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

→ 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 implementation for ESPResSo
Lattice Boltzmann in $3 + \epsilon$ Minutes

- Boltzmann equation

$$\frac{\partial}{\partial t} f(x, v, t) + v \cdot \nabla_x f(x, v, t) = C f(x, v, t)$$ (1)

$f(x, v, t)$: particle distribution function $\quad C$: collision operator

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

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

$n_i(x, t)$: particle population $\quad v_i$: velocity $\quad \tau$: time step $\quad L_{ij}$: collision matrix

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

Lattice Boltzmann implementation for ESPResSo
Lattice Boltzmann in \( 3 + \epsilon \) Minutes

- Hydrodynamic fields from populations

\[
\begin{align*}
\rho &= \sum_i n_i \\
\rho u &= \sum_i n_i c_i \\
\Pi &= \sum_i n_i c_i \otimes c_i
\end{align*}
\]

(3)

- Populations from hydrodynamic fields (pseudo-equilibrium distribution)

\[
n_i^* = A_i \rho + B_i \rho u \cdot c_i + C_i \Pi_{\gamma\gamma}' + D_i \Pi'_{\alpha\beta} c_{i\alpha} c_{i\beta}
\]

(4)
Lattice Boltzmann in $3 + \epsilon$ Minutes

- Collisions $\rightarrow$ relaxation of $\Pi$ (incompressible limit)

\[
\Pi'_{\alpha\beta} = \Pi^\text{eq}_{\alpha\beta} + (1 + \lambda) \left( \Pi_{\alpha\beta} - \Pi^\text{eq}_{\alpha\beta} \right)
\]  

(5)

$\lambda$: eigenvalue of collision matrix $L_{ij}$

- Streaming $\rightarrow$ propagation on the lattice

\[
n_i(x + c_i, t + \tau) = n_i^*(x, t)
\]  

(6)
Coupling of particles and fluid

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

\[ \mathbf{F} = -\zeta [\mathbf{V} - \mathbf{u}(\mathbf{R}, t)] \]  

(7)

• linear interpolation to determine \( \mathbf{u}(\mathbf{R}, t) \)

• ensure momentum conservation by transferring momentum to the fluid

• dissipative forces \( \rightarrow \) add stochastic forces to fulfill fluctuation-dissipation relation

Lattice Boltzmann implementation for ESPResSo
Coupling of particles and fluid

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

\[
    u(\mathbf{R}, t) = \sum_{x \in \text{Cell}} \delta_x u(x, t)
\]  \hspace{1cm} (8)

- momentum transfer

\[
    - \frac{1}{a^3} \mathbf{F} = \frac{\Delta j}{\Delta t} = \frac{\mu}{a^2 \tau \Delta t} \sum_{x \in \text{Cell}} \sum_i \Delta n_i(x, t) c_i
\]  \hspace{1cm} (9)
Algorithm

1. Lattice Boltzmann Dynamics (time step $\tau$)
   
   (a) collision step
   (b) streaming step

2. Coupling to Molecular Dynamics (time step $\Delta t$)
   
   (a) calculate particle force
   (b) transfer momentum to fluid
Integration in ESPResSo

- Lattice Boltzmann Dynamics → Integration loop

```c
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

```c
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

|  \( n_1 \) |  \( n_2 \) |  \( n_3 \) | ... | \( \rho \) | \( u_1 \) | \( u_2 \) | \( u_3 \) | \( \Pi_{11} \) | \( \Pi_{12} \) | \( \Pi_{22} \) | \( \Pi_{13} \) | \( \Pi_{23} \) | \( \Pi_{33} \) |

- 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

```c
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)

```c
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 being exchanged */
    MPI_Datatype datatype; /**< MPI datatype of data being 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 → definition of fluid temperature?)

  analyze fluid velprof
File Structure of LB Implementation

Core Part of LB

- lattice.h
- halo.h
- lb.h
- lb_boundaries.h
- lattice.c
- halo.c
- lb.c
- lb_boundaries.c

statistics_fluid.h
- statistics_fluid.c

Lattice Boltzmann implementation for ESPResSo
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 (→ random numbers)
- Momentum drift on Regatta (→ float=maf)
- Correct fluctuations?
- Accuracy of integrator?
- ...