WEST UNIVERSITY OF TIMISOARA

PHYSICS DEPARTMENT

MASTER THESIS

Supervisors:

Prof. univ. dr. Vizman Daniel

Dr. Sofonea Victor CS I

Author:

Biciusca Tonino

Timisoara

2013

WEST UNIVERSITY OF TIMISOARA

PHYSICS DEPARTMENT

LATTICE BOLTZMANN MODELS FOR MULTIPHASE SYSTEMS

Supervisors:

Prof. univ. dr. Vizman Daniel

Dr. Sofonea Victor CS I

Author:

Biciusca Tonino

Timisoara

2013

Contents

1 General theory of lattice Boltzmann **4**

1.1 Boltzmann equation 4

1.2 Chapman - Enskog method 5

1.3 Conservation equations 7

1.4 Discretization of the momentum space 7

1.5 Gauss-Hermite lattice Boltzmann models 9

1.6 Implementation of the non-ideal equation of state 13

2 Numerical algorithms for lattice Boltzmann models **14**

2.1 First order upwind 14

2.2 Flux limiter schemes 15

2.3 Corner transport upwind 17

2.4 Parallel computing using PETSc 21

2.5 Parallel computing using GPUs 24

3 Computing results **28**

3.1 Minkowski functionals 28

3.2 Simulation of phase separation in two - dimensional systems at constant temperature 33

3.2.1 Plane interface 33

3.2.2 Dynamics of phase separation 38

3.3 Three dimensional simulation 40

3.4 Performance studies 42

4 Conclusion **43**

A Appendix **44**

A.1 Hermite polynomials and Gauss-Hermite quadrature 44

ACKNOWLEDGEMENTS

I would like to thank to Dr. Victor Sofonea from Romanian Academy for his guidance and support throughout this process of completing the master thesis. I appreciate his confidance in me over the last year and for the time that he invested in assisting me. Furthermore I would like to thank Professor dr. Daniel Vizman for introducing me to the topic as well for the support on the way.

1 General theory of lattice Boltzmann

1.1 Boltzmann equation

The kinetic theory of Boltzmann, which connects fluid dynamics to thermodynamics and statistical physics, has been a milestone in the development of theoretical physics. In order to describe the kinetics of, e.g., an atomic gas, Boltzmann introduced, with great intuition, more than half century before the rise of quantum mechanics, a probabilistic description for the evolution of a single-particle distribution, which anticipated atomistic scattering concepts. In the kinetic theory we are interested to the dynamics of systems containing a large numbers of particle that interact during collisions. Because it is impossible to track particles individually, and since in practice one is more interested in the behavior of macroscopic state, we will concentrate on the particle number distribution function $f(\boldsymbol{x}, \boldsymbol{\xi}, t)$. The number of particles located in the volume d^3x around position x and which have a velocity located in the volume $d^3\xi$ around ξ at the time t, is defined as:

$$f(\boldsymbol{x}, \boldsymbol{\xi}, t) d^3x d^3\xi \tag{1.1}$$

If we take into account various moments of f, we will have the three well known conservation equation for mass, momentum and energy :

$$\rho(\mathbf{x}, t) = \int m f(\mathbf{x}, \xi, t) d\xi \tag{1.2}$$

$$\rho(\mathbf{x}, t) u(\mathbf{x}, t) = \int m \xi f(\mathbf{x}, \xi, t) d\xi \tag{1.3}$$

$$\rho(\mathbf{x}, t) e(\mathbf{x}, t) = \frac{1}{2} \int c^2 f(\mathbf{x}, \xi, t) dc \tag{1.4}$$

where m is the molecular mass, $\mathbf{c} = \xi - \mathbf{u}(\mathbf{x}, t)$ is the is the peculiar particle velocity vector and $c^2 = |\mathbf{c}|^2$ c .

In the presence of an external field $\mathbf{F}$, the time evolution of f is given by the well-known Boltzmann equation [1],

$$(\partial_t + \xi \cdot \nabla_x + \frac{\mathbf{F}}{m} \cdot \nabla_\xi) f(\mathbf{x}, \xi, t) = \Omega \tag{1.5}$$

where the collision operator Ω, represents the rate of change of the distribution function, due to interparticle collision. For a system not subjected to external forces, the Boltzmann equation becomes [2],

$$\partial_t + \xi \cdot \nabla_x f = \Omega \tag{1.6}$$

(note that ξ and ∇f are vectors). The Boltzmann equation represents an advection equation with a source term Ω and is very difficult to solve.

The collision term Ω in the Boltzmann equation is usually linearized using the Bhatnagar-Gross-Krook (BGK) approximation [3] after introducing a relaxation time τ. In this way, Equation (1.5) becomes the Boltzmann-BGK equation:

$$(\partial_t + \xi \cdot \nabla_x + \frac{\mathbf{F}}{m} \cdot \nabla_\xi) f(\mathbf{x}, \xi, t) = -\frac{1}{\tau} \left[f(\mathbf{x}, \xi, t) - f^{eq}(\mathbf{x}, \xi, t) \right] \tag{1.7}$$

where f^{eq} is the equilibrium distribution, known as the Maxwell-Boltzmann distribution function [4] :

$$f^{eq} = \rho \left(\frac{m}{2\pi k_B \theta} \right)^{\frac{3}{2}} \exp \left[-\frac{m}{2k_B \theta} \left(\xi - \mathrm{u}(\mathrm{x}, t) \right)^2 \right] \tag{1.8}$$

1.2 Chapman - Enskog method

The Chapman-Enskog method relies on the expansion of the distribution function f with respect to the Knudsen number

$$\mathrm{Kn} = \frac{\lambda}{L} \tag{1.9}$$

where λ is the mean free path of particles and L is the characteristic length of the fluid system. The Knudsen number provides a measure of the degree of deviation of the Boltzmann distribution from its local equilibrium. In this work, we restrict ourselves to small values of Kn (typically $Kn < 0.1$). According to Chapman and Cowling [4], we cant expand f asymptotically in powers of the Kundsen number Kn as follows

$$f = \sum_{i=0}^{\infty} Kn^i f^{(i)} \tag{1.10}$$

where $f^{(0)}$ is the Maxwell-Boltzmann equilibrium distribution function. Furthermore, the time and spatial variations are also scaled using the power of Kn

$$\partial_t = Kn\ \partial_t^{(0)} + Kn^2\ \partial_t^{(1)} + \ldots \quad and \quad \nabla = Kn\ \nabla \tag{1.11}$$

Introduction Eq. (1.10) into Eqs. (1.2) - (1.4) and matching the terms according to various powers of Kn gives

$$f^{(0)} = f^{eq} \tag{1.12}$$

as well as a number of useful relations involving the moments of functions $f^{(l)}, l > 0$

$$\int f^{(l)}\ d^D\xi = 0 \quad \forall l > 0 \tag{1.13}$$

$$\int f^{(l)} \xi_\alpha\ d^D\xi = 0 \quad \forall l > 0 \tag{1.14}$$

$$\delta_{\alpha\beta} \int f^{(l)} \xi_\alpha \xi_\beta\ d^D\xi = 0 \quad \forall l > 0 \tag{1.15}$$

Substitutino of Eqs.(1.10) and (1.11) into Eq. (1.7), followed by matching the corresponding powers of Kn gives the evolution equations to zero, first and second order:

$$\frac{1}{\tau}\left[f^0 - f^{eq}\right] = 0 \tag{1.16}$$

$$f^{(1)} = -\tau\left(\partial_t^{(0)} + \xi\cdot\nabla + F\cdot\nabla_\xi\right)\ f^{(0)} \tag{1.17}$$

$$f^{(2)} = -\tau\left[\left(\partial_t^{(0)} + \xi\cdot\nabla + F\cdot\nabla_\xi\right)\ f^{(1)} + \partial_t^{(1)} f^{(0)}\right] \tag{1.18}$$

1.3 Conservation equations

For isothermal fluid systems where compressibility effects con be neglected, the first and second order conservation equation for mass and momentum are obtained for (1.16 ... 1.18) after multiplication with :

$$\Phi(m,\ m\xi_\alpha,\ m\xi_\alpha\xi_\beta\delta_{\alpha\beta}/2) \tag{1.19}$$

and integration over the velocity space. The equation for mass conservation and momentum are obtained by multiplying eq. (1.7) with $\Phi = m$ and $\Phi = mv_\alpha$ after some algebra we have

$$\partial_t\rho + \partial_\beta(\rho u_\beta) = 0 \tag{1.20}$$

$$\partial_t(\rho u_\alpha) + \partial_\beta(\rho u_\alpha u_\beta) = -\partial_\alpha p + \nu\partial_\beta\left[\rho\partial_\alpha u_\beta + \rho\partial_\beta u_\alpha\right] + \rho a_\alpha \tag{1.21}$$

In eq. (1.21), p is the ideal gas pressure $(\rho = mn)$

$$p = nk_BT = \chi c^2\rho \tag{1.22}$$

and ν is the physical value of the kinematic viscosity of the single component fluid

$$\nu = \tau\chi c^2 = \tau k_BT/m \tag{1.23}$$

1.4 Discretization of the momentum space

For a system not too far from its equilibrium state, we may suppose

$$\nabla_{\xi_\alpha}\ f(\mathbf{x},\xi,t) \simeq \nabla_{\xi_\alpha}\ f(\mathbf{x},\xi,t) = -\frac{m}{k_BT}\left[\xi - \mathbf{u}(\mathbf{x},t)\right]\ f^{eq}(\mathbf{x},\xi,t) \tag{1.24}$$

After introducing expression (1.14) in Eq. (1.7), we get the following form of the Boltzmann equation:

$$\begin{aligned} \partial_t\ f(\mathbf{x},\xi,t)+\xi\cdot\nabla f(\mathbf{x},\xi,t) \quad &= \\ \frac{1}{k_BT}\mathbf{F}\cdot[\xi-\mathbf{u}(\mathbf{x},t)]\ f^{eq}(\mathbf{x},t)-\frac{1}{\tau}\left[f(\mathbf{x},\xi,t)-f^{eq}(\mathbf{x},\xi,t)\right] & \end{aligned} \tag{1.25}$$

The discretization of velocity space is the central idea of lattice Boltzmann models, where the velocity space is restricted to a finite set $(\mathbf{e}_i),\ i=0,1,...,N$ and the distribution functions f is replaced by a set of distribution functions $f_i(\mathrm{x},t)$ and $f(\mathbf{x},\xi,t)d^D\xi$ for $\xi=\mathbf{e}_i$. After discretization of both ξ and $\mathbf{x}$, the BGK Boltzmann equation (1.25) is replaced by system of N equations

$$\partial_t\ f_i(\mathbf{x},t)+\mathbf{e}_i\cdot\bigtriangledown f_i(\mathbf{x},t)=\frac{1}{k_BT}\mathbf{F}\cdot[\mathbf{e}_i-\mathbf{u}(\mathbf{x},t)]\ f_i^{eq}(\mathbf{x},t)-\frac{1}{\tau}\left[f_i(\mathbf{x},t)-f_i^{eq}(\mathbf{x},t)\right] \tag{1.26}$$

$$\text{with} \quad i=0,1,...,N$$

To solve this system numerically, the coordinate space is discretized, too, and the distribution function $f_i(\mathrm{x},t)$ are now defined only in the nodes $(\mathbf{x})$ of a lattice $\mathscr{L}$ that is either a square lattice (in 2D) or a cubic one (in 3D), with the lattice spacing δs. The main result of the discretization of the phase space is that integrals in this space are replaced by sums over the discrete velocity set $(\mathbf{e}_i)$ and expression (1.2) of the local number density becomes [5]

$$\rho\equiv\rho(\mathbf{x},t)=\sum_{i=0}^{N}f_i(\mathbf{x},t) \tag{1.27}$$

while the local velocity (1.3) is replaced by

$$\mathbf{u}\equiv\mathbf{u}(\mathbf{x},t)=\frac{1}{n(\mathbf{x},t)}\sum_{i=0}^{N}\mathbf{e}_i f_i(\mathbf{x},t) \tag{1.28}$$

1.5 Gauss-Hermite lattice Boltzmann models

According to Grad [1949b, 1952], the distribution function f can be projected on an orthogonal basis formed by Hermite polynomials. This method leads to two additional sets of equations for pressure tensor and for the energy flux and the whole system is known under the name of Grand's 13 - moments equations. We will not enter the details of the derivation of Grad, but we will use his idea of projecting the distribution function $f(\mathbf{x},\xi,t)$ on Hermite polynomials. Using the Hermite basis and the associated projection method, a set of kinetic equations can be obtained [6]. The projection of the velocity distribution function on the Hermite polynomial basis is given by [1]

$$f(\mathbf{x},\xi,t) = w(\xi)\sum_{n=0}^{\infty}\frac{1}{n!}\mathbf{a}_{\alpha}^{(n)}(\boldsymbol{x},t)\mathscr{H}_{\alpha}^{(n)}(\xi) \tag{1.29}$$

where $w(\boldsymbol{\xi})$ is the Hermite weight function and $\mathbf{H}^{(n)}$ and $\boldsymbol{a}^{(n)}$ denote the Hermite polynomial of degree n and the expansion coefficient of degree n, respectively (see the Appendix for the definition of $\mathbf{H}^{(n)}$). According to Grad's notation [1, 4], the subscript α is actually a multi-index $(\alpha_1,\alpha_2,...,\alpha_n)$ and the Einstein sum rule is used over repeated indices.
The expansion coefficients are obtained by taking the scalar product of f with the Hermite basis polynomials

$$\mathbf{a}_{\alpha}^{(n)}(\mathbf{x},t) = \int f(\mathbf{x},\xi,t)\mathscr{H}_{\alpha}^{(n)}(\xi)\,d\xi \tag{1.30}$$

Using the definitions of the moments of f,

$$\rho = \int f\,d(\xi), \qquad \rho\mathbf{u} = \int \xi\; f\,d\xi \tag{1.31}$$

one gets for [1] $a^{(n)}$ $(n = 0, ..., 4)$

$$
\begin{aligned}
a^{(0)} &= \rho && (1.32)\\
a^{(1)}_{\alpha} &= \rho\, u_{\alpha} && (1.33)\\
a^{(2)}_{\alpha\beta} &= \mathbf{P}_{\alpha\beta} + \rho(u_{\alpha}u_{\beta} - \delta_{\alpha\beta}) && (1.34)\\
a^{(3)}_{\alpha\beta\gamma} &= \mathbf{Q}_{\alpha\beta\gamma} + u_{\alpha}a^{(2)}_{\beta\gamma} + u_{\beta}a^{(2)}_{\alpha\gamma} + u_{\gamma}a^{(2)}_{\alpha\beta} + (1-D)\,\rho u_{\alpha}u_{\beta}u_{\gamma} && (1.35)\\
a^{(4)}_{\alpha\beta\gamma\delta} &= \mathbf{R}_{\alpha\beta\gamma\delta} - (\mathbf{P}_{\alpha\beta}\delta_{\gamma\delta} + \mathbf{P}_{\alpha\gamma}\delta_{\beta\delta} + \mathbf{P}_{\alpha\delta}\delta_{\beta\gamma} + \mathbf{P}_{\beta\gamma}\delta_{\alpha\delta} + \mathbf{P}_{\beta\delta}\delta_{\gamma\alpha} + \mathbf{P}_{\gamma\delta}\delta_{\alpha\beta})\\
&+ (\delta_{\alpha\beta}\delta_{\gamma\delta} + \delta_{\alpha\gamma}\delta_{\beta\delta} + \delta_{\alpha\delta}\delta_{\beta\gamma}) && (1.36)
\end{aligned}
$$

where the quantities **Q** and **R** are defined by

$$
\begin{aligned}
\mathbf{P} &= \int (\mathbf{c}\bigotimes\mathbf{c}) f\, d\mathbf{c},\\
\mathbf{Q} &= \int (\mathbf{c}\bigotimes\mathbf{c}\bigotimes\mathbf{c}) f\, d\mathbf{c},\\
\mathbf{R} &= \int (\mathbf{c}\bigotimes\mathbf{c}\bigotimes\mathbf{c}\bigotimes\mathbf{c}) f\, d\mathbf{c} && (1.37)
\end{aligned}
$$

where $\mathrm{c}\bigotimes\mathrm{c}$, $\mathrm{c}\bigotimes\mathrm{c}\bigotimes\mathrm{c}$ and $\mathrm{c}\bigotimes\mathrm{c}\bigotimes\mathrm{c}\bigotimes\mathrm{c}$ are the tensor products of c with themselves.
The thermohydrodynamic variables can be expressed in terms of the low-order Hermite expansion coefficients [6]:

$$\rho = \mathbf{a}^{(0)} \quad (1.38)$$

$$\rho\mathbf{u} = \mathbf{a}^{(1)} \quad (1.39)$$

$$\mathbf{P} = \mathbf{a}^{(2)} - \rho(\mathbf{u}^2 - \delta) \quad (1.40)$$

$$\mathbf{Q} = \mathbf{a}^{(3)} - \mathbf{u}\mathbf{a}^{(2)} + (D-1)\rho\mathbf{u}^{(3)} \quad (1.41)$$

while the internal energy is

$$\rho\varepsilon = \frac{1}{2}[a^{(2)}_{ii} - \rho(u^2 - D)] \quad (1.42)$$

Since the Hermite polynomials are orthogonal, the leading moments of the distribution function f up to order N are preserved by truncating the high-order terms in its Hermite series.

$$f(\boldsymbol{x},\xi,t) \cong f^N(\mathbf{x},\xi,t) = \omega(\xi)\sum_{n=0}^{N}\frac{1}{n!}\mathscr{H}^{(n)}_{\alpha}(\xi)\mathbf{a}^{(n)}_{\alpha}(\mathbf{x},t) \quad (1.43)$$

The approximation f^N will produce the same velocity moments as the original f. This guarantees that the fluid system can be described by a finite set of macroscopic variables (thermohydrodynamic moments).

By applying Gauss - Hermite quadrature (see the Appendix), the expansion coefficient $\mathbf{a}^{(n)}$ in (1.43) can be expressed as a weighted sum :

$$\mathbf{a}_{\alpha}^{(n)} = \int w(\xi) r(\mathbf{x},\xi,t)\, d\xi = \sum_{i=1}^{d} w_i r(\mathbf{x},\xi_i,t) = \sum_{i=1}^{d} \frac{w_i}{\omega(\xi_i)} f^N(\mathbf{x},\xi,t) \mathscr{H}_{\alpha}^{(n)}(\xi_{\mathbf{i}}) \tag{1.44}$$

where w_i and ξ_i are the weights and abscissae of the Gauss - Hermite quadrature of a degree $\geq 2N$. In case of the Gauss - Hermite quadrature the nodes are not equally spaced as in the standard lattice Boltzmann method. This rescaling procedure will give different Hermite polynomials and expansion coefficients.

The discretization of Eq. (1.7) and projection of $f^{(0)}$ on the truncated Hermite basis, gives the expansion coefficients

$$\begin{aligned}
a_0^{(0)} &= \rho & (1.45)\\
a_{0\ \alpha}^{(1)} &= \rho u_\alpha & (1.46)\\
a_{0\ \alpha\beta}^{(2)} &= \rho u_\alpha u_\beta + \rho(\theta-1)\delta_{\alpha\beta} & (1.47)\\
a_{0\ \alpha\beta\gamma}^{(3)} &= \rho u_\alpha u_\beta u_\gamma + \rho(\theta-1)(\delta_{\alpha\beta}u_\gamma + \delta_{\alpha\gamma}u_\beta + \delta_{\beta\gamma}u_\alpha) & (1.48)\\
a_{0\ \alpha\beta\gamma\delta} &= \rho u_\alpha u_\beta u_\gamma u_\delta + \rho(\theta-1)^2(\delta_{\alpha\beta}\delta_{\gamma\delta} + \delta_{\alpha\gamma}\delta_{\beta\delta} + \delta_{\alpha\delta}\delta_{\beta\gamma}) & \\
&+ \rho(\theta-1)(\delta_{\alpha\beta}u_\gamma u_\delta + \delta_{\alpha\gamma}u_\beta u_\delta + \delta_{\alpha\delta}u_\beta u_\gamma & \\
&+ \delta_{\beta\gamma}u_\alpha u_\delta + \delta_{\beta\delta}u_\alpha u_\gamma + \delta_{\gamma\delta}u_\alpha u_\beta) & (1.49)
\end{aligned}$$

With f_i defined as

$$f_i(\mathbf{x},t) \equiv \frac{w_i}{\omega(\xi_i)} f(\mathbf{x},\xi_i,t) \tag{1.50}$$

the discretized equilibrium distribution function, truncated to fourth order is

$$\begin{aligned}
f_i^{(0)} &= w_i \sum_{n=0}^{4} \frac{1}{c_l^{2n} n!}\, a_{0\,\alpha}^{(n)}\, \mathscr{H}_{i\alpha}^{(n)} \\
&= w_i \rho \{1 + \frac{\xi_i \cdot \mathbf{u}}{c_l^2} + \frac{1}{2c_l^4}\left[(\xi_i \cdot \mathbf{u})^2 - c_l^2 \mathbf{u}^2 + c_l^2(\theta-1)(\xi_i^2 - c_l^2 D)\right] . \\
&+ \frac{\xi_i \cdot \mathbf{u}}{6c_l^6}\left[(\xi_i \cdot \mathbf{u})^2 - 3c_l^2 \mathbf{u}^2 + 3c_l^2(\theta-1)(\xi_i^2 - c_l^2(D+2))\right] \\
&+ \frac{1}{24c_l^8}\left[(\xi_i \cdot \mathbf{u})^4 - 6c_l^2 \mathbf{u}^2 (\xi \cdot \mathbf{u})^2 + 3c_l^4 \mathbf{u}^4\right. \\
&+ 6c_l^2(\theta-1)\left((\xi_i \cdot \mathbf{u})^2(\xi_i^2 - c_l^2(D+4)) + c_l^2 \mathbf{u}^2(c_l^2(D+2) - \xi_i^2)\right) \\
&+ \left. .3c_l^4(\theta-1)^2\left(\xi_i^4 - 2c_l^2(D+2)\xi_i^2 + c_l^4 D(D+2)\right)\right\} \qquad (1.51)
\end{aligned}$$

where $\mathscr{H}_{i\alpha}^{(n)}$ are the discretized Hermite polynomials.

Observation.

All the terms proportional to $\theta - 1$ vanish for an *isothermal* system in which $\theta = 1$.

1.6 Implementation of the non-ideal equation of state

By examining the Boltzmann-BGK equation (1.25), it's easy to see that the third term on the left-hand side represents the force of an non-ideal fluid. This force term should be projectes on Hermite basis, too. Because this therm involves derivatives with respect to ξ, which cannot be expressed directly, one can be use the espresion (1.29) to get:

$$\begin{aligned} \nabla_\xi f &= \sum_{n=0}^{\infty} \frac{1}{n!}\, \mathbf{a}_\alpha^{(n)} \nabla_\xi (\omega \mathscr{H}_\alpha^{(n)}) = \sum_{n=0}^{\infty} \frac{(-1)^n}{n!}\, \mathbf{a}_\alpha^{(n)} \nabla_\xi^{n+1} \omega \\ &= -\omega \sum_{n=0}^{\infty} \frac{1}{n!} \mathbf{a}_\alpha^{(n)} \mathscr{H}_\alpha^{n+1} = -\omega \sum_{n=1}^{\infty} \frac{1}{n!} n \mathbf{a}_\alpha^{(n-1)} \mathscr{H}_\alpha^{(n)} \end{aligned} \tag{1.52}$$

A simplified version of Eq. (1.52) can be obtained by approximating $\nabla_\xi f \simeq \nabla_\xi f^{eq}$:

$$\nabla_\xi f \simeq \nabla_\xi f^{eq} = \frac{1}{n} \partial_\beta \left(p^i - p^w \right) + k \partial_\beta (\nabla^2 n) \tag{1.53}$$

Both expressions of $\nabla_\xi f$ will be considered during our simulations. These cases are specified in our code by the parameter key_tau, which takes the values 0 and 8, respectively.

From Eq. (1.53) we identify that:

$$p^i = \theta n \tag{1.54}$$

and

$$p^w = \frac{3\theta n}{3-n} - \frac{9}{8} n^2 \tag{1.55}$$

The equation (1.55) represent the equation of state, where the critical point is located at $\theta = 1$ and $n = 1$ and parameter k controls the surface tension.

2 | Numerical algorithms for lattice Boltzmann models

2.1 First order upwind

In two dimension, the all known D2Q9 model [] is a special case of the Hermite expansion. This model has been widely used for simulations of two-dimensional flows. When using a scheme based on characteristics , the forward Euler difference is used to compute the time derivative, but there are several possibilities to compute the term $\xi_i \cdot \nabla_i(\mathrm{x},t)$. We will restrict here to the first order upwind scheme.

First order upwind scheme applied to $\xi_i \cdot \nabla_i(\mathrm{x},t)$:

$$\xi_i \cdot \nabla_i(\mathrm{x},t) = \frac{c}{\delta s}\left[f_i(\mathrm{x},t) - f_i(\mathrm{x}-\delta s\xi_i/c,t)\right] \tag{2.1}$$

This numerical scheme is defined on the square lattice where δs is lattice spacing and is very easy to extend for other lattices. Special attention should be paid when applying the updating rule for values of f at each lattice node by denoting $f(\mathrm{x}-\delta s\xi_i/c,t)$ in accordance to :

$$\begin{aligned} f_i(\mathrm{x},t+\delta t) &= f_i(\mathrm{x},t) - \frac{c\delta t}{\delta s}\left[f_i(\mathrm{x},t) - f_i(\mathrm{x}-\delta s\xi_i/c,t)\right] \\ &+ \frac{\delta t}{\chi c^2}\mathrm{a}(\mathrm{x},t)\cdot\left[\xi_i - \mathrm{u}(\mathrm{x},t)\right] f_i^{eq}(\mathrm{x},t) \end{aligned} \tag{2.2}$$

The first-order upwind scheme is a good candidate for LB models because of its stability and when is associated to the forward time stepping rule, this scheme gives the updating rule for the distribution functions defined in Eq. (2.2).

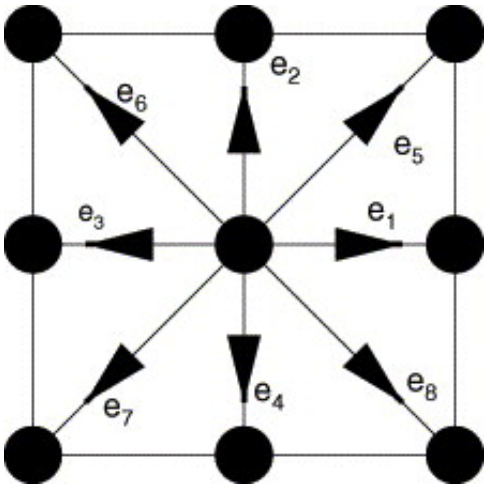


Figure 2.1: D2Q9 lattice Boltzman model, where $e_i = \xi_i$.

Upwind method is use because is more stable than the space centered scheme when dealing with large density gradients.

2.2 Flux limiter schemes

The important aspects of choosing numerical scheme in lattice Boltzmann model is to improve the numerical accuracy. We know that the upwind scheme is preferred since this one is more stable than the space centered scheme when dealing with large density gradients. However, the upwind scheme exhibits numerical diffusion and viscosity [8] which may affect the simulation results. When applying high order scheme like Lax - Wendorff [5][9] :

$$f_{i,j}^{n+1} = f_{i,j}^{n} - \frac{c\delta t}{\delta s}[f_{i,j+1}^{n} - f_{i,j-1}^{n}] + \frac{1}{2}\left(\frac{c\delta t}{\delta s}\right)^2 [f_{i,j+1}^{n} - 2\, f_{i,j}^{n} + f_{i,j-1}^{n}] \tag{2.3}$$

where δs and δt ar space lattice and time step, even if is more accurate then centered scheme, the wiggle phenomenon introduces unphysical oscillations of the fluid density.
Flux limiter techniques are used to improve the numerical accuracy and is a good alternative to the Lax-Wendroff scheme, because provide a possibility to overcome these well known problems of FDLB models. Note that, for all $i = 1, 2, ..., N$, we have $c = |\boldsymbol{e}_i|$ in the equation below:

$$CFL = c\frac{\delta t}{\delta s} \tag{2.4}$$

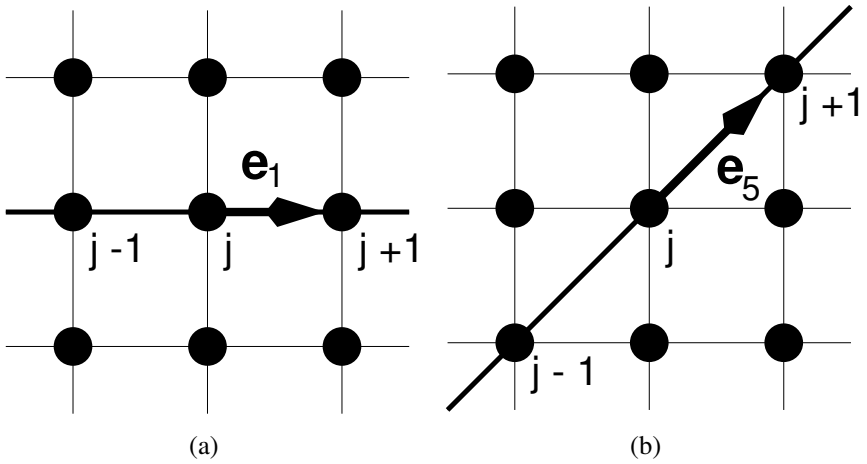


Figure 2.2: Lines of characteristics in the LB lattice, for the following distribution functions: a - $f_1(x,t)$; b - $f_5(x,t)$.

where CFL is the Courant - Friedrichs - Levy number. We rewrite the updating rule (2.6) in a conservative form using two fluxes [5][9]

$$f_{i,j}^{n+1} = f_{i,j}^{n} - CFL[F_{i,j+1/2}^{n} - F_{i,j-1/2}^{n}] \tag{2.5}$$

where

$$F_{i,j+1/2}^{n} = f_{i,j}^{n} + \frac{1}{2}(1-CFL)[f_{i,j+1}^{n} - f_{i,j}^{n}]\psi(\theta_{i,j}^{n}) \tag{2.6}$$

and

$$F_{i,j-1/2}^{n} = F_{i,(j-1)+1/2}^{n} = f_{i,j-1}^{n} + \frac{1}{2}(1-CFL)[f_{i,j}^{n} - f_{i,j-1}^{n}]\psi(\theta_{i,j-1}^{n}) \tag{2.7}$$

The flux limiter $\psi(\theta_{i,j}^{n})$ introduced in (2.9) is expressed as a function of the *smoothness*

$$\theta_{i,j}^{n} = \frac{f_{i,j}^{n} - f_{i,j-1}^{n}}{f_{i,j+1}^{n} - f_{i,j}^{n}} \tag{2.8}$$

In particular, the Lax - Wendroff scheme is recovered for the flux limiter $\psi(\theta_{i,j}^{n}) = 1$. The widely used first order upwind scheme is recovered as another particular case, when $\psi(\theta_{i,j}^{n}) = 0$.

2.3 Corner transport upwind

The flux limiter scheme described above can be viewed in a more geometric way that facilitates the extension to two space dimensions. Corner transport upwind (CTU) method was develop by P. Colella [10] for studying effects of diagonal flow.

We first consider the scalar advection equation

$$q_t + uq_x + vq_y = 0 \tag{2.9}$$

with u and v constant. We will generally assume that $u > 0$ and $v > 0$, and this case will sometimes be assumed when we wish to be specific, but most of the formulas will be presented in a manner that applies for flow in any direction. A quite general class of methods can be derived by the following sequence [10] [11] [12] of steps:

1. View the cell averages at time t_n as defining a piecewise constant function $\tilde{q}^n(x,y,t_n)$.
2. Solve the advection equation exactly with this data over a time step of length $\triangle t$, giving $\tilde{q}^n(\boldsymbol{x},\boldsymbol{y},t_{n+1}) = \tilde{q}^n(\boldsymbol{x}-\boldsymbol{u}\triangle t, \boldsymbol{y}-\boldsymbol{v}\triangle t, t_n)$.
3. Average this shifted function over the grid cells to obtain the new cell average and we assume a uniform grid with equal spacing h in both directions [11].

$$Q_{i,j}^{n+1} = \frac{1}{h^2}\int_{x_{i-1/2}}^{x_{i+1/2}}\int_{y_{j-1/2}}^{y_{j+1/2}} \tilde{q}^n(\boldsymbol{x}-\boldsymbol{u}\triangle t, \boldsymbol{y}-\boldsymbol{v}\triangle t, t_n)dxdy \tag{2.10}$$

$$\begin{aligned} Q_{i,j}^{n+1} &= \frac{1}{h^2}\int_{x_{i-1/2}}^{x_{i+1/2}}\int_{y_{j-1/2}}^{y_{j+1/2}} \tilde{q}^n(\boldsymbol{x}-\boldsymbol{u}\triangle t, \boldsymbol{y}-\boldsymbol{v}\triangle t, t_n)dxdy \\ &= \frac{1}{h^2}\int_{x_{i-1/2}-\boldsymbol{u}\triangle t}^{x_{i+1/2}-\boldsymbol{u}\triangle t}\int_{y_{j-1/2}-\boldsymbol{v}\triangle t}^{y_{j+1/2}-\boldsymbol{v}\triangle t} \tilde{q}^n(\boldsymbol{x},\boldsymbol{y},t_n)dxdy \end{aligned} \tag{2.11}$$

The new cell average $Q_{i,j}^{n+1}$ is given by the cell average of $\tilde{q}^n(\boldsymbol{x},\boldsymbol{y},t_n)$ over the shaded rea region shown in Fig. 2.3. Because $\tilde{q}^n(\boldsymbol{x},\boldsymbol{v},t_n)$ is constant in each grid cell, this reduces the problem to a simple convex combination of four cell values:

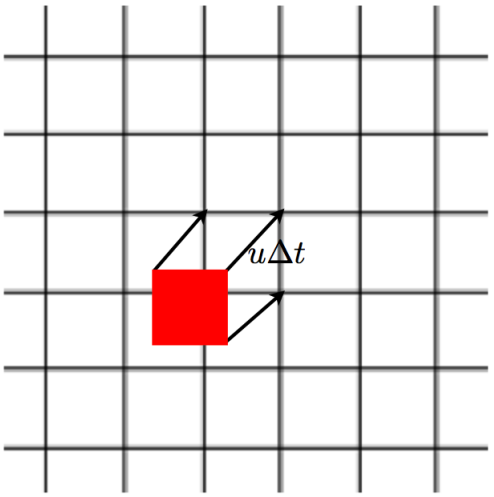


Figure 2.3: The corner transport upwind method is obtained by shifting the piecewise constant data by distance $(\boldsymbol{u}\triangle t, \boldsymbol{v}\triangle t)$ and averaging back on the grid.

$$\begin{aligned} Q_{i,j}^{n+1} &= \frac{1}{h^2}[(\triangle x - u\triangle t)(\triangle y - v\triangle t)Q_{i,j}^{n} + (\triangle x - u\triangle t)(v\triangle t)Q_{i,j-1}^{n} \\ &+ (\triangle y - v\triangle t)(u\triangle t)Q_{i-1,j}^{n} + (u\triangle t)(v\triangle t)Q_{i-1,j-1}^{n}] \end{aligned} \tag{2.12}$$

Equation (2.15) can be rearranged to yield

$$\begin{aligned} Q_{i,j}^{n+1} &= Q_{i,j} - \frac{\mathrm{u}\triangle t}{h}(Q_{i,j} - Q_{i-1,j}) - \frac{\mathrm{v}\triangle t}{h}(Q_{i,j} - Q_{i,j-1}) \\ &+ \frac{1}{2}(\triangle t)^2\left\{\frac{u}{h}\left[\frac{v}{h}(Q_{i,j} - Q_{i,j-1}) - \frac{v}{h}(Q_{i-1,j} - Q_{i-1,j-1})\right]\right. \\ &+ \left.\frac{v}{h}\left[\frac{u}{h}(Q_{i,j} - Q_{i-1,j}) - \frac{u}{h}(Q_{i,j-1} - Q_{i-1,j-1})\right]\right\} \end{aligned} \tag{2.13}$$

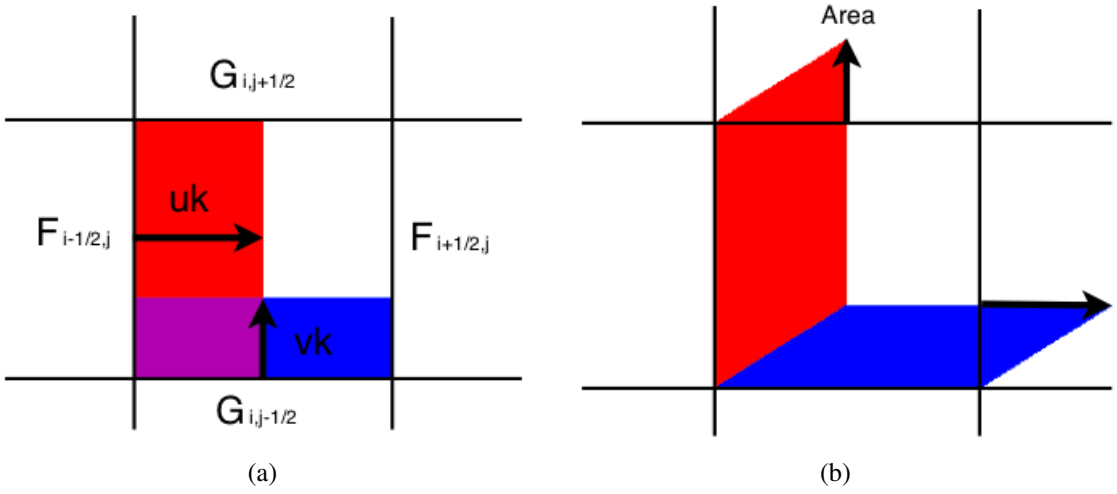
This method (2.16) is called *corner-transport upwind* (CTU) method (following Colella [10]), because define a proper transport across the corner from cell $C_{i-1,j-1}$ to $C_{i,j}$ and is still only first-order accurate. For a conservative finite volume method in flux-differencing form, we will have [12]

$$Q_{i,j}^{n+1} = Q_{i,j} - \frac{\triangle t}{h}[F_{i+1/2,j} - F_{i-1/2,j} + G_{i,j+1/2} - G_{i,j-1/2}] \tag{2.14}$$

where $\triangle t$ is the time step and $F_{i-1/2,j}$ represents theflux at the edge of the cell C_{ij} and $G_{i,j-1/2}$ is the flux at its bottom.

The simplest upwind method (the "donor-cell" method presented in the first line of Eq. 2.16) can be interpreted as wave propagation method (shown in Fig. 2.4. a) and we have

$$F_{i-1/2,j} = u\, Q_{i-1,j}$$
$$G_{i,j-1/2} = v\, Q_{i,j-1} \tag{2.15}$$

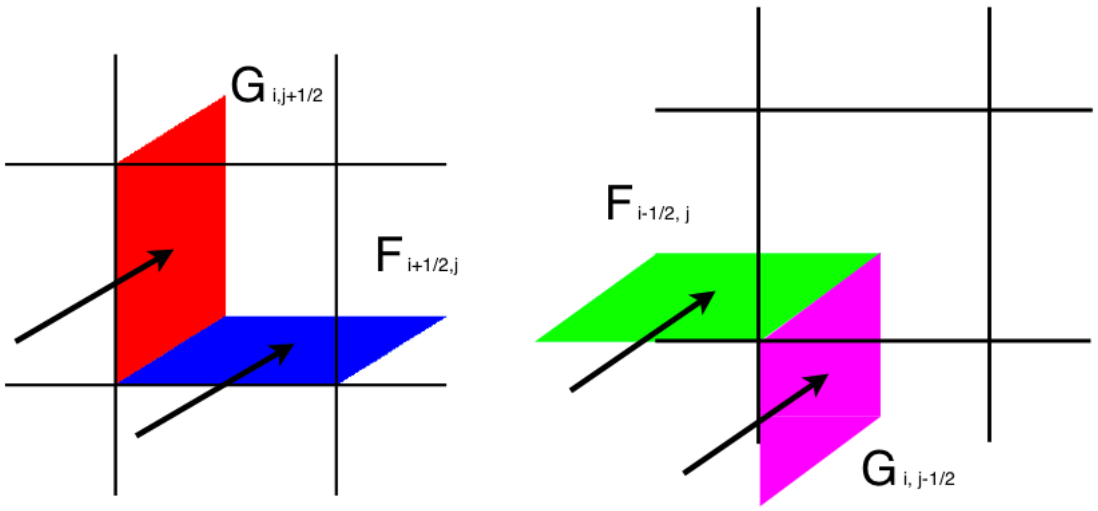


(a) (b)

Figure 2.4: Wave propagation interpretation with fluxes: a - first order upwind; b - first order corner transport upwind.

Waves propagate independently into the cell for the x and y directions, carrying jumps $(Q_{i,j} - Q_{i-1,j})$ and $(Q_{i,j} - Q_{i,j-1})$, at speeds given by the velocities u and v. Propagating each of these waves at the proper speed (u, v) oblique to the grid (Fig. 2.4 b) it can be easily implemented as two-step procedure. First the wave propagated normal to interface, giving a provisional value for the flux, and then the triangular pice of the wave that moves into an adjacent cell is used to update the flux between cell. If the area of this triangle is $Area = \frac{1}{2}(\triangle t)^2 \mathrm{uv}$ then the cell average is modified by $\frac{1}{2}\frac{\triangle t^2}{h^2}\mathrm{uv} \triangle Q$, where $\triangle Q$ is the jump cross the wave.

Because one cell average is increased by this amount while the other is decreased by this same amount, this transfer can be obtain by modifying the flux $G_{i,j+1/2}$ by $\frac{1}{2}\frac{\triangle t}{h}\mathrm{uv} \triangle Q$. Now the wave propagation from each interface affects two different fluxes and this is most easily implemented by initializing all $F_{i-1/2,j}$ and $G_{i,j-1/2}$ to zero and then looping over the cell interface , updating the appropriate fluxes.



(a) (b)

Figure 2.5: a. Transverse propagation affecting the fluxes $\tilde{F}_{i+1/2,j}$ and $\tilde{G}_{i,j+1/2}$ b. Transverse propagation affecting the fluxes $\tilde{F}_{i-1/2,j}$ and $\tilde{G}_{i,j-1/2}$.

The wave propagation from interface between cells $C_{i-1,j}$ and $C_{i,j}$ affects the fluxes $F_{i-1/2,j}$ and $G_{i,j+1/2}$ and similarly the wave from the interface between $C_{i,j-1}$ and $C_{i,j}$ update the $G_{i,j-1/2}$ and $F_{i+1/2,j}$ by [11]

$$\begin{aligned}
\tilde{F}_{i-1/2,j} &= -\frac{1}{2}\frac{\triangle t}{h}\,\mathrm{uv}(Q_{i-1,j}-Q_{i-1,j-1}) \\
\tilde{F}_{i+1/2,j} &= -\frac{1}{2}\frac{\triangle t}{h}\,\mathrm{uv}(Q_{i,j}-Q_{i,j-1}) \\
\tilde{G}_{i,j-1/2} &= -\frac{1}{2}\frac{\triangle t}{h}\,\mathrm{uv}(Q_{i,j-1}-Q_{i-1,j-1}) \\
\tilde{G}_{i,j+1/2} &= -\frac{1}{2}\frac{\triangle t}{h}\,\mathrm{uv}(Q_{i,j}-Q_{i-1,j})
\end{aligned} \tag{2.16}$$

The first order upwind method is widely used because it is more stable than the original version called "donor-cell" upwind, because modified method requires

$$\frac{\triangle t}{h} max(|\mathrm{u}|,|\mathrm{v}|) \leq 1 \tag{2.17}$$

whereas the original method requires

$$\frac{\triangle t}{h}(|\mathrm{u}|,|\mathrm{v}|) \leq 1 \tag{2.18}$$

The second order Corner Transport Upwind scheme is more elaborated and has better stability properties than the two-dimensional Lax-Wendroff scheme [7].

2.4 Parallel computing using PETSc

The Portable, Extensible Toolkit for Scientific computation (PETSc) is a suie of open source software libraries for parallel solution of linear and nonlinear equations. PETSc uses the Message Passing Interface (MPI) for all of its parallelism and consists of a variety of libraries (similar to classes in C++) where each library manipulates a particular family of objects (for example vectors) and the operations one would like top perform on the objects.
The development of PETSc was started in 1995 by Bill Gropp, Lois Curfman McInnes, and Barry Smith at Argonne National Laboratory and is used for scalable (parallel) solution of scientific applications modeled by partial differential equations (PDE) and is one of the most widely used parallel numerical software.
Linear algebraic systems arise when solving the continuum partial differential equation models using the finite element, finite volume, finite difference [13]. The main focus of PETSc is to solving linear systems arising from PDE-based models. Some of the PETSc modules deal with [14]:

- Index sets (IS), including permutations, for indexing into vectors, renumbering, etc.
- Vectors (Vec) are one of the simplest PETSc objects and are used to store discrete PDE solutions, right-hand sides for linear systems, etc.
- Matrices (Mat) where is possible to choose a variety of matrix implementations because no single matrix format is appropriate for all problems.
- Managing interactions between mesh data structures and vectors and matrices (DM).
- Over fifteen Krylov subspace methods (KSP).
- Dozens of pre conditioners, including multi grid, block solvers, and sparse direct solvers (PC).
- Nonlinear solvers (SNES).
- Time-steppers for solving time-dependent (nonlinear) PDEs, including support for differential algebraic equations (TS).

The use of PETSc is very easy when parallelism is achieved by domain decomposition. The domain where the PDE is defined is divided among the processes, and each process manages the unknowns and matrix elements associated with that domain. PETSc use MPI model for parallel programming and employ its routines as needed within an application code. The user does not need to manage the detailing with parallel objects such as vectors, matrices, and solvers.
The libraries enable easy customization and extension of both algorithms and implementations. PETSc follows the distributed-memory single program multiple data (SPMD) model of message passing interface, with the flexibility of having different types of computation running on different processes. The PETSc infrastructure create a foundation for building large-scale applications and is useful to consider the interrelationship among different pieces of this collection of libraries as shown in Fig. 2.6 and in Fig. 2.7 is shown several of the individual parts in more detail.

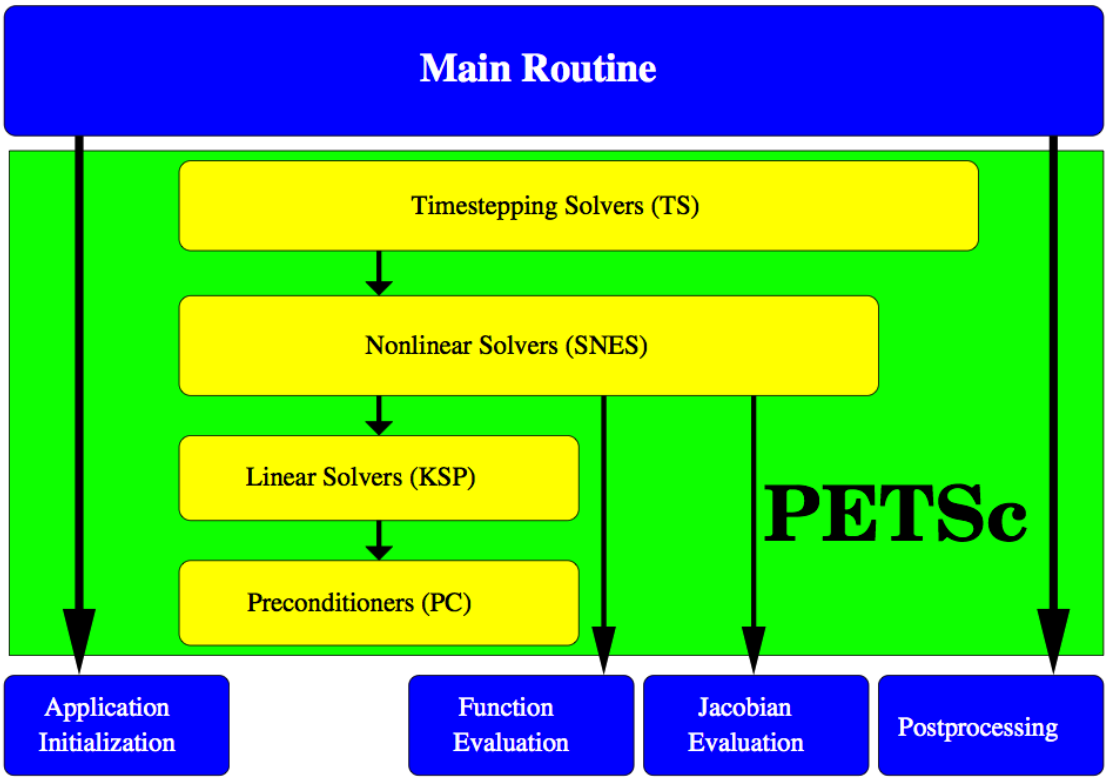


Figure 2.6: Flow control for PETSc applications [14]

These figures [14] illustrate the library's hierarchical organization, which enables users to employ the level of abstraction that is most appropriate for a particular problem.

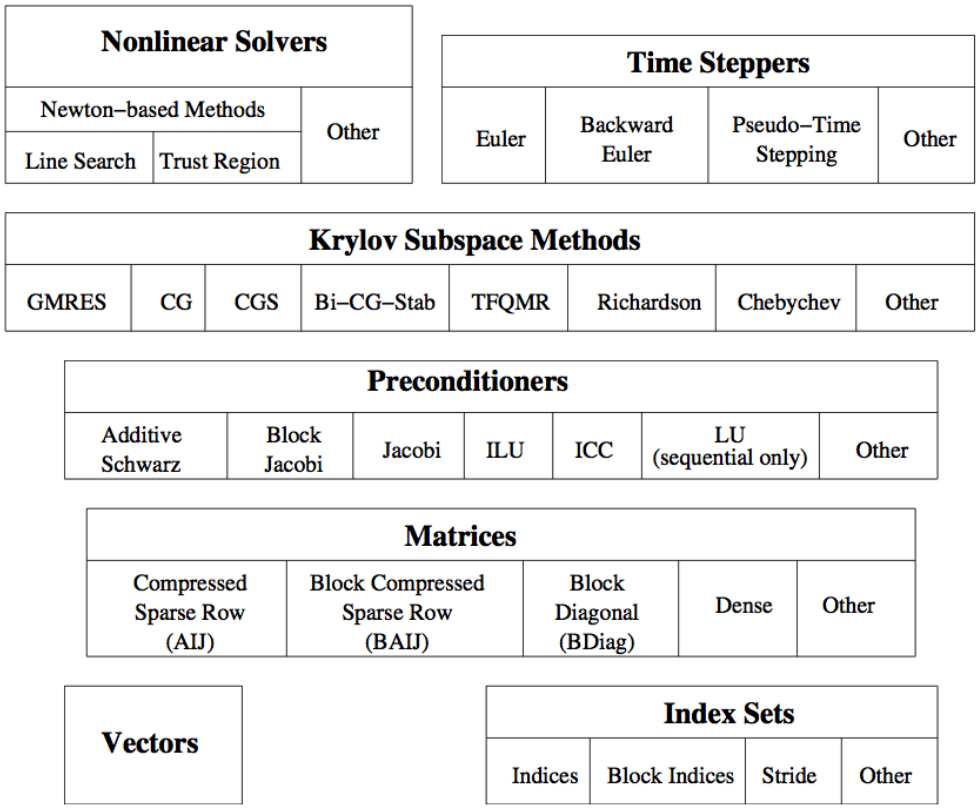


Figure 2.7: Numerical libraries of PETSc [14]

PETSc is written in C language using object-oriented programming techniques of data encapsulation, polymorphism, and inheritance. There are six main classes [**?**] like **Vec** vector class for managing the system solutions, the **Mat** matrix class for managing the sparse matrices, the **KSP** Krylov solver class for managing the iterative accelerators, the **PC** preconditioned class, the **SNES** nonlinear solver class, and the **TS** ordinary differential equations (ODE) integrator class. A wide variety of simulations have been written by using PETSc and these include fluid flow for aircraft, automobile design, blood flow simulation, porous media flow, etc.

An important role in PETSc are distributed arrays (DMDAs), which are used in conjunction with vectors. Logically regular rectangular grids are used by DMDAs when communication of nonlocal data is needed before certain local computations can occur. This DMDAs are not intended for parallelizing unstructured grid problems.

A typical situation that we can encounter, when solving PDEs equations in parallel, is to evaluate a local function $f(x)$, where each process requires its local portion of the vector x as well as its host points (the bordering portions of the vector that are owned by neighboring processes).

2.5 Parallel computing using GPUs

Parallel computing use the capability of a computer to execute operations concurrently and has been used to simplify the programming of certain applications which react to or simulate the parallelism of the natural world. At the same time, parallelism complicates programming when the objective is to take advantage of the existence of multiple hardware components to improve performance. Graphics processing units (GPUs) are devices that we find in most modern PCs, because they provide a number of basic operations to the CPU (central processing unit), such as rendering an image in memory and then displaying onto a screen. A GPU belongs to the architecture category of single-input multiple-data (SIMD) processors, which basically means that many processors do the same computations for different data in parallel and the GPU hardware consists of a number of key blocks:

- Memory (global, constant, shared)
- Streaming multiprocessors (SMs)
- Streaming processors (SPs)

In our days graphics processing units (GPUs) are very powerful and highly parallel. GPUs contain hundreds of processor cores that may run thousands of threads concurrently. For this reason, the intensive computing applications run much faster than on a CPU.
In 2007, NVIDIA saw an opportunity to bring GPUs into the mainstream by adding an easy-to-use programming interface, which is a parallel programming library named CUDA (Computing Unified Device Architecture). CUDA is an extension to the C language [15] that allows GPU code to be written in regular C (in fig. 2.8 we present a compilation process with CUDA using NVCC compiler). With this extensions is possible to use the GPU specific features that include new API calls, and some new type qualifiers that apply to functions and variables. In CUDA we find some specific functions, called *kernels* that can be invoked by the CPU. The GPU has its own internal scheduler that will then allocate the kernels to whatever GPU hardware is present. It is executed N number of times in parallel on GPU by using N number of threads, and also CUDA provides shared memory and synchronization among threads. The *threads* are organized by defining a grid and making a division of the grid in thread block or just *blocks* (Fig. 2.9).

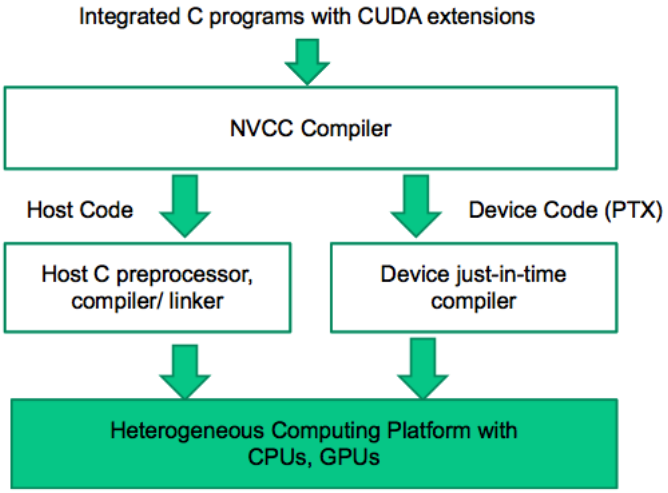


Figure 2.8: Compilation process in CUDA [15]

Because each block consists of a batch of threads, it can be a 1D, 2D or 3D object where the maximal number of threads which is allowed depends on the graphic card capability [15].

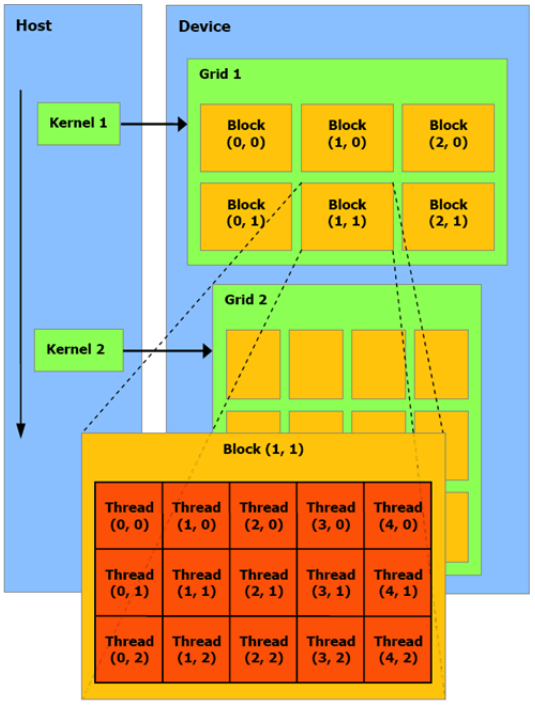


Figure 2.9: A multidimensional example of CUDA grid organization [18]

A block must execute from start to completion and may be run on one of N SMs (symmetrical multiprocessors). Blocks are allocated from the grid of blocks to any SM that has free slots. Blocks threads are grouped into *warps* that consists of typically of 32 threads with consecutive thread indices.

Since all these threads execute the same code, CUDA programming is an instance of the well-known SPMD (single program, multiple data) [15] parallel programming style which is not the same as SIMD, because in an SPMD system, the parallel processing units execute the same program on multiple parts of data. Typically when a host code launches a kernel, the CUDA runtime system generates a *grid* of threads that are organized in a two-level hierarchy. Furthermore the grid is organized into an array of thread blocks, and all blocks of a grid are the same size, where each block contain up to 1024 threads. The number of threads in each thread block is specified by the host code when a kernel is launched. The kernel can be launched with different numbers of threads at different parts of the host code and for a given grid of threads, the number of threads in a block is available in the *blockDim* variable. Is simply to view that each

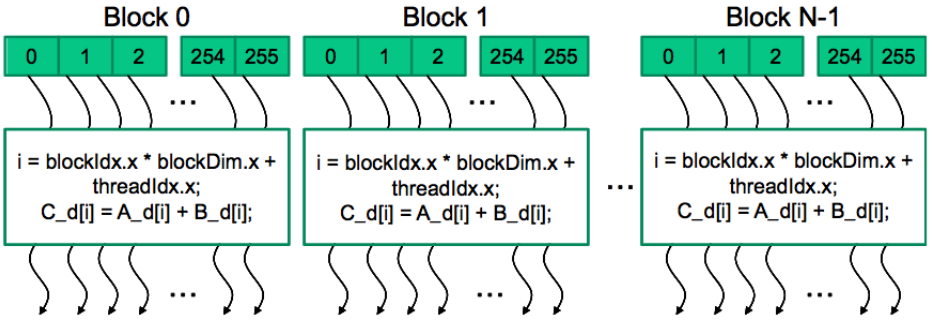


Figure 2.10: All threads in a grid execute the same code [15]

thread in block has a unique *threadIdx* value, for example, the first thread in block 0 has value 0 in its *threadIdx* variable, the second thread has value 1, etc. This allows each thread to combine its *threadIdx* and *blockIdx* values to create a unique global index for itself with the entire grid (example in Fig. 2.10 of a global index). This model of programming compels the programmer to organize threads and their data into hierarchical and multidimensional organizations.

High-performance GPU applications require fast memory transfer and for that we have to take in to account that GPU has different layers of memory. The main memory is the global memory which have the biggest capacity but the poor performance is due to the fact that tends to have long access latencies (hundreds of clock cycles) and finite access bandwidth. CUDA supports several types of memory that can be used by programmers to achieve high execution speed in their kernels (Fig. 2.11) show these CUDA device memories.

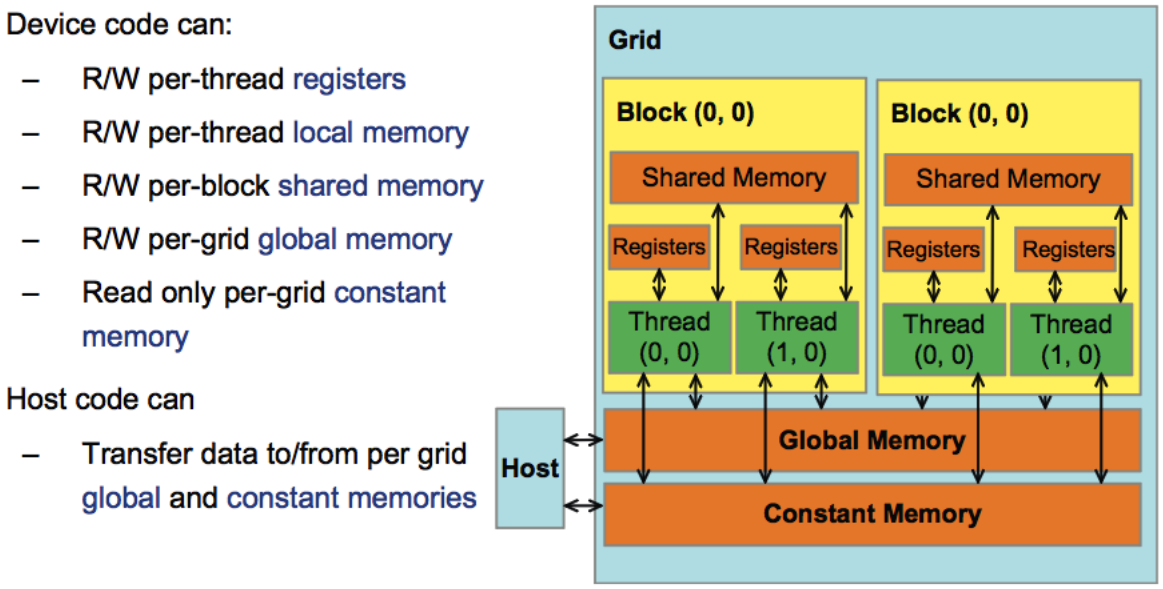


Figure 2.11: CUDA device memory model [18]

The constant memory support short-latency, high-bandwidth, read-only access by the device when all threads simultaneously access the same location. Variables that reside in registers and shared memory (on-chip memories) can be accessed at very high speed in a highly parallel manner. Registers are allocated to individual threads where each thread can only access its own registers and shared memory is allocated to thread blocks where all the threads in a block can access variables in this memory locations. By declaring a CUDA variable in one of the CUDA memory types (__device__ , __ global__ and __ host__), the programmer dictates the visibility and access speed of the variables.

3 Computing results

3.1 Minkowski functionals

Because we want to describe the morphology of the domains that are formed during phase separation in 2D and 3D, it is important to find a method for the characterization of these structures. Integral geometry can supply a method to find this morphological measures, known as Minkowski functionals. They provide the characteristic length scale L of patterns and also allow one to study the scaling behavior of the content, shape, and connectivity of spatial structures. The Minkowski functionals are well known and used very often in digital picture analysis and integral geometry [17] when dealing with the morphology of black and white discretized images. In two dimensions, Minkowski functionals are related to familiar geometric quantities: the area, the boundary length and the connectivity number. The four functionals for three dimensional structures are the volume, surface, integral mean curvature and connectivity of the spatial pattern. To understand the morphological characterization, to refer a picture element case and, for conveniece, we will use the term pixel to refer at pictures elements.

First, we define a 2D lattice used for Boltzmann simulations, filled with black pixels on a white background. Now let us to assume that the pixels are squares and that the linear size of each square has been normalized to one (Fig. 3.1). In order to get detailed information about each density level, we introduce a threshold [16] density $0 \leq \rho_{th} \leq \rho_{max}$. Each black pixel or grey level may be associated with the local density $\rho(x)$ in every corresponding node.

The Minkowski functionals, that describe the morphological content of the 2D pattern are the area A, the perimeter U and the Euler characteristic χ. For understanding the Euler characteristic in two dimensional case, we have to consider that two black pixels are connected if and only if they are nearest neighbors or next-nearest neighbors of each other or can be connected by a chain of black pixels that are nearest and/or next-nearest neighbors.

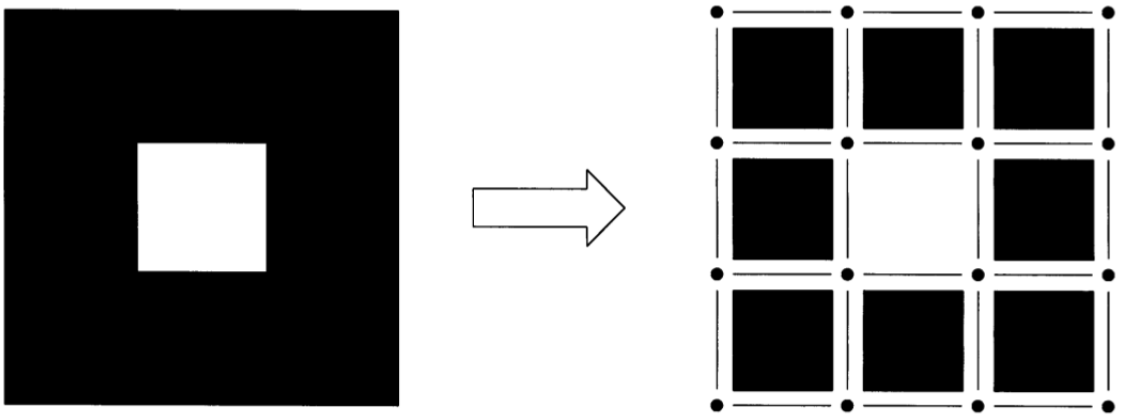


Figure 3.1: A black and white digital image consists of a square lattice [17]. For this example: number of squares $n_s = 8$, number of edges $n_e = 24$ and number of vertices $n_v = 16$.

Two steps are required to compute the values of the Minkowski functionals. First, let us to decompose each black pixel into 4 vertices, 4 edges and the interior of the pixel (Fig. 3.1). After that we count the total number of squares n_s, edges n_e and vertices n_v, and for computing the functionals [17] from

$$A = n_s \tag{3.1}$$

$$U = -4n_s + 2n_e \tag{3.2}$$

$$\chi = n_s - n_e + n_v \tag{3.3}$$

In case of a three dimensional cubic lattice filled with black and white pixels, the four Minkowski functionals are the volume V, the surface area S, the mean breadth B, and the Euler characteristic. To exemplify χ in 3D, we have to consider that number of regions of connected black pixels plus the number of completely enclosed regions of white pixels minus the number of tunnels, i.e. regions of white pixels piercing regions of connected black pixels. Now let's consider that each black pixel as the union of 6 faces, 8 vertices, 12 edges and the interior of the cube, it can be shown [17] that

$$V = n_c \tag{3.4}$$

$$S = -6n_c + 2n_f \tag{3.5}$$

$$2B = 3n_c - 2n_f + n_e \tag{3.6}$$

$$\chi = -n_c + n_f - n_e + n_v \tag{3.7}$$

where n_c and n_f are the number of cubes and faces, respectively.

The Minkowski functionals implemented with PETSc are computed for a two dimensional lattice, with a function called "minko2d" :

```
#undef __FUNC__
#define __FUNC__ "minko2d"

PetscErrorCode minko2d(void)
{
  PetscInt ix, iy;

  char filename[128];
  FILE *xhandle;

  DMDAVecGetArrayDOF(da, gf, &agf);
  for(iy=ystart; iy < yend; iy++)
    {
      for(ix=xstart; ix < xend; ix++)
        {
          if(agf[iy][ix][kn] > rth)
            {
              agf[iy][ix][kmpix] = one;
            }
          else
            {
              agf[iy][ix][kmpix] = zero;
            }
        }
    }
  DMDAVecRestoreArrayDOF(da, gf, &agf);
  DMGlobalToLocalBegin(da,gf,INSERT_VALUES,lf);
  DMGlobalToLocalEnd(da,gf,INSERT_VALUES,lf);
  DMDAVecGetArrayDOF(da,gf,&agf);
```

```
DMDAVecGetArrayDOF(da,lf,&alf);



rbpix = 0;
redge = 0;
rvertice = 0;
for(iy=ystart; iy < yend; iy++)
  {
    for(ix=xstart; ix < xend; ix++)
      {
        if(alf[iy][ix][kmpix])
          {
            rbpix++;
          }
        if(alf[iy][ix][kmpix] == one || alf[iy][ix+1][kmpix] == one)
          {
            redge++;
          }
        if(alf[iy][ix][kmpix] == one || alf[iy+1][ix][kmpix] == one)
          {
            redge++;
          }
        if(alf[iy][ix][kmpix] == one || alf[iy][ix+1][kmpix] == one ||
           alf[iy+1][ix][kmpix] == one || alf[iy+1][ix+1][kmpix] == one)
          {
            rvertice++;
          }
      }
  }
DMDAVecRestoreArrayDOF(da,gf,&agf);
DMDAVecRestoreArrayDOF(da,lf,&alf);
```

```
  VecSetValue(mfbpix,rstart,rbpix,INSERT_VALUES);
  VecSetValue(mfedge,rstart,redge,INSERT_VALUES);
  VecSetValue(mfvertice,rstart,rvertice,INSERT_VALUES);

  VecSum(mfbpix, &rbpix);
  VecSum(mfedge, &redge);
  VecSum(mfvertice, &rvertice);

  return 0;
}
```

In this function VecSetValue(Vec v,int row,PetscScalar value, InsertMode mode) has the role to set a single entry i.e. value into a vector v , where row set location of the entry. For computing the sum of all the components of a vector, we use VecSum(Vec v,PetscScalar *sum), where v is vector and sum is the result of this summation.

3.2 Simulation of phase separation in two - dimensional systems at constant temperature

3.2.1 Plane interface

The phase separation process was first investigated in the case of plane interfaces. For this purpose, we used a two-dimensional lattice with 512x4 nodes. The simulation parameters were $\delta s = 1/128$, etc, $\kappa = 10^{-4}$. Figure (3.2) shows the evolution of the density profiles at temperature T= 0.80, as recovered with $e\nabla = 2$ and $\tau = 10^{-3}$. One can see the formation of high density domains at the early stage of the separation process. These domains coalesce thereafter to reduce the interface energy.

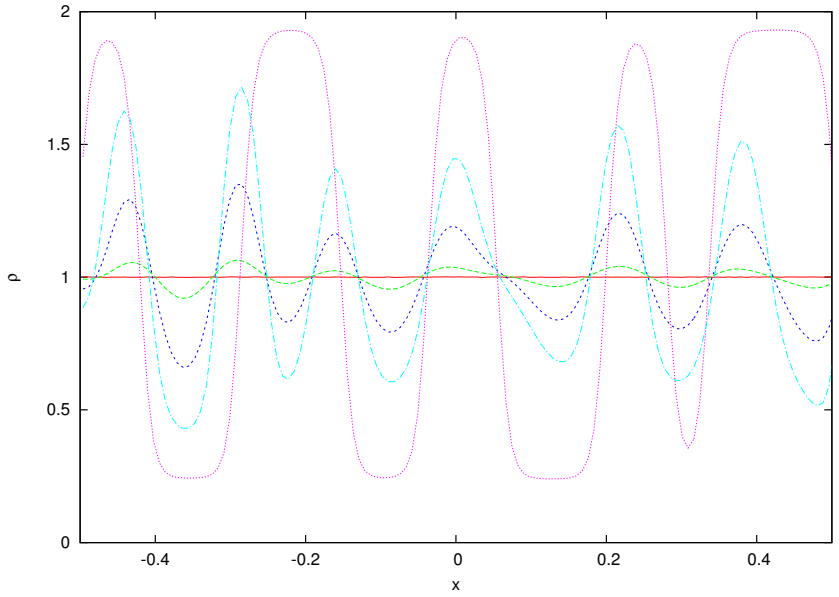
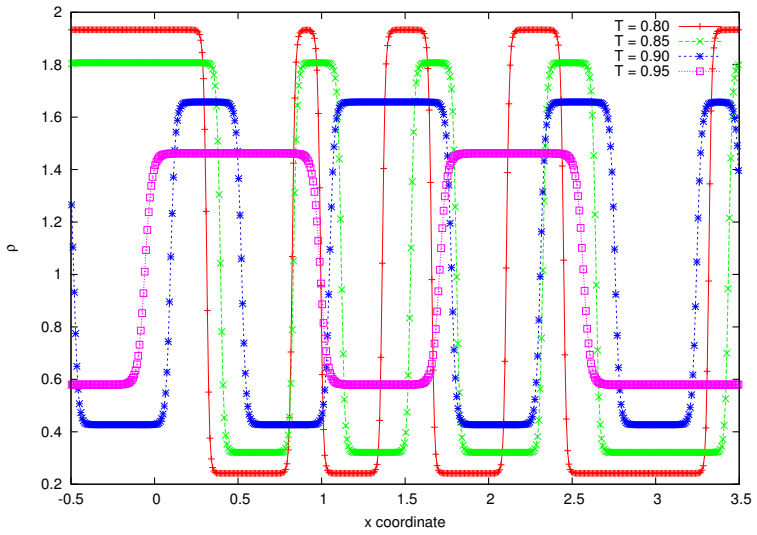


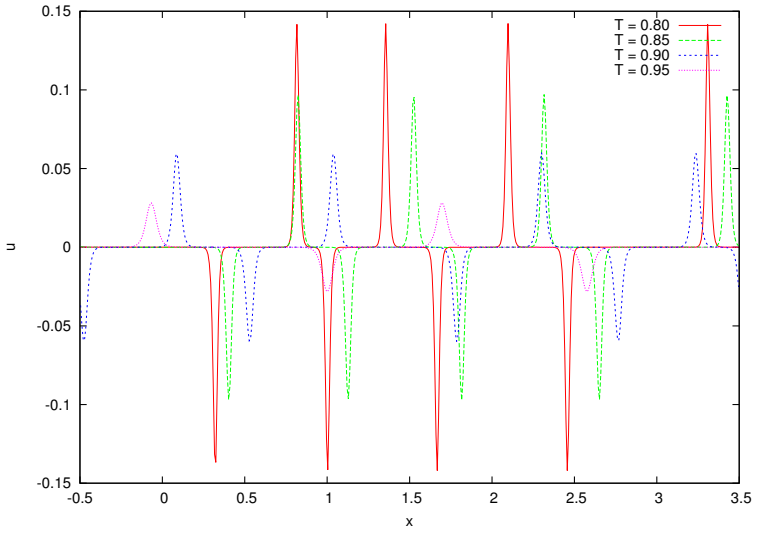
Figure 3.2: Denisty profiles for $e\nabla = 2, \tau = 0$ at different iterations.

Figures (3.3) and (3.4) show the density and velocity profiles after 10 millions iterations at various temperatures. Both versions of the corner transport upwind scheme (first and second order) were used. One can easily see that the amplitude of the spurious velocity is drastically reduced when using the second order scheme.

In figures (3.5) and (3.6) we plot the density profiles obtained by LB simulation after 10 millions iterations, for various values of the temperatures. These profiles agree very well to the corresponding values of the liquid and vapour density, as derived by the Maxwell construction.

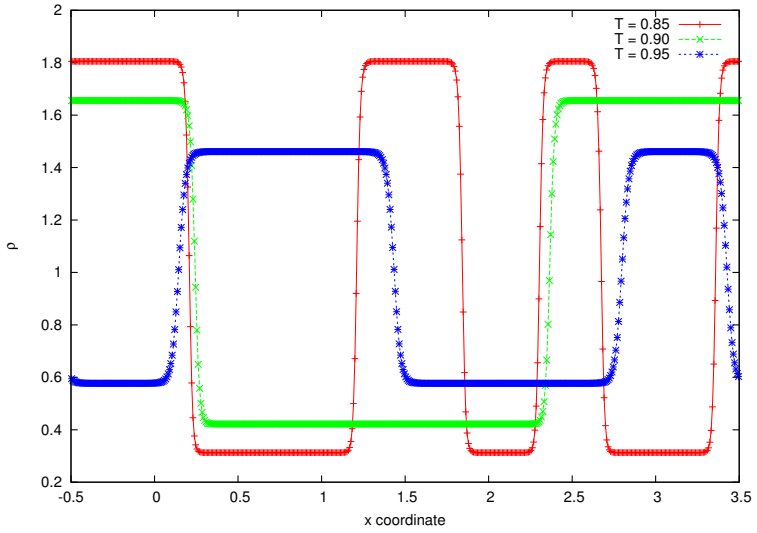


(a)

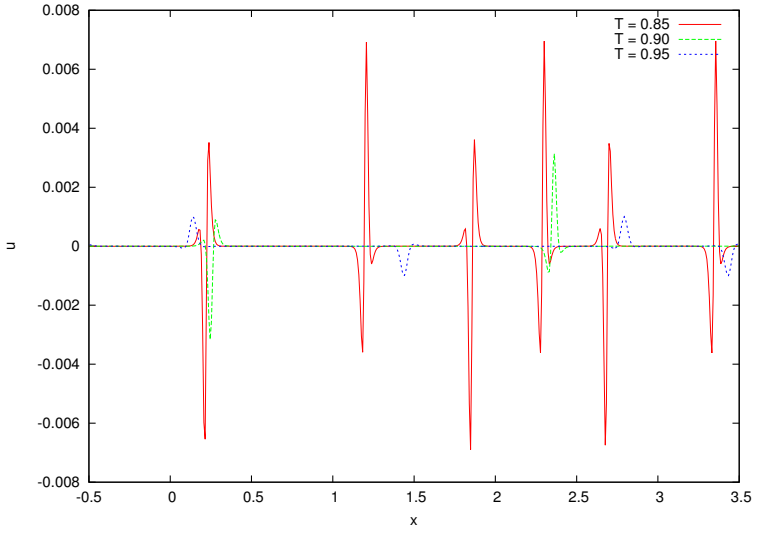


(b)

Figure 3.3: a. Equilibrium density profiles. b. Equilibrium velocity profiles. In both cases, it is $e\nabla = 2, \tau = 0$ and $T = 0.80$.



(a)



(b)

Figure 3.4: a. Equilibrium density profiles. b. Equilibrium velocity profiles. In both cases, it is $e\nabla = 4, \tau = 0$ and $T = 0.80$.

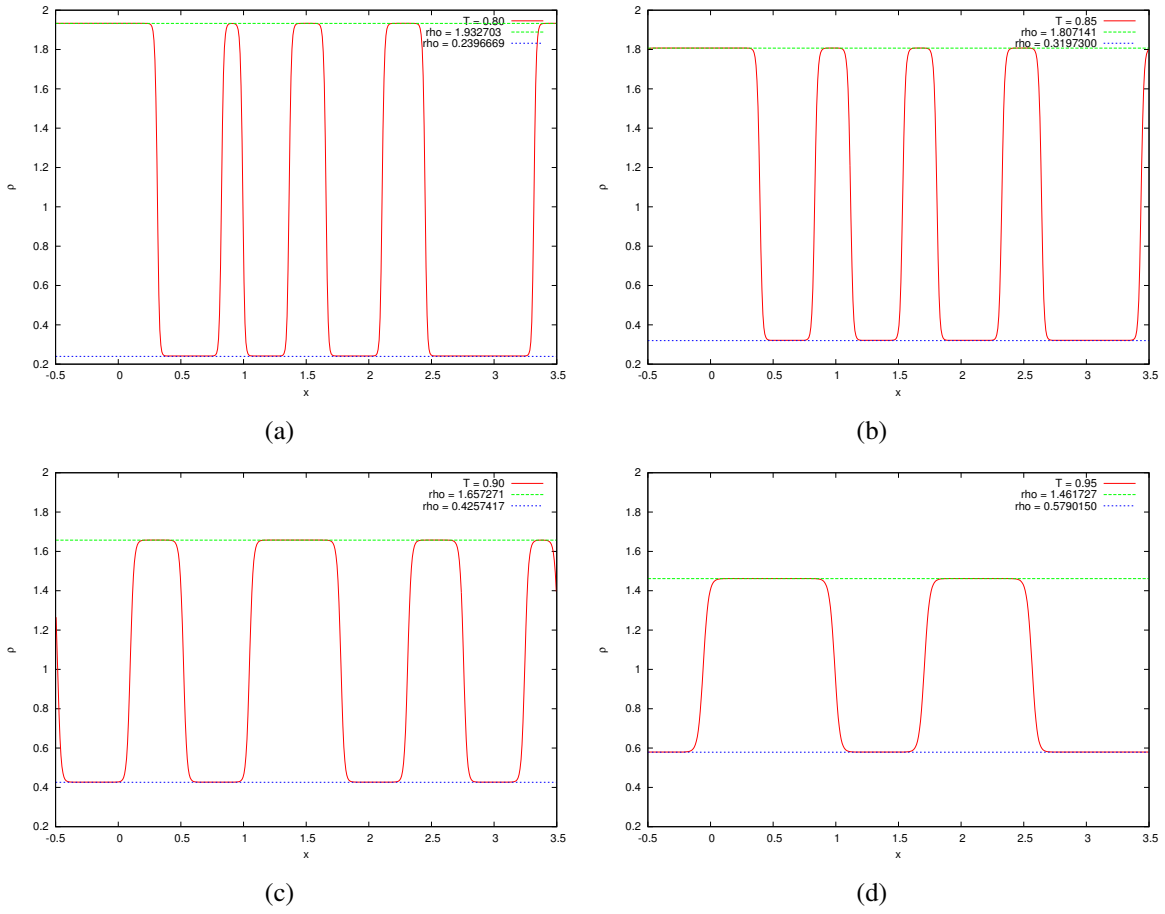
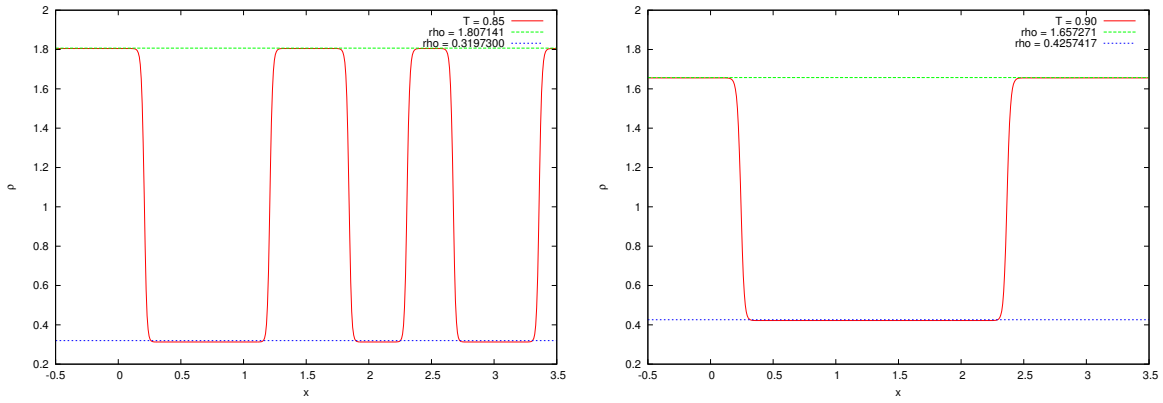
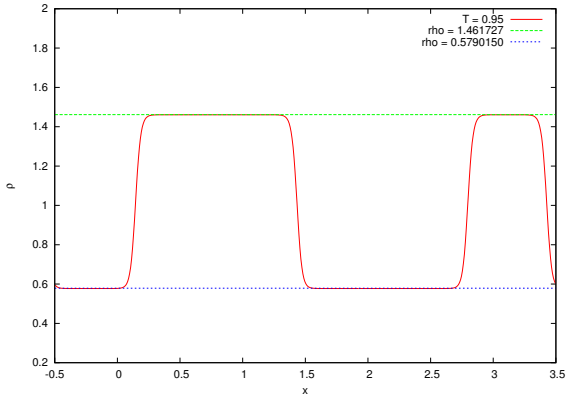


Figure 3.5: Comparison of the phase diagram at $e\nabla = 2$ with theoretic values for liquid and vapors.



(a)

(b)

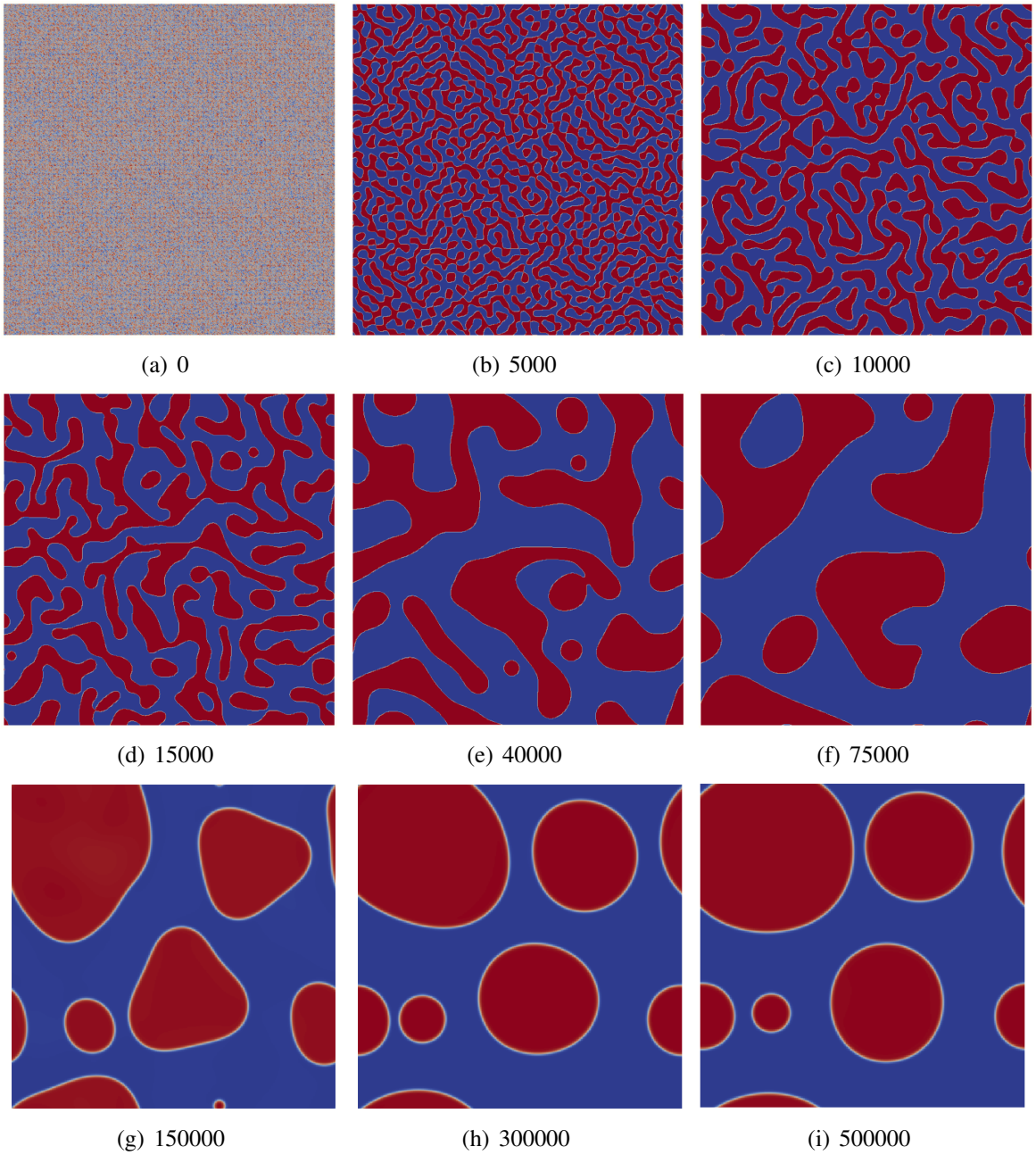


(c)

Figure 3.6: Comparison of the phase diagram at $e\nabla = 4$ with theoretic values for liquid and vapors.

3.2.2 Dynamics of phase separation

The evolution of the phase separation on a 2D lattice with 1024x1024 nodes is presented in Fig. (3.7). After separation, the liquid domains coalesce under the action of the surface tension force and form circular drops, as seen after 300.000 iterations.

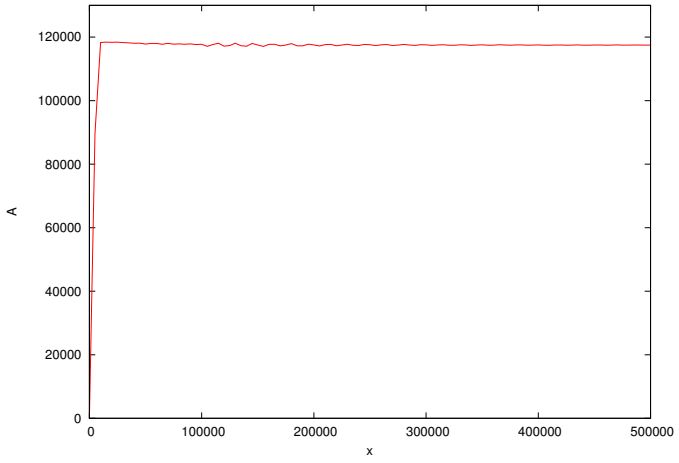


(a) 0 (b) 5000 (c) 10000

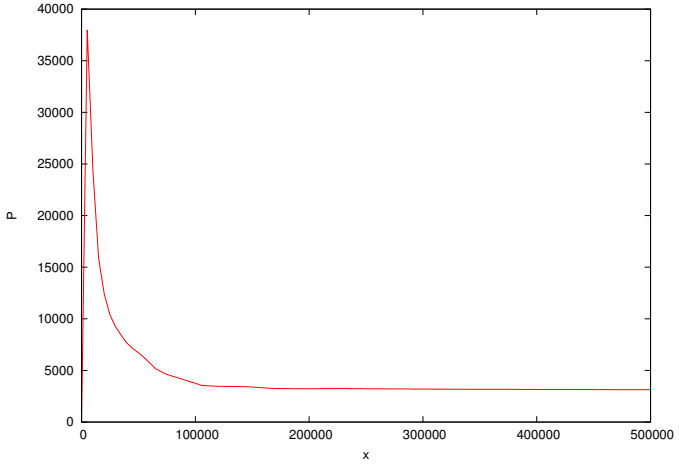
(d) 15000 (e) 40000 (f) 75000

(g) 150000 (h) 300000 (i) 500000

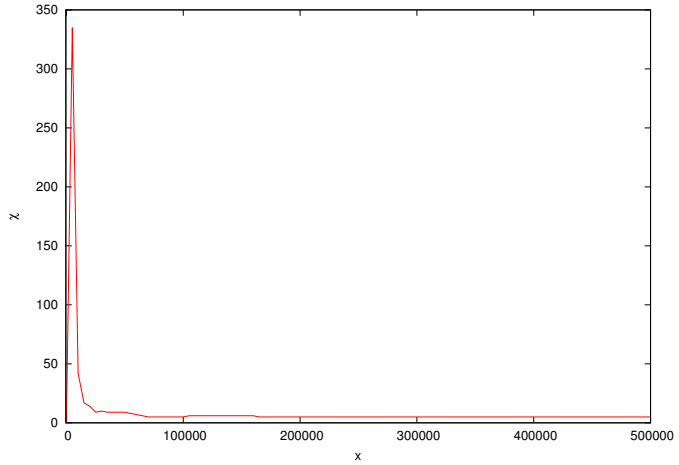
Figure 3.7: Evolution of liquid - vapour phase separation at temperature T = 0.80, $e\nabla = 2$ and $\tau = 0$ on a 2D lattice with 512 x 512 nodes: blue phase = vapour and red phase = liquid.



(a)



(b)



(c)

Figure 3.8: Minkowski functionals at $T = 0.80$ where a. area, b. perimeter, c. Euler characteristic

In Fig.(3.8) we present the evolution of the Minkowski functionals, as calculated during the simulation shown in Figure (3.7). The total area A of the domains grows very fast and becomes quite constant, while the total perimeter and the Euler characteristics evolve in a similar manner: after passing through a maximum, these quantities decrease slowly when the domains start to coalesce. This process is driven by the surface tension that minimizes the liquid-vapor interface.

3.3 Three dimensional simulation

In this thesis, we also used the Lattice Boltzmann method to perform 3D simulations. For this purpose, we used the D3Q27 model, which has 26 non-vanishing velocities and exhibits better numerical stability. Figures (3.9, 3.10, 3.11) show the liquid drops separated at temperature T = 0.85 on a 3D lattice with 128 x 128 x 128 nodes. The dimensionless critical temperature is $T_c = 1.00$. As a result of the coalescence process, only two liquid drops are observed after 215000 iterations (figure 3.12), and form of these drops is quite spherical because of the surface tension.

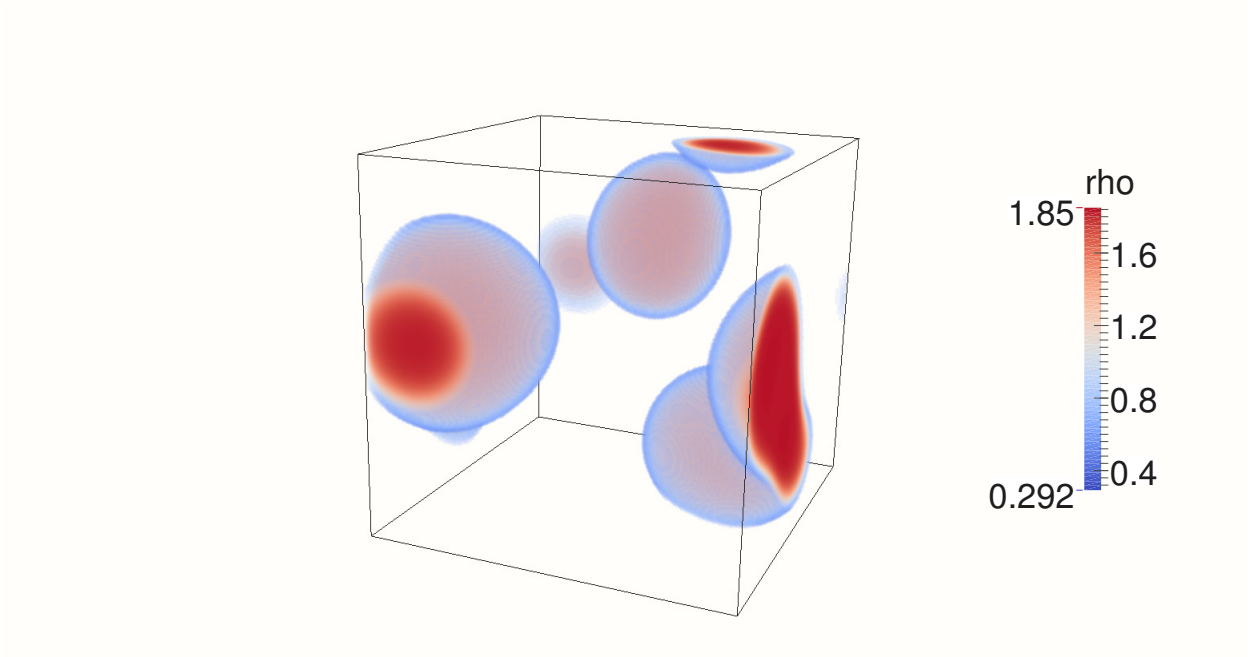


Figure 3.9: 150000 iterations

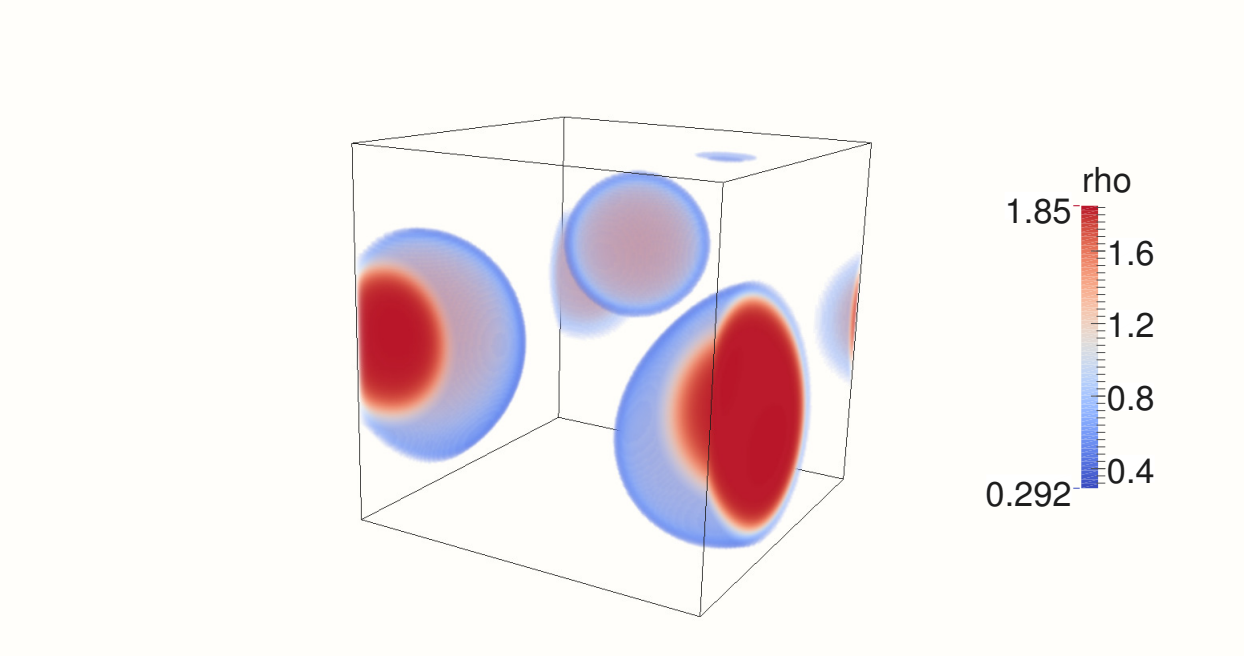


Figure 3.10: 180000 iterations

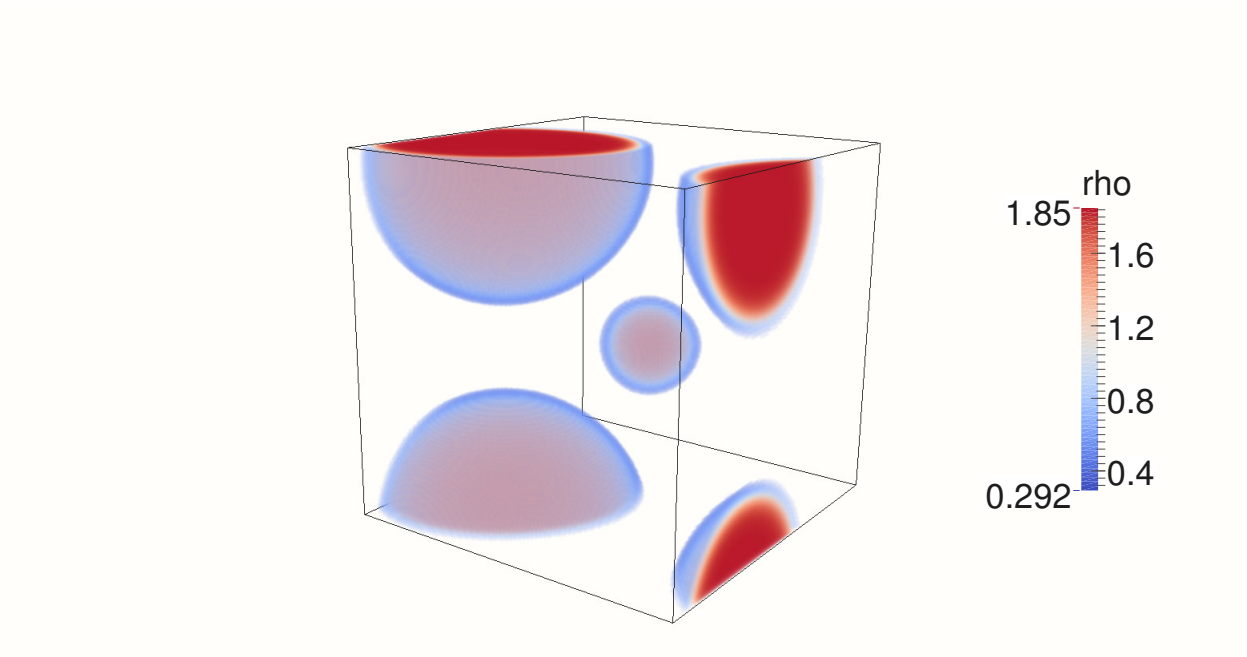


Figure 3.11: 215000 iterations

3.4 Performance studies

The typical dependence of the total run time in case of the lattice Boltzmann model (D2Q9) used on the IBM Blue Gene / P system is presented in Fig. (3.12). Increasing number of cores help us to reduce the total run time. We also build a GPU code for the lattice Boltzmann model (D2Q9) that has a better performance and power consumption in comparison with traditional CPUs. For a typical simulation performed on the GPU with 256 cores for 100.000 iterations, total run time was approximately 10 minutes and on the IBM Blue Gene / P system for the same amount of iteration but with only 128 cores, total run time was 3.5 hours.

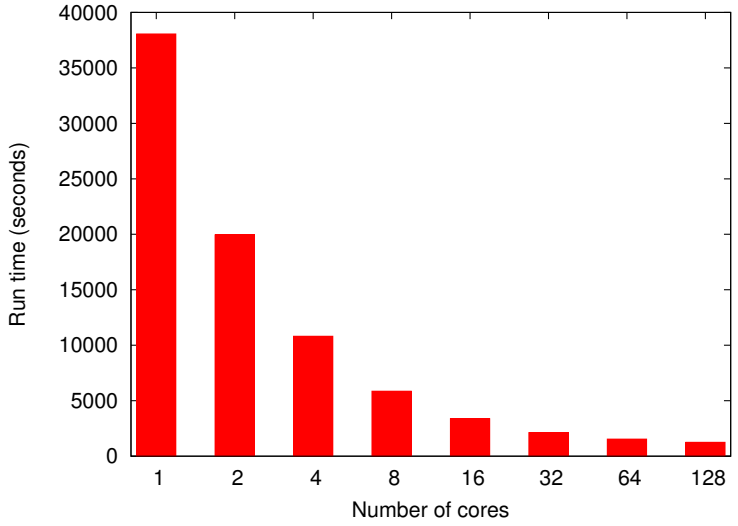


Figure 3.12: Typical dependence of the total run time vs. the number of cores used on the IBM Blue Gene / P system, when running the 2D lattice Boltzmann code for liquid - vapour systems.

4 | Conclusion

In this thesis, we provide a theoretical framework of lattice Boltzmann method using the Hermite expansion approach, where fluid flows can be systematically approximated by constructing high-order lattice Boltzmann models. These models are based on the physics at the mesoscopic scale and provide an alternative to current computational fluid dynamics methods. Due to their local nature, LB models are suitable for parallel computing.

LBM with Hermite polynomials can be used to study the dynamics of van der Waals fluids and phase separation of a isothermal system at various values of the temperature. The numerical scheme presented in chapter two, are used to reduce numerical errors (the spurious velocities in the interface region). Density profiles are not affected by the particular numerical scheme and interfaces are smooth. The interface width can be controlled by the surface tension. These results suggest that numerical schemes have to be carefully chosen when dealing with multiphase systems.

The Minkowski functionals are appropriate for the description of morphology when we want to describe the domains that are formed during phase separation. All the images presented here were obtained by processing the data in a very powerful visualization software, namely Paraview.

In order to approach the new computing technologies, we developed and tested a GPU implementation of our LBM code. Our first results revealed that the GPU code runs approximatively 10 times faster than the code running on the CPU-based Blue Gene P system. This encouraging result strongly support the development of GPU-based simulations to reduce the computational costs.

A Appendix

A.1 Hermite polynomials and Gauss-Hermite quadrature

In a D dimensional space using Rodrigues' formula, we can define the nth-order Hermite polynomial:

$$\mathscr{H}^{(n)}(\xi) = \frac{(-1)^n}{\omega(\xi)} \nabla^n \omega(\xi) \tag{A.1}$$

and the definition of the weight function $\omega(\xi)$ associated with the Hermite polynomials is

$$\omega(\xi) = \frac{1}{(2\pi)^{D/2}} \exp(-\xi^2/2) \tag{A.2}$$

where $\xi^2 = \xi \cdot \xi$. The first few polynomials are :

$$\begin{aligned}
\mathscr{H}^{(0)}(\xi) &= 1 && \text{(A.3)}\\
\mathscr{H}^{(1)}_{\alpha}(\xi) &= \xi_\alpha && \text{(A.4)}\\
\mathscr{H}^{(2)}_{\alpha_1\alpha_2}(\xi) &= \xi_{\alpha_1}\xi_{\alpha_2} - \delta_{\alpha_1\alpha_2} && \text{(A.5)}\\
\mathscr{H}^{(3)}_{\alpha_1\alpha_2\alpha_3}(\xi) &= \xi_{\alpha_1}\xi_{\alpha_2}\xi_{\alpha_3} - \xi_{\alpha_1}\delta_{\alpha_2\alpha_3} - \xi_{\alpha_2}\delta_{\alpha_1\alpha_3} - \xi_{\alpha_3}\delta_{\alpha_1\alpha_2} && \text{(A.6)}\\
\mathscr{H}^{(4)}_{\alpha_1\alpha_2\alpha_3\alpha_4}(\xi) &= \xi_{\alpha_1}\xi_{\alpha_2}\xi_{\alpha_3}\xi_{\alpha_4} - (\xi_{\alpha_1}\xi_{\alpha_2}\delta_{\alpha_3\alpha_4} + \xi_{\alpha_1}\xi_{\alpha_3}\delta_{\alpha_2\alpha_4} + \xi_{\alpha_1}\xi_{\alpha_4}\delta_{\alpha_2\alpha_3}\\
&+ \xi_{\alpha_2}\xi_{\alpha_3}\delta_{\alpha_1\alpha_4} + \xi_{\alpha_2}\xi_{\alpha_4}\delta_{\alpha_1\alpha_3} + \xi_{\alpha_3}\xi_{\alpha_4}\delta_{\alpha_1\alpha_2})\\
&+ (\delta_{\alpha_1\alpha_2}\delta_{\alpha_3\alpha_4} + \delta_{\alpha_1\alpha_3}\delta_{\alpha_2\alpha 4} + \delta_{\alpha_1\alpha_4}\delta_{\alpha_2\alpha_3}) && \text{(A.7)}
\end{aligned}$$

The recurrence relation of Hermite polynomials is :

$$\xi_{\alpha_0}\mathscr{H}^{(n)}_{\alpha_1\ldots\alpha_n} = \mathscr{H}^{(n+1)}_{\alpha_0\ldots\alpha_n} + \sum_{i=1}^{n} \delta_{\alpha_0\alpha_i}\mathscr{H}^{(n-1)}_{\alpha_1\ldots\alpha_{i-1},\alpha_{i+1},\ldots\alpha_n} \tag{A.8}$$

The Hermite polynomials form a set of orthonormal basis of the Hilbert space of function $f(\xi)$, with a scalar product defined by:

$$\int \omega(\xi)\mathscr{H}_{\alpha}^{(m)}(\xi)\mathscr{H}_{\beta}^{(n)}(\xi)\, d\xi = \delta_{mn}\delta_{\alpha\beta}^{n} \tag{A.9}$$

If a function is square integrable we can expand in terms of the Hermite polynomials, in our case $f(\xi)$ is :

$$f(\xi) = \sum_{n=0}^{\infty} \mathbf{a}_{\alpha}^{(n)}\mathscr{H}_{\alpha}^{(n)}(\xi) \tag{A.10}$$

By multiplying (A.10) with $\omega(\xi)\mathscr{H}_{\beta}^{(n)}(\xi)$ and integrating, we obtain:

$$\int \omega(\xi)f(\xi)\mathscr{H}_{\beta}^{(n)}(\xi)\, d\xi = \mathbf{a}_{\alpha}^{(n)}\delta_{\alpha\beta}^{n} = n!\mathbf{a}_{\beta}^{(n)} \tag{A.11}$$

The last equality holds because there are $n!$ distinct permutations of the multi-index $(i_1, i_2, ..., i_n)$ and $\mathbf{a}_{i}^{(n)}$ is symmetric. It is more convenient to use the expression

$$f(\xi) = \omega(\xi)\sum_{n=0}^{\infty}\frac{1}{n!}\mathbf{a}^{(n)}\mathscr{H}^{(n)}(\xi) \tag{A.12}$$

with the expansion coefficients given by

$$\mathbf{a}^{(n)} = \int f(\xi)\mathscr{H}^{(n)}(\xi)\, d\xi \tag{A.13}$$

Applying Gaussian quadrature [6] to a given function $f(\xi)$, we will obtain the best estimate of the integral $\int_a^b \omega(\xi)f(\xi)$ by choosing the optimal set of abscissae ξ_a, $a = 1, ..., n$, like that:

$$\int_a^b \omega(\xi)f(\xi)\, d\xi \cong \sum_{a=1}^{n} w_a f(\xi_a) \tag{A.14}$$

where $\omega(\xi)$ is an arbitrary weighting function and w_a a set of constant weights.
A theorem of numerical analysis on Gaussian quadrature state that the optimal abscissae of the n-point of quadrature are precisely the roots of nth corresponding orthogonal polynomial, and the weights are given by:

$$w_a = \frac{(p_{n-1}, p_{n-1})}{p_{n-1}(\xi_a) p_n^{'}(\xi_a)} \tag{A.15}$$

where $p_n^{'} = dp_n/d\xi$. The Gaussian - Hermite rule integrates exactly to a polynomial of degree $2n-1$. Taking the derivative [6] of (A 1), we obtain

$$\frac{d\mathscr{H}^{(n)}}{d\xi} = \xi\mathscr{H}^{(n)} - \mathscr{H}^{(n+1)} = n\mathscr{H}^{(n-1)} \tag{A.16}$$

With the help of (A 8) and using (A 9), the corresponding weights are

$$w_a = \frac{n!}{[n\mathscr{H}^{(n-1)}(\xi_a)]^2} \tag{A.17}$$

Bibliography

[1] Orestis P. Malaspinas, *Lattice Boltzmann Method for the Simulation of Viscoelastic Fluid Flows* , These N^o 4505 , Suisse (2009).

[2] A. A. Mohamad, *Lattice Boltzmann Method* , Springer (2011).

[3] P.L. Bhatnagar, E.P. Gross and M. Krook, *A model for collision processes in gases I: small amplitude processes in charged and neutral one-component system* , *Physical Review Vol. 94, (1954), pp. 511-525.*

[4] Michel O. Deville, Thomas B. Gatski, Mathematical Modeling for Complex Fluids and Flows *, Springer (2012).*

[5] D. Vizman, V. Sofonea, A. Cristea, Advanced numerical methods and applications *, Ed. EUROBIT (2008).*

[6] X. Shan, Xue-Feng Yuan, H. Chen, Kinetic theory representation of hydrodynamics: a way beyond the Navier-Stokes equation *, Journal of Fluid Mechanics (2006) , vol. 550, pp. 413-441.*

[7] V. Sofonea, A. Lamura, G. Gonnella and A. Cristea, Finite-difference lattice Boltzmann model with flux limiter for liquid-vapor systems *, Physical Review E 70, (2004).*

[8] V. Sofonea, R. F. Sekerka, Viscosity of finite fifference fattice Boltzmann models *, Journal of Computational Physics 184 (2003) 422.*

[9] A. Cristea, V. Sofonea, Two component lattice Boltzmann model with flux limiters *, Central European Science Journals of Physics, (2004).*

[10] P. Colella, Multidimensional Upwind Methods dor Hyperbolic Conservation Laws *, Journal of Computational Physics 87, (1990).*

[11] R. J. Leveque, Finite Volume Methods for Hyperbolic Problems *, Cambridge University, (2002).*

[12] R. J. Leveque, , Society for Industrial and Applied Mathematics Vol. 33, No. 2, (1996), pp. 627 - 665.

[13] D. Padua, Encyclopedia of Parallel Computing *, Springer (2011).*

[14] PETSc Manual, http://www.mcs.anl.gov/petsc/petsc-current/docs/manual.pdf

[15] D. B. Kirk, Wen-mei W. Hwu, Programming Massively Parallel Processors, Second Edition *, Nvidia (2012)*

[16] V. Sofonea and K. R. Mecke, Morphological characterization of spinodal decomposition kinetics *, The European Physical Journal B 8, pp. 99-112, (1999).*

[17] K. Michielsen, H. De Raedt, Integral - Geometry Morphological Image Analysis *, Physics Reports 347, pp. 461 - 538, (2001).*

[18] J. Sanders, E. Kandrot, Cuda by Example *, Nvidia (2011).*