

DL MESO USER MANUAL

R. S. Qin and W. Smith

Computational Science and Engineering Department
CCLRC Daresbury Laboratory
Warrington WA4 4AD, United Kingdom

Contents

1	DL_MESO General Information	6
1.1	Description	6
1.2	Requirements	6
1.2.1	Software requirements	6
1.2.2	System requirements	7
1.3	The DL_MESO Directory Structure	7
1.4	Disclaimer	7
1.5	Copyright	7
1.6	Authors	8
1.7	Suggestions and Bug Reports	8
1.8	Acknowledgements	8
2	The DL_MESO GUI	9
2.1	Getting Started with the DL_MESO GUI	9
2.2	Lattice Boltzmann and the DL_MESO GUI	10
2.2.1	Defining the System	10
2.2.2	Defining the Space Properties	11
2.3	Dissipative Particle dynamics and the DL_MESO GUI	11
2.3.1	Defining the System	11
2.4	Compiling and Running the LBE/DPD Program	12
2.5	Notes	13

3	The Lattice Boltzmann Equation: Basic Theory	14
3.1	Introduction	14
3.2	Basic Definitions	15
3.3	Derivation of Equilibrium	15
3.4	Structural Relaxation and Macroscopic Equations	17
3.5	Mesoscale Interaction	18
3.6	Summary of Lattice Boltzmann Equation	19
4	Dissipative Particle Dynamics: Basic Theory	20
4.1	Introduction	20
4.2	Outline of Method	21
4.3	Derivation of Equilibrium	22
4.4	Summary of Dissipative Particle Dynamics	23
5	DL_MESO_LBE Basic Definition	24
5.1	Lattice model	24
5.2	Data structure	27
5.2.1	Storage of particle distribution functions	27
5.2.2	Storage of space property	28
5.2.3	Storage of running information	29
5.3	The Parameters and their Function	32
6	DL_MESO Examples	38
6.1	The LBE Test Cases	38
6.1.1	2D_Pressure	38
6.1.2	2D_Shear	38
6.1.3	3D_PhaseSeparation	38
6.1.4	3D_Shear	38
6.2	The DPD Test Cases	39

6.2.1	Dim3_Spe2_Ple4000	39
7	The DL_MESO_LBE Package Reference	40
7.1	Overview	40
7.2	DL_MESO_LBE Subroutines and Functions	41
7.2.1	main	41
7.2.2	lbpMODEL	43
7.2.3	lbpBASIC	44
7.2.4	lbpGET	48
7.2.5	lbpIO	54
7.2.6	lbpBOUND	58
7.2.7	lbpSUB	64
7.2.8	lbpMPI	71
7.3	DL_MESO_LBE Data Files	74
7.3.1	Description of the INPUT files	74
7.3.2	Description of the OUTPUT files	75
8	The DL_MESO_DPD Package Reference	76

List of Tables

5.1	Boundary condition category	28
5.2	Letter Notation Method	29
5.3	Notation of boundary condition	30
5.4	Notation of boundary condition (continued)	31
5.5	System information	32
5.6	Domain information	33
5.7	Neighbor information	33
5.8	DL_MESO_LBE Parameters	34
5.9	DL_MESO_LBE Parameters-continued	35
5.10	DL_MESO_LBE Parameters-continued	36
5.11	DL_MESO_LBE Parameters-continued	37

Chapter 1

DL_MESO General Information

1.1 Description

DL_MESO is a general purpose mesoscopic simulation package developed at Daresbury Laboratory by Dr. Rongshan Qin under the auspices of the Engineering and Physical Sciences Research Council (EPSRC) for the EPSRC's Collaborative Computational Project for the Computer Simulation of Condensed Phases (CCP5). The package is the property of the Council for the Central Laboratory of Research Councils.

DL_MESO is issued free under licence to academic institutions pursuing scientific research of a non-commercial nature. All recipients of the code must first agree to the terms and conditions of the licence. Commercial organisations interested in acquiring the package should approach the Computational Science and Engineering Department, CCLRC Daresbury Laboratory in the first instance. Daresbury Laboratory is the sole centre for distribution of the package. Under no account is it to be redistributed to third parties without consent of the owners.

DL_MESO contains two mesoscale simulation methods:

- Lattice Boltzmann Equation (included with 1.0 version)
- Dissipative Particle Dynamics (included with 2.0 version)

1.2 Requirements

1.2.1 Software requirements

- JAVA 2 Version 1.4 above

- Standard C++ Compiler
- Message Passing Interface (if parallel execution required).

1.2.2 System requirements

DL_MESO has been tested on Solaris, Window XP and IBM p690+ HPCx machines.

1.3 The DL_MESO Directory Structure

The supplied version of DL_MESO is a gzipped tar file, which unpacks as directory **dl_meso_2.x**, where x is a generation number. Beneath the top level of this directory are a number of sub-directories:

- **LBE** - containing the LBE source code.
- **DPD** - containing the DPD source code.
- **JAVA** - containing the GUI source code.
- **MAN** - containing the DL_MESO user manual.
- **DEMO** - containing test cases for DL_MESO.
- **WORK** - an example 'working directory'.

1.4 Disclaimer

Neither the CCLRC, CCP5 nor any of the authors of the DL_MESO package guarantee that the package is free from error. Neither do they accept responsibility for any loss or damage that results from its use.

1.5 Copyright

CCLRC Daresbury Laboratory 2004.

1.6 Authors

Dr. Rongshan Qin
Computational Science & Engineering Department
CCLRC Daresbury Laboratory
Warrington WA4 4AD, England

1.7 Suggestions and Bug Reports

Please send suggestions or bug reports to r.qin@dl.ac.uk.

1.8 Acknowledgements

DL_MESO was developed under the auspices of the Engineering and Physical Sciences Research Council (EPSRC) for the EPSRC's Collaborative Computational Project for the Computer Simulation of Condensed Phases (CCP5).

The members of the CCP5 DL_MESO consortium were:

David M. Heyes, University of Surrey.
Chris M. Care, Sheffield Hallam University.
Peter V Coveney, University College London.
David Emerson, CCLRC Daresbury Laboratory.
Rob English, North East Wales Institute.
Andrea Ferrante, Unilever Port Sunlight.
Ian Halliday, Sheffield Hallam University.
John Harding, University of Sheffield.
Sebastian Reich, Imperial College.
W. Smith, CCLRC Daresbury Laboratory.
P. B. Warren, Unilever Port Sunlight.
Julia Yeomans, Oxford University.

Many other people have given advice and encouragement in the development of DL_MESO. We gratefully acknowledge the support of the following people: Maurice Leslie, Richard Wain, Alexandre Dupuis, Jonathan Chin, Michael Dupin and Weiming Liu.

Chapter 2

The DL_MESO GUI

2.1 Getting Started with the DL_MESO GUI

The DL_MESO GUI offers a convenient way of using the DL_MESO package, though it is not an essential tool for those who prefer command line operation. Working with the GUI requires the availability of java tools, particularly the *javac* compiler and the *java* runner for Java 2 version 1.4 (at least). These may be obtained from the *java.sun.com* website.

To build the GUI proceed as follows.

- Enter the DL_MESO/JAVA directory.
- Type “javac *.java” to compile the source code.
- Type “jar -cfm GUI.jar manifest.mf *.class” to create the GUI.jar executable JAR file.
- Move to your working directory.
- Launch the GUI.

A unix script called *makegui* that performs the build of the GUI can be found in the JAVA sub-directory.

Your working directory is the directory from which you wish to work when running DL_MESO. Working there will keep any files you generate separate from the DL_MESO source files. **Note** in the current version of DL_MESO the working directory should be at the same directory level as the JAVA directory i.e. within the DL_MESO top directory. An example of such a working directory (called *WORK* is present under the DL_MESO top directory.

In your working directory you can start the GUI with the command

- `java -jar ../JAVA/GUI.jar`

You may consider saving this command in a script for simple execution. An example script is present in the *WORK* sub-directory and is called *rungui*.

2.2 Lattice Boltzmann and the DL_MESO GUI

To access the LBE facilities in the DL_MESO GUI, proceed as follows.

- Click the “LBE” button to get the LBE panel.
- Click the “Define System” button and supply the required information. The file “lbin.sys” will be created by the step.
- Click the “Define Space” button to define the simulation space. The file “lbin.spa” will be created by this step.
- Click “EXIT” to finish the settings.

2.2.1 Defining the System

- The LBE model is defined by clicking the check boxes in “System dimension” and “LBE model”.
- The sizes of system are set by input to “Grid Numbers” and “Virtual Size X”. For 2D systems, the grid number in the z direction must be 1.
- The speed of sound is determined by the input values “Molar weight” and “Initial Temperature”.
- The “Noise Intensity” is mainly for initializing multiphase or multicomponent systems.
- The values for speed and pressure at the boundaries are only applied when you define the space with a “fixed speed” or “fixed pressure” boundary. Otherwise it is ignored.
- If a multi fluid system is to be studied, the value of the “Number of fluid” will need to be changed. The button “Set fluid parameter” must be clicked and the corresponding value input, followed by a click of the button “SAVE F”. In this case you cannot study fluid flow.
- If a solute parameter is required the “Number of fluid” must be set to 1 or 0, and the “Number of solute” must be set to a value larger than 0. The button “Set solute parameter” must be also be clicked. After all solute parameters have been input, button “SAVE C” must be clicked.

- To use temperature scaling, click the check box “yes” and then “Set thermal parameter”. Save the information using the “SAVE T” button.
- Click “SAVE” to save the input parameters.

2.2.2 Defining the Space Properties

- The data are created immediately after a click. The “Set Space” button should be re-clicked if a mistake has been made.
- The default boundary condition is periodic. Nothing will happen if the button “periodic boundary condition” is clicked.
- If a position is defined twice, the later definition will override the earlier one.
- Click the “CREATE SPACE” button to save all settings.

2.3 Dissipative Particle Dynamics and the DL_MESO GUI

To access the DPD facilities in the DL_MESO GUI, proceed as follows.

- Click the “DPD button to get the DPD panel.
- Click the “Define System” button and supply the required information. The file “dpdsys.dat” will be created by the step.
- Click “EXIT” to finish the settings.

2.3.1 Defining the System

- The text box labelled “DPD dimension” on the 1st line of the “Define DPD System” panel specifies the dimension of the system to be simulated 2 or 3 (for 2D or 3D) are acceptable inputs.
- The number of different species and the number of particles overall are specified in the text boxes labelled “number of species” and “number of particles” respectively on the 2nd line.
- The required cut off radius (i.e. the separation at which two particles are considered not to interact) is defined on line 3 in the text box “cut off radius”.
- The size of the simulation box (which has a periodic boundary condition) is defined on the 4th line in the three text boxes labelled “size of box: x: y: z:”.

- The 5th line defines the “time step” and the volume “flexible parameter” respectively.
- The number of steps required for the simulation and the interval for saving simulation data (in time steps) is specified on line 6 in text boxes labelled “total steps” and “save span”.
- The required system temperature and the interval between temperature rescales is defined on line 7 in text boxes “temperature” and “temp rescale interval”.
- The number of particles representing each species is defined on line 8 in text boxes labelled “species particle numbers: 0: 1: 2:”. Note that it is only necessary to specify nonzero numbers in all boxes if three different species are present etc. Note also the total numbers on all boxes should equal the total number of particles in the system.
- The masses of the particle species are specified in text boxes on line 9. The boxes are labelled “species masses: 0: 1: 2:”. Again there is no need to specify data in all boxes if insufficient species are present.
- On line 10 the “verlet factor” and “dissipative factor” are specified.
- The text boxes on lines 12-14 specify the interaction parameters for the DPD potential energy functions. Entries labelled “nm” where “n” and “m” are intergers specify interactions between particle types labelled “n” and “m” respectively. It is not necessary to specify values for nonexistant interactions.
- Click “SAVE” to save the input parameters in file “dpdsys.dat”

2.4 Compiling and Running the LBE/DPD Program

- Compiling the LBE/DPD code may be accomplished through the compiler panel which is activated from the “Compile LBE/DPD Code” button. The panel that appears allows you to select the operating system, the C++ compiler; and the version (serial or parallel) of the LBE/DPD code you wish to build. Clicking the “Do it” button will start the compilation. A message box will signal the completion.
- If the compilation fails, you may need to edit the code. An editing panel is available for this purpose using the “Change LBE/DPD Code”. Its function is identical to the “Compile LBE/DPD Code” panel in operation.
- Running the LBE/DPD code is made possible through the “Run LBE/DPD Program” button, which activates a panel that allows you to select the required submission command and then submit the job. Note you may need to create a suitable run script in your working directory beforehand.

- Collecting data (from multiple processors) and plotting results is possible using the “Gather LBE/DPD Data” and “Plot LBE/DPD Results” buttons. Be aware that these are not fully active options at present.

2.5 Notes

- There are some inactive buttons reserved for later use.
- Click “EXIT” to close down the GUI.

Chapter 3

The Lattice Boltzmann Equation: Basic Theory

3.1 Introduction

The Lattice Boltzmann method utilizes fully discretized space, time and velocity to describe the evolution of fluid. Space is represented by a regularly distributed grid. Time flow is obtained by integrating over discrete time steps. Discrete velocity vectors are defined to ensure that a particle moves from one grid point to another without falling between grid points.

The Lattice Boltzmann method inherits the discrete concept from its ancestor Lattice Gas Automata (LGA). The main difference between Lattice Boltzmann Equation (LBE) and LGA is that with LBE the physical state of a grid is described by a set of single particle distribution functions instead of a particle number. This allows LBE to simulate condensed matter fluids distinct from diluted fluids in which the mean free path of the component particles is much larger than the lattice distance.

The lattice Boltzmann algorithm can be summarized by the following:

- Fluid properties are mapped onto a discrete lattice
- The physical state at each lattice point is described by a set of particle distribution functions.
- Macroscopic fluid variables are defined via moments of the distribution functions.
- The system evolution towards equilibrium is through the relaxation of the distribution function to its equilibrium form.

3.2 Basic Definitions

Triangular and rectangular lattices are two of the most popular grid forms. The former has O6 rotation isotropy and has been widely applied in two-dimensional systems, e.g. the D2Q7 and D2Q13 models. The latter has only O4 rotation isotropy but it more easily handles the simulation of three dimensional systems with complex boundary conditions. The equilibrium for D2Q9 and D3Q27 lattice models can be derived *a priori* from the Maxwell equilibrium distribution. D3Q15 and D3Q19 models appear to be more popular than D3Q27 because the latter is much more expensive in terms of computing cost.

It is required that the equilibrium state should be able to reproduce elementary macroscopic fluid variables

$$\rho = \sum_{i=0}^q f_i \quad (3.1)$$

$$\rho u_\alpha = \sum_{i=0}^q f_i \hat{e}_{i\alpha} \quad (3.2)$$

where ρ is the density, f_i the i -th particle distribution function, \hat{e}_i the i -th speed vector and u_α the macroscopic velocity along the α -axis. In the mesoscale method, \hat{e}_i is not the thermal velocity of a particle and therefore

$$E \neq \frac{1}{2} \sum f_i (\hat{e}_i - u)^2 \quad (3.3)$$

Eq. 3.3 implies that the temperature cannot be derived from the mesoscale particle distribution function. However, temperature *can* be defined at each grid point.

3.3 Derivation of Equilibrium

There are basically two methods to construct lattice Boltzmann equilibria. The first is called the *bottom-up* method and equilibrium is derived from the Maxwell-Boltzmann equilibrium distribution. The second is called the *top-down* method and equilibrium is constructed so that the required macroscopic properties can be reproduced. Only the bottom-up method is demonstrated here.

The Maxwell-Boltzmann single particle equilibrium distribution function is

$$f^{eq} = \frac{\rho}{(2\pi\theta)^{\frac{D}{2}}} \exp\left[-\frac{(\xi - u)^2}{2\theta}\right] \quad (3.4)$$

where $\theta = k_B T/m$, k_B is the gas constant, T is temperature, m is molar mass, D is the space dimension, ξ is the thermal velocity of a particle and u the macroscopic velocity.

When $|\xi - u| \ll \sqrt{\theta}$, eq. 3.4 can be expanded into

$$f^{eq} = \frac{\rho}{(2\pi\theta)^{\frac{D}{2}}} \exp\left(-\frac{\xi^2}{2\theta}\right) \left[1 + \frac{\xi \cdot v}{\theta} + \frac{(\xi \cdot v)^2}{2\theta^2} - \frac{u^2}{2\theta}\right] \quad (3.5)$$

For a microscopic quantity $\psi(\xi)$, the associated macroscopic quantity Ψ is calculated by

$$\Psi = \int \psi(\xi) f^{eq} d\xi \quad (3.6)$$

Let $\xi_\alpha = \sqrt{2\theta} c_\alpha$ and $u_\alpha = \sqrt{2\theta} v_\alpha$, where α represents a Cartesian coordinate. Eqs. 3.5 and 3.6 have

$$\Psi = \int e^{-c^2} \psi(c) \frac{\sqrt{2\theta} \rho}{(2\pi\theta)^{\frac{D}{2}}} [1 + 2c \cdot v + 2(c \cdot v)^2 - v^2] dc \quad (3.7)$$

Using Gaussian quadrature, eq.3.7 changes into

$$\Psi = \sum_i \psi(c_i) \frac{\sqrt{2\theta} \rho}{(2\pi\theta)^{\frac{D}{2}}} w(c_i) [1 + 2c_i \cdot v + 2(c_i \cdot v)^2 - v^2] \quad (3.8)$$

Let

$$w_i = \frac{\sqrt{2\theta}}{(2\pi\theta)^{\frac{D}{2}}} w(c_i) \quad (3.9)$$

and

$$f^{eq} = w_i \rho [1 + 2c_i \cdot v + 2(c_i \cdot v)^2 - v^2] \quad (3.10)$$

The value of $w(c_i)$ can be obtained from Gauss-Hermite integration. Eq. 3.10 is the equilibrium particle distribution function in the discrete regime. w_i is called the weight factor for speed vector c_i . Eq. 3.10 can also be written as

$$f^{eq} = w_i \rho \left[1 + \frac{3(e_i \cdot u)}{c^2} + \frac{9(e_i \cdot u)^2}{2c^4} - \frac{3u^2}{2c^2}\right] \quad (3.11)$$

Where $c = \sqrt{3\theta} = \sqrt{3k_B T/m}$ is the modulus of the basic lattice vector. For water at 20 °C, $c = 367.8$ m/s.

3.4 Structural Relaxation and Macroscopic Equations

The Lattice Boltzmann method uses BGK (Bhatnager, Gross and Krook) approximation to describe the structural relaxation. The single particle distribution function evolves to the equilibrium state via

$$f_i(x_i + e_i \Delta t, t + \Delta t) - f_i(x_i, t) = -\frac{\Delta t}{\tau_f} [f_i(x_i, t) - f_i^{eq}] \quad (3.12)$$

where τ_f is called the relaxation time and is related to the kinetic viscosity of fluid.

To derive the macroscopic equations, the left hand side of eq. 3.12 can be expanded as

$$f_i(x_i + e_i \Delta t, t + \Delta t) - f_i(x_i, t) = \sum_{m=1}^{\infty} \frac{\Delta t^m}{m!} (\partial_t + e_{i\alpha} \partial_\alpha)^m f_i(x, t) \quad (3.13)$$

Expanding the instantaneous particle distribution function around its equilibrium and retaining only the first order gives

$$f_i(x_i, t) = f_i^{eq}(x_i, t) - \tau_f (\partial_t + e_{i\alpha} \partial_\alpha) f_i^{eq}(x, t) + O(\partial^2) \quad (3.14)$$

Substituting eqs. 3.13 and 3.14 into the left hand of eq. 3.12, gives the second order differential equation for the equilibrium distribution

$$\frac{f_i^{eq} - f_i}{\tau_f} = (\partial_t + e_{i\alpha} \partial_\alpha) f_i^{eq} - w_f (\partial_t + e_{i\alpha} \partial_\alpha)^2 f_i^{eq} + O(\partial^3) \quad (3.15)$$

where

$$w_f = \tau_f - \frac{\Delta t}{2} \quad (3.16)$$

Summing eq. 3.15 over i and ignoring the second order derivative we obtain

$$0 = \partial_t \rho + \partial_\alpha \rho u_\alpha - w_f \partial_\beta (\partial_\beta \rho u_\beta + \partial_\alpha \sum_i f_i^{eq} e_{i\alpha} e_{i\beta}) + O(\partial^3) \quad (3.17)$$

Summing eq. 3.17 times e_i over i we obtain

$$0 = \partial_t \rho u_\alpha + \partial_\beta \sum_i f_i^{eq} e_{i\alpha} e_{i\beta} - w_f \partial_\gamma (\partial_t \sum_i f_i^{eq} e_{i\alpha} e_{i\gamma} + \partial_\beta \sum_i f_i^{eq} e_{i\alpha} e_{i\beta} e_{i\gamma}) + O(\partial^3) \quad (3.18)$$

Eq. 3.18 shows that the second term in eq. 3.17 is of the third order in the derivative. Therefore, we have the continuity equation to the second order of the derivative

$$\partial_t \rho + \nabla \cdot \rho u = 0 \quad (3.19)$$

Defining the third and fourth order moments

$$\sum_i f_i^{eq} e_{i\alpha} e_{i\beta} = P_{\alpha\beta} + \rho u_\alpha u_\beta \quad (3.20)$$

$$\sum_i f_i^{eq} e_{i\alpha} e_{i\beta} e_{i\gamma} = P_{\alpha\beta} u_\gamma + P_{\alpha\gamma} u_\beta + P_{\beta\gamma} u_\alpha + \rho u_\alpha u_\beta u_\gamma \quad (3.21)$$

with these definitions, eq. 3.18 leads to the incompressible Navier Stokes equation

$$\rho \partial_t u_\alpha + \rho u_\beta \partial_\beta u_\alpha = -\partial_\beta P_{\alpha\beta} + \frac{\rho w_f D}{3} \partial_\beta \left(\frac{\delta_{\alpha\beta} - 3\partial_\alpha P_{\alpha\beta}}{D} \partial_\gamma u_\gamma + \partial_\alpha u_\beta + \partial_\beta u_\alpha \right) + o(\partial^3) \quad (3.22)$$

where the kinetic viscosity is given by $\mu = w_f/3 = \frac{1}{3}(\tau_f - \frac{1}{2})$.

Similarly, the convection diffusion equation governing the evolution of solute transfer and the convection conduction equation governing the evolution of thermal transfer can be obtained.

3.5 Mesoscale Interaction

The rate of change of momentum is proportional to the force

$$\rho u_\alpha(t + \Delta t) = \rho u_\alpha(t) + F_\alpha \Delta t \quad (3.23)$$

The lattice Boltzmann equation takes account the effect of relaxation and express eq. 3.23 into a new format

$$\rho u_\alpha(t + \Delta t) = \rho u_\alpha(t) + F_\alpha \tau_f \quad (3.24)$$

F can be a long ranged body force or any local interactions. In the Shan-Chen model, F is defined as

$$F^a = -\psi^a(x) \sum_b g_{ab} \sum_i w_i \psi^b(x + e_i) e_i \quad (3.25)$$

where g_{ab} is the interaction coefficient between element a and b . ψ^a is related to the density of element a and can take many different forms, e. g.

$$\psi^a(x) = 1 - \exp(-\rho^a) \quad (3.26)$$

3.6 Summary of Lattice Boltzmann Equation

Lattice Boltzmann is an established numerical methodology for handling hydrodynamics in fluid. It is particularly suitable for simulating systems with complex boundary conditions. It is also suitable for simulating systems with phase transitions because mesoscale interactions can be merged into the method easily.

Chapter 4

Dissipative Particle Dynamics: Basic Theory

4.1 Introduction

Dissipative Particle Dynamics (DPD) is an off-lattice, discrete particle method for modelling mesoscopic systems. It has little in common with Lattice Boltzmann methods, except in its application to systems of similar length and time scales.

The DPD method inherits its methodology from classical molecular dynamics (MD), particularly from Brownian Dynamics (BD). It differs from BD however in an important way: it is *Galilean invariant* and for this reason conserves hydrodynamic behaviour, while the BD method does not. Many systems are crucially dependent on hydrodynamic interactions and it is essential to retain this feature in the model. DPD is particularly useful for simulating systems on the near-molecular scale, such as polymers, biopolymers, lipids, emulsions and surfactants - systems in which large scale structure evolves on a time scale that is too long to be modelled effectively by molecular dynamics. DPD may also be used when such systems experience shear and flow gradients.

The DPD algorithm can be summarized by the following:

- A condensed phase system may be modelled as a system of free particles interacting directly through ‘soft’ forces.
- The system is coupled to a heat bath via stochastic forces, which act on the particles in a pairwise manner.
- The particles also experience a damping or drag force, which also acts in a pairwise manner.
- Thermodynamic equilibrium is maintained through the balance of the stochastic and drag

forces i.e. the method satisfies the fluctuation-dissipation theorem.

- At equilibrium (or steady state) the properties of the system are calculated as averages over the individual particles, as in molecular dynamics.

4.2 Outline of Method

In DPD ¹ the system is modelled as a system of free particles, which are spherical and interact over a range that is of the same order as their diameters. The particles can be thought of as assemblies or aggregates of molecules, such as solvent molecules or polymers, or more simply as carriers of momentum.

The equations governing the time evolution in a DPD simulation resemble those of ordinary MD.

$$\begin{aligned}\frac{d\vec{v}_i}{dt} &= \frac{1}{m_i}\vec{f}_i \\ \frac{d\vec{r}_i}{dt} &= \vec{v}_i\end{aligned}\tag{4.1}$$

in which \vec{r}_i , \vec{v}_i and \vec{f}_i are the position, velocity and force of the i 'th particle, which has mass m_i . The force on the particle is a sum of pair forces:

$$\vec{f}_i = \sum_{j \neq i}^N (\vec{f}_{ij}^C + \vec{f}_{ij}^D + \vec{f}_{ij}^R)\tag{4.2}$$

In which \vec{f}_{ij}^C , \vec{f}_{ij}^D and \vec{f}_{ij}^R are the *conservative*, *drag* and *random* (or *stochastic*) pair forces respectively. Each represents the force exerted on particle i due to the presence of particle j .

The conservative interactions are usually ‘soft’ (i.e. weakly interacting) so that the particles can pass by each other (or even through each other) relatively easily so that equilibrium is achieved quickly. A common form of interaction potential is an inverse parabola:

$$V(r_{ij}) = \frac{1}{2}A_{ij}(1 - r_{ij})^2\tag{4.3}$$

where $r_{ij} = |\vec{r}_i - \vec{r}_j|$ and A_{ij} is the interaction strength. A_{ij} may be the same for all particle pairs, or may be different for different particle types.

Equation 4.3 gives rise to a repulsive force of the form

$$\vec{f}_{ij}^C = A(1 - r_{ij})\frac{\vec{r}_{ij}}{r_{ij}}\tag{4.4}$$

¹The outline of the DPD method supplied here is based on the paper by R.D. Groot and P.B. Warren, J. Chem. Phys. **107** 4423 (1997).

This is the deterministic or *conservative* force \vec{f}_i^C , exerted on particle i by particle j . Note the force is zero when $r_{ij} > 1$ and thus the particles have an effective diameter of 1 in these units.

The stochastic forces experienced by the particles is again pairwise in nature and takes the form:

$$\vec{f}_{ij}^R = \sigma w^R(r_{ij}) \zeta_{ij} \Delta t^{-1/2} \frac{\vec{r}_{ij}}{r_{ij}} \quad (4.5)$$

in which Δt is the time step and $w^R(r_{ij})$ is a switching function which imposes a finite limit on the range of the stochastic force. ζ_{ij} is a random number with zero mean and unit variance. The constant σ is related to the temperature, as is understood from the role of the stochastic force in representing a heat bath.

Finally the particles are subject to a drag force, which depends on the relative velocity between interacting pairs of particles:

$$\vec{f}_{ij}^D = -\gamma w^D(r_{ij}) (\vec{r}_{ij} \cdot \vec{v}_{ij}) \frac{\vec{r}_{ij}}{r_{ij}} \quad (4.6)$$

where $w^D(r_{ij})$ is once again a switching function, and $\vec{v}_{ij} = \vec{v}_i - \vec{v}_j$. The constant γ is the drag coefficient. It follows from the fluctuation-dissipation theorem that for thermodynamic equilibrium to result from this method the following relations must hold.

$$\begin{aligned} \sigma^2 &= 2\gamma k_B T \\ w^D(r_{ij}) &= [w^R(r_{ij})]^2. \end{aligned} \quad (4.7)$$

In practice the switching functions are defined through

$$w^D(r_{ij}) = (1 - r_{ij})^2 \quad (r_{ij} < 1) \quad (4.8)$$

which ensures that all interactions are switched off at the range $r_{ij} = 1$.

4.3 Derivation of Equilibrium

The derivation of the DPD algorithm² is based on the Fokker-Planck equation

$$\frac{\partial \rho}{\partial t} = \mathcal{L} \rho \quad (4.9)$$

Where ρ is the equilibrium distribution function and \mathcal{L} is the evolution operator, which may be split into *conservative* and *dissipative* parts:

$$\mathcal{L} = \mathcal{L}^C + \mathcal{L}^D \quad (4.10)$$

²See P. Espan ol and P.B. Warren, Europhys Lett. **30** 191 (1995).

with

$$\begin{aligned}\mathcal{L}^C &= -\sum_{i=1}^N \frac{\vec{p}_i}{m_i} \frac{\partial}{\partial \vec{r}_i} - \sum_{i \neq j}^N \vec{f}_{ij}^C \frac{\partial}{\partial \vec{p}_i} \\ \mathcal{L}^D &= \sum_{i \neq j}^N \hat{e}_{ij} \cdot \frac{\partial}{\partial \vec{p}_i} \left[\gamma w^D(\hat{e}_{ij} \cdot \vec{v}_{ij}) + \frac{\sigma^2}{2} \{w^R(r_{ij})\}^2 \hat{e}_{ij} \cdot \left\{ \frac{\partial}{\partial \vec{p}_i} - \frac{\partial}{\partial \vec{p}_j} \right\} \right]\end{aligned}\quad (4.11)$$

where $\hat{e}_{ij} = \vec{r}_{ij}/r_{ij}$.

When $\alpha = \gamma = 0$ then equation 4.9 becomes:

$$\frac{\partial \rho}{\partial t} = \mathcal{L}^J \rho \quad (4.12)$$

for which the equilibrium solution is evidently

$$\rho^{eq} = \frac{1}{Z} \exp \left(\frac{1}{k_B T} \left[\sum_{i=1}^N \frac{p_i^2}{2m_i} + \frac{1}{2} \sum_{j \neq i}^N \phi(r_{ij}) \right] \right). \quad (4.13)$$

Which is, of course, the Boltzmann distribution function for an equilibrium system. Thus it is apparent that for the simulation based on equation 4.9 to maintain the same distribution function, the terms in the operator \mathcal{L}^D of equation 4.11 must sum to zero. It follows that the conditions given in equation 4.7 must apply.

4.4 Summary of Dissipative Particle Dynamics

DPD is a simple method. All that is required is a system of spherical particles enclosed in a periodic box undergoing time evolution as a result of the above forces. In implementation it differs very little from molecular dynamics. It should be noted that all computed interactions are pairwise, which means that the principle of the conservation of momentum in the system is preserved. (The term ‘Galilean invariance’ implies the same thing.) The conservation of momentum is required for the preservation of hydrodynamic forces.

Chapter 5

DL_MESO_LBE Basic Definition

5.1 Lattice model

DL_MESO_LBE utilises a right-handed cartesian coordinate system with the x-axis from left to right in the horizontal direction, the y-axis from low to high in vertical direction and the z-axis from back to front.

D2Q9, D3Q15, D3Q19 and D3Q27 lattice models have been included. The speed vectors and weight factors are

D2Q9

$$\begin{aligned} \text{weight factor: } w_0 &= \frac{4}{9} \\ w_i &= \frac{1}{9} \text{ for } i=1, 2, 3, 4 \\ w_i &= \frac{1}{36} \text{ for } i=5, 6, 7, 8 \end{aligned}$$

$$\begin{aligned} \text{speed vector : } v_{0x} &= 0 & v_{0y} &= 0 \\ v_{1x} &= 1 & v_{1y} &= 0 \\ v_{2x} &= 0 & v_{2y} &= 1 \\ v_{3x} &= -1 & v_{3y} &= 0 \\ v_{4x} &= 0 & v_{4y} &= -1 \\ v_{5x} &= 1 & v_{5y} &= 1 \\ v_{6x} &= -1 & v_{6y} &= 1 \\ v_{7x} &= -1 & v_{7y} &= -1 \\ v_{8x} &= 1 & v_{8y} &= -1 \end{aligned}$$

D3Q15

weight factor: $w_0 = \frac{2}{9}$
 $w_i = \frac{1}{9}$ for $i=1, 2, \dots, 6$
 $w_i = \frac{1}{72}$ for $i=7, 8, \dots, 14$

speed vector :

$v_{0x} = 0$	$v_{0y} = 0$	$v_{0z} = 0$
$v_{1x} = 1$	$v_{1y} = 0$	$v_{1z} = 0$
$v_{2x} = 0$	$v_{2y} = 1$	$v_{2z} = 0$
$v_{3x} = 0$	$v_{3y} = 0$	$v_{3z} = 1$
$v_{4x} = -1$	$v_{4y} = 0$	$v_{4z} = 0$
$v_{5x} = 0$	$v_{5y} = -1$	$v_{5z} = 0$
$v_{6x} = 0$	$v_{6y} = 0$	$v_{6z} = -1$
$v_{7x} = 1$	$v_{7y} = 1$	$v_{7z} = 1$
$v_{8x} = -1$	$v_{8y} = 1$	$v_{8z} = 1$
$v_{9x} = 1$	$v_{9y} = -1$	$v_{9z} = 1$
$v_{10x} = 1$	$v_{10y} = 1$	$v_{10z} = -1$
$v_{11x} = -1$	$v_{11y} = -1$	$v_{11z} = -1$
$v_{12x} = 1$	$v_{12y} = -1$	$v_{12z} = -1$
$v_{13x} = -1$	$v_{13y} = 1$	$v_{13z} = -1$
$v_{14x} = -1$	$v_{14y} = -1$	$v_{14z} = 1$

D3Q19

weight factor: $w_0 = \frac{1}{3}$
 $w_i = \frac{1}{18}$ for $i=1, 2, \dots, 6$
 $w_i = \frac{1}{36}$ for $i=7, 8, \dots, 18$

$$\begin{array}{lll}
\text{speed vector :} & v_{0x} = 0 & v_{0y} = 0 & v_{0z} = 0 \\
& v_{1x} = 1 & v_{1y} = 0 & v_{1z} = 0 \\
& v_{2x} = 0 & v_{2y} = 1 & v_{2z} = 0 \\
& v_{3x} = 0 & v_{3y} = 0 & v_{3z} = 1 \\
v_{4x} = -1 & v_{4y} = 0 & v_{4z} = 0 \\
& v_{5x} = 0 & v_{5y} = -1 & v_{5z} = 0 \\
& v_{6x} = 0 & v_{6y} = 0 & v_{6z} = -1 \\
& v_{7x} = 1 & v_{7y} = 1 & v_{7z} = 0 \\
v_{8x} = -1 & v_{8y} = 1 & v_{8z} = 0 \\
v_{9x} = -1 & v_{9y} = -1 & v_{9z} = 0 \\
& v_{10x} = 1 & v_{10y} = -1 & v_{10z} = 0 \\
& v_{11x} = 0 & v_{11y} = 1 & v_{11z} = 1 \\
& v_{12x} = 0 & v_{12y} = -1 & v_{12z} = 1 \\
& v_{13x} = 0 & v_{13y} = -1 & v_{13z} = -1 \\
& v_{14x} = 0 & v_{14y} = 1 & v_{14z} = -1 \\
& v_{15x} = 1 & v_{15y} = 0 & v_{15z} = 1 \\
v_{16x} = -1 & v_{16y} = 0 & v_{16z} = 1 \\
v_{17x} = -1 & v_{17y} = 0 & v_{17z} = -1 \\
& v_{18x} = 1 & v_{18y} = 0 & v_{18z} = -1
\end{array}$$

D3Q27

$$\begin{array}{l}
\text{weight factor: } w_0 = \frac{8}{27} \\
w_i = \frac{2}{27} \text{ for } i=1, 2, \dots, 6 \\
w_i = \frac{1}{54} \text{ for } i=7, 8, \dots, 18 \\
w_i = \frac{1}{216} \text{ for } i=19, 20, \dots, 26
\end{array}$$

speed vector :	$v_{0x} = 0$	$v_{0y} = 0$	$v_{0z} = 0$
	$v_{1x} = 1$	$v_{1y} = 0$	$v_{1z} = 0$
	$v_{2x} = 0$	$v_{2y} = 1$	$v_{2z} = 0$
	$v_{3x} = -1$	$v_{3y} = 0$	$v_{3z} = 0$
	$v_{4x} = 0$	$v_{4y} = -1$	$v_{4z} = 0$
	$v_{5x} = 0$	$v_{5y} = 0$	$v_{5z} = 1$
	$v_{6x} = 0$	$v_{6y} = 0$	$v_{6z} = -1$
	$v_{7x} = 1$	$v_{7y} = 1$	$v_{7z} = 0$
	$v_{8x} = -1$	$v_{8y} = 1$	$v_{8z} = 0$
	$v_{9x} = -1$	$v_{9y} = -1$	$v_{9z} = 0$
	$v_{10x} = 1$	$v_{10y} = -1$	$v_{10z} = 0$
	$v_{11x} = 1$	$v_{11y} = 0$	$v_{11z} = -1$
	$v_{12x} = -1$	$v_{12y} = 0$	$v_{12z} = -1$
	$v_{13x} = -1$	$v_{13y} = 0$	$v_{13z} = 1$
	$v_{14x} = 1$	$v_{14y} = 0$	$v_{14z} = 1$
	$v_{15x} = 0$	$v_{15y} = 1$	$v_{15z} = 1$
	$v_{16x} = 0$	$v_{16y} = -1$	$v_{16z} = 1$
	$v_{17x} = 0$	$v_{17y} = -1$	$v_{17z} = -1$
	$v_{18x} = 0$	$v_{18y} = 1$	$v_{18z} = -1$
	$v_{19x} = 1$	$v_{19y} = 1$	$v_{19z} = 1$
	$v_{20x} = 1$	$v_{20y} = 1$	$v_{20z} = -1$
	$v_{21x} = -1$	$v_{21y} = 1$	$v_{21z} = -1$
	$v_{22x} = -1$	$v_{22y} = 1$	$v_{22z} = 1$
	$v_{23x} = 1$	$v_{23y} = -1$	$v_{23z} = 1$
	$v_{24x} = 1$	$v_{24y} = -1$	$v_{24z} = -1$
	$v_{25x} = -1$	$v_{25y} = -1$	$v_{25z} = -1$
	$v_{26x} = -1$	$v_{26y} = -1$	$v_{26z} = 1$

5.2 Data structure

5.2.1 Storage of particle distribution functions

For a system with a square lattice, the total grid number = $lbsy.nx \times lbsy.ny \times lbsy.nz$, where $lbsy.nx$, $lbsy.ny$ and $lbsy.nz$ are the grid numbers along x-axis, y-axis and z-axis, respectively. The grids points are arranged in a serial order of

$g_{000}, g_{001}, \dots, g_{00 lbsy.nz}, g_{010}, g_{011}, \dots, g_{0 lbsy.ny lbsy.nz}, g_{100}, g_{101}, \dots, g_{lbsy.nx lbsy.ny lbsy.nz}$

At each grid point, DL_MESO_LBE arranges the particle distribution functions in the order of: fluid functions; solute functions; temperature functions; and phase field order parameter. For

Table 5.1: Boundary condition category

value	meaning
0	liquid
10	in the domain boundary for parallel computing
11	inside solid
12	bounceback boundary
100-199	constant speed, composition and temperature boundary
200-299	constant speed, Neumann composition and temperature boundary
300-399	constant speed and composition, Neumann temperature boundary
400-499	constant speed and temperature, Neumann composition boundary
500-599	constant pressure, composition and temperature boundary
600-699	constant pressure and temperature, Neumann composition boundary
700-799	constant pressure and composition, Neumann temperature boundary
800-899	constant pressure, Neumann composition and temperature boundary

example, for a D2Q9 lattice with two fluids, scalar temperature and phase-field, the distribution functions are in the order of

$$f_0^0, f_1^0, f_2^0, f_3^0, f_4^0, f_5^0, f_6^0, f_7^0, f_8^0, f_0^1, f_1^1, f_2^1, f_3^1, f_4^1, f_5^1, f_6^1, f_7^1, f_8^1, T_0, T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, pf$$

Therefore, the number of particle distribution functions in each grid

$$lbsitelength = (lbsy.nf + lbsy.nc + lbsy.nt) \times lbsy.nq + lbsy.ph$$

where *lbsy.nf*, *lbsy.nc*, *lbsy.nt*, *lbsy.nq* and *lbsy.ph* are respectively: number of fluid; number of solute; number of temperature scalar; number of discrete speed; and number of phase field order parameter. *lbsy.nt* and *lbsy.ph* can only take the values of 1 or 0, representing with temperature scalar and phase field parameter or without those parameters. Also, if *lbsy.nc* \neq 0, *lbsy.nf* cannot be set larger than 1.

5.2.2 Storage of space property

The space property is represented by an integer value in DL-MESO lattice Boltzmann method. For example, *lbphi*[100]=0 represents that the 100th grid is a liquid site and *lbphi*[101]=12 the bounce back boundary. Table 5.1 list the category of the space properties

The orientation of a solid-liquid boundary is also represented by the value of a integer. For exam-

Table 5.2: Letter Notation Method

letter	meaning
T	Normal Vector Point to Top
D	Normal Vector Point to Down
L	Normal Vector Point to Left
R	Normal Vector Point to Right
F	Normal Vector Point to Front
B	Normal Vector Point to Back
V	Constant Speed
P	Constant Pressure (Density)
C	Constant Solute Composition
T	Constant Temperature
PS	Planar surface
CC	Concave Corner
CE	Concave Edge

ple, a planar surface with normal vector along y-axis is denoted by 21. And a concave corner face to top-right-front corner is denoted by 31. It must be pointed out that only those space positions located in the surface of a FCC cube have been included and translated in DL_MESO_LBE. A point with random orientation, e.g. a 47 degree plane, has not been included.

The digital number is rather confusing and difficult to understand. DL_MESO_GUI, therefore, has built up a translator which enable translating a *word* to the corresponding integer number. The *word* is made up of defined letters as listed in Table 5.2. With the combination of category and orientation. The boundary conditions are listed in table 5.3. The *letters* are in the order of: 1. Fluid property of constant speed or constant pressure. 2. solute property of constant composition or Newmann boundary. 3. Temperature property of isolate or heat bath. 4. Geometric property of planar surface or concave corner or concave edge. 5. Orientation of boundary. For example, a shearing planar surface face down the y-axis with constant composition and isothermal on it is represented as *VCBPSD* and translated as 322.

5.2.3 Storage of running information

The DL_MESO lattice Boltzmann component defines three structures to store the system information (The system here means the physical object which you are simulating rather than the computer system.), domain information and neighbor information. The parameters in those structures are listed in table 5.5 5.6 and 5.7.

Table 5.3: Notation of boundary condition

CTPST	121	VCTPSD	122	VCTPSL	123
VCTPSR	124	VCTPSF	125	VCTPSB	126
VCTCCTRB	127	VCTCCTLB	128	VCTCCDLB	129
VCTCCDRB	130	VCTCCTRF	131	VCTCCTLF	132
VCTCCDLF	133	VCTCCDRF	134	VCTCETR	143
VCTCETL	144	VTCEDL	145	VTCEDR	146
VTCETF	147	VTCELF	148	VTCEDF	149
VTCERF	150	VTCETB	151	VTCELB	152
VTCEDB	153	VTCERB	154	VBBPST	221
VBBPSD	222	VBBPSL	223	VBBPSR	224
VBBPSF	225	VBBPSB	226	VBBCCTRB	227
VBBCCTLB	228	VBBCCDLB	229	VBBCCDRB	230
VBBCCTRF	231	VBBCCTLF	232	VBBCCDLF	233
VBBCDRF	234	VBBCETR	243	VBBCETL	244
VBBCEDL	245	VBBCEDR	246	VBBCETF	247
VBBCELF	248	VBBCEDF	249	VBBCERF	250
VBBCETB	251	VBBCELB	252	VBBCEDB	253
VBBCERB	254	VCBPST	321	VCBPSD	322
VCBPSL	323	VCBPSR	324	VCBPSF	325
VCBPSB	326	VCBCCTRB	327	VCBCCTLB	328
VCBCCDLB	329	VCBCCDRB	330	VCBCCTRF	331
VCBCCTLF	332	VCBCCDLF	333	VCBCCDRF	334
VCBCETR	343	VCBCETL	344	VCBCEDL	345
VCBCEDR	346	VCBCETF	347	VCBCELF	348
VCBCEDF	349	VCBCERF	350	VCBCETB	351
VCBCELB	352	VCBCEDB	353	VCBCERB	354
VBTPST	421	VBTPSD	422	VBTPSL	423
VBTPSR	424	VBTPSF	425	VBTPSB	426
VBTCCTRB	427	VBTCCTLB	428	VBTCCDLB	429
VBTCCDRB	430	VBTCCTRF	431	VBTCCTLF	432
VBTCDLF	433	VBTCDRF	434	VBTCETR	443
VBTCETL	444	VBTCEDL	445	VBTCEDR	446
VBTCETF	447	VBTCELF	448	VBTCEDF	449
VBTCERF	450	VBTCETB	451	VBTCELB	452
VBTCEDB	453	VBTCERB	454	PCTPST	521

Table 5.4: Notation of boundary condition (continued)

PCTPSD	522	PCTPSL	523	PCTPSR	524
PCTPSF	525	PCTPSB	526	PCTCCTRB	527
PCTCCTLB	528	PCTCCDLB	529	PCTCCDRB	530
PCTCCTRF	531	PCTCCTLF	532	PCTCCDLF	533
PCTCCDRF	534	PCTCETR	543	PCTCETL	544
PCTCEDL	545	PCTCEDR	546	PCTCETF	547
PCTCELF	548	PCTCEDF	549	PCTCERF	550
PCTCETB	551	PCTCELB	552	PCTCEDB	553
PCTCERB	554	PBTPST	621	PBTPSD	622
PBTPSL	623	PBTPSR	624	PBTPSF	625
PBTPSB	626	PBTCCTRB	627	PBTCCTLB	628
PBTCDLB	629	PBTCDRB	630	PBTCCTRF	631
PBTCCTLF	632	PBTCDLF	633	PBTCDRF	634
PBTCETR	643	PBTCETL	644	PBTCEDL	645
PBTCEDR	646	PBTCETF	647	PBTCSELF	648
PBTCEDF	649	PBTCERF	650	PBTCETB	651
PBTCELB	652	PBTCEDB	653	PBTCERB	654
CBPST	721	PCBPSD	722	PCBPSL	723
PCBPSR	724	PCBPSF	725	PCBPSB	726
PCBCCTRB	727	PCBCCTLB	728	PCBCDLB	729
PCBCDRB	730	PCBCCTRF	731	PCBCCTLF	732
PCBCDLF	733	PCBCDRF	734	PCBCETR	743
PCBCETL	744	PCBCEDL	745	PCBCEDR	746
PCBCETF	747	PCBCSELF	748	PCBCEDF	749
PCBCERF	750	PCBCETB	751	PCBCELB	752
PCBCEDB	753	PCBCERB	754	PBBPST	821
PBBPSD	822	PBBPSL	823	PBBPSR	824
PBBPSF	825	PBBPSB	826	PBBCCTRB	827
PBBCCTLB	828	PBBCDLB	829	PBBCDRB	830
PBBCCTRF	831	PBBCCTLF	832	PBBCDLF	833
PBBCDRF	834	PBBCETR	843	PBBCETL	844
PBBCEDL	845	PBBCEDR	846	PBBCETF	847
PBBCSELF	848	PBBCEDF	849	PBBCERF	850
PBBCETB	851	PBBCELB	852	PBBCEDB	853
PBBCERB	854				

Table 5.5: System information

parameter	meaning
lbsy.nd	space dimension
lbsy.nq	number of discrete speed
lbsy.nf	number of fluid
lbsy.nc	number of solute
lbsy.nt	temperature scalar
lbsy.pf	phase field order parameter
lbsy.nx	grid number in x direction
lbsy.ny	grid number in y direction
lbsy.nz	grid number in z direction ($lbsy.nz \equiv 1$ when $lbsy.nd = 2$)

5.3 The Parameters and their Function

Table 5.8 lists all the parameters defined in DL_MESO_LBE. The whole range parameters are named with the prefix '*lb*'. Because DL_MESO is an on going project and new parameters might be added to the package in the future, it is strongly suggested that users of DL_MESO would not name their own parameters with prefix '*lb*', '*dp*' or '*sp*'.

The notation column in table 5.8 gives the restriction of the parameters. '*A*' means the array of data, followed by the number of element in the array. For example, '*A lbsy.nf*' means the parameter is actually an array with lbsy.nf elements. ≥ 1 means the number must be greater or equivalent to one. '*1 or 0*' means the value of the parameter can either be one or zero.

Table 5.6: Domain information

parameter	meaning
lbdm.rank	name of the processor
lbdm.size	number of processor
lbdm.bwid	domain boundary width
lbdm.xcor	x coordinate of the processor
lbdm.ycor	y coordinate of the processor
lbdm.zcor	z coordinate of the processor ($lbdm.zcor \equiv 0$ when $lbsy.nd = 2$)
lbdm.xdim	number of processor along x-axis
lbdm.ydim	number of processor along y-axis
lbdm.zdim	number of processor along z-axis ($lbdm.zdim \equiv 1$ when $lbsy.nd = 2$)
lbdm.xs	x-coordinate of domain start position
lbdm.xe	x-coordinate of domain end position
lbdm.xinner	grid number along x-axis in the domain
lbdm.xouter	grid number along x-axis in the domain plus boundary
lbdm.ys	y-coordinate of domain start position
lbdm.ye	y-coordinate of domain end position
lbdm.yinner	grid number along y-axis in the domain
lbdm.youter	grid number along y-axis in the domain plus boundary
lbdm.zs	z-coordinate of domain start position
lbdm.ze	z-coordinate of domain end position
lbdm.zinner	grid number along z-axis in the domain
lbdm.zouter	grid number along z-axis in the domain plus boundary
lbdm.touter	total grid number in the domain plus in boundary

Table 5.7: Neighbor information

parameter	meaning
lbnb[k].rank	name of the k-neighbour processor
lbnb[k].spos	start position for sending message on distribution function
lbnb[k].rpos	start position for receiving message on distribution function
lbnb[k].bspos	start position for sending message on boundary condition
lbnb[k].brpos	start position for receiving message on boundary condition
k=0	right neighbour
k=1	left neighbour
k=2	upper neighbour
k=3	lower neighbour
k=4	front neighbour
k=5	back neighbour

Table 5.8: DL_MESO_LBE Parameters

function	parameter	data type	notation
system information	lbsy	structure	
domain information	lbdm	structure	
neighbor information	lbnb	structure	A 6
space dimension	lbsy.nd	int	
number of discrete speed	lbsy.nq	int	
number of fluid	lbsy.nf	int	≥ 1
number of solute	lbsy.nc	int	
temperature scalar	lbsy.nt	int	1 or 0
grid number in x direction	lbsy.nx	int	
grid number in y direction	lbsy.ny	int	
grid number in z direction	lbsy.nz	int	
domain boundary width	lbdm.bwid	int	≥ 1
system dimension along x	lbxsize	double	
total running steps	lbtotstep	int	
step for save	lbsave	int	
steering	lbsteer	int	1 or 0
system molar weight	lbsmw	double	
noise intensity	lbnoise	double	
system molar volume	lbsmv	double	
size of embryo	lbemradius	double*	A lbsy.nf
number of embryos	lbemnumber	int*	A lbsy.nf
initial system speed	lbiniv	double	A 3
top boundary speed	lbtov	double	A 3
bottom boundary speed	lbbotv	double	A 3
front boundary speed	lbfrov	double	A 3
back boundary speed	lbbakv	double	A 3
left boundary speed	lblefv	double	A 3
right boundary speed	lbrigv	double	A 3
initial fluid density	lbinip	double*	A lbsy.nf
top boundary fluid density	lbtovp	double*	A lbsy.nf
bottom boundary fluid density	lbbotp	double*	A lbsy.nf
front boundary fluid density	lbfrop	double*	A lbsy.nf
back boundary fluid density	lbbakp	double*	A lbsy.nf
left boundary fluid density	lblefp	double*	A lbsy.nf
right boundary fluid density	lbrigp	double*	A lbsy.nf

Table 5.9: DL_MESO_LBE Parameters-continued

function	parameter	data type	notation
initial composition	lbinic	double *	A lbsy.nc
top boundary composition	lbtopc	double *	A lbsy.nc
bottom boundary composition	lbbotc	double *	A lbsy.nc
front boundary composition	lbfroc	double *	A lbsy.nc
back boundary composition	lbbakc	double *	A lbsy.nc
left boundary composition	lblefc	double *	A lbsy.nc
right boundary composition	lbrigc	double *	A lbsy.nc
initial temperature	lbinit	double	
top boundary temperature	lbtop	double	
bottom boundary temperature	lbbot	double	
front boundary temperature	lbfrot	double	
back boundary temperature	lbbakt	double	
left boundary temperature	lbleft	double	
right boundary temperature	lbrigt	double	
system cooling rate	lbsysdt	double	
top boundary cooling rate	lbtopdt	double	
bottom boundary cooling rate	lbbotdt	double	
front boundary cooling rate	lbfrodt	double	
back boundary cooling rate	lbbakdt	double	
left boundary cooling rate	lblefdt	double	
right boundary cooling rate	lbrigdt	double	
inverse relaxation time fluid	lbt	double *	A lbsy.nf
inverse relaxation time solute	lbt	double *	A lbsy.nc
inverse relaxation time Temp	lbt	double *	A lbsy.nt
particle interaction	lbg	double *	A lbsy.nf*lbsy.nf
body force	lbbdforce	double *	A 3*lbsy.nf
interaction force	lbinterforce	double *	A 3*lbsy.nf
distribution function	lbf	double *	A
temporary function	lbft	double *	A lbdm.touter*(lbsy.nf+lbsy.nc)
equilibrium distribution	lbfeq	double *	A lbsy.nq
speed vector of the model	lbv	int*	A 3*lbsy.nq
index for the opposite speed	lbopv	int*	A lbsy.nq

Table 5.10: DL_MESO_LBE Parameters-continued

function	parameter	data type	notation
weight factor of speed vector	lbw	double *	A lbsy.nq
parameter number on each grid	lbsitlength	int	
lbdm.youter*lbdm.zouter	lbyz	int	
grid distance	lbdx	double	
time step	lbdt	double	
sound speed	lbsoundv	double	
Reynolds number	lbreynolds	double	
name of the processor	lbdm.rank	int	
total number of processor	lbdm.size	int	
x coordinate of the domain	lbdm.xcor	int	
y coordinate of the domain	lbdm.ycor	int	
z coordinate of the domain	lbdm.zcor	int	
number of process along x	lbdm.xdim	int	
number of process along y	lbdm.ydim	int	
number of process along z	lbdm.zdim	int	
domain start position x	lbdm.xs	int	
domain end position x	lbdm.xe	int	
domain start position y	lbdm.ys	int	
domain end position y	lbdm.ye	int	
domain start position z	lbdm.zs	int	
domain end position z	lbdm.ze	int	
inner grid number along x	lbdm.xinner	int	
outer grid number along x	lbdm.xouter	int	
inner grid number along y	lbdm.yinner	int	
outer grid number along y	lbdm.youter	int	
inner grid number along z	lbdm.zinner	int	
outer grid number along z	lbdm.zouter	int	
total grid numbers	lbdm.touter	int	
name of the neighbour	lbnb[].rank	int	
position for sending msg	lbdm[].spos	unsigned long	
position for receiving msg	lbdm[].rpos	unsigned long	
position for sending bmsg	lbdm[].bspos	unsigned long	

Table 5.11: DL_MESO_LBE Parameters-continued

function	parameter	data type	notation
position for receiving bmsg	lbdm[].brpos	unsigned long	
message type	lbmsg2x	MPI_Datatype	
message type	lbmsg2y	MPI_Datatype	
message type	lbmsg3x	MPI_Datatype	
message type	lbmsg3y	MPI_Datatype	
message type	lbmsg3z	MPI_Datatype	
message type	lbbmsg2x	MPI_Datatype	
message type	lbbmsg2y	MPI_Datatype	
message type	lbbmsg3x	MPI_Datatype	
message type	lbbmsg3y	MPI_Datatype	
message type	lbbmsg3z	MPI_Datatype	

Chapter 6

DL_MESO Examples

Test cases for DL_MESO are found in the *DEMO* subdirectory. This contains two further subdirectories: *LBE* and *DPD*.

6.1 The LBE Test Cases

These are found in the *LBE* subdirectory. They consist of the following cases.

6.1.1 2D_Pressure

This is a 2D simulation of a single fluid on a 42x42 grid with a fixed boundary condition.

6.1.2 2D_Shear

This is a 2D simulation of a single fluid on a 42x42 grid with a shear boundary condition.

6.1.3 3D_PhaseSeparation

This is a 3D simulation of a binary fluid on a 100x100x100 grid with a fixed boundary condition, showing phase separation of the fluids.

6.1.4 3D_Shear

This is a 3D simulation of a single fluid on a 40x30x25 grid with a shear boundary condition.

6.2 The DPD Test Cases

These are found in the *DPD* subdirectory. They consist of the following cases.

6.2.1 Dim3_Spe2_Ple4000

This simulation consists of 4000 particles with 2 species. Species 0 has a population of 2400 and species 1 has a population of 1600 particles.

Chapter 7

The DL_MESO_LBE Package Reference

7.1 Overview

DL_MESO_LBE consists of six packages

- lbpMODEL
Contains subroutines to assign lbw, lbv and lbopv for D2Q9, D3Q15, D3Q19 and D3Q27 lattice models.
- lbpBASIC
Contains general purpose subroutines which are applicable to any occasion. For example, to create a random number whose value between -1 and 1.
- lbpGET
Contains subroutines to calculate the macroscopic quantities. For examples, to calculate the macroscopic density, speed and momentum.
- lbpIO
Contains subroutines to read parameter or write the numerical results. For example, read the space definition.
- lbpBOUND
Contains subroutines to treat boundary conditions. For examples, calculate the particle distribution function in a shear boundary.
- lbpSUB
Contains the most important subroutines for lattice-Boltzmann calculation. For example, particle propagation and site collision.

- lbpMPI

Contains all subroutines necessarily for parallel computation. The package can be ignored if user uses only one processor workstation or windows PC.

It is suggested that DL_MESO users to put the subroutines defined by themselves into a package called lbpUSER so that the upgrade of DL_MESO will not interfere with their contribution.

7.2 DL_MESO_LBE Subroutines and Functions

7.2.1 main

Serial code

```
# include "slbe.hpp"
int main(int argc, char* argv[])
{
fDefineSystem();
fSetSerialDomain();
fStartDLMESO();
fMemoryAllocation();
fInputParameters();
fReadSpaceParameter();
fGetModel();
fInitialiseSystem();
fMarkBoundArea3D();
fsOutPutGrid();
for(int i=0; i<= lbtotstep; i++) {
if(i%lbsave == 0)
fsOutPutQ();
fPropagation();
fColliBoundary();
fCollision();
}
fFreeMemory();
fFinishDLMESO();
return 0;
}
```

Parallel code

```
# include "plbe.hpp"
int main(int argc, char** argv)
{
  fStartMPI(argc, argv);
  fDefineSystem("lbin.sys");
  fDefineDomain();
  fStartDLMESO();
  fMemoryAllocation();
  fDefineNeighbours();
  fDefineMessage();
  fInputParameters("lbin.sys");
  fReadSpaceParameter("lbin.spa");
  fBoundNonBlockCommunication();
  fAllReady();
  fGetModel();
  fInitialiseSystem();
  fOutPutInfo();
  fOutPutGrid();
  fMarkBoundArea3D();
  for(int i=0; i<= lbtotstep; i++) {
    fNonBlockCommunication();
    if(ifOutPutQ());
    fPropagation();
    fColliBoundary();
    fCollision();
    fAllReady();
  }
  fFreeMemory();
  fFinishDLMESO();
  fCloseMPI();
  return 0;
}
```

7.2.2 lbpMODEL

D2Q9

- Header records
int D2Q9()
- Function
Assign the weight factor lbw , speed vector lbv and the index for oposite speed vector $lbopv$ for D2Q9 lattice model
- Dependencies
None

D3Q15

- Header records
int D3Q15()
- Function
Assign the weight factor lbw , speed vector lbv and the index for oposite speed vector $lbopv$ for D3Q15 lattice Model
- Dependencies
None

D3Q19

- Header records
int D3Q19()
- Function
Assign the weight factor lbw , speed vector lbv and the index for oposite speed vector $lbopv$ for D3Q19 lattice Model
- Dependencies
None

D3Q27

- Header records
int D3Q15()

- Function
Assign the weight factor lbw , speed vector lbv and the index for opposite speed vector $lbopv$ for D3Q27 lattice Model
- Dependencies
None

7.2.3 lbpBASIC

- Package header records


```

/*****DL-MESO-LBE PACKAGE REFERENCE*****/
                DL-MESO Version 1.2
                Author   : R.S. Qin
                Copyright : Daresbury Laboratory
                        : 14/02/2005
                lbpBASIC.cpp
                *****/
/*
 * functions in this package are for general purpose. It has nothing
 * to do with Mesoscale Methods. If you found a function with same
 * purpose in C++ standard libeary, please don't hesitate to replace it.
 */

```

fCppAbs

- Header records


```

template <class T>
T fCppAbs(T a)

```
- Function
Calculate absolute value of numerical number a .
- Dependencies
None
- Arguments

a	input	any datatype
fCppAbs	output	the same data type as a

fGetNumberOrdered

- Header records

case 1:

```
int fGetNumberOrdered(int &iiox, int &iioy, int &iioz)
{
```

```
// to arrange three integers in a decreasing order
```

case 2:

```
int fGetNumberOrdered(int &iiox, int &iioy)
{
```

```
// to arrange two integers in a decreasing order
```

- Function

Rearrange the integers so that they appear in the decreasing order

- Dependencies

None

- Arguments

iox input and output integer reference

ioy input and output integer reference

ioz input and output integer reference

fGetNumberOrderFixed

- Header records

case 1:

```
int fGetNumberOrderFixed(int &iiox, int &iioy, int &iioz, int ix, int iy, int iz)
{
```

```
// put iiox, iioy and iioz in the order same to ix, iy and iz
```

case 2:

```
int fGetNumberOrderFixed(int &iiox, int &iioy, int ix, int iy)
{
```

```
// put iiox and iioy in the order same to ix and iy
```

- Function

Rearrange a set of intergers so that they appear in the order same as another set of integers.

- Dependencies

fGetNumberOrdered

- Arguments

iox	input and output	integer reference
ioy	input and output	integer reference
ioz	input and output	integer reference
ix	input	integer
iy	input	integer
iz	input	integer

fBestGrouping

- Header records


```
int fBestGrouping(int totalgrid, int totalgroup, int& indigrid, int& critigroup)
{
// when group < criticalgroup, grid number = indigrid
// when group ≥ criticalgroup, grid number =indigrid-1
```
- Function

Distribute grids to processes so that the maximum difference of the number of grids among processes is 1.
- Dependencies

None
- Arguments

totalgrid	input	integer
totalgroup	input	integer
indigrid	output	integer reference
critigroup	output	integer reference
- Comments

The totalgrid grids are distributed into totalgroup processes so that the first criticalgroup processes have indigrid grids and the other processes have indigrid-1 grids.

fCppMod

- Header records


```
cast 1:
int fCppMod(int a, int b)
{
// make sure a integer number a is in a range between 0 and b-1
cast 2:
```

```
int fCppMod(long a, long b)
{
// make sure a long integer number a is in a range between 0 and b-1
```

- Function

Tie the head and tail together, which means that the position next to the maximum equals to the minimum, and vice versa.

- Dependencies

None

- Arguments

```
a          input  integer
b          input  integer
fCppMod    output integer
```

- Comments

$fCppMod = 0$ when $a = b$ or $fCppMod = b - 1$ when $a = -1$. This function is useful for periodic boundary condition.

fPrintLine and fPrintDoubleLine

- Header records

```
int fPrintLine()
int fPrintDoubleLine()
```

- Function

Print a line of 72 character length.

- Dependencies

None

fRandom

- Header records

```
double fRandom()
{
// to create a random number with value between -1 and 1
```

- Function

Create a double precision random number between -1 and 1

- Dependencies

None

- Arguments
fRandom output double precision
- Comments
There is a build in seed in the function. The seed is only activated when the function is firstly called.

7.2.4 lbpGET

fGetNodePosi

- Header records
case 1 for 3D:
inline long fGetNodePosi(int xpos, int ypos, int zpos)
case 2 for 2D:
inline long fGetNodePosi(int xpos, int ypos)
- Function
Calculate the grid position in 1D array from its cartesian coordinate
- Dependencies
None
- Arguments

xpos	input	integer
ypos	input	integer
zpos	input	integer
fGetNodePosi	output	long
- Comments
The calculation follows C++ data structure.

fGetCoord

- Header records
int fGetCoord(long tpos, int& xpos, int& ypos, int& zpos)
- Function
Calculate the cartesian coordinate from its grid position in 1D array
- Dependencies
None

- Arguments

xpos	output	integer reference
ypos	output	integer reference
zpos	output	integer reference
tpos	input	long integer

fGetOneMassSite

- Header records


```
double fGetOneMassSite(double* startpos)
{
// get one mass started from startpos in lbf[] array
```
- Function

Calculate the mass density of one of the fluids at a grid
- Dependencies

None
- Arguments

startpos	input	double pointer
fGetOneMassSite	output	double
- Comments

Mass density is calculated according to the definition of mass density $\rho = \sum f_i$. The 'startpos' is the start point for the summation of particle distribution functions and must be assigned correctly. There are two other subroutines achieving the same purpose:

1. double fGetOneMassSite(int fpos, long tpos)

where 'fpos' means the fpos-th fluid, 'tpos' means the tpos-th grid.

2. double fGetOneMassSite(int fpos, int xpos, int ypos, int zpos)

where 'fpos' means the fpos-th fluid, 'xpos', 'ypos' and 'zpos' are cartesian coordinate of the grid.

Those two functions are more readable but a little slower.

fGetTotMassSite

- Header records


```
double fGetTotMassSite(double* startpos)
```

```
{
// get total mass started from startpos in lbf[] array
```

- Function
 - Calculate the mass density of all fluid at a grid
- Dependencies
 - None
- Arguments

startpos	input	double pointer
fGetTotMassSite	output	double
- Comments

The calculation is based on the definition of $\rho = \sum_i f_i$. There is another subroutine achieving the same purpose:

```
double fGetTotMassSite(long tpos)
where tpos means the tpos-th grid.
```

fGetOneMassDomain

- Header records


```
double fGetOneMassDomain(int fpos)
{
//get total mass in the domain
```
- Function
 - Calculate the total mass of fpos-th fluid through the domain
- Dependencies
 - None
- Arguments

fpos	input	integer
fGetOneMassDomain	output	double precision
- Comments

In the parallel calculation, the total mass of the domain does not include those boundary areas which are used for message passing.

fGetFracSite

- Header records

```
double fGetFracSite(int fpos, double* startpos)
{ // get mass fraction of fpos phase in the site started
  // from startpos in lbf[] array
```

- Function

Calculate the mass fraction of fluid fpos in the site.

- Dependencies

None

- Arguments

```
fpos      input integer
startpos  input          double pointer
```

- Comments

The calculation is based on $c = \rho_i / \sum_i \rho_i$. There is another subroutine which is slightly slower but more readable

```
double fGetFracSite(int fpos, long tpos)
```

fGetOneSpeedSite

- Header records

```
int fGetOneSpeedSite(double *speed, double* startpos)
{
  // speed of one fluid start from startpos
```

- Function

Calculate the macroscopic speed of one fluid in the grid.

- Dependencies

None

- Arguments

```
speed     output  array of double
startpos  input   double pointer
```

- Comments

The calculation is based on $v_\alpha = \sum f_i e_{i\alpha} / \rho$. There are two other more readable functions but are slightly slower.

1. `int fGetOneSpeedSite(double *speed, int fpos, int xpos, int ypos, int zpos);`
2. `int fGetOneSpeedSite(double *speed, int fpos, long tpos);`

fGetSpeedSite

- Header records


```
int fGetSpeedSite(double *speed, double* startpos)
{
// site speed of fluid start from startpos in lbf[] array
```
- Function

Calculate the fluid speed in the grid
- Dependencies

None
- Arguments

speed	output	array of double
startpos	input	double pointer
- Comments

$v_\alpha = (\sum \sum f_i e_{i\alpha}) / (\sum \rho)$. Other subroutines with same purpose are:

 1. `int fGetSpeedSite(double *speed, long tpos)`
 2. `int fGetSpeedSite(double *speed, int xpos, int ypos, int zpos)`

fGetOneMomentSite

- Header records


```
int fGetOneMomentSite(double *speed, double* startpos)
{
// momentum of one fluid start from startpos
```
- Function

Calculation of momentum of one of the fluids.
- Dependencies

None
- Arguments

speed	output	array of double
startpos	input	double pointer

- Comments

$K_i = \sum f_i e_{i\alpha}$. Other subroutines with same purpose are:

1. `int fGetOneMomentSite(double *speed, int fpos, long tpos)`
2. `int fGetOneMomentSite(double *speed, int fpos, int xpos, int ypos, int zpos)`

fGetTotMomentSite

- Header records

```
int fGetTotMomentSite(double *momen, double* startpos)
{
// momentum of grid start from startpos
```

- Function

Calculate grid momentum.

- Dependencies

None

- Arguments

momen output array of double
startpos input double pointer

- Comments

momen[0] is the momentum along x-axis, momen[1] is the momentum along y-axis, and momen[2] is the momentum along z-axis.

fGetTotMomentDomain

- Header records

```
int fGetTotMomentDomain(double *momentum)
{
//get total momentum in the domain
```

- Function

Calculate the momentum in the domain.

- Dependencies

fGetTotMomentSite

- Arguments

momentum output array of double

- Comments

This function is mainly used to check is the domain momentum along each axis is conserved.

fGetOneDirecSpeedSite

- Header records
float fGetOneDirecSpeedSite(int dire, double* startpos)
- Function
Calculate the grid speed along one of directions: 0 for x, 1 for y and 2 for z.
- Dependencies
None
- Arguments
dire input integer
startpos input double pointer
- Comments
Mainly used for output grid speed.

7.2.5 lbpIO**fDefineSystem**

- Header records
int fDefineSystem(char* filename = "lbin.sys")
{
/*
DL_MESO_LBE
Author : R.S. Qin
Copyright : Daresbury Laboratory
: 05/11/2004
/
// get system basic information
- Function
Read parameters in System package from input system file lbin.sys
- Dependencies
lbin.sys data file.
- Arguments
filename input array of characters
- Comments
The default file name is lbin.sys. You can change the argument as other name of the system datafile.

fInputParameters

- Header records

```
int fInputParameters(char* filename="lbin.sys")
{
// get system basic information
```
- Function
Read system parameters from lbin.sys datafile.
- Dependencies
lbin.sys data file
- Arguments
filename input array of characters
- Comments
The default file name is lbin.sys. You can change the argument as other name of the system datafile.

fReadSpaceParameter

- Header records

```
int fReadSpaceParameter(char* filename="lbin.spa")
```
- Function
Read 2D or 3D space parameters from datafile lbin.spa
- Dependencies

```
int fReadSpace2D(char* filename="lbin.spa")
int fReadSpace3D(char* filename="lbin.spa")
```
- Arguments
filename input array of characters
- Comments
The default file name is lbin.spa. You can change the argument to other name of the space datafile.

fSetoffSteer

- Header records

```
int fSetoffSteer()
{
```

```

/*
setup steer
/

```

- **Function**
to create a file with filename of "notsteer" so that DL_MESO_LBE code will not read new lbin.sys and lbin.spa files again if "notsteer" is existing.
- **Dependencies**
None
- **Comments**
You may or may not change the input datafiles. If you have changed, the code will run with new parameters.

fCheckSteer

- **Header records**
int fCheckSteer()
{
check if user want to steer
- **Function**
Check the existance of "notsteer" files. If true, read lbin.sys and lbin.spa files.
- **Dependencies**
fInputParameters("lbin.sys");
fReadSpaceParameter("lbin.spa");

fOutPutGrid3D

- **Header records**
int fOutPutGrid3D(char* filename="lbout")
- **Function**
Output grid position of 3D system.
- **Arguments**
lbout input array of characters
- **Comments**
The output file is named "lbout.xyz"

fOutPutQ3D

- Header records
int fOutPutQ3D(char* filename="lbout")
- Function
Output the macroscopic quantities mass density of fluid 0, grid speed along x, y and z directions, and space definition of 3D system.
- Arguments
lbout input array of characters
- Comments
The output file is named "lbout.q"

fOutPutCA3D

- Header records
int fOutPutCA3D(char* filename="lbout", int iprop=0)
- Function
Output the macroscopic quantities mass fraction of fluid iprop, grid speed along x, y and z directions, and space definition of 3D system.
- Arguments
lbout input array of characters
iprop input integer
- Comments
The output file is named "lbout.q"

fOutPutCB3D

- Header records
int fOutPutCB3D(char* filename="lbout", int iprop=0)
- Function
Output the macroscopic quantities of composition of solute iprop, grid speed along x, y and z directions, and space definition of 3D system.
- Arguments
lbout input array of characters
iprop input integer
- Comments
The output file is named "lbout.q"

fOutPutT3D

- Header records

```
int fOutPutT3D(char* filename="lbout")
```
- Function
 Output the macroscopic quantities of temperature, grid speed along x, y and z directions, and space definition of 3D system.
- Arguments

```
lbout  input  array of characters
```
- Comments
 The output file is named "lbout.q"

Other subroutines for output .q

- The subroutines with name of fOutPut*2D mean to output same parameters for a 2D system.
- The subroutines with name of fsOutPut*D is suitable for computer with only one processor.
- lbout2.q means the q file at second saving step
- lbout2at15 means the q file at second saving step and written by process 15.

7.2.6 lbpBOUND**fNextStep**

- Header records

```
long fNextStep(int q, int xpos, int ypos, int zpos)
{
  // moving the particle along q direction to its nearest neighbour
}
```
- Function
 find the particle position at next time step when is is now in (xpos, ypos, zpos) and moving along q-direction.
- Arguments

```
q          input  integer
xpos       input  integer
ypos       input  integer
zpos       input  integer
fNextStep  output  long integer
```

- Comments

For 2D system, the subroutine is

```
long fNextStep(int q, int xpos, int ypos)
```

fMoveNonzeroAway

- Header records

```
int fMoveNonzeroAway(long tpos)
```

- Function

Move those particle distribution functions at non-fluid site (the tpos-th grid) to their next neighbours.

- Arguments

```
tpos  input  long integer
```

- Comments

With this step, the full way bounce back will become the half way bounce back condition.

fBounceBackF

- Header records

```
int fBounceBackF(long tpos)
```

```
{ /*
```

```
set distribution function on a site with fullway bounce back
```

```
/
```

- Function

Perform the full way bounce back at the tpos-th grid

- Dependencies

None

- Arguments

```
tpos  input  long integer
```

- Comments

$$f(lbopv[i]) = f(lbv[i])$$

fBounceBackC

- Header records

```
int fBounceBackC(long tpos)
```

```
{ /*
set distribution function on a site with fullway bounce back
/
```

- Function

Perform the full way bounce back at the tpos-th grid
- Dependencies

None
- Arguments

tpos input long integer
- Comments

$f(lbopv[i]) = f(lbv[i])$

fBounceBackT

- Header records


```
int fBounceBackT(long tpos)
{ /*
set distribution function on a site with fullway bounce back
/
```
- Function

Perform the full way bounce back at the tpos-th grid
- Dependencies

None
- Arguments

tpos input long integer
- Comments

$f(lbopv[i]) = f(lbv[i])$

fSiteBlankF

- Header records


```
int fSiteBlankF(long tpos)
{
// set distribution function on a site inside a solid block
```

- Function
Set particle distribution functions into zero.
- Dependencies
None
- Arguments
tpos input long integer
- Comments
It is quite useful sometimes. For example, the flow inside a solid boundary is negligible.

fSiteBlankC

- Header records
int fSiteBlankC(long tpos)
- Function
Set particle distribution functions into zero.
- Dependencies
None
- Arguments
tpos input long integer
- Comments
It is quite useful sometimes. For example, the solute diffusion inside a bulk solid when compare with the diffusion in liquid.

fSiteBlankT

- Header records
int fSiteBlankT(long tpos)
- Function
Set particle distribution functions into zero.
- Dependencies
None
- Arguments
tpos input long integer
- Comments
It is quite useful sometimes. For example, the heat transfer in the insulator.

fFixedSpeedFluid

- Header records

```
int fFixedSpeedFluid(int tpos, int prop)
{
/*
suitable for boundary with fixed speed, fluid only
/
```

- Function

Calculate the particle distribution function at fixed speed boundary.

- Arguments

```
tpos  input  integer
prop  input  integer
```

- Comments

For planar surface, calculations are based on the theory reported in Zou QS, He XY, On pressure and velocity boundary conditions for the lattice Boltzmann BGK model, PHYS FLUIDS 9 (6): 1591-1598 JUN 1997.

For concave edge and concaved corner, equilibrium boundary conditions are used and the density on the edge and site is assumed to equal to the values at their nearest neighbors in bulk liquid.

fFixedDensityFluid

- Header records

```
int fFixedDensityFluid(int tpos, int prop)
{
/*
suitable for boundary with fixed pressure
/
```

- Function

Calculate the particle distribution function at fixed density boundary.

- Arguments

```
tpos  input  integer
prop  input  integer
```

- Comments

For planar surface, calculations are based on the theory reported in Zou QS, He XY, On pressure and velocity boundary conditions for the lattice Boltzmann

BGK model, PHYS FLUIDS 9 (6): 1591-1598 JUN 1997.

For concave edge and concaved corner, zero boundary speed is assumed.

fFixedSoluteConcen

- Header records


```
int fFixedSoluteConcen(int tpos, int prop)
{
/*
uitable for boundary with fixed solute concentration
/
```
- Function

Calculate the particle distribution function at fixed composition boundary.
- Arguments


```
tpos  input  integer
prop  input  integer
```
- Comments

For planar surface, calculations are based on the theory reported in
Zou QS, He XY, On pressure and velocity boundary conditions for the lattice Boltzmann
BGK model, PHYS FLUIDS 9 (6): 1591-1598 JUN 1997.
For concave edge and concaved corner, zero boundary speed is assumed.

fFixedTemperature

- Header records


```
int fFixedTemperature(int tpos, int prop)
{
/*
uitable for boundary with fixed temperature
/
```
- Function

Calculate the particle distribution function at fixed temperature boundary.
- Arguments


```
tpos  input  integer
prop  input  integer
```
- Comments

For planar surface, calculations are based on the theory reported in

Zou QS, He XY, On pressure and velocity boundary conditions for the lattice Boltzmann BGK model, PHYS FLUIDS 9 (6): 1591-1598 JUN 1997.

For concave edge and concaved corner, zero boundary speed is assumed.

fColliBoundary

- Header records
int fColliBoundary()
- Function
Calculate the particle distribution function at different boundary
- Dependencies
Many
- Comments
The user can always add their own algorithm for other boundary conditions.

7.2.7 lbpSUB

fWeekMemory

- Header records
inline void fWeekMemory()
- Function
To terminate calculation if system has not enough memory
- Dependencies
None
- Comments
It will print the error information of that
”-error, can’t allocate more memory.”

fMemoryAllocation

- Header records
int fMemoryAllocation()
{
/*
get memory allocated for lattice Boltzmann model
/

- Function
allocate memory for all parameters
- Dependencies
fWeekMemory()
- Comments
error message would appear and calculation stop if the memory allocation were not successful.

fFreeMemory

- Header records
int fFreeMemory()
{
/*
free all allocated memory
/
- Function
Free allocated memory
- Dependencies
None
- Comments
None

fSetSerialDomain

- Header records
int fSetSerialDomain()
- Function
To set the domain parameter for serial computer
- Dependencies
None
- Comments
None

fStartDLMESO

- Header records
int fStartDLMESO()
- Function
To announce that DL_MESO_LBE starts to run
- Comments
You can always ignore this subroutine if you prefer

fFinishDLMESO

- Header records
int fFinishDLMESO()
- Function
To announce that running of DL_MESO_LBE is finished
- Comments
You can always ignore this subroutine if you prefer

fGetModel

- Header records
int fGetModel()
- Function
initialize lbv, lbw, lbopv
- Comments
the parameters are specified according to the space dimension and number of discrete velocity.

fGetEquilibriumF

- Header records
int fGetEquilibriumF(double *feq, double *v, double rho)
{
/*
calculate equilibrium distribution function
suitable only for square lattice
not suitable for incompressible fluid

/

- Function
Calculate equilibrium distribution function for compressible fluid
- Dependencies
None
- Arguments

feq	output	double pointer
v	input	double point
rho	input	double
- Comments

$$f^{eq} = w_i \rho \left[1 + \frac{3(e_i \cdot u)}{c^2} + \frac{9(e_i \cdot u)^2}{2c^4} - \frac{3u^2}{2c^2} \right]$$
 If you want to change the equilibria to suitable other models, this is the place you must do.

fGetEquilibriumC

- Header records
int fGetEquilibriumC(double *feq, double *v, double rho)
- Function
Calculate equilibrium distribution function for solute
- Dependencies
None
- Arguments

feq	output	double pointer
v	input	double point
rho	input	double
- Comments
You can always change this subroutine to suitable to other lattice Boltzmann models. For example, change into Free Energy Model.

fGetEquilibriumT

- Header records
int fGetEquilibriumT(double *feq, double *v, double tem)
- Function
Calculate equilibrium distribution function for temperature

- Dependencies

None

- Arguments

```

feq  output  double pointer
v    input   double point
tem  input   double

```

- Comments

You can always change this subroutine to suitable to other lattice Boltzmann models. I have found following papers useful

1. Inamuro T, Yoshino M, Inoue H, Mizuno R, Ogino F, A lattice Boltzmann method for a binary miscible fluid mixture and its application to a heat-transfer problem, J COMPUT PHYS 179 (1): 201-215 JUN 10 2002.

2. Yoshino M, Inamura T, Lattice Boltzmann simulations for flow and heat/mass transfer problems in a three-dimensional porous structure, INT J NUMER METH FL 43 (2): 183-198 SEP 20 2003.

fGetEquilibriumIncom

- Header records

```

int fGetEquilibriumIncom(double *feq, double *v, double rho, double rho0)
{
/*
calculate equilibrium distribution function
suitable only for squre lattice
suitable only for incompressible fluid
/

```

- Function

Calculate equilibrium distribution function for incompressible fluids

- Dependencies

None

- Arguments

```

feq  output  double pointer
v    input   double point
rho  input   double
rho0 input   double

```

- Comments

$$f^{eq} = w_i \left[\rho + \rho_0 \frac{3(e_i \cdot u)}{c^2} + \frac{9(e_i \cdot u)^2}{2c^4} - \frac{3u^2}{2c^2} \right]$$

Users are recommended to read the paper

He xiaoyi and Luo Lishi, Lattice Boltzmann Model for incompressible Navier-Stokes Equation

fInitialiseSystem

- Header records

```
int fInitialiseSystem()
{
// initialise system
```

- Function

Initialize logical frame

- Comments

It is suggested to always change this subroutine so that more fast, stable and reasonable initials are set. However, you can use the default setting to achieve many of your purposes.

fSiteCollision

- Header records

```
int fSiteCollision(double* startpos, double* bodyforce)
{
// BGK single relaxation time
```

- Function

Calculation of collision at a grid.

- Arguments

```
startpos    input    double pointer
bodyforce   input    double pointer
```

- Comments

$$f_i(x_i, t) = f_i^{eq}(x_i, t) - \tau_f (\partial_t + e_{i\alpha} \partial_\alpha) f_i^{eq}(x, t)$$

You can change this subroutine to other relaxation regimes such as multy relaxation time method.

fCollision

- Header records
int fCollision()
- Function
Collision steps for all fluid.
- Comments
It is not suggested to change this subroutine/

fMarkBoundArea3D

- Header records
int fMarkBoundArea3D()
{
// this is mainly used for parallel code
- Function
To denote where the boundary areas for passing message are.
- Dependencies
None
- Comments
It is only for parallel computer.

fCalcPotential_ShanChen

- Header records
int fCalcPotential_ShanChen()
{
// Shan-Chen Model: $\phi = 1 - \exp(-\rho)$
- Function
Calculate the interaction potential suggested in Shan_Chen model
- Comments
It is a place to introduce new mesoscale interactions. Recommended to read
1. SHAN XW, CHEN HD, LATTICE BOLTZMANN MODEL FOR SIMULATING FLOWS WITH MULTIPLE PHASES AND COMPONENTS, PHYS REV E 47 (3): 1815-1819 MAR 1993.

2. SHAN XW, CHEN HD, SIMULATION OF NONIDEAL GASES AND LIQUID-GAS PHASE-TRANSITIONS BY THE LATTICE BOLTZMANN-EQUATION, PHYS REV E 49 (4): 2941-2948 Part A APR 1994.

fCalInteraction

- Header records
int fCalInteraction(int xpos, int ypos, int zpos)
- Function
Calculation of particle interaction forces
- Arguments
xpos input integer
ypos input integer
zpos input integer
- Comments
The detail algorithm can be found in
SHAN XW, CHEN HD, LATTICE BOLTZMANN MODEL FOR SIMULATING FLOWS WITH MULTIPLE PHASES AND COMPONENTS, PHYS REV E 47 (3): 1815-1819 MAR 1993.

7.2.8 lbpMPI

This package is only for parallel computer. Even for those parallel DL_MESO_LBE user, it is not really need to read or understand. JUST USE IT.

Many subroutines in this package is not described in the manu. Users with special interest are recommended to read the code.

fStartMPI

- Header records
int fStartMPI(int argc, char* argv[])
{
// start message passing interface
- Function
Start Message Passing Interface

fCloseMPI

- Header records

```
int fCloseMPI()
{
// close message passing environment
```
- Function
Close Massager Passing Interface.

fGlobalValue

- Header records

```
int fGlobalValue(double *vqua, int nnum, double *vtot)
{
// combine values from all processes and distribute to all
```
- Comments There are manu different formats of arguments included

```
int fGlobalValue(double *vqua, int nnum)
int fGlobalValue(int *vqua, int nnum, int *vtot)
int fGlobalValue(int *vqua, int nnum)
int fGlobalValue(long int *vqua, int nnum)
int fGlobalValue(long int *vqua, int nnum, long int *vtot)
int fGlobalProduct(double *vqua, int nnum)
int fGlobalProduct(int *vqua, int nnum)
```

fArrangeProcessors

- Header records

```
int fArrangeProcessors()
{
// arrange processes according to the system and space dimensions
```
- Function
Arrange processors according system dimension
- Comments
Calculations are based on

$$\frac{lbdm.xdim}{lbsy.nx} \simeq \frac{lbdm.ydim}{lbsy.ny} \simeq \frac{lbdm.zdim}{lbsy.nz}$$

$$lbdm.xdim \times lbdm.ydim \times lbdm.zdim = lbdm.size$$

fDefineDomain

- Header records
int fDefineDomain()
{
// to specify domain parameters

fDefineMessage

- Header records
int fDefineMessage()
{
//define vectors
- Function
Define vector messages

fDefineNeighbours

- Header records
int fDefineNeighbours()
{
// calculate parameters for lbnb[] objects
// 0- i +x, 1- i -x, 2- i +y, 3- i -y, 4- i +z, 5- i -z
- Function
Calculate the names of neighbours, the message sending and receiving start points.
- Comments
Strictly not allow to change.

fNonBlockCommnication

- Header records
int fNonBlockCommnication()
{
// passing boundary information for either 2d or 3d system

fOutPutInfo

- Header records
int fOutPutInfo()

```
{
// to get system information
```

- Function
Output length of integer and float, and number of processes
- Comments
This is subroutine is important for rearrange the lbout data.

fBoundNonBlockCommnication

- Header records
int fBoundNonBlockCommnication()
{
// passing boundary information for either 2d or 3d system
- Function
passing space information to those boundary areas for message passing.

7.3 DL_MESO_LBE Data Files

7.3.1 Description of the INPUT files

Define system – lbin.sys

It is suggested to use DL_MESO_GUI to construct the lbin.sys file. However, you can also edit the existing lbin.sys. You can delete those parameters you do not need. However, if you want to introduce any new parameter, make sure to add the parameter recognition sentence to fInputParameters subroutine.

Define space – lbin.spa

It is suggested to use DL_MESO_GUI to construct the lbin.spa file. The data stored in lbin.spa are in a format of

xcoordinate, ycoordinate, zcoordinate, gridproperty

An empty lbin.spa represents the periodic boundary conditions applies.

7.3.2 Description of the OUTPUT files

DL-MESO-LBE writes standard Plot3D Data Format files. For parallel computer, the data are in binary. For serial computer, the output data are in ANSI. You can always change the format to meet your need.

Out put grid position – lbout.xyz

nx, ny, nz

$x_{0,0,0}, \dots, x_{nx-1,ny-1,nz-1}$

$y_{0,0,0}, \dots, y_{nx-1,ny-1,nz-1}$

$z_{0,0,0}, \dots, z_{nx-1,ny-1,nz-1}$

nx = total number of grid point in x direction

ny = total number of grid point in y direction

nz = total number of grid point in z direction

x, y, z = coordinate in Cartesian axis

Out put macroscopic quantities – lbout.q

nx, ny, nz

sound_speed, 1.0, Reynolds_number, Time_step

$\rho_{0,0,0}, \dots, \rho_{nx-1,ny-1,nz-1}, U_{0,0,0}, \dots, U_{nx-1,ny-1,nz-1}$

$V_{0,0,0}, \dots, V_{nx-1,ny-1,nz-1}, W_{0,0,0}, \dots, W_{nx-1,ny-1,nz-1}$

$\phi_{0,0,0}, \dots, \phi_{nx-1,ny-1,nz-1}$

ρ = density

U = velocity along x direction

V = velocity along y direction

W = velocity along z direction

ϕ = space properties

Chapter 8

The DL_MESO_DPD Package Reference

To be written.