Aalto University

OPEN ACCESS

This is an electronic reprint of the original article.
This reprint may differ from the original in pagination and typographic detail.

Denniston, Colin; Afrasiabian, Navid; Cole-Andre, M. G.; Mackay, F. E.; Ollila, S. T. T.; Whitehead, T.

LAMMPS lb/fluid fix version 2: Improved hydrodynamic forces implemented into LAMMPS through a lattice-Boltzmann fluid

Published in:
Computer Physics Communications

DOI:
10.1016/j.cpc.2022.108318

Published: 01/06/2022

Document Version
Publisher's PDF, also known as Version of record

Published under the following license:
CC BY-NC-ND

Please cite the original version:
Denniston, C., Afrasiabian, N., Cole-Andre, M. G., Mackay, F. E., Ollila, S. T. T., & Whitehead, T. (2022). LAMMPS lb/fluid fix version 2: Improved hydrodynamic forces implemented into LAMMPS through a lattice-Boltzmann fluid. *Computer Physics Communications*, *275*, Article 108318. https://doi.org/10.1016/j.cpc.2022.108318

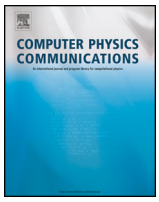
This material is protected by copyright and other intellectual property rights, and duplication or sale of all or part of any of the repository collections is not permitted, except that material may be duplicated by you for your research use or educational purposes in electronic or print form. You must obtain permission for any other use. Electronic or print copies may not be offered, whether for sale or otherwise to anyone who is not an authorised user.

Computer Physics Communications 275 (2022) 108318

Contents lists available at ScienceDirect

Computer Physics Communications

www.elsevier.com/locate/cpc



LAMMPS lb/fluid fix version 2: Improved hydrodynamic forces implemented into LAMMPS through a lattice-Boltzmann fluid ☆,☆☆

Colin Denniston [a,b,*], Navid Afrasiabian [a,b], M.G. Cole-André [a], F.E. Mackay [b], S.T.T. Ollila [b,c], T. Whitehead [b,d]

[a] Department of Physics and Astronomy, The University of Western Ontario, London, Ontario, N6A 5B8, Canada
[b] Department of Applied Mathematics, The University of Western Ontario, London, Ontario, N6A 5B7, Canada
[c] Department of Applied Physics, Aalto University School of Science and Technology, P.O. Box 11000, FIN-00076 Aalto, Espoo, Finland
[d] SHARCNET, Western Science Centre, The University of Western Ontario, London, Ontario, N6A 5B7, Canada

ARTICLE INFO

Article history:
Received 1 December 2021
Received in revised form 11 February 2022
Accepted 14 February 2022
Available online 22 February 2022

Keywords:
Lattice-Boltzmann algorithm
Molecular dynamics
Hydrodynamics
Micro-fluidics

ABSTRACT

The first version of this code (Mackay et al., 2013) [10] implemented long-range hydrodynamic interactions into the open-source molecular dynamics package LAMMPS. This was done through the creation of a fix, *lb/fluid* which was subsequently included as a user-package in the main LAMMPS distribution. Here we substantially update this package by making improvements to its accuracy, adding significant new features, and by simplifying the use of the package. A new two-pass interpolation and spreading scheme is introduced which results in the improved accuracy and numerical stability. New features include new output options, several added computes, and mesh geometry option suitable for micro- and nano-fluidic device simulations. The original package could require fairly careful calibration to obtain accurate thermostating and accurate reproduction of properties related to the hydrodynamic size of objects such as colloids. This process has now been largely automated so that the default settings should suffice for most applications.

Program summary

Program title: fix lb/fluid
CPC Library link to program files: https://doi.org/10.17632/2289cnrdtz.1
Licensing provisions: GPLv3
Programming language: C++
Journal reference of previous version: Comput. Phys. Commun. 184 (2013) 2021–2031.
Does the new version supersede the previous version?: Yes
Reasons for the new version: The new version improves accuracy, adds new features, and simplifies the use of the package.
Summary of revisions: A new two-pass interpolation and spreading scheme is introduced to relate properties on the fluid mesh to off-lattice particle properties. New features include output options, several added computes, and mesh geometry suitable for micro- and nano-fluidic device simulations. Calibration processes have been largely automated so that the default settings should suffice for most applications.
Nature of problem: The inclusion of long-range hydrodynamic effects into molecular dynamics simulations requires the presence of an explicit solvent. Prior to the implementation of this fix, the only option for incorporating such a solvent into a LAMMPS [1] simulation is the explicit inclusion of each of the individual solvent molecules. This is obviously quite computationally intensive, and for large system sizes can quickly become impractical.
Solution method: As an alternative, we have implemented a coarse-grained model for the fluid, simplifying the problem, while retaining the solvent degrees of freedom. We use a thermal lattice-Boltzmann model for the fluid, which is coupled to the molecular dynamics particles at each fluid time step.

☆ The review of this paper was arranged by Prof. Blum Volker.

☆☆ This paper and its associated computer program are available via the Computer Physics Communications homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author at: Department of Physics and Astronomy, The University of Western Ontario, London, Ontario, N6A 5B8, Canada.
E-mail address: cdennist@uwo.ca (C. Denniston).

https://doi.org/10.1016/j.cpc.2022.108318
0010-4655/© 2022 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

References

[1] S. Plimpton, J. Comput. Phys. 117 (1995) 1–19.

© 2022 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

1. Introduction

Particles and aggregates in fluids such as colloids and polymers can exhibit interesting behavior over a wide range of time scales. While local rearrangements can be quite fast, long range hydrodynamic interactions can lead to relatively slow dynamics. This can be true even for single particles in confined environments. For example, inertial migration of a particle in a channel [1] can be very slow process. Hydrodynamic effects are important for a wide range of systems including confined polymers in solutions [2,3], protein folding [4], colloidal aggregation [5] and sedimentation [6,7].

Classical molecular dynamics (MD) simulations use Newton's equations of motion to solve for the trajectories of particles interacting via intermolecular forces. LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) is an open-source MD package [8] that is parallelized using a spatial domain-decomposition [9]. It is especially attractive as it is fairly straightforward for a user to modify and extend due to its extensive documentation.

We implemented a lattice-Boltzmann (LB) fluid into LAMMPS in [10]. LB schemes are based on a discretized version of the Boltzmann equation and parallelize well in the domain-decomposition scheme used by LAMMPS. In order to couple the fluid to the MD particles, a force proportional to the local velocity difference between the fluid and particle was used [11–13]. This package was subsequently incorporated as a user-package in the standard LAMMPS distribution.

In this paper we describe a significant update to the LAMMPS *lb/fluid* package. Substantial improvements are made to the accuracy and stability by making use of a new algorithm described here for spreading the particle contributions onto the fluid mesh. This also substantially simplifies the use of the method as previous algorithmic values that had to be supplied by the user are now automatically calculated. We also add significant new features including boundaries suitable for simulations of micro and nanofluidic systems.

The following section describes the method and algorithms used. Section 3 describes how the code is installed and executed as part of a LAMMPS run, while Section 4 provides several test examples of the new package.

2. Theoretical background

2.1. The lattice-Boltzmann algorithm

The previous version of *lb/fluid* had multiple implementations of the lattice-Boltzmann (LB) algorithm: There was the "standard" integrator and an exponential integrator coupled with separate 15 and 19 velocity models [10]. The inclusion of the exponential integrator was primarily to improve stability that is tested by the coupling force to the particles. However, the current version relies on conservative forces without free parameters that results in a more stable method. As a result, we include only the classic lattice Boltzmann method in this version, again with separate 15 and 19 velocity implementations.

We briefly outline the standard LB algorithm here so that conventions and notation used are clear. Our description follows fairly closely to that given in [14]. The starting point is the BGK approximation of the Boltzmann equation which describes the time evolution of the fluid distribution function f [15],

$$\left(\frac{\partial}{\partial t}+\frac{\partial x_\gamma}{\partial t}\frac{\partial}{\partial x_\gamma}+\frac{\partial p_\gamma}{\partial t}\frac{\partial}{\partial p_\gamma}\right)f=-\frac{1}{\tau}\left(f-f^{eq}\right), \tag{1}$$

where the summation convention on repeated Greek indices is assumed, f^{eq} is the equilibrium distribution, x_γ is a position component, and p_γ is a component of the momentum density. Discretization is done first by expanding f in terms of tensor Hermite polynomials to second order in momentum and evaluating the expansion coefficients using a Gaussian quadrature [16]. This process defines N discrete velocities $\mathbf{e}_k$, and a set of weights w_k giving rise to a $D3QN$ discrete Boltzmann model. This produces the set of N lattice Boltzmann equations

$$\left(\partial_t+e_{k\gamma}\partial_\gamma\right)f_k=-\frac{1}{\tau}\left(f_k-f_k^{eq}\right)-F_{f\gamma}\partial_{p_\gamma}f_k, \tag{2}$$

where f_k is the distribution function corresponding to one of the discrete velocities $\mathbf{e}_k$ (contributing $f_k\mathbf{e}_k$ to the fluid momentum density), and $F_{f\gamma}=\partial p_\gamma/\partial t$ is the force density on the fluid (from forces external to the fluid). The lattice equilibria f_k^{eq} from the second order Hermite expansion are

$$\begin{aligned}f_k^{eq}=w_k\Bigg(&\rho+\frac{3}{c^2}p_\gamma e_{k\gamma}\\&+\frac{9}{2c^4}H_{\lambda\nu}\left(\rho\left(a_0-\frac{c^2}{3}\right)\delta_{\lambda\nu}+\frac{p_\gamma p_\nu}{\rho}\right)\Bigg),\end{aligned} \tag{3}$$

where $H_{\lambda\nu}=e_{k\lambda}e_{k\nu}-\frac{c^2}{3}\delta_{\lambda\nu}$, $c=\Delta x/\Delta t$ is the lattice velocity, the isotropic pressure is ρa_0 and, unless specified, a_0 is set to $\frac{c^2}{3}$. The mass density ρ and momentum density $\mathbf{p}=\rho\mathbf{u}$ are moments of the distribution function:

$$\rho=\sum_{k=0}^{N}f_k,\qquad \rho\mathbf{u}=\sum_{k=0}^{N}f_k\mathbf{e}_k. \tag{4}$$

The forcing term on the right hand side of Eq. (2) is approximated to lowest order as $F_{f\gamma}\partial_{p_\gamma}f_k\approx F_{f\gamma}\partial_{p_\gamma}f_k^{eq}$ which is then rolled into a term similar to the collision term making use of the fact that its lowest moments are [17,18]

$$\sum_k F_{f\gamma}\partial_{p_\gamma}f_k^{eq}=0, \tag{5}$$

$$\sum_k F_{f\gamma}\partial_{p_\gamma}f_k^{eq}e_{k\alpha}=F_{f\alpha}, \tag{6}$$

$$\sum_k F_{f\gamma}\partial_{p_\gamma}f_k^{eq}e_{k\alpha}e_{k\beta}=u_\alpha F_{f\beta}+F_{f\alpha}u_\beta. \tag{7}$$

Eq. (2) is a system of hyperbolic partial differential equations which can be converted to a system of first order ordinary differential equations (ODEs) using the method of characteristics (here the characteristics are straight lines along the $\mathbf{e}_i$ directions). The resulting ODEs can be solved to second-order using the implicit trapezoidal method to obtain

$$f_k(\mathbf{x}+\mathbf{e}_k\Delta t, t+\Delta t) - \frac{\Delta t}{2}R_k(\mathbf{x}+\mathbf{e}_k\Delta t, t+\Delta t) = f_k(\mathbf{x},t) - \frac{\Delta t}{2}R_k(\mathbf{x},t), \tag{8}$$

where $R_k(\mathbf{x},t)$ is the right hand side of Eq. (2). By defining auxiliary distribution functions,

$$\bar{f}_k(\mathbf{x},t) = f_k(\mathbf{x},t) - \frac{\Delta t}{2}R_k(\mathbf{x},t), \tag{9}$$

these equations can be made explicit:

$$\bar{f}_k(\mathbf{x}+\mathbf{e}_k\Delta t, t+\Delta t) = \bar{f}_k(\mathbf{x},t) + \Delta t\, R_k(\mathbf{x},t). \tag{10}$$

To evaluate the collision term in R_k using the $\bar{f}_k$ (rather than the f_k), Eq. (9) must be inverted. Doing this and then substituting into Eq. (10) yields [14,19]

$$\bar{f}_k(\mathbf{x}+\mathbf{e}_k\Delta t, t+\Delta t) = \bar{f}_k(\mathbf{x},t) + \frac{\Delta t}{\tau+\Delta t/2}\left[-\left(\bar{f}_k - f_k^{eq}\right) - \tau F_{f\gamma}\partial_{p_\gamma} f_k^{eq}\right]. \tag{11}$$

In order to construct f_k^{eq} we need ρ and $\mathbf{p}=\rho\mathbf{u}$ which can be found from the $\bar{f}_k$ using [14]

$$\rho = \sum_{k=0}^{N}\bar{f}_k \tag{12}$$

$$p_\alpha = \rho u_\alpha = \sum_{k=0}^{N}\bar{f}_k e_{k\alpha} + \frac{\Delta t}{2}F_{f\alpha}. \tag{13}$$

Another interpretation of the auxiliary distribution $\bar{f}_k$ is being the true distribution defined at the half time steps $\Delta t/2$, $3\Delta t/2$, $5\Delta t/2,\ldots$ while the f_k are the true distribution on the full time steps $\Delta t, 2\Delta t, 3\Delta t,\ldots$ [20]. Using this interpretation, one can define a "half-step" fluid velocity as

$$\rho\bar{u}_\alpha = \sum_{k=0}^{N}\bar{f}_k e_{k\alpha}, \tag{14}$$

which will be useful in constructing the particle-fluid coupling force in the next subsection.

How the steps of the LB algorithm fit into LAMMPS is illustrated in Algorithm 1, which is a modified version of the pseudo-code given in the LAMMPS documentation [21]. For comparison, we have also shown the steps for the standard LAMMPS NVE fix, which is typically used in conjunction with the *lb/fluid* fix to move the particles.

Algorithm 1 LAMMPS pseudo-code illustrating calls for lb/fluid.

```
for n = 1 to N_t do                                   ▷ Loop over N_t timesteps
  ev_set()

  fix→initial_integrate()
    nve:
      v_i(t+Δt/2) = v_i(t) + Δt/(2m_i) F_i(t)
      x_i(t+Δt) = x_i(t) + v_i(t+Δt/2) Δt
    lb/fluid:
      calculate f_k^eq(ρ(t), u(t)) on local subgrid
      calculate/comm f̄_k(t+Δt)
      calculate/comm ρ = Σ_k f̄_k, ρū = Σ_k f̄_k e_k
  fix→post_integrate()

  LAMMPS neighbor list update and particle communication

  force_clear()
  fix→pre_force()

  force→compute(),
    where force ∈ {pair, bond, angle,dihedral, improper, kspace}

  comm→reverse_comm()

  fix→post_force()
    lb/fluid: if (fixviscouslb)
                calculate and comm interpolation weights
                calculate and comm Force on fluid F_f
                comm particle F_H(t+Δt) mass field
              end if
    lb/viscous: F_i(t+Δt) = F_{H,i} + m_i/(m_i + m_{n,i}) F_{x,i}
  fix→final_integrate()
    nve: v_i(t+Δt) = v_i(t+Δt/2) + Δt/(2m_i) F_i(t+Δt)
    lb/fluid: u(t+Δt) = ū + Δt/(2ρ) F_f
  fix→end_of_step()
    lb/fluid: output fluid                            ▷ If output to write

  output→write()                                      ▷ If output to write
end for
```

2.2. Coupling MD particles to the fluid

Here we will derive the force coupling between the Molecular Dynamics (MD) particles and the fluid. This is the main change from the earlier version of the package. While we will see the form is similar to that in Ref. [22], the new derivation eliminates all free parameters from that formalism. The goal here is to achieve an immersed boundary style method [23] where the velocity of the particle (or node on the surface of the extended particle) exactly matches the velocity of the fluid (interpolated to the location of the particle/surface node) at the end of each time step. Similar to other constraints in LAMMPS such as the SHAKE algorithm [24,25], this is done by finding the set of constraining forces (equal and opposite for the particles/fluid) that ensure that the particles velocity match the (interpolated) velocity of the fluid at the particles location at the end of the time step. As we will see, the algorithm below does this exactly in two limits, first when the particles are far apart and second when they are very densely packed. Between these limits there is a finite, but bounded, error (difference between the velocities). Just as with other constraints in LAMMPS, the total number of degrees of freedom (DOF) in the system are reduced by these constraints. As such, this must be accounted for when doing things like computing the temperature. This accounting is also provided by fix outputs described in a later section.

An interpolation *stencil* is used to interpolate values from the fluid mesh to the particle location (which typically does not coincide with a mesh point). The weight from mesh site j at particle (node) i location is denoted as ξ_{ij}. By construction we require that $\sum_j \xi_{ij} = 1$. This can then be used to interpolate the value of the fluid density to particle i's location as

$$\rho_i = \sum_j \rho(x_j)\xi_{ij}, \tag{15}$$

where $\rho(x_j)$ is the density at the mesh site x_j. However, for the fluid velocity we use a mass-weighted average for the interpolation so that

$$\mathbf{u}_i = \frac{\sum_j \rho(x_j)\mathbf{u}(x_j)\xi_{ij}}{\sum_j \rho(x_j)\xi_{ij}}. \tag{16}$$

In other words, we actually interpolate the momentum density to the location of the particle and then divide by the mass density interpolated to the particle site. This helps ensure an exact conservation of momentum in the algorithm. *Spreading* is the converse process to interpolation, and is the term used to describe how the particle properties are distributed onto the fluid mesh. The main quantity that needs to be *spread* is the equal-and-opposite force to the hydrodynamic force on the i'th particle F_H^i. In the previous version of the package the spreading weight of particle i onto

fluid mesh site j was always equal to the interpolation weight ξ_{ij}. In the current version this will only be true for a limiting case. We denote the spreading weight of particle i at lattice site j as η_{ij}. We will not require this sum to one, but will need the normalization which is denoted

$$\eta_{iA} = \sum_j \eta_{ij}. \tag{17}$$

To derive the required constraining forces, we need to also consider the velocity-Verlet algorithm [26] typically used in MD (variants work fairly similarly). In LAMMPS, the particle velocity at the end of the step is computed from the half-step velocity (cf. Algorithm 1) in the `final_integrate` step:

$$\mathbf{v}_i(t+\Delta t) = \mathbf{v}_i(t+\Delta t/2) + \frac{\Delta t}{2m_i}\left[-\mathbf{F}_H^i + \frac{m_i}{m_i+m_n^i}\mathbf{F}_x^i\right], \tag{18}$$

where m_i is the particle mass, m_n^i is the fluid mass constrained to move with the particle (which we will determine below), $\mathbf{F}_H^i$ is the constraining force on the particle (making it move with the fluid), and $\mathbf{F}_x^i$ is the other forces (inter-particle or other MD forces) on the particle. Note that the particle and accompanying fluid move *together* so form a pseudo-particle with total mass $m_i + m_n^i$. The corresponding step for the fluid is from Eq. (13) which, in terms of the specific forces in Eq. (18) and converted from momentum densities to total momentum for fluid site j, is

$$\rho_j \mathbf{u}_j \Delta x^3 = \rho_j \bar{\mathbf{u}}_j \Delta x^3 + \frac{\Delta t}{2}\sum_i \left[\mathbf{F}_H^i + \frac{m_n^i}{m_i+m_n^i}\mathbf{F}_x^i\right]\frac{\eta_{ij}}{\eta_{iA}}. \tag{19}$$

As an aside, we note that these equations conserve the total momentum $\mathbf{P}$ in the system:

$$\begin{aligned}
\Delta P &= \Delta P_{particles} + \Delta P_{fluid} && (20)\\
&= \sum_i \left[m_i \mathbf{v}_i(t+\Delta t) - m_i \mathbf{v}_i(t+\frac{\Delta t}{2})\right] && (21)\\
&\quad + \sum_j \left[\rho_j \mathbf{u}(\mathbf{x}_j) - \rho_j \bar{\mathbf{u}}(\mathbf{x}_j)\right]\Delta x^3 && (22)\\
&= \frac{\Delta t}{2}\sum_i \left[-\mathbf{F}_H^i + \frac{m_i}{m_i+m_n^i}\mathbf{F}_x^i\right] && (23)\\
&\quad + \frac{\Delta t}{2}\sum_i \left[\mathbf{F}_H^i + \frac{m_n^i}{m_i+m_n^i}\mathbf{F}_x^i\right]\sum_j \frac{\eta_{ij}}{\eta_{iA}} && (24)\\
&= \frac{\Delta t}{2}\sum_i \mathbf{F}_x^i. && (25)
\end{aligned}$$

where in Eq. (24) we have made use of Eq. (18) and (19) and switched the order of summation for the second term. To go from Eq. (24) to Eq. (25) we have made use of Eq. (17) to find the sum over j to be one. Note that if Newton's third law is obeyed by the MD algorithm (which is the case) we should have either: a) $\sum_i \mathbf{F}_x^i = 0$ if the $\mathbf{F}_x^i$ are inter-particle forces only or, b) this sum will result in the total external force on the system if external forces are present. As a result, total momentum is conserved, or changes in accordance to Newton's laws in the presence of external forces.

We now turn to determining $\mathbf{F}_H^i$ and m_n^i using the constraint that we want the end-of-step velocities $\mathbf{v}_i(t+\Delta t)$ and $\mathbf{u}_j$ interpolated to the i'th particle position to be identical. Using the interpolated velocity definition Eq. (16) with the final fluid velocity we get

$$\begin{aligned}
&\mathbf{u}_i(t+\Delta t)\\
&= \frac{\sum_j \xi_{ij}\left(\rho_j \bar{\mathbf{u}}(\mathbf{x}_j) + \frac{\Delta t}{2\Delta x^3}\sum_s \left\{\mathbf{F}_H^s + \frac{m_n^s}{m_s+m_n^s}\mathbf{F}_x^s\right\}\frac{\eta_{sj}}{\eta_{sA}}\right)}{\sum_j \rho_j \xi_{ij}}, && (26)\\
&= \bar{\mathbf{u}}_i + \frac{\Delta t}{2\Delta x^3}\sum_s \left\{\mathbf{F}_H^s + \frac{m_n^s}{m_s+m_n^s}\mathbf{F}_x^s\right\}\frac{\sum_j \xi_{ij}\eta_{sj}}{\eta_{sA}\sum_j \rho_j \xi_{ij}}, && (27)
\end{aligned}$$

where the first term is the interpolated half-step fluid velocity at particle i's location, and we have switched the order of summation in the second term. Now taking the difference between the interpolated final fluid velocity and the particle velocity (Eq. (18)) we get

$$\begin{aligned}
&\mathbf{u}_i(t+\Delta t) - \mathbf{v}_i(t+\Delta t) = (\bar{\mathbf{u}}_i - \mathbf{v}_i(t+\Delta t/2)) && (28)\\
&\quad + \frac{\Delta t}{2}\mathbf{F}_H^i\left(\frac{\sum_j \xi_{ij}\eta_{ij}}{\eta_{iA}\sum_j \rho_j \xi_{ij}\Delta x^3} + \frac{1}{m_i}\right) && (29)\\
&\quad + \frac{\Delta t}{2}\frac{\mathbf{F}_x^i m_n^i}{m_i+m_n^i}\left(\frac{\sum_j \xi_{ij}\eta_{ij}}{\eta_{iA}\sum_j \rho_j \xi_{ij}\Delta x^3} - \frac{1}{m_n^i}\right) && (30)\\
&\quad + \frac{\Delta t}{2\Delta x^3}\sum_{s\neq i}\left\{\mathbf{F}_H^s + \frac{m_n^s}{m_s+m_n^s}\mathbf{F}_x^s\right\}\frac{\sum_j \xi_{ij}\eta_{sj}}{\eta_{sA}\sum_j \rho_j \xi_{ij}}, && (31)
\end{aligned}$$

where we have separated out the term in the sum over particles corresponding to particle i. At this point we have not specified m_n^i, $\mathbf{F}_H^i$, or η_{ij} and we will use the goal of minimizing this expression (difference between the final velocities) to determine these quantities.

We note two limiting cases:

1. *Isolated Particles case* where there is only one particle or all particles are separated. In this case for any mesh site j only one particle contributes and $\xi_{ij}\eta_{sj} = 0$ for $i \neq s$. As a result the last line is zero and the choice

$$m_n^i \stackrel{?}{=} \frac{\eta_{iA}\sum_j \rho_j \xi_{ij}\Delta x^3}{\sum_j \xi_{ij}\eta_{ij}} \tag{32}$$

will zero the third line (the $\stackrel{?}{=}$ is used here as we will modify this slightly below). Finally

$$\mathbf{F}_H^i = \frac{2}{\Delta t}\frac{m_i m_n^i}{m_i+m_n^i}(\bar{\mathbf{u}}_i - \mathbf{v}_i(t+\Delta t/2)) \tag{33}$$

will zero the first two lines yielding $\mathbf{u}_i(t+\Delta t) = \mathbf{v}_i(t+\Delta t)$ as desired. Note that this is the same coupling force expression derived in Ref. [22] except that now m_n^i and the times scale for the impulse are fully specified.

2. *Coinciding Particles case* where particles are either isolated or exactly coincide so for any mesh site j either only one particle i has $\xi_{ij} \neq 0$ or there are Q particles that exactly coincide. With a slight modification of m_n^i,

$$m_n^i \equiv \frac{\eta_{iA}^2\sum_j \rho_j \xi_{ij}\Delta x^3}{\sum_j \xi_{ij}\eta_{ij}} \tag{34}$$

and defining the spreading weight η_{ij} as

$$\eta_{ij} = \xi_{ij}\frac{|\xi_{ij}|}{\sum_{r=1}^{Q}|\xi_{rj}|}, \tag{35}$$

we will show below that this zeros the difference between the particle and interpolated fluid velocity in both this case and in the isolated particle case, with the choice of Eq. (33) in all cases.

To see that Equations (33), (34), and (35) work for both limiting cases, first note that in case 1 the sum in the denominator Eq. (35) always has only one term, which cancels with the term in the numerator and the spreading weight $\eta_{ij} = \xi_{ij}$, the interpolation weight. As a result, in case 1 the spreading normalization $\eta_{iA} = 1$ and so Eq. (34) and (32) become numerically identical. Thus these expressions again yield $\mathbf{u}_i(t+\Delta t) = \mathbf{v}_i(t+\Delta t)$ as desired for case 1.

For limiting case 2, the denominator in Eq. (35) is $Q\,|\xi_{ij}|$, where particle i is one of the Q coinciding particles (so all have same interpolation weight), and $\eta_{ij} = \frac{1}{Q}\xi_{ij}$. The normalization then becomes $\eta_{iA} = 1/Q$. In this case, the effect will be that the Q coinciding particles equally split the properties of one "super" pseudo-particle with a total full weight of ξ_{ij}. Further, the last line in the velocity difference equation, Eq. (31) can be rewritten as identical terms to the first terms in the brackets on the previous two lines (so a total of Q such terms). As $\eta_{iA} = 1/Q$ in this case, we once again get $\mathbf{u}_i(t+\Delta t) = \mathbf{v}_i(t+\Delta t)$ as desired.

In the more general case where the interpolation weights for multiple particles may overlap, but not perfectly coincide, Eq. (35) is a weighted sum of the Q particles contributing to a fluid mesh site j. In this case we may not get $\mathbf{u}_i(t+\Delta t) = \mathbf{v}_i(t+\Delta t)$ exactly, but it should be very close.

One downside to using Eq. (35) is that it requires two passes over the particles to compute. The first pass works out

$$w_{1,j} = \sum_s |\xi_{sj}|, \tag{36}$$

which is stored in an array (with dimensions of the fluid mesh size) and in the second pass over the particles we work out

$$\eta_{ij} = \frac{1}{w_{1,j}}\xi_{ij}|\xi_{ij}|, \tag{37}$$

for each particle. While more expensive than a single pass, the overall fluid-particle force calculation is still $\mathcal{O}(N)$ for N particles. In most examples of using the *lb/fluid* package this calculation is a small part of the total computational load so that the extra work is negligible and well worth the cost. Significantly, this removes the need for the user to specify a correct value for m_n^i. This seemed to be a significant issue for many users of the first version of *lb/fluid* as poor choices for m_n^i resulted in poor performance, especially in thermostating of the particles when a fluid with thermal fluctuations was used.

2.3. Interpolation stencils

The algorithm above requires a choice of interpolation stencil with weights ξ_{ij} which interpolated particle i information at location $\mathbf{r}_{pi} = (x_{pi}, y_{pi}, z_{pi})$ to fluid mesh site $\mathbf{r}_{fj} = (x_{fj}, y_{fj}, z_{fj})$. The stencil weight

$$\xi_{ij} = \phi(\Delta x_{ij})\phi(\Delta y_{ij})\phi(\Delta z_{ij})$$

where $\Delta\mathbf{r}_{ij} = (\mathbf{r}_{pi} - \mathbf{r}_{fj})/\Delta x$ and Δx is the lattice spacing. Three different stencils are provided in the package with the choice specified by the user in the input script:

1. *Trilinear stencil*: This is a standard 2-point linear interpolation (in each dimension) so

$$\phi(\Delta) = 1 - |\Delta|, \qquad |\Delta| < 1. \tag{38}$$

2. *Three-point immersed boundary stencil*: This is a commonly used immersed boundary stencil [23],

$$\phi(\Delta) = \begin{cases} \frac{1}{3}(1 + \sqrt{1 - 3\Delta^2}), & |\Delta| \le 0.5 \\ \frac{1}{6}(5 - 3\Delta - \sqrt{1 - 3(1-\Delta)^2}), & 0.5 < |\Delta| < 1.5. \end{cases} \tag{39}$$

3. *Keys' cubic spline interpolation stencil*: This 4-point stencil is often used for scaling images and video as it preserves detail better than linear interpolation. Keys showed that this produces third-order accuracy [27]. The stencil is

$$\phi(\Delta) = \begin{cases} \frac{3}{2}|\Delta|^3 - \frac{5}{2}\Delta^2 + 1, & |\Delta| \le 1 \\ -\frac{1}{2}|\Delta|^3 + \frac{5}{2}\Delta^2 - 4|\Delta| + 2, & 1 < |\Delta| < 2. \end{cases} \tag{40}$$

These stencils are all zero outside their defined range. The default stencil (used if not otherwise specified by the user) is the trilinear stencil.

There are advantages and disadvantages to each stencil. The trilinear stencil's main advantage is that it extends at most Δx from the particle which allows particles to approach walls (and each other) without significant overlap of the stencil. The disadvantage of the trilinear stencil is that it has stronger commensurability effects from the underlying mesh [18]. The immersed boundary stencil reduces these mesh effects [23] but at the cost of reduced accuracy. In addition, the re-weighting necessary for densely packed nodes in the spreading stencil η_{ij} somewhat negates the stencil smoothness over the mesh. The 4-point stencil produces similar results to the trilinear stencil. It produces a more accurate interpolation at the cost of extending up to $2\Delta x$ from the particle. However, the stencil accuracy appears to be one of the smaller discretization errors overall so usually one of the other two stencils would be preferred.

2.4. Degrees of freedom and temperature

As with the earlier package version, it is possible to add thermal fluctuations to *lb/fluid* (via the `noise` option). The algorithm for the noise in the first version of the package was documented in [13]. Here, as we have removed the exponential integrator and use the standard LB algorithm, the noise implementation is equivalent to that in [28]. The noise is that of an ideal fluid obeying the fluctuation dissipation relation

$$\langle s_{\alpha\beta}(\mathbf{r}, t)s_{\gamma\nu}(\mathbf{r}', t')\rangle = 2\eta_{\alpha\beta\gamma\nu}k_BT\delta(\mathbf{r} - \mathbf{r}')\delta(t - t'), \tag{41}$$

where $s_{\alpha\beta}$ is the fluctuating stress and $\eta_{\alpha\beta\gamma\nu}$ is the viscosity (tensor). Nonideal fluids would require augmenting this approach [29] which is not done here.

As noted above, the particle-fluid coupling acts as a constraint that forces the particle velocity and fluid velocity interpolated to the particle location to match. As a result, the i'th particle moves as a pseudo-particle with an effective mass $m_i + m_n^i$. If the temperature of the particle is then "measured" using the kinetic energy and equipartition, then this effective mass is what should be used in the kinetic energy. However, if two or more particles are in close proximity the particle-fluid constraint on each particle can make them dependent on each other. This is most clear for coinciding particles (case 2 in the previous subsection). In that case, if we have two coinciding particles, they are effectively both constrained to move together with their accompanying fluid. Two such particles should only count as one particle when counting the degrees of freedom (*dof*). Similar issues in counting degrees of freedom are encountered in other constraints in LAMMPS such as the SHAKE constraint. A fairly convenient measure for *dof* in this case would be

$$dof_{LB} = 3\sum_s \eta_{sA}. \tag{42}$$

For isolated particles $\eta_{sA} = 1$ so this just gives three degrees of freedom per particle (as would be expected for a point particle).

For Q coincident particles $\eta_{sA} = 1/Q$ so we would get just 3 degrees of freedom as would also be expected as they act as a single particle. The *lb/fluid* fix has a scalar output for the particle temperature using equipartition giving

$$T = \frac{\sum_i (m_i + m_n^i)\mathbf{v}_i^2}{dof_{LB} k_B}. \tag{43}$$

If other constraints, such as the rigid fix, are also present then the *dof* removed by the other constraint will not be accounted for. However, in that case the user can usually calculate the number of degrees of freedom fairly straightforwardly (e.g. 6 per rigid body that can translate and rotate freely in 3 dimensions) and so there is an option for the user to specify the *dof* to override the default calculation.

There is also a vector output that the user can access for output (for instance in a thermo command) that give the fluid temperature, total mass, and total momentum in the system. Total here includes fluid plus all particles.

3. Installation and execution

The LB fluid and particle force calculations are implemented in LAMMPS through the creation of a series of "fixes" in the *lb/fluid* package. In the LAMMPS formalism a "fix" is almost any operation performed during the execution of a normal time step of the program. This package is included as a user package and so must be included in the build either using the `make yes-latboltz` if building using make, or `cmake -D latboltz=yes` if using CMake (this copies the package source files into the main LAMMPS source directory). Further steps in compiling LAMMPS are described well in the standard LAMMPS documentation.

The main fix is *lb/fluid*. This is called with three required arguments and also has several optional arguments. The first required argument, as with all LAMMPS fixes, is `nevery` which is an integer which tells LAMMPS to call this fix every `nevery` time steps. In almost all cases, this should be set to 1. The second and third required arguments are the fluid `viscosity` and `density`. These should be specified in the same units as used in the rest of the input file (usually specified via the LAMMPS `units` command earlier in the input script). Convenient units when using the package include the `micro` and `nano` units (for micro and nano-fluidic simulations respectively).

The algorithm uses LAMMPS `MPI` domain decomposition to partition the lattice across processors. As a result, each processor is responsible for the atoms and LB lattice sites within its portion of the domain. In addition, each local lattice has a "ghost" region of width two `dx` around it that overlaps the domain in neighboring processors. Information in these ghost regions is communicated between the processors (indicated by `comm` in Algorithm 1) in each timestep. To ensure proper functioning of the algorithm, it is important that the communication cutoff in LAMMPS, for particles that interact with the LB fluid, is large enough to cover the 2`dx` ghost region of the LB lattice (i.e. LAMMPS needs to know about the ghost atoms in the ghost region covered by the LB lattice). This can be set via the LAMMPS `comm_modify cuttoff` command. Care should be taken, however, that you don't inadvertently set this too small based on the other potentials present in the system.

Optional arguments for *lb/fluid* include `dx`, the mesh spacing for the fluid grid, and `dm`, the mass unit for internal LB calculations. For computational efficiency it is typically best to select `dx` to be no smaller than absolutely necessary. Generally `dm` should be chosen so that the fluid density scaled to units of $\mathtt{dm}/\mathtt{dx}^3$ is of order one (to minimize roundoff errors). The lattice Boltzmann algorithm uses the same time step as the rest of LAMMPS which is set using the standard LAMMPS `timestep` command. For a number of algorithmic reasons, the speed of sound is set to $c/\sqrt{3}$ (see discussion of a_0 in Section 2.1). Generally this will *not* be the actual speed of sound in the fluid you are modeling. As the algorithm is not designed to accurately reproduce sound wave dynamics, this is not a problem as long as there is a separation of velocity scales in your problem (i.e. any fluid velocities that arise in your problem are roughly an order or magnitude, or more, smaller than $c/\sqrt{3}$). This is a common constraint of standard LB algorithms and not unique to the *lb/fluid* package. By default the algorithm uses a 15-velocity LB model but a 19-velocity model can be used by specifying the `D3Q19` option.

The `stencil` option can be used to select among the stencils discussed in Section 2.3. This option takes a second integer argument to select the 2, 3, or 4-point stencil (2-point is the default). Fluids exhibit thermal fluctuations at small length scales (nm) leading to Brownian motion of suspended particles. Thermal fluctuations are controlled via the `noise` option that takes two further arguments, a Temperature (typically in Kelvin) and a seed for the random number generator (integer). The resulting temperature in the fluid can be assessed via the array variable `f_ID[1]` where `ID` is the fix id (set by the user in the input script) for the *lb/fluid* fix.

There are also a variety of options for system boundaries. One could, in principle, use atoms in fixed positions to form the fluid boundaries as that is what the immersed boundary method is designed for. However, for flat boundaries along coordinate directions it is simpler and more computationally efficient to apply boundary conditions at these walls. The boundary conditions for *lb/fluid* are specified independently to those for the atoms. Usually the user will specify similar boundary conditions for both, but there are cases where it can be useful to specify different boundary conditions [3]. The simplest case, and default if not specified, is periodic boundary conditions in all directions for the fluid. The next simplest case is periodic boundaries in two directions and flat boundaries coinciding with the mesh in the other (taken to be along z) using the `zwall_velocity` option. In this case, the velocity of the fluid can be specified on the top/bottom boundary, providing a straightforward way of implementing Couette flow (shear) in the system between the walls. The user can also specify a body force using the `bodyforce` option on the fluid which, when between stationary walls, provides Poiseuille flow. Much more complex geometries can be created using the pit geometry options. In this case the boundary condition at the walls is always no-slip/no-flow and the boundaries are technically located halfway between lattice sites. The pit options allow channels, pits, barriers, and other geometries and will be illustrated in Section 4.

The main form of output is the `dumpxdmf` option that takes two arguments, an integer indicating the dump is done every this number of time steps, and a file name to write to. The full grid density and velocity field is written in xdmf format, a binary format that can be read by programs such as the open source Paraview [30]. In addition to plotting, Paraview also provides some tools for analysis of the density/fluid flow and can also read normal LAMMPS dump files written in the VTK format. Density and fluid flow plots in Section 4 were produced with Paraview.

Finally, there is a `write_restart` option to write data periodically to restart the *lb/fluid* fix and a corresponding `read_restart` option to read data to restart the fix. These restart files would need to be used in conjunction with corresponding restart commands for LAMMPS in order to completely restart the simulation. Care should be taken to ensure they are written at corresponding time steps. The *lb/fluid* restart files can be very large when the fluid mesh has a large number of sites.

A second fix that will almost always be used with the *lb/fluid* fix is the *lb/viscous* fix included as part of this package. This fix is analogous to the standard LAMMPS *viscous* fix. This fix applies

the hydrodynamic forces to the particles. It also rescales the non-hydrodynamic forces by $m_i/(m_i + m_n^i)$ so that the particles move as if they have a mass $m_i + m_n^i$, as discussed in Section 2.2 above. For the particles to actually move though, an additional standard LAMMPS fix such as the *NVE* or *RIGID* fix needs to be invoked. Note that during a timestep, fix operations run *in the order they are invoked* in the input script. As a result, any forces added by fixes such as the *addforce* fix, as opposed to standard pair/bond forces, will only be properly rescaled if they are called before the *lb/fluid* and *lb/viscous* fixes. If they are added after these fixes, *lb/fluid* will not know about these forces and you will not end the time step with matching velocities of the particle and fluid interpolated to the particle position.

There is also a fix, *lb/momentum* that works analogously to the built-in *momentum* fix for particle momentum only, but additionally includes the fluid momentum. This allows the user to subtract off the total (particle plus fluid) linear momentum from the system. The algorithm should conserve momentum to numerical precision so this fix is primarily useful for long runs and systems with large mesh sizes and/or large numbers of particles.

Documentation is provided in the */doc* directory in a format compatible with the standard LAMMPS documentation.

4. Test run descriptions

We provide here some tests that illustrate the use and performance of the package. These examples, along with some sample output, are included in the */examples/PACKAGES/latboltz* directory. Although this version of the package is faster than the original, and much faster than modeling the "solvent" explicitly using particles, the majority of the computational time is still typically spent updating the lattice Boltzmann fluid. As such, in most cases increasing the number of lattice sites will significantly increase the computation time while increasing the number of particles will only modestly increase the computational time. However, the package parallelizes well so using additional processors can ameliorate the added wall clock time. Note that the system dimensions must be divisible by Δx. The parallelization uses the standard LAMMPS domain decomposition so if you wish to use N processors in a given direction then $L/(N\Delta x)$ must also be an integer (where L is the system size in the given direction).

4.1. Subgrid particles

Force couplings between fluids and particles that are related to the velocity difference between the fluid and particle, such as the one we use in Eq. (33), are very closely related to the Debye-Bueche-Brinkman (DBB) model [31–33] (similar model except the linearized Navier-Stokes is used). In such a model, the drag force experienced by a spherical particle with radius R moving with speed v through an otherwise quiescent fluid is

$$F = \frac{2\beta^2}{2\beta^2 + 9} F_S, \tag{44}$$

where $F_S = 6\pi\eta R v$, and β is a dimensionless parameter given by $\beta = R\sqrt{\gamma\lambda/\eta}$. γ is the coefficient of the velocity difference in Eq. (33) so here

$$\gamma = \frac{2}{\Delta t}\frac{m_i m_n^i}{m_i + m_n^i}. \tag{45}$$

Finally, λ is the density of particles so for a uniform density we would have $\lambda = N/(4/3\pi R^3)$ if N is the total number of particles comprising our composite spherical particle. Putting this together one can obtain the following relation:

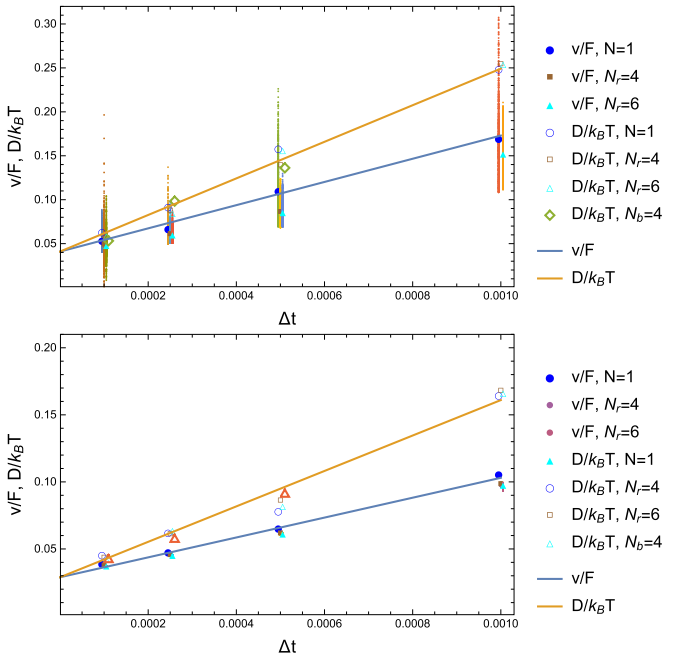


Fig. 1. Inverse of the drag coefficient measured using a constant force in the x-direction to pull the particle through the fluid (solid symbols) and using the fluctuation-dissipation relation valid in the no-slip limit (open symbols) using the trilinear (a) and three-point immersed boundary (b) stencils. Circles are for single point particles ($N = 1$), squares are for 4-point ($N_r = 4$) and triangles for 6-point ($N_r = 6$) composite particles, both with nodes within a shell of radius $0.204\Delta x$ and with fixed relative distances (implemented using the standard LAMMPS *rigid* fix), and diamonds for 4-point composite particles ($N_b = 4$) with harmonic bonds between the nodes (same average size as the corresponding rigid composite particles). Results for different particle constructions at the same value of Δt have been shifted slightly horizontally so that they can be distinguished. The lines correspond to linear fits through the corresponding data (blue for v/F and orange for D/k_BT). (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

$$\frac{v}{F} = \frac{1}{6\pi\eta R} + \frac{m_i + m_n^i}{2m_i m_n^i N}\Delta t. \tag{46}$$

The constant is the inverse of the drag coefficient for Stokes Drag limit that is achieved as $\Delta t \to 0$. Note that the term proportional to Δt is only the error related to the DBB-like coupling term between the fluid and particle. There could be other discretization errors although one would expect those to be proportional to Δt^2 as that is the order of discretization errors for the lattice-Boltzmann (and velocity-Verlet should be proportional to Δt^4).

We first examine "subgrid" particles, that is those of size less than Δx. As the resolution of the mesh is Δx, this limits the effective size of a particle [18]. To some degree, such particles represent the "worst case scenario", at least for relative errors, as the particle size is comparable to the resolution of the algorithm. We construct "composite" particles from one, four, and six nodes. Nodes are placed uniformly on a shell of radius $0.204\Delta x$ initially and their relative location is either held fixed using the LAMMPS *rigid* fix to integrate the composite particles motion or are bonded with a harmonic bond of unit strength. The single node particle and bonded particles are integrated using the LAMMPS *NVE* fix. Hydrodynamic forces are created via the *lb/fluid* and *lb/viscous* fixes as discussed above. The total mass of the composite particle is the same for all constructions. Here we use values for the fluid density and viscosity appropriate for water and use LAMMPS *nano* units for all measurements (nm, ns, ag).

The drag force is measured in two different ways. In the first, a constant force is applied in the x-direction to pull the particle through the fluid and measure its resulting speed v. To avoid bias

from selecting a specific particle path relative to the mesh sites a small oscillating force is applied in the transverse directions so that the particle samples distances of 1-2 Δx in the transverse direction. Results are shown using solid symbols in Fig. 1a for the trilinear stencil and in Fig. 1b for the three-point stencil. Note that all subgrid particle constructions (one, four, six nodes, rigid and elastically bonded) give similar average results for a given stencil indicating that the stencil overlap is accurately being accounted for. The results also follow the linear relationship expected from Eq. (46). One significant difference between the results for the trilinear and three-point stencil is the much wider range of values measured in the trilinear as the composite particles traverse the lattice. This is expected and was seen in earlier versions of the algorithm [18]. The three-point immersed boundary stencil is explicitly constructed to avoid this issue [23], at the cost of smearing the particle over a finite range. In particular, the maximum of the immersed boundary stencil is $1/2$ (along each coordinate direction) while the trilinear stencil maximum is one when the particle and fluid mesh node coincide. In this algorithm, the three point stencil has a uniform, and larger, m_n^i across the lattice whereas m_n^i varies for the trilinear stencil depending on where in a given lattice cell the particle is. The larger effective size of the particles for the 3-point stencil is also evident in the intercept (proportional to $1/R$ in Eq. (46)) for this case being smaller than for the trilinear stencil.

In the second method, we make use of the fluctuation-dissipation relation that is true in the no-slip limit, where the drag coefficient should be k_BT/D where D is the diffusion constant for the particle in a fluid with thermal fluctuations at temperature T. Technically this is only true in the no-slip limit so agreement between the value from the fluctuation-dissipation and drag measurements can be used as a test of whether or not we have achieved that limit, or at least an acceptable approximation to that limit. Here we put 100 subgrid particles (constructed in the same manner as for the drag test) in the box and measure their diffusion constant using the built-in LAMMPS *msd* compute (mean-square-displacement) and following closely to the DIFFUSE example included in the LAMMPS distribution. In principle, the diffusion constant will be affected by the presence of the other particles. However, this is a very dilute solution and we verified that a single particle (in a much longer run) has the same diffusion constant within the statistical errors of our measurement in this case. Not surprisingly, the drag coefficient values for the two methods only coincide as $\Delta t \to 0$. For all practical purposes though, a time step of 0.0001 yields statistically indistinguishable results and a time step of 0.00025 would probably be acceptable for many situations.

Considering that we are explicitly forcing the particle and fluid velocity at the particle to match at the end of the time step it might be surprising that the results above demonstrate that the "no-slip" limit is only achieved in the limit that $\Delta t \to 0$. The deviation comes from the fact that what we are actually simulating is a DBB style model where the force coupling, Eq. (33), necessarily contains the inverse of the time step so in the limit, $F_H \sim (\mathbf{u} - \mathbf{v})/\Delta t \to 0/0$ as $\Delta t \to 0$. The algorithm fairly faithfully reproduces the expected analytical results of the DBB model in this. Earlier implementations of the algorithm often gave poor temperature reproduction. That is not the case here. The average kinetic temperature (scalar fix output computed via the kinetic energy using Eq. (43)) is found to be reasonably independent of Δt and is within a few degrees of the set point (300 K in this example).

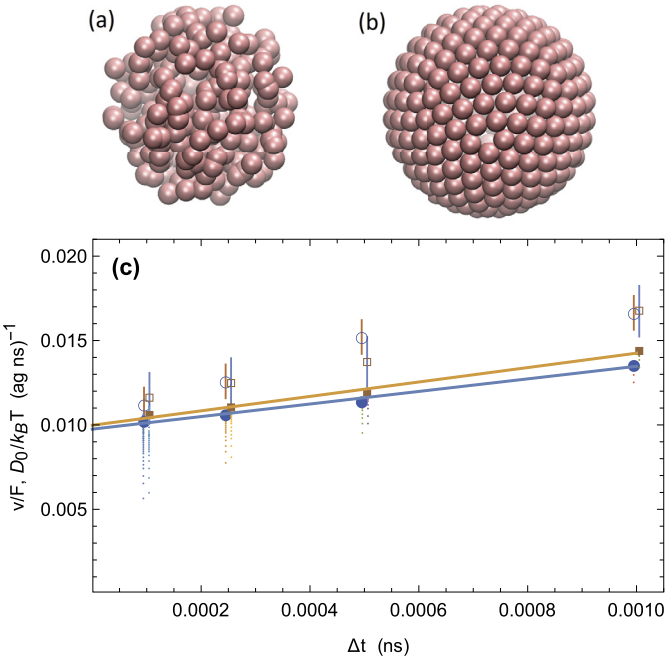


Fig. 2. (a) Starting configuration for 284 "atoms" in LAMMPS created inside a spherical region (using built-in LAMMPS commands). (b) Final configuration of atoms where they have condensed onto a spherical surface (making use of LAMMPS wall potential on the surface) after a normal MD run using a Langevin thermostat with a gradually decreasing temperature. This configuration is then used as the surface mesh of a colloid in an LB fluid simulation. (c) Inverse of the drag coefficient measured using a constant force in the x-direction to pull the colloid through the fluid (solid symbols) and using the fluctuation-dissipation relation valid in the no-slip limit (open symbols) using the trilinear (squares) and three-point immersed boundary (circles) stencils. The Diffusion constant is corrected for finite-size effects which are more significant in this simulation than for the subgrid particles in the previous section. Results for different particle constructions at the same value of Δt have been shifted slightly horizontally so that they can be distinguished.

4.2. Extended surface with mesh

This example illustrates how standard LAMMPS routines can be used to create a surface mesh to model a large colloidal sphere. A sequence of MD runs is first done to create the sphere. The initial configuration consists of 284 randomly created atoms (using LAMMPS *create_atoms* command) inside a spherical region (defined via the *region* command), as illustrated in Fig. 2a. LAMMPS *soft* potential (repulsive) is used during a short MD run (20000 steps) to push particles, some initially very close together, apart. At the same time, a Lennard-Jones potential is put in place on the wall of the sphere (using a LAMMPS *fix wall/region* command) which starts the condensation of atoms onto the wall of the sphere. This is followed by a somewhat longer longer run (500000 steps) where the particles interact with each other via a repulsive Lennard-Jones potential, interact with the sphere wall with an attractive Lennard-Jones potential, and experience a gradually decreasing temperature (using a *fix langevin* command). A quick minimization then finalizes the atom locations on the surface of the colloidal sphere leaving the configuration illustrated in Fig. 2b.

The colloid is then subject to simulations using the *lb/fluid* package similar to those for subgrid particles. We first measure the drag force coefficient for the colloid by applying a constant force in the x-direction and then measuring it's resulting speed through the fluid. This is plotted in Fig. 2c as solid symbols for both the trilinear stencil (square) and immersed-boundary 3-point stencil (circle). The colloid is $6.67\Delta x$ in diameter and the *relative* difference in the drag coefficient between the two stencils, or as a function of Δt is considerable less than seen for the subgrid parti-

cles. As mentioned above, this smaller relative error is expected as the particle size increases to a size substantially larger than Δx.

We also do a simulation with thermal noise in the fluid. In the x-direction we add an external harmonic potential centered at $x = 0$, $V = \frac{1}{2}ax^2$. The equipartition theorem then allows a temperature measurement independent of Eq. (43) as $T_x = a\langle x^2\rangle/k_B$. As the particle is unbound in the yz-plane it undergoes a random walk from which we can measure the diffusion constant. Similar to the subgrid particles, the fluctuation-dissipation theorem relates the inverse of the drag coefficient to D/k_BT in the no-slip limit. As there is only one colloid in this simulation and the drag force is considerably larger (hence smaller diffusion constant) for the colloid than for the subgrid particles, these simulations needed to be considerably longer than those for the subgrid particles to obtain similar statistical accuracy. The results for the diffusion measurement are shown in Fig. 2c as open spheres. As for the subgrid particles, the no-slip limit is achieved within statistical errors for $\Delta t = 0.0001$.

Eq. (43) is not as accurate in measuring the temperature for the large extended particles. This is not surprising as the derivation is only exact in the limit of very closely, or very widely, spaced nodes. For the trilinear stencil the temperature from Eq. (43) gave an average of 313 K, compared to a set point of 300 K while the three-point stencil gave an average of 259 K. The statistical error is estimated at 1-2 K so this suggests a systematic error. There was no discernible variation with Δt. To test whether the variation was due to the approximations in deriving the formula or an actual systematic error in the temperature we made use of the equipartition theorem for the particles diffusing in a harmonic potential to get an independent measure of the temperature. This yielded temperatures within 10 K (3%) of the 300 K setpoint for both stencils, smaller than the statistical errors for this measurement. (The correlation time for kinetic temperature is very short whereas the correlation time for a measurement based on positions is much longer, hence the larger statistical errors.) From this we conclude that the large colloid is well thermalized in the LB fluid.

4.3. Immersed boundary wall

Boundary conditions for the LB algorithm can be implemented using a number of methods, the most common of which is the bounce-back condition that will be discussed in the next subsection. However, since the LB fluid is coupled with MD procedures of LAMMPS, it is also possible to create complex bounding geometries using MD particles. In this section, we illustrate such *immersed boundaries* with the example of translocation of a single polymer chain through a solid-state pore.

Our system consists of a polymer chain with composite monomers, a solid-state nanopore carved out of an atomistic wall, and the LB fluid. The simulation box has dimensions of 48 nm $\times$ 30 nm $\times$ 30 nm. Snapshots of the system at the initial stages of the simulation and upon insertion of the polymer are shown in Fig. 3. Normally these simulations would be done with thermal noise, but here we illustrate the flow without noise to demonstrate the backflow effect (as mentioned above, it is straightforward to turn on the noise in the package with the `noise` option).

The polymer chain comprises of composite monomers with one central atom and 31 shell atoms. The shell atoms interact with the fluid via *fix lb/fluid* and *fix lb/viscous* while the excluded volume effects are ensured using a shifted-truncated Lennard-Jones (LJ) pairwise interaction between the central atoms, and the central and wall atoms. The presence of the shell gives the monomers a well defined hydrodynamic radius [18]. The composite monomers are treated as a rigid body by applying *fix rigid*. This decreases the computational cost of the simulations without compromising

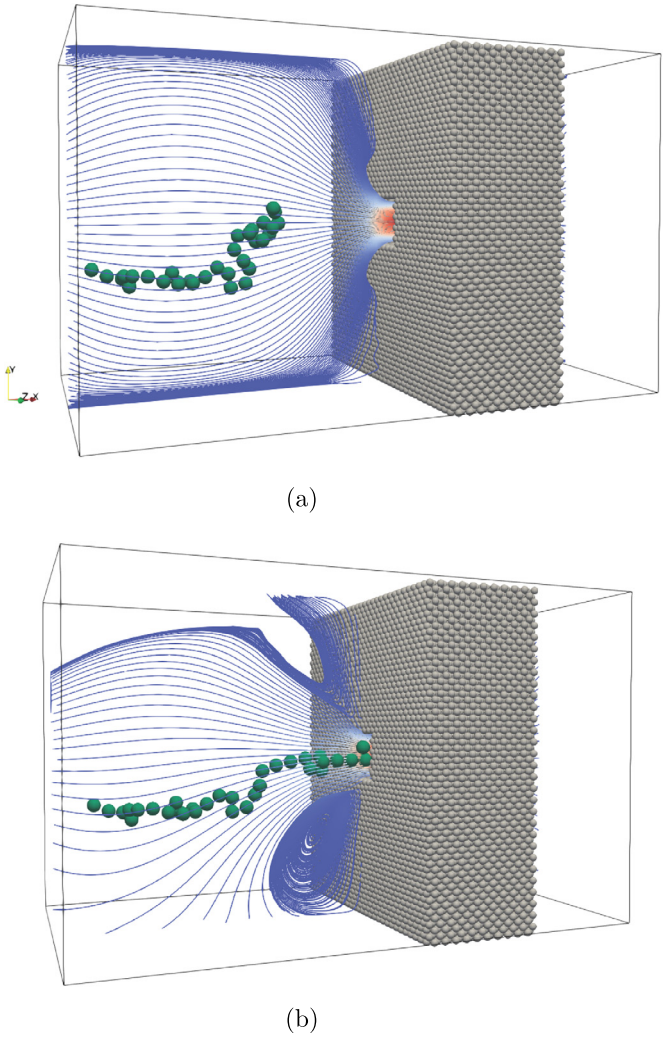


Fig. 3. A realization of polymer translocation simulated using multiscale MD-LB method. The blue curves show the fluid streamlines on the central plane, the polymer chain is shown as a succession of green spheres, and the grey block of spheres is the solid-state wall. The nanochannel is created by a hole in the middle of the wall. (a) Shows an initial state where the chain is far away from the pore while (b) shows the chain as it threads through the nanochannel. The presence of the chain inside the channel decreases the cross-section and creating a backflow which results in formation of vortices around the channel.

accuracy. The central atoms are connected via Finitely Extensible Nonlinear Elastic (FENE) bonds [34].

The wall atoms, similar to shell atoms, interact with the lb fluid via fixes *lb/fluid* and *lb/viscous*. The nanopore is made by deleting a 10 nm $\times$ 4 nm $\times$ 4 nm block from the middle of the wall. The calculated forces on the wall are set to zero at every timestep by applying *fix freeze* from LAMMPS routines.

The viscosity and density of the fluid is set to 1/10 of that of water to accelerate diffusive dynamics and reduce the simulation time while the kinematic viscosity is left identical to that of water. The *scaleGamma* flag with scale factor -1 is invoked for wall atoms which explicitly takes the limit $m_i \to \infty$ in Eq. (33), (18), and (19) consistent with the walls being immobile. This prevents any flow going through the wall as well as makes any momentum build-up in the system insignificant. This can also be obtained by setting the mass of the wall atoms to a large number [35]. However, using *scaleGamma* flag is less computationally costly. In this case, it is possible to hollow out the middle sections of the wall in order to improve computational speed and total wall time. Care must be taken to ensure the wall does not get too thin, however. A slight

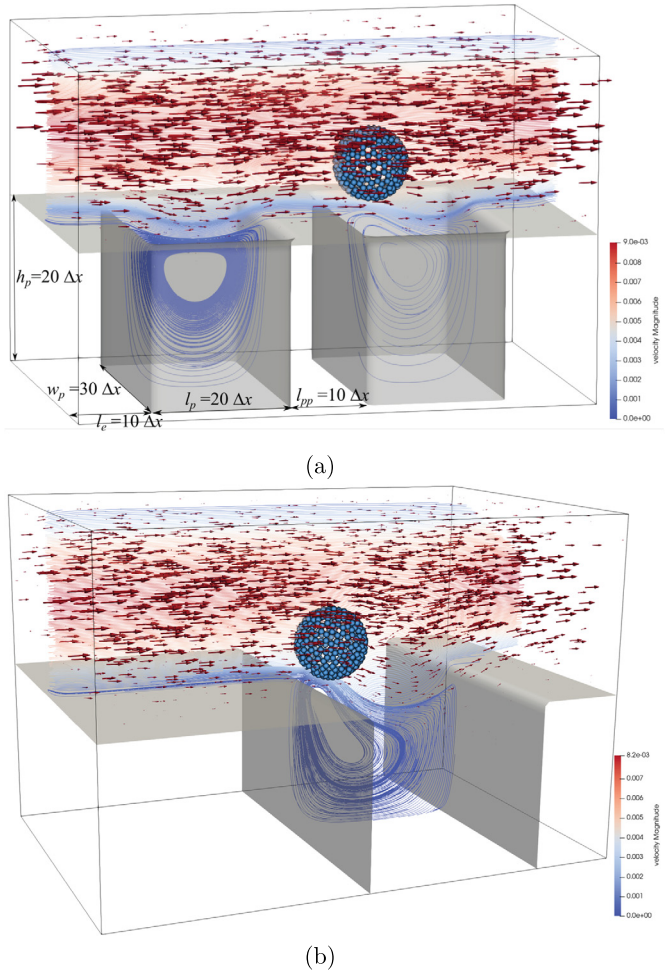


Fig. 4. Pit geometry examples.

change in the velocity profile indicating leakage is detectable for very thin walls (one or two atomic layers) so the number of layers should be tuned for particular purposes. The trilinear stencil is used as interpolation scheme due to advantages motioned in [36]. The pressure-driven flow is implemented by applying a pressure jump at the boundaries in the x-direction using the flag *pressurebcx*. The thermal noise was turned off for the tests presented in this section, but we expect the average velocity profile to be similar to that shown in 3 for fluid with thermal noise [35].

In Fig. 3, snapshots of the system at initial stages and while threading are shown. The fluid streamlines are illustrated on the central plane ($x = 15$). The streamlines are read from an `xdmf` file output by *fix lb/fluid* command and visualized in the open-source data analysis, and visualization tool Paraview [30]. This feature can be turned on by adding the flag *dumpxdmf*. The size of the polymer beads are exaggerated for illustration purposes. At the beginning when the chain is far away from the pore the streamlines are only slightly affected by the presence of the chain. However, due to backflow effects, vortices appear around the pore as the chain goes through the nanopore.

4.4. Pit geometry

The package has a *pits* option that implements bounce-back boundary conditions (no-flow, no-slip) at the top and bottom (in z), and front and back (in y), of the simulation domain. In addition, the lower wall(s) can have a more complex geometry specified by a number of pits whose dimensions can be set by the *pits* option arguments. A couple of examples are illustrated in Fig. 4. The dimension arguments are illustrated in Fig. 4(a).

In both of these examples, the flow along the x-direction is generated by the *pressurebcx* boundary condition that imposes a pressure jump at the entrance to the channel. In Fig. 4(a) there are periodic boundary conditions in the y-direction whereas in Fig. 4(b) there are sidewalls and the pit extends across the whole domain in the y-direction. The geometry in Fig. 4(b) is similar to that used in [37].

Lattice sites outside of the active domain are not updated to ensure efficient computation but are present in the arrays (so take up memory space). This should be considered when dividing up the domain among processors (which can be set via the LAMMPS *processors* command). For example, if you had four processors to assign to the problem illustrated in Fig. 4(b), dividing this domain into a 2×2 grid in the xz-directions would leave the processor in the lower left with essentially nothing to do. A linear division along y would balance the load among processors (at the cost of more communication at the processor boundaries) and a 2×2 grid in the xy-directions would give all processors some work but those that include the pit would have considerably more. Given the variation in processor communication versus compute speed for various hardware configurations, it is not *a priori* obvious which of these divisions will lead to the fastest runtime.

4.5. Restart

The *lb/fluid* fix also includes a restart option. This just restarts the fluid and the standard lammps restart procedure is required for restarting the particle portion of the simulation. The frequency of writing the restart for the particles should normally be the same as that for the fluid to restart where the simulation left off.

Unfortunately, the simulation will not restart exactly, though should provide statistically similar results. This is because the hydrodynamic forces used depend on the atom velocities at the half-step which are not saved by LAMMPS. This is similar to the situation with other LAMMPS pair styles such as granular, dpd, and lubricate pair styles where the same situation arises and leads to similar consequences on restart.

Restart files are written at the end of a step. As should be clear from the pseudo-code in Algorithm 1 and our discussion of the LB algorithm, it is the auxiliary distribution that is followed in the algorithm (cf. Eq. (9)), which is effectively the distribution at a half-step. At the end of a step we have updated the fluid velocity to the correct time point but the distribution is effectively half a step behind. To enable restarts with just saving the fluid distribution (i.e. and not also the fluid velocity and forces) we first compute the distribution at the end of the full step and then save that for the restart (using Eq. (9) and corresponding equations like Eq. (13)).

4.6. Scaling

We now turn to a couple of more complex simulations to demonstrate the timing and scalability in representative implementations of the method. The first example consists of colloidal particles in water sheared between two walls. This is a minimally updated example used to demonstrate the scalability of the first version of the *lb/fluid* package [10]. The second example is a toy car driving along a long channel (between two fixed walls) in air. The two examples are chosen to test different limits.

The colloidal example consists of 480 composite particles, each of radius 0.726 μm, in a system of size 16.8 μm × 16.8 μm × 6.0 μm. The system is periodic in the x and y directions and has walls which move at ±20 μm/μs to impose the shear. The density and viscosity is chosen to match water at standard conditions.

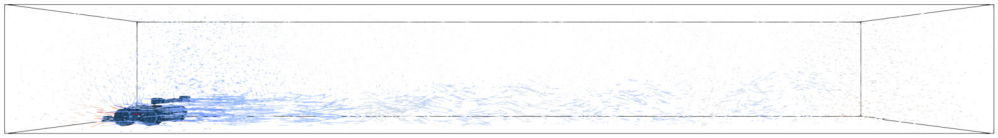


Fig. 5. Toy car example.

The system is discretized in space using $\Delta x = 0.06$ μm and uses a times step of $\Delta t = 0.001$ μs. Each composite particle is comprised of 3612 surface nodes. The resulting system contains 1734240 particles and 7.84 million lattice sites. More details, including an illustration can be found in [10] and a detailed examination of this system can be found in [38].

The second example is a model of a toy car driving down a long air-filled channel and is illustrated in Fig. 5. The system is 96 cm × 720 cm × 96 cm and the car is about 40 cm long, 15 cm wide and 12–15 cm high. The car was made by using LAMMPS *create_atoms* to put 5092 atoms in simple geometric *regions* (prisms, cylinders, blocks, and a sphere) and then using the LAMMPS *fix wall/region* in an annealing run to condense the atoms onto the surfaces (similar to what was done in Section 4.2). The system is discretized in space using $\Delta x = 1$ cm and uses a times step of $\Delta t = 0.025$ s. The car was set in motion with a velocity of 7.5 cm/s in the y-direction and maintains this velocity by using the *fix setforce* command to zero the forces on the car body. The wheels were allowed to freely rotate about the x-direction while having center-of-mass forces zeroed (rotation corresponding to "rolling" occurs naturally to reduce what would otherwise be very high shear rates between the bottom of the wall and the floor of the channel). The density (0.0012 g/cm^3) and viscosity (0.0002 Poise) are appropriate for air at standard conditions.

The simulations were run on a Compute Canada cluster (Narval) equipped with 1109 AMD Rome 7532 processors at 2.4 GHz linked with an InfiniBand Mellanox HDR network. The confined colloid example was timed for 400 timesteps and the toy car example was timed for 5000 timesteps (the time spent condensing the atoms onto the car's surfaces was not included). Fig. 6 shows both the speedup (t_1/t_p) and efficiency ($t_1/(p \cdot t_p)$), where t_i is the wall clock time for the code to run on i processors.

As can be seen, the efficiency for the confined colloid example is very good (greater than $\sim 75\%$) while the toy car is a bit lower (still greater than $\sim 50\%$). This difference is likely due to the fact that in the confined colloid example both the fluid and particles are homogeneously distributed across the system, making for a fairly uniform load for all processors (the system is parallelized using domain decomposition). In contrast, in the toy car example only the fluid is homogeneously distributed. The particles, however, are very localized (likely to 1 or 2 processors) meaning that the workload is imbalanced. As the number of particles is not too large, this imbalance is not severe, but is noticeable.

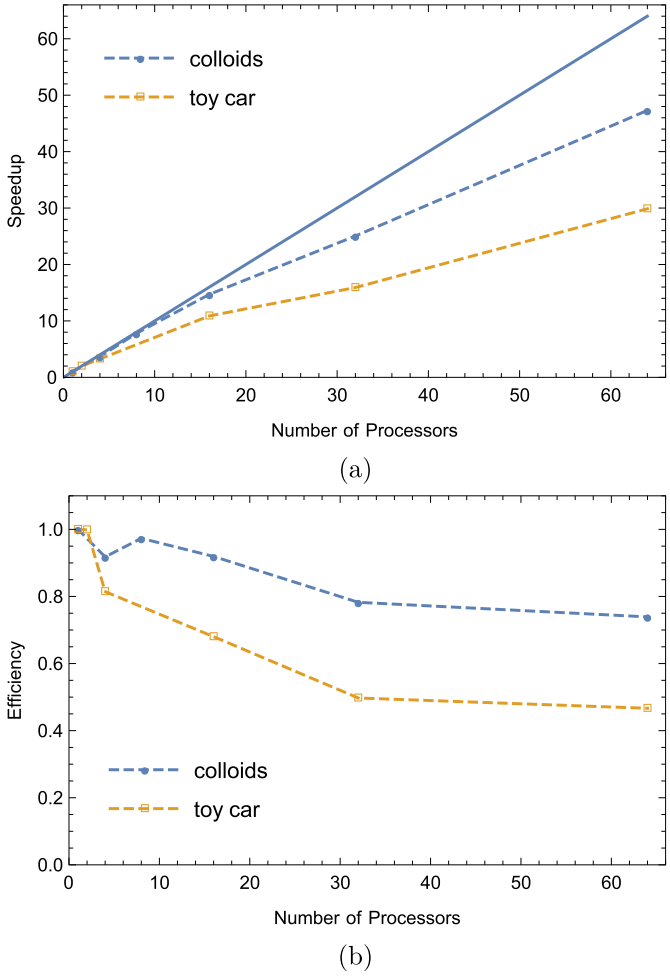


Fig. 6. Speedup (t_1/t_p) and efficiency ($t_1/(p \cdot t_p)$), where t_i is the wall clock time for the code to run on i processors.

5. Summary

The new version of the *lb/fluid* package implemented into LAMMPS is a significant improvement. The new spreading algorithm described in Section 2.2 is exact in both the dilute and concentrated limit of node densities and provides excellent performance, as demonstrated in the examples provided. The added features, such as compute outputs, and pit geometries greatly expand the variety of problems that can be solved. We also demonstrated the package shows good scaling performance with the number of processors available.

We plan on continuing to develop the *lb/fluid* package. Future anticipated features include a gpu accelerated version, a more general pit geometry, multiple relaxation times for viscoelastic fluids, and more general lattice partitions across different processors. The process for incorporating these changes are reasonably clear as there are examples from the literature. Others, such as multiphase fluids that also have widely tested algorithms could also be incorporated. However, more work needs to be done on understanding noise fluctuations in non-ideal fluids [29] to ensure these are added correctly. The current implementation's accuracy breaks down when the gap between immersed bodies becomes very small, at which point lubrication forces become dominant. There is some work on including such forces in LB [39] but it is not yet clear the best way to add the effect of thermal fluctuations to such lubrication forces.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was supported by the Natural Science and Engineering Council of Canada (NSERC). We would like to thank the Shared Hierarchical Academic Research Computing Network (SHARCNET) and Compute/Calcul Canada for the computational resources.

References

[1] B.P. Ho, L.G. Leal, J. Fluid Mech. 65 (2) (1974) 365–400, https://doi.org/10.1017/S0022112074001431.
[2] R.M. Jendrejack, D.C. Schwartz, J.J. de Pablo, M.D. Graham, J. Chem. Phys. 120 (5) (2004) 2513–2529, https://doi.org/10.1063/1.1637331.
[3] S.T.T. Ollila, C. Denniston, M. Karttunen, T. Ala-Nissila, Soft Matter 9 (2013) 3478–3487, https://doi.org/10.1039/C3SM27410A.
[4] M. Cieplak, S. Niewieczerzal, J. Chem. Phys. 130 (12) (2009) 124906.
[5] H. Tanaka, T. Araki, Phys. Rev. Lett. 85 (2000) 1338–1341, https://doi.org/10.1103/PhysRevLett.85.1338, https://link.aps.org/doi/10.1103/PhysRevLett.85.1338.
[6] P.N. Segrè, F. Liu, P. Umbanhowar, D.A. Weitz, Nature 409 (2001) 594–597, https://doi.org/10.1038/35054518.
[7] A.J.C. Ladd, J. Chem. Phys. 88 (8) (1988) 5051–5063, https://doi.org/10.1063/1.454658.
[8] A.P. Thompson, H.M. Aktulga, R. Berger, D.S. Bolintineanu, W.M. Brown, P.S. Crozier, P.J. in 't Veld, A. Kohlmeyer, S.G. Moore, T.D. Nguyen, R. Shan, M.J. Stevens, J. Tranchida, C. Trott, S.J. Plimpton, Comput. Phys. Commun. 271 (2022) 108171, https://doi.org/10.1016/j.cpc.2021.108171.
[9] S. Plimpton, J. Comput. Phys. 117 (1) (1995) 1–19, https://doi.org/10.1006/jcph.1995.1039, https://www.sciencedirect.com/science/article/pii/S002199918571039X.
[10] F. Mackay, S. Ollila, C. Denniston, Comput. Phys. Commun. 184 (8) (2013) 2021–2031, https://doi.org/10.1016/j.cpc.2013.03.024, https://www.sciencedirect.com/science/article/pii/S001046551300132X.
[11] V. Lobaskin, B. Dünweg, New J. Phys. 6 (2004) 54, https://doi.org/10.1088/1367-2630/6/1/054.
[12] M.E. Cates, K. Stratford, R. Adhikari, P. Stansell, J.-C. Desplat, I. Pagonabarraga, A.J. Wagner, J. Phys. Condens. Matter 16 (38) (2004) S3903–S3915, https://doi.org/10.1088/0953-8984/16/38/009.
[13] S.T.T. Ollila, C. Denniston, M. Karttunen, T. Ala-Nissila, J. Chem. Phys. 134 (6) (2011) 064902, https://doi.org/10.1063/1.3544360.
[14] R.W. Nash, R. Adhikari, M.E. Cates, Phys. Rev. E 77 (2008) 026709, https://doi.org/10.1103/PhysRevE.77.026709, https://link.aps.org/doi/10.1103/PhysRevE.77.026709.
[15] P.L. Bhatnagar, E.P. Gross, M. Krook, Phys. Rev. 94 (1954) 511–525, https://doi.org/10.1103/PhysRev.94.511, https://link.aps.org/doi/10.1103/PhysRev.94.511.
[16] X. He, L.-S. Luo, Phys. Rev. E 56 (1997) 6811–6817, https://doi.org/10.1103/PhysRevE.56.6811, https://link.aps.org/doi/10.1103/PhysRevE.56.6811.
[17] N.S. Martys, X. Shan, H. Chen, Phys. Rev. E 58 (1998) 6855–6857, https://doi.org/10.1103/PhysRevE.58.6855, https://link.aps.org/doi/10.1103/PhysRevE.58.6855.
[18] S.T.T. Ollila, C.J. Smith, T. Ala-Nissila, C. Denniston, Multiscale Model. Simul. 11 (1) (2013) 213–243, https://doi.org/10.1137/110858756.
[19] Z. Guo, C. Zheng, B. Shi, Phys. Rev. E 65 (2002) 046308, https://doi.org/10.1103/PhysRevE.65.046308, https://link.aps.org/doi/10.1103/PhysRevE.65.046308.
[20] P.J. Dellar, Phys. Rev. E 64 (2001) 031203, https://doi.org/10.1103/PhysRevE.64.031203, https://link.aps.org/doi/10.1103/PhysRevE.64.031203.
[21] S. Plimpton, Pseudo-code of lammps timestep, https://docs.lammps.org/Developer_flow.html.
[22] F. Mackay, C. Denniston, J. Comput. Phys. 237 (2013) 289–298, https://doi.org/10.1016/j.jcp.2012.11.038, https://www.sciencedirect.com/science/article/pii/S0021999112007188.
[23] C.S. Peskin, Acta Numer. 11 (2002) 479–517, https://doi.org/10.1017/S0962492902000077.
[24] J.-P. Ryckaert, G. Ciccotti, H.J. Berendsen, J. Comput. Phys. 23 (3) (1977) 327–341, https://doi.org/10.1016/0021-9991(77)90098-5, https://www.sciencedirect.com/science/article/pii/0021999177900985.
[25] H.C. Andersen, J. Comput. Phys. 52 (1) (1983) 24–34, https://doi.org/10.1016/0021-9991(83)90014-1, https://www.sciencedirect.com/science/article/pii/0021999183900141.
[26] L. Verlet, Phys. Rev. 159 (1967) 98–103, https://doi.org/10.1103/PhysRev.159.98, https://link.aps.org/doi/10.1103/PhysRev.159.98.
[27] R. Keys, IEEE Trans. Acoust. Speech Signal Process. 29 (6) (1981) 1153–1160, https://doi.org/10.1109/TASSP.1981.1163711.
[28] R. Adhikari, K. Stratford, M.E. Cates, A.J. Wagner, Europhys. Lett. (EPL) 71 (3) (2005) 473–479, https://doi.org/10.1209/epl/i2004-10542-5.
[29] M.R. Parsa, A.J. Wagner, Phys. Rev. Lett. 124 (2020) 234501, https://doi.org/10.1103/PhysRevLett.124.234501, https://link.aps.org/doi/10.1103/PhysRevLett.124.234501.
[30] J. Ahrens, B. Geveci, C. Law, in: Visualization Handbook, Elsevier, Butterworth-Heinemann, 2005.
[31] H.C. Brinkman, Appl. Sci. Res. A 1 (1947) 27–34.
[32] H.C. Brinkman, Appl. Sci. Res. A 1 (1947) 81–86.
[33] P. Debye, A.M. Bueche, J. Chem. Phys. 16 (6) (1948) 573–579, https://doi.org/10.1063/1.1746948.
[34] K. Kremer, G.S. Grest, J. Chem. Phys. 92 (8) (1990) 5057–5086, https://doi.org/10.1063/1.458541.
[35] N. Afrasiabian, C. Denniston, Soft Matter 16 (39) (2020) 9101–9112.
[36] N. Afrasiabian, The Journey of a Single Polymer Chain to a Nanopore, Electronic Thesis and Dissertation Repository 6966, 2020.
[37] S.T.T. Ollila, C. Denniston, M. Karttunen, T. Ala-Nissila, Phys. Rev. Lett. 112 (2014) 118301, https://doi.org/10.1103/PhysRevLett.112.118301, https://link.aps.org/doi/10.1103/PhysRevLett.112.118301.
[38] F.E. Mackay, K. Pastor, M. Karttunen, C. Denniston, Soft Matter 10 (2014) 8724–8730, https://doi.org/10.1039/C4SM01812E.
[39] N.-Q. Nguyen, A.J.C. Ladd, Phys. Rev. E 66 (2002) 046708, https://doi.org/10.1103/PhysRevE.66.046708, https://link.aps.org/doi/10.1103/PhysRevE.66.046708.