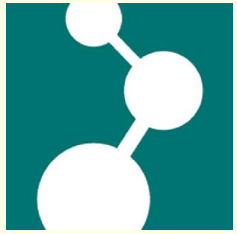


Max Planck Institute
for Polymer Research

Lattice Boltzmann implementation for ESPResSo

Ulf D. Schiller

uschille@mpip-mainz.mpg.de

July 14th, 2006

Overview

- Short introduction to Lattice Boltzmann
- Algorithm
- Integration in ESPResSo
- Data structures
- Parallelization
- Missing features and open issues

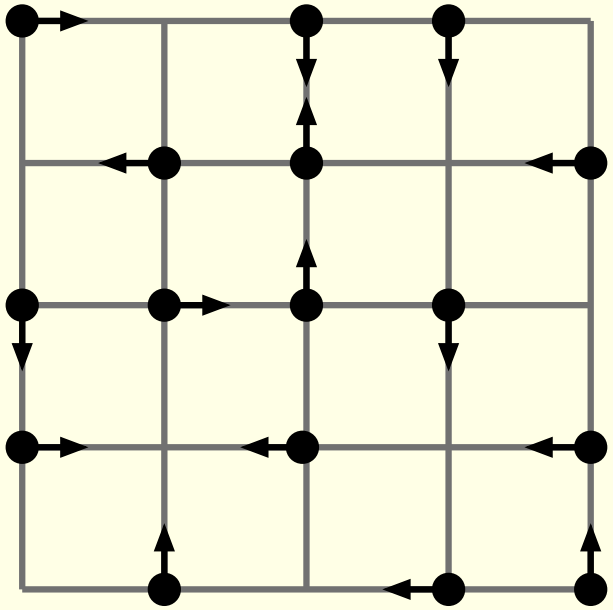
Introduction

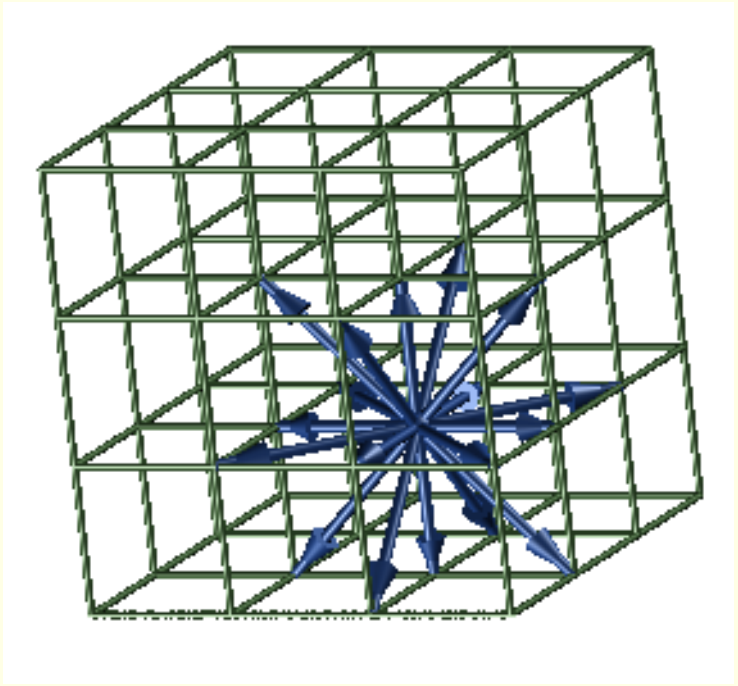
- Simulations of complex fluids have to account for hydrodynamic interactions
- Explicit solvent simulations are computationally demanding

$\longrightarrow$ Hybrid approach:

- Molecular Dynamics for the solute
- Lattice Boltzmann for the solvent
- Coupling via viscous momentum transfer

Lattice Boltzmann in $3 + \epsilon$ Minutes





- particle distributions evolving on a lattice
- streaming and collisions
- discrete set of velocities (D3Q18, D3Q19, etc.)

Lattice Boltzmann in $3 + \epsilon$ Minutes

- Boltzmann equation

$$\frac{\partial}{\partial t} f(\mathbf{x}, \mathbf{v}, t) + \mathbf{v} \cdot \nabla_{\mathbf{x}} f(\mathbf{x}, \mathbf{v}, t) = \mathcal{C} f(\mathbf{x}, \mathbf{v}, t) \tag{1}$$

$f(\mathbf{x}, \mathbf{v}, t)$: particle distribution function $\mathcal{C}$: collision operator

- space and time discretized version $\longrightarrow$ Lattice Boltzmann equation (LBE)

$$n_i(\mathbf{x} + \mathbf{v}_i \tau, t + \tau) = n_i(\mathbf{x}, \tau) + \sum_j L_{ij} \left(n_j - n_j^{\mathrm{eq}}(\rho, \mathbf{u}) \right) \tag{2}$$

$n_i(\mathbf{x}, t)$: particle population $\mathbf{v}_i$: velocity τ: time step L_{ij}: collision matrix

- mass & momentum conserved $\Rightarrow$ Navier-Stokes hydrodynamics

Lattice Boltzmann in $3 + \epsilon$ Minutes

- Hydrodynamic fields from populations

$$\rho = \sum_i n_i \qquad \rho \mathbf{u} = \sum_i n_i \mathbf{c}_i \qquad \mathbf{\Pi} = \sum_i n_i \mathbf{c}_i \otimes \mathbf{c}_i \tag{3}$$

- Populations from hydrodynamic fields (pseudo-equilibrium distribution)

$$n_i^* = A_i \rho + B_i \rho \mathbf{u} \cdot \mathbf{c}_i + C_i \Pi'_{\gamma\gamma} + D_i \Pi'_{\alpha\beta} c_{i\alpha} c_{i\beta} \tag{4}$$

Lattice Boltzmann in $3 + \epsilon$ Minutes

- Collisions $\longrightarrow$ relaxation of $\mathbf{\Pi}$ (incompressible limit)

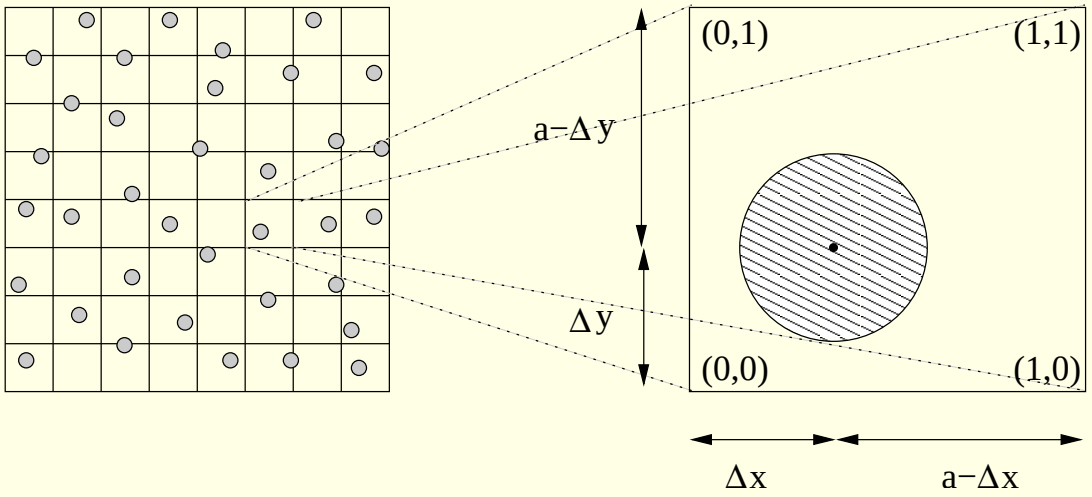
$$\Pi'_{\alpha\beta} = \Pi^{\mathsf{eq}}_{\alpha\beta} + (1 + \lambda)\left(\overline{\Pi}_{\alpha\beta} - \overline{\Pi}^{\mathsf{eq}}_{\alpha\beta}\right) \tag{5}$$

  λ: eigenvalue of collision matrix L_{ij}

- Streaming $\longrightarrow$ propagation on the lattice

$$n_i(\mathbf{x} + \mathbf{c}_i, t + \tau) = n_i^*(\mathbf{x}, t) \tag{6}$$

Coupling of particles and fluid

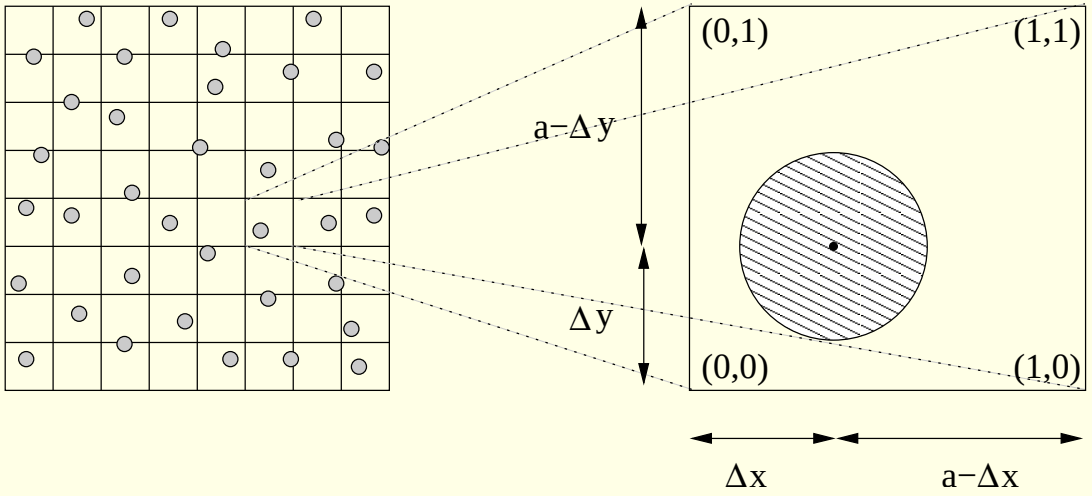


- Idea: treat monomers as point particles and apply Stokesian drag

$$\mathbf{F} = -\zeta \left[\mathbf{V} - \mathbf{u}(\mathbf{R}, t)\right] \tag{7}$$

- linear interpolation to determine $\mathbf{u}(\mathbf{R}, t)$
- ensure momentum conservation by transferring momentum to the fluid
- dissipative forces $\longrightarrow$ add stochastic forces to fulfill fluctuation-dissipation relation

Coupling of particles and fluid



- interpolation scheme for $\mathbf{u}(\mathbf{R}, t)$

$$\mathbf{u}(\mathbf{R}, t) = \sum_{\mathbf{x} \in \text{Cell}} \delta_{\mathbf{x}} \mathbf{u}(\mathbf{x}, t) \tag{8}$$

- momentum transfer

$$-\frac{1}{a^3}\mathbf{F} = \frac{\Delta \mathbf{j}}{\Delta t} = \frac{\mu}{a^2 \tau \Delta t} \sum_{\mathbf{x} \in \text{Cell}} \sum_{i} \Delta n_i(\mathbf{x}, t) \mathbf{c}_i \tag{9}$$

Algorithm

1. Lattice Boltzmann Dynamics (time step τ)
   (a) collision step
   (b) streaming step

2. Coupling to Molecular Dynamics (time step Δt)
   (a) calculate particle force
   (b) transfer momentum to fluid

Integration in ESPResSo

- Lattice Boltzmann Dynamics → Integration loop

```
void integrate_vv(int n_steps)
{

...

#ifdef LB
  if (lattice_switch & LATTICE_LB) lb_propagate();
  if (check_runtime_errors()) break;
#endif

...

}
```

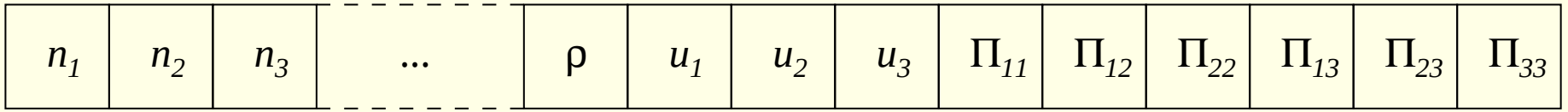
Integration in ESPResSo

- Coupling → Force calculation

```
void force_calc()
{

...

#ifdef LB
  if (lattice_switch & LATTICE_LB) calc_particle_lattice_ia() ;
#endif

...

}
```

Data Structures

- most operations are local → store data for lattice nodes contiguously



- data access via struct holding pointers (e.g. `node[i].n[k]`)

```
typedef struct {

  double *n;
  double *rho;
  double *j;
  double *pi;

} LB_FluidNode;
```

- leads to an additional indirection → inefficient? (discussion)

Data Structures

- General data structure for lattice algorithms

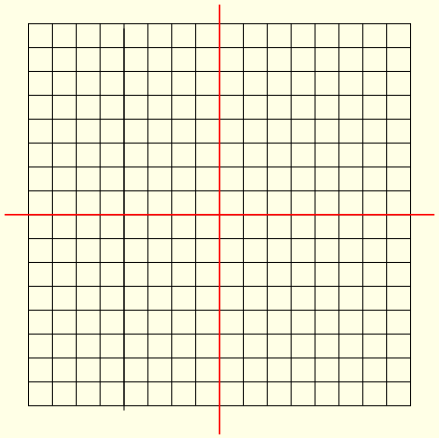
```
typedef struct _Lattice {

  int grid[3] ;              // dimensions of the lattice
  int halo_grid[3] ;

  int grid_volume ;          // volume of the lattice
  int halo_grid_volume ;
  int halo_grid_surface ;
  int halo_offset ;

  double agrid ;             // lattice spacing
  double tau ;               // lattice time step

  void *fields;              // pointer to the structs holding pointers
  void *data;                // pointer to the actual data

} Lattice;
```

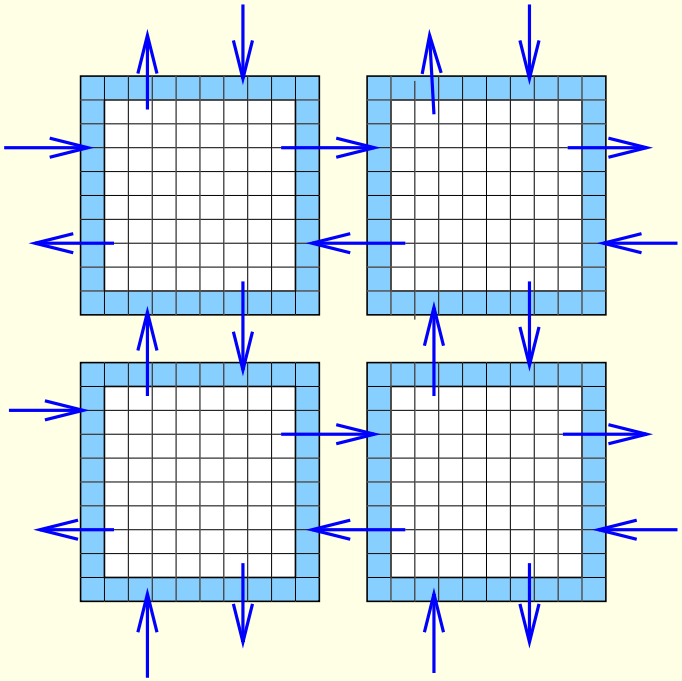
Data Structures

New lattice algorithms can be easily implemented:

- specify a struct for the data on the lattice nodes
- write routines for memory allocation and data initialization (`init`-routines)
- convenient data access in the algorithm (`lattice.fields.<whatever>`)
- direct (low-level) data access is also possible (`(<type_cast>*)lattice.data[index]`)

Parallelization





- domain decomposition scheme
- communication of halo regions between processors

Parallelization

- abstract halo communication scheme (independent of lattice data)

```
typedef struct {

  int type;               /**< type of halo communication */

  int source_node;        /**< index of processor which sends halo data */
  int dest_node;          /**< index of processor receiving halo data */

  void *send_buffer;      /**< pointer to data being sent */
  void *recv_buffer;      /**< pointer to data being received */

  Fieldtype fieldtype;    /**< type layout of the data beeing exchanged */
  MPI_Datatype datatype; /**< MPI datatype of data beeing communicated */

} HaloInfo ;
```

Parallelization

Usage of halo communication scheme:

- specify data layout of the lattice nodes (`fieldtype`, `datatype`)
- initialize communicator:

```
prepare_halo_communication(&halo_comm,&lattice,fieldtype,datatype);
```

- use it:

```
halo_communication(&halo_comm);
```

(this works, if domain decomposition scheme is feasible for the algorithm)

TCL Commands

- setting up the Lattice Boltzmann fluid

  `lbfluid ( agrid | tau | density | viscosity | friction | ext_force <value> )*`

- analysis routines for the fluid

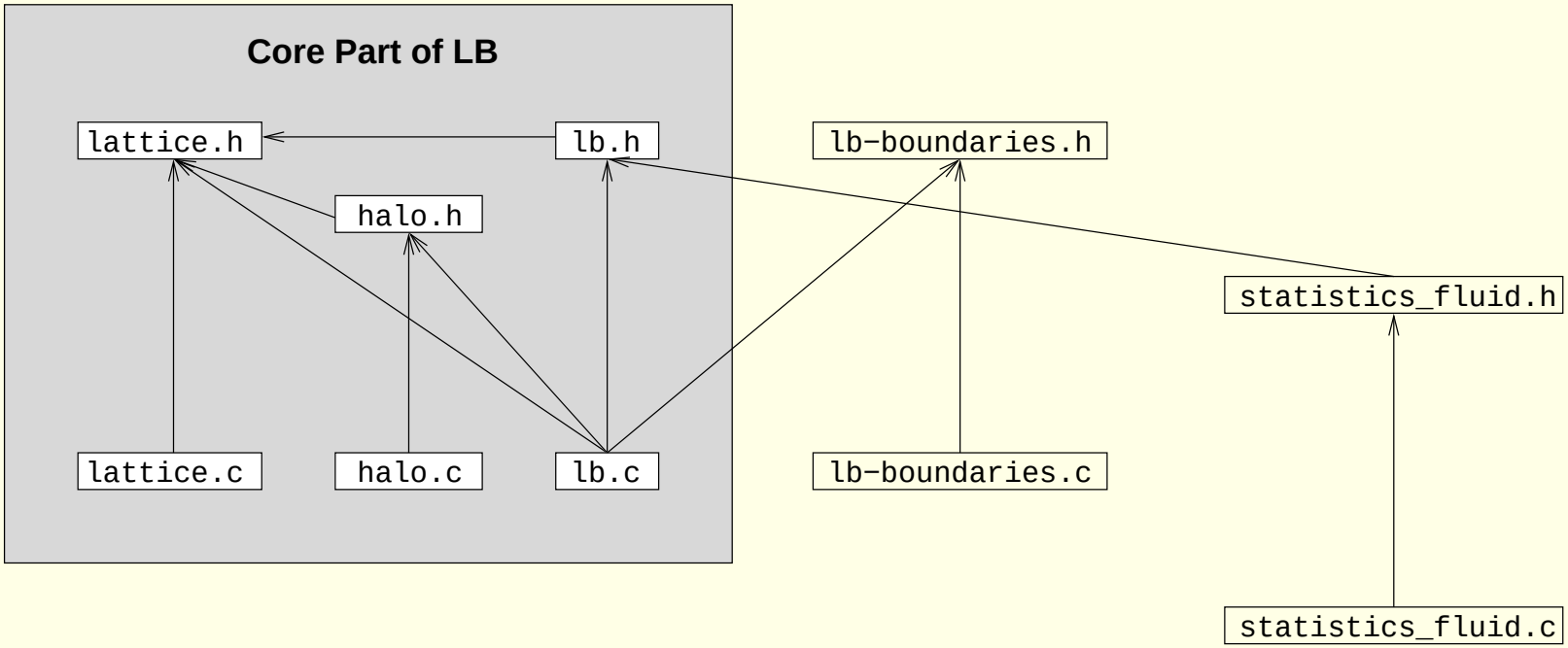
  `analyze fluid mass`

  `analyze fluid momentum`

  `analyze fluid temperature` (work in progress $\longrightarrow$ definition of fluid temperature?)

  `analyze fluid velprof`

File Structure of LB Implementation



Missing Features

- Checkpoints
- Analysis routines: Output of flow field, etc.
- Fourier transformation into k-space (FFTW $\rightarrow$ Torsten Stuehn, discussion)
- Flexible interactions: couple specific particle types ($\rightarrow$ virtual particle ID for fluid?)
- Boundary conditions (partly implemented)
- Simulation of flows (partly implemented)

Open Issues

- Performance benchmarks
- Reproducible trajectories ($\longrightarrow$ random numbers)
- Momentum drift on Regatta ($\longrightarrow$ `float=maf`)
- Correct fluctuations?
- Accuracy of integrator?
- ...