Computer Physics Communications 224 (2018) 1–21

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

Feature article

mdFoam+: Advanced molecular dynamics in OpenFOAM☆

S.M. Longshaw [a,*], M.K. Borg [b], S.B. Ramisetti [b], J. Zhang [b], D.A. Lockerby [c], D.R. Emerson [a], J.M. Reese [b]

[a] Scientific Computing Department, The Science & Technology Facilities Council, Daresbury Laboratory, Warrington, Cheshire WA4 4AD, UK
[b] School of Engineering, University of Edinburgh, Edinburgh EH9 3FB, UK
[c] School of Engineering, University of Warwick, Coventry, CV4 7AL, UK

ARTICLE INFO

Article history:
Received 6 March 2017
Received in revised form 17 August 2017
Accepted 25 September 2017
Available online 23 October 2017

Keywords:
OpenFOAM
Molecular dynamics
MD
Computational fluid dynamics
CFD
Nano-scale
Micro-scale
Nanofluidics

ABSTRACT

This paper introduces mdFoam+, which is an MPI parallelised molecular dynamics (MD) solver implemented entirely within the OpenFOAM software framework. It is open-source and released under the same GNU General Public License (GPL) as OpenFOAM. The source code is released as a publicly open software repository that includes detailed documentation and tutorial cases. Since mdFoam+ is designed entirely within the OpenFOAM C++ object-oriented framework, it inherits a number of key features. The code is designed for extensibility and flexibility, so it is aimed first and foremost as an MD research tool, in which new models and test cases can be developed and tested rapidly. Implementing mdFoam+ in OpenFOAM also enables easier development of hybrid methods that couple MD with continuum-based solvers. Setting up MD cases follows the standard OpenFOAM format, as mdFoam+ also relies upon the OpenFOAM *dictionary*-based directory structure. This ensures that useful pre- and post-processing capabilities provided by OpenFOAM remain available even though the fully Lagrangian nature of an MD simulation is not typical of most OpenFOAM applications. Results show that mdFoam+ compares well to another well-known MD code (e.g. LAMMPS) in terms of benchmark problems, although it also has additional functionality that does not exist in other open-source MD codes.

Program summary

Program title: mdFoam+
Program Files doi: http://dx.doi.org/10.17632/7b4xkpx43b.1
Licensing provisions: GNU General Public License 3 (GPLv3)
Programming language: C++
Nature of problem: mdFoam+ has been developed to help investigate complex fluid flow problems at the micro and nano scales using molecular dynamics (MD). It provides an easily extended, parallelised, molecular dynamics environment.
Solution method: mdFoam+ implements a classical molecular dynamics solution using an explicit time-stepping regime and inter-molecular force-field types appropriate for studying fluid dynamics problems down to the nano-scale.
References: All appropriate methodological references are contained in the section entitled **References**.

© 2017 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

Molecular dynamics (MD) solves Newton's equations of motion for ensembles of atoms and molecules that interact with each other in accordance with well-validated intermolecular potentials (or force fields). An MD simulation is typically deterministic, so an explicit time-integration scheme is used to advance molecules in time and space, producing quasi-continuous molecular trajectories. The cornerstone of any MD simulation is to translate these trajectories into measurable material properties that provide fundamental insight into the microscopic and macroscopic behaviour of the system under investigation.

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

* Corresponding author.
E-mail address: stephen.longshaw@stfc.ac.uk (S.M. Longshaw).

https://doi.org/10.1016/j.cpc.2017.09.029
0010-4655/© 2017 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

Table 1
Comparison of mdFoam and mdFoam+ capabilities.

Feature	mdFoam	mdFoam+
Arbitrary 2D geometries	✓	✓
Arbitrary 3D geometries	✓	✓
MD initialisation and pre-processing tools	✗	✓
Wide range of standard pair potentials in extensible framework	✗	✓
Parallel processing	✓	✓
MD control models (i.e. density, temperature etc.) in extensible framework	✗	✓
Wide range of standard MD measurement models in extensible framework	✗	✓
Default set of cyclic boundary conditions implemented in extensible framework	✗	✓
Extensible framework for implementing new time-integration methods	✗	✓

The software discussed in this article (mdFoam+) fulfils a similar role to other MD software such as LAMMPS [1], DL_POLY [2] or NAMD [3]. However, it is markedly different to these others as it has been designed primarily with non-equilibrium flow problems in mind and all of its functionality is built entirely within the OpenFOAM [4] framework. This provides an inherent level of modularity, as well as potential interoperability by way of coupling with a large selection of other solver types that are also built within OpenFOAM. mdFoam+ is released under the same General Public License (GPL) as the version of OpenFOAM that it is built upon and is publicly available via a Git-based repository [5] that is maintained by the developing group.

OpenFOAM [4] (**Open**-source **F**ield **O**peration **A**nd **M**anipulation) is an open-source suite of libraries and applications designed primarily to solve computational fluid dynamics (CFD) problems. Since its initial release towards the end of the 1980s, the core design concept of OpenFOAM has remained the same, to provide an open and extensible C++ based software package containing a wide range of libraries, pre- and post-processing tools and solvers, as well as an underlying framework that can be used to build new applications.

OpenFOAM has incorporated the mdFoam MD solver since its 1.6 release. This software was developed by Macpherson, Borg and Reese [6–10] and was accepted into OpenFOAM in 2009. Since its initial release, the core MD functionality of mdFoam has remained largely unchanged and can be found in the current public release of OpenFOAM.

This article is about a branch of mdFoam, which has continued development since the original was accepted into OpenFOAM and will be referred to as mdFoam+ to distinguish it from mdFoam. mdFoam+ provides an enhanced set of MD capabilities when compared to mdFoam, as well as improvements in terms of both serial and parallel performance. Table 1 gives an overview of some of the functional differences between mdFoam and mdFoam+.

In basing the core functionality of mdFoam+ around standard OpenFOAM libraries, the application is able to make use of the meshing capability provided by applications like *blockMesh* or *snappyHexMesh*, parallelised Lagrangian/mesh-tracking algorithms and pre- and post-processing methods. As an example, molecules are initially defined by linking their coordinates within cells of the mesh of a problem domain. As the application is built on top of this meshing capability, this also means that typical OpenFOAM pre- and post-processing applications can be used, such as domain-decomposition and reconstruction of the mesh for parallel processing based on cells (and hence the molecules), as well as visualising results via ParaView [11].

Access to the mesh processing capabilities of OpenFOAM provides a powerful tool to define MD simulations with complex mesh structures, which are often a requirement for coupling with CFD applications. However, although the application has been developed with CFD in mind, the underlying MD methodologies are applicable to any suitable problem and so the code should not be considered solely for solving coupled-CFD or CFD-related cases. A key strength of mdFoam+ comes from its strict adherence to OpenFOAM coding practices, this means its design is almost entirely modular in the form of C++ classes.

2. Molecular dynamics with mdFoam+

There are many fluid dynamics problems where a typical continuum-fluid approach is invalid. Examples include flows with nano-scale confinement (e.g. within nano-tubes) [12–17], multi-phase problems with moving contact lines [18–20] and the rheology of complex fluids [21,22]. Molecular dynamics is a high-fidelity computational method that provides molecular-level detail in order to provide scientific insight into nano-scale and multi-scale phenomena. Essentially, MD can be described as a discrete and deterministic simulation of matter modelled explicitly by atoms and molecules that interact with each other through intermolecular force potentials, and advance in time and space to produce trajectories, from which important material property measurements can be extracted.

2.1. Background

Consider a system of N molecules, indexed by $i = 1, 2, \ldots N$. The ith molecule contains n_i atomic sites and a centre of mass $\vec{r}_i$. Position ($\vec{r}_i$), as well as the velocity $\vec{v}_i$, of each molecule are a function of time, and the motion of all N molecules is governed by Newton's equations of motion:

$$\frac{d}{dt}\vec{r}_i = \vec{v}_i, \tag{1}$$

$$m_i\frac{d}{dt}\vec{v}_i = \vec{f}_i, \tag{2}$$

where m_i is the ith molecule's mass (a summation of its component atomic masses) and $\vec{f}_i$ is the total force. At heart an MD calculation within mdFoam+ is the numerical implementation of Eqs. (1) and (2) in order to obtain the discretised trajectory $\vec{r}_i(t)$ as a function of time. Time is discretised using a fixed time-step Δt, which is chosen to obtain acceptable accuracy and to avoid numerical instabilities. Some examples of time-steps used in our simulations are: $\Delta t = 5$ fs for monatomic fluids (such as argon), and $\Delta t = 2$ fs for a 4-site water model.

2.2. Intermolecular potentials

The force term, $\vec{f}_i$, in Eq. (2) represents the net force on molecule i due to intermolecular forces. Selecting the correct force field for a particular problem is the most vital task in any MD simulation; slight changes in the force field parameters can give substantially different dynamics and overall macroscopic results. How to select the best potential for a problem is beyond the scope of this article as there are so many possible approaches.

The potential energy U of a system generally comprises bonded and non-bonded terms. Bonded force-fields are *intra*-molecular potentials (i.e. within an individual molecule), which are applied to flexible molecules consisting of a number of atomic-sites. Examples include: covalent bond stretching (two-site), harmonic angle bending (three-site), and torsion (four-site). Non-bonded terms are *inter*-molecular potentials (i.e. between molecules). Here, the focus is on rigid, non-bonded molecules; however an extension to intra-molecule force fields has recently been implemented in a separate branch of mdFoam+ [23] that is likely to be included in the main application in the future.

In general the potential energy of a non-bonded system with rigid molecules is given by:

$$U(\vec{r}) = \sum_{i}^{N} U_1(\vec{r}_i) + \sum_{i}^{N}\sum_{j>i}^{N} U_2(\vec{r}_i, \vec{r}_j) + \sum_{i}^{N}\sum_{j>i}^{N}\sum_{k>j>i}^{N} U_3(\vec{r}_i, \vec{r}_j, \vec{r}_k)\ldots \tag{3}$$

Terms on the right hand side (from left to right) represent one-body interactions (i.e. external energies such as body forces and electric fields), two-body pair-interactions (i.e. van der Waals and electrostatics), three-body interactions and so on. Three and higher body interactions are usually ignored for simple fluids. There are various pair-potentials in the literature; the most common is the 12–6 Lennard-Jones (LJ) potential:

$$U_{\mathrm{LJ}}(r_{ij}) = 4\epsilon\left[\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right], \tag{4}$$

where $r_{ij} = \left|\vec{r}_{ij}\right| = \left|\vec{r}_i - \vec{r}_j\right|$ is the separation distance between two molecules (i, j), σ is the hard-sphere diameter (which corresponds to where the potential energy is zero) and ϵ is the energy well-depth. For molecules that have charged sites, the Coulomb (C) pair potential is usually included:

$$U_{\mathrm{C}}(r_{ij}) = \frac{1}{4\pi\epsilon_0}\frac{q_i q_j}{r_{ij}}, \tag{5}$$

where ϵ_0 is the electric constant, while q_i and q_j are the charges of sites i and j on a pair of different molecules.

In order to determine the force on each molecule, the system energy, Eq. (3), is differentiated with respect to the molecular position $\vec{r}_i$:

$$\vec{f}_i = -\frac{dU}{d\vec{r}_i}. \tag{6}$$

mdFoam+ provides access to a number of different types of potentials by default but adding new pair-potentials is a quick and straightforward task given the modular, object-oriented design it adheres to.

2.3. Initialisation

An MD simulation using mdFoam+ begins with a set of pre-located molecules. This process is achieved using pre-processing tools and involves specifying the positions of the domain extremities, velocities of the molecules and type of molecules (i.e. their mass, local site positions and potential parameters). mdFoam+ comes with a number of pre-processing utilities that allow the user to prepare an MD case, such as basic initialisation (e.g. models that produce lattice structures, graphene sheets and nanotubes), mapping, adding (i.e. combining several groups of molecular instances together, including rotation and shifting) and deletion (i.e. removing molecules in a variety of ways).

2.4. Algorithmic overview

Like the majority of MD solvers, mdFoam+ implements an explicit time-stepping scheme in order to discretise Eqs. (1) and (2). All MD simulations in this paper (as well as all previous publications using mdFoam+) use the well-known Velocity-Verlet [24] scheme, which is illustrated in Fig. 1 and can be described algorithmically for one MD time-step, $t \rightarrow t + \Delta t$, as follows:

Step 1 Estimate the velocity at the mid-step for all N molecules in the system:

$$\vec{v}_i\left(t + \Delta t/2\right) = \vec{v}_i\left(t\right) + (1/2)\,\vec{a}_i\left(t\right)\Delta t \tag{7}$$

Step 2 Update the positions of all N molecules in the system using OpenFOAM's inbuilt particle tracking algorithm [25], as discussed in Section 2.6, which handles motion of molecules across faces of the mesh (and also deals with boundaries). In its mathematical form, the move step is given by:

$$\vec{r}_i\left(t + \Delta t\right) = \vec{r}_i\left(t\right) + \vec{v}_i\left(t + \Delta t/2\right)\Delta t \tag{8}$$

Step 3 Create images (or copies) of molecules that are external to the main cell of the underlying computational mesh and applicable to the cyclic and processor boundaries.

Step 4 Compute the inter-molecular forces $\vec{f}_i$ at $t + \Delta t$ for all N molecules using Eq. (6). The numerical implementation of this step is described in Section 2.7.

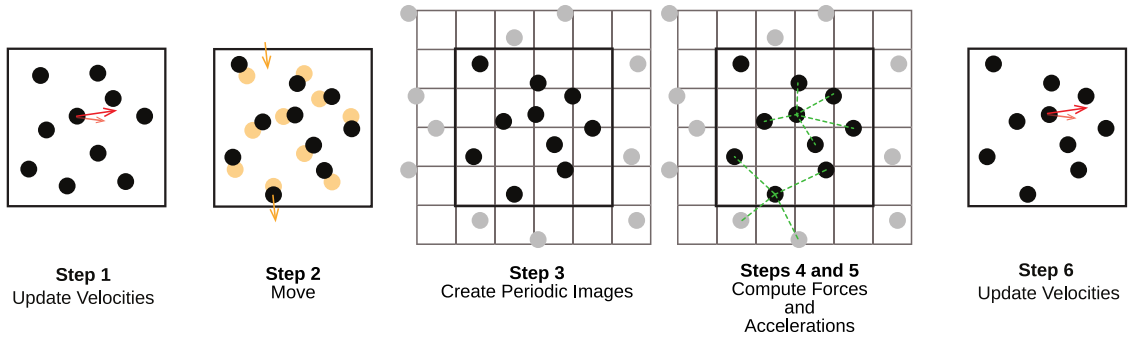


Fig. 1. Step-by-step illustration of the basic MD time-integration scheme used in mdFoam+, with periodic boundary conditions applied in all three directions.

Step 5 Compute a new acceleration for all N molecules in the system:

$$\vec{a}_i\left(t+\Delta t\right)=\vec{f}_i\left(t+\Delta t\right)/m_i \tag{9}$$

Step 6 Update the velocity at time $t+\Delta t$ for all N molecules:

$$\vec{v}_i\left(t+\Delta t\right)=\vec{v}_i\left(t+\Delta t/2\right)+(1/2)\,\vec{a}_i\left(t+\Delta t\right)\Delta t \tag{10}$$

Step 7 Return to **Step 1**, but with $t=t+\Delta t$, unless simulation end-time has been reached, in which case stop.

2.5. Rigid polyatomic molecules

The algorithm in Section 2.4 can be extended to deal with three-dimensional polyatomic molecules. mdFoam+ handles rigid multi-site molecules, which means the bond distances and angles between atomic sites on the molecule remain fixed. The equations of motion and the algorithm for polyatomic molecules need to be modified to allow for rotation.

In **Step 1** and **Step 6** above, the rotational velocity of the molecule is updated in a way similar to the translational velocity:

$$\vec{\pi}_i\left(t+\Delta t/2\right)=\vec{\pi}_i\left(t\right)+(1/2)\,\vec{\tau}_i\left(t\right)\Delta t, \tag{11}$$

where $\vec{\pi}_i$ and $\vec{\tau}_i$ are the angular momentum and torque of molecule i, respectively.

In **Step 4** the total force and total torque are calculated as sums over all atoms in a molecule. Finally, after the move step in **Step 2**, a splitting method is used to update rotational movement of the system of rigid bodies (see Appendix A in [26]).

2.6. Particle tracking

The particle tracking algorithm [25] underpinning mdFoam+ is based on existing OpenFOAM functionality for the discrete motion of particles within a mesh, with the two important objectives of knowing when particles move between cells in the mesh and dealing with particles as they hit boundaries (e.g. solid walls, inflow, outflow and periodic). Most other MD codes are different from mdFoam+ in how they track the motion of molecules, because MD domains are traditionally cuboid in shape and provide the simplest case of periodic boundaries. In the move step of standard MD codes (Step 2, Section 2.4), the position vector $\Delta\vec{r}_i=\vec{v}_i\Delta t$ is added to the molecules using Eq. (8), which is followed by a correction step that copies the molecules to the predefined opposing boundary if they are found to lie outside the boundary that they were near to. mdFoam+, however, is based on the premise of a general CFD mesh with an arbitrary number of boundaries, not necessarily of the simple periodic type.

To facilitate this more complex boundary type, a more flexible way of tracking molecules within a mesh is required, as illustrated in Fig. 2, showing how a particle moves between cells within a single time-step. Rather than applying $\Delta\vec{r}_i$ fully to a molecule, the number of cell faces that potentially cross the mesh patch that is associated with the molecule are first calculated. The molecule is then tracked to each face (and cell index) sequentially. After each intersection, the face's properties are checked to make a decision on the next part of its motion. If the face is internal, the molecule simply proceeds to its next position. However, if the face is part of a boundary, an action will be applied (during its move step). For example, if the boundary is a specular wall, the molecular trajectory is modified on contact with the boundary, and continues the rest of its movement within the domain. This has been found to be a robust algorithm that ensures molecules never stray outside the domain, it also enables mdFoam+ to deal with various complex boundaries that would not be possible otherwise.

2.7. Force calculation and performance considerations

MD simulations are notoriously computationally intensive. In theory, every molecule needs to interact with all other molecules in the system to obtain the net force (see Eqs (3) and (6)), making this an $\mathcal{O}(N^2)$ problem. When this is considered alongside the need to take relatively small time-steps (to ensure stability and accuracy) this means the computationally intensive force computation step can occur many millions of times within a simulation. There are, therefore, a number of practical modelling considerations that are usually adopted when running an MD simulation in order to reduce the amount of time it can take to compute the forces, and reduce the cost to $\mathcal{O}(N\ln(N))$. mdFoam+ follows a number of these best practices:

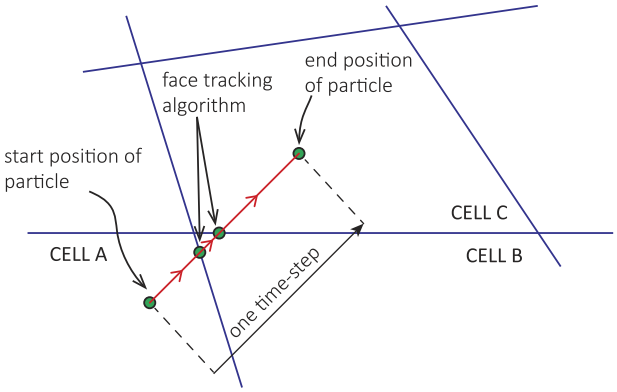


Fig. 2. Illustration of the particle tracking algorithm [25] within an unstructured mesh.

1. **Potential cut-off**: When using a potential function that has an attractive/repulsive tail that tends to zero at large molecular separation (such as the LJ potential), it is good practice to define a calculation 'cut-off' radius, r_{cut}. This ensures that pair-wise calculations are made only with those molecules that are within a specified surrounding region of influence. In order to facilitate this, an MD solver needs to implement a neighbour-list or cell-list algorithm [27]. mdFoam+ implements the latter, using the underlying OpenFOAM mesh structure in order to perform cell–cell searches [9]. This means that all molecules in a given cell only need to interact with molecules also in that cell and in the neighbouring (or touching) adjacent cells. If the cell sizes are chosen to be equal to r_{cut} this means each cell requires information about molecules from all touching cells only. Otherwise, a search algorithm for cells within r_{cut} is adopted [9].
2. **Utilise pair-wise symmetry**: Pair-wise interactions are typically symmetric: a force calculated for one molecule in a pair is experienced by the other molecule in equal magnitude but opposite direction. This allows calculations in a system of molecules within mdFoam+ to be reduced by half.
3. **Calculate in parallel**: Currently, mdFoam+ allows parallel processing using the MPI capability inherent within OpenFOAM and can be run on both distributed and shared memory computer architectures using domain-decomposition techniques. Pre-processing tools within OpenFOAM allow the initial meshed domain to be decomposed using a number of techniques that aim to minimise MPI communications. Several limitations are notable here, which are mainly inherited from OpenFOAM itself and include the fact that, for each MPI rank created, a separate set of output files is written to disk (this can sometimes cause problems with parallel file systems on large high performance computing (HPC) machines). In terms of parallel scalability, it has been shown that this can be problem specific and can be as low as a few hundred MPI ranks in some cases. Parallel performance is discussed in further detail in Section 2.10 but is considered an important future development area for mdFoam+, with options such as selective GPU optimisation and mixed shared/distributed parallelism being considered in the form of hybrid OpenMP/MPI acceleration.

2.8. Software implementation

Problems solved by software like mdFoam+ tend to fall into the category of *N-body* and are often described as embarrassingly parallel in nature. However, in the case of a complex MD problem, the best software implementation is not always clear. The primary reason for this is that while many N-body problems have very little data that needs to be stored per discrete body in the system (e.g. acceleration, velocity, position, etc.), the amount of data needed per atom or molecule for an MD simulation can be significantly higher. An example of this is the need to store data regarding orientation of a molecule comprising multiple atoms. More importantly however, it is often necessary for molecules in a system to store different sets of properties to others, due to MD simulations often containing different types (or species) of molecule. In other words, MD solvers need to be able to cope with large non-homogeneous sets of discrete bodies.

To add further complexity, it is a requirement for most MD scenarios that new molecules (or atoms) can be added or removed from a simulation during the run-time of a problem, meaning the data storage requirements of the software can vary significantly during the evolution of a problem. The end result is that the most flexible and easily considered implementation uses a linked-list type of data structure to create an array-of-structures (AoS). This provides an easily expandable (or contractable) data structure and allows each structure stored to be customised for each molecule in the system. However, in the case of an MD simulation involving pair-wise force calculations, this solution also provides problems in terms of CPU cache-coherency, as data contiguity is not guaranteed in a linked-list of pointers to instantiated objects. In real terms, this means run-time latency associated with memory access is notably higher than if a structure of arrays (SoA) were used.

In mdFoam+ the majority of the data stored in memory is encapsulated in a single *MoleculeCloud* class. This is an extension of a base OpenFOAM Lagrangian class known as the *Cloud*. When instantiated, the molecule cloud class creates a doubly-linked list, which stores pointers to instantiations of a mutable class known as the *Molecule* class, which in itself is an extension of the base OpenFOAM *Particle* class. Although this design provides flexibility and an easily understood data structure, it also does not guarantee contiguous memory access, which is the primary source of the software's cache misses.

In the case of mdFoam+, it is not entirely clear whether it is better to sacrifice run-time performance to enable easier programming by using an AoS design, or whether design complexity should be increased in order to reduce run-time by converting this to SoA. This could be a key area for future development, but hybrid approaches used in other N-body application areas [28] may provide the most suitable balance between usability and computational performance. However, it is important to note that this may mean deviating from building upon core OpenFOAM libraries such as the *Cloud* class.

Table 2
Example reference values for a simulation involving water molecules. A property x in S.I. units is converted into its non-dimensional reduced form x^* using its reference value x_r. The first column is the dimension, the second is the scaled variable used in the MD simulation, and the third is the reference value. The Boltzmann constant, k_b, is used to calculate the derived value *Temperature*.

Fundamental values		
Length	$l^* = l/l_r$	$l_r = 3.154 \times 10^{-10}$ m
Time	$t^* = t/t_r$	$t_r = 1.66 \times 10^{-12}$ s
Mass	$m^* = m/m_r$	$m_r = 2.987 \times 10^{-26}$ kg
Charge	$q^* = q/q_r$	$q_r = 1.602176487 \times 10^{-19}$ C
Derived values		
Energy	$\epsilon^* = \epsilon/\epsilon_r$	$\epsilon_r = \left(m_r l_r^2/t_r^2\right) = 1.08 \times 10^{-21}$ J
Force	$f^* = f/f_r$	$f_r = (\epsilon_r/l_r) = 3.42 \times 10^{-12}$ N
Velocity	$u^* = u/u_r$	$u_r = \left(\sqrt{\epsilon_r/m_r}\right) = 190$ m s^{-1}
Mass density	$\rho^* = \rho/\rho_r$	$\rho_r = \left(m_r/l_r^3\right) = 952.03$ kg/m^3
Temperature	$T^* = T/T_r$	$T_r = (\epsilon_r/k_b) = 78.1$ K
Pressure	$p^* = p/p_r$	$p_r = \left(\epsilon_r/l_r^3\right) = 34.368 \times 10^6$ N m^{-2}

2.9. Reduced units

As the numerical values calculated during an MD simulation can be very small, and may fall out of safe storage limits even for double precision floating point accuracy, it is common to scale all properties in the system to within a suitable non-dimensional range. In MD these are typically referred to as 'reduced units' [29]. mdFoam+ follows this approach, allowing a set of fundamental reference variables to be defined as part of a case. An example of this might be if the mass of one water molecule is $m_i = 2.9 \times 10^{-26}$ kg then a reference mass m_r can be chosen (e.g. $m_r = 3 \times 10^{-26}$ kg) such that $m_i^* = m_i/m_r$ gives a scaled value which is close to 1 for calculation purposes.

Four fundamental reference variables for performing scaling are defined within mdFoam+; length l_r, mass m_r, time t_r and charge q_r; all other reference variables can be derived from a combination of these. Reference values will vary depending on the problem; in Table 2 an example is shown of reference values used for MD simulations of water.

2.10. Parallel processing

mdFoam+ provides an MPI based domain-decomposition method for performing parallel MD simulations. Generally the concept is that the mesh, which represents the entire domain to be simulated, is first split up, or decomposed, into as many sub-parts as there are processor cores available. This process of domain-decomposition is well known and fundamental to OpenFOAM parallelism.

This functionality is built upon pre-existing OpenFOAM capability and, while it can provide notably improved performance when compared to running the same problem in serial, the scalability of the code is problem-specific and sometimes limited to the order of hundreds of MPI ranks. It should be noted this is not unique to mdFoam+ as the Lagrangian nature of an MD simulation means that any implementation using static domain decomposition is likely to see similar limitations. This is primarily due to the way MPI parallelism is implemented within OpenFOAM, combined with the need for relatively high numbers of MPI messages at domain borders due to the properties of a Lagrangian simulation. Improving parallelism within mdFoam+ is an important future task as HPC heads towards a fine-grained model. Currently, however, the parallel performance of mdFoam+ is sufficient to make it a useful tool for simulating MD problems involving a few million molecules over reasonable time periods, as shown later in this article.

A scalability study is provided for reference; this simple case is a fully periodic cube comprising water molecules in equilibrium (of side length L), with a 1 nm cut-off distance. Four identical cases, of increasing cube length (L = 22.1 nm, 31.6 nm, 47.3 nm and 63.1 nm), are considered and simulated using increasing numbers of MPI ranks. All calculations were performed on ARCHER, the UK's national supercomputing service. This system is a Cray XC30, giving access to nodes with two Intel Xeon E5—2697v2 CPUs (12 cores per CPU, 24 per node) and 64GB of RAM. For these results, mdFoam+ was compiled using GNU GCC 4.9 and the system default Cray MPI library, based on MPICH. Results can be seen in Fig. 3, where the computational time per MD time-step is plotted against the number of MPI ranks used. The computational time is measured using in-built OpenFOAM timing functionality and is the average time taken per step.

2.11. Extensible design

An important feature of mdFoam+ is its inherent ability to be extended in terms of MD functionality. This capability was one of the key reasons OpenFOAM was originally selected as the base development framework for the software. mdFoam+ is designed entirely in object-oriented C++ and in accordance with the OpenFOAM coding guidelines. The majority of its classes are therefore derived from existing OpenFOAM classes, primarily from the Lagrangian portion of the code. Consequently, the typical design for all of the software's modular elements is that a base class is provided (which often derives from a base OpenFOAM class) and a specialised class is then created to provide specific functionality. Typically, the amount of code provided in these is minimal, therefore each derived class often contains less than a hundred lines of unique code.

Adding extra functionality to mdFoam+ requires a copy of an appropriate specialised class to be created, given a new name and its functionality updated. The new class is then added to the compilation list, as with any OpenFOAM addition. Once mdFoam+ is re-compiled, the functionality provided by the new class can then be accessed by updating the appropriate line in the case dictionary files. While other MD codes also strive to provide this level of extensibility, the strict adherence of OpenFOAM (and therefore mdFoam+) to a fully object-oriented programming model means that any new addition will perform as well as the code it is built upon. It also ensures a level of

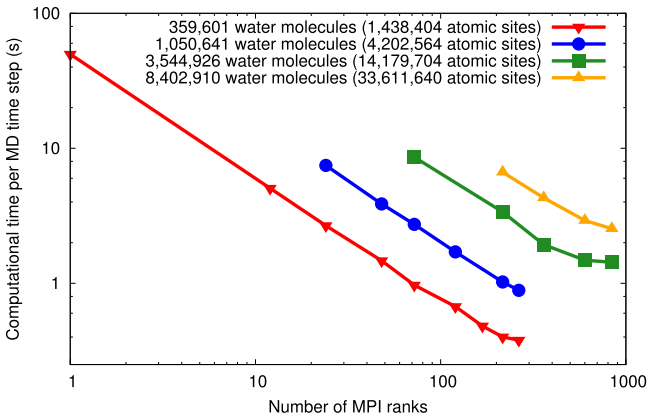


Fig. 3. Parallel performance of four example cases of a periodic cube of water molecules in equilibrium showing the average computational time per MD time-step for two simulations. The number of water molecules (and atomic sites) is indicated in the key of the graph and correspond to the cube length, where $L = 22.1$ nm refers to the case with 359,601 molecules, $L = 31.6$ nm the case with 1050641 molecules etc. Each node of the HPC resource used contained 24 processor cores. Results are shown on a log–log scale.

standardisation in any new addition and simplifies the process of adding new functionality. An example of where this ability is useful is when there is a need to add a new inter-molecular potential. Doing so within mdFoam+ can be achieved quickly and often through the addition of a negligible amount of new code.

The extensibility of mdFoam+ is also notable in the way cases are defined. As the standard OpenFOAM case dictionary model is followed, cases are specified as a series of input files known as dictionaries. These are plain text files which adopt a standardised format in which new variables can be defined by typing new text and a value associated with the name. Dictionary parsing is done within mdFoam+ using the basic OpenFOAM libraries, therefore the majority of variables used are free-form in nature (i.e. should new functionality be added that requires a new input parameter, this can be easily added to the corresponding dictionary file).

2.12. Coupling mdFoam+ with computational fluid dynamics

One of the important reasons why mdFoam+ has been developed within OpenFOAM is to enable the coupling of molecular dynamics simulations with standard CFD simulations. In these hybrid simulations, the goal is to introduce molecular resolution into CFD simulations where the continuum fluid equations are invalid or questionable, such as water flows inside, or around, nanotubes. A series of hybrid methods have been proposed that couple mdFoam+ with continuum equations in OpenFOAM [10,22,30–37], which have shown that the computational savings over a full MD simulation can be significant when applied to appropriate problems.

3. Downloading and installing mdFoam+

This article does not aim to provide detailed instructions on how to download and build mdFoam+ , as this process does not differ from building a standard implementation of OpenFOAM. The software can be downloaded through the associated journal library entry, or directly from a publicly available Git repository [5].

Once downloaded, building the software for a chosen platform is best achieved by following the detailed instructions included in the *doc/MicroNanoFlows* folder within the main repository directory, referring back to platform-specific instructions for the current build of OpenFOAM that mdFoam+ is released alongside. At the time of writing mdFoam+ is released as part of an OpenFOAM 2.4.0 repository. This provides compilation capability for a number of well-known platforms, however it primarily targets POSIX environments such as Linux and Unix, along with a number of hardware platforms such as x86 and PowerPC. It is possible to compile OpenFOAM 2.4.0 under alternative environments, such as Apple's MacOS or Microsoft's Windows, however it is beyond the scope of this document to detail how this can be achieved; the reader is directed to the OpenFOAM documentation.

Compilation of different parts of the OpenFOAM suite, and therefore mdFoam+, require a number of third party libraries. These are not bundled as part of the software download but are freely available for download in publicly available locations. Further details are provided in the *doc/MicroNanoFlows* folder.

4. Using mdFoam+

For those familiar with using an application in the OpenFOAM suite, usage of mdFoam+ will be relatively straightforward. In order to provide an illustrative example, the following section describes the process of defining, initialising and running (both in serial and parallel), and then post-processing a simple equilibrium case, consisting of a periodic cubic domain filled with water at standard atmospheric pressure and temperature. This case is a smaller version of the problem in Section 2.10 used for the parallel scalability study, as well as in Section 5 for cross MD-platform validation.

This MD simulation is a canonical *NVT* ensemble, which means that it has a constant number of molecules N, a constant volume V, and a constant temperature T. The case is illustrated in Fig. 4 for a domain consisting of a periodic cubic box of side length $L = 5.05$ nm, filled with $N = 4298$ water molecules in order to give a mass density of $\rho_M = 1000$ kg/m^3. A Berendsen thermostat [38] is applied to the entire domain in order to set the temperature $T = 300$ K.

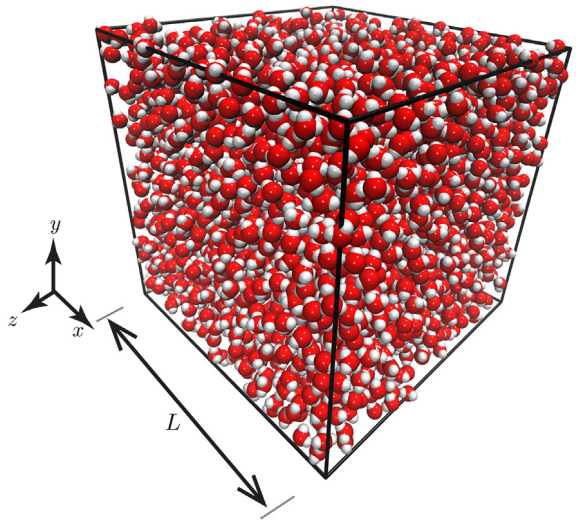


Fig. 4. MD example case: the simulation of water in a cube of side length L, with periodic boundary conditions applied in all directions.

4.1. Case definition

Using mdFoam+ begins by creating a new MD case. The case structure for mdFoam+ is similar to any OpenFOAM application, and is normally defined by first creating a new folder with an appropriate name (referred to henceforth as *[case]*), under which there are two more folders named *system* and *constant*. The former of these contains the majority of the OpenFOAM dictionary files that control most of the running parameters for the MD case (i.e. time-step control, case initialisation parameters, potentials etc.), while the latter contains details of the physical domain that is used to create and populate the underlying OpenFOAM mesh with molecules. While there are some files within this structure that are specific to mdFoam+, their nomenclature is designed to be self-descriptive. Alongside the example case described here, tutorial cases are also supplied as part of the software distribution and can be found within the *tutorials/discreteMethods/mdFoamPlus* folder.

4.1.1. Mesh creation

Simulations using mdFoam+ are defined in a three-dimensional Cartesian co-ordinate system and are most commonly processed using the blockMesh application, found within the OpenFOAM suite. More advanced meshing tools are available within OpenFOAM, but it is typically sufficient to use blockMesh. For more complex generation of meshes for mdFoam+, the reader is referred to [6–10].

The mesh itself is used by an mdFoam+ simulation in a number of ways: (a) to initially place molecules; (b) to provide a cell-list algorithm for neighbour lookup; (c) to decompose a domain for parallel execution; (d) to resolve macroscopic field properties from the underlying MD simulation; and (e) to assist coupling to other Eulerian (mesh-based) methods.

The file which controls the operation of the blockMesh application is *blockMeshDict*. This can be found at *[case]/constant/polyMesh/blockMeshDict*. An example of this dictionary is:

```
1  /*--------------------------------------------------------------------------*\
2  | =========                 |                                                |
3  | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox          |
4  |  \\    /   O peration     | Version:  2.4.0-MNF                            |
5  |   \\  /    A nd           | Web:      http://www.openfoam.org              |
6  |    \\/     M anipulation  |                                                |
7  \*--------------------------------------------------------------------------*/
8
9  FoamFile
10  {
11      version         2.0;
12      format          ascii;
13
14      root            "";
15      case            "";
16      instance        "";
17      local           "";
18
19      class           dictionary;
20      object          blockMeshDict;
21  }
22
```

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 1;

vertices
(
    (0 0 0)
    (16 0 0)
    (16 16 0)
    (0 16 0)
    (0 0 16)
    (16 0 16)
    (16 16 16)
    (0 16 16)
);

blocks
(
    hex (0 1 2 3 4 5 6 7) mdZone (5 5 5) simpleGrading (1 1 1)
);

boundary
(
    periodicX_half0
    {
        type cyclic;
        faces ((1 2 6 5));
        neighbourPatch periodicX_half1;
    }

    periodicX_half1
    {
        type cyclic;
        faces ((0 4 7 3));
        neighbourPatch periodicX_half0;
    }

    periodicY_half0
    {
        type cyclic;
        faces ((2 3 7 6));
        neighbourPatch periodicY_half1;
    }

    periodicY_half1
    {
        type cyclic;
        faces ((0 1 5 4));
        neighbourPatch periodicY_half0;
    }

    periodicZ_half0
    {
        type cyclic;
        faces ((4 5 6 7));
        neighbourPatch periodicZ_half1;
    }

    periodicZ_half1
    {
        type cyclic;
        faces ((0 3 2 1));
        neighbourPatch periodicZ_half0;
    }
);

```

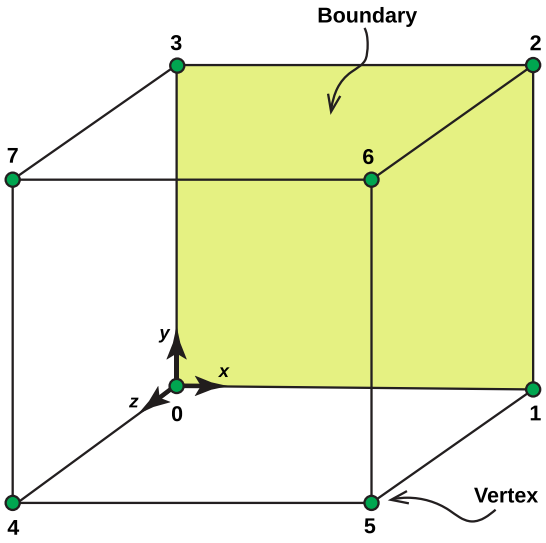


Fig. 5. Block structure of the cubic domain for the example case.

```
89  mergePatchPairs
90  ( );
92
93  // ************************************************************************* //
```

Referring to the example above, lines 1–24 are a standard header within OpenFOAM dictionary files, these are included in this first example for completeness but are omitted from all further examples. Lines 27–37 define 8 vertices, which define one block on the mesh. In this simple case only one block is needed as the domain is a cube, therefore 8 vertices only are required. These vertices are numbered 0 to 7 and their physical locations are depicted in Fig. 5; the value *convertToMeters* seen on line 25 allows the domain to be scaled, but in this case no scaling is applied as all values are already in reduced units.

Blocks are defined within the *blocks* region; in this case there is only one block, which can be seen on lines 39–42 where the vertex order is first defined followed by the keyword *mdZone* and then *simpleGrading*. The entry *mdZone* is the zone-name given to the list of cells that are to be created in the x, y and z directions, which is written as $\left(N_{c,x}, N_{c,y}, N_{c,z}\right)$; the total number of cells that will be created is $N_{c,x} \times N_{c,y} \times N_{c,z}$. The cell size is therefore given by $\Delta L_i = L_i/N_{c,i}$, where $i = x, y, z$; this is an important value when defining an MD simulation that uses the cell-list algorithm, because ideally the cell size should be as close as possible to the largest cut-off distance of all potentials used (i.e. r_{cut}). The *simpleGrading* parameter allows cells to be scaled in size in one or more directions. However, this rarely needs to be changed from the default of (1, 1, 1) as MD simulations typically use uniform cell distributions.

The boundary entries shown on lines 44–87 indicate the *patches* that form the domain boundaries. In this example of a periodic cube, there are 6 boundaries and 6 cyclic patches. Each patch contains the same three parameters. **type**, which defines the type of the patch, can be any viable OpenFOAM input, however it is most common to use either *patch* (a generic boundary that links to a user-defined boundary model) or *cyclic*. The **faces** of a patch are defined by the vertices. Since these are cyclic patches, they need to be coupled to another **neighbourPatch**. Once the boundaries have been defined geometrically, it is also necessary to link the boundaries to the correct MD functionality; this is done within the file *[case]/system/boundariesDict*, which for this example looks like:

```
28  polyPatchBoundaries
29  ( );
30
31  polyCyclicBoundaries
32  (
33      boundary
34      {
35          boundaryModel  polyStandardCyclic;
36          patchName      periodicX_half0;
37      }
38      boundary
39      {
40          boundaryModel  polyStandardCyclic;
41          patchName      periodicX_half1;
42      }
43      boundary
44      {
45          boundaryModel  polyStandardCyclic;
46          patchName      periodicY_half0;
```

```
    }
    boundary
    {
        boundaryModel  polyStandardCyclic;
        patchName      periodicY_half1;
    }
    boundary
    {
        boundaryModel  polyStandardCyclic;
        patchName      periodicZ_half0;
    }
    boundary
    {
        boundaryModel  polyStandardCyclic;
        patchName      periodicZ_half1;
    }
);

polyGeneralBoundaries
( );
```

Each boundary has a **boundaryModel** that specifies the type of model mdFoam+ will use. In this particular case, the *polyStandardCyclic* has been selected. Other models exist within mdFoam+, e.g. to modify the cyclic boundaries [39], or new ones can be added by extending its capabilities. The **patchName** links the names of the patches within the *blockMeshDict* to the MD boundary models chosen. Once these files are defined, the mesh and boundary data can be generated by first running the OpenFOAM application ***blockMesh*** in the base of the *[case]* folder, followed by running the mdFoam+ cyclic boundary condition tool ***createCyclicBoundaries***.

4.1.2. Defining unit scaling

As described in Section 2.9, the majority of MD simulations run with mdFoam+ use reduced units to ensure calculated values are within sensible bounds for floating point accuracy. The four fundamental reference values used for scaling are defined in the file *[case]/system/reducedUnitsDict*. An example of this file is:

```
refLength    3.154e-10;
refTime      1.66e-12;
refMass      2.987e-26;
refCharge    1.602176487e-19;
```

Full descriptions of these values, as well as how they relate to other derived scaling values, are in Section 2.9.

4.1.3. Molecular properties

Constant molecular properties can be set for each species of molecule involved in a simulation. These are defined in the dictionary file located at *[case]/constant/moleculeProperties*, an example of which is:

```
idList (water);

moleculeProperties
(
    water
    {
        cloudType               polyMoleculeCloud;
        siteIds                 (H H O M);
        pairPotentials          (0 0 1 0);
        siteReferencePositions
        (
            (7.56950327263661e-11 5.85882276618295e-11 0)
            (-7.56950327263661e-11 5.85882276618295e-11 0)
            (0 0 0)
            (0 1.546e-11 0)
        );
        siteMasses
        (
            1.67353255e-27
            1.67353255e-27
            2.6560176e-26
            0
        );
        siteCharges
```

```
43          (
44              8.91450997367e-20
45              8.91450997367e-20
46              0
47              -1.782901995e-19
48          );
49      }
50  );
```

Line 19 shows the definition of a new species of molecule in the form of an **idList**. Each species of molecule should be given a unique name within this list (in this case there is only *water*). The index of each unique name within this list is used within mdFoam+ as a unique identifying number in order to keep track of different molecule types; in this case *water* is given an ID of 0. The physical properties of all species of molecule are defined between lines 21–50. It is important to note that the properties of each species must be defined in the order they are listed in the idList.

4.1.4. Potentials

One of the most important choices in any MD simulation is the selection of the potential type used to calculate inter-molecular forces. Potential calculation within mdFoam+ is discussed in Section 2.2, but for the exemplar case presented here using the TIP4P/2005 water model, short-range Lennard-Jones and electrostatic pair-wise potentials are used. Definitions of these are made in the dictionary file *[case]/system/potentialDict*, for example:

```
51  pairs
52  (
53      O-O
54      {
55          pairPotential   lennardJones;
56          rCut            1.0e-9;
57          rMin            1e-15;
58          dr              1e-13;
59          lennardJonesCoeffs
60          {
61              sigma       3.1589e-10;
62              epsilon     1.286751503e-21;
63          }
64          energyScalingFunction   noScaling;
65          writeTables     yes;
66      }
67  );
68
69  electrostatic
70  {
71      pairPotential   coulomb;
72      rCut            1e-9;
73      rMin            1e-15;
74      dr              2e-12;
75
76      energyScalingFunction   shiftedForce;
77      writeTables     yes;
78  }
```

Here, lines 53–66 are defining the O–O (Oxygen to Oxygen) interactions for all species (in this case there is only *water*) to use Lennard-Jones, while lines 69–78 define the use of short-range electrostatic interactions between charged sites. If the capability of mdFoam+ is extended (as described in Section 2.11) to include a new type of potential, then it would be included in an MD simulation by updating the reference in this file. It should be noted that values defined here will be automatically scaled according to the values set in the reduced units dictionary file (see Section 4.1.2).

4.1.5. Time control

The correct choice of time-step is fundamental to achieving a stable and efficient MD simulation. A time-step that is too small will make the simulation too computationally expensive, while a time-step that is too large will significantly reduce the accuracy of the results (so that drifts in energy can occur) or create instabilities that lead to simulation blow-up.

All controls associated with time-stepping, simulation length and file output are held in a single dictionary located at *[case]/system/controlDict*, which is standard in OpenFOAM. An example of this file for the exemplar case is as follows:

```
17  application     mdFoamPlus;
18
19  startFrom       startTime; //Start simulation at startTime (available: latestTime)
20
```

```
21  startTime       0.0; //Start from 0 time (can be changed to any time)
22
23  stopAt          endTime; //End at endTime (other options available)
24
25  endTime         12.0; //Stops simulation when time has reached this value
26
27  deltaT          0.0012; //Size of MD time-step
28
29  writeControl    runTime;
30
31  writeInterval   0.012; //File write interval
32
33  purgeWrite      3; //Only last 3 times are stored (0: no deletion)
34
35  writeFormat     ascii; //Format can be ASCII or Binary
36
37  writePrecision  12; //Floating point precision to write
38
39  writeCompression uncompressed; //Files can be compressed upon write
40
41  timeFormat      general;
42
43  timePrecision   8; //Precision of values written
44
45  runTimeModifiable yes;
46
47  adjustTimeStep  no;
```

While the majority of this file is self-descriptive, there are a few important points to note. The first is that all units are assumed to be scaled (or reduced) units of time, i.e. to convert the time-step value of $\Delta t^* = 0.0012$ to S.I. units requires multiplying by the reference value of time *refTime* defined in Section 4.1.2 (in this case making $\Delta t = 2$ fs).

The majority of time-control functionality offered by mdFoam+ is built upon that provided by OpenFOAM, therefore capabilities like the ability to restart a simulation are inherent. This functionality can be an important tool when considering multiple scenarios for an MD case, as it allows an initial period to be simulated once (e.g. equilibration) and then multiple alternative scenarios to be restarted from the same initial state. The nature of this functionality is best described in the OpenFOAM user-guide; essentially, it involves setting the value of *startTime* equal to an existing time folder with a valid set of MD data contained within. For a case set to start at $t = 0$, the initial values are stored in a folder located at *[case]/0* and the values are created using a separate pre-processing tool described in Section 4.2. The total number of time-steps taken is controlled by the value *endTime*.

Another important consideration is how often to write data to disk. While it is possible for data to be written at every time-step (by setting *writeInterval* equal to *deltaT*), this should be avoided whenever possible as it will introduce significant processing overhead, as for each time-step written a new output folder has to be created and a number of individual files written out to disk within the new folder. This can significantly impact the final size of the case directory and the computational cost of the simulation, especially in a typical MD scenario where the simulation may be run for millions of time-steps. Often the important values obtained from the simulation are time-averages and not the exact molecular details at one point in time. It is therefore important to select appropriate values for how often data is output and also how often historical data is deleted. This is controlled by the *purgeWrite* value, which in this case will ensure only the last three time-steps written remain on disk. For performance reasons, and should disk space not be an issue, setting the value of *purgeWrite* equal to 0 will ensure all historical data remains and the file-system overhead of deleting directories will not be incurred.

A useful piece of functionality provided by mdFoam+ is the ability to modify the running state of an active MD simulation by editing the case's *controlDict* file so that it ends cleanly without the process having to be killed by the user. This can be most useful when running a long case on an HPC resource where it is perhaps becoming obvious that a simulation has been initially set to run for longer than is actually needed (i.e. perhaps it has reached steady-state more quickly than anticipated). In this case the value of *stopAt* can be edited to equal either *writeNow* which will make it stop after completing the current time-step being calculated, or it can be set to *nextWrite* which will make it stop following the next write-interval.

4.1.6. Molecular dynamics control

An important part of any MD simulation is control over the physical nature of the simulation beyond the basic definitions of the species of molecule and how they interact in terms of potentials. This control acts on different physical aspects of a simulation, such as the temperature using a thermostat, or inserting/deleting molecules in order to control density. mdFoam+ provides this functionality in the form of a control architecture [10] that is in-line with its over-arching goal to be a flexible and extensible MD framework.

Controllers are encapsulated in a set of C++ classes, so that new ones can be easily added; their use is specified in a dictionary file located at *[case]/system/controllersDict*. In the example case presented here there is only a need for one controller, a Berendsen thermostat [38], in order to control the temperature. The file for this is:

```
18  polyStateControllers
19  (
20      // thermostat
21
```

```
    controller
    {
       zoneName             mdZone;

        stateControllerModel polyTemperatureBerendsenBinsNew;

        polyTemperatureBerendsenBinsNewProperties
        {
            temperature             3.816;
            tauT                    0.04;
            molIds                  (water);
            peculiar                yes;

            binModel                uniformBins;

            uniformBinsProperties
            {
                startPoint              (0 0 0);
                endPoint                (16 0 0);
                nBins                   4;
                area                    1; //Dummy variable
            }
        }
    }
);

polyFluxControllers
( );
```

In this example the Berendsen thermostat is applied by dividing the domain into four regions (or bins) and temperature is controlled through molecular velocity re-scaling, resulting in the kinetic energy of the system being adjusted according to $\frac{1}{2}m_i v_i^2$. For this rescaling the modification factor is:

$$\chi = \left[1 + \frac{\Delta t_m}{\tau_T}\left(\frac{T_t}{\langle T\rangle} - 1\right)\right]^{1/2}, \tag{12}$$

where τ_T is a time-relaxation constant that defines the coupling strength between the measured system temperature $\langle T\rangle$ and the target temperature T_t. In this example, a target temperature of $T_t^* = 3.816$ in normalised units ($T_t = 300$ K in S.I. units) has been chosen and the time-relaxation constant is set to be a few times larger than the MD time-step $\Delta t^* = 0.0012$. The Boolean variable named *Peculiar* refers to whether the internal velocity for a molecule is calculated according to the real atomic velocities or its peculiar velocity.

4.1.7. Repeatability

It is usual for MD simulations to use pseudo-random numbers to create initial molecular formations, as well as to introduce random components to calculations during their evolution. In order to ensure simulations that rely on random number streams are repeatable (or deterministic), it is important that the pseudo-random number generator used can be re-seeded to produce an identical number stream. It is worth noting that there are other factors that affect code determinism, ranging from the computing hardware to the software used to compile the code and even the ordering in which parallel execution takes place. These factors require consideration when deciding whether a simulation can be considered repeatable. The point here, however, is that given the same execution platform and serial execution, mdFoam+ is deterministic.

Repeatability is ensured within mdFoam+ by utilising a customised version of the base OpenFOAM *Random* classes which takes in a seed, or self-seeds using the current system time, and creates a cache of values which is refreshed only once the initial pool is utilised. The design decision to create an initial cache of values rather than calculate new values as they are needed was taken in order to reduce run-time computational overhead at the expense of a small amount of memory. Many OpenFOAM applications utilise pseudo-random number generation through this class, however the functionality in *Random* is built upon underlying POSIX utilities. The OpenFOAM distribution that contains mdFoam+ therefore enforces a non-standard build rule in its pre-defined make rules which ensures that OpenFOAM uses a more precise set of POSIX random number generation tools, as by default OpenFOAM uses the fastest but weakest. mdFoam+ and any associated tool that utilises random numbers can also be allowed to self-seed (in which case the current system date and time is used), effectively meaning each run of the same case will be unique, or a dictionary file can be provided which allows the seed to be manually entered. Finally, the random number seed used by an application is automatically output as a file named *random_[application name]* in the base of the case directory.

At the time of writing, the three applications which utilise pseudo-random numbers are mdFoam+, mdInitialiseField and createWeightField. The dictionary files to control the seeding of these are located at *[case]/system/randomDict_[application name]* (i.e. for mdFoam+, this would be *[case]/system/randomDict_mdFoamPlus*). An example dictionary file is as follows:

```

randomSeed               5000; //Seed used to create pseudo-random numbers
randomCacheSizeMultiplier 20;   //Determines intial cache size according to molecule count

```

In this case the pseudo-random number generator will be seeded with the integer value 5000, and an initial cache of values will be created that is equal to the number of molecules in the simulation multiplied by 20. In the case of a self-seeded simulation (or when this value is not provided), the cache size will always be equal to this, however it may sometimes be preferable to decrease or increase this size. An increase in cache size will mean a greater memory footprint but a decrease in the number of times that a new cache has to be generated because the initial cache has been utilised.

4.2. Case initialisation

Once all of the dictionary files needed to define a new case are in place and the underlying mesh has been created, as per Section 4.1.1, it is necessary to create an initial state that the MD simulation evolves from. This is done using the pre-processing tool mdInitialiseField, which, like mdFoam+ itself, is designed to be extensible to enable new initialisation algorithms to be implemented.

For the example case presented here the goal is to insert N water molecules into the cubic domain in order to satisfy a target fluid density of $\rho_m = 1000$ kg/m^3 according to the equation of mass density:

$$\rho_m = \frac{M}{V} = \frac{Nm_i}{L^3} \tag{13}$$

where $m_i = 2.99 \times 10^{-26}$ kg is the mass of a single molecule (as defined in *[case]/constant/moleculeProperties*) and $L = 5.05$ nm.

mdInitialiseField is controlled using a dictionary file located at *[case]/system/mdInitialiseDict*. The file for this example case is:

```
25  polyConfigurations
26  (
27
28      configuration
29      {
30          type                    polyBCC;
31          temperature             3.816;
32          bulkVelocity            (0.0 0.0 0.0);
33          molId                   water;
34          frozen                  no;
35
36          boundBox
37          {
38              startPoint          (0.9 0.9 0.9);
39              endPoint            (16 16 16);
40          }
41
42          unitCellSize            1.1;
43          N                       4298;
44      }
45  );
```

As the example presented here only has one species of molecule (*water*), there is only one configuration entry in the above file, which refers to the correct molecules according to the pre-defined value *molID* (see Section 4.1.3). Should multiple species be involved then one entry for each is required. In this case the initialisation algorithm used is the *polyBCC* method, which produces a body-centred cubic lattice of molecules within the created mesh; other methods are also available by default.

When initialising a case, the main rule to follow is that molecules should not overlap with each other. This means that two molecules are not so close that their separation distance, when used within the chosen potential calculations, will result in an immediate simulation blow-up. Generally the goal is to produce an initial layout where the distance between any two molecules is not less than 0.3 nm, although this distance is dependent on the underlying MD properties.

The configuration for the presented example avoids molecule overlap by filling a bounding box which allows a slight gap at the periodic boundaries, ensuring potentials calculated across boundaries work as expected. This is achieved by off-setting the starting point of the fill using the variable *startPoint*. The variable *unitCellSize* indicates the size of a cell in the associated BCC lattice, effectively controlling molecule separation, and then setting $N = 4298$ ensures the tool will generate exactly this many molecules. This latter value should be determined according to appropriate calculations such as the available domain size, the size of the bounding box, the unit cell size and the desired molecular density.

Once all case dictionaries are in place, ***mdInitialiseField*** should be run in the base of the case directory; this will generate an initial time folder named according to the value of the variable *startTime* in the dictionary *[case]/system/controlDict*.

4.3. Running mdFoam+

Once all dictionary files have been created and the case initialised with the appropriate pre-processing tools as per Sections 4.1.1 and 4.2, the MD calculations can then be started in serial straight away. As this suggests, execution will use a single mdFoam+ process only.

While a serial run may be suitable in the case of a small MD simulation, it is preferable to make use of multiple processors wherever possible, especially for larger simulations. As mdFoam+ utilises MPI-based domain decomposition, parallel execution can happen both when running on a workstation with a few CPU cores or when running on a large distributed memory system over hundreds or even thousands of cores. In order to execute an mdFoam+ case in parallel there is an extra pre-processing step needed. This is achieved using

the OpenFOAM utility ***decomposePar***, which reads a dictionary at *[case]/system/decomposeParDict*; the layout of this dictionary follows the OpenFOAM standard, therefore reference to the official documentation is encouraged. However, the most important consideration when decomposing large or complex domains is which algorithm to use. OpenFOAM provides access to a number of domain decomposition techniques, the choice of which produces the best decomposition has to be made on a case-by-case basis.

When ***decomposePar*** is executed in the base of the case folder, it creates a number of new folders, one for each processor being executed upon. Each of these contains a working copy of the sub-set of the overall domain created in Section 4.2, a complete copy of which can always be found in the initialisation folder named according to the starting time.

Once domain decomposition has been achieved, a parallel run of mdFoam+ can be initialised using the appropriate MPI run-time for the MPI library that was originally used during its compilation. An important point to note is that mdFoam+, like other OpenFOAM applications, must be informed if it is to run in parallel. This is done using a new command line switch passed through at the point of execution; so an example of running 256 MPI ranks using a typical environment would be:

```
mpirun -np 256 mdFoamPlus -parallel
```

4.4. Post-processing results

One of the powerful inherited features of building mdFoam+ within OpenFOAM is the ability to post-process and visualise results. By default, OpenFOAM provides a wrapper around the visualisation environment ParaView [11], called ***paraFoam***. This automatically loads all appropriate data of an OpenFOAM case into ParaView and sets up a work-flow within the tool to enable quick visualisation of results. It is important to note that this can be done at almost any time that data exists (i.e. it is possible to view an initialised MD case before any MD calculations are performed) by simply executing ***paraFoam*** in the base of the case folder.

An alternative to running ***paraFoam*** is the utility ***foamToVTK***. This processes the data and creates a new folder named *VTK* in the case directory, the contents of which can be viewed in any visualisation package that supports the VTK format.

It is also possible to visualise results created with mdFoam+ using the more common MD visualisation tool VMD [40]. While it is beyond the scope of this document to provide full details of how to utilise either ParaView or VMD to best effect, detailed instructions can be found within the documentation provided in the software release, where a number of scripts to automate some time-consuming processes can also be found.

Should results have been generated by a parallel run of mdFoam+, then it is important to recognise that only a portion of the whole domain will be contained within each processor folder that resides within the overall case folder. It is possible to reconstruct the whole domain for each stored time-step using the OpenFOAM ***reconstructPar*** utility, or to treat each set of results independently. However, it is important to ensure any processing tool is called from the base of the correct folder (i.e. if ***paraFoam*** is executed in the base of the case folder before ***reconstructPar*** is executed then this would be invalid).

4.5. Source code structure

As mdFoam+ is distributed as part of a customised OpenFOAM repository, the structure of its source code and that of any associated processing applications is in-line with other applications built within OpenFOAM. It is beyond the scope of this article to describe in full the ethos behind the OpenFOAM coding style, therefore it is recommended that the OpenFOAM documentation is first reviewed in order to understand the general source-code structure. The remainder of this section assumes a level of understanding by the reader of how OpenFOAM organises its applications and classes.

As with all OpenFOAM applications, the source code for mdFoam+ and its associated processing tools are located in two base folders within the general repository directory. The applications themselves can be found in the *applications* folder while the underlying C++ classes can be found in the *src* folder. In terms of applications, mdFoam+ can be found in *applications/discreteMethods/molecularDynamics*, while associated processing applications can be found in *applications/utilities/preProcessing/molecularDynamics*. Underlying classes can be found in *src/lagrangian/molecularDynamics*.

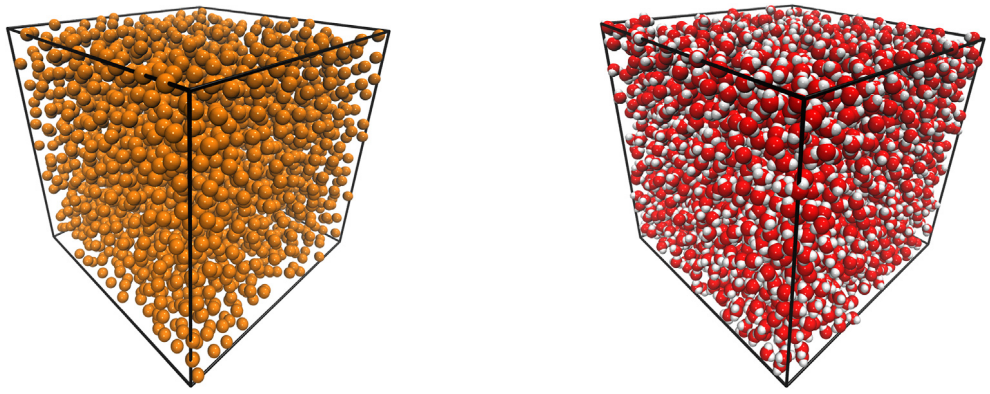
Documentation is also included with the repository, providing both installation instructions for a general workstation with a POSIX environment and in-depth technical details. These can be found in the *doc* folder, located at *doc/MicroNanoFlows*. Finally, tutorial example cases are also provided, and can be found within the *tutorials* folder located at *tutorials/discreteMethods*.

5. Validation

In this section, two examples are presented to validate mdFoam+ against another popular molecular dynamics solver, LAMMPS [1]. The Velocity-Verlet integration scheme is used with a time-step of 2 fs for all simulations, and a cut-off radius of 1.2 nm is used for all potentials. A cubic domain with side-length 5.1 nm and periodic boundary conditions in all three directions are used in both cases; this is similar to the example shown in Section 4 and illustrated in Fig. 4.

The first example consists of a monatomic system of 2744 argon atoms, as shown in Fig. 6(a). A Lennard-Jones (LJ) potential with parameters $\sigma = 3.405 \times 10^{-10}$ m and $\epsilon = 1.654 \times 10^{-21}$ J, is used to describe the interactions between the argon atoms. The system is first initialised from a simple lattice in mdFoam+ and allowed to equilibrate at a temperature of $T = 84$ K using a Berendsen thermostat. The equilibrated system is then simulated using both the mdFoam+ and LAMMPS solvers. The kinetic and potential energy per molecule is the basis for any MD software validation, and this is shown show in Fig. 7. Both mdFoam+ and LAMMPS give very similar kinetic and potential energies although mdFoam+ displays more fluctuations in results, which can be attributed to small differences in the sampling techniques used.

The second validation example consists of a molecular system of 4096 rigid water molecules, as shown in Fig. 6(b). The rigid TIP4P/2005 force field [41] is used to simulate the water molecules. This consists of four interaction sites: one oxygen site (no charge) with LJ parameters $\epsilon_{OO} = 1.2868 \times 10^{-21}$ J and $\sigma_{OO} = 0.31589$ nm, two hydrogen sites (0.5564 e), and one massless M site (-1.1128 e). A



(a) Argon cube. (b) Water cube.

Fig. 6. Molecular dynamics simulations of (a) a monatomic system of argon atoms, and (b) a molecular system of rigid water molecules, similar to the case illustrated in Fig. 4.

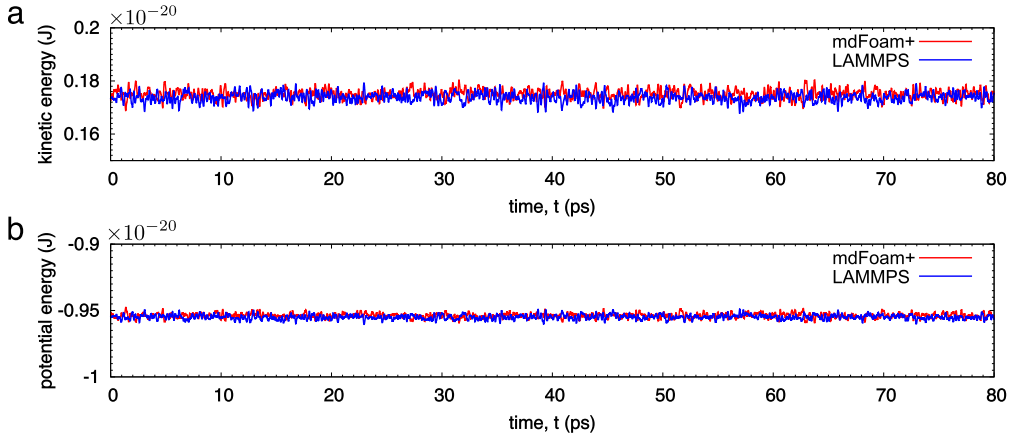


Fig. 7. Comparison of instantaneous (a) kinetic and (b) potential energies per atom for the argon validation system, simulated using mdFoam+ and LAMMPS.

smoothly truncated Coulomb potential is used for electrostatic interactions between different water molecules. The H–O–H bond angle and the O–H bond length in a water molecule are fixed at 104.52° and 0.09572 nm, respectively.

As in the previous example, the system is first initialised from a lattice using mdFoam+ and allowed to equilibrate at a temperature of $T = 300$ K. Simulations are then run using both mdFoam+ and LAMMPS on the equilibrated cases, using a Berendsen thermostat in mdFoam+ and a Nosé-Hoover thermostat in LAMMPS. In order to compare the fluctuations of the energies calculated by the two solvers, unbiased by thermal constraints, both thermostats are switched off after $t = 80$ ps. Results are shown in Fig. 8, where good agreement is shown in the mean value and fluctuations of the kinetic and potential energies.

6. Molecular dynamics examples

In order to demonstrate the functionality of mdFoam+, a number of example cases are now presented of typical MD problems. The first is the evaporation of a nanodroplet on a specific substrate; the second is the flow of water through a carbon nanotube; the third is the flow of water through a complex mixing channel.

6.1. Evaporating nanodroplet on a substrate

Droplet evaporation is not only ubiquitous in daily life but also related to a variety of engineering problems, such as spray combustion, coating, and 3D-printing [42]. With recent advances in nanotechnology, it is now possible to generate droplets at nanoscale lengths. Presented mdFoam+ simulations of salt water nanodroplets to show the potential for producing nanoscale features (e.g. nanocrystals, nanodots and nanowires) using evaporation [19,20].

The simulations are performed in a three-dimensional box with a side length of 18.8 nm in the $x-$ and $z-$directions (parallel to the solid surface) and 31.4 nm in the $y-$direction (perpendicular to the solid surface). Periodic boundary conditions are applied in all directions. To demonstrate the flexible features of initialising MD systems in mdFoam+, a cubic box of water molecules and Na^+ and Cl^- ions is first created, and then equilibrated at the target temperature. In a separate case, the platinum surface, with a face centred cubic (FCC) structure and a lattice constant of 3.92 Å, is built and then the supplied pre-processing utilities are used to shift the water cube such that it lies on the platinum substrate (avoiding overlaps). There are eight layers of atoms in the platinum surface: the bottom four are frozen, while the top four are treated using Lennard-Jones (LJ) potentials and coupled to a Berendsen thermostat to control the temperature.

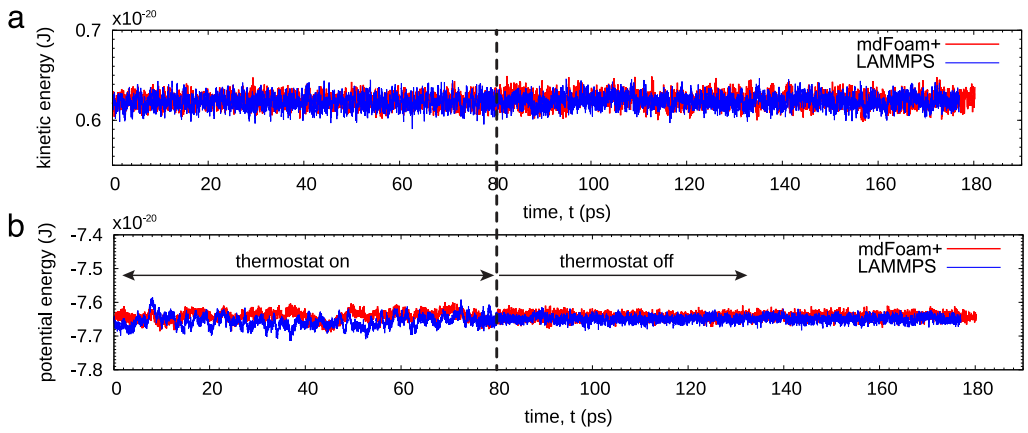
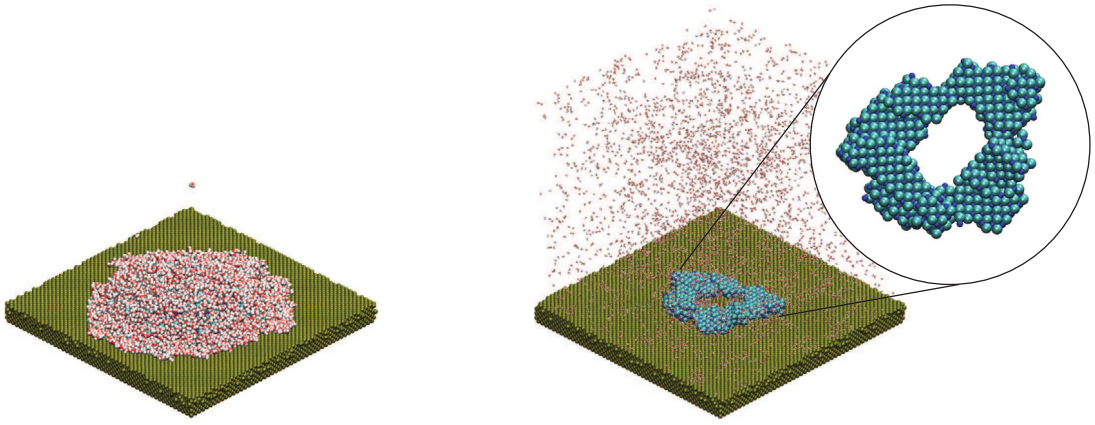


Fig. 8. Comparison of instantaneous (a) kinetic and (b) potential energies per atom for the water validation system, simulated using mdFoam+ and LAMMPS (thermostats are switched off at time $t = 80$ ps).



(a) Salt-water nanodroplet resting on substrate.

(b) Water evaporation from heated substrate.

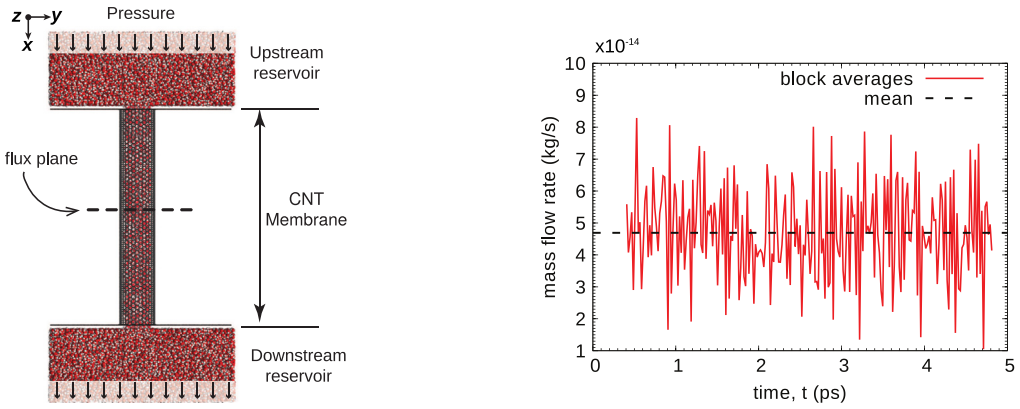
Fig. 9. A salt water nanodroplet MD simulation containing 4832 water molecules, 500 Na^+ and 500 Cl^- ions on a platinum surface. Snapshots of (a) initial equilibrium state; the droplet contact angle is 36.1° and the contact radius is 6.43 nm; and (b) after the substrate is heated and all water molecules have evaporated; inset: typical salt crystal ring deposit on the surface.

Table 3
Lennard-Jones potential parameters for atomic sites in water molecules, Na^+ and Cl^- ions, and for Pt atoms. These values are taken from [19,43].

Site	ε ($\times 10^{-21}$ J)	σ (nm)
H	0	0
O	1.287	0.31589
M	0	0
Na^+	0.325	0.245
Cl^-	1.043	0.41
Pt	110.99	0.2471
H-Na^+	0.299	0.155
H-Cl^-	0.536	0.2375

A rigid TIP4P/2005 model (as above) is used to simulate the water molecules. The Na^+ (1.0 e) and Cl^- (−1.0 e) ions are treated as non-polarised sites with fixed charges. The LJ parameters for all sites are listed in Table 3, and use a cut off radius of 1.2 nm for both the LJ and the Coulomb potentials. Lorentz–Berthelot mixing rules are used to determine the LJ cross-interactions between two different kinds of sites. There are usually no LJ interactions between an H site and a Na^+ or Cl^- ion site, however, as reported by Alejandre et al. [43], additional LJ potential interactions for H–Cl^- and H-Na^+ pairs need to be implemented in order to prevent the overestimation of ion hydration. The Lorentz–Berthelot mixing rules are also employed for the Pt-O, Pt-Na^+ and Pt-Cl^- pairs, but with rescaled energies in order to produce an artificial hydrophilic/hydrophobic surface.

The MD system is run for 4 ns of problem time at 300 K in order to reach an equilibrium state, when the box of salt water has turned into a droplet. Fig. 9(a) shows the specific case of a droplet with 4832 water molecules, 500 Na^+ and 500 Cl^- ions formed on the platinum surface. After the system reaches equilibrium, the temperature of the platinum surface is then increased linearly to 600 K within 4 ns. Within this period, the liquid water molecules in the droplets evaporate and become vapour, while all the Na^+ and Cl^- ions crystallise within the droplet, until they finally form a salt crystal deposit on the surface after complete evaporation, as shown in Fig. 9(b).



(a) MD setup.

(b) Mass flow rate measurements at flux plane.

Fig. 10. (a) MD simulation set-up of a carbon nanotube (CNT) membrane section; (b) mass flow rate measurements of water crossing the flux plane of the CNT.

Depending on the solid–liquid interaction strength and the evaporation rate, either a clump or a ring-like pattern of salt crystals is left on the surface after evaporation. In the case shown in Fig. 9(b), where the temperature rise rate is 75 K/ns, a ring-like pattern is deposited (a top view of the deposit pattern is shown inset in the figure). This is reminiscent of the so-called *coffee-ring* phenomenon. The reason for this pattern is that there is a pinned stage in the evaporation process, during which salt ions can move from the centre out to the rim of the droplets, following the outward capillary flow [19,20].

6.2. Water flowing through a carbon nanotube

Membranes of aligned carbon nanotubes (CNTs) offer the possibility of significant performance improvements over current polymeric membranes for reverse osmosis. As the diameter of the tubes can be as small as 1 nm, they act as a molecular sieve: they are narrow enough to block salt ions and other contaminants, but still wide enough to allow water molecules to flow through. Furthermore, the flow through CNTs can be much larger than expected by hydrodynamic theory, which means a high packing factor of aligned CNTs in a membrane could reduce running costs associated with desalination.

In this section, an exemplar simulation of pressure-driven flow through a CNT membrane is described. This is depicted in Fig. 10(a). The case is created by generating a CNT of length 20 nm and diameter 2 nm, as well as two graphene sheets, in a fully periodic cubic domain of dimensions 28.7 nm $\times$ 10.65 nm $\times$ 10.33 nm in the $x-$ $y-$ and $z-$directions respectively. The molecule deletion pre-processing utility ***deleteMolecules*** is then used to create holes in the graphene sheets that match the diameter and locations of the entrance and exit of the CNT. The graphene sheets then act as the bounding surface of this modelled membrane section. Water in a lattice is initialised at approximately 1000 kg/m^3 and 300 K on both sides of the membrane and within the CNT. The previously listed potentials for the TIP4P/2005 model are used, and LJ potentials between oxygen sites O and carbon atoms C are obtained from additional MD simulations calibrated against experiments of droplets on graphite [16,18], giving: $\sigma_{CO} = 0.319$ nm and $\epsilon_{CO} = 7.09 \times 10^{-22}$ J.

The MD system is first equilibrated with a Berendsen thermostat applied in bins in the upstream and downstream reservoirs, as well as a force controller at the periodic edges of the MD simulation (as shown in Fig. 10(a)). This imposes a pressure drop of around 200 MPa across the membrane section [13,37]. A further simulation is also run to change the absolute pressure in the upstream reservoir by controlling the number of water molecules using the FADE algorithm [21].

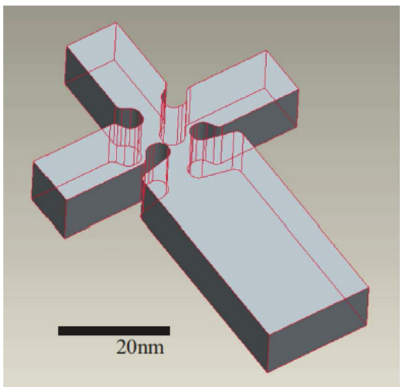
An MD simulation subsequent to the initial 5 ns of equilibration time is then run to measure the mass flow rate at a flux plane located within the CNT. The flow rate is the key performance parameter in this case; mass flow rate is measured at every time-step, and post-processed using averages over 10000 time-steps. A further mean is then taken, resulting in a predicted mass flow rate of 4.7 $(\pm 0.2) \times 10^{-14}$ kg/s, as shown in Fig. 10(b).

6.3. Flow through a complex mixing channel

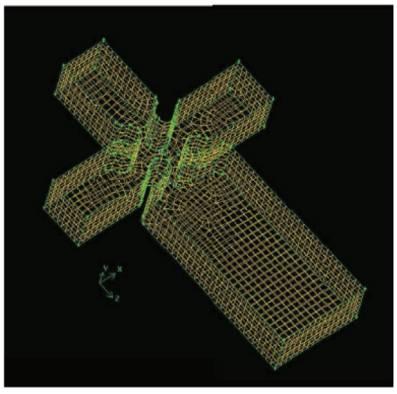
The premise of this final example is to demonstrate the capability of mdFoam+ when complex boundaries, typical of CFD simulations, are required. The platinum substrate in Section 6.1 and the carbon nanotube in 6.2 both contain solid-wall molecules. These provide a physical boundary for the liquid/vapour molecules to interact with, and make the flow through or around a nanofluidic device straightforward for most MD solvers. However, modelling larger nanofluidic devices and systems of arbitrary complexity is computationally prohibitive, as a large cost is associated with modelling the bounding wall molecules.

The ability to deal with complex mesh geometries is a key feature of mdFoam+. In this section, a possible solution to this problem is demonstrated, which involves replacing the wall molecules with simple reflective boundaries and boundary-force models that produce the effect of the solid–fluid short range interactions. This capitalises on OpenFOAM's ability to produce meshed domains of an arbitrary complexity.

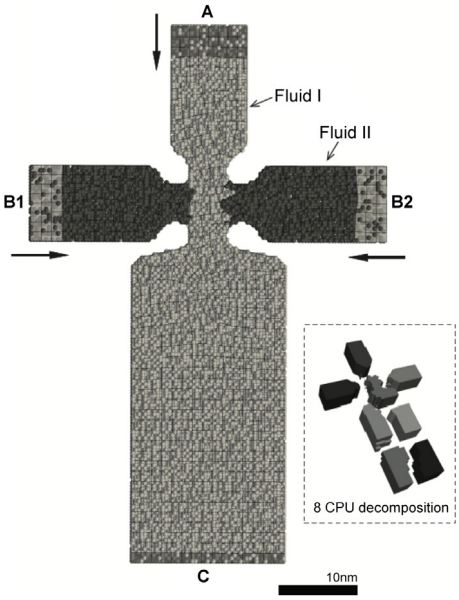
A mixing channel of three-inlet, one-outlet design is considered, taken from Hertzog et al. [44] but its scale is reduced down to nanometres so that a reasonable mixing time-scale may be simulated using molecular dynamics. The work-flow demonstrated for this modified nano-mixer geometry is defined in such a way that it can be generally adopted for other complex geometry cases.



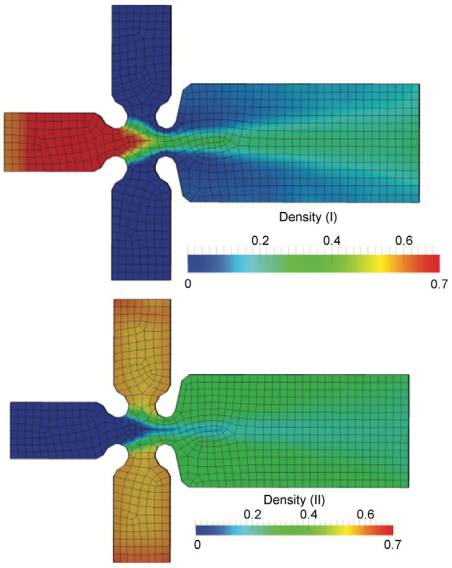
(a) CAD drawing of nanofluidic channel.



(b) Meshed version of CAD drawing.



(c) Initialised within mdFoam+.



(d) Final simulation results.

Fig. 11. MD set up of a nanofluidic mixing channel device: (a) part drawn in CAD package; (b) part meshed; (c) part exported to OpenFOAM and initialised with two species of molecules in mdFoam+; (d) simulation results of mixing (top: density for species I, bottom: density for species II).

Initially the part is drawn in a computer aided design (CAD) drawing application (see Fig. 11(a)). The geometry is then meshed using hexahedral cells (see Fig. 11(b), and subsequently exported into OpenFOAM, where it is filled with molecules of two liquid species, Fluid I and Fluid II, as shown in Fig. 11(c).

The two fluids are essentially modelled as isotopes of argon, and the Lennard-Jones potential is used for all I–I, II–I and II–II fluid interactions. Both fluids have identical physical properties but a different identification (ID) number, so that mixing can be observed and measured. The non-periodic boundary conditions at the inlets (A, B1, B2) are devised to supply molecules of these artificial fluids at a constant rate using constraints that control density $\rho^* = 0.6$, temperature $T^* = 2.4$ and velocity ($u^*_A = 0.25$; $u^*_{B1} = 0.07$; $u^*_{B2} = 0.07$) [7,10]. At the outlet C, molecules are removed at a constant rate, to satisfy continuity. The boundary force model that is applied between the liquid molecules and the internal surfaces of the device is obtained from an MD pre-simulation that measures the mean force on liquid molecules as a function normal to the wall. Field measurements for the two species densities after a typical simulation run are shown in Fig. 11(d). More details on this system, such as the materials used, can be found in references [7,10].

7. Conclusions and future work

This article has introduced mdFoam+, which has been developed in order to study complex fluid dynamics problems using molecular dynamics and to explore hybrid simulations that couple MD to continuum-fluid solvers. It is designed entirely within the open-source OpenFOAM framework, and implements a highly-extensible and fully object-oriented C++ based approach. The code is released under the same GPL license as the OpenFOAM base it is released alongside, and is available as a public software repository [5] that includes documentation and example cases. mdFoam+ is parallelised using an MPI-based domain-decomposition approach built upon the parallel capability provided by OpenFOAM.

To date the code has been used for various complex MD problems in nanofluidics and has proved robust, relatively computationally efficient (within the bounds of its current Array of Structures design) and reliable.

The mdFoam+ solver is intended to be a useful research and development platform for those utilising molecular dynamics in their work. It is designed to be easily approached in terms of setting up new cases, with OpenFOAM's mesh-based processing forming the basis of its domain description. Capability extension is inherent to mdFoam+ and typically involves creation of new small C++ classes that are derived from existing ones. Nearly all aspects of its functionality can be extended in this way. mdFoam+ is presented here in the hope that it will be used and its capabilities extended by different research and development groups from different disciplines.

The development state of mdFoam+ is ongoing, however, the current public release is stable and will not be expanded until any new significant features have been tested and validated. As mdFoam+ is built within OpenFOAM, it is likely that general optimisations to the latter's Lagrangian functionality will force the use of newer versions as and when this is deemed beneficial to mdFoam+. When this happens, backwards compatibility with older releases cannot be guaranteed. To mitigate this potential problem for users of older versions, any new version will be released as a separate entity, ensuring legacy versions are maintained. In the case of modifications to existing releases, backwards compatibility is ensured wherever possible, by not modifying case file input parameter requirements (meaning that existing cases can still be used without modification) and ensuring any new parameters introduced are not requirements for running a case. Algorithmic compatibility is tested by the authors prior to public release of any changes using the simple tutorial test cases provided with mdFoam+.

Optimisations and bug fixes are often applied to the repository however two major future developments of the code will centre on improving serial and then parallel performance; first through memory design optimisation, and then through hybridisation of parallelisation strategies beyond pure MPI. Another area that is expected to be developed in the future is a general solver coupling interface, primarily with the aim of enabling mdFoam+ to interface with other OpenFOAM applications but also, more generally, other applications outside of the OpenFOAM suite. This coupling strategy will provide an interface both to mesh (Eulerian) and mesh-free (Lagrangian) techniques.

Acknowledgements

This work is supported in the UK by the Engineering and Physical Sciences Research Council (EPSRC) under grants EP/K038621/1, EP/K038664/1, EP/K038427/1 and EP/N016602/1. MKB and JMR would also like to acknowledge the ARCHER Leadership grant which enabled scaling studies to be run on ARCHER.

References

[1] S. Plimpton, J. Comput. Phys. 117 (1995) 1–19. http://dx.doi.org/10.1006/jcph.1995.1039.
[2] I.T. Todorov, W. Smith, K. Trachenko, M.T. Dove, J. Mater. Chem. 16 (2006) 1911–1918. http://dx.doi.org/10.1039/B517931A.
[3] J.C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R.D. Skeel, L. Kale, K. Schulten, J. Comput. Chem. 26 (2005) 1781–1802. http://dx.doi.org/10.1002/jcc.20289.
[4] H.G. Weller, G. Tabor, H. Jasak, C. Fureby, Comput. Phys. 12 (1998) 620–631. http://dx.doi.org/10.1063/1.168744.
[5] M.K. Borg, S.M. Longshaw, S.B. Ramisetti, J. Zhang, D.A. Lockerby, D.R. Emerson, J.M. Reese, mdFoam+, https://github.com/MicroNanoFlows, 2016.
[6] G.B. Macpherson, Molecular Dynamics Simulation in Arbitrary Geometries for Nanoscale Fluid Mechanics (Ph.D. thesis), University of Strathclyde, Glasgow, 2008.
[7] M.K. Borg, Hybrid Molecular-Continuum Modelling of Nanoscale Flows (Ph.D. thesis), University of Strathclyde, Glasgow, 2010.
[8] G.B. Macpherson, M.K. Borg, J.M. Reese, Mol. Simul. 33 (15) (2007) 1199–1212. http://dx.doi.org/10.1080/08927020701730724.
[9] G.B. Macpherson, J.M. Reese, Mol. Simul. 34 (1) (2008) 97–115. http://dx.doi.org/10.1080/08927020801930554.
[10] M.K. Borg, G.B. Macpherson, J.M. Reese, Mol. Simul. 36 (10) (2010) 745–757. http://dx.doi.org/10.1080/08927021003752812.
[11] J. Ahrens, B. Geveci, C. Law, ParaView: An End-User Tool for Large Data Visualization, Elsevier, 2005.
[12] W.D. Nicholls, Molecular Dynamics Simulations of Liquid Flow in and Around Carbon Nanotubes (Ph.D. thesis), University of Strathclyde, Glasgow, 2012.
[13] W.D. Nicholls, M.K. Borg, D.A. Lockerby, J.M. Reese, Microfluidics Nanofluidics 12 (1–4) (2012) 257–264. http://dx.doi.org/10.1007/s10404-011-0869-3.
[14] W.D. Nicholls, M.K. Borg, D.A. Lockerby, J.M. Reese, Mol. Simul. 38 (10) (2012) 781–785. http://dx.doi.org/10.1080/08927022.2011.654205.
[15] K. Ritos, M.K. Borg, N.J. Mottram, J.M. Reese, Phil. Trans. R. Soc. A 374 (2060) (2015). http://dx.doi.org/10.1098/rsta.2015.0025.
[16] K. Ritos, Water Flow at the Nanoscale: A Computational Molecular and Fluid Dynamics Investigation (Ph.D. thesis), University of Strathclyde, Glasgow, 2014.
[17] K. Ritos, D. Mattia, F. Calabrò, J.M. Reese, J. Chem. Phys. 140 (2014) 014702. http://dx.doi.org/10.1063/1.4846300.
[18] K. Ritos, N. Dongari, M.K. Borg, Y. Zhang, J.M. Reese, Langmuir 29 (23) (2013) 6936–6943. http://dx.doi.org/10.1021/la401131x.
[19] J. Zhang, M.K. Borg, K. Sefiane, J.M. Reese, Phys. Rev. E 92 (2015) 052403. http://dx.doi.org/10.1103/PhysRevE.92.052403.
[20] J. Zhang, M.K. Borg, K. Ritos, J.M. Reese, Langmuir 32 (6) (2016) 1542–1549. http://dx.doi.org/10.1021/acs.langmuir.5b04424.
[21] M.K. Borg, D.A. Lockerby, J.M. Reese, J. Chem. Phys. 140 (2014) 074110. http://dx.doi.org/10.1063/1.4865337.
[22] M.K. Borg, D.A. Lockerby, J.M. Reese, J. Comput. Phys. 255 (2013) 149–165. http://dx.doi.org/10.1016/j.jcp.2013.08.022.
[23] A. Agius Anastasi, K. Ritos, G. Cassar, M.K. Borg, Mol. Simul. 42 (18) (2016) 1–10. http://dx.doi.org/10.1080/08927022.2016.1209753.
[24] L. Verlet, Phys. Rev. 159 (1967) 98–103. http://dx.doi.org/10.1103/PhysRev.159.98.
[25] G.B. Macpherson, Niklas Nordin, Henry G. Weller, Commun. Numer. Methods. Eng. 25 (3) (2009) 263–273. http://dx.doi.org/10.1002/cnm.1128.
[26] A. Dullweber, B. Leimkuhler, R. McLachlan, J. Chem. Phys. 107 (15) (1997) 5840–5851. http://dx.doi.org/10.1063/1.474310.
[27] D.C. Rapaport, The Art of Molecular Dynamics Simulation, second ed., Cambridge University Press, 2004.
[28] R. Vacondio, S.M. Longshaw, S. Siso, L. Mason, B.D. Rogers, Proc. 11th Smoothed Particle Hydrodynamics European Research Interest Community Conference (SPHERIC 2016), Vol. 11, 2016.
[29] M.P. Allen, D.J. Tildesley, Computer Simulation of Liquids, Oxford University Press, 1987.
[30] M.K. Borg, D.A. Lockerby, J.M. Reese, J. Comput. Phys. 233 (2013) 400–413. http://dx.doi.org/10.1016/j.jcp.2012.09.009.
[31] M.K. Borg, D.A. Lockerby, J.M. Reese, Microfluidics Nanofluidics 15 (4) (2013) 541–557. http://dx.doi.org/10.1007/s10404-013-1168-y.
[32] D.A. Lockerby, C.A. Duque-Daza, M.K. Borg, J.M. Reese, J. Comput. Phys. 237 (2013) 344–365. http://dx.doi.org/10.1016/j.jcp.2012.11.032.
[33] D.M. Holland, D.A. Lockerby, M.K. Borg, W.D. Nicholls, J.M. Reese, Microfluidics Nanofluidics 18 (3) (2014) 461–474. http://dx.doi.org/10.1007/s10404-014-1443-6.
[34] D. Stephenson, D.A. Lockerby, M.K. Borg, J.M. Reese, Microfluidics Nanofluidics 18 (5) (2014) 841–858. http://dx.doi.org/10.1007/s10404-014-1476-x.
[35] D.M. Holland, M.K. Borg, D.A. Lockerby, J.M. Reese, Comput. & Fluids 115 (2015) 46–53. http://dx.doi.org/10.1016/j.compfluid.2015.03.023.
[36] M.K. Borg, D.A. Lockerby, J.M. Reese, J. Fluid Mech. 768 (2014) 388–414. http://dx.doi.org/10.1017/jfm.2015.83.
[37] K. Ritos, M.K. Borg, D.A. Lockerby, D.R. Emerson, J.M. Reese, Microfluidics Nanofluidics 19 (5) (2015) 997–1010. http://dx.doi.org/10.1007/s10404-015-1617-x.
[38] H.J.C. Berendsen, J.P.M. Postma, W.F. Gunsteren, A. DiNola, J.R. Haak, J. Chem. Phys. 81 (1984) 3684–3690. http://dx.doi.org/10.1063/1.448118.
[39] J. Li, D. Liao, S. Yip, Phys. Rev. E 57 (1998) 7259–7267. http://dx.doi.org/10.1103/PhysRevE.57.7259.
[40] W. Humphrey, A. Dalke, K. Schulten, J. Molec. Graph. 14 (1996) 33–38. http://dx.doi.org/10.1016/0263-7855(96)00018-5.
[41] J.L.F. Abascal, C. Vega, J. Chem. Phys. 123 (23) (2005) 234505. http://dx.doi.org/10.1063/1.2121687.
[42] D. Brutin, Droplet Wetting and Evaporation: From Pure to Complex Fluids, Academic Press, 2015.
[43] J. Alejandre, G.A. Chapela, F. Bresme, J. Hansen, J. Chem. Phys 130 (17) (2009) 174505. http://dx.doi.org/10.1063/1.3124184.
[44] D.E. Hertzog, B. Ivorra, B. Mohammadi, O. Bakajin, J.G. Santiago, J. Anal. Chem. 78 (13) (2006) 4299–4306. http://dx.doi.org/10.1021/ac051903j.