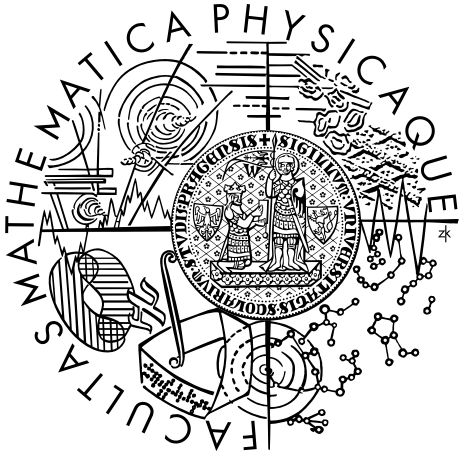
Charles University in Prague
Faculty of Mathematics and Physics

MASTER THESIS



Miroslav Tomasik

Simulation of two-dimensional flow past obstacles using lattice-gas cellular automata

Institute of Theoretical Physics

Supervisor of the master thesis: Martin Scholtz
Study programme: Physics
Study branch: Mathematical modelling

Prague 2017

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ..... signature of the author

Title: Simulation of two-dimensional flow past obstacles using lattice-gas cellular automata

Author: Miroslav Tomasik

Institute or Department: Institute of Theoretical Physics

Supervisor: Martin Scholtz, Institute of Theoretical Physics

Abstract: Cellular automata constitutes original computational methods, that found its application in various scientific disciplines. The special class of cellular automata, the lattice gas automata were succesfull in dealing with many challenges of hydrodynamic simulations, and they bootstraped one of the most perspective CFD methods, the Lattice Boltzmann models.

In the theoretical part, we follow the evolution of the lattice gas automata, explore the theory behind them, and from their microdynamics, we derive the hydrodynamic equations.

In the practical part, we implemented two most distinguished types of LGCA, the Pair-interaction automaton and FCHC. We applied them on the flow around obstacles of various shapes. The scientifically most relevant part concerns statistical properties of the turbulent flow simmulated by LGCA, but requires further research to conclude it.

Keywords: Cellular automata FCHC Pair-interaction Turbulence

Contents

Part I

Theory

1. Cellular automata

Years before DNA and replication mechanism was discovered in living cells, John von Neumann was investigating self-replicating systems in theory, and layed basis for the "New kind of science" [4].

1.1 Game of Life

John Conway, by significant simplification of von Neumann ideas, introduced Game of Life in 1970 that renewed general interest in cellular automata.

Depending on the initial conditions, evolution of this automaton can be chaotic, periodic or it can lead to the stable configurations.

The reason for this complexity is that Game of Life is Turing-complete – in principle it can simulate any computer program and thus various behavior can be observed.

For the purposes of this thesis, we have written a simple desktop application implementing this game, since we can easily explain the basic principles of cellular automata on it.

Let us have a rectangular grid, with black and white squares. White squares represent dead cells, black cells represent living cells. On the figure 1.1 we see such a grid, that we chose for the initial state of 'Life'.

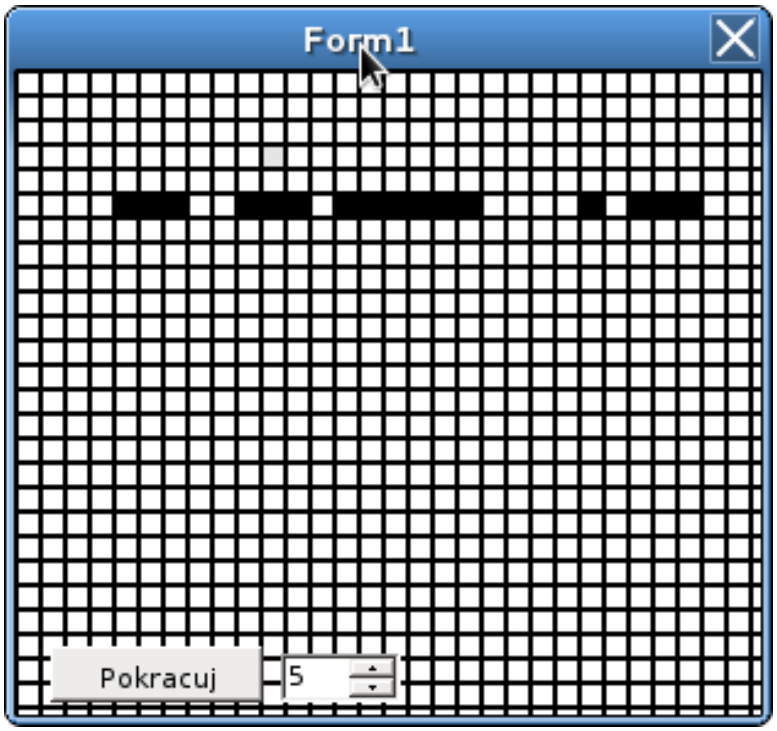


Figure 1.1: The initial state of 'Life' (at t=0)

Now we press 'Pokracuj' button and let the the Life evolve. In the discrete time steps, the grid is changing. We see that some cells are dying, but some cells are getting alive. What is the rule that kills the cell or leave it be?

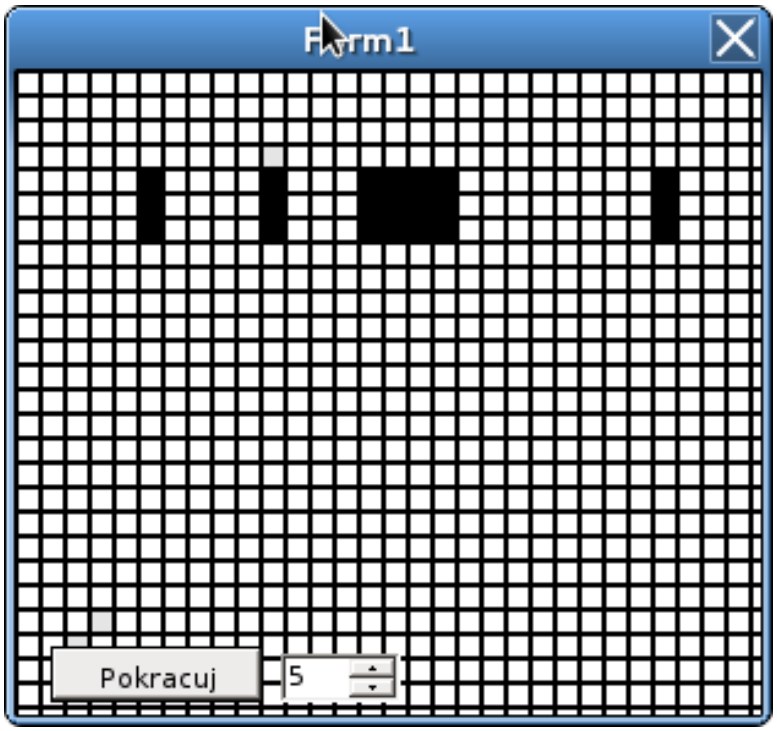


Figure 1.2: t=1

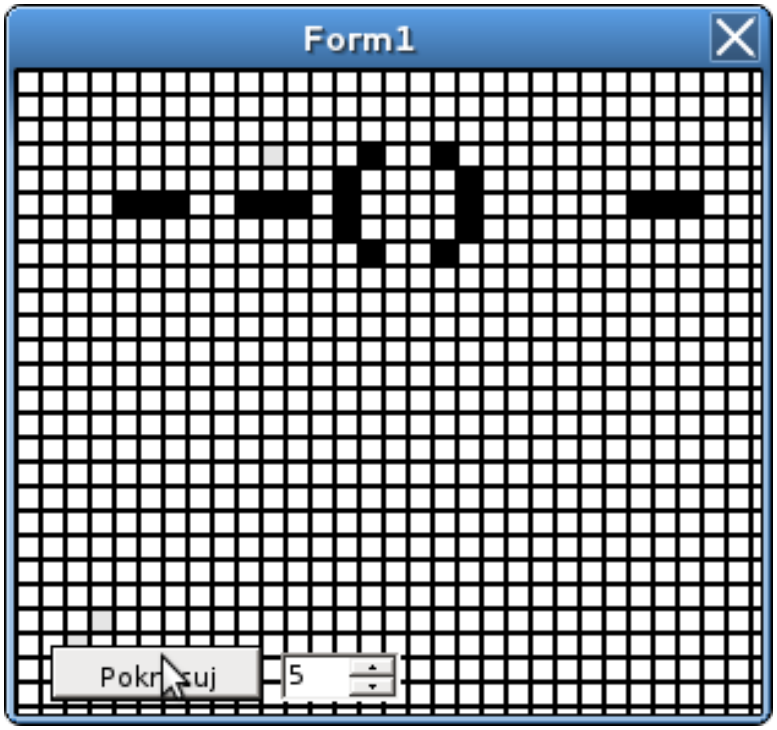


Figure 1.3: t=2

1. If the cell is alive, and 2 or 3 neighbouring cells are alive, the cell will stay alive in the next step. Otherwise it will die.

2. If the cell is dead, and **exactly** 3 neighboring cells are alive, the cell will get alive in the next step. Otherwise, it stays dead.

We see that the rule involves only the state of **the cell** and the states of **its eight neighboring cells**.

Let us proceed from this simple example. In the next section, we will generalize main features of 'Life' and formally define the cellular automaton.

1.2 Cellular automaton in general

1. **Position of the cells:**

   Instead of two dimensional, rectangular grid of 'Life', cells might be arranged on the regular grid of arbitrary dimensional. (Regularity follows from definition of Kubrid. In general, cells might be positioned really wildly, e.g. on Penrose lattice, or arbritrary as proposed by Feynman).

2. **Set of cell states Q:**

   In 'Life' cells can be dead or alive, but generally, set of states Q for the cell can have an arbitrary finite size.

3. **Neighborhood:**

   In 'Life', state of the cell in the next step was determined by the nearest neighboring cells. We call these cells neighborhood of range r = 1 (in the distance by 1 cell). However, the rule might involve neighborhood with the arbitrary range, see figure 3

4. **Update rule:**

   Update rule is an arbitrary bounded mapping from the *neighborhood* to the set of states Q. Since the state of the cell is determined only by the state of its neighborhood, update rules in cellular automata are local.

1.3 The most basic cellular automaton

In middle 1980s on the prestigious Princeton institute, Stephen Wolfram and his assistants were performing unusual computer experiments. They were simulating the evolution of the one dimensional cellular automata and they were analyzing the obtained patterns [18] (to a despair of their senior colleagues, who did not understand this "new kind of science")[4].

The most basic, one dimensional cellular automaton we can imagine, is the two state automaton with range $r = 1$.

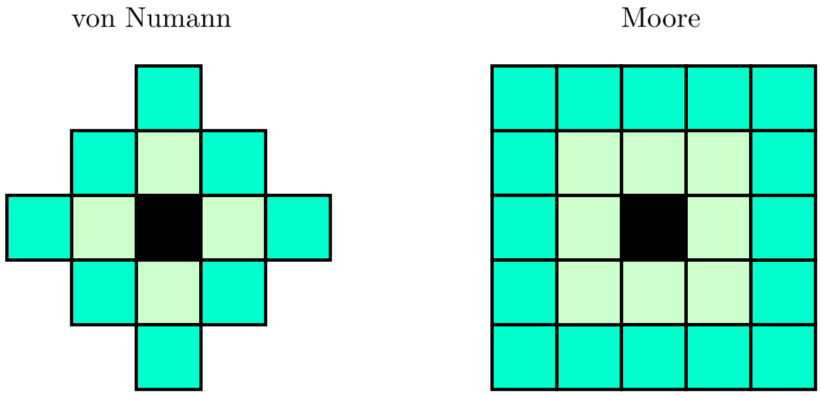


Figure 1.4: Moore's and von Neumann's neighborhood

Figure 1.5: A state of one dimensional cellular automaton

a_t^{i-1}	a_t^i	a_t^{i+1}	a_{t+1}^i
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	1
0	1	1	1
0	1	0	0
0	0	1	1
0	0	0	0

Table 1.1: Rule 90

One dimensional indicates that the cells are arranged in the row (figure 1.3), range $r = 1$ indicates that the update rule for a cell involves only its next neighbors. An example of an update rule is shown the table 1.1.

The three columns represent the state of a cell a_t^i and its left and right neighbor, a_t^{i-1} and a_t^{i+1} respectively. A living cell is denoted by 1, a dead cell is denoted by 0. The last column denotes the state of the middle cell a_i in the next step $t + 1$. The sequence of *one*s and *zero*s in the last column is the binary representation of a rule. In this particular case, it represents Rule 90. Since there is $2^8 = 256$ combinations for the last columns, there is 256 rules for this most basic cellular automaton.

The update rule that we have just described can be formally written as

$$a_i^{(t)} = f\big[\sum_{j=-r}^{j=r} \alpha_j a_{i+j}^{(t-1)}\big] \tag{1.1}$$

where a_i refer to the state of the j^{th} cell, α_k are the integer weights, and f is the function that takes integer as the single argument, and r is the range of the neighborhood.

1.4 Classification of 1D cellular automata

According to Wofram [2], these automata can be classified analogously to the dynamical systems.

(a) **Limit point:**
The final configuration is homogeneous.
Rules number $0, 4, 16, 32, 36, 48, 54, 60, 62$.

(b) **Limit cycles:**
Simpe time-periodic patterns.
Rules number $8, 24, 40, 56, 58$.

(c) **Strange attractors:**
Chaotic patterns, see the figure 1.6.
Rules number $2, 6, 10, 12, 14, 18, 22, 26, 28, 30, 34, 38, 42, 44, 46, 50$.

The rule 110 is special among them – as Cook has proved [1],this rule is equivalent to the Turing machine, so depending on the initial conditions, it could be classified in any of the three classes above.

Below, we present figures capturing the evolution of several of the mentioned rules, that were obtained by our implementation of 1D cellular automata.

All of them were initialized by the row of dead cells, only the middle cell was alive.

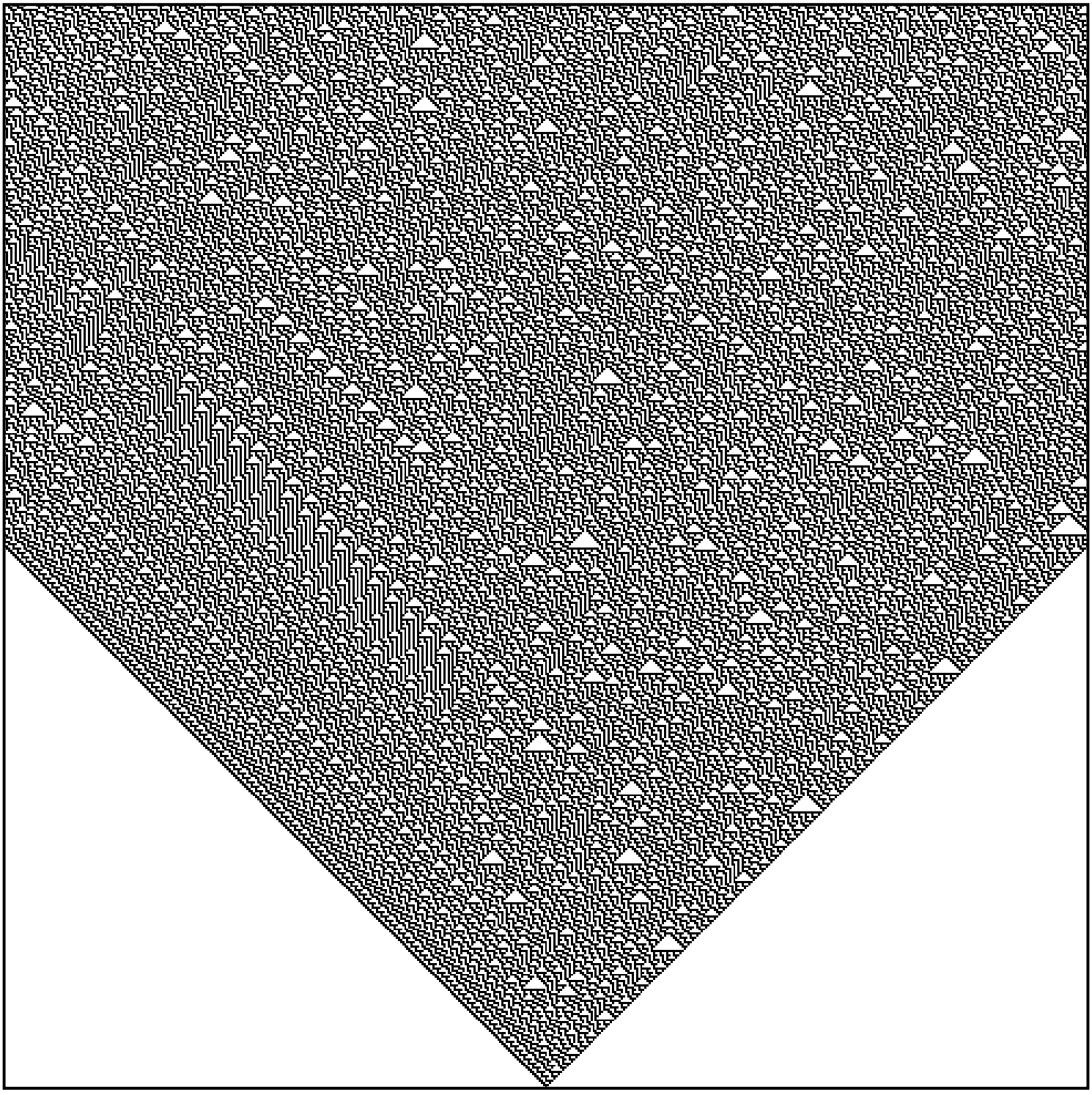


Figure 1.6: Rule 30

1.5 Cellular automata in physics

In his visionary book New kind of Science [2], Wolfram suggest possibility, that in the infinitely large world of cellular automata, there is an automaton that

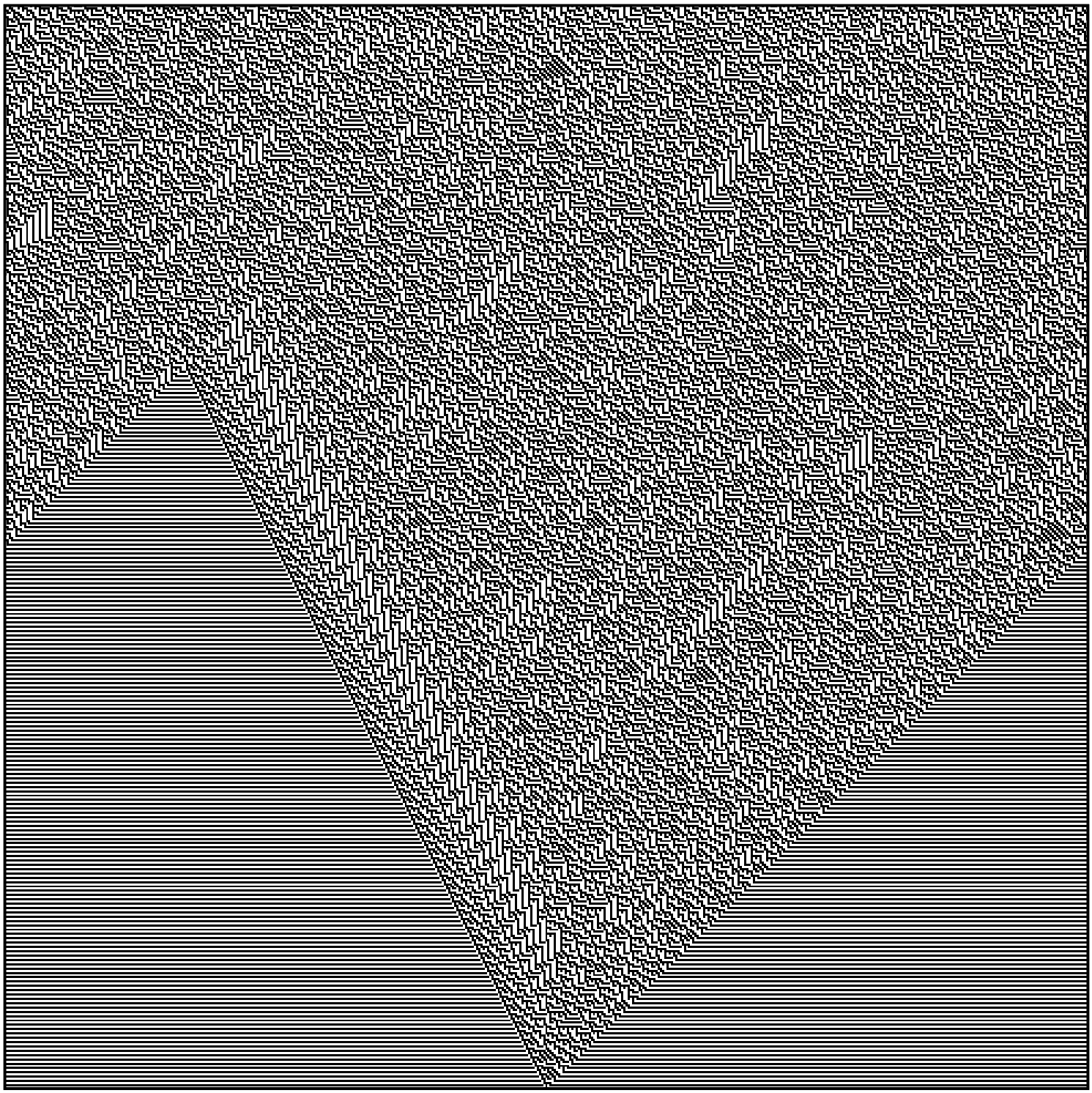


Figure 1.7: Rule 45

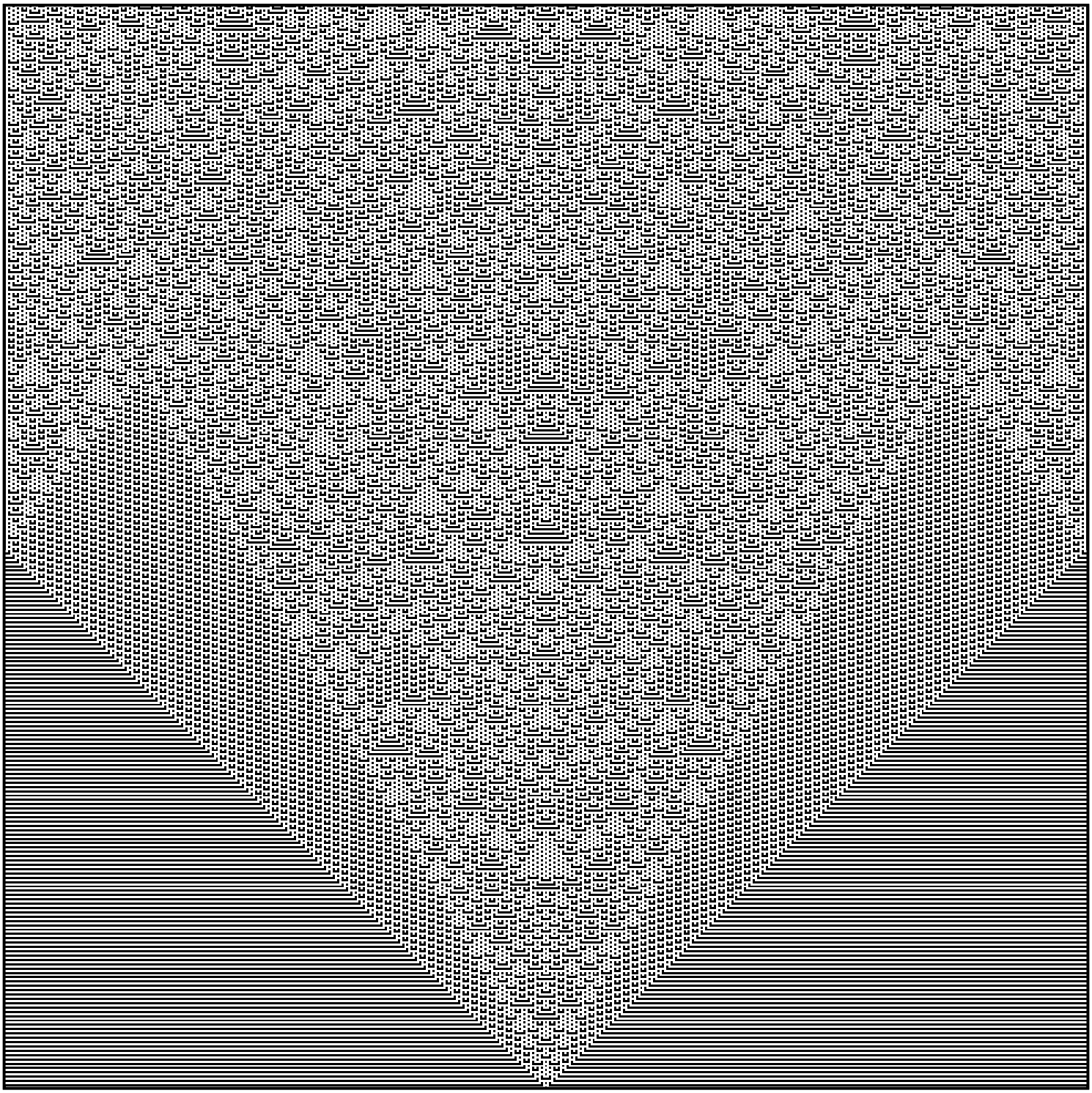


Figure 1.8: Rule 73

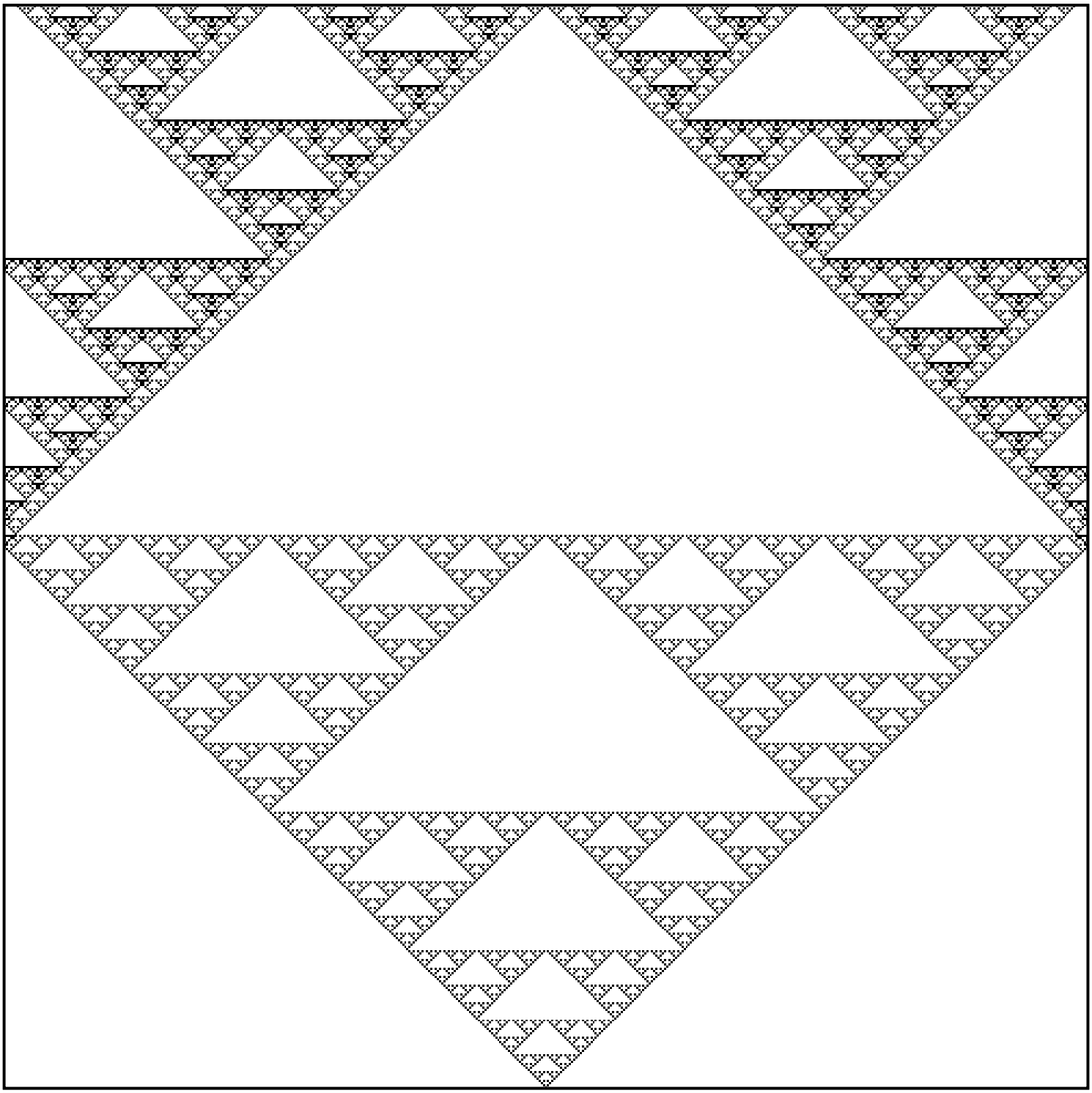


Figure 1.9: Sierpinski carpet - rule 90

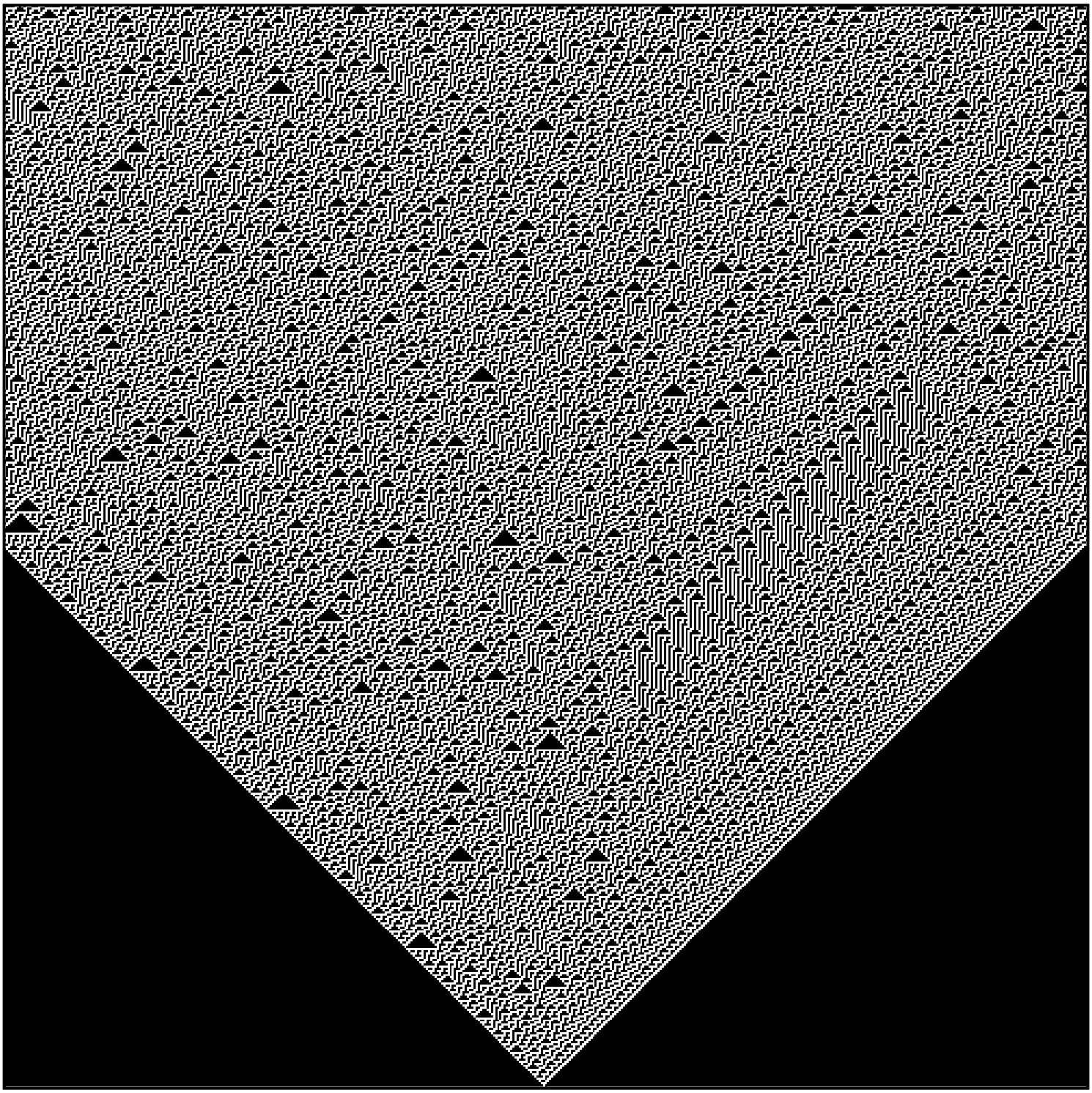


Figure 1.10: Rule 149

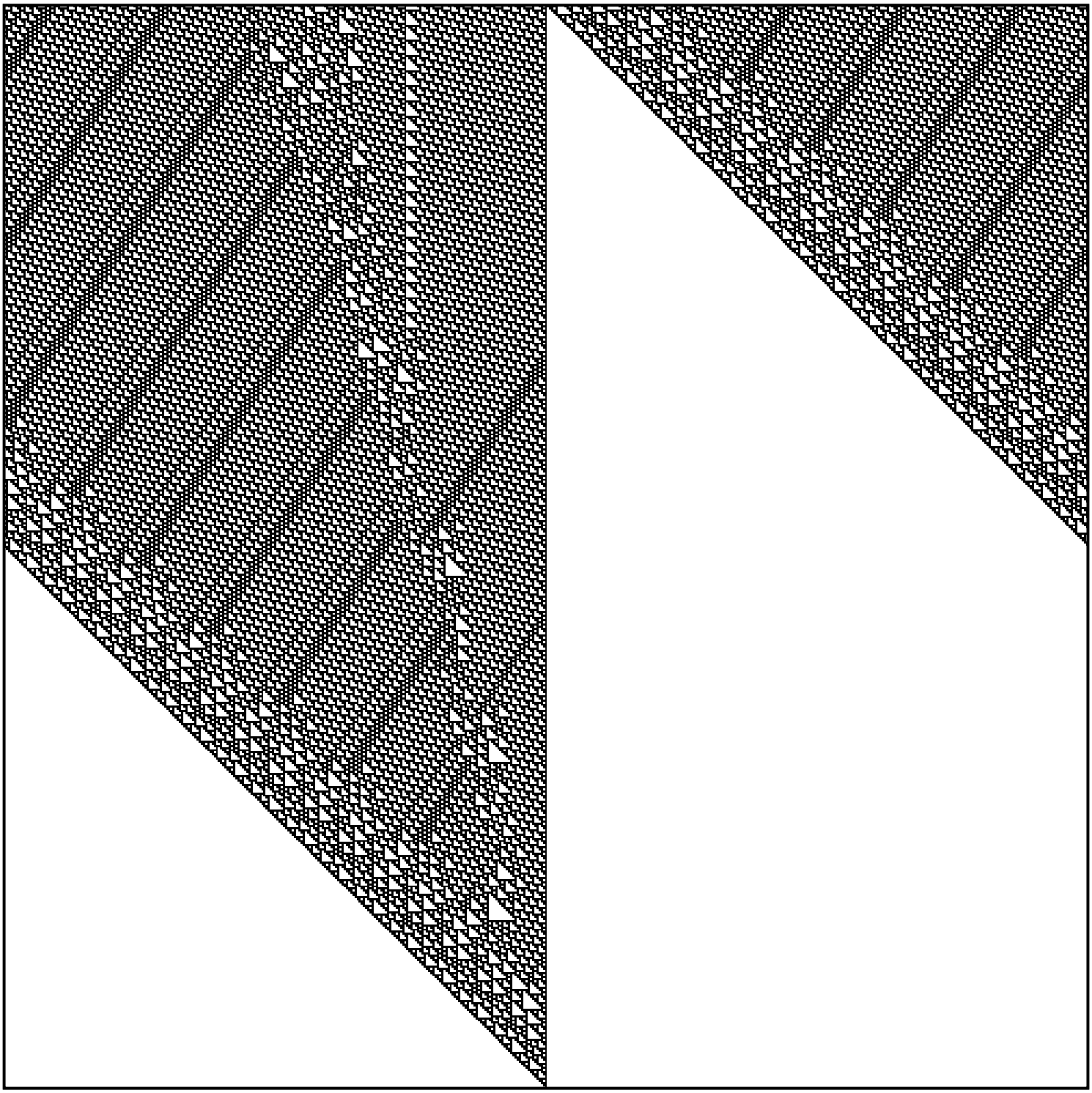


Figure 1.11: Rule 110

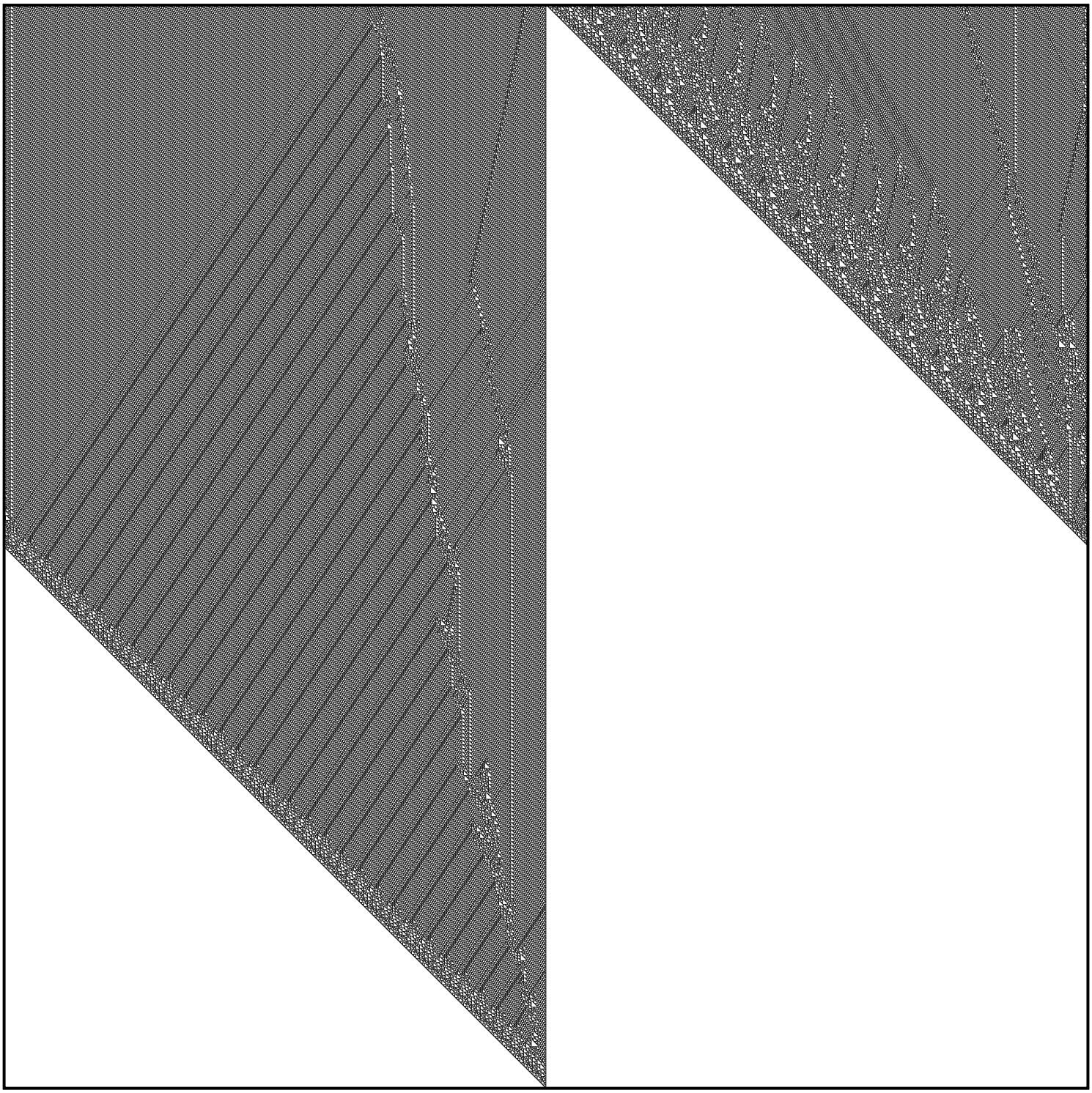


Figure 1.12: Rule 110 – 2000 × 2000

constitute a unified theory of the physics. Although the details of his idea met with rejection(see [16]), many notable physicists are attempting to construct such cellular automaton [17].

However, focus of our work is much more modest that cellular automata describing universe. We are interested in the cellular automata that could model flow of the fluids.

There is already a formal similarity between cellular automata we presented, and discrete partial differential equations. Consider the diffusion equation

$$\frac{\partial C}{\partial t} = \kappa \frac{\partial^2 C}{\partial x^2}.$$

By discretization forward in time, it is transformed to

$$C_i^{(t)} = f\big[\sum_{j=-1}^{j=1} \alpha_j \, C_{i+j}^{(t-1)}\big]$$

Formally, this equation corresponds to the rule of the one dimensional cellular automaton 1.1. However, in case of these equation, C_i is a continuous value, not discrete, the α_i are reals, not integers, which would cause instability of cellular automaton, as discussed in [2] and thus the similarity is only formal.

The connection of cellular automata with the fluid mechanics is not this straight-forward, but it is grounded in the symmetries and conservation laws found in the hydrodynamic equations and in the special class of cellular automata alike.

In these symmetries lies not only beauty of this method and their attractiveness for physicists, but their principal advantage over the better-established CFD methods.

2. Lattice gass cellular automata

The first lattice gass cellular automaton was proposed in 1973 by Hardy, Pomeau and de Pazis [11], and so it is called the HPP model.

Unfortunately, it could not do its job sufficiently well. For the reasons that we will sketch in this chapter, it does not converge to Navier-Stokes equations in macroscopic limit.

In the subsequent chapters we will explore two different approaches how to make functional LGCA. They both build on the idea of this imperfect HPP.

Therefore, we will explain the basic principles of HPP first, and later on, we will upgrade it to FHP, FCHC and our favorite Pair-interaction cellular automaton.

2.1 From CA to LGCA

The lattice of HPP is the simple rectangular 2D grid. At every point of a grid is a node, and this node is composed of 4 cells, see Figure 2.1.

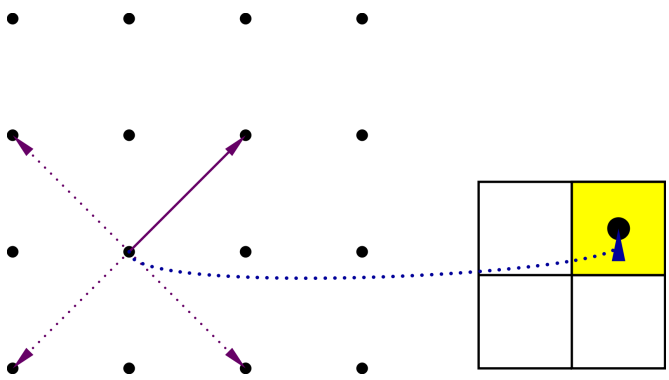


Figure 2.1: Rectangular grid

Each of these cells can be in two states– empty (white square) or occupied by the particle (yellow square). The particle in this cell is heading to the diagonal node along the corresponding lattice vector.

2.2 Update rule

At every time-step, the position of the particles is changing in two subsequent steps – collision and propagation.

During collision, particles are swapped inside the single node, respecting two constraints - number of particle and the total momentum is fixed.

From these constraints follows that there are only two collision configurations for HPP, see Figure 2.2.

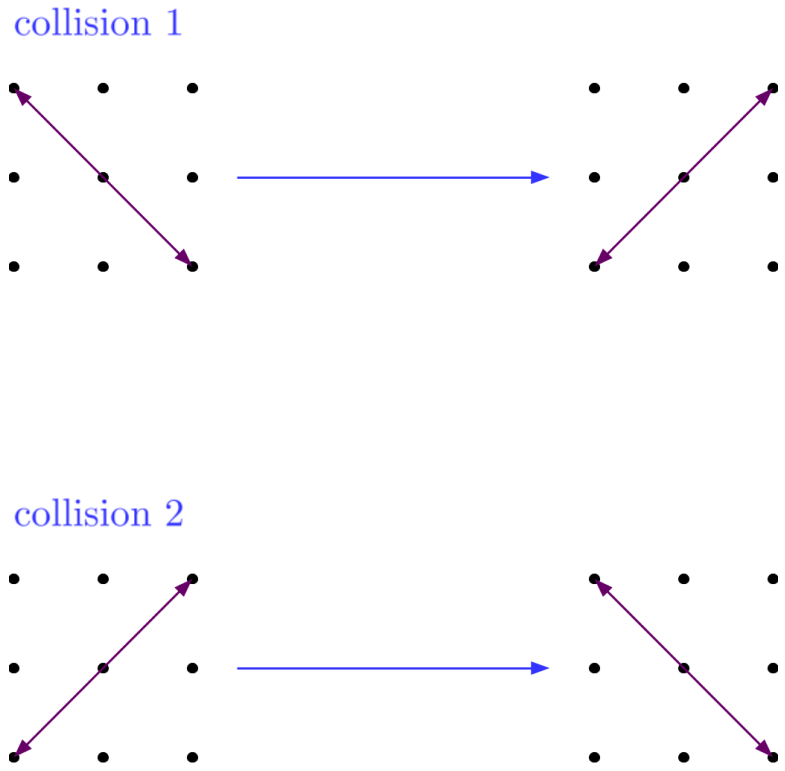


Figure 2.2: HPP colisions

These configurations are symmetric – the first one is resolved to the other and vice versa.

2.3 Propagation:

After the collisions are resolved in every node, propagation follows, see Figure 2.3. During propagation, particles are streamed along the lattice vectors corresponding to the cells they occupy.

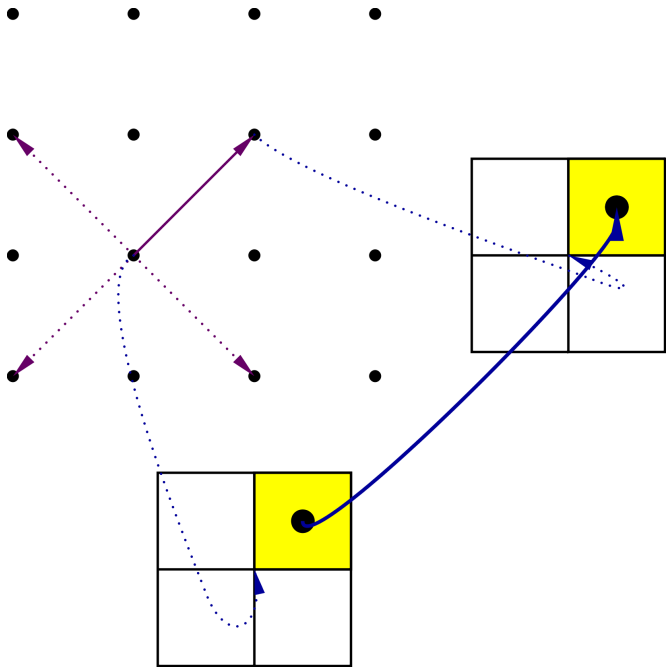


Figure 2.3: Propagation of particle from upper-left cell

2.4 Conservation laws

Let us inspect the mass and momentum conservation of this model in more depth by considering its symmetries. We assume that lattice is infinite (or finite, but periodic boundary condition are used). Then if we shift the lattice by multiple of lattice vector, we get the same lattice – the lattice is invariant with respect to translation. As we know by Norther's theorems, translational symmetry implies conservation of momentum.

Also, the grid possesses a rough rotational symmetry - rotation by the 90° leads to the same lattice. However, this rectangular symmetry is not sufficient. When we will derive the hydrodynamic equations in the following chapter, we may observe that four lattice vectors lead to unrealistic equations, comparing to the models with six and more lattice vectors.

Another problem caused by low rotational symmetry are the non-physical quantities that are conserved nevertheless – so called spurious invariants [1].

Let us decompose the total momentum of the collision configuration 2.2 into the cardinal directions:

$$P = P_N + P_S + P_E + P_W. \tag{2.1}$$

The total momentum P is correctly conserved by the collision, but also quantities

$$P_{spur1} = P_N + P_E - P_S - P_W \tag{2.2}$$

[1] The improved LGCA also posses the spurious *Zanetti's invariants*, but they are under level of noise, due to higher symmetry and additional collision rules

and

$$P_{spur2} = P_N + P_W - P_S - P_E \tag{2.3}$$

are conserved, although these quantities have no physical counterparts.

2.5 Conclusion

To conclude this chapter and finish-off the HPP, it is physically implausible because

1. angular momentum is not conserved due to insufficient rotational symmetry,
2. other non-physical quantities, so called *spurious invariants* are conserved.

Although it is a flawed model, it sparked interest of the wider community and various successful LGCAs evolved from HPP. In the next chapter, we will introduce the first successful branch of LGCAs – the FHP model.

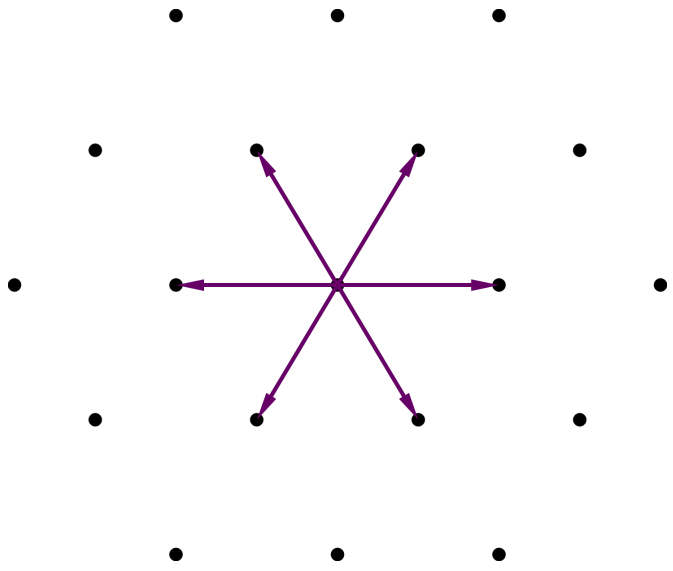
3. FHP

This enhanced lattice gas cellular automaton is also named after its inventors – Frisch, Humpfrey and Pomeu. They proposed it in 1986 [9] together with its three dimensional variant - the FCHC. In following section, we will graphically explain the basic principles of FHP, the later sections will be more general and the obtained results will be valid for arbitrary dimensional FHP-like automaton (most importantly FCHC).

3.1 The lattice of FHP

The whole improvement of FHP lies in a simple change - instead of the square grid, FHP builds on a hexagonal grid. All other properties are implied by the increased symmetry of the lattice.

On the figure 3.1, we have a part of the hexagonal grid, and from one of the nodes, six lattice vectors point to the neighboring nodes. Let us denote the set of the lattice vectors by $c_i,\ i = 1, 2, 3, 4, 5, 6$.



The node is a set of the six cells and each of the cells corresponds to one of the lattice vectors. Let us denote the state of the node by $\boldsymbol{n} = (n_1, n_2, n_3, n_4, n_5, n_6)$, where $n_i = 0$ stands for an empty i^{th} cell, and $n_i = 1$ implies that there is a particle in the i^{th} cell.

State of a node with the position $\mathbf{r}$ on lattice will be denoted by $\mathbf{n}(\mathbf{r})$, whereas the state of the *whole lattice* will be denoted by $\mathbf{n(.)}$.

3.2 Update rule

As we know, the update happens in discrete time steps ($t = 1, 2, 3...$) and consists of two subsequent steps - collision and propagation. Both of these steps are local, so they can be treated node by node.

3.2.1 Propagation

Propagation is the straight-forward phase and can be captured by the simple equation

$$Sn_i(\boldsymbol{r}) = n_i(\boldsymbol{r} + \boldsymbol{c_i}).$$

If the cell is occupied by a particle, that means $n_i(\boldsymbol{r}) = 1$, it propagates along the corresponding lattice vector to the neighboring node, see the figure 3.1.

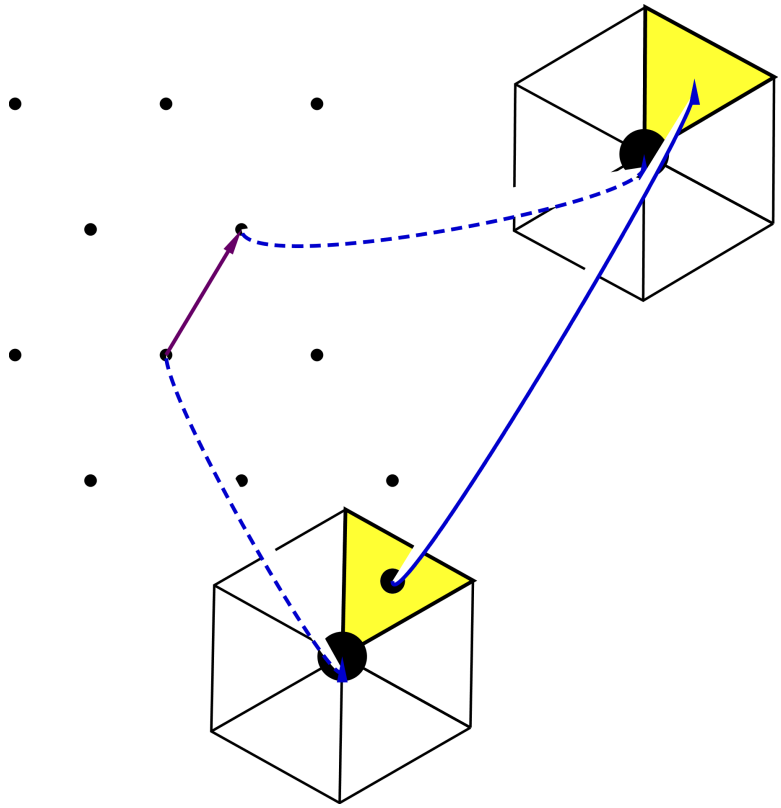


Figure 3.1: FHP collisions without rest particle.

The propagation, however, is preceded by the more difficult step – the collision.

3.2.2 Collision

The purpose of the collision is to swap as many particles in the node as possible. The only constraint on the collision rule is to preserve the number of particles (conservation of the mass) and to preserve the total momentum in the node.

These requirements lead only to a handful of collision configurations, see figure 3.2. For the simplicity, we are considering the FHP-I model, that does not included the "rest particles", otherwise there would be few more rules to add.

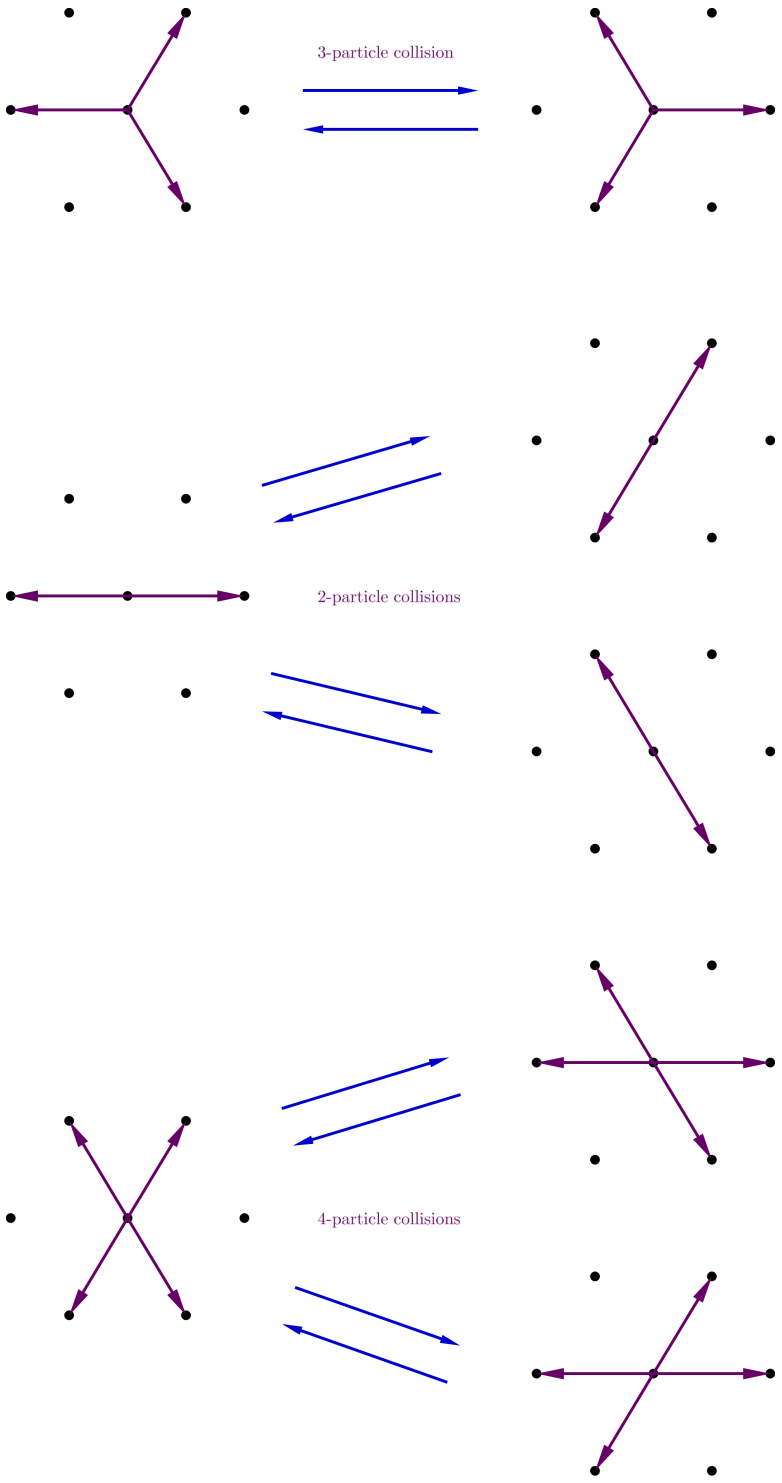


Figure 3.2: FHP collisions without rest particle.

We see that two and four particle configurations can be resolved in two different configurations. The resulting state need to be chosen randomly, with probability 1/2 for each state to preserve the parity symmetry of the model. If we were systematically preferring one of them, we would introduce additional, non-

physical invariant - chirality. Hence, we need to introduce non-determinism to the model.

We will express the probabilities of transition from state n to state n' by the probability matrix

$$A(n \to n') \geq 0.$$

As we have 64 possible states of the node, matrix A is of dimension 64×64. For example, the submatrix of A that governs the head-on collisions is

$$A' = \begin{bmatrix} 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & 0 \end{bmatrix}$$

Since the collisions are symmetric, matrix A is symmetric as well.

Also, collisions are invariant to rotations and reflections of the node

$$A(g(\boldsymbol{n}) \to g(\boldsymbol{n'})) = A(\boldsymbol{n} \to \boldsymbol{n'}),$$

where $g \in G$, and G is the symmetry group of the node. The normalization of probability leads to the semi-detailed balance

$$\sum_{\boldsymbol{n}} A(\boldsymbol{n} \to \boldsymbol{n'}) = 1 \tag{3.1}$$

3.3 Collision operator

Interestingly, we can express the whole update step in one simple equation using collision operator Δ_i:

$$n_i(t+1, r+c_i) = n_i(t,r) + \Delta_i(t,r) \tag{3.2}$$

where Δ_i is

1. $\Delta_i = 0$ if no collision is happening in $n(t,r)$. The state of the cell $n_i(t,r)$ only propagates to $n_i(t+1, r+c_i)$.
2. $\Delta_i = 1$ if there is not particle in $n_i(t,r)$ yet, but gets there after collision.
3. $\Delta_i = -1$ if there is particle in the $n_i(t,r)$, but after the collision, cell gets empty.

By this reasoning, we obtain the collision operator for FHP-I model

$$\begin{aligned} \Delta_i = {} & n_{i+1}n_{i+3}n_{i+5}(1-n_i)(1-n_{i+2})(1-n_{i+4}) \\ & -n_i n_{i+2} n_{i+4}(1-n_{i+1})(1-n_{i+3})(1-n_{i+5}) \\ & +\xi n_{i+1}n_{i+4}(1-n_i)(1-n_{i+2})(1-n_{i+3})(1-n_{i+5}) \\ & +(1-\xi)n_{i+2}n_{i+5}(1-n_i)(1-n_{i+1})(1-n_{i+3})(1-n_{i+4}) \\ & -n_i n_{i+3}(1-n_{i+1})(1-n_{i+2})(1-n_{i+4})(1-n_{i+5}). \end{aligned} \tag{3.3}$$

3.4 Microscopic conservation laws

We can easily prove that

$$\sum_i \Delta_i(t,r) = 0,$$
$$\sum_i c_i \Delta_i(t,r) = 0.$$

by substituting 3.3 into these equations. Combining with 3.2, these equations imply the conservation of mass and momentum

$$\begin{aligned} \sum_i n_i(t+1, r+c_i) &= \sum_i n_i(t,r), \\ \sum_i c_i n_i(t+1, r+c_i) &= \sum_i c_i n_i(t,r). \end{aligned} \tag{3.4}$$

By employing the aparatus of the statistical physics we will show that these microscopic conservation laws lead to physically plausible macroscopic description.

3.5 Conservation of probabilities

Let us define the phase space Γ as the set of all possible states of the lattice $n(.)$. Imagine we want to initialize cellular automaton with some macroscopic velocity $\boldsymbol{v_0}$, macroscopic pressure p_0, and macroscopic density ρ_0. We can realize this macrostate by very many microstates of lattice $\boldsymbol{n(.)}$. We assign initial probability to each of these microstates

$$P(0, \boldsymbol{n(.)}) \geq 0.$$

As always, probabilities over whole lattice are normalized:

$$\sum_{\boldsymbol{n(.)}} P(0, \boldsymbol{n(.)}) = 1.$$

In the statistical mechanics, Liouville's space state theorem postulates that the density of the phase space is constant. Microdynamics of our model implies equivalent theorem for LGCA:

$$P(t+1, \mathcal{E}\boldsymbol{n}(.)) = P(t, \boldsymbol{n}(.))$$

It is obtained directly by applying the update formula

$$\mathcal{E}\boldsymbol{n}(t,.) = \boldsymbol{n}(t+1,.)$$

However, for FHP and other non-deterministic models, the conservation of probability is governed by the more general Chapman-Kolmogorov equation

$$P(t+1, \mathcal{S}\boldsymbol{n'}(.)) = \sum_{\boldsymbol{n}(.)\in\Gamma} \prod_{\boldsymbol{r}} A(\boldsymbol{n}(\boldsymbol{r}) \to \boldsymbol{n'}(\boldsymbol{r})) P(t, \boldsymbol{n}(.)), \tag{3.5}$$

that reduces to Liouville equations in deterministic case, where $A(\boldsymbol{n(r)} \to \boldsymbol{n'(r)})$ selects unique configuration.

3.6 Mean occupation numbers

Motivated by the ensemble formalism of statistical physics, we define mean occupation numbers

$$N_i = \langle n_i \rangle = \sum_{n_i(.)\in\Gamma} n_i(.) P(t, n_i(.))$$

that naturally implies formula for mean mass density

$$\rho(t, r) = \sum_i N_i(t, \boldsymbol{r}) \tag{3.6}$$

and the momentum density

$$j(t, r) = \sum_i c_i N_i(t, \boldsymbol{r}). \tag{3.7}$$

Due to conservation of probabilities, the conservation laws 3.4 implies conservation of the mean populations

$$\sum_i N_i(t+1, \boldsymbol{r} + \boldsymbol{c_i}) = \sum_i N_i(t, \boldsymbol{r}), \tag{3.8}$$

$$\sum_i c_i N_i(t+1, \boldsymbol{r} + \boldsymbol{c_i}) = \sum_i c_i N_i(t, \boldsymbol{r}). \tag{3.9}$$

3.7 Equilibrium occupation numbers

We are interested in the equilibrium solutions of the Chapman-Kolmogorov equation 3.5. Because collisions on the lattice are purely local, we are looking for the solution of the form

$$P(\boldsymbol{n}(.)) = \prod_{\boldsymbol{r}\in\mathcal{L}} p(\boldsymbol{n}(\boldsymbol{r})). \tag{3.10}$$

Maximization of the entropy suggest $p(n(.))$ of the form

$$p(\boldsymbol{n}(\boldsymbol{r})) = \prod_i N_i^{n_i} (1 - N_i)^{(1-n_i)}. \tag{3.11}$$

where $N_i^{n_i}(1 - N_i)^{(1-n_i)}$ is probability that mean occupation number is N_i.

Substituting the formula 3.10 and 3.11 into Chapman-Kolmogorov equation 3.5 gives

$$N_i^{n_i'}(1 - N_i)^{(1-n_i')} = \sum_{\boldsymbol{n}\in\Gamma} \prod_{\boldsymbol{r}} A(\boldsymbol{n} \to \boldsymbol{n'}) N_i^{n_i} (1 - N_i)^{(1-n_i)}, \tag{3.12}$$

for each state $\boldsymbol{n}$ of 64 possible states (or 2^b for automaton with b lattice vectors). So in fact the last equation is a set of 64 equations.

It is a non-trivial fact, that this set has a solution, and the following lemma states, that it is given by Fermi-Dirac distribution. The theorem is stated for D dimensional cellular automaton with b lattice vectors c_i, $i = 1, 2...b$.

Theorem 1: The following statements are equivalent:

1. N_i^{eq} are solutions of 3.12.

2. N_i^{eq} are solutions of the set of b equations:

$$\Delta_i(N) = \sum_{nn'}(n'_i - n_i)A(n \to n') \prod_j N_j^{n_j}(1 - N_j)^{1-n_j} \tag{3.13}$$

3. N_i^{eq} are given by Fermi-Dirac distribution

$$N_i^{eq} = \frac{1}{1 + \exp(h + \boldsymbol{q}.\boldsymbol{c_i})}, \tag{3.14}$$

where h is real number and $\mathbf{q}$ is D-dimensional vector.

The proof of this theorem can be found in the Appendix C of [5]. Further, non-deterministic collisions and semi-detailed balance 3.1 lead to the *Universality theorem*, stating that these automata admit universal equilibrium solutions, completly factorized over all nodes and cells, with mean populations N_i given by the Fermi-Dirac distribution, dependent only on the density ρ and mass current $j = \rho\boldsymbol{u}$ and independent of the transition probabilities $A(\boldsymbol{n} \to \boldsymbol{n'})$ [5].

The idea of the proof is as follows:

Unknown parameters h and $\boldsymbol{q}$ can be calculated in terms of mass density and velocity by inversion of the relations

$$\rho = \sum_i N_i = \sum_i \frac{1}{1 + exp(h + \boldsymbol{q}.\boldsymbol{c_i})} \tag{3.15}$$

$$\boldsymbol{u} = \sum_i \boldsymbol{c_i}\boldsymbol{N_i} = \sum_i \frac{\boldsymbol{c_i}}{1 + exp(h + \boldsymbol{q}.\boldsymbol{c_i})} \tag{3.16}$$

For FHP, explicit solutions are available only in few special cases [5]. It is not surprising to obtain Fermi-Dirac distribution in equilibrium, due to the exclusion principle (cells are empty or occupied by one particle).

To proceed, we will use perturbation of equilibrium solutions N_i^{eq} in powers of $\boldsymbol{u}$ for small Mach numbers ($\boldsymbol{u}/c_{\text{sound}}$). Up to the second order[1], the expansion reads [5]

$$N_i^{eq}(\rho, \boldsymbol{u}) = \frac{\rho}{b} + \frac{D\rho}{c^2 b}\boldsymbol{c_i}.\boldsymbol{u} = \rho G(\rho) Q_{i\alpha\beta} u_\alpha u_\beta + O(u^3) \tag{3.17}$$

where

$$Q_{i\alpha\beta} = c_{i\alpha}c_{i\beta} - \frac{c^2}{D}\delta_{\alpha\beta} \tag{3.18}$$

and

$$G(\rho) = \frac{D^2}{2c^4 b}\frac{b - 2\rho}{b - \rho}. \tag{3.19}$$

We see that for $\rho = b/2$, the coefficient $G(\rho)$ vanishes.

[1]The expansio to the second order is required, because the advection in Navier-Stokes equations will emerge from the quadratic term

3.8 Chapman-Enskog expansion

Because relaxation towards equilibrium values happens in a few updates of the automaton, it is standard procedure to expand the occupation numbers $N_i(t, \boldsymbol{r})$ around the equilibrium occupation numbers $N_i^{eq}(\rho, \boldsymbol{u})$:

$$N_i(t, \boldsymbol{r}) = N_i^0(t, \boldsymbol{r}) + \epsilon N_i^1(t, \boldsymbol{r}) + \mathcal{O}(\epsilon^2) \tag{3.20}$$

The equations of mass and momentum conservation 3.8 and 3.9 can be equivalently stated in the form

$$\sum_i N_i(t+1, r+c_i) - N_i(t, r) = 0, \tag{3.21}$$

$$\sum_i c_i(N_i(t+1, r+c_i) - N_i(t, r)) = 0, \tag{3.22}$$

that is more suitable for our purpose.

Expansion of $N_i(t+1, r+c_i)$ around $N_i(t, r)$ leads to

$$\begin{aligned} N_i(t+1, r+c_i) = N_i(t, r) + \partial_t N_i(t, r) + c_{i\alpha}\partial_\alpha N_i(t, r) \\ +\frac{1}{2}\partial_t\partial_t N_i(t, r) + \frac{1}{2}c_{i\alpha}c_{i\beta}\partial_\alpha\partial_\beta N_i(t, r) + c_{i\alpha}\partial_t\partial_\alpha N_i(t, r) + \mathcal{O}(\partial^3). \end{aligned} \tag{3.23}$$

The expression above is the mixture of various physical phenomena – diffusion, advection, propagation of the sound waves or relaxation towards local equilibria. Each phenomena has its typical spatial and temporal scale, that corresponds to different powers of ϵ, see the tables below.

TEMPORAL SCALES		
Scale	**Rescaling of time**	**Phenomena**
1 step	t	Relaxation towards local equilibrium
100 steps	$t_1 = \frac{1}{100}t = \epsilon t$	advection, sound waves (perturbation of mass and densi
10 000 steps	$t_2 = \frac{1}{10000}t = \epsilon^2 t$	diffusion

SPATIAL SCALES		
Scale	**Rescaling of length**	**Phenomena**
1 lattice unit	**r**	Relaxation towards local equilibrium
100 lattice units	$\boldsymbol{r_1} = \frac{1}{100}\boldsymbol{r} = \epsilon\boldsymbol{r}$	diffusion, advection, sound waves

To derive the hydrodynamical equation, we will exploit the multi-scale technique. It starts by grouping-up the terms of hydrodynamical temporal and spatial scales.

Using the rescaled length and time, we deduce the rescaled differential operators

$$\begin{aligned} \partial_t &= \epsilon\partial_t^{(1)} + \epsilon^2\partial_t^{(2)} \\ \partial_\alpha &= \epsilon\partial_\alpha^{(1)} \end{aligned} \tag{3.24}$$

Now, we are ready to expand the conservation laws 3.21 and 3.22 in Chapman-Enskog series. We insert expansion 3.23 of $N_i(t+1, \boldsymbol{r}+\boldsymbol{c_i})$, then we insert expansion of $N_i(t, \boldsymbol{r})$ according to 3.20. Finally, we substitute differential operators according to 3.24.

From the expansion, we present only the terms of order ϵ, that we are interested in.

From the conservation of mass 3.21 we get

$$\partial_t^{(1)} \sum_i N_i^{(0)} + \partial_\beta \sum_i c_{i\beta} N_i^{(0)} = 0, \tag{3.25}$$

and from the conservation of momentum 3.22 we get

$$\partial_t^{(1)} \sum_i c_{i\alpha} N_i^{(0)} + \partial_\beta \sum_i c_{i\alpha} c_{i\beta} N_i^{(0)} = 0. \tag{3.26}$$

Using definition of mass density, and the following definition of the *momentum flux tensor*

$$P_{\alpha\beta}^{(0)} := \sum_i c_{i\alpha} c_{i\beta} N_i^{(0)} = \sum_i c_{i\alpha} c_{i\beta} N_i^{eq}(\rho, \boldsymbol{u}). \tag{3.27}$$

we can write the conservation laws in the shorter form

$$\begin{aligned} \partial_t^{(1)} \rho + \nabla^{(1)}(\rho \boldsymbol{u}) &= 0, \\ \partial_t^{(1)}(\rho u_\alpha) + \nabla^{(1)} P_{\alpha\beta}^{(0)} &= 0 \end{aligned} \tag{3.28}$$

In the first equation, we can recognize the familiar continuity equation, but we have some work to do with the second one.

Substituting 3.17 for N_i^{eq} into momentum flux tensor, its components read (after some algebraic simplification)

$$\begin{aligned} P_{xx}^{(0)} &= \frac{\rho}{2} g(\rho)(u_x^2 - u_y^2) + \frac{\rho}{2}, \\ P_{yy}^{(0)} &= \frac{\rho}{2} g(\rho)(u_x^2 - u_y^2) + \frac{\rho}{2}, \\ P_{xy}^{(0)} &= P_{yx}^{(0)} = \rho g(\rho) u_x u_y, \end{aligned} \tag{3.29}$$

where we defined $g(\rho) = \frac{3-\rho}{6-\rho}$.

Unfortunately, it does not match the momentum flux tensor of Navier-Stokes equation

$$\begin{aligned} P_{xx} &= \rho\, u_x^2 + p \\ P_{yy} &= \rho\, u_y^2 + p \\ P_{xy} = P_{yx} &= \rho u_x u_y \end{aligned} \tag{3.30}$$

Let us examine why are these tensors different.

For low values of $\boldsymbol{u}^2$, pressure p is given by isothermal relation $p = \frac{\rho}{2} = \rho c_s^2$, where $c_s = \frac{1}{\sqrt{2}}$ is the speed of sound[2].

What about $g(\rho)$?

This is the disease of FHP, FCHC and all the lattice-gas cellular automata to follow, that $g(\rho)$ is never equal to 1, as it is in Navier-Stokes equations.

The reason lies in the broken *Galilei invariance* in these automata as they all have discrete rotational symmetry (by 60° in FHP and between 60° and 45° in FCHC).

Symptomatic treatment, such as rescaling of time

$$t \to \frac{t}{g(\rho)}, \tag{3.31}$$

does not solve all the associated problems (D'Humier et al. 1987). This flaw requires fundamental treatment, and leads to the new generation of automata that we will present in the final chapter on LGCA theory.

However, using this rescaled time, and setting density to be constant ($\rho = \rho_0$) for all terms except the pressure, we get the FHP version of incompressible Euler equation

$$\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u}\boldsymbol{\nabla})\boldsymbol{u} = -\boldsymbol{\nabla} P, \tag{3.32}$$

but the pressure P is still the function of density and velocity

$$P = (\frac{\rho}{2\rho_0 g(\rho_0)} - \boldsymbol{u}^2). \tag{3.33}$$

This is as far as we can get in the first approximation. The derivation of the Navier-Stokes equations

$$\begin{aligned} \boldsymbol{\nabla}.\boldsymbol{u} &= 0, \\ \partial_t \boldsymbol{u} + (\boldsymbol{u}\boldsymbol{\nabla})\boldsymbol{u} &= -\boldsymbol{\nabla} P + \nu \nabla^2 \boldsymbol{u} \end{aligned} \tag{3.34}$$

would require terms of the order ϵ^2 from the Chapman-Enskog expansion to add in the equations 3.21 and 3.22, as shown in [5].

4. FCHC

In two dimensions, we are lucky to find hexagon – a symmetric polygon that covers whole 2D plane and possesses sufficient rotational symmetry to make a good lattice for FHP.

In three dimensions, we have five symmetric candidates for LGCA, so–called Plato solids, but none of them works. Dodecahedron and icosahedron might have sufficient rotational symmetry (interestingly, they share the group of symmetry) but they do not cover whole 3D space uniformly. Only the cube does but LGCA build on a rectangular lattice would suffer from the same diseases as HPP – insufficient rotational symmetry and insufficient degrees of freedom in the nodes.

Short analysis [2] shows that in 5 and more dimensions, we have only three symmetric solids – simplex, hypercube and its dual solid. None of them fits.

Fortunately, in four dimensions, we have three extra symmetric solids and one of them actually works.

4.1 Face-centered hypercube

Face-centered hypercube (we will use the shortcut FCHC) is the suitable solid for LGCA in four dimensions. It can be defined by its 24 vertices with the Cartesian coordinates

$$\begin{array}{cccc}
(\pm 1, & \pm 1, & 0, & 0), \\
(\pm 1, & 0, & \pm 1, & 0), \\
(\pm 1, & 0, & 0, & \pm 1), \\
(0, & \pm 1, & \pm 1, & 0), \\
(0, & \pm 1, & 0, & \pm 1), \\
(0, & 0, & \pm 1, & \pm 1).
\end{array}$$

These 24 vertices correspond to 24 lattice vectors in four dimensions but if we project them into three dimensions (by deleting the fourth coordinate q_4) we get only 18 lattice vectors

$$\begin{array}{ccc}
(\pm 1, & \pm 1, & 0), \\
(\pm 1, & 0, & \pm 1), \\
(\pm 1, & 0, & 0), \\
(\ 0, & \pm 1, & \pm 1), \\
(\ 0, & \pm 1, & 0), \\
(\ 0, & 0, & \pm 1).
\end{array}$$

For $q_4 = 0$ we have 12 lattice vectors (red lines on the figure 4.1), each vector corresponds to the single cell only, so at most one particle propagates along each.

For both $q_4 = -1$ and $q_4 = 1$ we obtain same six lattice vectors (blue lines on the figure 4.1). Each vector corresponds to the two cells, hence two particles can propagate along each vector.

In the end, the $3D$ projection of the FCHC lattice leads to the rectangular grid, so the position of the nodes can be denoted by 3 Cartesian integer coordinates.

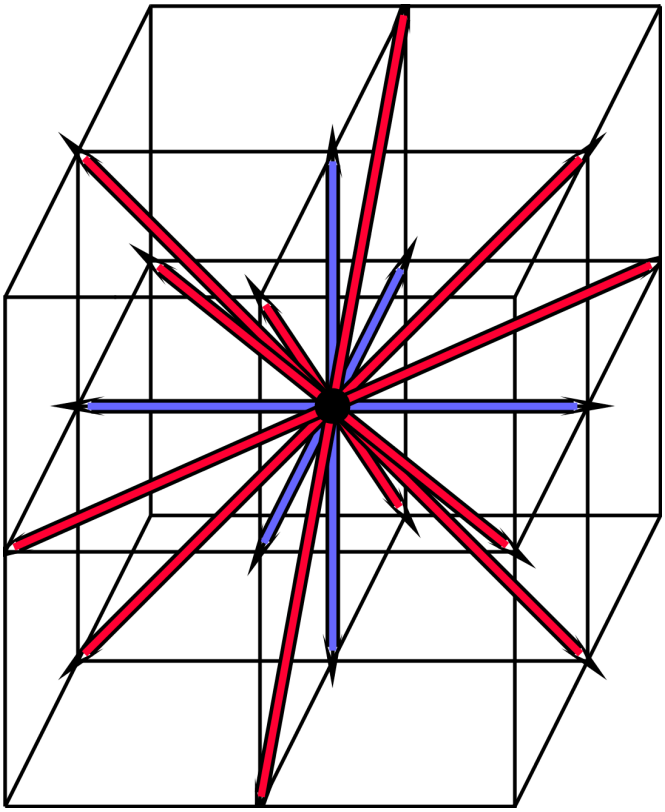


Figure 4.1: Projection of the lattice vectors into 3D. Red arrows are projections of vectors with $q_4 = 0$, blue arrows with $q_4 = \pm 1$.

Each node is connected to 18 neighbors by the lattice vectors of Fig. 4.1, and 24 particles can propagate along them in the single step.

4.2 Collision rules for FCHC

The FCHC lattice was proposed already by Frisch et. al. in 1986. However, they did not propose any recipe for collision. It is easy to understand why it is not a simple task.

There are 24 cells in each node, so the node can acquire $2^{24} = 16\ 777\ 216$ different states. It was easy to specify the collision rule for FHP in one simple table, but how do you specify rules for millions of states?

In the subsequent section we will show how Henon answered to this challenge in his famous article [8].

4.2.1 Necessary conditions

Let us review what are the necessary conditions that collision rules must fulfill:

1. number of particles is preserved;
2. total momentum is preserved;
3. no other quantity is preserved;

4. exclusion principle – no two particles occupy one cell in the new state;

5. collision rules are invariant with respect to rotations and reflections of the node;

6. collisions satisfy the semi-detailed balance.

Henon proposed his own conditions on the collisions and showed that all conditions above are automatically fulfilled.

1. All collisions are isometries of FCHC – such transformations that preserve set of 24 lattice vectors above. It can be shown that this condition actually defines FHP collision set. For FCHC, this condition guarantees that conditions will be simple and leads us to the simple collision algorithm.

2. The choice of isometry is restricted by the momentum conservation only. There are 7009 possible values of momentum. But considering that collision rules are invariant under isometries of FCHC, and due to first restriction that collisions are isometries, we can consider 37 momenta only.

3. We want shear viscosity to be as low as possible, so that we can go to higher Reynolds numbers in our simulations. Suppressing the viscosity means is achieved by shortening the mean free path of the particles. In other words, we wish to change as many bits in the collision as possible. We will call such collisions "optimal isometries".

4. The isometry is randomly chosen among optimal isometries.

Having discussed the role of the isometries, let us define them properly.

4.3 Isometries of FCHC

By G we denote the group of isometries that preserve all 24 lattice vectors (or equivalently all vertexes of FCHC). Any isometry can be represented by 4×4 matrix

$$M = \begin{bmatrix} a_{11} & \dots & a_{14} \\ \dots & & \dots \\ a_{41} & \dots & a_{44} \end{bmatrix}. \tag{4.1}$$

Although the group G is of order 1152, it can be generated by five elements only, but it will be more comfortable to employ the generating set consisting of 12 elements to be described in the following section.

4.3.1 Generating set

Isometries $S_\alpha,\ \alpha = 1, 2, 3, 4$, are reflections over(?) plane $x_\alpha = 0$. For example, S_1 is represented by the matrix

$$S_1 = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

that flips the sign of the first momentum coordinate q_1.

Another six isometries $P_{\alpha\beta}$ are reflections over plane $x_\alpha = x_\beta$. For example, P_{12} can be represented by matrix

$$P_{12} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4.3}$$

that swaps the first and second coordinates of the momentum.

Another isometry Σ_1 is reflection over plane $x_1 + x_4 = x_2 + x_3$ represented by matrix

$$\Sigma_1 = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & -1 \\ 1 & 1 & -1 & 1 \\ 1 & -1 & 1 & 1 \\ -1 & 1 & 1 & 1 \end{bmatrix}. \tag{4.4}$$

The last generating isometry Σ_2 is reflection over plain $x_1 = x_2 + x_3 + x_4$. In matrix representation it reads

$$\Sigma_2 = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}. \tag{4.5}$$

Any isometry of FCHC can be composed from these 12 isometries, so it can be uniquely expressed by

$$M = \begin{pmatrix} I \\ S_4 \end{pmatrix} \begin{pmatrix} I \\ S_3 \end{pmatrix} \begin{pmatrix} I \\ S_2 \end{pmatrix} \begin{pmatrix} I \\ S_1 \end{pmatrix} \begin{pmatrix} I \\ P_{34} \end{pmatrix} \begin{pmatrix} I \\ P_{23} \\ P_{24} \end{pmatrix} \begin{pmatrix} I \\ P_{12} \\ P_{13} \\ P_{14} \end{pmatrix} \begin{pmatrix} I \\ \Sigma_1 \\ \Sigma_2 \end{pmatrix} \tag{4.6}$$

where the brackets mean "one of the isometries inside the bracket".

4.3.2 Normalized momenta

Normalized momenta are such momenta that their components fulfill the following two conditions:

1. $q_1 \geq q_2 \geq q_3 \geq q_4 \geq 0$,
2. $q_0 = 0$ or $q_1 + g_4 = q_2 + q_3$.

Henon specified collision rules for normalized momenta only. There is 37 normalized momenta (out of 7009 total) so he significantly simplified the job.

If the state has normalized momentum (i.e. fulfilling 1), we just look into table and pick any isometry that is offered for that momentum. However, if the state doesn't have normalized momentum,

1. we normalize it by applying an appropriate isometry on the state. Let us denote this isometry by Γ. In fact, this isometry is equivalent to the change of the coordinate system.

2. Now that we have state with normalized momenta, search in the table and pick an optimal isometry.

3. We go back to the previous coordinate system by applying isometry Γ^{-1}.

4.3.3 Optimal isometries

However, in the step 2, we do not want to use all the isometries that conserve the momentum. Optimally, we would like to change as many bits as possible (that minimizes mean free path of particles, and that leads to desired low viscosity).

Although no further reduction beyond 37 normalized momenta is possible, some of these momenta share same optimal isometries, that lead to minimal viscosity and preserve momentum.

Based on that, momenta can be divided into 12 classes, as in the table below. First column specifies momentum, the second column specifies isometries that can be applied to this momentum. Before the simulations starts, we calculate optimal states for each of the 2^{24} initial states. Then, the collision step reduces to searching in this huge table and randomly choosing from the available states.

In the early nineties when this algorithm was proposed and tested, the look-up table was considered too big, and several optimizations were proposed and implemented in CRAY-Asssebler [2]. Despite that, the achieved update rate was lower then of the Pair-iteraction model, that we are presenting in the next chapter.

Normalized momenta	Optimal isometries
$q_1 = q_2 > q_3 > q_4 > 0$	P_{12}
$q_1 = q_2 = q_3 > q_4 > 0$	$P_{23}P_{12}$, $P_{23}P_{13}$
$q_1 > q_2 > q_3 > q_4 = 0$, $q_1 = q_2 + q_3$	$S_4\Sigma_1$, $S_4\Sigma_2$
$q_1 > q_2 > q_3 > q_4 = 0$, $q_1 \neq q_2 + q_3$	S_4
$q_1 = q_2 > q_3 > q_4 = 0$	S_4P_{12}
$q_1 > q_2 = q_3 > q_4 = 0$	$S_4\Sigma_1$, $S_4\Sigma_2$, $S_4P_{23}\Sigma_1, S_4P_{23}\Sigma_2$
$q_1 > q_2 = q_3 > q_4 = 0$, $q_q = 2q_2$	S_4P_{23}
$q_1 > q_2 = q_3 > q_4 = 0$, $q_1 \neq 2q_2$	$P_{23}P_{12}$, $P_{23}P_{13}$, $S_4P_{23}P_{12}$, $S_4P_{23}P_{12}$
$q_1 = q_2 = q_3 > q_4 = 0$	S_4S_3, S_3P_{34}, S_4P_{34}
$q_1 > q_2 = q_3 = q_4 = 0$	$S_3P_{34}P_{12}$, $S_4P_{34}P_{12}$, $S_4S_3\Sigma_1$, $S_4S_3P_{34}P_{12}\Sigma_1$, $S_4S_3\Sigma_2$, $P_{34}P_{12}\Sigma_2$
$q_1 > q_2 = q_3 = q_4 = 0$	$S_4S_2P_{23}$, $S_4S_3P_{23}$, $S_3S_2P_{24}$, $S_4S_3P_{24}$, $S_3S_2P_{34}$, $S_4S_2P_{34}$
$q_1 = q_2 = q_3 = q_4 = 0$	$S_3S_1P_{34}P_{12}$, $S_4S_1P_{34}P_{12}$, $S_3S_2P_{34}P_{12}$, $S_4S_2P_{34}P_{12}$, $S_2S_1P_{24}P_{13}$, $S_4S_1P_{13}P_{13}$, $S_3S_2P_{24}P_{13}$, $S_4S_3P_{24}P_{13}$, $S_2S_1P_{23}P_{14}$, $S_3S_1P_{23}P_{14}$, $S_4S_2P_{23}P_{14}$, $S_4S_3P_{23}P_{14}$

5. Pair Interaction LGCA

Pair-interaction automata (PI) constitute another branch that successfully evolved from HPP model, somewhat in contrast to FHP. To put it in a nutshell, FHP switched from the rectangular grid to hexagonal, and thus obtained better rotational symmetry, increased the degree of freedom of the nodes and added new collision states.

The Pair-interaction model preserves the HPP rectangular lattice, but the degrees of freedom in the node are incremented by the artificial definition of momentum. Momenta has no longer same direction as the lattice velocities, but they constitute an inner degree of freedom of the particle, that might change in the collision, independently of the lattice velocity. Although the differentiation in momentum and velocity might seem odd at the first glance, we will show that it leads to the physically plausible microdynamics and offers us very efficient algorithm for implementation.

We will explore theory of Pair Interaction automaton in arbitrary dimension later on, but to get good understanding of the pair-interactions, we start with its 2D version, so we can explain the theory graphically.

5.1 User-friendly guide to Pair interactions

5.1.1 Pair-interactions automaton in two dimensions

Geometry of the lattice is same as for HPP model. It consists of nodes arranged in a rectangular grid.

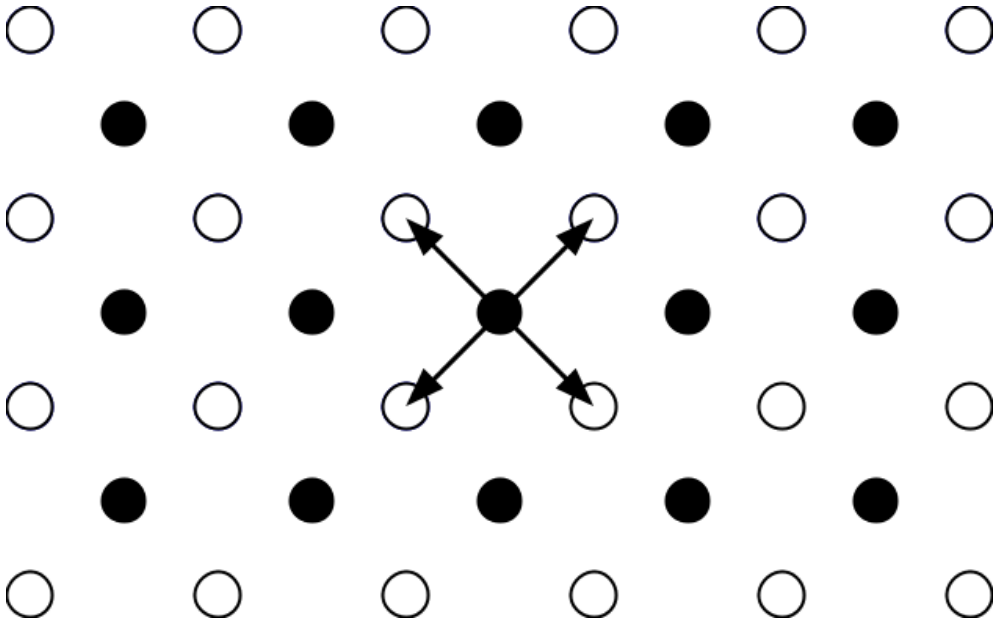


Figure 5.1: Lattice of 2D Pair Interaction automaton

Let us assume that in the time-step t only the black nodes are occupied (e.g. with odd Cartesian idices). Then in the next step $t + 1$, only the white nodes (with even indices) are occupied. So the set of occupied nodes alternate

with every time step, which offers very efficient implementation, because we can perform propagation on the single data-structure. What is more important, Pair-interaction model is free of Zanetti's staggered invariants ([2], page 132) due to this alternating lattice.

5.1.2 Node in detail

Let us look at the node in a more detail.

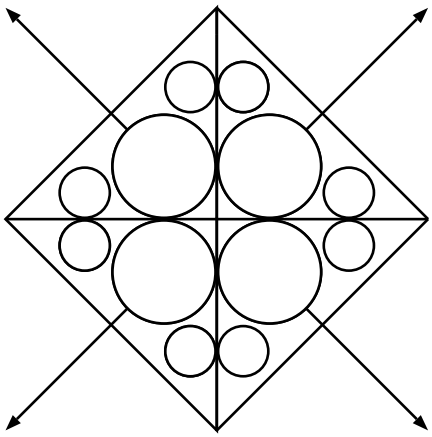


Figure 5.2: An empty node

It consists of 4 cells that are represented by 3 bits: one mass bit (a big circle) and two momentum bits (small circles).

The node on the picture above is empty (all bits are set to zero).

Here we have another example of a node:

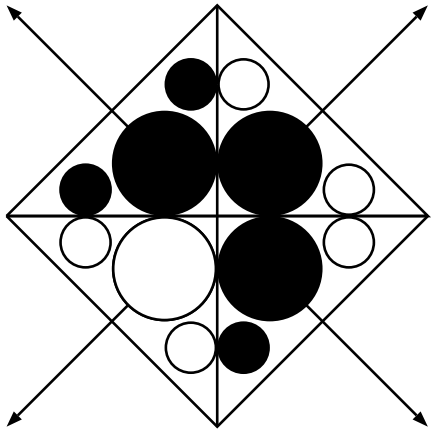


Figure 5.3: State of a node before collision

In this node, there are particles in the three cells (mass bit is set to 1).

Particle in the lower-left cell is standing (both momentum bits are 0).

Particle in the upper-left cell has momentum in direction [1,1] and particle in the lower-right cell has momentum in direction [0,1].

5.1.3 Update rules

Following rules are true of every lattice-gas cellular automaton.

1. Lattice is changing in discrete time steps
2. Update rules are **local**. It means that state of the cell in the next step is determined by the current state of the cell itself and its four neighbors. Hence we can resolve update of the lattice node by node, which can be carry out parallely.
3. If this cellular automaton really simulates fluid dynamics and is physically realistic, update rule needs to conserve **mass, momentum and angular momentum**.

Update of the lattice is accomplished in two distinct steps – collision and propagation.

5.1.4 Collision

Collision changes configuration of the single nodes respecting two constraints – number of particles and total momentum in the node is conserved. In this model, it is performed by the pair-interaction algorithm:

1. The cells are paired in X direction:

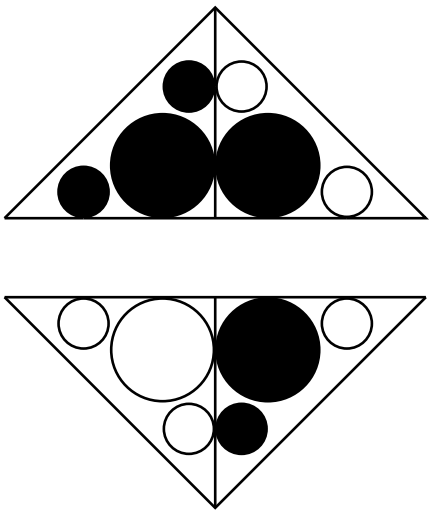


Figure 5.4: Pairs in X-direction

2. Then we swap the bits in the pairs so that total momentum in the pair is preserved. Therefore, node in 5.1.2 changes to

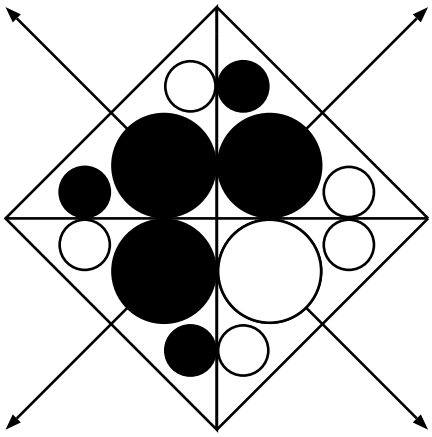


Figure 5.5: State of the node after pair-interaction in X direction

3. Then, the cells are paired in Y direction

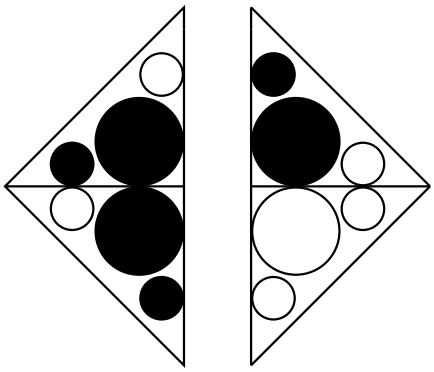


Figure 5.6: Pairs in Y-direction

4. We change the configuration in these pairs preserving mass and momentum in the node. Hence, the node in 2 changes into:

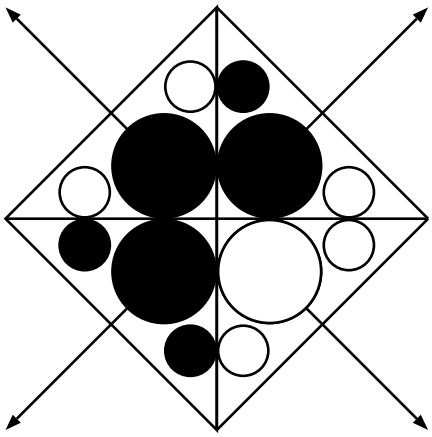


Figure 5.7: State of the node after pair-interaction in Y-direction

In total, there is only 17 possible configurations of the pairs, so the pair-interaction can be resolved by looking in the figure 5.1.4. This collision set is same for Pair-interaction automata of any dimension, which is the great advantage comparing to other LGCA methods.

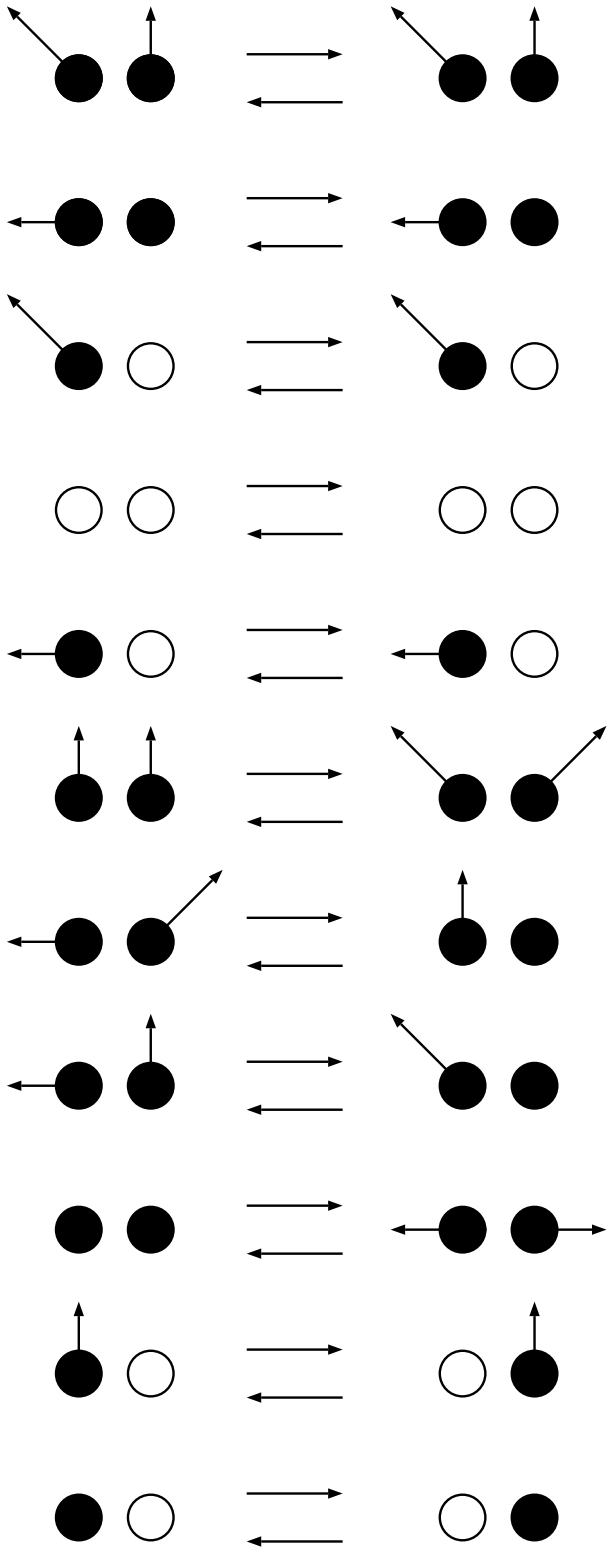


Figure 5.8: All admissible pair-interactions

5.2 Propagation:

Same as for HPP, particles propagates from a node along 4 lattice vectors to the neighboring nodes. In case of Pair-interaction, it carries its mass and momentum bits to the new node.

5.3 Pair-interaction cellular automaton in arbitrary dimension D

COMPARISON OF **2** AND ***D*** DIMENSIONAL PAIR INTERACTION AUTOMATON		
dimension of PI	**2**	***D***
nodes	4 cells arranged into square	2^D cells arranged in hypercube
cells	1 mass bit, 2 momentum bits	1 mass bit, D momentum bits
neighbors	node has 4 neighbors	node has 2^D neighbors

Generalization of this automaton to the arbitrary dimension D is very straightforward, if we chose a proper formalism. Instead of notation that Nasilowski used in his founding article [12], we will stick to the formalism that we already established for FHP model.

State of the node will be denoted by $\boldsymbol{n}(t, \boldsymbol{r})$, where $\boldsymbol{r}$ is its position on the lattice and t is the current time step. $\boldsymbol{n}(t, .)$ represents the whole lattice at time t. When possible, we will skip the time variable and use only $\boldsymbol{n}(.)$.

Each node consists of 2^D cells

$$\boldsymbol{n} = (n_1, n_2, ..., n_{2^D}) \tag{5.1}$$

or $2^D(D+1)$ bits

$$\boldsymbol{n} = \left\{ n_{i\alpha} \in \left\{0, 1\right\},\ i = 1..2^D,\ \alpha = 0, 1, ...D \right\} \tag{5.2}$$

where i is index of the cell, index $\alpha = 0$ stands for the mass bit, and $\alpha = 1, 2, ..D$ stands for the momenta bits of the cell.

5.4 Update

The evolution of the lattice happens in discrete time steps $\mathcal{U}$

$$\mathcal{U}\boldsymbol{n}(t, .) = \boldsymbol{n}(t+1, .) \tag{5.3}$$

Update operator $\mathcal{U}$ is the composition of two operators, collision $\mathcal{C}$ and propagation $\mathcal{P}$

$$\mathcal{U} = \mathcal{P} \circ \mathcal{C}. \tag{5.4}$$

5.5 Collision

The purpose of collision is to change the state of the node $\boldsymbol{n}$ into new state $\boldsymbol{n'}$ such that 2 conditions are fulfilled:

1. **Collision operator is one-to-one**

   If $\mathcal{C} : \boldsymbol{n} \to \boldsymbol{n'}$, then $\mathcal{C} : \boldsymbol{n'} \to \boldsymbol{n}$.

   This reversibility condition allows us to apply Gibbs formalism later on.

2. **Collision preserves mass and momentum in the nodes**

   Mass and momentum are given by

$$m_0 = \sum_i n_{i0} \tag{5.5}$$

$$p_j = \sum_i c_{ij} n_{ij}, \ j = 1, 2, ...D \tag{5.6}$$

   Using these definitions, conservation of mass and momentum can be expressed by

$$\begin{aligned} m_o = m'_o \quad &\Leftrightarrow \quad \sum_i n_{i0} = \sum_i n'_{i0}, \\ p_j = p'_j \quad &\Leftrightarrow \quad \sum_i c_{ij} n_{ij} = \sum_i c_{ij} n'_{ij}, \ j = 1, 2, ...D \end{aligned} \tag{5.7}$$

The simplest collision rule fulfilling these conditions would be identity

$$\mathcal{I} : \boldsymbol{n} \to \boldsymbol{n} \tag{5.8}$$

but this "non-collision" would lead to undesired, non-physical invariants. On the contrary, we want to change as many bits in the node as possible, to reduce the mean free path of the particles. Physically, it minimizes the shear viscosity and allow us to reach higher Reynolds numbers in the simulations.

5.5.1 Collision algorithm

Collision rule that fulfills all the requirements above turned out to be really simple. It was explained graphically in the previous section, now we state it more formally for an arbitrary dimension D. The algorithm consists of D steps. In each step ($d = 1...D$), we create pairs of the cells in d^{th} direction, such that $c_{id} = -c_{jd}$ and $c_{i\alpha} = c_{j\alpha}$ for $\alpha \neq d$. If $n_{id} = n_{jd}$ (it means that in direction d, momentum of the pair is zero), we swap the states of the cells n_i and n_j without changing the total momentum of the pairs. Because all the pair-interactions conserved the momentum, the total momentum in the node is conserved by the collision.

5.6 Equilibrium statistics

We start by revoking the basic equation of the microdynamics

$$\boldsymbol{n}(t + 2, .) = \mathcal{U}^2 \boldsymbol{n}(t, .) \tag{5.9}$$

We remind that $n(.)$ represents the whole lattice, while $n(\boldsymbol{r})$ represents the state of the node at $\boldsymbol{r}$.

As we already mentioned, the occupied parts on the lattice alternate every time-step, so we can compare only states over the two time steps.

The microdynamical equation 5.9 implies conservation of probabilities

$$P(t, \boldsymbol{n}(.)) = P(t+2, \mathcal{U}^2 \boldsymbol{n}(.)) \tag{5.10}$$

where $P(t, n(.))$ is the probability that lattice is found in the state $n(.)$ from the statistical ensable of automata.

5.6.1 Gibbs distribution

Before we proceed, it is handy to define the **hypermomentum** m_α, where m_0 is the mass, and $(m_1, m_2, ..., m_D)$ is the momentum.

From now on, the Greek index α will run trough the $\{0, 1, 2, ..., D\}$ and will signalize that hypermomentum is used, while Latin j runs over $\{1, 2, ...D\}$ and signalizes the standard momentum.

The Gibbs distribution is defined for this model as

$$P^E(n(.)) = \frac{W(n(.))}{Z} \tag{5.11}$$

where

$$W(n(.)) = \exp\left(-\sum_{\alpha=0}^{D} \mu_\alpha m_\alpha(.) \right), \tag{5.12}$$

Z is the partition function

$$Z = \sum_{n(.))} W(n(.)). \tag{5.13}$$

and the μ_α are the intensive parameters conjugated to the hypermomentum m_α.

Let us inspect if the Gibbs distribution fulfills the conservation of probabilities required by 5.10.

The total momentum of the whole lattice $\mathcal{L}$ is given by

$$m_\alpha(.) = \sum_{\boldsymbol{r} \in \mathcal{L}} c_{i\alpha} n_{i\alpha}(\boldsymbol{r}), \ \alpha = 0, 1, ..D. \tag{5.14}$$

Conservation of local momenta 5.7 implies conservation of the total momentum

$$m_\alpha(.) = m'_\alpha(.), \ \alpha = 0, 1, ..D \tag{5.15}$$

that immediately implies conservation of the Gibbs probability

$$P^E(t+2, \mathcal{E}^2 n(.)) = P^E(t, n(.)). \tag{5.16}$$

. So the $P^E(t, (n.))$ is the stationary solution of 5.10, which means it is the equilibrium distribution for our model. However, the Gibbs distribution represents only one class of the stationary solutions. If there were additional extensive dynamical invariants independent of momenta and mass (e.g. Zanetti's invariants

for FHP and FCHC model), there would arise conjugated intensive parameters, corresponding to other equilibrium distributions. However, due to the alternating lattice, this model is free of Zanetti's invariants.

The additivity of the momentum cell by cell on the lattice (equation 5.14) implies

$$P^E(n(.)) = \prod_{i,\boldsymbol{r}\in\mathcal{L}} P^E(n_i(\boldsymbol{r})) \tag{5.17}$$

which means that there is no statistical correlation between the cells. Again,

$$P^E(n_i(\boldsymbol{r})) = \frac{W(n_i(\boldsymbol{r}))}{Z}, \tag{5.18}$$

with

$$W(n_i(\boldsymbol{r})) = exp(-\sum_{\alpha=0}^{D} \mu_\alpha c_{i\alpha} n_{i\alpha}(\boldsymbol{r})) = \prod_{\alpha=0}^{D} exp(-\mu_\alpha c_{i\alpha} n_{i\alpha}(\boldsymbol{r})) = \prod_{\alpha=0}^{D} W(n_{i\alpha}(\boldsymbol{r})) \tag{5.19}$$

and

$$Z_i = \sum_{n_i} W(n_i(\boldsymbol{r})).$$

The probability that the cell is occupied by a particle is

$$\rho_i \equiv \langle n_{i0} \rangle = \frac{1}{Z_i}$$

and the probability that the particle has non-zero i^{th} component of the momentum

$$\sigma_{ij} \equiv \langle n_{ij} | n_{i0} = 1 \rangle = \frac{W(n_{ij}}{Z_i}.$$

Rather technical solution of the two equations above shown in the Appendix B of [12] leads to

$$\rho_i := f\big(\mu_0 + \sum_{j=1}^{d} lnf(-\mu_j v_j)\big), \quad \sigma_{ij} = f(\mu_j v_j), \tag{5.20}$$

where

$$f(\lambda) := \frac{1}{1+e^\lambda}.$$

Because the lattice vectors are normalized and the single node contains 2^D lattice vectors, the volume of the node is

$$V = 2^D.$$

Finally, we can define the mass density

$$\rho = 2^{-D} \sum_i \rho_i \tag{5.21}$$

and the momentum density

$$q_j := 2^{-D} \sum_i c_{ij} \langle n_{ij} \rangle = 2^{-D} \sum_i c_{ij} \rho_i \sigma_{ij}, \quad j = 1, 2...D. \tag{5.22}$$

In the next section, we will also use **hypermomentum** density

$$q_\alpha := 2^{-D} \sum_i c_{i\alpha} \langle n_{i\alpha} \rangle, \ \alpha = 0, 1, 2...D, \tag{5.23}$$

where q_0 is the mass density ρ, and $(q_1, q_2, ...q_D)$ is the momentum density.

We can use the above definitions to solve equations 5.20 in terms of ρ and $\boldsymbol{q}$. To find the solution in the closed-form is not possible, but one can find the asymptotic solution for small $\boldsymbol{q}$, derived in the Appendix B of [12].

$$\begin{aligned} \rho_i &= \rho + 2\frac{1-\rho}{2-\rho} \boldsymbol{v}.\boldsymbol{q} + 2\,\frac{(1-\rho)(1-2\rho}{(2-\rho)^2\rho} \left[(\boldsymbol{v}.\boldsymbol{q})^2 - q^2 \right] + O(q^3), \\ \sigma_{ij} &= \frac{1}{2} + \frac{v_j q_j}{(2-\rho)\rho} + O(q^3). \end{aligned} \tag{5.24}$$

5.7 Hydrodynamic description

The quantities of our interest are the so-called *collisional invariant*. These quantities do not change by the collisions, but vary slowly in time and space. On the most general level, they are temperature, mass and momentum, and the system they describe is in local equilibrium.

In this section, the system is assumed to be near its local equilibrium, so in the first approximation, it can be described as the local equilibrium state given by 5.24 with the hypermomentum as its only parameter:

$$\begin{aligned} \rho_i^0 &= \rho_i^0(q_\alpha), \\ \sigma_{i\alpha}^0 &= \sigma_{i\alpha}^0(q_\alpha), \end{aligned}$$

We are looking for the hydrodynamic equations of the form

$$\frac{\partial q_\alpha}{\partial t} = -\sum_k \nabla_k Q_{\alpha k}, \tag{5.25}$$

where the $Q_{\alpha k}$ is the hypermomentum flux tensor. The first approximation of this equation

$$\frac{\partial q_\alpha}{\partial t} = -\sum_k \nabla_k Q_{\alpha k}^0(q_\alpha)$$

is time-reversal and symmetric, so it cannot capture the viscosity and other friction effects.

In order to proceed, we need to expand the functionals $p_{i\alpha} := (\rho_i, \sigma_{ij})$ and $Q_{\alpha k}$ in the Chapman-Enskog series

$$p_{i\alpha} = p^0_{i\alpha}(\boldsymbol{q}) + \sum_{\beta j} R_{i\alpha\beta j}(\boldsymbol{q})\nabla_j q_\beta + O(\nabla^2), \tag{5.26}$$

$$Q_{\alpha k} = Q^0_{\alpha k}(\boldsymbol{q}) - \sum_{\beta j} T_{\alpha\beta k j}(\boldsymbol{q})\nabla_j q_\beta + O(\nabla^2), \tag{5.27}$$

where the tensor $T_{\alpha\beta kj}$ plays the role of the diffusion coefficients. Note that this expansion is justified only for sufficiently small lattice spacing Δx, because

$$O(\nabla^k) = O((\Delta x)^{-k}).$$

The unknown terms in the equations 5.26 and 5.27 must be consistent with the microdynamics of our model, and that is the recipe to calculate them. The details of this calculation are shown in Appendix A of [12], we will just summarize the main results.

The momentum flux tensor in the first approximation reads

$$\begin{aligned} g^0_k := Q^0_{0k} &= 2\frac{1-\rho}{2-\rho}q_k + O(\boldsymbol{q}^3), \\ Q^0_{jk} &= \Big(\frac{\rho}{2} - 2\frac{(1-\rho)(1-2\rho)}{(2-\rho)^2\rho}q_j^2\Big)\delta_{jk} + 4\frac{(1-\rho)^2}{(2-\rho)^2}q_j q_k + O(\boldsymbol{q}^3). \end{aligned} \tag{5.28}$$

We have split the hypermomentum flux tensor into its mass component $\boldsymbol{g}$ and the ordinary momentum tensor Q_{jk} to obtain the hydrodynamic equations in the familiar form.

5.8 Euler equations

If we substitute these $\boldsymbol{g}^0$ and Q^0_{jk} into the hydrodynamic equations 5.25, we obtain

$$\begin{aligned} \partial_t\rho + \boldsymbol{\nabla.}\Big[\Big(\frac{10}{9} - \frac{8}{9}\Big)\boldsymbol{q}\Big] &= 0, \\ \partial_t q_j + \sum_k \nabla_k\Big[\frac{\rho}{2}\delta_{jk} + \frac{8}{9}q_j\, q_k\Big] &= 0, \end{aligned} \tag{5.29}$$

that are analogous to the compressible Euler equations

$$\begin{aligned} \partial_t\rho + \boldsymbol{\nabla.q} &= 0, \\ \partial_t q_j + \sum_k \nabla_k\big[p(\rho)\,\delta_{jk} + \frac{1}{\rho}\,q_j\, q_k\big] &= 0. \end{aligned} \tag{5.30}$$

Unfortunately, the tensor Q^0_{jk} from 5.28 possesses non-physical anisotropy, due to the term proportional to $q_j^2\delta_{jk}$, and thus the automaton's hydrodynamics is not equivalent to that of physical fluid.

Notice, however, that if we set density to be constant $\rho = 1/2$, the anisotropic term vanishes and we obtain the incompressible inviscid equations

$$\begin{aligned} \boldsymbol{\nabla}.\boldsymbol{q} &= 0, \\ (\partial_t + \frac{8}{9}\boldsymbol{q}.\boldsymbol{\nabla})\boldsymbol{q} + \frac{1}{2}\boldsymbol{\nabla}\rho &= 0. \end{aligned} \tag{5.31}$$

Comparing them to the physical incompressible Euler equations

$$\begin{aligned} \boldsymbol{\nabla}.\boldsymbol{u} &= 0, \\ (\partial_t + \boldsymbol{u}.\boldsymbol{\nabla})\boldsymbol{u} + \nabla\Phi &= 0, \end{aligned} \tag{5.32}$$

we can easily see that they are equivalent and can be transformed into each other by

$$\Phi = \frac{4}{9}\rho, \quad \boldsymbol{u} = \frac{8}{9}\boldsymbol{q}. \tag{5.33}$$

So our model perfectly simulates the ideal fluid (obeying incompressible Euler equations) with the kinematic pressure Φ and the hydrodynamic velocity $\boldsymbol{u}$ defined by 5.33. This hydrodynamic velocity can be interpreted as the momentum convection velocity, which differs from the mean velocity $\boldsymbol{w}$ of the particles

$$\boldsymbol{w} = \frac{\langle \sum_i v_i n_{i0} \rangle}{\langle \sum_i n_{i0} \rangle} \approx \frac{2(1-\rho)}{(2-\rho)\rho}\boldsymbol{q}. \tag{5.34}$$

The difference is caused by the broken Galilean invariance due to discrete symmetry of our model. The ratio between the velocities is called the "g-factor" ($g(\rho)$), and for our special case $\rho = \frac{1}{2}$, it is equal to 2/3:

$$\boldsymbol{w} = \frac{4}{3}\boldsymbol{q} = \frac{2}{3}\boldsymbol{u}. \tag{5.35}$$

It already arised in FHP and FCHC model (where $g(\rho) \sim 1/2$ for small ρ).

5.9 Navier-Stokes equations

For most applications, the inviscid incompressible approximation is too much idealized, and we would like to find analogous equations to the Navier-Stokes

$$\begin{aligned} \boldsymbol{\nabla}.\boldsymbol{u} &= 0, \\ (\partial_t + \boldsymbol{u}.\boldsymbol{\nabla})\boldsymbol{u} + \nabla\Phi &= \nu\nabla^2\boldsymbol{u} \end{aligned} \tag{5.36}$$

with the friction term on the right hand side, containing the shear viscosity ν.

We need to substitute the momentum flux tensor in the second approximation 5.27 into the hydrodynamic equations 5.25 to obtain

$$\begin{aligned} \boldsymbol{\nabla}.\boldsymbol{q} &= 0, \\ (\partial_t + \frac{8}{9}\boldsymbol{q}.\boldsymbol{\nabla})q_j + \frac{1}{2}\nabla_j\rho &= \sum_{klm} T_{jklm}\nabla_l\nabla_m q_k. \end{aligned} \tag{5.37}$$

If we could show that the viscosity tensor T_{jklm} is isotropic, these equations would be equivalent to Navier-Stokes 5.36 by means of the transformation 5.33, but our model do not posses symmetries that forces T_{jklm} to be isotropic.

The more detailed calculation of the friction terms, that spans pages 24-32 in [12], leads to

$$\begin{aligned} T_{00km} &= \frac{1}{9}\delta_{kl}, \\ T_{jklm} &= \frac{1}{6}(3\delta_{jl}\delta_{km} + \delta_{jm}\delta_{lk}(1+\gamma_{mk}) - 2\delta_{jklm}). \end{aligned} \tag{5.38}$$

To prove that the tensor is not isotropic, we consider a simple example of two-dimensional flow in x_1-direction:

$$q_1 = q_1(t, x_2), \qquad q_2 = 0. \tag{5.39}$$

Then, the Navier-Stokes equations 5.36 reduces to

$$\partial_t q_1 = \nu(\nabla_2)^2 q_1, \tag{5.40}$$

where we introduced the shear viscosity coefficient $\nu := T_{1122}$.

If we rotate the flow by the angle α, the equation changes to

$$\partial_t q_1' = \nu'(\nabla_2')^2 q_1', \tag{5.41}$$

The prime quantities above were obtained by rotation

$$\begin{aligned} \boldsymbol{x} &= R\,\boldsymbol{x'}, \\ \nabla &= R\nabla, \\ \boldsymbol{q} &= R\boldsymbol{q'}, \\ T'_{j'k'l'm'} &= \sum_{jklm} T_{jklm} R_{jj'} R_{kk'} R_{ll'} R_{mm'}, \end{aligned} \tag{5.42}$$

using the orthogonal matrix

$$R = \begin{pmatrix} cos(\alpha) & sin(\alpha) \\ -sin(\alpha) & cos(\alpha) \end{pmatrix}. \tag{5.43}$$

The shear viscosity transformed into rotated coordinate system reads

$$\begin{aligned} \nu' = T'_{1122} = T_{1122}cos^4(\alpha) + T_{2211}sin^4(\alpha) \\ +(T_{1111} + T_{2222} - T_{1212} - T_{2121} - T_{1221} - T_{2112})cos^2(\alpha)sin^2(\alpha) \end{aligned} \tag{5.44}$$

Into this equation, we plug the viscosity tensor in original coordinate system

$$\begin{aligned} T_{1122} = T_{2211} &= \frac{1}{2}, \\ T_{1111} = T_{2222} &= \frac{1}{3}, \\ T_{1212} = T_{2121} &= 0, \\ T_{1221} &= \frac{1}{6}, \\ T_{2112} &= \frac{1}{2}. \end{aligned} \tag{5.45}$$

and find out that

$$\nu' = \frac{1}{2}(cos^4(\alpha) + sin^4(\alpha)). \tag{5.46}$$

It clearly shows that the shear viscosity is angle-dependent and proves anisotropy of the viscosity tensor 5.38.

6. From Liouville to Navier-Stokes

In the previous chapters, we derived hydrodynamic equations for LGCA to find out that they do not match the Eulear and Navier-Stokes equation precisely. This flaw is caused by discrete microdynamics of LGCA, which leads to the Fermi-Dirac distribution. To treat this flaw, the Lattice-Boltzmann models have emerged.

We may say that Lattice-Boltzmann method stands on the two pillars – Lattice gas cellular automata are the first pillar, and in this chapter we will establish the second one – kinetic theory of gasses [10].

6.1 Liouville's equation

Let us start by invoking the Hamiltonian for the N particle system

$$H = \frac{1}{2m}\sum_{i=1}^{N} p_i^2 + \sum_{i=1}^{N} V(\boldsymbol{r_i}) + \sum_{i<j} U(\boldsymbol{r_i} - \boldsymbol{r_j})$$

where V is the potential of the force $\boldsymbol{F} = -\nabla V$, that affects all the particles equally, and U is the potential of the 2-particle interactions. We require that U is short-ranged, meaning $U(r) \approx 0$ for r of order much larger then is the atomic distance.

As usual in the framework of continuum mechanics, N is a ridiculously large number, something like $N \approx 10^{23}$.

Hence, we represent the state of the system by the probability density function $f(\boldsymbol{r}, \boldsymbol{p}, t)$. It specifies the probability that the system is found in the vicinity of point $(\boldsymbol{r}, \boldsymbol{p}) = (\boldsymbol{r_1}, ..., \boldsymbol{r_N}, \boldsymbol{p_1}, ..., \boldsymbol{p_N})$.

As usual, the probability is normalized:

$$\int f(\boldsymbol{r}, \boldsymbol{p}, t)\, \mathrm{d}\boldsymbol{r}\, \mathrm{d}\boldsymbol{p} = 1, \quad \mathrm{d}\boldsymbol{p}\mathrm{d}\boldsymbol{r} = \prod_{i=1}^{N} \mathrm{d}\boldsymbol{r_i}\mathrm{d}\boldsymbol{p_i}.$$

Since the microdynamics under consideration is deterministic, the probability density function is locally conserved and we may write its continuum equation

$$\frac{\partial f}{\partial t} + \frac{\partial}{\partial \boldsymbol{r}}(\dot{\boldsymbol{r}} f) + \frac{\partial}{\partial \boldsymbol{p}}(\dot{\boldsymbol{p}} f) = 0 \tag{6.1}$$

If we substitute the Hamilton's equations

$$\frac{\partial \boldsymbol{p}_i}{\partial t} = -\frac{\partial H}{\partial \boldsymbol{r}_i}, \; \frac{\partial \boldsymbol{r}_i}{\partial t} = \frac{\partial H}{\partial \boldsymbol{p}_i}$$

into the equation 6.1, we obtain the Liouville's equation

$$\frac{\partial f}{\partial t} + \sum_{i=1}^{N} \frac{\partial f}{\partial \boldsymbol{r_i}} \frac{\partial H}{\partial \boldsymbol{p_i}} - \frac{\partial f}{\partial \boldsymbol{p_i}} \frac{\partial H}{\partial \boldsymbol{r_i}} = 0.$$

It is often written using the Poisson brackets

$$\frac{\partial f}{\partial t} = \{H,\, f\}. \tag{6.2}$$

6.2 The BBGKY hierarchy

So far, the transition to probability description did not simplified the problem much – we are still dealing with the function of $N \approx 10^{23}$ variables.

We need to limit our ambitions and we will focus on the one-particle distribution, defined by

$$f_1(\boldsymbol{r_1}, \boldsymbol{p_1}, t) = N \int \prod_{i=2}^{N} \mathrm{d}\,\boldsymbol{r_i} \mathrm{d}\,\boldsymbol{p_i} f(\boldsymbol{r}, \boldsymbol{p}, t). \tag{6.3}$$

Leaving the particle with index 1 from the product was arbitrary, because all the particles are identical. This is also the reason why f_1 is normalized to N:

$$\int f_1(\boldsymbol{r_1}, \boldsymbol{p_1}, t) \mathrm{d}\boldsymbol{r_1} \mathrm{d}\boldsymbol{p_1} = N.$$

To simplify the notation, we will use $(\boldsymbol{p}, \boldsymbol{r})$ instead $(\boldsymbol{p_1}, \boldsymbol{r_1})$ from now on.

For many purposes, the function f_1 captures all we need to know. For example, it gives us the average density of particles at the spatial point $\boldsymbol{r}$

$$n(\boldsymbol{r}, t) = \int f_1(\boldsymbol{r}, \boldsymbol{p}, t) \mathrm{d}\boldsymbol{p}, \tag{6.4}$$

the average velocity of the particle

$$u(\boldsymbol{r}, t) = \int \frac{\boldsymbol{p}}{m} f_1(\boldsymbol{r}, \boldsymbol{p}, t) \mathrm{d}\boldsymbol{p},$$

and the energy flux

$$\varepsilon(\boldsymbol{r}, t) = \int \frac{\boldsymbol{p}}{m} E(\boldsymbol{p}) f_1(\boldsymbol{r}, \boldsymbol{p}, t) \mathrm{d}\boldsymbol{p}.$$

To derive the equation governing the evolution of f_1, let us see how it varies in time:

$$\frac{\partial f_1}{\partial t} = N \int \prod_{i=2}^{N} \mathrm{d}\,\boldsymbol{r_i} \mathrm{d}\,\boldsymbol{p_i} \frac{\partial f}{\partial t} = N \int \prod_{i=2}^{N} \mathrm{d}\,\boldsymbol{r_i} \mathrm{d}\,\boldsymbol{p_i} \{H,\, f\}.$$

Arranging the terms on the right, we can turn this equation in the form

$$\frac{\partial f_1}{\partial t} = \{H_1, f_1\} + \left(\frac{\partial f_1}{\partial t}\right)_{coll}, \tag{6.5}$$

where

$$H_1 = \frac{p^2}{2m} + V(\boldsymbol{r}).$$

We see that the evolution of f_1 is governed by the Liouville's equation plus the extra term, the *collision integral*

$$\left(\frac{\partial f_1}{\partial t}\right)_{coll} = N(N-1) \int \mathrm{d}\boldsymbol{r_2} \mathrm{d}\boldsymbol{p_2} \frac{\partial U(\boldsymbol{r} - \boldsymbol{r_2})}{\partial \boldsymbol{r}} \cdot \frac{\partial}{\partial \boldsymbol{p}} \int \prod_{i=3}^{N} \mathrm{d}\boldsymbol{r_i} \mathrm{d}\boldsymbol{p_i} f(\boldsymbol{r}, \boldsymbol{r_2}, ..., \boldsymbol{p}, \boldsymbol{p_2}, ..., t).$$

Not surprisingly, it is not determined by the 1-particle distribution, because it does not contain the relation of one particle to all other particles. Some of that information is carried by the *2-particle distribution*

$$f_2(\boldsymbol{r}, \boldsymbol{r_2}, \boldsymbol{p}, \boldsymbol{p_2}, t) := N(N-1) \int f(\boldsymbol{r}, \boldsymbol{r_2}, \boldsymbol{r_3}, ..., \boldsymbol{p}, \boldsymbol{p_2}, \boldsymbol{p_3}, ..., t) \prod_{i=3}^{N} \mathrm{d}\boldsymbol{r_i} \mathrm{d}\boldsymbol{p_i}.$$

Using the 2-particle distribution, we can rewritten the collision integral as

$$\Big(\frac{\partial f_1}{\partial t}\Big)_{coll} = \int \mathrm{d}\boldsymbol{r_2} \mathrm{d}\boldsymbol{p_2} \frac{\partial U(\boldsymbol{r} - \boldsymbol{r_2})}{\partial \boldsymbol{r}} . \frac{\partial f_2}{\partial \boldsymbol{p}}.$$

We see that to follow the evolution of one-particle distribution f_1, we also need to know something about the two-particle distribution f_2. And again, to know the evolution of the two-particle distribution, we repeat the same procedure as before and find the Liouville-like equation for f_2 with additional term containing the three-particle distribution f_3.

In general, the evolution of *n-particle distribution* is described by

$$\frac{\partial f_n}{\partial t} = \big\{H_n, f_n\big\} + \sum_{i=1}^{n} \int \mathrm{d}\boldsymbol{p_{n+1}} \mathrm{d}\boldsymbol{r_{n+1}} \frac{\partial U(\boldsymbol{r_i} - \boldsymbol{r_{n+1}})}{\partial \boldsymbol{r_i}}) . \frac{\partial f_{n+1}}{\partial \boldsymbol{p_i}}, \tag{6.6}$$

where the Hamiltonian

$$H_n = \sum_{i=1}^{n} \Big(\frac{\boldsymbol{p_i}}{2m} + V(\boldsymbol{p_i}) \Big) + \sum_{i<j\leq n} U(\boldsymbol{r_i} - \boldsymbol{r_j})$$

includes external forces V and interactions between the n particles.

The set of equations 6.6 is called the *BBGKY hierarchy*. Instead of the Liouville's equation governing the function of $N \approx 10^{23}$ variables, we have now the set of 10^{23} functions.

However, it is more advantageous to work with the hierarchy of equations 6.6, as it is ready to be approximated. Depending on the particular problem at hands, we decide which terms are important and which terms can be ignored, thus truncating the problem to something manageable.

6.3 Boltzmann equation

Let us derive the most simple and useful of these truncations, the Boltzmann equation

$$\frac{\partial f_1}{\partial t} = \big\{H_1, f_1\big\} + \Big(\frac{\partial f_1}{\partial t}\Big)_{coll},$$

in the closed form, where collision integral is the expression of f_1 alone.

To proceed, let us define two time scales: the time between particle collisions τ, and the time of the collision itself, τ_{coll}. In a dilute gas, we have

$$\tau >> \tau_{coll},$$

so most of the time, f_1 follows the Hamiltonian evolution, with occasional, but abrupt perturbations by the collision.

What is the rate at which the collision happens?

Let the two particles with momenta $\boldsymbol{p_1}$ and $\boldsymbol{p_2}$ collide at the point $\boldsymbol{r}$, and during the collision, they gain velocities $\boldsymbol{p_1'}$ and $\boldsymbol{p_2'}$. We can define the rate of the collision by

$$Rate = \omega(\boldsymbol{p_1}, \boldsymbol{p_2}|\boldsymbol{p_1'}, \boldsymbol{p_2'}) f_2(\boldsymbol{r}, \boldsymbol{r}, \boldsymbol{p_1}, \boldsymbol{p_2}),$$

where ω carries information about the dynamics of the collision, and can be computed from the inter-atomic potential U [10].

Next, we focus on the collisions where

1. one of the particles has specific momentum $\boldsymbol{p_1}$ before collision, and gains some unspecified momentum $\boldsymbol{p_1'}$ in the collision, or
2. the particle had unspecified momentum $\boldsymbol{p_1'}$ before the collision, but gains the specific momentum $\boldsymbol{p_1}$ in the collision.

The collision integral governing these collisions contains two terms:

$$\Big(\frac{\partial f_1}{\partial t}\Big)_{coll} = \int \big[\omega(\boldsymbol{p_1'}, \boldsymbol{p_2'}|\boldsymbol{p_1}, \boldsymbol{p_2}) f_2(\boldsymbol{p_1'}, \boldsymbol{p_2'}) - \omega(\boldsymbol{p_1}, \boldsymbol{p_2}|\boldsymbol{p_1'}, \boldsymbol{p_2'}) f_2(\boldsymbol{p_1}, \boldsymbol{p_2})\big] \mathrm{d}\boldsymbol{p_2} \mathrm{d}\boldsymbol{p_1'} \mathrm{d}\boldsymbol{p_2'}.$$

Considering the symmetries of the process (invariance under time reversal and parity), we have

$$\omega(\boldsymbol{p_1'}, \boldsymbol{p_2'}|\boldsymbol{p_1}, \boldsymbol{p_2}) = \omega(\boldsymbol{p_1}, \boldsymbol{p_2}|\boldsymbol{p_1'}, \boldsymbol{p_2'}). \tag{6.7}$$

Finally, in order to express the collision integral in terms of f_1 instead of f_2, we make a non-trivial assumption, that velocities of colliding particles are uncorrelated:

$$f_2(\boldsymbol{r}, \boldsymbol{r}, \boldsymbol{p_1}, \boldsymbol{p_2}) = f_1(\boldsymbol{r}, \boldsymbol{p_1})\, f_1(\boldsymbol{r}, \boldsymbol{p_2}). \tag{6.8}$$

This assumption is known as a *molecular chaos hypothesis*, and seems quite innocent. However, by supposing that particles are uncorrelated before the collision, but they inevitably become correlated in the collision, we introduce the arrow of time on the scene.

Finally, using 6.7 and 6.8, we can write the collision integral in the form

$$\Big(\frac{\partial f_1}{\partial t}\Big)_{coll} = \int \omega(\boldsymbol{p_1'}, \boldsymbol{p_2'}|\boldsymbol{p_1}, \boldsymbol{p_2}) \big[f_1(\boldsymbol{r}, \boldsymbol{p_1'}) f_1(\boldsymbol{r}, \boldsymbol{p_2'}) - f_1(\boldsymbol{r}, \boldsymbol{p_1}) f_1(\boldsymbol{r}, \boldsymbol{p_2})\big] \mathrm{d}\boldsymbol{p_2} \mathrm{d}\boldsymbol{p_1'} \mathrm{d}\boldsymbol{p_2'} \tag{6.9}$$

and we can express the *Boltzmann equation*

$$\frac{\partial f_1}{\partial t} = \big\{H_1, f_1\big\} + \Big(\frac{\partial f_1}{\partial t}\Big)_{coll} \tag{6.10}$$

in the closed form.

6.4 Equilibrium and detailed balance

The *equilibrium distribution* is defined such that it does not explicitly depend on the time:

$$\frac{\partial f_1^{eq}}{\partial t} = 0.$$

Using the Boltzmann equation, we can express this condition equivalently as

$$\{f_1, H_1\} + \Big(\frac{\partial f_1}{\partial t}\Big)_{coll} = 0.$$

If we restrict ourselves to the case with vanishing external force

$$V(\boldsymbol{r}) = 0,$$

any function of momentum commutes with Hamiltonian and the streaming term $\{f_1, H_1\}$ vanishes. According to 6.9, to make the collision integral vanish, we need to satisfy the *detailed balance* condition

$$f_1^{eq}(\boldsymbol{r}, \boldsymbol{p_1}) f_1^{eq}(\boldsymbol{r}, \boldsymbol{p_2}) = f_1^{eq}(\boldsymbol{r}, \boldsymbol{p_1'}) f_1^{eq}(\boldsymbol{r}, \boldsymbol{p_2'}), \tag{6.11}$$

or equivalently

$$\log(f_1^{eq}(\boldsymbol{r}, \boldsymbol{p_1})) + \log(f_1^{eq}(\boldsymbol{r}, \boldsymbol{p_2})) = \log(f_1^{eq}(\boldsymbol{r}, \boldsymbol{p_1'})) + \log(f_1^{eq}(\boldsymbol{r}, \boldsymbol{p_2'})).$$

According to this equation, the sum of logarithms is conserved in the collision.

As we know, such quantities are either momenta or energy, hence the logarithms are the linear combinations of energy and momenta

$$log(f_1^{eq}(\boldsymbol{r}, \boldsymbol{p})) = \beta(\mu - E(\boldsymbol{p}) + \boldsymbol{u} \cdot \boldsymbol{p}),$$

where

$$E(\boldsymbol{p}) = \frac{1}{2m} p^2$$

and β, μ and $\boldsymbol{u}$ are constants. We need to set the constant μ such that f_1 is normalized, which gives us

$$f_1^{eq} = \frac{N}{V} \Big(\frac{\beta}{2\pi m}\Big)^{3/2} e^{-\beta m (\boldsymbol{v} - \boldsymbol{u})^2/2}. \tag{6.12}$$

Supposing that β is the inverse temperature, this is the Maxwell-Boltzmann distribution.

If we forget about the streaming term $\{H_1, f_1\}$, then the detailed balance condition 6.11 leads to the much larger class of solutions, so called *local equilibrium distributions*

$$f_1^{loc}(\boldsymbol{r}, \boldsymbol{p}, t) = n(\boldsymbol{r}, t) \Big(\frac{\beta(\boldsymbol{r}, t)}{2\pi m}\Big)^{3/2} \exp\big(-\beta(\boldsymbol{r}, t) \frac{m}{2} [\boldsymbol{v} - \boldsymbol{u}(\boldsymbol{r}, t)]^2\big) \tag{6.13}$$

where n, β and $\boldsymbol{u}$ are promoted to the functions. By inserting these distributions into the Boltzmann equation, the collision integral vanishes, but the streaming term stays.

6.5 Hydrodynamics

On the most coarse-grained level, the system in local equilibrium is described by hydrodynamics, with the parameters that do not relax to their equilibrium values immediately, but vary slowly in space and time. In this section we will show that these parameters are density $\rho(\boldsymbol{r}, t)$, temperature $T(\boldsymbol{r}, t)$ and velocity $\boldsymbol{u}(\boldsymbol{r}, t)$ (thus showing that the functions in 6.13 have their usual meaning).

Let us have an arbitrary function $A(\boldsymbol{r}, \boldsymbol{u})$. To see how does the function A vary in space, we integrate over the momentum space

$$\langle A(\boldsymbol{r}, t)\rangle = \frac{\int A(\boldsymbol{r}, \boldsymbol{p}) f_1(\boldsymbol{r}, \boldsymbol{p}, t)\mathrm{d}\boldsymbol{p}}{\int f_1(\boldsymbol{r}, \boldsymbol{p}, t)\mathrm{d}\boldsymbol{p}}. \tag{6.14}$$

We are already familiar with the denominator – it is the particle density function, so we can write

$$\langle A(\boldsymbol{r}, t)\rangle = \frac{1}{n(\boldsymbol{r}, t)} \int A(\boldsymbol{r}, \boldsymbol{p}) f_1(\boldsymbol{r}, \boldsymbol{p}, t)\mathrm{d}\boldsymbol{p}.$$

Note that the resulting function is a function of space and time, since we computed average only over the momentum variables.

We are interested in the functions A that vanish, when integrated against the collision integral

$$\int A(\boldsymbol{r}, \boldsymbol{p}) \Big(\frac{\partial f_1}{\partial t}\Big)_{coll} \mathrm{d}\boldsymbol{p} = 0.$$

Intuitively, this condition is required because we are interested in the quantities that vary slowly, but the collisions are abrupt processes, and the term involving collision integral vary fast.

Substituting the expression for the collision integral 6.9 to the last equation, we get

$$\int A(\boldsymbol{r}, \boldsymbol{p_1}) \omega(\boldsymbol{p_1'}, \boldsymbol{p_2'} | \boldsymbol{p_1}, \boldsymbol{p_2}) \big[f_1(\boldsymbol{r}, \boldsymbol{p_1'}) f_1(\boldsymbol{r}, \boldsymbol{p_2'}) - f_1(\boldsymbol{r}, \boldsymbol{p_1}) f_1(\boldsymbol{r}, \boldsymbol{p_2})\big] \mathrm{d}\boldsymbol{p_1}\mathrm{d}\boldsymbol{p_2}\mathrm{d}\boldsymbol{p_1'}\mathrm{d}\boldsymbol{p_2'} = 0.$$

Considering the symmetries of ω, this condition is equivalent to

$$\begin{aligned} \int \omega(\boldsymbol{p_1'}, \boldsymbol{p_2'} | \boldsymbol{p_1}, \boldsymbol{p_2}) \big[f_1(\boldsymbol{r}, \boldsymbol{p_1'}) f_1(\boldsymbol{r}, \boldsymbol{p_2'}) - f_1(\boldsymbol{r}, \boldsymbol{p_1}) f_1(\boldsymbol{r}, \boldsymbol{p_2})\big] \\ \times \big[A(\boldsymbol{r}, \boldsymbol{p_1}) + A(\boldsymbol{r}, \boldsymbol{p_2}) - A(\boldsymbol{r}, \boldsymbol{p_1'}) - A(\boldsymbol{r}, \boldsymbol{p_2'})\big] \mathrm{d}\boldsymbol{p_1}\mathrm{d}\boldsymbol{p_2}\mathrm{d}\boldsymbol{p_1'}\mathrm{d}\boldsymbol{p_2'} = 0 \end{aligned},$$

which holds if

$$A(\boldsymbol{r}, \boldsymbol{p_1}) + A(\boldsymbol{r}, \boldsymbol{p_2}) = A(\boldsymbol{r}, \boldsymbol{p_1'}) + A(\boldsymbol{r}, \boldsymbol{p_2'}).$$

So A is a quantity that is conserved during the collision, which is true for the energy, momenta, and trivially, for a constant function, e.g. $A = 1$. These are the *collision invariants* of our interest.

If we plug any collision invariant to the Boltzmann equation 6.10 and integrate over the momenta variables, the term involving collision integral vanishes

$$\int A(\boldsymbol{r}, \boldsymbol{p}) \Big(\frac{\partial}{\partial t} + \frac{\boldsymbol{p}}{m} \cdot \frac{\partial}{\partial \boldsymbol{r}} + \boldsymbol{F} \cdot \frac{\partial}{\partial \boldsymbol{p}}\Big) f_1(\boldsymbol{r}, \boldsymbol{p}, t)\mathrm{d}\boldsymbol{p} = 0$$

Because A do not explicitly depend on time, we can put it inside the time derivative, and using integration by part on other terms *per partes* leads to

$$\frac{\partial}{\partial t}\int Af\mathrm{d}\boldsymbol{p} + \frac{\partial}{\partial \boldsymbol{r}}\cdot\int \frac{\boldsymbol{p}}{m}Af\mathrm{d}\boldsymbol{p} - \int \frac{\boldsymbol{p}}{m}\cdot\frac{\partial A}{\partial \boldsymbol{r}}f\mathrm{d}\boldsymbol{p} - \int \boldsymbol{F}\cdot\frac{\partial A}{\partial \boldsymbol{p}}f\mathrm{d}\boldsymbol{p} = 0$$

Finally, applying the expression for the average 6.14 on this equation, we can write

$$\frac{\partial}{\partial t}\langle nA\rangle + \frac{\partial}{\partial \boldsymbol{r}}\cdot\langle n\boldsymbol{u}A\rangle - n\langle \boldsymbol{u}\cdot\frac{\partial A}{\partial \boldsymbol{r}}\rangle - n\langle \boldsymbol{F}\cdot\frac{\partial A}{\partial \boldsymbol{u}}\rangle = 0, \tag{6.15}$$

which is our master equation governing the evolution of the collisional invariant. In between, we have introduced notation for the velocity

$$\boldsymbol{v} = \frac{\boldsymbol{p}}{m},$$

and now we introduce the average velocity

$$\boldsymbol{u} = \langle \boldsymbol{v}\rangle.$$

Substituting the constant $A = 1$ into the master equation yields

$$\frac{\partial n}{\partial t} + \frac{\partial}{\partial \boldsymbol{r}}\cdot(n\boldsymbol{u}).$$

Multiplying by particle mass m, this is the continuity equation.

Substituting the next collisional invariant, the momentum $A = m\boldsymbol{v}$, yields

$$\frac{\partial}{\partial t}(mnu_i) + \frac{\partial}{\partial r_j}\langle mnv_iv_j\rangle - \langle nF_i\rangle = 0 \tag{6.16}$$

If we expand the middle term like

$$\langle v_iv_j\rangle = \langle (u_j - v_j)(u_i - v_i)\rangle + v_i\, v_j$$

and if we define the quantity

$$P_{ij} = P_{ji} = \rho\langle (u_j - v_j)(u_i - v_i)\rangle$$

known as the pressure tensor, we can rewrite the equation 6.16 as

$$\rho\Big(\frac{\partial}{\partial t} + u_j\frac{\partial}{\partial x_j}\Big)u_i = \frac{\rho}{m}F_i - \frac{\partial}{\partial x_j}P_{ij}. \tag{6.17}$$

This is the familiar form of the momentum conservation law. We can interpret it according to the Newton's law – the left-hand side is the acceleration of the fluid element, and the right-hand side involves the external force $\boldsymbol{F}$ and the internal pressure of the fluid

If we evaluate the pressure tensor on the equilibrium distribution, we find out that it is proportional to the temperature

$$P_{ij} = nk_BT\delta_{ij},$$

which is the ideal gas law.

The last collisional invariant is the kinetic energy of particles. Instead of absolute kinetic energy, we shall use the relative kinetic energy

$$A = \frac{1}{2}m(\boldsymbol{u} - \boldsymbol{v})^2,$$

so when we substitute it into our master equation 6.15, we get the compact form

$$\frac{1}{2}\frac{\partial}{\partial t}\langle\rho(\boldsymbol{v} - \boldsymbol{u})\rangle + \frac{1}{2}\frac{\partial}{\partial r_i}\langle\rho v_i(\boldsymbol{v} - \boldsymbol{u})\rangle - \rho\langle v_i\frac{\partial u_j}{\partial r_i}(\boldsymbol{v} - \boldsymbol{u})^2\rangle = 0. \tag{6.18}$$

Using the idea of equipartition, we can define the temperature for the non-equilibrium system

$$\frac{3}{2}k_B T(\boldsymbol{r}, t) = \frac{1}{2}m\langle(\boldsymbol{v} - \boldsymbol{u}(\boldsymbol{r}, t))^2.\rangle$$

Let us define another quantity, the *heat flux*

$$\frac{1}{2}m\rho\langle(v_i - u_i)(\boldsymbol{v} - \boldsymbol{u})^2\rangle.$$

Using the definition of heat flux and pressure tensor, we rewrite the equation 6.18 as

$$\frac{1}{2}m\rho\langle v_i(\boldsymbol{v} - \boldsymbol{u})^2\rangle = q_i + \frac{3}{2}\rho u_i k_B T,$$

Since the pressure tensor is symmetric, we can replace $\partial u_j/\partial r_i$ with the symmetric *rate of strain*

$$U_{ij} = \frac{1}{2}\big(\frac{\partial u_j}{\partial r_i} + \frac{\partial u_i}{\partial r_j}\big).$$

Further, using the continuity equation, we obtain the conservation of energy in its usual form

$$\rho\big(\frac{\partial}{\partial t} + u_i\frac{\partial}{\partial r_i}\big)k_B T + \frac{2}{3}\frac{\partial q_i}{\partial r_i} + \frac{2m}{3}U_{ij}P_{ij} = 0.$$

Finally, we have the three equations, describing the time evolution of the particle density n, momentum $\boldsymbol{u}$ and the temperature T. Although they hold for any distribution f_1, they do not constitute the closed set of equations – equation for n depends on $\boldsymbol{u}$, equation for momentum depends on the pressure tensor P_{ij} and the equation for temperature depends on the heat flux q. To determine any of these, we need to compute the distribution f_1 from the Boltzmann equation, which is not easy.

Since we are looking for the solution that varies slowly, the first approximation would be the local equilibrium distribution 6.13, for which collision integral vanishes. Using this distribution, we can compute that

$$P_{ij} = k_b n(\boldsymbol{r}, t)T(\boldsymbol{r}, t)\delta_{ij} = P(\boldsymbol{r}, t)\delta_{ij}$$

and

$$q = 0.$$

Inserting these terms into the conservation laws, the continuity equation stays unchanged, we just rewrite it using $\rho = m\,n$:

$$\big(\frac{\partial}{\partial t} + u_j \frac{\partial}{\partial r_j}\big)\rho + \rho \frac{\partial u_i}{\partial r_i}. \tag{6.19}$$

However, the momentum conservation 6.17 turns into the Euler equation

$$\big(\frac{\partial}{\partial t} + u_j \frac{\partial}{\partial r_j}\big)u_i + \frac{1}{\rho}\frac{\partial P}{\partial x_i} = \frac{F_i}{m}. \tag{6.20}$$

However, we are still missing dissipation in these equations. To show it, we can combine equations 6.19 and 6.20 to obtain

$$\big(\frac{\partial}{\partial t} + u_j \frac{\partial}{\partial r_j}\big)(\rho T^{-2/3}) = 0$$

This equation implies that $\rho T^{-2/3}$ is constant along the streamlines, which means that the motion of the fluid is adiabatic – not increasing entropy. Hence we have no mechanism for the fluid to return to the equilibrium yet.

It is easy to see what we are missing. We chose local equilibrium as an approximation for the distribution f_1, so that the collision integral vanishes, but it does not solve the streaming term. However, if we stick to the long wave-length variations in the velocity, we need to add only little correction term

$$f_1 = f_1^{(0)} + \delta f_1. \tag{6.21}$$

The correction δf_1 contributes to the collision integral

$$\Big(\frac{\partial f_1}{\partial t}\Big)_{coll} = \int \omega(\boldsymbol{p_1'}, \boldsymbol{p_2'}|\boldsymbol{p_1}, \boldsymbol{p_2})\big[f_1(\boldsymbol{p_1'})f_1(\boldsymbol{p_2'}) - f_1(\boldsymbol{p_1})f_1(\boldsymbol{p_2})\big]\mathrm{d}\boldsymbol{p_2}\mathrm{d}\boldsymbol{p_1'}\mathrm{d}\boldsymbol{p_2'}$$

where we neglected the quadratic terms $O(\delta f_1^2)$ and used the fact that $f_1^{(0)}$ vanish in the collision integral.

To proceed further, the proper way would be to use *Chapman-Enskog expansion* of δf_1 to treat the collision operator properly, but there is an easier way to progress, the so called BGK [1] approximation. We simply set the collision integral to be

$$\Big(\frac{\partial f_1}{\partial t}\Big)_{coll} = -\frac{\delta f_1}{\tau}, \tag{6.22}$$

where τ is the relaxation time, and we take it to be a constant (although in general, it might be function of momentum). With this replacement, the Boltzmann equation changes to

$$\frac{\partial(f_1^{(0)} + \delta f_1)}{\partial t} - \big\{H_1, f_1^{(0)} + \delta f_1\big\} = -\frac{\delta f_1}{\tau}$$

[1] Bhatnagar-Gross-Krook

6.6 Where is the viscosity?

Let us consider the flow of the fluid with the velocity gradient $\partial u_x/\partial z \neq 0$. According to the Newton's law, the shear viscosity is associated with the flux of x-momentum to the z-direction:

$$\Pi_{xz} = -\eta \frac{\partial u_x}{\partial r_z}, \tag{6.23}$$

which is the off-diagonal component of the pressure tensor

$$P_{xz} = \rho \langle (v_x - u_x)(v_z - u_z) \rangle.$$

As we already know, the local equilibrium distribution gives us the diagonal pressure tensor, but if we use the corrected distribution, we have

$$P_{ij} = P\delta_{ij} + \Pi_{ij}, \tag{6.24}$$

with the additional term Π_{ij}, which is a non-diagonal *stress tensor*

$$\Pi_{ij} = \frac{m\tau\rho}{k_B T} U_{kl} = \Big[\langle v_i v_j v_k v_l \rangle_0 - \frac{1}{3} \delta_{kl} \langle v_i v_j v^2 \rangle_0 \Big].$$

In fact, the Π_{ij} is traceless, because

$$\langle v^2 v_k v_l \rangle_0 = \delta_{kl} \langle v_i v_j v^2 \rangle_0.$$

and it depends linearly on the U_{kl}. These two properties determine the form of the tensor

$$\Pi_{ij} = -2\mu \big(U_{ij} - \frac{1}{3} \delta_{ij} \nabla . \boldsymbol{u} \big). \tag{6.25}$$

For $\partial u_x/\partial z \neq 0$, we can evaluate

$$\Pi_{xz} = \frac{2m\tau\rho}{15 k_B T} U_{xx} \langle v^4 \rangle_0.$$

Comparing it to the Newton's law 6.23, we get an expression for the shear viscosity

$$\eta = n k_B T \tau. \tag{6.26}$$

Note that this expression is just an estimation, as it relies on the relaxation time approximation τ.

6.7 Navier-Stokes equations

The corrected distribution 6.21 does not change the density equation

$$\frac{\partial \rho}{\partial t} + \nabla . (\rho \boldsymbol{u}) = 0,$$

but due to the non-diagonal stress tensor 6.25, the momentum equation transforms to the Navier-Stokes equation

$$\Big(\frac{\partial}{\partial t} + \boldsymbol{u} \cdot \nabla \Big) \boldsymbol{u} = \frac{\boldsymbol{F}}{m} - \frac{1}{\rho} \nabla P + \frac{\eta}{\rho} \nabla^2 \boldsymbol{u} + \frac{\eta}{3\rho} \nabla (\nabla \cdot \boldsymbol{u}) \tag{6.27}$$

Although our derivation relied on the dilute gas approximation, the Navier-Stokes equation are more general then that and can be considered to be the most general expression for the momentum transport [10].

7. Lattice Boltzmann method

In the previous chapter, we showed that Boltzmann equation is superior to the Navier-Stokes equations – if we solve the Boltzmann equation, we can indirectly obtain the solution for the Navier-Stokes equations (NSE).

But since the distribution $f(\boldsymbol{x}, \boldsymbol{u}, t)$ is the function of seven variables, it is difficult equation to solve analytically. Surprisingly, the numerical method for the Boltzmann equation turns out to be quiet simple – both to implement and parallelise, and it results in the exact advection (as opposed to NSE solvers, that have major difficulty with the advection term).

This numerical scheme evolved from the lattice-gas cellular automata that are the focus of our thesis, so we will explain it along the lines of LGCA, although explanation from various perspectives is possible.

7.1 Basics

The basic quantity for LBM is the velocity distribution $f_i(\boldsymbol{r}, t)$. Using the probability density function 6.3, it can be defined as

$$f_i(\boldsymbol{r}, t) = f_1(\boldsymbol{r}, \boldsymbol{c_i}, t).$$

and represents the density of the particles, but only at the discrete time steps and points of the lattice.

The LBM lattice is often denoted by the symbol $DdQq$, where d is the dimension of the lattice, and q is the number of lattice vectors, e.g. $D1Q3$, $D2Q9$ or $D3Q19$, which is the most common 3D variant with 19 lattice vectors. Interestingly, it is the projection of the four dimensional FCHC lattice that we already described, see figure 4.1.

The mass and momentum density at the discrete point $(\boldsymbol{r}, t)$ is given by

$$\rho(\boldsymbol{r}, t) = \sum_i f_i(\boldsymbol{r}, t) \tag{7.1}$$

and

$$\boldsymbol{j}(\boldsymbol{r}, t) = \sum_i c_i f_i(\boldsymbol{r}, t), \tag{7.2}$$

which correspond to 3.6 and 3.7, that we derived for the FHP model.

For isotermal lattice Boltzmann equation, the speed of sound is determined by the simple formula

$$p = c_s^2 \rho,$$

and for the lattice sets we introduced above, it is

$$c_s^2 = \frac{1}{3}\frac{\Delta x^2}{\Delta t^2},$$

It is a good custom to set the lattice spacing $\Delta x = 1$ and time step $\Delta t = 1$, and rescale the result to the physical units in demand.

The Boltzmann equation, discretized in space, time and velocity space 6.10 reads

$$f_i(\boldsymbol{x} + \boldsymbol{c_i}\Delta t, t + \Delta t) = f_i(\boldsymbol{x}, t) + \Omega_i(\boldsymbol{x}, t), \tag{7.3}$$

which states that particles from the population $f_i(\boldsymbol{x}, t)$ move to the point $\boldsymbol{x} + \boldsymbol{c_i}$ at the next time step. Before that, particles are affected by the collision operator Ω that redistributes particles among the populations f_i. Notice that this corresponds to the equation 3.2 with one major difference – in FHP the respective quantities were boolean (empty cell or particle inside), in LBM, these are continuous values of particle densities that collide and propagate.

The LBM version of the BGK collision operator 6.22 is

$$\Omega_i(f) = -\frac{f_i - f_i^{eq}}{\tau}\Delta t, \tag{7.4}$$

and gives us a good intuition about what collisions do – the population numbers relax towards the equilibrium values at a rate given by τ. The equilibrium distribution is given by

$$f_i^{eq}(\boldsymbol{x}, t) = w_i\rho\Big(1 + \frac{\boldsymbol{u}\cdot\boldsymbol{c_i}}{c_s^2} + \frac{(\boldsymbol{u}\cdot\boldsymbol{c_i})^2}{2c_s^4} - \frac{\boldsymbol{u}\cdot\boldsymbol{u}}{c_s^2}\Big), \tag{7.5}$$

where w_i are the weights specific to the LBM lattice used, $\boldsymbol{u} = \frac{\boldsymbol{j}}{\rho}$ with ρ and $\boldsymbol{j}$ given by 7.1 and 7.2.

In the previous chapter, we established the link between Boltzmann and Navier-Stokes equations, and derived the expressions for the shear viscosity 6.26 and the stress tensor 6.25. Their discretized LBM variants are

$$\eta = c_s^2\big(\tau - \frac{\Delta t}{2}\big)$$

and

$$\Pi_{\alpha\beta} \approx -\big(1 - \frac{\tau}{2\tau}\big)\sum_i c_{i\alpha}c_{i\beta}(f_i - f_i^{eq}).$$

7.2 Collision and propagation

Analogously to lattice-gas cellular automata, the update given by the Boltzmann equation 7.3 can be decomposed to

1. **Collision**

$$f_i^*(\boldsymbol{x}, t) = f_i(\boldsymbol{x}, t) - \frac{\Delta t}{\tau}\big(f_i(\boldsymbol{x}, t) - f_i^{eq}(\boldsymbol{x}, t)\big) \tag{7.6}$$

   where f_i^* is the particle distribution after the collision.

2. **Propagation**

$$f_i(\boldsymbol{x} + \boldsymbol{c_i}\Delta t, t + \Delta t) = f_i^*(\boldsymbol{x}, t).$$

Less formally, the update step can be described in the few points:

1. Use f_i to compute ρ and $\boldsymbol{j}$ via 7.1 and 7.2.
2. Compute Maxwell-Boltzmann distribution f_i^{eq} via 7.5.
3. Carry out the collision and the propagation.

7.3 Conclusion

We are tempted to say that LBM is the the lattice-gas cellular automaton with continuous values of the cells and collision operator given by 7.4. Physically, however, this is a huge leap and they are not cellular automata in the true sense. The lattice-gas cellular automata are the microscopic toy models of the fluid, but due to their symmetries, realistic behavior emerges on the macroscopic level, while Lattice-Boltzmann method should be viewed as a numerical scheme for solving the Boltzmann equation. Thus it is beyond the scope of this thesis, and concludes our introduction to cellular automata.

Part II

Implementation and applications

8. Practical part

In the theoretical part of the thesis, we have introduced the notion of cellular automaton and basic models of lattice-gas cellular automata that mimic the flow of physical fluids governed by the Euler equations and, to certain extent, the Navier-Stokes equations. We have described the algorithms behind the microdynamics of these models and emphasized the importance of the symmetries of the lattice for reproducing correct macroscopic behavior.

In this part, we implement the aforementioned algorithms in the language C++ and visualize the results using the *Matplotlib*, *Mathematica* and *GNU*plot. The original results of this thesis are summarized in the points to follow.

1. To motivate the applications of cellular automata, we implemented 1D and 2D cellular automata that exhibit wide range of behavior. These results were already included in the theoretical part and will not be discussed further.

2. We implement FCHC and PI – the two distinct 3D LGCA introduced in theoretical part. These implementations are based only on the schemes presented in theoretical part.

3. We propose and implement Pair-interaction automaton with non-deterministic collision rule. We compare it with its classical variant on the "exploding cube", Taylor-Green vortex decay and simulation of the fully developed turbulence.

4. We inspect the flow around the obstacles.

5. We measure parallel scaling performance of our implementation.

6. We study fully-developed turbulent flow and compare its statistical properties to the Kolmogorov–Obuchov K41 theory [7].

9. Implementation

According to [2], the most effective language for the implementation of LGCA is C, with slightly better performance than Fortran. In our implementation, we could not resist to use some comfortable features of C++, but we have sticked to the low-level C-style of programming.

9.1 Parallelization

Languages C and C++ offer three widely used interfaces for parallelization:

1. OpenMP
2. MPI
3. Cuda

MPI was our hot candidate from the beginning and we have even used it on the $1D$ cellular automaton with range 2. But in the end, we concluded that OpenMP would be more appropriate choice. Since we have opporunity to use clusters at Metacenter (the network of academic clusters in the Czech republic) we expected we could employ huge amount of processors, if we use MPI and compute on various clusters parallely. But we learnt that in the practice we can hardly employ more then two hundreds of processors. Using OpenMP, we can run the computation on SMP machine with 64 processors, which is not significantly lower.

The drawbacks of MPI are obvious, it requires vast modification of the code, contrary to OpenMP that usually requires only several lines of code to add.

Moreover, parallelization with OpenMP can be easily included to Python extension module.

In the following section, we provide a useful measure of our parallel implementation, the *scaling efficiency*. It indicates the speedup of the computation with the increasing number of processing units employed. We applied two distinct concepts of scaling efficiency.

9.2 Strong scaling

In strong scaling measurement, we vary the number of processing units, while the problem size stays fixed.

The formula is as simple as

$$s_N = \frac{t_1}{N t_N} 100\%,$$

where t_1 is a time to finish the work with one processing unit and t_N is time to finish the same amount of work with N processing units.

The idea of strong scaling is to find a "sweet spot" – optimal number of computational units to compensate for the overhead of parrallelisation.

9.3 Weak scaling

In this measurement, we vary number of processing units, but we vary also the size of the problem accordingly, so that amount of work per one processing unit is constant.

The weak scaling formula reads

$$w_N = \frac{t_1}{t_N} 100\%,$$

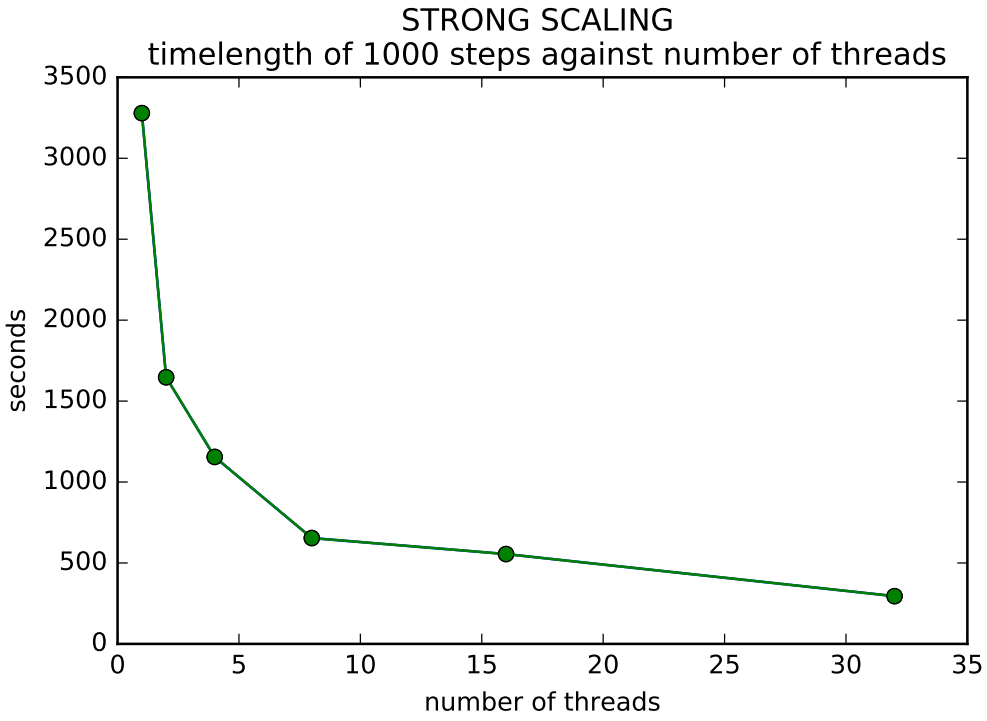
where t_1 is time to complete 1 unit of workload with 1 processing unit, while t_N is time to complete N units of workload with N processing units.

Therefore, the weak scaling tells us how requirements on memory bandwidth and other resources depend on the size of the problem.

9.4 Flow on the sphere

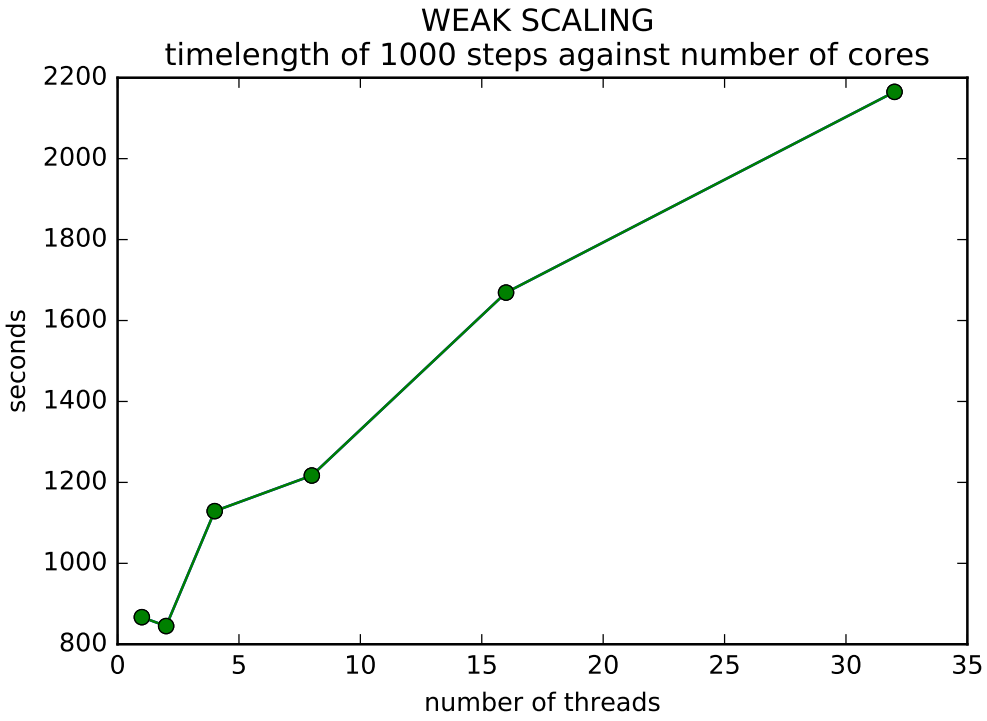
The most time–consuming simulation that we performed was the turbulent flow inside the sphere simulated by Pair-interaction, so the measurement of scaling on this problem is of the highest importance.

The strong scaling was measured on the sphere with diameter $d = 500$ over 3000 time-steps. On the graph below, we plotted time–length of the last 1000 steps against the number of cores employed.



For the weak scaling, the diameter of the sphere varied, so that the volume of lattice was increasing proportionally to the number of threads. However, side of the lattice must be multiple of 10, because macroscopic velocity is computed over the cube with side of 10. Therefore, the number of nodes per one thread differs, but no more then by 5%.

Threads	Diameter [nodes]	Nodes per one thread
1	320	32 768 000
2	400	32 000 000
4	500	31 250 000
8	630	31 255 875
16	800	32 000 000
32	1000	31 250 000



All computations were performed on the cluster **Alfrid** of *University of West Bohemia*.

Alfrid cluster	
Architecture	x86_64
CPU op-mode(s)	32-bit, 64-bit
Byte Order	Little Endian
CPU(s)	32
On-line CPU(s) list	0-31
Thread(s) per core	2
Core(s) per socket	8
Socket(s)	2
NUMA node(s)	2
Vendor ID	GenuineIntel
CPU family	6
Model	62
Model name	Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz
Stepping	4
CPU MHz	1216.007
CPU max MHz	3400.0000
CPU min MHz	1200.0000
BogoMIPS	5201.43
Virtualization	VT-x
L1d cache	32K
L1i cache	32K
L2 cache	256K
L3 cache	20480K
NUMA node0 CPU(s)	0-7,16-23
NUMA node1 CPU(s)	8-15,24-31

10. Implementation of the Pair-Interaction LGCA in 3D

> *"Programmers waste enormous amounts of time thinking about, or worrying about, the speed of noncritical parts of their programs, and these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered. We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%."*
>
> – Donald Knuth, *Structured programming with GoTo statement*

Inspired by this quote and best programming practice presented in [6] on Lattice-Boltzmann implementation, we did our best to implement functions *collision* and *propagations* efficiently, since they are the core of the simulation and overall performance depends largely on them.

Excerpts of the source code that we present are already parallelized by OpenMP.

10.1 Implementation of the collision algorithm

```
#define A 1
#define B 2
#define C 4
#define D 8
#define E 16
#define F 32
#define G 64
#define H 128

unsigned char cell[8] = {A, B, C, D, E, F, G, H};


/* Node consists of 8 mass bits (char m) and 3x8 momentum bits (char
    p[3]). If there is an obstacle in the node <=> node.o == 1. */
typedef struct
{
   unsigned char m;
   unsigned char p[3];
   int o;
} Node;
// We access the bits of the of the node by bitwise AND opperation
//e.g. node.m & A == 1 means there is particle in the cell A
//      node.p[1] & A == 0 means that this particle has no momentum in
    Y-direction

// 3 directions (X, Y, Z), 4 pair-interactions in each direction, 2
    cells in each pair
```

```
unsigned char Pair[3][4][2] =
{
   {
      {A,E}, {B,F}, {C,G}, {D,H}
   },
   {
      {A,C}, {B,D}, {E,G}, {F,H}
   },
   {
      {A,B}, {C,D}, {E,F}, {G,H}
   }
};

// The collision algorithm
void collision(Node &node)
{
   //d,u ... index for downer and upper momentum
   int d; //index of momentum in direction of PI
   int u; //index of momentum orthogonal to direction of PI

   // l, r ... left/right cell in the pair
   // ml, mr ... mass bits in the pair
   // lu, ld, ri, ru, rd, ri ... momenta bits in the pair
   unsigned char l, r, ml, mr, lu, ld, ru, rd, li, ri;

   for (int i = 0; i < 3; ++i)
   {
      d = (i + 1) % 3;
      u = (i + 2) % 3;
      for (int j = 0; j < 4; ++j)
      {
         l = Pair[i][j][0];
         r = Pair[i][j][1];

         ml = node.m&l;
         mr = node.m&r;
         ld = node.p[d] & l;
         lu = node.p[u] & l;
         li = node.p[i] & l;
         rd = node.p[d] & r;
         ru = node.p[u] & r;
         ri = node.p[i] & r;

         /* PAIR INTERACTIONS */
         if (!ml && !mr)
            continue;
         if (ml && mr)
         {
            // alternate momenta in direction of the pair-interaction
            if (!ld && !rd)
            {
```

```
            node.p[d] |= l;
            node.p[d] |= r;
         }
         else if (ld && rd)
         {
            node.p[d] ^= l;
            node.p[d] ^= r;
         }
         // alternate momenta in all other directions
         if (lu && !ru)
         {
            node.p[u] ^= l;
            node.p[u] |= r;
         }
         else if (!lu && ru)
         {
            node.p[u] |= l;
            node.p[u] ^= r;
         }
         if (li && !ri)
         {
            node.p[i] ^= l;
            node.p[i] |= r;
         }
         else if (!li && ri)
         {
            node.p[i] |= l;
            node.p[i] ^= r;
         }
      }
      else if (!ml && !rd)
      {
         node.m |= l;
         node.m ^= r;
         if (ru)
         {
            node.p[u] |= l;
            node.p[u] ^= r;
         }
         if (ri)
         {
            node.p[i] |= l;
            node.p[i] ^= r;
         }
      }
      else if (!mr && !ld)
      {
         node.m |= r;
         node.m ^= l;
         if (lu)
         {
```

```
            node.p[u] |= r;
            node.p[u] ^= l;
         }
         if (li)
         {
            node.p[i] |= r;
            node.p[i] ^= l;
         }
      }
   }
}

// 'Collision' runs over the lattice and calls function 'collision' on
    each node.
void Collision(Node***array, int X, int Y, int Z, int start)
{
   int x,y,z;

#pragma omp parallel for private (x,y,z)
   for (x = start; x < X; x+=2)
   {
      for(y = start; y < Y; y+=2)
      {
         for(z = start; z < Z; z+=2)
         {
            // if the node is obstacle, skip it
            if(array[x][y][z].o)
               continue;
            collision(array[x][y][z]);
         }
      }
   }
}
```

10.2 Implementation of the Propagation

As we explained in the teoretical part, the PI-3D lattice splits into 4 sub-lattices, that are not connected. We are using only one of these sub-lattices. At even times $t = 2, 4, ...$ only nodes with even indices are populated, and at odd times, nodes with odd indices are populated.

Theoretically, this setting guarantees that spurious Zannetti's invariants are not present ([12]), practically, we can use single data structure to store the lattice before and after propagation. Comparing to other lattices (FHP, FCHC, DdQq), this provides additional speed-up by lowering memory bandwidth and saving other resources.

The excerpts of the source code governing Propagation follows.

```
/* Group of cells that propagates in positive X-direction, Y-direction
```

```
    and Z-direction. */
#define dirX (B+D+F+H)
#define dirY (E+F+G+H)
#define dirZ (A+B+E+F)
/* For example, A & dirX == 0, so particle from A propagates in
    negative X direction,
    but A & dirZ == 1, so it propagates in positive Z direction. */

/* At start == 0, particles are streamed from even nodes
    ([0,2,124]...) to odd nodes([1,1,123]...)
   At start == 1, from odd to even nodes. */
void Propagation(Node***array, int X, int Y, int Z, int start)
{

#pragma omp parallel for
   for (int x = start; x < X; x += 2)
   {
      for (int y = start; y < Y; y += 2)
      {
         for (int z = start; z < Z; z += 2)
         {
            // for cells in the node
            for (int k = 1; k <= H; k <<= 1)
            {
               //if there is particle in the cell, we propagate it
               if (array[x][y][z].m & k)
               {
                  int xN = k & dirX ? (x + 1) % X : (x - 1 + X) % X;
                  int yN = k & dirY ? (y + 1) % Y : (y - 1 + Y) % Y;
                  int zN = k & dirZ ? (z + 1) % Z : (z - 1 + Z) % Z;

#pragma omp atomic
                  array[xN][yN][zN].m |= k & array[x][y][z].m;
#pragma omp atomic
                  array[xN][yN][zN].p[0] |= k & array[x][y][z].p[0];
#pragma omp atomic
                  array[xN][yN][zN].p[1] |= k & array[x][y][z].p[1];
#pragma omp atomic
                  array[xN][yN][zN].p[2] |= k & array[x][y][z].p[2];
               }
            }
            //we set the old node to 0
            array[x][y][z].m = 0;
            array[x][y][z].p[0] = 0;
            array[x][y][z].p[1] = 0;
            array[x][y][z].p[2] = 0;
         }
      }
   }
}
```

11. Non-deterministic PI

Pair-Interaction automaton that Nasilowski proposed is deterministic, and it might be considered to be its advantage.

The pair-interaction leads

The collisions usually lead to the maximal change of the state, which minimizes the viscosity, as we previously discussed. Moreover, it offers theoretical ground for using Gibbs distribution 5.11 in derivation of hydrodynamic equations.

But let us explore following examples. First, consider node with one standing particle.

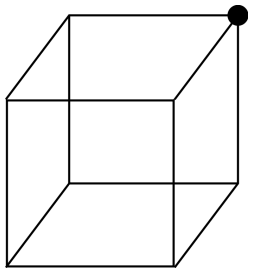


Figure 11.1: Node before collision

By deterministic pair-interactions in X,Y and Z direction, it is always resolved into the state

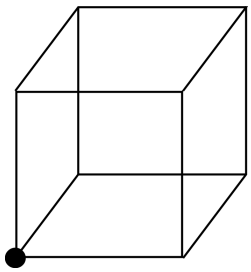


Figure 11.2: Node after deterministic collision

But it is no better then

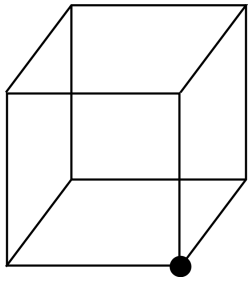


Figure 11.3: Another acceptable state after collision

or any other state with one standing particle.

Even better example is the node with two particles with momenta in X direction.

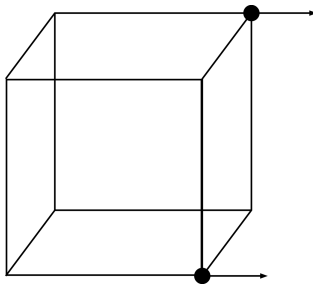


Figure 11.4: Node before collision, and after deterministic collision

Deterministic collision do not change its state (pair interaction in Y direction, followed by pair interaction in Z direction leads to the same state for this particular example).

However, collision that would be non-deterministic can lead to the following state

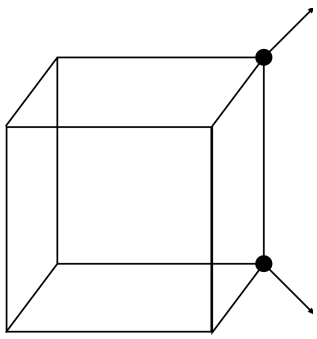


Figure 11.5: Node after non-deterministic collision

where both particles gained additional momenta, one in Z, and other in -Z direction. We could continue with the examples showing deterministic PI do not realize all available states, although some of them are more desirable.

On the other hand, non-deterministic model, has an obvious drawback. Generation of the random numbers significantly slows the collisions.

To speed up the generation of random number, we hard-coded *Marsaglia's xorshift* generator [13], which significantly improved performance comparing to *rand*() function of C. It may not be appropriate for serious Monte-Carlo simulations, but it is more then enough for our purpose.

```
static unsigned long x=123456789, y=362436069, z=521288629;

unsigned long xorshf96(void) {
   unsigned long t;
   x ^= x << 16;
   x ^= x >> 5;
   x ^= x << 1;
    t = x;
```

```
   x = y;
   y = z;
   z = t ^ x ^ y;

   return z;
}
```

11.1 Non-deterministic collision

At the beginning of collision we generate random number by `long rand = xorshf96()`. Before each pair-interaction, we bit-wise shift this number `rand >>= 1` , and read the first bit. If it is 1, pair-interaction is performed, otherwise it is skipped.

```
void collision(Node &node)
{
   int d, u;
   unsigned char l, r, ml, mr, lu, ld, ru, rd, li, ri;

   // we generate random long
   long rand = xorshf96();

   for (int i = 0; i < 3; ++i)
   {
      d = (i + 1) % 3;
      u = (i + 2) % 3;
      for (int j = 0; j < 4; ++j)
      {
         l = Pair[i][j][0];
         r = Pair[i][j][1];

         ml = node.m&l;
         mr = node.m&r;
         ld = node.p[d] & l;
         lu = node.p[u] & l;
         li = node.p[i] & l;
         rd = node.p[d] & r;
         ru = node.p[u] & r;
         ri = node.p[i] & r;

         /* PAIR INTERACTIONS */
         if (!ml && !mr)
            continue;
         if (ml && mr)
         {
            // alternate momenta in direction of the pair-interaction
            if (!ld && !rd)
            {
               //we shift the random number, to the right, and look at
                  the first bit
               rand >>= 1;
```

```
   if(rand & 1)
   {
      //if the first bit is 1, the pair-interaction is
          performed
      node.p[d] |= l;
      node.p[d] |= r;
   }
}
else if (ld && rd)
{
   rand >>= 1;
   if(rand & 1)
   {
      node.p[d] ^= l;
      node.p[d] ^= r;
   }
}
// alternate momenta in all other directions
if (lu && !ru)
{
   rand >>= 1;
   if(rand & 1)
   {
      node.p[u] ^= l;
      node.p[u] |= r;
   }
}
else if (!lu && ru)
{
   rand >>= 1;
   if(rand & 1)
   {
      node.p[u] |= l;
      node.p[u] ^= r;
   }
}
if (li && !ri)
{
   rand >>= 1;
   if(rand & 1)
      continue;
   node.p[i] ^= l;
   node.p[i] |= r;
}
else if (!li && ri)
{
   rand >>= 1;
   if(rand & 1)
      continue;
   node.p[i] |= l;
   node.p[i] ^= r;
```

```
            }
         }
         else if (!ml && !rd)
         {
            rand >>= 1;
            if(rand & 1)
               continue;
            node.m |= l;
            node.m ^= r;
            if (ru)
            {
               node.p[u] |= l;
               node.p[u] ^= r;
            }
            if (ri)
            {
               node.p[i] |= l;
               node.p[i] ^= r;
            }
         }
         else if (!mr && !ld)
         {
            rand >>= 1;
            if(rand & 1)
               continue;
            node.m |= r;
            node.m ^= l;
            if (lu)
            {
               node.p[u] |= r;
               node.p[u] ^= l;
            }
            if (li)
            {
               node.p[i] |= r;
               node.p[i] ^= l;
            }
         }
      }
   }
}
```

11.2 Exploding cube

To demonstrate the difference of deterministic and non-deterministic automaton in the crystal form, we simulated the "explosion of the cube".

The simulations were performed on the lattice $240 \times 240 \times 240$ with periodic boundary conditions. Into the cube with side 3 times smaller then the size of lattice (thus containing 1/27 of all nodes), we put particles with zero momentum

in all available cells.

The evolution of the system for both versions of PI is to be seen on the following pictures.

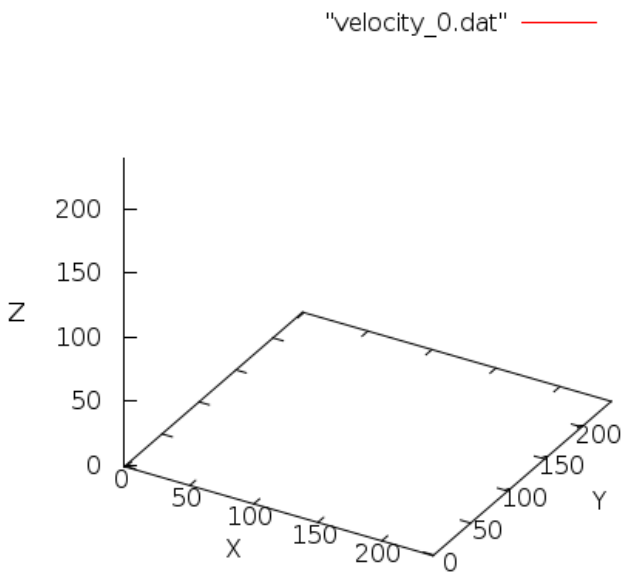


Figure 11.6: Deterministic PI - time 0

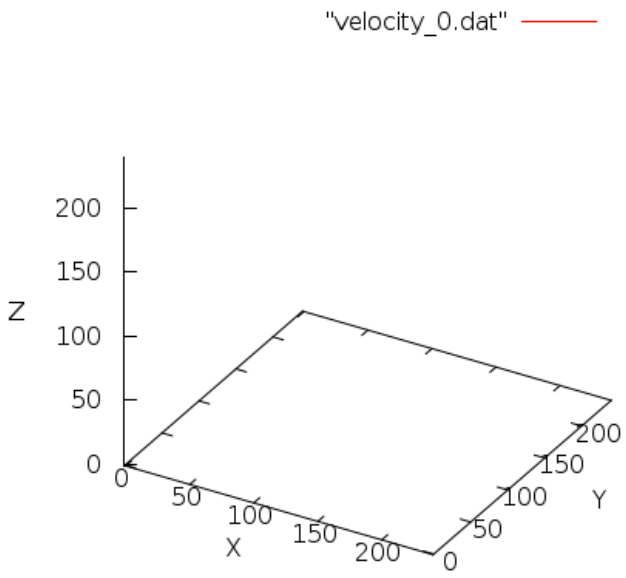


Figure 11.7: Non-deterministic PI - time 0

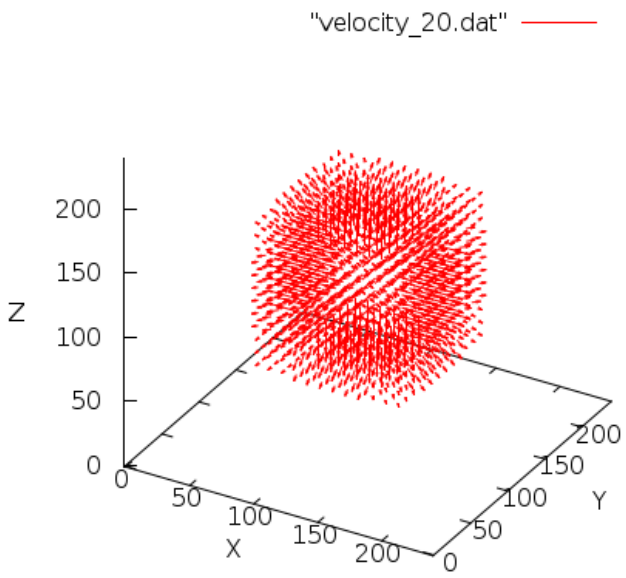


Figure 11.8: Deterministic PI - time 20

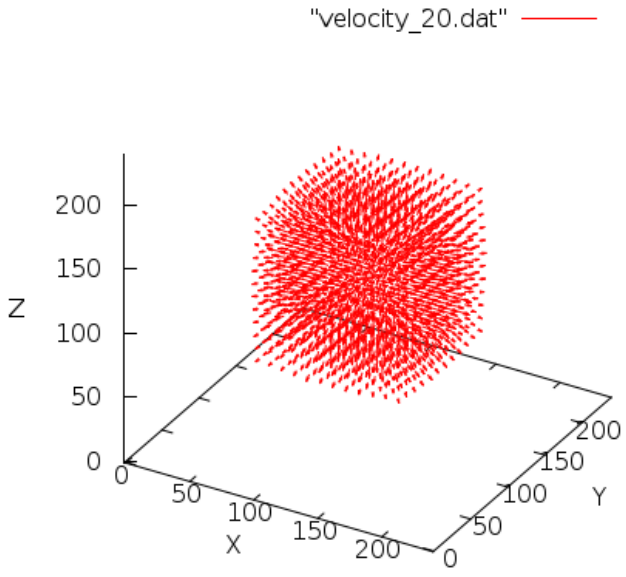


Figure 11.9: Non-deterministic PI - time 20

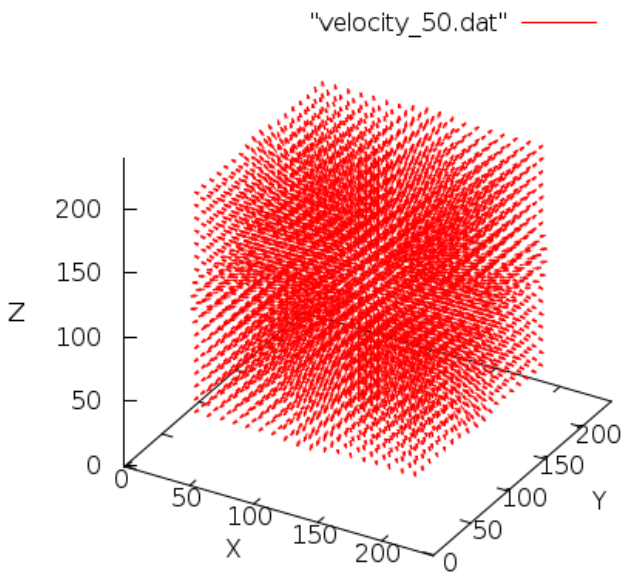


Figure 11.10: Deterministic PI - time 50

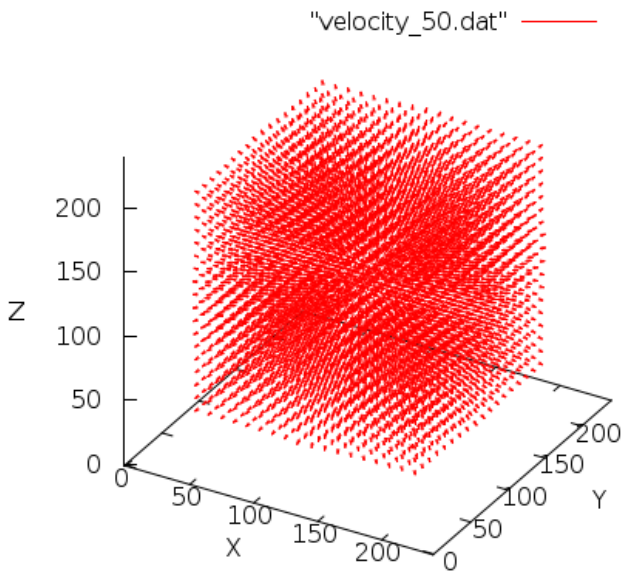


Figure 11.11: Non-deterministic PI - time 50

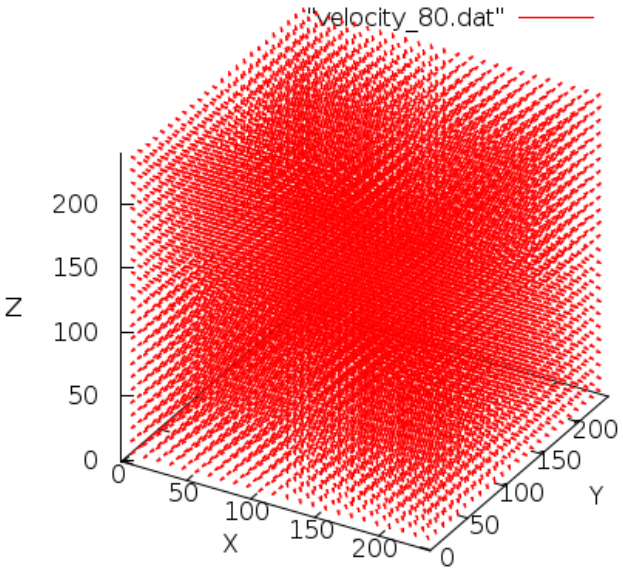


Figure 11.12: Deterministic PI - time 80

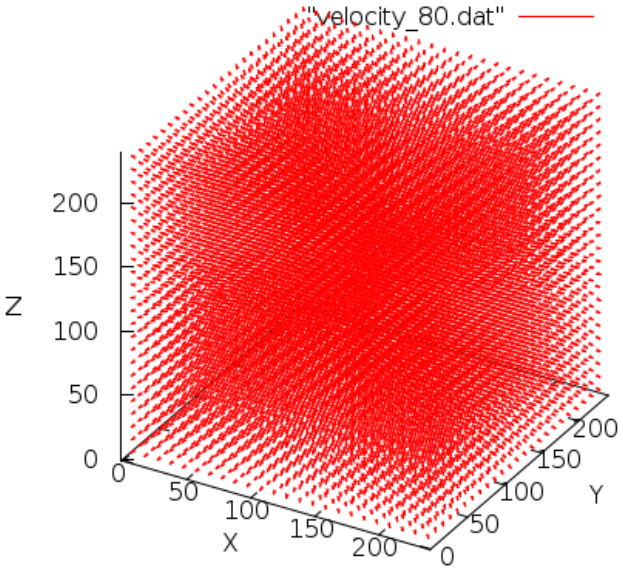


Figure 11.13: Non-deterministic PI - time 80

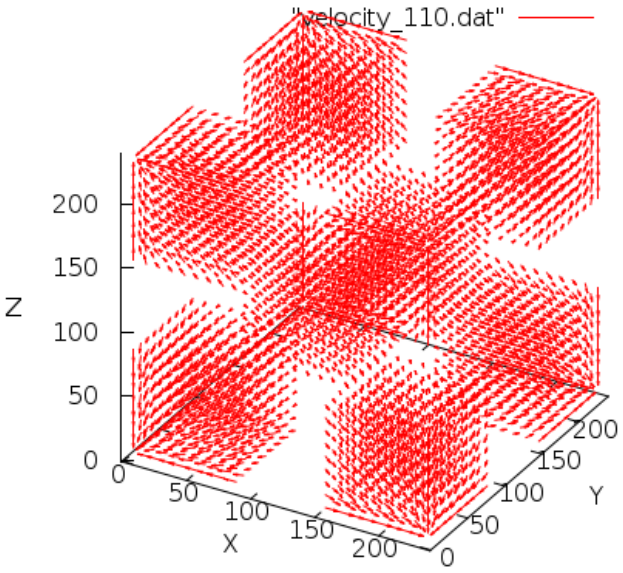


Figure 11.14: Deterministic PI - time 110

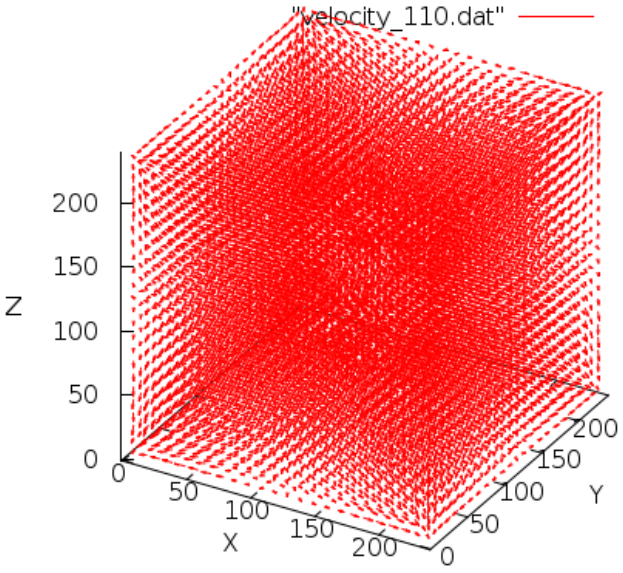


Figure 11.15: Non-deterministic PI - time 110

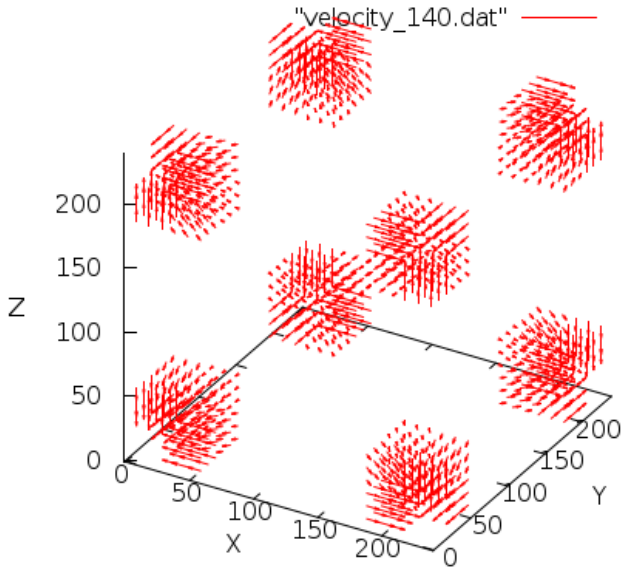


Figure 11.16: Deterministic PI - time 140

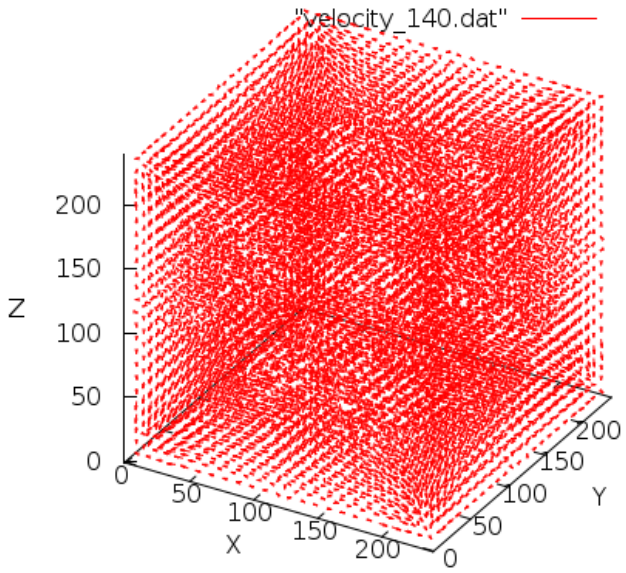


Figure 11.17: Non-deterministic PI - time 140

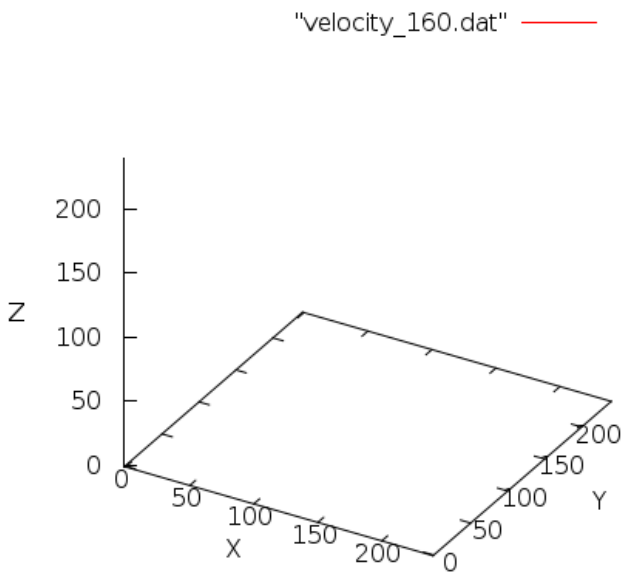


Figure 11.18: Deterministic PI - time 160

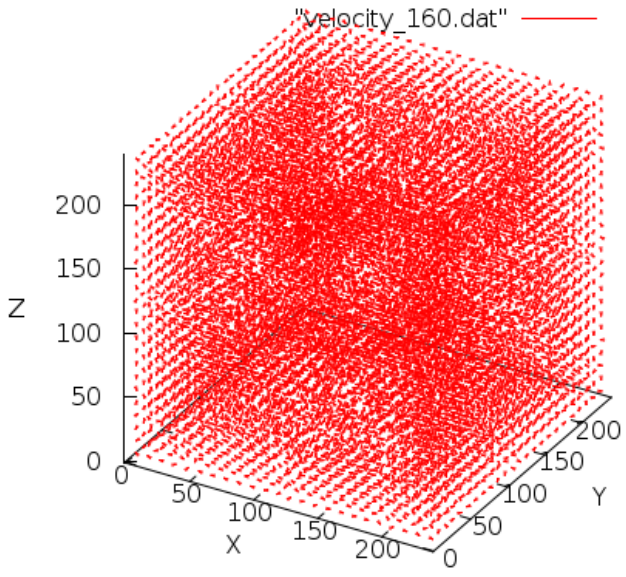


Figure 11.19: Non-deterministic PI - time 160

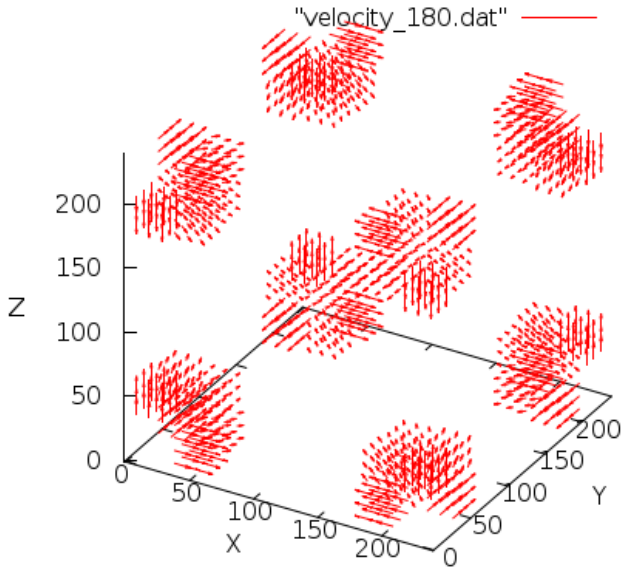


Figure 11.20: Deterministic PI - time 180

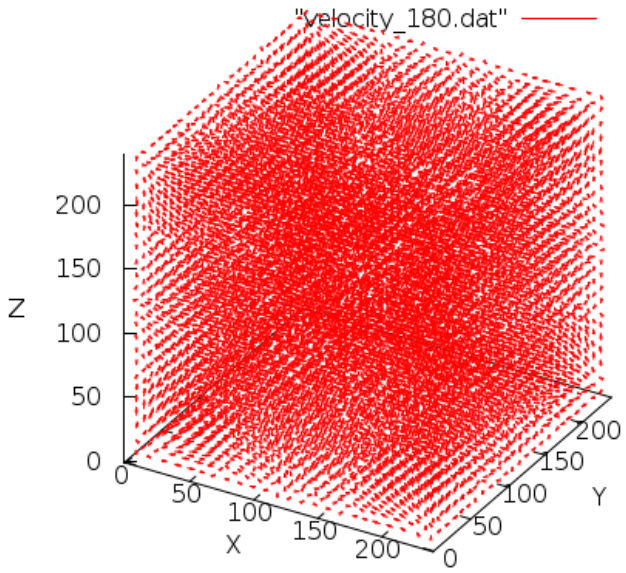


Figure 11.21: Non-deterministic PI - time 180

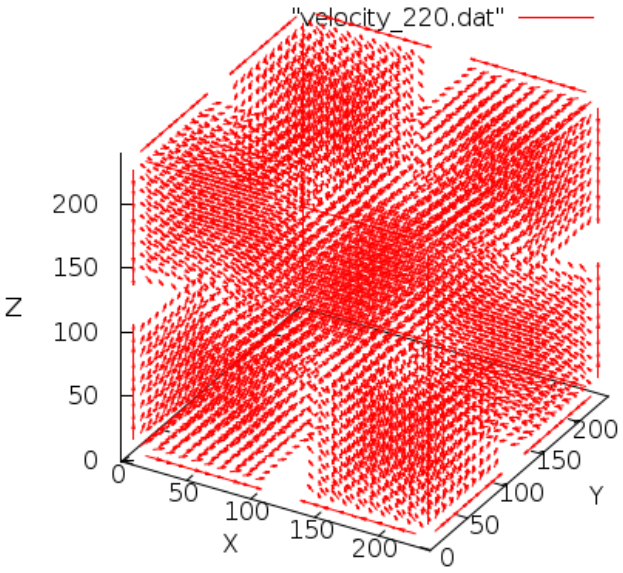


Figure 11.22: Deterministic PI - time 220

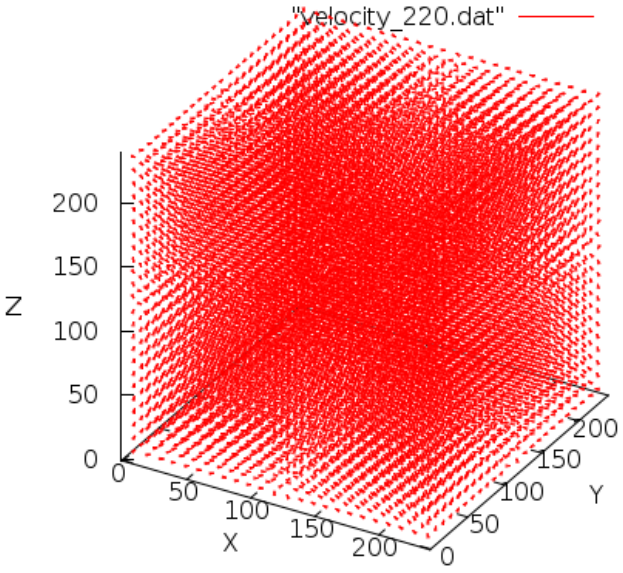


Figure 11.23: Non-deterministic PI - time 220

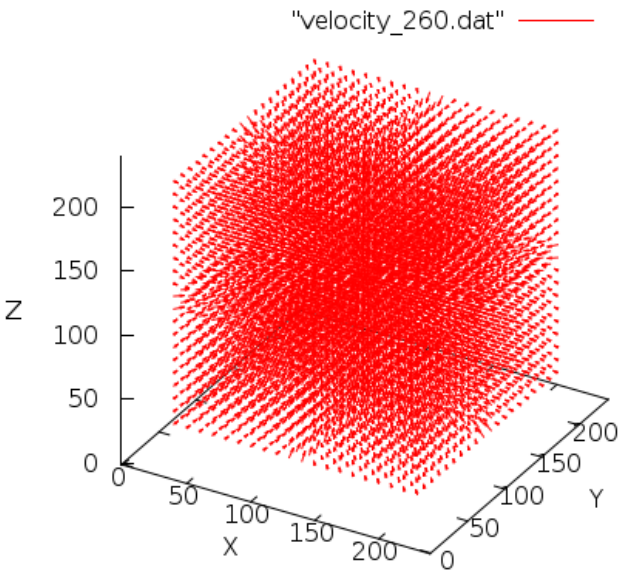


Figure 11.24: Deterministic PI - time 260

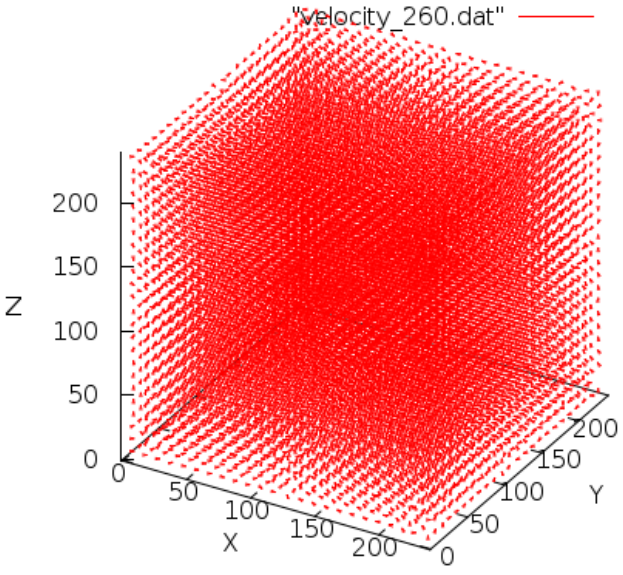


Figure 11.25: Non-deterministic PI - time 260

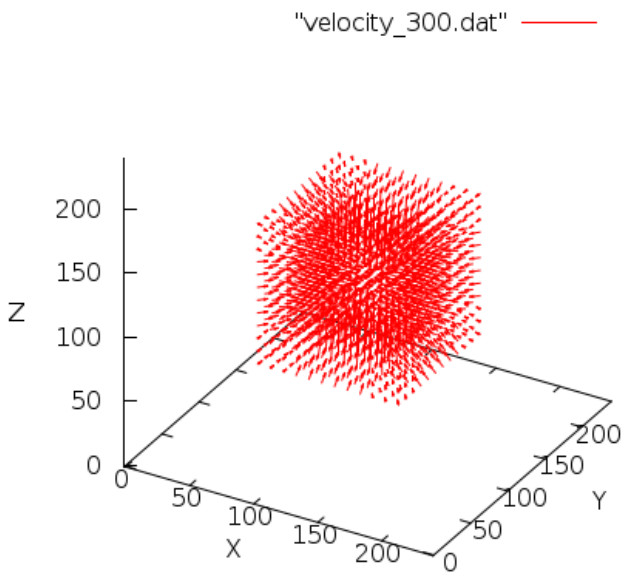


Figure 11.26: Deterministic PI - time 300

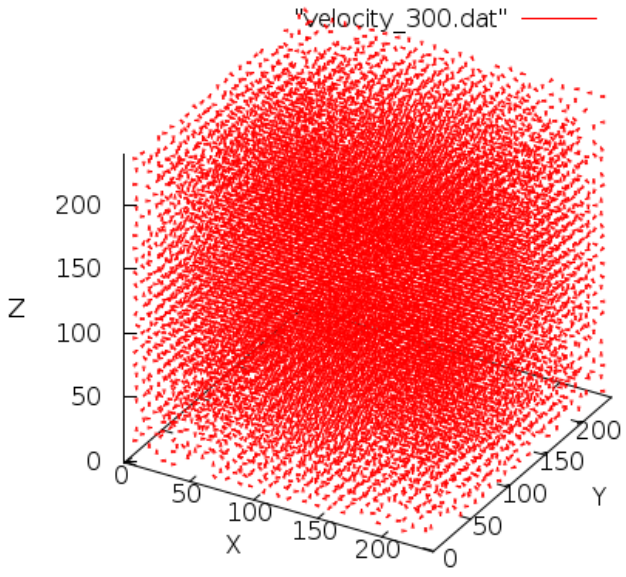


Figure 11.27: Non-deterministic PI - time 300

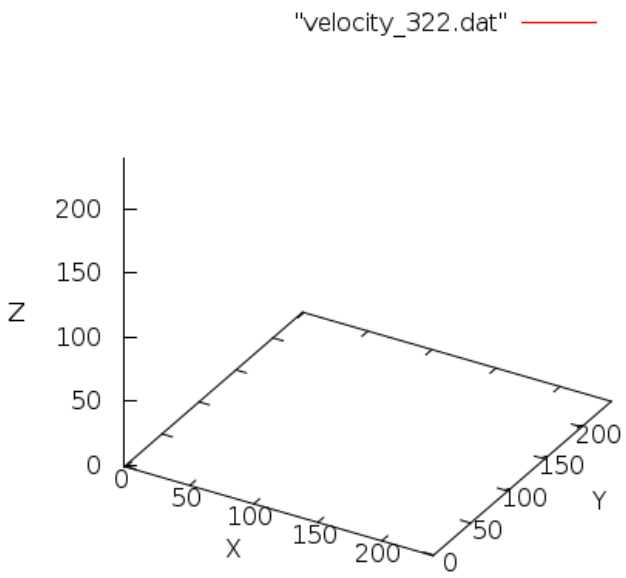


Figure 11.28: Deterministic PI - time 0

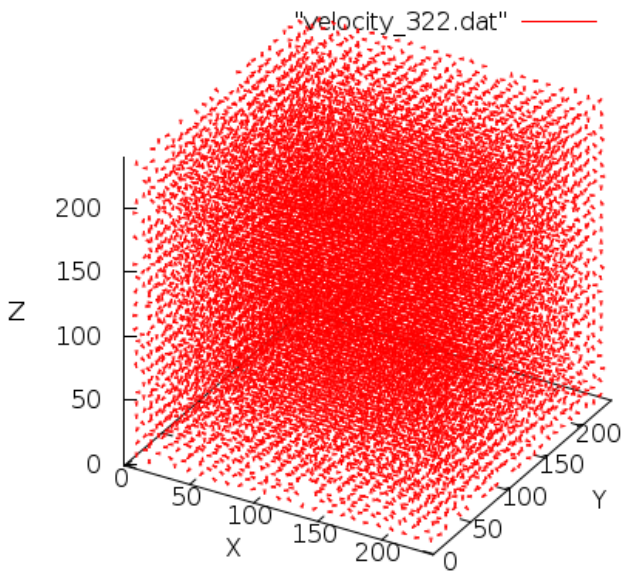


Figure 11.29: Non-deterministic PI - time 0

The patterns that you see for deterministic PI were repeating in the infinite loop. We performed simulation up to 9000 steps and the pattern did not change.

On the other hand, the pattern in the non-deterministic PI broke after the first

explosion already and exhibits the realistic "spilling" of the particles, in contrast to the perfect periodic evolution of deterministic automaton.

12. Taylor-Green vortex decay using Pair-interaction CA

Taylor-Green vortex decay is standard benchmark problem used for evaluation of accuracy of CFD methods. We shall see how deterministic and non-deterministic PI fare on this problem.

The lattice was set-up with the periodic boundary conditions, and with initial velocity field

$$\begin{aligned} v_x &= \sin\left(\frac{2\pi}{I}\,x\right)\,\cos\left(\frac{2\pi}{I}\,y\right)\,\cos\left(\frac{2\pi}{I}\,z\right), \\ v_y &= -\cos\left(\frac{2\pi}{I}\,x\right)\,\sin\left(\frac{2\pi}{I}\,y\right)\,\cos\left(\frac{2\pi}{I}\,z\right), \\ v_z &= 0, \end{aligned} \tag{12.1}$$

that obeys the continuity equation

$$\nabla \cdot \boldsymbol{v} = 0.$$

Functions below set-up the Taylor-Green vortex with the initial conditions 12.1.

```
//sets macroscopic velocity (vx,vy,vz) at (i,j,k)
void set_speed(Node***a, int i, int j, int k, int dx, int dy, int dz,
    double vx, double vy, double vz)
{
   double n = 0;
   int mx = 0;
   int my = 0;
   int mz = 0;

   unsigned char sx = vx >= 0 ? dirX : mirX;
   unsigned char sy = vy >= 0 ? dirY : mirY;
   unsigned char sz = vz >= 0 ? dirZ : mirZ;

   double absx = vx >= 0 ? vx : -vx;
   double absy = vy >= 0 ? vy : -vy;
   double absz = vz >= 0 ? vz : -vz;

   for (int x = i; x < i + dx; x += 2)
   {
      for (int y = j; y < j + dy; y += 2)
      {
         for (int z = k; z < k + dz; z += 2)
         {
            n += 1;
            if (mx / n < absx)
            {
               ++mx;
```

```
            a[x][y][z].m |= sx;
            a[x][y][z].p[0] |= sx;
         }
         if (my / n < absy)
         {
            ++my;
            a[x][y][z].m |= sy;
            a[x][y][z].p[1] |= sy;
         }
         if (mz / n < absz)
         {
            ++mz;
            a[x][y][z].m |= sz;
            a[x][y][z].p[2] |= sz;
         }
      }
   }
}

// sets initial conditions for taylor-green vortex
void taylor_green_vortex(Node***a, int I, int J, int K, int X, int Y,
    int Z, int dx, int dy, int dz)
{
   double ampl = 0.5;
   double kx = 2 * PI / I;
   double ky = 2 * PI / J;
   double kz = 2 * PI / K;

   double ax = 1 * ampl;
   double ay = -1 * ampl;
   double az = 0;

   double vx, vy, vz;

   for (int i = 0; i < I; ++i)
      for (int j = 0; j < J; ++j)
         for (int k = 0; k < K; ++k)
         {
            vx = ax * sin(kx * i) * cos(ky * j) * cos(kz * k);
            vy = ay * cos(kx * i) * sin(ky * j) * cos(kz * k);
            vz = 0;

            set_speed(a, i*dx, j*dy, k*dz, dx, dy, dz, vx, vy, vz);
         }
}
```

12.1 Simulation of the vortex decay by deterministic and non-deterministic PI

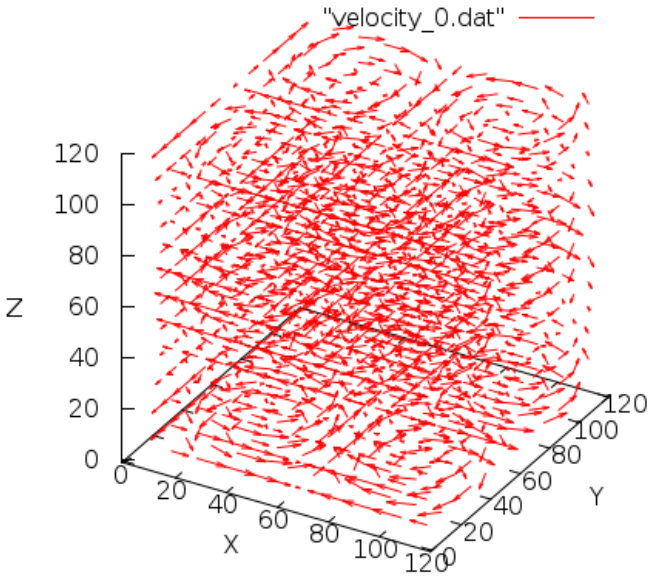


Figure 12.1: Time 0 – deterministic PI

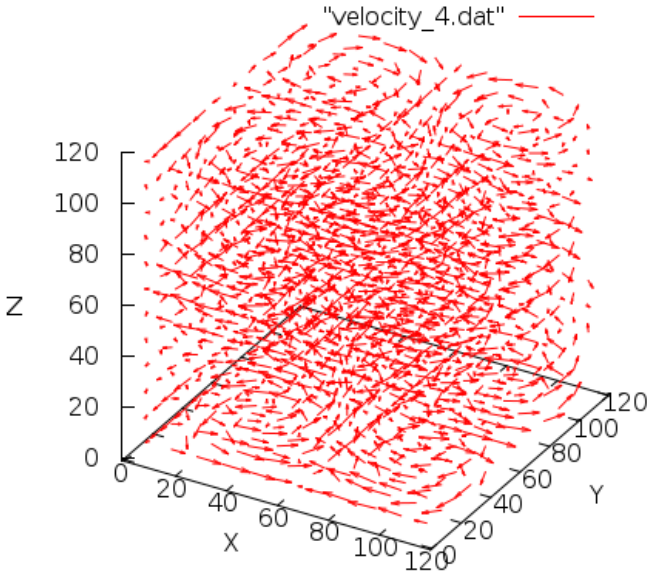


Figure 12.2: Time 4 – deterministic PI

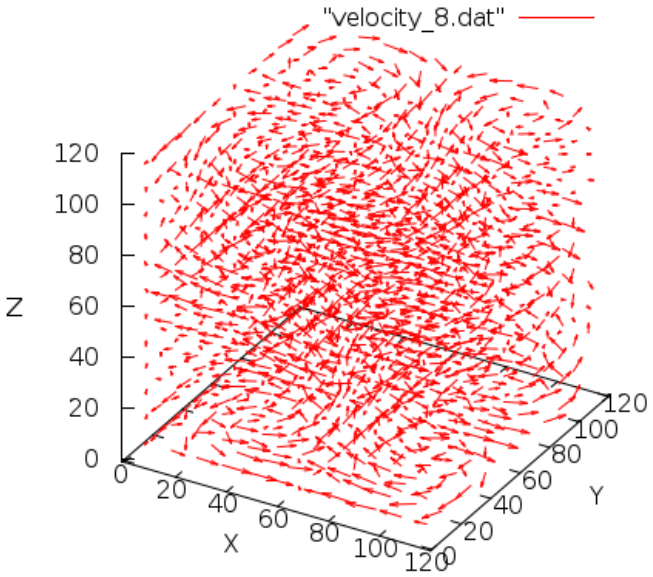


Figure 12.3: Time 8 – deterministic PI

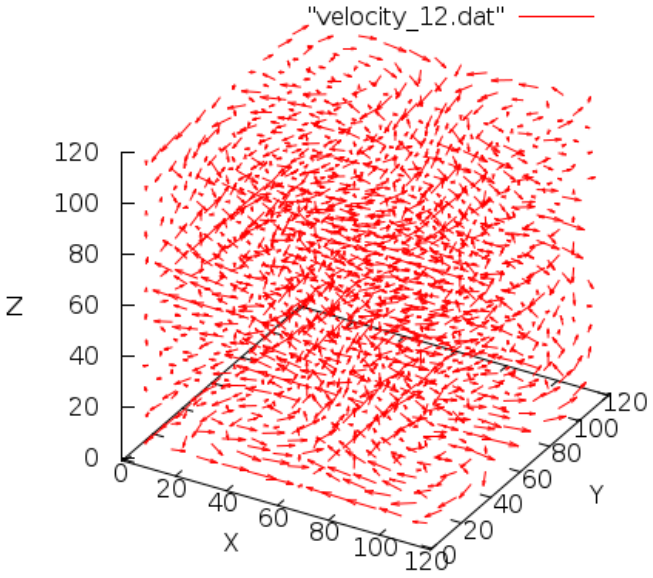


Figure 12.4: Time 12 – deterministic PI

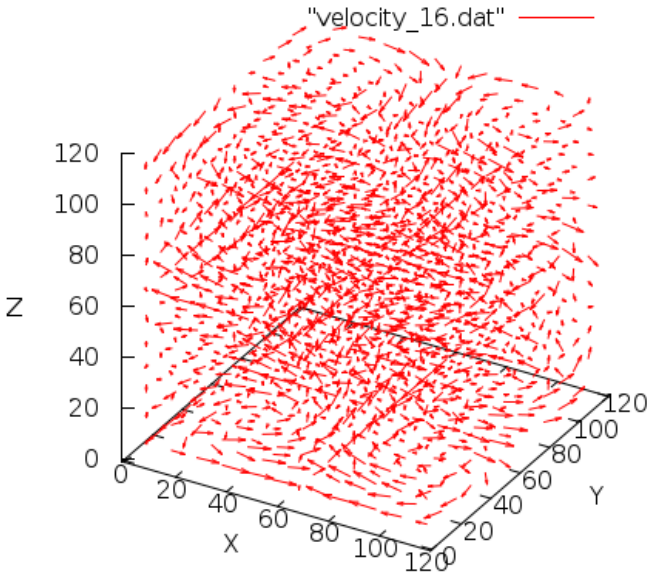


Figure 12.5: Time 16 – deterministic PI

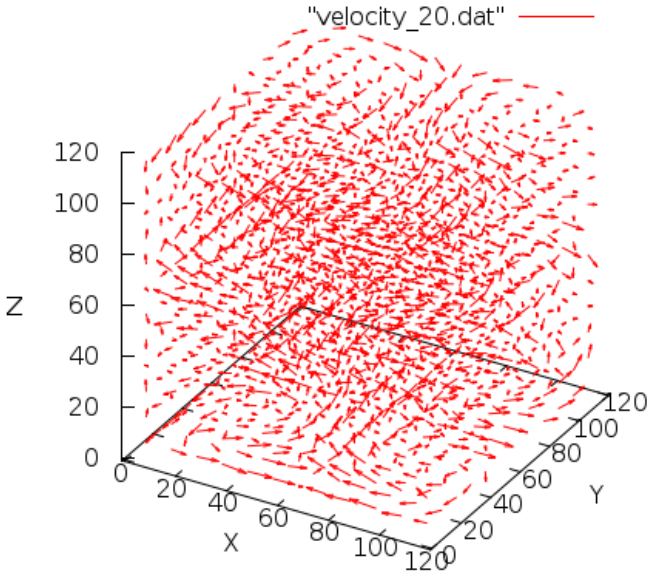


Figure 12.6: Time 20 – deterministic PI

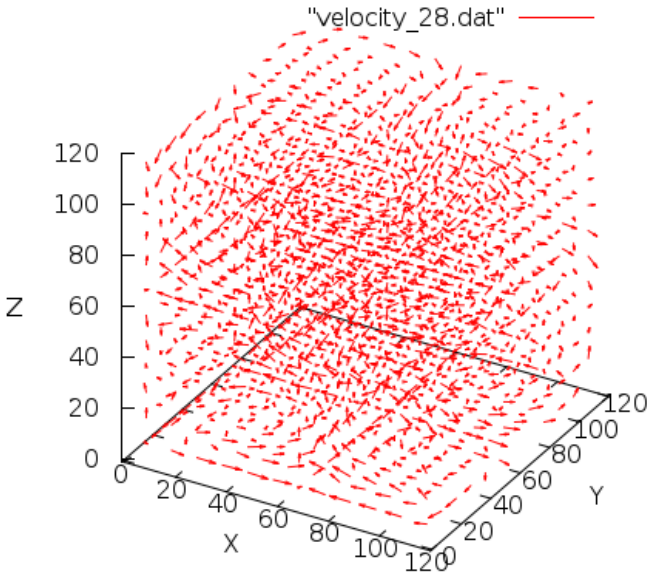


Figure 12.7: Time 28 – deterministic PI

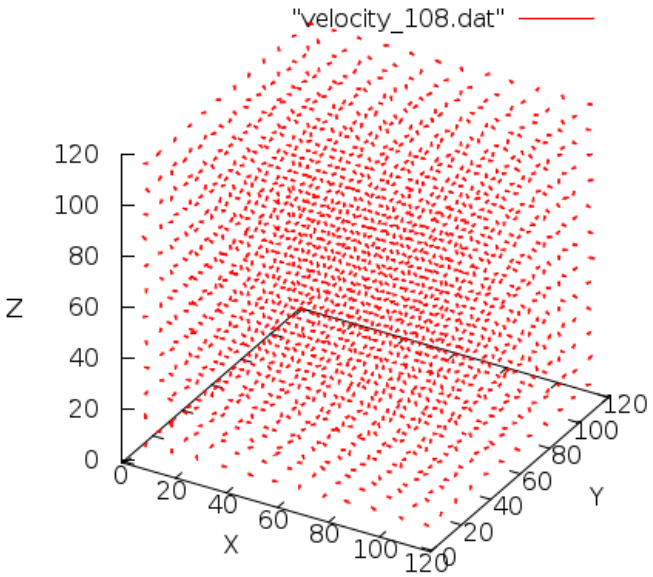


Figure 12.8: Time 108 – deterministic PI

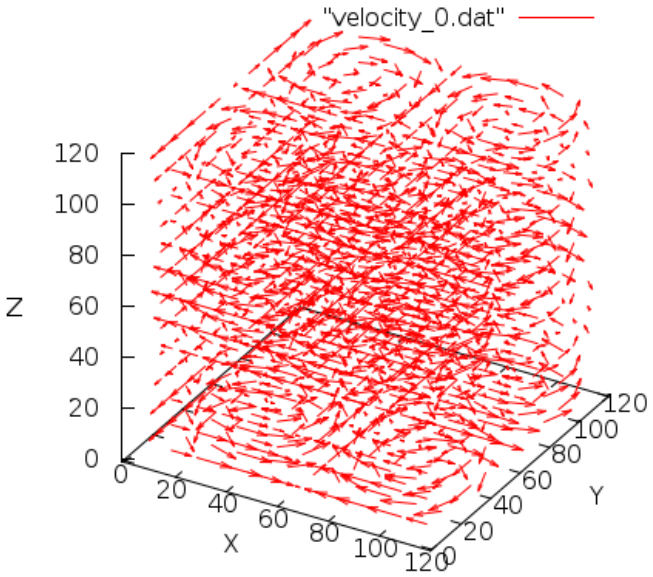


Figure 12.9: Time 0 – NON-deterministic PI

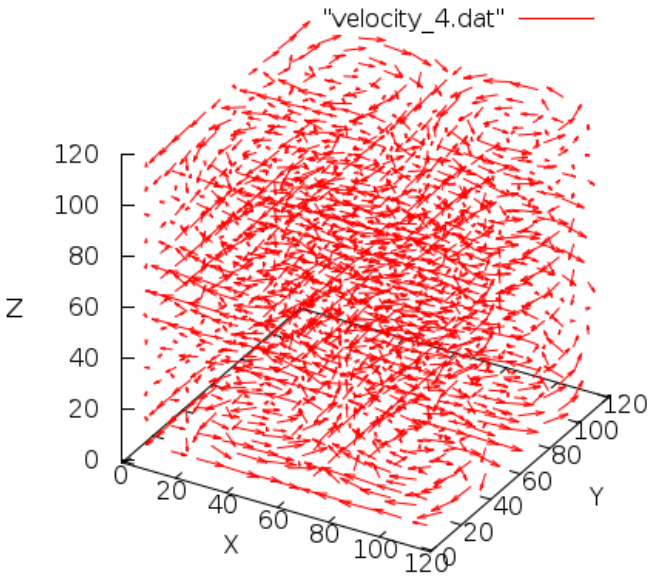


Figure 12.10: Time 4 – NON-deterministic PI

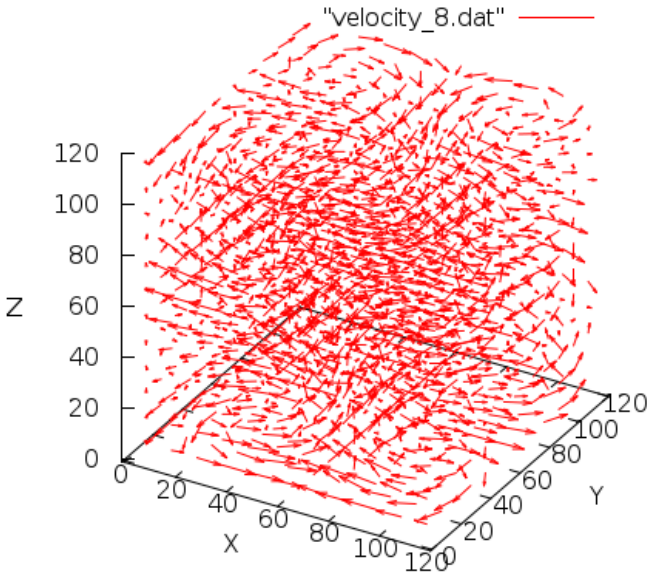


Figure 12.11: Time 8 – NON-deterministic PI

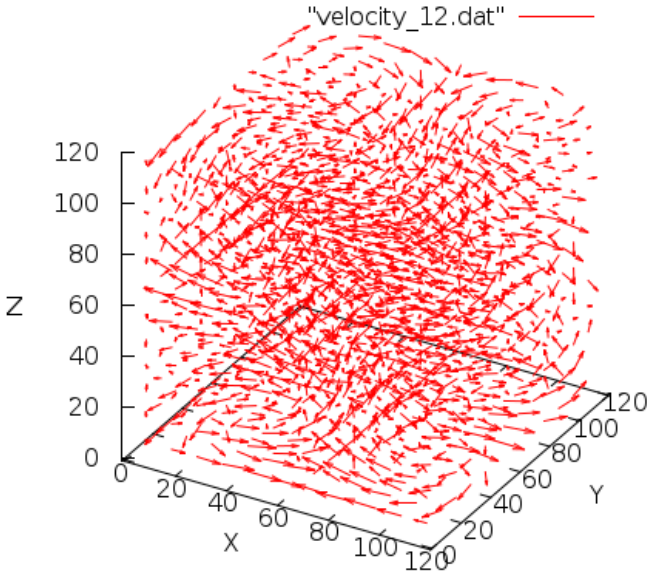


Figure 12.12: Time 12 – NON-deterministic PI

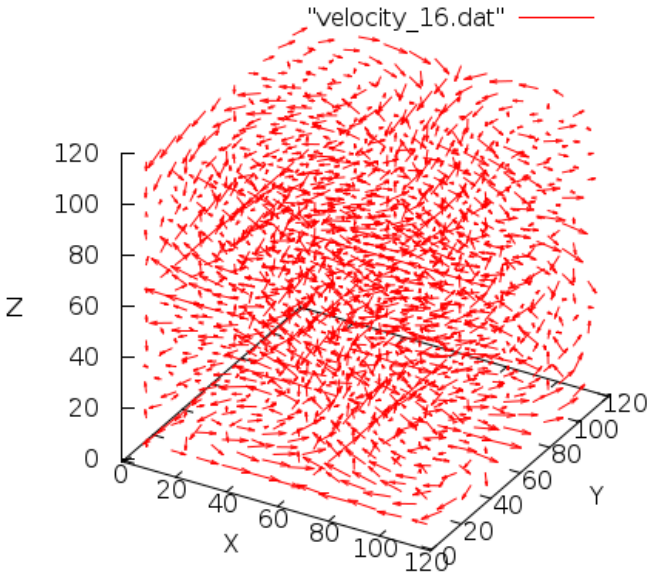


Figure 12.13: Time 16 – NON-deterministic PI

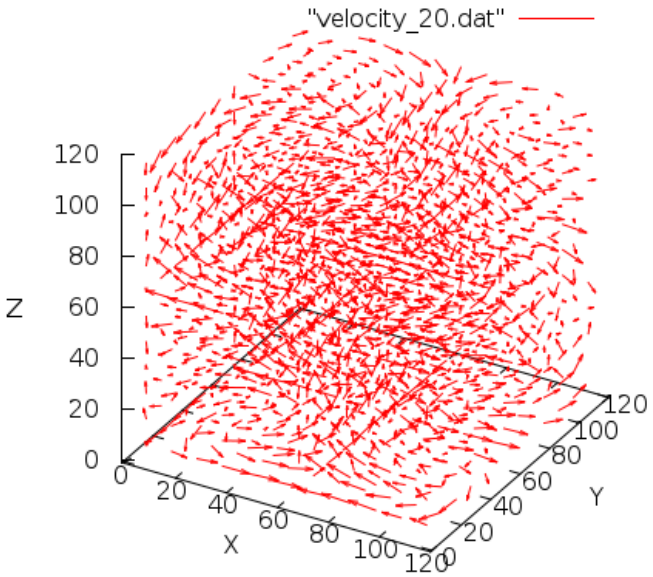


Figure 12.14: Time 20 – NON-deterministic PI

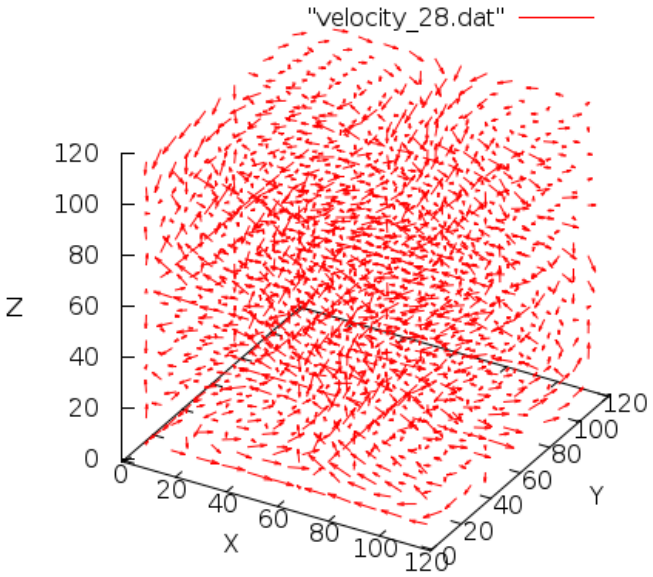


Figure 12.15: Time 28 – NON-deterministic PI

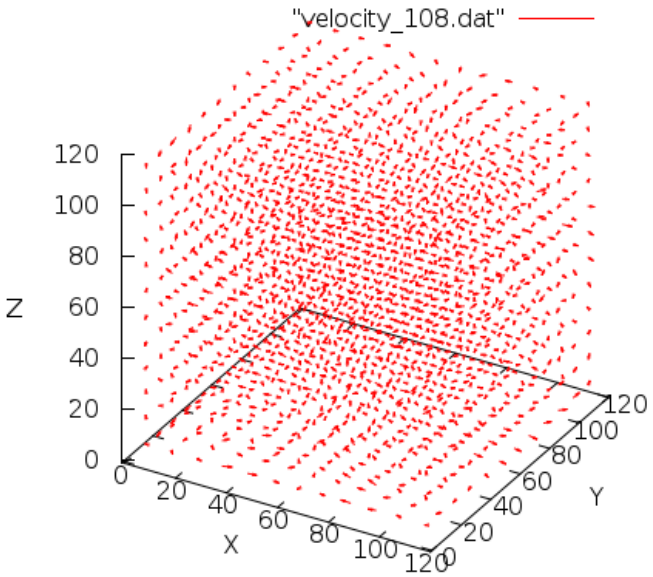


Figure 12.16: Time 108 – NON-deterministic PI

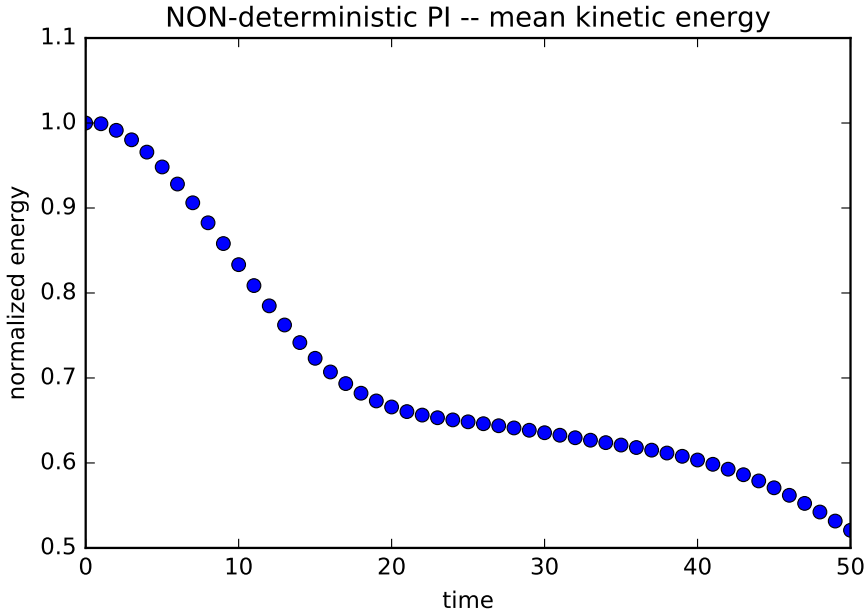
12.2 Dissipation of kinetic energy

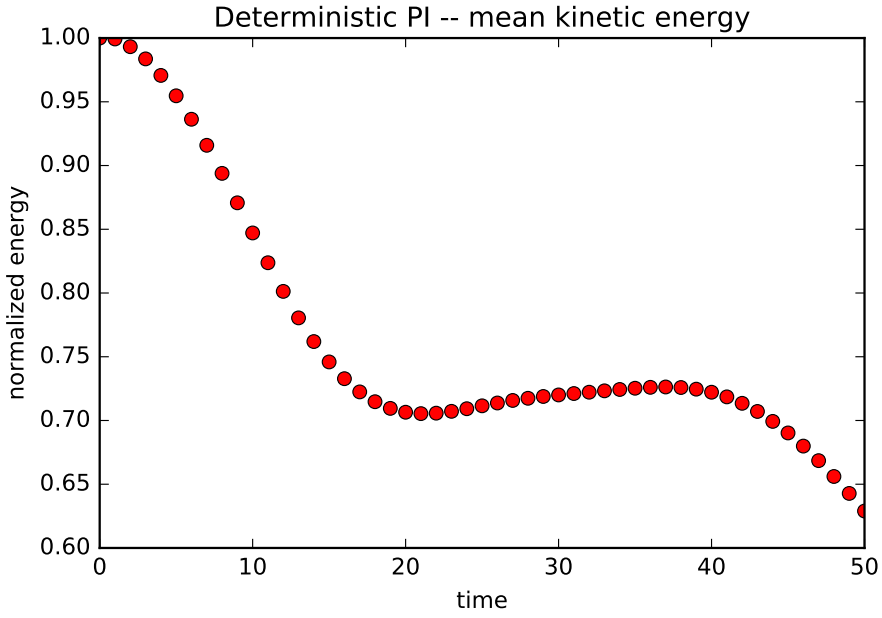
As the diagnostic quantity of our model, we computed mean kinetic energy of the vortex

$$E_k = \int_V \frac{1}{2}\rho u^2 \mathrm{d}V$$

and followed its time evolution. Our results can be compared with the reference solutions computed by nodal Discontinuous Galerkin and Spectral difference methods presented in [14], that used same initial conditions.

The size of the lattice that we used was $1000 \times 1000 \times 1000$ nodes.





In non-deterministic model, kinetic energy descends faster and more consistently, which agrees with our hypotheses that non-deterministic collision changes free path of the particles more frequently.

12.3 Computation speed and scaling efficiency

13. Implementation of FCHC

In the chapter 3 and 4, we theoretically analyzed FCHC automaton and explained its collision algorithm proposed by Henon. In this chapter, we briefly comment on our implementation.

1. Before the simulation of the flow starts, we compute the table with 2^{24} rows. Integer index of the row in the table is the state of the node. Using Henon's algorithm, we compute the optimal isometries for each state.

2. Collisions are resolved by choosing random optimal state from the table, instead of computing the optimal isometries during collision. To avoid overhead of the random numbers generator, we were choosing optimal states one after another, so that each state is used with the same frequency.

Excerpts of the source code for the computation of table would span 700 lines of text, so we do not show it here. As all source code presented in this chapter, it can be found in `https://github.com/tuniak/fchc`.

13.1 Algorithm for collision

Once we have the table specifying optimal isometrical states ready, the collision reduces to randomly choosing an optimal state from the table. To avoid an expensive generation of random numbers, we are alternating isometries one by one and so we cover all final states uniformly.

```
/* For all nodes 'n' on the grid, this function looks in the table and
    choose from optimal final states */

void Collision(int***grid, int**table, int t, int X, int Y, int Z)
{
   int x,y,z;

   int n;
   int k;
#pragma omp parallel for private (x,y,z,n,k)
   for (x = 0; x < X; ++x)
      for (y = 0; y < Y; ++y)
         for (z = 0; z < Z; ++z)
         {
            //We find that node with position [x][y][y] on the grid is
                in the state n
            n = grid[x][y][z];
            // If there is obstacle in the node, we skip it.
            if (n & OBS)
               break;
            /* k is initialized by the external parameter t,
                and k is moduled by number of optimal isometries for the
                   state n */
            k = (t % table[n][0]) + 1;
```

```
            // we assign the k-th optimal state to the node
            grid[x][y][z] = table[n][k];

            //every step, we increment the external parameter, so we
                hopefully achieve the uniform distribution of k (there
                is huge number of nodes on the grid)
            ++t;
         }
}
```

13.2 Propagation in FCHC

Although the collisions are resolved in the four dimensional space (and the particles are the four dimensional objects), the propagation is performed on the 3D projection of FCHC lattice 4.1.

```
/*
The propagation is supposed to happen simultaneously in all nodes at
    once. Because it is performed sequentially in the computation, we
    need to propagate particles from the current array to a different
    one. Every time step, these two arrays alternate.
*/
void Propagation(int***from, int***to, int**table, int X, int Y, int Z)
{
   int x,y,z;
   int i;
   int n;
   int new_x, new_y, new_z;

#pragma omp parallel for private (n, new_x, new_y, new_z, x, y, z, i)
   for ( x = 0; x < X; ++x)
      for ( y = 0; y < Y; ++y)
         for ( z = 0; z < Z; ++z)
         {
            // The current state of the node at [x][y][z]
            n = from[x][y][z];
            // We check every cell
            for ( i = 0; i < 24; ++i)
            {
               // If the cell C[i] is occupied by particle, we propagate
                   it to the corresponding node.
               if (n & C[i])
               {
                  new_x = PeriodicBC(x + c[i][0], X);
                  new_y = PeriodicBC(y + c[i][1], Y);
                  new_z = PeriodicBC(z + c[i][2], Z);
                  // If there is an obstacle in the node, particle's
                      velocity is reversed.
                  // e.g. particle with velocity [1,0,-1,0] gains
```

```
                    velocity [-1,0,1,0] by the reflection from the
                    obstacle
                // Hence, in the next step, it propagates to the node
                    where it come from
                if (to[new_x][new_y][new_z] & OBS)
                   to[new_x][new_y][new_z] |= Reverse[i];
                else
                   to[new_x][new_y][new_z] |= C[i];
             }
          }
          from[x][y][z] = 0;
       }
}
```

14. Simulation of the flow around the obstacles

The microdynamics of the lattice-gas cellular automata is non-physical, and to obtain the physical velocity, we need to compute its mean value over thousands of nodes.

Functions that compute velocity are trivial and similar for Pair-interaction and FCHC alike, so we comment only on FCHC implementation.

```
void compute_velocity(int***grid, double****v, int X, int Y, int Z,
    int side, int I, int J, int K)
{
   // The mean velocity is computed over 'side'*'side'*'side' nodes..
   double N = side*side*side;
   int i, j, k, l, m, n;
   int x, y, z;

#pragma omp parallel for private (i,j,k,l,m,n,x,y,z)
   for (i = 0; i < I; ++i)
   {
      for (j = 0; j < J; ++j)
      {
         for (k = 0; k < K; ++k)
         {
            // Null the velocities from previous time-step
            v[i][j][k][0] = 0;
            v[i][j][k][1] = 0;
            v[i][j][k][2] = 0;

            for (x = i*side; x < (i + 1)*side; ++x)
            {
               for (y = j*side; y < (j + 1)*side; ++y)
               {
                  for (z = k*side; z < (k + 1)*side; ++z)
                  {
                     n = grid[x][y][z];
                     for (m = 0; m < 24; ++m)
                     {
                        if (n & C[m])
                        {
                           for (l = 0; l < 3; ++l)
                           {
                              v[i][j][k][l] += c[m][l];
                           }
                        }
                     }
                  }
               }
            }
```

```
            v[i][j][k][0] /= 0;
            v[i][j][k][1] /= 0;
            v[i][j][k][2] /= 0;
         }
      }
   }
}
```

In the next section, we will present the simulation of the flow around the spherical obstacle and round plate. Particles with velocity in Z direction are flowing from the plain $z = 1$, and they propagate freely out of the tunnel from $z = max - 1$.

The flow is enclosed in the tunnel with the square intersection. The no-slip condition is imposed on the tunnel and obstacles alike, and it is guided by the Propagation function.

```
void Propagation(int***from, int***to, int**table, int X, int Y, int Z)
{
....
....
....
....
                // If there is an obstacle in the node, particle's
                    velocity is reversed.
                // e.g. particle with velocity [1,0,-1,0] gains
                    velocity [-1,0,1,0] by the reflection from the
                    obstacle
                // Hence, in the next step, it propagates to the node
                    where it come from
                if (to[new_x][new_y][new_z] & OBS)
                   to[new_x][new_y][new_z] |= Reverse[i];
                else
                   to[new_x][new_y][new_z] |= C[i];
             }
....
....
....
....
}
```

14.1 Flow around the sphere

Each velocity vector is computed as a mean over $N = 80^3/8 = 64000$ nodes for PI and $N = 80^3 = 521000$ for FCHC.

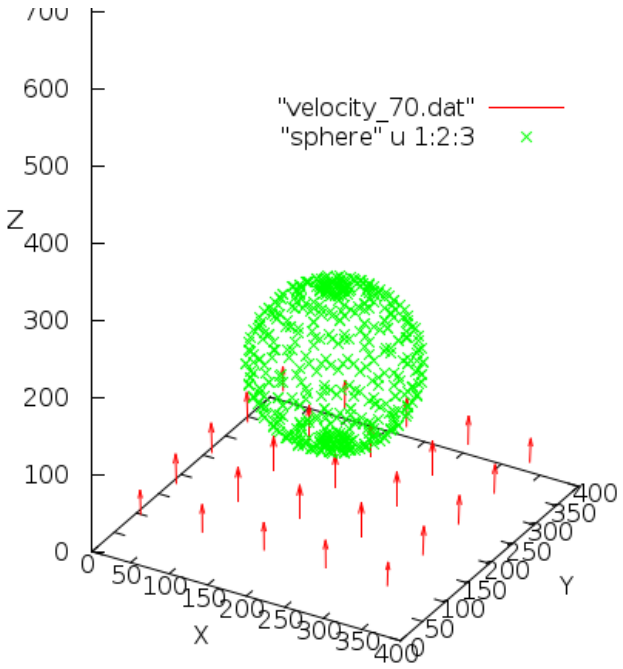


Figure 14.1: PI - flow around sphere, time 70

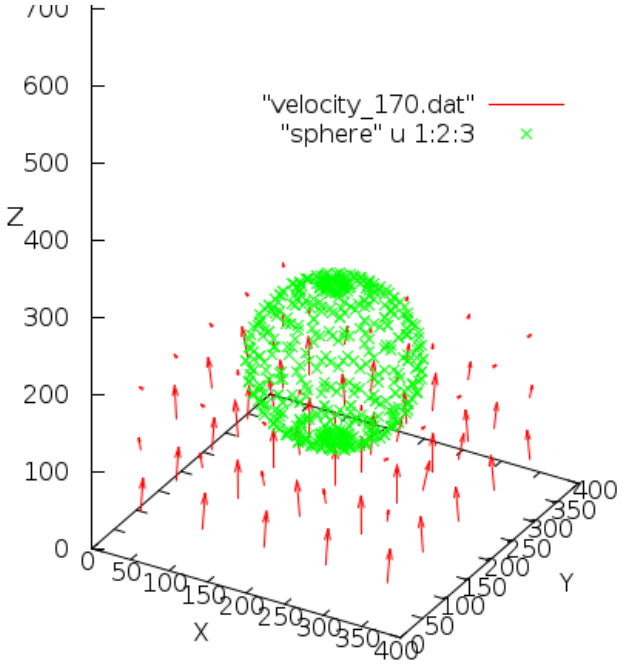


Figure 14.2: PI - flow around sphere, time 170

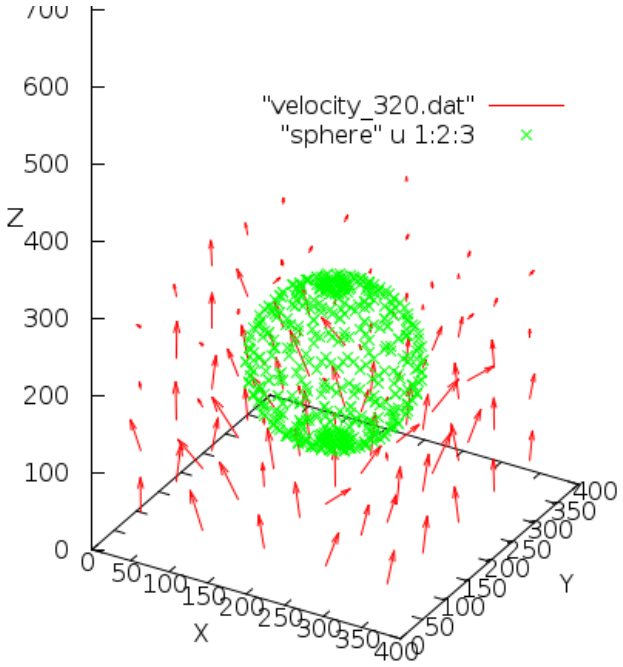


Figure 14.3: PI - flow around sphere, time 320

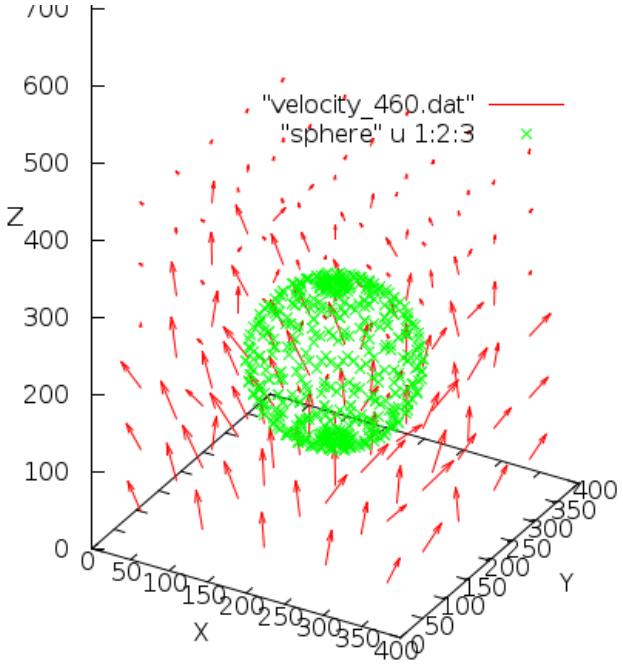


Figure 14.4: PI - flow around sphere, time 460

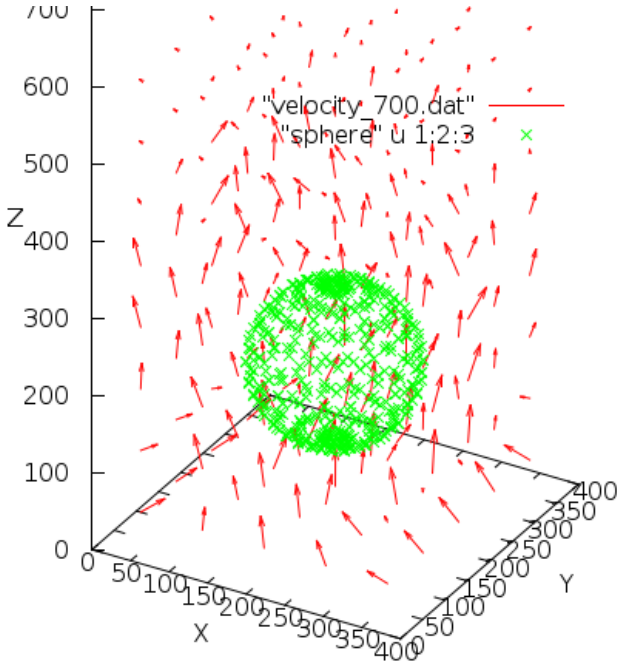


Figure 14.5: PI - flow around sphere, time 700

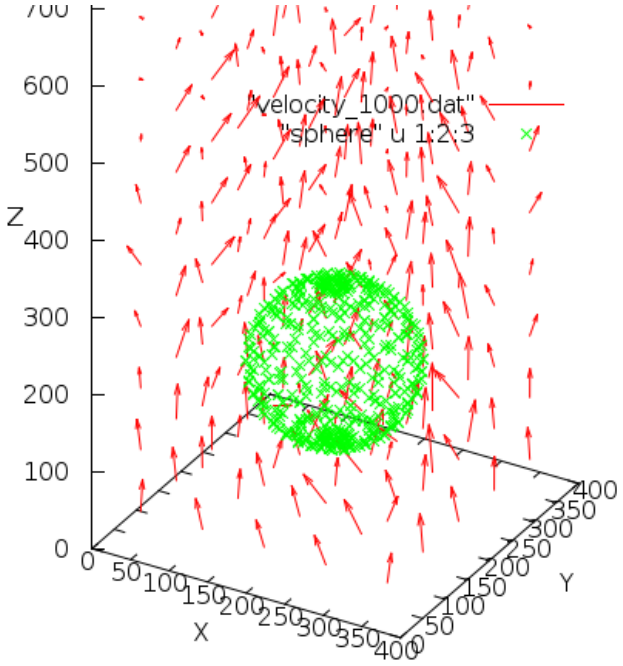


Figure 14.6: PI - flow around sphere, time 1000

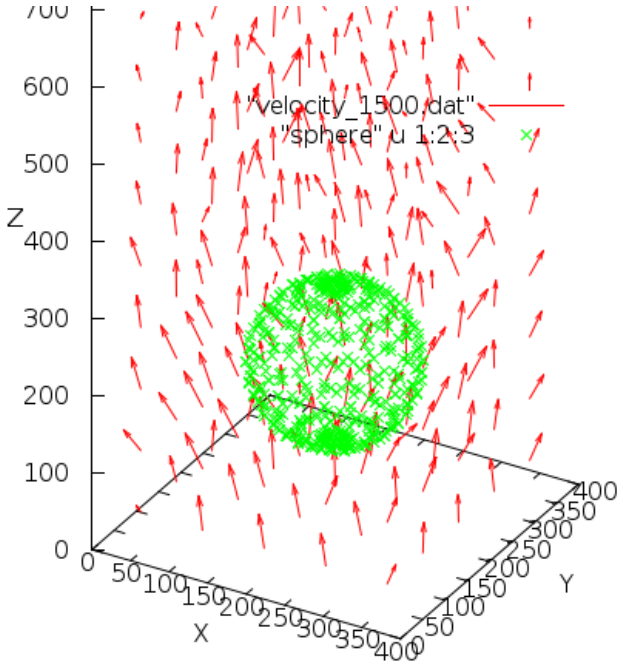


Figure 14.7: PI - flow around sphere, time 1500

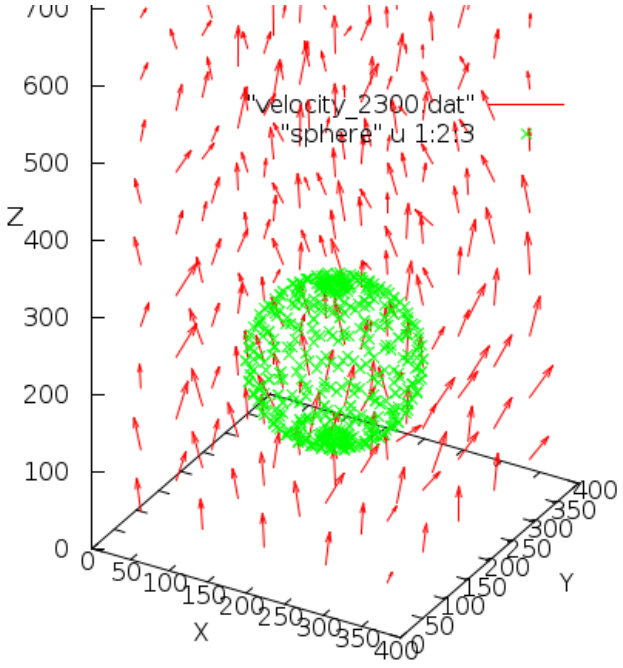


Figure 14.8: PI - flow around sphere, time 2300

14.2 Flow around the disk

Flow around the disk was performed on the Pair-interaction model only, the macroscopic velocity field was obtained by averaging over $N = 40^3/8 = 8000$ nodes.

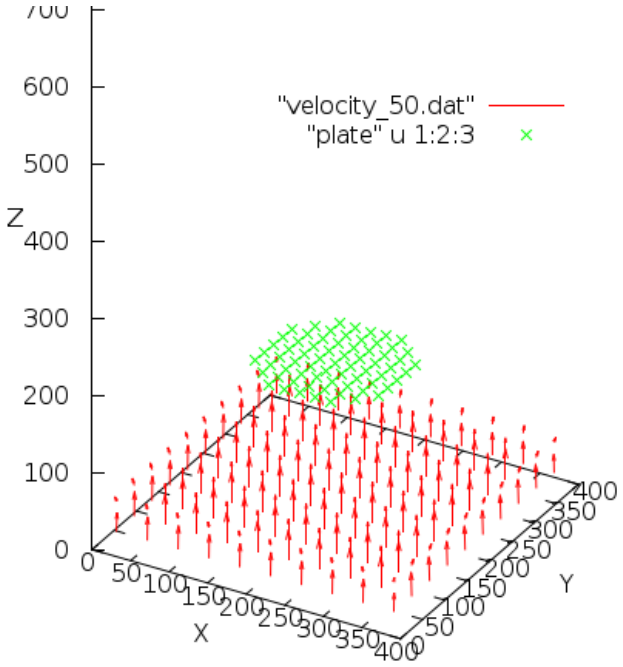


Figure 14.9: PI - flow around plate, time 50

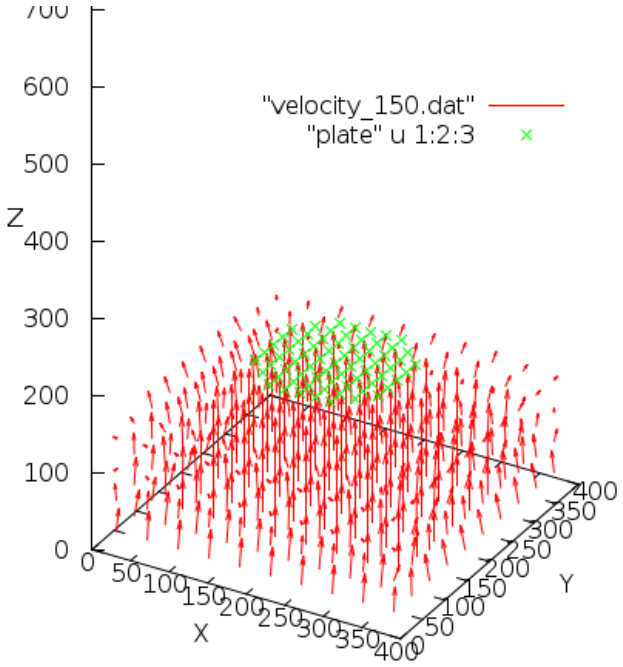


Figure 14.10: PI - flow around plate, time 150

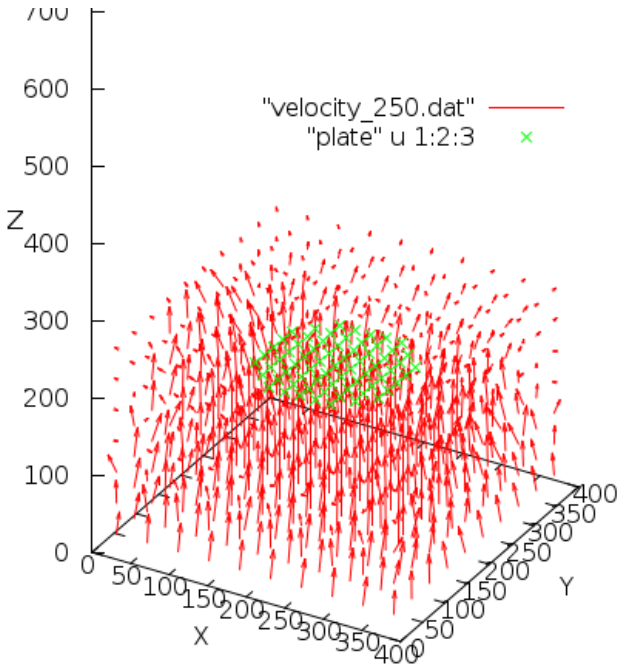


Figure 14.11: PI - flow around plate, time 250

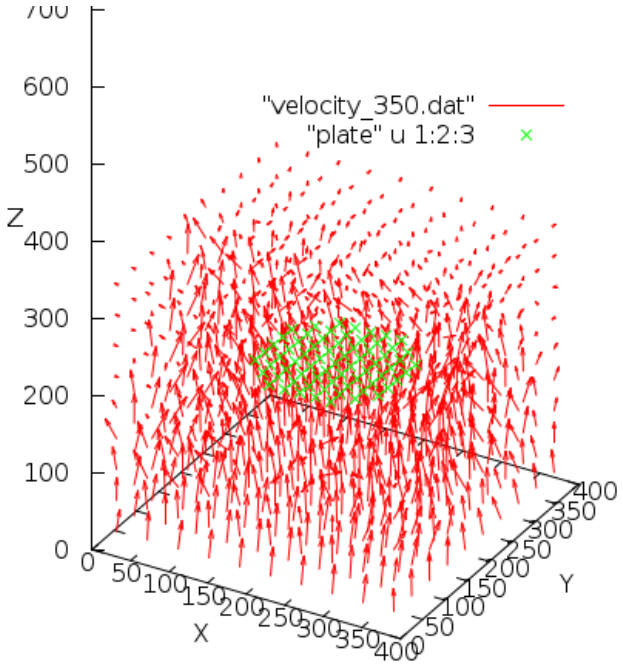


Figure 14.12: PI - flow around plate, time 350

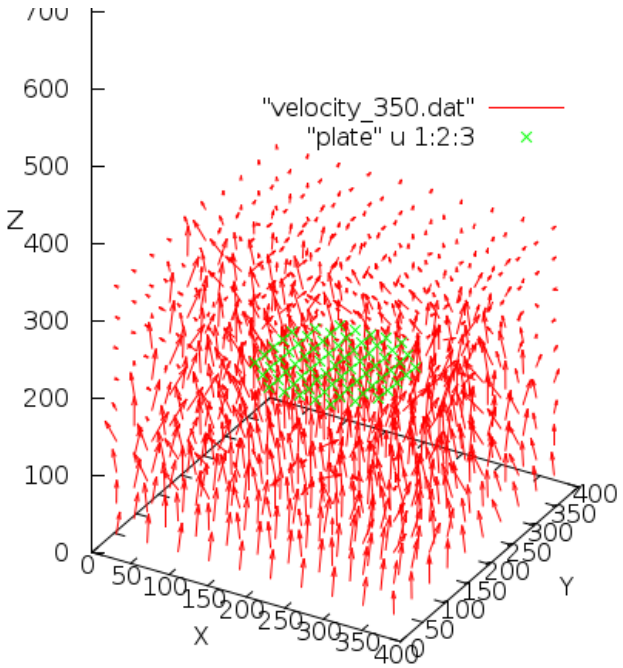


Figure 14.13: PI - flow around plate, time 350

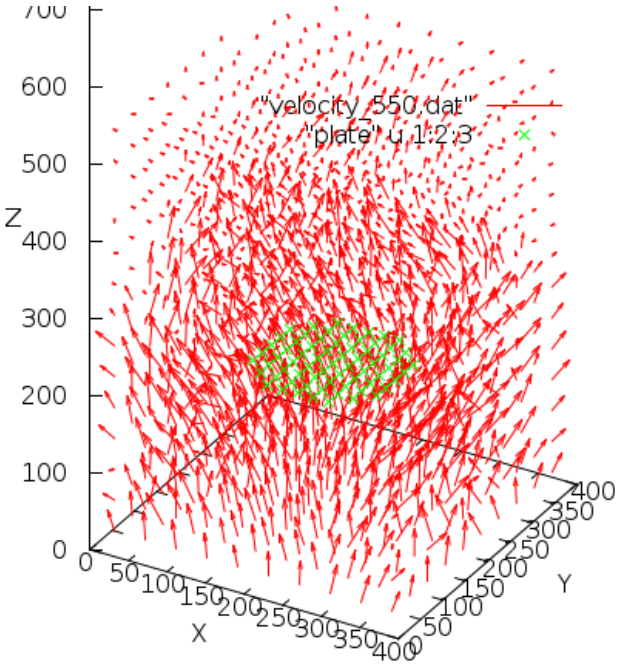


Figure 14.14: PI - flow around plate, time 550

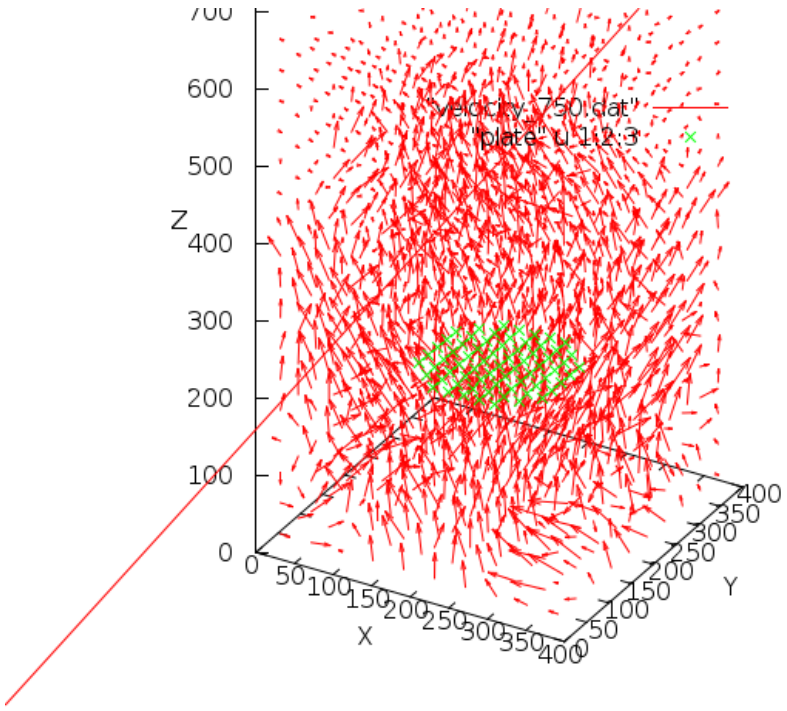


Figure 14.15: PI - flow around plate, time 750

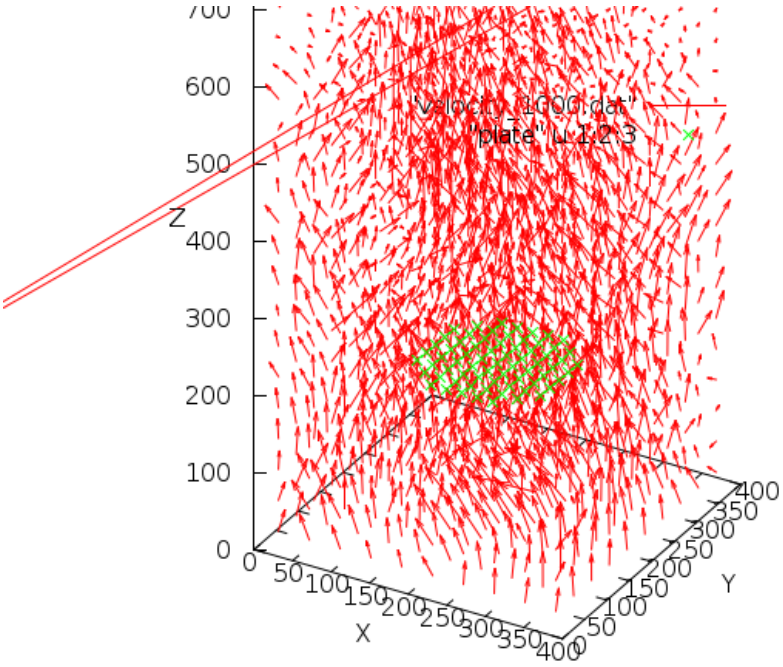


Figure 14.16: PI - flow around plate, time 1000

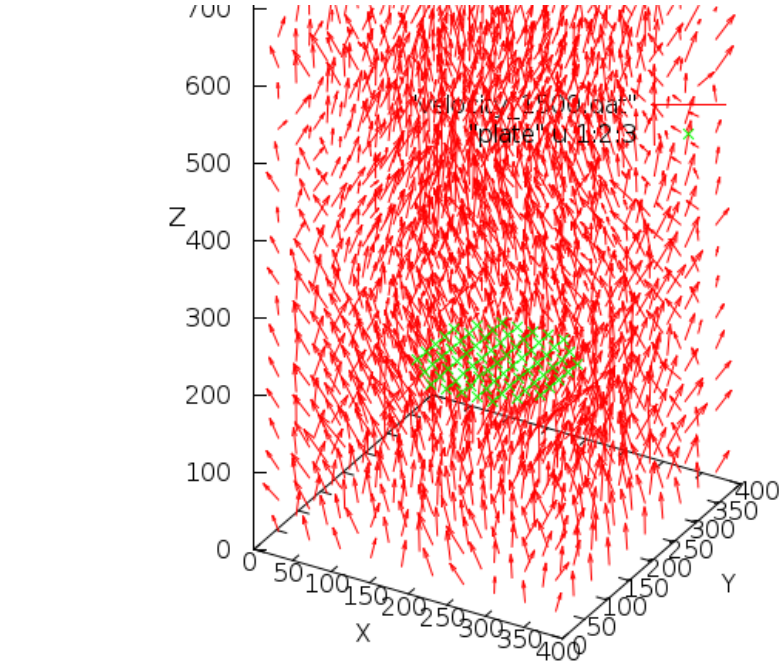


Figure 14.17: PI - flow around plate, time 1500

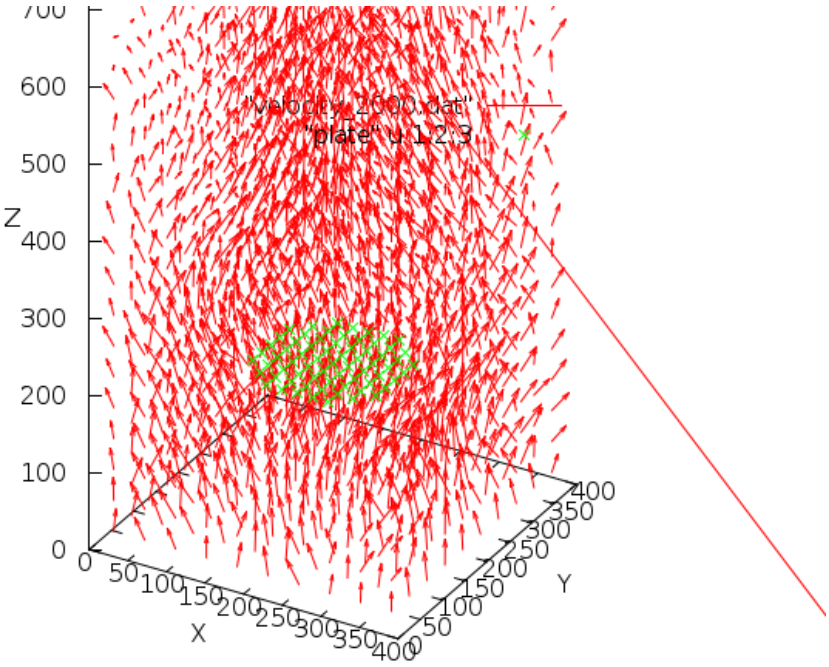


Figure 14.18: PI - flow around plate, time 2000

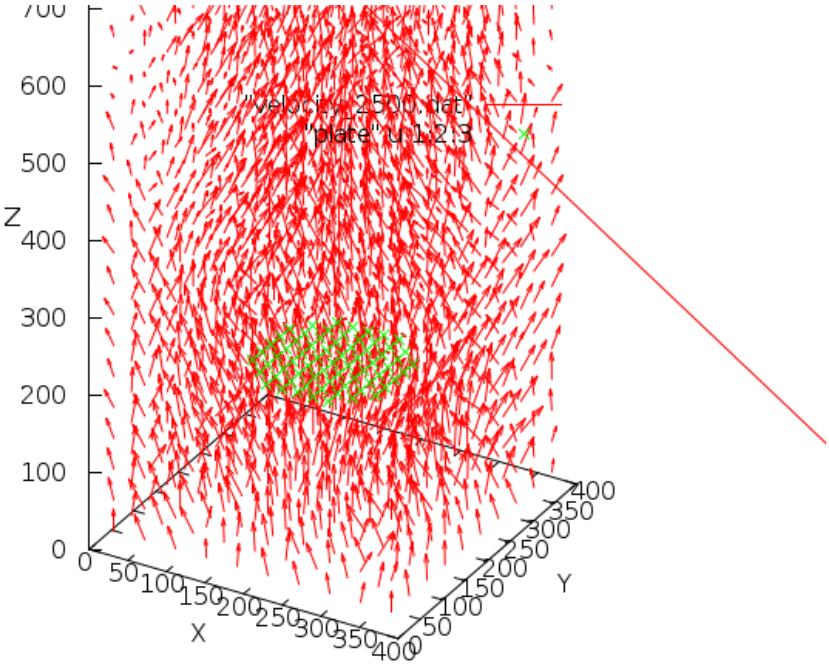


Figure 14.19: PI - flow around plate, time 2500

15. Fully developed turbulence simulated on LGCA

We are intending to simulate fully developed turbulent flow using FCHC and both PI variants and see how close we can get to the theoretical prediction of K41 theory and experimental results.

15.1 Inward flow on the sphere

To develop the turbulent flow that would be isotropic, we chose the following setting:

1. On the sphere inscribed in the lattice, a layer of particles is created every time-step.
2. The flow is directed in the middle of the sphere.
3. If the particle strayed out the sphere, it was forgotten.

Since the particles have discrete momenta, most of them cannot be directed precisely in the middle. To see what directions of particle momenta are allowed, consider the cube inscribed in the sphere as on the figure 15.1.

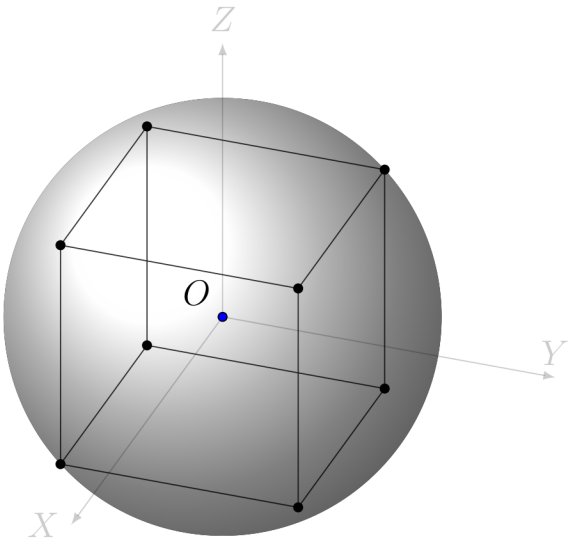


Figure 15.1: State of a node before collision

From a single node, we can direct particle in 26 basic direction, corresponding to the faces, edges and vertices of the cube.

Six directions correspond to the vectors, that are normal of the faces of the cube

$$[1,0,0],\ [-1,0,0],\ [0,1,0],\ [0,-1,0],\ [0,0,1],\ [0,0,-1]. \tag{15.1}$$

Eight directions corresponds to the "diagonal" vectors

$$[1,1,1],\ [-1,1,1],...,[-1,-1,-1] \tag{15.2}$$

and the other twelve vectors read

$$[1,1,0],\ [1,0,1],\ [-1,1,0],...,[0,-1,-1]. \tag{15.3}$$

Therefore, we divide the sphere on the 26 surface areas. On each of these areas, particles have momentum in the direction corresponding to one of the vectors above.

Simulation on the smaller lattice indicates that our setting results in the flow that is symmetric and directs to the middle of the sphere.

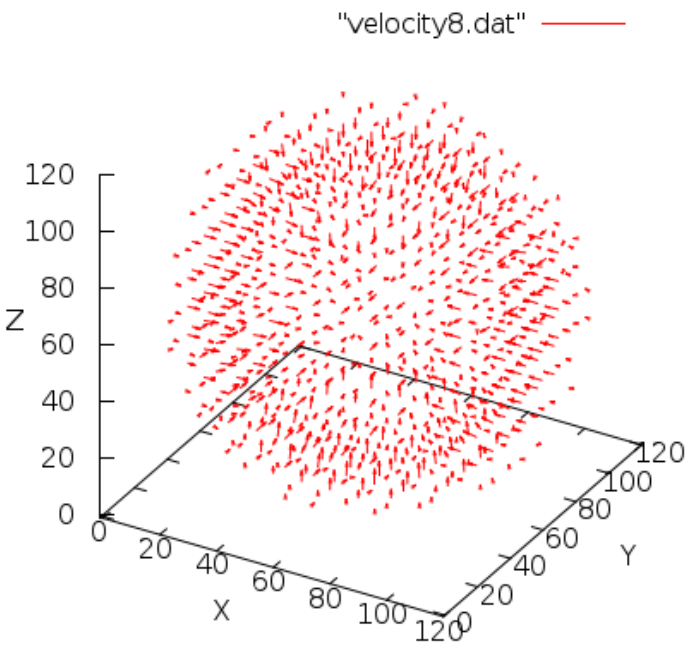


Figure 15.2: Velocity field at time 8 - FCHC inward flow

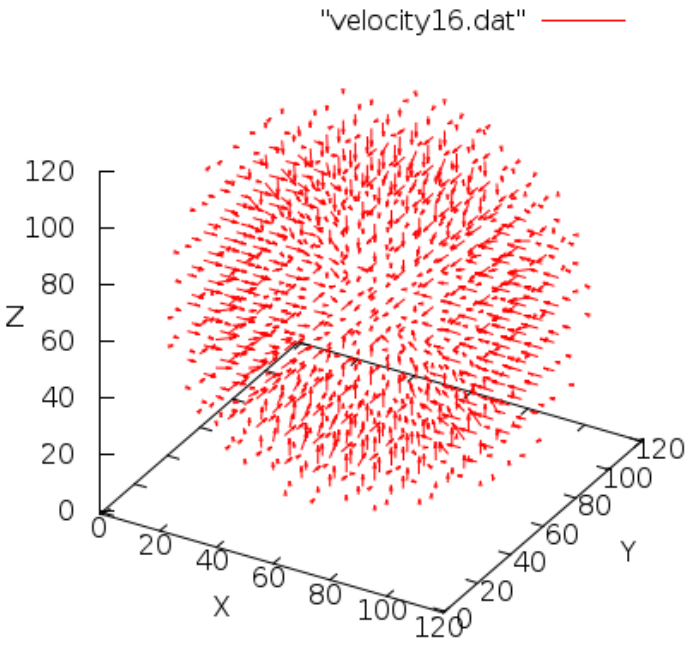


Figure 15.3: Velocity field at time 16 - FCHC inward flow

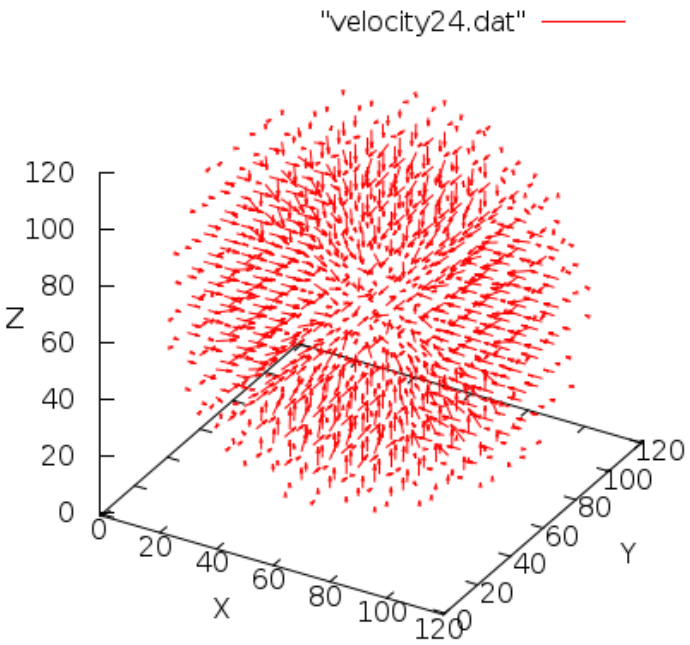


Figure 15.4: Velocity field at time 24 - FCHC inward flow

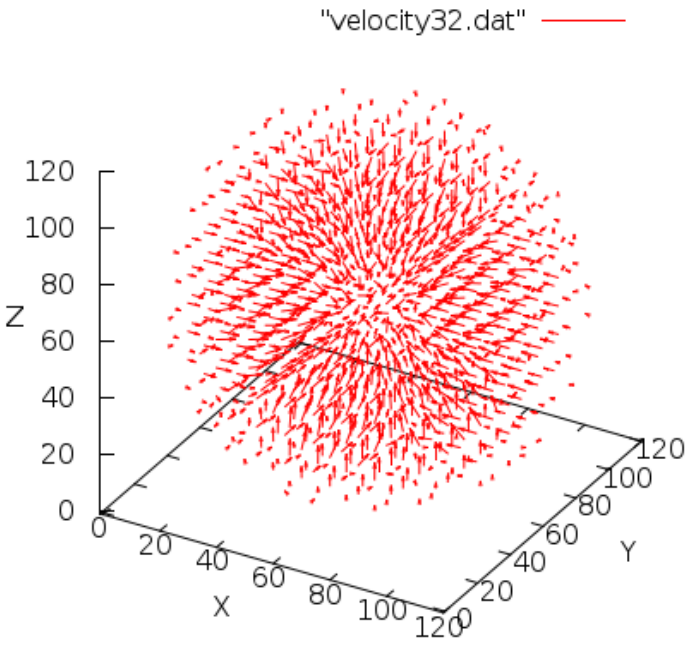


Figure 15.5: Velocity field at time 32 - FCHC inward flow

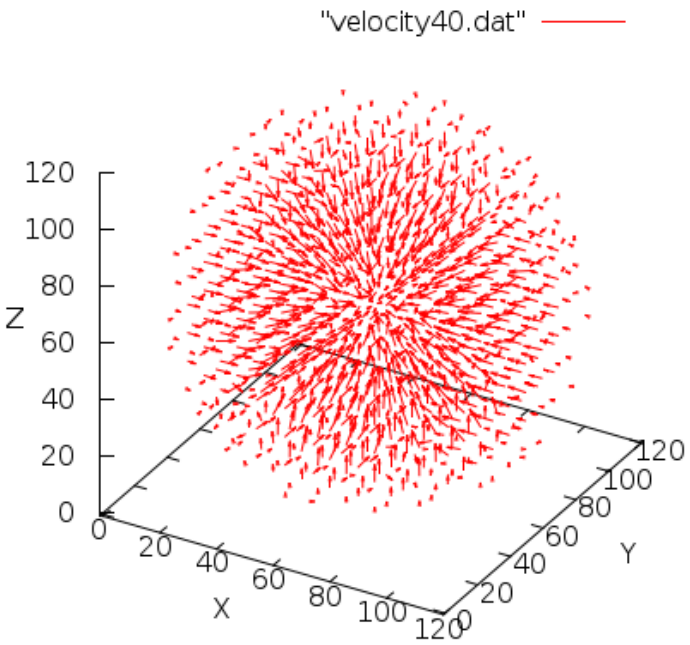


Figure 15.6: Velocity field at time 40 - FCHC inward flow

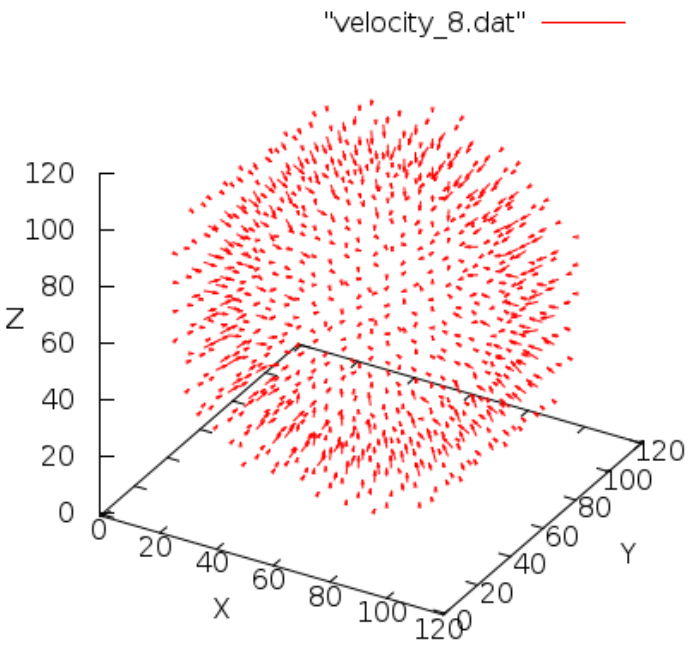


Figure 15.7: Velocity field at time 8 - PI inward flow

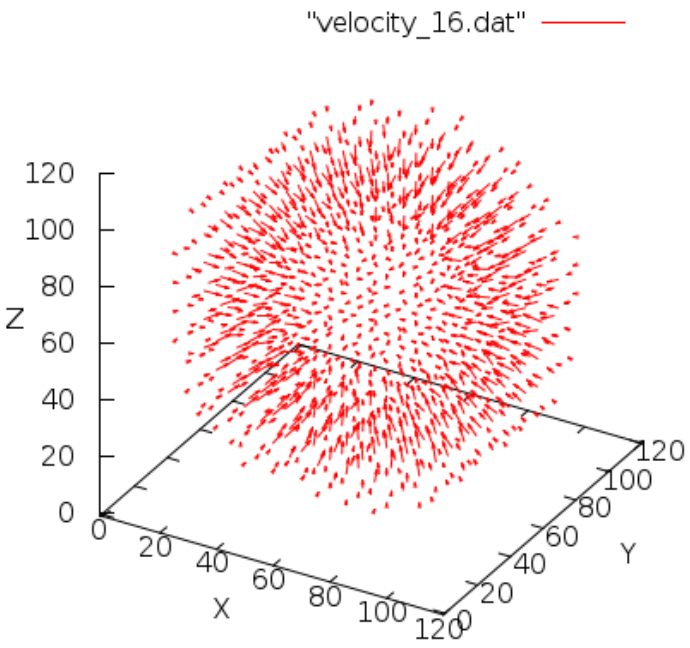


Figure 15.8: Velocity field at time 16 - PI inward flow

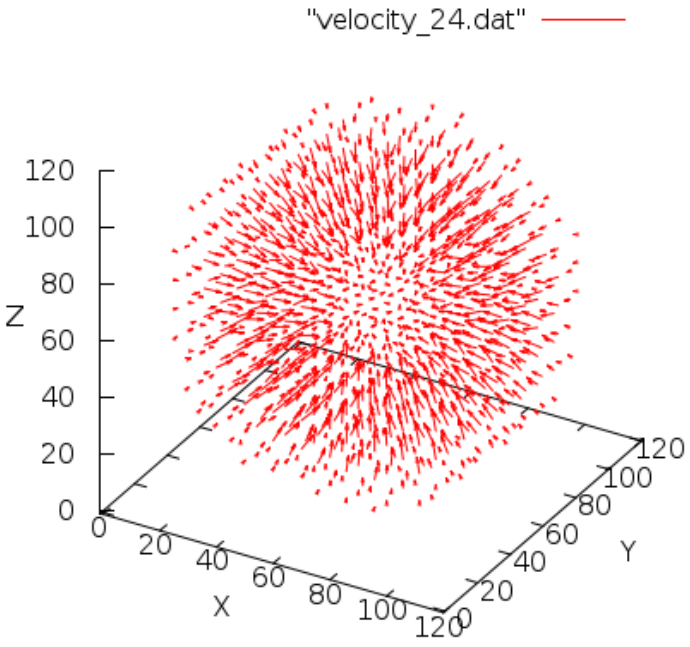


Figure 15.9: Velocity field at time 24 - PI inward flow

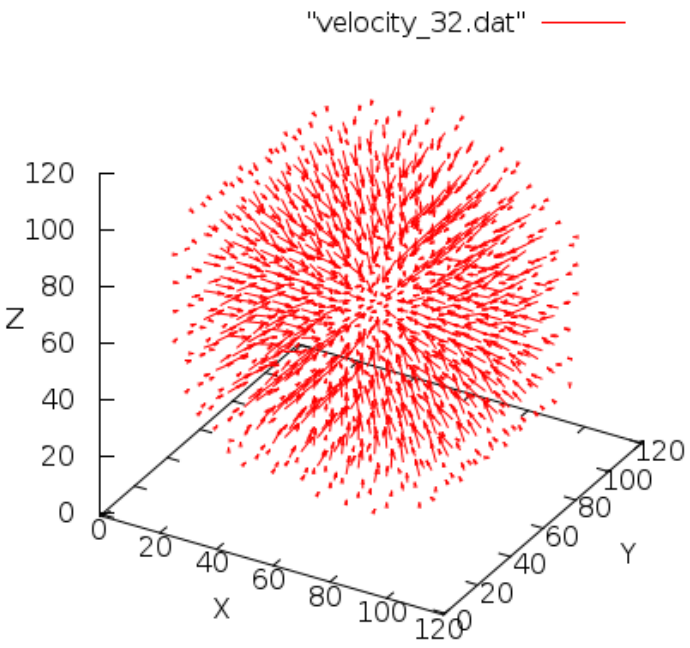


Figure 15.10: Velocity field at time 32 - PI inward flow

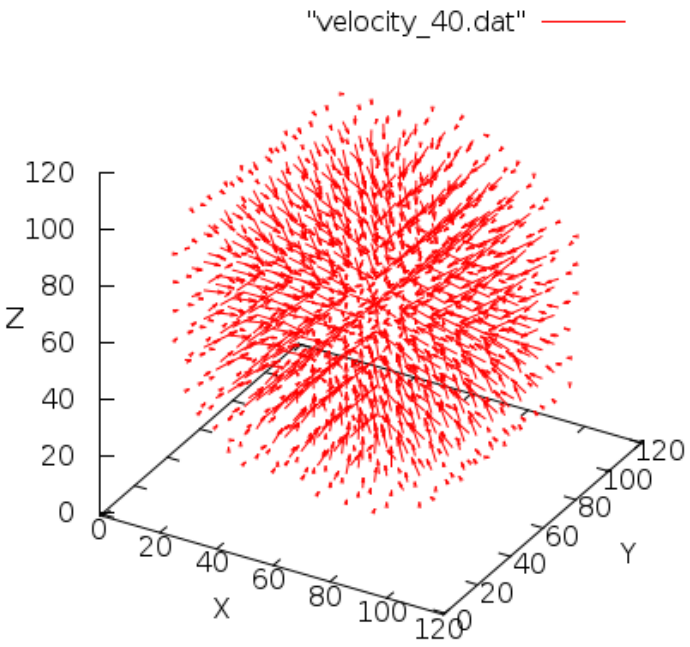


Figure 15.11: Velocity field at time 40 - PI inward flow

15.2 Statistical properties of the flow

In this chapter, we will inspect statistical properties of this flow to see whether isotropy and homogenity was recovered in our model, as predicted by K41 theory and confirmed by experimental data.

All the statistical quantities that we computed and visualized were obtained by averaging over 10 000 steps for lattice $1000 \times 1000 \times 1000$, and 4 000 steps for lattice $600 \times 600 \times 600$.

15.2.1 First statistical moment - the mean velocity field

For our model, the first statistical moment – mean velocity, is defined as

$$\langle v(\boldsymbol{r}) \rangle = \sum_{t=T_1}^{T_2} v(t, \boldsymbol{r}). \tag{15.4}$$

The second moment – covariance tensor, is defined for the centered vector functions as

$$\Gamma^c_{ij} = \langle v_{ij} \rangle, \tag{15.5}$$

but for the functions with non-zero mean value, which is our case, covariance needs to be centered:

$$\Gamma_{ij} = \langle v_{ij} \rangle - \langle v_i \rangle \langle v_j \rangle \,. \tag{15.6}$$

Implementation of the formulas 15.4 and 15.5 was performed inside the `compute_velocity` function, centering of the covariance tensor in `finalize_covariance_tensor`.

```
void compute_velocity(Node***array, double****v, double****mean,
    double*****gamma, int dx, int dy, int dz, int I, int J, int K)
{
   double N = dx*dy*dz;

   int i,j,k;
   int x,y,z;
   int c;

#pragma omp parallel for private (i,j,k,x,y,z,c)
   for (i = 0; i < I; ++i)
   {
      for (j = 0; j < J; ++j)
      {
         for (k = 0; k < K; ++k)
         {
            v[i][j][k][0] = 0;
            v[i][j][k][1] = 0;
            v[i][j][k][2] = 0;
            for (x = i*dx; x < (i + 1)*dx; x += 2)
            {
               for (y = j*dy; y < (j + 1)*dy; y += 2)
```

```
   {
      for (z = k*dz; z < (k + 1)*dz; z += 2)
      {
         for (c = 1; c <= H; c <<= 1)
         {
            if (c & array[x][y][z].m)
            {
               if (c & array[x][y][z].p[0])
               {
                  if (c & dirX)
                  {
                     ++v[i][j][k][0];
                  }
                  else
                  {
                     --v[i][j][k][0];
                  }
               }
               if (c & array[x][y][z].p[1])
               {
                  if (c & dirY)
                  {
                     ++v[i][j][k][1];
                  }
                  else
                  {
                     --v[i][j][k][1];
                  }
               }
               if (c & array[x][y][z].p[2])
               {
                  if (c & dirZ)
                  {
                     ++v[i][j][k][2];
                  }
                  else
                  {
                     --v[i][j][k][2];
                  }
               }
            }
         }
      }
   }
}

v[i][j][k][0] /= N;
v[i][j][k][1] /= N;
v[i][j][k][2] /= N;

// mean velocity
```

```
            mean[i][j][k][0] += v[i][j][k][0];
            mean[i][j][k][1] += v[i][j][k][1];
            mean[i][j][k][2] += v[i][j][k][2];

            // covariance tensor
            gamma[i][j][k][0][0] += (v[i][j][k][0]*v[i][j][k][0]);
            gamma[i][j][k][0][1] += (v[i][j][k][0]*v[i][j][k][1]);
            gamma[i][j][k][0][2] += (v[i][j][k][0]*v[i][j][k][2]);
            gamma[i][j][k][1][0] += (v[i][j][k][1]*v[i][j][k][0]);
            gamma[i][j][k][1][1] += (v[i][j][k][1]*v[i][j][k][1]);
            gamma[i][j][k][1][2] += (v[i][j][k][1]*v[i][j][k][2]);
            gamma[i][j][k][2][0] += (v[i][j][k][2]*v[i][j][k][0]);
            gamma[i][j][k][2][1] += (v[i][j][k][2]*v[i][j][k][1]);
            gamma[i][j][k][2][2] += (v[i][j][k][2]*v[i][j][k][2]);
         }
      }
   }
}

void finalize_covariance_tensor(double****mean, double*****gamma, int
    I, int J, int K, int T)
{
   int i,j,k,d,e;
#pragma omp parallel for private (i,j,k,d,e)
   for(i=0; i<I; ++i)
      for(j=0; j<J; ++j)
         for(k=0; k<K; ++k)
            for(d=0; d<3; ++d)
               for(e=0; e<3; ++e)
                  gamma[i][j][k][d][e] = (gamma[i][j][k][d][e] / T ) -
                      mean[i][j][k][d]*mean[i][j][k][e];
}
```

In order to interpret the data obtained in the simulation, we have to compare the correlation tensor to one predicted by Kolmogorov's K41 theory of fully-developed turbulence.

In the isotropic situation, there is no preferred direction and, thus, the only second-order tensor at our disposal is the Kronecker delta δ_{ij}. Hence, the correlation tensor must be of the form

$$\Gamma_{ij} = K\,\delta_{ij}, \tag{15.7}$$

where K is, in general, function of spatial coordinates, having dimension of velocity-squared. However, the requirement of global isotropy implies global homogeneity, so that K is in fact constant. Physically speaking, this form of correlation tensor means that different Cartesian components of the velocity field at given point are uncorrelated, i.e. independent.

In the anisotropic situation, on the other hand, there is a preferred direction which can be represented by a unit vector $\boldsymbol{n}$. In that case, the most general second-order tensor is given by

$$\Gamma_{ij} = K_1\,\delta_{ij} + K_2\,n_i\,n_j, \tag{15.8}$$

where K_1 and K_2 are functions in general. In other words, there are just two independent components of the correlation tensor.

In the most general situation, when the anisotropy is specified by three linearly independent vectors, we get six independent components, which simply corresponds to a general second-order symmetric tensor.

Any such tensor defines a quadratic form and a family of surfaces parametrized by constant C via equation

$$\Gamma_{ij}\, x_i\, x_j = C. \tag{15.9}$$

In the isotropic case these surfaces are concentric spheres. In the anisotropic case, we will get ellipsoids, because correlation tensor is positive-definite[1]. We can conclude that the deviation of the surfaces from spherical shape measures the failure of the system to be isotropic.

The Kolmogorov theory then makes assumptions known as Kolmogorov hypotheses, asserting that the correlation functions inside the inertial interval are independent of the external scale (energy injection) and of the dissipation scale.

These assumptions, together with isotropy and dimensional analysis, then restrict possible functional dependence of correlators on the parameters of the system, namely the dissipation rate and viscosity (see, e.g. [15] and references therein). In this thesis, we do not go into details of Kolmogorov's theory, instead we focus on the question to what extent the assumption of isotropy is satisfied in the cellular automaton model.

[1]This can be easily seen by the following argument. In the frame of main axes of Γ_{ij}, this tensor is diagonal and its diagonal terms are $\Gamma_{ii} = \langle v_i^2 \rangle \geq 0$. Since quadratic forms are inertial with respect to rotations, this property is preserved in any frame.

15.3 Graphical representation of the obtained results

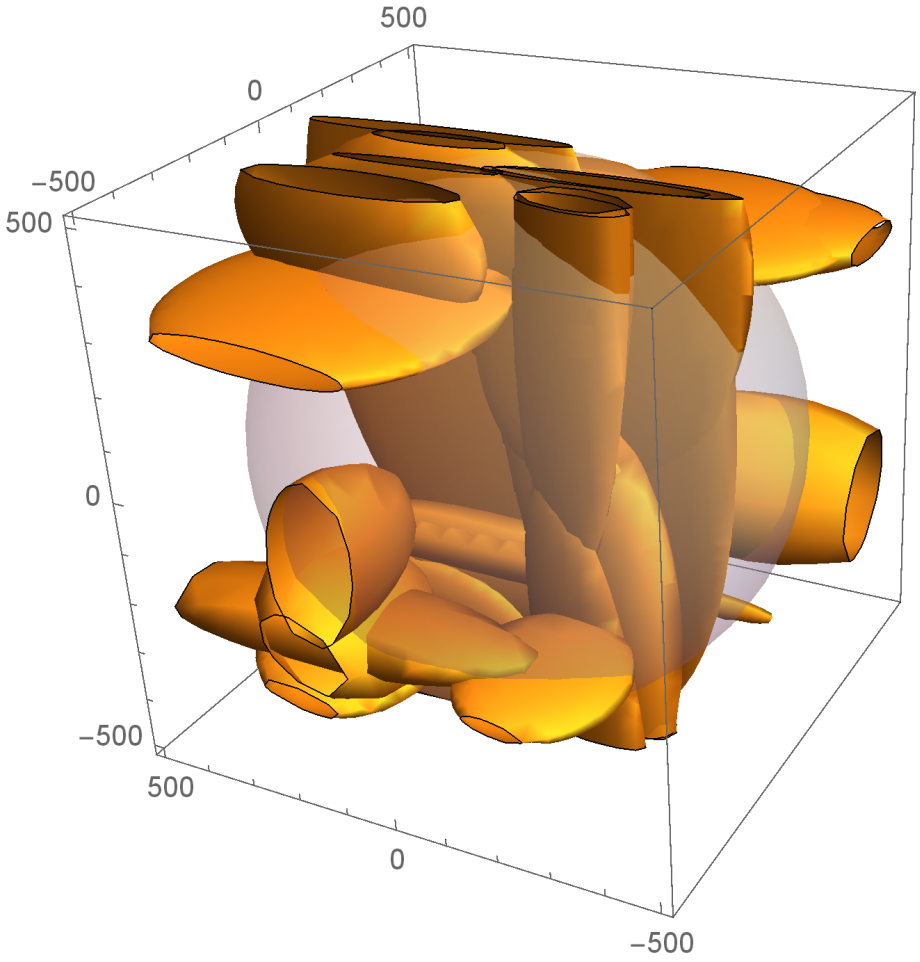


Figure 15.12: Covariance tensor field for deterministic PI on lattice $1000 \times 1000 \times 1000$ – only tensors with highest norm are presented

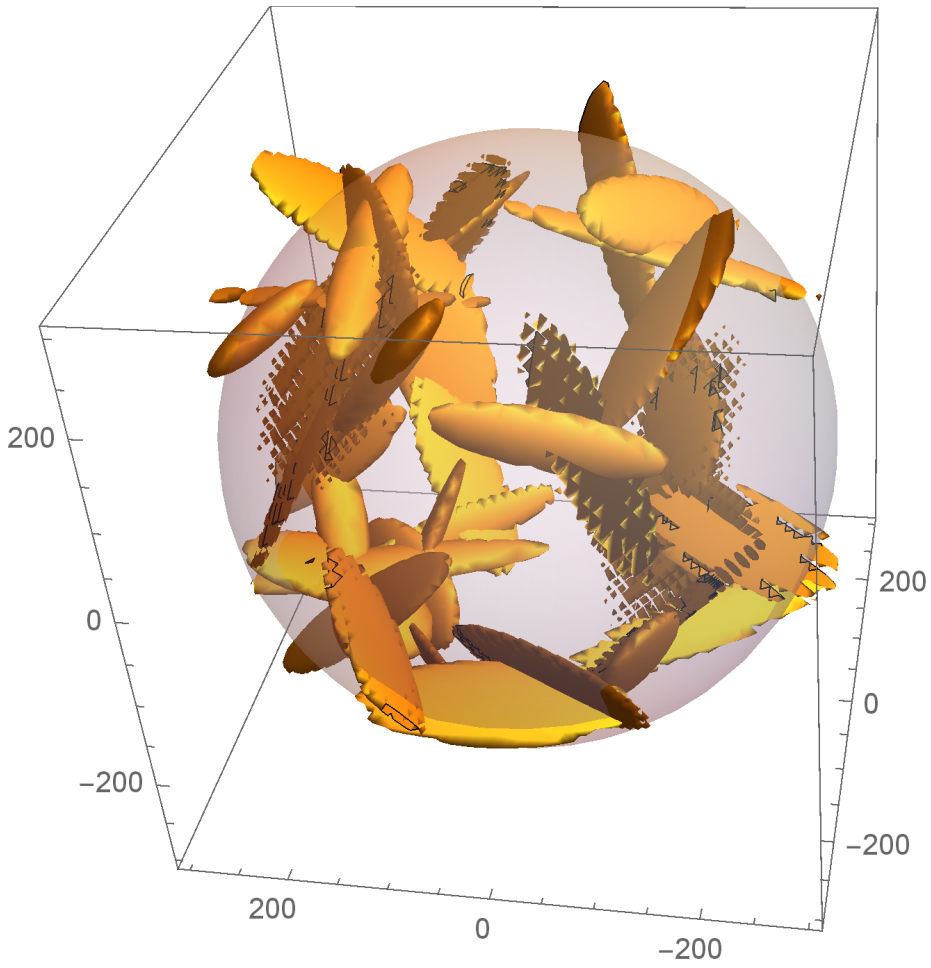


Figure 15.13: Covariance tensor field for deterministic PI on lattice $600 \times 600 \times 600$ – only tensors with highest norm are presented

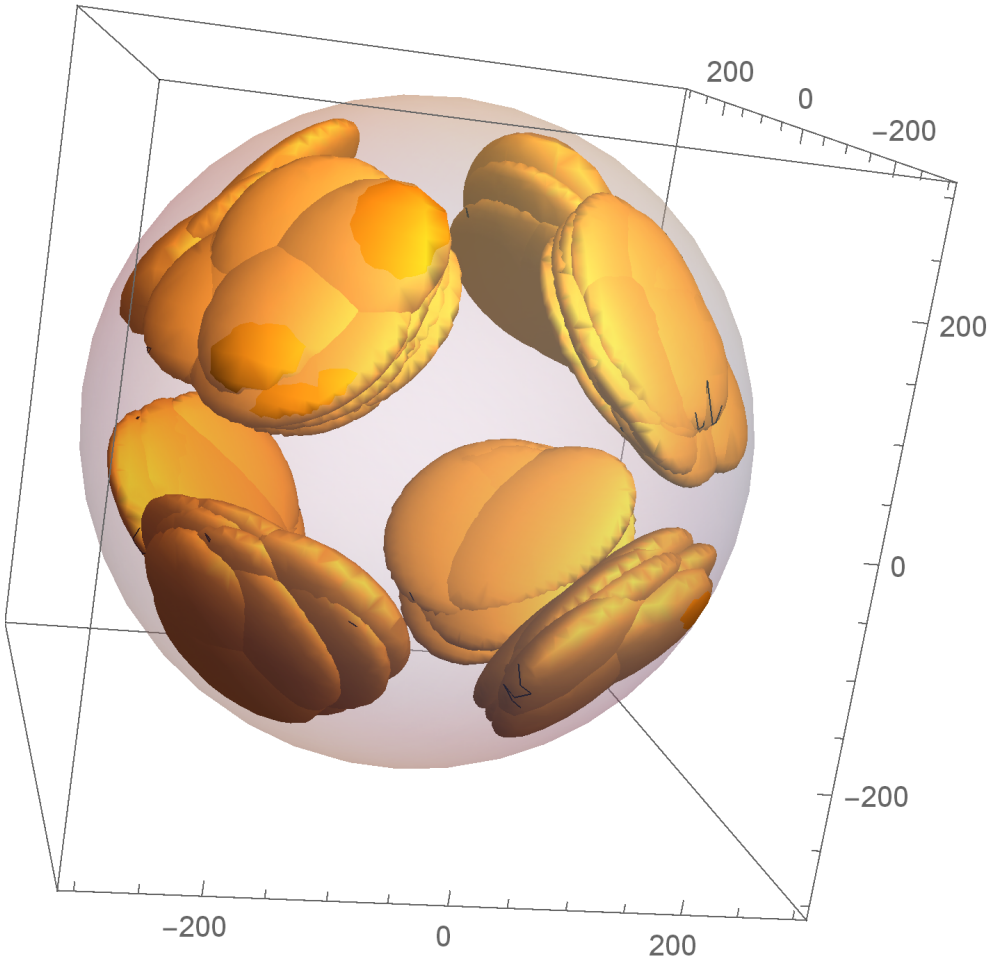


Figure 15.14: Covariance tensor field for non-deterministic PI on lattice $600 \times 600 \times 600$ – only the tensors with highest norm are presented

The figures above show that the correlation tensor field is represented by highly ellipsoidal surfaces and hence, the isotropy seems to be significantly violated.

In case of non-deterministic automaton, the ellipsoids are tangent to the surface of the sphere 15.3. They are found at the corners of inscribed cube 15.1, where the density of the inward flow was lowest. This is the imperfection of the boundary conditions on the sphere, and it is a reasonable explanation of the observed anisotropy.

On the other hand, the orientation of ellipsoids for deterministic automaton is rather chaotic and their shape is irregular, in contrast to the symmetric, cranky cookies of non-deterministic automaton.

Because of the flawed boundary conditions on the sphere, we cannot conclude whether non-deterministic PI is appropriate model for the simulation of fully developed turbulence. This is left for future investigation.

Nevertheless, the boundary conditions were same for both automaton and so we can conclude that non-deterministic PI outperformed its classical variant in all simulations that have been conducted.

Bibliography

[1] Cook, M.: *Universality in Elementary Cellular Automata.* Complex Systems 15, 1-40, 2004.

[2] Wolf-Gladrow, Dieter A.: *Lattice-Gas Cellular Automata and Lattice Boltzmann Models – An Introduction* . New York: Springer, 2005.

[3] Henon, Michel: *Isometric collision rules for the four dimensional FCHC Lattice Gas.* Complex Systems 1, 1987

[4] Wolfram, Stephen: *New kind of science* . Wolfram Media, Inc., 2002

[5] Frisch, Uriel: *Lattice Gas Hydrodynamics in Two and Three Dimensions.* Complex Systems 1, 1987

[6] Timm Krüger, Halim Kusumaatmaja, Alexandr Kuzmin, Orest Shardt, Goncalo Silva, Erlend Magnus Viggen: *The Lattice Boltzmann Method Principles and Practice.* Springer I, 2017

[7] Frisch, Uriel: *Turbulence: The Legacy of A. N. Kolmogorov.* Cambridge University Press, 1995

[8] Bodenheimer, Michel, Laughlin, Gregory P., Rozyczka, Michal, Yorke, Harold W.: *Numerical methods in Astrophysics - An Introduction.* Taylor & Francis Group, LLC, 2007

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

[10] Tong, David: *Lectures on Kinetic Theory, part 2.* http://www.damtp.cam.ac.uk/user/tong/kinetic.html

[11] Hardy, J., de Pazzis , O., Pomeau, Y.: *Time evolution of a two-dimensional model system I. Invariant states and time correlation functions.* J. Math. Phys., 14 (1973), pp. 1746–1759

[12] Nasilowski, Ralf : *A Cellular-Automaton Fluid Model with Simple Rules in Arbitrarily Many Dimensions.* Journal of Statistical Physics, Vol. 65, Nos. 1/2, 1991

[13] Marsaglia, George: *Random Numbers for C: End at last?.* http://www.cse.yorku.ca/ oz/marsaglia-rng.html

[14] Bull, J.R., Jameson, A.: *Simulation of the Compressible Taylor Green Vortex using High-Order Flux Reconstruction Schemes.* 7th AIAA Theoretical Fluid Mechanics Conference, June 2014

[15] Scholtz, Martin: *Vplyv anizotropie na stabilitu skalovacich rezimov v modeli advekcie pasivnej vektorovej primesi.* Kosice, 2007

[16] Aaronson, Scott : *BOOK REVIEW on A New Kind of Science.* Quantum Information and Computation, Vol. 1, No. 0, 2001

[17] 'T HOOFT, Gerald : *The Cellular Automaton Interpretation of Quantum Mechanics*. arXiv:1405.1548v3 [quant-ph] 21 Dec 2015

[18] LEVY, Steven : *The man who cracked the code to everything.* https://www.wired.com/2002/06/wolfram/

List of Figures

List of Tables

List of Abbreviations

Attachments