \int^{1}_{-1} \int^{1}_{-1} d\xi \, d\eta \, dt$$

to both sides of (3.18)–(3.20), and then invoking the definitions of pseudo-source terms (equations (3.37)–(3.39), (3.47)–(3.49), (3.54)–(3.55)). The three equations thus obtained are

$$\overline{S^t_{\xi}} + \overline{S^t_{\eta}} + \overline{S^\xi_{\eta}} + f_1 = 0$$

(3.96)

$$\overline{S^t_{\xi}} + \overline{S^t_{\eta}} + \overline{S^\xi_{\eta}} + f_2 = 0$$

(3.97)

and

$$\overline{S^t_{\xi}} + \overline{S^t_{\eta}} + f_3 = 0,$$

(3.98)
respectively, where $f$ terms are defined as

\begin{align*}
    f_1 & \equiv \xi, x_0 (u_0 - u_p) + \xi, y_0 (v_0 - v_p) \frac{1}{2} (\bar{u}_{i,j} - \bar{u}_{i-1,j}) \\
    & + [\eta, x_0 (u_0 - u_p) + \eta, y_0 (v_0 - v_p)] \frac{1}{2} (\bar{v}_{i,j} - \bar{v}_{i-1,j}) \\
    & + \frac{\xi, x_0}{\rho} \frac{1}{2} (\bar{p}_{t,i,j} - \bar{p}_{t,i-1,j}) + \frac{\eta, y_0}{\rho} \frac{1}{2} (\bar{p}_{t,i,j} - \bar{p}_{t,i-1,j}) + \bar{b}_{1i,j}^t,
\end{align*}

\begin{align*}
    f_2 & \equiv \xi, x_0 (u_0 - u_p) + \xi, y_0 (v_0 - v_p) \frac{1}{2} (\bar{v}_{i,j} - \bar{v}_{i-1,j}) \\
    & + [\eta, x_0 (u_0 - u_p) + \eta, y_0 (v_0 - v_p)] \frac{1}{2} (\bar{v}_{i,j} - \bar{v}_{i-1,j}) \\
    & + \frac{\xi, y_0}{\rho} \frac{1}{2} (\bar{p}_{t,i,j} - \bar{p}_{t,i-1,j}) + \frac{\eta, y_0}{\rho} \frac{1}{2} (\bar{p}_{t,i,j} - \bar{p}_{t,i-1,j}) + \bar{b}_{2i,j}^t.
\end{align*}

and

\begin{align*}
    f_3 & \equiv \rho \left[ \xi, x_0 \frac{1}{2} (\bar{u}_{i,j} - \bar{u}_{i-1,j}) + \eta, x_0 \frac{1}{2} (\bar{v}_{i,j} - \bar{v}_{i-1,j}) \right]^2 \\
    & + 2 \rho \left[ \xi, y_0 \frac{1}{2} (\bar{u}_{i,j} - \bar{u}_{i-1,j}) + \eta, y_0 \frac{1}{2} (\bar{v}_{i,j} - \bar{v}_{i-1,j}) \right] \\
    & \cdot \left[ \xi, x_0 \frac{1}{2} (\bar{u}_{i,j} - \bar{u}_{i-1,j}) + \eta, x_0 \frac{1}{2} (\bar{v}_{i,j} - \bar{v}_{i-1,j}) \right] \\
    & + \rho \left[ \xi, y_0 \frac{1}{2} (\bar{v}_{i,j} - \bar{v}_{i-1,j}) + \eta, y_0 \frac{1}{2} (\bar{v}_{i,j} - \bar{v}_{i-1,j}) \right]^2 \\
    & + \rho \left[ \xi, x_0 \frac{1}{2} (\bar{b}_{1,i,j}^t - \bar{b}_{1,i-1,j}^t) + \eta, x_0 \frac{1}{2} (\bar{b}_{1,i,j}^t - \bar{b}_{1,i-1,j}^t) \right] \\
    & + \rho \left[ \xi, y_0 \frac{1}{2} (\bar{b}_{2,i,j}^t - \bar{b}_{2,i-1,j}^t) + \eta, y_0 \frac{1}{2} (\bar{b}_{2,i,j}^t - \bar{b}_{2,i-1,j}^t) \right].
\end{align*}

Note that the approximation (3.40) used in section 3.4 is also used here to arrive at the above $f$’s. Taking the term

\[
    \xi, x (u - u_p) \frac{\partial u}{\partial \xi}
\]

in equation (3.18) as an example, this term is locally averaged and then approximated
through the following steps:

\[
\frac{1}{8t_0} \int_{-t_0}^{t_0} \int_{-1}^{1} \int_{-1}^{1} \xi_x (u - u_p) \frac{\partial u}{\partial \xi} \, d\xi \, d\eta \, dt \\
\approx \left( \frac{1}{8t_0} \int_{-t_0}^{t_0} \int_{-1}^{1} \int_{-1}^{1} \xi_x \, d\xi \, d\eta \, dt \right) \left( \frac{1}{8t_0} \int_{-t_0}^{t_0} \int_{-1}^{1} \int_{-1}^{1} (u - u_p) \frac{\partial u}{\partial \xi} \, d\xi \, d\eta \, dt \right)
\]

\[
= \xi_{x_0} \left\{ \frac{1}{2} \int_{-1}^{1} \left[ \frac{1}{4t_0} \int_{-t_0}^{t_0} \int_{-1}^{1} (u - u_p) \, d\eta \, dt \right] \left[ \frac{1}{4t_0} \int_{-t_0}^{t_0} \int_{-1}^{1} \frac{\partial u}{\partial \xi} \, d\eta \, dt \right] \, d\xi \right\}
\]

\[
\approx \xi_{x_0} \left\{ \frac{1}{2} \int_{-1}^{1} \left( \bar{w}^{nt}(\xi) - u_p \right) \frac{d\bar{w}^{nt}(\xi)}{d\xi} \, d\xi \right\}
\]

(3.102)

\[
= \xi_{x_0} \left\{ \frac{1}{2} \int_{-1}^{1} \left( \bar{w}^{nt}(\xi) - u_p \right) \frac{d\bar{w}^{nt}(\xi)}{d\xi} \, d\xi \right\}
\]

\[
= \xi_{x_0} \left( u_0 - u_p \right) \frac{1}{2} \left( \bar{u}_{i,j}^{nt} - \bar{u}_{i-1,j}^{nt} \right),
\]

where the coefficient \( \xi_{x_0} \) is formally defined as:

\[
\xi_{x_0} = \frac{1}{8t_0} \int_{-t_0}^{t_0} \int_{-1}^{1} \int_{-1}^{1} \xi_x \, d\xi \, d\eta \, dt = \frac{1}{4} \int_{-1}^{1} \int_{-1}^{1} \xi_x \, d\xi \, d\eta,
\]

(3.103)

and \( u_0 \) is the element-averaged \( u \)-velocity at the current time step, given by

\[
u_{0,i,j} = \frac{1}{4} \left( u_{i,j}^{nt} + u_{i,j}^{nt} + u_{i,j-1}^{nt} + u_{i-1,j}^{nt} \right).
\]

(3.104)

Approximations to the other terms can be similarly applied; and the geometric coefficients \( \xi_{,y_0}, \eta_{,x_0} \) and \( \eta_{,y_0} \) in equations (3.99)–(3.101) have definitions similar to \( \xi_{,x_0} \).

The requirement for uniqueness of the transverse-integrated unknowns indepen-

---

\(^1\)In the numerical implementation, \( \xi_{,x_0} \) is calculated using two-point Gauss quadrature.

\(^2\)The absence of cross derivative terms in the \( f \)'s is due to the approximation that the cross derivatives diminish after being transverse-integrated in both spatial directions [32].

---

38
dent of the order of integration, from the second constraint conditions, can be math-
ematically formulated as below:

\[ u_{\xi t} \equiv \frac{1}{2} \int_{-1}^{1} u_{\xi t}(\eta) \, d\eta = \frac{1}{2t_0} \int_{-t_0}^{t_0} u_{\xi}(t) \, dt \equiv u^{\xi t}, \quad (3.105) \]

\[ u_{\eta t} \equiv \frac{1}{2} \int_{-1}^{1} u_{\eta t}(\xi) \, d\eta = \frac{1}{2t_0} \int_{-t_0}^{t_0} u_{\eta}(t) \, dt \equiv u^{\eta t}, \quad (3.106) \]

\[ v_{\xi t} \equiv \frac{1}{2} \int_{-1}^{1} v_{\xi t}(\eta) \, d\eta = \frac{1}{2t_0} \int_{-t_0}^{t_0} v_{\xi}(t) \, dt \equiv v^{\xi t}, \quad (3.107) \]

\[ v_{\eta t} \equiv \frac{1}{2} \int_{-1}^{1} v_{\eta t}(\xi) \, d\eta = \frac{1}{2t_0} \int_{-t_0}^{t_0} v_{\eta}(t) \, dt \equiv v^{\eta t}, \quad (3.108) \]

\[ p_{\xi t} \equiv \frac{1}{2} \int_{-1}^{1} p_{\xi t}(\eta) \, d\eta = \frac{1}{2} \int_{-1}^{1} p_{\xi}(\xi) \, d\xi \equiv p^{\xi t}, \quad (3.109) \]

where the unknowns are consequently substituted by the local solutions (see section 3.5), leading to the following five equations\(^\dagger\):

\[ A_{c1} u_{i,j-1}^{\xi t} + A_{c2} u_{i,j}^{\xi t} + A_{c3} S_{1i,j}^{\xi t} - t_0 S_{1i,j}^{\xi \eta} - u_{i,j,k-1}^{\xi \eta} = 0, \quad (3.110) \]

\[ A_{d1} u_{i-1,j}^{\eta t} + A_{d2} u_{i,j}^{\eta t} + A_{d3} S_{1i,j}^{\eta t} - t_0 S_{2i,j}^{\eta \eta} - u_{i,j,k-1}^{\eta \eta} = 0, \quad (3.111) \]

\[ A_{c1} v_{i,j-1}^{\xi t} + A_{c2} v_{i,j}^{\xi t} + A_{c3} S_{2i,j}^{\xi t} - t_0 S_{2i,j}^{\xi \eta} - v_{i,j,k-1}^{\xi \eta} = 0, \quad (3.112) \]

\[ A_{d1} v_{i-1,j}^{\eta t} + A_{d2} v_{i,j}^{\eta t} + A_{d3} S_{2i,j}^{\eta t} - t_0 S_{2i,j}^{\eta \eta} - v_{i,j,k-1}^{\eta \eta} = 0, \quad (3.113) \]

\[ A_{g1} p_{i,j-1}^{\xi t} + A_{g2} p_{i,j}^{\xi t} - A_{g3} p_{i-1,j}^{\eta t} + A_{g4} p_{i,j}^{\eta t} + A_{g5} S_{1i,j}^{\xi \eta} + A_{g6} S_{2i,j}^{\eta \eta} = 0. \quad (3.114) \]

Thus, eight equations—equations (3.96)–(3.98) and (3.110)–(3.114)—are obtained from the constraint conditions. The pseudo-source terms are solved using these equations. Then the solutions of the pseudo-source terms, in terms of the transverse-

\(^\dagger\)The definition of the coefficients \(A\)'s in these equations are listed in appendix C.
integrated unknowns, are substituted into equations (3.87)–(3.94), leading to the elimination of the pseudo-source terms therein, and most importantly, to the final set of eight discrete equations containing the same number of unknowns for each element.

## 3.8 Final Set of Discrete Equations

The final set of discrete, algebraic equations for the eight transverse-averaged unknowns are

\[ F_{11} \tilde{u}_{i,j}^{\xi t} = F_{12} \tilde{u}_{i-1,j}^{\eta t} + F_{13} \tilde{u}_{i+1,j}^{\eta t} + F_{14} \tilde{u}_{i,j+1}^{\xi t} + F_{15} \tilde{u}_{i,j-1}^{\xi t} + F_{16} \tilde{u}_{i+1,j}^{\xi t} \]

\[ + F_{17} \tilde{u}_{i+1,j-1}^{\eta t} + F_{18} (\tilde{v}_{i,j}^{\xi \eta} + \tilde{v}_{i,j,k-1}^{\xi \eta}) + F_{19} (\tilde{v}_{i+1,j}^{\xi \eta} + \tilde{v}_{i+1,j,k-1}^{\xi \eta}) \]  \hspace{1cm} (3.115)

\[ F_{11} \tilde{v}_{i,j}^{\eta t} = F_{12} \tilde{v}_{i-1,j}^{\eta t} + F_{13} \tilde{v}_{i+1,j}^{\eta t} + F_{14} \tilde{v}_{i,j+1}^{\eta t} + F_{15} \tilde{v}_{i,j-1}^{\eta t} + F_{16} \tilde{v}_{i+1,j}^{\eta t} \]

\[ + F_{17} \tilde{v}_{i+1,j-1}^{\eta t} + F_{18} (\tilde{u}_{i,j}^{\xi \eta} + \tilde{u}_{i,j,k-1}^{\xi \eta}) + F_{19} (\tilde{u}_{i+1,j}^{\xi \eta} + \tilde{u}_{i+1,j,k-1}^{\xi \eta}) \]  \hspace{1cm} (3.116)

\[ F_{31} \tilde{p}_{i,j}^{\eta t} = F_{32} \tilde{p}_{i-1,j}^{\eta t} + F_{33} \tilde{p}_{i+1,j}^{\eta t} + F_{34} \tilde{p}_{i,j+1}^{\eta t} + F_{35} \tilde{p}_{i,j-1}^{\eta t} + F_{36} \tilde{p}_{i+1,j}^{\eta t} \]

\[ + F_{37} \tilde{p}_{i+1,j-1}^{\eta t} + F_{38} \tilde{f}_{3i,j} + F_{39} \tilde{f}_{3i+1,j} \]  \hspace{1cm} (3.117)

\[ F_{41} \tilde{u}_{i,j}^{\xi t} = F_{42} \tilde{u}_{i,j-1}^{\xi t} + F_{43} \tilde{u}_{i,j+1}^{\xi t} + F_{44} \tilde{u}_{i,j}^{\xi t} + F_{45} \tilde{u}_{i,j-1}^{\xi t} + F_{46} \tilde{u}_{i,j+1}^{\xi t} \]

\[ + F_{47} \tilde{u}_{i-1,j}^{\xi t} + F_{48} (\tilde{v}_{i,j}^{\xi \eta} + \tilde{v}_{i,j,k-1}^{\xi \eta}) + F_{49} (\tilde{v}_{i+1,j}^{\xi \eta} + \tilde{v}_{i+1,j,k-1}^{\xi \eta}) \]  \hspace{1cm} (3.118)

\[ F_{41} \tilde{v}_{i,j}^{\xi t} = F_{42} \tilde{v}_{i,j-1}^{\xi t} + F_{43} \tilde{v}_{i,j+1}^{\xi t} + F_{44} \tilde{v}_{i,j}^{\xi t} + F_{45} \tilde{v}_{i,j-1}^{\xi t} + F_{46} \tilde{v}_{i,j+1}^{\xi t} \]

\[ + F_{47} \tilde{v}_{i-1,j}^{\xi t} + F_{48} (\tilde{u}_{i,j}^{\xi \eta} + \tilde{u}_{i,j,k-1}^{\xi \eta}) + F_{49} (\tilde{u}_{i+1,j}^{\xi \eta} + \tilde{u}_{i+1,j,k-1}^{\xi \eta}) \]  \hspace{1cm} (3.119)

\[ F_{61} \tilde{p}_{i,j}^{\xi t} = F_{62} \tilde{p}_{i-1,j}^{\xi t} + F_{63} \tilde{p}_{i,j+1}^{\xi t} + F_{64} \tilde{p}_{i,j}^{\xi t} + F_{65} \tilde{p}_{i,j-1}^{\xi t} + F_{66} \tilde{p}_{i,j+1}^{\xi t} \]

\[ + F_{67} \tilde{p}_{i-1,j+1}^{\xi t} + F_{68} \tilde{f}_{3i,j} + F_{69} \tilde{f}_{3i+1,j} \]  \hspace{1cm} (3.120)

\[ F_{71} \tilde{u}_{i,j}^{\xi \eta} = F_{72} \tilde{u}_{i,j,k-1}^{\xi \eta} + F_{73} \tilde{u}_{i,j}^{\eta t} + F_{74} \tilde{u}_{i-1,j}^{\eta t} + F_{75} \tilde{u}_{i,j}^{\xi t} + F_{76} \tilde{u}_{i,j-1}^{\xi t} + f_{1i,j} \]  \hspace{1cm} (3.121)

\[ F_{71} \tilde{v}_{i,j}^{\xi \eta} = F_{72} \tilde{v}_{i,j,k-1}^{\xi \eta} + F_{73} \tilde{v}_{i,j}^{\eta t} + F_{74} \tilde{v}_{i-1,j}^{\eta t} + F_{75} \tilde{v}_{i,j}^{\xi t} + F_{76} \tilde{v}_{i,j-1}^{\xi t} + f_{2i,j} \]  \hspace{1cm} (3.122)
Here, the coefficients $F$’s\textsuperscript{†} are functions of intermediate coefficients $A$’s defined in the previous sections, e.g.,

$$F_{11} \equiv -A_{14} - \frac{A_{16}A_{d2}}{A_{d3}} + \frac{A_{17}A_{d1,i+1}}{A_{d3,i+1}}; \quad F_{12} \equiv -A_{13} + \frac{A_{16}A_{d1}}{A_{d3}}. \quad (3.123)$$

A schematic illustration of the computational stencil for the final set of discrete equations is shown in figure 3.4. The unknown $\overline{\phi}_{i,j}^\eta (\phi = u,v,p)$, evaluated at the solid circle point in figure 3.4(a), depends on the transverse-integrated variables evaluated at the six surrounding, triangle points within elements $(i, j)$ and $(i + 1, j)$. Similarly, the unknown $\overline{\phi}_{i,j}^\xi$ ($\phi = u,v,p$), represented by the solid circle point in figure 3.4(b), depends on the transverse-integrated variables at the six triangle points within elements $(i,j)$ and $(i,j + 1)$. However, the unknown $\overline{\phi}_{i,j}^{\xi \eta}$ ($\phi = u,v$), averaged in the two spatial dimensions, depends on the four triangle points at each edge of the single element $(i,j)$ (see figure 3.4(c)). As shown in the figure, complete symmetry of the developed numerical scheme exists in the $\xi$ and $\eta$ directions, which conforms to the MNIM for rectangular elements [35]. This completes the formulation of the numerical scheme for the Navier-Stokes equations.

One of the major differences in the current formulation from that of the MNIM for rectangular elements is the emergence of the significantly larger number of intermediate coefficients (such as $K$’s and $A$’s) with much more complex expressions. The complexity arises naturally from the demand of treating distorted quadrilateral elements. The coefficients $F$’s appearing in the final set of discrete equations in the present work resemble the $F$’s in reference [35]. Comparing the set of discrete equations obtained here and that in reference [35], the corresponding $F$ pairs are easily identified, and listed in table 3.1.

\textsuperscript{†}The definitions of all the $F$’s are listed in appendix D.
Figure 3.4: Computational mesh for the transverse-integrated unknowns.
Table 3.1: Correspondence of $F$ coefficients

<table>
<thead>
<tr>
<th>$F$ coefficients in present work</th>
<th>$F$ coefficients in reference [35]</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_{11}$</td>
<td>$F_{57}$</td>
</tr>
<tr>
<td>$F_{12}$</td>
<td>$F_{51}$</td>
</tr>
<tr>
<td>$F_{13}$</td>
<td>$F_{52}$</td>
</tr>
<tr>
<td>$F_{18}$</td>
<td>$F_{53}$</td>
</tr>
<tr>
<td>$F_{19}$</td>
<td>$F_{54}$</td>
</tr>
<tr>
<td>$F_{31}$</td>
<td>$F_{27}$</td>
</tr>
<tr>
<td>$F_{32}$</td>
<td>$F_{21}$</td>
</tr>
<tr>
<td>$F_{33}$</td>
<td>$F_{22}$</td>
</tr>
<tr>
<td>$F_{34}/F_{35}$</td>
<td>$F_{23}$</td>
</tr>
<tr>
<td>$F_{36}/F_{37}$</td>
<td>$F_{24}$</td>
</tr>
<tr>
<td>$F_{38}$</td>
<td>$F_{25}$</td>
</tr>
<tr>
<td>$F_{39}$</td>
<td>$F_{26}$</td>
</tr>
<tr>
<td>$F_{61}$</td>
<td>$F_{17}$</td>
</tr>
<tr>
<td>$F_{62}$</td>
<td>$F_{11}$</td>
</tr>
<tr>
<td>$F_{63}$</td>
<td>$F_{12}$</td>
</tr>
<tr>
<td>$F_{64}/F_{65}$</td>
<td>$F_{13}$</td>
</tr>
<tr>
<td>$F_{66}/F_{67}$</td>
<td>$F_{14}$</td>
</tr>
<tr>
<td>$F_{68}$</td>
<td>$F_{15}$</td>
</tr>
<tr>
<td>$F_{69}$</td>
<td>$F_{16}$</td>
</tr>
<tr>
<td>$F_{71}$</td>
<td>$F_{77}$</td>
</tr>
<tr>
<td>$F_{72}$</td>
<td>$F_{73}$</td>
</tr>
<tr>
<td>$F_{73}$</td>
<td>$F_{74}$</td>
</tr>
<tr>
<td>$F_{74}$</td>
<td>$F_{75}$</td>
</tr>
</tbody>
</table>
Table 3.1: (continued)

<table>
<thead>
<tr>
<th>$F$ coefficients in present work</th>
<th>$F$ coefficients in reference [35]</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F75$</td>
<td>$F71$</td>
</tr>
<tr>
<td>$F76$</td>
<td>$F72$</td>
</tr>
<tr>
<td>$F41$</td>
<td>$F37$</td>
</tr>
<tr>
<td>$F42$</td>
<td>$F31$</td>
</tr>
<tr>
<td>$F43$</td>
<td>$F32$</td>
</tr>
<tr>
<td>$F48$</td>
<td>$F33$</td>
</tr>
<tr>
<td>$F49$</td>
<td>$F34$</td>
</tr>
<tr>
<td>$F14$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$F15$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$F16$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$F17$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$F44$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$F45$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$F46$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$F47$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

Two points need to be emphasised here: 1) not all the $F$’s in this dissertation have counterparts in reference [35], e.g., $F_{14}$ and $F_{44}$; 2) some $F$’s such as $F_{13}$ and $F_{23}$ in reference [35] have multiple counterparts in the present work, due to the different ways of defining coefficients. Also notice that the $F$ terms that are unique to the present work should in theory be vanishing when the generic quadrilateral elements are restricted to rectangles, and thus the set of discrete equations obtained here should be reduced to the equations obtained in reference [35]. This statement is further
verified in a numerical experiment using rectangular mesh in the next chapter, where the values of $F$’s in both work are calculated and found to be identical.

Similar to the MNIM for rectangular elements, the final set of discrete algebraic equations are solved iteratively using the Gauss-Seidel iterative procedure in conjunction with a SIMPLE-like algorithm [17, 35, 39]. The developed numerical scheme is benchmarked in the next chapter using several 2D, steady-state and time-dependent fluid flow problems with quadrilateral elements distorted to various degrees.

3.9 Summary

Using a simple isoparametric geometry mapping and thus transforming irregular four-node quadrilateral elements into square elements, the modified nodal integral scheme is applied to the transformed Navier-Stokes equations. This approach has often been used in finite volume and finite element schemes. While the transformation of the quadrilateral elements to square elements is straightforward, the transformed set of the Navier-Stokes equations are much more complicated, including significantly large number of additional, linear and nonlinear terms. The appearance of the additional terms can be explained based on physical grounds. No approximations are introduced in the transformation of the element and the governing equations from $(x, y)$ to $(\xi, \eta)$ space. Approximations similar to those made in conventional nodal schemes are then introduced in the development of the numerical scheme. The complexity of the transformed equations leads to more complex expressions for the intermediate coefficients of the new scheme compared to those for MNIM for rectangular elements. The final set of algebraic, discrete equations for generic quadrilateral elements have more terms than those in the conventional MNIM; yet the former should be reduced to the latter in the case of rectangular elements.
4 Modified Nodal Integral Method Incorporated with Quadrilateral Elements for Navier-Stokes Equations—Numerical Results

4.1 Introduction

To verify the numerical scheme developed in the previous chapter and study its performance especially towards quadrilateral elements with different distortions, the following problems are solved in this chapter: Poiseuille flow between parallel plates, modified lid-driven cavity flow problem, tangential annular flow, and the time-dependent periodic flow problem. These problems have been used extensively by other researchers to test numerical schemes solving the incompressible Navier-Stokes equations. Among the four problems, the first three are steady-state problems solved by marching in time, and the last is a time-dependent problem. The new nodal scheme was coded in the C language. The implementations for all test problems were complied by the GCC compiler with the “-O3” optimization option under Linux environment, and run on a PC with an Intel Core 2 Duo CPU running at 2.50 GHz.

4.2 Poiseuille Flow Between Parallel Plates

Poiseuille flow between parallel plates refers to a laminar flow of a viscous fluid moving in a 2D rectangular channel. The flow is driven by a constant pressure gradient in the horizontal direction. No-slip conditions at the top and bottom plates impose viscous drag on the fluid, which balances the pressure gradient. The flow becomes fully developed if the channel is long enough. The steady-state velocity and pressure...
distributions can be analytically derived by setting the time derivatives as well as derivatives with respect to x equal to zero in the Navier-Stokes equations. Assuming the distance between the two parallel plates is $2H$, the analytic solution of the flow velocity is given by

$$u(y) = u_0 \left(1 - \frac{y^2}{H^2}\right)$$  \hspace{1cm} (4.1)$$

and $v = 0$.  

$$u_0 = u(y = 0) = -\frac{H^2}{2\mu} \frac{dp}{dx}.$$  \hspace{1cm} (4.3)$$

The Reynolds number $Re$ for the channel flow is determined by fluid density $\rho$, viscosity $\mu$, half distance between the two plates $H$, and the maximum velocity on the centerline $u_0$:

$$Re = \frac{\rho H}{\mu} u_0.$$  \hspace{1cm} (4.4)$$

Substituting equation (4.3) into (4.4) yields

$$Re = \frac{\rho H}{\mu} \left(-\frac{H^2}{2\mu} \frac{dp}{dx}\right) = -\frac{\rho H^3}{2\mu^2} \frac{dp}{dx}.$$  \hspace{1cm} (4.5)$$

which indicates that $Re$ is proportional to the pressure gradient if the other parameters are fixed. In the numerical experiment, $Re$ is varied by changing the magnitude of the pressure gradient. The other parameters are:

$$H = 0.5, \rho = 1, \mu = 0.005.$$  \hspace{1cm} (4.6)$$
Figure 4.1 shows a schematic diagram of Poiseuille flow between parallel plates. Shown in figure 4.2 are two computational domains, over which the MNIM for generic quadrilateral elements developed in this work is applied to solve the incompressible Navier-Stokes equations.

As the new scheme is based on an algebraic mapping that is not conformal (or to say, angle-preserving), it is desirable to see how pure rotation of the computational elements over an orthogonal grid can effect the accuracy of the MNIM. Hence the first domain chosen is the oblique square $A'B'C'D'$ shown in figure 4.2(a). $A'B'C'D'$ is obtained by rotating the square $ABCD$ 30° in the counter-clockwise direction. Both squares have a uniform side length of 0.4, and are centered at $(x = 1, y = 0)$. The calculations are performed with $\frac{dp}{dx} = -0.2$, leading to $Re = 500$. A reference pressure $p = 1.0$ is imposed at the inlet $x = 0$. The boundary conditions along the external edges are specified according to equations (4.1) and (4.2). The new nodal scheme is employed to calculate the velocity distribution of the channel flow over the oblique square $A'B'C'D'$, which is discretized by uniform square elements. Three different grid sizes, namely, $4 \times 4$, $8 \times 8$, and $16 \times 16$ are used. The $u$ velocity profiles obtained numerically along the line $E'F'$, as shown in figure 4.2(a), are plotted and
Figure 4.2: Computational domains for the Poiseuille flow problem.

compared with the analytic solution for the $4 \times 4$ and $8 \times 8$ mesh in figure 4.3. $E'$ and $F'$ are midpoints of edge $A'B'$ and $C'D'$, respectively. As the points $E'$ and $F'$ are symmetrically located with respect to the point $(1,0)$, the velocity distribution along $E'F'$ should also possess symmetry on both sides of the center line. Complete symmetry of the numerical values along $E'F'$ is observed for all three meshes tested.
Figure 4.3: Comparison of numerical and exact $u$ velocity along $E'F'$ (see figure 4.2(a)) in the rotated square domain $A'B'C'D'$ for the Poiseuille flow problem.
The plots of the numerical results show good agreement with the exact solution, at Re = 500. When the grid size is reduced by one half, the numerical and exact results almost become indistinguishable. Meanwhile, to compare the performance of the new scheme and the original MNIM in terms of accuracy, calculations are also done on the square region $ABCD$ using the old MNIM. The root mean square (RMS) errors of the transverse-integrated $u$ velocity $\overline{u}^{\eta t}$ obtained from both calculations over $A'B'C'D'$ and $ABCD$ are plotted versus the size of the element $\Delta x$ on a log-log scale in figure 4.4. The dashed lines shown in figure 4.4 are lines of best-fit (in the least-squares sense) for the discrete data points. The red line represents the line of best-fit for the RMS errors over domain $ABCD$. It has a slope of 2.03 (with an asymptotic standard error of 0.8604%), which is consistent with the second order nature of MNIM for rectangular elements [35]. The RMS errors over the domain $A'B'C'D'$ calculated using the scheme developed in this dissertation, however, are slightly greater than the RMS errors over $ABCD$, for all three mesh sizes. The slope of the blue best-fit
line for the RMS errors over $A'B'C'D'$ is 1.91 with an asymptotic standard error of 3.208%, indicating a slight drop in the order of accuracy for the new scheme. Although the comparison suggests that the accuracy of the MNIM is impacted when using rotated square (or more generally, rectangular) elements, such impact is not very significant. The results show that when the square elements are rotated by as much as $30^\circ$, the MNIM maintains a near second order accuracy when solving the relatively simple Poiseuille flow problem.

The same fluid flow problem is also solved over the more irregular domain $IJKL$ shown in figure 4.2(b). This domain is similar to that used to test the control volume method [40], the Hybrid Nodal-Integral/Finite-Element Method [41], and the Nodal Integral Method for Quadrilateral Elements [32], all for the convection-diffusion equation. The four vertex points $I, J, K$ and $L$ are assigned global Cartesian coordinates $(x = 0.75, y = 0.25), (x = 0.25, y = -0.25), (x = 0.5, y = -0.5), (x = 1.5, y = -0.5)$, respectively. The edges $IL$ and $JK$ are both at an angle of $45^\circ$ from the $x$-axis, and hence parallel to each other. Edge $IJ$ is perpendicular to $IL$ and $JK$. The generic $n \times n$ grid is created by equally dividing each of the four edges of the trapezoid $IJKL$ using $n - 1$ points and then connecting point pairs on opposite edges. Figure 4.2(b) shows a $4 \times 4$ grid constructed in this way. Grids of different sizes can be similarly constructed. By imposing a reference pressure at the inlet and varying the value of the pressure gradient $\frac{dp}{dx}$, two Reynolds numbers (500 and 1000) are used in the following numerical test. For each Reynolds number, the problem is solved using the new nodal scheme with generic quadrilateral elements on three different mesh sizes: $4 \times 4$, $8 \times 8$ and $16 \times 16$. No-slip boundary condition is applied on the edge $KL$, and Dirichlet boundary conditions along the other three exterior edges are specified using the analytical solution. Figure 4.5 demonstrates the comparison between numerical and exact solutions along the line $MN$ shown in figure 4.2(b), with points $M$ and $N$ bisecting edges $IJ$ and $KL$, respectively. The numerical results agree with the exact
Figure 4.5: Comparison of numerical and exact $u$ velocity along $MN$ (see figure 4.2(b)) in the trapezoidal domain $IJKL$ for the Poiseuille flow problem.
values quite well in case of the coarse, $4 \times 4$ mesh, for both Reynolds numbers (see figure 4.5(a)). Improvement in the accuracy of the numerical results can be observed when the mesh is refined to $8 \times 8$ (see figure 4.5(b)). It is more difficult but still possible to see that the numerical values are less accurate in the higher Reynolds number case (especially for the coarse mesh), which is indicated in the RMS error plots as well. Figure 4.6 shows a log-log plot of the RMS errors in the $u$ velocity over domain $IJKL$. It should be noted that unlike in figure 4.4, the $x$-axis in figure 4.6 represents logarithm of the smallest element size along the shortest edge $JK$, since the mesh over $IJKL$ is nonuniform. RMS errors in the high Reynolds number case are greater than the low Reynolds number case, for all three mesh sizes. The difference of RMS errors at $Re = 1000$ and $Re = 500$, however, generally decreases as the element size decreases. Two dashed lines in different colors representing the lines of best-fit for the two sets of data for different Reynolds numbers are also plotted in figure 4.6. The red line has a slope of 1.86 with an asymptotic standard error of

![Figure 4.6: RMS error of the $u$ velocity in the trapezoidal domain $IJKL$ for the Poiseuille flow problem.](image)

4.6
2.823%, while the blue line has a slope of 2.12 with an asymptotic standard error of 5.660%. Interestingly, the results seem to suggest an increase in the order of the numerical scheme from 1.86 to 2.12, when the Reynolds number increases from 500 to 1000. This is most likely due to fortuitous cancellation of errors. Moreover, for the Re = 1000 case, the asymptotic standard error of the best-fit line, 5.660%, is not small, indicating a relatively strong deviation of the discrete data points from the best-fit line. The RMS error for Re = 1000 case is almost twice as large as that for Re = 500 case, for the 4 × 4 mesh; while the order of magnitude of the RMS error remains the same for both Reynolds numbers, in the case of the other two finer meshes. These indicate that the present nodal scheme deteriorates to some extent for the larger Reynolds number, but only for very coarse meshes when numerical diffusion becomes significant. In the case of small numerical diffusion, the scheme still maintains a near second order accuracy for the test problem.

4.3 Modified Lid-Driven Cavity Problem

Although the robustness of the nodal scheme developed in the present work has been demonstrated in the previous numerical experiment, the Poiseuille flow solved therein is nevertheless a simple flow. For complex flows such as the flow in a lid-driven cavity (see figure 4.7), the distortion of irregular quadrilateral elements from the ideal, square shape may affect the behavior of the new scheme even more. Hence, a variation of the classical lid-driven cavity problem—the modified lid-driven cavity problem—is used to test the present nodal scheme in this section. This problem was first proposed by Shin et al. to study the performance of nine numerical schemes developed to solve the Navier-Stokes equations [42]. It has an exact analytic solution with a prescribed lid velocity, as well as artificial body force terms varying within the cavity. The well-known singularity at the top corners where the lid meets the vertical
Figure 4.7: Schematic diagram and computational mesh for the modified lid-driven cavity problem.

Walls in the classical lid-driven cavity problem is eliminated here, as the prescribed lid velocity has zero values at those two points. Exact solutions for both the velocity and pressure field of the modified lid-driven cavity problem are given by [42]

\[ u(x, y) = 8(x^2 - 2x^3 + x^4)(-2y + 4y^3), \]  
\[ v(x, y) = -8(2x - 6x^2 + 4x^3)(-y^2 + y^4), \]  
\[ p(x, y) = \frac{8}{Re} \left( 24 \left( \frac{x^3}{3} - \frac{x^4}{2} + \frac{x^5}{5} \right) y + (x^2 - 2x^3 + x^4)(-2y + 4y^3) \right) 
+ 64 \left( \frac{x^4}{2} - 2x^5 + 3x^6 - 2x^7 + \frac{x^8}{2} \right) (-2y + 4y^3)^2 + (-2 + 12y^2)(-y^2 + 4y^4), \]  
(4.9)
respectively. The space-dependent body forces are respectively given by

\[
b_y = -\frac{192}{\text{Re}}\left[\left(\frac{x^3}{3} - \frac{x^4}{2} + \frac{x^5}{5}\right) + (2x - 6x^2 + 4x^3)(y^2 - \frac{1}{6}) + (x - \frac{1}{2})(y^4 - y^2)\right] \\
- 64 \left\{ (2x^2 - 8x^3 + 14x^4 - 12x^5 + 4x^6)(4y^3 - 2y)(y^4 - y^2) \\
+ (x^4 - 4x^5 + 6x^6 - 4x^7 + x^8)[12(y^5 - y^3) - (6y^2 - 1)(4y^3 - 2y)] \right\}
\] (4.10)

and \( b_x = 0 \). (4.11)

Note that the lid velocity can be obtained as follows:

\[
u_{\text{lid}}(x) = u(x, y = 1) = 16(x^2 - 2x^3 + x^4).
\] (4.12)

The availability of analytical solutions allows for straightforward error analysis. To take full advantage of this benchmark problem, it was first solved over the entire square domain using both the original MNIM and the present MNIM incorporated with generic quadrilateral elements. The intention is to test whether the coefficients in the final set of discrete equations for these two schemes resemble each other, when the computational domain is discretized by square elements. A uniform 20 × 20 mesh is used for the whole cavity area. No-slip boundary conditions are applied at the other three walls of the cavity, except for the moving lid. The boundary condition along the lid is specified according to the exact solutions. The steady-state problem is solved in a time-dependent manner by starting from the initial condition with zero uniform velocity and pressure fields. Reported in table 4.1 are the numerical values of two families of \( F \) coefficients for element (5, 5), at steady-state. The \( F' \)s are found to be strictly identical in both cases, verifying the theoretical prediction made in chapter 3. Figure 4.8 shows the calculated velocity field for the 20 × 20 square mesh using the two schemes.
Next, the new implementation of the MNIM is solved and tested over a parallelogram within the unit cavity. The parallelogram $ABCD$, shown in figure 4.7, is centered in the cavity, with a uniform edge length of 0.5. It can deform continuously as the angle $\theta$ measured between the edge $AD$ and the horizontal line is continuously changed. Simulations are carried out for three $\theta$ values—$\frac{5\pi}{12}, \frac{\pi}{3}, \frac{\pi}{4}$. The uniform $n \times n$ mesh used in the test is constructed by dividing each edge equally. Figure 4.7 shows such an $8 \times 8$ mesh. Dirichlet boundary conditions are imposed upon the four external edges of the parallelogram $ABCD$ using the exact solutions given by the equations (4.7)–(4.9). Again, the time-independent, modified lid-driven cavity problem is solved by marching in time from a uniform zero initial condition for all variables till steady state is achieved. Numerical results reported below are for Reynolds number $Re = 1$. Note that the velocity field in this manufactured problem is not affected by the Reynolds number. The numerical results showed very good agreement with exact solution for all eight unknowns in this problem. As an example, figure 4.9 shows a global, qualitative comparison between exact and numerical solutions for $\bar{u}^{\xi t}$ and $\bar{p}^{\eta t}$ ($8 \times 8$ mesh, $\theta = \frac{\pi}{4}$). The $x$- and the $y$-axis in the figure indicate the edge number along the boundaries of the parallelogram $ABCD$. Figure 4.10 shows the numerical and exact values of the edge-averaged velocities and pressure ($\bar{u}^{\eta t}, \bar{v}^{\eta t}$ and $\bar{p}^{\eta t}$) along the line $EF$ (as shown in figure 4.7) on a $4 \times 4$ mesh. Points $E$ and $F$ bisect edges $AB$ and $CD$ respectively. It can be observed that numerical results agree with exact solution very well even for a mesh as coarse as $4 \times 4$. Figure 4.11 shows the numerical and exact transverse-integrated $u$ velocities $\bar{u}^{\eta t}$ and $\bar{v}^{\xi t}$ along the two bisectors $EF$ and $GH$ (see figure 4.7) respectively for three $\theta$ values ($8 \times 8$ mesh, $\theta = \frac{5\pi}{12}, \frac{\pi}{3}, \frac{\pi}{4}$). The numerical profiles of the velocity field over the $ABCD$ domain are plotted in figure 4.12 for $\theta = \frac{\pi}{3}$ and $\frac{\pi}{4}$. RMS errors in $\bar{u}^{\xi t}, \bar{v}^{\eta t}, \bar{p}^{\xi t}$ are reported for different mesh sizes as well as $\theta$ values in table 4.2. It should be pointed out that the exact transverse-integrated values in the error calculations in this dissertation are
evaluated using line integrals (unlike the point values used in reference [43]). The “Factor” column in table 4.2 represents the reduction factor of the RMS errors as the mesh resolution is increased, for each $\theta$ value. Mesh refinement study shows that RMS errors increase for all variables as skewness of the elements increases, but the absolute deterioration is not very significant. The scheme has a second order (or sometimes even higher) accuracy when element shape is close to square/rectangular. It deteriorates somewhat as the angle $\theta$ is changed from $\frac{5\pi}{12}$ to $\frac{\pi}{4}$, indicated by the damping of reduction factors to approximately $\sim 3.5$. Furthermore, the scheme seems to be slightly less accurate in $v$ velocity. This is probably due to artificial diffusion induced by complex flow directions. Previous researchers have found that numerical diffusion usually intensifies transport in the direction normal to the flow [44]. The flow over the $ABCD$ domain has a stronger trend of moving horizontally than vertically (see figure 4.12), which may result in larger errors in $v$ velocity. However, it is known that most numerical schemes applicable to nonorthogonal meshes will lose some accuracy if the elements used are badly distorted, and almost all schemes suffer more or less from the inevitable numerical diffusion. The modified lid-driven cavity problem solved here is a relatively difficult problem involving complex flows. Actually the results and error analysis clearly show that the new nodal scheme can predict flows beyond simple flow using severely distorted elements, without considerable loss of accuracy.

Table 4.1: Numerical values of corresponding $F$ coefficients for element (5,5) in the square cavity domain. Elements are numbered from lower left to top right.

<table>
<thead>
<tr>
<th>$F$’s in present work</th>
<th>$F$’s in reference [35]</th>
<th>Numerical values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_{11}$</td>
<td>$F_{57}$</td>
<td>-1.6001e2</td>
</tr>
<tr>
<td>$F_{12}$</td>
<td>$F_{51}$</td>
<td>3.9951e1</td>
</tr>
<tr>
<td>$F_{13}$</td>
<td>$F_{52}$</td>
<td>4.0064e1</td>
</tr>
</tbody>
</table>
Table 4.1: (continued)

<table>
<thead>
<tr>
<th>$F$'s in present work</th>
<th>$F$'s in reference [35]</th>
<th>Numerical values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F18$</td>
<td>$F53$</td>
<td>-5.9951e1</td>
</tr>
<tr>
<td>$F19$</td>
<td>$F54$</td>
<td>-6.0064e1</td>
</tr>
<tr>
<td>$F31$</td>
<td>$F27$</td>
<td>-1.0000e2</td>
</tr>
<tr>
<td>$F32$</td>
<td>$F21$</td>
<td>1.0000e1</td>
</tr>
<tr>
<td>$F33$</td>
<td>$F22$</td>
<td>1.0000e1</td>
</tr>
<tr>
<td>$F34/F35$</td>
<td>$F23$</td>
<td>-3.0000e1</td>
</tr>
<tr>
<td>$F36/F37$</td>
<td>$F24$</td>
<td>-3.0000e1</td>
</tr>
<tr>
<td>$F38$</td>
<td>$F25$</td>
<td>-0.1250e-1</td>
</tr>
<tr>
<td>$F39$</td>
<td>$F26$</td>
<td>-0.1250e-1</td>
</tr>
<tr>
<td>$F61$</td>
<td>$F17$</td>
<td>-1.0000e2</td>
</tr>
<tr>
<td>$F62$</td>
<td>$F11$</td>
<td>1.0000e1</td>
</tr>
<tr>
<td>$F63$</td>
<td>$F12$</td>
<td>1.0000e1</td>
</tr>
<tr>
<td>$F64/F65$</td>
<td>$F13$</td>
<td>-3.0000e1</td>
</tr>
<tr>
<td>$F66/F67$</td>
<td>$F14$</td>
<td>-3.0000e1</td>
</tr>
<tr>
<td>$F68$</td>
<td>$F15$</td>
<td>-0.1250e-1</td>
</tr>
<tr>
<td>$F69$</td>
<td>$F16$</td>
<td>-0.1250e-1</td>
</tr>
<tr>
<td>$F71$</td>
<td>$F77$</td>
<td>-4.9000e3</td>
</tr>
<tr>
<td>$F72$</td>
<td>$F73$</td>
<td>4.7000e3</td>
</tr>
<tr>
<td>$F73$</td>
<td>$F74$</td>
<td>-2.4019e3</td>
</tr>
<tr>
<td>$F74$</td>
<td>$F75$</td>
<td>-2.3981e3</td>
</tr>
<tr>
<td>$F75$</td>
<td>$F71$</td>
<td>-2.3985e3</td>
</tr>
<tr>
<td>$F76$</td>
<td>$F72$</td>
<td>-2.4015e3</td>
</tr>
<tr>
<td>$F41$</td>
<td>$F37$</td>
<td>-6.4007e1</td>
</tr>
<tr>
<td>$F42$</td>
<td>$F31$</td>
<td>1.5949e1</td>
</tr>
</tbody>
</table>
Table 4.1: (continued)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( F^{32} )</td>
<td>( F^{33} )</td>
<td>Numerical values</td>
</tr>
<tr>
<td>( F^{43} )</td>
<td>1.6058e1</td>
<td></td>
</tr>
<tr>
<td>( F^{48} )</td>
<td>-2.3949e1</td>
<td></td>
</tr>
<tr>
<td>( F^{49} )</td>
<td>-2.4058e1</td>
<td></td>
</tr>
<tr>
<td>( F^{14} )</td>
<td>( \emptyset )</td>
<td>0.0</td>
</tr>
<tr>
<td>( F^{15} )</td>
<td>( \emptyset )</td>
<td>0.0</td>
</tr>
<tr>
<td>( F^{16} )</td>
<td>( \emptyset )</td>
<td>0.0</td>
</tr>
<tr>
<td>( F^{17} )</td>
<td>( \emptyset )</td>
<td>0.0</td>
</tr>
<tr>
<td>( F^{44} )</td>
<td>( \emptyset )</td>
<td>0.0</td>
</tr>
<tr>
<td>( F^{45} )</td>
<td>( \emptyset )</td>
<td>0.0</td>
</tr>
<tr>
<td>( F^{46} )</td>
<td>( \emptyset )</td>
<td>0.0</td>
</tr>
<tr>
<td>( F^{47} )</td>
<td>( \emptyset )</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Table 4.2: RMS errors for the modified lid-driven cavity problem.

<table>
<thead>
<tr>
<th>Mesh</th>
<th>( \Pi^{\epsilon \eta} )</th>
<th>Factor</th>
<th>( \Pi^{\epsilon \eta} )</th>
<th>Factor</th>
<th>( P^{\xi \ell} )</th>
<th>Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \theta = \frac{5\pi}{12} )</td>
<td>4 \times 4</td>
<td>1.623 \times 10^{-3}</td>
<td>7.913 \times 10^{-3}</td>
<td>1.048 \times 10^{-2}</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8 \times 8</td>
<td>4.017 \times 10^{-4}</td>
<td>2.055 \times 10^{-3}</td>
<td>3.85</td>
<td>2.562 \times 10^{-3}</td>
<td>4.09</td>
</tr>
<tr>
<td></td>
<td>16 \times 16</td>
<td>9.657 \times 10^{-5}</td>
<td>5.257 \times 10^{-4}</td>
<td>3.91</td>
<td>6.358 \times 10^{-4}</td>
<td>4.03</td>
</tr>
<tr>
<td>( \theta = \frac{\pi}{3} )</td>
<td>4 \times 4</td>
<td>1.920 \times 10^{-3}</td>
<td>1.042 \times 10^{-2}</td>
<td>1.091 \times 10^{-2}</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8 \times 8</td>
<td>5.079 \times 10^{-4}</td>
<td>2.903 \times 10^{-3}</td>
<td>3.59</td>
<td>2.933 \times 10^{-3}</td>
<td>3.72</td>
</tr>
<tr>
<td></td>
<td>16 \times 16</td>
<td>1.369 \times 10^{-4}</td>
<td>8.018 \times 10^{-4}</td>
<td>3.62</td>
<td>8.035 \times 10^{-4}</td>
<td>3.65</td>
</tr>
<tr>
<td>( \theta = \frac{\pi}{4} )</td>
<td>4 \times 4</td>
<td>2.182 \times 10^{-3}</td>
<td>1.407 \times 10^{-2}</td>
<td>1.159 \times 10^{-2}</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8 \times 8</td>
<td>6.181 \times 10^{-4}</td>
<td>4.175 \times 10^{-3}</td>
<td>3.37</td>
<td>3.321 \times 10^{-3}</td>
<td>3.49</td>
</tr>
<tr>
<td></td>
<td>16 \times 16</td>
<td>1.731 \times 10^{-4}</td>
<td>1.220 \times 10^{-3}</td>
<td>3.42</td>
<td>9.328 \times 10^{-4}</td>
<td>3.56</td>
</tr>
</tbody>
</table>

61
Figure 4.8: Numerical results for the velocity vectors of the modified lid-driven cavity problem on a $20 \times 20$ uniform square mesh for $Re = 1$. 

(a) Vector length proportional to velocity magnitude

(b) Uniform vector length
Figure 4.9: Comparison of exact and numerical solutions for $u^{\xi\eta}$ and $p^{\eta t}$ over the parallelogram $ABCD$ for the modified lid-driven cavity problem ($8 \times 8$ mesh, $\theta = \frac{\pi}{4}$).
Figure 4.10: Comparison of exact and numerical solutions along line $EF$ (see figure 4.7) for the modified lid-driven cavity problem.
Figure 4.11: Comparison of exact and numerical solutions of the edge-averaged $u$ velocities along lines $EF$ and $GH$ (see figure 4.7) for the modified lid-driven cavity problem.
Figure 4.12: Numerical results for the velocity vectors of the modified lid-driven cavity problem over the $ABCD$ domain for $8 \times 8$ mesh cases.
4.4 Tangential Annular Flow

The modified lid-driven cavity problem was solved in the previous section over a parallelogram sub-domain that necessitates skewed elements for efficient coarse discretization permitted by the nodal scheme. For problems over a circular or annular area, however, it will be more efficient to approximate the computational domain using tapered elements. The tangential annular flow provides an ideal test problem to study the present nodal scheme’s compatibility with tapered elements—an important subclass of quadrilaterals.

![Figure 4.13: Schematic diagram and computational mesh for the tangential annular flow problem.](image)

The tangential annular flow in this section refers to an incompressible fluid flow between two coaxial cylinders that are infinite in length. As shown in figure 4.13, the rotating outer cylinder has a constant angular velocity of $\Omega_0$, and thus drags the enclosed fluid to move in the $\theta$ direction. The inner cylinder is kept stationary. The ratio of outer and inner radii is chosen as 2 in the numerical experiment. The
tangential annular flow not only is commonly utilized in instruments measuring the viscosity of Newtonian flows [45], but also has analytic solutions that are handy in testing new numerical schemes. The fully-developed velocity and pressure fields of this flow are described by the following equations (in cylindrical coordinates):

\[
v_\theta(r) = \frac{4}{3} \Omega_0 R \left( \frac{r}{R} - \frac{1}{4} \frac{R}{r} \right),
\]

(4.13)

\[
p(r) = \frac{8}{9} R^2 \rho \Omega_0^2 \ln(r) + \frac{1}{18 r^2} \left( 32 p_0 r^2 - 16 p_0 r^2 + 2 p_0 r^2 + 16 r^4 \rho \Omega_0^2 \right)
\]

\[-16 r^2 R^2 \rho \Omega_0^2 + r^2 R^2 \rho \Omega_0^2 - R^4 \rho \Omega_0^2 - 16 r^2 R^2 \rho \Omega_0^2 \ln(r) \right),
\]

(4.14)

in which \( R \) is the radius of the outer cylinder, and \( p_0 \) is the reference pressure on the outer cylinder. The problem can be reformulated for numerical simulations using the non-dimensional variables:

\[
\tilde{v}_\theta(r) = \frac{v_\theta(r)}{\Omega_0 R},
\]

(4.15)

\[
\tilde{p}(r) = \frac{p(r)}{\rho(\Omega_0 R)^2}.
\]

(4.16)

In the above equations, the non-dimensional variables \( \tilde{v}_\theta(r) \) and \( \tilde{p}(r) \) can respectively reproduce the original variables \( v_\theta(r) \) and \( p(r) \), if the parameters \( \Omega_0, R, \) and \( \rho \) are chosen to be unity. The Reynolds number then becomes equal to the inverse of the viscosity \( \nu \).

The numerical simulation is carried out over the quarter annulus domain shown in figure 4.13. A typical computational mesh used in the simulation is constructed by equally dividing the angle, \( \frac{\pi}{2} \), and the width of the annulus, \( \frac{R}{2} \), into \( m \) and \( n \) pieces, respectively. In this way, the quarter annulus domain can be approximated by \( m \times n \) tapered elements. Two mesh sizes—15 × 10 and 30 × 20—are used in the numerical experiment. The steady-state solution is reached by marching in time (with
a step size of $\Delta t = 0.01$ for the fine mesh and $\Delta t = 0.02$ for the coarse mesh), using the present implementation of the time-dependent MNIM for quadrilateral elements. The initial condition is set uniformly for all variables as $u = v = p = 0.1$. No-slip boundary conditions are applied on the boundaries corresponding to the inner and outer cylinders. A reference pressure $p_0 = 1$ is imposed on the outer cylinder boundary. Dirichlet conditions for both velocity and pressure are enforced on the vertical and horizontal boundaries of the quarter annulus. The viscosity $\nu$ used in the simulation is 0.02, leading to a Reynolds number of $Re = 50$.

Shown in figures 4.14 and 4.15 are the velocity results for the coarse and fine mesh cases. Each figure has two sub-plots obtained using the same data set—one sub-plot with vector length proportional to the velocity magnitude, and the other with uniform vector length in order to reveal the flow direction more clearly at locations where velocity approaches zero. According to the analytic prediction of the velocity field (see equation (4.13)), the flow should consist of laminar concentric layers in between the two cylinders. These layers can be easily identified in the velocity vector plots. In figure 4.14(b), slight yet recognizable deviations of the plotted vector directions from the concentric layers’ tangent directions can be observed†, especially at the inner-most layer where the velocity is very close to zero (so the numerical error is highly magnified under the uniform vector length plot). However, the velocity vectors in figure 4.15(b) are generally more consistent with the tangent vectors of the concentric layers, implying a higher accuracy obtained under the finer mesh. Figure 4.16 presents the percentage error of the element-averaged velocity for those elements located along the center line of the quarter annulus domain in the $\theta$ direction (see the red line in figure 4.13). The $x$-axis in the figure indicates the element index starting from the vertical boundary. The results quantitatively show the improvement in

---

†Readers are recommended to refer to the electronic version of this dissertation, so that the figures can be arbitrarily enlarged for scrutiny.
Figure 4.14: Flow map of the tangential annular flow problem for $15 \times 10$ mesh.
Figure 4.15: Flow map of the tangential annular flow problem for $30 \times 20$ mesh.
Figure 4.16: Percentage error of element-averaged velocity for elements along the center line (see figure 4.13) in the tangential annular flow problem.
accuracy when the computational mesh is refined. Even for the $15 \times 10$ mesh, the percentage error never exceeds 5% for both $u$ and $v$ velocities. In the finer mesh case, the percentage error is further reduced to less than 1% at most locations. Reported in table 4.3 are the RMS errors and CPU time for the tangential annular flow problem. The “Factor” column in the table has the same definition as that in table 4.2. For all variables reported, the reduction factors are slightly less than 4. Once again, the near second order accuracy can be observed in this test problem, confirming that the tapered elements can be very effectively used in the scheme developed here.

Table 4.3: RMS errors and CPU time for the tangential annular flow problem.

<table>
<thead>
<tr>
<th>Mesh</th>
<th>$\bar{u}^{\xi \eta}$</th>
<th>Factor</th>
<th>$\bar{v}^{\xi \eta}$</th>
<th>Factor</th>
<th>$\bar{p}^{\xi t}$</th>
<th>Factor</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$15 \times 10$</td>
<td>$3.204 \times 10^{-2}$</td>
<td>2.187 $\times 10^{-2}$</td>
<td>4.450 $\times 10^{-2}$</td>
<td>28.2s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$30 \times 20$</td>
<td>$8.322 \times 10^{-3}$</td>
<td>3.85</td>
<td>$5.637 \times 10^{-3}$</td>
<td>3.88</td>
<td>$1.174 \times 10^{-2}$</td>
<td>3.79</td>
<td>407.7s</td>
</tr>
</tbody>
</table>

4.5 Time-Dependent Periodic Flow

The superior performance of the MNIM incorporated with quadrilateral elements developed in this dissertation has been partially demonstrated in the previous numerical examples. All of the three problems solved, nevertheless, are steady-state problems that can hardly take full advantage of the new nodal method as a time-dependent scheme. Therefore, to further test the new implementation of the MNIM, it is employed to solve a time-dependent incompressible fluid flow problem (referred to as the time-dependent periodic flow problem) with the following exact solution:

$$u(t, x, y) = \cos(t) \cos^2(\pi x/2) \sin(\pi y),$$

(4.17)
\[ v(t, x, y) = -\cos(t) \sin(\pi x) \cos^2(\pi y/2), \quad (4.18) \]

\[ p(t, x, y) = \cos(t) \cos(\pi x/2) \sin(\pi y/2). \quad (4.19) \]

This problem has been used by other researchers to perform temporal accuracy checks for collocative spectral method [46]. The velocity and pressure fields represent a large eddy centered at the origin, rotating periodically and alternately in opposite directions as time varies. Figure 4.17 shows the distribution of the \( u \) and \( v \) velocities as well as pressure at the initial time \( t = 0 \) over the square \([-1, 1] \times [-1, 1]\). The computational domain chosen in the current study is \([-1, 1] \times [-1, 1] \setminus \{x^2 + y^2 \leq 0.4^2\}\), over which the generic mesh\(^1\) used for the problem is shown in figure 4.18, and the velocity vector plot at time \( t = 2k\pi \) \((k = 0, 1, 2, \ldots)\) is shown in figure 4.19.

The problem was solved with initial conditions specified by equations (4.17)–(4.19). Dirichlet conditions for all variables are enforced on all mesh boundaries. Numerical results for the velocity vectors at \( t = \frac{3}{4}\pi \) are plotted in figures 4.20 and 4.21, corresponding to a \( 160 \times 15 \) mesh and a \( 180 \times 20 \) mesh, respectively. The time step \( \Delta t = \frac{\pi}{1000} \) for the coarser mesh case and \( \Delta t = \frac{\pi}{2000} \) for the finer mesh case. Apparent improvement in accuracy of the results can be recognized by comparing figures 4.20 and 4.21, illustrated particularly by the velocity vectors at locations near the external boundary\(^2\) in the uniform vector length plots. The plot in figure 4.22(a) shows the results for \( \bar{u}^\eta \) along the vertical center line \( x = 0 \) after one time period, \( t = 2\pi \). The results for \( \bar{v}^\eta \) along the horizontal center line \( y = 0 \) for the same time is shown in figure 4.22(b). Though numerical diffusion can be seen in the reduction of the peak value in both \( u \) and \( v \) velocities, the peak value does not drop significantly.

In the \( 160 \times 50 \) mesh case, the peak value reduces from 1 to 0.9678 for the \( u \) velocity

\(^1\)Constant mesh spacing is applied in discretizing the \( \theta \) and \( r \) directions. For an \( m \times n \) mesh in this section, \( m \) represents the mesh size in the \( \theta \) direction while \( n \) in the \( r \) direction.

\(^2\)The velocity is zero everywhere on the outer boundary in this problem.
Figure 4.17: Velocity and pressure fields for the time-dependent periodic flow problem at $t = 0$. Dashed lines show the contour map for each field.
Figure 4.18: Schematic diagram of the computational mesh for the time-dependent periodic flow problem.

Figure 4.19: Schematic velocity vector plot of the time-dependent periodic flow problem at $t = 2k\pi$ ($k = 0, 1, 2 \cdots$).
Figure 4.20: Flow map of the time-dependent periodic flow problem for $160 \times 15$ mesh at $t = \frac{3}{4} \pi$. 

(a) Vector length proportional to velocity magnitude

(b) Uniform vector length
Figure 4.21: Flow map of the time-dependent periodic flow problem for 180 × 20 mesh at $t = \frac{3}{4}\pi$.
Figure 4.22: Comparison of numerical and exact velocity results of the time-dependent periodic flow problem at $t = 2\pi$. 

(a) $u$ velocity along the line $x = 0$ at $t = 2\pi$

(b) $v$ velocity along the line $y = 0$ at $t = 2\pi$
and to 0.9644 for the $v$ velocity, at time $t = 2\pi$. In the finer $180 \times 20$ mesh case, the peak value reduces to 0.9837 for the $u$ velocity and to 0.9862 for the $v$ velocity. The numerical diffusion decreases with increasing number of elements, implying that mesh refinement can effectively reduces the numerical diffusion of this scheme. The RMS error of the element-averaged velocities $\overline{u}^{\xi\eta}$ and $\overline{v}^{\xi\eta}$ for the $160 \times 15$ mesh at $t = 2\pi$ is $2.41 \times 10^{-2}$ and $1.86 \times 10^{-2}$, respectively. The RMS error is reduced to $6.65 \times 10^{-3}$ for $\overline{u}^{\xi\eta}$ and $5.27 \times 10^{-3}$ for $\overline{v}^{\xi\eta}$, over a refined mesh of $320 \times 30$, for the same period of time. Again, the new nodal scheme is shown to have a near second order accuracy.

It should be further noted that the original MNIM developed for rectangular elements was shown to have “inherent upwinding” [17,35]. For flow in a convection-dominated domain, the flow at a certain location should be prevalingly affected by the velocity on the upstream side. Hence for a numerical scheme for the Navier-Stokes equations, to have an upwinding feature means that the velocity variables in the discrete equations should possess coefficients that are automatically weighted by the scheme in favor of the upstream direction. The “inherent-upwinding” feature of the original MNIM was demonstrated in [35] using the Taylor’s Decaying Vortices problem. To demonstrate that the new nodal scheme developed here has the same feature mentioned-above, the coefficients in the discrete equation for the $u$ velocity $\pi^{u\eta}$ were evaluated at a location $A$ in the computational domain of the present study, as shown in figure 4.19. Note that the flow in this problem is always in the horizontal direction at $A$†, making it an ideal location for this investigation. Also shown in figure 4.19 is the schematic of two neighbouring elements sharing the interface on which the discrete variable $\pi^{u\eta}_{i,j}$ resides. Algebraic equation (3.115), developed in the previous

†The flow direction may be in the positive or negative $x$ direction as time changes.
chapter to evaluate $\overline{u}^{nt}_{i,j}$, is reproduced below for easy reference:

$$
F_{11} \overline{u}^{nt}_{i,j} = F_{12} \overline{u}^{nt}_{i-1,j} + F_{13} \overline{u}^{nt}_{i+1,j} + F_{14} \overline{u}^{et}_{i,j} + F_{15} \overline{u}^{et}_{i,j-1} + F_{16} \overline{u}^{et}_{i+1,j} \\
+ F_{17} \overline{u}^{et}_{i,j+1} + F_{18} (\overline{u}^{η}_{i,j} + \overline{u}^{η}_{i,j,k-1}) + F_{19} (\overline{u}^{η}_{i+1,j} + \overline{u}^{η}_{i+1,j,k-1}).
$$

(4.20)

For convenience sake, the above equation is re-written in the following form:

$$
\overline{u}^{nt}_{i,j} = l_1 \overline{u}^{nt}_{i-1,j} + l_2 \overline{u}^{nt}_{i+1,j} + l_3 \overline{u}^{et}_{i,j} + l_4 \overline{u}^{et}_{i+1,j} + l_5 \overline{u}^{et}_{i,j-1} \\
+ l_6 \overline{u}^{et}_{i+1,j-1} + l_7 (\overline{u}^{η}_{i,j} + \overline{u}^{η}_{i,j,k-1}) + l_8 (\overline{u}^{η}_{i+1,j} + \overline{u}^{η}_{i+1,j,k-1}),
$$

(4.21)

where $l_1 = \frac{F_{12}}{F_{11}}$, $l_2 = \frac{F_{13}}{F_{11}}$, $l_3 = \frac{F_{14}}{F_{11}}$, $l_4 = \frac{F_{15}}{F_{11}}$, $l_5 = \frac{F_{16}}{F_{11}}$, $l_6 = \frac{F_{17}}{F_{11}}$, $l_7 = \frac{F_{18}}{F_{11}}$, $l_8 = \frac{F_{19}}{F_{11}}$. These new $l$ coefficients, varying with location and time, demonstrate how the unknown on the LHS is affected in the scheme by the neighbouring variables from both upstream and downstream sides. Table 4.4 reports the numerical value of $l$ coefficients at location $A$ corresponding to the simulation over the $180 \times 20$ mesh at time $t = 2\pi$. It is clearly seen that the value of $l_1$—the coefficient of the upstream variable $\overline{u}^{nt}_{i-1,j}$—is greater than that of $l_2$—the coefficient of the downstream variable $\overline{u}^{nt}_{i+1,j}$—by four orders of magnitude, indicating a significant weighting of the coefficients in favor of the upstream direction. Similar observation is made in the comparison of the other upstream-downstream coefficient pairs $l_3$ and $l_4$, $l_5$ and $l_6$, $l_7$ and $l_8$, respectively. It deserves emphasis that the weighting adjustment of the coefficient is not restricted only for the unknown variable $\overline{u}^{nt}$ to be evaluated using equation (4.21). The coefficients of the element-averaged variable $\overline{u}^{η}$, as well as the variable $\overline{u}^{et}$ (transverse-integrated in the other direction), are also adjusted by the scheme according to the flow direction. To illustrate the dynamic nature of the built-in upwinding, the same $l$ coefficients are reported in table 4.5 at $t = 3\pi$, when the flow at location $A$ is in the (horizontal) direction opposite to the direction at $t = 2\pi$. Comparing the coefficients in table 4.4 and 4.5, it is clear that as the flow
changes direction, the coefficients evolve to retain the upwinding characteristic. All these observations have illustrated the upwinding feature of the new nodal scheme developed in this work.

Table 4.4: $l$ coefficients in equation (4.21) at $t = 2\pi$.

<table>
<thead>
<tr>
<th>$l$ coefficients</th>
<th>Value at location A</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_1$</td>
<td>146.590</td>
</tr>
<tr>
<td>$l_2$</td>
<td>0.076</td>
</tr>
<tr>
<td>$l_3$</td>
<td>24.582</td>
</tr>
<tr>
<td>$l_4$</td>
<td>0.429</td>
</tr>
<tr>
<td>$l_5$</td>
<td>19.201</td>
</tr>
<tr>
<td>$l_6$</td>
<td>0.653</td>
</tr>
<tr>
<td>$l_7$</td>
<td>$-183.917$</td>
</tr>
<tr>
<td>$l_8$</td>
<td>$-2.887$</td>
</tr>
</tbody>
</table>

Table 4.5: $l$ coefficients in equation (4.21) at $t = 3\pi$.

<table>
<thead>
<tr>
<th>$l$ coefficients</th>
<th>Value at location A</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_1$</td>
<td>0.071</td>
</tr>
<tr>
<td>$l_2$</td>
<td>148.054</td>
</tr>
<tr>
<td>$l_3$</td>
<td>0.437</td>
</tr>
<tr>
<td>$l_4$</td>
<td>24.910</td>
</tr>
<tr>
<td>$l_5$</td>
<td>0.646</td>
</tr>
<tr>
<td>$l_6$</td>
<td>20.182</td>
</tr>
<tr>
<td>$l_7$</td>
<td>$-2.853$</td>
</tr>
<tr>
<td>$l_8$</td>
<td>$-186.401$</td>
</tr>
</tbody>
</table>

4.6 Summary

In this chapter, the Modified Nodal Integral Method incorporated with generic quadrilateral elements developed in the dissertation work has been tested by solving a series of incompressible flow problems. A classic problem to start with, the Poiseuille flow
problem was solved over two domains: a rotated square and a trapezoid, for different Reynolds numbers. Then the modified lid-driven cavity problem was solved to verify that the new nodal scheme is capable of recovering the set of discrete equations obtained in the original MNIM when treating square/rectangular elements. The same problem was next solved over a parallelogram constituted by skewed quadrilaterals under various degrees of distortion. The third test problem—the tangential annular flow problem—was solved over a quarter annulus approximated by the union of tapered elements. Last, the time-dependent periodic flow problem was solved in a square with a circular inner boundary, mainly to investigate the numerical dissipation of the scheme in a time-dependent problem. Results obtained for all these problems are in very good agreement with analytical results. The present implementation of the MNIM can work very well with generic quadrilateral elements of different shapes and degrees of distortion, without significant loss of accuracy. Actually the new scheme maintains a near second order accuracy in most test cases. Moreover, the time-dependent periodic flow problem was used to check the characteristic upwinding feature of the new nodal scheme, and the inherent upwinding in the scheme is found to be evident.
5 An Overview of GPU-Based Parallel Computing

5.1 Introduction to GPGPU

Faster speed and higher accuracy are, and will always be pursued by computational scientists and engineers. To achieve these goals, micro-processors were assembled across a network to arrive at “parallel computing”. This computing technology, due to its superb performance over the old sequential computing, has been serving the science and engineering community for decades. The commonly used parallel computing models include but are not restricted to: 1) shared memory model; 2) message passing model, e.g., MPI [47]; 3) threads model, e.g., POSIX Threads [48] and OpenMP [49]; 4) data parallel model; and 5) hybrid model. Although these models can in theory be implemented on different types of machines as well as memory architectures irrespective of the underlying hardware [50], they were mostly implemented on clusters running multiple, single-core Central Processing Units (CPUs). The concept of parallel computing, however, has evolved over the past few years from the traditional parallelism obtained at processor level to the chip level, with the emergence of multi- and many-core architectures. Roughly speaking, multi-core and many-core processing technique means adding more cores† onto a single microprocessor chip. The idea is both innovative and straightforward, as single core processors have been rapidly approaching the physical limits of potential complexity and speed, and thus become less efficient for modern compute intensive challenges. Shown in figure 5.1(a) is an Intel

†A “core” represents an independent actual processor—the basic unit that reads and executes machine instructions.
Core Duo CPU, one of the mainstream multi-core CPUs on the market. This CPU has a central processor integrating two independent cores, enabling multi-threading capability and on-chip parallelism. There also have been quad-, hexa-, octo- and 12-core CPUs available in the market, and CPU prototypes with even more cores in research labs. The practice of adding cores onto the CPU chip seems to be promising and effortless. However, such practice becomes increasingly difficult to realize when the core number exceeds a certain amount, as a result of the limitation from the original design in CPU architecture.

![Intel Core Duo CPU](image)

(a) Intel Core Duo CPU

![NVIDIA Tesla GPU](image)

(b) NVIDIA Tesla GPU

Figure 5.1: Current mainstream (a) multi-core CPU and (b) many-core GPU.

It should be noted that CPU is not the only type of processor capable of performing computing tasks on a computer, nor is it the processor with the best computing capability. Surprisingly, the Graphics Processing Unit (GPU)—a co-processor to the CPU—has in recent years acquired the strongest computing power. GPUs were originally designed for rendering tasks which are essentially about parallel computation and usually involve massive amount of real-time calculation. The design of GPU
architecture makes it easy to add large number of independent actual processors onto a GPU chip. Figure 5.1(b) shows the flagship NVIDIA Tesla GPU with 240 processor cores. Because a modern GPU generally carries substantially more cores than a CPU, the terminology “many-core”† has been reserved to characterize GPUs, so as to differentiate from the “multi-core” term used for CPUs. Figure 5.2 shows the floating-point computational capability of the two kinds of processors. The peak single precision performance of NVIDIA GPUs was just slightly higher than that of Intel CPUs in the year of 2004. However, by the end of the year 2009, NVIDIA GPUs already had speeds that are nearly an order of magnitude higher in single precision calculations. For the double precision case, GPUs also have delivered increasingly better performance than CPUs since 2008, and the trend is expected to continue in the coming years.

†Many-core roughly refers to several tens of cores, or above.
Powerful in floating-point computation, GPUs had been used exclusively in computer graphics. The idea of making use of GPUs for high-performance computing, however, was put into practice as early as 1990’s. Researchers at that time were encouraged by the impressive speedups obtained on GPUs for some of the early non-graphic applications. As a result, the so-called general-purpose computing on GPUs, or GPGPU, has emerged [52]. Though highly innovative, the GPGPU underwent a slow development for a rather long time. The use of GPUs in general-purpose computations was limited due to the fixed hardware specifically designed for the rendering purpose, and due to the difficulties in programming using graphics-oriented languages such as OpenGL. Also, the computational scientists and engineers had to become very familiar with the underlying hardware before being able to exploit the full capacity of GPUs. The situation has changed dramatically with the introduction of new GPU architectures by NVIDIA and ATI/AMD that are fully programmable. The release of GPU programming models and tools such as NVIDIA’s Compute Unified Device Architecture (CUDA) further simplified the development of GPU-based computational applications. CUDA is a programming platform that programmers can use to realize algorithms on GPUs. CUDA’s inherent powerful control over the modern GPU’s many-core architecture, makes it a strong candidate for GPGPU tasks. Although several other GPU programming environments (e.g., Brook, Sh, and CTM) are also available, CUDA is the most popular as it manages to maintain a less-steep learning curve. Since 2007, many research codes from various disciplines have been rewritten in CUDA and consequently yielded substantial performance benefit. Literature search shows that GPGPU has been reported to accelerate computational applications in the following fields: methods for the numerical solution of partial differential equations [53], computational fluid dynamics [54–60], computational electro-magnetics [61], computational plasma physics [62, 63], molecular dynamics simulations [64–67], Monte Carlo simulations [68], fast evaluation of

87
special functions in theoretical physics [69,70], atmospheric sciences [71,72], seismic wave studies [73], etc. Attracted by the impressive results reported in these applications, the GPGPU community is under rapid expansion, and scientific computing is moving fast toward the many-core era.

Modeling and simulation in nuclear science and engineering relies heavily on computational power. Simulations of neutronics has benefited greatly from parallel computing [74,75]. Other fields in nuclear engineering, e.g., reactor thermal hydraulics, nuclear materials, plasma physics, also took advantage of parallelization to advance the simulations. The field is expected to benefit substantially from the new many-core GPU technology. Some of the pioneering applications of GPU-computing to nuclear engineering problems have been reported. Kodama et al. realized a CUDA implementation of neutron diffusion code using the response matrix formulation to solve the multi-group diffusion equation [76]. They used the IAEA LWR benchmark problem to test the new implementation on GPU and obtained speedups of $\sim 40\times$ over a CPU implementation for two mesh sizes on a 20 cm $\times$ 20 cm fuel assembly. Huang et al. developed accelerated program to solve the Burgers equation in 2009, using single precision calculations [77]. Huang and Rizwan-uddin reported the first, double precision, nodal scheme based GPU implementation that solves the 3D incompressible Navier-Stokes equations in 2011 [78]. Heimlich et al. presented Monte Carlo based neutron shielding simulations in slab geometry and 2D heat equation solution using central difference solver under the CUDA environment [79]. The reported speedup results were obtained by comparing the GPU implementations with the corresponding multi-threaded CPU implementations. Gong et al. used an NVIDIA GTX280 processor to accelerate 3D simulations of deterministic particle transport using the discrete ordinates ($S_n$) method [80]. Kirschenmann et al. developed a CUDA program that can conduct 3D neutronics simulations based on the simplified $P_n$ equations [81]. Due to the limited scope of literature search, many other GPGPU applications in nu-
clear engineering may remain uncovered here. It is believed that the trend to take advantage of the “on desk” super-computing power realized by GPGPU will continue to bring benefit to this field.

5.2 GPU Architecture

![Comparison of CPU and GPU architectures](image)

Figure 5.3: Comparison of CPU and GPU architectures [51].

As emphasized in the previous section, the original design of GPU was aimed at highly specialized graphics rendering tasks. Hence the architecture of a GPU differs significantly from that of a CPU, which is designed as a general purpose processor. A comparison of CPU and GPU architectures is demonstrated in figure 5.3. Generally speaking, GPUs devote more transistors to data processing than caching and controlling, while CPUs need to balance the number of transistors responsible for different functions. The basic unit to conduct arithmetic operations on a microprocessor is called an arithmetic logic unit (ALU). As can be seen in figure 5.3, there are significantly more ALU’s† (yet less control transistors and cache) in GPU than CPU. This explains the superior performance for GPUs when loaded with intensive arithmetic

†ALU’s are indicated by the green blocks in the figure.
computations. It should also be noted that the disadvantage for GPUs to have less control and cache abilities hardly impacts their performance if the computation tasks are highly parallel and do not have sophisticated flow control.

Figure 5.4: NVIDIA Tesla C1060 GPU architecture.

Although an abstract programming model that makes the GPU hardware somewhat transparent† to the programmers has been provided by CUDA (which will be described in detail in the next section), it still requires a good understanding of the structure and components of a modern GPU to exploit the full capability of GPU-computing. As an example, figure 5.4 shows the architecture of Tesla C1060, one of the latest GPUs from NVIDIA that are primarily used not as a graphics processor, but as a compute co-processor. The basic processing unit of NVIDIA Tesla

†By transparent, it is to say that programmers can be more focused on algorithms than on how to manipulate the GPU hardware.
GPU is the streaming processor (SP), a fully pipelined, single-issue, in-order microprocessor complete with two arithmetic logic units and a floating-point unit (FPU). A group of 8 SPs, 2 additional special function units (SFUs)—processor cores that have floating-point multiply units used for transcendental operations—and 16 KB of shared memory form a streaming multiprocessor (SM). A group of 3 SMs with some additional memory form texture/processor cluster (TPC). Ten such clusters form the streaming processor array. In total, Tesla C1060 GPU has 240 SPs running at 1.3 GHz, thus delivering nearly 1 TeraFLOP in single-precision floating-point arithmetic and 78 GFLOPS in double precision. Additionally, its 512-bit interface to the off-chip GDDR3 memory (4 GB) provides 102 GB/s bandwidth, which is an order of magnitude greater than the memory bandwidth provided by a mainstream CPU†.

5.3 The CUDA Programming Model for GPGPU

GPGPU relies not only on the development of the GPU architecture, but also on the evolution of the software environment under which the graphics processors are programmed. CUDA is a such a programming platform that has released the GPGPU researchers from having to learn the details of the graphics libraries. More specifically, CUDA provides an abstract programming model to describe and control the GPU hardware for scientific computation tasks. Programmers can therefore access and manipulate, for example, the transistors and arithmetic units, on a GPU through the components provided by this programming model, for example, the many-core “threads”.

CUDA exposes a single GPU as a number of multiprocessors consisting of a group of SIMD (Single Instruction, Multiple Data) processors. The union of these SIMD processors (cores) are abstracted as “grid” in the CUDA model. The grid can be

†The memory bandwidth supported on CPUs is more restricted by operating systems, whereas such restriction to GPUs is minimized [82].
decomposed into substructures called “blocks” arranged in 1D or 2D arrays. A block can further be decomposed into 1D, 2D, or 3D arrays of threads, which are the basic executing units in a CUDA code. As shown in figure 5.5(a), each block is uniquely numbered and identified by the block index which is similar to the array index in a C program. Each thread must belong to a block and is uniquely numbered and identified grid-wise by a linear combination of the thread index and the block index. At runtime, the SIMD processors schedule and load a certain number† of threads during each core clock period. These threads then read and execute the same instructions in the “kernel”—the counterpart of a C subroutine for CUDA—simultaneously. For example, the Tesla C1060 GPU has 240 SIMD streaming processors in total. Each of the streaming processor is taking care of just one thread block at any moment. It loads and processes a “warp” of 32 threads from that block in parallel, at the speed of core frequency. Hence, to reach the full parallelism capacity of a GPU, programmers are advised to specify: the grid size (namely, the total number of blocks) to be greater than the number of streaming processors available on the GPU; and the block size (namely, the number of threads in each block) to be multiples of (or less than) the warp size.

It should be noted that in the “grid-block-thread” hierarchy of CUDA, synchronization among threads is achievable only within a thread block. The most secure way of synchronizing all threads is to terminate and invoke the kernel again. However, threads belonging to different blocks are still able to communicate during the life span of the kernel by using the global memory on GPU. The CUDA model has provided a memory hierarchy (shown in figure 5.5(b)) corresponding to that of threads. Global memory can be visited by all multiprocessors on the GPU while shared memory is available for all SIMD processors within a multiprocessor. This means the global

†This number partially determines the capacity of parallelism and is specified by the GPU hardware.
Figure 5.5: Modeling of thread and memory in CUDA [79].
memory is reachable for any thread from any block in the grid, whereas the shared memory can only be used by threads grouped in the same block. These two memories also differ in size and the speed of access. Global memory has a huge capacity, but is slow to visit as it is not cached. Shared memory is cached, hence may become very helpful in accelerating the kernel when threads within a block reuse a set of data frequently. One obvious limitation of shared memory is its size; normally 16KB for each block on the present generation of GPUs. Moreover, numerical algorithms implemented using shared memory are usually less straightforward than using global memory. The constant and texture memory are also available grid-wise on a GPU. Both of them have faster access than the global and shared memory. Yet they are read-only memory that can not be utilized for updating the variables. Except for the several types of memory mentioned-above, threads are associated with a few registers to store local variables with the shortest access time.

The CUDA programming model comes with a C-like programming language often referred to as “CUDA C” [83]. In a CUDA program, the most important part handling the parallel execution of GPU threads is called a kernel, which has been repeatedly mentioned in the above paragraphs. To illustrate the difference of a kernel and a usual subroutine, listings 5.1 and 5.2 compare a CUDA code and C code carrying out the same task of matrix addition. The kernel “matrix_add” defined in line 9 in listing 5.1 is lead by the “__global__” declaration specifier—a unique indicator for kernels. The built-in variables threadIdx.x and threadIdx.y in lines 11-12 are the indices of the threadID assigned by the system to each thread. The matrix addition is completed thread-wise in parallel in GPU-computing. Therefore the higher dimensional matrix addition, which has to resort to a couple of for-loops in the serial C code (see listing 5.2), is reduced to an algebraic addition simply expressed by line 13 in the CUDA code, indicating that every thread is taking care of just one addition operation. Unlike a C subroutine, a CUDA kernel is called by the main function using a <<<...>>>
syntax, which contains the dimension information of the grid and blocks. In this example, the grid and block dimensions are respectively set in lines 4 and 5 of listing 5.1.

![Memory Transfer Diagram](image.png)

Figure 5.6: Schematic diagram for memory transfer between host and device.

The execution of a CUDA program starts from the host side where CPU executes the serial part of the code. Once the kernel is launched, CPU invokes GPU on the device side and parallel threads are loaded to run the instructions from the kernel. It deserves emphasis that as CPU and GPU own separate memory spaces, it is obliged to transfer the required data in the computation task from host to device before the GPU is invoked. The same process will have to be repeated once again in the opposite direction, after the computation is finished on the device side and control is returned to CPU. Figure 5.6 shows the schematic of memory transfer between the host and device. This memory transfer is very expensive in terms of execution time. Some amount of information transfer between device and host is inevitable in order to acquire the real-time status of GPU during a long simulation. The transfer should be otherwise avoided to the possible extent.
Listing 5.1: CUDA code of matrix addition.

```c
int main()
{
    ...
    dim3 gridSize(1);
    dim3 blockSize(N, N);
    matrix_add<<<gridSize, blockSize>>>(A, B, C);
    ...
}

__global__ void matrix_add(float **A, float **B, float **C)
{
    int i = threadIdx.x;
    int j = threadIdx.y;
    C[i][j] = A[i][j] + B[i][j];
}
```

Listing 5.2: C code of matrix addition.

```c
int main()
{
    ...
    matrix_add(A, B, C);
    ...
}

void matrix_add(float **A, float **B, float **C)
{
    int i, j;
    for (i=0; i<N; i++)
    {
        for (j=0; j<N; j++)
        {
            C[i][j] = A[i][j] + B[i][j];
        }
    }
}
```
5.4 Case Studies: CUDA Implementation of the 2-D Burgers Solver

Burgers equation is a non-linear model of the convection-diffusion process similar to the Navier-Stokes equations, but without the pressure term. It has been used to test new numerical schemes [84] and new computer architectures prior to their applications to, for example, the Navier-Stokes equations. The two-dimensional Burgers equation is given by

\[
\frac{Dw}{Dt} = \nu \nabla^2 w, \quad (5.1)
\]

where \( w = (u, v)^T \) and \( u \) and \( v \) denote velocity components in \( x \) and \( y \) directions respectively. The GPU-accelerated Burgers solver is implemented on NVIDIA graphics processors. Explicit Euler and second order central difference schemes are applied to discretize time and space, respectively. To parallelize, an individual thread created on the GPU grid is assigned to each finite difference grid point in the computational domain. GPU threads work simultaneously to update the unknown variables at each time step. Hence the update process, namely, calling the kernel function, is executed with the time evolution without sweeping the space domain at each single step, which is the most computationally demanding part in a serial implementation.

Numerical solution is obtained over the unit square \( 0 \leq x \leq 1, 0 \leq y \leq 1 \) with homogeneous initial conditions and the following boundary conditions [85]: \( u(x, 0) = u(x, 1) = v(1, y) = 0, \ v(0, y) = 1, \ u(0, y) = u(1, y) = \sin(2\pi y), \ v(x, 0) = v(x, 1) = 1 - x \). Steady state solution can be achieved by marching in time. To make the problem more computationally demanding, a relatively small time step (\( \Delta t = 10^{-5} \text{ s} \)) is chosen. The time step affects both CPU and GPU based simulations the same way. Hence, its only effect is to increase the compute time for both; allowing better ratio.
comparison. The viscosity $\nu$ is 0.015 in the simulation. The GPU implementation of the numerical scheme is coded in CUDA. Figure 5.7 shows the flow chart of the CUDA Burgers solver. The Burgers kernel is given in appendix E. A CPU serial implementation for the same problem is coded in C for comparison purposes. The CUDA code was compiled using the NVCC compiler, while the C code using the GCC compiler with the “-O3” optimization option. The C code runs under the single-threaded mode† on a PC with an Intel Core 2 Duo CPU running at 2.50 GHz. The CUDA code is accelerated by NVIDIA Quadro FX5600 and Tesla C1060 GPUs in different tests. The reason to perform GPU simulation on different graphics processors is to illustrate how GPGPU can be affected and benefit from the rapid development of GPU hardware. The two graphics processors used typically represent the current and the previous generation of GPU computing technology. Table 5.1 lists and compares the major specifications of Quadro FX5600 and Tesla C1060. Except for the remarkable advancement from Quadro to Tesla in almost all aspects that are usually adopted to evaluate GPU hardware, the latter also has the double precision floating-point ability that the former does not possess at all. Additionally, the Tesla GPU supports the upgraded platform CUDA 1.3 whereas the Quadro only supports the original CUDA 1.0.

Numerical results for $u$ and $v$ velocities at different times are shown in figure 5.8. Table 5.2 shows performance comparison of the single precision implementation on CPU and GPUs. The “speedup” columns are defined as the ratio of CPU time to GPU time. A speedup of $\sim 2 \times$ to $\sim 7 \times$ is obtained on the Quadro FX5600 GPU for meshes from $22 \times 22$ to $64 \times 64$. On the Tesla C1060 GPU, the speedup is from $\sim 5 \times$ to $\sim 18 \times$ for the same mesh sizes. It is observed that speedup increases with mesh resolution on both GPUs. As is well known from parallel computing exercises,†

†This means only one of the two cores available on the Intel Core 2 Duo CPU is used in the simulation.
this trend is expected to continue for even larger size problems that involve more threads and thus can better exploit the parallel computation abilities of GPUs. The speedup obtained on the newer Tesla GPU, for any mesh size tested, is higher than the speedup obtained on Quadro GPU, with a ratio between 2 to 3. This is consistent with the peak single precision performance of the two GPUs listed in table 5.1. Table 5.3 reports the double precision performance of the C code running on Intel CPU and the CUDA code running on Tesla GPU. Comparing the CPU time reported in tables 5.2 and 5.3, it is found that the CPU C code using double precision runs slightly slower than its single precision version for the first three mesh sizes, but a bit faster for the last mesh size case. Theoretically, CPU code should have the same performance toward single and double precision calculations as long as the memory bandwidth is not the bottleneck for a specific problem. The reason is that modern 32-bit CPUs are equipped with 80-bit ALUs, which take care of all arithmetic calculations. Hence in each CPU clock cycle, an ALU reads and operates a 32-bit single precision number or a 64-bit double precision number with no difference in operation speed. In the test, the single and double precision CPU code have slightly different execution time, which is reasonable because the execution time varies with the status of the multi-task operating system as well as the status of CPU. It is also observed that the double precision CUDA code runs significantly slower than its single precision counterpart on Tesla C1060. This is due to the limitation of GPU hardware. On the Tesla GPU, there is only one floating-point unit that can deal with double precision calculations on each multiprocessor and is shared by all eight streaming processors, whereas each streaming processor has its own floating-point unit that deals with single precision calculations. However, the double precision CUDA code still obtains an impressive speedup from $\sim 3 \times$ to $\sim 10 \times$ for the meshes tested in this problem.

Though the performance of the CUDA implementation is sacrificed to some ex-
tent in the double precision case, the improvement in precision greatly compensates the loss of acceleration. Figures 5.9 and 5.10 show the difference between GPU simulation results obtained using double and single precision. It can be seen that due to the symmetry in the boundary condition and in the solution of this problem, the distribution of the difference in double and single precision results exhibits fairly symmetrically. It is also found that the double-single precision difference for the $u$ velocity varies between $-0.002$ to $0.002$ over the whole domain, whereas such difference for the $v$ velocity reaches as high as $0.0075$ at some locations. The above-discussed difference, though may be of less importance for the tested Burgers equation, may affect the accurate prediction of the more complex Navier-Stokes equations in some cases. To minimize such impact, the double precision calculation is used in the CUDA implementation of the Modified Nodal Integral Method for the 3D, incompressible Navier-Stokes equations on GPUs, which is detailed in the next chapter.
Table 5.1: Characteristics of the Tesla C1060 and Quadro FX5600 GPUs.

<table>
<thead>
<tr>
<th></th>
<th>Tesla C1060</th>
<th>Quadro FX5600</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of streaming processors</td>
<td>240</td>
<td>128</td>
</tr>
<tr>
<td>Peak core clock</td>
<td>1.3 GHz</td>
<td>1.35 GHz</td>
</tr>
<tr>
<td>Single precision performance</td>
<td>933 GFLOPs</td>
<td>346 GFLOPs</td>
</tr>
<tr>
<td>Double precision performance</td>
<td>78 GFLOPs</td>
<td>N/A</td>
</tr>
<tr>
<td>Total memory</td>
<td>4 GB GDDR3</td>
<td>1.5 GB GDDR3</td>
</tr>
<tr>
<td>Memory clock</td>
<td>800 MHz</td>
<td>400 MHz</td>
</tr>
<tr>
<td>Memory interface</td>
<td>512-bit</td>
<td>384-bit</td>
</tr>
<tr>
<td>Memory bandwidth</td>
<td>102 GB/s</td>
<td>76.8 GB/s</td>
</tr>
<tr>
<td>Peak energy consumption</td>
<td>225 W</td>
<td>171 W</td>
</tr>
<tr>
<td>Support platform</td>
<td>NVIDIA CUDA 1.3</td>
<td>NVIDIA CUDA 1.0</td>
</tr>
<tr>
<td>Market price</td>
<td>$~1,400</td>
<td>$~950</td>
</tr>
</tbody>
</table>

Table 5.2: Single precision performance of GPU and CPU implementations for the Burgers solver.

<table>
<thead>
<tr>
<th>Mesh</th>
<th>CPU time (s)</th>
<th>Quadro FX5600 time (s)</th>
<th>Tesla C1060 time (s)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>22 x 22</td>
<td>30.91</td>
<td>14.49</td>
<td>6.68</td>
<td>4.63</td>
</tr>
<tr>
<td>32 x 32</td>
<td>61.67</td>
<td>18.94</td>
<td>7.99</td>
<td>7.72</td>
</tr>
<tr>
<td>46 x 46</td>
<td>147.88</td>
<td>28.82</td>
<td>10.15</td>
<td>14.57</td>
</tr>
<tr>
<td>64 x 64</td>
<td>251.03</td>
<td>34.67</td>
<td>13.84</td>
<td>18.14</td>
</tr>
</tbody>
</table>

Table 5.3: Double precision performance of GPU and CPU implementations for the Burgers solver.

<table>
<thead>
<tr>
<th>Mesh</th>
<th>CPU time (s)</th>
<th>GPU time (s)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>22 x 22</td>
<td>31.37</td>
<td>10.85</td>
<td>2.89</td>
</tr>
<tr>
<td>32 x 32</td>
<td>63.29</td>
<td>13.30</td>
<td>4.76</td>
</tr>
<tr>
<td>46 x 46</td>
<td>150.03</td>
<td>18.72</td>
<td>8.01</td>
</tr>
<tr>
<td>64 x 64</td>
<td>248.94</td>
<td>25.64</td>
<td>9.71</td>
</tr>
</tbody>
</table>
Initialization on host (CPU):
- Dynamic memory allocation on host
- Set initial and boundary conditions
- Calculate constant coefficients

Initialization on device (GPU):
- Set up device execution configuration
- Dynamic memory allocation on device

Transfer data to device memory

Invoke GPU solver

time marching loops

Call Burgers kernel

Transfer data back to host memory

Write output files

Figure 5.7: Flow chart of the CUDA Burgers solver.
Figure 5.8: Snapshots showing $u$ and $v$ velocity results of the Burgers equation.
Figure 5.9: Difference of steady-state $u$ velocity results of the Burgers equation calculated using double and single precision on a $32 \times 32$ mesh.
Figure 5.10: Difference of steady-state $v$ velocity results of the Burgers equation calculated using double and single precision on a $32 \times 32$ mesh.
Modified Nodal Integral Method for 3D Navier-Stokes Equations on a Graphics Processing Unit

6.1 Introduction

The nodal methods have become the backbone and workhorse of the production codes used in the nuclear industry for decades. Although highly efficient as a subclass of coarse mesh schemes, the potential of the nodal methods can be further exploited from the computing hardware aspect. A variation of the nodal schemes—the Modified Nodal Integral Method (MNIM)—has been developed for 3D time-dependent incompressible Navier-Stokes equations [36]. To take advantage of the superior acceleration performance obtained using GPU-computing technique, a new implementation of the MNIM on a graphics processing unit is developed in this chapter. The chapter begins with a review of the MNIM for 3D Navier-Stokes equations. The development of the new GPU implementation for the MNIM is discussed next. Numerical results are consequently presented in section 6.4. Chapter is finally concluded with a summary.

6.2 MNIM for the 3D, Time-Dependent, Incompressible Navier-Stokes Equations

Presented in this part is a brief description of the Modified Nodal Integral Method as applied to 3D, time-dependent, incompressible, isothermal Navier-Stokes equations. Reference [36] is referred to for a comprehensive description of the derivation of the scheme.
The 3D incompressible Navier-Stokes equations are given by

\[ \nabla \cdot \mathbf{v} = 0, \]  
\[ \frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} = \nu \nabla^2 \mathbf{v} - \frac{1}{\rho} \nabla p + \mathbf{g}, \]  

where \( \mathbf{v} = (u, v, w)^T \) and \( u, v, w \) denote velocity components in the \( x, y, \) and \( z \) directions respectively. In the formulation of the MNIM, the continuity equation (6.1) is replaced by an equivalent Poisson-type equation for pressure, given as:

\[ \nabla^2 p = -\rho \nabla \cdot (\mathbf{v} \cdot \nabla \mathbf{v} - \nu \nabla^2 \mathbf{v} - \mathbf{g}). \]  

To numerically solve equations (6.2) and (6.3), the global computational domain \( (X, Y, Z, T) \) is discretized in rectangular space-time elements \((i, j, k, n)\) of size \((2a_i \times 2b_j \times 2c_k \times 2\tau_n)\) with element-centered local coordinates. The derivation of the MNIM for the momentum equation (6.2) is shown here only for the \( x \)-direction, where the N-S equation is written as

\[
\frac{\partial u}{\partial t} + u_p \frac{\partial u}{\partial x} + v_p \frac{\partial u}{\partial y} + w_p \frac{\partial u}{\partial z} - \nu \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \right) = -\frac{1}{\rho} \frac{\partial p}{\partial x} + g_x(x, y, z, t) - (u - u_p) \frac{\partial u}{\partial x} - (v - v_p) \frac{\partial u}{\partial y} - (w - w_p) \frac{\partial u}{\partial z}. \]  

Note that \( u_p, v_p \), and \( w_p \) in equation (6.4) are the element-averaged \( u, v, \) and \( w \) velocities at the previous time step, respectively, introduced as the “delayed coefficients” to treat the non-linear convection terms.

The next step is the Transverse Integration Procedure (TIP), in which the PDE is integrated with respect to all independent variables except one. The TIP is repeated for all space-time directions and thus yields a set of corresponding ODE’s—three ODE’s in space variables and one ODE in time. For instance, operating equation
(6.4) by
\[
\frac{1}{8a_ib_j\tau_n} \int_{-\tau_n}^{\tau_n} \int_{-b_j}^{b_j} \int_{-c_k}^{c_k} dy \, dz \, dt
\]
yields
\[
\frac{d\bar{u}^{yzt}(x)}{dx} - \nu \frac{d^2\bar{u}^{yzt}(x)}{dx^2} = \bar{S}^{yzt}(x), \tag{6.5}
\]
where \(\bar{u}^{yzt}(x)\) is by definition
\[
\bar{u}^{yzt}(x) \equiv \frac{1}{8a_ib_j\tau_n} \int_{-\tau_n}^{\tau_n} \int_{-b_j}^{b_j} \int_{-c_k}^{c_k} u(x, y, z, t) \, dy \, dz \, dt, \tag{6.6}
\]
and the pseudo-source term on the RHS is formally defined as
\[
\bar{S}^{yzt}(x) = - \frac{1}{8a_ib_j\tau_n} \int_{-\tau_n}^{\tau_n} \int_{-b_j}^{b_j} \int_{-c_k}^{c_k} \left[ \frac{\partial u}{\partial t} + (u - u_p) \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} + w \frac{\partial u}{\partial z} 
\right.
\]
\[
\left. - \nu \left( \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \right) + \frac{1}{\rho} \frac{\partial p}{\partial x} - g_x(x, y, z, t) \right] \, dy \, dz \, dt. \tag{6.7}
\]
Similarly, the TIP is applied to momentum equations in the \(y\)- and \(z\)-directions as well as to the pressure Poisson equation (6.3), yielding fifteen ODE’s in total. The resulting ODE’s become decoupled and hence solvable after the expansion of the pseudo-source terms in Legendre polynomials and truncating at zeroth order. The decoupled ODE’s are solved analytically within each element for the corresponding transverse-integrated variables. Continuity of surface-averaged unknowns and their derivatives at interfaces of neighbouring elements are imposed to obtain a set of discrete algebraic equations in terms of truncated pseudo-source terms. The final set of discrete algebraic equations for the transverse-integrated unknowns are then obtained after eliminating the pseudo-source terms by applying the constraint equations similar to those presented in section 3.7, but in 3D.
6.3 GPU Implementation of the MNIM for the 3D, Incompressible Navier-Stokes Equations

Unlike operations such as matrix addition that are fully parallelizable, not all of the steps in a numerical scheme for solving the Navier-Stokes equations can be easily formulated as data-parallel algorithms. The MNIM is no exception. Hence it is significantly more demanding to adapt the MNIM to the massively parallel architecture of GPUs than to implement the scheme on a CPU. Described in detail below are the main steps of the GPU implementation of the MNIM for the 3D N-S equations, including the parallel decomposition strategies, the floating-point precision considerations, the design of a parallel pressure Poisson solver, and the design of a new iteration strategy on GPUs.

6.3.1 Parallel Decomposition Strategies

Decomposing the problem into a large number of independent threads is considered to be one of the key aspects to maximize parallelism on the GPU. Hence parallel decomposition strategies need to be carefully designed to make efficient use of the hundreds of streaming processors available on modern graphics processors. In the present GPU-implementation of the MNIM, the computational domain is decomposed into brick-like elements in such a way as to make each element have a one-to-one mapping to a thread on the GPU grid (see figure 6.1). Therefore, the number of threads in any application in this work will be equal to the number of computational elements. Also, the computational grid is topologically equivalent to the GPU grid, allowing easy and straightforward design in parallel algorithms. As mentioned in the previous chapter, thread blocks can be arranged in 1D or 2D arrays in CUDA while threads within a single block can be arranged in 1D, 2D or 3D arrays. This CUDA feature gives the programmers the flexibility to manage pieces of data in various
ways according to the need. In the present work, the threads are arranged in 3D arrays in each block, and the blocks in 2D arrays in the whole grid. A set of simple relationships given below is needed to evaluate the global Cartesian index \((i, j, k)\) of a certain thread using its associated \(threadID\) and \(blockID\):

\[
i = blockIdx.x \times blockDim.x + threadIdx.x,
\]

\[
j = blockIdx.z,
\]

\[
k = blockIdx.y \times blockDim.y + threadIdx.y,
\]

where \(i, j,\) and \(k\) denote the global index in the \(x, y,\) and \(z\) directions, respectively; the variables \(blockIdx,\) \(blockDim\) and \(threadIdx\) are built-in variables respectively representing the identifier of the block, the dimension of the block, and the identifier of the thread.
6.3.2 Floating Point Precision Considerations

Since early GPU hardware only supported single precision floating point calculations that often can not satisfy the requirements of scientific computing, the floating point precision has been one of the major concerns for any GPGPU application. The numerical difference in results obtained using single and double precision calculations on GPUs has been particularly demonstrated in the Burgers solver presented in the previous chapter. Except for affecting the accuracy of the simulation results, the aforementioned difference, however, can also cause fatal numerical errors in certain cases. For example, consider that the following arithmetic operation sequence is carried out on a GPU [60]:

\[
A = 1E2 + 1E-5; \\
B = A \ast 10; \\
C = B - 1E3;
\]

If variable A in this operation sequence is originally defined as a double precision number, then variable C will have the (expected) correct result of .0001. But if A is defined as a single precision number in the beginning, variable C would correspondingly become 0, which can further lead to fatal errors if C appears in the denominator in the following steps. The reason is that a double precision number contains 16 significant figures whereas a single precision number contains only 7. Therefore variable A will be truncated to 1E2 in the single precision case, consequently leading to the incorrect result for C. The above example also ideally explains why the MNIM performs much better with double precision floating-point than with single precision. The MNIM maintains high accuracy on coarse meshes due to the significantly larger number of unknowns per element as well as the associated coefficients in the final set of discrete equations than those in the traditional fine mesh schemes. The coefficients,
particularly, need to be updated at the beginning of every time step, and sometimes can be very small. Thus the coefficients’ truncation error due to the use of single precision floating-point may result in unreliable values for the numerical solution, which may be very sensitive to the values of some of those coefficients\footnote{Recall the exponential terms appearing in the analytical solution of the ODE’s obtained after the TIP in the development of the MNIM. These exponential terms help to accurately capture steep gradients within a coarse mesh element, but also result in the scheme’s sensitive dependence on the coefficients.}. Fortunately the state-of-the-art GPUs now have support for double precision capability that is becoming comparable to the single precision capability in terms of computing performance, making GPGPU even more powerful. Hence double precision calculations are adopted in the present GPU implementation of the MNIM for the Navier-Stokes equations.

6.3.3 Pressure Poisson Solver

In the development of the MNIM, a pressure Poisson equation is solved in place of the continuity equation. This treatment is commonly used in numerical schemes for the Navier-Stokes equations. However, due to the poor convergence rate, solution of

![Red-Black ordering of Cartesian grid points.](image)
the Poisson-type pressure equation usually represents the most time-consuming part of the code\(^\dagger\) and thus requires an efficient solver. Essentially, solving the discrete pressure Poisson equations means finding solution to a linear system of equations that can be solved iteratively. The Gauss-Seidel iterator was used in the CPU implementation of the MNIM [35]. To suit the multi-threaded nature of computation on GPUs, a parallel variation of the Gauss-Seidel scheme—the Red-Black Gauss-Seidel—is used in the present GPU implementation, where the computational grid is colored with red and black nodes in the alternative order as shown in figure 6.2. Instead of sequentially updating the variables from bottom-left to top-right by rows in the conventional Gauss-Seidel, the update process in the Red-Black Gauss-Seidel is alternatively conducted in parallel for all the nodes of the same color once at a time. Specifically in the present implementation, the following pressure update sequence is executed first for all the red nodes, and then all the black nodes, within a single iteration:

\[
\begin{align*}
\bar{p}^{\text{rgt}}_{i,j,k}^{(n+1)} &= \sum_{ae} (a_{1ae} \bar{p}^{\text{rgt}}_{ae}^{(n)} + b_{1ae} \bar{p}^{\text{zxt}}_{ae}^{(n)} + c_{1ae} \bar{p}^{\text{zxt}}_{ae}^{(n)} + d_{1ae} f_{1ae}) , \\
\bar{p}^{\text{yzt}}_{i,j,k}^{(n+1)} &= \sum_{ae} (a_{2ae} \bar{p}^{\text{rgt}}_{ae}^{(n+1)} + b_{2ae} \bar{p}^{\text{yzt}}_{ae}^{(n)} + c_{2ae} \bar{p}^{\text{yzt}}_{ae}^{(n)} + d_{2ae} f_{1ae}) , \\
\bar{p}^{\text{zxt}}_{i,j,k}^{(n+1)} &= \sum_{ae} (a_{3ae} \bar{p}^{\text{rgt}}_{ae}^{(n+1)} + b_{3ae} \bar{p}^{\text{yzt}}_{ae}^{(n+1)} + c_{3ae} \bar{p}^{\text{zxt}}_{ae}^{(n)} + d_{3ae} f_{1ae}) .
\end{align*}
\]

Note that equations (6.8)-(6.10) are reproduced using a compact form from equations (4.91)-(4.93) in reference [35]. The subscript “ae” in the above equations refers to the indices of all adjacent elements required by the stencil to evaluate the unknown at \((i,j,k)\), while the superscript \((n)\) refers to the iteration number. Furthermore, due to the mutual dependence relationship of the transverse-integrated variables for

\(^{\dagger}\text{The claim will become more clear in the performance analysis part of section 6.4 in this chapter.}\)
pressure in the MNIM, parallelism is not achieved among the three pressure unknowns \( \overline{p}^{xyt}, \overline{p}^{yzt} \) and \( \overline{p}^{zxt} \). These three unknowns are updated one by one, indicated by the sequential order of equations (6.8)–(6.10). The algorithm for the Red-Black Gauss-Seidel scheme to solve the pressure Poisson equation is shown in Algorithm 1.

<table>
<thead>
<tr>
<th>1</th>
<th>while ( n &lt; ) maximum iteration number do</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>if element ((i, j, k)) is colored red then</td>
</tr>
<tr>
<td>3</td>
<td>( \overline{p}^{xyt}<em>{i,j,k}(n+1) = \sum</em>{ae \in \text{black}} \left( a_{1ae} \overline{p}^{xyt}<em>{ae}(n) + b</em>{1ae} \overline{p}^{yzt}<em>{ae}(n) + c</em>{1ae} \overline{p}^{zxt}<em>{ae}(n) + d</em>{1ae} f_{1ae} \right) )</td>
</tr>
<tr>
<td>4</td>
<td>( \overline{p}^{yzt}<em>{i,j,k}(n+1) = \sum</em>{ae \in \text{black}} \left( a_{2ae} \overline{p}^{xyt}<em>{ae}(n+1) + b</em>{2ae} \overline{p}^{yzt}<em>{ae}(n) + c</em>{2ae} \overline{p}^{zxt}<em>{ae}(n) + d</em>{2ae} f_{1ae} \right) )</td>
</tr>
<tr>
<td>5</td>
<td>( \overline{p}^{zxt}<em>{i,j,k}(n+1) = \sum</em>{ae \in \text{black}} \left( a_{3ae} \overline{p}^{xyt}<em>{ae}(n+1) + b</em>{3ae} \overline{p}^{yzt}<em>{ae}(n+1) + c</em>{3ae} \overline{p}^{zxt}<em>{ae}(n) + d</em>{3ae} f_{1ae} \right) )</td>
</tr>
<tr>
<td>6</td>
<td>end</td>
</tr>
<tr>
<td>7</td>
<td>if element ((i, j, k)) is colored black then</td>
</tr>
<tr>
<td>8</td>
<td>( \overline{p}^{xyt}<em>{i,j,k}(n+1) = \sum</em>{ae \in \text{red}} \left( a_{1ae} \overline{p}^{xyt}<em>{ae}(n) + b</em>{1ae} \overline{p}^{yzt}<em>{ae}(n) + c</em>{1ae} \overline{p}^{zxt}<em>{ae}(n) + d</em>{1ae} f_{1ae} \right) )</td>
</tr>
<tr>
<td>9</td>
<td>( \overline{p}^{yzt}<em>{i,j,k}(n+1) = \sum</em>{ae \in \text{red}} \left( a_{2ae} \overline{p}^{xyt}<em>{ae}(n+1) + b</em>{2ae} \overline{p}^{yzt}<em>{ae}(n) + c</em>{2ae} \overline{p}^{zxt}<em>{ae}(n) + d</em>{2ae} f_{1ae} \right) )</td>
</tr>
<tr>
<td>10</td>
<td>( \overline{p}^{zxt}<em>{i,j,k}(n+1) = \sum</em>{ae \in \text{red}} \left( a_{3ae} \overline{p}^{xyt}<em>{ae}(n+1) + b</em>{3ae} \overline{p}^{yzt}<em>{ae}(n+1) + c</em>{3ae} \overline{p}^{zxt}<em>{ae}(n) + d</em>{3ae} f_{1ae} \right) )</td>
</tr>
<tr>
<td>11</td>
<td>end</td>
</tr>
<tr>
<td>12</td>
<td>( n \leftarrow n + 1 )</td>
</tr>
<tr>
<td>13</td>
<td>end</td>
</tr>
</tbody>
</table>

**Algorithm 1:** Red-Black Gauss-Seidel iteration scheme for solving the pressure Poisson equation

### 6.3.4 Iteration Strategy

In the original CPU implementation of the MNIM, the final set of discrete algebraic equations is solved iteratively using Gauss-Seidel scheme in conjunction with a SIMPLE-like algorithm that couples the field variables [35]. The corresponding algorithm in the present GPU implementation is described below. The coefficients and the transverse-averaged unknowns for each element in the computational domain are dynamically allocated and stored in a linear array in the host memory. During each
execution of the code, all the source data is transferred to the memory on the device side by calling the CUDA function `cudaMemcpy()`. The coefficients in the scheme’s final set of discrete equations vary with time and are calculated at the beginning of every time step, which is referred to as an outer iteration. On the other hand, GPU

```
1 for m = 1 to maximum time step do
2     t ← t + Δt
3     evaluate all coefficients
4     while not converged do
5         for n_v = 1 to N_v do
6             update velocity variables
7             for n_p = 1 to N_p do
8                 update pressure variables using Red-Black Gauss-Seidel (Algorithm 1)
9                 n_p ← n_p + 1
10             end
11             n_v ← n_v + 1
12         end
13         convergence check
14     end
15 end
```

**Algorithm 2**: Red-Black Gauss-Seidel in conjunction with a SIMPLE-like algorithm

threads are invoked in each inner iteration by the inner kernels including two kernel functions: one is in charge of simultaneously updating the velocity variables and the other is the pressure kernel\(^{\dagger}\). Similar to the case with the iterative approach adopted by the original CPU implementation of the MNIM as well as many other iterative approaches, the velocity kernel in the present GPU implementation is iterated only once for fixed pressure field every time it is invoked, while the pressure kernel is iterated 10 times for fixed velocity field to reach optimum convergence\(^{\ddagger}\). At the end of the outer

\(^{\dagger}\)In fact the pressure kernel is further decomposed into a red- and black-kernel, corresponding to the Red-Black Gauss-Seidel iteration scheme.

\(^{\ddagger}\)To achieve optimum convergence, the pressure update is iterated 15 times for given velocity field in the CPU implementation of the MNIM [35]. The better convergence rate obtained in the present GPU implementation is due to the adoption of the Red-Black Gauss-Seidel scheme instead of the traditional Gauss-Seidel.
iteration, if either the convergence criterion is satisfied (for a steady-state problem) or the time step reaches the maximum, the GPU terminates all the running kernels and copies the discrete unknowns back to the CPU memory. Then the CPU takes over to complete the task of post-processing. Algorithm 2 shows the above-discussed algorithm for outer and inner iterations.

### 6.4 Validation Cases

To validate the GPU implementation of the MNIM for the 3D Navier-Stokes equations, the lid-driven cavity flow problem in a cube and in a prism with aspect ratio of two are solved in this section. Both problems are solved using the new GPU implementation for various Reynolds numbers. Comparison between the numerical results of the present simulations and the results from literature is made. The computational performance of the MNIM-based Navier-Stokes solver on GPU is also analyzed and compared with the performance of the corresponding CPU implementation. All the GPU simulations are done on one of the 32 nodes of the Accelerator Cluster (AC) at the National Center for Supercomputing Applications (NCSA).

#### 6.4.1 3D Lid-Driven Cavity Flow in a Cube

The simulation of the three-dimensional lid-driven cavity flow in a cube is carried out as the first validation case. The problem has been extensively used to test numerical schemes for the Navier-Stokes equations [86–89]. The computational domain and cavity configurations are shown in figure 6.3. The top lid ($y = 1$) moves in the $x$ direction at a constant speed $u_{\text{lid}}$. The other walls of the cavity are stationary. As the Reynolds number is given by $\text{Re} = UL/\nu = u_{\text{lid}}/\nu$, hence different Reynolds numbers can be achieved by varying the lid speed $u_{\text{lid}}$. Simulation results are reported below for $\text{Re} = 100$, 400, and 1000. The $u$ velocity profile along the vertical centerline
(x = 0.5, y ∈ [0, 1], z = 0.5) for Re = 100 is shown in figure 6.4, while the v velocity profile along the horizontal centerline (x ∈ [0, 1], y = 0.5, z = 0.5) for the same Reynolds number is shown in figure 6.5. The corresponding velocity profiles for Re = 400 and 1000 are shown in figures 6.6–6.7 and 6.8–6.9, respectively. The GPU simulation results are compared with the reference data as well in each of the plot. Very good agreement is observed between the results obtained in this work and those reported in reference [86], despite the fact that a 20 × 20 × 20 coarse mesh is used in the present GPU simulation for all Reynolds numbers tested. It should be noted that the 20 × 20 × 20 mesh used here is non-uniform, with the geometric factor of 1.1, 1.2, and 1.3 for Re = 100, 400, and 1000, respectively. Uniform mesh of the same size is also tested but the numerical results are found to be less accurate, indicating that a uniform 20 × 20 × 20 mesh may still be too coarse to capture the variation within the

Figure 6.3: Configuration and coordinate system of the lid-driven cavity flow in a cube.
boundary layer. Moreover, the flow structures in the center $yz$ plane at $x = 0.5$ are shown in figures 6.10–6.12 for different Reynolds numbers. The two primary vortices located symmetrically with respect to the vertical centerline in the $x = 0.5$ plane are found to move toward the lower corners of the cavity as Reynolds number increases from 100 to 1000. It is observed as well that two smaller secondary vortices gradually form and evolve in a symmetric arrangement near the upper corners with increasing Reynolds numbers. In figures 6.10–6.12, the flow structures for different Reynolds numbers obtained in the present GPU simulation are also compared with those of reference [86] and very good qualitative agreement is observed.

6.4.2 3D Lid-Driven Cavity Flow in a Prism

The three-dimensional lid-driven cavity flow in a prism is used as the second validation problem to test the GPU implementation of the MNIM-based Navier-Stokes solver. Figure 6.13 shows the computational domain and cavity configuration for the simulation. The cavity configuration is the same as in the previous validation case, except that the aspect ratio of the prism is increased to two, which not only doubles the size of the computational domain but also causes significant difference in the flow structures. Three Reynolds numbers ($Re = 100, 400,$ and $1000$) are achieved for the simulations by varying the lid speed $u_{\text{lid}}$ at $y = 2$. A $30 \times 60 \times 30$ non-uniform mesh with the geometric factor of 1.1 is used for all Reynolds numbers tested. Figure 6.14 shows the resulting $u$ velocity profile along the vertical centerline ($x = 0.5, y \in [0, 2], z = 0.5$) for $Re = 100$. The flow structure for the same Reynolds number in the $z = 0.5$ plane of the prismatic cavity is shown in figure 6.15. The corresponding velocity profiles and flow maps for $Re = 400$ and 1000 are shown in figures 6.16–6.17 and 6.18–6.19, respectively. By comparing figures 6.4 and 6.14, it is found that for $Re = 100$, the centerline $u$ velocity profile obtained in the prismatic cavity contains an inflection point around $y = 1$, whereas the same profile obtained in the unit cavity
Figure 6.4: $u$ velocity profile for Re = 100 along the vertical centerline of the cubic cavity.

Figure 6.5: $v$ velocity profile for Re = 100 along the horizontal centerline of the cubic cavity.
Figure 6.6: $u$ velocity profile for $Re = 400$ along the vertical centerline of the cubic cavity.

Figure 6.7: $v$ velocity profile for $Re = 400$ along the horizontal centerline of the cubic cavity.
Figure 6.8: $u$ velocity profile for Re = 1000 along the vertical centerline of the cubic cavity.

Figure 6.9: $v$ velocity profile for Re = 1000 along the horizontal centerline of the cubic cavity.
Figure 6.10: Velocity field for $\text{Re} = 100$ in the $x = 0.5$ plane of the cubic cavity.
Figure 6.11: Velocity field for Re = 400 in the $x = 0.5$ plane of the cubic cavity.
Figure 6.12: Velocity field for Re = 1000 in the $x = 0.5$ plane of the cubic cavity.
has no such point. Similar comparison for higher Reynolds number cases (Re = 400 and 1000) can also be made to observe that the $u$ velocity profile along the vertical centerline in the prismatic cavity contains more inflection points than the unit cavity, indicating that the flow pattern of the lid-driven cavity flow is significantly more complex in a prism with aspect ratio larger than 1. Such complexity can further be revealed by studying the evolution of vortices in the flow. The flow map plots (figures 6.15, 6.17, and 6.19) show that a primary vortex develops in the upper half of the cavity close to the top lid and shifts downwards with increasing Reynolds numbers, while a secondary vortex, much weaker than the primary vortex in strength, gradually evolves in the lower half of the cavity and moves leftwards as the Reynolds number is increased from 100 to 1000. Additionally, the centerline $u$ velocity profiles obtained in the present GPU simulation are compared with reference data obtained using a $35 \times 35 \times 70$ mesh on CPU, as shown in figures 6.14, 6.16, and 6.18. The velocity

![Figure 6.13: Configuration and coordinate system of the lid-driven cavity flow in a prism.](image)
profiles in this work and in reference [87] are almost identical for the low Reynolds number case (Re = 100). The GPU simulation results still agree reasonably well with the results of reference [87] at Reynolds number as high as 1000.

### 6.4.3 Performance Analysis

The GPU simulations in this work are carried out on the 32-node Accelerator Cluster at NCSA. Each node of the AC cluster has: 2 dual-core AMD Opterons processors running at 2.4 GHz, 8 GB of CPU memory, 1 NVIDIA Tesla S1070 GPU computing server including 4 Tesla C1060 GPUs and 16 GB of graphics memory in total. All the GPU simulations are performed on one of the AC nodes, and are restricted to only one AMD processor accelerated by one Tesla C1060 GPU with 4 GB of GPU memory on that node. This restriction is intentionally imposed to obtain fair speedup ratio between single CPU and single GPU simulations. (Generally, simulations running on a multi-GPU platform such as Tesla S1070 can generate better speedup when compared with simulations running on a single GPU.) The CUDA code was compiled using the NVCC compiler, with support of CUDA SDK (Software Development Kit) 3.0. The CPU implementation of the MNIM for the 3D N-S equations was originally coded in Fortran without optimization. Since the GPU implementation is coded in CUDA which is a variation of the C programming language, considerable amount of effort was made to rewrite the Fortran code (approximately 4000 lines) in the C language. The intension of this effort is to eliminate the performance difference caused by the different programming languages and compilers. The C code is also optimized during coding. Tests have shown that the C code runs approximately twice as fast as the original Fortran code. In addition, the C code was compiled using the GCC compiler with the level three optimization option “-O3”, and runs under the single-threaded mode on a PC with an Intel Core 2 T9300 CPU.

The performance analysis is conducted by simulating the lid-driven cavity flow in
Figure 6.14: $u$ velocity profile for $Re = 100$ along the vertical centerline of the prismatic cavity with aspect ratio of 2.

Figure 6.15: Velocity field for $Re = 100$ in the $z = 0.5$ plane of the prismatic cavity with aspect ratio of 2.
Figure 6.16: $u$ velocity profile for $Re = 400$ along the vertical centerline of the prismatic cavity with aspect ratio of 2.

Figure 6.17: Velocity field for $Re = 400$ in the $z = 0.5$ plane of the prismatic cavity with aspect ratio of 2.
Figure 6.18: $u$ velocity profile for $Re = 1000$ along the vertical centerline of the prismatic cavity with aspect ratio of 2.

Figure 6.19: Velocity field for $Re = 1000$ in the $z = 0.5$ plane of the prismatic cavity with aspect ratio of 2.
a unit cube for the Re = 1000 case. Figure 6.20 shows the percentages of computation time for different kernels and operations in the GPU simulation. As mentioned in section 6.3.3, solving the pressure Poisson equation is taking the largest portion of computation time for the Navier-Stokes solver. Figure 6.20 shows that the two pressure kernels—the “red_pressure_kernel” and “black_pressure_kernel”—together take up to over 70% of the total GPU time. The performance comparison of the CUDA code and the C code is shown in table 6.1 for two mesh sizes (20 × 20 × 20 and 40 × 40 × 40). The GPU time and CPU time reported in table 6.1 are computational time for different simulations until steady-state is achieved. It is found that similar to the case of the GPU Burgers solver discussed in the previous chapter, the speedup obtained here increases with increasing mesh resolution. The speedup goes from 9.77 for the 20 × 20 × 20 mesh to 12.49 for the 40 × 40 × 40 mesh. The speedups are quite impressive, especially considering that double precision is retained in the
present GPU implementation of the MNIM. The efficiency of the coarse mesh method is thus shown to be greatly strengthened by the state-of-the-art computational power.

Table 6.1: Performance comparison of GPU and CPU implementations using the MNIM to simulate the lid-driven cavity flow in a cube.

<table>
<thead>
<tr>
<th>Mesh</th>
<th>GPU time (s)</th>
<th>CPU time (s)</th>
<th>Overall Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 × 20 × 20</td>
<td>113.87</td>
<td>1112.07</td>
<td>9.77</td>
</tr>
<tr>
<td>40 × 40 × 40</td>
<td>852.08</td>
<td>10642.51</td>
<td>12.49</td>
</tr>
</tbody>
</table>

6.5 Summary

This chapter presents a GPU implementation of the Modified Nodal Integral Method for the 3D, time-dependent, incompressible Navier-Stokes equations using the CUDA programming model. After a brief review of the numerical scheme, several major aspects crucial to the new GPU implementation are discussed in detail. A one-to-one mapping is applied to decompose the computational domain and GPU grid, leading to an optimized parallelism among GPU threads. Double precision floating point calculations are adopted to ensure the precision needed for flow simulations using the MNIM. A pressure Poisson solver is incorporated with the Red-Black Gauss-Seidel iteration scheme, which ideally suits the parallel nature of GPU-computing. An iteration strategy using the Red-Black Gauss-Seidel in conjunction with a SIMPLE-like algorithm is proposed for the present implementation. The GPU implementation is then applied to simulate the 3D lid-driven cavity flows in a unit cube and a prism with aspect ratio of two for validation purposes. Numerical results obtained in this work are compared with reference data for various Reynolds numbers. Very good agreement is found. Finally a performance analysis is conducted for the new imple-
mentation of the MNIM on graphics processors, indicating that performance of the MNIM on GPU can be an order of magnitude faster than on a mainstream CPU.
7 Summary, Conclusions and Future Work

7.1 Summary and Conclusions

Nodal methods, a class of coarse mesh methods, usually require less computing time to achieve a given accuracy than many other conventional numerical schemes that rely on fine meshes. As a variation of the nodal methods, the modified nodal integral method (MNIM) is capable of providing efficient solution to the time-dependent, incompressible Navier-Stokes (N-S) equations with high accuracy. However, the transverse integration procedure (TIP) required in the formulation of the MNIM limits the scheme to be only applicable to rectangular elements. In order to remove this limitation and extend the nodal scheme to non-rectangular meshes/domains, a modified nodal integral method incorporated with generic quadrilateral elements for the time-dependent, incompressible N-S equations is developed in the first part of this dissertation, using a simple isoparametric geometry mapping. The mapping is used to transform: 1) the irregular four-node quadrilateral elements into square elements; 2) the original set of N-S equations to a set of transformed equations valid over the transformed computational domain. Then the new nodal scheme is formulated for the transformed equations. While the transformation of the quadrilateral elements to the square elements is straightforward, the transformed set of N-S equations are much more complicated, including significantly large number of additional, linear and nonlinear terms that are justified based on physical grounds. The complexity of the transformed equations not only leads to more complex expressions for the intermediate coefficients but also adds additional terms to the final set of algebraic,
discrete equations of the new nodal scheme, compared to those of the original MNIM for rectangular elements. The numerical scheme developed is applied to several test problems of increasing complexity. Results show that the scheme works very well with quadrilateral elements of different shapes and degrees of distortion, maintaining the high accuracy and efficiency of the MNIM; and that the new scheme has inherent upwinding.

To further enhance the computational capabilities, one needs to exploit the latest developments in computing hardware. Realizing that graphics processing units (GPUs) can provide superior computational power over conventional CPUs, the cutting-edge GPU-computing and the highly efficient nodal scheme are married in a double precision GPU implementation of the MNIM for the 3D, incompressible N-S equations in the second part of this dissertation. In the present GPU implementation: a one-to-one mapping is applied to decompose the computational domain and GPU grid, leading to optimal parallelism; a pressure Poisson solver that fits the parallel nature of GPU-computing is developed incorporated with the Red-Black Gauss-Seidel iteration scheme; an iteration strategy using the Red-Black Gauss-Seidel in conjunction with a SIMPLE-like algorithm is used. The GPU implementation is applied to simulate the lid-driven cavity flows in a unit cube and a prism with an aspect ratio of two for various Reynolds numbers. Good agreement is found between simulation results and reference solutions, despite the fact that considerably coarser meshes are used in the present work. A performance analysis indicates that the MNIM on GPU can be an order of magnitude faster than on a CPU.

7.2 Future Work

An immediate extension of the present work is to extend the nodal scheme developed in this dissertation to three spatial dimensions. Instead of the generic quadrilateral
elements used here, the hexahedral elements will be required to incorporate with the nodal scheme for three dimensions. In addition, the formulation of the nodal scheme in 3D will require a different mapping that transforms the generic hexahedral elements to unit cubes, and transforms the governing equations to a set of new equations. It can be foreseen that the transformed equations will be significantly more complicated than those in the 2D case, bringing additional complexity to the intermediate coefficients as well as the final set of discrete equations.

Another possible extension of the present work is to use more general, rather than linear, mapping to allow the use of elements with curved boundaries for the nodal scheme. The nodal scheme developed in this dissertation is incorporated with quadrilateral elements by using a simple isoparametric mapping that is linear. However, curve-sided elements may constitute better approximation for irregular computation domains than quadrilaterals. Thus the nodal scheme can be more efficient and accurate if incorporated with curve-sided elements by using a different, more general mapping.

A third logical extension of this research work is to include turbulence models into the nodal scheme developed here. The present nodal scheme, although works accurately and efficiently on complicated meshes, is only applicable to laminar flows, as turbulence modeling has been completely ignored in the development of the scheme. In fact, an MNIM for the $k$-$\epsilon$ equations was successfully developed earlier to simulate turbulent flows [19], which can be taken great advantage of in the future work to extend the nodal scheme presented in this dissertation for turbulence simulations.

A last possible extension of the present work is to develop a multi-GPU implementation of the MNIM. The GPU implementation presented in this dissertation is accelerated by only one graphics processor with one level of on-chip parallelism. A future multi-GPU implementation of the MNIM is expected to involve calculations on multiple GPUs, where an extra level of inter-GPU parallelism is added so as to
obtain better speedups for larger scale problems. The extension of such implementa-
tion from single GPU to multiple GPUs is straightforward but by no means trivial.
Message passing among the multiple GPUs requires the incorporation with other parallel computing models such as MPI, and this can bring considerable amount of complexity to the algorithm design, coding, and testing for the new implementation.
Bilinear Function

Bilinear Lagrange interpolation functions for two-dimensional, 4-node interpolation is given by

\[ l_i(\xi, \eta) = \prod_{j=1}^{4} \prod_{k=1}^{4} \frac{(\xi - \xi_j)(\eta - \eta_k)}{(\xi_i - \xi_j)(\eta_i - \eta_k)} \quad (i = 1, 2, 3, 4), \tag{A.1} \]

where \( \xi_i \) and \( \eta_i \) indicate the nodal values of the \( i^{th} \) node’s \( \xi \) and \( \eta \) coordinates.
Derivation of the Derivatives of the Transformed Coordinates

The first and second derivatives of the transformed coordinates (ξ and η) with respect to the original coordinates (x and y) are derived below. Equation (3.1) can be rewritten in a compact form as

\[
\begin{align*}
  x &= a_0 + a_1 \xi + a_2 \eta + a_3 \xi \eta \\
  y &= b_0 + b_1 \xi + b_2 \eta + b_3 \xi \eta
\end{align*}
\]  

(B.1)

where

\[
\begin{align*}
  a_0 &= \frac{x_1 + x_2 + x_3 + x_4}{4} \\
  a_1 &= \frac{x_1 - x_2 - x_3 + x_4}{4} \\
  a_2 &= \frac{x_1 + x_2 - x_3 - x_4}{4} \\
  a_3 &= \frac{x_1 - x_2 + x_3 - x_4}{4} \\
  b_0 &= \frac{y_1 + y_2 + y_3 + y_4}{4} \\
  b_1 &= \frac{y_1 - y_2 - y_3 + y_4}{4} \\
  b_2 &= \frac{y_1 + y_2 - y_3 - y_4}{4} \\
  b_3 &= \frac{y_1 - y_2 + y_3 - y_4}{4}
\end{align*}
\]  

(B.2)

Differentiating equation (B.1) with respect to x and y yields

\[
\begin{align*}
  1 &= a_1 \xi_x + a_2 \eta_x + a_3 (\xi_x \eta + \eta_x \xi) \\
  0 &= b_1 \xi_x + b_2 \eta_x + b_3 (\xi_x \eta + \eta_x \xi)
\end{align*}
\]  

(B.3)
and

\[
\begin{aligned}
0 &= a_1 \xi_{,y} + a_2 \eta_{,y} + a_3 (\xi_{,y} \eta + \eta_{,y} \xi) \\
1 &= b_1 \xi_{,y} + b_2 \eta_{,y} + b_3 (\xi_{,y} \eta + \eta_{,y} \xi),
\end{aligned}
\]  
(B.4)

respectively. Equations (B.3) and (B.4) can be combined and written as

\[
\begin{bmatrix}
a_1 + a_3 \eta & a_2 + a_3 \xi \\
b_1 + b_3 \eta & b_2 + b_3 \xi
\end{bmatrix}
\begin{bmatrix}
\xi_{,x} & \xi_{,y} \\
\eta_{,x} & \eta_{,y}
\end{bmatrix} = \mathbf{I},
\]  
(B.5)

Solving equation (B.5) for the first derivatives of \(\xi\) and \(\eta\) yields

\[
\begin{bmatrix}
\xi_{,x} & \xi_{,y} \\
\eta_{,x} & \eta_{,y}
\end{bmatrix} = \begin{bmatrix}
a_1 + a_3 \eta & a_2 + a_3 \xi \\
b_1 + b_3 \eta & b_2 + b_3 \xi
\end{bmatrix}^{-1},
\]  
(B.6)

the LHS of which is the Jacobian \(J(x, y)\). The second derivatives can be derived using the same approach. Differentiating equation (B.3) with respect to \(x\) and \(y\) yields

\[
\begin{aligned}
0 &= a_1 \xi_{,xx} + a_2 \eta_{,xx} + a_3 (\xi_{,xx} \eta + \eta_{,xx} \xi + 2 \xi_{,x} \eta_{,x}) \\
0 &= b_1 \xi_{,xx} + b_2 \eta_{,xx} + b_3 (\xi_{,xx} \eta + \eta_{,xx} \xi + 2 \xi_{,x} \eta_{,x}),
\end{aligned}
\]  
(B.7)

and

\[
\begin{aligned}
0 &= a_1 \xi_{,yy} + a_2 \eta_{,yy} + a_3 (\xi_{,yy} \eta + \eta_{,yy} \xi + 2 \xi_{,y} \eta_{,y}) \\
0 &= b_1 \xi_{,yy} + b_2 \eta_{,yy} + b_3 (\xi_{,yy} \eta + \eta_{,yy} \xi + 2 \xi_{,y} \eta_{,y}),
\end{aligned}
\]  
(B.8)

respectively. Equations (B.7) and (B.8) can be combined and written as

\[
\begin{bmatrix}
a_1 + a_3 \eta & a_2 + a_3 \xi \\
b_1 + b_3 \eta & b_2 + b_3 \xi
\end{bmatrix}
\begin{bmatrix}
\xi_{,xx} & \xi_{,yy} \\
\eta_{,xx} & \eta_{,yy}
\end{bmatrix} = -2\begin{bmatrix}
a_3 \xi_{,x} \eta_{,x} & a_3 \xi_{,y} \eta_{,y} \\
b_3 \xi_{,x} \eta_{,x} & b_3 \xi_{,y} \eta_{,y}
\end{bmatrix},
\]  
(B.9)
Solving equation (B.9) for the second derivatives of $\xi$ and $\eta$ yields

$$
\begin{bmatrix}
\xi_{,xx} & \xi_{,yy} \\
\eta_{,xx} & \eta_{,yy}
\end{bmatrix} = -2 \begin{bmatrix}
a_1 + a_3 \eta & a_2 + a_3 \xi \\
b_1 + b_3 \eta & b_2 + b_3 \xi
\end{bmatrix}^{-1} \begin{bmatrix}
a_3 \xi_{,x} \eta_{,x} & a_3 \xi_{,y} \eta_{,y} \\
b_3 \xi_{,x} \eta_{,x} & b_3 \xi_{,y} \eta_{,y}
\end{bmatrix}, \quad (B.10)
$$

or

$$
\begin{bmatrix}
\xi_{,xx} & \xi_{,yy} \\
\eta_{,xx} & \eta_{,yy}
\end{bmatrix} = -2J(x, y) \begin{bmatrix}
a_3 \xi_{,x} \eta_{,x} & a_3 \xi_{,y} \eta_{,y} \\
b_3 \xi_{,x} \eta_{,x} & b_3 \xi_{,y} \eta_{,y}
\end{bmatrix}. \quad (B.11)
$$
The definition of coefficients $A$ appearing in chapter 3 are given below.

\[ A_{11} \equiv \frac{\eta m_1 \text{csch} \left( \frac{K_1}{K_2} \right)}{2K_2} K_1 \] (C.1)

\[ A_{12} \equiv \frac{\eta m_1 \text{csch} \left( \frac{K_{1+1}}{K_{2+1}} \right)}{2K_{2+1}} K_{1+1} \] (C.2)

\[ A_{13} \equiv \frac{\xi m_1 K_5}{2K_6} \left[ 1 + \coth \left( \frac{K_5}{K_6} \right) \right] \] (C.3)

\[ A_{14} \equiv \frac{\xi m_1}{2K_6 K_{6+1}} \left\{ K_{5+1} K_{6+1} \left[ -1 + \coth \left( \frac{K_{5+1}}{K_{6+1}} \right) \right] + K_5 K_{6+1} \left[ 1 + \coth \left( \frac{K_5}{K_6} \right) \right] \right\} \] (C.4)

\[ A_{15} \equiv \frac{\xi m_1 K_{5+1}}{2K_{5+1}} \left( 1 - e^{\frac{K_{6+1}}{K_{6+1}}} \right) K_{6+1} \] (C.5)

\[ A_{16} \equiv \frac{\xi m_1}{K_5 K_6} \left\{ K_5 \left[ 1 + \coth \left( \frac{K_5}{K_6} \right) \right] - K_6 \right\} \] (C.6)

\[ A_{17} \equiv \frac{\xi m_1}{K_{5+1} K_{6+1}} \left\{ K_{5+1} \left[ -1 + \coth \left( \frac{K_{5+1}}{K_{6+1}} \right) \right] - K_{6+1} \right\} \] (C.7)

\[ A_{18} \equiv \eta m_1 \left[ \frac{1}{K_1} - \frac{\text{csch} \left( \frac{K_1}{K_2} \right)}{K_2} \right] \] (C.8)

\[ A_{19} \equiv -\eta m_1 \left[ \frac{1}{K_{1+1}} - \frac{\text{csch} \left( \frac{K_{1+1}}{K_{2+1}} \right)}{K_{2+1}} \right] \] (C.9)

\[ A_{31} \equiv \frac{\eta m_1 \text{csch} \left( \frac{K_3}{K_4} \right)}{2K_4} K_3 \] (C.10)

\[ A_{32} \equiv \frac{\eta m_1 \text{csch} \left( \frac{K_{3+1}}{K_{4+1}} \right)}{2K_{4+1}} K_{3+1} \] (C.11)
\[ A_{33} \equiv \frac{\xi_m K_7}{2K_8} \left[ 1 + \coth \left( \frac{K_7}{K_8} \right) \right] \] (C.12)

\[ A_{34} \equiv \frac{\xi_m}{2K_8 K_{8i+1}} \left\{ K_{7i+1} K_8 \left[ -1 + \coth \left( \frac{K_{7i+1}}{K_{8i+1}} \right) \right] + K_7 K_{8i+1} \left[ 1 + \coth \left( \frac{K_7}{K_8} \right) \right] \right\} \] (C.13)

\[ A_{35} \equiv \frac{\xi_m K_{7i+1}}{(1 - e^{\frac{K_8}{K_{8i+1}}}) K_{8i+1}} \] (C.14)

\[ A_{36} \equiv \frac{\xi_m}{K_7 K_8} \left\{ K_7 \left[ 1 + \coth \left( \frac{K_7}{K_8} \right) \right] - K_8 \right\} \] (C.15)

\[ A_{37} \equiv \frac{\xi_m}{K_{7i+1} K_{8i+1}} \left\{ K_{7i+1} \left[ -1 + \coth \left( \frac{K_{7i+1}}{K_{8i+1}} \right) \right] - K_{8i+1} \right\} \] (C.16)

\[ A_{38} \equiv \eta_m \left[ \frac{1}{K_3} - \frac{\csch \left( \frac{K_3}{K_4} \right)}{K_4} \right] \] (C.17)

\[ A_{39} \equiv -\eta_m \left[ \frac{1}{K_{3i+1}} - \frac{\csch \left( \frac{K_{3i+1}}{K_{4i+1}} \right)}{K_{4i+1}} \right] \] (C.18)

\[ A_{41} \equiv \frac{\eta_m K_1}{2K_2} \left[ 1 + \coth \left( \frac{K_1}{K_2} \right) \right] \] (C.19)

\[ A_{42} \equiv \frac{\eta_m}{2K_2 K_{2j+1}} \left\{ K_{1j+1} K_2 \left[ -1 + \coth \left( \frac{K_{1j+1}}{K_{2j+1}} \right) \right] + K_1 K_{2j+1} \left[ 1 + \coth \left( \frac{K_1}{K_2} \right) \right] \right\} \] (C.20)

\[ A_{43} \equiv \frac{\eta_m K_{1j+1}}{2K_{1j+1}} \] (C.21)

\[ A_{44} \equiv \frac{\xi_m \csch \left( \frac{K_{5j+1}}{K_{6j+1}} \right)}{2K_{0j+1}} \] (C.22)

\[ A_{45} \equiv \frac{\xi_m \csch \left( \frac{K_5}{K_6} \right) K_5}{2K_6} \] (C.23)

\[ A_{46} \equiv \xi_m \left[ \frac{1}{K_5} - \frac{\csch \left( \frac{K_5}{K_6} \right)}{K_6} \right] \] (C.24)

\[ A_{47} \equiv -\xi_m \left[ \frac{1}{K_{5j+1}} - \frac{\csch \left( \frac{K_{5j+1}}{K_{6j+1}} \right)}{K_{6j+1}} \right] \] (C.25)

\[ A_{48} \equiv \frac{\eta_m}{K_1 K_2} \left\{ K_1 \left[ 1 + \coth \left( \frac{K_1}{K_2} \right) \right] - K_2 \right\} \] (C.26)
\[ A_{49} \equiv \frac{\eta_{m2}}{K_{1j+1}K_{2j+1}} \left\{ K_{1j+1} \left[ -1 + \coth \left( \frac{K_{1j+1}}{K_{2j+1}} \right) \right] - K_{2j+1} \right\} \] (C.27)

\[ A_{61} \equiv \frac{\eta_{m2}K_3}{2K_4} \left[ 1 + \coth \left( \frac{K_3}{K_4} \right) \right] \] (C.28)

\[ A_{62} \equiv \frac{\eta_{m2}}{2K_4 K_{1j+1}} \left\{ K_{3j+1}K_4 \left[ -1 + \coth \left( \frac{K_{3j+1}}{K_{4j+1}} \right) \right] + K_3K_{4j+1} \left[ 1 + \coth \left( \frac{K_3}{K_4} \right) \right] \right\} \] (C.29)

\[ A_{63} \equiv \frac{\eta_{m2}K_3 K_4}{2K_4 K_{1j+1}} \left[ 1 + \coth \left( \frac{K_{3j+1}}{K_{4j+1}} \right) \right] \] (C.30)

\[ A_{64} \equiv \frac{\xi_{m2} \text{csch} \left( \frac{K_7}{K_8} \right)}{2K_8} \] (C.31)

\[ A_{65} \equiv \frac{\xi_{m2} \text{csch} \left( \frac{K_7}{K_8} \right)}{2K_8} \] (C.32)

\[ A_{66} \equiv \xi_{m2} \left[ \frac{1}{K_7} - \frac{\text{csch} \left( \frac{K_7}{K_8} \right)}{K_8} \right] \] (C.33)

\[ A_{67} \equiv -\xi_{m2} \left[ \frac{1}{K_{7j+1}} - \frac{\text{csch} \left( \frac{K_{7j+1}}{K_{8j+1}} \right)}{K_{8j+1}} \right] \] (C.34)

\[ A_{68} \equiv \frac{\eta_{m2}}{K_3 K_4} \left\{ K_3 \left[ 1 + \coth \left( \frac{K_3}{K_4} \right) \right] - K_4 \right\} \] (C.35)

\[ A_{69} \equiv \frac{\eta_{m2}}{K_3 K_{4j+1}} \left\{ K_{3j+1} \left[ -1 + \coth \left( \frac{K_{3j+1}}{K_{4j+1}} \right) \right] - K_{4j+1} \right\} \] (C.36)

\[ A_{c1} \equiv \frac{K_1 \left[ 1 + \coth \left( \frac{K_1}{K_2} \right) \right] - K_2}{2K_1} \] (C.37)

\[ A_{c2} \equiv \frac{K_1 \left[ 1 - \coth \left( \frac{K_1}{K_2} \right) \right] + K_2}{2K_1} \] (C.38)

\[ A_{c3} \equiv \frac{K_1 \coth \left( \frac{K_1}{K_2} \right) - K_2}{K_1^2} \] (C.39)

\[ A_{d1} \equiv \frac{K_5 \left[ 1 + \coth \left( \frac{K_5}{K_6} \right) \right] - K_6}{2K_5} \] (C.40)

\[ A_{d2} \equiv \frac{K_5 \left[ 1 - \coth \left( \frac{K_5}{K_6} \right) \right] + K_6}{2K_5} \] (C.41)

\[ A_{d3} \equiv \frac{K_5 \coth \left( \frac{K_5}{K_6} \right) - K_6}{K_5^2} \] (C.42)
\[ A_{g1} \equiv \frac{K_3 \left[ 1 + \coth \left( \frac{K_3}{K_4} \right) \right] - K_4}{2K_3} \] (C.43)

\[ A_{g2} \equiv \frac{K_3 \left[ 1 - \coth \left( \frac{K_3}{K_4} \right) \right] + K_4}{2K_3} \] (C.44)

\[ A_{g3} \equiv \frac{K_7 \left[ 1 + \coth \left( \frac{K_7}{K_8} \right) \right] - K_8}{2K_7} \] (C.45)

\[ A_{g4} \equiv \frac{1}{2} \left[ -1 + \frac{K_8}{K_7} + \coth \left( \frac{K_7}{K_8} \right) \right] \] (C.46)

\[ A_{g5} \equiv \frac{K_3 \coth \left( \frac{K_3}{K_4} \right) - K_4}{K_3^2} \] (C.47)

\[ A_{g6} \equiv -\frac{K_7 \coth \left( \frac{K_7}{K_8} \right) - K_8}{K_7^2} \] (C.48)
Definition of Coefficients $F$

The definition of coefficients $F$ appearing in chapter 3 are given below.

\begin{align*}
F_{11} &\equiv -A_{14} - \frac{A_{16} A_{d2}}{A_{d3}} + \frac{A_{17} A_{d1+i}}{A_{d3+i}} & (D.1) \\
F_{12} &\equiv -A_{13} + \frac{A_{16} A_{d1}}{A_{d3}} & (D.2) \\
F_{13} &\equiv A_{15} - \frac{A_{17} A_{d2+i}}{A_{d3+i}} & (D.3) \\
F_{14} &\equiv A_{11} - \frac{A_{18} A_{c2}}{A_{c3}} & (D.4) \\
F_{15} &\equiv -A_{11} - \frac{A_{18} A_{c1}}{A_{c3}} & (D.5) \\
F_{16} &\equiv -A_{12} - \frac{A_{19} A_{c2+i}}{A_{c3+i}} & (D.6) \\
F_{17} &\equiv A_{12} - \frac{A_{19} A_{c1+i}}{A_{c3+i}} & (D.7) \\
F_{18} &\equiv \frac{1}{2} \left( \frac{A_{18}}{A_{c3}} - \frac{A_{16}}{A_{d3}} \right) & (D.8) \\
F_{19} &\equiv \frac{1}{2} \left( \frac{A_{19}}{A_{c3+i}} + \frac{A_{17}}{A_{d3+i}} \right) & (D.9) \\
F_{31} &\equiv -A_{34} + \frac{(A_{36} + A_{38}) A_{g4}}{A_{g5} - A_{g6}} + \frac{(A_{37} - A_{39}) A_{g3+i}}{A_{g5+i} - A_{g6+i}} & (D.10) \\
F_{32} &\equiv -A_{33} + \frac{(A_{36} + A_{38}) A_{g3}}{A_{g5} - A_{g6}} & (D.11) \\
F_{33} &\equiv A_{35} + \frac{(A_{37} - A_{39}) A_{g4+i}}{A_{g5+i} - A_{g6+i}} & (D.12) \\
F_{34} &\equiv A_{31} - \frac{(A_{36} + A_{38}) A_{g2}}{A_{g5} - A_{g6}} & (D.13) \\
F_{35} &\equiv -A_{31} - \frac{(A_{36} + A_{38}) A_{g1}}{A_{g5} - A_{g6}} & (D.14)
\end{align*}
\[ F_{36} \equiv -A_{32} + \frac{(A_{37} - A_{39})A_{g2^{i+1}}}{A_{g5^{i+1}} - A_{g6^{i+1}}} \]  
(D.15)

\[ F_{37} \equiv A_{32} + \frac{(A_{37} - A_{39})A_{g1^{i+1}}}{A_{g5^{i+1}} - A_{g6^{i+1}}} \]  
(D.16)

\[ F_{38} \equiv \frac{A_{36} A_{g5} + A_{38} A_{g6}}{A_{g5} - A_{g6}} \]  
(D.17)

\[ F_{39} \equiv -\frac{A_{g5^{i+1} - A_{g6^{i+1}}}{A_{g5} - A_{g6}} \]  
(D.18)

\[ F_{41} \equiv -A_{42} \frac{A_{48} A_{c2}}{A_{c3}} + \frac{A_{49} A_{c1_{j+1}}}{A_{c3_{j+1}}} \]  
(D.19)

\[ F_{42} \equiv -A_{41} + \frac{A_{48} A_{c1}}{A_{c3}} \]  
(D.20)

\[ F_{43} \equiv A_{43} - \frac{A_{49} A_{c2_{j+1}}}{A_{c3_{j+1}}} \]  
(D.21)

\[ F_{44} \equiv A_{45} - \frac{A_{46} A_{d2}}{A_{d3}} \]  
(D.22)

\[ F_{45} \equiv -A_{45} - \frac{A_{46} A_{d1}}{A_{d3}} \]  
(D.23)

\[ F_{46} \equiv -A_{44} - \frac{A_{47} A_{d2_{j+1}}}{A_{d3_{j+1}}} \]  
(D.24)

\[ F_{47} \equiv A_{44} - \frac{A_{47} A_{d1_{j+1}}}{A_{d3_{j+1}}} \]  
(D.25)

\[ F_{48} \equiv \frac{1}{2} \left( \frac{A_{46}}{A_{d3}} - \frac{A_{48}}{A_{c3}} \right) \]  
(D.26)

\[ F_{49} \equiv \frac{1}{2} \left( \frac{A_{49}}{A_{c3_{j+1}}} + \frac{A_{47}}{A_{d3_{j+1}}} \right) \]  
(D.27)

\[ F_{61} \equiv -A_{62} - \frac{(A_{66} + A_{68})A_{g2}}{A_{g5} - A_{g6}} - \frac{(A_{67} - A_{69})A_{g1_{j+1}}}{A_{g5_{j+1}} - A_{g6_{j+1}}} \]  
(D.28)

\[ F_{62} \equiv -A_{61} + \frac{(A_{66} + A_{68})A_{g1}}{A_{g5} - A_{g6}} \]  
(D.29)

\[ F_{63} \equiv A_{63} + \frac{(A_{67} - A_{69})A_{g2_{j+1}}}{A_{g5^{j+1}} - A_{g6^{j+1}}} \]  
(D.30)

\[ F_{64} \equiv A_{65} + \frac{(A_{66} + A_{68})A_{g4}}{A_{g5} - A_{g6}} \]  
(D.31)

\[ F_{65} \equiv -A_{65} - \frac{(A_{66} + A_{68})A_{g3}}{A_{g5} - A_{g6}} \]  
(D.32)

\[ F_{66} \equiv -A_{64} + \frac{(A_{67} - A_{69})A_{g4_{j+1}}}{A_{g5^{j+1}} - A_{g6^{j+1}}} \]  
(D.33)

\[ F_{67} \equiv A_{64} - \frac{(A_{67} - A_{69})A_{g3_{j+1}}}{A_{g5^{j+1}} - A_{g6^{j+1}}} \]  
(D.34)
\begin{align*}
F_{68} & \equiv - \frac{A_{66} A_{g5} + A_{68} A_{g6}}{A_{g5} - A_{g6}} & (D.35) \\
F_{69} & \equiv - \frac{A_{67} A_{g5_{j+1}} - A_{69} A_{g6_{j+1}}}{A_{g5_{j+1}} - A_{g6_{j+1}}} & (D.36) \\
F_{71} & \equiv - \frac{1}{2} \left( \frac{1}{A_{c3}} + \frac{1}{A_{d3}} + \frac{1}{t_0} \right) & (D.37) \\
F_{72} & \equiv \frac{1}{2} \left( \frac{1}{A_{c3}} + \frac{1}{A_{d3}} - \frac{1}{t_0} \right) & (D.38) \\
F_{73} & \equiv - \frac{A_{d2}}{A_{d3}} & (D.39) \\
F_{74} & \equiv - \frac{A_{d1}}{A_{d3}} & (D.40) \\
F_{75} & \equiv - \frac{A_{c2}}{A_{c3}} & (D.41) \\
F_{76} & \equiv - \frac{A_{c1}}{A_{c3}} & (D.42)
\end{align*}
```c
__global__ void burgers_kernel(float* u, float* v,
                              float* u_old, float* v_old)
{
    // Block indices
    int bx = blockIdx.x;
    int by = blockIdx.y;
    // Thread indices
    int tx = threadIdx.x;
    int ty = threadIdx.y;
    // Global thread indices
    int global_tx = bx * BLOCK_SIZE + tx;
    int global_ty = by * BLOCK_SIZE + ty;
    // Mapping from thread indices to mesh indices
    int row = global_ty + 1;
    int col = global_tx + 1;
    int k = row * NX + col;
    // Explicit Euler update rules
    u[k] = u_old[k] + dt*(Mu*(u_old[k+NX] - 2*u_old[k] + u_old[k-NX])
                         /(dx*dx) + (u_old[k+1] - 2*u_old[k] + u_old[k-NX])/(dy*dy))
           - u_old[k]*(u_old[k+NX] - u_old[k-NX])/(2*dx)
           - v_old[k]*(u_old[k+1] - u_old[k-1])/(2*dy));
    v[k] = v_old[k] + dt*(Mu*(v_old[k+NX] - 2*v_old[k] + v_old[k-NX])
                         /(dx*dx) + (v_old[k+1] - 2*v_old[k] + v_old[k-NX])/(dy*dy))
           - u_old[k]*(v_old[k+NX] - v_old[k-NX])/(2*dx)
           - v_old[k]*(v_old[k+1] - v_old[k-1])/(2*dy));
}
```
References


Author’s Biography

Kai Huang was born in Changzhou, Jiangsu Province, China on April 9, 1982. Kai received his Bachelor of Science degree in Computer Science and Technology from Nanjing University of Posts and Telecommunications, Nanjing, China in 2004. He then completed his Master of Science in Nuclear Science and Technology from Tsinghua University, Beijing, China. In August 2006, Kai was admitted to the Department of Nuclear, Plasma, and Radiological Engineering at the University of Illinois at Urbana-Champaign to pursue Doctor of Philosophy in Nuclear Engineering. Kai completed his Ph.D. in 2011.