Technical Note: Data Structures of ILBDC Lattice Boltzmann Solver

Markus Wittmann [*]
Thomas Zeiser
Erlangen Regional Computing Center, 91058 Erlangen, Germany

February 25, 2011

1 Lattice Boltzmann Method

The lattice Boltzmann method (LBM) is used in computational fluid mechanics (CFD) to simulate flows. Introduced by McNamara [3] the LBM historically is based on lattice gas automata. Instead simulating each particle a statistical average for a sufficient small control volume is applied. The LBM can also be considered as the solution to a velocity-discrete Boltzmann equation with an appropriate collision operator.

As the original collision operator is computational intensive the BGK operator [1] can be used instead. This results in the lattice BGK methods [2, 4].

For simulation the simulation domain is discretized by an equally spaced Cartesian grid. The resulting cells are then either fluid or solid. Every cell has N velocity vectors $\mathbf{e}_i$ ($i = 0, \dots, N-1$) and N particle distribution functions f_i ($i = 0, \dots, N-1$) which describe the probability of a certain number of particles being in the velocity range around $\mathbf{e}_i$.

Depending on the dimensionality d of the simulation domain (2D or 3D) and the number of velocity vectors N several models exists. They are usually referred to as DdQN, e.g. D2Q7 or D3Q19.

The lattice Boltzmann method solves the equation

$$f_i(\mathbf{r} + \mathbf{e}_i \Delta t, t + \Delta t) = f_i(\mathbf{r}, t) + \Omega_i(f(\mathbf{r}, t)), \qquad i = 0, \dots, N-1, \tag{1}$$

where Ω_i is the collision operator. This is an explicit FD scheme with only next-neighbor interaction.

Algorithm One time step of the algorithm can logically be divided into a *collision* and a *propagation* step. During collision an intermediate particle distribution function $\tilde{f}_i$ is computed:

$$f_i(\mathbf{r}, t) \to \tilde{f}_i(\mathbf{r}, t). \tag{2}$$

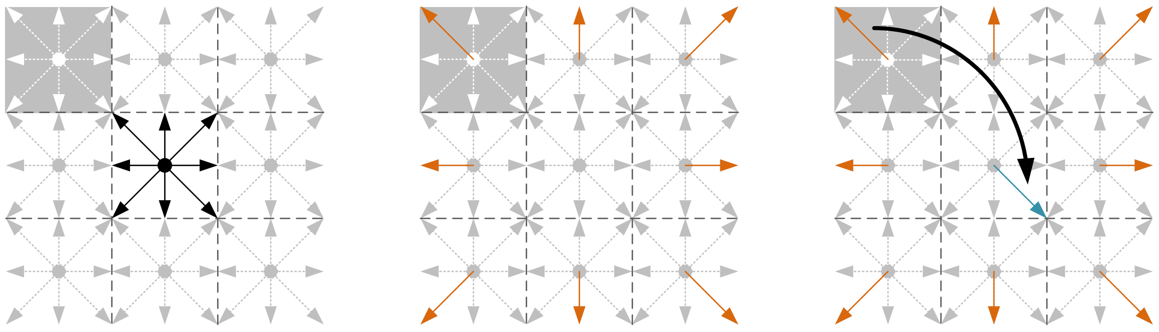
In the propagation step $\tilde{f}_i$'s of a fluid cell are populated to the neighboring cells

[*]Funded by BMBF grand no. 01 IH08003A (SKALB)

$$\tilde{f}_i(\mathbf{r}, t) \to f_i(\mathbf{r} + \mathbf{e}_i \Delta t, t + \Delta t), \tag{3}$$

as in Fig. 1.

For performance reasons collision and propagation should be carried out in one combined step [5]. The simplest approach to work around data dependencies is to use two grids.



(a) After collision and before propagation.

(b) After propagation and before bounce back.

(c) After bounce back.

Figure 1: During propagation the local particle distribution functions f_i in (a) are moved along their corresponding velocity vectors $\mathbf{e}_i$ (black arrows) to the neighbor cells (b). Particle distribution functions f_i that were populated to solid cells (gray cell) are reflected back in the opposite direction as in (c).

Boundary condition If a particle distribution function f_i is propagated to a solid cell then it is reflected to the original but with opposite direction of the velocity vector $\mathbf{e}_i$ as depicted in Fig. 1(c). This can be optimized by storing the intermediate value directly at the final position:

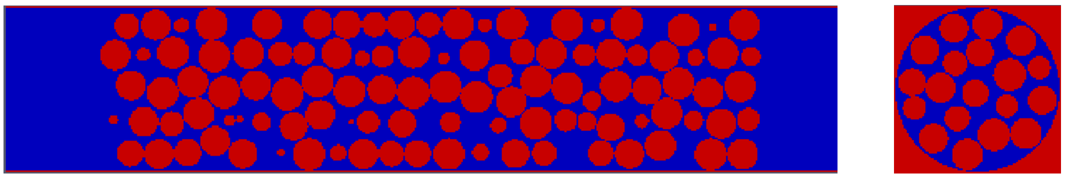
$$f_j(\mathbf{r}, t+1) = \tilde{f}_i(\mathbf{r}, t+1), \quad \text{with} \quad \mathbf{e}_j = -\mathbf{e}_i. \tag{4}$$

This kind of boundary condition is known as *bounce back* and is a mesoscopic equivalent of the macroscopic non-slip boundary condition.

2 Implementation in the ILBDC code

The ILBDC code (International Lattice Boltzmann Development Consortium) [6, 7] is designed to efficiently simulate the flow through porous media with a low fraction of fluid space. A typical example of such a porous media are packed bed reactors where a tube is randomly filled with spheres, cylinders, etc. as visualized in Fig. 2. The carrier material acts as catalyst for fluid pumped through the tube.

For simulation the whole domain can be allocated as a multi-dimensional array together with a flag field indicating which cell is fluid or solid. This so called marker and cell (MAC) method is suitable for geometries with high fluid space.

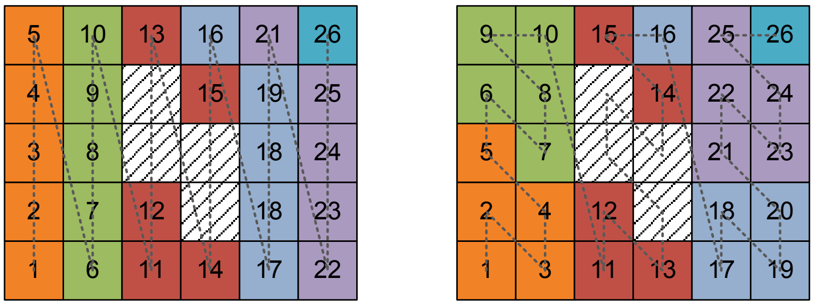

(a) Slice in the xy plane.

(b) Slice in the yz plane.

Figure 2: Slices of a packed bed reactor of spheres. Blue indicates fluid cells and red solid cells.

For porous media a lot of memory is wasted due to the fact that solid cells are not necessary for simulation. In general only the information that a neighboring cells is solid is needed. Hence the ILBDC code uses a *sparse representation* to store the domain. The fluid cells, i.e. the particle distribution functions f_i, are stored in a one-dimensional array along with an adjacency list.

The processing order of the cells is determined by their appearance in the 1D list, which depends on the schema chosen for numbering the cells during list's setup. Several schemas for numbering can be applied, like *lexicographic ordering* with or without blocking as shown in Fig. 3 or *discrete space-filling curves*, e.g. Z curve (Morton order) or Hilbert curve.



(a) Blocking factor 1.

(b) Blocking factor 2.

Figure 3: Different blocking factors as in (a) and (b) for a lexicographic ordering of cells yield different processing orders of fluid cells. Dashed cells represent solid material. For load balancing the list of fluid cells is cut into equally sized chunks. Cells with the same color represent such a chunk assuming that the size of chunk is five cells.

Load balancing Distributing the load equally is currently done by cutting the 1D list of fluid cells into equally sized chunks. This is done without any knowledge about the resulting topology of such a chunk as Fig. 3 illustrates. Beside other effects like inefficient cache line usage increased communication between chunks can hinder performance.

An alternative approach is the usage of graph partitioning algorithms for the decomposition. Hereby partitions can be created with nearly minimal surface and communication partners.

Graph partitioning To transfer the simulation domain into a graph fluid cells are considered as vertices. Velocity vectors $\mathbf{e}_i$ to neighboring fluid cells represent undirected edges. The complete transformation can be seen in Fig. 4. It is possible to consider just a reduced neighborhood for the graph. For the often used D3Q19 model six, 18, or all 26 surrounding neighbors can be chosen. On this graph a partitioning algorithm can then be applied.

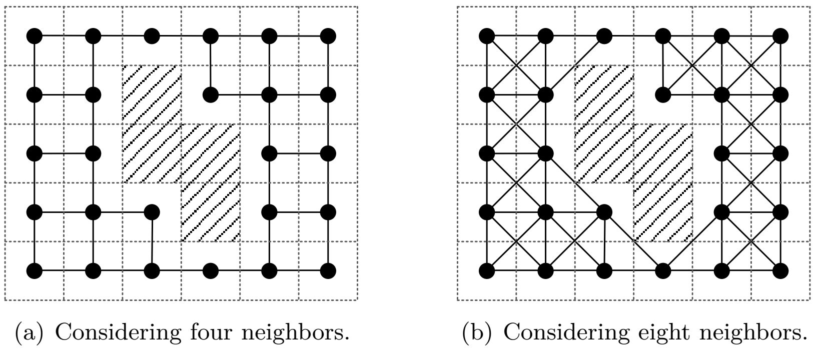

(a) Considering four neighbors. (b) Considering eight neighbors.

Figure 4: Transformation of simulation domain into an undirected graph. Fluid cells become vertices and velocity vectors $\mathbf{e}_i$ to non-solid neighbors become undirected edges. Depending on how much velocity vectors are used, like four in (a) or eight in (b), the resulting graph differs.

References

[1] P. L. Bhatnagar, E. P. Gross, and M. Krook. A Model for Collision Processes in Gases. I. Small Amplitude Processes in Charged and Neutral One-Component Systems. *Physical Review Online Archive (Prola)*, 94(3):511–525, May 1954.

[2] S. Chen and G. D. Doolen. Lattice Boltzmann Method for Fluid Flows. *Annual Review of Fluid Mechanics*, 30(1):329–364, 1998.

[3] Guy R. McNamara and Gianluigi Zanetti. Use of the Boltzmann Equation to Simulate Lattice-Gas Automata. *Physical Review Letters*, 61(20):2332–2335, November 1988.

[4] S. Succi. *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond.* Oxford University Press, August 2001.

[5] Gerhard Wellein, Thomas Zeiser, Stefan Donath, and Georg Hager. On the single processor performance of simple lattice Boltzmann kernels. *Computers & Fluids*, 35:910–919, 2006. 10.1016/j.compfluid.2005.02.008.

[6] Thomas Zeiser, Georg Hager, and Gerhard Wellein. Benchmark analysis and application results for lattice Boltzmann simulations on NEC SX vector and Intel Nehalem systems. *Parallel Processing Letters*, 19(4):491–511, 2009. 10.1142/S0129626409000389.

[7] Thomas Zeiser, Georg Hager, and Gerhard Wellein. Vector computers in a world of commodity clusters, massively parallel systems and many-core many-threaded cpus: Recent

experience based on an advanced lattice boltzmann flow solver. In Wolfgang E. Nagel, Dietmar B. Kröner, and Michael M. Resch, editors, *High Performance Computing in Science and Engineering '08*, pages 333–347. Springer Berlin Heidelberg, 2009. 10.1007/978-3-540-88303-6_24.