Master Thesis

General boundary condition in dissipative particle dynamics

Author(s):
Altenhoff, Adrian M.

Publication Date:
2005

Permanent Link:
https://doi.org/10.3929/ethz-a-005146314

Rights / License:
In Copyright - Non-Commercial Use Permitted
Master Thesis

General Boundary Condition in Dissipative Particle Dynamics

Adrian M. Altenhoff
adrian.altenhoff@gmail.com

ETH Zürich, September 30, 2005

Faculty: Institute of Computational Science
Prof. Petros Koumoutsakos
Author: Adrian M. Altenhoff
Supervisor: Dr. Jens H. Walther
Abstract

Dissipative particle dynamics (DPD) is a particle-based mesoscopic simulation method to model hydrodynamic behavior in the field of complex fluids. Here we present a novel method for the imposition of the no-slip boundary condition at fluid-solid interfaces. The method uses the probability distribution function of the individual force contributions as measured in separate simulations. The results measured for exemplary steady and unsteady flow patterns demonstrate a significant reduction in the density fluctuations previously observed. In addition, the method meets the no-slip condition with a negligible impact on the temperature of the wall.
Contents

1 Introduction 3

2 Dissipative Particle Dynamics 3

3 Periodic DPD-Client 4

4 Overview of Boundary Conditions 7

5 A New Wall Boundary Condition using Wall Potentials 7
  5.1 Semi Analytical Approach 8
  5.2 Measured Force Probabilities 10
  5.3 Particle Reflexion Mechanism 11

6 Results of the New Boundary Method 11
  6.1 Fluid at Rest 13
  6.2 Couette Flow 13
  6.3 Poiseuille Flow 14
  6.4 Stokes Flow Near an Oscillating Flat Plate 15
  6.5 Lid-driven Cavity Flow 16

7 Conclusion 16

8 Future Work 18

A Overview of the ppm DPD-client 19
  A.1 Introduction 19
  A.2 Input / Output 19
    A.2.1 Required Input Files 19
    A.2.2 Generated Output Files 21
    A.2.3 Additional Tools 22
  A.3 Flow Charts 23
  A.4 DPD Modules 23
    A.4.1 dpd_module_boundary 23
    A.4.2 dpd_module_comp 27
    A.4.3 dpd_module_jo 28
    A.4.4 dpd_module_profiler 29
    A.4.5 dpd_module_ranDist 29
    A.4.6 dpd_module_random 30
    A.4.7 dpd_module_rdf 30
  A.5 Known bugs 30
1 Introduction

The modelling of complex fluids and multiphase flows requires microscopic (molecular) information of the fluid behaviour. Direct numerical simulation of these systems may thus be performed using Molecular Dynamics (MD) simulations. However, flow phenomena occurring on a meso-scale often depend only on a few aspects of molecular behaviour. Therefore MD simulations are too detailed and computationally very expensive. On the other hand, conventional continuum-based simulation techniques are based on solving the Navier–Stokes equations and have difficulties to model the necessary microscopic detail. To improve the computational efficiency, the dynamic behaviour of molecules may be combined into a mesoscale description, that is especially well-suited for problems having a time- or length scales either too big or too small for the aforementioned methods. Dissipative Particle Dynamics (DPD) is one example of a mesoscale method. It is particle-based and was initially introduced by Hoogerbrugge and Koelman [7] to avoid the lattice artefacts of Lattice Gas Automata and in the same time to capture hydrodynamic time and space scales much larger than those available with MD. In the last decade, the method has been improved, generalised and has been applied to a wide variety of complex fluid systems. Flekkøy et al. [4] also provide a link between the DPD forces and the hydrodynamics of the underlying molecular system.

Previous DPD studies have mainly considered fluids in periodic systems. Solid walls have been modeled using layers of frozen “solid” DPD-particles, combined with reflecting schemes to prevent leakage of fluid DPD-particles out of the system. The desired behaviour of such a wall depends on the length scale of the system. Most problems currently addressed to DPD simulations, take place on a length scale where molecular effects of the wall on the fluid should not be visible. Our central goal in the present work is to provide a solid wall on such a length scale. The wall should be impermeable (i.e. no particles are allowed to enter the wall), show no slip (i.e. the wall should impose the correct velocity on the particles) and the wall should not affect the properties of the fluid, like pressure, density or temperature. Most of the current wall schemes induce strong density variations of up to 100% which reduces the bulk density in small confined systems or the schemes are not generally applicable.

2 Dissipative Particle Dynamics

The DPD method describes a fluid in a coarse-grained fashion, by representing molecular clusters as a single particle which shows the collective dynamic behaviour of all the molecules it contains. The time evolution of DPD particles is described by Newton’s equation of motion

\[
\frac{d\mathbf{r}_i}{dt} = \mathbf{v}_i, \quad \frac{d\mathbf{v}_i}{dt} = \mathbf{f}_i
\]

where the mass of the particles is unity. The total force acting on a particle \(i\) is composed of three pairwise additive forces, a conservative (\(\mathbf{F}^C_{ij}\)), a dissipative (\(\mathbf{F}^D_{ij}\)) and a stochastic force (\(\mathbf{F}^R_{ij}\))

\[
\mathbf{f}_i = \sum_{j \neq i} \left( \mathbf{F}^C_{ij} + \mathbf{F}^D_{ij} + \mathbf{F}^R_{ij} \right).
\]

All forces are truncated beyond a finite cutoff radius \(r_c\), which is the intrinsic length scale of the system. The conservative force is given by a soft repulsion:

\[
\mathbf{F}^C_{ij} = \begin{cases} 
a_{ij} \left(1 - \frac{r_{ij}}{r_c} \right) \hat{r}_{ij} & r_{ij} < r_c, \\
0 & r_{ij} \geq r_c,
\end{cases}
\]

where \(r_{ij} = |\mathbf{r}_i - \mathbf{r}_j|\), \(\hat{r}_{ij} = \mathbf{r}_{ij}/r_{ij}\) and \(a_{ij}\) is the maximum repulsion between a pair of particles. Soft interactions have been proposed by Forrest and Suter [5] and are based on the average behaviour of atoms during short time intervals.
The dissipative and the stochastic force

\[ F^D_{ij} = -\gamma \omega^D(r_{ij}) \left( \mathbf{r}_{ij} \cdot \mathbf{v}_{ij} \right) \mathbf{r}_{ij}, \]

\[ F^R_{ij} = \sigma \omega^R(r_{ij}) \xi_{ij} \mathbf{r}_{ij}, \]

where \( \omega^D \) and \( \omega^D \) are distance dependent weight functions vanishing for \( r > r_c = 1 \), \( \mathbf{v}_{ij} = \mathbf{v}_i - \mathbf{v}_j \) is the relative velocity between the particles, \( \sigma \) the noise amplitude, \( \gamma \) the friction coefficient and \( \xi_{ij} = \xi_{ji} \) are normally distributed random fluctuations with zero mean and variance \( 1/\Delta t \).

Español and Warren [2] showed that one of the two weight functions can be chosen arbitrarily and that this choice fixes the other. Thus the system relaxes to a Gibbs–Boltzmann equilibrium distribution [2] if the dissipative and stochastic forces are balanced according to the fluctuation–dissipation theorem

\[ \omega^D(r) = \left[ \omega^R(r) \right]^2, \quad \text{and} \quad \sigma^2 = 2\gamma k_B T. \]

As a simple choice we take

\[ \omega^D(r) = \begin{cases} (1-r)^2 & r < 1, \\ 0 & r \geq 1, \end{cases} \]

Several methods for integrating the DPD evolution exist (see [11, 13]). The present implementation uses a modified velocity–Verlet algorithm, presented by Groot and Warren [6], commonly used in DPD simulations. Thus

\[ \mathbf{r}_i(t + \Delta t) = \mathbf{r}_i(t) + \Delta t \mathbf{v}_i(t) + \frac{1}{2}(\Delta t)^2 \mathbf{f}_i(t) \]

\[ \mathbf{v}_i(t + \Delta t) = \mathbf{v}_i(t) + \lambda \Delta t \mathbf{f}_i(t) \]

\[ \mathbf{f}_i(t + \Delta t) = \mathbf{f}_i(t + \Delta t), \mathbf{v}_i(t + \Delta t) \]

\[ \mathbf{v}_i(t + \Delta t) = \mathbf{v}_i(t) + \frac{1}{2} \Delta t(\mathbf{f}_i(t) + \mathbf{f}_i(t + \Delta t)). \]

where \( \lambda \) is an empirical factor. Groot and Warren [6] observed that the algorithm behaves best with \( \lambda = 0.65 \). As the error in the temperature does not converge monotonically but increases again when the time step is decreased, we decided to perform all the reported simulations in this work with \( \lambda = 1/2 \). The velocity dependent particle force requires an additional corrector step to achieve second order. Nevertheless, the forces still have to be computed only once per time step.

Without the stochastic or the dissipative force, this integration scheme would be exact up to order \( O(\Delta t^2) \) at \( \lambda = 1/2 \). Because of the stochastic nature of the DPD process, the order of the algorithm becomes unclear.

### 3 Periodic DPD-Client

We have implemented a DPD client within the framework of the ppm-library [18]. The current implementation is limited to mono-particle fluids and has been validated on standard test cases in fluid mechanics. Special care has to be taken on how to generate the stochastic force Eq. (5) between two particles, i.e. the relation \( \xi_{ij} = \xi_{ji} \) must hold to ensure momentum conservation. To assure reproducibility (same sequence of pseudo random numbers) during parallel simulations, we introduce a new method to ensure momentum conservation.

We have achieved this by storing a seed \( s_i \) for each particle. When two particles interact, the seed for the random number generator is chosen from a mapping \( s_i \otimes s_j \) of the particles’ seeds. A simple and fast choice for the mapping function is the “sum” or the “xor” of the individual seeds. After each time step, the particle’s seed is updated according to the algorithm of the particular random number generator, in order to generate new random numbers in the next time step.
Figure 1: Autocorrelation of the generated Gaussian random numbers $\xi_{ij}$ (red) after using the sum as a mapping function for the particles’ seed and for a series of uniform random numbers (green). The bounds give a 95% confidential interval for white noise. No correlation can be measured.

The built-in random number generator of the Intel Fortran compiler\(^1\) does not behave as expected, i.e. the system temperature does not converge to the expected mean temperature when the time step is reduced; it remains 5% above the expected value. Choosing the random number generator proposed by Knuth [8]

$$I_{j+1} = aI_j + c \pmod{2^{32}}$$

with $a = 1664525$ and $c = 1013904223$ resolved this problem. Fig. 1 shows a sample autocorrelation of the random numbers generated for a particle. No correlation can be observed.

To validate the implementation (ppm client) for the periodic system, we compared a number of observables to analytical solutions of previously reported results. Thus, the temperature $k_B T$ of the system, whose conservation is a main condition for reliable simulations, is measured as

$$\langle k_B T \rangle = \frac{\langle \mathbf{v}^2 \rangle}{3} \quad (9)$$

where $\langle \ldots \rangle$ is a an ensemble average over all particles. As the temperature is both, an input and an output parameter, the difference is an indicator for the quality of the simulation. Fig. 2 shows the deviations of the measured to the expected temperature for different time steps. For $\Delta t \leq 0.02$ the error is less than 1%.

As a next test, the radial distribution function $g(r)$ has been studied. It is defined as the ratio of the actual density of particles in a shell at distance $r$ around a test particle to the mean density and is a well suited quantity to study structural properties of a fluid. In a DPD-fluid without any conservative forces, i.e. $a_{ij} = 0$, referred to as an ideal DPD-fluid, the radial distribution function provides an excellent test for the integrator, since $g(r) \equiv 1$. In Fig. 3(a) the measured $g(r)$ with $\Delta t = 0.01$, $k_B T = 1$ and $\sigma = 3$ is shown. The deviations from 1 are less than 4% and only present at small ($r < 0.2$) distances and similar to the results obtained in Ref. [11].

For DPD-fluids with conservative interactions, i.e. $a \neq 0$, there does not exist any theoretical value for the radial distribution function. We show in Fig. 3(b) a measured $g(r)$ for $a = 25$ and $\rho = 4$, together with the one measured by Groot and Warren [6]. Our values are very close to the ones reported previously. This is an important error measure as our proposed boundary treatment (see Section 5.1) is based on the $g(r)$.

\(^1\)Intel Fortran Compiler, Version 8.1, on a Linux x86 architecture.
Figure 2: Temperature as a function of time step for $\sigma = 3$, $\rho = 4$. Shown are results for the Euler-algorithm (red, green) and the modified Velocity Verlet algorithm (black, blue) of our implementation (solid) and results published in [6] (dotted).

Figure 3: Radial distribution function for an ideal DPD fluid (a) of our implementation (blue) and the results presented by Nikunen et al. [11] (green). The results for the repulsion coefficient $a = 25$ and $\rho_n = 3$ is shown in (b). The red line are measured values by us, compared with the result of Groot and Warren [6] (blue points).
4 Overview of Boundary Conditions

Currently, the main challenge in DPD simulations is to provide accurate boundary schemes for the imposition of the no-slip boundary condition at fluid-solid interfaces. In the last few years there has been an intense focus on modelling such boundaries. In this section we want to provide a brief overview over previous published methods.

In 1999, Revenga et al. [16] proposed a general classification for the main approaches to impose no-slip boundary conditions as follows:

1. The Lees-Edwards method [9, 1] to impose planar shear.
2. Freezing regions of the fluid to create a rigid wall.
3. Combine layers of particles with reflection mechanism.

The Lees-Edwards method is essentially a way to avoid directly modelling the physical boundary. This method assumes a shear flow imposed through an upper wall moving with velocity $U_x$ and a lower wall moving in the opposite direction with the velocity $-U_x$. A particle that is leaving the computational domain through the upper boundary is inserted similar to the periodic boundary method through the lower one, but, as proposed by Lees and Edwards [9], at a position shifted by $-2U_x t$ and adjusted velocity $-2U_x$. In a recently published method by Visser et al. [19] they extend the idea of the Lees-Edwards method and describe a general way on how to map two or more twin systems back-to-back to produce the desired wall behaviour.

Freezing regions were already introduced by Hoogerbrugge et al. [7] for DPD simulations. They have used this method to simulate a flow through a square array of cylinders. In a similar way, confined simulation domains can be modelled.

The third category is quite broad. All the methods have in common, that they use immobile particles to model a wall. Because of the soft forces, the wall particles cannot prevent the particles from leaking the domain unless the wall density is increased, the repulsion parameter $a_{ij}$ is increased for wall particles or a reflection mechanism is applied to particles leaving the simulation domain. Naturally, these schemes can also be combined. Pivkin et al. [14] have compared different ways to combine them. While these schemes succeed in confining the particles and satisfying the no-slip boundary, they cannot suppress the generation of large density fluctuations which persist beyond a cutoff distance as one can see in Fig. 4.

5 A New Wall Boundary Condition using Wall Potentials

The enforcement of the no-slip boundary condition is currently ensured by placing immobile “wall” particles in the solid region combined with bounce-back reflection. These methods all have the disadvantage, that they introduce strong density fluctuations near the walls. Besides this, frozen particles are a waste of computational resources.

If solid walls have to be modelled, two important points have to be taken into account:

1. The hard walls confines the particles, i.e. secures conservation of mass and secures the kinetic pressure by conserving the velocity (i.e. the temperature).
2. Point (1) alone excludes the viral term which is important for dense fluids and results in strong density fluctuations at the fluid–solid interface, cf. Fig. 4.

This viral contribution we approximate using an effective wall force $F_m$ as proposed by Werder et al. [20], i.e.,

$$P = P_K + P_U = \rho_n k_B T + \rho_n \int_0^r F_m(r) dr$$

where $P_K$ denotes the ideal part of the pressure.
Figure 4: Density profile obtained by Pivkin et al. [14] in a Poiseuille flow scenario (red). There are large density fluctuations of approximately 100%. The dashed line shows the exact uniform density.

5.1 Semi Analytical Approach

We propose to apply an effective mean boundary force that minimises local disturbance (e.g. in the density or temperature fields) by accounting for local structures of the fluid, which is, for monoatomic fluids, described by the radial distribution function \( g(r) \), cf. Section 3. In this model, we integrate to satisfy the mean pressure the force components normal to the wall and, for the no-slip condition, the tangential force components, weighted by \( g(r) \) over the part of the radial cutoff that lies outside of the computation domain - therefore the contributions to the virial pressure is a cutoff sphere, cf. Fig. 5.

As the conservative force \( \Psi_{ij}^C \) only depends on the position of the particles, we can integrate the components itself using polar coordinates. Thus the normal conservative wall force component is

\[
\Psi_{N}^C(r_w) = 2\pi \rho_n \int_{z=r_w}^{0} \int_{x=0}^{r_c} g(r) F^C_N(r) \frac{z}{r} x \, dx \, dz
\]

where \( r = \sqrt{x^2 + z^2} \). The tangential component vanish due to symmetry.

Because the dissipative force components depend also on the relative velocity of the particles, we normalise the force by the magnitude of the relative velocity. The dissipative wall force components then becomes

\[
\Psi_{N}^D(r_w) = 2\pi \rho_n \int_{z=r_w}^{0} \int_{x=0}^{r_c} g(r) \frac{F^D_N(r)}{v_{ij}} \frac{z}{r} x \, dx \, dz
\]

\[
\Psi_{T}^D(r_w) = \rho_n \int_{z=r_w}^{0} \int_{-\sqrt{r_c^2-z^2}}^{\sqrt{r_c^2-z^2}} \int_{-\sqrt{r_c^2-z^2-y^2}}^{\sqrt{r_c^2-z^2-y^2}} g(r) \frac{F^D_T(r)}{v_{ij}} \frac{|x|}{r} \, dx \, dy \, dz
\]

with \( v_{ij} = |v_{ij}| \). Later on during the simulation, we assume that the wall consists of a Maxwellian distributed particle pool. A particle \( i \) coming into the interaction range of the wall will feel a force \( F_{m,i}^D \)

\[
F_{m,i}^D = \left( 2 - \frac{r_w}{r_c} \right) \Psi_{T}^D(r_w) (\theta - v_i)
\]

where \( \theta \) is a maxwellian distributed random number. The factor \( \left( 2 - \frac{r_w}{r_c} \right) \) is a simple way to approximately extend the velocity profile of the computational domain into the wall, which is necessary to fulfill the no-slip condition (see [21]). The idea of the factor is, that a particle far away from the wall sees the wall being at rest and the velocity difference is simply it’s own
Figure 5: Integration domains for effective boundary forces. \( r_c \) is the cutoff distance and \( r_w \) the
distance to the wall. The normal force contributions along \( z \) are integrated over the shaded area
using polar coordinates, the tangential force contributions along \( x \) using cartesian coordinates.

velocity. If in contrast a particle is close to the wall, the particles outside the wall should move
in the opposite direction. By the choice of 2 in the factor, we assume the particle outside the
domain would move with the same velocity in the other direction, cf. Fig. 6. We expect even
better results by using a more sophisticated way to implement the extension of the velocity
profile, e.g. by using the running average of the particles’ velocity to model the slope near the
boundary.

As a last contribution of the force exerted by the wall, we look at the stochastic term of
the pair force Eq. (5). Since this is a random force with zero mean, the normal as well as the
tangential components of the stochastic force are zero. Integrating the force over the cutoff sphere
would therefore lead to a mean wall force that is zero again. But of course, the contributions of
the stochastic force are necessary. The important point is the non zero variance. Therefore we
describe the force as a single random variable, that has a zero mean and a distance depending
variance, i.e.

\[
F_{ij}^R = \theta(r_{ij}) \text{ and } \text{Var} (\theta(r_{ij})) = \sigma \omega^R(r_{ij}) \frac{1}{\Delta t} \hat{\mathbf{r}}_{ij}
\]

The integral for the stochastic contributions to the mean wall force becomes

\[
\Psi_N^R(r_w) = 2\pi \rho_n \int_{z=r_w}^{r_c} \int_{x=0}^{\sqrt{r_c^2-z^2}} g(r) \text{Var} (\theta(r)) \frac{z}{r} dx \, dz \tag{15}
\]

\[
\Psi_T^R(r_w) = \rho_n \int_{z=r_w}^{r_c} \int_{\sqrt{r_c^2-z^2}}^{\sqrt{r_c^2-z^2-y^2}} \int_{-\sqrt{r_c^2-z^2-y^2}}^{\sqrt{r_c^2-z^2-y^2}} g(r) \text{Var} (\theta(r)) \left| \frac{x}{r} \right| dx \, dy \, dz \tag{16}
\]

The total normal and tangential forces are:

\[
F_{m,i}^N(r) = \Psi_N^C(r) + \left( 2 - \frac{r_w}{r_c} \right) \Psi_N^D(r_w) (\theta - v_{N,i}) + \Psi_N^R(r_w) \zeta
\]

\[
F_{m,i}^T(r) = \Psi_T^C(r) + \left( 2 - \frac{r_w}{r_c} \right) \Psi_T^D(r_w) (\theta - v_{T,i}) + \Psi_T^R(r_w) \zeta \tag{17}
\]

where \( \zeta \) and \( \theta \) are both random variables with zero mean and variance 1 and \( k_B T \) respectively.
All these wall force contributions can also be obtained by sampling the fluctuating forces in both
Figure 6: Sketch of the wall force factor used in Eq. (14). A particle near the wall with a certain instantaneous velocity (red arrow) should feel the force (green + yellow). With our approximation, the wall moves with a distance from the wall depending velocity into the opposite direction (green+red).

Figure 7: Comparison of the ensemble average (solid lines) and the g(r) approach (circles) for the normal (red) and tangential (blue) wall forces acting on a DPD particle at a distance r from the fictitious wall. Shown are the contributions of the conservative, dissipative and stochastic components (from left to right).

the normal and tangential directions, acting on a fictitious wall in a periodic simulation at the same state point. We adopted both approaches, compared them to each other and found an excellent agreement for the conservative and the dissipative force. Deviations in the variance of the stochastic force are quite big, as one can see in Fig. 7. Probably, they arise from uncertainty in the radial distribution function and in the proceeding work only the measured forces are used.

5.2 Measured Force Probabilities

The g(r) approach is useful as it elucidates the mean forces — but it is not clear how to extend this to take into account the fluctuations of the conservative and dissipative forces. This can only be achieved by sampling the probability function of the force components in the periodic simulation. Fig. 8 shows the density functions for all the force contributions. Further analysis of the skewness and the kurtosis indicate, that these distributions are in general non-Gaussian (cf. Fig. 9) with a non-zero mean. Moreover, the measured forces at a certain distance are not correlated in time.

Thus, our final way to implement the boundary condition is therefore to sample the proba-
Figure 8: Probability of the normal (top) and tangential (bottom) of the conservative, dissipative and stochastic forces (from left to right).

probability distributions for each of the force contributions and to apply a wall force in the following way:

\[
F_i^W(r_w) = \left( \frac{F_{N,i}^W}{F_{T,i}^W} \right) = \begin{pmatrix}
\chi_N(r_w) + \left( 2 - \frac{r_w}{r_c} \right) \chi_N(r_w) (\theta - \nu_{N,i}) + \chi_N(r_w) \\
\chi_T(r_w) + \left( 2 - \frac{r_w}{r_c} \right) \chi_T(r_w) (\theta - \nu_{T,i}) + \chi_T(r_w)
\end{pmatrix}
\]

(18)

where \( \chi \) is a random variable according to the sampled probability distribution and \( \theta \) a normal distributed random number with variance \( k_BT \).

5.3 Particle Reflexion Mechanism

Because of the soft potential used in DPD, the wall force itself cannot prevent the particles from leaving the computational domain. For impermeable walls this behaviour is not desirable. Particles that are crossing the wall have therefore to be reinsert into the computational domain. Revenga et al. [15, 16] investigated specular reflection, maxwellian reflection and bounce-back reflection. Visser et al. [19] introduced the bounce-forward as an other method, that should diminish the effect of the wall on the temperature of the fluid, but still produce impermeability and no-slip. We have adopted this method, where the position of the particle is updated in the same manner as in the specular reflection but velocity and acceleration are inverted, cf. Fig. 10(a). Nevertheless, we can still observe a slight influence of the mechanism to the temperature of the fluid: We placed a soft wall in the middle of a periodic system. The reintroduction mechanism is the only thing, that prevents the particles from crossing the soft wall. As one can see in Fig. 10(b), there is a slight, \( 2\% \) increase of the tangential temperature components near the wall. Using specular reflection, this increase is not present.

6 Results of the New Boundary Method

In this section we discuss the results of our new boundary method applied to some well known flow problems. First, we look at a fluid at rest, then consider the Couette and Poiseuille flow problems and finally looking at the results of the flow past an oscillating flat plate and the lid-driven cavity flow.
Figure 9: Analysis of the probability distributions by looking at the Skewness (left) and the Kurtosis coefficient (right) versus the distance from the fictitious wall. The dashed line shows the coefficients of a Gaussian distribution. The coloured samples are the conservative (normal: green, tangential: blue), dissipative (normal: cyan, tangential: red) and stochastic (normal: yellow, tangential: magenta) force components. Especially the dissipative force is clearly non-Gaussian.

Figure 10: Sketch of the bounce-forward reflection mechanism (a). A particle jumping out of the simulation domain is reinsert according to the specular reflection, but the velocity and force components are inverted. Fig. (b) shows the temperature disturbance we observed using bounce-forward reflection.
Figure 11: DPD simulations of a fluid at rest and confined between two walls. The wall force is implemented using the means of the sampled forces (see section 5.1). From left to right is shown the density, temperature and velocity profiles. The observed fluctuations of 6% in the density are artifacts induced by the presence of the walls, while the cooling down effect of 10% near the wall arises due to the missing fluctuations of the wall force.

Figure 12: DPD simulations of a fluid at rest and confined between two walls. The wall force is implemented according to our final proposition (see section 5.2), by using the sampled probability distributions to generate the force. From left to right is shown the density, temperature and velocity profiles. The observed fluctuations of 6% and 2% in the density and temperature are artifacts induced by the presence of the walls.

6.1 Fluid at Rest

As a first test we look at a fluid between two walls located at $x = 0$ and $x = 10$. The simulation domain is a cube of size $10 \times 10 \times 10$. The density $\rho$ is set to 3 and the temperature $k_B T$ to 1. In Fig. 11 the results using the mean wall force components to model the wall as described in section 5.1 together with the bounce-forward reflection mechanism are shown, where as in Fig. 12 the results using the probability functions as described in section 5.2 are shown. Only applying the mean forces cools down the system about 10% near the wall. By adding the fluctuations in the force the discrepancy in the temperature profile is reduced to less than 2%, were as the density shows deviations of less than 6% ins both cases. To illustrate that the bounce-forward mechanism has no effect on the density fluctuations, we show in Fig. 13 the results of the same simulation scenario but with specular reflection. No difference in the density profile can be observed.

6.2 Couette Flow

As a first test for the no-slip condition, we study the steady velocity profile for a shear flow between flat walls. Again, the simulation is performed in a cube of size 10. In this case, the left wall at $x = 0$ is moving with a constant velocity of $U_z = 1.0$ where the right wall at $x = 10$ is at rest. A shear profile will develop having a slope $-U_z/h$ where $h$ is the distance between the walls. The results shown in Fig. 14 are obtained with a simulation at $\rho = 3$, $k_B T = 2.0$
Figure 13: DPD simulations of a fluid at rest and confined between two walls. The wall force is implemented by using the sampled probability distributions to generate the force and specular reflection is applied. From left to right is shown the density, temperature and velocity profiles. The observed fluctuations in the density profile are not smaller as in the case with bounce-forward reflection and the wall is not non-slip anymore.

Figure 14: Couette flow: Shown are profiles between the walls for density, temperature and velocity separately for each dimension x (blue), y (green) and z (red). In cyan, the analytic solution is given for reference.

and \( a = 25 \), and the bouncing mechanism is applied in a reference system, where the wall is stationary. The results are in excellent agreement with the analytic solution and are found to satisfy the no-slip condition.

### 6.3 Poiseuille Flow

The next case for validating our boundary condition is the Poiseuille flow. Usually the Poiseuille flow is understood to be driven by an externally imposed pressure gradient, but for isothermal flow [17], it is equivalent to apply a gravitational force \( mg \) on each particle. In this scenario, we again have two parallel walls at \( x = 0 \) and \( x = 10 \). Both are at rest. A gravity force \( g = 0.02 \) acts on each particle in the z-direction. Starting from a fluid at rest, there will arise a steady velocity profile with a parabolic shape (cf. for example [12]) according to

\[
V_z(x) = \frac{\rho gh^2}{8\mu} \left[ \frac{x}{h} - \left( \frac{x}{h} \right)^2 \right] \tag{19}
\]

where \( \mu \) is the viscosity and \( h \) the distance between the walls. In Fig. 15 we show results with the same parameters (\( \rho = 3, a = 25, \sigma = 3 \) and \( k_BT = 1 \)) as Pivkin [14] used. Our density profile shows much smaller fluctuations than reported by Pivkin et al. (6\% compared to 100\%). The deviations in the temperature profile are again less than 2\% and the velocity profile shows no slip at the boundary. All the quantities are averaged during 40000 time steps after an initial equilibration period.
Figure 15: Poiseuille flow: Shown are profiles between the walls of density, temperature and velocity (from left to right). The density of our boundary method (blue) has much smaller fluctuations near the wall as the one Pivkin et al. [14] (red) reported recently. The temperature and velocity profiles are again given separately for each dimension x (blue), y (green) and z (red). In cyan, the Navier–Stokes solution is given for reference.

Figure 16: Poiseuille flow according to Fan et al. [3]: On the left are shown profiles between the walls for the density of the present work (blue) and results of Fan et al. (red). On the right are the velocity profiles after $t = 500$ (blue), $t = 1000$ (green) and the steady profile reached after $t = 2000$ (black) together with the Navier–Stokes solution (black circles) and steady state solution of Fan et al. (red). They correspond almost perfectly to each other.

As a second validation, we repeated the Poiseuille flow problem as presented in Fan et al. [3]. Simulation parameters are set according to the original paper to $\rho = 4$, $k_B T = 1$, $a = 18.75$ and $\sigma = 3$ in a simulation box of size $30.5 \times 5 \times 10$. Our results are again in very good agreement with [3], in particular the measured velocity in the middle of the channel is 8.650 and the viscosity 1.076 in comparison to 8.639 and 1.077 respectively predicted by [3]. The density has again much smaller fluctuations near the wall, cf. Fig. 16.

6.4 Stokes Flow Near an Oscillating Flat Plate

An other well-known exemplary flow, for which an analytical solution exists [12], is Stokes flow near an oscillating flat plat. We use the parameters used in [14] with a cubic domain of size 10, solid walls at $y = 0$ and $y = 10$ and periodic boundary conditions in the other two directions. The lower wall is oscillating with a velocity $U_x = \sin(\Omega t)$, where $\Omega = \pi/20$. The density $\rho$ is set to 10, the temperature $k_B T = 1/3$, $\sigma = \sqrt{3}$ and the maximum repulsion $a = 3$. The kinematic
Figure 17: Stokes flow near an oscillating flat plate: We show the velocity profiles at 16 instances during the period. On the left: Time \( t = 2k\pi/8, \) \( k = 0, \ldots, 7 \). On the right: \( t = (2k+1)\pi/8, \) \( k = 0, \ldots, 7 \). DPD simulations are indicated with squares, analytic solutions with lines.

Viscosity can be estimated according to [6] to be

\[
\nu = \frac{45(k_B T)^2}{2\pi \rho \sigma^2 r_c^5} + \frac{\pi \sigma^2 \rho r_5^5}{1575 k_B T} = 0.2060. \tag{20}
\]

The computational domain was subdivided into 20 equally sized bins in the y-direction, and data were collected at 16 points during the periodic cycle by phase-averaging over the last 5 time steps over 50 periods.

In Fig. 17 the velocity profiles at the 16 different points in the period are shown together with the analytical solutions. The normalised distance is defined as \( Y = y\sqrt{\Omega/\nu} \). The results of the simulation are in excellent agreement with the analytic solution. The largest absolute error in the velocity is below 0.01.

6.5 Lid-driven Cavity Flow

As last test we consider flow in a lid-driven cavity. For that, we repeated the simulation proposed by Pivkin et al. [14]. The DPD parameters are the same as in the previous case, but now there are solid walls in two directions \( (x \text{ and } y) \). The wall located at \( y = 0 \) is moving with a constant velocity of \( U_x = 0.5475 \). The corresponding Reynolds number is 25. The computational domain is subdivided into \( 30 \times 30 \times 1 \) equally sized bins. In Fig. 18 we show the measured density distribution, the velocity vector field and two velocity profiles along the cuts at \( y = 5 \) and \( x = 5 \) respectively. The density fluctuations in the corners are artefacts of the wall force, which is not yet properly implemented for corners, see section 8. Nevertheless, the velocity shows a good agreement with the spectral element simulation performed by Pivkin et al as one can see especially in the lower graphs of Fig. 18.

7 Conclusion

In this work we have presented a new method to model solid walls in DPD simulation. It is based on an effective wall potential obtained by sampling in an additional simulation at the same state point but with periodic boundary conditions the force contributions acting on a fictitious wall. These boundary forces have to be applied afterward according Eq. (18). We validated the proposed method on a set of exemplary flow pattern. This new boundary algorithm is found to produce no-slip and to significantly reduce the density fluctuations previously observed in DPD simulations.
Figure 18: Cavity flow: On the top we show density distribution (left) and velocity vector field (right). Below are velocity profiles extracted along a vertical and horizontal cut. Triangles pointing upwards are velocity components in x-direction, while downwards pointing indicate the velocities in y-direction. The solid lines are reference values that Pivkin et al. [14] obtained by a spectral element simulation. The coordinates are normalised by the domain size, velocity by the wall velocity $U_x$. 
8 Future Work

Further work is possible at several edges of this work. A few ideas are listed in the following.

- First, one could improve the stability of the method at bigger time steps using an alternative integrator. Two approaches [13, 16] have recently been published and seem to be stable over a bigger time step interval. Both methods are not yet widely used and look at DPD in a different manner than the classical way. Anyhow, it would be worth to have a deeper look at these methods.

- Up to know, the simulation client can only handle identical particles. Most interesting DPD applications look at so-called soft matter physics, e.g. the mesoscopic phenomena of liquid crystals, colloids or biomembranes. All these applications need the possibility to model different particles together. An extention of our client is therefore an important point.

- In section 5.2, the correlation of the forces in time given a certain distance from the wall has been analysed and was found to be uncorrelated. But the forces between the particles are correlated. Further work should be done in studying, if the uncorrelated wallforce reduces the correlation between the particles near the wall and how one could correct that.

- The experiment with the lid-driven cavity flow has shown, that the corner regions are not yet implemented fully correctly. The imposed wallforce is too strong. Therefore one has to find a mechanism to decrease the force in such regions. This is obviously only a special case of the next point.

- The biggest open issue is clearly to implement non rectangular walls, especially curved walls. The proposed way to impose a wallforce is in principle confined to flat walls. Nevertheless, one could imagine that the sampling part is also done for a set of fictitious walls with a certain curvatures. Walls with very small curvatures could maybe also simply be linearised. Curved boundaries are in the area of DPD simulations nearly not investigated.
A Overview of the ppm DPD-client

A.1 Introduction

Our ppm DPD program is a client for the ppm library [18], a library for parallel particle methods. In this appendix we give a brief overview on the main aspects of this client. We first describe the necessary input and output files and we present the scripts and tools for handling these files and create human readable output respectively. Afterwards we will demonstrate the simplified simulation steps using some flow charts, before describing each module of the DPD client. And in the end we will mention the already known bugs of the client.

A.2 Input / Output

In this section we present the required input files for the DPD client on the one hand and on the other hand the output that the simulation will produce. For both we present the file formats. Having showed that, we describe the scripts to post process these files and how they can be used to generate input files for the simulation again, i.e. to generate the wall forces.

A.2.1 Required Input Files

To perform a DPD simulation, one has to provide at least a control file, specifying the simulation parameters, and an input file, describing the initial state of the system. For simulations with non periodic boundaries, a third file is required which contains a description of the wall forces.

Control File  The control file is an ASCII text file containing entries of the form “key” = “value” where key is case insensitive. In the sample control file, all the keys and their possible values are explained as comments. Please check there for all possible settings. The most important ones are:
inputfile
freqoutput
outputfile
domain
cutoff
nsteps
dt
freqdiag
diagnosticsfile
rho
bTemp
sigma
repulsion
rdf
profiling
prof_domain
prof_bins
prof_freq.out
prof_file
bForceFile
reflect
simMeth
boundMeas

Initial System State File  The input file to describe the starting state of the system looks like follows:

The first line contains the total number of particles (Ntot), the number of dimensions for position, number of dimensions for velocities and the number of attributes a particle has. The second line contains only the precision (4 = single, 8 = double) and the third line the simulation starting time. On the next Ntot lines follow the positions of the particles, then their velocities and finally the seeds, in total 3N_{tot} + 3 lines.
$$
\begin{array}{|c|c|c|c|}
\hline
\text{Ntot} & \text{nDim} & \text{nSpec} & \text{nAp} \\
\hline
\text{precision} & \text{time} & & \\
\hline
\text{pos}_{1,1} & \cdot & \cdot & \text{pos}_{nDim,1} \\
\vdots & \cdot & \cdot & \vdots \\
\text{pos}_{1,Ntot} & \cdot & \cdot & \text{pos}_{nDim,Ntot} \\
\text{v}_{1,1} & \cdot & \cdot & \text{v}_{nSpec,1} \\
\vdots & \cdot & \cdot & \vdots \\
\text{v}_{1,Ntot} & \cdot & \cdot & \text{v}_{nSpec,Ntot} \\
\text{seed}_{1,1} & \cdot & \cdot & \text{seed}_{nAp-1,1} \\
\vdots & \cdot & \cdot & \vdots \\
\text{seed}_{1,Ntot} & \cdot & \cdot & \text{seed}_{nAp-1,Ntot} \\
\hline
\end{array}
$$

**Wall Force File**  The input files for the force components are in binary format. There exist matlab scripts (wallForces and genDist) to generate the files. The file format can easily be seen in these scripts.

**A.2.2 Generated Output Files**

In this section, we provide a brief overview of the output files the simulation creates. Most of them are self-explanatory and quite simple in their structure.

**Diagnostics**  Two diagnostic files are created, one measuring the tracer diffusion coefficient (TDC), the other some energy quantities. The format of the TDC file is simply

$$
\begin{array}{|c|c|c|c|c|c|}
\hline
\text{time} & \text{coefficient} \\
\hline
\end{array}
$$

The format for the Energy file is:

$$
\begin{array}{|c|c|c|c|c|c|}
\hline
\text{time} & \text{Ekin} & \langle v^2 \rangle & \langle v_x^2 \rangle & \langle v_y^2 \rangle & \langle v_z^2 \rangle & \langle k_B T \rangle \\
\hline
\end{array}
$$

Both files are removed at the beginning of each simulation and in each time step, the measured quantities in this step are appended at the end of the files.

**System State**  The file specified by outputfile in the control file contains the instantaneous state of the system at the time step it was created. the format is simply:

$$
\begin{array}{|c|c|c|c|}
\hline
\text{Number} & \text{time step} & \text{Number of} & \text{number of} \\
\text{of total} & & \text{species} & \text{dimensions} \\
\text{particles} & & & \\
\hline
\text{simulation} & & & \\
\text{time} & & & \\
\hline
\text{pos}_{x,1} & \text{pos}_{y,1} & \text{pos}_{z,1} & \text{v}_{x,1} & \text{v}_{y,1} & \text{v}_{z,1} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\text{pos}_{x,Ntot} & \text{pos}_{y,Ntot} & \text{pos}_{z,Ntot} & \text{v}_{x,Ntot} & \text{v}_{y,Ntot} & \text{v}_{z,Ntot} \\
\hline
\end{array}
$$

**Profiling**  Profiling output is stored in different files according to the observed variable: The density is written to a file ending on "RHO.out", the velocity to "VEL.out", the squared velocity to "VEL2.out" and the number of samples to "SAMP.out". The file containing the density contains the same information as the sample file, only multiplied by a factor to get the number density. The other files contain non-averaged data. The format of the sample file is:

\[21\]
where \(c(x, y, z, t)\) is the number of observation in this specific bin, \(nx\) is the total number of bins in the \(x\), \(ny\) in \(y\) and \(nz\) in \(z\) direction and \(nt\) are the number of time bins.

The velocity profiling files look similar. The 3 velocity components are on the same line, and the other dimensions are appended:

\[
\begin{array}{ccc}
| v_x(1,1,1) & v_y(1,1,1) & v_z(1,1,1) \\
| v_x(nx,1,1) & v_y(nx,1,1) & v_z(nx,1,1) \\
| v_x(2,1,1) & v_y(2,1,1) & v_z(2,1,1) \\
| v_x(nx,ny,nz,nt) & v_y(nx,ny,nz,nt) & v_z(nx,ny,nz,nt) \\
\end{array}
\]

**RDF output**  This file contains for every distance the the radial distribution function value.

<table>
<thead>
<tr>
<th>distance</th>
<th>rdf value</th>
</tr>
</thead>
</table>

### A.2.3 Additional Tools

In this section we present some tools to generate the input and post process the output files.

**Input File Generator** The tool InputGen.java is a generator for the input file. It generates randomly distributed particles on the whole domain, gives them a Maxwellian distributed velocity and generates the seeds. The program needs as input the total number of particles, the corner coordinates of the domain, an output filename and the desired temperature of the system. The tool is written in Java to make sure, that the random number generator is not the same as the one used in the simulation. This is important for the use of the particles’ seed, cf. Sec. A.4.2.

**Computed Mean Wall Force Generator** The matlab function wallForces generates from a polynomial RDF description the computed wall force file. The function parameters are:

- filename: filename of the wall force file
- resolution: the number of sample points that should be generated in the file.
- rho: the density of the system \(\rho\)
- sigma: the noise amplitude \(\sigma\)
- gamma: the friction coefficient \(\gamma\)
- rep: the repulsion coefficient \(a_{ij}\)
- cutoff: the interaction distance \(r_c\)
- p: the polynomial coefficient for \(g(r)\)

and \(p\) can be obtained by the matlab built-in function polyfit(…).

**Measured Mean Wall Force Generator** The measured mean wall force is generated directly by the client when performing a simulation with periodic boundary conditions. The measured forces are also be written into a human readable file called dpd_boundary_FORCE.
**Probitility Distribution Wall Force Generator** Using the measured forces created by the DPD client if the control flag `boundMeas` is set to `.TRUE.`, the matlab script `genDist` generates a cumulated density function. The file name of the file containing the measurements has to be edited directly at the beginning of the script.

**Profile Visualisator**n The matlab scripts `density`, `velocity` and `temperature` generate form the profiling files the same 1D profiles as those presented in the result Sec. 6. These functions take as input:

- `filename` the base of the profile filename as set in the control file.
- `dim` the number of bins in each direction. Time bins are not yet implemented in these scripts.
- `along` in which direction the profile should be generated.

The temperature can be computed from the velocity and the squared velocity profiles by looking at temperature as the variance of the velocity. Thus the time average of the temperature is

\[
\langle k_B T \rangle = \frac{(v - \langle v \rangle)^2}{3} = \frac{1}{3} \sum_{i=1}^{3} (v_i - \langle v_i \rangle)^2 = \langle \text{Var}(v) \rangle = \langle v^2 \rangle - \langle v \rangle^2 .
\]

Two dimensional density or velocity plots can be created with the `contourf(…)` and `quiver(…)` functions. Examples for this can also be found in the script `cavity`.

### A.3 Flow Charts

In this section we present the structure of the ppm DPD client in an abstract way by indicating the sequence of steps, that are necessary for the simulation. Fig. 19 gives a global view of the simulation process: After an initialisation step, we advance in time by repeating the integration loop. In this loop, one can recognise the modified velocity-Verlet algorithm from Sec. 2 again.

Fig. 20 and 21 explain the steps “compute interparticle forces” and “add wall forces” in more detail: Before the interparticle forces can be computed, one has to update the seeds as proposed in Sec. 3, then map the particles and the ghosts to the right processor and build the new neighbour list. Finally, one can start computing the forces. Fig. 21 shows the steps that are necessary to add the wall forces. Depending on the settings made by the user in the control file, a different path is taken in the flow chart.

### A.4 DPD Modules

In the following we give an introduction to all the modules and their subroutines used in the DPD client.

#### A.4.1 `dpd_module_boundary`  
This module contains all the subroutines handling the boundary condition, i.e. the reflection and the wall force mechanism. A flag called `dpd_bound_MODE` stores, whether the program is in the wall force measurement or application state. The application mode is subdivided into 4 different modes: applying the measured mean wall force, applying the computed mean wall force, using the measured probability distribution without correlation or using the measured probability distribution with correlation. The work is accomplished by the following subroutines:

- `dpd_bound_init`  
  This subroutine only decides according settings in the control file, which boundary mode should be used and either initialises the resources to measure the forces on the fictitious wall, loads the mean wall forces or loads the inverse probability lookup table, cf. Sec. A.4.5.
Figure 19: Flowchart showing the global structure of the program.
Figure 20: Flowchart showing in more detail the part “compute interparticle forces” in the global view.
Figure 21: Flowchart showing in more detail the part “add wall force” in the global view.
• **dpd_read.bforce**
  This is the subroutine that loads the mean wall force, either the measured or the computed one, depending on the **dpd_bound.MODE**.

• **dpd_bc.intro**
  This subroutine handles the reintroduction of particles leaving the computational domain. The user can specify which mechanism should be used. Implemented are the periodic-, the bounce-forward-, the specular- and the Lees-Edwards methods.

• **dpd_leesEdwards_ghost**
  This subroutine only adjusts the velocity of the ghost particles if using Lees-Edwards boundary condition.

• **dpd_bound.apply**
  This subroutine processes all the particles near a non-periodic wall by using a the cell list, see Sec. A.4.2, and adds the force components computed by the following subroutine (**dpd_wallForce**). The routine assumes that the cell list is up to date. This means, that the user has to call the routine **dpd_build_list** itself or be sure that the position of any particles has not changed since the last call of the routine. **dpd_wallForce** needs as input parameter the distance to the wall, the normal and 2 tangential velocities in the coordinate system of the wall. Therefore, this subroutine has also to map these quantities correctly.

• **dpd_wallForce**
  Depending on the **dpd_bound.MODE** flag, either the mean- or the probability force components (**dpd_cumDist**) are looked up and linearly interpolated between the nearest two sampling points. If the user has selected the correlated probability mode, then the conservative and the dissipative contributions are weighted according the correlation coefficient \( \alpha \) with the “brain”, a simple state buffer containing the previous generated forces: 
  
  \[
  \text{Brain}_i = \alpha \text{Brain}_{i-1} + (1 - \alpha) f_i,
  \]
  
  where \( f_i \) is the looked up uncorrelated force and the correlated contribution is then \( \text{Brain}_i \). All the contributions are finally added according to Eq. (18) and returned.

A.4.2 **dpd_module.comp**

The module **dpd_module.comp** contains the main logic presented in the global flowchart. It contains all the necessary routines for the time integration, the mapping of the particles to the different processors and the interparticle force calculation. The module contains the following subroutines:

• **dpd_step**
  This subroutine implements the outermost loop over the time steps. In the main loop, the routine calls the **dpd_vV_step** to do the time integration according to the velocity–Verlet algorithm and the **dpd_step.output** to update the diagnostics and generate output informations. Before entering the loop, several quantities are initialised according to the integrator’s requirements, i.e. the force of the particles at time \( t = 0 \) has to be computed.
  To abort the program, it looks at the end of each step in the main loop, if the “abort” file exist, whose filename can be specified in the control file.

• **dpd_step.output**
  In this subroutine, the diagnostics like the radial distribution function (RDF), see Sec. A.4.7, the profiling of the physical quantities like velocity, density and temperature, see Sec. A.4.4, or the tracer diffusion constant are updated. Beside this, also instantaneous output of particles position and velocity are created. For every quantity one can decide in the control file, whether to turn it on or off and how often the output should be generated respectively.
**dpd_vv_step**
This subroutine implements the velocity–Verlet time integration step as described in the flow chart. The following presented routines have to be called in exactly this order after having updated of the positions and the velocity guess, but before correcting the velocities. In the last step of the integration scheme the forces of the last timestep have to be known, cf. Eq. (8). Therefore they have to be passed by `dpd_step` to this routine and updated at the end of each step.
The velocity–Verlet scheme is not (yet) implemented as an ODE-solver in the ppm library. Because of it's properties, i.e. 2nd order accuracy with only one force computation per time step, one should think of moving the scheme into the ppm library.

**dpd_set_particle_seed**
In this routine, the seed of the particles is changed. This is done by seeding the random number generator with the old particle seed, throwing a dummy random number and taking the new seed of the random number generator as the new particle seed. In important point is, that the initial particle seeds are not generated with the same random number generator! Otherwise one is only shifting the seeds from one particle to the next.

**dpd_map_ghost**
This subroutine maps the position, velocity and seed of the particles within a cutoff distance of any boundary to the ghost particles. The mapping mechanism itself is implemented in the ppm library.

**dpd_build_list**
This routine creates a neighbour list with cells of size $r_c \times r_c \times r_c$. It is used by the following routine in order to locate neighbouring particles efficiently. The lists itself are generated by the ppm methods `ppm_neighlist_clist` and `ppm_neighlist_MxNeighIdx`.

**dpd_comp_pp_cell** and **dpd_part_interaction**
The routine makes use of the cell list. All particles that are potentially within the interaction distance $r_c$ are located in the same or one of the 26 neighbouring cells. The code in `dpd_comp_pp_cell` processes all the cells and their particles and finds pair of particles within the interaction distance. For these pairs, the code in `dpd_part_interaction` is executed. There, the interparticle forces are computed according to Eq. (3–5)

### A.4.3 dpd_module_io
This module contains most of the I/O routines. The most important ones are listened below.

**dpd_read_ctrl**
This subroutine reads the control file. All the user defineable values are read from this file. The routine ignores lines starting with a "#". Other lines are interpreted in the form "key" = "value". The key is case insensitive.

**dpd_read_input**
This subroutine reads the initial condition of the system, i.e. position, velocity and seed of all the particles. The routine distributes the particles onto the different processors. A detailed description of the file format is given in Sec. A.2.1.

**dpd_write_output**
This subroutine writes the current state of the system into a file. The routine collects all the quantities on one processor and writes them afterward to disc. For a detailed description of the file format, see Sec. A.2.2
• dpd_write_diag
  This subroutine updates the diagnostics. Currently it creates two output files, one measuring the tracer diffusion coefficient and the other the energy quantities like temperature, mean velocities and kinetic energy. Again, the quantities are collected form all the processors and then written to disc.

A.4.4 dpd_module_profiler
The profiler is a module to measure physical quantities over several time steps and average them. It is in principle a buffer that is collecting positions and velocities of particles according to their spatial distribution. The profiler itself does not build any average. This is done in a post processing step, cf. A.2.2

• dpd_profiler_init
  This subroutine simply initialises the profiler according to the users settings.

• dpd_profiler_getTimeBin
  In the control file, the user can specify a profiling scenario. In this subroutine, one needs to decide in which time bin the quantities should be collected. Using this mechanism, one can collect data at different times in the same run.

• dpd_profiler_step
  In this routine the position, velocity and the squared velocity of each particle in the profiling domain is added to the bin in which the particle is located. From these quantities, one can compute the density, temperature and velocity, cf. Sec. A.2.2.

• dpd_profiler_write and dpd_profiler_finalize
  These two subroutines simply write the measured quantities into a file and deallocate the resources respectively.

A.4.5 dpd_module_ranDist
This module handles the probability wall force. As mentioned in Sec. A.2.1, the input for this mechanism is a file containing the cumulated distributions for each force component generated by the script genDist. The following subroutines process this input and provide an interface to generate force components distributed according the measured forces.

• dpd_loadProbs
  This subroutine initialises the module, loads the input file and generates an inverse probability lookup table for each force component at a set of different distances from the wall. The number of different lookup tables is specified in the input file.

• dpd_cumDist
  This is a function which returns the cumulate distribution value. It is passed by the routine dpd_loadProbs to a bisection algorithm to compute the inverse probability lookup table.

• dpd_getProbDist
  This function is the main interface to the module. To get an instance of a certain probability force component, the user simply calls the function with 2 arguments, the ID of the force component, and the distance of the particle form the wall.

• dpd_ranDist_finalize
  This routine simply frees all resources it has taken.
A.4.6 dpd_module_random

This module provides access to the random number generator and implements some transformations of random numbers. It contains the following functions.

- **uniform**
  This is a function which returns a uniform random number. Currently a random number generator by Knuth [8] is used. But other random number generators are also possible, as long as they provide a way to retrieve and set the state (seed).

- **set_seed** and **get_seed**
  These are the subroutines to set and get the seed of the random number generator.

- **normal_bm**
  This function returns a Gaussian distributed random number. The implementation of the transformation is performed according to the Box-Muller method.

- **pseudo_gauss**
  As the stochastic forces generated with uniform random numbers with mean zero and variance 1 are indistinguishable from forces generated with Gaussian random numbers [6], we use uniform random numbers to generate the forces. This function transforms the uniform random numbers u to have mean zero and variance 1, i.e. \( u' = 2u - 1 \) if \( u \) is uniformly distributed in the range \([0, 1)\).

A.4.7 dpd_module_rdf

The task of this module is to measure the radial distribution function. It makes again use of the neighbour list presented in Sec. A.4.2. The user can specify the range in which the RDF is measured by the **rdf_cutoff**-key in the control file. If this distance is greater than the cutoff distance of the forces, one has to make sure that there exist ghost particles in the whole band of width **rdf_cutoff** around the computational domain. Otherwise, the result will be wrong. The following subroutines belong to this module:

- **dpd_compRDF**
  By processing all the cells in the cell list, the distance between each pair of particles is computed and discretised. In contrast to the force calculation, the subroutine builds the neighbour list by it’s own each time it is called.

- **dpd_rdf_output**
  At the end of the simulation, the measured number of particle pairs is compared to the number of pairs one would expect if the particles are uniformly distributed in the domain. The fraction of the measured to the expected number of pairs is written to disc for every distance bin.

A.5 Known bugs

In the following we list the known bugs in no particular order:

- The DPD client was never tested in parallel. This never happens because the MPI installation is buggy (not a single ppm client could be run in parallel) and no one could fix it. The symptom is, that all processors think the are the only running processor and hence crashes.

- The force computation can not be done using symmetry due to a wrong or missing implementation in the ppm library.
• The written restart files can not properly be restarted. Further more, the profiling state as well as the RDF measurement state are not written into the restart file. Without these information, a restart does not make that much sense. As a quick fix, there exist the routine \texttt{dpd\_write\_newStart}. It generates a new input file of the current particle configuration. This can be useful if one wants to generate a well calibrated input file.

• The simulation is only implemented for 3D problems.

• The radial distribution function can only be computed efficiently for \( \texttt{rdf\_cutoff} \leq r_c \).
References


