Convergence Acceleration of Fluid-Flow Computation with Special Emphasis on Multigrid Technique

A thesis

submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

by

Subhra Sankar Kalita

(Roll No: 126103006)

Department of Mechanical Engineering

Indian Institute of Technology Guwahati

Guwahati - 781039

November, 2022

Declaration

I declare that the work contained in the thesis entitled **Convergence Acceleration of Fluid-Flow Computation with Special Emphasis on Multigrid Technique** is carried under the suprvison of **Professor Anoop Kumar Dass**, in the Department of Mechanical Engineering, Indian Institute of Technology Guwahati for the award of the degree of Doctor of Philosophy and this work has not been submitted elsewhere for a degree.

Subhra Sankar Kalita
Research Scholar
Department of Mechanical Engineering
Indian Institute of Technology Guwahati

November, 2022

Certificate

It is certified that the work contained in the thesis entitled **Convergence Acceleration of Fluid-Flow Computation with Special Emphasis on Multigrid Technique** by Subhra Sankar Kalita, a student of the Department of Mechanical Engineering, India Institute of Technology Guwahati, for the award of the degree of Doctor of Philosophy has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Dr. Anoop Kumar Dass
Professor
Department of Mechanical Engineering
Indian Institute of Technology Guwahati

November, 2022

DEDICATED TO MY PARENTS

Acknowledgement

A PhD work can only be accomplished through a lot of support, encouragement and inspiration from the well-wishers. At the outset, I express my gratitude to the Almighty for blessing me with the strength and patience required to complete a PhD work.

I would like to offer my sincere gratitude to my thesis supervisor, Prof. Anoop K. Dass. He has been a constant source of support, inspiration and invaluable guidance. I thank him from the bottom of my heart for imparting the basics of the relevant subjects in the best possible way before I embarked on my thesis work. He has imbibed in me the habit of scientific thinking and systematic analysis. He gave me ample freedom to carry out my work on my own while providing necessary suggestions from time to time. At this juncture, I also take the opportunity to express my sincere thanks to Prof. Dass's family members for being very affectionate and for their kind hospitality.

I thank the members of my Doctoral Committee; the chairman, Prof. Manmohan Pandey, the committee member, Prof. Amaresh Dalal (Department of Mechanical Engineering) and external member, Prof. Siddhartha Pratim Chakrabarty (Department of Mathematics) for their comments and suggestions during my progress seminars. Their invaluable suggestions were critical in shaping the present form of the thesis.

I would like to acknowledge the Department of Mechanical Engineering, Indian Institute of Technology Guwahati for providing me with the fellowship and facilities to carry out my research work. Besides, I sincerely thank the High-Performance Computing Environment (PARAM-ISHAN) of IIT Guwahati for providing the computational facility to carry out the computations in my research work.

I wish to acknowledge the contribution of anonymous reviewers of my conference papers for their valuable suggestions which certainly improve my research work further.

I take this opportunity to convey my gratefulness to my parents and my wife, Pallabi for their unbounded love, sacrifice, support, encouragement and patience during the course of my thesis work. All my family members have immensely contributed directly or indirectly towards the smooth completion of my thesis work. I gratefully acknowledge the patience, support and encouragement of my father-in-law and mother-in-law and Maina.

A special thanks to Mr Sandip Kumar Sarma and Dr. Ashif Iqbal for uplifting my confidence during my difficult times and being constant support throughout the journey. The encouragement and suggestions from Dr. Paragmoni Kalita, Dr. Bhaskor Jyoti Bora, Dr. Dhrubajyoti Kashyap help every aspect of my PhD days. I am lucky to have friends like Dr. Ujjwol Barman, Dr. Raushan Kumar, Dr. Mirzaul Hussian, Dr. Debarshi Mallick, Dr. Kiran Saikia, Dr. Jai Manik, and many more, who all have been my co-research-scholars in the Department of Mechanical Engineering. They have always provided me with great help and support. The motivation, and support from my friends- Mr. Manasjyoti Kalita, Mr. Manas Pratim Deva Sarma, Ms. Namita Das, Ms. Sanjukta Malakar, Mr. Jyotirmoy Kakati always inspired me during this time.

I feel fortunate to have received the love, cooperation and support from so many people in the completion of my PhD that it would not be possible for me to name them all. I thank all of them from the core of my heart.

Subhra Sankar Kalita

November, 2022

Indian Institute of Technology Guwahati

Abstract

This thesis is concerned with the study and analysis of various convergence-acceleration strategies that can help CFD practitioners to obtain their simulation results quickly. Convergence-acceleration can be achieved either by choosing a fast algorithm or by parallelizing the solution strategy. The multigrid method lies in the former category and is considered as one of the most powerful convergence-acceleration strategies. CUDA, which is a relatively new method for parallelizing CFD codes falls under the second category and as a part of the convergence-acceleration efforts embodied in this thesis, CUDA is used to parallelize LBM-based computations.

Multigrid (MG) methods, which are based on many levels of grids, have the ability to overcome the slow convergence characteristic of the single-grid methods for large-scale problems. This method has been established as a powerful tool for accelerating numerical convergence and, thus reducing the computational time. The two versions of multigrid methods are the linear multigrid method, known as the correction scheme applicable to linear equations and the nonlinear multigrid method also known as the full-approximation storage (FAS) multigrid method applicable to nonlinear equations. At the heart of computation of nonlinear problems, many times there lies a linearized problem. Therefore, this thesis work starts with numerical experiments for the 2D heat conduction problem using multigrid. The result shows that though successive overrelaxation (SOR) with the optimum relaxation parameter substantially accelerates Gauss-Seidel iterations, linear multigrid does it even better and its performance relative to SOR improves with grid refinement.

Apart from adopting the right strategies in the context of multigrid itself, we observe that the final computational time depends greatly on the selection of the original flow solution schemes and algorithms themselves. The streamfunction-velocity formulation is a relatively new method to solve the Navier-Stokes equations. The formulation allows Dirichlet boundary conditions to be used for both streamfunction and velocity, which are iteration-independent. This makes the method particularly amenable for multigrid acceler-

ation as unchanging boundary information is quickly passed to the interior. The accuracy of the results is established through a careful comparison exercise and faster convergence is indicated by time records of single and multigrid computations. Overall the combination of multigrid and streamfunction-velocity formulation affords a highly efficient solution procedure. To demonstrate the power of the solution procedure, a number of fluid-flow and heat-transfer problems are solved. The staggered cavity flow problem has been solved by the present computational procedure for the first time. Here, the laminar investigations are seen to be carried out at the highest value of Re = 3200 and for this Re, the present solution procedure obtains results in only 41 seconds with a speed-up of about 28. For the first time multigrid accelerated $\psi - V$ formulation has been used to compute multiple steady solutions in two- and four-sided cavity flows, demonstrating that multigrid has the ability to compute flows in those classes of problems that exhibit nonunique solutions. Even though the convergence is slower near the critical Re at which the flow bifurcates, the present method achieves fast convergence. In all the situations, solutions are obtained in a few seconds on a grid that provides accurate solutions, giving substantial convergence acceleration over a single-grid.

A part of the thesis is concerned with the lattice-Boltzmann simulation of various fluid flows and heat transfer problems. Self-written C/C++ codes have been developed for both the single-relaxation-time (SRT) and multiple-relaxation-time (MRT) lattice Boltzmann method (LBM). The comparative study between MRT-and SRT-based LBM indicates that in most cases, both models provide comparable solutions while dealing with laminar flow regimes. Nevertheless, MRT-LBM has the edge over SRT-LBM due to its better stability in solving complex flow problems. The LBM algorithm is highly amenable to acceleration through parallelization. We have used graphics processing unit (GPU) to parallelize the LBM code. A number of two- and three-dimensional fluid-flow and heat-transfer problems have been parallelized. Once two-dimensional LBM has been successfully parallelized using CUDA (Compute Unified Device Architecture), extending it to three-dimensional is found to be relatively straightforward. The acceleration of two-dimensional steady mixed convection heat transfer in a differentially heated square cavity for both aiding and opposing mechanisms has been successfully carried out using CUDA. It has been found that parallelized LB code computes about 8 times faster than the serial one and it signifies that LB code is very efficient in GPU environment. Hence, CUDA-based LBM computations are very efficient at a high value of Re that requires a finer grid. Finally, to gain

an understanding of how the multigrid-assisted LBM method stands with respect to other multigrid-assisted computations, a two-dimensional heat diffusion problem was solved using MG-LBM and it was found that with a four-level multigrid-LBM we can achieve a speed-up of around 269.

Overall, in the present work, a wide variety of fluid flow and heat transfer problems of varying complexities have been studied using multigrid accelerated techniques and the parallel lattice Boltzmann method. Both SRT and MRT versions of LBM are used to compute flow problems involving natural- and mixed convection. Furthermore, the scope of using the multigrid assisted $\psi - V$ algorithm has been extended to flow configuration having nonunique solutions. Sufficient care has been taken to obtain highly accurate solutions. All presented results are grid-independent. All the results in this thesis are produced using computer codes written in C/C++ (for parallel codes, CUDA was used) that have been carefully developed by the author from scratch and no help from any third party tool or library has been taken. The results are validated through an extensive comparison with the published results. The ability of multigrid to efficiently compute the wide variety of flows included in this thesis demonstrates not only the utility of the algorithm but also its robustness. Some of the applications deal with problems in which the multigrid method was not used earlier.

Contents

List of Figures

List of Tables

Nomenclature

c	Lattice speed ($\Delta x/\Delta t$)
c_θ, c_e, c_v	Relaxation parameters in moment space for D2Q5 MRT model
C	Diagonal relaxation matrix in momentspace of g
C_s	Speed of sound in lattice scale
e	Kinetic energy
$\mathbf{e}_i$	Discrete lattice velocity
f_i	Density distribution function
g_i	Temperature distribution function
$\mathbf{g}$	Acceleration due to gravity, m s^{-2}
Gr	Grashof number ($\mathbf{g}\beta\Delta T L^3/\nu^2$)
k	Thermal conductivity, W m^{-1} K^{-1}
L	Length of the cavity, m
$\mathbf{m}$	Moment function for density distribution function in vector form
$\mathbf{M}$	Transformation matrix for density distribution
$\mathbf{n}$	Moment function for temperature distribution function in vector form
$\mathbf{N}$	Transformation matrix for temperature distribution
Nu	Nusselt number
Pr	Prandtl number (ν/α)
$r_e, r_\epsilon, r_q, r_\nu$	Relaxation parametersin moment space for D2Q9 MRT model
$\mathbf{r}$	Position vector of arbitrary lattice node, $[x, y]$
$\mathbf{R}$	Diagonal relaxation matrix in moment space of f
Ra	Rayleigh number ($\mathbf{g}\beta\Delta T L^3/(\nu\alpha)$)
Re	Reynolds number ($V_0^* L/\nu$)
Ri	Richardson number ($\mathrm{Gr}/\mathrm{Re}^2$)
S_i	Total force term in LBM
t	Lattice time

T	Dimensional temperature, K
$\mathbf{u}$	Dimensionless velocity vector, $[u, v]$
V_0^*	Lid velocity, m s^{-1}
V_{lb}	Characteristic velocity in the lattice unit
w_i	Weighting factor

Greek Symbols

α	Thermal diffusivity, m^2s^{-1}
β	Volume expansion coefficient, K^{-1}
Δt	Lattice time step
Δx	Lattice spacing
ϵ	Square of kinetic energy
θ	Dimensionless temperature
τ_v	Viscous relaxation time
τ_θ	Thermal relaxation time
ψ	Dimensionless stream function
Ω	Collision matrix
μ	Coefficient of dynamic viscosity, Ns/m^2
ν	Coefficient of kinematic viscosity, m^2/s
ρ	Density of the fluid, kg m^{-3}

Subscripts and Superscripts

b	Bottom wall
c	Cold
h	Hot
i	Number of the distribution function
l	Left aall

m	Mean value
max	Maximum
min	Minimum
r	Right wall
ref	Reference value
t	Top wall
eq	Equilibrium

Abbreviations

ADI	Alternating Direction Implicit
AMG	Algebraic Multigrid
BC	Boundary Condition
BE	Boltzmann Equation
BGK	Bhatnagar-Gross-Krook
CFD	Computational Fluid Dynamics
CPU	Central Processing Unit
CS	Correction Scheme
CUDA	Compute Unified Device Architecture
DF	Distribution Function
DDF	Double-Distribution-Function
DmQn	m-Dimensional n-Velocity Model
D2Q5	Two-Dimensional Five-Velocity Model (for temperature DF in MRT)
D2Q9	Two-Dimensional Nine-Velocity Model
D3Q19	Three-Dimensional Nineteen-Velocity Model
FAS	Full Approximation Storage
FDM	Finite Difference Method
FEM	Finite Element Method
FVM	Finite Volume Method
GDDR	Graphics Double Data Rate
GPU	Graphics Processing Unit
GS	Gauss-Seidel

LB	Lattice Boltzmann
LBE	Lattice Boltzmann Equation
LBM	Lattice Boltzmann Method
LGA	Lattice Gas Automata
MD	Molecular Dynamics
MGM	Multigrid Method
MLUPS	Million Lattice Update Per Second
MRT	Multiple Relaxation Time
PDF	Particle Distribution Function
SM	Streaming Multiprocessor
SRT	Single Relaxation Time
WU	Work Units

Chapter 1

Introduction

1.1 Background

Computational fluid dynamics (CFD) mathematically models a physical phenomenon involving fluid flow and has been a major area of research in fluid mechanics for the last few decades. It is imperative that the computations should be accurate and can be efficiently carried out within a reasonable amount of time. As CFD is used to optimize experimental efforts, in the preliminary design cycles the numerical simulations are required to be run for a large number of times. In this context, convergence-acceleration algorithms become important, since they have the potential of cutting short the simulation time by months or even years. In most numerical procedures for solving partial differential equations, the analyst first discretizes the problem, thus choosing approximated algebraic equations on a finite-dimensional approximation space, and then devises a numerical process to (nearly) solve this huge system of discrete equations. Usually, no real interplay is allowed between the discretization and solution processes. The discretization process, being unable to predict the proper resolution and the proper order of approximation at each location, generates a mesh that is usually too fine. As a result, the algebraic system thus becomes unnecessarily large in size, since the local smoothness of the solution is not being properly exploited. On the other hand, the solution process fails to take advantage of the fact that the algebraic system to be solved does not stand by itself, but is actually an approximation to continuous equations, and therefore can itself be similarly approximated by other (much simpler) algebraic system [6].

The resolution, and hence numerical accuracy, can be improved by refining the grid supporting the solution. However, even if computer memory resources permit such a re-

finement, convergence towards the exact steady-state numerical solution tends to slow down dramatically with increasing grid density if conventional iterative algorithms such as Jacobi, Gauss-Seidel or ADI methods are used. This is, of course, a particularly serious constraint in the context of three-dimensional Navier-Stokes calculations. More advanced solvers, such as preconditioned conjugate gradient method and Stones strongly-implicit procedure [7], offer some advantages, but are not always uniformly efficient and suffer from difficulties of being adapted to vector computers.

In all the above-mentioned algorithms, the initial rate of convergence for the first few iterations is rapid, but the convergence slowly deteriorates, as the iterations progress. The main problem is the slow smoothing of long-wave (low-frequency) error components relative to a much faster erosion of short-wave (high-frequency) components, having wavelengths comparable to the mesh distance. As the grid is refined, these low-frequency errors dominate the overall rate of convergence. For a solution scheme to offer fast convergence, all error components must decay uniformly fast. There is an increasing amount of evidence that the multigrid method (MGM) goes some way towards meeting the aforementioned objectives. The method is peculiar in that it can use any one of the traditional relaxation methods to achieve convergence rates far above those possible with the same relaxation variant operating on a single grid. This is rooted in the fact that the role of the relaxation scheme is not primarily to reduce the error but rather to smooth it, so that it can be well supported by the grid. The basic idea of multigrid is to work not with a single grid, but with a sequence of grids ("levels") of increasing fineness, each of which may be introduced and changed in the process, and constantly interact with each other. The solution progresses along a sequence of coarsening and refining grids, with only a few relaxation sweeps applied on each level to remove high-frequency error components. Relaxation can be effectively performed by a simple point-explicit method, such as the Gauss-Seidel technique which can be easily vectorized. Similarly, inter-grid transfer (interpolation) processes, termed restriction and prolongation, can be effectively vectorized and parallelized, if careful attention is paid to the handling of boundary information.

There are two broad categories of MGM, namely, correction scheme (CS) and full approximation scheme (FAS). The former operates with corrections to the solution on coarse grids, which are eventually added to the absolute fine-grid solution. This method is applicable to linear systems only. In contrast, the latter method can be used to solve nonlinear problems by operating on the absolute solution (rather than its correction) on

all grid levels. The Other extensions of multigrid methods include techniques where no partial differential equation nor geometrical problem background is used to construct the multilevel hierarchy [1]. Such algebraic multigrid methods (AMG) construct their hierarchy of operators directly from the system matrix, and the levels of the hierarchy are simply subsets of unknowns without any geometric interpretation. Thus, AMG methods become true black-box solvers for sparse matrices. However, AMG is regarded as advantageous mainly where geometric multigrid is too difficult to apply. Before describing the multigrid method, the available methods of analysis are briefly surveyed in the present thesis. This survey is not intended to be comprehensive and is intended to indicate the motivation for the developments that are described in this chapter.

The transport equations of fluid flow and heat transfer can be formulated on three modes of scales, namely macroscopic, microscopic and mesoscopic scales based on the Knudsen number. At each level, there is a definite model to represent the fluid flow by satisfying the three fundamental physical principles: conservation of mass, conservation of momentum and conservation of energy. The finite difference method (FDM), finite volume method (FVM) [8, 9, 10] and finite element method (FEM) [11] are the conventional CFD techniques and they are based on a macroscopic scale. The presence of the nonlinear convective terms in these equations makes the solution algorithm complex. The instability of the algorithms is a key concern for these numerical schemes. Molecular dynamics (MD) [12] is a microscopic scale-based solution technique and is governed by Hamilton's equation. In MD, the time evolution of a system of interacting particles is predicted. However, the solution of a problem with detailed molecular/atomic-level information leads to massive data utilisation. Therefore, the main drawbacks of MD models are massive computational resources and data reduction. The mesoscopic scale connects the macro-scale and micro-scale through approximation to the Boltzmann equation of the kinetic theory of gases. The lattice Boltzmann method (LBM) [13, 14] is based on discrete equations derived from the Boltzmann equation. The basic idea of the LBM is to construct simplified kinetic type models that solve the time evolution of the particle distribution function by obeying the desired macroscopic equations. The use of simplified models to predict macroscopic fluid flows is based on the fact that the macroscopic dynamics of a fluid are the result of the collective behaviour of many microscopic particles in the system. Moreover, the macroscopic dynamics are not highly sensitive to the underlying details of microscopic physics. The last two decades have observed a dramatic improvement in the LBM as

a new effective and alternative approach to CFD. The LBM has achieved considerable success in simulating a wide variety of engineering problems especially in treating single component hydrodynamics, multiphase and multicomponent fluids [15, 16], convective heat transfer [17], flows through porous media [13, 18], magnetohydrodynamic [19] and many more complex flow problems [20].

Many modern computational systems that are used to solve CFD problems are parallel systems. They are either multicore CPUs (Central Processing Units) or many-core GPUs (Graphical Processing Units). In recent years, GPUs have been used a lot as a tool for computational acceleration. It is best suited for data-parallel computations. It doesn't have sophisticated flow control as opposed to the CPUs. GPUs are organized into highly-threaded several Streaming Multiprocessors (SMs). CUDA (Compute Unified Device Architecture) was introduced by NVIDIA in 2006. It is a general-purpose parallel programming architecture that harnesses the processing power of NVIDIA GPUs [2]. We can get around 10 times more peak floating-point calculations on a GPU compared to a CPU that comes for a comparable price. Also, GPUs have a very high memory bandwidth (the rate at which data is transferred from the memory to processors) compared to CPUs. For example, Tesla K40 comes with 288GB/s of memory bandwidth, 12GB of memory size and 2880 CUDA cores. Owing to their high computing power, modern GPUs are a very convenient CFD tool for analyzing large problems. It will be seen later that LBM computations go very well with GPU as LBM has an inherent parallelizability.

1.2 Literature Review

Fedorenko [21, 22] is the originator of the idea of multigrid. However, initially his idea did not draw much attention. Later on, interest in this technique was reawakened by Brandt and he gave a sound mathematical foundation in his path-breaking paper [6] in 1977. He was the first to demonstrate the practical efficiency and generality of multigrid methods. He discretized the governing equations on each grid and introduced the formation of the coarse grid equations. Then he demonstrated the interpolation techniques to transfer the fine grid residuals to the coarse grid and the coarse grid corrections to the fine grid. Since then the multigrid technique has become one of the most extensively used convergence-acceleration tools in CFD. At about the same period as Brandt [6], Hackbusch [23] independently rediscovered the multigrid method. Hackbusch [24, 25] obtained the mathematical convergence proofs for the MGM in the case of elliptic boundary-value

problems. The potential of the method has been realized and demonstrated in the solution of various elliptic model problems in [26]. An efficient multigrid algorithm for the Poisson equation has been clearly described in [27].

Detailed analysis of simple model problems, along with some interesting historical notes is contained in the review article of Stuben and Trottenberg [28]. Multigrid methods have been analysed, applied and generalized in many ways since their introduction, and are gradually becoming recognized as a powerful tool in applied mathematics. A package of multigrid solvers for solving a simply structured elliptic boundary value problems was developed by Foester and Witsch [29]. They have shown superior performance of the multigrid method to that of fast direct solvers. Fuchs [30], in 1981, developed a numerical solution of 2D Navier-Stokes equation with the emphasis on multigrid. He examined the smoothing properties of different relaxation schemes as well as the effect of stretched grids on multigrid performance. The benefits of using multigrid methods in solving the Poisson and Navier-Stokes equations were demonstrated by Miller and Schmidt [31].

Considerable efforts have been made by researchers toward the development of different computational fluid dynamics (CFD) techniques, for the solution of the incompressible N-S equations. For testing the accuracy and performance of newly developed numerical methods, for computing 2D incompressible viscous flows, the two-dimensional lid-driven cavity flow has been extensively used as a benchmark problem [4, 32, 33, 34, 35]. Not only many of the incompressible flow phenomena can be observed in the driven cavity, but it is quite relevant to industrial applications [36, 34]. Kuhlmann and other researchers extended it to the two-sided lid-driven cavity [37, 38, 39, 40, 41]. Of late, researchers are taking an interest in the double lid-driven staggered cavity which was first introduced by Hinatsu and Ferziger [42]. Zhou et al. [43] used the discrete singular convolution (DSC) method for the solution of the two-sided lid-driven staggered cavity, and obtained benchmark quality solutions for Reynolds numbers (Re) between 50 and 3200, for the antiparallel motion of the top and bottom walls. Later on, Nithiarasu and Liu [44] studied the same problem using an artificial compressibility-based characteristic-based split scheme. Tekić et al. [45] employed the lattice Boltzmann method (LBM) to simulate the flows in the lid-driven staggered cavity, for both parallel and antiparallel motions. Kalita and Gogoi [46] carried out their analysis of the staggered-cavity flow with a high-order compact approach for both the parallel and antiparallel motions of the walls. Zhou [43] has mentioned that different fluid flow features like symmetry, elliptic instability and numerically induced symmetry

breaking can be observed with the staggered cavity problem.

The one-sided lid-driven cavity problem was extended by Kuhlmann et al. [37, 38] to a two-sided problem, where the flow is driven by the parallel or antiparallel motion of two facing walls. The facing walls could be either the left and right walls or the upper and lower walls. At low Reynolds number, the flow consists of separate co- or counter-rotating primary vortices that form adjacent to each moving wall. At higher Reynolds numbers, instabilities arise in the flow due to the interaction between the two primary vortices. Moreover, their results showed that multiple flow solutions may exist, depending on the cavity aspect ratio and the value of the Reynolds number [3]. Alleborn et al. [36] and Luo and Yang [47] carried out numerical simulations of two-sided lid-driven cavity flow with temperature gradient and accompanied by heat and mass transport. More recently, the investigation carried out by Wahba [48, 49] revealed that the multiplicity of flow states exists for two-sided non-facing lid-driven cavity flow and four-sided lid-driven cavity flow. He found that the critical Reynolds number for the two-sided lid-driven cavity is 1073 and for the four-sided lid-driven cavity is 129.

In the past two decades, the biharmonic form of the $\psi - V$ formulation is gaining popularity as it has a number of advantages which are demonstrated by various authors [50, 51, 52, 53, 54, 55, 56, 57]. For instance, the computational stencil is a nine-point compact one and the discretized equation can achieve a fourth-order accuracy [58, 50, 54]. The implementation of the $\psi - V$ formulation of the N-S equation yields very accurate results as reported in the works of many authors [54, 56, 59, 60, 61, 62, 63, 64, 65].

For more than a decade, an explicit numerical scheme for the discrete Boltzmann equation, the so-called Lattice-Bathnagar-Gross-Krook(LBGK) approach has been used quite successfully to obtain approximate solutions for weakly compressible and incompressible Navier-Stokes problems. To explore the benefits of state-of-the-art numerical techniques for the solution of the stationary discrete Boltzmann equation, Tölke et al. [66] in the year 2002 introduced a nonlinear multigrid solution approach discretized by an implicit second-order Finite Difference scheme. The procedure uses prolongation and restriction on a geometrical basis, so the mapping from one level to another is only possible for geometries of moderate complexity. They obtained drastically improved efficiency in comparison to the widely used Lattice-Bathnagar-Gross-Krook (LBGK) approach. Dimitri J. Mavriplis [67] in 2006 investigated efficient solution strategies for the steady-state lattice Boltzmann equation. He has adopted a nonlinear multigrid approach, which makes use of the non-

linear LBE time-stepping scheme on each grid level. Rapid convergence to steady-state is achieved by the nonlinear algorithm, resulting in one or more orders of magnitude increase in solution efficiency over the LBE time-integration approach. In 2014, Patil et al. [68] presented a new solver for second-order elliptic partial differential equations (PDEs) based on the lattice Boltzmann method (LBM) and the multigrid (MG) technique. It was observed that the use of a high-order stencil (for smoothing) improves convergence and accuracy for an equivalent number of smoothing sweeps. In terms of grid scheduling, the W-cycle was found to be more efficient than V-cycle. It was found that the parallel MG-LBM exhibits good speed-ups that are quite similar to those for the classical single-grid LBM for up to 16 processors tested on a multi-core cluster. Such a parallel solver can thus significantly augment (via specialized hardware/software) the convergence acceleration already achieved by the use of multigrid (via an optimal numerical scheme) with the LBM. In 2019, Armstrong and Peng [69] extended the multigrid LBM to MRT (multiple relaxation time) LBM. They validated the new method using Poiseuille flow and then extended it to three dimensions.

It is found from the literature that the structure of the LBM algorithm is such that it can be very easily parallelized using a GPU. In 2003 Lie et al. [70] made the first attempt to use GPU for the acceleration of LBM. Fan et al. [71] applied LBM on a 32 nodes cluster made of NVIDIA GeForce 5800 ultra in 2004. At that time, as CUDA was unavailable, they used OpenGL to map the lattice Boltzmann variables to two-dimensional textures. With 32 nodes, Fan et al. found an efficiency in their implementation of 49.2 Million Lattice Update Per Second (MLUPS). An NVIDIA 8800 Ultra graphics card was used by Tölke [72] to implement a two-dimensional LBM code. He found a speed up (ratio between GPU time to CPU time) of about 23X for the same test case. In 2008, Tölke and Krafczyk [73] implemented a three-dimensional LBM with D3Q13 lattice model on GPU. However, studies concerning mixed convection with CUDA implementation are very limited.

In 2013, Vanka [74] has provided a lucid discussion on the history of CFD on GPUs and different issues and advantages of using GPU to solve fluid flow problems. In one of his own experimentations, the author has solved Large Eddy and Direct Numerical Simulations on GPU and achieved speed up of more than a factor of 20. Jin et al. [75] solved a three-dimensional flow of power-law fluids in a driven cube with the help of a GPU. The authors have used NVidia Tesla K20 GPU and CUFLOW solver which uses an explicit algorithm for momentum equations and a red-black SOR algorithm for the pressure-Poisson equation. Since, both of these methods are data parallel algorithms and can easily be mapped to a

GPU, they have observed a gain in computational speed upwards of 18 over serial running on an Intel Xeon E5-2650v2 2.6-GHz CPU.

The present work studies the viability of CUDA implementation on the mixed convection flow phenomenon in a square cavity.

1.3 Motivation and Objectives

In computational fluid dynamics (CFD), the need for obtaining a quickly converged solution cannot be overemphasized. For example, in the design of aerodynamic objects such as aerofoils, missiles, etc., a computer code needs to be run over and over again with various input parameters, to obtain the final geometry of the object that is expected to give the desired performance. It is easily seen that if convergence acceleration makes it possible for a CFD result to be obtained in a day instead of a week, and if this result needs to be obtained multiple times in the design cycle of an object, such acceleration has the potential of saving considerable time and money. Not only in aerodynamics but in other design areas where the governing equations of fluid dynamics and heat transfer are numerically solved repeatedly, obtaining a faster-converged solution is of seminal importance. It is mainly for this reason that from the nineteen-eighties onwards convergence acceleration of CFD code has assumed particular interest and importance as a CFD activity.

The present work can be viewed to be an effort in this direction. One area of investigation here is the multigrid method, which is considered as one of the most general convergence-acceleration devices. It is general because it can be used to accelerate the convergence of CFD codes based on the finite difference method, finite volume method and also finite element method. Apart from developing new ideas and techniques in the context of multigrid itself, we observe that the final computational time depends greatly on the selection of the original flow solution schemes and algorithms themselves. In this context, the present thesis applies multigrid to work in conjunction with a fluid flow algorithm that is seen to be used somewhat sparingly, which happens to be the streamfunction-velocity method of two-dimensional flow computation. Also, there is another area where multigrid is in the nascent stage of application - the lattice Boltzmann method (LBM). Some efforts will therefore be devoted in this thesis to the application of multigrid to LBM as well. This is to gain an understanding of how the multigrid-assisted LBM method stands with respect to other multigrid-assisted computations. The importance of very fast multigrid-assisted 2D computations can also be realised when we see that these can be many times used for

obtaining a good initial guess for some three-dimensional computations.

In the broad context of obtaining a faster solution for desired purposes, parallel-code computations are frequently carried out. In this context, of late CUDA is gaining importance. Therefore, a part of the convergence-acceleration efforts embodied in this thesis is also directed to the application of CUDA to LBM-based computations.

Overall, the main thrust of this thesis is to gain experience (so as to share with the readers) in the acceleration of various flow computations and to compare their relative merits whenever possible so as to be of assistance to CFD practitioners in their design effort. Based on the above context, the objectives of the present investigation are laid out as follows:

1. Developing a solver based on multigrid accelerated streamfunction-velocity formulation to solve incompressible fluid flow problems using C/C++.

2. To demonstrate the ability of the above formulation to capture multiple steady solutions for certain flow configurations.

3. Extension of the multigrid accelerated streamfunction-velocity formulation to solve heat transfer problems with remarkable ease and accuracy.

4. Implementation of C/C++ codes based on both single-relaxation-time (SRT) and multiple-relaxation time (MRT) lattice Boltzmann model to realise their full scope of application in different fluid flow and heat transfer problems for a wide range of Reynolds (Re), Rayleigh (Ra) and Prandtl numbers (Pr).

5. Development of parallel lattice Boltzmann code for two- and three-dimensional fluid flow and heat transfer problems with the help of CUDA (Compute Unified Device Architecture) to run on a GPU (Graphics Processing Unit) device.

6. To examine the feasibility of integrating the full-approximate storage (FAS) multigrid with the lattice Boltzmann method.

Furthermore, the current work proposes a computationally efficient solution procedure that also produces a highly accurate solution. The geometric multigrid is used to achieve computational efficiency and the $\psi - V$ formulation is chosen to obtain high accuracy. It may be noted that the possibility of application of Dirichlet boundary condition for just one variable, i.e. streamfunction plays an important role in its inherent accuracy. For these

reasons, we use this combination to compute various fluid flow and heat transfer problems with remarkable ease and accuracy. For some of the problems studied, the present solution procedure has been used for the first time. Also, we have studied for the first time how this accelerated solution procedure can be easily extended to those problems which have multiple steady solutions. We have shown that the Gauss-Seidel (GS) procedure, despite being known for relatively slow convergence, can be very effective as a geometric multigrid smoother and that the combination of $\psi - V$ formulation and multigrid results in a powerful computational procedure.

1.4 Organization of the Thesis

The thesis is organized into seven chapters. The contents of the chapters are briefly described below.

Chapter 1 presents a brief introduction, the relevant literature survey, the motivation of the thesis and its structure.

Chapter 2 contains the brief background of the multigrid method, lattice Boltzmann method and GPU parallelisation using CUDA. The different terminologies associated with multigrid theory, algorithms of various multigrid cycles, the difference between linear and nonlinear multigrid methods and the performance of the multigrid method for linear equations are explained. In this chapter, the implementation techniques of both SRT and MRT lattice Boltzmann method, and different boundary condition treatments are described. Furthermore, a brief introduction to heterogeneous parallel computing and the CUDA programming model is elaborated here.

Chapter 3 presents the essence of streamfunction-velocity formulation and its coupling strategies with the multigrid method. The performance of this coupled formulation applied to the lid-driven cavity and staggered lid-driven cavity is discussed. The accuracy and speed of multigrid accelerated streamfunction-velocity formulation by obtaining the multiple stable solutions for a two-sided non-facing lid-driven cavity and a four-sided lid-driven cavity is demonstrated.

Chapter 4 extends the streamfunction-velocity formulation coupled with multigrid to solve natural convection and mixed convection in a square cavity.

Chapter 5 simulates the isothermal and thermal benchmarks cavity flow problems using an LBM-C code based on both SRT- and MRT- lattice Boltzmann models. The satisfactory agreement of present results with the previously published results shows its

competence and gives us the confidence to parallelize our code with the help of a graphics processing unit (GPU). The performance gain by two- and three-dimensional lattice Boltzmann method on implementation on GPUs using CUDA C programming framework is studied with the help of heat diffusion, two- and three-dimensional lid-driven cavity flow, two-dimensional natural convection and the mixed convection flow in a differentially heated lid-driven square cavity filled with a low Prandtl number (Pr) fluid.

Chapter 6 discusses the strategy to accelerate the lattice Boltzmann method with the help of multigrid. The acceleration is demonstrated with the help of a two-dimensional heat-diffusion problem.

Chapter 7 includes a summary of the findings of the present thesis and possible extensions of current work for uncovering the unseen facts.

Chapter 2

Multigrid, Lattice Boltzmann Method and CUDA in Brief

2.1 Multigrid Method

This chapter introduces the essential principles of the multigrid method and emphasises its potential for solving computational fluid dynamics problems. At the heart of most mathematical models, arising from the engineering, physical or applied mathematical problems, lies the partial differential equations which give rise to extensive computations. With the increase in the complexity of the problems, the number of grid points required to capture the physics increases. The advancement in the computational power of modern computers, in both speed and memory, has facilitated the exploration of more challenging computations of various branches of science and mathematics, considered too large to compute a few years ago. A number of new techniques for solving fluid flow problems have been developed in the last few decades. These advancements have also brought many new challenges to light. The conventional solution procedures on a single-grid suffer from slow convergence for problems involving a large number of grid points. This slow convergence severely affects the three-dimensional problems, where a nominal grid refinement in any coordinate direction leads to a tremendous increase in the total number of grid points. The main reason is that many standard iterative methods like Jacobi, Gauss-Seidel etc. have the smoothing property. They can quickly eliminate the high-frequency error components but fails to smooth out the low-frequency error components at the same rate. The "multigrid method" is very effective on all error components. Multigrid uses a hierarchy of grids with different levels in order to decay all the error components uniformly fast. Thus,

the multigrid methods have overcome the slow convergence characteristic of the single-grid methods for large-scale problems, and hence they have been established as a powerful numerical convergence acceleration tool. The multigrid method has its origin in the early work of Fedorenko [21, 22] and has been extensively studied and developed by Brandt [6]. Since then the popularity of the multigrid method has been increasing enormously, especially amongst the CFD practitioners. Further details on the multigrid method can be found in [1, 27, 25]. The present chapter outlines the main principles and the practical utility of multigrid methods. Also, the relative performance of the multigrid method with Gauss-Seidel and successive overrelaxation method in the numerical solution of the Laplace equation is examined in some detail.

2.1.1 Linear Multigrid

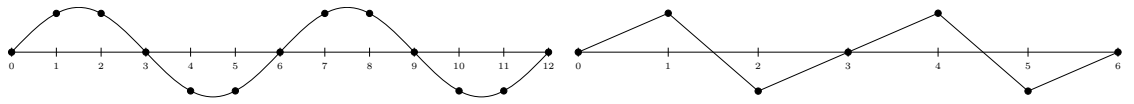
The multigrid method is one of the most effective ways to accelerate convergence and is a very fast linear iterative solver based on the multilevel paradigm. This method iteratively solves a system of discrete (finite-difference or finite-element or finite-volume) equations on a given grid, by constant interactions with a hierarchy of coarser grids, taking advantage of the relation between different discretizations of the same continuous problem. The typical application of multigrid is in the numerical solution of elliptic partial differential equations in two or more dimensions. Multigrid algorithms are not difficult to program if the various grids are suitably organized. There are two basic principles of multigrid approach [27]:

- Error smoothing is one of them. Many classical iterative methods (Gauss-Seidel etc.) have a strong smoothing effect on the error of an approximation if the iterative methods are appropriately applied to discrete elliptic problems.

- The other principle is that a smooth quantity on a fine grid can be approximated on a coarser grid without any essential loss of information. For instance, the error that becomes smooth on a fine grid after some iteration steps can be approximated by a suitable method on a coarser grid. Computation on a coarse grid is substantially cheaper than on a fine grid.

Figure 2.1 demonstrate that the error components appearing smooth on a fine grid become more oscillatory when the error is projected on a coarse grid.

Assume that a relaxation scheme on the fine grid with N = 12 shown in figure 2.1a removes the oscillatory components of the error leaving a relatively smooth error. These



(a) *Fine Grid* (b) *Coarse Grid*

Figure 2.1: *Smooth error in a fine grid appears oscillatory in a coarse grid*

smooth components are then projected directly to the coarser grid with N = 6 and the smooth wave on the fine grid looks more oscillatory on the coarser grid (figure 2.1b). This suggests that when relaxation begins to stall, signalling the predominance of smooth error modes, it is advisable to move to a coarser grid, on which those smooth error modes appear more oscillatory and relaxation will be more effective.

Multigrid methods can be of two types: a) Geometric multigrid, and b) Algebraic multigrid. Each of them can again be classified into two categories, namely, the correction scheme (CS) and the full approximation storage (FAS) method.

The CS is applicable to linear equations only; they operate with corrections to the solution on coarse grids, which are eventually added to the absolute fine-grid solution. The FAS is used for solving non-linear problems. It solves equations for approximations to the solutions rather than for corrections at each grid. The three important steps that should be carried out in order to apply multigrid using a coarse grid are:

1. Transfer of the solution and residuals from the fine to the coarse grid.

2. Calculation of new solution (corection/approximation) on the coarse grid.

3. Solution interpolation from the coarse to the fine grid.

It is important to introduce two terms that are frequently used hereafter - restriction and prolongation. The restriction is defined as the transfer of variables from the fine to coarse grid denoted by the operator I_h^{2h} and prolongation is defined as the transfer of variables from the coarse to fine grid denoted by the operator I_{2h}^f with the suffixes h and $2h$ standing for fine and coarse grids.

Consider a two-dimensional Poisson equation

$$\nabla^2 u = f \tag{2.1}$$

where f is the source term and $\mathbf{u}$ is the solution vector. Using finite difference method to

discretize, the equation (2.1) can be written as

$$A\mathbf{u} = \mathbf{f}. \tag{2.2}$$

Equation 2.2 denotes a system of linear equations, where A is the discretization operator. Let us assume $\mathbf{u}$ and $\mathbf{v}$ as an exact solution and a computed approximation to $\mathbf{u}$ respectively, generated by some iterative method. The bold symbols $\mathbf{u}$ and $\mathbf{v}$ represent vectors and u_j and v_j denote the j^{th} components of these vectors. $\mathbf{u}^h$ and $\mathbf{v}^h$ mean that $\mathbf{u}$ and $\mathbf{v}$ are associated with a particular grid, say Ω^h. The algebraic error is the difference between the exact and the approximate solution of equation (2.2) and is computed as

$$\mathbf{e} = \mathbf{u} - \mathbf{v}. \tag{2.3}$$

The error $\mathbf{e}$ is a vector and any of the standard vector norms can be used to measure its magnitude.

The *residual* vector $\mathbf{r}$ is a measure of how well $\mathbf{v}$ approximates $\mathbf{u}$ and is given by

$$\mathbf{r} = \mathbf{f} - A\mathbf{v} \tag{2.4}$$

By the uniqueness of the solution, $\mathbf{r} = \mathbf{0}$ if and only if $\mathbf{e} = \mathbf{0}$. However, it may *not* be true that when $\mathbf{r}$ is small in norm, $\mathbf{e}$ is also small in norm. [1]

Subtracting equation (2.4) from equation (2.2) yields

$$A(\mathbf{u} - \mathbf{v}) = \mathbf{r} \tag{2.5}$$

using equation (2.3),

$$A\mathbf{e} = \mathbf{r} \tag{2.6}$$

This relationship is known as the *residual equation*. It says that the error satisfies the same set of equations as the unknown $\mathbf{u}$ when $\mathbf{f}$ is replaced by the residual $\mathbf{r}$. The residual equation plays a vital role in multigrid methods.

We can now anticipate, in an imprecise way, how the residual equation can be used to great advantage. Suppose that an approximation $\mathbf{v}$ has been computed by some method. It is easy to compute the residual $\mathbf{r} = \mathbf{f} - A\mathbf{v}$. To improve the approximation $\mathbf{v}$, we might solve the residual equation for $\mathbf{e}$ and then compute a new approximation using the

definition of the error

$$\mathbf{u} = \mathbf{v} + \mathbf{e} \tag{2.7}$$

2.1.1.1 Restriction and Prolongation

There are two types of intergrid transfer operations used in multigrid. In one we need to move the vectors from a fine grid to a coarse grid and are generally called as *restriction* operators and are denoted by I_h^{2h}. The most commonly used restriction operator is *injection*. It is defined by $I_h^{2h}\mathbf{v}^h = \mathbf{v}^{2h}$, where

$$v_{i,j}^{2h} = v_{2i,2j}^{h}. \tag{2.8}$$

In a finite difference method, all nodes on the fine grid coincide with those of the coarse grid and therefore the transfer of variables from the fine to the coarse grid by injection is achieved by directly copying the values of the coarse grid variables from the corresponding fine grid points. The arrangement of variables is shown in figure 2.2. There is one more

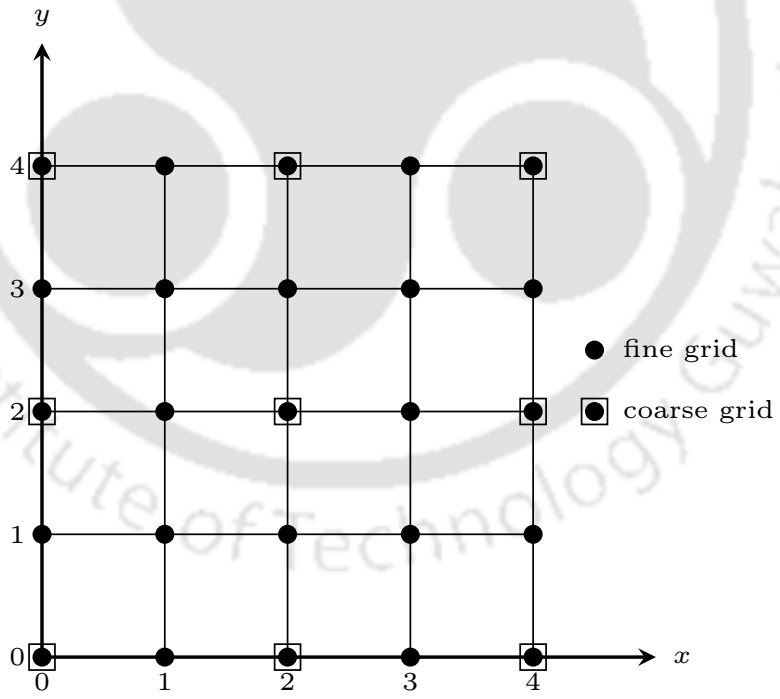


Figure 2.2: *Location of fine and coarse grid points for a 2D square domain.*

restriction operator called *full weighting* which is defined by $I_h^{2h}\mathbf{v}^h = \mathbf{v}^{2h}$ and for one-dimensional problem it is represented as

$$v_j^{2h} = \frac{1}{4}\left(v_{2j-1}^h + 2v_{2j}^h + v_{2j+1}^h\right), \quad 1 \le j \le \frac{n}{2} - 1 \tag{2.9}$$

If Red-Black Gauss-Seidel is used as smoother than instead of injection, *full weighting* should be used as restriction operator [76]. The *full weighting* operator in two dimensions is the averaging of the fine-grid nearest neighbours and is given by:

$$v_{i,j}^{2h} = \frac{1}{16}\Big[v_{2i-1,2j-1}^{h} + v_{2i-1,2j+1}^{h} + v_{2i+1,2j-1}^{h} + v_{2i+1,2j+1}^{h} + 2\left(v_{2i,2j-1}^{h} + v_{2i,2j+1}^{h} + v_{2i-1,2j}^{h} + v_{2i+1,2j}^{h}\right) + 4v_{2i,2j}^{h}\Big], \qquad 1 \le i,j \le \frac{n}{2} - 1. \tag{2.10}$$

The second type of intergrid transfer operations involves the transfer of variables from the coarse to the fine grid by the linear interpolation and is called *prolongation*. The interpolation operator is denoted by I_{2h}^{h} through which the corrections obtained from the coarse-grid solutions are passed onto the fine grid to update the solution on it. If we let $I_{2h}^{h}\mathbf{v}^{2h} = \mathbf{v}^{h}$, then the components of $\mathbf{v}^{h}$ are given by

$$v_{2i,2j}^{h} = v_{i,j}^{2h}, \qquad 0 \le i,j \le \frac{n}{2} - 1 \tag{2.11}$$

$$v_{2i+1,2j}^{h} = \frac{1}{2}\left(v_{i,j}^{2h} + v_{i+1,j}^{2h}\right), \tag{2.12}$$

$$v_{2i,2j+1}^{h} = \frac{1}{2}\left(v_{i,j}^{2h} + v_{i,j+1}^{2h}\right), \tag{2.13}$$

$$v_{2i+1,2j+1}^{h} = \frac{1}{4}\left(v_{i,j}^{2h} + v_{i+1,j}^{2h} + v_{i,j+1}^{2h} + v_{i+1,j+1}^{2h}\right), \tag{2.14}$$

At even-numbered fine grid points both row and column wise (equation (2.11)) values of $v_{2i,2j}^{h}$ are transferred directly from coarse to fine grid (figure 2.2) because these points coincide with one another. At odd-numbered columns and even-numbered rows of fine grid points like $(1,0)$, $(1,2)$, etc., the values of $v_{2i+1,2j}^{h}$ are calculated by taking the average of adjacent coarse-grid values (equation (2.12)). Similarly at odd-numbered rows and even-numbered columns of fine grid points such as $(0,1)$, $(2,1)$, etc., the values of $v_{2i,2j+1}^{h}$ are calculated using equation (2.13). In the remaining odd-numbered (both row and column) fine grid points the values of $v_{2i+1,2j+1}^{h}$ are calculated using equation (2.14) [1]. It is very important to transfer the corrections accurately back to the fine grid because if interpolation works well the correction of the fine grid solution would be effective.

2.1.1.2 Linear Two-Grid Correction Scheme

A two grid correction scheme algorithm for linear problems consists of smoothing on the fine grid, approximation of the required correction on the coarse grid, prolongation of the

coarse grid correction to the fine grid and again smoothing on the fine grid (figure 2.3). Now having a well-defined way to transfer vectors between fine and coarse grids, we define the two-grid correction scheme as below.

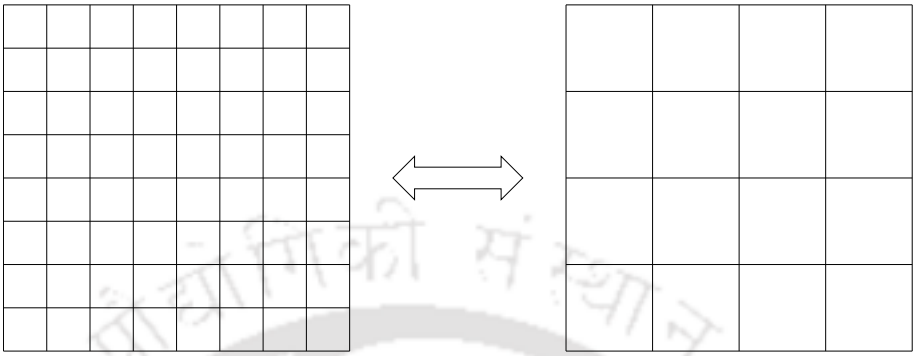


Figure 2.3: *Grid arrangement in two-level scheme*

$$\mathbf{v}^h \leftarrow MG\left(\mathbf{v}^h, \mathbf{f}^h\right)$$

- Iterate $A^h\mathbf{u}^h = \mathbf{f}^h$, ν_1 times on the fine grid (Ω^h) with initial guess $\mathbf{v}^h$. A low value for ν_1 is used and usually it is kept between 1 to 3.
- Compute the residual $\mathbf{r}^h = \mathbf{f}^h - A^h\mathbf{v}^h$ and store it at each point of the fine-grid. Restricted the residual to the coarse grid by $\mathbf{r}^{2h} = I_h^{2h}\mathbf{r}^h$ to convert the low-frequency error to high-frequency error that can be smoothed on the coarse grid.
- Solve the correction equation $A^{2h}\mathbf{e}^{2h} = \mathbf{r}^{2h}$ on the coarse grid, Ω^{2h} using zero initial guess, i.e., $\mathbf{e}^{2h} = \mathbf{0}$.
- Prolongate the correction obtained on the coarse-grid to the fine grid by $\mathbf{e}^h = I_{2h}^h\mathbf{e}^{2h}$.
- Correct the fine-grid approximation by $\mathbf{v}^h \leftarrow \mathbf{v}^h + \mathbf{e}^h$.
- Relax ν_2 times the fine-grid equation $A^h\mathbf{u}^h = \mathbf{f}^h$ on Ω^h with corrected values $\mathbf{v}^h$. ν_2 value is also kept low and is normally between 1 and 3.

It is necessary to repeat the above two-level cycle until desired convergence is achieved.

2.1.1.3 Multi-level Correction Scheme

The two-grid correction scheme as outlined above consists of only one coarse grid. In order to get significant improvements in computational economy use of additional levels is

required. The extension to an additional level is straightforward in principle but requires careful interpretation of the multigrid concept in order to correctly implement the procedure. If multiple coarse grids are present, the extension can be made recursively, i.e., steps 2 and 3 of the section 2.1.1.2 are repeated until the coarsest grid is reached. In this way, the residual of the finest grid controls the solution on all coarser grids. Therefore, we can apply the two-grid correction scheme to the residual equation on Ω^{2h}, which means relaxing there and then moving to Ω^{4h} for the correction step. We can repeat this process on successively coarser grids until a direct solution of the residual equation is possible. Once the coarsest grid relaxation is over steps 4-6 in section 2.1.1.2 are successively repeated until the finest grid is reached again.

This procedure is known as the V-cycle in which the calculation proceeds from the finest grid down to the coarsest and back to the finest. One sweep of the multigrid method, from the finest to the coarsest grid and back to the finest grid again is called a cycle. There are various multigrid cycles, namely, V-cycle (figure 2.4(a)), W-cycle (figure 2.4(b)), S-cycle (also called the saw-tooth cycle), Brandt's method (also called R-cycle) and FMG (full multigrid V-cycle) (figure 2.4(c))[33]. Only V-, W- and FMV-cycles are shown in figure 2.4 with grid spacing $h, 2h, 4h, ... = 2^{l-1}h$ where l represents number of levels.

Among the cycles, V- and W-cycles are widely used. W-cycles are usually more robust and they are easier to analyze in the classical multigrid convergence theory. However, they are expensive, and in local refinement applications (or in 1D) they may fail to have optimal work count [77]. W-cycles are especially expensive in parallel algorithms when frequent coarse grid visits lead to poor processor utilization. A full multigrid method starts solving the problem on the coarsest grid and uses that solution as the initial approximation for the next finer level. This process continues until it reaches the finest level where the solution to the problem is required. It should be noted that the order of interpolation has to be chosen in an appropriate way. Either V- or W-cycles can be used in the full multigrid method. The optimal choice of the number of relaxation sweeps required at any grid level is not explicitly available. Various experiments have to be performed to find it. In the present thesis, V-cycle has been used in all the cases.

2.1.1.4 Multigrid Implementation to Heat Conduction Problem

As a first step towards learning the multigrid method, this technique is implemented to obtain the solution to Laplace's equation with Dirichlet boundary conditions. Laplace

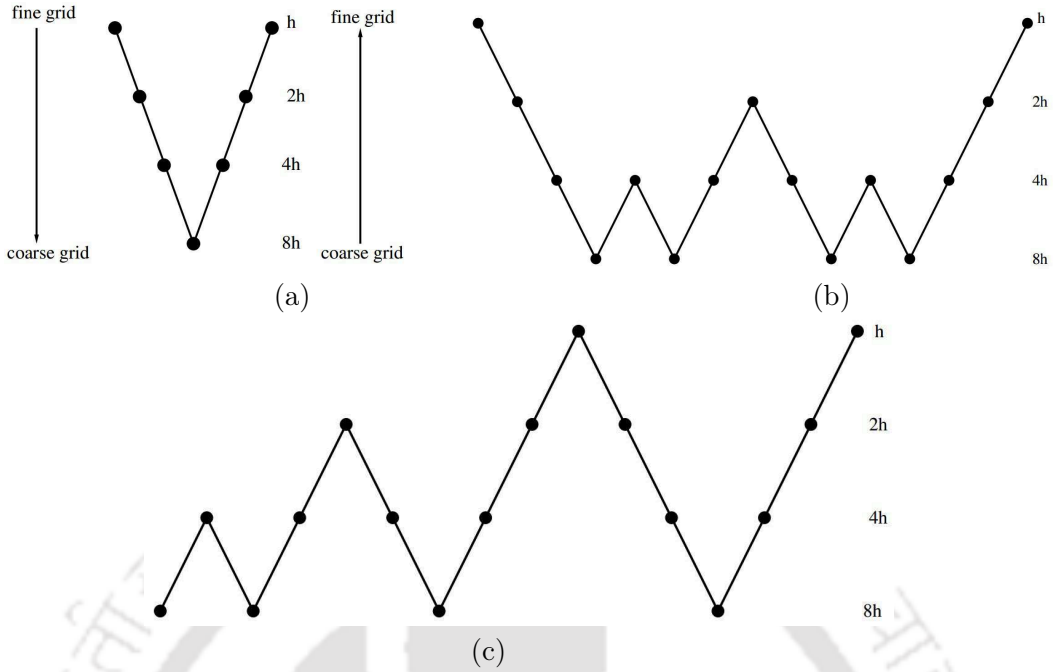


Figure 2.4: *(a) V-Cycle (b) W-Cycle (c) FMG-Cycle.*[1]

equation is the governing equation for the two-dimensional heat conduction problem

$$\frac{d^2\phi}{dx^2} + \frac{d^2\phi}{dy^2} = 0 \tag{2.15}$$

here ϕ is unknown variable to be found out. The computational effort for the multigrid scheme is compared with that required for the simple Gauss-Seidel scheme and for the Gauss-Seidel scheme with optimum over-relaxation. A square domain is considered (figure 2.5) with the boundary conditions

$\phi = 50 \quad$ at $\quad x = 0$

$\phi = 100 \quad$ at $\quad y = 1$

$\phi = 150 \quad$ at $\quad x = 1$

$\phi = 200 \quad$ at $\quad y = 0$

The Laplace equation is numerically solved using the Gauss-Seidel method, Gauss-Seidel with SOR and the multigrid method with Gauss-Seidel as the smoother. In the multigrid method, it is convenient to successively halve the number of uniformly distributed grid points on each side of the grid to obtain the coarser grid levels. Hence grids of 9×9, 17×17, 33×33, 65×65 and 129×129 is used. The over-relaxation factors computed for

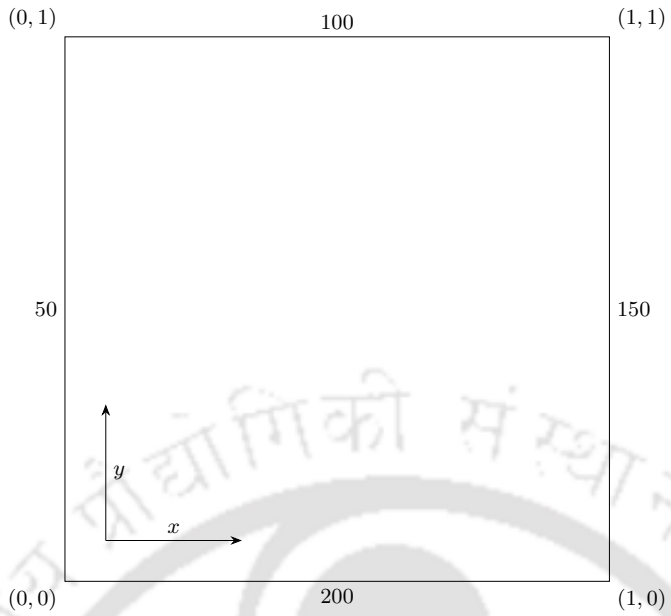


Figure 2.5: *Square Domain with Boundary Condition.*

these grids are 1.45, 1.67, 1.82, 1.91 and 1.95 respectively. These will be used with the Gauss-Seidel scheme without multigrid.

Over-relaxation is not used with multigrid, as it doesn't seem to improve the convergence rate. The computational effort, i.e. work unit is found out in terms of equivalent fine-grid sweeps. The convergence parameter used in the calculations is the maximum change in the computed variable (u or Δu) between two successive sweeps divided by the maximum value of the dependent variable on the boundary. (Here maximum boundary value is 200). Convergence is declared when this parameter is less than 10^{-5}.

An attempt is also made to observe the variation of the work unit with the number of sweeps. To gain experience with the different multigrid cycles, code is developed for V-cycle, W-cycle and FMG-cycle and a comparison is carried out.

The standard central-difference discretization for equation (2.15) yields,

$$\frac{\phi_{i-1,j} - 2\phi_{i,j} + \phi_{i+1,j}}{(\Delta x^2)} + \frac{\phi_{i,j-1} - 2\phi_{i,j} + \phi_{i,j+1}}{(\Delta y^2)} = 0 \tag{2.16}$$

Let an operator A be defined such that $A\phi_{i,j}$ is the standard difference representation; then equation (2.16) can be written as

$$A\phi_{i,j} = \frac{\phi_{i-1,j} - 2\phi_{i,j} + \phi_{i+1,j}}{(\Delta x^2)} + \frac{\phi_{i,j-1} - 2\phi_{i,j} + \phi_{i,j+1}}{(\Delta y^2)} = 0 \tag{2.17}$$

Figure 2.6 shows the grid arrangement in a 4-level multigrid method. Here, level-1 indicates

the finest grid, level-2 and level-3 indicates the coarser grid and level-4 indicates the coarsest grid. The steps to be followed for a 4-level multigrid method applied to the 2D heat conduction problem are as follows:

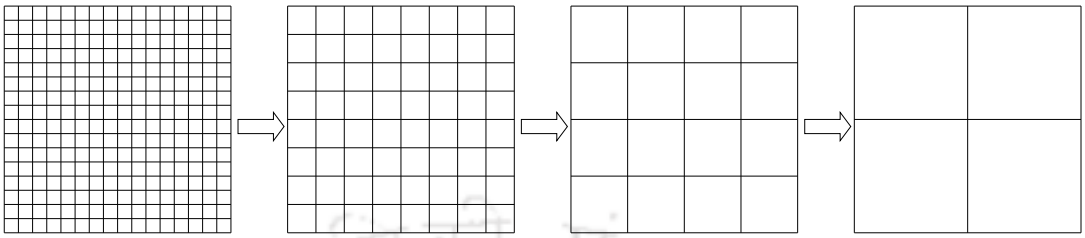


Figure 2.6: *Grid arrangements in the 4-level multigrid method.*

1. The difference equation, $A\phi_{i,j} = 0$ is iterated k times ($k = 3$ or 4) on the finest grid.

2. The residual, $r^1_{i,j} = -A\phi^k_{i,j}$ is computed and stored at each point. This residual is then restricted by injection to the next coarse grid. The restricted residual is denoted as $r^2_{i,j} = I^2_1 r^1_{i,j}$, where I denotes the transfer operator, its subscript indicates the level of origin and its superscript indicates the level of destination and the superscript on the r indicates the grid level upon which the residual is computed.

3. The correction equation:

$$Ae^2_{i,j} = r^2_{i,j} = I^2_1 r^1_{i,j} \tag{2.18}$$

is iterated n times on grid level 2 using zero as the initial guess while keeping the residual fixed at each grid point. The solution after n iterations, $e^2_{i,j}$, represents a correction to the fine-grid solution. This solution, as well as the residual used to obtain it, are stored for future use in the prolongation phase. In order to transfer the problem to a next coarser grid, an updated residual needs to be computed on grid level 2. The updated residual at level 2 is:

$$r^2_{i,j} = r^2_{i,j} - Ae^2_{i,j} = I^2_1 r^1_{i,j} - Ae^2_{i,j} \tag{2.19}$$

where $e^2_{i,j}$ is the solution obtained on the grid level 2 after n iterations. The newly updated residual is then restricted to the next coarser grid (level 3) as $I^3_2 r^2_{i,j}$.

4. The correction equation:

$$Ae^3_{i,j} = r^3_{i,j} = I^3_2 r^2_{i,j} \tag{2.20}$$

is iterated n times on grid level 3 using zero as the initial guess. The solution after n iterations can be thought of as a correction to the correction obtained on grid level 2, which of course represents a further correction to the fine-grid solution. This solution and the residual used to obtain it are stored for use in the prolongation phase. A new residual needs to be computed on grid level 3 in order to transfer to the coarsest grid (level 4). The new residual at level 3 is:

$$r^3_{i,j} = r^3_{i,j} - Ae^3_{i,j} = I^3_2 r^2_{i,j} - Ae^3_{i,j} \tag{2.21}$$

where $e^3_{i,j}$ is the solution obtained on the grid level 3 after n iterations. The newly updated residual is then restricted to the next coarser grid (level 4). The correction equation $Ae^4_{i,j} = r^4_{i,j} = I^4_3 r^3_{i,j}$ is iterated to convergence on the coarsest grid accurately using zero as the initial guess. As there is no other coarser grid, there is no need to find the residual at this grid level.

5. Now the corrections obtained on the coarsest grid are transferred (prolongated) onto the next finer grid and are represented as $I^3_4 e^4_{i,j}$. These are added to the corrections $e^3_{i,j}$ obtained earlier at level 3 . The sum of the two corrections is used as the initial guess at each point, relaxed a few times using equation (2.20). The new solution represents improved corrections. The corrections from level 3 are prolongated onto the next finer grid (level 2). These corrections are added to the already obtained corrections $e^2_{i,j}$ at level 2. Using these corrections as the initial guess, n iterations are made on grid level 2 using equation (2.18). It is to be noted that, no new residuals are computed in the prolongation phase of moving up from the coarser to the finer grid.

6. The improved corrections from level 2 are prolongated onto grid-level 1 (finest grid) and added to the previous solution obtained on the finest grid $\phi_{i,j}$. The corrected solution is then relaxed for n sweeps. This completes one cycle. Now check for convergence. If convergence has not been reached, this cycle has to be repeated till desired convergence is achieved.

2.1.2 Results and Discussions

The number of iterations (in terms of equivalent fine-grid sweeps or work units) required for convergence for all the schemes is given in Table 2.1. For the multigrid technique, three

sweeps are made on the fine and all intermediate grids before a transfer is made and convergence is achieved on the coarsest grid for each cycle. The first column, labelled GS, gives results obtained with the conventional Gauss-Seidel scheme with no over-relaxation. The second column labelled GSω_{opt} gives the result obtained with the Gauss-Seidel scheme using the optimum over-relaxation factor. The column labelled MG2 gives the result obtained with the two-level multigrid with V-Cycle (two-level W-Cycle is the same as V-Cycle), and the column labelled MGMAX provides the multigrid result for all the three cycles obtained using the maximum number of levels, i.e., taking the calculation down to one internal grid point. This results in use of seven, six, five, four, and three levels for the 129×129, 65×65, 33×33, 17×17, and 9×9 grids, respectively.

The detailed result is presented in Table 2.2. It consists of all the grid-levels for all the three multigrid cycles in the 129×129, 65×65, 33×33, 17×17, and 9×9 grids.

A number of interesting points can be made from the results shown in Table 2.1. The number of iterative sweeps required by the standard Gauss-Seidel scheme can be seen to be almost proportional to the number of grid points used. Of course, the computational effort per sweep is also proportional to the number of points used. The use of the optimum over-relaxation factor reduces the computational effort substantially, especially as the number of grid points increases. For the finest grid, the use of over-relaxation reduces the computational effort by a factor of about 24 which is significant. The two-level multigrid is seen to provide a significant reduction in computational effort, but it does not perform quite as well as the Gauss-Seidel scheme with optimum over-relaxation. However, it is general, whereas the optimum over-relaxation factor can only be computed in advance for special cases.

The performance of the multigrid with the maximum number of levels (which can be called the $n-$level scheme) is truly amazing. The number of sweeps is seen to be nearly independent of the number of grid points used. For the finest grid, only 21 sweeps were required for the $V-$Cycle, 33 for $W-$Cycle, and 17 for $FMG-$Cycle, compared to 6826 for the conventional Gauss-Seidel scheme. This is a reduction in effort by a factor a 325 for $V-$Cycle, 207 for $W-$Cycle, and 402 for $FMG-$Cycle. It requires only 1/13th (for $V-$Cycle), 1/9th (for $W-$Cycle), and 1/17th (for $FMG-$Cycle) as much effort as the Gauss-Seidel scheme with the optimum over-relaxation. This reduction in effort or "speed-up-factor" is shown graphically in figure 2.7. All multigrid schemes required 5 cycles except for $FMG-$Cycle which required 3, for convergence for this problem, which utilized only

Grid	GS	GSω_{opt}	MG2	MGMAX		
				$V-$Cycle	$W-$Cycle	$FMG-$Cycle
9×9	62	19	23	18	24	16
17×17	215	40	44	20	29	16
33×33	715	75	102	21	32	17
65×65	2282	137	267	21	33	17
129×129	6826	282	744	21	33	17

Table 2.1: Number of equivalent fine-grid iterations required for convergence

Levels	Cycles	129×129	65×65	33×33	17×17	9×9
2	$V-$cycle	744	267	102	44	23
	$W-$cycle	744	267	102	44	23
	$FMG-$cycle	–	–	–	–	–
3	$V-$cycle	85	42	27	22	18
	$W-$cycle	100	51	34	28	24
	$FMG-$cycle	62	31	22	17	16
4	$V-$cycle	26	22	21	20	–
	$W-$cycle	37	32	30	29	–
	$FMG-$cycle	20	18	17	16	–
5	$V-$cycle	21	21	21	–	–
	$W-$cycle	33	32	32	–	–
	$FMG-$cycle	17	17	17	–	–
6	$V-$cycle	21	21	–	–	–
	$W-$cycle	33	32	–	–	–
	$FMG-$cycle	17	17	–	–	–
7	$V-$cycle	21	–	–	–	–
	$W-$cycle	33	–	–	–	–
	$FMG-$cycle	17	–	–	–	–

Table 2.2: Detailed results of $V-$, $W-$, and $FMG-$Cycle.

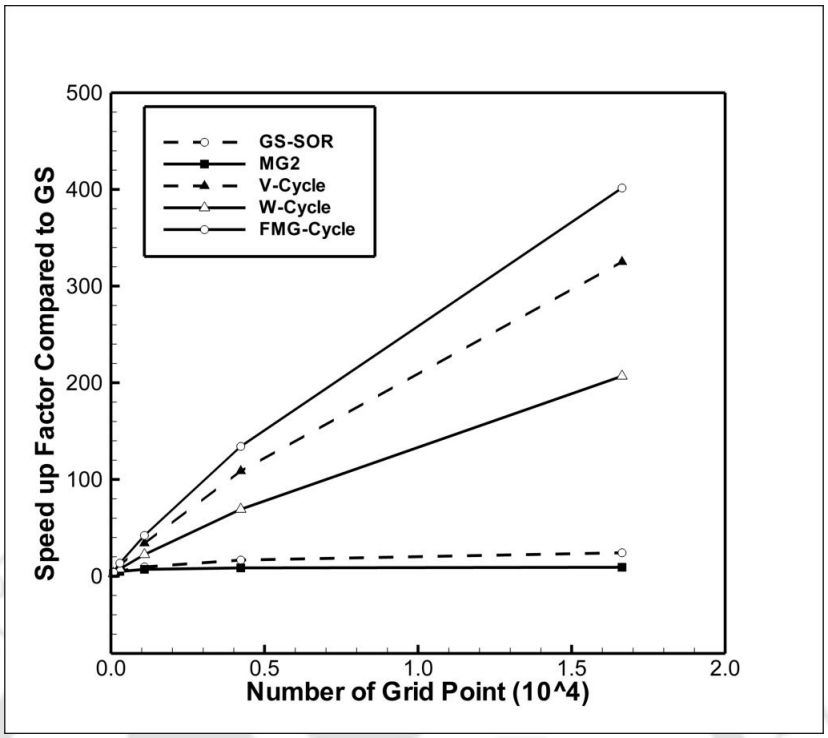


Figure 2.7: *Comparison of effort for Dirichlet problem.*

13 to 14 sweeps through the finest grid ($FMG-$Cycle required only 11).

For the 129×129 grid, it was observed that using four or five levels gave nearly the same performance as using seven levels. Specially it was observed for $V-$Cycle from Table 2.2 that 744, 85, 26, 21, 21, and 21 work units (equivalent fine-grid sweeps) were required using 2, 3, 4, 5, 6, and 7 levels, respectively. Similar observations can also be made for $W-$Cycle and $FMG-$Cycle. This trend is illustrated in figure 2.8. The especially large improvement observed when moving from two to three levels is noteworthy.

The results were fairly insensitive to the number of sweeps for the fine and intermediate grids. The following trend was observed for the $n-$level scheme on the 129×129 grid. For use of 2, 3, 4, and 5 sweeps, the number work units required was 21, 21, 22, and 25, respectively for the $V-$Cycle. The performance is found to deteriorate as the number of sweeps was increased above 5. This can be seen in figure 2.9.

For the two-level scheme, performance was actually improved by not converging on the coarse grid. For the 129×129 grid and for $V-$Cycle, the number of equivalent fine grid sweeps required for convergence was reduced to 521, 503, 513, and 524 if the maximum number of coarse-grid sweeps was limited to 75, 100, 125, and 150 respectively. This represents a reduction of about one-third in the computational effort for the two-level scheme.

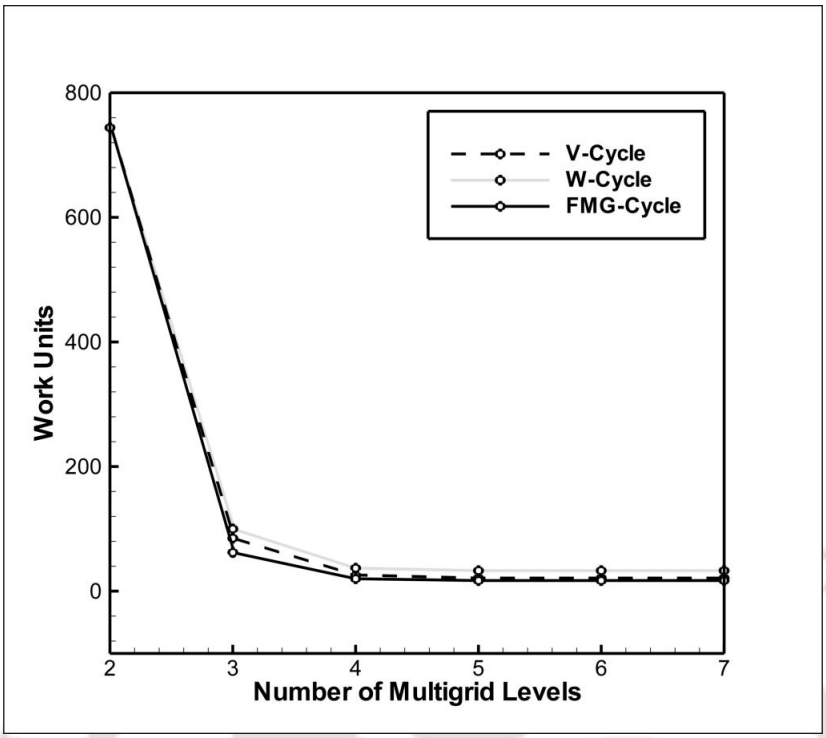


Figure 2.8: *Effect of Multigrid levels on Dirichlet problems for* 129×129 *grid.*

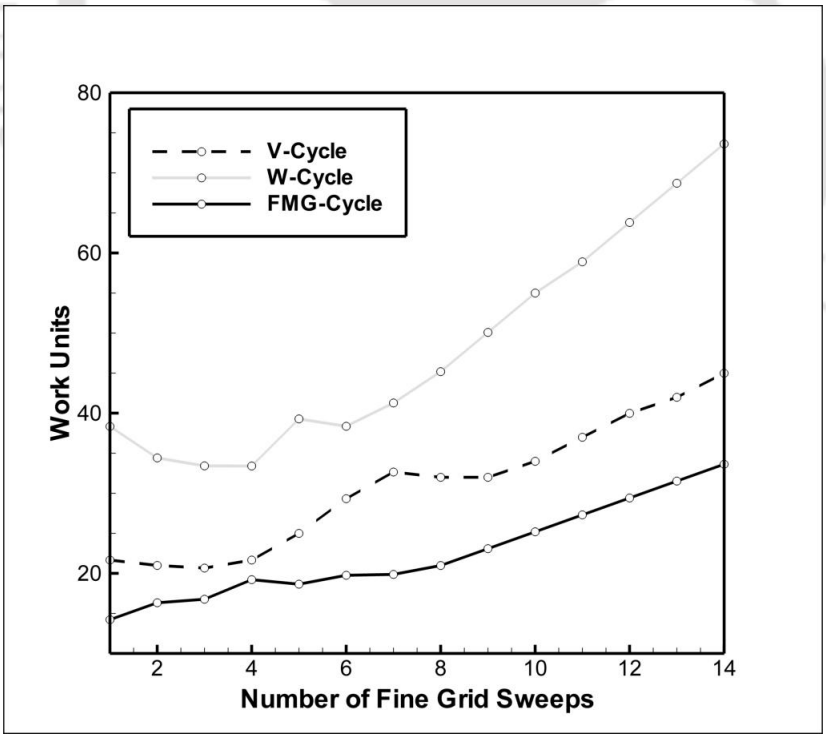


Figure 2.9: *Variation of Work Unit with number of Fine Grid Sweeps for* 129×129 *grid.*

2.1.3 Optimization of the Storage and CPU Time in Multigrid

The storage costs of multigrid algorithms decrease relatively as the dimension of the problem increases. Multigrid requires half the number of grid points in each dimension in the subsequent coarser grid. Therefore, the effective number of variables required to perform the computation is reduced by about four times for a two-dimensional problem. For example, suppose a five-level multigrid V-cycle is used for a computation and it uses grids of 9×9, 17×17, 33×33, 65×65 and 129×129. Therefore, we need to store the variables for a total of 22325 points. One suitable way is to use an array of size 22325 and of type double for each of the solution variables. Apart from the solution variables, additional arrays are required to store the residuals and the prolongation arrays.

In a multigrid system, there are a number of parameters that can be varied. For instance,

1. Number of multigrid cycles
2. Number of multigrid levels
3. Number of pre-smooths
4. Number of post-smooths
5. Number of fixed iterations given on the coarsest grid
6. Under/Over relaxation of the restricted values
7. Under/Over relaxation of the prolongated values

By properly optimizing the above parameters, the CPU time taken by a multigrid system can be significantly reduced. The reduction of CPU time often outweighs the slight increase in the storage requirement.

2.1.4 Nonlinear Multigrid

The multigrid method discussed in the previous section is applicable to linear equations, where we transfer only the residual from one grid to another. If the equation is nonlinear we should transfer both the solution as well as the residual from the fine to coarse grid and operate on the absolute solution on all grid levels. This type of multigrid is known as the full-approximation storage (FAS) multigrid method. The solution procedure for the FAS

scheme is the same as that of the correction scheme. Now we will outline the commonly used FAS multigrid method.

Consider a system of nonlinear algebraic equations,

$$A(\mathbf{u}) = \mathbf{f} \tag{2.22}$$

where $\mathbf{u}, \mathbf{f} \in \mathbf{R}^n$. Here A represents a nonlinear operator. Let us assume $\mathbf{v}$ is an approximation to the exact solution $\mathbf{u}$, then the error $\mathbf{e}$ and the residual $\mathbf{r}$ are given by

$$\mathbf{e} = \mathbf{u} - \mathbf{v} \tag{2.23}$$

$$\mathbf{r} = \mathbf{f} - A(\mathbf{v}) \tag{2.24}$$

Now subtracting equation (2.22) from equation (2.24), we find that

$$A(\mathbf{u}) - A(\mathbf{v}) = \mathbf{r} \tag{2.25}$$

Using equation (2.23), equation (2.25) can be written as,

$$A(\mathbf{v} + \mathbf{e}) - A(\mathbf{v}) = \mathbf{r} \tag{2.26}$$

Suppose we have found an approximation, $\mathbf{v}^h$, to the original fine-grid problem

$$A^h\left(\mathbf{u}^h\right) = \mathbf{f}^h \tag{2.27}$$

we now want to use the residual equation on the coarse grid Ω^{2h} to approximate $\mathbf{e}^h$, the error in $\mathbf{v}^h$. Using the above argument, the residual equation on the coarse grid appears as

$$A^{2h}\left(\mathbf{v}^{2h} + \mathbf{e}^{2h}\right) - A^{2h}\left(\mathbf{v}^{2h}\right) = \mathbf{r}^{2h} \tag{2.28}$$

where A^{2h} denotes the coarse-grid operator, $\mathbf{r}^{2h}$ is the coarse-grid residual, $\mathbf{v}^{2h}$ is a coarse-grid approximation to $\mathbf{v}^h$, and $\mathbf{e}^{2h}$ is a coarse-grid approximation to $\mathbf{e}^h$. Once $\mathbf{e}^{2h}$ is computed, the fine-grid approximation can be updated by $v^h \leftarrow v^h + I_{2h}^h e^{2h}$. Since $\mathbf{r}^{2h}$ in equation (2.28) is the restricted residual from the fine grid, it can be written as

$$\mathbf{r}^{2h} = I_h^{2h}\mathbf{r}^h = I_h^{2h}\left(\mathbf{f}^h - A^h\left(\mathbf{v}^h\right)\right) \tag{2.29}$$

where I_h^{2h} is the restriction operator. It has already been mentioned that in FAS method not only the residuals but the solutions should also be transferred from the fine to coarse grid. The approximation $\mathbf{v}^{2h}$ in equation (2.28) is also restricted from the fine grid. Thus it can be represented as

$$\mathbf{v}^{2h} = I_h^{2h}\mathbf{v}^h \tag{2.30}$$

Making use of equation (2.29) and (2.30) in equation (2.28) yields

$$A^{2h}\left(I_h^{2h}\mathbf{v}^h + \mathbf{e}^{2h}\right) = A^{2h}\left(I_h^{2h}\mathbf{v}^h\right) + I_h^{2h}\left(\mathbf{f}^h - A^h\left(\mathbf{v}^h\right)\right) \tag{2.31}$$

where $I_h^{2h}\mathbf{v}^h + \mathbf{e}^{2h} = \mathbf{u}^{2h}$. The right side of this nonlinear system is known. The goal is to find or approximate the solution to this system, which we have denoted $\mathbf{u}^{2h}$. The coarse-grid error approximation, $\mathbf{e}^{2h} = \mathbf{u}^{2h} - I_h^{2h}\mathbf{v}^h$, can then be interpolated up to the fine grid and used to correct the fine-grid approximation $\mathbf{v}^h$. This correction step takes the form

$$\mathbf{v}^h \leftarrow \mathbf{v}^h + I_{2h}^h\mathbf{e}^{2h} \tag{2.32}$$

or

$$\mathbf{v}^h \leftarrow \mathbf{v}^h + I_{2h}^h\left(\mathbf{u}^{2h} - I_h^{2h}\mathbf{v}^h\right) \tag{2.33}$$

This is the essence of the full approximation storage multigrid method.

2.1.5 Algorithms of the Different Cycles

The algorithms of the different cycles are outlined below in the recursive form. These are general algorithms valid for any number of grid levels (except for FAS).

2.1.6 V-Cycle Scheme

The algorithm of the V-Cycle Scheme is given in recursive form as follows[1]

$$\mathbf{v}^h \leftarrow V^h\left(\mathbf{v}^h, \mathbf{f}^h\right).$$

1. Relax ν_1 times on $A^h\mathbf{u}^h = \mathbf{f}^h$ with a given initial guess $\mathbf{v}^h$.

2. If Ω^h = coarsest grid, then go to step 4.

Else

$$\begin{aligned} \mathbf{f}^{2h} &\leftarrow I_h^{2h}\left(\mathbf{f}^h - A^h\mathbf{v}^h\right), \\ \mathbf{v}^{2h} &\leftarrow \mathbf{0} \\ \mathbf{v}^{2h} &\leftarrow V^{2h}\left(\mathbf{v}^{2h}, \mathbf{f}^{2h}\right), \end{aligned}$$

3. Correct $\mathbf{v}^h \leftarrow \mathbf{v}^h + I_{2h}^h\mathbf{v}^{2h}$.

4. Relax ν_2 times on $A^h\mathbf{u}^h = \mathbf{f}^h$ with initial guess $\mathbf{v}^h$.

2.1.6.1 $\mu-$Cycle Scheme

The V-Cycle is just one of a family of multigrid cycling schemes. The entire family is called the *μ-cycle* method and is defined recursively by the following.[1]

$$\mathbf{v}^h \leftarrow M\mu^h\left(\mathbf{v}^h, \mathbf{f}^h\right).$$

1. Relax ν_1 times on $A^h\mathbf{u}^h = \mathbf{f}^h$ with a given initial guess $\mathbf{v}^h$.

2. If Ω^h = coarsest grid, then go to step 4.
   Else

$$\begin{aligned} \mathbf{f}^{2h} &\leftarrow I_h^{2h}\left(\mathbf{f}^h - A^h\mathbf{v}^h\right), \\ \mathbf{v}^{2h} &\leftarrow \mathbf{0} \\ \mathbf{v}^{2h} &\leftarrow M\mu^{2h}\left(\mathbf{v}^{2h}, \mathbf{f}^{2h}\right) \mu \text{ times.} \end{aligned}$$

3. Correct $\mathbf{v}^h \leftarrow \mathbf{v}^h + I_{2h}^h\mathbf{v}^{2h}$.

4. Relax ν_2 times on $A^h\mathbf{u}^h = \mathbf{f}^h$ with initial guess $\mathbf{v}^h$.

In practice, only $\mu = 1$ (which gives the V-cycle) and $\mu = 2$ are used. Figure 2.4(b) shows the schedule of grids for $\mu = 2$ and the resulting *W-cycle*. A V-cycle with ν_1 relaxation sweeps before the correction step and ν_2 relaxation sweeps after the correction step is referred as a $V(\nu_1, \nu_2)$-cycle, with a similar notation for W-cycles.

2.1.6.2 Full Multigrid V-Cycle

In the case of Full Multigrid, we use coarse grids to obtain better initial guesses. The idea of using coarser grids to generate improved initial guesses is the basis of a strategy called *nested iteration. Full multigrid V-cycle (FMG)* joins the nested iteration with the V-cycle. In the case of FMG, we first solve or relax on the coarsest grid. Then the result is interpolated to the next finer grid and V-cycle is invoked. The procedure is repeated till we reach the finest grid. Expressed recursively, the algorithm has the following compact form[1].

$$\mathbf{v}^h \leftarrow FMG^h\left(\mathbf{f}^h\right).$$

1. If Ω^h = coarsest grid, set $\mathbf{v}^h \leftarrow \mathbf{0}$ and go to step 3.
   Else
   $$\begin{aligned}\mathbf{f}^{2h} &\leftarrow I_h^{2h}\left(\mathbf{f}^h\right),\\ \mathbf{v}^{2h} &\leftarrow FMG^{2h}\left(\mathbf{f}^{2h}\right).\end{aligned}$$
2. Correct $\mathbf{v}^h \leftarrow I_{2h}^h \mathbf{v}^{2h}$.
3. $\mathbf{v}^h \leftarrow V^h\left(\mathbf{v}^h, \mathbf{f}^h\right)$ ν_0 times.

We initialize the coarse-grid right sides by transferring $\mathbf{f}^h$ from the fine grid. Another option is to use the original right-side function f. The cycling parameter, ν_0, sets the number of V-cycles done at each level. It is generally determined by a previous numerical experiment; $\nu_0 = 1$ is the most common choice.

Figure 2.4(c) shows the schedule of grids for FMG with $\nu_0 = 1$. Each V-cycle is preceded by a coarse-grid V-cycle designed to provide the best initial guess possible. The extra work done in these preliminary V-cycles is not only inexpensive but easily pays for itself[1].

The Full Multigrid can also use the more general μ-cycle in place of the V-cycle. (In fact, V-cycle is a special case of the μ-cycle with $\mu = 1$).

2.1.6.3 Full Approximate Storage (FAS)

A two-grid version of this scheme is described as follows:

- Restrict the current approximation and its fine-grid residual to the coarse grid: $\mathbf{r}^{2h} = I_h^{2h}\left(\mathbf{f}^h - A^h\left(\mathbf{v}^h\right)\right)$ and $\mathbf{v}^{2h} = I_h^{2h}\mathbf{v}^h$.

- Solve the coarse-grid problem $A^{2h}\left(\mathbf{u}^{2h}\right) = A^{2h}\left(\mathbf{v}^{2h}\right) + \mathbf{r}^{2h}$.
- Compute the coarse-grid approximation to the error: $\mathbf{e}^{2h} = \mathbf{u}^{2h} - \mathbf{v}^{2h}$.
- Interpolate the error approximation up to the fine grid and correct the current fine-grid approximation: $\mathbf{v}^{h} \leftarrow \mathbf{v}^{h} + I_{2h}^{h}\mathbf{e}^{2h}$.

It is worth noting that if A is a linear operator, then the FAS scheme reduces directly to the (linear) two-grid correction scheme. Thus, FAS can be viewed as a generalization of the two-grid correction scheme to nonlinear problems.

Since the second step in the above procedure involves a nonlinear problem itself, FAS involves an inner and an outer iteration; the outer iteration is the FAS correction scheme, while the inner iteration is usually a standard relaxation method such as nonlinear Gauss-Seidel. A true multilevel FAS process would be done recursively by approximating solutions to the Ω^{2h} problem using the next coarsest grid, Ω^{4h}. Thus, FAS, like its linear counterparts, is usually implemented as a V-cycle or W-cycle scheme.

2.2 Lattice Boltzmann Method

2.2.1 Introduction

The relation between the Boltzmann equation and the Navier-Stokes equation for the study of fluid dynamics has been an active and popular topic of research in the past few decades. The Boltzmann equation relates the time evolution and spatial variation of a collection of molecules to a collision operator that describes the interaction of the molecules. Compared to the Navier-Stokes equation, the Boltzmann equation is known to provide a more efficient representation of gaseous flows for a whole range of flow regimes. However, the conventional numerical methods (finite difference method, finite volume method, finite element method, etc.) based on the discretization of partial differential equations (Navier-Stokes equations) are generally preferred by the researchers in continuum regime to solving the Boltzmann equations. This is because the solution of the Boltzmann equation is a non-trivial task owing to the complexity of the collision term.

The development of Lattice Gas Automata (LGA) [14] and Lattice Boltzmann Method (LBM) [78] are the promising methods that use nonconventional techniques for applications in CFD. However, the LGA suffered from some drawbacks such as statistical noise, the lack of Galilean invariance and non-physical solution (e.g., pressure depending on velocity). To overcome the difficulties of LGA, the LBM uses the simple model of linearized

collision operator based on the Bhatnagar-Gross-Krook (LBGK) collision model [79]. The justification for the LBM approach is the fact that the collective behaviour of many microscopic particles is behind the macroscopic dynamics of fluid and this dynamics is not particularly dependent on the details of the microscopic phenomena exhibited by the individual molecules. It is the solution of a minimal Boltzmann kinetic equation, rather than the discretization of the Navier-Stokes equations of continuum mechanics. It provides stable and efficient numerical computations for the macroscopic behaviour of fluids, although describing the fluid in a microscopic way.

2.2.2 Development of LBM

The LBM has evolved from the Boolean fluid model known as the Lattice Gas Cellular Automata (LGCA). LGA (or LGCA) implements particle streaming and colliding in a fully discrete dimension. By satisfying mass conservation (number of Boolean particles) and momentum conservation, LGCA can yield Navier-Stokes (NS) equations. However, the derivation of the lattice Boltzmann model from the continuum Boltzmann equation makes it a self-standing research subject in connection with statistical physics or kinetic theory equation. The LBM approximates the continuous Boltzmann equation by discretising physical space with a set of uniformly spaced lattice nodes and velocity space by a discrete set of microscopic velocity vectors. The time- and space-averaged microscopic movements of particles are modelled using molecular populations called distribution functions, which obey specific particle interaction rules to satisfy Navier-Stokes equations.

2.2.2.1 Boltzmann transport equation

Ludwig Boltzmann formulated the Boltzmann equation (BE) [80, 81] to describe the statistical distribution of particles in a fluid. It is one of the essential kinetic equation for non-equilibrium statistical mechanics that deals with systems far from thermodynamic equilibrium. The BE uses a statistical description known as distribution function, $f(\mathbf{r}, \mathbf{e}, t)$ which is the probable number of particles in a unit volume of fluid positioned about $\mathbf{r}$ in the physical space element $d^3\mathbf{r} = d\mathbf{x}d\mathbf{y}d\mathbf{z}$ with velocity $\mathbf{e}$ in the velocity space element $d^3\mathbf{e} = d\mathbf{u}d\mathbf{v}d\mathbf{w}$ at time t, to describe the mechanics of fluid. Mathematically, it is expressed by the following relation

$$\int f(\mathbf{r}, \mathbf{e}, t) d^3\mathbf{r} d^3\mathbf{e} = N \tag{2.34}$$

where N is the total number of molecules in the container.

If an external force $\mathbf{F} = m\mathbf{a}(\mathbf{r}, t)$ (m is mass and $\mathbf{a}$ is the acceleration of a particle) is applied to a particle, the number of particles will remain the same (neglecting collision of particles) and it is presented as [81]:

$$\int f\left(\mathbf{r} + \mathbf{e}dt, \mathbf{e} + \left(\frac{\mathbf{F}}{m}\right)dt, t + dt\right) d\mathbf{r}d\mathbf{e} - f(\mathbf{r}, \mathbf{e}, t)\, d\mathbf{r}d\mathbf{e} = 0 \tag{2.35}$$

Considering the collision between particles, the equation becomes:

$$\int f\left(\mathbf{r} + \mathbf{e}dt, \mathbf{e} + \left(\frac{\mathbf{F}}{m}\right)dt, t + dt\right) d\mathbf{r}d\mathbf{e} - f(\mathbf{r}, \mathbf{e}, t)\, d\mathbf{r}d\mathbf{e} = \Omega(f)\, d\mathbf{r}d\mathbf{e}dt. \tag{2.36}$$

Here, $\Omega(f)$ is the collision operator. Dividing the above equation by $d\mathbf{r}d\mathbf{e}dt$ and taking the limit of $dt \to 0$, the rate of change of number of particles, i.e., distribution function is given by

$$\frac{dt}{dt} = \Omega(f) \tag{2.37}$$

The total rate of change of distribution function can be expressed as,

$$df = \frac{\partial f}{\partial \mathbf{r}}d\mathbf{r} + \frac{\partial f}{\partial \mathbf{e}}d\mathbf{e} + \frac{\partial f}{\partial t}dt \tag{2.38}$$

Therefore, the Boltzmann transport equation can be stated as

$$\frac{\partial f}{\partial t} + \mathbf{e} \cdot \frac{\partial f}{\partial \mathbf{r}} + \frac{\mathbf{F}}{m} \cdot \frac{\partial f}{\partial \mathbf{e}} = \Omega(f) \tag{2.39}$$

For a system without an external force, the Boltzmann equation can be given as,

$$\frac{\partial f}{\partial t} + \mathbf{e} \cdot \frac{\partial f}{\partial \mathbf{r}} = \Omega(f) \tag{2.40}$$

Distribution function $f(\mathbf{r}, \mathbf{e}, t)$ for single-particle depends on the scattering of a particle due to collision with another particle. The collision operator, $\Omega(f)$ requires position and speed of both particles at time t (i.e., requires two body distribution function $f^2(\mathbf{r}, \mathbf{e}, t)$) which indeed require $f^3(\mathbf{r}, \mathbf{e}, t)$ and so on up to N body distribution function $f^N(\mathbf{r}, \mathbf{e}, t)$. This results in Liouville equation [80] which exactly describe Newtonian behaviour of N-particles. This hierarchy of coupled N equations is known as the BBGKY hierarchy [80]. In order to simplify this complicated collision term, Boltzmann has taken several key assumptions based on dilute gas of point-like particles having only localised binary collision, due to which $Q(f, f)$ depends only on $f^2(\mathbf{r}, \mathbf{e}, t)$ and by taking the assumption of molecular

chaos (i.e. no correlation of particles entering short-lived collision), he replaced $f^2(\mathbf{r}, \mathbf{e}, t)$ by the product of $f^1(\mathbf{r}, \mathbf{e}, t)$ with velocity $\mathbf{e}_1$ and $f^2(\mathbf{r}, \mathbf{e}, t)$ with velocity $\mathbf{e}_2$. The simplified collision operator becomes

$$Q(f, f) = \int dC \int \mathbf{e}_2 \sigma(C) |\mathbf{e}_1 - \mathbf{e}_2| \left(f_1' f_2' - f_1 f_2\right) d\Omega \tag{2.41}$$

where, C is the scattering angle, f and f' are distribution function before and after a collision, $\sigma(C)$ is the differential collision cross section for the two-particle collision.

2.2.2.2 Linearization of collision operator

This complicated collision integral, i.e. $Q(f, f)$ is the major hurdle when dealing with the Boltzmann equation. To facilitate numerical and analytical solutions of the Boltzmann equation, simpler operator $J(f)$ is used in place of $Q(f, f)$. This simplified collision integral $J(f)$ has to satisfy two constraints [14] as given below:

1. $J(f)$ has to conserve the five collision invariants ψ_k of $Q(f, f)$, i.e.

$$\int \psi_k J(f) d^3\mathbf{r} d^3\mathbf{e} = 0 \tag{2.42}$$

   where, $(k = 0, 1, 2, 3, 4)$, $\psi_0 = 1$ for conservation of mass, $\psi_{1,2,3} = \mathbf{e}$ for conservation of momentum and $\psi_4 = \mathbf{e}_2$ for conservation of energy.

2. The collision term should have the capability of approximating the distribution function close to the equilibrium distribution function, i.e. Maxwellian distribution by H-theorem. (https://www.thphys.uni-heidelberg.de/~wolschin/statsem21_4s.pdf)

If no external forces are present and the distribution function is independent of $\mathbf{r}$ and time t (i.e., for $t \to \infty$ under an arbitrary initial condition), then the distribution function $f(\mathbf{e})$ approaches equilibrium distribution function $f^{eq}(\mathbf{e})$. The equilibrium distribution function satisfies the Boltzmann Transport equation (2.40), when the right-hand side of the BE is zero, i.e.,

$$\int dC \int \mathbf{e}_2 \sigma(C) |\mathbf{e}_1 - \mathbf{e}_2| \left(f_1' f_2' - f_1 f_2\right) = 0 \tag{2.43}$$

and there is only one possibility for this to be true, which is given as,

$$f_1^{'} f_2^{'} - f_1 f_2 = 0 \tag{2.44}$$

Equilibrium distribution function $f^{eq}(\mathbf{e})$ which satisfies this above condition is called Maxwell Boltzmann Distribution Function [81] which is given as,

$$f^{eq} = \rho \left(\frac{m}{2\pi k_B T} \right)^{\frac{D}{2}} exp \left(- \frac{m (\mathbf{e} - \mathbf{u})^2}{2 k_B T} \right) \tag{2.45}$$

where m is mass of one molecule, T is the temperature of the gas, $\mathbf{u}$ is a macroscopic velocity, k_B is the Boltzmann constant and D is the dimension of the space.

If the external body force is neglected, both of the two constraints on collision integral are fulfilled by the widely used BGK model developed by Bhatnagar, Gross and Krook [79]. A simple approximation for collision term which is obtained by exponentially relaxes to local equilibrium with a time constant as given below:

$$J(f) = - \left(\frac{f - f^{eq}}{\tau} \right) \tag{2.46}$$

where τ is single relaxation time and f^{eq} is a local Maxwellian function.

2.2.2.3 Derivation of lattice Boltzmann equation

The LBE can be thought of as a special finite difference form of the discrete Boltzmann equation, where a first-order finite difference scheme is applied to both time and space derivatives. If a set of discrete velocities and the time step and lattice increment are appropriately chosen, the convection term of the LBE becomes a simple moving of the distribution density from one lattice node to another. LBE can be obtained from the simplified BE by following three necessary steps.

1. A suitable local equilibrium function which plays an essential role in the LBM is to be determined. This function determines the kind of flow conditions that are solved employing LBE. Equilibrium distribution function, f^{eq} in the BGK collision operator is primarily the low Mach number Taylor series expansion of Maxwellian function.

$$f^{eq} = \frac{\rho}{(2\pi RT)^{\frac{D}{2}}} exp \left(- \frac{\mathbf{e}^2}{2RT} \right) \left\{ 1.0 + \frac{\mathbf{e} \cdot \mathbf{u}}{RT} + \frac{(\mathbf{e} \cdot \mathbf{u})^2}{2 (RT)^2} - \frac{\mathbf{u}^2}{2RT} \right\} + (u^3) \tag{2.47}$$

where R is the gas constant.

From the kinetic theory, the specific internal energy can be expressed as,

$$E = \frac{1}{2}\mathbf{e}^2 = \frac{3}{2}RT \tag{2.48}$$

Therefore, RT becomes $\frac{\mathbf{e}^2}{3}$ which is the value of the lattice speed of sound C_s in LBM. The f^{eq} can be viewed as a multiplication of weighting function $\psi(\mathbf{e}) = \frac{1}{(2\pi RT)^{\frac{D}{2}}}$ with a polynomial in $\mathbf{e}$. The continuous integrals are evaluated as sums of a discrete set of weight functions w_i with the help of Gauss quadrature corresponding to each discrete velocity $\mathbf{e}_i$ and is given by

$$\int \psi(\mathbf{e})\,\mathbf{e}^n d\mathbf{e} = \sum_i w_i \mathbf{e}_i^{eq} \tag{2.49}$$

This leads to discrete equilibrium distribution function and is calculated by prescribed macroscopic density and velocity as follows [82]

$$f_i^{eq} = \rho w_i \left\{ 1.0 + \frac{\mathbf{e}_i \cdot \mathbf{u}}{C_s^2} + \frac{(\mathbf{e}_i \cdot \mathbf{u})^2}{2C_s^4} - \frac{\mathbf{u}^2}{2C_s^2} \right\} \tag{2.50}$$

For the D_2Q_9 lattice structure, the lattice weights (w_i) are [83]

$$w_i = \left\{ \begin{array}{cc} \frac{4}{9} & i = 0 \\ \frac{1}{9} & i = 1, \cdots, 4 \\ \frac{1}{36} & i = 5, \cdots, 8 \end{array} \right\} \tag{2.51}$$

and the discrete velocity set $\mathbf{e}_i$ is specified as

$$\mathbf{e}_i = \left\{ \begin{array}{cc} (0,0) & i = 0 \\ c(\cos[(i-1)\pi/2], \sin[(i-1)\pi/2]) & i = 1, \cdots, 4 \\ \sqrt{2}c(\cos[(2i-9)\pi/4], \sin[(2i-9)\pi/4]) & i = 5, \cdots, 8 \end{array} \right\} \tag{2.52}$$

For the D_3Q_{15} lattice structure, the lattice weights are given by [84]:

$$w_i = \left\{ \begin{array}{cc} \frac{2}{9} & i = 0 \\ \frac{1}{9} & i = 1, \cdots, 6 \\ \frac{1}{72} & i = 7, \cdots, 14 \end{array} \right\} \tag{2.53}$$

and the discrete velocity set $\mathbf{e}_i$ is

$$\mathbf{e}_i = \left\{ \begin{array}{cc} (0,0,0) & i=0 \\ c(\pm1,0,0), c(0,\pm1,0), c(0,0,\pm1) & i=1,\cdots,6 \\ c(\pm1,\pm1,\pm1) & i=7,\cdots,14 \end{array} \right\} \tag{2.54}$$

For the D_3Q_{19} lattice structure, the lattice weights are given by [85]:

$$w_i = \left\{ \begin{array}{cc} \frac{1}{3} & i=0 \\ \frac{1}{18} & i=1,\cdots,6 \\ \frac{1}{36} & i=7,\cdots,18 \end{array} \right\} \tag{2.55}$$

and the discrete velocity set $\mathbf{e}_i$ is

$$\mathbf{e}_i = \left\{ \begin{array}{cc} (0,0,0) & i=0 \\ c(\pm1,0,0), c(0,\pm1,0), c(0,0,\pm1) & i=1,\cdots,6 \\ c(\pm1,\pm1,0), c(0,\pm1,\pm1), c(\pm1,0,\pm1) & i=7,\cdots,18 \end{array} \right\} \tag{2.56}$$

And for the D_3Q_{27} lattice structure, the lattice weights are given by [85]:

$$w_i = \left\{ \begin{array}{cc} \frac{8}{27} & i=0 \\ \frac{2}{27} & i=1,\cdots,6 \\ \frac{1}{54} & i=7,\cdots,18 \\ \frac{1}{216} & i=19,\cdots,26 \end{array} \right\} \tag{2.57}$$

and the discrete velocity set $\mathbf{e}_i$ is

$$\mathbf{e}_i = \left\{ \begin{array}{cc} (0,0,0) & i=0 \\ c(\pm1,0,0), c(0,\pm1,0), c(0,0,\pm1) & i=1,\cdots,6 \\ c(\pm1,\pm1,0), c(0,\pm1,\pm1), c(\pm1,0,\pm1) & i=7,\cdots,18 \\ c(\pm1,\pm1,\pm1) & i=19,\cdots,26 \end{array} \right\} \tag{2.58}$$

2. The discretisation of velocity space $\mathbf{e}$ to obtain the necessary and minimum number of discrete velocities $\mathbf{e}_i$.

$$\frac{\partial f_i}{\partial t} + \mathbf{e}_i \cdot \frac{\partial f_i}{\partial \mathbf{r}} = \left(\frac{f_i^{eq} - f_i}{\tau} \right) \tag{2.59}$$

This equation is termed as the discrete Boltzmann equation.

3. The discretisation of time and space is carried out to obtain the lattice Boltzmann equation, which works as the governing equation for LBM. The LBE is given as [82]

$$f_i\left(\mathbf{r}+\mathbf{e}_i\Delta t, t+\Delta t\right)-f_i\left(\mathbf{r},t\right)=\frac{\Delta t}{\tau}\left(f_i^{eq}-f_i\right) \tag{2.60}$$

2.2.2.4 Algorithm of LBM

The LBM algorithm consists of two main steps: streaming and collision. The algorithm starts with an initialisation step to set the initial values of the distribution functions using a discrete equilibrium distribution function. The general LBM algorithm can be described with the following steps:

1. Collision: In this step the collision operator (figure 2.10b) is implemented. Collision step models the interactions between pseudo-particles (distribution functions) by conserving mass and momentum and bring back them to their equilibrium states. The collision process is local concerning each lattice site.

$$f_i\left(\mathbf{r},t+\Delta t\right)=f_i\left(\mathbf{r},t\right)+\frac{\Delta t}{\tau}\left(f_i^{eq}-f_i\right) \tag{2.61}$$

2. Streaming: The streaming step (figure 2.10c) models the shifting of distribution functions in the direction of motion from one node to the adjacent nodes.

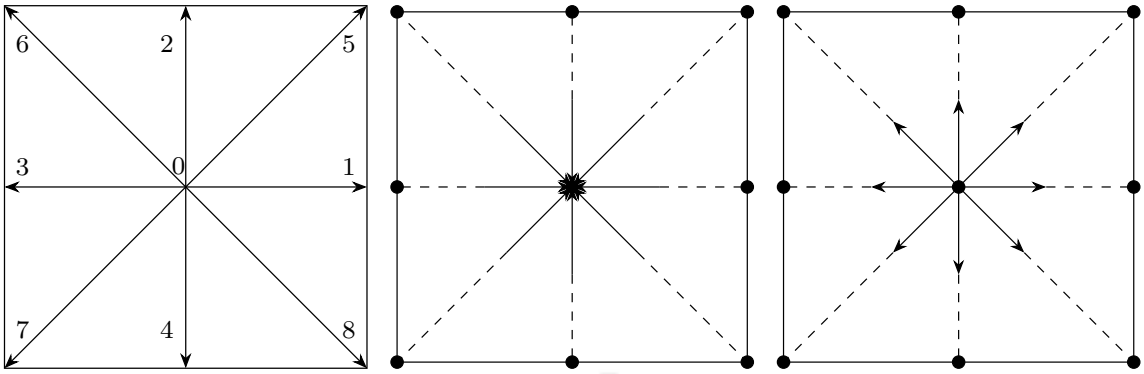
$$f_i\left(\mathbf{r}+\mathbf{e}_i\Delta t, t+\Delta t\right)=f_i\left(\mathbf{r},t+\Delta t\right) \tag{2.62}$$

3. After the collision and streaming the boundary is implemented, which defines the unknown distribution functions at the boundary lattice nodes.

4. These three steps are completed in one time-step of the LBM simulation. Hence, this sequence is repeated in a loop until the final time is reached.

2.2.3 Lattice structure in LBM

In LBM, it is assumed that all the particles reside at the fixed location called the lattice nodes. These lattices are the regular arrangement of the particles in the domain and hence all particles should follow or move along these fixed paths thereby reducing the degree of freedom of the system. All particles stream along with these lattice links and collide at lattice sites. The lattice structure in LBM is denoted by $DmQn$, where m is



(a) $D2Q9$ lattice model (b) Collision process (c) Streaming process

Figure 2.10: *Collision and streaming process of lattice Boltzmann method in a $D2Q9$ lattice.*

the number of dimensions and n denotes the number of particle velocities, i.e., the number of linkages of a node. As an example, the popular one-dimensional (1-D) model is referred to as the one-dimensional two-velocity ($D1Q2$) model. Other 1-D models are the one-dimensional three-velocity ($D1Q3$) and one-dimensional five-velocity ($D1Q5$) models. Two-dimensional lattice models are two-dimensional five-velocity ($D2Q5$), two-dimensional seven-velocity ($D2Q7$) and two-dimensional nine-velocity ($D2Q9$) models. Figure 2.10a shows $D2Q9$ lattice models with nine discrete velocities ($\mathbf{e}_i$) [86]. In a $D2Q9$ model, each node has eight neighbours connected by eight links. The particle distribution functions (PDF) with their respective velocity ($\mathbf{e}_i$) streams from one node to its neighbouring node in a streaming process along with these links in unit time. The three-dimensional counterparts are the three-dimensional fifteen-velocity ($D3Q15$, figure 2.11), three-dimensional nineteen-velocity ($D3Q19$) and three-dimensional twenty seven-velocity ($D3Q27$, figure 2.11) models. For three-dimensional problems, commonly $D3Q19$ lattice model [85] is used, as shown in figure 2.12. In this model, each node has eighteen neighbours connected by eighteen links. These 9 speeds and 19 speeds provide sufficient isotropy to the 2D and 3D models respectively and are sufficient for weakly compressible fluid applications. Besides, the use of a rest particle in a discrete velocity set is to improve the computational stability and reliability of the model. It is known that, through a Chapman-Enskog analysis, one can recover the governing continuity and momentum equations in the low Mach number limit [81, 85].

2.2.4 Methodology of basic LBM model

The two fundamental steps of the Lattice Boltzmann algorithm (LBE) are streaming and collision. The streaming step shifts the particle from one lattice node to the adjacent

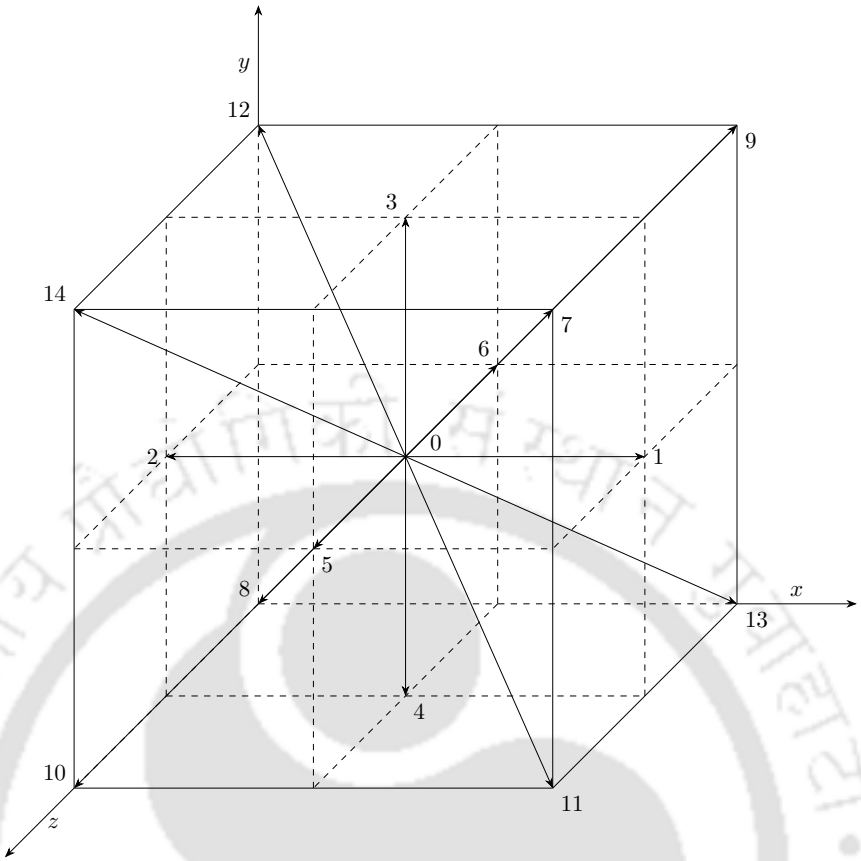


Figure 2.11: *D3Q15 lattice model.*

node. Whereas, the collision step models the interactions between molecules and can be modified. Depending on the linearised collision operator used in the lattice Boltzmann equation two approaches are mainly studied by researchers. The most popular and simple is the Bhatnagar-Gross-Krook (BGK) [87, 88] collision model in which both bulk and shear viscosity are determined by a single relaxation time (SRT) parameter. The SRT parameter is viscosity dependent and it relaxes the different physical modes to their equilibria at the same rate. Thus, it leads to deficiencies concerning its accuracy and stability. To overcome this limitation, a collision matrix with multiple relaxation times can be used which allows choosing the bulk viscosity independently of the shear viscosity. d'Humiéres [89] introduced the multiple relaxation time (MRT) approach which improves the numerical stability of LBM by relaxing the hydrodynamic and non-hydrodynamic moments with different relaxation times. However, the proper implementation of the MRT method is 15% to 20% slower than the SRT method as suggested by Lallemand and Luo [90]. The methodology of both models for isothermal and thermal flows is provided in the following sections. In order to facilitate numerical calculation and analysis, the proper conversion from the physical unit to the lattice unit has to be made. For the simulation of lid-

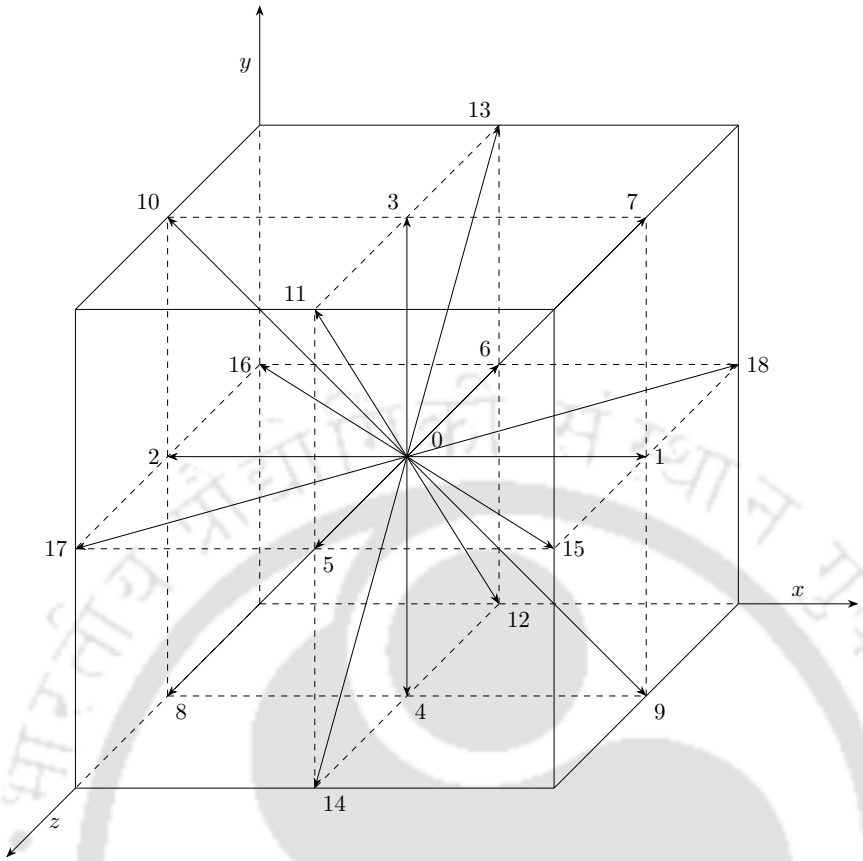


Figure 2.12: *D3Q19 lattice model.*

driven (temperature term is ignored) and mixed convection problems, the non-dimensional quantities are expressed as follows: q.

$$x = \frac{x^*}{L}; \qquad y = \frac{y^*}{L}; \qquad u = \frac{u^*}{V_0^*}; \qquad v = \frac{v^*}{V_0^*}; \qquad \theta = \frac{T - T_c}{T_h - T_c}; \tag{2.63}$$

whereas, for the simulation of buoyancy induced convection, the macroscopic quantities are presented in their dimensionless form as:

$$x = \frac{x^*}{L}; \qquad y = \frac{y^*}{L}; \qquad u = \frac{u^* L}{\alpha_f}; \qquad v = \frac{v^* L}{\alpha_f}; \qquad \theta = \frac{T - T_c}{T_h - T_c}; \tag{2.64}$$

where L and V_0^* are taken as the reference length and characteristic velocity of the fluid respectively. The symbols with '*' indicate the parameters in their dimensional form and without '*' indicate the same in their non-dimensional form. θ stands for non-dimensional temperature and α_f is the thermal diffusivity of the fluid (or base fluid in case of nanofluid).

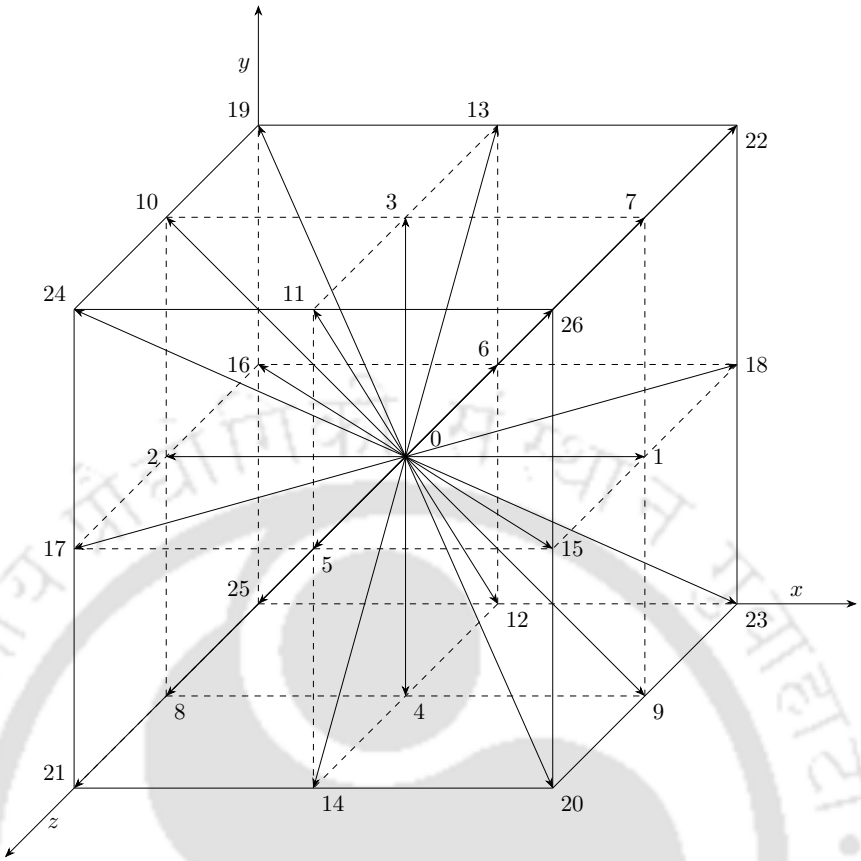


Figure 2.13: *D3Q27 lattice model.*

2.2.4.1 SRT-LBM model for isothermal flows

The governing equation for the lattice BGK model with a single relaxation time (SRT) is [86]

$$f_i\left(\mathbf{r}+\mathbf{e}_i\Delta t, t+\Delta t\right) = f_i\left(\mathbf{r},t\right) + \frac{1}{\tau_\nu}\left(f_i^{eq}\left(\mathbf{r},t\right) - f_i\left(\mathbf{r},t\right)\right) \tag{2.65}$$

The suffix i ($0 \leq i \leq 8$ for $D2Q9$ model) refers to a particle distribution function (PDF) with velocity $\mathbf{e}_i$ at which the distriution function streams from one node to its neighbouring node in a streaming process. Here, $\mathbf{r} = [x, y]$ is the position vector of an arbitrary lattice node, and Δt is the lattice time step. The single relaxation time (τ_ν) determine the kinetic viscosity, $\nu = C_s^2\Delta t\left(\tau_\nu - \frac{1}{2}\right)$ and it relaxes discrete density distribution function (f_i) towards its equilibrium distribution function (f_i^{eq}) in the collision process (given by the right-hand side of the equation).

2.2.4.2 MRT-LBM model for isothermal flows

The principle of MRT is to perform the collision process in the moment space rather than in the velocity space, as the collision operator $\Omega(f)$ is generally a full matrix. The MRT-LBM

introduces different relaxation times that relax the respective moment of the distribution functions, thereby improving numerical stability and accuracy compared to SRT-LBM. The evolution equation for the LBM-MRT model is given as [89]:

$$f_i\left(\mathbf{r}+\mathbf{e}_i\Delta t, t+\Delta t\right)-f_i\left(\mathbf{r},t\right)=-\mathrm{M}^{-1}\mathrm{R}\left\{\mathbf{m}\left(\mathbf{r},t\right)-\mathbf{m}^{eq}\left(\mathbf{r},t\right)\right\} \tag{2.66}$$

For $D2Q9$ model, the nine discretised distribution function corresponding to $\mathbf{e}_i$ and their nine moments in vector form are given as

$$\mathbf{f}\left(\mathbf{r},t\right)=\left\{f_0\left(\mathbf{r},t\right),f_1\left(\mathbf{r},t\right),\cdots,f_8\left(\mathbf{r},t\right)\right\}^T \tag{2.67}$$

$$\begin{aligned}\mathbf{m}\left(\mathbf{r},t\right)&=\left\{m_0\left(\mathbf{r},t\right),m_1\left(\mathbf{r},t\right),\cdots,m_8\left(\mathbf{r},t\right)\right\}^T\\&=\left(\rho,e,\in,j_x,q_x,j_y,q_y,P_{xx},P_{xy}\right))^T\end{aligned} \tag{2.68}$$

where ρ is the fluid density, e and $\in$ are related to the energy and its square, j_x and j_y are components of the momentum, q_x and q_y are the components of the energy flux, P_{xx} and P_{xy} correspond to the diagonal and off-diagonal components of the stress tensor. The superscript T denotes transpose operator.

The transformation matrix M is a 9×9 matrix that linearly transforms the density distribution functions ($\mathbf{f}$) in velocity space to their velocity moments ($\mathbf{m}$) in moment space and vice-versa follows:

$$\mathbf{m}=\mathrm{M}\mathbf{f},\quad \mathbf{f}=\mathrm{M}^{-1}\mathbf{m} \tag{2.69}$$

The transformation matrix M is given by equation (2.70).

$$\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\
-4 & -1 & -1 & -1 & -1 & 2 & 2 & 2 & 2\\
4 & -2 & -2 & -2 & -2 & 1 & 1 & 1 & 1\\
0 & 1 & 0 & -1 & 0 & 1 & -1 & -1 & 1\\
0 & -2 & 0 & 2 & 0 & 1 & -1 & -1 & 1\\
0 & 0 & 1 & 0 & -1 & 1 & 1 & -1 & -1\\
0 & 0 & -2 & 0 & 2 & 1 & 1 & -1 & -1\\
0 & 1 & -1 & 1 & -1 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1
\end{bmatrix} \tag{2.70}$$

The rows of M are distributed according to the order of tensor of the 9 moments instead

of the order of moment of the distribution functions. The first three rows correspond to three scalars ρ, e and $\in$ which are the 0th, 2nd and 4th order moments of f_i respectively. The next four rows are related to four vectors j_x, q_x, j_y and q_y respectively where j_x and j_y are the first-order moments and q_x and q_y are the third-order ones. The last two rows are related to the second-order stress tensors which are second-order moments.

The equilibrium moments $\mathbf{m}^{eq}$ in vector form is given by

$$\mathbf{m}^{eq} = \left\{\rho\left(1, -2 + 3\mathbf{u}^2, 1 + 3\mathbf{u}^2, u_x, -u_x, u_y, -u_y, u_x^2 - u_y^2, u_x, u_y\right)\right\}^T \tag{2.71}$$

In moment space the diagonal relaxation matrix R given as

$$\mathrm{R} = \mathrm{diag}\left(1, r_e, r_\in, 1, r_q, 1, r_q, r_\nu, r_\nu\right) \tag{2.72}$$

These relaxation parameters are chosen flexibly $(0 < r < 2)$ while depending on the problem formulation. For the current isothermal flow simulation, they are set as $r_e = r_\in = 1,64$ and $r_q = 1.2$. The parameter r_ν is linked to kinetic viscosity ν as follows:

$$\frac{1}{r_\nu} = \frac{\nu}{C_s^2} + \frac{1}{2} \tag{2.73}$$

The MRT model reduces to the usual lattice BGK equation if all the relaxation parameters are set to single relaxation time τ_ν where $\mathrm{R} = \frac{I}{\tau_\nu}$ (I is the identity matrix).

In LBM, the macroscopic variables are obtained by taking moments of the distribution function. Fluid density ρ and velocity $\mathbf{u}$ are calculated by taking 0th and 1st moments of f_i respectively [91].

$$\rho = \sum_{i=0}^{8} f_i \tag{2.74}$$

$$\rho\mathbf{u} = \sum_{i=0}^{8} f_i \mathbf{e}_i \tag{2.75}$$

2.2.4.3 SRT-LBM model for thermal flows

Peng et al. [18] simplified DDF approach based on SRT-LBM is implemented to simulate heat transfer flow problems. The governing discretised lattice Boltzmann equations for

density and temperature distribution function are given as

$$f_i\left(\mathbf{r}+\mathbf{e}_i\Delta t, t+\Delta t\right) = f_i\left(\mathbf{r},t\right) + \frac{1}{\tau_\nu}\left(f_i^{eq}\left(\mathbf{r},t\right) - f_i\left(\mathbf{r},t\right)\right) + S_i \tag{2.76}$$

To incorporate the buoyancy force in this thermal lattice Boltzmann model, the force term (S) is added to the collision process. A number of LB forcing schemes are proposed in the literature to model external force terms for different thermal problems. Among these, three popular forcing schemes are given as follows:

1. Luo's force model (conventional model) [92]

$$S_i = dt\, w_i \frac{\mathbf{e}_i \cdot \mathbf{G}}{C_s^2} \tag{2.77}$$

2. Shan and Chen force model [93]

$$S_i = dt\, w_i \left\{\frac{\mathbf{e}_i - \mathbf{u}}{C_s^2} + \frac{\mathbf{e}_i\left(\mathbf{e}_i \cdot \mathbf{u}\right)}{C_s^4}\right\} \cdot \mathbf{G} \tag{2.78}$$

3. Guo's force model [17]

$$S_i = dt\, w_i \left\{1 - \frac{1}{2\tau_\nu}\right\}\left\{\frac{\mathbf{e}_i - \mathbf{u}}{C_s^2} + \frac{\mathbf{e}_i\left(\mathbf{e}_i \cdot \mathbf{u}\right)}{C_s^4}\right\} \cdot \mathbf{G} \tag{2.79}$$

Among these three models, Guo's force model is found to be more appropriate one as it can correctly represent the continuity and momentum equations on the macroscopic scale, i.e., Navier-Stokes equation using Chapman-Enskog expansion.

$\mathbf{G}$ is defined as the external force per unit volume due to the presence of buoyancy force. According to the Boussinesq approximation, all the properties of the fluid are considered constants except the density variation in the buoyancy force term. The buoyancy force per unit volume depends linearly on the temperature as follows [94]

$$\mathbf{G} = \rho \mathbf{g} \beta \left(\theta - \theta_m\right) \tag{2.80}$$

where ρ is the density of the fluid at the mean temperature θ_m, $\mathbf{g}$ is the acceleration due to gravity and β is the coefficient of thermal expansion.

For temperature distribution function the governing discretised lattice Boltzmann equa-

tion is given as [18]

$$g_i\left(\mathbf{r}+\mathbf{e}_i\Delta t, t+\Delta t\right)=g_i\left(\mathbf{r},t\right)+\frac{1}{\tau_\theta}\left(g_i^{eq}\left(\mathbf{r},t\right)-g_i\left(\mathbf{r},t\right)\right) \tag{2.81}$$

The τ_ν and τ_θ are single relaxation time (SRT) parameters that determine the kinetic viscosity, $\nu=C_s^2\Delta t\left(\tau_\nu-\frac{1}{2}\right)$ and thermal diffusivity, $\alpha=C_s^2\Delta t\left(\tau_\theta-\frac{1}{2}\right)$ respectively.

2.2.4.4 MRT-LBM model for thermal flows

The governing equation for the MRT-LBM model for flow field with source term is [95]:

$$f_i\left(\mathbf{r}+\mathbf{e}_i\Delta t, t+\Delta t\right)-f_i\left(\mathbf{r},t\right)=-\mathrm{M}^{-1}\mathrm{R}\left\{\mathbf{m}\left(\mathbf{r},t\right)-\mathbf{m}^{eq}\left(\mathbf{r},t\right)\right\}+\mathrm{M}^{-1}\left(\mathrm{I}-\frac{\mathrm{R}}{2}\right)\mathbf{S} \tag{2.82}$$

The diagonal relaxation matrix R in the moment space is given as:

$$\mathrm{R}=\mathrm{diag}\left(1,r_e,r_\in,1,r_q,1,r_q,r_\nu,r_\nu\right) \tag{2.83}$$

For all the thermal flow simulations of this thesis, the relaxation parameters are chosen as $r_e=1.64$, $2>r_\in\geq\frac{2}{1+\mathrm{Pr}}$ and $r_q=1.2$, where Pr is the Prandtl number. The total force term $\mathbf{S}$ is [96]:

$$\mathbf{S}=\left\{\rho\left(0,6\mathbf{u}\cdot\mathbf{F},-6\mathbf{u}\cdot\mathbf{F},F_x,-F_x,F_y,-F_y,2(u_xF_x-u_yF_y),(u_xF_x+u_yF_y)\right)\right\}^T \tag{2.84}$$

where,

$$F_x=0;\quad F_y=\mathbf{G}=\rho\mathbf{g}\beta(\theta-\theta_m) \tag{2.85}$$

For temperature field, the evolution equation for MRT-LBM model is [96]:

$$g_i\left(\mathbf{r}+\mathbf{e}_i\Delta t, t+\Delta t\right)-g_i\left(\mathbf{r},t\right)=-\mathrm{N}^{-1}\mathrm{C}\{\mathbf{n}(\mathbf{r},t)-\mathbf{n}^{eq}(\mathbf{r},t)\} \tag{2.86}$$

For D2Q5 model, the suffix i refers to 5 discrete velocities $\mathbf{e}_i$ for D2Q5 lattice and the velocities are [95]:

$$\mathbf{e}_i=\left\{\begin{array}{ll}(0,0) & i=0\\ c(\cos[(i-1)\pi/2],\sin[(i-1)\pi/2]) & i=1,\cdots,4\end{array}\right\} \tag{2.87}$$

The diagonal relaxation matrix C is [96]:

$$\mathrm{C} = \mathrm{diag}\,(1, c_\theta, c_\theta, c_e, c_\nu) \tag{2.88}$$

The relaxation parameters are chosen as $c_e = c_\nu = 1.8$ in this thesis work. The relaxation parameter c_θ in equation (2.88) is determined by the thermal diffusivity α as:

$$\frac{1}{c_\theta} = \frac{10\alpha}{(4+a)} + \frac{1}{2} \tag{2.89}$$

where the value of constant term a should be less than one to avoid numerical instability.

The transformation matrix N is a 5×5 matrix and it linearly transforms the temperature distribution functions ($\mathbf{g}$) to their moments ($\mathbf{n}$) and vice-versa as follows:

$$\mathbf{n} = \mathrm{N}\mathbf{g}, \quad \mathbf{g} = \mathrm{N}^{-1}\mathbf{n} \tag{2.90}$$

where N is given by

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & -1 \\ -4 & 1 & 1 & 1 & 1 \\ 0 & 1 & -1 & 1 & -1 \end{bmatrix} \tag{2.91}$$

The equilibrium moments $\mathbf{n}^{eq}$ in vector form is [95]:

$$\mathbf{n}^{eq} = (\theta, u\theta, v\theta, a\theta, 0)^T \tag{2.92}$$

The characteristic velocity for mixed convection flow is taken as the lid velocity ($V_0^* = \sqrt{\mathrm{R}i\mathbf{g}\Delta TL}$), whereas, the characteristic velocity of the flow for natural convection flow is $\sqrt{\mathbf{g}\Delta TL}$. The value of characteristic velocity in the lattice unit (V_{lb}) must be small in comparison to the lattice speed of sound C_S ($= 1/\sqrt{3}$) to satisfy the incompressibility condition. In Thermal LBM, the velocity $\mathbf{u}$ is modified by taking into account the force effect as follows [17]:

$$\rho\mathbf{u} = \sum_{i=0}^{8} f_i\mathbf{e}_i + \frac{\mathbf{G}}{2} \tag{2.93}$$

Temperature (θ) is the only conserved quantity which is calculated as 0_{th} moments of g_i

[18] as given below:

$$\theta = \sum_{i=0}^{4} g_i \tag{2.94}$$

2.2.5 Initial and boundary condition treatments in LBM

In order to obtain a proper solution to any flow problem, the initial and boundary conditions (BC) play a vital role. The macroscopic variables are generally directly implemented at these conditions in solving continuum flow problems using computational fluid dynamics techniques such as finite difference or finite volume method. However, in LBM, these physical conditions are needed to be expressed in terms of an only unknown parameter known as the discrete distribution function. Different approaches have been proposed by researchers to obtain the distribution function from known macroscopic variables while maintaining the same physical meaning. To model the initial condition in LBM, the equilibrium dis-

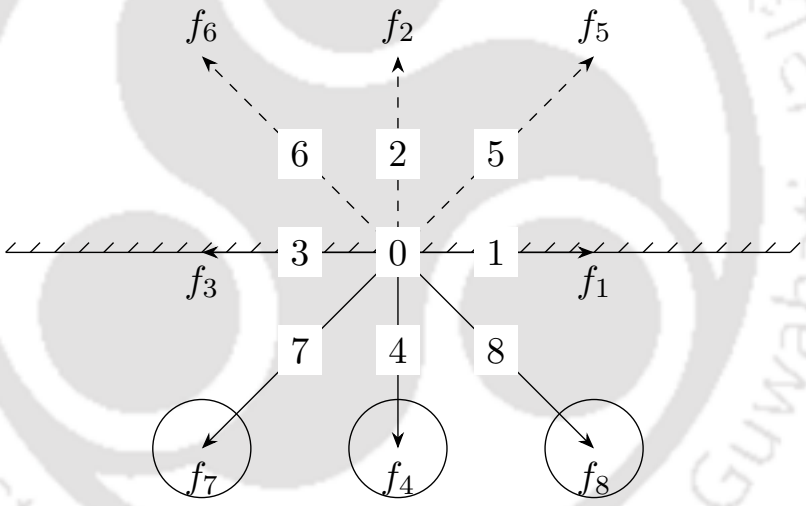


Figure 2.14: *Solid-fluid surface interaction (Top side solid, bottom side fluid).*

tribution function is most widely used [86] to initialise both density- and temperature-distribution function, which is represented as follows:

$$f_i(\mathbf{r}, t) = f_i^{eq}(\mathbf{r}, t); \qquad g_i(\mathbf{r}, t) = g_i^{eq}(\mathbf{r}, t) \tag{2.95}$$

A number of boundary schemes are available in LBM literature. These boundary conditions can be divided into two categories: link-wise and wet-node [83, 85]. The popular bounce-back (BB) model falls under link-wise schemes. This model simulates the no-slip boundary condition on the stationary wall located midway on the link between lattice nodes (termed as a half-way bounce-back scheme). The enforcement of bounce back model

on density distribution function is given as follows:

$$f_i\left(\mathbf{r}_b, \mathbf{e}_i, t\right) = f_i\left(\mathbf{r}_b, -\mathbf{e}_i, t\right) \tag{2.96}$$

where $\mathbf{r}_b$ is the position vector of the boundary node. In this approach, the distribution function is reversed back to its original location with the same speed after meeting a rigid boundary during its streaming.

Ladd et al. [97] proposed a revised half-way bounce-back scheme to incorporate wall velocity into LBM framework as follows:

$$f_i\left(\mathbf{r}_b, \mathbf{e}_i, t\right) = f_i\left(\mathbf{r}_b, -\mathbf{e}_i, t\right) - 2w_i\rho_b\left\{\frac{\mathbf{e}_i \cdot \mathbf{u}_w}{C_s^2}\right\} \tag{2.97}$$

where b and w denote the boundary node and wall node. It is obvious that wall velocity $\mathbf{u}_w = 0$ for a stationary boundary.

On the other hand, in wet-node boundary schemes, the boundary is placed on a boundary node. The most common wet-node schemes are the equilibrium scheme, nonequilibrium extrapolation method and non-equilibrium bounce-back method. In the equilibrium scheme, the unknown distribution functions at the boundary are set equal to their equilibrium values as obtained from the prescribed density and velocity at the boundary.

$$f_i\left(\mathbf{r}_b, t\right) = f_i^{eq}\left(\mathbf{r}_b, \rho_b, \mathbf{u}_w, t\right) \tag{2.98}$$

This BC provides satisfactory accuracy when relaxation time τ approaches 1.

Guo et al. [98] proposed a non-equilibrium extrapolation method (NEEM) based on non-equilibrium distribution functions. The basic idea of this scheme is to decompose the distribution functions at a boundary node into its equilibrium and nonequilibrium parts. The equilibrium part is constructed based on the specified macroscopic boundary conditions, while the non-equilibrium part is approximated with a first-order extrapolation. It is expressed as follows:

$$f_i\left(\mathbf{r}_b, t\right) = f_i^{eq}\left(\mathbf{r}_b, \rho_f, \mathbf{u}_w, t\right) + \left(f_i\left(\mathbf{r}_f, t\right) - f_i^{eq}\left(\mathbf{r}_f, t\right)\right) \tag{2.99}$$

Where $f, \mathbf{r}_f$ and ρ_f represent the fluid node, position vector and density at the neighbouring fluid (inner) node respectively.

Zou and He [99] proposed a non-equilibrium bounce-back method (NEBB), as the name

suggests it is the bounce-back of the non-equilibrium distribution functions.

$$f_i^{neq}(\mathbf{r}_b, \mathbf{e}_i, t) = f_i^{neq}(\mathbf{r}_b, -\mathbf{e}_i, t) \tag{2.100a}$$

$$f_i(\mathbf{r}_b, \mathbf{e}_i, t) - f_i^{eq}(\mathbf{r}_b, \mathbf{e}_i, t) = f_i(\mathbf{r}_b, -\mathbf{e}_i, t) - f_i^{eq}(\mathbf{r}_b, -\mathbf{e}_i, t) \tag{2.100b}$$

In the thesis work, the application of LB boundary schemes is limited to model BC at solid walls. However, different LB boundary schemes are proposed to treat curved boundary [100] and inflow/outflow conditions [101].

For thermal flow conditions, second-order accurate finite difference approximation (Neumann condition) on macroscopic temperature is implemented on thermally insulated walls, and Dirichlet condition is used for constant temperature walls [18]. To transform these temperature conditions to their distribution function (DF) counterpart, a second-order accurate non-equilibrium-extrapolation approach proposed by Guo et al. [102] is applied. In this approach, both equilibrium and non-equilibrium parts of DF are calculated from known temperatures at a boundary node and at a neighbouring fluid node.

$$g_i(\mathbf{r}_b, t) = g_i^{eq}(\mathbf{r}_b, \theta_w, \mathbf{u}_w, t) + (g_i(\mathbf{r}_f, t) - g_i^{eq}(\mathbf{r}_f, t)) \tag{2.101}$$

2.3 Parallelization using Graphics Processing Unit and CUDA

2.3.1 Parallelism using GPU and CUDA in Brief

2.3.1.1 Introduction

Traditionally, most of the computer programs are written sequentially and they only run on one of the processor cores and rely on the advances in hardware in order to achieve increased speed. Therefore, a sequential program will not become significantly faster from generation to generation. Since 2003, energy consumption and heat dissipation issues limited the increase of the clock frequency and the level of productive activities that can be performed in each clock period within a single CPU. Since then, the microprocessor industry moved to multiple processing units (processor cores) in each chip to increase the processing power. The high-performance computing community has been developing parallel programs for decades. These programs typically ran on a large scale, expensive computers. Only a few elite applications could justify the use of these expensive computers, thus limiting the

practice of parallel programming to a small number of application developers. Now that all new microprocessors are parallel computers, the number of applications that need to be developed as parallel programs has increased dramatically.

2.3.1.2 Heterogeneous Parallel Computing

Since 2003, the semiconductor industry has settled on two main streams for designing microprocessors [103]. The *multicore* stream seeks to maintain the execution speed of sequential programs while moving into multiple cores. The multicores began with two-core processors with the number of cores increasing with each semiconductor processor generation. In contrast, the many-thread stream focuses more on the execution throughput of parallel applications. The many-threads began with a large number of threads and the number of threads increases with each generation. For example, the NVIDIA Tesla P100 graphics processing unit (GPU) with thousands of threads, executing in a large number of simple, in order pipelines. As of 2016, the ratio of peak floating-point calculation throughput between many-thread GPUs and multicore CPUs is about 10. These are not necessarily application speeds, but the raw speed that the execution resources can potentially support in these chips.

Such a large performance gap between parallel and sequential execution has motivated many applications developers to move the computationally intensive parts of their software to GPU for execution. The many-threaded GPUs and general-purpose multicore CPUs differ in the fundamental design philosophies between them, as illustrated in figure 2.15. The GPU is specialized for highly parallel computations and therefore designed such that more transistors are devoted to data processing rather than data caching and flow control. The

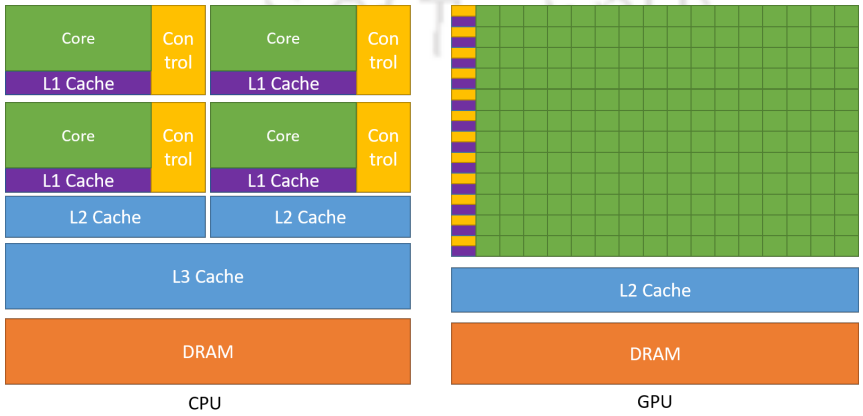


Figure 2.15: *Differences in distribution of chip resouces in a CPU vs a GPU [2].*

design of a CPU is optimized for sequential code performance. It makes use of sophisticated control logic to allow instructions from a single thread to execute in parallel or even out of their sequential order while maintaining the appearance of sequential execution. Large cache memories are provided to reduce the instruction and data access latencies of large complex applications. Neither control logic nor cache memories contribute to the peak calculation throughput.

The speed of many applications is limited by the rate at which data can be delivered from the memory system into the processors (*Memory bandwidth*). The memory bandwidth of graphics chips is approximately 10× more than contemporaneously available CPU chips. A GPU is capable of moving extremely large amounts of data in and out of its main Dynamic Random Access Memory (DRAM) because of graphics frame buffer requirements. In contrast, general-purpose processors have to satisfy requirements from legacy operating systems, applications, and I/O devices that make memory bandwidth more difficult to increase.

GPUs being designed as parallel, throughput-oriented computing engines, will not perform better than CPUs on some tasks which are optimized for CPUs. For programs that have one or very few threads, CPUs with lower operation latencies can achieve much higher performance than GPUs. When a program has a large number of threads, GPUs with higher execution throughput can achieve much higher performance than CPUs. Therefore, many applications use both CPUs and GPUs, executing the sequential parts on the CPU and numerically intensive parts on the GPUs. CUDA programming model, introduced by NVIDIA in 2007, is designed to support joint CPU-GPU execution of an application. The demand for supporting joint CPU-GPU execution is further reflected in more recent programming models such as OpenCL, OpenACC, and C++AMP.

Since 2006, GPU starts supporting the IEEE Floating-Point Standard and as a result, more numerical applications have been ported to GPUs. Till 2009, the GPU floating-point arithmetic units were primarily single precision. However, the recent GPUs support double-precision and their execution speed approaches about half that of single-precision, a level that only high-end CPU cores achieve. Therefore, modern GPUs are suitable for numerical applications.

Until 2006, graphics chips were very difficult to use because programmers had to use the equivalent of graphics application programming interface (API) functions to access the processing units, meaning that OpenGL or Direct3D techniques were needed to program

these chips. A computation must be expressed as a function that paints a pixel in some way in order to execute on these early GPUs. This technique was called GPGPU, for *General-Purpose Programming using a GPU*. Even with a higher level programming environment, the underlying code still needs to fit into the APIs that are designed to paint pixels. These APIs limit the kinds of applications that one can actually write for early GPUs. But everything changed in 2007 with the release of CUDA by NVIDIA. A new general-purpose parallel programming interface on the silicon chip serves the requests of CUDA programs and the programmers can use the familiar C/C++ programming tools.

2.3.1.3 Architecture of a Modern GPU

The NVIDIA GPU architecture is organized around a scalable array of multithreaded *streaming multiprocessors* (SMs). However, the number of SMs in a building block can vary from one generation of GPU to another. Each SM has a number of *streaming processors* (SPs) that share control logic and instruction cache. Each GPU comes with gigabytes of Graphics Double Data Rate (GDDR), Synchronous DRAM (SDRAM), referred to as Global Memory. These GDDR SDRAMs differ from the system DRAMs on the CPU motherboard in that they are essentially the frame buffer memory that is used for graphics. For graphics applications, they hold video images and texture information for 3D rendering. For computing, they function as very high-bandwidth off-chip memory, though with somewhat longer latency than typical system memory. For massively parallel applications, the higher bandwidth makes up for the longer latency.

2.3.1.4 Speed vs Parallelism

When an application is suitable for parallel execution, a good implementation on a GPU can achieve more than 100 times (100×) speedup over sequential execution on a single CPU core. If the application contains "data parallelism", it is possible to achieve a 10× speedup. The amount of speedup obtained from parallelizing an application depends on the portion of the application that can be parallelized and is given by Amdahl's law. It states that in parallelization if P is the portion of a program that can be parallelized and $1 - P$ is the serial (cannot be parallelized) portion of the program, then the maximum speedup $S(N)$ that can be achieved using N processors is:

$$S(N) = \frac{1}{(1-P) + \frac{P}{N}} \tag{2.102}$$

For example, if the amount of time spent in the parallelized portion of a program is 30%, even an infinite amount of processors can only reduce 30% off execution time and thus, achieving a maximum speed up of 1.43×. On the other hand, if 99% of the execution time is in the parallel portion, using 100 CPUs gives the entire application a 52× speedup. Therefore, in order to effectively increase the speed up of an application, it is very important that the application has the majority of its execution in the parallel portion.

Straightforward parallelization of applications can often saturate the memory (DRAM) bandwidth. Therefore, to get around the memory bandwidth limitations, one needs to exploit the different on-chip memories available on the GPUs to drastically reduce the number of accesses to the DRAM. This also involves further optimizing the code to get around limitations such as limited on-chip memory capacity. Most applications have portions where CPUs perform very well and are very difficult to speed up the performance using a GPU. Therefore, the program should be written in such a way that the GPUs complement the CPU, thus properly exploiting the heterogeneous parallel computing capabilities of the combined CPU/GPU system.

2.3.1.5 Challenges in Parallel Programming

Designing a parallel algorithm with the same level of algorithmic (computational) complexity as sequential algorithms can be difficult. Sometimes the parallel algorithms introduce large overheads compared to their sequential counterparts and it leads to slower execution for large input data.

The execution speed of many applications is limited by memory access speed. These are known as memory-bound applications, as opposed to compute-bound, which are limited by the number of instructions performed per byte of data.

The execution speed of parallel programs is often more sensitive to the input data characteristics than their sequential counterparts. Many real-world applications need to deal with inputs with widely varying characteristics, such as erratic or unpredictable data rates, and very high data rates.

Many real-world problems are most naturally described with mathematical recurrences. Parallelizing these problems often requires non-intuitive ways of thinking about the problem and may require redundant work during execution.

2.3.1.6 Parallel Programming Languages and Models

The two most widely used parallel programming interfaces are: i) message passing interface (MPI) [104] for scalable cluster computing systems, and ii) OpenMP [105] for shared memory multiprocessor systems. OpenMP compilers generate parallel code based on the directives (commands) and pragmas (hints) provided by a programmer.

MPI is a model where computing nodes in a high-performance computing (HPC) cluster do not share memory [104]. In MPI all data sharing and interaction are done through explicit message passing. Modern HPC clusters are heterogeneous CPU/GPU nodes. While each node contains GPU cards, the developers need to use MPI to communicate between the nodes (at the cluster level).

Porting an application into MPI requires a lot of effort. First, one needs to perform domain decomposition to partition the input and output data into nodes. After that, based on the domain decomposition, one needs to call message sending and receiving functions in order to exchange the data between nodes. In order to address this difficulty, CUDA provides shared memory functionality for parallel execution in the GPU. CUDA allows declaring variables and data structures as shared between CPU and GPU. The runtime hardware and software transparently maintain coherence by automatically performing optimized data transfer operations on behalf of the programmer as needed. This significantly reduces the programming complexity involved in overlapping data transfer with computation and I/O activities.

2.3.2 CUDA Programming Model

2.3.2.1 CUDA C Program Structure

As shown in the figure 2.16, the NVCC compiler processes a CUDA C program, using the CUDA keywords to separate the host code and device code. The host code is compiled with the host's standard C/C++ compilers and is run as a CPU process. The device code is called *kernels* and marked with CUDA keywords for data-parallel functions. The device code is further compiled by a run-time component of NVCC and executed on a GPU device. If there is no hardware available for a kernel, it can be executed on a CPU.

The execution of a heterogeneous CUDA program is illustrated in figure 2.17. The execution starts with host code (CPU serial code). When a kernel function (parallel device code) is called, it is executed by a large number of *threads* on a device. All the threads generated by a kernel are collectively called a *grid*. *Threads* are the primary vehicle of

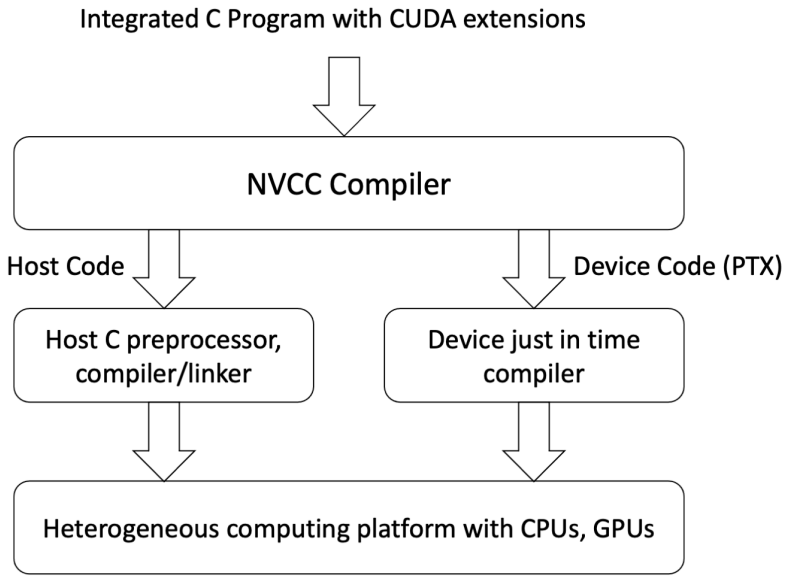


Figure 2.16: *Overview of the compilation process of a CUDA C Program [3].*

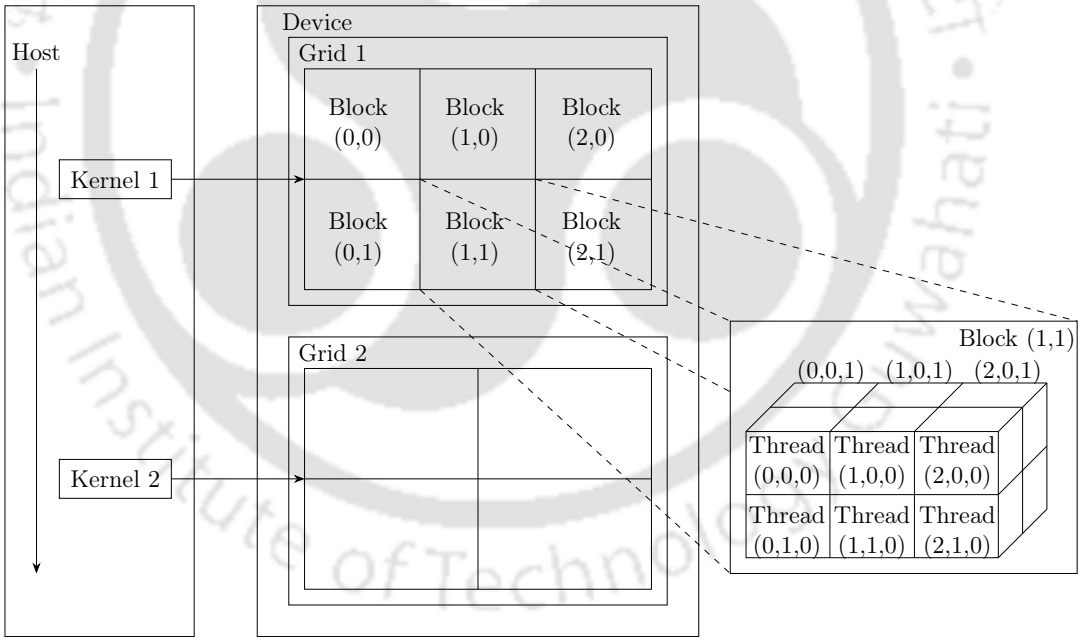


Figure 2.17: *Execution of a heterogeneous CUDA program [3].*

parallel execution in CUDA. Figure 2.17 shows the execution of two grids of threads. When all threads of a kernel complete their execution, the corresponding grid terminates, the execution continues on the host until another kernel is launched. Heterogeneous computing applications normally overlap the CPU and GPU execution to take advantage of both CPUs and GPUs.

2.3.2.2 A few important CUDA terminologies

2.3.2.2.1 CUDA function declarations CUDA C extends the C function declaration syntax to support heterogeneous parallel computing. The extensions are summarized in Table 2.3. Using one of "`__global__`" "`__device__`", or "`__host__`", a CUDA C programmer can instruct the compiler to generate a kernel function, a device function, or a host function. All function declarations without any of these keywords default to host functions. If both "`__host__`" and "`__device__`" are used in a function declaration, the compiler generates two versions of the function, one for the device and one for the host.

Table 2.3: CUDA C keywords for function declaration

	Executed on the:	Only callable from the:
`__device__` float DeviceFunc()	device	device
`__global__` void KernelFunc()	device	host
`__host__` float HostFunc()	host	host

2.3.2.2.2 Device Global Memory and Data Transfer Currently, the GPU cards have their own dynamic random access memory (DRAM). For a kernel execution on a device, the global memory needs to be allocated on the device. The data is then transferred from the host memory to the allocated device memory. Similarly, after the kernel execution is finished, the data from the device memory needs to be transferred back to the host memory. The device memory that is no longer needed is freed up. Following CUDA run-time system API functions perform these activities:

- `cudaMalloc`: It is called from the host code to allocate device global memory for an object.
- `cudaFree`: It frees up the storage from the global memory.
- `cudaMemcpy`: It transfers data between host and device memories. The cudaMemcpy function takes four parameters. The first parameter is a pointer to the destination location for the data object to be copied. The second parameter points to the source location. The third parameter specifies the number of bytes to be copied. The fourth parameter indicates the types of memory involved in the copy: from host memory to host memory, from host memory to device memory, from device memory to host memory, and from device memory to device memory.

2.3.2.2.3 Kernel Launch CUDA C kernels are surrounded by <<< and >>>. These execution configuration parameters are only used during a call to a *kernel function*, or a *kernel launch*. The first parameter within <<< and >>> is the total number of *block*s and the second parameter is the total number of *thread*s in each *block*. For example, in the following kernel function, there are 4 *block*s and each *block* consists of 256 *thread*s. Hence, the *grid* consists of total 1024 *thread*s.

```
vecAddKernel<<<4, 256>>>(A, B, C, N);
```

2.3.2.2.4 Threads A thread is a simplified view of how a processor executes a sequential program in a computer. A thread consists of the code of the program, the particular point in the code that is being executed, and the values of its variables and data structures. Since all these threads execute the same code, the CUDA programming style is the Single-Program Multiple-Data (SPMD) parallel programming. A CUDA parallel execution is initiated by launching *kernel functions*, which creates a lot of threads that operate on different parts of the data in parallel.

2.3.2.2.5 Built-In Variables CUDA kernels can access a set of built-in, predefined read-only variables that allow each thread to distinguish among themselves and to determine the area of data each thread is to work on. A few built-in variables are: `threadIdx`, `blockIdx`, `blockDim` and `gridDim`.

2.3.2.2.6 CUDA Thread Organization All CUDA threads in a grid execute the same kernel function; they rely on coordinates to distinguish themselves from one another and identify the appropriate portion of data to process. These threads are organized into *grid* and *block*: a *grid* consists of one or more *block*s, and each *block* consists of one or more *thread*s. All threads in a block share the same block index, which is the value of the `blockIdx` variable in a kernel. Each thread has a thread index, which can be accessed as the value of the `threadIdx` variable in a kernel. When a thread executes a kernel function, references to the `blockIdx` and `threadIdx` variables return the coordinates of the thread. The execution configuration parameters in a kernel launch statement specify the dimensions of the grid and the dimensions of each block. These dimensions are the values of the variables `gridDim` and `blockDim` in kernel functions.

In general, a grid is a three-dimensional array of blocks, and each block is a three-dimensional array of threads. When launching a kernel, the program needs to specify the

size of the grid and blocks in each dimension. The programmer can use fewer than three dimensions by setting the size of the unused dimensions to 1. The exact organization of a grid is determined by the execution configuration parameters (within `<<< >>>`) of the kernel launch statement. The first execution configuration parameter specifies the dimensions of the grid in the number of blocks. The second specifies the dimensions of each block in the number of threads. Each such parameter is of the `dim3` type, which is a C `struct` with three unsigned integer fields: x, y, and z.

Figure 2.17 illustrates how CUDA threads are arranged in a *grid*. In this example, the first grid consists of 6 blocks arranged in a two-dimensional fashion. There are three blocks along the x-direction and two along the y-direction.

Threads within a block cooperate by sharing data through shared memory and by synchronizing their execution to coordinate memory accesses. The synchronization points in the kernel can be specified by calling the `__syncthreads()` function. `__syncthreads()` acts as a barrier at which point all threads in the block must wait before proceeding.

2.3.2.2.7 Memory Hierarchy Figure 2.18 shows the overview of the CUDA memory hierarchy. There are four kinds of memory that are mostly used in numerical computation in CUDA: global memory, constant memory, shared memory and register. As in figure 2.18, global memory and constant memory are used to communicate with the host device.

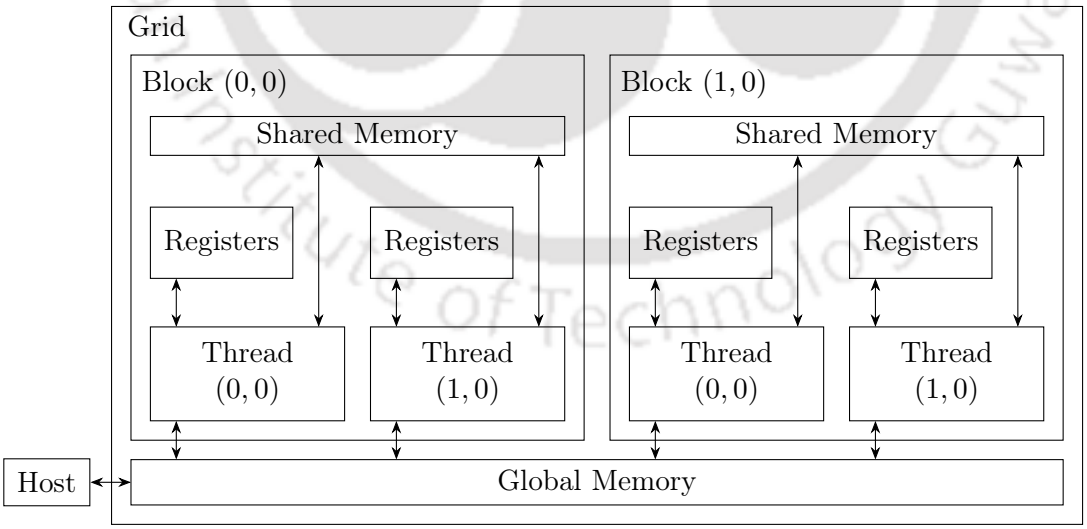


Figure 2.18: *CUDA memory hierarchy [3]*

Global memory is of read and write type in a kernel while constant memory is read-only. They are persistent across kernel launches by the same application. However, access to the constant memory is much faster than the global memory since it can be cached.

Shared memory is fast and is designed for data communication for threads in a block. Shared memory is visible to all threads of the block and with the same lifetime as the block. All register is private to threads and is the fastest memory in CUDA. It is used for the most frequently used data in the program. Since memory access contributes a lot to the computation time of the program, developers should take advantage of different kinds of memory. The key rule is that registers and share memory should be used as much as possible and data that do not to be modified in the execution of the program should be stored in the constant memory for faster access.

2.3.2.3 Basic Optimizations

Memory management is one of the most important considerations when using CUDA for GPU computing. The memory access of the registers and shared memory are fast, and they normally do not have a significant impact on overall performance. However, accessing the global memory is slow and has latencies of several hundred clock cycles. Retrieving data from the global memory is sensitive to the memory access patterns. Therefore, optimizing the access pattern of the global memory is critical in improving the performance of a CUDA program running on a GPU. This is especially the case for memory-bound algorithms such as the LBM.

Chapter 3

Multigrid Assisted Streamfunction Velocity $(\psi - V)$ Formulation

3.1 Introduction

Primitive variables and streamfunction-vorticity $(\psi - \omega)$ formulations of the N-S equations are the two most widely used traditional methods to solve incompressible viscous fluid flows. Primitive variables formulation makes an accurate representation of the fluid flow phenomena, but the occurrence of the pressure term in the momentum equations makes it computationally expensive and time-consuming. The $\psi - \omega$ formulation has the benefit of the computational economy, and the lack of computationally intensive pressure term makes it a popular choice for the CFD practitioners [4] for flows in 2D. However, implementation of the vorticity boundary conditions sometimes may be problematic as there is a difference in the level of accuracy in the numerical approximations at the interior and at the boundary. Frequently one uses higher-order finite difference approximation at the interior points and lower-order approximation at the boundary.

The last two decades have observed an increase in popularity of the $\psi - V$ formulation over the $\psi - \omega$, as it has to deal with a single fourth-order partial differential equation in streamfunction and its gradients [50, 51, 55]. The fourth-order biharmonic form of the N-S equation can be discretized with a compact nine-point stencil with second- and fourth-order accuracy [58, 50, 54]. The use of this method exactly satisfies the boundary conditions and hence removes the drawback of $\psi - \omega$ formulation where vorticity is required to be approximated at the boundaries. Moreover, owing to the compact nature of the approximation no special care at grid points near the boundaries is required [50, 54]. The resulting

system of linear equations after discretization with the nine-point second-order compact scheme is not diagonally dominant and when solved using a standard iteration method like Gauss-Seidel (GS) or Jacobi suffers from slow convergence [50]. As a remedy, researchers [54, 59, 60, 64] have used the biconjugate gradient stabilized (BiCGStab) method [106] to solve the algebraic system of equations.

The multigrid method is known to be one of the most general convergence-acceleration techniques, which is why it is frequently used in conjunction with a finite difference, finite volume and even finite element discretizations. The convergence of a linear or nonlinear iterative procedure can be accelerated using a multigrid method which uses a sequence of coarser grids [6, 25, 76, 1, 27]. As an alternative to the BiCGStab method, multigrid can be used to mitigate the slow convergence problem of standard Jacobi like iteration methods. Tian and Yu [60] developed a new second-order compact finite difference scheme based on the $\psi - V$ formulation and accelerated the streamfunction with the aid of multigrid. Later on, Santiago et al. [107] investigated the performance of the geometric multigrid method using a number of test cases.

The objective of this chapter is to propose a computationally efficient solution procedure that also produces a highly accurate solution. We use geometric multigrid to achieve computational efficiency and choose the $\psi - V$ formulation to obtain high accuracy. It may be noted that the possibility of application of Dirichlet boundary condition for just one variable, i.e. streamfunction plays an important role in its inherent accuracy. For these reasons, we use this combination to compute various fluid flow problems with remarkable ease and accuracy. For the staggered cavity flow and the multiple stable solution problems, the present solution procedure has been used for the first time. We have adopted the second-order nine-point compact-scheme reported by Gupta and Kalita [54]. We show that Gauss-Seidel (GS) procedure, despite being known for relatively slow convergence, can be very effective as a geometric multigrid smoother and that the combination of $\psi - V$ formulation and multigrid results in a powerful computational procedure.

We have reproduced the multiple stable solution results of Wahba [48] to show the ability, speed and accuracy of the multigrid accelerated streamfunction-velocity method. Furthermore, we have also shown that for all Reynolds numbers the symmetric solution exists, whereas the multiplicity of states of one symmetric and two asymmetric solutions for the two-sided and four-sided cavity flows are identified above the critical Reynolds number. Here we have captured multiple steady solutions after the critical Reynolds numbers

for both the two-sided and four-sided lid-driven cavity flows. The computation strategy employed to obtain these solutions is described in a following section.

3.2 Computational Methodology

3.2.1 Streamfunction-velocity formulation

The non-dimensional unsteady two-dimensional incompressible Navier-Stokes equations in the traditional primitive variable form are:

$$u_x + v_y = 0, \tag{3.1}$$

$$u_t + uu_x + vu_y = -p_x + \frac{1}{\mathrm{Re}}\nabla^2 u, \tag{3.2}$$

$$u_t + uv_x + vv_y = -p_y + \frac{1}{\mathrm{Re}}\nabla^2 v, \tag{3.3}$$

where Re is the Reynolds number, p is the pressure, u and v are velocities along x- and y-directions respectively and ∇^2 is the Laplacian operator. The appearance of the pressure term in equations (3.2) and (3.3) makes the solution of this formulation difficult and computationally time-consuming. To avoid the pressure term, for two-dimensional problems, the $\psi - \omega$ formulation of the N-S equations has been extensively used for a long time. The non-dimensional form of the formulation is given by

$$\psi_{xx} + \psi_{yy} = -\omega\,(x, y)\,, \tag{3.4}$$

$$\omega_t + u\omega_x + v\omega_y = \frac{1}{\mathrm{Re}}\left(\omega_{xx} + \omega_{yy}\right), \tag{3.5}$$

where ψ and ω are streamfunction and vorticity respectively. u and v are velocities along x- and y-directions respectively and are defined as

$$u\,(x, y) = \psi_y, \qquad v\,(x, y) = -\psi_x. \tag{3.6}$$

At the boundaries, the no-slip condition ensures that the stream function ψ has a constant value, usually assumed zero. The boundary conditions for ω are derived in terms of ψ using equation (3.4).

The streamfunction-velocity form of the steady-state N-S equations can be obtained by substituting equation (3.4) into equation (3.5) and after simplification we get the fourth-

order partial differential equation in ψ as [54]

$$\frac{\partial^4\psi}{\partial x^4} + 2\frac{\partial^4\psi}{\partial x^2\partial y^2} + \frac{\partial^4\psi}{\partial y^4} - \mathrm{Re}\, u\left(\frac{\partial^3\psi}{\partial x^3} + \frac{\partial^3\psi}{\partial x\partial y^2}\right) - \mathrm{Re}\, v\left(\frac{\partial^3\psi}{\partial x^2\partial y} + \frac{\partial^3\psi}{\partial y^3}\right) = 0.$$

The second-order compact finite difference discretization for the biharmonic form of ψ can be summarized as follows

$$\begin{aligned}&\psi_{i+1,j+1} + \psi_{i-1,j+1} + \psi_{i-1,j-1} + \psi_{i+1,j-1} + 28\psi_{i,j} - 8(\psi_{i+1,j} + \psi_{i,j+1} + \psi_{i-1,j} + \psi_{i,j-1})\\ &- 0.5\mathrm{Re}\, h^2 v_{i,j}(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1}) + 0.5\mathrm{Re}\, h^2 u_{i,j}(v_{i+1,j} + v_{i-1,j} + v_{i,j+1} + v_{i,j-1})\\ &- 3h(u_{i,j-1} - u_{i,j+1} + v_{i+1,j} - v_{i-1,j}) = 0,\end{aligned} \tag{3.7}$$

and the velocities appearing in the equation (3.7) are evaluated using fourth-order approximations as given below [54]

$$u_{i,j} = \frac{3}{4h}(\psi_{i,j+1} - \psi_{i,j-1}) - \frac{1}{4}(u_{i,j+1} + u_{i,j-1}), \tag{3.8}$$

$$v_{i,j} = \frac{3}{4h}(\psi_{i-1,j} - \psi_{i+1,j}) - \frac{1}{4}(v_{i+1,j} + v_{i-1,j}). \tag{3.9}$$

The detailed derivation is provided in the Appendix A

3.2.2 Solution Procedure

This section illustrates the solution procedure of the algebraic systems stemming from the finite-different approximations of the $\psi - V$ formulation. When the conventional iterative methods are used to solve the above algebraic equations, they exhibit slow convergence. To overcome this, equations (3.7 – 3.9) are solved in [54] in an outer-inner iteration manner. First the streamfunction (equation (3.7)) is solved using BiCGStab and then velocities (equations (3.8) and (3.9)) are solved using tri-diagonal matrix algorithm (TDMA). These steps consist of one outer iteration. Then this procedure is repeated till convergence.

To improve the convergence even further, Tian and Yu [60, 61] employed the multigrid method in order to solve the sparse linear systems. Being independent of the grid size multigrid method very efficiently solves the sparse systems of equations stemming from the elliptic PDEs [76, 1, 27]. Multigrid uses a hierarchy of coarse grids to compute corrections by rapidly smoothing out different error components. The notation Ω^h and Ω^H can be used to denote a fine and a coarse grid, respectively. The multigrid method consists of the following three main components [1]:

1. **Restriction:** Transferring the solution variables and residual from the fine to the coarse grid, (I_h^{2h}). Two commonly used restriction operations are injection and full-weighting.

2. **Smoothing:** Calculation of the new solution variables on the fine or coarse grid using a solver, e.g., Gauss-Seidel, which quickly removes the high-frequency error components. The smoothing operation of multigrid can be split into three parts, (i) presmooth (ii) coarsesmooth, and (iii) postsmooth.

3. **Prolongation:** Interpolation of the solution variables from the coarse to the fine grid, (I_{2h}^{h}). Bilinear interpolation is a common choice of prolongation.

Different multigrid cycles are available, for instance, in a V-cycle we move down from the finest to the coarsest grid and then move back up from the coarsest to the finest grid. Presmooth sweeps consist of one, two or three sweeps and are carried out at each grid level except on the coarsest grid before the residuals are transferred to the next coarser grid. Similarly, the postsmooth sweeps also consist of one or two sweeps and are carried out after performing the prolongation operation, i.e., after interpolating the coarse-grid corrections back to the next finer grid. On the coarsest grid, the algebraic equations can either be solved up to the machine precision or a certain number of sweeping steps can be fixed which are called here as coarsesmooth. A multigrid V(1,1)-cycle means a multigrid V-cycle with one presmooth and one postsmooth operation [76, 1, 27].

The present work has adopted a multigrid V-cycle with full-weighting as restriction operation, Gauss-Seidel as smoother to solve the equation (3.7) and bilinear interpolation as prolongation operation. Moreover, at the coarsest grid, rather than solving up to machine precision, a certain number of sweeps are fixed. The equations (3.8) and (3.9) are solved using the tri-diagonal matrix algorithm. Hence, the equations (3.7), (3.8) and (3.9) are solved in an outer-inner fashion. All the computations are carried out on an Intel i5-4690K CPU having 16GB RAM and with Intel Compiler 19.1 (package 166). All times mentioned in this thesis are either CPU time or GPU time required for the completion of the computation. To calculate the CPU time, we recorded the system time twice. Once, just before the start of the computation and secondly, just after the completion of the computation and then calculated their differences. This gave us the exact time required for the completion of a computation. The GPU time has been also calculated in a similar fashion. However, for GPU time, this difference also includes the memory transfer time between the CPU and GPU along with the execution time of the kernels. In some of

the tables, even if we have mentioned only time it means CPU time only. In order to demonstrate the ease of use and benefits in terms of speed and accuracy of the multigrid assisted $\psi - V$ formulation, we have chosen four test cases in the following sections.

3.3 Two-Dimensional Lid-Driven Cavity

The popular one-sided lid-driven cavity with a moving upper wall, as shown in Figure 3.1 is chosen to validate and check the performance of our code. The domain is chosen as a unit

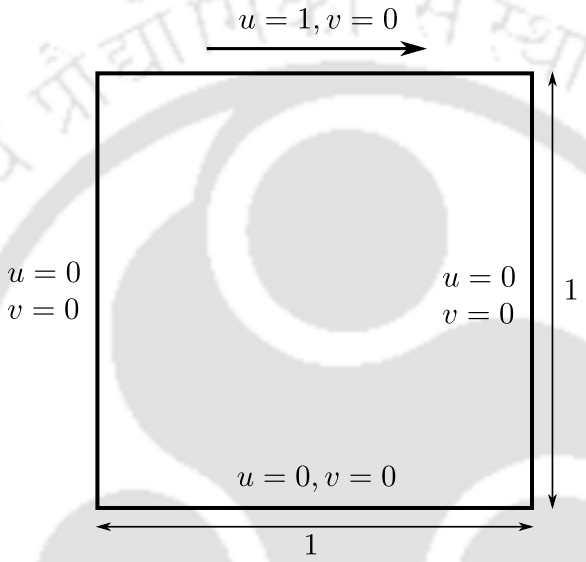
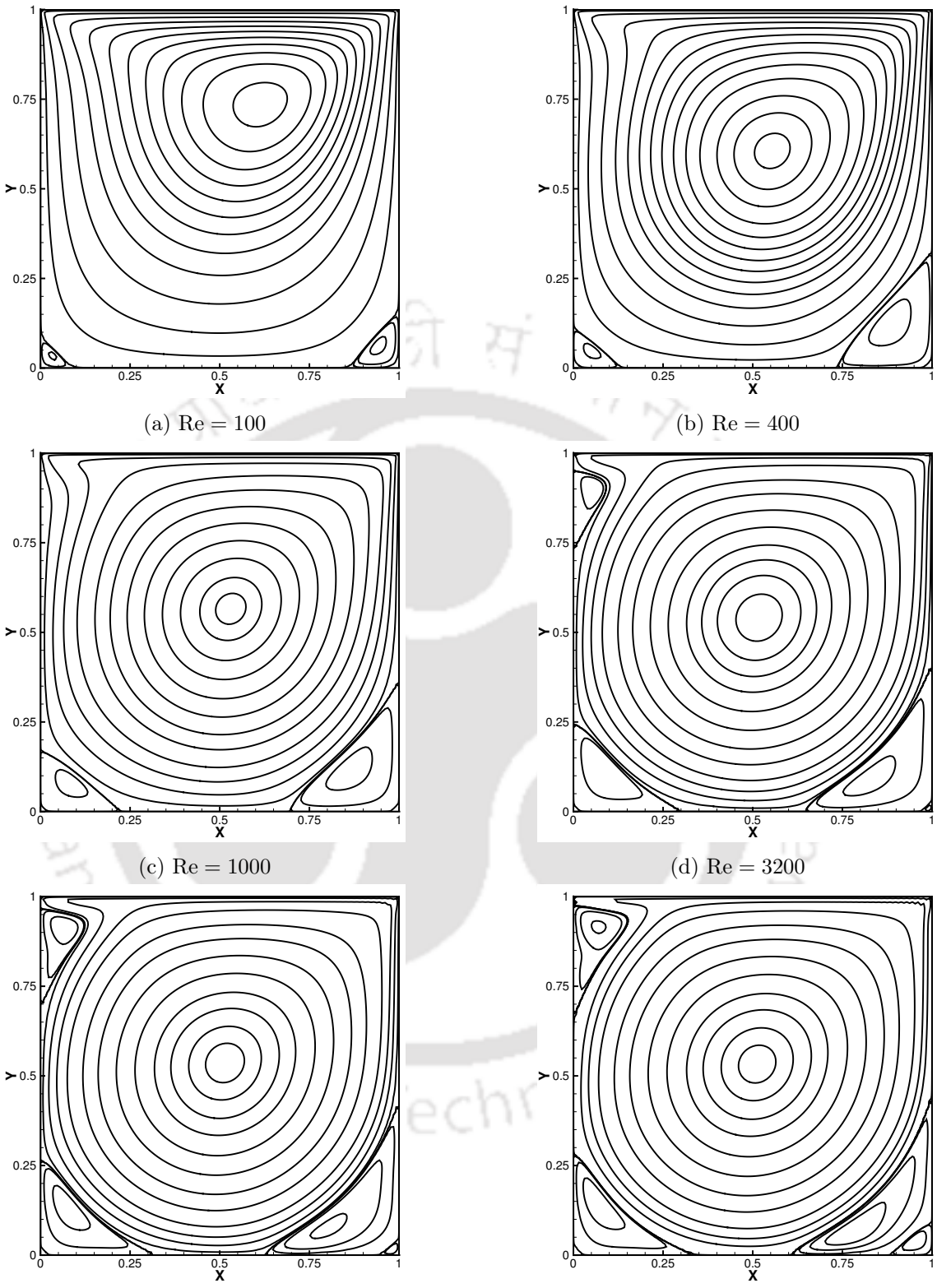


Figure 3.1: *Computational domain of lid-driven cavity.*

square $0 \leq x, y \leq 1$ [54]. The top wall of the cavity ($y = 1$) moves from left to tight with a unit velocity ($u = 1, v = 0$). The velocities on the other walls are zero ($u = 0, v = 0$). For the $\psi - V$ based computation ψ is taken as zero at all the walls.

Equation (3.7) has been solved at inner iteration using five-level multigrid V-cycle that uses grids of 9×9, 17×17, 33×33, 65×65 and 129×129. Gauss-Seidel has been used as smoother. At the coarsest grid, 50 number of iterations is fixed, and at every inner iteration, one multigrid cycle is used. Then u and v are computed at the grid points using equations (3.8) and (3.9) with TDMA. That completes one outer iteration. These steps are repeated until the maximum ψ-error between two successive outer iterations is smaller than 10^{-8}.

Figure 3.2 shows the streamlines for different Reynolds numbers. The comparison of u-velocity along a vertical line through the geometric centre for different Reynolds numbers has been plotted in Figures 3.3a - 3.3d. The results are found to match well with that of the published results of Ghia et al. [4]. Similarly, in Figures 3.4a - 3.4d the comparison

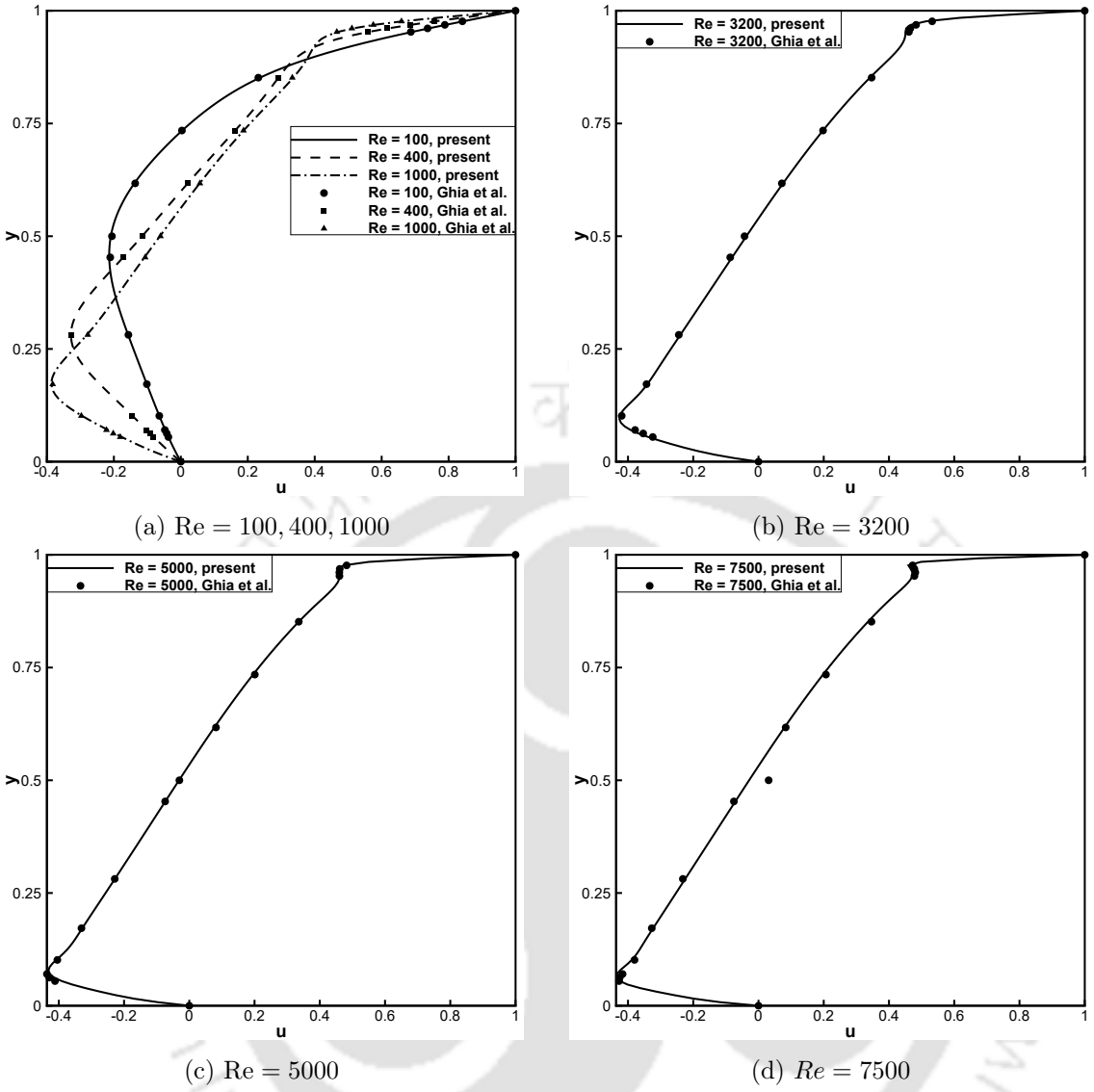


(a) Re = 100 (b) Re = 400

(c) Re = 1000 (d) Re = 3200

(e) Re = 5000 (f) Re = 7500

Figure 3.2: *Streamlines for lid-driven cavity flow for different Reynolds number on a* 129×129 *grid.*



(a) Re = 100, 400, 1000

(b) Re = 3200

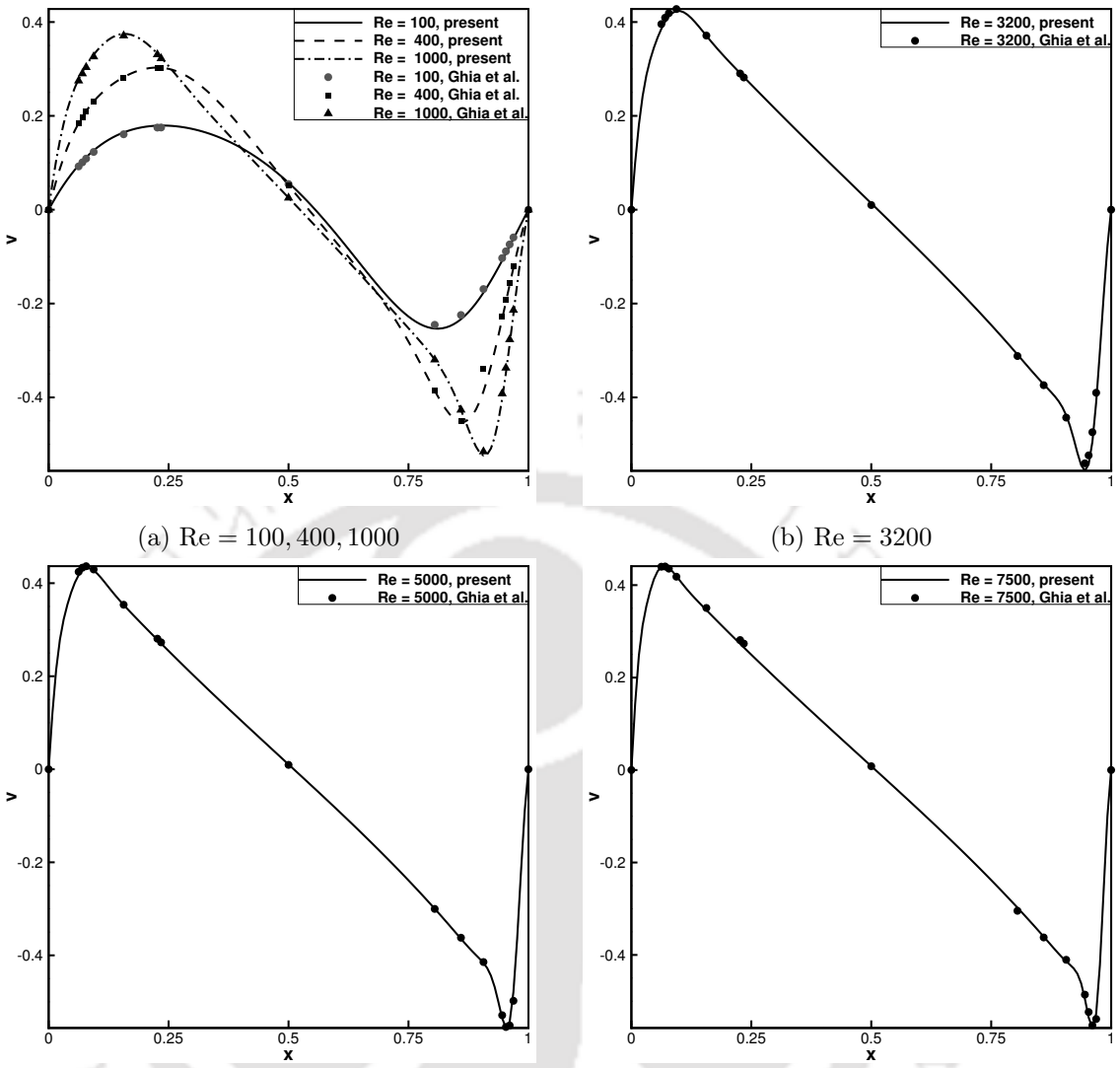
(c) Re = 5000

(d) *Re* = 7500

Figure 3.3: *Comparison of u-velocity along a vertical line through geometric centre for lid-driven cavity flow on a* 129×129 *grid for different Re.*

of v-velocity along a horizontal line through the geometric centre for different Reynolds numbers has been plotted, which shows good agreement with the result of Ghia et al. [4].

To compare the performance of multigrid with respect to single-grid methods, the problem is first solved on a single grid using BiCGStab as the algebraic-equation solver, as BiCGStab is supposed to be a very efficient method. The present multigrid code uses Gauss-Seidel as the smoother and achieves substantial acceleration over the single-grid solver, as can be seen from Table 3.1. The table shows that multigrid converges very fast. For instance, for the Reynolds numbers 100 and 400 the solution is obtained within 2 seconds, whereas the single-grid solution with BiCGStab obtains the solution for these Reynolds numbers in 86 and 92 seconds respectively. For Re = 1000 multigrid takes only



(a) Re = 100, 400, 1000

(b) Re = 3200

(c) Re = 5000

(d) Re = 7500

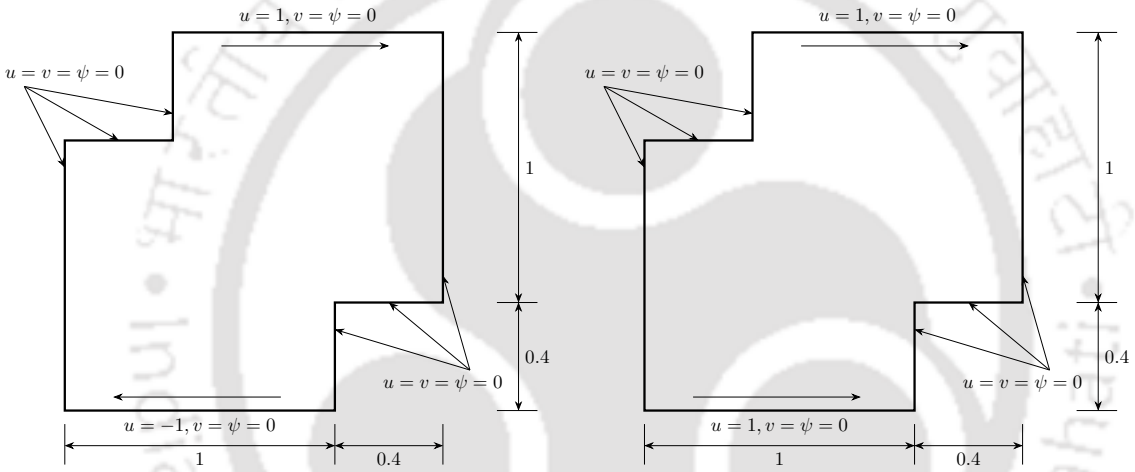
Figure 3.4: *Comparison of v-velocity along a horizontal line through geometric centre for lid-driven cavity flow on a* 129×129 *grid for different Re.*

Table 3.1: Time comparison of multigrid and single-grid on a 129×129 grid for lid-driven cavity flow problem.

Re	Time Taken (s)	
	Multigrid	Single-grid
100	1.833	86.341
400	1.924	92.123
1000	6.472	312.159
3200	29.102	1504.233
5000	73.185	4250.116
7500	262.7053	18053.088

6.5 seconds and the single-grid solver takes 312 seconds. As the Reynolds number increases convergence time for both multigrid and single-grid increases; however, for all Reynolds numbers multigrid takes substantially less time to converge compared to the single-grid. For this particular problem, speed-up achieved by multigrid is seen to progressively increase with Reynolds number. For example at Re = 100, the speed-up is 47, which increases to 68.7 at Re = 7500. Thus the present multigrid strategy achieves extremely high convergence acceleration.

3.4 Two-sided Staggered Lid-Driven Cavity

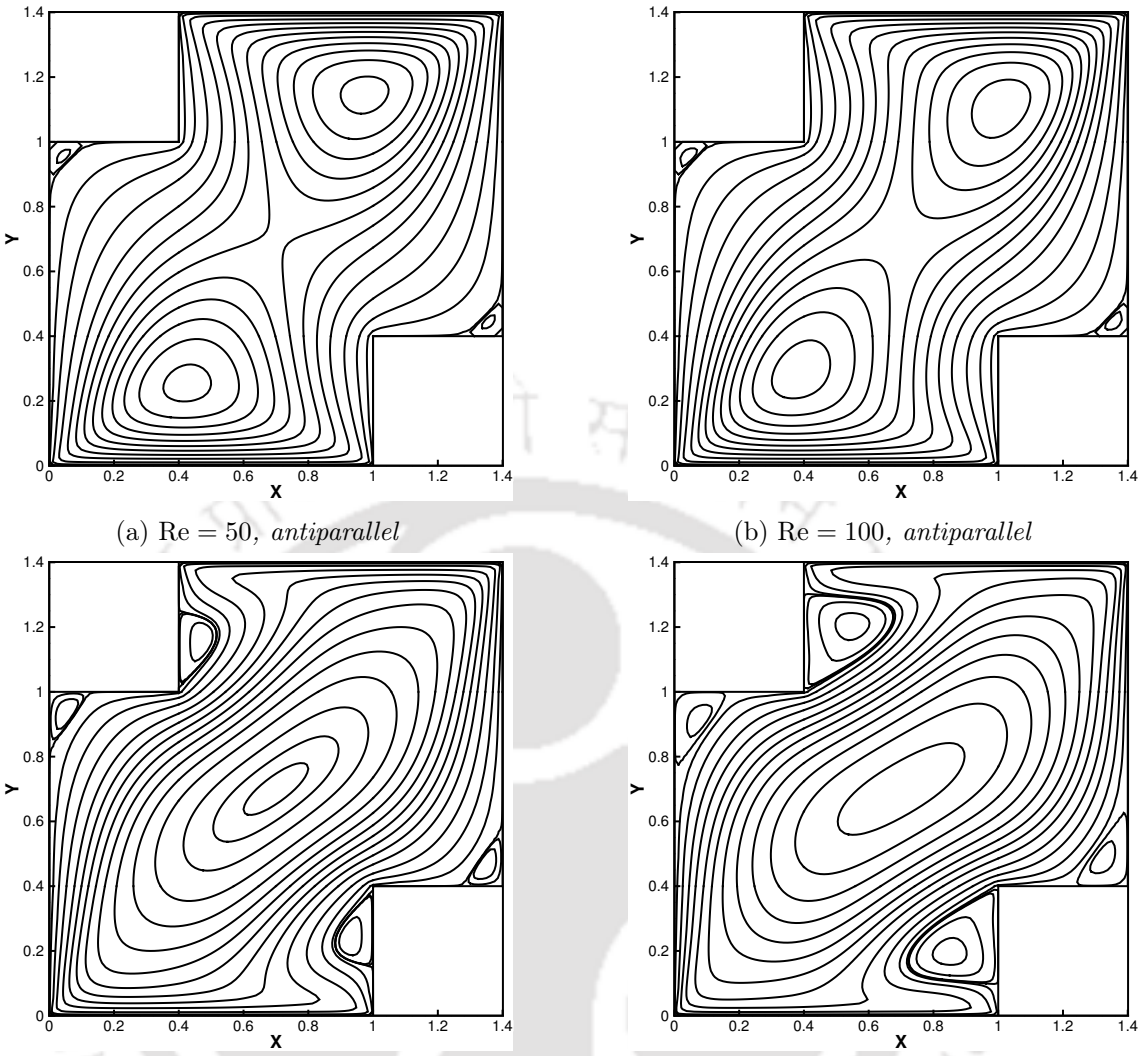


(a) *Antiparallel Motion* (b) *Parallel Motion*

Figure 3.5: *Computational domain of two-sided staggered lid-driven cavity.*

The computational domain of a staggered two-sided lid-driven cavity which is diagonally symmetric, as shown in figure 3.5, with its top and bottom walls (lids) moving at equal velocity [43, 44, 45, 46]. In the staggered cavity, two unit-square lid-driven cavities have been superimposed and are diagonally offset by 40%. For the parallel motion, Figure 3.5b, both the top ($y = 1$) and bottom ($y = 0$) walls move from left to right with unit velocity ($u = 1, v = 0$). Whereas, for the antiparallel motion, Figure 3.5a, the bottom wall moves from right to left with unit velocity ($u = -1, v = 0$). At all the other six walls, the velocities are assumed to be zero ($u = 0, v = 0$). For the $\psi - V$ based computation ψ is taken as zero at all the walls (Figure 3.5) for both the cases. The governing equations are given by (3.7), (3.8) and (3.9) and the boundary condition is depicted as in Figure 3.5.

To facilitate easy implementation of multigrid method to the streamfunction, equation (3.7), grids of 113×113, 57×57, 29×29, 15×15 and 8×8, i.e., five multigrid levels are



(a) Re = 50, *antiparallel*

(b) Re = 100, *antiparallel*

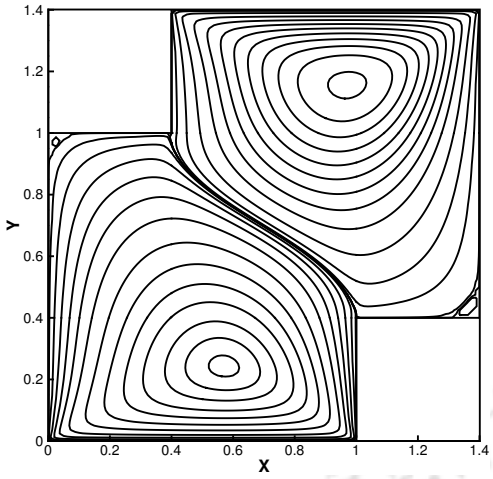
(c) Re = 400, *antiparallel*

(d) Re = 1000, *antiparallel*

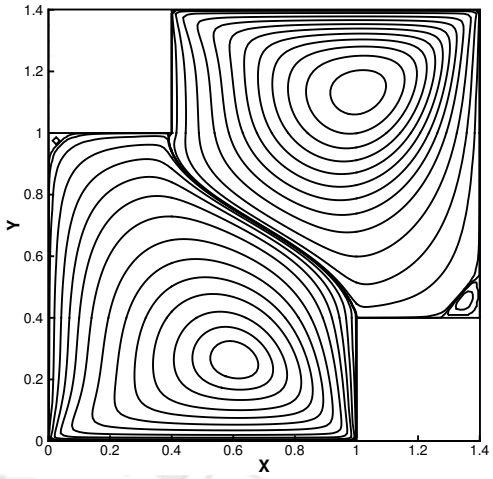
Figure 3.6: *Streamlines for different Reynolds numbers for the antiparallel motion on a* 129×129 *grid for a two-sided staggered lid-driven cavity.*

chosen. Gauss-Seidel has been used as multigrid smoother. At every inner iteration, one multigrid V-cycle is used. Then u and v are computed at the nodes from equations (3.8) and (3.9) using TDMA. It completes one outer iteration. These steps are repeated until the maximum ψ-error between two successive outer iterations was smaller than 10^{-7}.

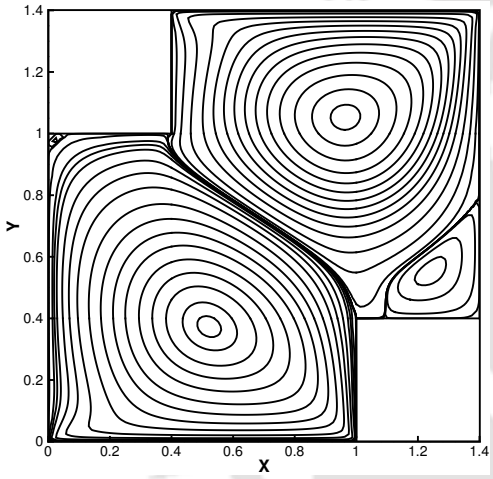
Figures 3.6a - 3.6d show the streamlines for the antiparallel motion of the staggered cavity for Re $= 50, 100, 400$ and 1000. Two secondary vortices appear on the top left and bottom right sides of the cavity for Re $= 400$ and 1000. With an increase in Re, the corner vortices grow in size. The comparison of u-velocity along a vertical line and v-velocity along a horizontal line through geometric centre for different Re have been plotted in the Figures 3.8a and 3.8b for the antiparallel motion. The results were found to match well



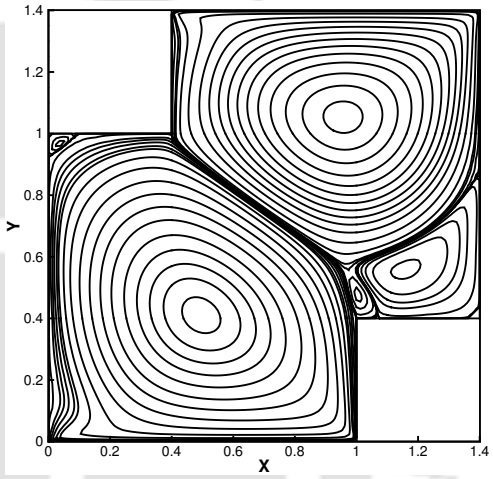
(a) Re = 50, *parallel*



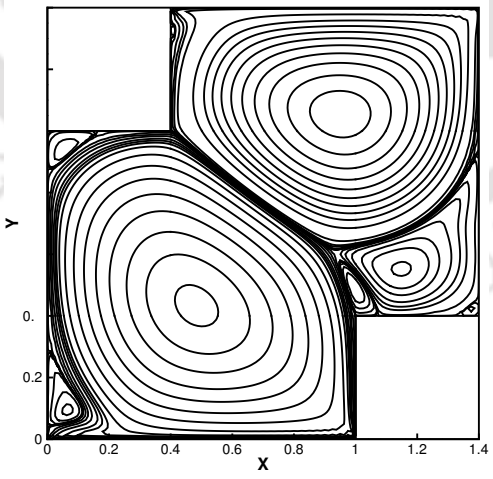
(b) Re = 100, *parallel*



(c) Re = 400, *parallel*



(d) Re = 1000, *parallel*



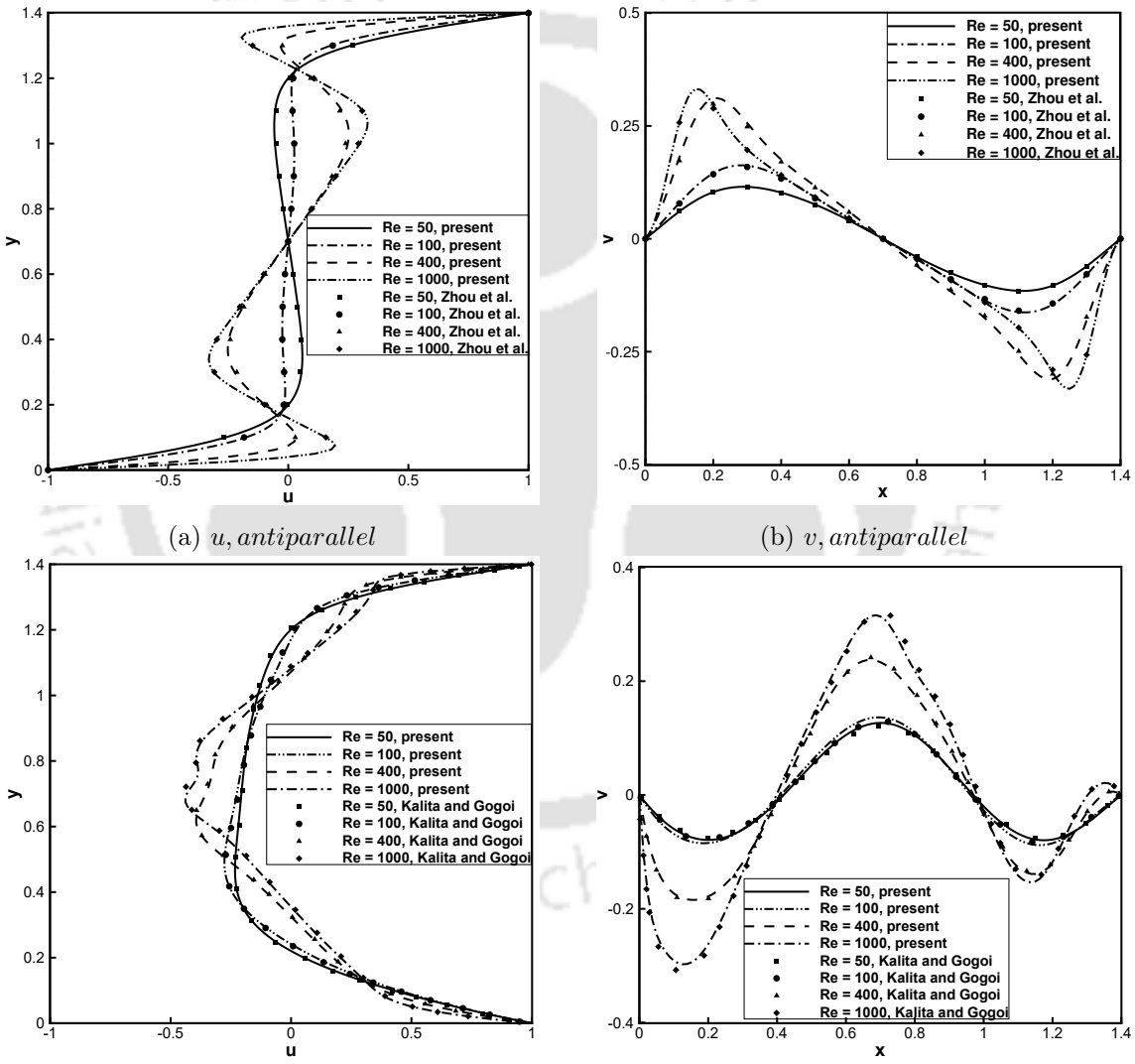
(e) Re = 3200, *parallel*

Figure 3.7: *Streamlines for different Reynolds numbers for the parallel motion on a* 129×129 *grid for a two-sided staggered lid-driven cavity.*

with that of the published results of Zhou et al. [43].

Figures 3.7a - 3.7e show the streamlines for the parallel motion of the staggered cavity

for Re = 50, 100, 400, 1000 and 3200. The secondary vortices are visible for all the Reynolds numbers and the bottom right vortex grows in size considerably with the increase in the Reynolds number. For Re = 1000 and 3200, additional vortices appear. In case of the parallel motion, no symmetry flow pattern was observed. Unlike antiparallel motion, for the whole range of Re, the presence of two primary vortices can be observed. In the Figures 3.8c and 3.8d the comparison of v-velocity along a horizontal line through geometric centre for different Re has been plotted for parallel motion of the lids, and it is in good agreement with the result of Zhou et al. [43].



(a) $u, antiparallel$ (b) $v, antiparallel$

(c) $u, parallel$ (d) $v, parallel$

Figure 3.8: *Comparison of u-velocity along a vertical line at* $x = 0.7$ *and v-velocity along a horizontal line at* $y = 0.7$ *for different Reynolds number, (a) and (b) for antiparallel motion, (c) and (d) for parallel motion on a* 113×113 *grid for two-sided staggered lid-driven cavity.*

For both the antiparallel and parallel configurations, to compare the performance of

Table 3.2: Comparison of multigrid and single-grid on a 113×113 grid for two-sided staggered lid-driven cavity flow problem.

Re	Time Taken (s), *Antiparallel*		Time Taken (s), *Parallel*	
	Multigrid	Single-grid	Multigrid	Single-grid
50	0.392	25.805	0.374	16.632
100	0.412	26.268	0.382	17.237
400	0.712	28.414	0.448	29.238
1000	1.533	58.715	1.720	129.941
3200	−−	−−	40.833	1123.404

multigrid with respect to single-grid solvers, the problem is first solved on a single grid using BiCGStab. The present multigrid code uses GS as the smoother and from Table 3.2 it can be seen that multigrid achieves substantial acceleration over the single-grid solver. The table shows that multigrid converges very fast. For instance, for both antiparallel and parallel motion of the cavity, for Reynolds numbers 50, 100 and 400, the solution is obtained within 1 second. For antiparallel motion, compared to multigrid the single-grid solution for these Reynolds numbers converges in 25.8, 26.2 and 28.4 seconds respectively, and for parallel motion, 16.6, 17.2 and 29.2 seconds respectively. As the Reynolds number increases convergence-time for both multigrid and single-grid increases. For Re = 1000 and the antiparallel motion, multigrid takes only 1.53 seconds, which is about 38 times faster as the single-grid solver takes 58.7 seconds. Similarly, for the parallel motion and Reynolds number 3200, the multigrid solution is about 28 times faster than the solution obtained by the single-grid solver. Thus the current multigrid strategy achieves extremely high convergence acceleration.

3.5 Multiple Solution for Two-Sided Non-Facing Lid-Driven Cavity

The flow configuration for the two-sided non-facing lid-driven cavity is shown in figure 3.9. The dimensionless boundary conditions are given as [33]

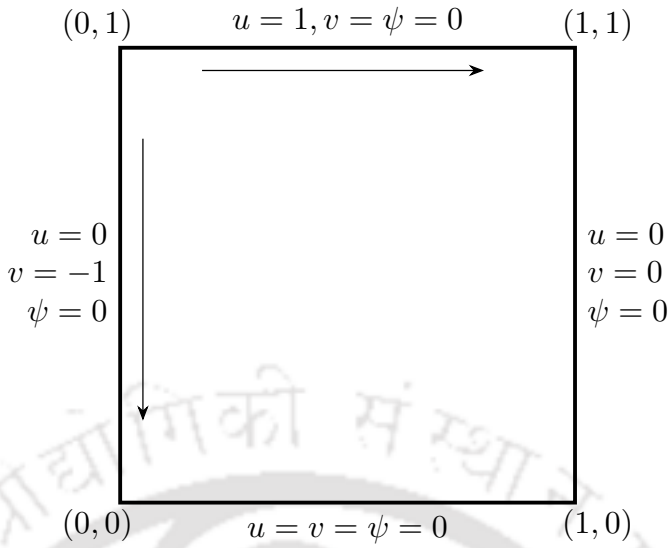


Figure 3.9: *Schematic diagram of the two-sided non-facing lid-driven cavity.*

$$x = 0 : u = 0.0, v = -1.0$$

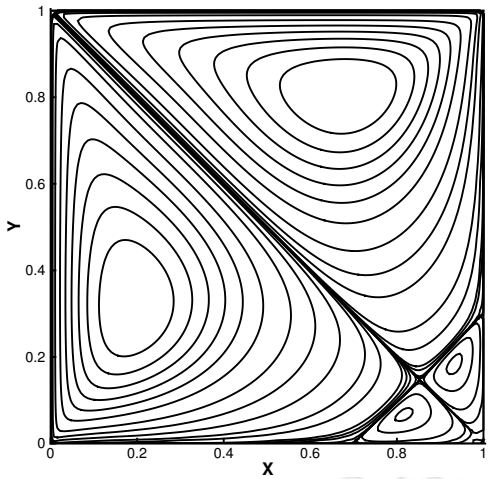
$$x = 1 : u = 0.0, v = 0.0$$

$$y = 0 : u = 0.0, v = 0.0$$

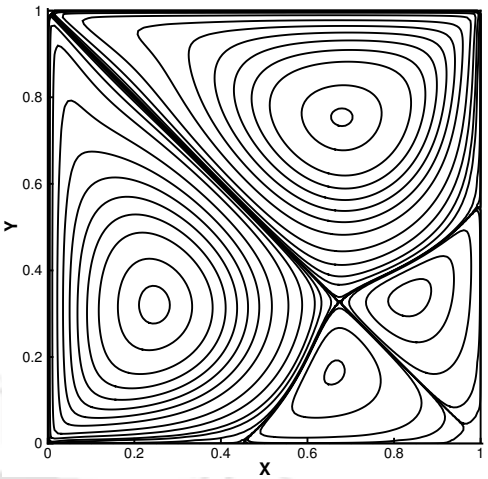
$$y = 1 : u = 1.0, v = 0.0$$

Two moving walls drive the flow. The top wall moves from left to right, and the left wall moves from top to bottom. The top wall drags the fluid to the right, and this fluid then gets deflected by the right wall causing it to circulate in the clockwise direction. Similarly, the left wall drags the fluid to the bottom, and it is deflected by the bottom wall causing it to circulate in the counterclockwise direction. Apart from the two counter-rotating primary vortices, two secondary vortices are formed at Re = 100 with the resulting flow field being symmetric about the diagonal passing through the point where the moving walls meet. As Reynolds number increases to a critical value and beyond, the flow produces multiple symmetric and steady asymmetric solutions.

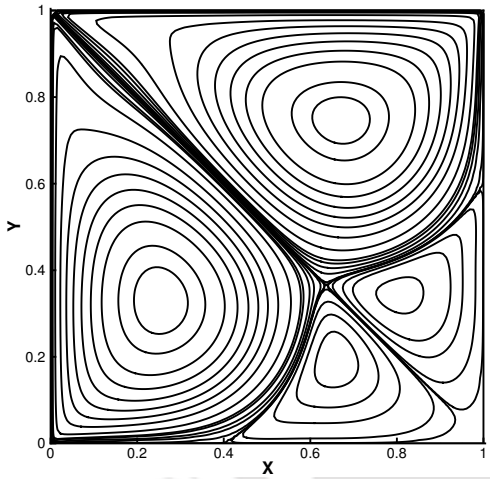
For the $\psi - V$ based computation, the streamfunction is taken as zero at all the walls. The equation (3.7) is solved using linear multigrid with V-cycle that uses five multigrid levels i.e. grids of 9×9, 17×17, 33×33, 65×65 and 129×129. GS has been used as the multigrid smoother. At the coarsest grid, 50 number of iterations is fixed, and at every inner iteration, one multigrid cycle is used. Then u and v are computed at the nodes from equations (3.8) and (3.9) using TDMA. These steps are repeated until the maximum ψ-error between two successive outer iterations was smaller than 10^{-7}.



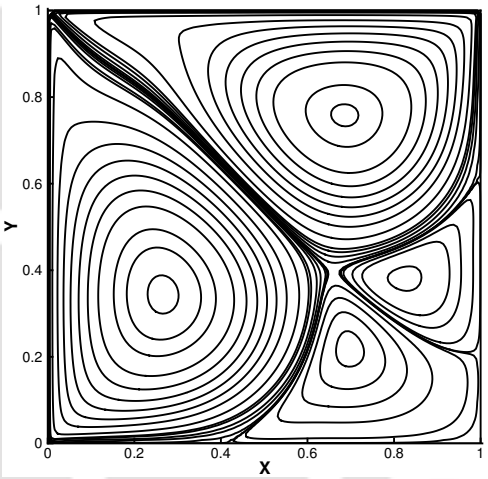
(a) Re = 100



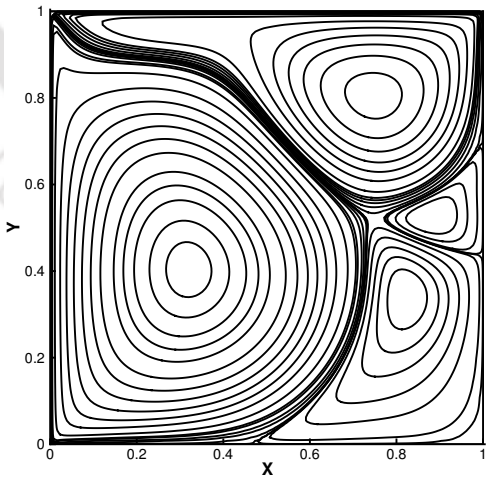
(b) Re = 500



(c) Re = 1000

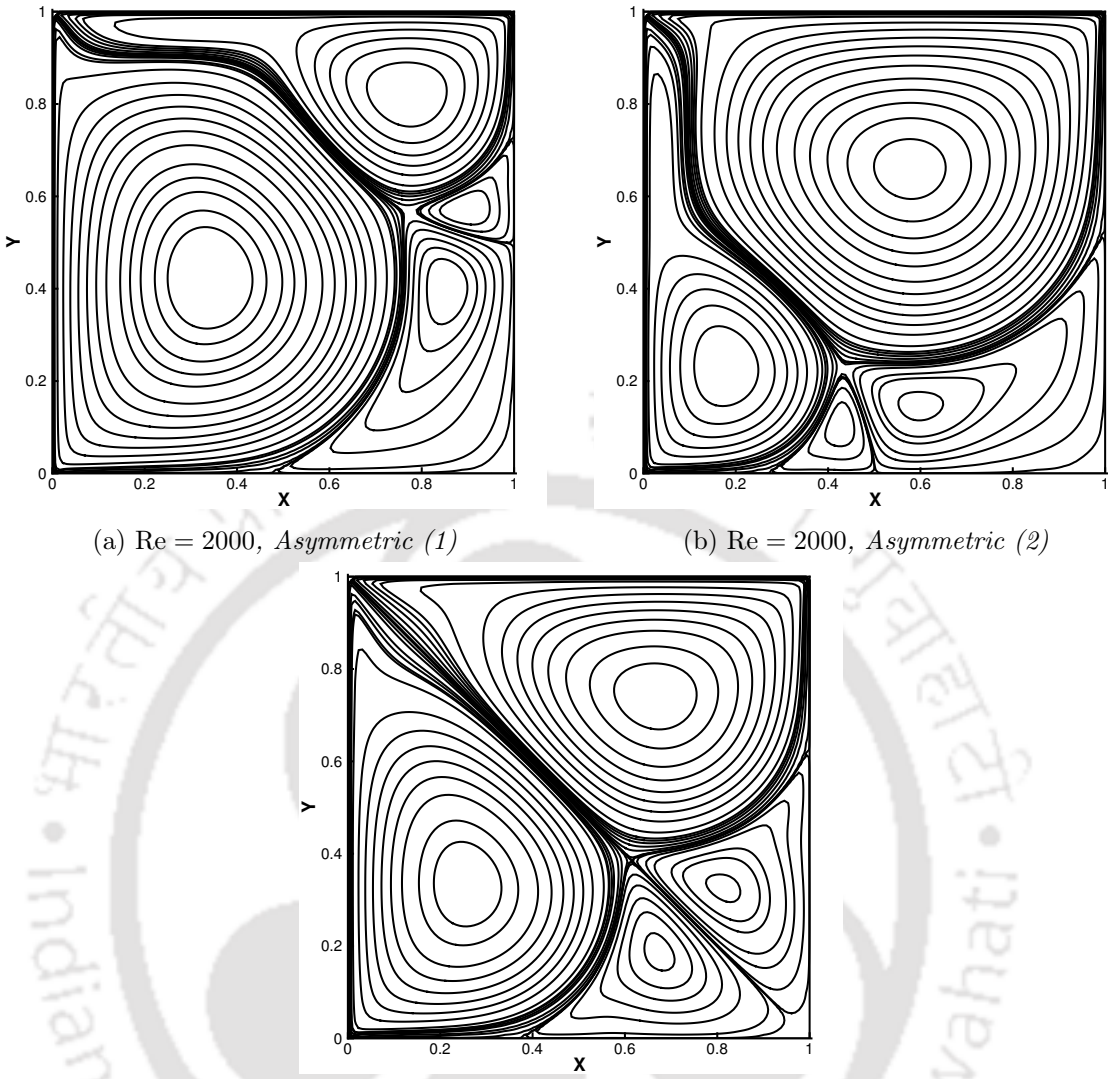


(d) Re = 1075, *Asymmetric*



(e) Re = 1500, *Asymmetric*

Figure 3.10: *Streamlines for two-sided non-facing lid-driven cavity for different Reynolds numbers on a* 129×129 *grid using* $\psi - V$ *formulation and multigrid.*



(a) Re = 2000, *Asymmetric (1)*

(b) Re = 2000, *Asymmetric (2)*

(c) Re = 2000, *Symmetric*

Figure 3.11: *Streamlines for two-sided non-facing lid-driven cavity for Re* = 2000 *on a* 129×129 *grid using* $\psi - V$ *formulation and multigrid.*

Figures 3.10 and 3.11 shows the streamlines for different Reynolds numbers. At lower Reynolds numbers (Figures 3.10a - 3.10c) the streamline patterns are symmetric about a diagonal and the size of the secondary vortices grow in size, as the Reynolds number increases, at the expense of the primary vortices. As seen from Figures 3.10d - 3.11c at larger Reynolds numbers asymmetry may develop even though symmetric solution also exists. Wahba [48] showed that between Reynolds number values of 1071 and 1075, the flow bifurcates from a stable symmetric state to stable asymmetric states and the critical Reynolds number for flow bifurcation is 1073. Figure 3.10d shows that at Re = 1075 there is a departure from symmetry with the upper vortices becoming slightly smaller than the corresponding lower ones if the sweeping direction in the iterative solver is from left to

right and our result is consistent with those reported in [48]. If the same sweeping direction is maintained, this trend continues and the uppper primary vortex becomes smaller and smaller with increase in Reynolds number. Two representative cases (Figures 3.10e- 3.11a) demonstrates this for Re = 1500 and Re = 2000. Figures 3.11a - 3.11c show the multiple-steady flow patterns with one symmetric and two asymmetric solutions at Re = 2000.

Table 3.3: ψ_{LPV} (at left primary vortex centre) and ψ_{RPV} (at right primary vortex centre) values for two-sided non-facing lid-driven cavity flow (Re = 2000) on a 129 × 129 grid.

	Wahba [48] (200 × 200) Grid	Present Study (129 × 129) Grid
Asymmetric (1)	$\psi_{\mathrm{LPV}} = 0.1138$	$\psi_{\mathrm{LPV}} = 0.1153$
	$\psi_{\mathrm{RPV}} = -0.068$	$\psi_{\mathrm{RPV}} = -0.0685$
Asymmetric (2)	$\psi_{\mathrm{LPV}} = 0.068$	$\psi_{\mathrm{LPV}} = 0.0685$
	$\psi_{\mathrm{RPV}} = -0.1138$	$\psi_{\mathrm{RPV}} = -0.1153$
Symmetric	$\psi_{\mathrm{LPV}} = 0.0944$	$\psi_{\mathrm{LPV}} = 0.0954$
	$\psi_{\mathrm{RPV}} = -0.0944$	$\psi_{\mathrm{RPV}} = -0.0954$

Table 3.4: Time taken by the multigrid method for the two-sided lid-driven cavity flow at various Reynolds numbers on a 129 × 129 grid.

Reynolds number	Multigrid (s)	Single-grid (s)
100	0.760116	38
500	1.051080	42
1000	9.295	87
1075 (asymmetric 1)	50.226	–
1500 (asymmetric 1)	8.975	–
2000 (asymmetric 1)	13.175	–

The streamfunction values at vortex centres for Reynolds number 2000 at the left primary vortex (LPV) and the right primary vortex (RPV) are given in Table 3.3 for all the three multiple-steady solutions. The multigrid results obtained at grid 129 × 129 are found to match well with that of the published results of Wahba [48] at grid 200 × 200. It can be seen that using the $\psi - V$ form of the N-S equation yields very accurate results even at a relatively coarser grid. One reason for this is definitely the absence of vorticity boundary conditions that injects some error into the solution procedure that uses $\psi - \omega$ formulation. Table 3.4 shows the time taken by the multigrid method for different Reynolds numbers. From the table, it can be seen that for Reynolds numbers 100 and 500, the solution converges very fast taking about a second. Compared to multigrid, the single-grid solution with BiCGStab requires approximately 38 and 42 seconds for Reynolds numbers

100 and 500 respectively. It shows that the multigrid accelerated $\psi - V$ formulation is not only accurate but also quite fast. At Reynolds numbers 1000 and 1500, the time required for convergence is about 9 seconds, which appears to be quite fast. At Reynolds number 1075, we can observe that since it is near the critical Reynolds number 1073 (Wahba [48]), more time is needed for the solution to convergence.

3.6 Multiple Solution for Four-sided Lid-Driven Cavity

Figure 3.12 shows the flow configuration for the four-sided lid-driven cavity. The dimensionless boundary conditions are given as [33]

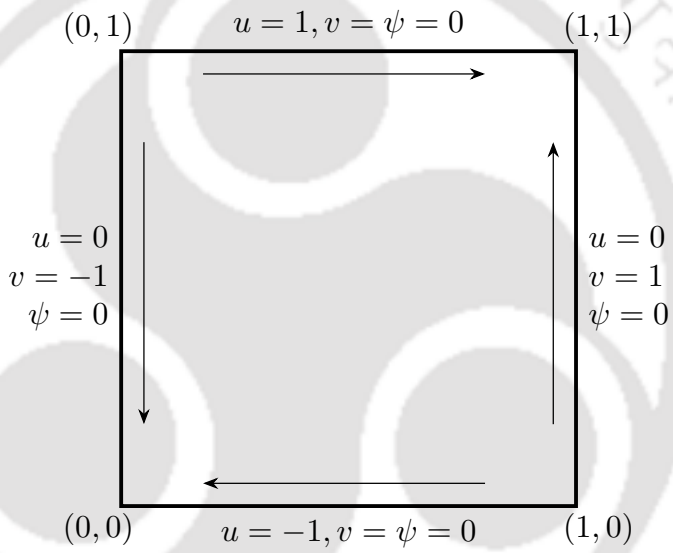


Figure 3.12: *Schematic diagram of the four-sided lid-driven cavity.*

$$x = 0 : u = 0.0, v = -1.0$$

$$x = 1 : u = 0.0, v = 1.0$$

$$y = 0 : u = -1.0, v = 0.0$$

$$y = 1 : u = 1.0, v = 0.0$$

Here the speed of all the four walls are equal. The upper wall moves to the right and the lower wall moves towards left, while the left and right walls move downwards and upwards respectively. The fluid dragged towards right by the top wall is deflected by the right wall causing it to circulate in the clockwise direction. The right wall drags the fluid upwards and the fluid is then deflected by the top wall causing it to circulate in the

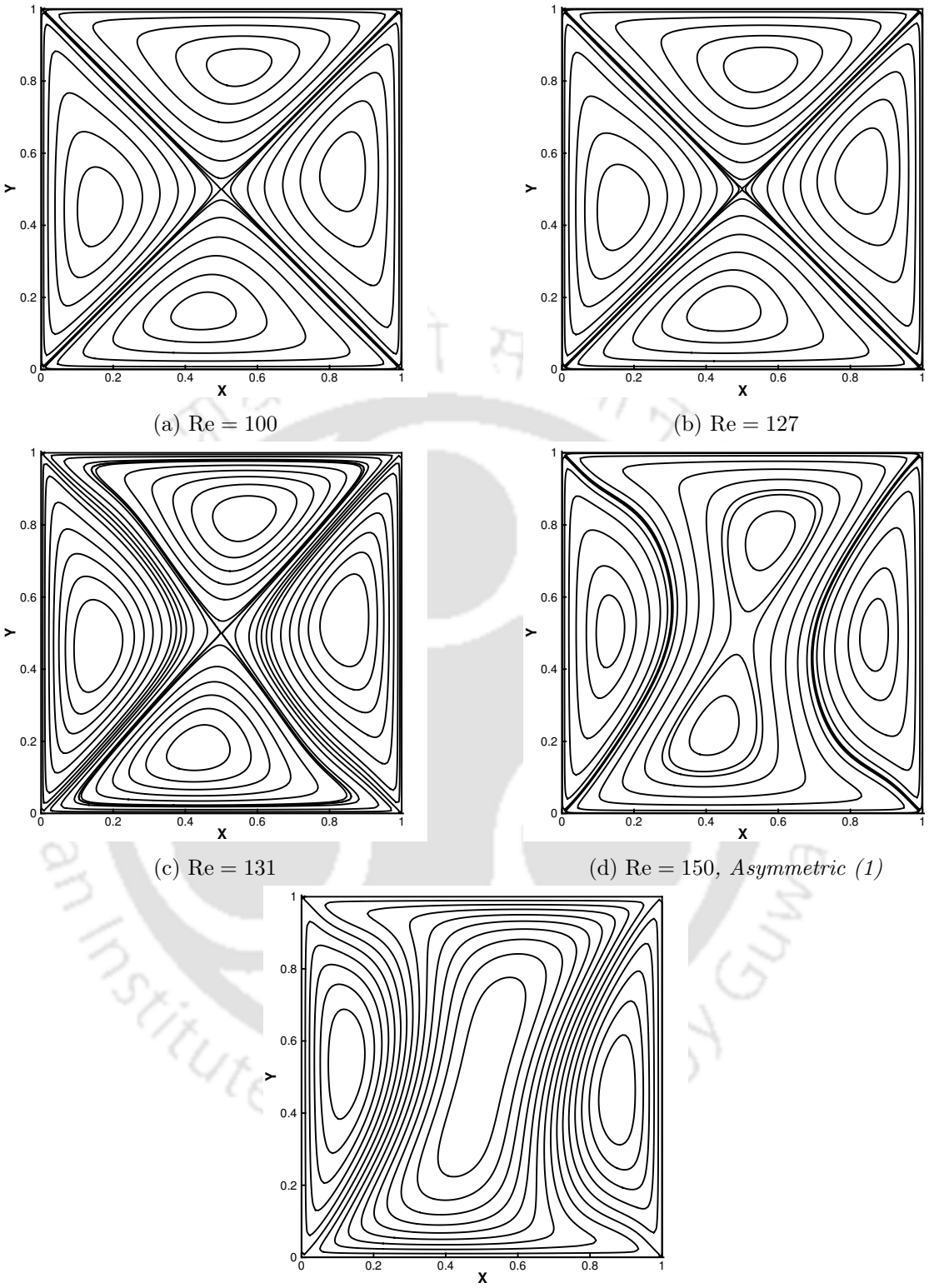
counterclockwise direction. Similarly, the fluid dragged by the bottom wall towards left and the left wall towards top gets rotated in the clockwise and counterclockwise direction respectively. As the speed of all the four walls are equal, four distinct similar sized vortices develop symmetric to both the diagonals.

For the $\psi - V$ based computation, the streamfunction is taken as zero at all the walls. The equation (3.7) is solved using multigrid V-cycle that uses five multigrid levels i.e. grids of 9×9, 17×17, 33×33, 65×65 and 129×129. GS has been used as the multigrid smoother. At the coarsest grid, 30 number of iterations is fixed, and at every inner iteration, one multigrid cycle is used. Then u and v are computed at the nodes from equations (3.8) and (3.9) using TDMA. These steps are repeated until the maximum ψ-error between two successive outer iterations is smaller than 10^{-8}.

Figures 3.13 and 3.14 shows the streamlines for different Reynolds numbers. The streamlines for Re = 100 and Re = 127 (Figure 3.13a - 3.13b) are symmetric about both the diagonals. As seen from Figures 3.13c - 3.14c at larger Reynolds number asymmetry may develop. Wahba [48] showed that for a four-sided lid-driven cavity, the flow bifurcates from a stable symmetric state to a stable asymmetric state between Reynolds number 127 and 131, and the value of the critical Reynolds number for bifurcation is 129. A departure from symmetry is observed at Re = 131 from Figure 3.13c, where the upper and lower vortices show a tendency to merge into one, and our result is consistent with those reported in [48]. Figures 3.13d - 3.14c show that with the increase in Reynolds number the merging process continues. Figure 3.13e shows a asymmetric state when the merging results in a single vortex with a single-core which is considerably larger than the right and left vortices. If the sweep direction is reversed then the resulting asymmetric flow patterns would have merged right and left vortices and the resultant vortex would be larger than the upper and lower vortices. All the three multiple-steady flow patterns for Re = 300 can be seen from Figures 3.14a - 3.14c with one symmetric and two asymmetric solutions.

Table 3.5: ψ_{GC} (Geometric Center of Gravity) values for four-sided lid-driven cavity flow (Re = 300) on a 129×129 grid.

	Wahba [48] (200×200) Grid	Present Multigrid Study (129×129) Grid
Asymmetric (1)	$\psi_{GC} = -0.1127$	$\psi_{GC} = -0.11328$
Symmetric	$\psi_{GC} = 0.0$	$\psi_{GC} = 0.0$
Asymmetric (2)	$\psi_{GC} = 0.1127$	$\psi_{GC} = 0.11328$



(a) Re = 100

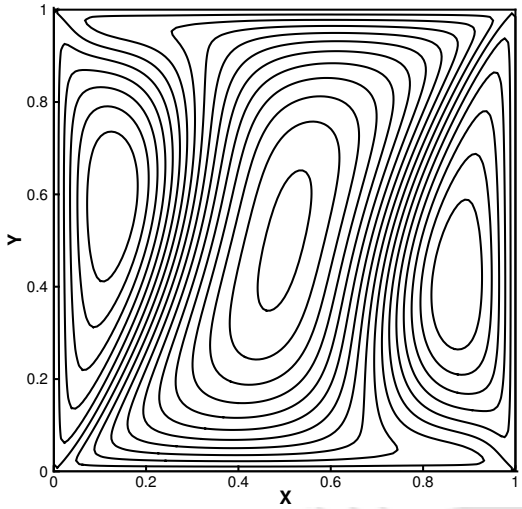
(b) Re = 127

(c) Re = 131

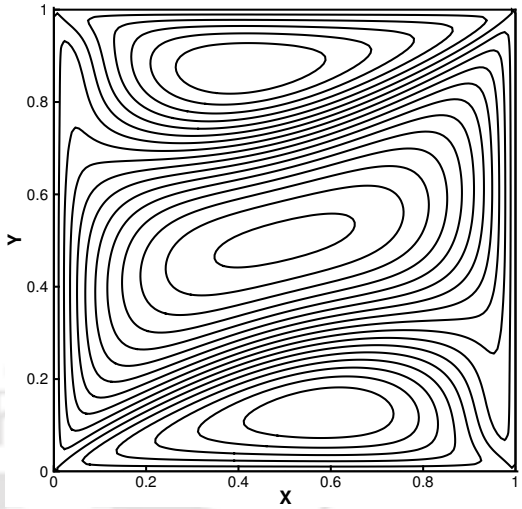
(d) Re = 150, *Asymmetric (1)*

(e) Re = 200, *Asymmetric (1)*

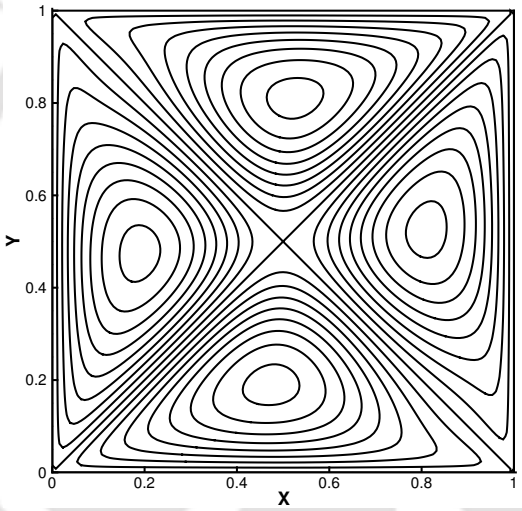
Figure 3.13: *Streamlines for a four-sided lid-driven cavity for different Reynolds numbers on a* 129×129 *grid using* $\psi - V$ *formulation and multigrid.*



(a) Re = 300, *Asymmetric (1)*



(b) Re = 300, *Asymmetric (2)*



(c) Re = 300, *Symmetric*

Figure 3.14: *Streamlines for a four-sided lid-driven cavity for Re* = 300 *on a* 129 × 129 *grid using* $\psi - V$ *formulation and multigrid.*

Table 3.6: Time taken by the multigrid method for the four-sided lid-driven cavity flow on a 129 × 129 grid at various Reynolds numbers.

Reynolds number	CPU Time (s)
100	1.852
127	7.448
131	96.129
200 (asymmetric 1)	2.912
300 (asymmetric 1)	2.447

The streamfunction values for Reynolds number 300 at the geometric centre of the cavity is given in Table 3.5 for all the three multiple-steady solutions. The multigrid results obtained at grid 129 × 129 are found to match well with that of the published

results of Wahba [48] at grid 200×200. Again, it can be seen that using the $\psi - V$ form of the N-S equation yields very accurate results even at a relatively coarser grid. Table 3.6 shows the time taken by the multigrid method for different Reynolds numbers. From the table, very fast convergence can be observed for Reynolds numbers 100, 200 and 300 (about two to three seconds). At Reynolds numbers 127 and 131, the times required for convergence are comparatively large and are about 7.5 and 96 seconds, respectively as both the Reynolds numbers lie near the critical value of 129.

3.7 Summary

The combination of the second-order accurate compact discretization of the biharmonic form of the N-S equations and the multigrid method yields an immensely accelerated solution procedure. The main reason for this appears to be the iteration-independent fixed Dirichlet boundary condition for ψ, in which information is spread all over the domain very quickly by multigrid. The result is that multigrid achieves tremendous speed-up over the single-grid methods. It appears that the performance of multigrid in combination with the $\psi - V$ formulation for the N-S equations has not received adequate attention so far and therefore this work aims at carrying out a systematic study of this multigrid-accelerated solution procedure. Accordingly, to elaborate on various aspects of the solution procedure, a number of fluid-flow problems have been solved in the laminar regime. In the standard lid-driven square-cavity problem the converged solution for $\mathrm{Re} = 7500$ is obtained by multigrid in about 263 seconds in an intel i5-based PC with a speed-up of 69 over the single-grid. For the staggered-cavity flow problem, the present solution procedure has been used for the first time. For this problem, laminar investigations are seen to be carried out at the highest value of $\mathrm{Re} = 3200$ and for this Re the present solution procedure obtains results in only 41 seconds with a speed-up of about 28. The accuracy and robustness of the current solution procedure can be evidenced by the fact that it can be comfortably used to obtain multiple stable solutions for two- and four-sided driven cavity flow. It may be noted that nonlinear problems are known at times to give multiple solutions and the traditional mathematical concept of wellposedness does not apply here. To examine the applicability of the present method to such problems the two-sided and four-sided driven cavity flow problems are studied, which are known to give multiple stable solutions. Near the critical Re at which the flow bifurcates, the convergence is invariably slow and the present method achieves fast convergence even under these situations. Thus, we see that the multigrid-assisted $\psi - V$

formulation performs very well in a variety of flow situations that include pure fluid flow and fluid-flow problems giving multiple stable solutions. In all the situations solutions are obtained in a few seconds on a grid that provides accurate solutions, giving substantial convergence acceleration over a single-grid. We believe that iteration-independent and physically correct Dirichlet boundary conditions for both streamfunction and velocity allow multigrid to propagate boundary information to the interior very quickly so that speed-up over single-grid computation is substantial indeed. As the Dirichlet boundary conditions are physically correct and the discretization is at least second-order accurate, the solutions are very accurate as shown by the comparisons. For certain situations, for example, the staggered cavity flow and the multiple-stable-solution problems, this method was not used earlier and the results had to be compared with results obtained by other methods. Even for these problems, our computations are likely to be very efficient and accurate. These observations indicate that the present multigrid-assisted streamfunction-velocity method is a very fast and accurate solution procedure for a variety of incompressible viscous flow problems.

Chapter 4

Multigrid Assisted $\psi - V$ Formulation for Heat Transfer Problems

4.1 Introduction

The natural and mixed convection flow phenomena in a differentially heated square cavity have been extensively studied by researchers for testing and validating new algorithms and codes owing to its wide range of industrial and engineering applications such as heat exchangers, building ventilation systems, cooling of electronic components, nuclear reactor insulation and collector of solar energy [108, 109]. Natural convection in enclosures has been extensively studied numerically by de Vahl Davis [110] for laminar flows, Henkes and Hoodendoorn [111] and Marakatos and Pericleous [112] for transition and turbulent flows. Peng et al. [18], D'Orazio et al. [113], Dixit and Babu [114], Kashyap and Dass [115] used LBM to study the natural convection phenomena in a differentially heated square cavity. In order to understand the complex interaction of lid-driven flow and buoyancy-driven flow dealt by mixed convection flow in enclosures, extensive experimental [116] and numerical studies [5, 117, 118] have been carried out. Prasad and Koseff [116] experimentally demonstrated the influence of Richardson numbers on the mixed convection flow. Iwatsu et al. [5] used the finite difference method to simulate mixed convection in a square lid-driven cavity. The vertical stationary walls were kept in adiabatic condition, and the top lid of the cavity moved from left to right. Kashyap and Dass [118] used a two-phase lattice Boltzmann model to study mixed-convection flow in a cavity filled with hybrid nanofluid. This chapter is concerned with the study of a computationally efficient solution procedure for natural and mixed convection problems. The method involves streamfunction-velocity

$(\psi - V)$ formulation for the Navier-Stokes (N-S) equations, as it has been observed to produce highly accurate solutions. We use the multigrid method to accelerate the convergence of these problems in section 4.3 and 4.4, which results in a highly efficient and accurate solution procedure.

4.2 Computational Methodology

The streamfunction-velocity formulation is explained in details in the section 3.2.1.

4.2.1 Governing Equations of Natural Convection

The steady 2D primitive variable form of the mass, momentum and energy equations, using Boussinesq approximation are:

$$u_x + v_y = 0, \tag{4.1}$$

$$uu_x + vu_y = -\frac{1}{\rho} p_x + \nu \nabla^2 u, \tag{4.2}$$

$$uv_x + vv_y = -\frac{1}{\rho} p_y + \nu \nabla^2 v + g\beta\,(T - T_0), \tag{4.3}$$

$$uT_x + vT_y = \alpha \nabla^2 T \tag{4.4}$$

where T is the temperature, ρ is the density, ν is the dynamic viscosity, β is the coefficient of thermal expansion, and α is the thermal diffusivity of the fluid.

In case of the natural convection problem, to obtain the non-dimensional form of the equations (4.1 – 4.4) the following non-dimensional parameters are introduced:

$$x^* = \frac{x}{L}, \quad y^* = \frac{x}{L}, \quad u^* = \frac{uL}{\alpha}, \quad v^* = \frac{vL}{\alpha}, \quad p^* = \frac{pL^2}{\rho\alpha^2}, T^* = \frac{T - T_0}{T_h - T_c} \tag{4.5}$$

where, L is the characteristic length, T_h and T_c are the temperatures at the hot and cold wall respectively. After dropping the asterisks the non-dimensional equations become:

$$u_x + v_y = 0, \tag{4.6}$$

$$uu_x + vu_y = -p_x + \mathrm{Pr}\nabla^2 u, \tag{4.7}$$

$$uv_x + vv_y = -p_y + \mathrm{Pr}\nabla^2 v + \mathrm{Ra}\,\mathrm{Pr}\,T, \tag{4.8}$$

$$uT_x + vT_y = \nabla^2 T \tag{4.9}$$

where Pr is the Prandlt number and Ra is the Rayleigh number.

Eliminating the pressure terms from the equations (4.7) and (4.8), the equations (4.6 – 4.9) can be written in streamfunction-vorticity form as

$$u\omega_x + v\omega_y = \mathrm{Pr}\,\nabla^2\omega + \mathrm{Ra}\,\mathrm{Pr}\,T_x \tag{4.10}$$

$$\psi_{xx} + \psi_{yy} = -\omega\,(x, y)\,, \tag{4.11}$$

$$uT_x + vT_y = \nabla^2 T \tag{4.12}$$

Now, substituting equation (4.11) in (4.10), the non-dimensional momentum equation in its biharmonic form can be obtain as

$$\frac{\partial^4\psi}{\partial x^4} + 2\frac{\partial^4\psi}{\partial x^2\partial y^2} + \frac{\partial^4\psi}{\partial y^4} - \frac{1}{\mathrm{Pr}}\,u\left(\frac{\partial^3\psi}{\partial x^3} + \frac{\partial^3\psi}{\partial x\partial y^2}\right) - \frac{1}{\mathrm{Pr}}\,v\left(\frac{\partial^3\psi}{\partial x^2\partial y} + \frac{\partial^3\psi}{\partial y^3}\right) = \mathrm{Ra}\,\frac{\partial T}{\partial x}. \tag{4.13}$$

4.2.2 Governing Equations of Mixed Convection

For the mixed convection problem, the following non-dimensional parameters are introduced to obtain the non-dimensional form of the governing equations (4.1 – 4.4):

$$x^* = \frac{x}{L}, \quad y^* = \frac{x}{L}, \quad u^* = \frac{u}{U}, \quad v^* = \frac{v}{U}, \quad p^* = \frac{p}{\rho U^2}, T^* = \frac{T - T_0}{T_h - T_c} \tag{4.14}$$

where U is the characteristic velocity. The non-dimensional form of the governing equations after dropping the asterisks are:

$$u_x + v_y = 0, \tag{4.15}$$

$$uu_x + vu_y = -p_x + \frac{1}{\mathrm{Re}}\nabla^2 u, \tag{4.16}$$

$$uv_x + vv_y = -p_y + \frac{1}{\mathrm{Re}}\nabla^2 v + \frac{\mathrm{Gr}}{\mathrm{Re}^2}\,T, \tag{4.17}$$

$$uT_x + vT_y = \frac{1}{\mathrm{Pr}\,\mathrm{Re}}\nabla^2 T \tag{4.18}$$

where Gr is the Grashof number. The non-dimensional momentum equation in its biharmonic form is

$$\frac{\partial^4\psi}{\partial x^4} + 2\frac{\partial^4\psi}{\partial x^2\partial y^2} + \frac{\partial^4\psi}{\partial y^4} - \mathrm{Re}\,u\left(\frac{\partial^3\psi}{\partial x^3} + \frac{\partial^3\psi}{\partial x\partial y^2}\right) - \mathrm{Re}\,v\left(\frac{\partial^3\psi}{\partial x^2\partial y} + \frac{\partial^3\psi}{\partial y^3}\right) = \frac{\mathrm{Gr}}{\mathrm{Re}}\frac{\partial T}{\partial x}. \tag{4.19}$$

4.2.3 Solution Procedure

The present work has adopted a multigrid V-cycle with full-weighting as restriction operation, Gauss-Seidel as smoother to solve the equation (3.7) and bilinear interpolation as prolongation operation. Moreover, at the coarsest grid, rather than solving up to machine precision, a certain number of sweeps are fixed. The equations (3.8) and (3.9) are solved using the TDMA. Hence, the equations (3.7), (3.8) and (3.9) are solved in an outer-inner fashion. All the computations are carried out on an Intel i5-4690K CPU having 16GB RAM and with Intel Compiler 19.1 (package 166). All the time mentioned in the tables in the following sections are CPU time only. In order to demonstrate the ease of use and benefits in terms of speed and accuracy of the multigrid assisted $\psi - V$ formulation, we have chosen two different test cases in the following sections.

4.3 Two-Dimensional Natural Convection in a Square Cavity

The schematic of the 2D computational domain of the natural convection in a square domain is shown in Figure 4.1. Here, the left wall is at a higher temperature, and the right wall is at a lower temperature. The top and bottom walls are insulated, and at the boundaries, both the velocity components are zero. The boundary conditions are as follows

$$\begin{aligned}
&\psi = u = v = 0, \qquad T = 1 \ \text{ at } \ x = 0 \ \ (0 \le y \le 1) \\
&\psi = u = v = 0, \qquad T = 0 \ \text{ at } \ x = 1 \ \ (0 \le y \le 1) \\
&\psi = u = v = 0, \qquad \frac{\partial T}{\partial y} = 0 \ \text{ at } \ y = 0 \ \ (0 \le x \le 1) \\
&\psi = u = v = 0, \qquad \frac{\partial T}{\partial y} = 0 \ \text{ at } \ y = 1 \ \ (0 \le x \le 1)
\end{aligned} \tag{4.20}$$

On the boundary, because $u = 0$ and $v = 0$, the temperature (equation (4.9)) can be reduced as

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0. \tag{4.21}$$

The solutions are computed in a grid of size 129×129 for Prandtl number 0.71 and Rayleigh numbers $10^3, 10^4, 10^5, 10^6$ and 10^7. A five-level multigrid V-cycle with Gauss-Seidel smoother is used to solve the streamfunction (equation (4.13)). At every inner iteration, one multigrid cycle (with one presmooth, one postsmooth, and fifty coarsesmooth iterations) is used. Then u and v are computed at the nodes from equations (3.8) and (3.9) using TDMA. The temperature (equation (4.21)) is then solved using Gauss-Seidel.

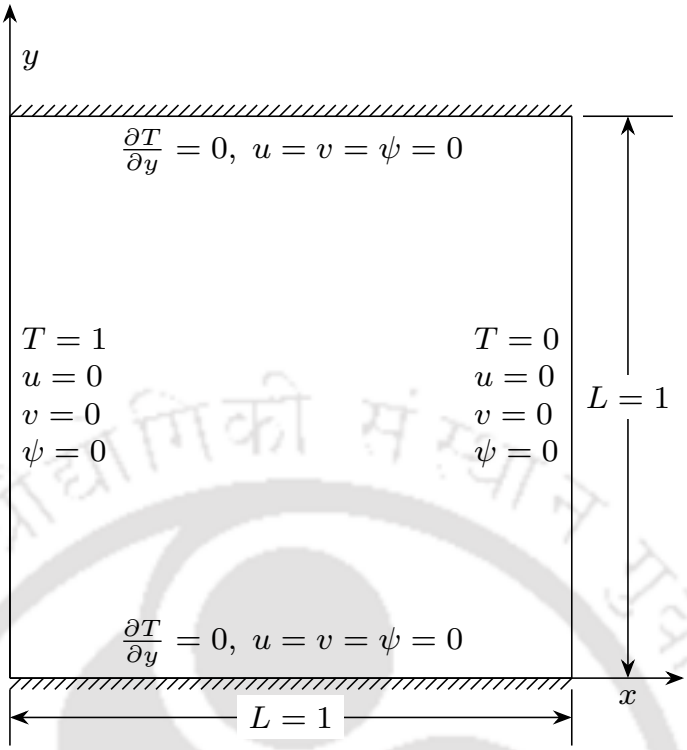


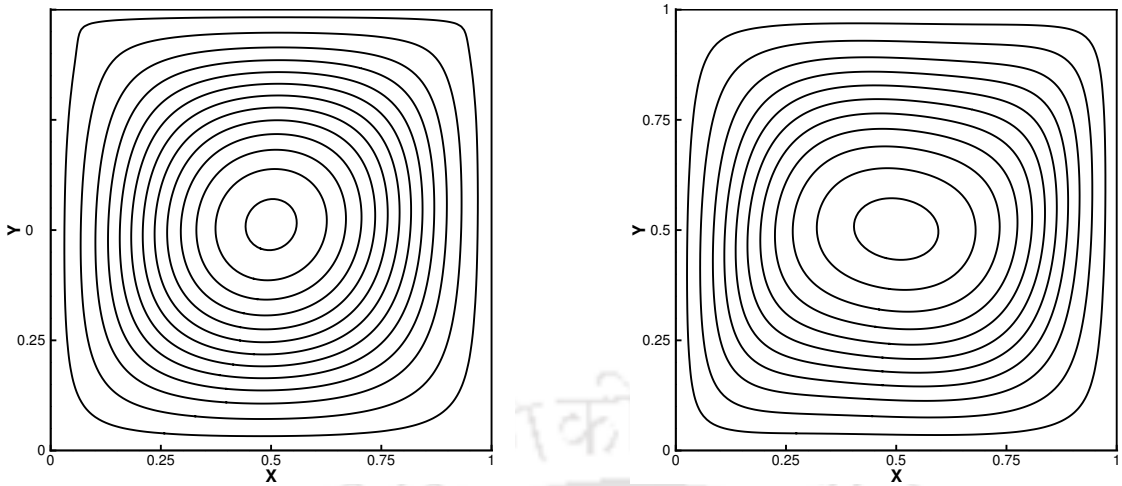
Figure 4.1: *Schematic of the natural convection problem in a square domain with boundary conditions.*

It completes one outer iteration. These steps are repeated until the maximum ψ-error between two successive outer iterations was smaller than 10^{-6}.

Table 4.1: Comparison of time taken by multigrid and single-grid on a 129×129 Grid and the average Nusselt number (Nu) at the left wall for natural convection problem.

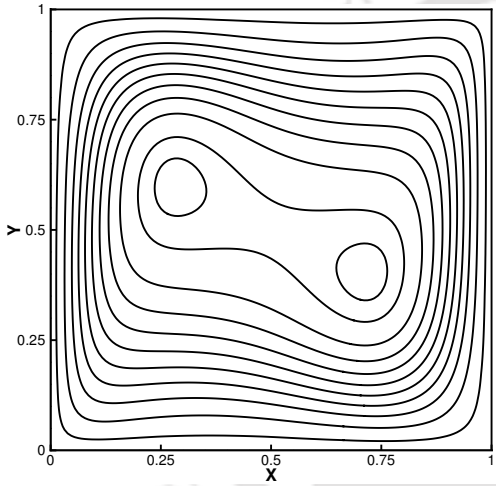
Ra	Time Taken (s)		Average Nusselt number at the left wall		
	Multigrid	Single-grid	Present	de Vahl Davis [110]	Dixit & Babu [114]
10^3	4.371	56.946	1.132	1.116	1.127
10^4	3.664	44.939	2.256	2.242	2.247
10^5	2.945	29.700	4.532	4.523	4.523
10^6	2.075	17.126	8.890	8.928	8.805
10^7	10.214	81.496	17.185	–	16.79

The streamline plots and the temperature plots for different Rayleigh numbers are presented in Figures 4.2 and 4.3 respectively. Table 4.1 shows the performance of the multigrid and its speed-up over the single-grid for different Rayleigh numbers. The multigrid solution is quite fast; for instance, using multigrid the converged solution for the Rayleigh number 10^7 can be obtained in about 10 seconds. It can be observed from the table that the multigrid is about eight to ten times faster than the single-grid solution. The average Nusselt

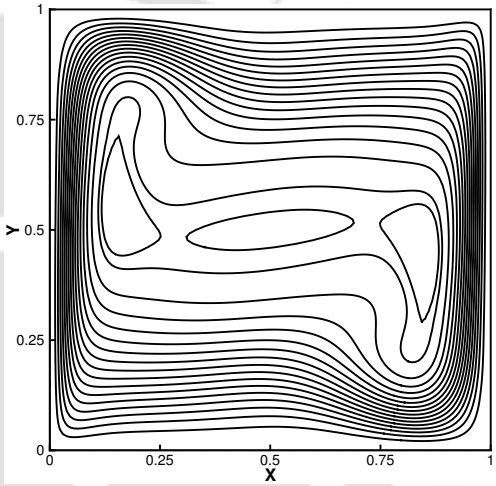


(a) Ra = 10^3

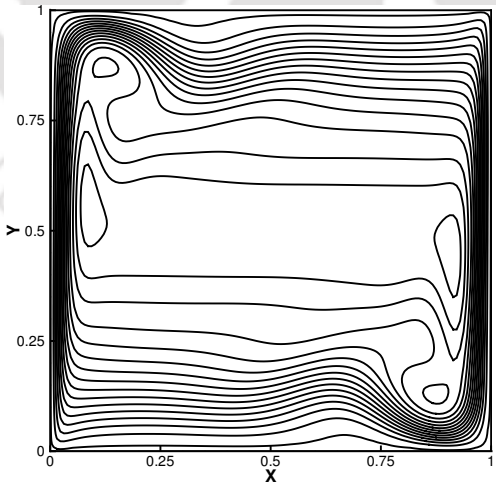
(b) Ra = 10^4



(c) Ra = 10^5

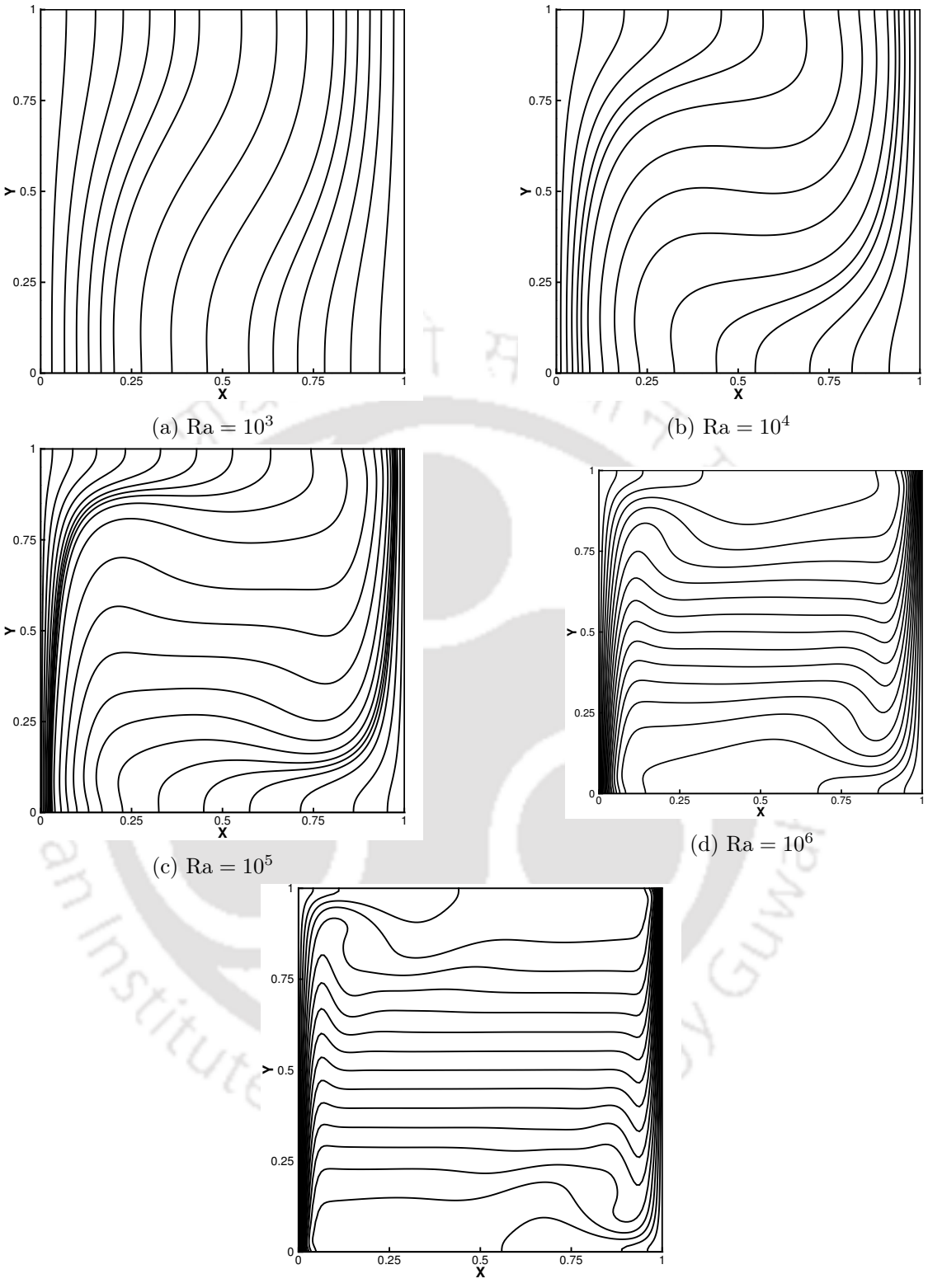


(d) Ra = 10^6



(e) Ra = 10^7

Figure 4.2: *Streamlines for different Rayleigh number on a* 129×129 *grid for natural convection problem.*



(a) Ra = 10^3

(b) Ra = 10^4

(c) Ra = 10^5

(d) Ra = 10^6

(e) Ra = 10^7

Figure 4.3: *Temperature contours for different Rayleigh number on a* 129×129 *grid for natural convection problem.*

number Nu on the left boundary is obtained by numerical integration using Simpson's $\frac{1}{3}$ rule and is shown in Table 4.1. It can be observed from the table that the present Nu values are in good agreement with the published results of de Vahl Davis [110] and Dixit and Babu [114].

4.4 Two-Dimensional Mixed Convection in a Square Cavity

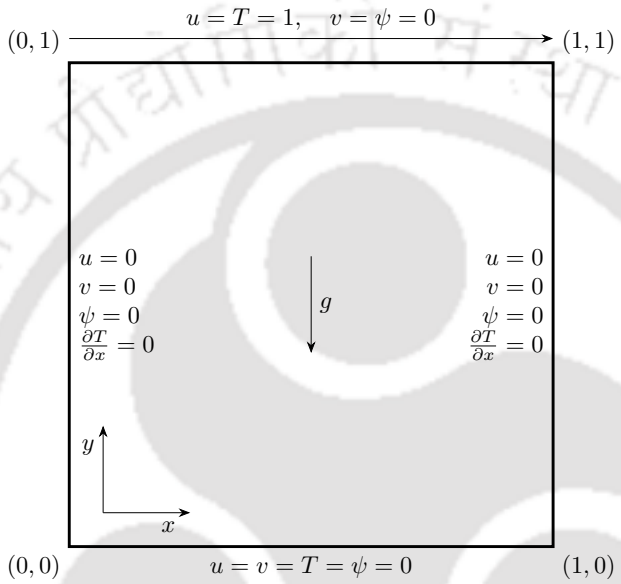


Figure 4.4: *Computational domain of mixed convection problem.*

The computational domain of the mixed convection problem is a unit square and is as shown in Figure 4.4. Here, the top wall is maintained at a higher temperature and the bottom wall is at a lower temperature. The left and right walls are insulated. The top wall is moving from the left to right with a constant unit velocity and the other three walls are stationary. The boundary conditions are as follows

$$\begin{aligned}
&\psi = u = v = 0, && T = 0 \ \text{ at } \ y = 0 \ \ (0 \le x \le 1) \\
&\psi = v = 0, && u = T = 1 \ \text{ at } \ y = 1 \ \ (0 \le x \le 1) \\
&\psi = u = v = 0, && \frac{\partial T}{\partial x} = 0 \ \text{ at } \ x = 0 \ \ (0 \le y \le 1) \\
&\psi = u = v = 0, && \frac{\partial T}{\partial x} = 0 \ \text{ at } \ x = 1 \ \ (0 \le y \le 1)
\end{aligned} \tag{4.22}$$

Conduction heat transfer could have been prevalent if the top wall remains stationary. However, the movement of the top wall induces a fluid flow inside the cavity. The fluid flow and the heat transfer of the mixed convection problem are characterized by Re, Gr

and Pr. The solutions are computed in a grid of size 129×129 for Prandtl number 0.71. Three different Reynolds numbers $100, 400$ and 1000 and three different Grashof numbers $10^2, 10^4$ and 10^6 have been chosen. Multigrid with Gauss-Seidel smoother is used to solve the streamfunction, equation (4.19). At every inner iteration, one multigrid cycle (with one presmooth, one postsmooth, and five coarsesmooth iterations) is carried out. Then u and v are computed at the nodes from equations (3.8) and (3.9) using one GS iterations. The temperature, equation (4.18), is then solved using TDMA. It completes one outer iteration. These steps are repeated until the maximum ψ-error between two successive outer iterations is smaller than 10^{-6}.

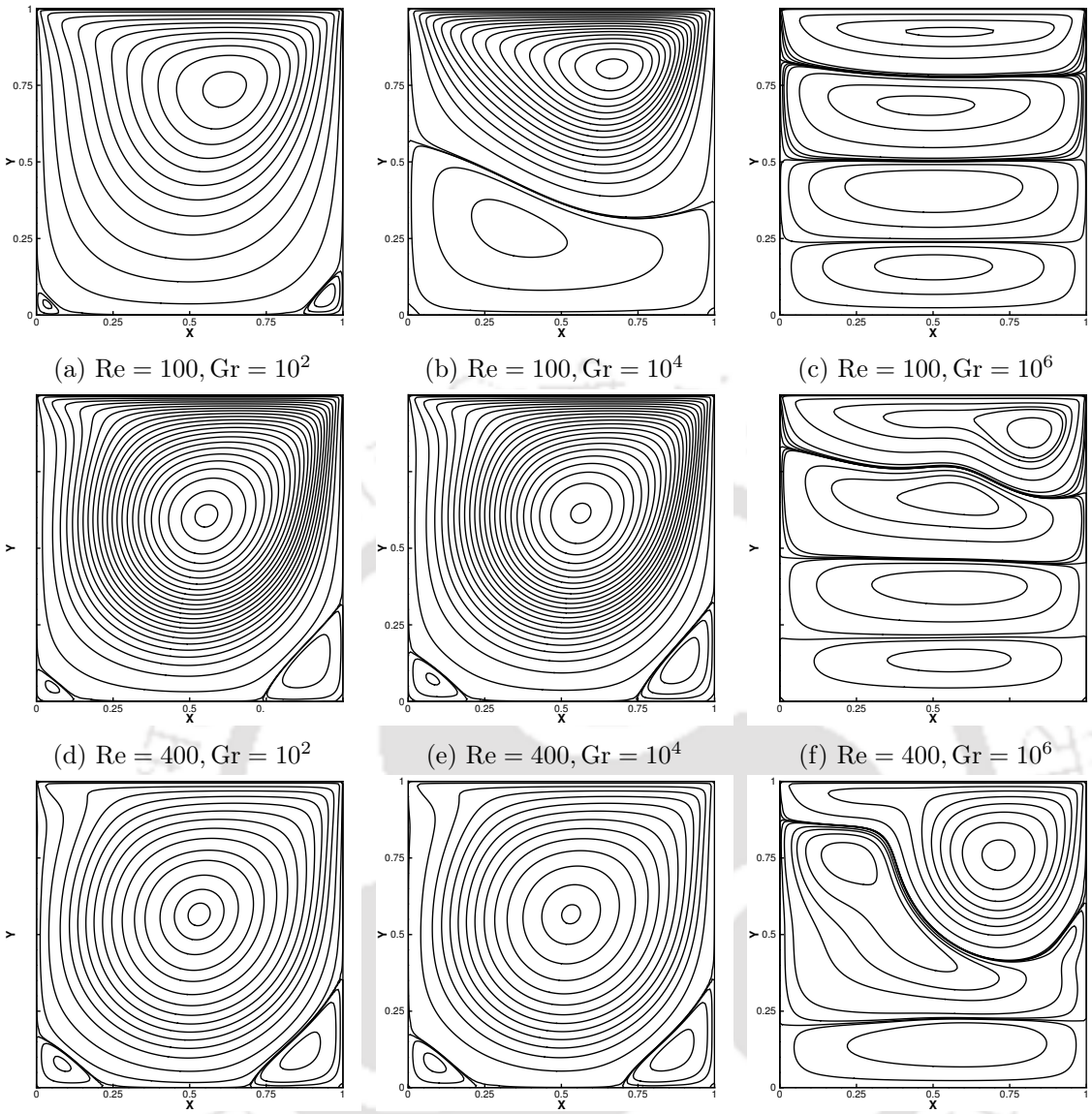
Table 4.2: Comparison of Multigrid and single-grid time on a 129×129 Grid for mixed convection problem.

Re	Gr	Time Taken (s)	
		Multigrid	Single-grid
100	10^2	2.643	85.701
	10^4	2.703	47.822
	10^6	2.692	29.187
400	10^2	2.531	90.520
	10^4	2.672	84.934
	10^6	3.109	43.189
1000	10^2	2.849	315.75
	10^4	2.913	286.870
	10^6	5.160	211.851

Table 4.3: Average Nusselt number at the top wall on a 129×129 grid for mixed convection problem.

Re		$\mathrm{Gr} = 10^2$	$\mathrm{Gr} = 10^4$	$\mathrm{Gr} = 10^6$
100	Present study	2.051679	1.410021	1.027810
	Iwatsu et al. [5]	1.94	1.34	1.021
400	Present study	4.011045	3.772329	1.222706
	Iwatsu et al. [5]	3.84	3.62	1.22
1000	Present study	6.430063	6.375177	1.807966
	Iwatsu et al. [5]	6.33	6.29	1.771

The streamline plots and the temperature plots for different Reynolds numbers at different Grashof numbers are presented in Figures 4.5 and 4.6 respectively. Table 4.3 shows the average Nusselt number at the top wall. The Nusselt number values are in good agreement with the published results of Iwatsu et al. [5]. The time required to get the converged



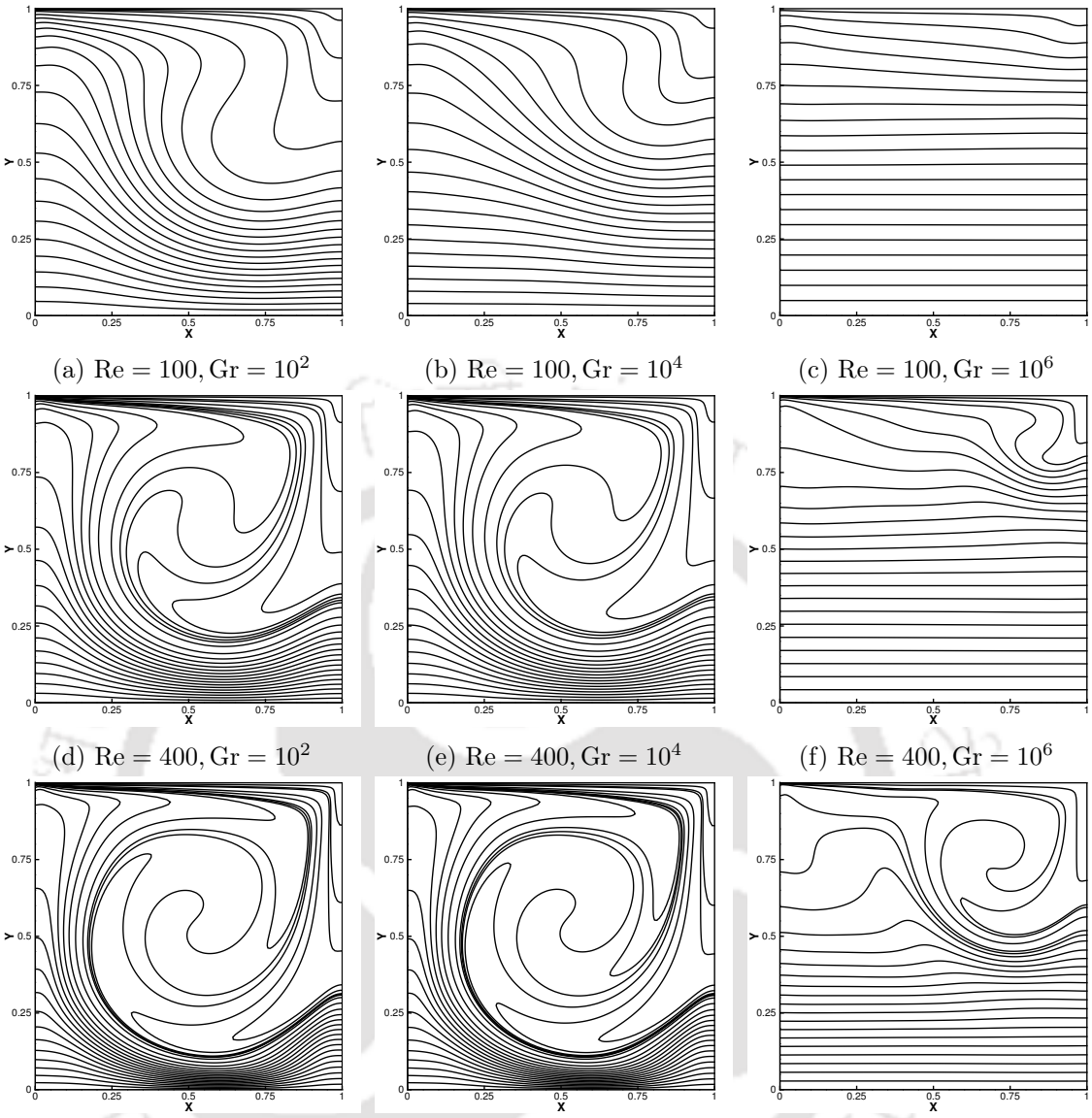
(a) $\mathrm{Re} = 100, \mathrm{Gr} = 10^2$ (b) $\mathrm{Re} = 100, \mathrm{Gr} = 10^4$ (c) $\mathrm{Re} = 100, \mathrm{Gr} = 10^6$

(d) $\mathrm{Re} = 400, \mathrm{Gr} = 10^2$ (e) $\mathrm{Re} = 400, \mathrm{Gr} = 10^4$ (f) $\mathrm{Re} = 400, \mathrm{Gr} = 10^6$

(g) $\mathrm{Re} = 1000, \mathrm{Gr} = 10^2$ (h) $\mathrm{Re} = 1000, \mathrm{Gr} = 10^4$ (i) $\mathrm{Re} = 1000, \mathrm{Gr} = 10^6$

Figure 4.5: *Streamlines for different Reynolds and Grashof number on a* 129×129 *grid for mixed convection problem.*

solution by the multigrid method, and single-grid is compared in Table 4.2. From the table, it can be observed that the time required by the multigrid method for all the Reynolds number and Grashof number combinations are almost similar and its around three seconds except for $\mathrm{Re} = 1000$ and $\mathrm{Gr} = 10^6$ which requires about 5.16 seconds to converge. For a particular value of the Reynolds number, with an increase in the Grashof number, a very subtle increase in the time required by multigrid for convergence can be observed from Table 4.2. On the other hand, for a particular Reynolds number, the time taken by single-grid decreases with the increase in the Grashof number. Another observation that can be drawn from Table 4.2 is that with an increase in the Reynolds number, the performance of



(a) $\mathrm{Re} = 100, \mathrm{Gr} = 10^2$ (b) $\mathrm{Re} = 100, \mathrm{Gr} = 10^4$ (c) $\mathrm{Re} = 100, \mathrm{Gr} = 10^6$

(d) $\mathrm{Re} = 400, \mathrm{Gr} = 10^2$ (e) $\mathrm{Re} = 400, \mathrm{Gr} = 10^4$ (f) $\mathrm{Re} = 400, \mathrm{Gr} = 10^6$

(g) $\mathrm{Re} = 1000, \mathrm{Gr} = 10^2$ (h) $\mathrm{Re} = 1000, \mathrm{Gr} = 10^4$ (i) $\mathrm{Re} = 1000, \mathrm{Gr} = 10^6$

Figure 4.6: *Temperature contours for different Reynolds and Grashof numberon a* 129×129 *grid for mixed convection problem.*

multigrid increases compared to single-grid.

4.5 Summary

From the above discussions, it is clear that the multigrid accelerated streamfunction-velocity formulation can be easily extended to natural and mixed convection problems. For heat transfer problems there is an additional equation for temperature that is being solved using the standard Gauss-Seidel method. In the mixed-convection problem for all Re-Gr combinations up to $\mathrm{Re} = 1000$ and $\mathrm{Gr} = 10^6$, the converged solution is obtained

at about 5 seconds or less, which is a very fast convergence. For example at Re = 1000 and Gr = 10^4, speed-up obtained over single-grid is nearly 99. The method performs very well also for the natural convection problem in a square cavity in which the solution on a 129×129 grid is obtained in about 10 seconds with a speed-up of about 8 over the single-grid. Thus, we see that the multigrid-assisted $\psi - V$ formulation performs very well in a variety of flow situations that include pure fluid flow, natural and mixed convection and fluid-flow problems giving multiple stable solutions. These observations indicate that the present multigrid-assisted streamfunction-velocity method is a very fast and accurate solution procedure for a variety of incompressible viscous flow and heat transfer problems.

Chapter 5

Parallel LBM Using Graphics Processing Unit (GPU)

5.1 CUDA Accelerated Lattice Boltzmann Method

The lattice Boltzmann method is very suitable for implementation on GPUs due to its data locality. Generally, each lattice node of the computational domain is assigned to a thread. The collision and streaming steps are performed by invoking corresponding CUDA kernels. However, these two steps can be combined together into one step, thus, reducing the number of global memory accesses by a factor of two. The main challenge in the GPU implementation of the LBM is the memory access pattern associated with the streaming step. During streaming, threads must access the distribution functions stored in their neighbouring nodes. LBM data structure can be organized mainly in two ways.

1. **Array of Structures (AoS)**: For D3Q19 lattice it is equivalent to $L^3 \times 19$ arrays.
2. **Structure of Arrays (SoA)**: For D3Q19 lattice it is equivalent to $19 \times L^3$ arrays.

For CPU implementations, the AoS is more relevant. However, for GPU implementation of LBM, since each lattice node of the computational domain is assigned to a thread to take advantage of the massively parallel structure of the GPU, the SoA arrangement ensures coalescence of global memory accesses.

The performance of the LBM is commonly measured using Million Lattice Updates Per Second (MLUPS). Tölke and Krafczyk [73, 72] split the domain into an array of one-dimensional blocks. They performed the streaming by shifting in the minor directions using shared memory. Since the shared memory is only accessible to the treads of a block, the

distribution functions that are leaving the block along the minor direction are temporarily stored as incoming distribution functions from the opposite side of the block. A separate kernel is then invoked to place these distributions in their correct locations. This approach ensures that all global memory accesses are fully coalesced, but has the disadvantage of requiring an additional kernel to be launched. Their implementation achieved a maximum performance of 592 MLUPS for the D3Q13 lattice on the GeForce 8800 Ultra.

Obrecht et al. [119] achieved high memory bandwidths on devices with higher compute capabilities using a relatively straightforward "reversed" scheme. This method relies on the fact that the penalties incurred from misaligned memory accesses have been alleviated on the higher compute capability devices (compute capability 1.2 and above), and on the observation that misaligned writes are more costly than misaligned reads. The implementation consists of each thread performing a series of global memory reads that involve misalignments, applying the collision operator, and writing the updated distributions onto global memory in a fully coalesced fashion. Their method achieved a maximum performance of 516 MLUPS for the D3Q19 lattice on the GeForce GTX 295.

Both of the implementations mentioned above require two sets of arrays for the domain. They update one array based on the values of the other and swap them after every time step to avoid data overwriting. Other algorithms exist that perform the read and write operations in place. This allows the simulation to use only a single array, thereby reducing memory requirements by half. Myre et al. [120] compared the "ping-pong" scheme that alternates between two sets of grids, with the "flip-flop" and "Lagrangian" schemes that only use one set of data. The "ping-pong" pattern used in their study is similar to the "reversed" scheme used by Obrecht et al., but it performs the collision operation first and then writes the updated distributions to the neighbouring points to complete the propagation. The "Lagrangian" pattern assigns a global memory location for each fluid packet, and the threads are assigned to a fixed point in the simulation domain. At each time step, the threads selectively read the fluid packets that would arrive at its location, perform the collision step, and write the updated values to the original memory locations. In the "flip-flop" pattern, two different read/write methods are used alternately for each time step. In the first time step, each thread reads the distribution functions from its node, applies the collision operator, and writes the updated information in the same node location, but in reversed directions. In the second time step, the threads now read the incoming distributions from the adjacent nodes, and subsequently write to the adjacent

nodes. Their study showed that the "Lagrangian" method is the fastest among the three memory access patterns, which achieved a maximum performance of 444 MLUPS for the D3Q19 lattice on the Tesla C1060.

Astorino et al. [121] presented a single grid method that employs the swapping technique [122] for streaming operation. This technique involves a series of swapping operations between distributions in the opposite directions on adjacent nodes, thus eliminating data overwriting. The limitation of this streaming method in the context of GPU computing is that it does not allow the collision and streaming steps to be combined into one kernel. This is because the swapping technique assumes knowledge of the order of the nodes at which the algorithm is applied. Since all threads execute concurrently, one cannot know the order in which the nodes are executed a priori, and will have to rely on kernel synchronizations to ensure that the collision operation is only applied to the nodes that have completed the swapping with all adjacent nodes. With this method, they report a maximum performance of 375 MLUPS on the D3Q19 lattice using the GeForce GTX 480.

Schönherr et al. [123] was the first to report on the implementation of a non-uniform grid LBM on the GPU. Their grid refinement scheme is based on the method by Filippova and Hänel [124], and their GPU implementation is based on the shared memory method proposed by Tölke and Krafczyk [73, 72]. For the interpolations involved at the grid interface, they use a compact second-order spatial interpolation, where the moments are interpolated from the four (eight for 3D) closest nodes by using a quadratic bubble function, and subsequently converted back into distribution functions [125, 126]. They also avoid using time interpolations by selectively discarding the invalid outer nodes after each fine mesh time step. The details regarding the algorithm used to complete the data transactions for the interpolations were not given, but it involves using a pre-computed list of interpolation cells that provide the indices of the associated nodes at the grid interface.

In Wang and Aoki's [127] large scale multi-node GPU implementation study, the LBM was run using up to 100 Tesla S1070 GPUs, with 2 GPUs per node. The inter-GPU communication was managed using a combination of CUDA API `cudaMemcpy` and MPI. They compared the efficiencies of 1D, 2D, and 3D domain partitioning methods, and concluded that when the number of GPUs is < 10, 1D decomposition proves to be sufficient, but for > 10 GPUs, 2D and 3D partitioning becomes important, as the size of the inter-GPU communication can be reduced. Obrecht et al. [128] also investigated the scalability of multi-GPU LBM codes. Their hardware configuration consisted of a single node connected

to eight GPUs. With 1D domain partitioning, they observed a nearly perfect strong scalability for a sufficiently large domain.

5.2 Implemetation Details

In the present work, we have invoked two kernels: one for collision and one for streaming. After the completion of the streaming kernel, a swap function is called to swap the array pointers. The updating of the macroscopic variables, the implementation of the boundary condition and the collision operation are clubbed into the collision kernel. The optimal block size is dependent on the size and shape of the domain. In this work, for 2D and 3D blocks of size 16×16 and $8 \times 8 \times 8$ respectively yield good performance.

In this study, the number of GPUs to be used was limited to two, due to hardware availability. Since the nodes on the machine Param-Ishan's 'GPU' cluster (HPC cluster of IIT Guwahati) are each equipped with two Tesla K40 GPUs, the current implementation does not require the use of MPI for inter-node communication. For inter-GPU communication between GPUs that share the same node, memory transfer is achieved by first copying the source GPU data back to the host memory via `cudaMemcpy`, and subsequently invoking an additional `cudaMemcpy` from host memory to the destination GPU. This two-step procedure can be completed in a single CUDA command by using CUDA's peer-to-peer memory access feature, `cuMemcpyPeerAsync`, which is one of NVIDIA's developments to promote multi-GPU programming. For efficient parallelization, it is critical to perform the inter-GPU memory transfers in an asynchronous fashion to ensure that the communication is overlapped with computation. To do so, two *CUDA streams* are created to separately manage each zone's halo nodes and inner nodes. While the halo data is being exchanged, the inner nodes are marched in time. As long as the inner node computation takes a longer time to complete than the data exchange in the halo region, the computational resources of the GPU will always be fully utilized, thus leading to efficient parallelization.

In a GPU, a collection of 32 threads are grouped together and is called a *warp* and all the threads of a warp are executed simultaneously. To utilize the full possible power of a GPU, one needs to invoke a large number of threads so that threads per streaming multiprocessor (SM) is significantly more than the streaming processors (SP)s per SM. SPs are also known as CUDA cores. For each compute capability of the GPU, there is a certain number of threads that can reside in one SM at a time. All the blocks that are defined are queued and they wait for an SM to have the resources (number of SPs) free, then it is

loaded to the SM and then the SM executes the warps. It loads the warps which have all the 32 threads ready with their data. Since one warp only has 32 threads and the SM of GTX 970 GPU has 128 SPs, one SM can execute 4 warps at a given time. If a thread in a warp has to perform memory access for its computation, the thread will block the warp's execution until its memory request is satisfied. In the meanwhile, another ready warp will be executed by the SM.

For example, an arithmetic calculation on the SP has a latency of 18-22 cycles while non-cached global memory access can take up to 300-400 cycles. This means if the threads of one warp are waiting for its data, only a subset of the 128 SPs would work. Therefore, the scheduler switches to execute another warp if available. And if this warp is not ready with its data it executes the next one and so on. This concept is called *latency hiding*. The number of warps and the block size determine the occupancy. If the occupancy is high, it is more likely that SPs are busy with computation.

All the computations are carried out on a system having intel-i5 4690K 4-core CPU @3.50 GHz, 16 GB DDR3 @1600 MHz RAM and nVIDIA Geforce GTX 970 Graphics Card. The GTX 970 GPU has 13 Streaming Multiprocessors (SMs) with 128 CUDA cores each, resulting in a total of 1664 CUDA cores. This GPU has a 1050 MHz base clock, 4 GB GDDR5 @7GHz memory with a 256-bit memory bus @224 GB/s and CUDA compute capability 5.2. The GTX 970 GPU has the following hardware limitations:

1. Maximum thread blocks per multiprocessor = 32
2. Maximum threads per multiprocessor = 2048
3. Maximum threads per thread block = 1024
4. Threads per Warp = 32
5. Maximum warps per Multiprocessor = 64
6. Max shared memory per block (bytes) = 49152
7. Shared memory per multiprocessor (bytes) = 65536
8. Max Registers per Thread Block = 32768 byte
9. Registers per Multiprocessor = 65536 byte

Since a high occupancy is expected from a parallel CUDA code, the selection of the optimum number of threads and blocks is crucial. Let us take an example to select an

optimum number of threads and blocks referring to the above hardware limitations for the GTX 970 card. If we select a thread block of size 32×32, the total number of threads per block will be 1024, which is the maximum possible threads one block can accommodate. In this case, one SM will execute 2 thread blocks, since the maximum number of threads per SM can be 2048. Even though an SM can accommodate 32 thread blocks, only 2 (2048 threads per SM/1024 threads per block) blocks can occupy per SM at a time because of the limitation imposed by the number of threads per SM. Now, if we select a thread block of size 16×16 or 8×8, the total number of threads per block will be 256 and 64 respectively. Here, an SM will execute 8 (2048/256) and 16 (2048/64) thread blocks respectively. On the other extreme, for a block of size 4×4, each block will have 16 threads. In this case, 2048 threads can be grouped into 128 (2048/16) blocks, but only 32 blocks (i.e., a total of 512 threads) out of these 128 will go to an SM at a time, leaving some CUDA cores sitting idle. Furthermore, based on the fluid flow problem to be solved, the amount of shared and register memory can also impose restrictions on the selection of thread block size. Hence, depending on the simulation, a selection of proper thread block size can increase the occupancy of threads per SM and also increase the speedup.

A number of different benchmark problems are solved using parallel LBM on a GTX 970 GPU. The problem of heat diffusion in a two-dimensional domain, the lid-driven fluid flow in square and cubic cavities, buoyancy-induced two-dimensional natural convection, heated lid-driven cavity and mixed convection in a differentially heated square cavity have been chosen for the study.

5.3 Benchmark Problems for Lattice Boltzmann Method

Here, three different benchmark problems are simulated for isothermal and thermal flows to check the applicability of the self-developed LBM codes (C/C++). The problem of the lid-driven cavity, buoyancy-induced natural convection and mixed convection in the cavity are considered the fundamental benchmark problems to test computational schemes for isothermal and thermal flows.

5.3.1 Lid-driven cavity flow in a square and cubic cavity

The flow configuration with boundary condition for the lid-driven square cavity flow is shown in Figure 5.1a. In this configuration, the upper horizontal lid moves towards the right with constant speed U while the other three walls are stationary. The flow is characterised

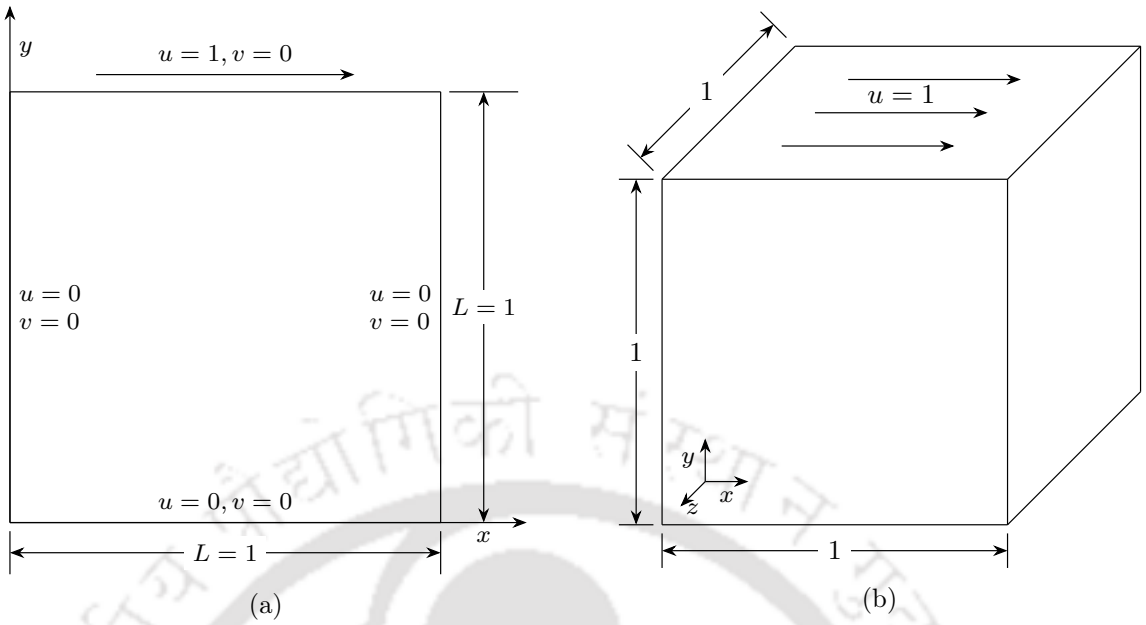


Figure 5.1: *Schematic diagram of the flow configuration with boundary conditions for (a) single lid-driven square cavity, (b) single lid-driven cubic cavity.*

by the Reynolds number, $\mathrm{Re} = UL/\nu$, where L is the length of the cavity. The results are presented for $\mathrm{Re} = 100$, 3200 and 7500 in a uniform grid size of 201×201.

To check the validity of both isothermal SRT-LBM and MRT-LBM models, the comparison of present results is made with the results of Ghia et al. [4] in terms of the horizontal velocity u along the vertical centerline and the vertical velocity v along the horizontal centerline of the cavity (Figure 5.2). The results show a close agreement. However, with the MRT-LBM model, the converged solution is obtained with less time which signifies the fact that it is more stable compared to the SRT-LBM model. The SRT-LBM and MRT-LBM models are then used to perform three-dimensional computation of lid-driven problem in a cubic cavity (Figure 5.1b) for $\mathrm{Re} = 100, 400$ and 1000 with uniform grid size of $81 \times 81 \times 81$. Figure 5.3 shows the comparison of SRT-LBM and MRT-LBM results with the published results of Ku et al. [129] in terms of centreline u-velocity and v-velocity at mid-span (i.e., $z = 0.5$) of the cubic cavity and it displays a good agreement.

5.3.2 Natural convection in a square cavity

Figure 5.4 shows the flow configuration and boundary conditions for natural convection flow in an air-filled square cavity ($\mathrm{Pr} = 0.7$). The flow configuration involves a closed square cavity of unit length where all the walls of the cavity are stationary. The horizontal walls are thermally insulated, and the left vertical wall is maintained at an isothermally

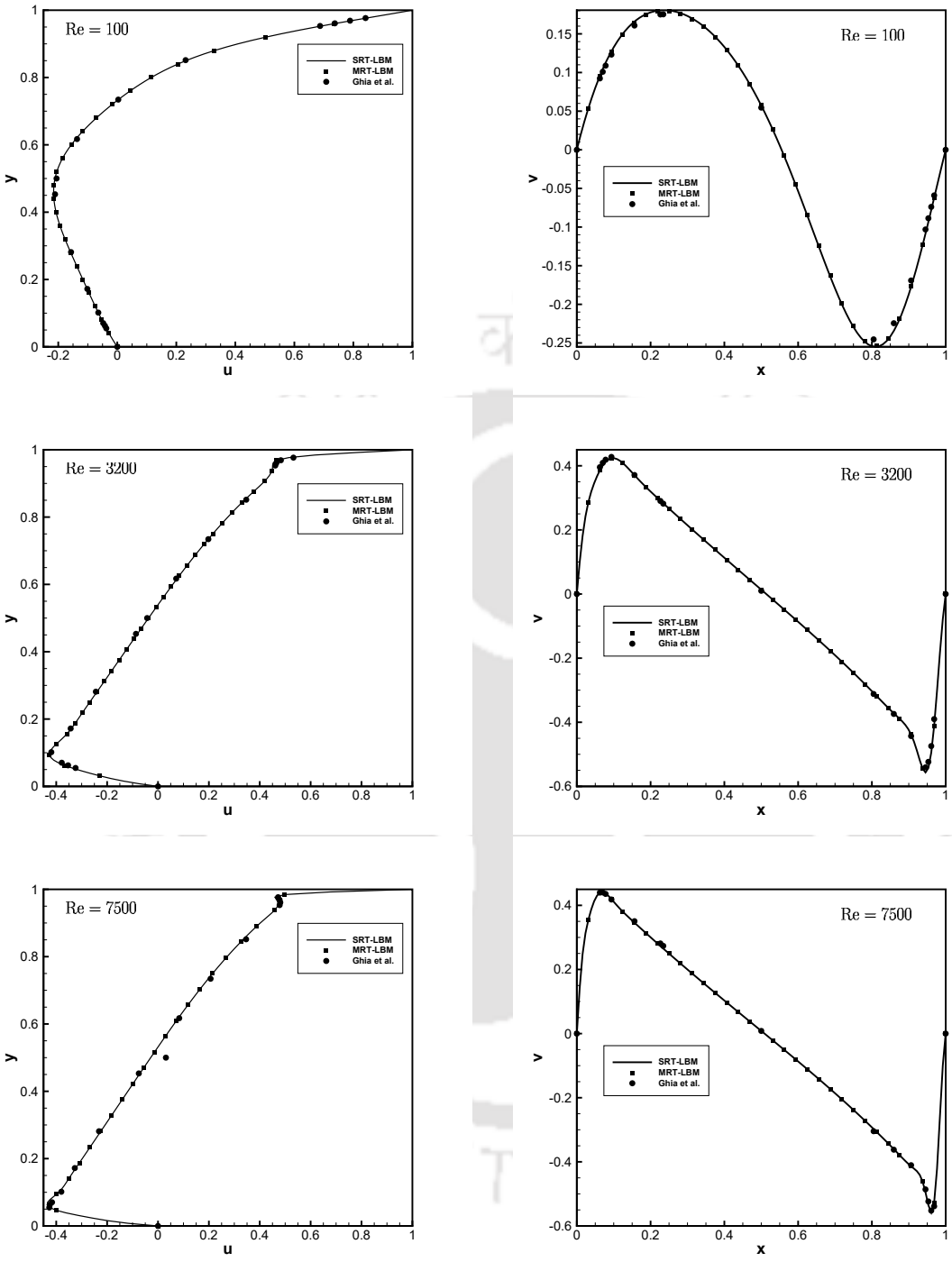


Figure 5.2: *Comparison of results in terms of dimensionless horizontal velocity u at $y = 0.5$ (left panel) and vertical velocity v at $x = 0.5$ (right panel) with Ghia et al. [4] for $Re = 100$ (top panel), $Re = 3200$ (middle panel) and $Re = 7500$ (bottom panel).*

hot and the right vertical wall at an isothermally cold temperature.

Grid sensitivity analysis is carried out to obtain grid-independent solutions for each of three different values of Ra for the steady-state natural convection in the air-filled square

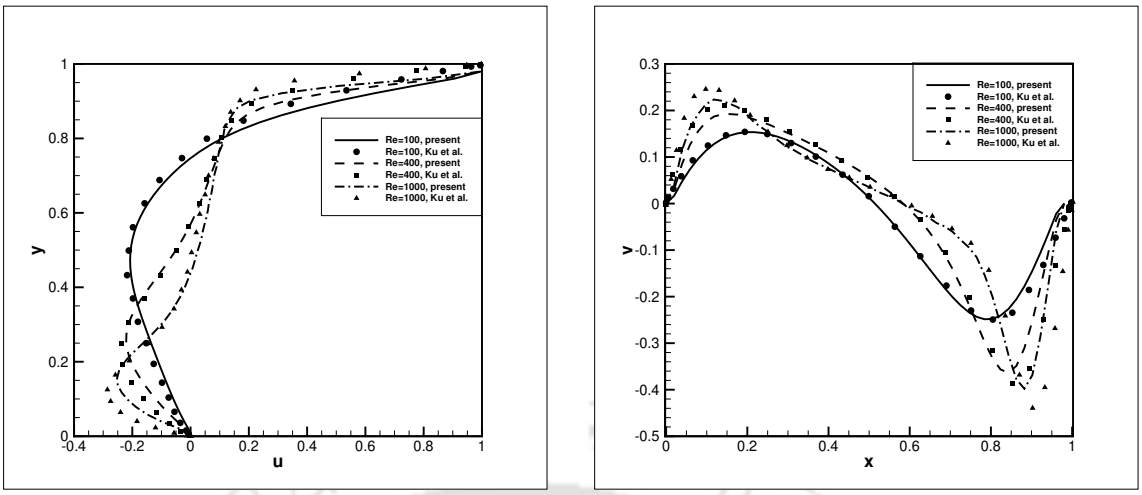


Figure 5.3: *Comparison of velocity profiles on vertical centerline (left) and horizontal centerline (right) of cubic cavity at* $Re = 100, 400$ *and* 1000.

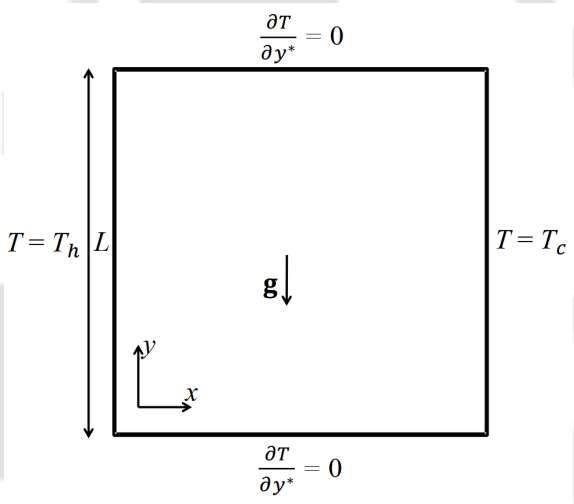


Figure 5.4: *Schematic diagram of the flow configuration with boundary conditions for natural convection in a square cavity.*

cavity. Table 5.1 gives the comparison of the results on various grid sizes in terms of the dimensionless maximum vertical velocity (v_{max}) at the horizontal midline of the cavity, and the average Nu along the hot left wall of the cavity denoted by $\overline{\mathrm{Nu}}_{lh}$. The grid of size 151×151 for $\mathrm{Ra} = 10^3$, 201×201 for $\mathrm{Ra} = 10^4$ and 301×301 for $\mathrm{Ra} = 10^5$ are found to be suitable for all the computations while taking note of the time-wise efficiency of the numerical simulations.

Table 5.2 shows that the present code successfully reproduces the close approximation of published results of de Vahl Davis [110] and Dixit & Babu [114] in terms of Nusselt number along the vertical hot wall. Figure 5.5 depicts the comparison of the SRT-LBM and MRT-LBM model for varying Ra from 10^3 to 10^5 and close agreement is observed.

Table 5.1: Grid independence study

Variables	Grid Sizes								
	Ra = 10^3			Ra = 10^4			Ra = 10^5		
	101× 101	151× 151	201× 201	151× 151	201× 201	251× 251	251× 251	301× 301	351× 351
v_{max}	3.601	3.636	3.653	19.298	19.388	19.441	67.545	67.714	67.843
$\overline{Nu_{lh}}$	1.096	1.102	1.105	2.188	2.200	2.207	4.414	4.431	4.442

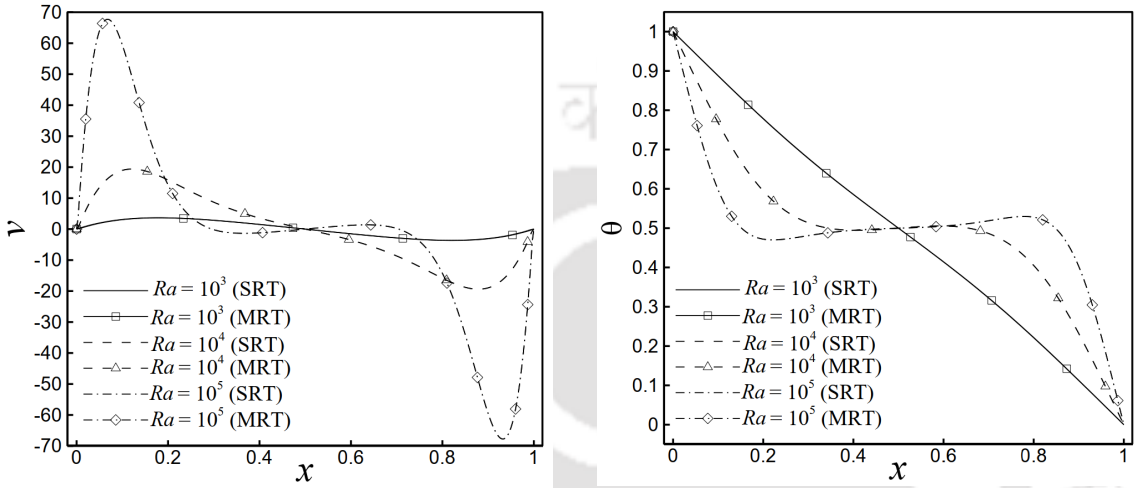


Figure 5.5: *Comparison of dimensionless vertical velocity v (left) and temperature θ (right) in the horizontal mid-plane of square cavity at different Ra* = $10^3, 10^4$ *and* 10^5.

Table 5.2: Comparison of the average Nusselt number (Nu) at the left wall for natural convection problem.

Ra	Average Nusselt number at the left wall		
	Present	de Vahl Davis [110]	Dixit & Babu [114]
10^3	1.102	1.116	1.127
10^4	2.20	2.242	2.247
10^5	4.431	4.523	4.523

Table 5.3: Comparison of present work with published results for higher Ra.

Ra	Method	y for u_{max}	u_{max}	x for v_{max}	v_{max}	$\overline{Nu}$
10^6	Present SRT	0.850	63.56	0.040	215.6	8.546
	Present MRT	0.851	63.852	0.0399	216.07	8.579
	de Vahl Davis [110]	0.850	64.63	0.0379	219.36	8.799
10^7	Present SRT	0.880	142.41	0.024	669.56	15.62
	Present MRT	0.884	143.66	0.022	676.90	15.847
	LeQuéré [130]	0.879	148.58	0.021	699.23	16.52
10^8	Present SRT	0.930	300.56	0.013	2089.48	27.935
	Present MRT	0.931	304.08	0.013	2109.22	28.342
	LeQuéré [130]	0.928	321.87	0.012	2222.39	30.225

To check the robustness of both SRT-LBM and MRT-LBM calculations at higher Rayleigh numbers, the present study deals with the simulation of natural convection from Ra = 10^6 to Ra = 10^8 as depicted in Table 5.3. A comparison of current results with the solutions available in the literature shows satisfactory agreement.

5.3.3 Mixed convection in a square cavity

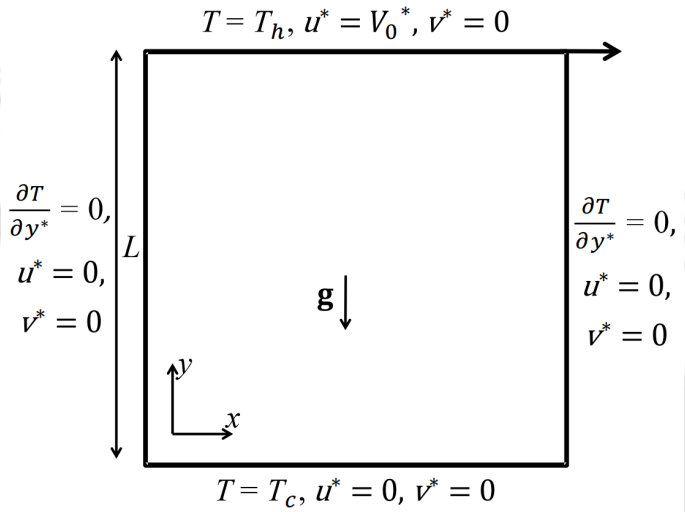


Figure 5.6: *Schematic diagram of the flow configuration with boundary conditions for single lid-driven mixed convection in cavity.*

The present LBM code is validated for mixed convection flow problem in a clear square cavity with a horizontally moving hot top lid and adiabatic sidewalls, as shown in Figure 5.6. The grid size of 301 × 301 is considered to carry out the simulations.

Figure 5.7 shows the variation of dimensionless temperature along the vertical midplane of square cavity for Re = 10^3 and Gr = 10^6 at Pr = 0.01 and 0.71, which provides a good match of the present result with Iwatsu et al. [5].

The results are then compared with the published works of Iwatsu et al. [5] for fixed Gr = 100 and different Re of 100, 400 and 1000 respectively. The results are presented in Table 5.4 in terms of minimum dimensionless horizontal velocity (u_{min}) at vertical mid-plane, minimum and maximum dimensionless vertical velocity (v_{min} and v_{max}) at horizontal mid-plane and the average Nusselt number ($\overline{\mathrm{Nu}_t}$) along the top wall. The comparison of results shows a good agreement.

These computations are carried out so as to be able to assess the timewise performance of present CUDA parallelization.

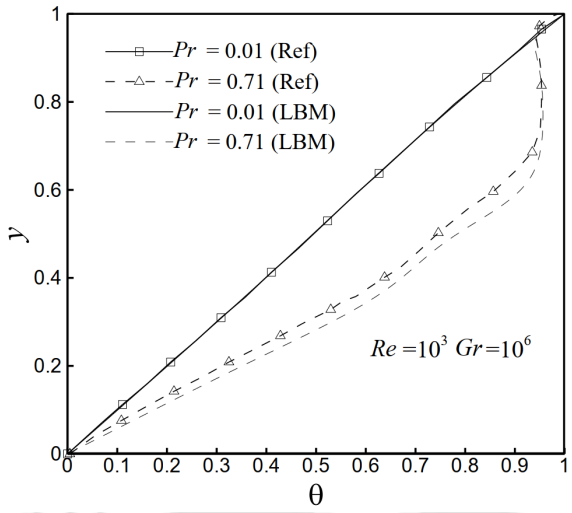


Figure 5.7: *Comparison of present results (LBM) with Iwatsu et al. [5] (Ref) in terms of dimensionless temperature (θ) profile along the vertical mid-plane of square cavity for* Pr = 0.01 *and* 0.71 *at* Re = 10^3 *and* Gr = 10^6.

Table 5.4: Comparison of present work with published results of Iwatsu et al. [5]

Variables	Re = 100		Re = 400		Re = 1000	
	LBM	Reference	LBM	Reference	LBM	Reference
u_{min}	−0.2129	−0.2037	−0.3280	−0.3197	−0.3880	−0.3782
v_{min}	−0.2495	−0.2448	−0.4521	−0.4459	−0.5258	−0.5178
v_{max}	0.1764	0.1699	0.3029	0.2955	0.3767	0.3658
$\overline{Nu_t}$	2.02	1.94	4.029	3.84	6.454	6.33

*Reference (Iwatsu et al. [5])

5.4 CUDA-Parallelized Lattice Boltzmann Computations

5.4.1 Two-Dimensional Heat Diffusion

The steady-state temperature distribution on a 2D rectangular plate is obtained using LBM and is accelerated with the help of CUDA. The governing equation for steady 2D heat conduction without heat generation is a Laplace equation:

$$\nabla^2 T = \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0 \tag{5.1}$$

The computational domain is shown in figure 5.8. Here, the temperature of the bottom wall (T_1) is 100 and the temperatures of all the other walls (T_2, T_3 and T_4) are 0. The total number of grid points along $x-$ and $y-$ directions (i_{max} and j_{max}) are 101 and 201 respectively. D2Q9 lattice model is used to solve this problem using LBM. For the heat-

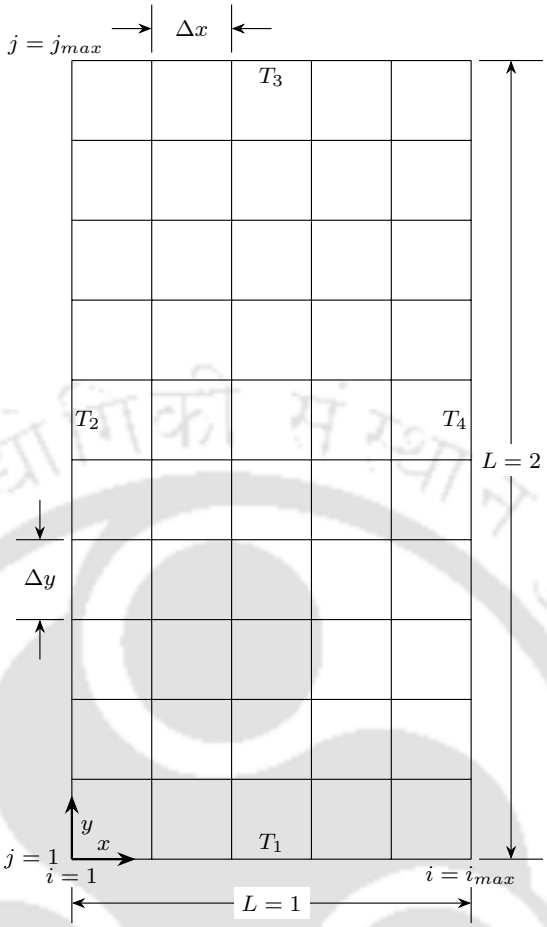


Figure 5.8: *Two-dimensional heat diffusion problem domain*

diffusion problem, we have used a CUDA block size of 16×16 and obtained a speed of $16\times$ when compared to the serial LBM code. The temperature contours is given in figure 5.9.

5.4.2 Lid-driven cavity flow in a square cavity

Revisiting the lid-driven cavity problem, the flow configuration with boundary condition for the lid-driven square cavity flows is shown in figure 5.10. The upper horizontal lid moves towards the right with constant speed U while the other three walls are stationary. A uniform grid of size 201×201 and a D2Q9 LBM-lattice model is used to carry out the simulation.

Table 5.5 shows the speedup of CUDA-LBM with serial LBM for different CUDA blocks and Reynolds numbers. From the table, it can be seen that for all the Reynolds numbers, with the increase in the CUDA block size from 4×4 to 16×16, the speed up increases. For instance, for Re= 100 the speedup increases from $17.89\times$ to $21.12\times$. However, with a further increase in the block size to 20×20 the speed up deteriorates. For Re= 100, speed

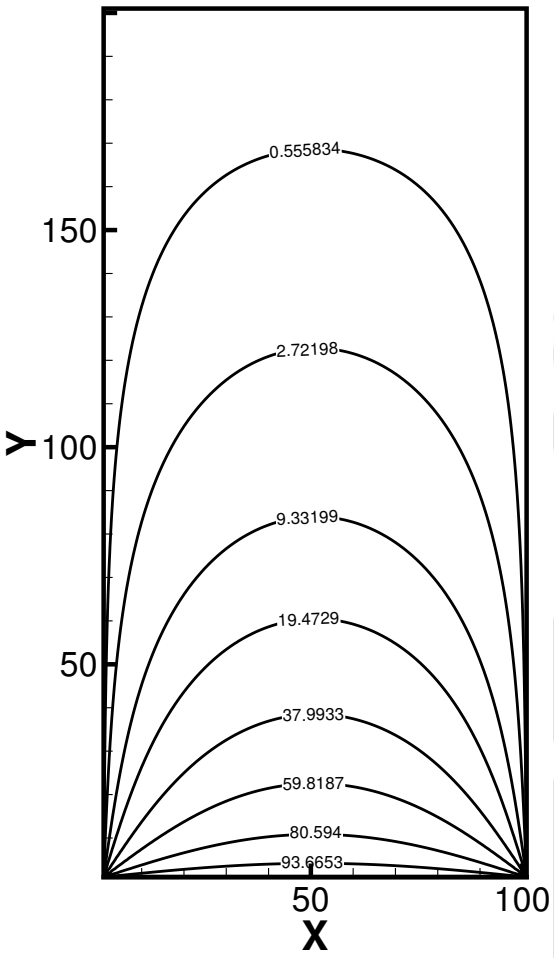


Figure 5.9: *Temperature contours of the two-dimensional heat diffusion problem*

up reduces to 19.86×. Moreover, with an increase in the Reynolds number from 100 to 7500, a decrease in speedup can be observed for all the thread block configurations.

Table 5.5: Speed of GPU compared to CPU for different CUDA block configurations for lid-driven square cavity

Re	CUDA Block Size			
	4×4	8×8	16×16	20×20
100	17.89X	19.47X	21.12X	19.86X
3200	16.15X	17.74X	19.08X	18.02X
7500	14.06X	15.83X	17.27X	16.14X

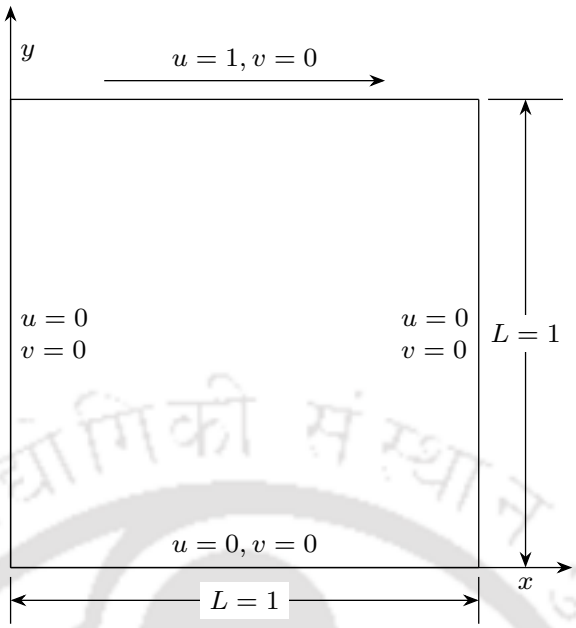


Figure 5.10: *Schematic diagram of the flow configuration with boundary conditions for single lid-driven square cavity*

Table 5.6: Speed of GPU compared to CPU for different CUDA block configurations for lid-driven cubic cavity

Re	CUDA Block Size		
	$4 \times 4 \times 4$	$8 \times 8 \times 8$	$16 \times 16 \times 16$
100	14.3X	15X	14.17X
1000	13.5X	14.23X	13.40X

5.4.3 Lid-driven cavity flow in a cubic cavity

The flow configuration with boundary condition for the lid-driven cubic cavity flows is shown in Figure 5.11. The upper horizontal lid moves towards the right with constant speed U while all the other walls are stationary. A uniform grid size of $49 \times 49 \times 49$ and $51 \times 51 \times 51$ are selected for Reynolds numbers 100 and 1000 respectively and the D3Q19 LBM-lattice model is used.

Table 5.6 shows the speedup of CUDA-LBM with serial LBM for different CUDA blocks and Reynolds numbers 100 and 1000. From the table, it can be seen that the CUDA thread block of size $8 \times 8 \times 8$ yields the highest speed-up compared to $4 \times 4 \times 4$ and $16 \times 16 \times 16$ thread blocks. Moreover, with an increase in the Reynolds number from 100 to 1000, a decrease in speed up can be observed for all the thread block configurations.

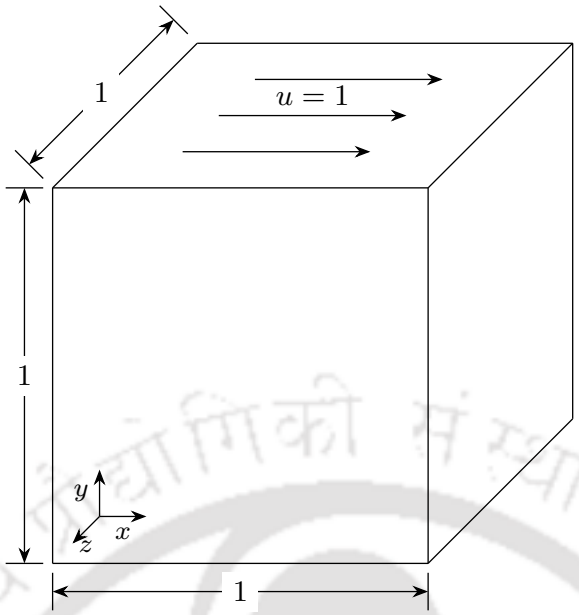


Figure 5.11: *Schematic diagram of the flow configuration with boundary conditions for single lid-driven cubic cavity.*

5.4.4 Two-Dimensional Natural Convection

The schematic of the 2D computational domain of the natural convection in a square domain is as shown in Figure 5.12. Here, the left wall is at a higher temperature, and the right wall is at a lower temperature. The top and bottom walls are insulated, and at the boundaries, both the velocity components are zero. Grids of size 129×129 for $Ra = 10^3$, 161×161 for $Ra = 10^4$, 191×191 for $Ra = 10^5$ and 221×221 for $Ra = 10^6$ are used.

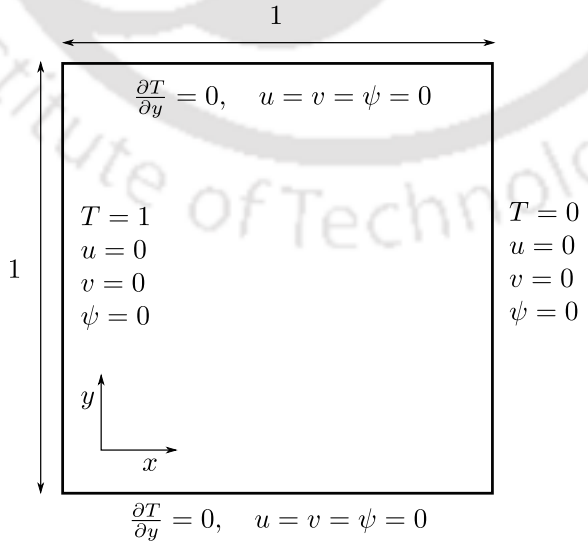


Figure 5.12: *Schematic diagram of the flow configuration with boundary conditions for two-dimensional natural convection.*

Table 5.7: Speed of GPU compared to CPU for different CUDA block configurations for 2D natural convection.

Ra	CUDA Block Size			
	4×4	8×8	16×16	20×20
10^3	9.86X	11.38X	12.93X	11.85X
10^4	9.34X	10.89X	12X	11.24X
10^5	8.17X	9.32X	10.51X	9.67X
10^6	6.67X	7.24X	8.42X	7.61X

Table 5.7 shows the speed up of CUDA-LBM with serial LBM for different CUDA blocks and Rayleigh numbers. From the table, it can be seen that for all the Rayleigh numbers, with the increase in the CUDA block size from 4×4 to 16×16, the speed up increases and then decreases with further increase in the block size to 20×20. For instance, for Ra= 10^4 the speedup increases from 9.34× to 12× and then decreases to 11.14× with the change in the block size. Moreover, with increase in the Rayleigh number from 10^4 to 10^6, a decrease of speedup can be observed for all the thread block configurations.

5.4.5 Two-Dimensional Heated Single Lid-Driven Cavity

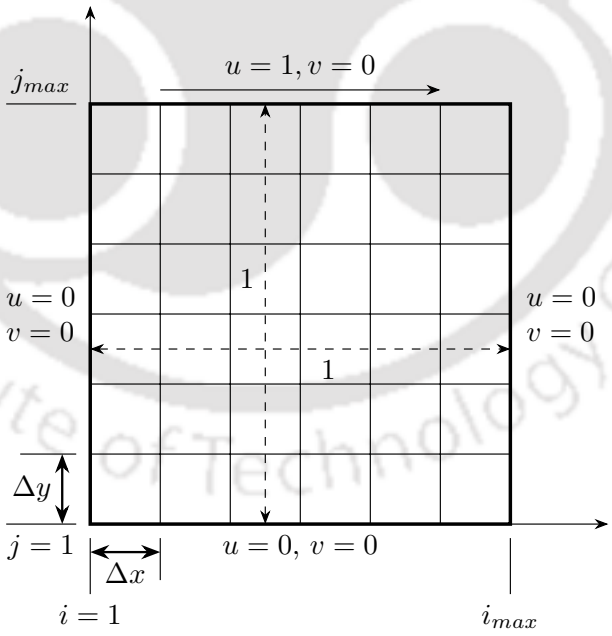


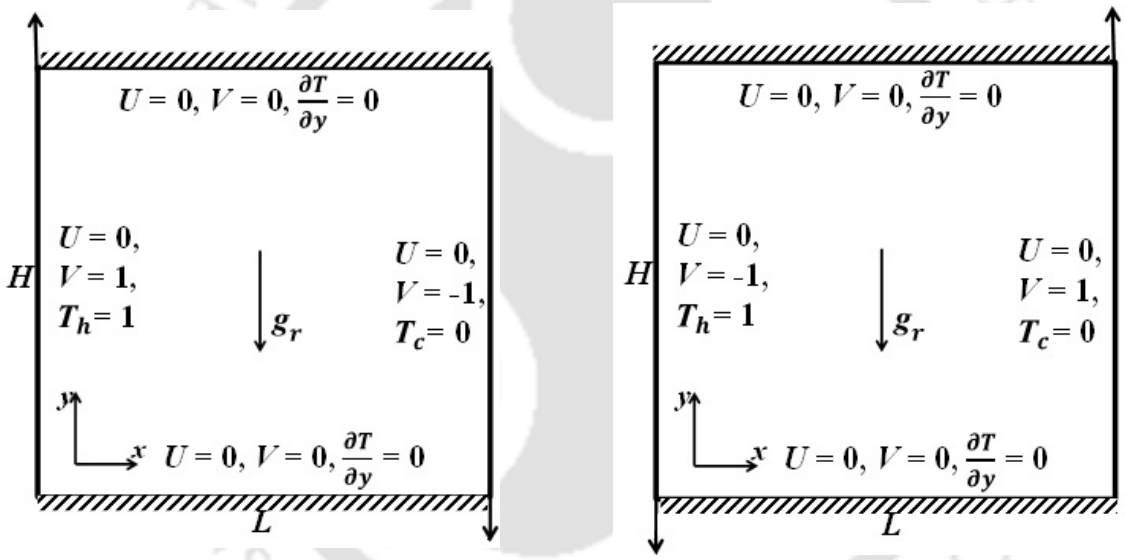
Figure 5.13: *Schematic diagram of the flow configuration with boundary conditions for single lid-driven square cavity.*

The computational domain for the single lid-driven square cavity flows is shown in figure 5.13. The square cavity is filled with air having viscosity $\nu = 0.02$(lattice units), and Prandtl number $Pr = 0.71$. The upper horizontal lid moves towards the right with

constant speed $U_{lid} = 0.2$ while all the other walls are stationary. The moving lid is heated to a normalized temperature of 1.0 and the east and west walls are kept at cold normalized temperature of 0.0. The bottom wall is thermally isolated. A uniform grid size of 129×129 is selected for Reynolds number 1000 and D2Q9 LBM-lattice model is used for the computation.

For this problem, with a CUDA thread block size of 16×16, a speedup of 9.5X has been observed.

5.4.6 Two-Dimensional Double Lid-Driven Mixed Convection in a Differentially Heated Square Cavity with Aiding and Opposing Effects



(a) aiding mechanism

(b) opposing mechanism

Figure 5.14: *Flow configuration for mixed convection in a differential heated square cavity.*

Two cases depending on the relative directions of shear-driven flow are studied to discuss the combined effect of both aiding and opposing mechanisms of mixed convection flow in a square cavity by varying Richardson number (Ri) discretely from 0.01 to 100, keeping Prandtl number (Pr) fixed at 0.7 and Reynolds number (Re) at 100. 191×191 grid size is used for all computations. For the termination of all computations, the convergence criterion for steady state solution is taken as $\left|\max\left(T^{t+\Delta t}\left(\mathbf{r}\right) - T^{t}\left(\mathbf{r}\right)\right)\right| < 10^{-8}$, where t is the time. The model configurations and boundary conditions for aiding and opposing mechanism of mixed convection flow in a square cavity are shown in figure 5.14a and 5.14b respectively.

The simulation is carried out on a grid of size 191×191 for Richardson number varying from $0.01 - 100$ at fixed Re$(= 100)$ and Pr $(= 0.7)$. Table 5.8 indicates that average Nusselt number (Nu) throughout the cavity $(\overline{Nu})$ and the average Nu along the vertical midplane of the cavity $(\overline{Nu_{\frac{1}{2}}})$ increase with Ri for a Re fixed at 100 and a Pr at 0.7 due to the aiding mechanism of mixed convection flow. For the opposing mechanism, Nu decreases and its value approaches 1 with an increase in value of Ri up to 1 since both forced- and free-convection influence the heat transfer but oppose each other resulting in conduction-like heat transfer. With further increase in values of Ri, average Nu increases as free convection dominates the flow and heat transfer.

Table 5.8: Variation of $\overline{Nu}$ and $\overline{Nu_{\frac{1}{2}}}$ with different Ri for two mechanisms

Ri	Aiding		Opposing	
	$\overline{Nu}$	$\overline{Nu_{\frac{1}{2}}}$	$\overline{Nu}$	$\overline{Nu_{\frac{1}{2}}}$
0.01	4.569	4.570	4.567	4.568
0.1	4.590	4.592	4.545	4.548
1.0	4.759	4.771	1.279	1.282
10.0	5.875	5.878	2.666	2.668
100.0	8.996	9.033	6.835	6.841

Table 5.9: Grid Independence test

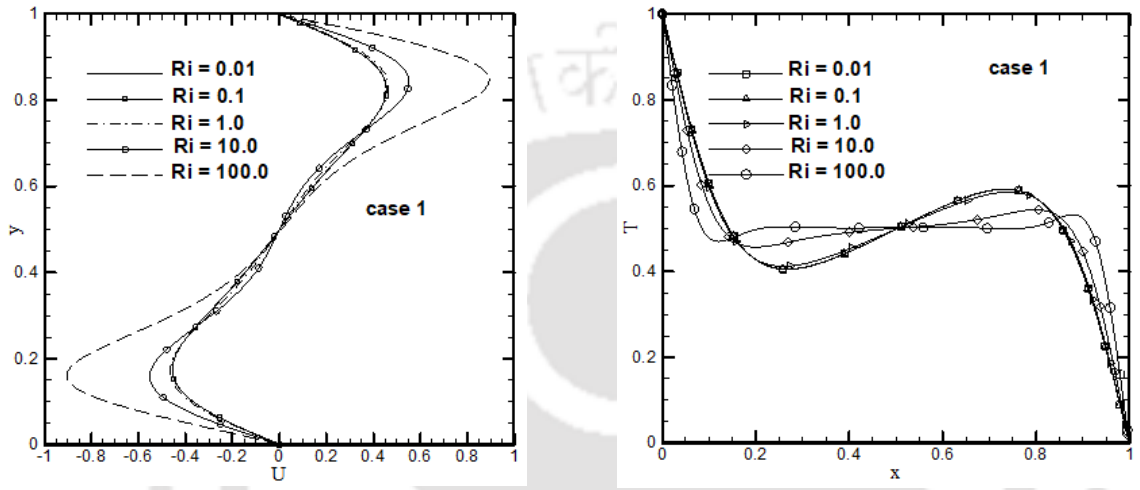
Variables	Grid Size		
	129×129	191×191	257×257
U_{max}	0.464	0.465	0.465
$\overline{Nu}$	4.740	4.759	4.769
$\overline{Nu_{\frac{1}{2}}}$	4.761	4.771	4.777

The result of the grid independence test is shown in table 7.5 shows the grid independence test. Here, we have selected three different grids 129×129, 191×191 and 257×257 to carry out the computation. From the table we can see that the difference of U_{max}, $\overline{Nu}$ and $\overline{Nu_{\frac{1}{2}}}$ are small between the grids 191×191 and 257×257 and hence the grid of size 191×191 is selected for this problem. The table 5.10 shows the speedup for mixed convection problem for grid size 191×191 with different block sizes. It is observed from the table that the block size of 16×16 gives the highest speed up and 8×8 thread block gives the poorest speed up. However, increasing the thread block size from 16×16 to 20×20 shows a reduction in performance. With the change of Ri from 0.01 to 100, the change in speedup for each block size is observed to be negligible. Moreover, for both aiding and

opposing mechanisms of mixed convection, we have observed a similar speedup as reported in the table 5.10.

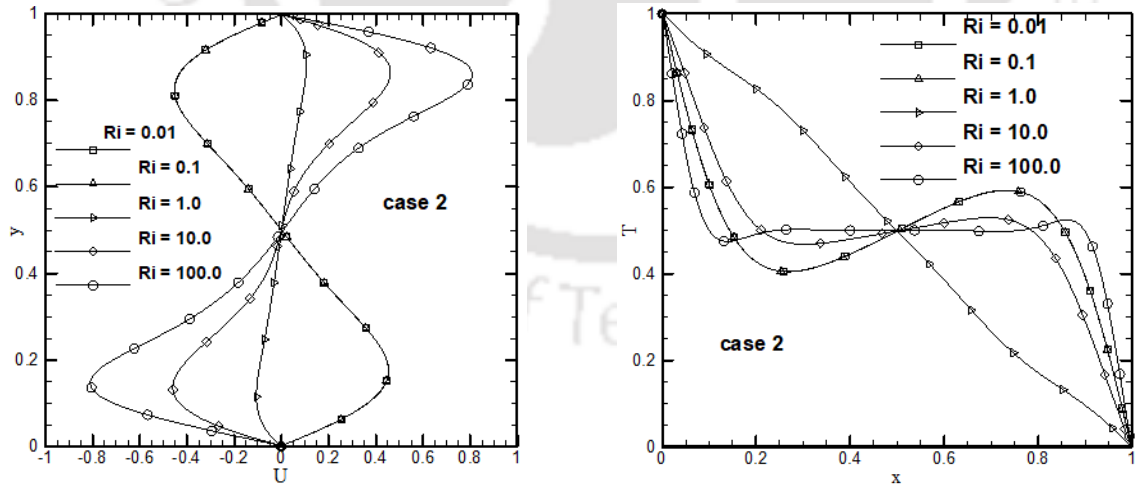
Table 5.10: Speed of GPU compared to CPU for Ri=1

Lattice Size	GPU Block Size			
	8×8	12×12	16×16	20×20
191×191	7.12X	7.61X	8.07X	7.8X



(a) (b)

Figure 5.15: *Comparisons of (a) velocity U in the vertical mid-plane and (b) Temperature T in horizontal mid-plane for Aiding Mechanism (case 1).*



(a) (b)

Figure 5.16: *Comparisons of (a) velocity U in the vertical mid-plane and (b) Temperature T in horizontal mid-plane for Opposing Mechanism (case 2).*

Figures 5.15a and 5.16a show the u-velocities along a vertical line through the geometric centre for both the aiding the opposing mechanism respectively for different Richardson

numbers. Similarly, the figures 5.15b and 5.16b shows the temperatures in a horizontal mid-plane for both the aiding and opposing mechanisms respectively at different Ri. The results are found to match well with that of the published results of Ghia et al. [4]. Similarly, in Figures 3.4a - 3.4d the comparison of v-velocity along a horizontal line through the geometric centre for different Reynolds numbers has been plotted, which shows good agreement with the result of Ghia et al. [4]. Table 7.7 shows the comparison of the present work with the published work of Iwatsu et al. [5]. From the table, it can be seen that our results are in good agreement with the published results.

Table 5.11: Comparison of the present work with the published results of Iwatsu et al. [5] at $Re = 400$

Variables	Method	$\mathrm{Gr} = 10^2$	$\mathrm{Gr} = 10^4$	$\mathrm{Gr} = 10^6$
U_{min}	Iwatsu (FDM)	-0.3197	-0.3095	-0.2632
	LBM	-0.3213	-0.3108	-0.2634
V_{max}	Iwatsu (FDM)	0.2955	0.2837	0.00724
	LBM	0.2958	0.2834	0.00758
$\overline{Nu_T}$	Iwatsu (FDM)	3.84	3.62	1.22
	LBM	3.87	3.63	1.21

5.4.7 Summary

In this chapter, the simulations of different benchmark isothermal and thermal flow problems are carried out to validate the developed LBM codes (written in C/C++) based on both single-relaxation-time (SRT) and multiple-relaxation-time (MRT) lattice Boltzmann models. The comparison of the current result shows a good match with the published results obtained through conventional numerical techniques. The comparison of SRT and MRT results indicates that both models provide comparable solutions while dealing with the laminar flow regime specified by Rayleigh number of the order of 10^6. However, MRT-LBM is found to be more stable than the SRT-LB model in solving flow with high Ra.

After validating with the published literature, the LBM code is parallelized using CUDA so that it can be run on GPUs. The parallel LBM code has been tested using a number of two- and three-dimensional fluid-flow and heat-transfer problems. Different combinations of thread blocks have been tested and it was found that for two- and three-dimensional problems thread blocks of size 16×16 and $8 \times 8 \times 8$ respectively gave the best performance for both fluid-fluid and heat-transfer cases. Once two-dimensional LBM is successfully parallelized using CUDA, extending it to three-dimensional is relatively straightforward.

After parallelizing the basic benchmark problems, the acceleration of two-dimensional steady mixed convection heat transfer and fluid flow analysis in a differentially heated square cavity has been successfully carried out using CUDA. Both aiding and opposing mechanisms of mixed convection are considered by providing two different velocity boundary conditions at the vertical lids. It is found that parallelized LB code computes about 8 times faster than the serial one and it signifies that LB code is very efficient in GPU technique. Average Nusselt numbers are tabulated to show the effect of heat transfer keeping Re fixed at 100 and Ri varying between 0.01 and 100.

Chapter 6

Multigrid Accelerated Lattice Boltzmann Method

6.1 Introduction

Computational efficiency is always a challenging area of the numerical solution procedure. The advancements in multicore CPUs and many-core GPUs have facilitated the scientific community to solve more complicated and challenging problems within a reasonable time frame. However, the necessity of accelerating the convergence of the underlying algorithms cannot be undermined and hence the development of fast and accurate numerical algorithms is still an active research area in computational fluid dynamics. The lattice Boltzmann method (LBM) is a kinetic theory-based computational approach. Due to a simple two-step "collision and streaming" algorithm, LBM can be very easily parallelized on both CPUs and GPUs. However, while solving steady-state or time-dependent flows with large separations in the relevant time and spatial scales, due to the time marching nature LBM suffers from slow convergence. Multigrid methods, which have been used in traditional CFD methods to implement solvers that exhibit linear scaling with respect to the problem size, offer a potential remedy to this issue.

In 2002, Tölke et al. [66] used multigrid method to solve the discrete Boltzmann equation with a second order finite difference discretization. Their implementation even though improved the convergence of the LBM for low Reynolds numbers (Re = 10), but it compromise the parallel property of the original LBM. Moreover, the efficiency gains diminished for Re $\geq$ 100. Later on, Mavriplis [67] proposed a nonlinear multigrid LBM that retained the two-step procedure of the traditional LBM. This method improved the computational

efficiency of the LBM significantly up to Re = 1000. Patil et al. [68] analyzed the efficiency and implemented a parallel version of the algorithm proposed by Mavriplis for the solution of elliptic PDEs. Armstrong and Peng [69] extended the algorithm proposed by Mavriplis to MRT-LBM and proposed an implementation in three-dimension.

6.2 Multigrid Accelerated Lattice Boltzmann Solver

The multigrid accelerated lattice Boltzmann method uses the nonlinear multigrid i.e., FAS procedure described in section 2.1.4 and 2.1.6.2. This section introduces the MG-LBM algorithm for a general class of problems. Chai and Shi [131] investigated the application of LBM for the Poisson equation. The current section follows the work of Chai and Shi and is illustrated for the D2Q5 lattice. The extension to the D2Q9 lattice and the three-dimensional lattices (for example, D3Q27) is straightforward. Let us consider a variable coefficient Poisson equation as given below:

$$\nabla \cdot (\beta \nabla \phi) = S \tag{6.1}$$

Here, β can be a function of the spatial coordinates. At steady-state, the above equation resembles the variable coefficient diffusion equation. If $\beta = 1$ the above equation simplifies to

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = S. \tag{6.2}$$

Here, ϕ represents the physical quantity of interest and S is the source term. Equation (6.2) is iteratively solved using a pseudo-time-stepping LBM in a multigrid framework. For a single-grid, the equation for the LBM for solution of the Poisson equation is:

$$f_\alpha \left(\mathbf{x} + \mathbf{e}_\alpha \Delta t, t + \Delta t\right) - f_\alpha \left(\mathbf{x}, t\right) = -\frac{1}{\tau}\left[f_\alpha \left(\mathbf{x}, t\right) - f_\alpha^{eq} \left(\mathbf{x}, t\right)\right] + \Delta t \bar{\omega}_\alpha \mathscr{D} S \left(\mathbf{x}, t\right) \tag{6.3}$$

where τ is the dimensionless relaxation time and $\Delta t = \Delta x$ is the lattice time-step. $\tau = 1.0$ signifies instantaneous relaxation to the local equilibrium. $\bar{\omega}_\alpha$ represents the diffusional-weights associated with the lattice directions $\mathbf{e}_0 = (0, 0)$ and $\mathbf{e}_\alpha = (cos[(\alpha - 1)\pi/2], sin[(\alpha - 1)\pi/2]), (\alpha = 1\tilde{}4)$. $\mathscr{D}$ is artificial diffusion coefficient, related to τ as, $\mathscr{D} = \kappa(0.5 - \tau)t$ and considering $CFL = 1.0$. The chosen value of τ should be greater than 0.5. Therefore, $\mathscr{D}$ always becomes negative [131] and effectively proportionally reducing the value of distribution functions. Using the Chapman-Enskog expansion the relationship between τ and

$\mathscr{D}$ can be derived [131]. For the D2Q5 model, $\bar{\omega}_\alpha = 0$, $\bar{\omega}_\alpha = 1/4(\alpha = 1\check{}4)$, and $\kappa = 1/2$. In equation (6.3), the equilibrium distribution function is defined as,

$$f_\alpha^{eq} = \begin{cases} \omega_0 - 1.0)\phi, & \alpha = 0 \\ \omega_\alpha \phi, & \alpha = 1, \cdots, b \end{cases} \tag{6.4}$$

Here, ω_α are the weights associated with the lattice directions and b is the number of lattice directions. For the D2Q5 model, $\omega_\alpha = \bar{\omega}_\alpha$ and $b = 4$. The physical quantity, ϕ is defined as,

$$\phi = \frac{1}{1-\omega_0} \sum_{\alpha=1}^{b} f_\alpha^{eq} = \frac{1}{1-\omega_0} \sum_{\alpha=1}^{b} f_\alpha \tag{6.5}$$

The weighting functions and the value of κ follow certain properties [131] and depend on the specific lattice geometry. Chai and Shi [131] used the Chapman-Enskog expansion to show that the formulation is free from the undesirable time-derivative term and equation (6.2) can be recovered as long as $\mathscr{D} \neq 0$. The classical LB method converges very slowly and is inefficient while solving equation (6.2). However, with the help of the multigrid technique, LBM can achieve a faster convergence towards the "steady-state" solution for the Poisson equation.

In order to apply multigrid equation (6.3) is rearranged in a residual form and equating it to zero for convergence, we have

$$R_\alpha(\mathbf{x}, t) = f_\alpha\left(\mathbf{x} + \mathbf{e}_\alpha \Delta t, t + \Delta t\right) - \left(1 - \frac{1}{\tau}\right) f_\alpha\left(\mathbf{x}, t\right) - \frac{1}{\tau} f_\alpha^{eq}\left(\mathbf{x}, t\right) - \Delta t \bar{\omega}_\alpha \mathscr{D} S\left(\mathbf{x}, t\right) = 0 \tag{6.6}$$

where $R_\alpha(\mathbf{x}, t)$ is the residual along the discrete α direction. Equation (6.3) is analogous to using the Jacobi iterative method to solve the LBE for a steady flow. Therefore, after the collision and streaming steps of the LBM, Mavriplis [67] introduced a under-relaxation step in the MG-LBM algorithm, creating a three-step iterative method analogous to the Jacobi under-relaxation scheme for linear systems. The MG-LBM algorithm still maintains the locality of the LB algorithm while allowing a rapid convergence.

Multigrid requires a number of hierarchically coarse grids. Let us consider a uniform two grid scenario, where h denotes the fine grid and H denotes the coarse grid and the grid spacing (distance between two adjacent grid points) of the coarse grid is twice compared to the fine grid. The microscopic particle velocities for the fine and coarse mesh are represented as $\mathbf{e}_\alpha^h$ and $\mathbf{e}_\alpha^H$, respectively. The multigrid accelerated LB algorithm for the fine grid may

be written in terms of the following three-step procedure:

1. LBM collision with a source term

$$f_{\alpha_h}(\mathbf{x}, \mathbf{t}^*) = f_{\alpha_h}(\mathbf{x}, \mathbf{t}) - \frac{1}{\tau}\left[f_{\alpha_h}(\mathbf{x}, \mathbf{t}) - f^{eq}_{\alpha_h}(\mathbf{x}, \mathbf{t})\right] - \Delta t \bar{\omega}_\alpha \mathscr{D} S_h(\mathbf{x}, t) \tag{6.7}$$

2. Streaming or advection

$$f_{\alpha_h}\left(\mathbf{x} + \mathbf{e}^h_\alpha \Delta t, t^{**}\right) = f_{\alpha_h}(\mathbf{x}, t^*) \tag{6.8}$$

3. Under-relaxation

$$f_{\alpha_h}(\mathbf{x}, t + \Delta t) = \gamma f_{\alpha_h}(\mathbf{x}, t^{**}) + (1 - \gamma) f_{\alpha_h}(\mathbf{x}, t)\,. \tag{6.9}$$

Here, the subscript 'h' represents fine grid quantities and the γ is the under-relaxation factor. Usually, in two-dimensions (2D), $\gamma = 0.8$ for optimal convergence rates based on a theoretical analysis [1]. The solution on the fine grid is,

$$\phi_h = \frac{1}{1 - \omega_0} \sum_{\alpha=1}^{b} f^{eq}_{\alpha_h} = \frac{1}{1 - \omega_0} \sum_{\alpha=1}^{b} f_{\alpha_h}, \tag{6.10}$$

where, $f^{eq}_{\alpha_h}$ is obtained from equation (6.4) by replacing ϕ by ϕ_h.

Let $L_h(u_h) = g$ be a nonlinear problem on the fine grid. The corresponding nonlinear residual is defined as

$$R_h = \mathbf{L}_h(u_h) - g, \tag{6.11}$$

and the coarse grid problem is

$$\mathbf{L}_H(u_H) = \mathbf{L}_H(\hat{I}^H_h u_h) - I^H_h R_h. \tag{6.12}$$

This comprises the Full Approximation Storage (FAS) scheme [76, 1, 27]. The Gauss-Seidel (GS) or Red-Black Gauss-Seidel (RBGS) can also be used as a high-frequency error damping smoother. However, it could compromise the modularity and the parallel scalability properties of the LBM. In the above equation, u_h and u_H are the solution variables on fine and coarse grids, respectively. The operator $\hat{I}^H_h$ represents the point-wise injection. The fine level residuals are transferred to the coarser grid using the restriction operator I^H_h.

Based on the above, a defect correction is defined for the coarse grid as,

$$D_h = R_H\left(\hat{I}_h^H u_h\right) - 2I_h^H R_h\left(u_h\right). \tag{6.13}$$

Here, $R_H\left(\hat{I}_h^H u_h\right)$ is the residual computed on the coarse grid based on the injected fine grid solution. For the LB approach, the solution variable u_h or u_H is replaced by distribution functions f_{α_h} or f_{α_H} along discrete lattice directions. The last term on the right hand side of the above equation is multiplied by a factor of 2 in order to account for different scaling of relaxation and diffusivity variations in the coarse and fine grids. This avoids the numerical instabilities incurred due to successively reducing the relaxation time on the coarse levels [67].

The coarse-grid equation is solved in a three-step procedure similar to that of fine grid. This three-step procedure is analogous to an under-relaxed Jacobi iteration. However, the under-relaxation step is modified to incorporate the defect correction. Thus, the coarse grid solution procedure becomes [68]

1. LBM collision with a source term

$$f_{\alpha_H}\left(\mathbf{x}, \mathbf{t}^*\right) = f_{\alpha_H}\left(\mathbf{x}, \mathbf{t}\right) - \frac{1}{\tau}\left[f_{\alpha_H}\left(\mathbf{x}, \mathbf{t}\right) - f_{\alpha_H}^{eq}\left(\mathbf{x}, \mathbf{t}\right)\right] - \Delta t \bar{\omega}_\alpha \mathscr{D} S_H\left(\mathbf{x}, t\right) \tag{6.14}$$

2. Advection

$$f_{\alpha_H}\left(\mathbf{x} + \mathbf{e}_\alpha^H \Delta t, t^{**}\right) = f_{\alpha_H}\left(\mathbf{x}, t^*\right) \tag{6.15}$$

3. Under-relaxation

$$f_{\alpha_H}\left(\mathbf{x}, t + \Delta t\right) = \gamma\left[f_{\alpha_H}\left(\mathbf{x}, t^{**}\right) + D_{\alpha_H}\right] + \left(1 - \gamma\right) f_{\alpha_H}\left(\mathbf{x}, t\right). \tag{6.16}$$

This yields the coarse grid solution via taking the moments as,

$$\phi_H = \frac{1}{1 - \omega_0} \sum_{\alpha=1}^{b} f_{\alpha_H}. \tag{6.17}$$

The first two steps of the coarse grid iteration are identical to the time-integration of the LBM equation on a fine grid and can be implemented by calling the standard LBM functions. The last step rapidly dampens the high-frequency errors and is chosen to maintain the overall locality of the LB approach. Once the coarsest grid equations are

solved, the coarse grid corrections to the fine grids are carried out recursively using the prolongation operator I_H^h:

$$f_{\alpha_h} = f_{\alpha_H} + I_H^h \left(f_{\alpha_H} - \hat{I}_h^H f_{\alpha_h} \right). \tag{6.18}$$

Here, I_H^h is a bilinear interpolation operator [27, 67]. The defect correction, D_H, is zero when the governing equations become linear. Hence, they are set to zero in the implementation for linear problems to avoid redundant computations.

The value of the relaxation time τ and time step δt are kept constant at all grid levels. This is a reasonable approximation as the MG-LB procedure is used to solve a steady-state problem and avoids numerical instability problems on coarser grid levels if the relaxation times are correspondingly scaled-down [67]. On the other hand, by scaling the restriction operator I_h^H by a factor of 2, the algorithm implicitly maintains physical consistency [67]. However, for unsteady problems, these parameters are required to be varied using appropriate numerical treatments in order to represent the flow Reynolds number accurately on all grid levels. The treatment of boundary conditions at each grid level is similar to the common LB formulations and no additional consideration for multigrid implementation is required.

6.2.1 Multigrid MRT-LBM

In the MRT-LBE the distribution function is mapped to the moment space prior to relaxation, allowing for greater flexibility in the choice of relaxation parameters. If we define $\mathbf{M}$ as the matrix mapping the distribution function to its moments and $\mathbf{R}$ as a diagonal matrix containing the relaxation parameter for the k^{th} moment along the k^{th} diagonal, then we can write the MRT-LBE as (refer to equation (2.66))

$$f_i\left(\mathbf{r} + \mathbf{e}_i \Delta t, t + \Delta t\right) - f_i\left(\mathbf{r}, t\right) = -\mathbf{M}^{-1}\mathbf{R}\left\{\mathbf{m}\left(\mathbf{r}, t\right) - \mathbf{m}^{eq}\left(\mathbf{r}, t\right)\right\} \tag{6.19}$$

where $i = 0, \cdots, q$, $\mathbf{m} = \mathbf{Mf}$, $\mathbf{m}^{eq}$ is a vector containing the equilibrium moments, and q is the number of discrete velocities of the system. Focusing only on the steady state solutions to equation (6.19) allows us to ignore time dependence and re-write the equation in an iterative form, moving from the n^{th} iteration to the $n + 1^{th}$ iteration:

$$f_i^{n+1}\left(\mathbf{r} + \mathbf{e}_i \Delta t\right) - f_i^n\left(\mathbf{r}\right) = -\mathbf{M}^{-1}\mathbf{R}\left\{\mathbf{m}^n\left(\mathbf{r}\right) - \mathbf{m}^{eq}\left(\mathbf{r}\right)\right\} \tag{6.20}$$

On the initial iteration we set the distribution function equal to the Maxwell equilibrium moments given by:

$$f_i^{eq} = w_i \rho \left[1 + \frac{3\mathbf{c}\cdot\mathbf{u}}{c_s^2} + \frac{(9\mathbf{c}\cdot\mathbf{u})^2}{2c_s^4} - \frac{3\mathbf{u}^2}{2c_s^2} \right], \quad i = 1, 2, \cdots, q, \tag{6.21}$$

where ρ and u are dictated by the initial conditions of the system, c_s is the speed of sound in the medium and here its value is taken as $\frac{1}{\sqrt{3}}$, q is the number of discrete velocities in the system, and $w_0 = \frac{4}{9}$, $w_{1\cdots4} = \frac{1}{9}$, and $w_{5\cdots8} = \frac{1}{36}$ in two dimensions and $w_0 = \frac{1}{3}$, $w_{1\cdots6} = \frac{1}{18}$, and $w_{7\cdots18} = \frac{1}{36}$ in three dimensions.

On subsequent iterations, the macroscopic variables, ρ and u, are computed from the following moments of the distribution function:

$$\begin{aligned} \rho &= \sum_{i=1}^{q} f_i, \\ \rho\mathbf{u} &= \sum_{i=1}^{q} f_i \mathbf{c}_i, \end{aligned} \tag{6.22}$$

where q is the number of discrete velocities in the LB discretization.

Equation (6.20) is analogous to using the Jacobi iterative method [69] to solve the LBE for a steady flow. Therefore, while integrating multigrid with LBM, Mavriplis [67] introduced a relaxation step after the completion of the collision and streaming steps of the LBM, This iterative method is analogous to the Jacobi under-relaxation scheme for linear systems [69]. On the fine grid, the FAS-MRT-LBM scheme is carried out in the following three-step procedure:

$$\begin{aligned} &\text{Collision:} && f_i^*(\mathbf{r}) = f_i^n(\mathbf{r}) - \mathbf{M}^{-1}\mathbf{R}\left\{\mathbf{m}^n(\mathbf{r}) - \mathbf{m}^{eq}(\mathbf{r})\right\} \\ &\text{Streaming:} && f_i^{**}(\mathbf{r} + \mathbf{e}_i\Delta t) = f_i^*(\mathbf{r}) \\ &\text{Relaxation:} && f_i^{n+1}(\mathbf{r}) = \gamma f_i^{**}(\mathbf{r}) + (1-\gamma) f_i^n(\mathbf{r}), \end{aligned} \tag{6.23}$$

where γ is the relaxation parameter and $0 < \gamma < 1$. In the current work, γ is set as 0.8 [67]. For the steady state LBE on the fine grid the residual equation is given by:

$$\mathbf{r}_h(\mathbf{f}_h) = 0, \tag{6.24}$$

where the residual, $\mathbf{r}_h(\mathbf{f}_h)$ is computed as follows:

$$\mathbf{r}(f_i) = f_i(\mathbf{r}) - f_i(\mathbf{r} - \mathbf{e}_i\Delta t) + \mathbf{M}^{-1}\mathbf{S}\left[\mathbf{m}^n(\mathbf{r} - \mathbf{e}_i\Delta t) - \mathbf{m}^{eq}(\mathbf{r} - \mathbf{e}_i\Delta t)\right]. \tag{6.25}$$

The pointwise injection was used to transfer the distribution function to the coarse grid and full-weighting is used to transfer the residual to the coarse grid.

For the FAS LBM, the coarse grid correction equation is written as:

$$\begin{aligned} \mathbf{r}_H(\mathbf{f}_H) &= \mathbf{D}_H \\ \mathbf{D}_H &= \mathbf{r}_H\left(\hat{I}_h^H \mathbf{f}_h\right) - 2I_h^H \mathbf{r}_h(\mathbf{f}_h) \end{aligned} \tag{6.26}$$

where $\mathbf{r}_H$ is computed using equation (6.25) and $\mathbf{D}_H$ is the defect correction. The factor of 2 in the second term on the right-hand side of the computation of $\mathbf{D}_H$ is used to scale the relaxation parameters on the coarse grids. Equation (6.26) is solved for $\mathbf{f}_H$ in almost the same manner as equation (6.23), with a slight modification to the relaxation step so that $\mathbf{D}_H$ can be added to the distribution function as follows:

$$\text{Relaxation:} \quad f_i^{n+1}(\mathbf{r}) = \gamma\left(f_i^{**}(\mathbf{r}) + \mathbf{D}_H\right) + (1-\gamma) f_i^n(\mathbf{r}). \tag{6.27}$$

6.3 Two-Dimensional Heat Diffusion

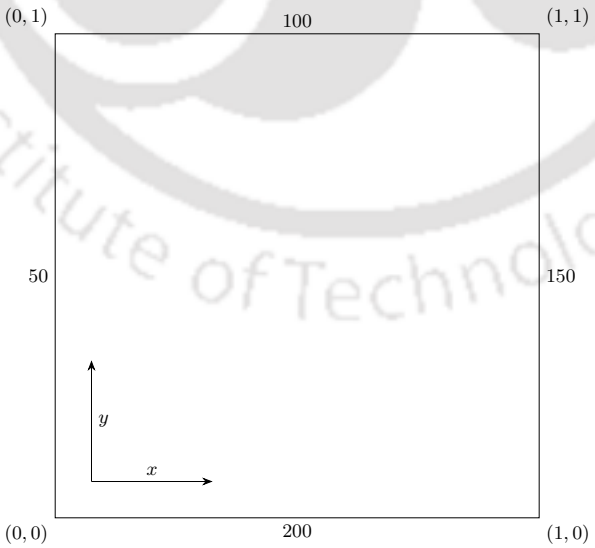


Figure 6.1: *Square Domain with Boundary Condition.*

The solution to Laplace's equation with Dirichlet boundary conditions is chosen as the only test problem [10]. Laplace equation is the governing equation for the two-dimensional

heat conduction problem

$$\frac{d^2\phi}{dx^2} + \frac{d^2\phi}{dy^2} = 0 \tag{6.28}$$

here ϕ is the temperature. The computational effort for the MG-LBM is compared with single-grid LBM. The computational domain with the boundary conditions is shown in figure 6.1 The boundary conditions are:

$\phi = 50$ at $x = 0$

$\phi = 100$ at $y = 1$

$\phi = 150$ at $x = 1$

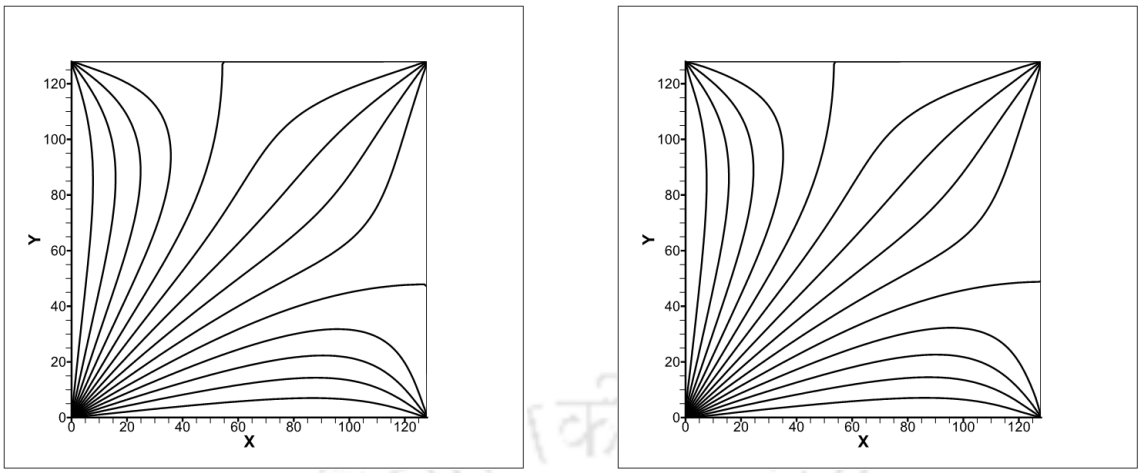
$\phi = 200$ at $y = 0$

A grid of 129×129 and a D2Q5 lattice structure are used in order to solve the problem using both single-grid LBM and MG-LBM. For MG-LBM, grids of 17×17, 33×33, 65×65 and 129×129, i.e., four multigrid levels are used. Three presmooth iterations are given before restricting the solution variable and the residual to the next coarser level. No postsmooth iteration is given.

Table 6.1: Comparison of multigrid LBM and without multigrid LBM for heat diffusion problem

Multigrid Levels	Coarsest Grid	Work Units	Speed-up
1	129×129	17232	-
2	65×65	3640	4.7
3	33×33	247.5	69.6
4	17×17	64.12	268.75

The temperature contours are shown in figure 6.2 [10]. Figure 6.2a is the single-grid LBM solution and the figure 6.2b is the solution with four-level MG-LBM. We can see that MG-LBM results are matching well with the results of single-grid LBM. Table 8.1 shows the performance of the MG-LBM compared to the single-grid LBM in terms of work units. From the table, we can see that with an increase in the multigrid level the work units required to solve the problem decreases. With a four-level multigrid, we have achieved a speed-up of 268.75 compared to the single-grid LBM. This speed up is calculated by dividing the work units required for the single-grid LBM by the work units required for a 4-level MG-LBM.



(a) Single-grid LBM (b) With four-level MG-LBM

Figure 6.2: *Temperature contours with single-grid LBM and 4-level MG-LBM.*

6.4 Summary

In the previous chapter, we accelerated the LBM using GPU parallelization. Here, we are accelerating the LBM with the help of Multigrid. Even though a direct comparison between GPU and multigrid acceleration is not appropriate, however, we would like to point out here that a properly tweaked multigrid has the potential of greatly reducing the effort of LBM compared to a parallel implementation. Furthermore, a GPU parallelized MG-LBM will be able to harness the benefits of both types of convergence-acceleration techniques and will result in a very fast LBM solver.

Chapter 7

Summary and Conclusion

7.1 Conclusions

This chapter describes the major achievements of the thesis and what future work can be taken up based on the experience gained from this work. Whenever an observation is made or an inference drawn, to prepare the background, the work done is also briefly described. Throughout the thesis, qualitative and quantitative aspects of results are systematically analysed with the help of a large number of figures and tables. All the results are produced using computer codes in C/C++ (for parallel codes, CUDA was used) that have been carefully developed by the author from scratch and no help from any third party tool or library has been taken. The results are validated through an extensive comparison with the published results.

This thesis is concerned with study and analysis of various convergence-acceleration strategies that can help CFD practitioners to obtain their simulation results quickly. Convergence-acceleration can be achieved either by choosing a fast algorithm or by parallelizing the solution strategy. The multigrid method lies in the former category and is considered as one of the most powerful convergence-acceleration strategy. Furthermore, this thesis uses CUDA, which is a relatively new method for parallelizing CFD codes. Therefore, a part of the convergence-acceleration efforts embodied in this thesis is also directed to the application of CUDA to LBM-based computations.

At the heart of computation of nonlinear problems, many times there lies a linearized problem. Therefore, this thesis work starts with numerical experiments for the 2D heat conduction problem. The result shows that though successive overrelaxation (SOR) with the optimum relaxation parameter substantially accelerates Gauss-Seidel iterations, linear

multigrid does it even better and its performance relative to SOR improves with grid refinement. Apart from adopting the right strategies in the context of multigrid itself, we observe that the final computational time depends greatly on the selection of the original flow solution schemes and algorithms themselves. This is because some of the basic solution algorithms are such that they form a very efficient combination with multigrid. In this context, the present thesis applies multigrid to work in conjunction with a fluid flow algorithm that is seen to be used somewhat sparingly, which happens to be the streamfunction-velocity ($\psi - V$) method of two-dimensional flow computation. The $\psi - V$ method uses a second-order accurate compact discretization of the biharmonic form of the N-S equations. The presence of the iteration-independent fixed Dirichlet boundary condition for ψ in this formulation was exploited using multigrid and a number of fluid-flow and heat-transfer problems have been solved in the laminar region with remarkable timewise efficiency.

For a properly tweaked multigrid based solution procedure and for the standard lid-driven square-cavity problem the converged solution for Re = 7500 is obtained by multigrid in about 263 seconds in an intel i5-based PC with a speed-up of 69 over the single-grid. The present solution procedure has been used to solve the staggered-cavity flow problem for the first time. Here, the laminar investigations are seen to be carried out at the highest value of Re = 3200 and for this Re, the present solution procedure obtains results in only 41 seconds with a speed-up of about 28.

For the first time multigrid accelerated $\psi - V$ formulation has been used to compute multiple steady solutions in two- and four-sided cavity flows, demonstrating that multigrid has the ability to compute flows in those classes of problems that exhibit nonunique solutions. Even though the convergence is slower near the critical Re at which the flow bifurcates, the present method achieves fast convergence. In all the situations, solutions are obtained in a few seconds on a grid that provides accurate solutions, giving substantial convergence acceleration over a single-grid.

The above formulation has also been extended to natural and mixed convection problems. In the mixed-convection problem, the converged solution is obtained at about 5 seconds or less for all Re-Gr combinations up to Re = 1000 and Gr = 10^6. For example at Re = 1000 and Gr = 10^4, speed-up obtained over single-grid is nearly 99. For the natural convection problem in a square cavity, the solution on a 129×129 grid is obtained in about 10 seconds with a speed-up of about 8 over the single-grid. It is quite evident that from the study that the multigrid-assisted $\psi - V$ formulation performs very well in a variety of

flow situations that include pure fluid flow, natural and mixed convection and fluid-flow problems giving multiple stable solutions.

A variety of fluid flow and thermal problems have been solved using the lattice Boltzmann method which is a relatively new numerical technique. Different isothermal and thermal flow problems have been solved to validate the developed LBM code by comparing the present results with the published results obtained through conventional numerical techniques. The current work has used both single-relaxation-time (SRT) and multiple-relaxation-time (MRT) lattice Boltzmann models. The comparison of SRT and MRT results indicates that both models provide comparable solutions while dealing with the laminar flow regime specified by Rayleigh number (Ra) of the order of 10^6. However, MRT-LBM is found to be more stable and accurate than SRT-LBM in solving flow with high Ra.

The LBM algorithm is such that it is highly amenable to acceleration through parallelization. In this thesis, we have used GPUs to parallelize the LBM code using CUDA parallel computing platform and programming model. A number of two- and three-dimensional fluid-flow and heat-transfer problems have been parallelized. Different combinations of thread blocks have been tested and it was found that for two- and three-dimensional problems thread blocks of size 16×16 and $8 \times 8 \times 8$ respectively gave the best performance for both fluid-fluid and heat-transfer cases. Once two-dimensional LBM is successfully parallelized using CUDA, extending it to three-dimensional is found to be relatively straightforward.

The acceleration of two-dimensional steady mixed convection heat transfer in a differentially heated square cavity for both aiding and opposing mechanisms has been successfully carried out using CUDA. It has been found that parallelized LB code computes about 8 times faster than the serial one and it signifies that LB code is very efficient in GPU technique. Average Nusselt numbers are tabulated to show the effect of heat transfer keeping Re fixed at 100 and Ri varying between 0.01 and 100. Also, we made another observation that timewise speed-up achieved by CUDA increases with the grid size. Hence, CUDA-based LBM computations are very efficient at a high value of Re that requires a finer grid.

A part of the thesis work is devoted to gain an understanding of how the multigrid-assisted LBM method stands with respect to other multigrid-assisted computations. For a two-dimensional heat diffusion problem, it was found that with a four-level multigrid-LBM we can achieve a speed-up of around 268.75.

Overall, in the present work, a wide variety of fluid flow and heat transfer problems of varying complexities have been studied using multigrid accelerated techniques and the parallel lattice Boltzmann method. Both SRT and MRT versions of LBM are used to compute flow problems involving natural- and mixed convection. Furthermore, the scope of using the multigrid assisted $\psi - V$ algorithm has been extended to flow configuration having nonunique solutions. Sufficient care has been taken to obtain highly accurate solutions. All presented results are grid-independent. The ability of multigrid to efficiently compute the wide variety of flows included in this thesis demonstrates not only the utility of the algorithm but also its robustness.

Thus it can be concluded that the goal of the work outlined at the beginning of the thesis has largely been met. The volume of work done and the elements of novelty therein both suggest that the work has been brought to a logical conclusion.

7.1.1 A Few Limitations and Advantages of the Current Methods

In the following section we have listed out a few of the advantages of the current studies over traditional NS solvers:

1. The streamfunction-velocity method uses a second-order accurate compact discretization of the biharmonic form of the Navier-Stokes equations. It may be noted that the possibility of application of Dirichlet boundary condition for just one variable, i.e. streamfunction, plays an important role in its inherent accuracy and multigrid was able to exploit the iteration-independent fixed Dirichlet boundary condition with remarkable ease and accuracy.

2. The LBM algorithm is known to be as embarrassingly parallelizable which makes it easy and convenient to parallelize using CUDA. Extending the LBM algorithm to three-dimensional case is also relatively straightforward and can be easily taken care of in GPU.

However, the study also suffers from certain limitations of its uasge and they are listed as follows:

1. The multigrid assisted streamfunction-velocity method in its current form can not be directly extended to three-dimensional problems. Pure three dimensional stream functions exist physically but at present there is no known way to represent then mathematically. Yih [132] in 1957 suggested using two stream functions to represent

the three dimensional flow. The only exception is a stream function for three dimensional flow exists but only for axisymmetric flow i.e the flow properties remains constant in one of the direction (say z axis).

2. The collision term and the equilibrium distribution function of the lattice Boltzmann method (LBM) which has been considered in this thesis work, pose a limitation of using it directly for high speed flows like supersonic shock-dominated flows. In order to compute high speed flows like supersonic shock-dominated flows, there are specialized lattice Boltzmann methods available, e.g., HRRP-LBM (Hybrid Regularized Recursive Pressure based LBM). In these specialized LB methods, the collision operator and the equilibrium distribution functions are evaluated differently to simulate high Mach number flows.

3. The parallel LBM-CUDA used in this thesis has the limitation of the problem size. Since, we are solving in one node/PC, we can only solve a problem whose memory requirement falls within the hardware memory limit (4 GB for GTX 970 and 12GB for Tesla K40). For larger problem, we will need to implement CUDA-MPI which will be able to use multiple nodes of the HPC cluster.

7.2 Scope for future work

This dissertation has been an attempt to develop and expand the field of application of different convergence-acceleration strategies. However, in what follows, we describe some possible scopes for future work as the present study opens up some interesting possibilities for further investigations

1. The present multigrid assisted $\psi - V$ code is based on the uniform grid. However, the extension of the current model to the non-uniform grids is beneficial in implementing this model more efficiently for flows with complex geometry such as flow through a solid circular/elliptic cylinder while incorporating curved boundary treatment.

2. The current parallel CUDA-LBM code is restricted to a single node machine. Therefore, MPI (message passing interface) can be used to scale up the parallel code's potential to use multiple nodes to be able to solve large and complex 3D flow problems.

3. The MG-LBM code has the potential to be accelerated using the CUDA framework. This will result in an even more accelerated solution.

4. Since the problems presently considered in the thesis are benchmark problems whose results are well known, different practical and complex problems can be solved to validate the robustness of the study.

5. Effect of grid size on the performance of the algorithms can be investigated.

6. The current study did not examine the 3D effect in details and mainly concentrated on how to make the GPU-based computations work for the flow configuration and see its effectiveness. This study can be extended to understand the 3D effects, secondary flows and the details about the z-variations in a cubical cavity.

Appendix A

Derivation of the Streamfunction-Velocity Equations

A.1 Second Order Equations (Stephenson Paper)

The number of terms in u is thirteen as listed below:

$$u = a_{0,0} + a_{1,0}x + a_{0,1}y + a_{2,0}x^2 + a_{1,1}xy + a_{0,2}y^2 + a_{3,0}x^3 + a_{2,1}x^2y + a_{1,2}xy^2 + a_{0,3}y^3 + a_{4,0}x^4 + a_{2,2}x^2y^2 + a_{0,4}y^4 \tag{A.1}$$

The interpolation conditions give:

$$\begin{aligned}
u_1 &= a_{0,0} + a_{1,0}h + a_{2,0}h^2 + a_{3,0}h^3 + a_{4,0}h^4 \\
u_2 &= a_{0,0} + a_{0,1}h + a_{0,2}h^2 + a_{0,3}h^3 + a_{0,4}h^4 \\
u_3 &= a_{0,0} - a_{1,0}h + a_{2,0}h^2 - a_{3,0}h^3 + a_{4,0}h^4 \\
u_4 &= a_{0,0} - a_{0,1}h + a_{0,2}h^2 - a_{0,3}h^3 + a_{0,4}h^4 \\
u_5 &= a_{0,0} + a_{1,0}h + a_{0,1}h + a_{2,0}h^2 + a_{1,1}h^2 + a_{0,2}h^2 + a_{3,0}h^3 \\
&\quad + a_{2,1}h^3 + a_{1,2}h^3 + a_{0,3}h^3 + a_{4,0}h^4 + a_{2,2}h^4 + a_{0,4}h^4 \\
u_6 &= a_{0,0} - a_{1,0}h + a_{0,1}h + a_{2,0}h^2 - a_{1,1}h^2 + a_{0,2}h^2 - a_{3,0}h^3 \\
&\quad + a_{2,1}h^3 - a_{1,2}h^3 + a_{0,3}h^3 + a_{4,0}h^4 + a_{2,2}h^4 + a_{0,4}h^4 \\
u_7 &= a_{0,0} - a_{1,0}h - a_{0,1}h + a_{2,0}h^2 + a_{1,1}h^2 + a_{0,2}h^2 - a_{3,0}h^3 \\
&\quad - a_{2,1}h^3 - a_{1,2}h^3 - a_{0,3}h^3 + a_{4,0}h^4 + a_{2,2}h^4 + a_{0,4}h^4 \\
u_8 &= a_{0,0} + a_{1,0}h - a_{0,1}h + a_{2,0}h^2 - a_{1,1}h^2 + a_{0,2}h^2 + a_{3,0}h^3
\end{aligned}$$

$$
\begin{aligned}
&- a_{2,1}h^3 + a_{1,2}h^3 - a_{0,3}h^3 + a_{4,0}h^4 + a_{2,2}h^4 + a_{0,4}h^4 \\
u_{x1} &= a_{1,0} + 2a_{2,0}h + 3a_{3,0}h^2 + 4a_{4,0}h^3 \\
u_{x3} &= a_{1,0} - 2a_{2,0}h + 3a_{3,0}h^2 - 4a_{4,0}h^3 \\
u_{y2} &= a_{0,1} + 2a_{0,2}h + 3a_{0,3}h^2 + 4a_{0,4}h^3 \\
u_{y4} &= a_{0,1} - 2a_{0,2}h + 3a_{0,3}h^2 - 4a_{0,4}h^3 \qquad \text{(A.2)}
\end{aligned}
$$

For a square grid element, we are able to make use of the following symmetric expression:

$$
\begin{aligned}
\Diamond u_0 &= u_1 + u_2 + u_3 + u_4 \\
&= 4a_{0,0} + 2(a_{2,0} + a_{0,2})h^2 + 2(a_{4,0} + a_{0,4})h^4 \qquad \text{(A.3)} \\
\Box u_0 &= u_5 + u_6 + u_7 + u_8 \\
&= 4a_{0,0} + 4(a_{2,0} + a_{0,2})h^2 + 4(a_{4,0} + a_{2,2} + a_{0,4})h^4 \qquad \text{(A.4)} \\
h\Diamond u_0^{'} &= u_{x1} - u_{x3} + u_{y2} - u_{y4} \\
&= 4(a_{2,0} + a_{0,2})h^2 + 8(a_{4,0} + a_{0,4})h^4 \qquad \text{(A.5)}
\end{aligned}
$$

Equations (A.1), (A.3), (A.4) and (A.5) can be considered as four linear equations in the four unknowns $a_{0,0}$, $(a_{2,0}+a_{0,2})h^2$, $a_{2,2}h^4$ and $(a_{4,0}+a_{0,4})h^4$. Elimination gives us $a_{0,0}$ and hence a finite difference scheme for u_0

$$
u = a_{0,0} = \frac{2}{7}\Diamond u_0 - \frac{1}{28}\Box u_0 - \frac{3h}{28}\Diamond u_0^{'} + \frac{1}{56}f_{0,0}h^4 \qquad \text{(A.6)}
$$

In a similar fashion, from

$$
u_1 - u_3 = 2a_{1,0}h + 2a_{3,0}h^3
$$

and

$$
h(u_{x1} + u_{x3}) = 2a_{1,0}h + 6a_{3,0}h^3
$$

we derive an expression for $a_{1,0}h$, which gives a difference scheme for hu_{x0},

$$
hu_{x0} = \frac{3}{4}(u_1 - u_3) - \frac{h}{4}(u_{x1} + u_{x3}). \qquad \text{(A.7)}
$$

Similarly, we get

$$hu_{y0} = \frac{3}{4}(u_2 - u_4) - \frac{h}{4}(u_{y2} + u_{y4}). \tag{A.8}$$

A.2 Streamfunction-Velocity Formulation

The governing equations for the streamfunction-Vorticity formulation is

$$\frac{\partial^2\psi}{\partial x^2} + \frac{\partial^2\psi}{\partial y^2} = -\omega \tag{A.9}$$

$$u\frac{\partial\omega}{\partial x} + v\frac{\partial\omega}{\partial y} = \frac{1}{Re}\left(\frac{\partial^2\omega}{\partial x^2} + \frac{\partial^2\omega}{\partial y^2}\right) \tag{A.10}$$

Now putting the (A.9) into (A.10) and after simplification we get

$$\frac{\partial^4\psi}{\partial x^4} + 2\frac{\partial^4\psi}{\partial x^2\partial y^2} + \frac{\partial^4\psi}{\partial y^4} = Re\left[u\left(\frac{\partial^3\psi}{\partial x^3} + \frac{\partial^3\psi}{\partial x\partial y^2}\right) + v\left(\frac{\partial^3\psi}{\partial x^2\partial y} + \frac{\partial^3\psi}{\partial y^3}\right)\right] \tag{A.11}$$

Now, lets assume A and B be the first and the second term of the RHS of the equation (A.11). The term A can be discretized as

$$\begin{aligned}
A &= \left[u\left(\frac{\partial^3\psi}{\partial x^3} + \frac{\partial^3\psi}{\partial x\partial y^2}\right)\right]_{i,j} \\
&= \left[u\left(\frac{\partial^2}{\partial x^2}\left(\frac{\partial\psi}{\partial x}\right) + \frac{\partial^2}{\partial y^2}\left(\frac{\partial\psi}{\partial x}\right)\right)\right]_{i,j} \\
&= -\left[u\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right)\right]_{i,j} \\
&= -u_{i,j}\left(\frac{v_{i-1,j} - 2v_{i,j} + v_{i+1,j}}{h^2} + \frac{v_{i,j-1} - 2v_{i,j} + v_{i,j+1}}{h^2})\right) + O\left(h^2\right) \\
&= -\frac{u_{i,i}}{h^2}\left(v_{i-1,j} + v_{i+1,j} + v_{i,j-1} + v_{i,j+1} - 4v_{i,j})\right) + O\left(h^2\right)
\end{aligned} \tag{A.12}$$

Similarly, the term B can be discretized as

$$\begin{aligned}
B &= \left[v\left(\frac{\partial^3\psi}{\partial x^2\partial y} + \frac{\partial^3\psi}{\partial y^3}\right)\right]_{i,j} \\
&= \left[v\left(\frac{\partial^2}{\partial x^2}\left(\frac{\partial\psi}{\partial y}\right) + \frac{\partial^2}{\partial y^2}\left(\frac{\partial\psi}{\partial y}\right)\right)\right]_{i,j} \\
&= \left[v\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)\right]_{i,j} \\
&= v_{i,j}\left(\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2} + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h^2})\right) + O\left(h^2\right) \\
&= \frac{v_{i,j}}{h^2}\left(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j})\right) + O\left(h^2\right)
\end{aligned} \tag{A.13}$$

Before moving to discretizing the equation (A.11), we can simplify the equation (A.6) as

$$28u_0 = 8\Diamond u_0 - \Box u_0 - 3h\Diamond u_0^{'} + 0.5f_{0,0}h^4$$

The above equation in discretized form in $\{i, j\}$ yields

$$\begin{aligned} 28u_{i,j} = {} & 8\left(u_{i+1,j} + u_{i,j+1} + u_{i-1,j} + u_{i,j-1}\right) \\ & - \left(u_{i+1,j+1} + u_{i-1,j+1} + u_{i-1,j-1} + u_{i+1,j-1}\right) \\ & - 3h\left(u_{x_{i+1,j}} - u_{x_{i-1,j}} + u_{y_{i,j+1}} - u_{y_{i,j-1}}\right) + \frac{h^4}{2} f_{i,j} \end{aligned}$$

or,

$$\begin{aligned} & 28u_{i,j} - 8\left(u_{i+1,j} + u_{i,j+1} + u_{i-1,j} + u_{i,j-1}\right) \\ & + \left(u_{i+1,j+1} + u_{i-1,j+1} + u_{i-1,j-1} + u_{i+1,j-1}\right) \\ & + 3h\left(u_{x_{i+1,j}} - u_{x_{i-1,j}} + u_{y_{i,j+1}} - u_{y_{i,j-1}}\right) = \frac{h^4}{2} f_{i,j} \end{aligned} \tag{A.14}$$

Furthermore, the equations (A.7) and (A.8) can be written as

$$hu_{xi,j} = \frac{3}{4}\left(u_{i+1,j} - u_{i-1,j}\right) - \frac{h}{4}\left(u_{x_{i+1,j}} + u_{x_{i-1,j}}\right) \tag{A.15}$$

$$hu_{y_{i,j}} = \frac{3}{4}\left(u_{i,j+1} - u_{i,j-1}\right) - \frac{h}{4}\left(u_{y_{i,j+1}} + u_{y_{i,j-1}}\right) \tag{A.16}$$

Denoting the RHS term as f, the equation (A.11) can be written as

$$\frac{\partial^4 \psi}{\partial x^4} + 2\frac{\partial^4 \psi}{\partial x^2 \partial y^2} + \frac{\partial^4 \psi}{\partial y^4} = f$$

The above equation resembles the biharmonic equation. Hence, the above equation can be discretized in the same way as given by equation (A.14) and it becomes

$$\begin{aligned} & 28\psi_{i,j} - 8\left(\psi_{i+1,j} + \psi_{i,j+1} + \psi_{i-1,j} + \psi_{i,j-1}\right) \\ & + \left(\psi_{i+1,j+1} + \psi_{i-1,j+1} + \psi_{i-1,j-1} + \psi_{i+1,j-1}\right) \\ & + 3h\left(\psi_{x_{i+1,j}} - \psi_{x_{i-1,j}} + \psi_{y_{i,j+1}} - \psi_{y_{i,j-1}}\right) = \frac{h^4}{2} f_{i,j} \end{aligned}$$

or,

$$\begin{aligned} & 28\psi_{i,j} - 8\left(\psi_{i+1,j} + \psi_{i,j+1} + \psi_{i-1,j} + \psi_{i,j-1}\right) \\ & + \left(\psi_{i+1,j+1} + \psi_{i-1,j+1} + \psi_{i-1,j-1} + \psi_{i+1,j-1}\right) \\ & - 3h\left(u_{i,j-1} - u_{i,j+1} - v_{i-1,j} + v_{i+1,j}\right) = \frac{h^4}{2} f_{i,j} \end{aligned}$$

Now, bringing $f_{i,j}$ which is the RHS term of the above equation to the LHS and replacing it with equations (A.12) and (A.13) and with small simplification we get

$$\begin{aligned} 28\psi_{i,j} &- 8\left(\psi_{i+1,j} + \psi_{i,j+1} + \psi_{i-1,j} + \psi_{i,j-1}\right) \\ &+ \left(\psi_{i+1,j+1} + \psi_{i-1,j+1} + \psi_{i-1,j-1} + \psi_{i+1,j-1}\right) \\ &- 3h\left(u_{i,j-1} - u_{i,j+1} - v_{i-1,j} + v_{i+1,j}\right) \\ &- 0.5Reh^2\left[v_{i,j}\left(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1}\right)\right] \\ &+ 0.5Reh^2\left[u_{i,j}\left(v_{i-1,j} + v_{i+1,j} + v_{i,j-1} + v_{i,j+1}\right)\right] = 0 \end{aligned} \tag{A.17}$$

The velocities are obtained using equations (A.15) and (A.16) as

$$u_{i,j} = \frac{3}{4h}\left(\psi_{i,j+1} - \psi_{i,j-1}\right) - \frac{1}{4}\left(u_{i,j+1} + u_{i,j-1}\right) \tag{A.18}$$

$$v_{i,j} = -\frac{3}{4h}\left(\psi_{i+1,j} - \psi_{i-1,j}\right) - \frac{1}{4}\left(v_{i+1,j} + v_{i-1,j}\right) \tag{A.19}$$

The equations (A.17), (A.19) and (A.19) are the discretized equations for the streamfunction, u-velocity and v-velocity respectively.

A.3 Alternative derivation of the streamfunction velocity formulation

Let's start again with the governing equation in the streamfunciton-velocity form,

$$\frac{\partial^4\psi}{\partial x^4} + 2\frac{\partial^4\psi}{\partial x^2\partial y^2} + \frac{\partial^4\psi}{\partial y^4} = Re\left[u\left(\frac{\partial^3\psi}{\partial x^3} + \frac{\partial^3\psi}{\partial x\partial y^2}\right) + v\left(\frac{\partial^3\psi}{\partial x^2\partial y} + \frac{\partial^3\psi}{\partial y^3}\right)\right] \tag{A.20}$$

Now, using equations (A.12) and (A.13) and after simplication, the right hand side of the equation (A.20) becomes

$$\begin{aligned} &-\frac{\mathrm{Re}\, u_{i,j}}{h^2}\left(v_{i+1,j} + v_{i,j+1} + v_{i-1,j} + v_{i,j-1}\right)) \\ &+ \frac{\mathrm{Re}\, v_{i,j}}{h^2}\left(u_{i+1,j} + u_{i,j+1} + u_{i-1,j} + u_{i,j-1}\right)) + O\left(h^2\right) \end{aligned} \tag{A.21}$$

The second order discretization of the second term of the left hand side (LHS) of the equation (A.20) can be carried out as outlined below

$$\frac{\partial^4\psi}{\partial x^2\partial y^2} = \frac{\partial^2}{\partial x^2}\left(\frac{\partial^2\psi}{\partial y^2}\right)$$

$$
\begin{aligned}
&= \frac{\partial^2}{\partial x^2}\left(\frac{\psi_{i,j+1} - 2\psi_{i,j} + \psi_{i,j-1}}{h^2}\right) + O\left(h^2\right) \\
&= \frac{1}{h^2}\left[\frac{\partial^2}{\partial x^2}\left(\psi_{i,j+1}\right) - 2\frac{\partial^2}{\partial x^2}\left(\psi_{i,j}\right) + \frac{\partial^2}{\partial x^2}\left(\psi_{i,j-1}\right)\right] + O\left(h^2\right) \\
&= \frac{1}{h^2}\left[\left(\frac{\psi_{i+1,j+1} - 2\psi_{i,j+1} + \psi_{i-1,j+1}}{h^2}\right) - 2\left(\frac{\psi_{i+1,j} - 2\psi_{i,j} + \psi_{i-1,j}}{h^2}\right)\right. \\
&\qquad \left. + \left(\frac{\psi_{i+1,j-1} - 2\psi_{i,j-1} + \psi_{i-1,j-1}}{h^2}\right) + O\left(h^2\right)\right]
\end{aligned}
$$

$$
\begin{aligned}
\implies \frac{\partial^4 \psi}{\partial x^2 \partial y^2} &= \frac{1}{h^4}\left[\left(\psi_{i+1,j+1} + \psi_{i-1,j+1} + \psi_{i-1,j-1} + \psi_{i+1,j-1}\right) + 4\psi_{i,j}\right. \\
&\quad \left. -2\left(\psi_{i+1,j} + \psi_{i,j+1} + \psi_{i-1,j} + \psi_{i,j-1}\right)\right] + O\left(h^2\right) \qquad \text{(A.22)}
\end{aligned}
$$

Now, we need to discretize the first term, $\frac{\partial^4 \psi}{\partial x^4}$, of the LHS of the equation (A.20). From Taylor's series expansion of $\psi_{i+1,j}$ and $\psi_{i-1,j}$ we get

$$
\begin{aligned}
\psi_{i+1,j} &= \psi_{i,j} + h\frac{\partial \psi}{\partial x}\bigg|_{i,j} + \frac{h^2}{2}\frac{\partial^2 \psi}{\partial x^2}\bigg|_{i,j} + \frac{h^3}{6}\frac{\partial^3 \psi}{\partial x^3}\bigg|_{i,j} + \frac{h^4}{24}\frac{\partial^4 \psi}{\partial x^4}\bigg|_{i,j} + \frac{h^5}{120}\frac{\partial^5 \psi}{\partial x^5}\bigg|_{i,j} + O\left(h^6\right), \\
\psi_{i-1,j} &= \psi_{i,j} - h\frac{\partial \psi}{\partial x}\bigg|_{i,j} + \frac{h^2}{2}\frac{\partial^2 \psi}{\partial x^2}\bigg|_{i,j} - \frac{h^3}{6}\frac{\partial^3 \psi}{\partial x^3}\bigg|_{i,j} + \frac{h^4}{24}\frac{\partial^4 \psi}{\partial x^4}\bigg|_{i,j} - \frac{h^5}{120}\frac{\partial^5 \psi}{\partial x^5}\bigg|_{i,j} + O\left(h^6\right).
\end{aligned}
$$

Addition of the above two equation yields

$$
\begin{aligned}
&\implies \psi_{i+1,j} + \psi_{i-1,j} = 2\psi_{i,j} + h^2\frac{\partial^2 \psi}{\partial x^2}\bigg|_{i,j} + \frac{h^4}{12}\frac{\partial^4 \psi}{\partial x^4}\bigg|_{i,j} + O\left(h^6\right) \\
&\implies \frac{h^4}{12}\frac{\partial^4 \psi}{\partial x^4}\bigg|_{i,j} = \psi_{i+1,j} - 2\psi_{i,j} + \psi_{i-1,j} - h^2\frac{\partial^2 \psi}{\partial x^2}\bigg|_{i,j} + O\left(h^6\right) \\
&\implies \frac{\partial^4 \psi}{\partial x^4}\bigg|_{i,j} = \frac{12}{h^4}\left[\psi_{i+1,j} - 2\psi_{i,j} + \psi_{i-1,j} - h^2\frac{\partial^2 \psi}{\partial x^2}\bigg|_{i,j}\right] + O\left(h^2\right) \\
&\implies \frac{\partial^4 \psi}{\partial x^4}\bigg|_{i,j} = \frac{12}{h^4}\left(\psi_{i+1,j} - 2\psi_{i,j} + \psi_{i-1,j}\right) - \frac{12}{h^2}\frac{\partial}{\partial x}\left(\psi_x\right)\bigg|_{i,j} + O\left(h^2\right) \qquad \text{(A.23)}
\end{aligned}
$$

Again, from Taylor's series expansion we get

$$
\begin{aligned}
\left(\psi_x\right)_{i+1,j} &= \left(\psi_x\right)_{i,j} + h\frac{\partial\left(\psi_x\right)}{\partial x}\bigg|_{i,j} + \frac{h^2}{2}\frac{\partial^2\left(\psi_x\right)}{\partial x^2}\bigg|_{i,j} + \frac{h^3}{6}\frac{\partial^3\left(\psi_x\right)}{\partial x^3}\bigg|_{i,j} + O\left(h^4\right) \\
\left(\psi_x\right)_{i-1,j} &= \left(\psi_x\right)_{i,j} - h\frac{\partial\left(\psi_x\right)}{\partial x}\bigg|_{i,j} + \frac{h^2}{2}\frac{\partial^2\left(\psi_x\right)}{\partial x^2}\bigg|_{i,j} - \frac{h^3}{6}\frac{\partial^3\left(\psi_x\right)}{\partial x^3}\bigg|_{i,j} + O\left(h^4\right)
\end{aligned}
$$

Subtracting them we get

$$\begin{aligned}
&\implies (\psi_x)_{i+1,j} - (\psi_x)_{i-1,j} = 2\,h\frac{\partial\,(\psi_x)}{\partial x}\bigg|_{i,j} + \frac{h^3}{3}\frac{\partial^3\,(\psi_x)}{\partial x^3}\bigg|_{i,j} + O\left(h^5\right)\\
&\implies 2\,h\frac{\partial\,(\psi_x)}{\partial x}\bigg|_{i,j} = (\psi_x)_{i+1,j} - (\psi_x)_{i-1,j} - \frac{h^3}{3}\frac{\partial^3\,(\psi_x)}{\partial x^3}\bigg|_{i,j} + O\left(h^5\right)\\
&\implies \frac{\partial}{\partial x}\,(\psi_x)\bigg|_{i,j} = \frac{1}{2h}\left((\psi_x)_{i+1,j} - (\psi_x)_{i-1,j} - \frac{h^3}{3}\frac{\partial^3\,(\psi_x)}{\partial x^3}\bigg|_{i,j}\right) + O\left(h^4\right) \qquad \text{(A.24)}
\end{aligned}$$

Now, using equation (A.24) to replace $\frac{\partial}{\partial x}\,(\psi_x)\big|_{i,j}$ in equation (A.23) as

$$\begin{aligned}
\frac{\partial^4\psi}{\partial x^4}\bigg|_{i,j} =& \frac{12}{h^4}\,(\psi_{i+1,j} - 2\psi_{i,j} + \psi_{i-1,j})\\
&- \frac{12}{h^2}\left[\frac{1}{2h}\left((\psi_x)_{i+1,j} - (\psi_x)_{i-1,j} - \frac{h^3}{3}\frac{\partial^3\,(\psi_x)}{\partial x^3}\bigg|_{i,j}\right) + O\left(h^4\right)\right] + O\left(h^2\right)
\end{aligned}$$

$$\begin{aligned}
\implies \frac{\partial^4\psi}{\partial x^4}\bigg|_{i,j} =& \frac{12}{h^4}\,(\psi_{i+1,j} - 2\psi_{i,j} + \psi_{i-1,j})\\
&- \frac{6}{h^3}\left((\psi_x)_{i+1,j} - (\psi_x)_{i-1,j} - \frac{h^3}{3}\frac{\partial^3\,(\psi_x)}{\partial x^3}\bigg|_{i,j}\right) + O\left(h^2\right)
\end{aligned}$$

$$\begin{aligned}
\implies \frac{\partial^4\psi}{\partial x^4}\bigg|_{i,j} =& \frac{12}{h^4}\,(\psi_{i+1,j} - 2\psi_{i,j} + \psi_{i-1,j})\\
&- \frac{6}{h^3}\left((\psi_x)_{i+1,j} - (\psi_x)_{i-1,j} - \frac{h^3}{3}\frac{\partial^4\psi}{\partial x^4}\bigg|_{i,j}\right) + O\left(h^2\right)
\end{aligned}$$

$$\begin{aligned}
\implies \frac{\partial^4\psi}{\partial x^4}\bigg|_{i,j} =& \frac{12}{h^4}\,(\psi_{i+1,j} - 2\psi_{i,j} + \psi_{i-1,j})\\
&- \frac{6}{h^3}\left((\psi_x)_{i+1,j} - (\psi_x)_{i-1,j}\right) + 2\,\frac{\partial^4\psi}{\partial x^4}\bigg|_{i,j} + O\left(h^2\right)
\end{aligned}$$

$$\implies \frac{\partial^4\psi}{\partial x^4}\bigg|_{i,j} = \frac{6}{h^3}\left((\psi_x)_{i+1,j} - (\psi_x)_{i-1,j}\right) - \frac{12}{h^4}\,(\psi_{i+1,j} - 2\psi_{i,j} + \psi_{i-1,j}) + O\left(h^2\right) \qquad \text{(A.25)}$$

Similarly, for $\left.\frac{\partial^4\psi}{\partial y^4}\right|_{i,j}$ we would get

$$\implies \left.\frac{\partial^4\psi}{\partial y^4}\right|_{i,j} = \frac{6}{h^3}\left((\psi_y)_{i,j+1} - (\psi_y)_{i,j-1}\right) - \frac{12}{h^4}\left(\psi_{i,j+1} - 2\psi_{i,j} + \psi_{i,j-1}\right) + O\left(h^2\right) \tag{A.26}$$

Now, putting equations (A.25), (A.22) and (A.26) in equation (A.20) and writing its RHS term as $f_{i,j}$ we get,

$$\begin{aligned}
&\frac{6}{h^3}\left((\psi_x)_{i+1,j} - (\psi_x)_{i-1,j}\right) - \frac{12}{h^4}\left(\psi_{i+1,j} - 2\psi_{i,j} + \psi_{i-1,j}\right) \\
&+\frac{6}{h^3}\left((\psi_y)_{i,j+1} - (\psi_y)_{i,j-1}\right) - \frac{12}{h^4}\left(\psi_{i,j+1} - 2\psi_{i,j} + \psi_{i,j-1}\right) \\
&+\frac{2}{h^4}\left[\left(\psi_{i+1,j+1} + \psi_{i-1,j+1} + \psi_{i-1,j-1} + \psi_{i+1,j-1}\right) + 4\psi_{i,j}\right. \\
&\left.-2\left(\psi_{i+1,j} + \psi_{i,j+1} + \psi_{i-1,j} + \psi_{i,j-1}\right)\right] = f_{i,j}
\end{aligned}$$

$$\begin{aligned}
\implies &\frac{56}{h^4}\psi_{i,j} - \frac{16}{h^4}\left(\psi_{i+1,j} + \psi_{i,j+1} + \psi_{i-1,j} + \psi_{i,j-1}\right) \\
&+ \frac{2}{h^4}\left[\left(\psi_{i+1,j+1} + \psi_{i-1,j+1} + \psi_{i-1,j-1} + \psi_{i+1,j-1}\right)\right] \\
&+ \frac{6}{h^3}\left((\psi_x)_{i+1,j} - (\psi_x)_{i-1,j} + (\psi_y)_{i,j+1} - (\psi_y)_{i,j-1}\right) = f_{i,j}
\end{aligned}$$

$$\begin{aligned}
\implies &\frac{56}{h^4}\psi_{i,j} - \frac{16}{h^4}\left(\psi_{i+1,j} + \psi_{i,j+1} + \psi_{i-1,j} + \psi_{i,j-1}\right) \\
&+ \frac{2}{h^4}\left[\left(\psi_{i+1,j+1} + \psi_{i-1,j+1} + \psi_{i-1,j-1} + \psi_{i+1,j-1}\right)\right] \\
&+ \frac{6}{h^3}\left((\psi_x)_{i+1,j} - (\psi_x)_{i-1,j} + (\psi_y)_{i,j+1} - (\psi_y)_{i,j-1}\right) \\
&= -\frac{\mathrm{Re}\, u_{i,j}}{h^2}\left(v_{i+1,j} + v_{i,j+1} + v_{i-1,j} + v_{i,j-1}\right)) \\
&+ \frac{\mathrm{Re}\, v_{i,j}}{h^2}\left(u_{i+1,j} + u_{i,j+1} + u_{i-1,j} + u_{i,j-1}\right))
\end{aligned} \tag{A.27}$$

which is the discretized equation of the streamfunction ψ. The discretized velocity equations are (A.18) and (A.19). Together they complete the entire discretization of the streamfunction-velocity formulation.

Bibliography

[1] William L. Briggs, Van Emden. Henson, and Steve F. McCormick. *A Multigrid Tutorial: Second Edition*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2000.

[2] NVIDIA Corporation. NVIDIA CUDA C Programming Guide, August 2019. Version 10.1.

[3] Wen mei Hwu, David Kirk, and Izzat El Hajj. *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 4th edition, 2022.

[4] U. Ghia, K. N. Ghia, and C. T. Shin. High–Re Solutions for Incompressible Flow Using the Navier–Stokes Equations and a Multigrid Method. *Journal of Computational Physics*, 48:387–411, 1982.

[5] R. Iwatsu and J. M. Hyun and K. Kuwahara. Mixed convection in a driven cavity with a stable vertical temperature gradient. *International Journal of Heat and Mass Transfer*, 36(6):1601 – 1608, 1993.

[6] Achi Brandt. Multi-Level Adaptive Solutions to Boundary-Value Problems. *Mathematics of Computation*, 31(138):333–390, 1977.

[7] H. L. Stone. Iterative solution of implicit approximations of multidimensional partial differential equations. *SIAM Journal of Numerical Analysis*, 5:530–558, 1968.

[8] Suhas V. Patankar. *Numerical Heat Transfer and Fluid Flow*. CRC Press, first edition, 1980.

[9] H. Versteeg and W. Malalasekra. *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*. Pearson Education Ltd, second edition, 2007.

[10] Dale Anderson, John C. Tannehill, and Richard H. Pletcher. *Computational Fluid Mechanics and Heat Transfer*. CRC Press, fourth edition, 2020.

[11] O.C. Zienkiewicz, R.L. Taylor, and David Fox. *The Finite Element Method for Solid and Structural Mechanics*. Butterworth-Heinemann, Oxford, seventh edition edition, 2014.

[12] Daan Frenkel and B. Smit. *Understanding Molecular Simulation*. Elsevier, second edition edition, 2001.

[13] Sauro Succi. *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond*. Clarendon Press Oxford, Oxford, 2001.

[14] Dieter A. Wolf-Gladrow. *Lattice Gas Cellular Automata and Lattice Boltzmann Models*, volume 1725 of *Lecture Notes in Mathematics*. Springer, Berlin, Heidelberg, 2000.

[15] Xiaowen Shan and Hudong Chen. Lattice boltzmann model for simulating flows with multiple phases and components. *Phys. Rev. E*, 47:1815–1819, Mar 1993.

[16] Haibo Huang, Michael Sukop, and Xiyun Lu. *Multiphase Lattice Boltzmann Methods: Theory and Application*. Hoboken: Wiley-Blackwell, 2015.

[17] Zhaoli Guo, Chuguang Zheng, and Baochang Shi. Discrete lattice effects on the forcing term in the lattice Boltzmann method. *Physical Review E*, 65:046308, Apr 2002.

[18] Y. Peng, C. Shu, and Y. T. Chew. Simplified thermal lattice Boltzmann model for incompressible thermal flows. *Phys. Rev. E*, 68:026701, Aug 2003.

[19] Paul J. Dellar. Lattice Kinetic Schemes for Magnetohydrodynamics. *Journal of Computational Physics*, 179(1):95–126, 2002.

[20] Cyrus K. Aidun and Jonathan R. Clausen. Lattice-Boltzmann Method for Complex Flows. *Annual Review of Fluid Mechanics*, 42(1):439–472, 2010.

[21] R. P. Fedorenko. A relaxation method for solving elliptic difference equations. *USSR Computational Mathematics and Mathematical Physics*, 1(4):1092–1096, 1962.

[22] R. P. Fedorenko. The speed of convergence of one iterative process. *USSR Computational Mathematics and Mathematical Physics*, 4(3):227–235, 1964.

[23] W. Hackbusch. On the multigrid method applied to difference equation. *Computing*, 20:291–306, 1978.

[24] W. Hackbusch. Convergence of multigrid iterations applied to difference equation. *Mathematics of Computation*, 34:425–440, 1980.

[25] Wolfgang Hackbusch. *Multi-Grid Methods and Applications*, volume 4. Springer-Verlag Berlin Heidelberg, first edition, 1985.

[26] P. Wesseling. *A robust and efficient multigrid method*, volume 960 of *Lecture Notes in Mathematics*. Springer-Verlag, New York, 1981.

[27] Anton Schüller Ulrich Trottenberg, Cornelius W. Oosterlee. *Multigrid*. Academic Press, San Diego, first edition, 2001.

[28] Klaus Stüben and Ulrich Trottenberg. Multigrid methods: Fundamental algorithms, model problem analysis and applications. In W. Hackbusch and U. Trottenberg, editors, *Multigrid Methods*, pages 1–176, Berlin, Heidelberg, 1982. Springer Berlin Heidelberg.

[29] H. Foester and K. Witsch. On efficient multigrid software for elliptic problems on rectangular domain. *Mathematics and Computers in Simulation*, XXIII:293–298, 1981.

[30] L. Fuchs. Multigrid solution of the Navier-Stokes equation on non-uniform grids. In *Multigrid Methods*, pages 83–100, Moffett Field, CA, 1981. Ames research Center, NASA Conference Publication 2202.

[31] T.F. Miller and F.W. Schmidt. Evaluation of a multilevel technique applied to the poisson and navier-stokes equations. *Numerical Heat Transfer*, 13:1–26, 1988.

[32] P. N. Shankar and M. D. Deshpande. Fluid Mechanics in the Driven Cavity. *Annual Review of Fluid Mechanics*, 32(1):93–136, 2000.

[33] D. Santhosh Kumar, Anoop K. Dass, and Anupam Dewan. A Multigrid-Accelerated Three-Dimensional Transient-Flow Code and its Application to a New Test Problem. *Journal of Hydrodynamics*, 22(6):838–846, Dec 2010.

[34] D. Arumuga Perumal and A. K. Dass. Multiplicity of steady solutions in two-dimensional lid-driven cavity flows by Lattice Boltzmann Method. *Computers & Mathematics with Applications*, 61(12):3711 – 3721, 2011.

[35] J. C. Kalita and S. Sen. Unsteady seperation leading to secondary and tertiary vortex dynamics: the sub-α-and sub-β-phenomena. *Journal of Fluid Mechanics*, 730:19–51, 2013.

[36] N. Alleborn, H. Raszillier, and F. Durst. Lid-driven cavity with heat and mass transport. *International Journal of Heat and Mass Transfer*, 42(5):833 – 853, 1999.

[37] H. C. Kuhlmann, M. Wanschura, and H. J. Rath. Flow in two-sided lid-driven cavities: non-uniqueness, instabilities, and cellular structures. *Journal of Fluid Mechanics*, 336:267–299, 1997.

[38] H. C. Kuhlmann, M. Wanschura, and H. J. Rath. Elliptic instability in two-sided lid-driven cavity flow. *European Journal of Mechanics - B/Fluids*, 17(4):561 – 569, 1998.

[39] S. Albensoeder, H.C. Kuhlmann, and H.J. Rath. Multiplicity of Steady Two-Dimensional Flows in Two-Sided Lid-Driven Cavities. *Theoretical and Computational Fluid Dynamics*, 14(4):223–241, Jan 2001.

[40] Ch. Blohm and H. C. Kuhlmann. The two-sided lid-driven cavity: experiments on stationary and time-dependent flows. *Journal of Fluid Mechanics*, 450:67–95, 2002.

[41] S. Albensoeder and H. C. Kuhlmann. Linear stability of rectangular cavity flows driven by anti-parallel motion of two facing walls. *Journal of Fluid Mechanics*, 458:153–180, 2002.

[42] M. Hinatsu and J. H. Ferziger. Numerical computation of unsteady incompressible flow in complex geometry using a composite multigrid technique. *International Journal for Numerical Methods in Fluids*, 13(8):971–997, 1991.

[43] Y. C. Zhou, B. S. V. Patnaik, D. C. Wan, and G. W. Wei. DSC solution for flow in a staggered double lid driven cavity. *International Journal for Numerical Methods in Engineering*, 57(2):211–234, 2003.

[44] P. Nithiarasu and C.-B. Liu. Steady and unsteady incompressible flow in a double driven cavity using the artificial compressibility (AC)-based characteristic-based split (CBS) scheme. *International Journal for Numerical Methods in Engineering*, 63(3):380–397, 2005.

[45] P. M. Tekić, J. B. Rađenović, N. L. Lukić, and S. S. Popović. Lattice Boltzmann simulation of two-sided lid-driven flow in a staggered cavity. *International Journal of Computational Fluid Dynamics*, 24(9):383–390, 2010.

[46] J. C. Kalita and B. B. Gogoi. Global two-dimensional stability of the staggered cavity flow with an HOC approach. *Computers & Mathematics with Applications*, 67(3):569 – 590, 2014.

[47] Win-Jet Luo and Ruey-Jen Yang. Multiple fluid flow and heat transfer solutions in a two-sided lid-driven cavity. *International Journal of Heat and Mass Transfer*, 50:2394–2405, 2007.

[48] E. M. Wahba. Multiplicity of states for two-sided and four-sided lid driven cavity flows. *Computers & Fluids*, 38(2):247 – 253, 2009.

[49] E. Wahba. Numerical simulations of flow bifurcations inside a driven cavity. *CFD Letters*, 3(2):100–110, 2011.

[50] I. Altas, J. Dym, M.M. Gupta, and R.P. Manohar. Multigrid Solution of Automatically Generated High-Order Discretizations for the Biharmonic Equation. *SIAM Journal of Scientific Computing*, 19:1575–1585, 1998.

[51] R. Kupferman. A Central-Difference Scheme for a Pure Stream Function Formulation of Incompressible Viscous Flow. *SIAM Journal on Scientific Computing*, 23(1):1–18, 2001.

[52] V. I. Bubnovich, C. E. Rosas, and N. O. Moraga. A stream function implicit finite difference scheme for 2D incompressible flows of Newtonian fluids. *International Journal for Numerical Methods in Engineering*, 53(9):2163–2184, 2002.

[53] M. H. Kobayashi and J. M. C. Pereira. A computational stream function method for two-dimensional incompressible viscous flows. *International Journal for Numerical Methods in Engineering*, 62(14):1950–1981, 2005.

[54] M. M. Gupta and J. C. Kalita. A new paradigm for solving Navier-Stokes equations: streamfunction-velocity formulation. *Journal of Computational Physics*, 207:52–68, 2005.

[55] M. Ben-Artzi, J. P. Croisille, D. Fishelov, and S. Trachtenberg. A pure-compact scheme for the streamfunction formulation of Navier-Stokes equations. *Journal of Computational Physics*, 205:640–664, 2005.

[56] M. M. Gupta and J. C. Kalita. New Paradigm Continued: Further Computations with Streamfunction-velocity Formulations for Solving Navier-Stokes Equations. *Communications in Applied Analysis*, 10(4):461–490, 2006.

[57] M. Ben-Artzi, J. P. Croisille, and D. Fishelov. Convergence of a Compact Scheme for the Pure Streamfunction Formulation of the Unsteady Navier-Stokes System. *SIAM Journal on Numerical Analysis*, 44(5):1997–2024, 2006.

[58] J. W. Stephenson. Single cell discretizations of order two and four for biharmonic problems. *Journal of Computational Physics*, 55(1):65–80, 1984.

[59] J. C. Kalita and M. M. Gupta. A streamfunction-velocity approach for 2D transient incompressible viscous flows. *International Journal for Numerical Methods in Fluids*, 62(3):237–266, 2010.

[60] Zhen F. Tian and P.X. Yu. An efficient compact difference scheme for solving the streamfunction formulation of the incompressible Navier-Stokes equations. *Journal of Computational Physics*, 230(17):6404 – 6419, 2011.

[61] P.X. Yu and Zhen F. Tian. A compact streamfunction-velocity scheme on nonuniform grids for the 2D steady incompressible Navier-Stokes equations. *Computers & Mathematics with Applications*, 66(7):1192 – 1212, 2013.

[62] S. K. Pandit and H. Karmakar. An Efficient Implicit Compact Streamfunction Velocity Formulation of Two Dimensional Flows. *Journal of Scientific Computing*, 68(2):653–688, Aug 2016.

[63] D. Fishelov. A new fourth-order compact scheme for the Navier-Stokes equations in irregular domains. *Computers & Mathematics with Applications*, 74(1):6 – 25, 2017. 5th European Seminar on Computing ESCO 2016.

[64] P. Kumar and J. C. Kalita. A transformation-free $\psi - v$ formulation of the Navier-Stokes equations on compact nonuniform grids. *Journal of Computational and Applied Mathematics*, 353:292 – 317, 2019.

[65] P. X. Yu and Z. F. Tian. A high-order compact scheme for the pure streamfunction (vector potential) formulation of the 3D steady incompressible Navier-Stokes equations. *Journal of Computational Physics*, 382:65 – 85, 2019.

[66] J. Tölke, M. Krafczyk, and R. Rank. A Multigrid-Solver for the Discrete Boltzmann Equation. *Journal of Statistical Physics*, 107:574–591, 2002.

[67] Dimitri J. Mavriplis. Multigrid solution of the steady-state lattice Boltzmann equation. *Computers & Fluids*, 35(8):793–804, 2006.

[68] Dhiraj V. Patil, Kannan N. Premnath, and Sanjoy Banerjee. Multigrid lattice Boltzmann method for accelerated solution of elliptic equations. *Journal of Computational Physics*, 265:172–194, 2014.

[69] Charles Armstrong and Yan Peng. An MRT Extension to the Multigrid Lattice Boltzmann Method. *Communications in Computational Physics*, 26(4):1178–1195, 2019.

[70] W. Lie, X. Wei, and A. Kaufmann. Implementing lattice Boltzmann computation on graphics hardware. *The Visual Computer*, 19(7):444–456, 2003.

[71] Zhe Fan, Feng Qiu, Arie E. Kaufman, and Suzanne Yoakum-Stover. GPU Cluster for High Performance Computing. *Proceedings of the ACM/IEEE SC2004 Conference*, pages 47–47, 2004.

[72] Jonas Tölke. Implementation of a Lattice Boltzmann kernel using the Compute Unified Device Architecture developed by nVIDIA. *Comput. Visual Sci.*, 13:29–39, 2010.

[73] J. Tölke and M. Krafczyk. TeraFLOP computing on a desktop PC with GPUs for 3D CFD. *International Journal of Computational Fluid Dynamics*, 22(7):443–456, 2008.

[74] S. P. Vanka. 2012 Freeman Scholar Lecture: Computational Fluid Dynamics on Graphics Processing Units. *Journal of Fluids Engineering*, 135(6), 04 2013.

[75] K. Jin, S.P. Vanka, R.K. Agarwal, and B.G. Thomas. GPU accelerated simulations of three-dimensional flow of power-law fluids in a driven cube. *International Journal of Computational Fluid Dynamics*, 31(1):36–56, 2017.

[76] Achi. Brandt and Oren E. Livne. *Multigrid Techniques: 1984 Guide with Applications to Fluid Dynamics, Revised Edition*. Society for Industrial and Applied Mathematics, 2011.

[77] Ulrich Rüde. *Mathematical and Computational Techniques for Multilevel Adaptive Methods*. Society for Industrial and Applied Mathematics, 1993.

[78] Guy R. McNamara and Gianluigi Zanetti. Use of the Boltzmann Equation to Simulate Lattice-Gas Automata. *Phys. Rev. Lett.*, 61:2332–2335, Nov 1988.

[79] P. L. Bhatnagar, E. P. Gross, and M. Krook. A Model for Collision Processes in Gases. I. Small Amplitude Processes in Charged and Neutral One-Component Systems. *Phys. Rev.*, 94:511–525, May 1954.

[80] Sauro Succi. *The Lattice Boltzmann Equation: For Fluid Dynamics and Beyond* . Oxford University Press, Oxford, first edition, 2013.

[81] A. A. Mohamad. *Lattice Boltzmann Method : Fundamentals and Engineering Applications with Computer Codes* . Springer, London, second edition, 2019.

[82] Xiaoyi He and Li-Shi Luo. Theory of the lattice Boltzmann method: From the Boltzmann equation to the lattice Boltzmann equation. *Phys. Rev. E*, 56:6811–6817, Dec 1997.

[83] Zhaoli Guo and Chang Shu. *Lattice Boltzmann Method and Its Applications in Engineering*. WORLD SCIENTIFIC, 2013.

[84] Renwei Mei, Wei Shyy, Dazhi Yu, and Li-Shi Luo. Lattice Boltzmann Method for 3-D Flows with Curved Boundary. *Journal of Computational Physics*, 161(2):680–699, 2000.

[85] Timm Krüger nad Halim Kusumaatmaja, Alexandr Kuzmin, Orest Shardt, Goncalo Silva, and Erlend Magnus Viggen. *The Lattice Boltzmann Method : Principles and Practice*. Graduate Texts in Physics. Springer, Cham, first edition, 2016.

[86] Shuling Hou, Qisu Zou, Shiyi Chen, Gary Doolen, and Allen C. Cogley. Simulation of Cavity Flow by the Lattice Boltzmann Method. *Journal of Computational Physics*, 118(2):329–347, 1995.

[87] Shiyi Chen, Hudong Chen, Daniel Martnez, and William Matthaeus. Lattice Boltzmann model for simulation of magnetohydrodynamics. *Phys. Rev. Lett.*, 67:3776–3779, Dec 1991.

[88] Y. H Qian, D D'Humières, and P Lallemand. Lattice BGK Models for Navier-Stokes Equation. *Europhysics Letters (EPL)*, 17(6):479–484, feb 1992.

[89] D'Humières. Generalized lattice Boltzmann Equations, Rarefied Gas Dynamics: Theory and Simulations. *Progress in Astronautics and Aeronautics*, 159:450–458, 1992.

[90] Pierre Lallemand and Li-Shi Luo. Theory of the lattice Boltzmann method: Dispersion, dissipation, isotropy, Galilean invariance, and stability. *Phys. Rev. E*, 61:6546–6562, Jun 2000.

[91] Shiyi Chen and Gary D. Doolen. Lattice boltzmann method for fluid flows. *Annual Review of Fluid Mechanics*, 30(1):329–364, 1998.

[92] Li-Shi Luo. *Lattice-Gas Automata and Lattice Boltzmann Equations for Two-Dimensional Hydrodynamics*. PhD thesis, Georgia Institute of Technology, Jan 1993.

[93] Xiaowen Shan and Hudong Chen. Simulation of nonideal gases and liquid-gas phase transitions by the lattice Boltzmann equation. *Phys. Rev. E*, 49:2941–2948, Apr 1994.

[94] Y. A. Cengel. *Heat Transfer a Practical Approach*. McGraw Hill, New York, 2003.

[95] Ahmed Mezrhab, Mohammed Amine Moussaoui, Mohamed Jami, Hassan Naji, and M'Hamed Bouzidi. Double MRT thermal lattice Boltzmann method for simulating convective flows. *Physics Letters A*, 374(34):3499–3507, 2010.

[96] Jia Wang, Donghai Wang, Pierre Lallemand, and Li-Shi Luo. Lattice boltzmann simulations of thermal convective flows in two dimensions. *Computers & Mathematics with Applications*, 65(2):262–286, 2013.

[97] Anthony J. C. Ladd. Numerical simulations of particulate suspensions via a discretized Boltzmann equation. Part 1. Theoretical foundation. *Journal of Fluid Mechanics*, 271:285–309, 1994.

[98] Z.L. Guo, C.G. Zheng, and B.C. Shi. Non-Equilibrium Extrapolation Method for Velocity and Pressure Boundary Conditions in the Lattice Boltzmann Method. *Chinese Physics*, 11:366–374, 2002.

[99] Qisu Zou and Xiaoyi He. On pressure and velocity boundary conditions for the lattice boltzmann bgk model. *Physics of Fluids*, 9:1591–1598, 1995.

[100] Renwei Mei, Li-Shi Luo, and Wei Shyy. An Accurate Curved Boundary Treatment in the Lattice Boltzmann Method. *Journal of Computational Physics*, 155(2):307–330, 1999.

[101] D. Yu, R. Mei, L. S. Luo, and W. Shyy. Viscous flow computations with the method of lattice Boltzmann equation. *Progress in Aerospace Sciences*, 39:329–367, 2003.

[102] Zhaoli Guo, Baochang Shi, and Chuguang Zheng. A coupled lattice BGK model for the Boussinesq equations. *International Journal for Numerical Methods in Fluids*, 39(4):325–342, 2002.

[103] Wen-mei Hwu, Kurt Keutzer, and Timothy G. Mattson. "the concurrency challenge". *IEEE Design Test of Computers*, 25(4):312–320, 2008.

[104] Message Passing Interface Forum. MPI-a message passing interface standard version 2.2, September 2009. Version 2.2.

[105] OpenMP Architecture Review Board. OpenMP application program interface, May 2005. Version 2.5.

[106] H. A. van der Vorst. Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems. *SIAM Journal on Scientific and Statistical Computing*, 13(2):631–644, 1992.

[107] C. D. Santiago, C. H. Marchi, and L. F. Souza. Performance of geometric multigrid method for coupled two-dimensional systems in CFD. *Applied Mathematical Modelling*, 39(9):2602 – 2616, 2015.

[108] F. P. Incropera and D. P. DeWitt. *Fundamentals of Heat and Mass Transfer*. John Wiley & Sons, Inc., New York City, New York, 4th edition edition, 1996.

[109] A. Baïri, E. Zarco-Pernia, and J.-M. García de María. A review on natural convection in enclosures for engineering applications. the particular case of the parallelogrammic diode cavity. *Applied Thermal Engineering*, 63(1):304 – 322, 2014.

[110] G. De Vahl Davis. Natural convection of air in a square cavity: A bench mark numerical solution. *International Journal for Numerical Methods in Fluids*, 3(3):249–264, 1983.

[111] R. A. W. M. Henkes and C. J. Hoogendoorn. Scaling of the laminar natural-convection flow in a heated square cavity. *International Journal of Heat and Mass Transfer*, 36(11):2913 – 2925, 1993.

[112] N. C. Markatos and K. A. Pericleous. Laminar and turbulent natural convection in an enclosed cavity. *International Journal of Heat and Mass Transfer*, 27(5):755 – 772, 1984.

[113] A. D'Orazio, M. Corcione, and G. P. Celata. Application to natural convection enclosed flows of a lattice Boltzmann BGK model coupled with a general purpose thermal boundary condition. *International Journal of Thermal Sciences*, 43:575–586, 2004.

[114] H. N. Dixit and V. Babu. Simulation of high Rayleigh number natural convection in a square cavity using the lattice Boltzmann method. *International Journal of Heat and Mass Transfer*, 49(3):727 – 739, 2006.

[115] D. Kashyap and A. K. Dass. Two-phase lattice Boltzmann simulation of natural convection in a Cu-water nanofluid-filled porous cavity: Effects of thermal boundary conditions on heat transfer and entropy generation. *Advanced Powder Technology*, 29(11):2707 – 2724, 2018.

[116] A. K. Prasad and J. R. Koseff. Combined forced and natural convection heat transfer in a deep lid-driven cavity flow. *International Journal of Heat and Fluid Flow*, 17(5):460 – 467, 1996.

[117] H. F. Oztop and I. Dagtekin. Mixed convection in two-sided lid-driven differentially heated square cavity. *International Journal of Heat and Mass Transfer*, 47(8):1761 – 1769, 2004.

[118] D. Kashyap and A. K. Dass. Effect of boundary conditions on heat transfer and entropy generation during two-phase mixed convection hybrid Al2O3-Cu/water nanofluid flow in a cavity. *International Journal of Mechanical Sciences*, 157-158:45 – 59, 2019.

[119] C. Obrecht, F. Kuznik, B. Tourancheau, and J. J. Roux. A new approach to the lattice Boltzmann method for graphics processing units. *Computers & Mathematics with Applications*, 61(12):3628 – 3638, 2011.

[120] J. Myre, S. D.C. Walsh, D. Lilja, and M. O. Saar. Performance analysis of single-phase, multiphase, and multicomponent lattice-Boltzmann fluid flow simulations on GPU clusters. *Concurrency Computation: Practice and Experience*, 23(4):332–350, March 2011.

[121] M. Astorino, J.B. Sagredo, and A. Quarteroni. A modular lattice Boltzmann solver for GPU computing processors. Technical report, Mathematics Institute of Computational Science and Engineering, Ecole Polytechnique Féderale de Lausanne, 2011.

[122] Keijo Mattila, Jari Hyväluoma, Tuomo Rossi, Mats Aspnäs, and Jan Westerholm. An efficient swap algorithm for the lattice Boltzmann method. *Computer Physics Communications*, 176(3):200–210, 2007.

[123] M. Schönherr, K. Kucher, M. Geier, M. Stiebler, S. Freudiger, and M. Krafczyk. Multi-thread implementations of the lattice Boltzmann method on non-uniform grids for CPUs and GPUs. *Computers & Mathematics with Applications*, 61(12):3730–3743, 2011.

[124] Olga Filippova and Dieter Hänel. Grid Refinement for Lattice-BGK Models. *Journal of Computational Physics*, 147(1):219–228, 1998.

[125] M. Geier, A. Greiner, and J.G. Korvink. Bubble functions for the lattice Boltzmann method and their application to grid refinement. *The European Physical Journal Special Topics*, 171(1):173–179, 2009.

[126] J. Tölke and M. Krafczyk. Second order interpolation of the flow field in the lattice boltzmann method. *Computers & Mathematics with Applications*, 58(5):898–902, 2009.

[127] Wang Xian and Aoki Takayuki. Multi-GPU performance of incompressible flow computation by lattice Boltzmann method on GPU cluster. *Parallel Computing*, 37(9):521–535, 2011.

[128] Christian Obrecht, Frédéric Kuznik, Bernard Tourancheau, and Jean-Jacques Roux. Multi-GPU implementation of the lattice Boltzmann method. *Computers & Mathematics with Applications*, 65(2):252–261, 2013.

[129] Hwar C Ku, Richard S Hirsh, and Thomas D Taylor. A pseudospectral method for solution of the three-dimensional incompressible Navier-Stokes equations. *Journal of Computational Physics*, 70(2):439–462, 1987.

[130] P. LeQuéré. Accurate Solutions to the Square Thermally Driven Cavity at High Rayleigh Number. *Comput. Fluids*, 20(1):29–41, Aug 1991.

[131] Zhenhua Chai and Baochang Shi. A novel lattice Boltzmann model for the Poisson equation. *Applied Mathematical Modelling*, 32(10):2050–2058, 2008.

[132] Chia-Shun Yih. Stream functions in three-dimensional flows. *La Houille Blanche*, 43(3):439–450, 1957.