ANALYSIS OF SINGLE PHASE FLUID FLOW AND HEAT TRANSFER IN SLIP FLOW REGIME BY PARALLEL IMPLEMENTATION OF LATTICE BOLTZMANN METHOD ON GPUS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

SITKI BERAT ÇELİK

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
MECHANICAL ENGINEERING

SEPTEMBER 2012

Approval of the thesis:

ANALYSIS OF SINGLE PHASE FLUID FLOW AND HEAT TRANSFER IN SLIP FLOW REGIME BY PARALLEL IMPLEMENTATION OF LATTICE BOLTZMANN METHOD ON GPUS

submitted by **SITKI BERAT ÇELİK** in partial fulfillment of the requirements for the degree of
Master of Science in Mechanical Engineering Department, Middle East Technical University by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Suha Oral
Head of Department, **Mechanical Engineering** _____

Asst. Prof. Dr. Cüneyt Sert
Supervisor, **Mechanical Engineering Department** _____

Asst. Prof. Dr. Barbaros Çetin
Co-supervisor, **Mechanical Engineering** _____

Examining Committee Members:

Asst. Prof. Dr. Cüneyt Sert
Mechanical Engineering, METU _____

Asst. Prof. Dr. Barbaros Çetin
Mechanical Engineering, Bilkent University _____

Asst. Prof. Dr. Nevsan Şengil
Astronautical Engineering, UTAA _____

Assoc. Prof. Dr. Almıla Güvenç Yazıcıoğlu
Mechanical Engineering, METU _____

Assoc. Prof. Dr. İlker Tarı
Mechanical Engineering, METU _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: SITKI BERAT ÇELİK

Signature :

ABSTRACT

ANALYSIS OF SINGLE PHASE FLUID FLOW AND HEAT TRANSFER IN SLIP FLOW REGIME BY PARALLEL IMPLEMENTATION OF LATTICE BOLTZMANN METHOD ON GPUS

Çelik, Sıtkı Berat

M.S., Department of Mechanical Engineering

Supervisor : Asst. Prof. Dr. Cüneyt Sert

Co-Supervisor : Asst. Prof. Dr. Barbaros Çetin

September 2012, 97 pages

In this thesis work fluid flow and heat transfer in two-dimensional microchannels are studied numerically. A computer code based on Lattice Boltzmann Method (LBM) is developed for this purpose. The code is written using MATLAB and Jacket software and has the important feature of being able to run parallel on Graphics Processing Units (GPUs). The code is used to simulate flow and heat transfer inside micro and macro channels. Obtained velocity profiles and Nusselt numbers are compared with the Navier-Stokes based analytical and numerical results available in the literature and good matches are observed. Slip velocity and temperature jump boundary conditions are used for the micro channel simulations with Knudsen number values covering the slip flow regime. Speed of the parallel version of the developed code running on GPUs is compared with that of the serial one running on CPU and for large enough meshes more than 14 times speedup is observed.

Keywords: Lattice Boltzmann Method, GPU computing, Jacket, Microchannel Flows

ÖZ

KAYGAN AKIŞ REJİMİNDEKİ TEK FAZLI AKIŞ VE ISI TRANSFERİNİN LATTICE BOLTZMANN METODU İLE GRAFİK KARTLARI ÜZERİNDE PARALEL ÇALIŞACAK ŞEKİLDE ANALİZİ

Çelik, Sıtkı Berat

Yüksek Lisans, Makina Mühendisliği Bölümü

Tez Yöneticisi : Yard. Doç. Dr. Cüneyt Sert

Ortak Tez Yöneticisi : Yard. Doç. Dr. Barbaros Çetin

Eylül 2012, 97 sayfa

Bu projede iki boyutlu mikrokanallarda, kaygan akış rejiminde tek fazlı akış ve ısı transferi problemi sayısal olarak çalışılmıştır. Bu amaçla Lattice Boltzmann Metodu (LBM) kullanılarak bir kod geliştirilmiştir. MATLAB ve Jacket yazılımları kullanılarak geliştirilen kodun önemli bir özelliği grafik kartları (GPU) üzerinde paralel çalışabilmesidir. Geliştirilen kod ile mikro ve makro kanallarda benzetimler yapılmış, elde edilen hız profilleri ve Nusselt sayıları literatürdeki Navier-Stokes tabanlı analitik ve sayısal sonuçlarla karşılaştırılmış ve uyumlu sonuçlar alınabildiği gösterilmiştir. Kaygan akış rejimini kapsayan Knudsen sayısı aralığı için yapılan mikrokanal benzetimlerinde hız kayması ve sıcaklık atlaması sınır şartları kullanılmıştır. Geliştirilen kodun grafik kartı üzerindeki paralel performansı ana işlemci üzerinde çalışan hali ile kıyaslanmış ve belli bir büyüklükten sonraki ağlarda 14 kattan daha büyük hızlanmalar elde etmenin mümkün olabildiği gösterilmiştir.

Anahtar Kelimeler: Lattice Boltzmann Metodu, GPU hesaplama, Jacket, Mikrokanal akışları

To my Family

ACKNOWLEDGMENTS

I would like to thank my teachers, Dr. Çetin and Dr. Sert, for their support, patience and directions during the past several years it has taken me to graduate. I would like to thank my parents for their unending love and support. I would like to thank Northern Cyprus Campus of METU for their association via BAP-FEN 10 Campus Research Project, to Dr. Volkan Esat, Abdullah Önal and Kaya Yorulmaz for their precious assistance, and my colleagues/friends, especially Erşan Erdoğan and Arif Koray Koska who companioned me during my studies. Last but not least, i thank to ESN METU family for their lovely friendship and support.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

Introduction

The ability of human beings to make tools has differentiated them from other kinds of creatures. In the early ages, this ultimate ability allowed to create tools in the order of magnitude of a human [1]. Houses, temples, knifes, etc. were at most two order of magnitude bigger or smaller than their makers. In long time, we have learnt to build much bigger and smaller tools and buildings. With the help of fast developing technology in manufacturing industry after 13th century, today, it is possible both to build hundreds of meters tall buildings and to build atomic size tools. While dealing with such massive and minute structures, the physics and engineering behind switch from the conventional everyday life approaches to more complex and completely different practices. Micro-Electro-Mechanical-Systems (MEMS) are the tools having dimensions in the size of microns (smaller than one millimetre and larger than one micron) and including electrical/mechanical components. The attractiveness of MEMS depends on their miniaturization, multiplicity and microelectronics [2]. Beside being light weighted, small volumed and power saving [3], it is also possible to produce thousands and millions of samples of a single device easily by lithography-based techniques. However, MEMS get their major power from the integration with microelectronics.

MEMS are used in numerous engineering-medical applications. **(i)** pressure sensors, **(ii)** inertial sensors, **(iii)** fluid regulation and control, **(iv)** optical switching, and **(v)** mass data storage are considered to be the five dominant applications (see Table 1.1). These applications have found roles in transportation, medicine, telecommunication, computers [4] as well as defence industry [4, 5]. Microsystems for radio frequency (RF-MEMS) applications, are not listed in Table 1.1, yet have entered in the commercialization phase in 2003, and have promising application in both communication, space and defence industries [6].

The market size of MEMS reached one billion dollars in 1996 [2] and have billions of dollars market size worldwide today [1].

Table 1.1: **Market Projections for MEMS**, Adapted from [4].

Application	**Market**
Pressure Sensors	
Blood Pressure Transducers	Medical
IUP Sensors	Medical
Angioplasty Pressure Transducers	Medical
Infusion Pressure Sensors	Transportation
Pressure Sensors: Automobile tires	Transportation
Inertial Sensors	
Airbag Accelerometer	Transportation
Suspension Accelerometer	Transportation
Braking Accelerometer	Transportation
IVHS Navigation Gyros	Transportation
Smart Munitions	Military
Pacemakers	Medical
Machine Monitoring	Manufacturing
Motion Control	Numerous
Fluid Regulation Control	
Medical Infusion Pumps	Medical
Industrial valves	Numerous
Fluid Meters	Numerous
Micromechanical Valves	Numerous
Ink/Bubble Printers	Computer/Printer
Optical Switches	
Access Switches	Telecommunication
Floating Switches	Telecommunication
Mass Data Storage	
Rigid Disk Drive	Computers
Optical Disk Drives	Computers
Flash Memory	Computers
Other Applications	
Analytical Instruments	Numerous
Displays	Numerous
Blood Oxygen Sensors	Medical
Other Medical Applications	Medical
Threshold Sensors	Numerous
Temperature Sensors	Transportation

Some of the MEMS devices involve fluid flows in microchannels such as microducts, micropumps, microturbines, microvalves, microcombustors, synthetic jets and lab-on-a-chip devices [1]. The behaviour of flows in such tiny systems are usually alter from the ones encountered in everyday life. Therefore different approaches are needed to model fluid flows in microchannels. The fluid flow modelling is needed to make cheap designs and estimates. Before performing experiments and manufacture the products the simulation of the system can save time and money.

1.1 Fluid Flow Modelling

The physics of fluids flows can be modelled by several different sets of mathematical equations. All those different sets can be successful even though the logic behind them are completely different. The idea behind those equations can be split in two major topics: Continuum models and particle models (see Fig.1.1).

1.1.1 Continuum Models

Continuum assumption states that every single point in a flow field has a finite physical property (temperature, pressure, density etc.) by accepting fluids to be divisible into sub-fluids indefinitely. Hence, there occurs no discontinuity and the assumption is called continuum. Assuming flow fields and time are infinitely divisible enables the of use differential calculus which leads to sets of partial differential equations as governing equations. The discovered non-linear partial differential equations are to conserve mass, energy and momentum. Navier-Stokes, Euler, and Burnett equations are the accomplished ones. Note that the solutions of dependent variables of those equations are continuous functions. The continuum assumption inherently causes the sets of equations to be incomplete. In other words, there are more unknowns than equations. Therefore, one is supposed to employ constitutive relations to relate stress and rate of strain; heat flux and temperature gradient, and in some situations equations of state to relate density and internal energy to pressure and temperature [1]. Also, it is necessary to consider boundary and initial conditions.

In the continuum model the flow field needs to be in thermodynamic equilibrium, by which, it is understood that there is always enough time for particles (molecules/atoms) to adjust them-

selves according to surrounding variations. For instance, when the velocity or temperature of the flow is begun to change, the particles need some time to undergo sufficient number of collisions to reflect the proper physics of the changes and the time should be incomparable (very small) with the macroscopic time scales. In reality, it is impossible to have perfect equilibrium state due to the fact that particles in motion continuously change energy and momentum. It is more acceptable to say quasi-equilibrium. Thermodynamic equilibrium assumption spawns no-slip and no-temperature-jump boundary condition (i.e. the boundaries have the same velocity and temperature with the barrier of the fluid).

The continuum and equilibrium assumptions may break down in some conditions. For instance, shock wave lengths are comparable with average distance of molecules; hence, it is not possible to have thermodynamic equilibrium or to assume continuum. Also, rarefied gas media have large intermolecular distances which is usually seen at the outer layers of the atmosphere. In MEMS, it is common not to have continuum and thermal equilibrium due to the tiny length scales of the fluid channels. These examples set forward the importance of average molecular distance and it is a common practise to define the characteristics of such problems with a non-dimensional parameter called Knudsen number (Kn). If we describe λ (mean-free-path) to be the average distance of molecules migrate between two successive collisions, Knudsen number is the ratio λ/H, where H is the characteristic length of the system, usually the hydraulic diameter for channel flows [7]. Increasing Kn is either due to increase in mean-free-path or due to decrease in characteristic length. Increasing Kn violates the assumptions of the continuum model equations. High Knudsen number flows may be encountered in flows in narrow channels (microchannels) and/or flows at low density. Different approaches are needed to model fluid flow at high Knudsen numbers for which continuum assumption fails [1].

1.1.2 Molecular Models

Molecular models -unlike continuum models- accept flow fields compose of many particles. The particles have their individual mass and velocity and exert Coulombic force to neighbouring particles. Although the physics seems to be simple, the system in interest is composed of billions of particles. (Imagine 10.000 particles in a cube having 65 nm ($65\text{x}10^{-9}$ meter side). Particles undergo numerous collisions in such a short time that is far beyond human percep-

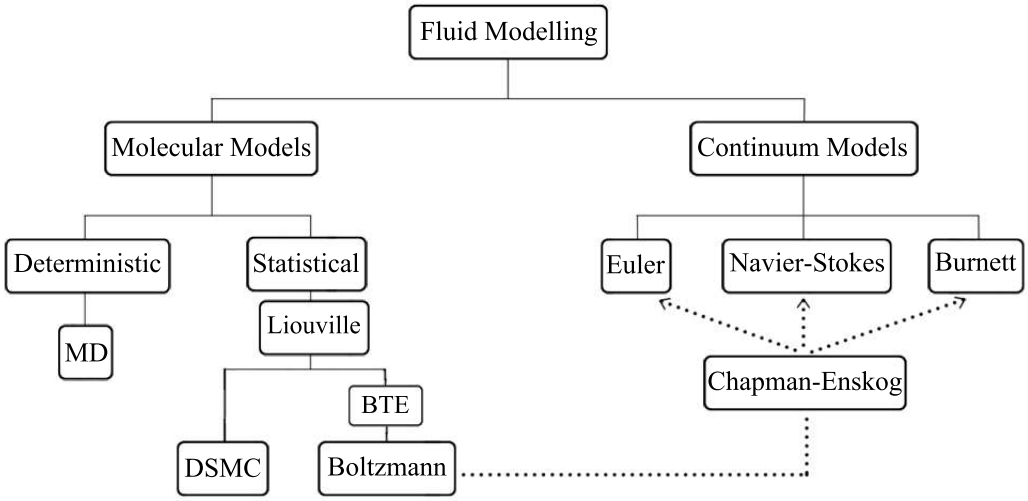

Figure 1.1: **Fluid Modelling**, Adapted from [1].

tion. In this chaotic environment, the feasible properties are not the velocities of the molecules but their collective and average behaviour. In continuum models the average behaviours are named as temperature, pressure, viscosity etc.

Molecular models are examined under two subtopics; deterministic methods and statistical methods. The former one uses basic Newtonian mechanics to evaluate spatial coordinates of particles for every discrete time step. A well studied numerical tool among deterministic methods is called Molecular Dynamics (MD). The pros of the method are very valuable. It usually reflects the physics successfully. It is not need to worry about the macroscopic properties such as temperature, pressure, Newtonian/non-Newtonian behaviours, slip velocity on boundaries, etc. They appear inherently as a result of the chaotic behaviour of the particles. Furthermore, the relation between stress and rate of strain, and heat flux and temperature gradient are also possible to evaluate. In theory, MD has works in every Knudsen number regime. It is valid for Kn numbers from zero to infinity. However, the number of molecules is supposed to be big enough to get average quantities. The example of 10.000 particles indicates the minimum number of air particles in standard conditions to get macroscopic properties with a 1% statistical variation [8]. Differentiating the Newtonian equations of motion and solve them for thousands of particles is easy, yet solving for billions of particles is required most of the time. The computational power, today, however, is not ready to deal with such extreme numbers. 1 second realistic simulation including vibration modes, orientation of polymer molecules and collisions takes hundreds of years of CPU time [9].

Another disadvantage of MD is the potential function definitions. Potential functions like Lennard-Jones, are used to calculate the potential energy of particles and its spatial derivative of them result in the force exerted on the other particles. It is needed to fit a potential empirically or according to numerical experiments. The function is supposed to be fit well to reflect the appropriate physics. It has no unique definition and varies depending on the problem solved. The latter methods, statistical ones, are dedicated to calculate the probability of the particles to be in a spatial volume (between $\mathbf{x}$ and $\mathbf{x} + \partial\mathbf{x}$) and to be in a finite velocity interval (between $\mathbf{v}$ and $\mathbf{v} + \partial\mathbf{v}$) at any time. The mathematical corresponding of the probabilities are expressed in distribution functions using the Sturm-Liouville theorem [10]. The distribution functions are used to find out the macroscopic properties. A successful tool among numerous statistical methods is Direct Simulation Monte Carlo (DSMC). The method uses random and uniform numbers, and hence the name "Monte Carlo" is given to it [11]. It randomly assigns the velocities of particles using the Boltzmann distribution function. Assigning velocities randomly may be seem irrelevant but if the chaotic interactions of billions of molecules are considered, one can confess that the velocities and positions are random in a sense. Not surprisingly, the method results in very accurate simulations for high Knudsen number flows. As discussed previously, small Kn may refer to large characteristic length and therefore, such solutions require an increase of the number of particles. DSMC also suffers from the lack of computational power with the increase of the number of simulated particles.

Finally, the Lattice Boltzmann method (LBM) is also a statistical simulation tool. The origin of LBM is Cellular Gas Automata [12], which was not a successful tool until the idea of using it to solve Boltzmann Transport Equation (BTE). In theory, BTE is capable to cover all flows with Kn ranging from zero to infinity. Fig. 1.2 compares the working regimes of continuum equations and BTE. Flows having Kn smaller than 0.01 are in continuum regime. Euler equations are very successful to imitate the physics in this regime. With the increasing Kn slip-flow regime starts and compressibility and slip-velocity effects come into the picture. Navier-Stokes equations can force the limits of slip-flow regime; but, fluid flows in transition and free molecular regimes can not be simulated using continuum equations. On the other hand, BTE is capable to govern all of the regimes. Moreover, a simplified version of BTE can be adopted for free molecular regime. More information about LBM, which is based on BTE is available in the next chapter.

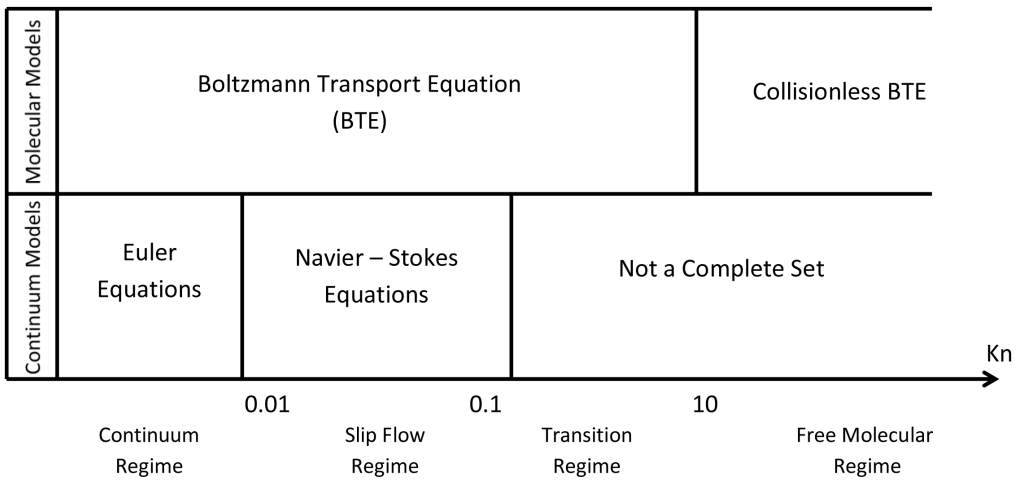

Figure 1.2: Flow Regimes by Knudsen Number [11].

1.2 Scope of the Current Work

The major objective of the current work was to develop a Lattice Boltzmann Method code to simulate slip flow regime which is encountered in microchannel flows. To start with, an LBM code for macrochannels was developed. Different boundary condition implementations and collision schemes from the literature were tested. The code is then enhanced to solve microchannel flows. Poiseuille flow with and without heat transfer in macrochannels and microchannels were solved. Results were well suited to Navier-stokes and analytical solutions in the literature. The second major aim was to develop a code that can utilize the computational power of GPUs. The code was originally running on a single CPU. It was modified to run on a single GPU. A further modification was done to run the code on Multiple-GPU. The codes were written in MATLAB. GPU computations were achieved by the use of the software Jacket.

1.3 Outline of the Dissertation

Chapter 1 is a brief introduction to the Micro-Electro-Mechanical-Systems (MEMS). The importance of MEMS and its application fields are emphasised. The simulation techniques of the flows in MEMS are discussed. An appropriate and fast one is put forward.

Chapter 2 is an introduction to the Lattice Boltzmann Method. The origins of the method from kinetic theory and the concept of statistical mechanics is discussed. Liouville's Equation and Boltzmann Transport Equation are derived and explained. The link between microscopic and macroscopic properties is explained.

Chapter 3 gives theoretical information about the Lattice Boltzmann Method (LBM). The details from the statistical thermodynamics to modern form of LBM is discussed. Besides the derivation, a historical perspective is provided. The concept of lattice units, collision, streaming and macroscopic property calculations are explained. Various boundary conditions are derived and the ones used in the current work are provided in detail.

Chapter 4 includes the results obtained by the developed LBM code. The results are compared by these obtained by classical Navier-Stokes solvers and also by analytical solutions.

Chapter 5 is about the implementation details of LBM on the CPU and the GPU. Starting from the history of computing it compares the CPU and GPU computing. The effects of parallel computing on LBM is reported in this chapter. The structure of the parallel LBM code is also studied here.

Chapter 6 tells about the conclusion of the project and the possible future work to further enhance the developed code.

CHAPTER 2

Kinetic Theory

2.1 Statistical Mechanics

Statistical mechanics deals with enormous number of particles obeying the classical mechanics rules [10]. Therefore, the Newton's second law of motion is applicable to each particle.

$$\frac{\partial^2 \mathbf{x}_i}{\partial t^2} = \hat{\mathbf{F}}_i \tag{2.1}$$

$$\frac{\partial \mathbf{x}_i}{\partial t} = \mathbf{v}_i \tag{2.2}$$

where $\mathbf{x}_i$ is position vector, $\mathbf{v}_i$ is velocity vector and $\hat{\mathbf{F}}_i$ is force vector per mass acting on the i^{th} particle and $i = 1, 2, ..., \mathrm{N}$ with N being the number of particles. $\hat{\mathbf{F}}_i$ is sum of inertial, electric, magnetic etc. forces, and is a known function of position that makes the equations coupled. There are 3 positions and 3 velocity components for each particle, which means the solution can be obtained solving 6N second-order differential equations simultaneously, and also, 6N initial conditions are needed. Note that for reasonable simulations, the order of magnitude of N starts from billions. Solving billions of differential equations with billions of initial conditions simultaneously is far beyond the computational power acquired today. The only concern is not the computational power; but there are also the issue of selecting initial datum, initial positions and velocities. It is nearly impossible to simultaneously determine the positions and velocities initially without affecting the states of particles. Therefore, mathematical approaches such as Maxwellian distribution may be employed to approximate the initial conditions; however, the solution of the equations would result in positions and velocities of each particle, and do not provide practical data such as temperature, pressure, stresses etc. Here is where the statistical mechanics is used. Averaging of momentum and energy of particles, and probability of particles to exist in a small control volume and in a small velocity

range are the main ideas. Cercignani provides detailed answers to questions such as “Why do we need to know the probability in some range of space and velocity?” or ”Can we find a probability for exact position and exact velocity?” [13]. While tossing a coin we know that we will get either heads (H) or tails (T). In terms of probability, the result lay between 0 and 1. The probability of getting heads or tails is $1/2$. If we denote probability with P, the probabilities of getting heads and tails are equal, i.e. $P(H) = P(T) = 1/2$. The summation of all the possibilities is 1 which ensures that one of the possibilities will definitely occur. In statistical mechanics, on the other hand, the possibility is a continuous function rather than discrete sets of values. There exist infinitely many possibilities corresponding to infinitely many states.

If we were able to increase the variables of coin example from 2 to many, the possibilities would decrease from 1/2. Further increase to infinity will create a continuous function of variables and probability of any variable on that continuous function would be zero. However the sum of the probabilities is still 1. This is not surprising and resembles to a finite line segment composed of many points with zero length. Therefore, it is needed to deal with the probability of the values which lie in an interval. If we define the probability density of this curve $P(\mathbf{z})$, the multiplication $P(\mathbf{z})d\mathbf{z}$ represents the probability of the smallest interval. Note that z is the coordinates and $\mathbf{z} = z_1, z_2, ..., z_\infty$. The integral of the multiplication over all variables results unity, which means absolutely one of the possibilities will occur. This is analogous with that integral of mass density over the volume results in total mass.

$$\int_Z P(\mathbf{z})d\mathbf{z} = 1 \tag{2.3}$$

The probability density is used to get the averages. When the probability density is known we can calculate the average of functions.

$$\bar{\varphi}(\mathbf{z}) = \int_Z P(\mathbf{z})\varphi(\mathbf{z})d\mathbf{z} \tag{2.4}$$

where $\bar{\varphi}(\mathbf{z})$ is the average of a function $\varphi(\mathbf{z})$

2.2 Phase Space and the Liouville’s Equation

Now, let’s draw two pictures. First one is a picture of collection of many particles having positions and velocities. Second one is a 6N dimensional coordinate system where dimensions are $\mathbf{x}_k$ and $\mathbf{v}_k$ ($k = 1, 2, ...N$). It is easier to imagine the former one. From now on; however,

the derivation of equations will depend on the second picture which will ease our work to represent the probabilities. The new picture is the so-called phase space. In terms of "certainty" (rather than probability), the state (velocity and position) of particles represents only points in such a system. While talking about "probability", these points have density-like property, in other words they are like spread points in 6N dimensional space. We also can define the variables in the phase space picture with one variable i.e. $\mathbf{p} = \mathbf{p}(\mathbf{x}_k, \mathbf{v}_k)$. Phase space is introduced due to the fact that the Liouville's theorem is derived in phase [14]. First, the number of phase points inside an arbitrary but fixed volume is determined.

$$n_p = \int_V P d\mathbf{p}. \tag{2.5}$$

The rate of change of the number of phase points is

$$\frac{dn_p}{dt} = \int_V \frac{\partial P}{\partial t} d\mathbf{p}. \tag{2.6}$$

Another expression for n_p can be obtained by equating the net rate of change of phase points to the phase points passing through the surface S. Also $\dot{\mathbf{p}}$ expresses 6N dimensional flow vector in phase space.

$$\frac{dn_p}{dt} = -\int_S (\hat{\mathbf{n}} \cdot \dot{\mathbf{p}}) P dS, \tag{2.7}$$

where $\hat{\mathbf{n}}$ denotes the unit normal vector, and due to the fact that it points out we have a minus sign. Using the Gauss theorem the above expression can be rewritten as

$$\frac{dn_p}{dt} = -\int_S \nabla_p \cdot (\dot{\mathbf{p}} P) d\mathbf{p}. \tag{2.8}$$

Equating Eq. 2.5 to Eq. 2.8 results in

$$\frac{\partial P}{\partial t} + \nabla_p \cdot (\dot{\mathbf{p}} P) = 0. \tag{2.9}$$

Expanding the gradient operator and writing the flow vector in open form gives

$$\frac{\partial P}{\partial t} + \sum_{i=1}^{N} \frac{\partial P}{\partial \mathbf{x_i}} \cdot \dot{\mathbf{x}}_\mathbf{i} + \sum_{i=1}^{N} \frac{\partial P}{\partial \mathbf{v_i}} \cdot \dot{\mathbf{v}}_\mathbf{i} = 0. \tag{2.10}$$

To be consistent on the previous notation, the equation can be written as

$$\frac{\partial P}{\partial t} + \sum_{i=1}^{N} \frac{\partial P}{\partial \mathbf{x_i}} \cdot \mathbf{v}_\mathbf{i} + \sum_{i=1}^{N} \frac{\partial P}{\partial \mathbf{v_i}} \cdot \hat{\mathbf{F}}_\mathbf{i} = 0 \tag{2.11}$$

which is known as the Liouville's equation. The solution of this equation for every particle results in the distribution density. However, we have only focused on the positions and the velocities of the particles, and interpret them as mathematical points in space, which move but have no interaction with their surroundings. They are just independently moving particles. The question of "What is the physical system corresponding to the solution of this equation?" arrives at this point. To give an answer to this question, remember the ideal gas assumption. Ideal gas means that the intermolecular potential energy is negligible. The particles move in straight paths. The interaction occurs only during the collisions which is in a confined region called action sphere bounded by molecular distance, σ. σ is very small compared with the mean free path. Another assumption is that the collisions are always monatomic, which means that the molecules have only linear momentum and energy. One can say, therefore, the behaviour of monatomic ideal gasses in thermal equilibrium resembles the solution of Liouville's equation. Those kind of particle systems are found in free-molecular regime (see Fig. 1.2). In real applications, we can hardly find a flow in free-molecular regime. However, it is quiet possible to encounter a flow of a real gas in non-equilibrium conditions.

2.3 The Boltzmann Transport Equation

For the flows in free-molecular regime, the Liouville's equation with appropriate boundary conditions is enough to calculate the density distribution. But using this density distribution, we cannot calculate the temperature or the pressure of the system. The macroscopic behaviours cannot be explained by Liouville's theorem, and the equation cannot be used in the case of non-equilibrium. By non-equilibrium, it should be understood that the particles collide continuously and hence, reach to different states. The non-equilibrium level is related to the collision time of particles. In order to find a solution to such a system, the collisions of particles are supposed to be modelled mathematically. This mathematical model is the Boltzmann equation and it also makes a link between the microscopic and macroscopic properties. Integrating the Liouville's equation over $\mathbf{x}_i$ and ξ_i and manipulating it to model binary collisions result in the following Boltzmann transport equation [10].

$$\frac{\partial P}{\partial t} + \xi \cdot \frac{\partial P}{\partial \mathbf{x}} = N \int [P(\xi^{'})P(\xi_1^{'}) - P(\xi)P(\xi_1)] V r dr d\epsilon d\xi_1 \tag{2.12}$$

where N is the number of molecules, prime indicates the property before the collision,ξ and ξ_1 are the molecular speeds of particles, V is the difference of the speeds of the particles, $|\xi - \xi_1|$

(relative speed), r is the radial coordinate of the "action sphere" bounded by the onset and end of collision, ϵ is the half of the angle between the entering and leaving paths of the moving particles. The maximum distance of particles that can interact with others forms the action sphere and one of the particles is assumed to be in rest during collision. While dealing with the Boltzmann equation, it is a common practice to define a new variable, f

$$f = NmP, \tag{2.13}$$

where m is the mass of a particle and f is the mass density in phase space. Remember that the sum of all the probabilities equals to unity. With the recently defined variable, total mass can be obtained as unity.

$$\int P \, d\mathbf{x} \, d\xi = 1 \tag{2.14}$$

$$\int f \, d\mathbf{x} \, d\xi = Nm = M \tag{2.15}$$

where M is total mass of the N particles . Hence the Boltzmann equation can be written as

$$\frac{\partial f}{\partial t} + \xi \cdot \frac{\partial f}{\partial \mathbf{x}} = \frac{1}{m} \int (f' f_1' - f f_1) V r dr d\epsilon d\xi_1 \tag{2.16}$$

Our interest is in this form of the Boltzmann equation.

2.4 Bridge Between Microscopic and Macroscopic Worlds

The solution of the Boltzmann equation, Eq. 2.16, results in distribution function. It basically stores the data of the probability of a particle (resting in $\mathbf{x}$ and $\mathbf{x} + \mathbf{dx}$; ξ and $\xi + d\xi$) times the total mass of the gas at time t. Integration of f over only the velocity vector, one can obtain the volume independent mass, in other words, density

$$\int f d\xi = \rho(\mathbf{x}, t) \tag{2.17}$$

This equation is also described as the "zeroth moment" of the distribution function.

Table 2.1 summarizes all the moments used. Next, the first moment is going to be calculated. It was mentioned that the probability functions are used to calculate the averages of functions. After successfully calculating the distribution function, one can calculate the average velocity of the particles in the system. The average velocity can also be defined as the bulk velocity. Let's take the first moment to get the average of the velocity.

$$\int \xi f d\xi = \rho \mathbf{v} \tag{2.18}$$

Table 2.1: Moments of Distribution Function in Kinetic Theory

Zeroth Moment	$\int f d\xi = \rho$
First Moment	$\int \xi f d\xi = \rho \mathbf{v}$
Second Moment	$\frac{1}{2}\int \mathbf{c}^2 f d\xi = \frac{1}{2}\int \xi^2 f d\xi - \frac{1}{2}\rho \mathbf{v}^2$

Dividing both sides by ρ results in the macroscopic velocity, **v**. The division is not demonstrated in the formula intentionally to visualize the moments of distribution functions clearly. Before proceeding to the second moment let's introduce a new variable

$$\mathbf{c} = \xi - \mathbf{v} \tag{2.19}$$

The velocity of the particles can be separated into two components. **v** is the bulk velocity (average velocity) and **c** is the random or peculiar velocity. For instance, gas molecules enclosed in a box has zero bulk velocity, **v**, however they are moving in the box with velocity **c**. Now, let's calculate the second moment of the distribution function f

$$\frac{1}{2}\int \mathbf{c}^2 f d\xi = \frac{1}{2}\int \xi^2 f d\xi - \frac{1}{2}\rho \mathbf{v}^2 \tag{2.20}$$

The term on the left hand side is the internal energy per volume, first term on the right hand side is the total energy per volume and the second one is the kinetic energy per volume. After addressing internal energy per unit mass as e, the first term on the left can be written as

$$\frac{1}{2}\int \mathbf{c}^2 f d\xi = \rho e \tag{2.21}$$

Pressure is directly related to the energy per unit mass and is given with the formula

$$p = \frac{2}{3}\rho e \tag{2.22}$$

Macroscopic relation between pressure and temperature is

$$p = \rho R T \tag{2.23}$$

where p is pressure, R is gas constant and T is temperature. From equations 2.22 and 2.23 temperature as a function of microscopic velocity becomes

$$T = \frac{2}{3}\frac{e}{R} \tag{2.24}$$

All the above formulations demonstrate that from the distribution functions, practical macroscopic properties like density, mass, velocity, pressure, temperature, and of course, some more which are not mentioned here can be determined.

CHAPTER 3

Lattice Boltzmann Method

The Boltzmann equation is an integro-differential equation with 7 dimensions which is very complicated and challenging to solve; however, there are some analytical solutions available [10, 14, 15, 8, 16]. For almost every solution, the collision term, which is the most problematic term is simplified to a linear function. If this is not done, one has to solve a differential equation containing a double integral. The approximated linear functions, as expected, do not carry the complicated physics into mathematics, but can result in solutions with desired resolution. Unfortunately, the analytical solutions are available only for very simplified cases. Some examples are steady/unsteady Poiseuille flow, shock wave structure, Rayleigh-Benard convection, Couette flow [14] and traffic flow [17]. The hurdle in analytical solutions lead researchers to seek for numerical solutions. There have been considerable affords to solve the equation numerically in terms of hydro-dynamics [18, 19], electron motion [20, 21], plasma physics [22], etc.

Beside numerical approaches to solve the Boltzmann transport equation, there were also some other techniques to simulate fluid flows. One of them is Lattice Gas Automata (LGA) which was introduced by von Neuman in 1966. However, it had been losing its popularity until parallel computing technology was invented [23]. LGA was constructed on boolean numbers (0 or 1) which are representing the speed of particles travelling on lattices and undergo collisions on lattice cross-roads according to some collision rules to conserve particle number and linear momentum [12]. While the method was in redeveloping age, late in 1980's, the idea to use the Boltzmann equation in Lattice Gas methods became popular. MacNamara and Zanetti were the first researchers who succeed to solve the Boltzmann Transport Equation in Lattice Gas methods numerically [24]. Hence, a new method combining LGA and Boltzmann equation

arose. The recently revolutionized technique was able to overcome the drawbacks of LGAs such as statistical noise and lack of Galilean invariance, and it inherently satisfied the H theorem by the use of BTE. What's more, the massively parallelisable property of LGAs was possible to be carried to the new one [25]. McNamara and Zanetti were loyal to the collision rules common in LGA. The issue with the collision was how to simplify it. Some affords [26, 27] have been tried but an efficient linearisation was obtained by Qian et al. [28]. They proposed the use of relaxation parameter, which is a known procedure in computational fluid dynamics community, for the solution of the Navier-Stokes equations. The relaxation parameter approach, in fact,is exactly the same as the approximation done by Bhatnagar et al. [29], namely BGK, with a properly selected equilibrium distribution function. Further more, they replaced the Fermi-Dirac distribution function with Maxwellian distribution function. All in all, this revised LGA method solving BTE with BGK approximation and using Maxwellian distribution function is now called the Lattice Boltzmann Method.

3.1 From Boltzmann Equation to the Lattice Boltzmann Method

Although the Lattice Boltzmann Equation is primarily derived from Lattice Gas methods, it is proved that, a decade later, LBE can be derived from the Boltzmann equation directly [30, 31]. The derivation below is based on Abe's works [31]. The Boltzmann equation with BGK approximation is

$$\frac{\partial f}{\partial t} + \xi \cdot \frac{\partial f}{\partial \mathbf{x}} = \omega(f^{eq} - f) \tag{3.1}$$

for which the right hand side of Eq. 2.16 is replaced by a linearised approximation [29]. f^{eq} is a Maxwellian equilibrium distribution function. The subtraction term stands for the departure from the equilibrium. Frequency ω, controls the rate of reaching to equilibrium and is related to pressure and kinematic viscosity [8].

$$f^{eq} = \frac{\rho}{2\pi RT} \exp\left(-\frac{1}{2RT}(\xi - \mathbf{v})^2\right) \tag{3.2}$$

$$\omega = \frac{p(\mathbf{x}, t)}{\nu(T)} \tag{3.3}$$

Equilibrium distribution function stands on macroscopic values. Note that the aim is to solve for macroscopic properties from microscopic velocities; but initially, the density ρ, temperature T and velocity $\mathbf{v}$ of the flow are needed to be known. The velocities in Eq. 3.2 are

normalized by a factor $\sqrt{3RT}$ and gets the form

$$f^{eq} = \frac{\rho}{2\pi/3} \exp\left(-\frac{3}{2}(\xi - \mathbf{v})^2\right) \tag{3.4}$$

Similarly, the acoustic sound, $c_s = \sqrt{RT}$ reduces to $1/\sqrt{3}$. f^{eq} is simplified up to second order accuracy by Taylor series expansion [32].

$$f^{eq} = \frac{\rho}{(2\pi/3)^{D/2}} \exp\left(-\frac{3}{2}\xi^2\right)\left[1 + 3(\xi \cdot \mathbf{v}) + \frac{9}{2}(\xi \cdot \mathbf{v})^2 - \frac{3}{2}|\mathbf{v}|^2\right], \tag{3.5}$$

where D is the number of space dimensions, which is in our case. Note that Eq. 3.1 requires velocity, space and time discretisation. The velocity discretisation is obtained by selecting 9 velocities as shown in Fig. 3.1. The system is 2 dimensional and has 9 discrete velocities; hence, it will be addressed as D2Q9 from now on.

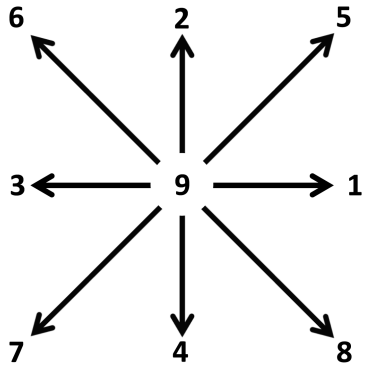

Figure 3.1: Discrete Velocities of D2Q9 LBM formulation

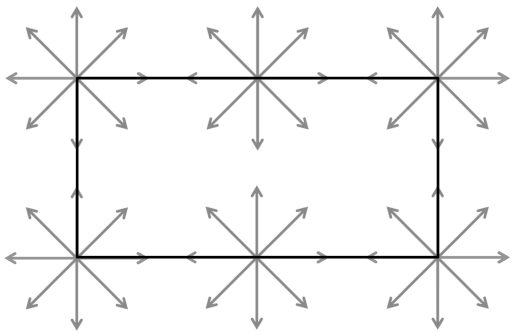

Figure 3.2: Discretization of Space

Fig. 3.2 shows the discretization of space for a channel flow problem in a 2D rectangular domain. The domain in the figure has 6 nodes and each node has 9 discrete velocities.

Eq. 3.1 in discrete velocities can be written in the form

$$\frac{\partial f_i}{\partial t} + \mathbf{e}_i \cdot \frac{\partial f_i}{\partial \mathbf{x}} = \omega(f_i^{eq} - f_i) \tag{3.6}$$

where $i = 1, 2, ..., 9$ and $\mathbf{e}_i$ refer to the discrete velocities shown in Fig. 3.1. ($e_1 = i, e_2 = j, ..., e_8 = i - j, e_9 = 0$). Note that now the BTE is reduced to a system of differential equations. The moment integrals, Eqs. 2.17 and 2.18, after the discretisation of velocity, are approximately calculated using Gaussian-type quadrature on 9 points.

$$\rho(\mathbf{x}, t) = \sum_{i=1}^{9} W_i f_i(\mathbf{x}, t) \tag{3.7}$$

$$\rho\mathbf{v}(\mathbf{x}, t) = \sum_{i=1}^{9} W_i \mathbf{e}_i f_i(\mathbf{x}, t) \tag{3.8}$$

where W_i's are the Gauss-type quadrature weight functions. Let's define another variable

$$\check{f}_i(\mathbf{x}, t) = W_i f_i(\mathbf{x}, t) \tag{3.9}$$

which also satisfies the same equation as $f_i(\mathbf{x}, t)$

$$\frac{\partial \check{f}_i}{\partial t} + \mathbf{e}_i \cdot \frac{\partial \check{f}_i}{\partial \mathbf{x}} = \omega(\check{f}_i^{eq} - \check{f}_i) \tag{3.10}$$

where

$$\check{f}_i^{eq} = w_i \rho[1 + 3(\mathbf{e}_i \cdot \mathbf{v}) + \frac{9}{2}(\mathbf{e}_i \cdot \mathbf{v})^2 - \frac{3}{2}|\mathbf{v}|^2] \tag{3.11}$$

$$w_i = \frac{W_i}{2\pi/3} \exp\left(-\frac{3}{2}\mathbf{e}_i^2\right) \tag{3.12}$$

Equilibrium distribution function, $\check{f}_i^{eq}$ is valid for low speeds. In other words, the equilibrium is satisfied for small Mach Numbers (Ma). He et al. and Abe independently calculated the new weight functions w_i's as [30, 31]

$$w_i = 1/9 \quad for \quad i = 1, 2, 3, 4 \tag{3.13}$$

$$w_i = 1/36 \quad for \quad i = 5, 6, 7, 8 \tag{3.14}$$

$$w_i = 4/9 \quad for \quad i = 9 \tag{3.15}$$

The cartesian mesh with uniform and equal spacing in x and y directions ($\Delta x = \Delta y = 1$) is used. The spatial derivative of Eq. 3.10 can be approximated by a first order upwind difference and the time derivative with a first order explicit difference with time step $\Delta t = 1$. Equating Δx, Δy and Δt to 1 simplifies the equation and this does not create a jeopardy for convergence. The equation yields to

$$\check{f}_i(\mathbf{x} + \Delta t\mathbf{e}_i, t + \Delta t) = \omega \check{f}_i^{eq} + (1 - \omega)\check{f}_i(\mathbf{x}, t) \tag{3.16}$$

where

$$\check{f}_i^{eq} = w_i\rho[1 + 3(\mathbf{e}_i \cdot \mathbf{v}) + \frac{9}{2}(\mathbf{e}_i \cdot \mathbf{v})^2 - \frac{3}{2}|\mathbf{v}|^2] \tag{3.17}$$

$$\omega = \frac{1}{3\nu + 0.5} \tag{3.18}$$

$$\rho(\mathbf{x}, t) = \sum_{i=1}^{9} \check{f}_i(\mathbf{x}, t) \tag{3.19}$$

$$\rho\mathbf{v}(\mathbf{x}, t) = \sum_{i=1}^{9} \mathbf{e}_i \check{f}_i(\mathbf{x}, t) \tag{3.20}$$

Eqs. 3.16 - 3.20 are the Lattice Boltzmann Equations, and they are exactly the same as the ones obtained from LGA. ω is defined only as a function of lattice kinematic viscosity for incompressible flows [33]. Also, it was mathematically proved that obtaining Navier-Stokes's Equations from Boltzmann Equation is possible for small Knudsen numbers [34]. For simplicity, a variable change given below will be applicable for the rest of the work.

$$\check{f}_i(\mathbf{x}, t) \longrightarrow f_i(\mathbf{x}, t) \tag{3.21}$$

3.2 Thermal Lattice Boltzmann Equation

The derivation from BTE to LBE in Sec. 3.1 is built on constant temperature approximation, in other words temperature has no effect on Eqs. 3.16 through 3.20, but only the density and velocity. In the pioneering work to enhance LBE to solve for temperature, Alexander used a D2Q13 model on a hexagonal lattice to simulate viscous, compressible, heat-conducting flows of an ideal monatomic gas [35]. He calculated the second moment, Eq. 2.20, after calculating a modified collision operator. He revised the equilibrium distribution function, f^{eq}, to be correct up to 3^{rd} order in velocity, $\mathbf{v}$. But he obtained good results only for small temperature variations only. Besides, the method was suffering from numerical instability and was valid for a fixed Prandtl number.

There have been several authors studied thermal LBE to avoid the above mentioned drawbacks [36, 37, 38]. Pavlo studied the instability of thermal LBE in detail for both hexagonal and square grids [39]. All those works were also available for fixed Prandtl number due to

the fact that the relaxation times of energy and momentum were the same. Vahala used multiple relaxation time for the collision operator [40], yet a promising result was not obtained until a new scheme was published by He [41]. They defined another distribution function for energy transport, internal energy density distribution function, $g = \frac{(\xi-\mathbf{v})^2}{2} f$. The idea to use two different distribution functions for momentum and energy resulted better stability than the previous methods. The relaxation parameters of momentum and heat transfer were also separated. In the current work, a simplified, hence computationally efficient, version of energy density distribution function with Boussinesq approximation is used [33]. The energy equations are as follows:

$$g_i(\mathbf{x} + \Delta t\mathbf{e}_i, t + \Delta t) = \omega_{th} g_i^{eq} + (1 - \omega_{th}) g_i(\mathbf{x}, t) \tag{3.22}$$

where

$$g_i^{eq} = w_i \phi(\mathbf{x}, t) \left[1 + 3(\mathbf{e}_i \cdot \mathbf{v})\right] \tag{3.23}$$

$$\omega_{th} = \frac{1}{3\alpha + 0.5} \tag{3.24}$$

$$\phi(\mathbf{x}, t) = \sum_{i=1}^{9} g_i(\mathbf{x}, t) \tag{3.25}$$

where the weight functions, w_i, are the same as momentum weight functions, ω_{th} is collision frequency for thermal LBE, α is lattice thermal diffusivity and $\phi(\mathbf{x}, t)$ is lattice temperature.

3.3 Lattice Units

Lattice Boltzmann Methods have their own non-dimensional parameters. The real parameters like temperature, viscosity, length, etc. first should be converted to non-dimensional parameters used in classical fluid mechanics. A further process is applied to those properties to obtain Lattice Units [42] as seen in Fig. 3.3.

Physical System $\longleftrightarrow$ Dimensionless System $\longleftrightarrow$ Lattice System

Figure 3.3: System exchange

An example can make the process clear. Think about a channel in 2-D which is 10 meters long (L) and is 1 meter in height (H). Air enters in uniformly with 2 meters per second speed (U). The kinematic viscosity of air is $2\text{x}10^{-5} m^2/s$ (ν). All the properties up to here belong to

the Physical System in Fig. 3.3. Let's define two non-dimensional parameters to govern the system, aspect ratio and Reynolds number. Aspect ratio is basically the ratio of the length and height of the tunnel. Reynolds Number is the ratio of inertial and viscous forces.

$$r = L/H = \frac{10}{1} = 10 \tag{3.26}$$

$$Re = \frac{UH}{\nu} = \frac{2\text{x}1}{2\text{x}10^{-5}} = 10^5 \tag{3.27}$$

Aspect ratio and Reynolds number are enough to describe such a simple system and they define the Dimensionless System in Fig. 3.3. Now it is time to calculate the Lattice Units. First of all, height of the channel should be determined. The height is given in terms of grid point spacing. Increasing the number of grid points allows to work with higher Reynolds numbers and heals the numerical error; however, brings a computational cost. Usually this value is determined by numerical experimentation after running a few simulations. For now, lets chose a grid of 11 nodes along the channel height. Hence the height of the channel becomes 10. Secondly, the velocity of the flow needs to be determined. The velocity should be smaller than the speed of sound, $1/\sqrt{3}$, defined in the method. LBM usually works fine with a velocity of $v^* = 0.02$. This velocity has no physical dimension but is in the Lattice System. (*) indicates that the property is in Lattice System. So far, two variables are defined, lattice height (H^*) and lattice velocity (v^*). After determining them, one can proceed to determine the other lattice properties such as length of channel and viscosity of fluid. Using aspect ratio and Reynolds number determined previously the Lattice Units are calculated as follows

$$L^* = H^* r = 10\text{x}10 = 100 \tag{3.28}$$

$$\nu^* = \nu^* H^*/Re = \frac{0.02\text{x}10}{10^5} = 2\text{x}10^{-6} \tag{3.29}$$

The Lattice Units indicates that the computational domain is composed of 10x100 = 1000 grid points, and the Eqn. 3.18 is calculated using lattice kinematic viscosity, ν^*. Same procedure is applicable to calculate Lattice thermal diffusivity using Prandtl number (Pr) equality.

$$\alpha^* = \nu^*/Pr \tag{3.30}$$

$$Pr = Pe/Re \tag{3.31}$$

where Pe is Péclet number.

The return to physical system from lattice system is done by the inverse of this procedure. For example, consider a channel flow. If the velocity on a node is calculated as $v^* = 0.05$, the physical velocity can be calculated as

$$Re = Re^* \tag{3.32}$$

$$\frac{UH}{\nu} = \frac{v^* H^*}{\nu^*} \tag{3.33}$$

$$U = \frac{v^* H^*}{\nu^*} \frac{\nu}{H} \tag{3.34}$$

$$U = \frac{0.05\text{x}10}{2\text{x}10^{-6}} \frac{2\text{x}10^{-5}}{1} \tag{3.35}$$

$$U = 5 \ m/s \tag{3.36}$$

where U is the physical velocity.

3.4 Collision and Streaming

LBM can be considered to be composed of 4 different parts; collision, streaming, implementation of boundary conditions and calculation of macroscopic properties. Streaming (updating) process is actually a part of collision calculations. However, it is simpler if handled separately. During the collision calculations, the i^{th} distribution function of any grid node can be calculated as follows

$$f_i(\mathbf{x}, t + \Delta t) = \omega f_i^{eq} + (1 - \omega) f_i(\mathbf{x}, t) \tag{3.37}$$

which actually results in the distribution functions of the next time step. Remember that there is also a spatial discretization. This is performed by properly transferring the values of distribution functions between neighbouring nodes

$$f_i(\mathbf{x} + \Delta\mathbf{x}, t + \Delta t) = f_i(\mathbf{x}, t + \Delta t), \tag{3.38}$$

and this process is addressed as streaming in LBM. The schematic of the process is demonstrated in Fig. 3.4. It shows the pre-streaming and post-streaming of the first distribution function, f_1, on every node of a sample 6 node mesh. The direction of the first distribution function is [1,0], and they are moved in that direction. Every distribution function is going to be moved according to their directions. Note that after a streaming step, there exists some missing distribution functions demonstrated with dashed arrows in Fig. 3.4 and some functions need to leave the domain. The distribution functions leaving the domain are not

important but the missing ones must be recalculated. At this point the boundary conditions take the stage for the missing distribution functions.

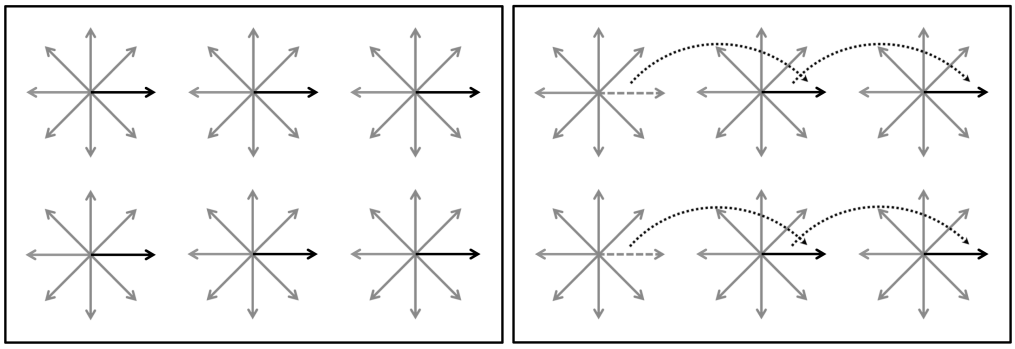

Figure 3.4: Streaming Process of f_1 on a sample grid of 6 points. Left: Before Streaming, Right: After Streaming

3.5 Boundary Conditions

Boundary conditions can be obtained by applying the mass and momentum conservations at the boundaries [43]:

$$\rho = \sum_{i=1}^{9} f_i \tag{3.39}$$

$$\rho \mathbf{v} = \sum_{i=1}^{9} \mathbf{e}_i f_i \tag{3.40}$$

The derivations below are based on Eqns. 3.39 and 3.40 where $\mathbf{v} = [v_x, v_y]$. Fig. 3.5 shows the boundary conditions for the 2D channel flow problem considered in this study. For simplicity the * sign being used to denote lattice units is removed for the rest of the work. In Fig. 3.5 ϕ is the lattice temperature.

3.5.1 Inlet Velocity Boundary Condition

While simulating the inlet boundary condition of a flow in a channel like the one shown in Fig. 3.5, it is assumed that there is no velocity component in y-direction but there is a non zero velocity in x-direction. After the streaming step, some distribution functions remain unknown. On the left boundary the unknown distribution functions are f_1, f_5 and f_6. Due to the fact that, we have some unknown distribution functions, the density, ρ, is an unknown,

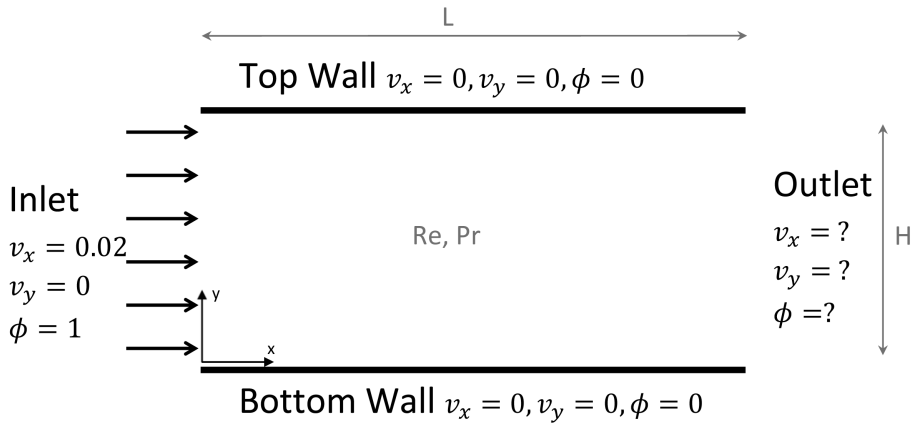

Figure 3.5: Boundary Conditions for a 2D Channel Flow

too. To solve 4 unknowns, at least 4 equations are needed. The first equation we will use is the equality of the non-equilibrium distribution functions, $f^{non-eq} = f - f^{eq}$, which are perpendicular to the wall. Remaining equations are Eqs. 3.39 and 3.40.

$$f_1 - f_1^{eq} = f_3 - f_3^{eq} \tag{3.41}$$

$$\rho = f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8 + f_9 \tag{3.42}$$

$$\rho v_x = f_1 - f_3 + f_5 - f_6 - f_7 + f_8 \tag{3.43}$$

$$\rho v_y = f_2 - f_4 + f_5 + f_6 - f_7 - f_8 = 0 \tag{3.44}$$

Solution of these 4 equation provides the following results for the unknowns at the inlet boundary

$$\rho = \frac{f_2 + f_4 + f_9 + 2(f_3 + f_6 + f_7)}{1 - v_x} \tag{3.45}$$

$$f_1 = f_3 + \frac{2}{3}\rho v_x \tag{3.46}$$

$$f_5 = f_7 + \frac{1}{2}(f_4 - f_2) + \frac{\rho v_x}{6} \tag{3.47}$$

$$f_8 = f_6 - \frac{1}{2}(f_4 - f_2) + \frac{\rho v_x}{6} \tag{3.48}$$

where v_x is the known inlet velocity.

3.5.2 No-slip Boundary Condition

No slip boundary condition can be obtained by equating the x and y velocities to zero for the nodes on necessary boundaries. The non-equilibrium parts of the distribution functions also

need to be used as

$$f_2 - f_2^{eq} = f_4 - f_4^{eq} \tag{3.49}$$

$$\rho = f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8 + f_9 \tag{3.50}$$

$$\rho v_x = f_1 - f_3 + f_5 - f_6 - f_7 + f_8 = 0 \tag{3.51}$$

$$\rho v_y = f_2 - f_4 + f_5 + f_6 - f_7 - f_8 = 0 \tag{3.52}$$

Note that when v_x and v_y velocities are set to zero, the unknown ρ multiplies with zero in two of the equations and ρ becomes unimportant for those equations. Solution of the Eqs. 3.49, 3.51 and 3.52 results in the following relation for the unknown $f's$ of the bottom wall.

$$f_2 = f_4 \tag{3.53}$$

$$f_5 = f_7 - \frac{f_1 - f_3}{2} \tag{3.54}$$

$$f_6 = f_8 + \frac{f_1 - f_3}{2} \tag{3.55}$$

3.5.3 Outlet Boundary Condition

Contrary to the inlet boundary, the velocity at the outlet is unknown. However, we can assume that the velocity is no more varying in the stream-wise direction in a long enough channel. That is equivalent to say that the flow is fully developed. For such cases, Succi suggests two methods [44]. One is to directly copy the distribution functions closest to the exit on the ones at exit. The other one is to calculate the unknowns by extrapolating the ones in the flow domain. It is observed that both schemes add extra numerical error to the method and they will not be used in this work. On the other hand, assuming that the velocity distribution close to the exit does not change in the stream-wise direction, we can safely assign the velocity close to exit to the exit nodes. The simultaneous solution of Eqs. from 3.49 to 3.52 result in the following relation when velocity on y direction is assumed to be zero.

$$\rho = \frac{f_2 + f_4 + f_9 + 2(f_1 + f_5 + f_8)}{1 - v_x} \tag{3.56}$$

$$f_3 = f_1 + \frac{2}{3}\rho v_x \tag{3.57}$$

$$f_6 = f_8 + \frac{1}{2}(f_4 - f_2) - \frac{\rho v_x}{6} \tag{3.58}$$

$$f_7 = f_5 - \frac{1}{2}(f_4 - f_2) - \frac{\rho v_x}{6} \tag{3.59}$$

3.5.4 Temperature Boundary Condition

Temperature boundary condition is implemented using the equality of the non-equilibrium distribution functions [33]. The equation set below is derived for the bottom wall.

$$g_2 = \phi\,(w(2) + w(4)) - g_4 \tag{3.60}$$

$$g_5 = \phi\,(w(5) + w(7)) - g_7 \tag{3.61}$$

$$g_6 = \phi\,(w(6) + w(8)) - g_8 \tag{3.62}$$

The outlet boundary condition for temperature can be implemented using the methods of Succi [44] which are discussed earlier. The extrapolation scheme among them is employed for the current work. End results are tabulated in Appendix B.

3.5.5 Boundary Conditions for Microchannels

3.5.5.1 Slip Boundary Condition

Upon the simultaneous solution of the mass and momentum equality, Tian formulated the Maxwell first-order slip boundary condition without thermal creep [45]. The non-dimensional form of slip boundary condition is given as

$$v_{y=0}^{slip} = v_{x,y=0} - v_{x,w} = \sigma Kn \left(\frac{\partial u}{\partial y}\right)_{y=0} \tag{3.63}$$

$$v_{y=H}^{slip} = v_{x,w} - v_{x,y=H} = \sigma Kn \left(\frac{\partial u}{\partial y}\right)_{y=H} \tag{3.64}$$

where σ is momentum-accommodation coefficient and assumed to be unity to simulate completely diffuse reflection. In most of the engineering applications the momentum-accommodation coefficient for gas-solid interactions is close to unity [46]. $v_{y=0}^{slip}$, $v_{y=H}^{slip}$, $v_{x,y=0}$, $v_{x,y=H}$ and $v_{x,w}$ are the slip velocities at bottom and top walls of the channel, x component of flow velocity at bottom and top walls, and the x component of the corresponding wall velocity, respectively. In this work the walls are stationary and $v_{x,w}$ is set to zero. The derivatives of the velocity appearing in Eqs. 3.63 and 3.64 are calculated using a second-order implicit scheme. The slip

boundary condition for bottom wall is derived as follows

$$\rho_w = f_1 + f_3 + f_9 + 2(f_7 + f_4 + f_8) \tag{3.65}$$

$$f_2 = f_4 \tag{3.66}$$

$$f_5 = \frac{\rho_w(1 + v_x) - (f_2 + f_4 + f_9)}{2} - (f_1 + f_8) \tag{3.67}$$

$$f_6 = \frac{\rho_w(1 - v_x) - (f_2 + f_4 + f_9)}{2} - (f_3 + f_7) \tag{3.68}$$

$$v_x = \lambda \frac{(4v_{x,1} - v_{x,2})}{2 + 3\lambda} \tag{3.69}$$

where $v_{x,1}$ and $v_{x,2}$ are the first two velocity values after the fluid velocity on the wall. λ is KnH. Fig. 3.6 demonstrates the rows of which the velocities belong to.

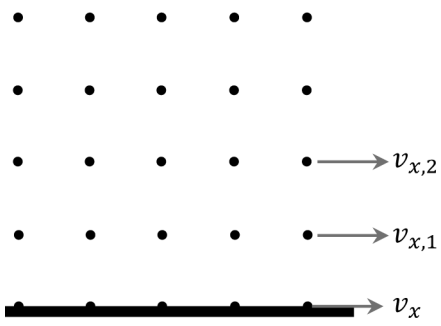

Figure 3.6: Rows of velocities used in slip boundary condition

3.5.5.2 Temperature Jump Boundary Condition

The work of Tian explains how to obtain a temperature jump boundary condition [45], which is similar to the slip velocity boundary condition. Mathematical representation of temperature jump is

$$\phi_{y=0}^{jump} = \phi_{y=0} - \phi_w = \alpha \left(\frac{2\gamma}{\gamma + 1}\right)\left(\frac{Kn}{Pr}\right)\left(\frac{\partial \phi}{\partial y}\right)_{y=0} \tag{3.70}$$

$$\phi_{y=H}^{jump} = \phi_w - \phi_{y=H} = \alpha \left(\frac{2\gamma}{\gamma + 1}\right)\left(\frac{Kn}{Pr}\right)\left(\frac{\partial \phi}{\partial y}\right)_{y=H} \tag{3.71}$$

where α is thermal-accommodation coefficient and is assumed to be one. γ is the specific heat ratio and Pr is Prandtl number. $\phi_{y=0}^{jump}$, $\phi_{y=H}^{jump}$, $\phi_{y=0}$, $\phi_{y=H}$, and ϕ_w are the temperature jump at

bottom and top walls, fluid temperature on the walls and the wall temperature respectively. Similar to velocity derivative the spatial derivative of temperature is also calculated using a second-order implicit scheme. For simplicity a temperature jump coefficient can be defined as

$$C_j = \kappa Kn = Kn\alpha\left(\frac{2\gamma}{(\gamma+1)Pr}\right) \tag{3.72}$$

Hence, the equations of temperature jump for bottom wall are

$$\phi_{y=0} = \frac{\left[C_j(4\phi_1 - \phi_2) + 2\phi_w\right]}{(2+3C_j)} \tag{3.73}$$

$$g_2 = \phi_{y=0}\,(w(2) + w(4)) - g_4 \tag{3.74}$$

$$g_5 = \phi_{y=0}\,(w(5) + w(7)) - g_7 \tag{3.75}$$

$$g_6 = \phi_{y=0}\,(w(6) + w(8)) - g_8 \tag{3.76}$$

Necessary equations for all BCs, including the ones that are not described in this section can be found in Appendix B.

3.6 Calculation of Macroscopic Properties

Density and velocity of the flow field are calculated using the mass and momentum conservations for each node.

$$\rho(\mathbf{x}, t) = \sum_{i=1}^{9} f_i(\mathbf{x}, t) \tag{3.77}$$

$$\mathbf{v}(\mathbf{x}, t) = \frac{\sum_{i=1}^{9} \mathbf{e}_i f_i(\mathbf{x}, t)}{\rho(\mathbf{x}, t)} \tag{3.78}$$

The temperature, on the other hand, is calculated using the thermal distribution function.

$$\phi(\mathbf{x}, t) = \sum_{i=1}^{9} g_i(\mathbf{x}, t) \tag{3.79}$$

CHAPTER 4

Results

Validity of the developed LBM code is tested by using Poiseuille flow in micro and macro scales. Poiseuille flow is, basically, the fluid flow between two parallel plates shown in Fig. 3.5. The velocity profiles, temperature distributions and Nusselt number variations obtained by LBM are compared against analytical solutions when available. There is no analytical solution for the developing regime of the channel; hence, other numerical solutions were employed.

4.1 Analytical Solution of Fully Developed Poiseuille Flow

The results for Poiseulle flow is discussed in detail in the rest of the chapter. But first, some definitions and analytical solution to velocity profile will be provided. Velocity component in the x direction can be obtained by the x-component of the linear momentum conservation, which can be simplified as

$$\mu \frac{\partial^2 v_x}{\partial y^2} = \frac{\Delta p}{L} \tag{4.1}$$

which is supported by the following boundary conditions

$$Bottom\ \ wall : v_0 = Kn \frac{\partial v}{\partial y} \tag{4.2}$$

$$Top\ \ wall : v_H = -Kn \frac{\partial v}{\partial y} \tag{4.3}$$

Δp is the pressure drop along the channel section of lenght L. The solution of this equation is a parabolic curve. When a non-dimensional length is defined as $\eta = y/H$ the solution is

$$\frac{v_x}{v_{x,mean}} = -\frac{6(\eta^2 - \eta - Kn)}{6Kn + 1} \tag{4.4}$$

Note that in the case of zero Kn the solution is still valid and corresponds to the case with no-slip on the walls. Heat transfer characteristic of the flow, on the other hand, can be determined

using the Nusselt number. Nusselt number is the ratio of convective and conductive heat transfers. For a fully developed channel flow the Nusselt number converges to a constant value. The mathematical derivation starts with the following equation written for a cross section of the flow

$$h(T_w - T_{mean}) - k\left(\frac{\partial T}{\partial y}\right)_{x=0} = 0 \tag{4.5}$$

where h and k are convective and conductive heat transfer constants, T_w is the temperature on the wall and T_{mean} is the mean temperature of the flow at a cross section defined as

$$T_{mean} = \frac{\int_A \rho v T dA}{\int_A \rho v dA} \tag{4.6}$$

Using the previously defined non-dimensional lenght η, and defining the following non-dimensional temperature

$$\theta = \frac{T_{mean} - T_w}{T_i - T_w} \tag{4.7}$$

where T_i is the constant temperature of the fluid flow at the inlet Eq. 4.5 becomes

$$\frac{k}{H}(T_i - T_w)\left(\frac{\partial \theta}{\partial \eta}\right)_{x=0} = h(T_w - T_{mean}) \tag{4.8}$$

which can be arranged to get the following non-dimensional temperature profile

$$\left(\frac{\partial \theta}{\partial \eta}\right)_{x=0} = \frac{2Hh}{k}\frac{(T_w - T_{mean})}{2(T_i - T_w)} \tag{4.9}$$

Previously mentioned Nusselt number is defined as $Nu = 2Hh/k$, which in this case turns into

$$Nu = -\frac{2}{\theta}\left(\frac{\partial \theta}{\partial \eta}\right)_{x=0} \tag{4.10}$$

4.2 Fluid Flow and Heat Transfer Results in Macrochannel

The Poiseuille flow domain in macro scale (see Fig. 4.1) has no-slip boundary conditions at the walls. We assume to have uniform velocity profile at the inlet and at the exit flow is assumed to be fully developed. The non-dimensional temperature at the inlet and at the walls are 1 and 0 respectively. The aspect ratio is 20, and Re = 10, Pr = 10. Although the problem is solved by a number of different meshes, the results presented here are obtained with a mesh of 81x1620 nodes.

The velocity profile in the developing regime is compared with results from a commercial software COMSOL which uses Finite Element Method (See Fig. 4.2). Normalized velocity in the plot is generated via the division of velocity to the mean velocity over the cross

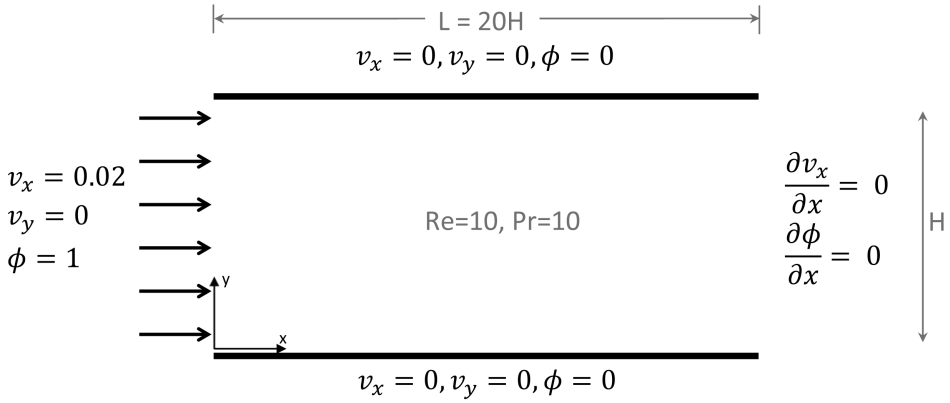

Figure 4.1: Schematic Drawing and Parameters of Poiseuille Flow in Macro Scale

section. The lines belong to LBM solution and signs belong to FEM. Due to the use of a small Reynolds number the velocity profile reaches fully developed regime fast. The lines of $x/L = 0.04$ and $x/L = 0.5$ are overlapping. The difference of the profile $x/L = 0.04$ from analytical solution is less than 1% and the profile at $x/L = 0.5$ covers the analytical solution very well. Temperature profile in the developing regime is compared with FEM in Fig. 4.3.

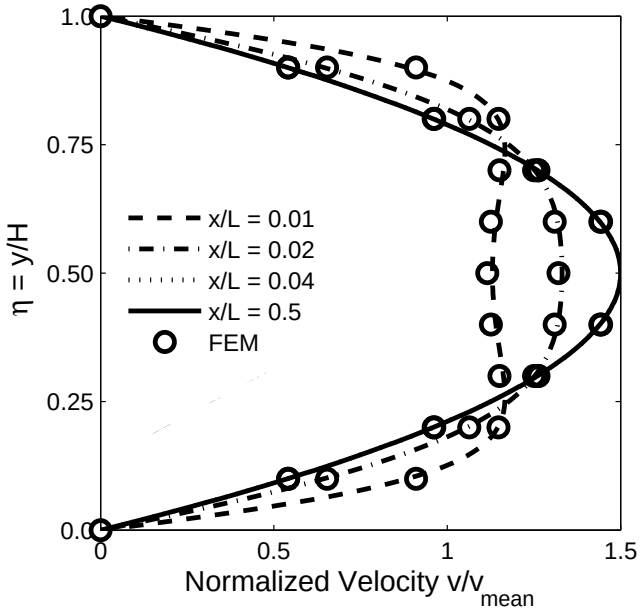

Figure 4.2: FEM and LBM Comparison of Developing Velocity at Macro Scale

The temperature profile at $x/L = 0.2$ is on top of the profile at $x/L = 0.5$, with less than 1% difference, hence after $x/L = 0.2$ the flow is thermally fully developed.

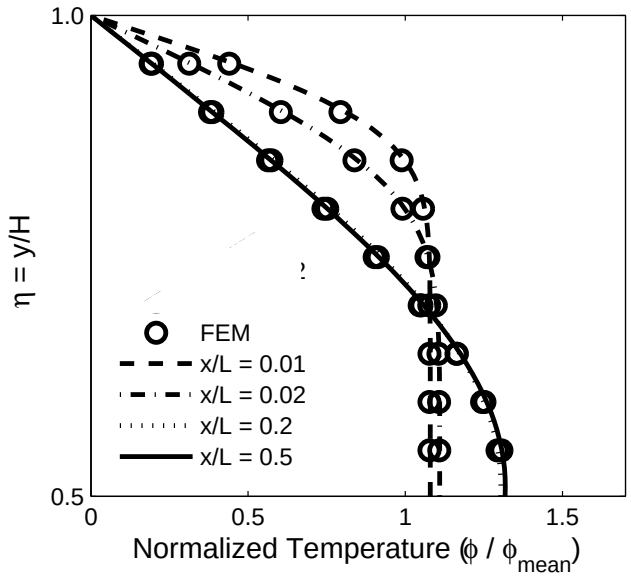

Figure 4.3: FEM and LBM comparison of Developing Temperature at Macro Scale

The Nusselt number variation along the channel is given in Fig. 4.4 where LBM solution is compared with the solution of Bejan [47]. The variation in the developing regime is close to the one given in the reference and by the end of the channel LBM converges to the analytical solution of 7.54.

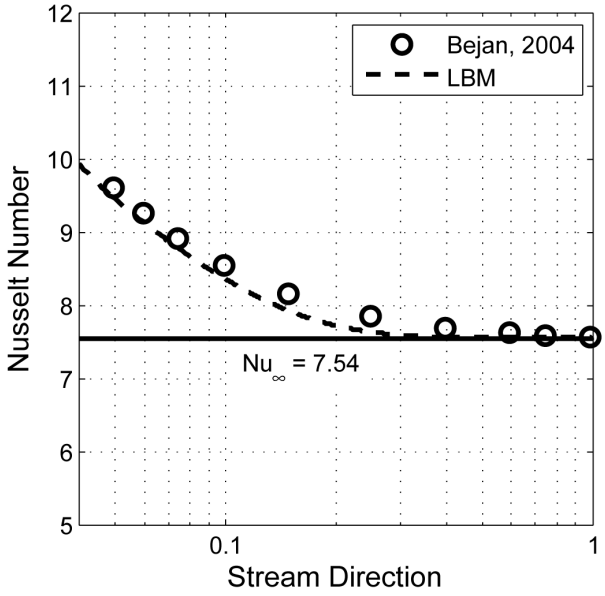

Figure 4.4: Nusselt Number Variation Along the Macrochannel

4.3 Fluid Flow and Heat Transfer Results in Microchannels

After having credible result from LBM for macroflows, the code is modified to simulate microflows in the slip-flow regime. The same problem, Poiseuille flow, is simulated. However, for this case, the flow have different characteristics with slip velocity and temperature jump at the boundaries. The boundary conditions are shown in Fig. 4.5. Aspect ratio, Re and Pr are kept unchanged while the boundary conditions had non-zero value. Boundary velocity and temperature values depend on Knudsen number, and therefore we repeated the simulation for several Knudsen numbers between 0 and 0.1.

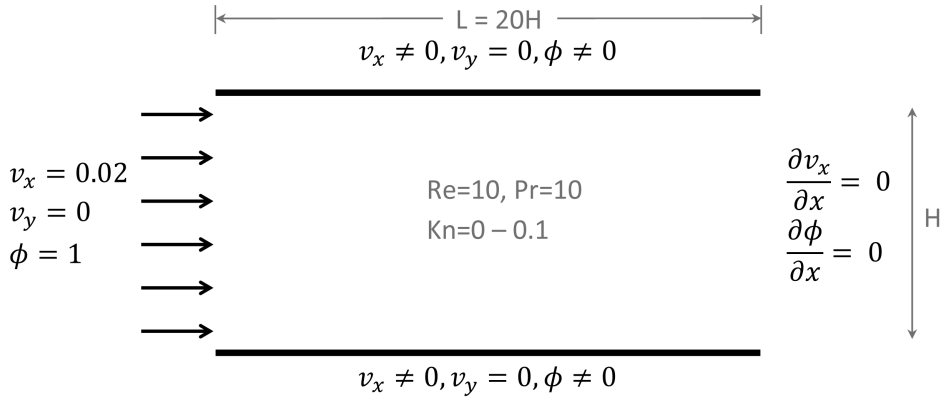

Figure 4.5: Schematic Drawing of Poiseuille flow in micro scale

To observe the validity of the boundary condition implementation, first the velocity profile for different Knudsen numbers are plotted together with the analytical solution, where perfect agreement is observed (Fig. 4.6). The Nusselt number depends on the velocity and temperature profiles. For the developing part of the flow, Nusselt number seems to converge to a value as soon as both the velocity and temperature become fully developed. The characteristic of velocity is determined by the Kn and the temperature profile is determined by Kn and κ defined in Eq. 3.72. Results are obtained for different κ and Knudsen numbers.

Fig. 4.7 shows Nusselt number variation along the channel for developing and developed regimes for $\kappa = 0$ and for $Kn = 0$, 0.04, 0.08. As expected, the $Kn = 0$ case converges to the analytical result for macro channels. Increasing Kn results in higher Nusselt number. Also note that $\kappa = 0$ indicates that there exists no temperature jump which is a fictitious case but is

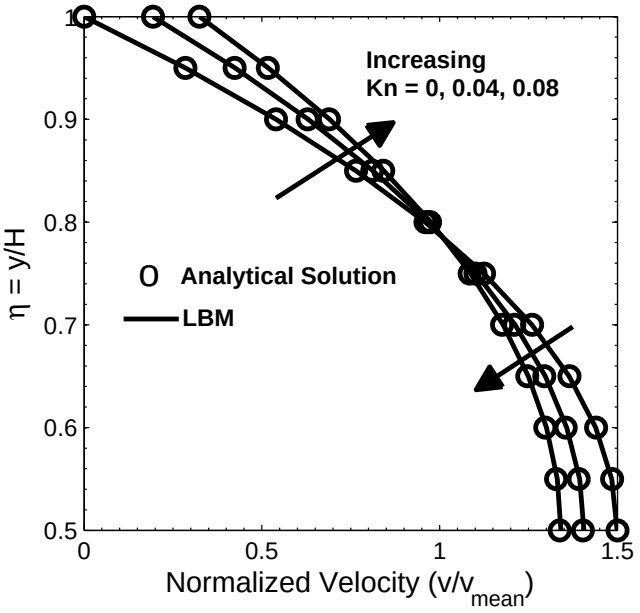

Figure 4.6: Analytical Solution and LBM Comparison of Developed Velocity inside Microchannel

used in data validation in the literature [48].

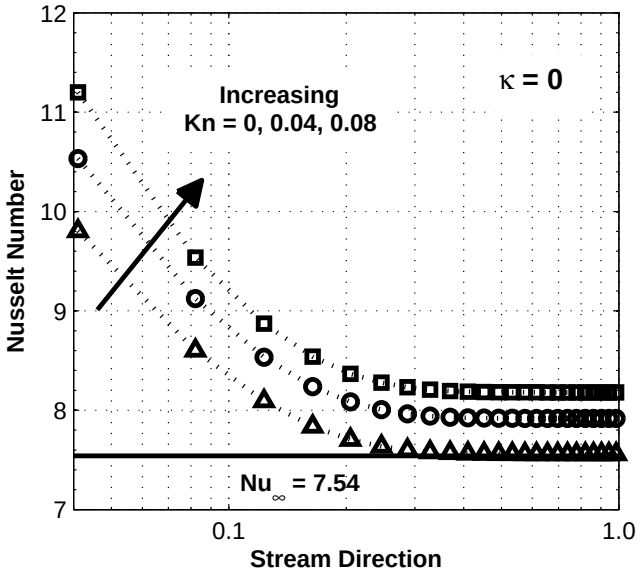

Figure 4.7: Nusselt Number Variation along the Channel for Different Knudsen Numbers at $\kappa = 0$

Fig. 4.8 has the results for $\kappa = 1.667$ which is a typical value for air. A non zero κ and hence

temperature jump, changed the characteristic of Nusselt number to a decreasing trend with increasing Knudsen number. The heat transfer in the channel occurs in two ways: Convection and conduction. The value of Nu indicates the ratio of heat transfer due to convection and conduction. Note that larger κ and Kn resulted in decrease of Nu which means that heat convection rate is decreased.

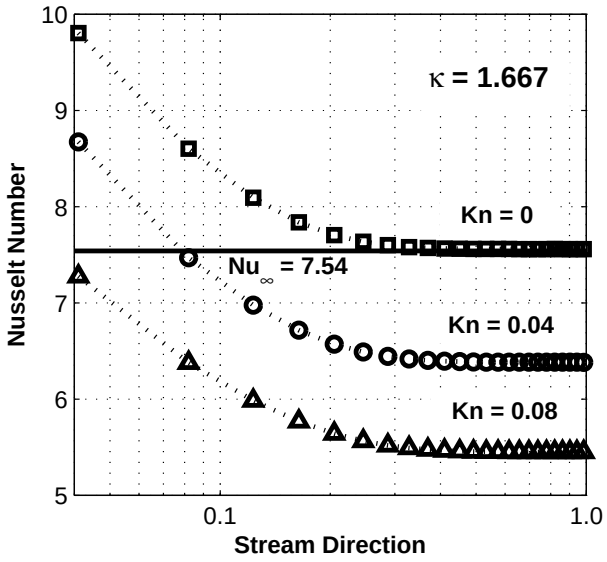

Figure 4.8: Nusselt Number Variation along the Channel for Different Knudsen Numbers at $\kappa = 1.667$

Fig. 4.9 provides the variation of Nu for $\kappa = 10$ which corresponds to a large temperature jump case. The variation along the channel is much smaller than the previous cases and Nu has lower values. As discussed earlier, the conduction rate increases with κ and Kn which resulted in decrease in Nu. Increasing the Kn of the flow makes the flow to approach the transition Knudsen regime where heat transfer by convection loses its importance and conduction becomes more significant due to the rarefaction effects.

The values of Nusselt number for fully developed flow calculated by the developed LBM code are tabulated in Table 4.1.

A grid convergence test was also performed and tabulated in Table 4.2. A set of runs for several different mesh resolution indicated that the solution is converged to the expected values.

Finally, it is worth to mention that the developed code is also used to solve problems other than

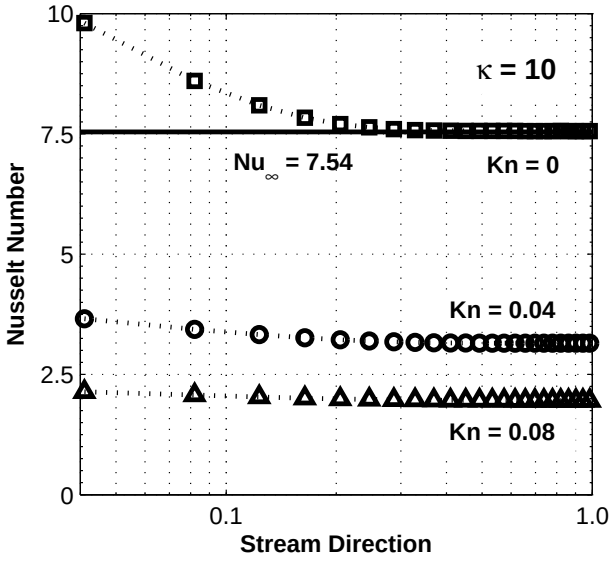

Figure 4.9: Nusselt Number Variation along the Channel for Different Knudsen Numbers at $\kappa = 10$

Table 4.1: Fully Developed Nusselt Numbers in 2D Channel Flow

	Kn					
κ	**0.00**	**0.02**	**0.04**	**0.06**	**0.08**	**0.10**
0.000	7.55	7.75	7.91	8.05	8.18	8.28
1.667	7.55	6.93	6.38	5.88	5.44	5.06
10.00	7.55	4.48	3.14	2.41	1.95	1.64

Table 4.2: Grid Convergence Test at $\kappa = 1.667$

	Kn					
Grid Size	**0.00**	**0.02**	**0.04**	**0.06**	**0.08**	**0.10**
21x420	7.57	6.94	6.38	5.88	5.43	5.04
41x820	7.56	6.94	6.38	5.88	5.44	5.06
61x1220	7.55	6.93	6.38	5.88	5.44	5.06
81x1620	7.55	6.93	6.38	5.88	5.44	5.06
Çetin [48]	7.54	6.92	6.37	5.88	5.44	5.05

channel flows. For example for the classical lid-driven Cavity benchmark problem successful results are obtained up to Reynolds numbers of 5000 [49]. Vortex shedding behind a circular cylinder is also simulated successfully with the developed LBM code. For these relatively high Re flows an additional benefit of LBM is observed. Unlike finite volume or finite element techniques, LBM does not need to be altered for convection dominated flows. There is no need to use artificial diffusion or the formulation does need to be modified by the use of stabilization techniques. Also due to the purely explicit nature of the LBM algorithm, time consuming tasks such as the solution of a linear equation system is not required.

CHAPTER 5

GPU Computing

The struggle in product design, analysis and academic research are usually restricted by oppositions and regulatory barriers like the need to reduce the design cycles, costs and environmental influence, satisfy the governmental rules, improve quality and safety and complexity of physics/mathematics. Those barriers made people require the field of accurate (and fast) product and system simulations [50]. The simulations can comprise coupled, complex and non-linear physical phenomena, very fine time steps, large physical space and complicated geometries to investigate. Researchers make use of all the goods in terms of computational power which is sometimes not satisfactory. Nowadays the use of graphical processing units (GPUs) for general purpose parallel scientific computing is seen to be a promising way of accelerating number crunching codes.

Central Processing Units (CPU) are where the mathematical operations are performed in computers and they are designed to perform operations in a sequential order. One of their properties is that with the duplication of computing frequency the speed of CPU is also doubled [51]. The technology in engineering limits the frequency not to exceed 4.0 GHz due to the extreme heat generation close to a heat density of nuclear reactor core [52]. The barrier of extreme heat generation drove people to explore various paths such as parallel programming. Instead of working on a single computer processor serially, researchers began to use multi processors in parallel.

Graphical Processing Units (GPU) was started to be used for general purpose computations in 1990's [51]. GPUs were known as capable to perform basic mathematical operations only. Even though their main purpose was to visualize a 3D virtual world on a 2D screen, appropriate programming languages made them challengers to CPUs. For almost a decade the

manufacturers of GPUs design them to easily handle mathematical operations. Unlike CPUs, GPUs can accommodate hundreds of processors and do not generate heat that we cannot deal with. The working principle of GPUs is also parallel by the nature of the problem they are designed for. During 2000's the use of GPUs is having increasing attention in the field of parallel computing. A drawback of GPU computing could be considered as that the programming languages were too complex. There had been several languages and most of them were abandoned. After the invention of CUDA and OpenCL programming tools, researchers no more need to be experts in computer graphics to harness the computational power of GPUs [53]. However, the nature of parallel computing is not similar to serial programming and needs a different approach to the design of algorithms, a familiarity to cache memory and core sharing features.

Cellular Gas Automata which is the origin of LBM, is known with its highly parallelizable algorithm. This feature is carried to LBM. Many researchers, during the last decade, tested this property of LBM and reported their findings. Tolke developed an LBM code using CUDA. He wrote his code in C language to run on a single GPU and obtained 1 order of magnitude speedup compared to his serially running LBM code [54]. Obrecht et al. wrote their 3DQ19 LBM code and ran it on a nVidia GTX295 GPU [55]. They also obtained nearly 2 fold increase and they reported GPU computing to be a cheap and fast solution. Riegel et al. developed an LBM solver called LBultra and tested it for the 3D benchmark problem of flow over a cylinder [56]. Their code, written in C++ ran on 3 Tesla C1060 GPUs and they reported about 19 fold increase in speed. Their parallel CPU code ran on 4 AMD cores or 2 Intel cores but those could not approach the speed of multi-GPU code. 4 AMD cores provided 1.8 fold speed up and 2 Intel cores provided 2 fold of increase in speed. They also observed that GPU programming saves more energy, space and money than parallel CPU computing. Baieley et al. [57] compared their LBM code running on a single GPU with an OpenMP version that runs on quad-core CPU, and reported a speedup of 28 times.

In this work, we used m language which is served by MathWorks company via the software MATLAB. m language is serial inherently and hence works on CPU. Another program named as Jacket, which is created by AccelerEyes company, is an add-on application to MATLAB and allows MATLAB to perform mathematical operations on GPUs. Basically, Jacket is a link from m language to GPU programming through the use of CUDA technology. The advantage of Jacket is the ease of its use. With a very little afford, measured in usually seconds and

minutes, the standard m language gets ready to run on GPU.

5.1 Run Time Comparison LBM on CPU and GPU

For the Poiseuille flow benchmark problem 3 different programs are developed. These codes are given in App. A and can be downloaded from the website http://code.google.com/p/lbm-jacket-microchannel/. First code solves the problem serially on a single core of a CPU. The CPU used is Intel Xeon E5620 Quad-Core 2.40 GHz. Second code runs parallel on a single GPU that is Tesla C1060. Tesla C1060 has 240 cores and capable of providing 933 GFLOPs/s of performance with 4 GB of GDDR3 memory at 102 GB/s bandwidth. The last one is parallel too, yet distributed among 4 Tesla C1060s and referred as Multi GPU (MGPU) version. Due to the fact that Tesla C1060 GPUs work more efficiently on single precision numbers, all the programs are written to run in single precision. Nevertheless the accuracy of the solutions are checked to be not effected by this.

All the parts of the code, collision, streaming, boundary conditions and macroscopic properties are well suited to parallel computing. There is no node and no distribution function that depends on the others. For example, while calculating the speed of the flow at a grid point we do not need to know the velocities at the neighbouring nodes. This property of LBM makes it convenient to parallel programming.

An important point is that we are comparing the speed of single CPU core with hundreds of GPU cores. The comparison doesn't seem to be fair. One can use several CPU cores and get higher speeds against hundreds of GPU cores. But in order to do so one has to deal with complicated parallel programming algorithms. Using m language with Jacket, we do not worry about the details of parallel programming. In minutes, the code becomes ready to run in parallel. Defining the variables on GPU memory is enough to make the operations performed on GPUs. The difference can be seen in the codes which are present in Appendix A.

If the CPU and GPU programs were written in complied languages such as C or Fortran, most probably they would run much more faster. m language is not compiled and that is a disadvantage if the concern is the speed. However, the code development, post processing, profiling and debugging is much faster in MATLAB environment. The time saved from those processes is thought to eliminate the speed disadvantage of m language.

The physical properties of the flow do not change the run time of the CPU, GPU and MGPU versions, therefore only the results of a single flow are reported. The parameters are: Re = 10, Pr = 10, Kn = 0, aspect ratio = 20. All three LBM codes are composed of 2 parts as seen in Fig. 5.1: momentum and heat transfer calculations, both of which include collision, streaming, boundary conditions and macroscopic property calculation subsections. A pseudo code can be found in Appendix B.

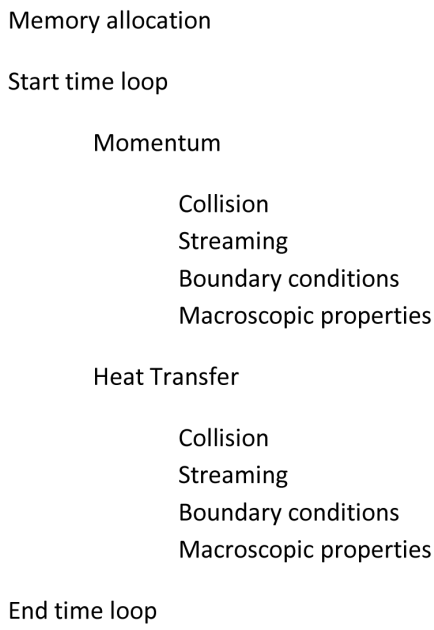

Figure 5.1: Structure of the Developed LBM Code

The solution domain is discretized into MxN grid points. 9 distribution functions which are MxN in size are allocated for each collision part. In the collision the new values of the distribution functions are calculated. Even though the operation on one distribution function does not affect the other distribution functions on the same or other grid points, MATLAB performs these calculations serially. However, Jacket distributes the operations of one distribution function among 240 cores of the GPU. In other words the domain is divided into 240 sub-domains for each distribution function.

The streaming part shifts every distribution function one row and/or column. For instance, consider the first distribution function as a matrix MxN in size. The elements of that matrix are shifted one column to the right, similarly the elements of the second distribution function matrix are shifted one row upward. The other ones are also shifted according to their

directions as demonstrated in Fig. 3.4. CPU computing starts from the first grid point and ranges over all the others serially. On the other hand, GPU scatters the operations evenly through all the cores available to it. Similar to collision, the domain is divided into hundreds of sub-domains and all those sub-domains are handled at the same time.

The boundary conditions are the parts where minimum effort is spent. Only 3 distribution functions on 2x(M+N) nodes are edited. The operation time for BC implementation on both serial and parallel computing is negligible.

In macroscopic property calculations, 9 distribution functions are added up at every grid points. If it is the momentum part, also the vectorial sum of the distribution functions are performed. GPU again distributes the operations into 240 cores.

Table 5.1: Time in Percentages which are Consumed in Different Parts of the Codes for Mesh Size 4000 x 1000

	CPU Code	GPU Code
Collision	82%	15%
Streaming	2%	76%
Boundary Conditions	0.3%	8%
Macroscopic Properties	16%	1%

Time consumed for different tasks performed by the LBM codes are provided in Table 5.1. Note that this table is generated for a mesh of 4000 x 1000 nodes. Percentage times for different mesh sizes resulted almost the same. For the code which runs on CPU, the collision part takes 82% of the time. This part is where the most of the mathematical calculations are performed. Macroscopic property calculation section consumes 16% of the time where streaming and boundary condition implementation consume 2% and 0.3% of the total elapsed time respectively. On the other hand these percentages change significantly in GPU codes. Streaming takes 76%, collision takes 15%, boundary condition implementation part takes 8% and macroscopic property calculations take 1% of the total run time. Parallelism decreased the time spent in collision and macroscopic property calculation parts significantly and hence other parts, streaming and boundary condition implementation, became more dominant.

The Multi GPU code has a little different structure than the CPU and single GPU code. In order to utilize all 4 GPUs, the domain is divided into 4 parts as seen in Fig. 5.2. Every part is solved on a different GPU. Due to the fact that every device has its own memory, the variables should be allocated on each device with different names. After the collision, the nodes on

GPU boundaries are updated using ghost nodes. The update is a data exchange between neighbouring GPUs as seen in Fig. 5.2. The data is first transferred to CPU memory and than to the other GPU. This process only exists in Multi GPU code and brings extra load to the program. Separating two adjacent nodes ceases the intercommunication on the boundaries of sub-domains. The lack of communication can be eliminated by ghost nodes. First step is memory allocation. To create ghost nodes, add one more column (or row) for each boundary of the sub-domains. The new nodes, in our case are in a column, are called ghost nodes which are represented as circles in Fig. 5.3. The variables on ghost nodes are assigned from the neighbouring sub-domain. They are going to be used just to calculate new data in each sub-domain and just after they need to be updated using the neighbouring sub-domain. After the variables in each sub-domain are calculated (after the collision), the variables on ghost nodes are updated using the nodes in neighbouring sub-domain. The dashed arrows indicates the transfer origins and destinies of the variables.

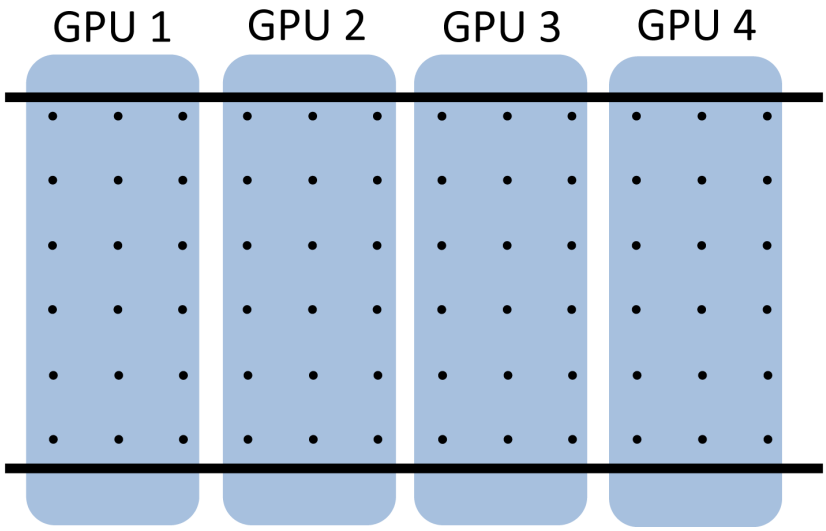

Figure 5.2: Sharing of the Domain among GPUs

The computational time demonstrated in Fig. 5.4 is not the convergence time but rather it is a time that covers a smaller number of time steps which is enough to compare the speeds. It is seen that with the increasing mesh density CPU time increase continuously . Single GPU parallel code has a constant elapsed time up to a mesh size of 10^5. Before that mesh size, single GPU consumes more time than a single CPU core. After a mesh size of $3x10^7$ the GPU time approaches to CPU time due to memory issues. The memory of CPU is 24GB whereas

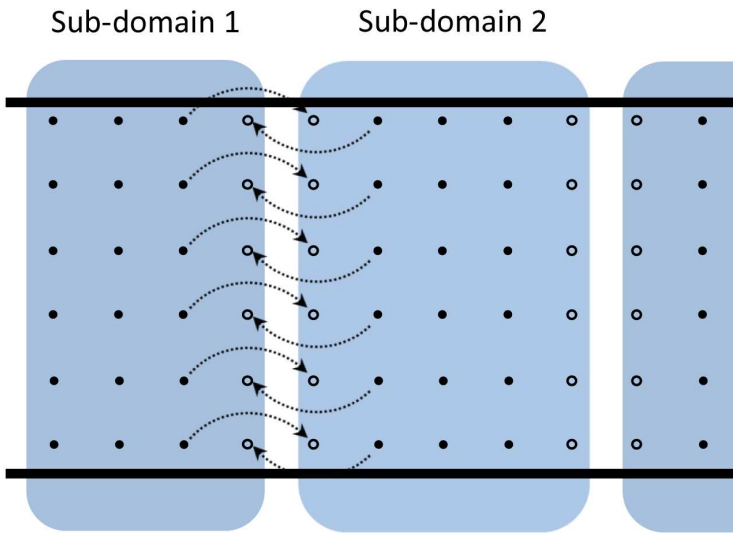

Figure 5.3: Ghost Nodes

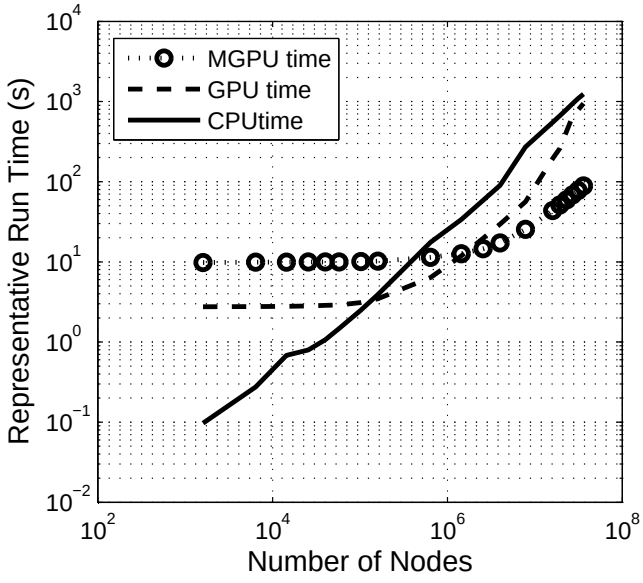

Figure 5.4: Comparison of MGPU, GPU and CPU Run Times

a single Tesla C1060 has only 4GB memory. In other words, for mesh sizes greater than 10^5 and smaller than $3x10^7$, it is feasible to run on GPU. The curve for MGPU is steady up to a mesh size of 10^6. After this point both CPU and GPU times increase faster, so MGPU can be used for meshes greater than 10^6.

The GPU and MGPU programs have constant speeds for smaller mesh sizes. This is a common picture in GPU computing. Even though the code runs in GPU, there should be variable transfer between CPU memory and GPU memory. The time due to variable transfer is dominant for small mesh sizes. In the Fig. 5.4, most of the time spent in the steady region is due to the variable transfer. A mesh size of 10^5 for GPU and 10^6 for MGPU programs are the limits where the time spent on computation begins to overcome the time spent on variable transfer. Due to this fact, domains with small mesh number cannot be accelerated; also, they run slower than serial computing.

The numerical results of Fig. 5.4 are tabulated in Table 5.2. Table 5.3 shows the speed ups. Speed up values are calculated as a ratio of run times. Compared to the serial program, the single GPU parallel program can consume about 5 fold less time at $7.84x10^7$ grid points and 14 fold increase in the speed of MGPU program is observed for the maximum mesh size.

During the current study, Accelereyes company released several versions of Jacket. Surprisingly, older versions of Jacket performed better. The Jacket library is a closed box, users cannot modify or explore what is written inside. The same LBM code showed worse performance on latest versions of Jacket.

LBM works with discrete time, space and velocity. The codes are written in 2 dimensional space with 9 speeds. The scheme is addressed as D2Q9. Beside D2Q9 there are other schemes like D2Q4, D2Q5, D2Q8 etc. D2Q9 is a commonly used scheme. It results in good accuracy but decreasing the number of discrete velocities can cause a speed up. If the code is enhanced two solve three dimensional problems the scheme changes. Typical 3 dimensional versions are D3Q15, D3Q19 and D3Q27. Increasing number of discrete velocities reduces the numerical error, yet it also brings computational load. Using a different scheme should be accompanied with appropriate boundary conditions. The boundary condition equations are needed to be re-derived. Note that all the boundary conditions in this work are for D2Q9 model. Collision, streaming and macroscopic property calculation parts do not require severe alterations.

Table 5.2: Representative MGPU, GPU and CPU Run Times in Seconds for Different Mesh Sizes

Mesh Size	MGPU	GPU	CPU
$1.60x10^3$	10.12	2.850	0.153
$6.40x10^3$	10.16	2.850	0.276
$2.56x10^3$	10.19	2.897	0.819
$4.00x10^4$	10.67	2.962	1.274
$5.76x10^4$	10.21	2.996	1.477
$1.02x10^5$	10.35	3.142	2.582
$1.60x10^5$	10.43	3.541	3.792
$6.40x10^5$	11.65	6.396	15.51
$1.44x10^6$	12.96	11.92	33.07
$2.56x10^6$	14.85	19.66	58.53
$4.00x10^6$	17.45	29.59	90.14
$7.84x10^6$	25.72	56.09	271.6
$1.60x10^7$	43.78	147.9	553.2
$1.94x10^7$	51.23	331.3	670.2
$2.30x10^7$	59.40	335.9	794.4
$2.70x10^7$	68.53	563.5	933.2
$3.14x10^7$	78.01	701.3	1084
$3.60x10^7$	88.80	798.9	1261

Table 5.3: MGPU, GPU and CPU Speedups for Different Mesh Sizes

Mesh Size	MGPU vs CPU	GPU vs CPU	MGPU vs GPU
$1.60x10^3$	0.01	0.05	0.28
$6.40x10^3$	0.03	0.09	0.28
$2.56x10^3$	0.08	0.28	0.28
$4.00x10^4$	0.12	0.43	0.28
$5.76x10^4$	0.14	0.49	0.29
$1.02x10^5$	0.25	0.82	0.30
$1.60x10^5$	0.36	1.07	0.34
$6.40x10^5$	1.33	2.43	0.55
$1.44x10^6$	2.55	2.77	0.92
$2.56x10^6$	3.94	2.98	1.32
$4.00x10^6$	5.17	3.05	1.70
$7.84x10^6$	10.6	4.84	2.18
$1.60x10^7$	12.7	3.74	3.38
$1.94x10^7$	13.1	2.02	6.47
$2.30x10^7$	13.4	2.37	5.66
$2.70x10^7$	13.6	1.66	8.22
$3.14x10^7$	13.9	1.55	8.99
$3.60x10^7$	14.2	1.58	9.00

As a final note it is worth to mention that couple of the finest meshes used in this project is too much for a basic 2D benchmark problem. The intention was to demonstrate the computational power comparison of CPU and GPUs. But such large meshes can be used in three dimensional problems. Three dimensional version of the code would have higher computational time due to the increase in mathematical operations and it is anticipated that the speed up of GPU would be higher.

CHAPTER 6

Conclusion and Future Work

Micro Electro Mechanical Systems combine electronics, magnetics, acoustics, biology, chemistry and fluid mechanics in very small dimensions. This fruitful and compact blend provides them to have promising future in many fields like transportation, medicine, telecommunication, computer, military, manufacturing, etc. Design of such systems, of course, are challenging. Not well prepared designs invite time and money lose in most cases. The heart of a well prepared design is accurate and fast simulation. There are numerous methods to simulate systems and products, yet proper simulation tool should be selected for proper systems. For MEMS, the assumptions of physics are different than many other engineering applications. The assumptions of mathematical model and the assumption of the physics of MEMS must match. The commonly used mathematical approaches to MEMS are modified Navier-Stokes equations as well as Newtonian mechanics equations and finally the Boltzmann Transport Equation.

In theory, the Boltzmann Transport Equation governs continuum, slip-flow, transition and free molecular regimes yet BTE is very hard to solve if it is possible. Therefore, researchers are studying numerical solutions of BTE. The numerical solution of BTE is capable to simulate continuum and slip flow regimes like, Burnett and Navier-Stokes equations do and also transition and free molecular regimes like Newtonian mechanics equations do. A powerful numerical tool to solve the BTE is the Lattice Boltzmann Method. In this project, LBM is studied for continuum and slip flow regimes in microchannels which are encountered in MEMS. Using the LBM we have simulated Poiseuille flow in continuum regime. The validity of the method in continuum regime is proved by the comparison of the velocity profile with the analytical solution and the comparison of the temperature profile from a numerical result via the Finite Element method. The Nusselt number in developing and fully developed flows are compared

with the solution from a reliable source e.g. Bejan [47]. After obtaining well agreed results the program is advanced to simulate slip-flow regime: slip velocity and temperature jump boundary conditions are implemented. The velocity profile is well agreed with analytical solution and also for the $Kn = 0$ case the Nu converged to the analytical solution. In the light of these information the Nusselt number variation along the channel is generated for different Kn and κ values.

Three different Lattice Boltzmann codes have been developed. First one runs serially on CPU. Second one runs on a single GPU and the last one is capable to run on 4 GPUs. The LBM codes are written in MATLAB which runs on CPU by default. However, using a commercial software, Jacket, the serial MATLAB codes are converted to the parallel ones easily. It is reported that the usage of Jacket with MATLAB saves time in the stages of developing a new code, as well as running the codes. Using the parallel computing technology, it is observed that a simulation time can be reduces 14 times for very large mesh sizes.

The consistent results which are obtained from this study encouraged us to enhance the developed codes. The other rarefaction effect, viscous heating, will be added in the method for a more proper microflow simulation. Further more, the ability of LBM to solve for mutliphase flows will be examined. In the beginning, we will focus on a two-phase problem: The simulation of bubble formation which is an important phenomena encountered in heat removal processes. Those problems will also be solved in 3D domains and hence more realistic solutions will be obtained. Due to the fact that working on a 3D domain requires more computational power and memory, the codes will be written in C language and they will run parallel on GPUs.

REFERENCES

[1] M. Gad-elHak. *Advances in Multiphysics Simulation and Experimental Testing of MEMS*. Imperial College Press, 2008.

[2] D. A. Koester, K. W. Markus, and M. D. Walters. MEMS: Small machines for the microelectronics age. *Computer*, 29:93–94, 1996.

[3] R. R. Mansour. RF MEMS for space applications. In *International Conference on MEMS, NANO and Smart Systems (ICMENS05)*, 2005.

[4] H. Helvajian. *Microengineering Technology for Space Systems*. The Aerospace Press, Los Angeles, 1997.

[5] J. L. Zunino, D. Skelton, and R. Mason. Micro-electromechanical systems (MEMS) reliability assessment program for department of defende activities. *Nanotech*, 3:463–466, 2005.

[6] J. Bouchaud. RF MEMS: status of the industry and roadmaps. In *Radio Frequency integrated Circuits (RFIC) Symposium*, pages 379–384, 2005.

[7] Y. H. Zhang, X. J. gu, R. W. Barber, and D. R. Emerson. Capturing knudsen layer phenomena using a lattice Boltzmann model. *Physical Review E*, 74(046704), 2006.

[8] G. Karniadakis, A. Beskok, and N. Aluru. *Microflows and Nanoflows Fundamentals and Simulation*. Springer, 2005.

[9] M. Gad-elHak. Gas and liquid transport at microscale. *Heat Transfer Engineering*, 27(4):13–29, 2006.

[10] C. Cercignani. *Mathematical Methods in Kinetic Theory*. Plenum Press, New York, 1969.

[11] G. A. Bird. Monte Carlo simulation of gas flows. *Ann. Rev. Fluid Mech.*, 10:11–31, 1978.

[12] U. Frisch, B.Hasslacher, and Y. Pomeau. Lattice-gas automata for the Navier-Stokes equation. *Phys. Rev. Lett.*, 56(14):1505–1508, 1986.

[13] C. Cercignani. *The Boltzmann Equation and Its Applications (Applied Mathematical Sciences)*. Springer-Verlag, 1988.

[14] S. Harris. *An Introduction to the Theory of the Boltzmann Equation*. Holt, Rinehart and Winston, Inc, USA, 1971.

[15] Richard L. Liboff. *Kinetic Theory Classical, Quantum, and Relavistic Descriptions*. Springer, New York, 3rd edition, 2003.

[16] P. V. Panat. *Thermodynamics and Statistical Mehanics*. alpha Science, Oxford, 2008.

[17] I. Prigogine and F. C. Andrews. A Boltzmann-like approach for traffic flow. *Operations Research*, 8(6):789–797, 1960.

[18] S.M. YEN. Numerical solution of the nonlinear Boltzmann equation for nonequilibrium gas flow problems. *Annual Review of Fluid Mechanics*, 16(1):67–97, 1984.

[19] A. J. C. Ladd. Numerical simulations of particulate suspensions via a discretized Boltzmann equation. *Journal of Fluid Mechanics*, 271:311–339, 1994.

[20] N. Goldsman, L. Henrickson, and J.Frey. A physics-based analytical/numericalsolution to the Boltzmann transport equation for use in device simulation. *Solid-State Electronics*, 34(4):389–396, 1991.

[21] I. D. Reid. An investigation of the accuracy of numerical solutions of Boltzmann's equation for electron swarms in gases with large inelastic cross sections. *Australian Journal of Physics*, 32(3):231–254, 1979.

[22] C. Busch and U. Kortshagen. Numerical solution of the spatially inhomogeneous Boltzmann equation and verification of the nonlocal approach for an argon plasma. *Phys. Rev. E*, 51(1):280–288, 1995.

[23] U. Frisch, D. D'Humieres, B. Hasslacher, P. Lallemand, Y. Pomeau, and J. Rivet. Lattice gas hydrodynamics in two and three dimensions. *Complex Systems*, 1:649–707, 1987.

[24] G. R. McNamara and G. Zanetti. Use of Boltzmann equation to simulate lattice-gas automata. *Phys. Rev. Let.*, 61(20):2332–2335, 1988.

[25] F. J. Higuera and J. Jimenez. Boltzmann approach to lattice gas simulations. *Europhys. Lett.*, 9(7):663–668, 1989.

[26] F. J. Higuera, S. Succi, and R. Benzi. Lattice gas dynamics with enhanced collisions. *Europhys. Lett.*, 9(4):345–349, 1989.

[27] S. Chen, H. Chen, D. Martinez, and W. Matthaeus. Lattice Boltzmann model for simulation of magnetohydrodynamics. *Physc. Rev. Lett.*, 67(27):3776–3779, 1991.

[28] Y. H. Qian, D. D'Humieres, and P. Lallemand. Lattice BGK models for Navier-Stokes equations. *Europhysc. Lett.*, 17(6):479–484, 1992.

[29] P. L. Bhatnagar, E. P. Groos, and M. Krook. A model for collision processes in gases. i. small amplitude processes in charged and neutral one-component systems. *Physc. Rev.*, 94(3):511–525, 1954.

[30] X. He and L. Luo. Theory of the lattice Boltzmann method: From the Boltzmann equation to the lattice Boltzmann equation. *Physc. Rev. E*, 56(6):6811–6817, 1997.

[31] T. Abe. Derivation of the lattice Boltzmann method by means of the discrete ordinate method for the Boltzmann equation. *J. Comp. Physc.*, 131(1):241–246, 1997.

[32] S. Chen and G. Doolen. Lattice Boltzmann method for fluid flows. *Annu. Rev. Fluid Mech.*, 30(1):329–364, 1998.

[33] A. A. Mohamad. *Lattice Boltzmann Method Fundamentals and Engineering Applications with Computer Codes*. Springer, 2011.

[34] S. Hou, Q. Zou, S. Chen, G. Doolen, and A. C. Cogley. Simulation of cavity flow by the lattice Boltzmann method. *J. Comput. Physics*, 118(2):329–347, 1994.

[35] F. J. Alexander, S. Chen, and J. D. Sterling. Lattice Boltzmann thermohydrodynamics. *Physical Review E*, 47(4), 1993.

[36] Y. Chen, H. Ohashi, and M. Akiyama. Thermal lattice Bhatnagar-Gross-Krook model without nonlinear deviations in macrodynamic equations. *Phys. Rev. E*, 50(4):2776–2783, 1994.

[37] G. R. McNamara, A. L. Garcia, and B. J. Alder. Stabilization of thermal lattice Boltzmann models. *J. of Stat. Physc.*, 81(1-2):395–408, 1995.

[38] X. Shan. Simulation of rayleigh-benard convection using a lattice boltzman method. *Physc. Rev. E*, 55(3):2780–2788, 1997.

[39] P. Pavlo, G. Vahala, L. Vahala, and M. Soe. Linear stability analysis of thermo-lattice Boltzmann models. *J. of Com. Physc.*, 139:79–91, 1998.

[40] G. Vahala, P. Pavlo, L. Vahala, and N. S. Martys. Thermal lattice Boltzmann models (tlbm) for compressible flows. *Int. J. of M. Physics C*, 9(8):1247–1261, 1998.

[41] X. He, S. Chen, and G. D. Doolen. A novel thermal model for the lattice Boltzmann method in incompressible limit. *J. of Com. Physics*, 146:282–300, 1998.

[42] Jonas Latt. *Choice of units in lattice Boltzmann simulations*, April 2008.

[43] X. He, Q. Zou, L. Luo, and M. Dembo. Analytic solutions of simple flows and analysis of nonslip boundary conditions for the lattice Boltzmann BGK model. *J. of Stat. Physc.*, 87(1/2), 1997.

[44] S. Succi. *The Lattice Boltzmann Equation: For Fluid Dynamics and Beyond (Numerical Mathematics and Scientific Computation)*. Clarendon, 2001.

[45] Z. Tian, C. Zou, Z. Liu, Z. Guo, H. Liu, and C. Zheng. Lattice Boltzmann method in simulation of thermal micro-flow with temperature jump. *Int. J. of Modern Physc. C*, 17(5):603–614, 2006.

[46] R. M. Jr. Drake E. G. R. Eckert. *Analysis of Heat and Mass Transfer*. McGraw-Hill, Newyork, 1972.

[47] A. Bejan. *Convective Heat Transfer*. John Wiley & Sons, 2004.

[48] B. Cetin. Anaylsis of single phase convective heat transfer in microtubes and microchannels. Master's thesis, Middle East Technical University, 2005.

[49] S. B. Celik, C. Sert, and B. Cetin. Simulation of lid-driven cavity flow by parallel implementation of lattice Boltzmann method on GPUs. In *International Symposium on Computing in Science and Engineering (ISCSE 2011)*, 2011.

[50] M. Eldredge, T. J. R. Hughes, R. M. Ferencz, S. M. Rifai, A. Raefsky, and B. Herndon. High-performance parallel computing in industry. *Parallel Computing*, 23:1217–1233, 1997.

[51] A. R. Brodtkorb, T. R. Hagen, and M. L. Saetra. Graphics processing unit (GPU) programming strategies and trends in GPU. *J. Parallel Distrib. Comput.*, 2012.

[52] G. Taylor. Energy efficient circuit design and the future of power delivery. *EPEPS'09*, 2011.

[53] J. E. Stone, D. J. Hardy, I. S. Ufimtsev, and K. Schulten. GPU-accelerated molecular modelling comming of age. *Journal of Molecular Graphics and Modelling*, 29:116–125, 2010.

[54] Jonas Tolke. Implementation of a Lattice Boltzmann kernel using the compute unified device architecture developed by nVidia. *Comput Visual Sci*, 13:29–39, 2010.

[55] C. Obrecht, F. Kuznik, B. Tourancheau, and J. J. Roux. A new approach to the Lattice Boltzmann Method for graphics processing units. *Computers and Mathematics with Applications*, 61:3628–3638, 2011.

[56] E. Riegel, T. Indinger, and N. A. Adams. Implementation of a Lattice Boltzmann Method for numerical fluid mechanics using the nVIDIA CUDA technology. *CSRD*, 23:241–247, 2009.

[57] International Conference on Parallel Processing. *Accelerating Lattice Boltzmann Fluid flow simulations using Graphics processors*. IEEE, 2009.

Appendix A

Developed LBM Codes

All the codes in Appendix A are available online: http://code.google.com/p/lbm-jacket-microchannel/

A.1 CPU Code

```
function time = LBM_CPU(nY)
    % nY is the Horizontal Node Number
    % Slit Channel
    % Runs on CPU
    % Flow + Heat Transfer
    % Calculates Nusselt Number

    nX = nY*4; % nX is the Vertical Node Number.
%     For this example it is choosen to be 4 times of height
    tstep = 100;
    Kn = 0;
    Kappa = 0;
    Lambda = Kn*(nY-1);
    C = Kappa*Lambda;
%     Nu = zeros(nX,1); % Nusselt Number Calculation is omitted. When this
%     variable definition and the commented out parts before Momentum section are
%     uncommented the code can calculate the Nusselt number.
    Re = 10;
    Pe = 100;
    Ulattice = 0.02;
    Pr = Pe/Re;
    Viscosity=Ulattice*(nY-1)/Re;
    alpha = Viscosity/Pr;
    omega = 1/(3*Viscosity+0.5);
    oneMinusOmega = 1-omega;
    omegat = 1/(3*alpha+0.5);
    oneMinusOmegat = 1-omegat;
```

```
    rho = zeros(nX,nY);
    f1 = rho;f2 = rho;f3 = rho;f4 = rho;f5 = rho;f6 = rho;f7 = rho;f8 = rho;f9 = rho;
    u = rho; v=rho;
    T = zeros(nX,nY);
    g1 = rho;g2 = rho;g3 = rho;g4 = rho;g5 = rho;g6 = rho;g7 = rho;g8 = rho;g9 = rho;
    rho = rho+5;

    tic
    for kk=1:tstep
%         if mod(kk,1000)==0
%           %---------Nusselt-------------
%             [Nu, Nuold] = NusseltcalculatorD2Q9(u,rho,Ulattice,U_in,T,nX,nY,Height,Kn,Nu);
%             NuoldSum = sum(abs(Nuold));
%             NuSum = sum(abs(Nu));
%             diff = abs(NuSum-NuoldSum);
%             if diff<0.001
%                 display('Program converged and paused');
%                 savestring = ['Nusselt_','Kn',num2str(Kn),'Kappa',num2str(Kappa),'Pe',num2str(Pe),'Re',num2str(Re),'N',num2str(nY),'r',num2
%                 save(savestring)
%                 break
%             end
%             disp(kk)
%             disp(Nu(max(nX)-100))
%             Kn
%             Kappa
%             nY
%         end

%% Momentum

        % Collision
        [f1,f2,f3,f4,f5,f6,f7,f8,f9] = CollisionD2Q9(u,v,rho,f1,f2,f3,f4,f5,f6,f7,f8,f9,omega,oneMinusOmega);

        % Streaming
```

```
        [f1,f2,f3,f4,f5,f6,f7,f8] = StreamingD2Q9(f1,f2,f3,f4,f5,f6,f7,f8,nX,nY);

        % Boundary Conditions
        [f1(1,:),f2(1,:),f4(1,:),f5(1,:),f6(1,:),f7(1,:),f8(1,:)] = InletconstantVelocityD2Q9(f2(1,:),f3(1,:),f4(1,:),f6(1,:),f7(1,:),f9(1,:)
        [f3(nX,:),f6(nX,:),f7(nX,:)] = OutletConstantVelocity( f1(nX,:),f2(nX,:),f4(nX,:),f5(nX,:),f8(nX,:),f9(nX,:),u(nX-10,:));
        [f2(:,1),f5(:,1),f6(:,1)] = SlipBottomWallD2Q9(f1(:,1),f3(:,1),f4(:,1),f7(:,1),f8(:,1),f9(:,1),u(:,2:3),Kn,nY);
        [f4(:,nY),f7(:,nY),f8(:,nY)] = SlipTopWallD2Q9(f1(:,nY),f2(:,nY),f3(:,nY),f5(:,nY),f6(:,nY),f9(:,nY),u(:,nY-2:nY-1),Kn,nY);

        % Macroscopic Values
        [u,v,rho] = MacroscopicD2Q9(f1,f2,f3,f4,f5,f6,f7,f8,f9,nX,nY);

    %% Heat Diffiusion

        % Collision
        [g1,g2,g3,g4,g5,g6,g7,g8,g9] = CollisionTD2Q9(u,v,T,g1,g2,g3,g4,g5,g6,g7,g8,g9,omegat,oneMinusOmegat);

        % Streaming
        [g1,g2,g3,g4,g5,g6,g7,g8] = StreamingD2Q9(g1,g2,g3,g4,g5,g6,g7,g8,nX,nY);

        % Boundary Conditions for Temperature
        g1(1,2:nY-1)=2/9-g3(1,2:nY-1);
        g5(1,2:nY-1)=1/18-g7(1,2:nY-1);
        g8(1,2:nY-1)=1/18-g6(1,2:nY-1);

        % Outlet zero-flux (Extrapolation)
        g3(nX,2:nY-1)=2*g3(nX-1,2:nY-1)-g3(nX-2,2:nY-1);
        g6(nX,2:nY-1)=2*g6(nX-1,2:nY-1)-g6(nX-2,2:nY-1);
        g7(nX,2:nY-1)=2*g7(nX-1,2:nY-1)-g7(nX-2,2:nY-1);

        % Wall Boundaries
        [g4(:,nY),g7(:,nY),g8(:,nY)] = TJumpTop(g2(:,nY),g5(:,nY),g6(:,nY),T(:,[nY-2:nY-1]),0,C);
        [g2(:,1),g5(:,1),g6(:,1)] = TJumpBottom(g4(:,1),g7(:,1),g8(:,1),T(:,2:3),0,C);  % 0 = Wall teperature

        % Macroscopic
```

```
        T = MacroscopicTD2Q9(g1,g2,g3,g4,g5,g6,g7,g8,g9);
    end
time = toc;
fprintf('CPU: time = %6.4f for %d X %d\n',time,nX,nY);
end
%%%%%%%% End of Main Function %%%%%%%%

function [g4,g7,g8] = TJumpTop(g2,g5,g6,T,Tw,C)
    tw = (C*(4*T(:,2)-T(:,1))+2*Tw)/(2+3*C);
    g8=tw/18-g6;
    g7=tw/18-g5;
    g4=2*tw/9-g2;
end

function [g2,g5,g6] = TJumpBottom(g4,g7,g8,T,Tw,C)
    tw = (C*(4*T(:,1)-T(:,2))+2*Tw)/(2+3*C);
    g6=tw/18-g8;
    g5=tw/18-g7;
    g2=2*tw/9-g4;
end

function [f1,f2,f3,f4,f5,f6,f7,f8,f9] = CollisionD2Q9(u,v,rho,f1,f2,f3,f4,f5,f6,f7,f8,f9,omega,oneMinusOmega)
    t10 = u.*u + v.*v;
    t10 = 1.5*t10;
    t1 =  u;
    t2 =  v;
    t3 = -u;
    t4 = -v;
    t5 =  u + v;
    t6 = -u + v;
    t7 = -u - v;
    t8 =  u - v;
    feq1 = rho/9 .*  (1 + 3*t1 + 4.5*t1.*t1 - t10);
    feq2 = rho/9 .*  (1 + 3*t2 + 4.5*t2.*t2 - t10);
```

```
    feq3 = rho/9 .*  (1 + 3*t3 + 4.5*t3.*t3 - t10);
    feq4 = rho/9 .*  (1 + 3*t4 + 4.5*t4.*t4 - t10);
    feq5 = rho/36 .* (1 + 3*t5 + 4.5*t5.*t5 - t10);
    feq6 = rho/36 .* (1 + 3*t6 + 4.5*t6.*t6 - t10);
    feq7 = rho/36 .* (1 + 3*t7 + 4.5*t7.*t7 - t10);
    feq8 = rho/36 .* (1 + 3*t8 + 4.5*t8.*t8 - t10);
    feq9 = 4*rho/9 .*(1 - t10);
    f1 = omega*feq1 + oneMinusOmega*f1;
    f2 = omega*feq2 + oneMinusOmega*f2;
    f3 = omega*feq3 + oneMinusOmega*f3;
    f4 = omega*feq4 + oneMinusOmega*f4;
    f5 = omega*feq5 + oneMinusOmega*f5;
    f6 = omega*feq6 + oneMinusOmega*f6;
    f7 = omega*feq7 + oneMinusOmega*f7;
    f8 = omega*feq8 + oneMinusOmega*f8;
    f9 = omega*feq9 + oneMinusOmega*f9;
end

function [g1,g2,g3,g4,g5,g6,g7,g8,g9] = CollisionTD2Q9(u,v,th,g1,g2,g3,g4,g5,g6,g7,g8,g9,omegat,oneMinusOmegat)
    t1 =  u;
    t2 =  v;
    t3 = -u;
    t4 = -v;
    t5 =  u + v;
    t6 = -u + v;
    t7 = -u - v;
    t8 =  u - v;
    feq1 = th/9 .* (1 + 3*t1 );
    feq2 = th/9 .* (1 + 3*t2 );
    feq3 = th/9 .* (1 + 3*t3 );
    feq4 = th/9 .* (1 + 3*t4 );
    feq5 = th/36 .* (1 + 3*t5 );
    feq6 = th/36 .* (1 + 3*t6 );
    feq7 = th/36 .* (1 + 3*t7 );
```

```
    feq8 = th/36 .* (1 + 3*t8 );
    feq9 = 4*th/9;
    g1 = omegat*feq1 + oneMinusOmegat*g1;
    g2 = omegat*feq2 + oneMinusOmegat*g2;
    g3 = omegat*feq3 + oneMinusOmegat*g3;
    g4 = omegat*feq4 + oneMinusOmegat*g4;
    g5 = omegat*feq5 + oneMinusOmegat*g5;
    g6 = omegat*feq6 + oneMinusOmegat*g6;
    g7 = omegat*feq7 + oneMinusOmegat*g7;
    g8 = omegat*feq8 + oneMinusOmegat*g8;
    g9 = omegat*feq9 + oneMinusOmegat*g9;
end

function [f1,f2,f4,f5,f6,f7,f8] = InletconstantVelocityD2Q9(f2,f3,f4,f6,f7,f9,Ulattice,nY)
    rhow = (f9 +f2 +f4 +2*(f3 +f6 +f7 ))/(1-Ulattice);
    f1  = f3  + 2*rhow*Ulattice/3;
    f5  = f7  + 0.5*(f4-f2) + rhow*Ulattice/6;
    f8  = f6  + 0.5*(f2-f4) + rhow*Ulattice/6;
end

function [u,v,rho] = MacroscopicD2Q9(f1,f2,f3,f4,f5,f6,f7,f8,f9,nX,nY)
    rho = f1+f2+f3+f4+f5+f6+f7+f8+f9;
    usum =  f1-f3+f5-f6-f7+f8;
    vsum =  f2-f4+f5+f6-f7-f8;
    u = usum./rho;
    v = vsum./rho;
end

function th = MacroscopicTD2Q9(g1,g2,g3,g4,g5,g6,g7,g8,g9)
    th = g1+g2+g3+g4+g5+g6+g7+g8+g9;
end

function [Nu,Nuold] = NusseltcalculatorD2Q9(u,rho,Ulattice,U_real,T,nX,nY,Height,Kn,Nu)
% Ulattice=umean
```

```
Nuold = Nu;
Umean = zeros(nX,1);
dy = 1;
ut = zeros(nX,nY);
T_mean = zeros(nX,1);
A = zeros(nX,1);
H = nY-1;
        for i=1:nX
            dummy=0;
            for j=1:5:nY-1
                dummy=dummy+(5*dy)*(19*rho(i,j)+75*rho(i,j+1)+...
                    50*rho(i,j+2)+50*rho(i,j+3)+75*rho(i,j+4)+19*rho(i,j+5))/288;
            end
            Rhomean(i)=dummy/H;
            dummy=0;
            for j=1:5:nY-1
                dummy=dummy+(5*dy)*(19*rho(i,j)*u(i,j)+75*rho(i,j)*u(i,j+1)+...
                    50*rho(i,j)*u(i,j+2)+50*rho(i,j)*u(i,j+3)+75*rho(i,j)*u(i,j+4)+19*rho(i,j)*u(i,j+5))/288;
            end

            Umean(i)=dummy/Rhomean(i)/H;

            dummy=0;
            ut(i,:) = rho(i,:).*u(i,:).*T(i,:);
            for j=1:5:nY-1
                dummy=dummy+5*dy*(19*ut(i,j)+75*ut(i,j+1)+50*ut(i,j+2)+...
                    50*ut(i,j+3)+75*ut(i,j+4)+19*ut(i,j+5))/288;
            end
            T_mean(i)=(dummy/Rhomean(i)/Umean(i)/H);
            A(i)= 49/20*T(i,1)-6*T(i,2)+15/2*T(i,3)-20/3*T(i,4)+...
                  15/4*T(i,5)-6/5*T(i,6)+1/6*T(i,7); % derivative
            Nu(i) = -A(i)/T_mean(i)*2*H;

        end
```

```
            subplot(2,2,1)
            imagesc(T')
            subplot(2,2,2)
            plot(Nu)
            drawnow
end

function [ f3,f6,f7] = OutletConstantVelocity( f1,f2,f4,f5,f8,f9,u_out )
rho_o = (f9+f2+f4+2*(f1+f5+f8))./(1.0+u_out);
    f3 =f1 -0.667*rho_o.*u_out;
f7 =f5 +0.5*(f2 -f4 )- rho_o.*u_out/6.0;
f6 =f8 +0.5*(f4 -f2 )- rho_o.*u_out/6.0;
end

function [f2,f5,f6] = SlipBottomWallD2Q9(f1,f3,f4,f7,f8,f9,u,Kn,nY)
    Lamda = Kn*(nY-1);
    uslip = Lamda*(4*u(:,1)-u(:,2))/(2+3*Lamda);
    rhow = (f1+f3+f9+2*(f4+f7+f8));
    f2 = f4;
    f5 = rhow.*(1+uslip)/2 - (f1+f8) - (f2+f4+f9)/2;
    f6 = rhow.*(1-uslip)/2 - (f3+f7) - (f2+f4+f9)/2;
end

function [f4,f7,f8] = SlipTopWallD2Q9(f1,f2,f3,f5,f6,f9,u,Kn,nY)
    Lamda = Kn*(nY-1);
    uslip = Lamda*(4*u(:,2)-u(:,1))/(2+3*Lamda);
    rhow = (f1+f3+f9+2*(f2+f5+f6));
    f4 = f2;
    f7 = rhow.*(1-uslip)/2 - (f3+f6) - (f2+f4+f9)/2;
    f8 = rhow.*(1+uslip)/2 - (f1+f5) - (f2+f4+f9)/2;
end

function [f1a,f2a,f3a,f4a,f5a,f6a,f7a,f8a] = StreamingD2Q9(f1,f2,f3,f4,f5,f6,f7,f8,nX,nY)
    f1a = f1([1,1:nX-1],1:nY);
```

```
    f2a = f2(1:nX,[1,1:nY-1]);
    f3a = f3([2:nX,nX],1:nY);
    f4a = f4(1:nX,[2:nY,nY]);
    f5a = f5([1,1:nX-1],[1,1:nY-1]);
    f6a = f6([2:nX,nX],[1,1:nY-1]);
    f7a = f7([2:nX,nX],[2:nY,nY]);
    f8a = f8([1,1:nX-1],[2:nY,nY]);

end
```

A.2 GPU Code

```
function time = LBM_GPU(nY)
    % nY is the Horizontal Node Number
    % Slit Channel
    % Runs on GPU
    % Flow + Heat Transfer
    % Calculates Nusselt Number
    % Linux:
%       addpath /usr/local/jacket/engine
    % Windows:
%    addpath C:\Progra~1\AccelerEyes\Jacket\engine
nY = gsingle(nY);
    nX = nY*4; % nX is the Vertical Node Number
%     For this example it is choosen to be 4 times of height
    Kn = gsingle(0);
    Kappa = gsingle(0);
    Lambda = Kn*(nY-1);
    C = Kappa*Lambda;
    tstep = 100;
    Re = gsingle(10);
    Pe = gsingle(100);
    Ulattice = gsingle(0.02);
    Pr = Pe/Re;
    Viscosity=Ulattice*(nY-1)/Re;
    alpha = Viscosity/Pr;
    omega = 1/(3*Viscosity+0.5);
    omegat = 1/(3*alpha+0.5);
    oneMinusOmegat = 1-omegat;
    oneMinusOmega = 1-omega;
    rho = gzeros(nX,nY);
    f1 = rho;f2 = rho;f3 = rho;f4 = rho;f5 = rho;f6 = rho;f7 = rho;f8 = rho;f9 = rho;
    u = rho; v=rho;
```

```
    T = gzeros(nX,nY);
    g1 = rho;g2 = rho;g3 = rho;g4 = rho;g5 = rho;g6 = rho;g7 = rho;g8 = rho;g9 = rho;
    rho = rho+5;
    tic
    for kk=1:tstep
%% Momentum

        % Collision
        [f1,f2,f3,f4,f5,f6,f7,f8,f9] = CollisionD2Q9(u,v,rho,f1,f2,f3,f4,f5,f6,f7,f8,f9,omega,oneMinusOmega);

        % Streaming
        [f1,f2,f3,f4,f5,f6,f7,f8] = StreamingD2Q9(f1,f2,f3,f4,f5,f6,f7,f8,nX,nY);

        % Boundary Conditions
        [f1(1,:),f2(1,:),f4(1,:),f5(1,:),f6(1,:),f7(1,:),f8(1,:)] = InletconstantVelocityD2Q9(f2(1,:),f3(1,:),f4(1,:),f6(1,:),f7(1,:),f9(1,:)
        [f3(nX,:),f6(nX,:),f7(nX,:)] = OutletConstantVelocity( f1(nX,:),f2(nX,:),f4(nX,:),f5(nX,:),f8(nX,:),f9(nX,:),u(nX-100,:));
        [f2(:,1),f5(:,1),f6(:,1)] = SlipBottomWallD2Q9(f1(:,1),f3(:,1),f4(:,1),f7(:,1),f8(:,1),f9(:,1),u(:,2:3),Kn,nY);
        [f4(:,nY),f7(:,nY),f8(:,nY)] = SlipTopWallD2Q9(f1(:,nY),f2(:,nY),f3(:,nY),f5(:,nY),f6(:,nY),f9(:,nY),u(:,nY-2:nY-1),Kn,nY);

        % Macroscopic Values
        [u,v,rho] = MacroscopicD2Q9(f1,f2,f3,f4,f5,f6,f7,f8,f9);

%% Heat Diffiusion

        % Collision
        [g1,g2,g3,g4,g5,g6,g7,g8,g9] = CollisionTD2Q9(u,v,T,g1,g2,g3,g4,g5,g6,g7,g8,g9,omegat,oneMinusOmegat);

        % Streaming
        [g1,g2,g3,g4,g5,g6,g7,g8] = StreamingD2Q9(g1,g2,g3,g4,g5,g6,g7,g8,nX,nY);

        % Boundary Conditions for Temperature
        g1(1,2:nY-1)=2/9-g3(1,2:nY-1);
        g5(1,2:nY-1)=1/18-g7(1,2:nY-1);
        g8(1,2:nY-1)=1/18-g6(1,2:nY-1);
```

```
        % Outlet zero-flux (Extrapolation)
        g3(nX,2:nY-1)=2*g3(nX-1,2:nY-1)-g3(nX-2,2:nY-1);
        g6(nX,2:nY-1)=2*g6(nX-1,2:nY-1)-g6(nX-2,2:nY-1);
        g7(nX,2:nY-1)=2*g7(nX-1,2:nY-1)-g7(nX-2,2:nY-1);

        %Top & Bottom Walls
        [g4(:,nY),g7(:,nY),g8(:,nY)] = TJumpTop(g2(:,nY),g5(:,nY),g6(:,nY),T(:,nY-2:nY-1),0,C);
        [g2(:,1),g5(:,1),g6(:,1)] = TJumpBottom(g4(:,1),g7(:,1),g8(:,1),T(:,2:3),0,C);  % 0 = Wall teperature

        % Macroscopic
        T = MacroscopicTD2Q9(g1,g2,g3,g4,g5,g6,g7,g8,g9);
    end
time = toc;
nXC = double(nX);
nYC = double(nY);
fprintf('GPU: time = %6.4f for %d X %d\n',time,nXC,nYC);
end
%%%%%%%% End of Main Function %%%%%%%%


% All the Subroutines are the same as the CPU code.
```

A.3 MGPU Code

```
function time = LBM_MGPU(Y)
% Y is the Horizontal Node Number
%  addpath /usr/local/jacket/engine
% addpath C:\Progra~1\AccelerEyes\Jacket\engine

% gactivate
X = Y*4;
x = X/4;
y = Y;
tstep = 100;
Kn = single(0);
Kappa = 1.67;
Lambda = Kn*(Y-1);
C = Kappa*Lambda;
Re = 10;
Pe = 100;
Pr = Pe/Re;
Viscosity=0.02*(Y-1)/Re;
alpha = Viscosity/Pr;
omega = 1/(3*Viscosity+0.5);
oneMinusOmega = 1-omega;
omegat = 1/(3*alpha+0.5);
oneMinusOmegat = 1-omegat;


gselect(1)
g1u = gzeros(x+1,y,'single');
g1v = g1u;
g1rho = g1u+5;
g1f1 = g1u;
g1f2 = g1u;
```

```
g1f3 = g1u;
g1f4 = g1u;
g1f5 = g1u;
g1f6 = g1u;
g1f7 = g1u;
g1f8 = g1u;
g1f9 = g1u;

g1T = g1u;
g1g1 = g1u;
g1g2 = g1u;
g1g3 = g1u;
g1g4 = g1u;
g1g5 = g1u;
g1g6 = g1u;
g1g7 = g1u;
g1g8 = g1u;
g1g9 = g1u;


gselect(2)
g2u = gzeros(x+2,y,'single');
g2v = g2u;
g2rho = g2u+5;
g2f1 = g2u;
g2f2 = g2u;
g2f3 = g2u;
g2f4 = g2u;
g2f5 = g2u;
g2f6 = g2u;
g2f7 = g2u;
g2f8 = g2u;
g2f9 = g2u;
```

```
g2T = g2u;
g2g1 = g2u;
g2g2 = g2u;
g2g3 = g2u;
g2g4 = g2u;
g2g5 = g2u;
g2g6 = g2u;
g2g7 = g2u;
g2g8 = g2u;
g2g9 = g2u;

gselect(3)
g3u = gzeros(x+2,y,'single');
g3v = g3u;
g3rho = g3u+5;
g3f1 = g3u;
g3f2 = g3u;
g3f3 = g3u;
g3f4 = g3u;
g3f5 = g3u;
g3f6 = g3u;
g3f7 = g3u;
g3f8 = g3u;
g3f9 = g3u;

g3T = g3u;
g3g1 = g3u;
g3g2 = g3u;
g3g3 = g3u;
g3g4 = g3u;
g3g5 = g3u;
g3g6 = g3u;
g3g7 = g3u;
g3g8 = g3u;
```

```
g3g9 = g3u;

gselect(4)
g4u = gzeros(x+1,y,'single');
g4v = g4u;
g4rho = g4u+5;
g4f1 = g4u;
g4f2 = g4u;
g4f3 = g4u;
g4f4 = g4u;
g4f5 = g4u;
g4f6 = g4u;
g4f7 = g4u;
g4f8 = g4u;
g4f9 = g4u;

g4T = g4u;
g4g1 = g4u;
g4g2 = g4u;
g4g3 = g4u;
g4g4 = g4u;
g4g5 = g4u;
g4g6 = g4u;
g4g7 = g4u;
g4g8 = g4u;
g4g9 = g4u;
gsync('all');
tic

for kk = 1: tstep % Time Loop

% Collision
    gselect(1)
    g1t10 = g1u.*g1u + g1v.*g1v;
```

```
g1t10 = 1.5*g1t10;
g1t1 =  g1u;
g1t2 =  g1v;
g1t3 = -g1u;
g1t4 = -g1v;
g1t5 =  g1u + g1v;
g1t6 = -g1u + g1v;
g1t7 = -g1u - g1v;
g1t8 =  g1u - g1v;
g1feq1 = g1rho/9 .*  (1 + 3*g1t1 + 4.5*g1t1.*g1t1 - g1t10);
g1feq2 = g1rho/9 .*  (1 + 3*g1t2 + 4.5*g1t2.*g1t2 - g1t10);
g1feq3 = g1rho/9 .*  (1 + 3*g1t3 + 4.5*g1t3.*g1t3 - g1t10);
g1feq4 = g1rho/9 .*  (1 + 3*g1t4 + 4.5*g1t4.*g1t4 - g1t10);
g1feq5 = g1rho/36 .* (1 + 3*g1t5 + 4.5*g1t5.*g1t5 - g1t10);
g1feq6 = g1rho/36 .* (1 + 3*g1t6 + 4.5*g1t6.*g1t6 - g1t10);
g1feq7 = g1rho/36 .* (1 + 3*g1t7 + 4.5*g1t7.*g1t7 - g1t10);
g1feq8 = g1rho/36 .* (1 + 3*g1t8 + 4.5*g1t8.*g1t8 - g1t10);
g1feq9 = 4*g1rho/9 .*(1 - g1t10);
g1f1 = omega*g1feq1 + oneMinusOmega*g1f1;
g1f2 = omega*g1feq2 + oneMinusOmega*g1f2;
g1f3 = omega*g1feq3 + oneMinusOmega*g1f3;
g1f4 = omega*g1feq4 + oneMinusOmega*g1f4;
g1f5 = omega*g1feq5 + oneMinusOmega*g1f5;
g1f6 = omega*g1feq6 + oneMinusOmega*g1f6;
g1f7 = omega*g1feq7 + oneMinusOmega*g1f7;
g1f8 = omega*g1feq8 + oneMinusOmega*g1f8;
g1f9 = omega*g1feq9 + oneMinusOmega*g1f9;
gselect(2)
g2t10 = g2u.*g2u +g2v.*g2v;
g2t10 = 1.5*g2t10;
g2t1 =  g2u;
g2t2 =  g2v;
g2t3 = -g2u;
g2t4 = -g2v;
```

```
g2t5 =  g2u + g2v;
g2t6 = -g2u + g2v;
g2t7 = -g2u - g2v;
g2t8 =  g2u - g2v;
g2feq1 = g2rho/9 .*  (1 + 3*g2t1 + 4.5*g2t1.*g2t1 - g2t10);
g2feq2 = g2rho/9 .*  (1 + 3*g2t2 + 4.5*g2t2.*g2t2 - g2t10);
g2feq3 = g2rho/9 .*  (1 + 3*g2t3 + 4.5*g2t3.*g2t3 - g2t10);
g2feq4 = g2rho/9 .*  (1 + 3*g2t4 + 4.5*g2t4.*g2t4 - g2t10);
g2feq5 = g2rho/36 .* (1 + 3*g2t5 + 4.5*g2t5.*g2t5 - g2t10);
g2feq6 = g2rho/36 .* (1 + 3*g2t6 + 4.5*g2t6.*g2t6 - g2t10);
g2feq7 = g2rho/36 .* (1 + 3*g2t7 + 4.5*g2t7.*g2t7 - g2t10);
g2feq8 = g2rho/36 .* (1 + 3*g2t8 + 4.5*g2t8.*g2t8 - g2t10);
g2feq9 = 4*g2rho/9 .*(1 - g2t10);
g2f1 = omega*g2feq1 + oneMinusOmega*g2f1;
g2f2 = omega*g2feq2 + oneMinusOmega*g2f2;
g2f3 = omega*g2feq3 + oneMinusOmega*g2f3;
g2f4 = omega*g2feq4 + oneMinusOmega*g2f4;
g2f5 = omega*g2feq5 + oneMinusOmega*g2f5;
g2f6 = omega*g2feq6 + oneMinusOmega*g2f6;
g2f7 = omega*g2feq7 + oneMinusOmega*g2f7;
g2f8 = omega*g2feq8 + oneMinusOmega*g2f8;
g2f9 = omega*g2feq9 + oneMinusOmega*g2f9;
gselect(3)
g3t10 = g3u.*g3u +g3v.*g3v;
g3t10 = 1.5*g3t10;
g3t1 =  g3u;
g3t2 =  g3v;
g3t3 = -g3u;
g3t4 = -g3v;
g3t5 =  g3u + g3v;
g3t6 = -g3u + g3v;
g3t7 = -g3u - g3v;
g3t8 =  g3u - g3v;
g3feq1 = g3rho/9 .*  (1 + 3*g3t1 + 4.5*g3t1.*g3t1 - g3t10);
```

```
g3feq2 = g3rho/9 .*  (1 + 3*g3t2 + 4.5*g3t2.*g3t2 - g3t10);
g3feq3 = g3rho/9 .*  (1 + 3*g3t3 + 4.5*g3t3.*g3t3 - g3t10);
g3feq4 = g3rho/9 .*  (1 + 3*g3t4 + 4.5*g3t4.*g3t4 - g3t10);
g3feq5 = g3rho/36 .* (1 + 3*g3t5 + 4.5*g3t5.*g3t5 - g3t10);
g3feq6 = g3rho/36 .* (1 + 3*g3t6 + 4.5*g3t6.*g3t6 - g3t10);
g3feq7 = g3rho/36 .* (1 + 3*g3t7 + 4.5*g3t7.*g3t7 - g3t10);
g3feq8 = g3rho/36 .* (1 + 3*g3t8 + 4.5*g3t8.*g3t8 - g3t10);
g3feq9 = 4*g3rho/9 .*(1 - g3t10);
g3f1 = omega*g3feq1 + oneMinusOmega*g3f1;
g3f2 = omega*g3feq2 + oneMinusOmega*g3f2;
g3f3 = omega*g3feq3 + oneMinusOmega*g3f3;
g3f4 = omega*g3feq4 + oneMinusOmega*g3f4;
g3f5 = omega*g3feq5 + oneMinusOmega*g3f5;
g3f6 = omega*g3feq6 + oneMinusOmega*g3f6;
g3f7 = omega*g3feq7 + oneMinusOmega*g3f7;
g3f8 = omega*g3feq8 + oneMinusOmega*g3f8;
g3f9 = omega*g3feq9 + oneMinusOmega*g3f9;
gselect(4)
g4t10 = g4u.*g4u +g4v.*g4v;
g4t10 = 1.5*g4t10;
g4t1 =  g4u;
g4t2 =  g4v;
g4t3 = -g4u;
g4t4 = -g4v;
g4t5 =  g4u + g4v;
g4t6 = -g4u + g4v;
g4t7 = -g4u - g4v;
g4t8 =  g4u - g4v;
g4feq1 = g4rho/9 .*  (1 + 3*g4t1 + 4.5*g4t1.*g4t1 - g4t10);
g4feq2 = g4rho/9 .*  (1 + 3*g4t2 + 4.5*g4t2.*g4t2 - g4t10);
g4feq3 = g4rho/9 .*  (1 + 3*g4t3 + 4.5*g4t3.*g4t3 - g4t10);
g4feq4 = g4rho/9 .*  (1 + 3*g4t4 + 4.5*g4t4.*g4t4 - g4t10);
g4feq5 = g4rho/36 .* (1 + 3*g4t5 + 4.5*g4t5.*g4t5 - g4t10);
g4feq6 = g4rho/36 .* (1 + 3*g4t6 + 4.5*g4t6.*g4t6 - g4t10);
```

```
    g4feq7 = g4rho/36 .* (1 + 3*g4t7 + 4.5*g4t7.*g4t7 - g4t10);
    g4feq8 = g4rho/36 .* (1 + 3*g4t8 + 4.5*g4t8.*g4t8 - g4t10);
    g4feq9 = 4*g4rho/9 .*(1 - g4t10);
    g4f1 = omega*g4feq1 + oneMinusOmega*g4f1;
    g4f2 = omega*g4feq2 + oneMinusOmega*g4f2;
    g4f3 = omega*g4feq3 + oneMinusOmega*g4f3;
    g4f4 = omega*g4feq4 + oneMinusOmega*g4f4;
    g4f5 = omega*g4feq5 + oneMinusOmega*g4f5;
    g4f6 = omega*g4feq6 + oneMinusOmega*g4f6;
    g4f7 = omega*g4feq7 + oneMinusOmega*g4f7;
    g4f8 = omega*g4feq8 + oneMinusOmega*g4f8;
    g4f9 = omega*g4feq9 + oneMinusOmega*g4f9;

% End of Collision

%% Streaming Ghost Node

gselect(2)
g1f1_r = double(g2f1(2,:));
g1f2_r = double(g2f2(2,:));
g1f3_r = double(g2f3(2,:));
g1f4_r = double(g2f4(2,:));
g1f5_r = double(g2f5(2,:));
g1f6_r = double(g2f6(2,:));
g1f7_r = double(g2f7(2,:));
g1f8_r = double(g2f8(2,:));
g1f9_r = double(g2f9(2,:));
g1u_r = double(g2u(2,:));

gselect(1)
g1f1(end,:)=g1f1_r;
g1f2(end,:)=g1f2_r;
g1f3(end,:)=g1f3_r;
g1f4(end,:)=g1f4_r;
```

```
g1f5(end,:)=g1f5_r;
g1f6(end,:)=g1f6_r;
g1f7(end,:)=g1f7_r;
g1f8(end,:)=g1f8_r;
g1f9(end,:)=g1f9_r;
g1u(end,:)=g1u_r;

gselect(1)
g2f1_l = double(g1f1(end-1,:));
g2f2_l = double(g1f2(end-1,:));
g2f3_l = double(g1f3(end-1,:));
g2f4_l = double(g1f4(end-1,:));
g2f5_l = double(g1f5(end-1,:));
g2f6_l = double(g1f6(end-1,:));
g2f7_l = double(g1f7(end-1,:));
g2f8_l = double(g1f8(end-1,:));
g2f9_l = double(g1f9(end-1,:));
g2u_l = double(g1u(end-1,:));

gselect(2)
g2f1(1,:) = g2f1_l;
g2f2(1,:) = g2f2_l;
g2f3(1,:) = g2f3_l;
g2f4(1,:) = g2f4_l;
g2f5(1,:) = g2f5_l;
g2f6(1,:) = g2f6_l;
g2f7(1,:) = g2f7_l;
g2f8(1,:) = g2f8_l;
g2f9(1,:) = g2f9_l;
g2u(1,:) = g2u_l;

gselect(3)
g2f1_r = double(g3f1(2,:));
g2f2_r = double(g3f2(2,:));
```

```
g2f3_r = double(g3f3(2,:));
g2f4_r = double(g3f4(2,:));
g2f5_r = double(g3f5(2,:));
g2f6_r = double(g3f6(2,:));
g2f7_r = double(g3f7(2,:));
g2f8_r = double(g3f8(2,:));
g2f9_r = double(g3f9(2,:));
g2u_r = double(g3u(2,:));

gselect(2)
g2f1(end,:) = g2f1_r;
g2f2(end,:) = g2f2_r;
g2f3(end,:) = g2f3_r;
g2f4(end,:) = g2f4_r;
g2f5(end,:) = g2f5_r;
g2f6(end,:) = g2f6_r;
g2f7(end,:) = g2f7_r;
g2f8(end,:) = g2f8_r;
g2f9(end,:) = g2f9_r;
g2u(end,:) = g2u_r;

gselect(2)
g3f1_l = double(g2f1(end-1,:));
g3f2_l = double(g2f2(end-1,:));
g3f3_l = double(g2f3(end-1,:));
g3f4_l = double(g2f4(end-1,:));
g3f5_l = double(g2f5(end-1,:));
g3f6_l = double(g2f6(end-1,:));
g3f7_l = double(g2f7(end-1,:));
g3f8_l = double(g2f8(end-1,:));
g3f9_l = double(g2f9(end-1,:));
g3u_l = double(g2u(end-1,:));

gselect(3)
```

```
g3f1(1,:) = g3f1_l;
g3f2(1,:) = g3f2_l;
g3f3(1,:) = g3f3_l;
g3f4(1,:) = g3f4_l;
g3f5(1,:) = g3f5_l;
g3f6(1,:) = g3f6_l;
g3f7(1,:) = g3f7_l;
g3f8(1,:) = g3f8_l;
g3f9(1,:) = g3f9_l;
g3u(1,:) = g3u_l;

gselect(4)
g3f1_r = double(g4f1(2,:));
g3f2_r = double(g4f2(2,:));
g3f3_r = double(g4f3(2,:));
g3f4_r = double(g4f4(2,:));
g3f5_r = double(g4f5(2,:));
g3f6_r = double(g4f6(2,:));
g3f7_r = double(g4f7(2,:));
g3f8_r = double(g4f8(2,:));
g3f9_r = double(g4f9(2,:));
g3u_r = double(g4u(2,:));

gselect(3)
g3f1(end,:) = g3f1_r;
g3f2(end,:) = g3f2_r;
g3f3(end,:) = g3f3_r;
g3f4(end,:) = g3f4_r;
g3f5(end,:) = g3f5_r;
g3f6(end,:) = g3f6_r;
g3f7(end,:) = g3f7_r;
g3f8(end,:) = g3f8_r;
g3f9(end,:) = g3f9_r;
g3u(end,:) = g3u_r;
```

```
gselect(3)
g4f1_l = double(g3f1(end-1,:));
g4f2_l = double(g3f2(end-1,:));
g4f3_l = double(g3f3(end-1,:));
g4f4_l = double(g3f4(end-1,:));
g4f5_l = double(g3f5(end-1,:));
g4f6_l = double(g3f6(end-1,:));
g4f7_l = double(g3f7(end-1,:));
g4f8_l = double(g3f8(end-1,:));
g4f9_l = double(g3f9(end-1,:));
g4u_l = double(g3u(end-1,:));

gselect(4)
g4f1(1,:) = g4f1_l;
g4f2(1,:) = g4f2_l;
g4f3(1,:) = g4f3_l;
g4f4(1,:) = g4f4_l;
g4f5(1,:) = g4f5_l;
g4f6(1,:) = g4f6_l;
g4f7(1,:) = g4f7_l;
g4f8(1,:) = g4f8_l;
g4f9(1,:) = g4f9_l;
g4u(1,:) = g4u_l;
gsync('all');

gselect(1)
[g1f1,g1f2,g1f3,g1f4,g1f5,g1f6,g1f7,g1f8] = StreamingGND2Q9(g1f1,g1f2,g1f3,g1f4,g1f5,g1f6,g1f7,g1f8,x+1,y);
[g1f1(1,2:y-1),g1f2(1,2:y-1),g1f4(1,2:y-1),g1f5(1,2:y-1),g1f6(1,2:y-1),g1f7(1,2:y-1),g1f8(1,2:y-1)] =...
     InletconstantVelocityD2Q9(g1f2(1,2:y-1),g1f3(1,2:y-1),g1f4(1,2:y-1),g1f6(1,2:y-1),g1f7(1,2:y-1),...
    g1f9(1,2:y-1),0.02); % 0.02=Ulattice
[g1f2(:,1),g1f5(:,1),g1f6(:,1)] = SlipBottomWallD2Q9(g1f1(:,1),g1f3(:,1),g1f4(:,1),g1f7(:,1),g1f8(:,1),g1f9(:,1),g1u(:,2:3),Kn,y);
[g1f4(:,y),g1f7(:,y),g1f8(:,y)] = SlipTopWallD2Q9(g1f1(:,y),g1f2(:,y),g1f3(:,y),g1f5(:,y),...
    g1f6(:,y),g1f9(:,y),g1u(:,y-2:y-1),Kn,y);
```

```
g1rho = g1f1+g1f2+g1f3+g1f4+g1f5+g1f6+g1f7+g1f8+g1f9;
g1usum =  g1f1-g1f3+g1f5-g1f6-g1f7+g1f8;
g1vsum =  g1f2-g1f4+g1f5+g1f6-g1f7-g1f8;
g1u = g1usum./g1rho;
g1v = g1vsum./g1rho;

gselect(2)
[g2f1,g2f2,g2f3,g2f4,g2f5,g2f6,g2f7,g2f8] = StreamingGND2Q9(g2f1,g2f2,g2f3,g2f4,g2f5,g2f6,g2f7,g2f8,x+2,y);
[g2f2(:,1),g2f5(:,1),g2f6(:,1)] = SlipBottomWallD2Q9(g2f1(:,1),g2f3(:,1),g2f4(:,1), g2f7(:,1),g2f8(:,1),g2f9(:,1),g2u(:,2:3),Kn,y);
[g2f4(:,y),g2f7(:,y),g2f8(:,y)] = SlipTopWallD2Q9(g2f1(:,y),g2f2(:,y),g2f3(:,y), g2f5(:,y),g2f6(:,y),g2f9(:,y),g2u(:,y-2:y-1),Kn,y);
g2rho = g2f1+g2f2+g2f3+g2f4+g2f5+g2f6+g2f7+g2f8+g2f9;
g2usum =  g2f1-g2f3+g2f5-g2f6-g2f7+g2f8;
g2vsum =  g2f2-g2f4+g2f5+g2f6-g2f7-g2f8;
g2u = g2usum./g2rho;
g2v = g2vsum./g2rho;

gselect(3)
[g3f1,g3f2,g3f3,g3f4,g3f5,g3f6,g3f7,g3f8] = StreamingGND2Q9(g3f1,g3f2,g3f3,g3f4,g3f5,g3f6,g3f7,g3f8,x+2,y);
[g3f2(:,1),g3f5(:,1),g3f6(:,1)] = SlipBottomWallD2Q9(g3f1(:,1),g3f3(:,1),g3f4(:,1),...
    g3f7(:,1),g3f8(:,1),g3f9(:,1),g3u(:,2:3),Kn,y);
[g3f4(:,y),g3f7(:,y),g3f8(:,y)] = SlipTopWallD2Q9(g3f1(:,y),g3f2(:,y),g3f3(:,y),g3f5(:,y),g3f6(:,y),g3f9(:,y),g3u(:,y-2:y-1),Kn,y);
g3rho = g3f1+g3f2+g3f3+g3f4+g3f5+g3f6+g3f7+g3f8+g3f9;
g3usum =  g3f1-g3f3+g3f5-g3f6-g3f7+g3f8;
g3vsum =  g3f2-g3f4+g3f5+g3f6-g3f7-g3f8;
g3u = g3usum./g3rho;
g3v = g3vsum./g3rho;

gselect(4)
[g4f1,g4f2,g4f3,g4f4,g4f5,g4f6,g4f7,g4f8] = StreamingGND2Q9(g4f1,g4f2,g4f3,g4f4,g4f5,g4f6,g4f7,g4f8,x+1,y);
[g4f3(x,:),g4f6(x,:),g4f7(x,:)] = OutletConstantVelocity( g4f1(x,:),g4f2(x,:),g4f4(x,:),g4f5(x,:),g4f8(x,:),g4f9(x,:),g4u(x-5,:));
[g4f2(:,1),g4f5(:,1),g4f6(:,1)] = SlipBottomWallD2Q9(g4f1(:,1),g4f3(:,1),g4f4(:,1),g4f7(:,1),g4f8(:,1),g4f9(:,1),g4u(:,2:3),Kn,y);
[g4f4(:,y),g4f7(:,y),g4f8(:,y)] = SlipTopWallD2Q9(g4f1(:,y),g4f2(:,y),g4f3(:,y), g4f5(:,y),g4f6(:,y),g4f9(:,y),g4u(:,y-2:y-1),Kn,y);
g4rho = g4f1+g4f2+g4f3+g4f4+g4f5+g4f6+g4f7+g4f8+g4f9;
g4usum =  g4f1-g4f3+g4f5-g4f6-g4f7+g4f8;
```

```
g4vsum =  g4f2-g4f4+g4f5+g4f6-g4f7-g4f8;
g4u = g4usum./g4rho;
g4v = g4vsum./g4rho;

gsync('all');



%% Heat Transfer
% Collision
    gselect(1)
    g1t1 =  g1u;
    g1t2 =  g1v;
    g1t3 = -g1u;
    g1t4 = -g1v;
    g1t5 =  g1u + g1v;
    g1t6 = -g1u + g1v;
    g1t7 = -g1u - g1v;
    g1t8 =  g1u - g1v;
    g1feq1 = g1T/9 .*  (1 + 3*g1t1 );
    g1feq2 = g1T/9 .*  (1 + 3*g1t2 );
    g1feq3 = g1T/9 .*  (1 + 3*g1t3 );
    g1feq4 = g1T/9 .*  (1 + 3*g1t4 );
    g1feq5 = g1T/36 .* (1 + 3*g1t5 );
    g1feq6 = g1T/36 .* (1 + 3*g1t6 );
    g1feq7 = g1T/36 .* (1 + 3*g1t7 );
    g1feq8 = g1T/36 .* (1 + 3*g1t8 );
    g1feq9 = 4*g1T/9 ;
    g1g1 = omegat*g1feq1 + oneMinusOmegat*g1g1;
    g1g2 = omegat*g1feq2 + oneMinusOmegat*g1g2;
    g1g3 = omegat*g1feq3 + oneMinusOmegat*g1g3;
    g1g4 = omegat*g1feq4 + oneMinusOmegat*g1g4;
    g1g5 = omegat*g1feq5 + oneMinusOmegat*g1g5;
    g1g6 = omegat*g1feq6 + oneMinusOmegat*g1g6;
```

```
g1g7 = omegat*g1feq7 + oneMinusOmegat*g1g7;
g1g8 = omegat*g1feq8 + oneMinusOmegat*g1g8;
g1g9 = omegat*g1feq9 + oneMinusOmegat*g1g9;
gselect(2)
g2t1 =  g2u;
g2t2 =  g2v;
g2t3 = -g2u;
g2t4 = -g2v;
g2t5 =  g2u + g2v;
g2t6 = -g2u + g2v;
g2t7 = -g2u - g2v;
g2t8 =  g2u - g2v;
g2feq1 = g2T/9 .*  (1 + 3*g2t1);
g2feq2 = g2T/9 .*  (1 + 3*g2t2);
g2feq3 = g2T/9 .*  (1 + 3*g2t3);
g2feq4 = g2T/9 .*  (1 + 3*g2t4);
g2feq5 = g2T/36 .* (1 + 3*g2t5);
g2feq6 = g2T/36 .* (1 + 3*g2t6);
g2feq7 = g2T/36 .* (1 + 3*g2t7);
g2feq8 = g2T/36 .* (1 + 3*g2t8);
g2feq9 = 4*g2T/9;
g2g1 = omegat*g2feq1 + oneMinusOmegat*g2g1;
g2g2 = omegat*g2feq2 + oneMinusOmegat*g2g2;
g2g3 = omegat*g2feq3 + oneMinusOmegat*g2g3;
g2g4 = omegat*g2feq4 + oneMinusOmegat*g2g4;
g2g5 = omegat*g2feq5 + oneMinusOmegat*g2g5;
g2g6 = omegat*g2feq6 + oneMinusOmegat*g2g6;
g2g7 = omegat*g2feq7 + oneMinusOmegat*g2g7;
g2g8 = omegat*g2feq8 + oneMinusOmegat*g2g8;
g2g9 = omegat*g2feq9 + oneMinusOmegat*g2g9;
gselect(3)
g3t1 =  g3u;
g3t2 =  g3v;
g3t3 = -g3u;
```

```
g3t4 = -g3v;
g3t5 =  g3u + g3v;
g3t6 = -g3u + g3v;
g3t7 = -g3u - g3v;
g3t8 =  g3u - g3v;
g3feq1 = g3T/9 .*  (1 + 3*g3t1);
g3feq2 = g3T/9 .*  (1 + 3*g3t2);
g3feq3 = g3T/9 .*  (1 + 3*g3t3);
g3feq4 = g3T/9 .*  (1 + 3*g3t4);
g3feq5 = g3T/36 .* (1 + 3*g3t5);
g3feq6 = g3T/36 .* (1 + 3*g3t6);
g3feq7 = g3T/36 .* (1 + 3*g3t7);
g3feq8 = g3T/36 .* (1 + 3*g3t8);
g3feq9 = 4*g3T/9;
g3g1 = omegat*g3feq1 + oneMinusOmegat*g3g1;
g3g2 = omegat*g3feq2 + oneMinusOmegat*g3g2;
g3g3 = omegat*g3feq3 + oneMinusOmegat*g3g3;
g3g4 = omegat*g3feq4 + oneMinusOmegat*g3g4;
g3g5 = omegat*g3feq5 + oneMinusOmegat*g3g5;
g3g6 = omegat*g3feq6 + oneMinusOmegat*g3g6;
g3g7 = omegat*g3feq7 + oneMinusOmegat*g3g7;
g3g8 = omegat*g3feq8 + oneMinusOmegat*g3g8;
g3g9 = omegat*g3feq9 + oneMinusOmegat*g3g9;
gselect(4)
g4t1 =  g4u;
g4t2 =  g4v;
g4t3 = -g4u;
g4t4 = -g4v;
g4t5 =  g4u + g4v;
g4t6 = -g4u + g4v;
g4t7 = -g4u - g4v;
g4t8 =  g4u - g4v;
g4feq1 = g4T/9 .*  (1 + 3*g4t1);
g4feq2 = g4T/9 .*  (1 + 3*g4t2);
```

```
    g4feq3 = g4T/9 .*  (1 + 3*g4t3);
    g4feq4 = g4T/9 .*  (1 + 3*g4t4);
    g4feq5 = g4T/36 .* (1 + 3*g4t5);
    g4feq6 = g4T/36 .* (1 + 3*g4t6);
    g4feq7 = g4T/36 .* (1 + 3*g4t7);
    g4feq8 = g4T/36 .* (1 + 3*g4t8);
    g4feq9 = 4*g4T/9;
    g4g1 = omegat*g4feq1 + oneMinusOmegat*g4g1;
    g4g2 = omegat*g4feq2 + oneMinusOmegat*g4g2;
    g4g3 = omegat*g4feq3 + oneMinusOmegat*g4g3;
    g4g4 = omegat*g4feq4 + oneMinusOmegat*g4g4;
    g4g5 = omegat*g4feq5 + oneMinusOmegat*g4g5;
    g4g6 = omegat*g4feq6 + oneMinusOmegat*g4g6;
    g4g7 = omegat*g4feq7 + oneMinusOmegat*g4g7;
    g4g8 = omegat*g4feq8 + oneMinusOmegat*g4g8;
    g4g9 = omegat*g4feq9 + oneMinusOmegat*g4g9;

% End of Collision

% Streaming Ghost Node

gselect(2)
g1g1_r = double(g2g1(2,:));
g1g2_r = double(g2g2(2,:));
g1g3_r = double(g2g3(2,:));
g1g4_r = double(g2g4(2,:));
g1g5_r = double(g2g5(2,:));
g1g6_r = double(g2g6(2,:));
g1g7_r = double(g2g7(2,:));
g1g8_r = double(g2g8(2,:));
g1g9_r = double(g2g9(2,:));
g1u_r = double(g2u(2,:));

gselect(1)
```

```
g1g1(end,:)=g1g1_r;
g1g2(end,:)=g1g2_r;
g1g3(end,:)=g1g3_r;
g1g4(end,:)=g1g4_r;
g1g5(end,:)=g1g5_r;
g1g6(end,:)=g1g6_r;
g1g7(end,:)=g1g7_r;
g1g8(end,:)=g1g8_r;
g1g9(end,:)=g1g9_r;
g1u(end,:)=g1u_r;

gselect(1)
g2g1_l = double(g1g1(end-1,:));
g2g2_l = double(g1g2(end-1,:));
g2g3_l = double(g1g3(end-1,:));
g2g4_l = double(g1g4(end-1,:));
g2g5_l = double(g1g5(end-1,:));
g2g6_l = double(g1g6(end-1,:));
g2g7_l = double(g1g7(end-1,:));
g2g8_l = double(g1g8(end-1,:));
g2g9_l = double(g1g9(end-1,:));
g2u_l = double(g1u(end-1,:));

gselect(2)
g2g1(1,:) = g2g1_l;
g2g2(1,:) = g2g2_l;
g2g3(1,:) = g2g3_l;
g2g4(1,:) = g2g4_l;
g2g5(1,:) = g2g5_l;
g2g6(1,:) = g2g6_l;
g2g7(1,:) = g2g7_l;
g2g8(1,:) = g2g8_l;
g2g9(1,:) = g2g9_l;
g2u(1,:) = g2u_l;
```

```
gselect(3)
g2g1_r = double(g3g1(2,:));
g2g2_r = double(g3g2(2,:));
g2g3_r = double(g3g3(2,:));
g2g4_r = double(g3g4(2,:));
g2g5_r = double(g3g5(2,:));
g2g6_r = double(g3g6(2,:));
g2g7_r = double(g3g7(2,:));
g2g8_r = double(g3g8(2,:));
g2g9_r = double(g3g9(2,:));
g2u_r = double(g3u(2,:));

gselect(2)
g2g1(end,:) = g2g1_r;
g2g2(end,:) = g2g2_r;
g2g3(end,:) = g2g3_r;
g2g4(end,:) = g2g4_r;
g2g5(end,:) = g2g5_r;
g2g6(end,:) = g2g6_r;
g2g7(end,:) = g2g7_r;
g2g8(end,:) = g2g8_r;
g2g9(end,:) = g2g9_r;
g2u(end,:) = g2u_r;

gselect(2)
g3g1_l = double(g2g1(end-1,:));
g3g2_l = double(g2g2(end-1,:));
g3g3_l = double(g2g3(end-1,:));
g3g4_l = double(g2g4(end-1,:));
g3g5_l = double(g2g5(end-1,:));
g3g6_l = double(g2g6(end-1,:));
g3g7_l = double(g2g7(end-1,:));
g3g8_l = double(g2g8(end-1,:));
```

```
g3g9_l = double(g2g9(end-1,:));
g3u_l = double(g2u(end-1,:));

gselect(3)
g3g1(1,:) = g3g1_l;
g3g2(1,:) = g3g2_l;
g3g3(1,:) = g3g3_l;
g3g4(1,:) = g3g4_l;
g3g5(1,:) = g3g5_l;
g3g6(1,:) = g3g6_l;
g3g7(1,:) = g3g7_l;
g3g8(1,:) = g3g8_l;
g3g9(1,:) = g3g9_l;
g3u(1,:) = g3u_l;

gselect(4)
g3g1_r = double(g4g1(2,:));
g3g2_r = double(g4g2(2,:));
g3g3_r = double(g4g3(2,:));
g3g4_r = double(g4g4(2,:));
g3g5_r = double(g4g5(2,:));
g3g6_r = double(g4g6(2,:));
g3g7_r = double(g4g7(2,:));
g3g8_r = double(g4g8(2,:));
g3g9_r = double(g4g9(2,:));
g3u_r = double(g4u(2,:));

gselect(3)
g3g1(end,:) = g3g1_r;
g3g2(end,:) = g3g2_r;
g3g3(end,:) = g3g3_r;
g3g4(end,:) = g3g4_r;
g3g5(end,:) = g3g5_r;
g3g6(end,:) = g3g6_r;
```

```
g3g7(end,:) = g3g7_r;
g3g8(end,:) = g3g8_r;
g3g9(end,:) = g3g9_r;
g3u(end,:) = g3u_r;

% GPU4
gselect(3)
g4g1_l = double(g3g1(end-1,:));
g4g2_l = double(g3g2(end-1,:));
g4g3_l = double(g3g3(end-1,:));
g4g4_l = double(g3g4(end-1,:));
g4g5_l = double(g3g5(end-1,:));
g4g6_l = double(g3g6(end-1,:));
g4g7_l = double(g3g7(end-1,:));
g4g8_l = double(g3g8(end-1,:));
g4g9_l = double(g3g9(end-1,:));
g4u_l = double(g3u(end-1,:));

gselect(4)
g4g1(1,:) = g4g1_l;
g4g2(1,:) = g4g2_l;
g4g3(1,:) = g4g3_l;
g4g4(1,:) = g4g4_l;
g4g5(1,:) = g4g5_l;
g4g6(1,:) = g4g6_l;
g4g7(1,:) = g4g7_l;
g4g8(1,:) = g4g8_l;
g4g9(1,:) = g4g9_l;
g4u(1,:) = g4u_l;
gsync('all');

% Streaming, Boundary Conditions and Macroscopic
gselect(1)
[g1g1,g1g2,g1g3,g1g4,g1g5,g1g6,g1g7,g1g8] = StreamingGND2Q9(g1g1,g1g2,g1g3,g1g4,g1g5,g1g6,g1g7,g1g8,x+1,y);
```

```
g1g1(1,2:y-1)=1*2/9-g1g3(1,2:y-1); % 1 = Inlet Temperature
g1g5(1,2:y-1)=1/18-g1g7(1,2:y-1);  % 1 = Inlet Temperature
g1g8(1,2:y-1)=1/18-g1g6(1,2:y-1);  % 1 = Inlet Temperature
[g1g2(:,1),g1g5(:,1),g1g6(:,1)] = TJumpBottom(g1g4(:,1),g1g7(:,1),g1g8(:,1),g1T(:,2:3),0,C);  % 0 = Wall teperature
[g1g4(:,y),g1g7(:,y),g1g8(:,y)] = TJumpTop(g1g2(:,y),g1g5(:,y),g1g6(:,y),g1T(:,y-2:y-1),0,C); % 0 = Wall teperature
g1T = g1g1+g1g2+g1g3+g1g4+g1g5+g1g6+g1g7+g1g8+g1g9;
gselect(2)
[g2g1,g2g2,g2g3,g2g4,g2g5,g2g6,g2g7,g2g8] = StreamingGND2Q9(g2g1,g2g2,g2g3,g2g4,g2g5,g2g6,g2g7,g2g8,x+2,y);
[g2g2(:,1),g2g5(:,1),g2g6(:,1)] = TJumpBottom(g2g4(:,1),g2g7(:,1),g2g8(:,1),g2T(:,2:3),0,C);  % 0 = Wall teperature
[g2g4(:,y),g2g7(:,y),g2g8(:,y)] = TJumpTop(g2g2(:,y),g2g5(:,y),g2g6(:,y),g2T(:,y-2:y-1),0,C);
g2T = g2g1+g2g2+g2g3+g2g4+g2g5+g2g6+g2g7+g2g8+g2g9;

gselect(3)
[g3g1,g3g2,g3g3,g3g4,g3g5,g3g6,g3g7,g3g8] = StreamingGND2Q9(g3g1,g3g2,g3g3,g3g4,g3g5,g3g6,g3g7,g3g8,x+2,y);
[g3g2(:,1),g3g5(:,1),g3g6(:,1)] = TJumpBottom(g3g4(:,1),g3g7(:,1),g3g8(:,1),g3T(:,2:3),0,C);  % 0 = Wall teperature
[g3g4(:,y),g3g7(:,y),g3g8(:,y)] = TJumpTop(g3g2(:,y),g3g5(:,y),g3g6(:,y),g3T(:,y-2:y-1),0,C);
g3T = g3g1+g3g2+g3g3+g3g4+g3g5+g3g6+g3g7+g3g8+g3g9;

gselect(4)
[g4g1,g4g2,g4g3,g4g4,g4g5,g4g6,g4g7,g4g8] = StreamingGND2Q9(g4g1,g4g2,g4g3,g4g4,g4g5,g4g6,g4g7,g4g8,x+1,y);
% Outlet zero-flux (Extrapolation)
g4g3(x+1,2:y-1)=2*g4g3(x,2:y-1)-g4g3(x-1,2:y-1);
g4g6(x+1,2:y-1)=2*g4g6(x,2:y-1)-g4g6(x-1,2:y-1);
g4g7(x+1,2:y-1)=2*g4g7(x,2:y-1)-g4g7(x-1,2:y-1);
[g4g2(:,1),g4g5(:,1),g4g6(:,1)] = TJumpBottom(g4g4(:,1),g4g7(:,1),g4g8(:,1),g4T(:,2:3),0,C);  % 0 = Wall teperature
[g4g4(:,y),g4g7(:,y),g4g8(:,y)] = TJumpTop(g4g2(:,y),g4g5(:,y),g4g6(:,y),g4T(:,y-2:y-1),0,C);
g4T = g4g1+g4g2+g4g3+g4g4+g4g5+g4g6+g4g7+g4g8+g4g9;
% End of Streaming, Boundary Conditions and Macroscopic

% gsync('all');

%% End of Heat Trasnfer

end % enTime Loop
```

```
time = toc;
fprintf('MGPU: time = %6.4f for %d X %d\n',time,X,Y);

end

%%% End of Main Function %%%

function [f1a,f2a,f3a,f4a,f5a,f6a,f7a,f8a] = StreamingGND2Q9(f1,f2,f3,f4,f5,f6,f7,f8,X,Y)
    f1a = f1([1,1:X-1],1:Y);
    f2a = f2(1:X,[1,1:Y-1]);
    f3a = f3([2:X,X],1:Y);
    f4a = f4(1:X,[2:Y,Y]);
    f5a = f5([1,1:X-1],[1,1:Y-1]);
    f6a = f6([2:X,X],[1,1:Y-1]);
    f7a = f7([2:X,X],[2:Y,Y]);
    f8a = f8([1,1:X-1],[2:Y,Y]);
end

function [ f3,f6,f7] = OutletConstantVelocity( f1,f2,f4,f5,f8,f9,u_out )
rho_o = (f9+f2+f4+2*(f1+f5+f8))./(1.0+u_out);
    f3 =f1 -0.667*rho_o.*u_out;
f7 =f5 +0.5*(f2 -f4 )- rho_o.*u_out/6.0;
f6 =f8 +0.5*(f4 -f2 )- rho_o.*u_out/6.0;
end

function [f2,f5,f6] = SlipBottomWallD2Q9(f1,f3,f4,f7,f8,f9,u,Kn,Y)
    Lamda = Kn*(Y-1);
    uslip = Lamda*(4*u(:,1)-u(:,2))/(2+3*Lamda);
    rhow = (f1+f3+f9+2*(f4+f7+f8));
    f2 = f4;
    f5 = rhow.*(1+uslip)/2 - (f1+f8) - (f2+f4+f9)/2;
    f6 = rhow.*(1-uslip)/2 - (f3+f7) - (f2+f4+f9)/2;
end
```

```
function [f4,f7,f8] = SlipTopWallD2Q9(f1,f2,f3,f5,f6,f9,u,Kn,Y)
    Lamda = Kn*(Y-1);
    uslip = Lamda*(4*u(:,2)-u(:,1))/(2+3*Lamda);
    rhow = (f1+f3+f9+2*(f2+f5+f6));
    f4 = f2;
    f7 = rhow.*(1-uslip)/2 - (f3+f6) - (f2+f4+f9)/2;
    f8 = rhow.*(1+uslip)/2 - (f1+f5) - (f2+f4+f9)/2;
end

function [f1,f2,f4,f5,f6,f7,f8] =...
     InletconstantVelocityD2Q9(f2,f3,f4,f6,f7,f9,Ulattice)
    rhow = (f9 +f2 +f4 +2*(f3 +f6 +f7 ))/(1-Ulattice);
    f1  = f3  + 2*rhow*Ulattice/3;
    f5  = f7  + 0.5*(f4-f2) + rhow*Ulattice/6;
    f8  = f6  + 0.5*(f2-f4) + rhow*Ulattice/6;
end

function [g4,g7,g8] = TJumpTop(g2,g5,g6,T,tw,C)
    tw = (C*(4*T(:,2)-T(:,1))+2*tw)/(2+3*C);
    g8=tw/18-g6;
    g7=tw/18-g5;
    g4=tw*2/9-g2;
end

function [g2,g5,g6] = TJumpBottom(g4,g7,g8,T,Tw,C)
    tw = (C*(4*T(:,1)-T(:,2))+2*Tw)/(2+3*C);
    g6=tw/18-g8;
    g5=tw/18-g7;
    g2=2*tw/9-g4;
end
```

Appendix B

Implementation of Boundary Conditions

B.1 Macro Channel Boundary Conditions

B.1.1 Inlet Velocity Boundary Condition (Left)

$$\rho = \frac{f_2 + f_4 + f_9 + 2(f_3 + f_6 + f_7)}{1 - v_x} \tag{B.1}$$

$$f_1 = f_3 + \frac{2}{3}\rho v_x \tag{B.2}$$

$$f_5 = f_7 + \frac{1}{2}(f_4 - f_2) + \frac{\rho v_x}{6} \tag{B.3}$$

$$f_8 = f_6 - \frac{1}{2}(f_4 - f_2) + \frac{\rho v_x}{6} \tag{B.4}$$

B.1.2 No-slip Boundary Condition (Bottom)

$$f_2 = f_4 \tag{B.5}$$

$$f_5 = f_7 - \frac{f_1 - f_3}{2} \tag{B.6}$$

$$f_6 = f_8 + \frac{f_1 - f_3}{2} \tag{B.7}$$

B.1.3 No-slip Boundary Condition (Top)

$$f_4 = f_2 \tag{B.8}$$

$$f_7 = f_5 + \frac{f_1 - f_3}{2} \tag{B.9}$$

$$f_8 = f_6 - \frac{f_1 - f_3}{2} \tag{B.10}$$

B.1.4 Outlet Boundary Condition (Right)

$$\rho = \frac{f_2 + f_4 + f_9 + 2(f_1 + f_5 + f_8)}{1 - v_x} \tag{B.11}$$

$$f_3 = f_1 + \frac{2}{3}\rho v_x \tag{B.12}$$

$$f_6 = f_8 + \frac{1}{2}(f_4 - f_2) - \frac{\rho v_x}{6} \tag{B.13}$$

$$f_7 = f_5 - \frac{1}{2}(f_4 - f_2) - \frac{\rho v_x}{6} \tag{B.14}$$

B.1.5 Temperature Boundary Condition (Left)

$$g_1 = \phi\,(w(1) + w(3)) - g_3 \tag{B.15}$$

$$g_5 = \phi\,(w(5) + w(7)) - g_7 \tag{B.16}$$

$$g_8 = \phi\,(w(8) + w(6)) - g_6 \tag{B.17}$$

B.1.6 Temperature Boundary Condition (Bottom)

$$g_2 = \phi\,(w(2) + w(4)) - g_4 \tag{B.18}$$

$$g_5 = \phi\,(w(5) + w(7)) - g_7 \tag{B.19}$$

$$g_6 = \phi\,(w(6) + w(8)) - g_8 \tag{B.20}$$

B.1.7 Temperature Boundary Condition (Top)

$$g_4 = \phi\,(w(2) + w(4)) - g_2 \tag{B.21}$$

$$g_7 = \phi\,(w(5) + w(7)) - g_5 \tag{B.22}$$

$$g_8 = \phi\,(w(6) + w(8)) - g_6 \tag{B.23}$$

B.1.8 Zero Flux Temperature Boundary Condition (Right)

$$g_{3,X} = 2g_{3,X-1} - g_{3,X-2} \tag{B.24}$$

$$g_{6,X} = 2g_{6,X-1} - g_{6,X-2} \tag{B.25}$$

$$g_{7,X} = 2g_{7,X-1} - g_{7,X-2} \tag{B.26}$$

(B.27)

B.2 Micro Channel Boundary Conditions

B.2.1 Slip Boundary Conditions (Bottom)

$$\lambda = Kn * H \tag{B.28}$$

$$v_x = \lambda \frac{(4v_{x,1} - v_{x,2})}{2 + 3\lambda} \tag{B.29}$$

$$\rho_w = f_1 + f_3 + f_9 + 2(f_7 + f_4 + f_8) \tag{B.30}$$

$$f_2 = f_4 \tag{B.31}$$

$$f_5 = \frac{\rho_w(1 + v_x) - (f_2 + f_4 + f_9)}{2} - (f_1 + f_8) \tag{B.32}$$

$$f_6 = \frac{\rho_w(1 - v_x) - (f_2 + f_4 + f_9)}{2} - (f_3 + f_7) \tag{B.33}$$

B.2.2 Slip Boundary Conditions (Top Wall)

$$\lambda = Kn * H \tag{B.34}$$

$$v_x = \lambda \frac{(4v_{x,1} - v_{x,2})}{2 + 3\lambda} \tag{B.35}$$

$$\rho_w = f_1 + f_3 + f_9 + 2(f_7 + f_4 + f_8) \tag{B.36}$$

$$f_4 = f_2 \tag{B.37}$$

$$f_7 = \frac{\rho_w(1 - v_x) - (f_2 + f_4 + f_9)}{2} - (f_3 + f_6) \tag{B.38}$$

$$f_8 = \frac{\rho_w(1 + v_x) - (f_2 + f_4 + f_9)}{2} - (f_1 + f_5) \tag{B.39}$$

B.2.3 Temperature Jump Boundary Conditions (Bottom)

$$\lambda = Kn * H \tag{B.40}$$

$$C_j = \kappa\lambda \tag{B.41}$$

$$\phi_0 = \frac{\left[C_j(4\phi_1 - \phi_2) + 2\phi_w\right]}{(2 + 3C_j)} \tag{B.42}$$

$$g_2 = \phi_0\,(w(2) + w(4)) - g_4 \tag{B.43}$$

$$g_5 = \phi_0\,(w(5) + w(7)) - g_7 \tag{B.44}$$

$$g_6 = \phi_0\,(w(6) + w(8)) - g_8 \tag{B.45}$$

B.2.4 Temperature Jump Boundary Conditions (Top)

$$\lambda = Kn * H \tag{B.46}$$

$$C_j = \kappa\lambda \tag{B.47}$$

$$\phi_Y = \frac{\left[C_j(4\phi_{Y-1} - \phi_{Y-2}) + 2\phi_w\right]}{(2 + 3C_j)} \tag{B.48}$$

$$g_4 = \phi_Y\,(w(2) + w(4)) - g_2 \tag{B.49}$$

$$g_7 = \phi_Y\,(w(5) + w(7)) - g_5 \tag{B.50}$$

$$g_8 = \phi_Y\,(w(6) + w(8)) - g_6 \tag{B.51}$$

Appendix C

Pseudo Code

Algorithm C.0.1: LATTICEBOLTZMANNMETHOD()

for $time \leftarrow 1$ **to** $ntStep$

1. Momentum Transfer

1.1 Collision

for $i \leftarrow 1$ **to** nX

for $j \leftarrow 1$ **to** nY

do
$$\begin{cases} f_k^{eq}(i,j) = w_k \rho(i,j)[1 + 3(\mathbf{e}_k \cdot \mathbf{v}(i,j)) + \frac{9}{2}(\mathbf{e}_k \cdot \mathbf{v}(i,j))^2 - \frac{3}{2}|\mathbf{v}(i,j)|^2] \\ f_k(i,j) = \omega f_k^{eq}(i,j) + (1-\omega) f_k(i,j) \end{cases}$$

1.2 Streaming

See Appendix A

1.3 Implementation of Boundary Conditions

See Appendix B

1.4 Macroscopic Property Calculation

for $i \leftarrow 1$ **to** nX

for $j \leftarrow 1$ **to** nY

do
$$\begin{cases} \rho(i,j) = \sum_{k=1}^{9} f_k(i,j) \\ \mathbf{v}(i,j) = \sum_{k=1}^{9} \mathbf{e}_k f_k(i,j)/\rho(i,j) \end{cases}$$

2. Heat Transfer

2.1 Collision

for $i \leftarrow 1$ **to** nX

for $j \leftarrow 1$ **to** nY

do
$$\begin{cases} g_k^{eq}(i,j) = w_k \phi(i,j)\,[1 + 3(\mathbf{e}_k \cdot \mathbf{v}(i,j))] \\ g_k(i,j) = \omega_t g_k^{eq}(i,j) + (1-\omega_t) g_k(i,j) \end{cases}$$

2.2 Streaming

See Appendix A

2.3 Implementation of Boundary Conditions

See Appendix B

2.4 Macroscopic Property Calculation

for $i \leftarrow 1$ **to** nX

for $j \leftarrow 1$ **to** nY

do $\phi(i,j) = \sum_{k=1}^{9} g_k(i,j)$