Computer Physics Communications 224 (2018) 22–43

Contents lists available at ScienceDirect

Computer Physics Communications

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

Feature article

dsmcFoam+: An OpenFOAM based direct simulation Monte Carlo solver☆

C. White [a,*], M.K. Borg [b], T.J. Scanlon [c], S.M. Longshaw [d], B. John [d], D.R. Emerson [d], J.M. Reese [b]

[a] School of Engineering, University of Glasgow, Glasgow G12 8QQ, UK
[b] School of Engineering, University of Edinburgh, Edinburgh EH9 3FB, UK
[c] Department of Mechanical and Aerospace Engineering, University of Strathclyde, Glasgow G1 1XJ, UK
[d] Scientific Computing Department, The Science & Technology Facilities Council, Daresbury Laboratory, Warrington, Cheshire WA4 4AD, UK

ARTICLE INFO

Article history:
Received 6 March 2017
Received in revised form 21 August 2017
Accepted 25 September 2017
Available online 24 October 2017

Keywords:
dsmcFoam+
OpenFOAM
direct simulation Monte Carlo
DSMC
Hypersonics
Nano-scale
Micro-scale
Rarefied gas dynamics

ABSTRACT

dsmcFoam+ is a direct simulation Monte Carlo (DSMC) solver for rarefied gas dynamics, implemented within the OpenFOAM software framework, and parallelised with MPI. It is open-source and released under the GNU General Public License in a publicly available software repository that includes detailed documentation and tutorial DSMC gas flow cases. This release of the code includes many features not found in standard dsmcFoam, such as molecular vibrational and electronic energy modes, chemical reactions, and subsonic pressure boundary conditions. Since dsmcFoam+ is designed entirely within OpenFOAM's C++ object-oriented framework, it benefits from a number of key features: the code emphasises extensibility and flexibility so it is aimed first and foremost as a research tool for DSMC, allowing new models and test cases to be developed and tested rapidly. All DSMC cases are as straightforward as setting up any standard OpenFOAM case, as dsmcFoam+ relies upon the standard 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 a DSMC simulation is not typical of most OpenFOAM applications. We show that dsmcFoam+ compares well to other well-known DSMC codes and to analytical solutions in terms of benchmark results.

Program summary
Program title: dsmcFoam+
Program Files doi: http://dx.doi.org/10.17632/7b4xkpx43b.1
Licensing provisions: GNU General Public License 3 (GPL)
Programming language: C++
Nature of problem: dsmcFoam+ has been developed to help investigate rarefied gas flow problems using the direct simulation Monte Carlo (DSMC) method. It provides an easily extended, parallelised, DSMC environment.
Solution method: dsmcFoam+ implements an explicit time-stepping solver with stochastic molecular collisions appropriate for studying rarefied gas flow problems.
References: All appropriate methodological references are contained in the section entitled **References**.

1. Introduction

The direct simulation Monte Carlo (DSMC) technique is a stochastic particle-based method for simulating dilute gas flow problems. The method was pioneered by Bird [1] in the 1960s and has since become one of the most accepted methods for solving gas flows in the non-equilibrium Knudsen number regime. In a DSMC simulation, a single particle represents a large number of real gas atoms or molecules, reducing the computational expense relative to a fully deterministic method such as molecular dynamics (MD). Each of these particles is

☆ 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: craig.white.2@glasgow.ac.uk (C. White).

https://doi.org/10.1016/j.cpc.2017.09.030

Table 1
Comparison of dsmcFoam and dsmcFoam+ capabilities.

Feature	dsmcFoam	dsmcFoam+
Arbitrary 2D geometries	✓	✓
Arbitrary 3D geometries	✓	✓
Arbitrary axisymmetric geometries	✗	✓
Parallel processing	✓	✓
Rotational energy	✓	✓
Vibrational energy	✗	✓
Electronic energy	✗	✓
Chemical reactions	✗	✓
Gravitational force controller	✗	✓
Mass flow rate measurement	✗	✓
Simulation quality reports	✗	✓
Dynamic load balancing	✗	✓

free to move in space, according to its own velocity and the local time step, and to interact with other particles and the boundaries of the system. Inter-particle collisions are handled in a stochastic manner after all particle movements have taken place. In this way, an evolving simulation emulates the physics of a real gas rather than attempting to solve Newton's equations of motion for a very large number of individual atoms/molecules, as in MD.

The software application described in this article, dsmcFoam (later, dsmcFoam+), fulfils a similar role to alternatives such as MONACO [2], DAC [2], or SPARTA [3] (to name a few). However, it is distinct from these others because all of its functionality is built entirely within OpenFOAM [4]. This provides a built-in level of modularity, as well as potential interoperability by way of coupling with a large selection of other solver types that are also available in OpenFOAM. It is released open-source under the same General Public License (GPL) as the version of OpenFOAM it currently uses as its base. Up-to-date versions of dsmcFoam+, along with the OpenFOAM base it uses, are publicly available via a repository [5].

OpenFOAM [4] (or **Open**-source **F**ield **O**peration **A**nd **M**anipulation) is an open-source suite of libraries and applications initially designed to solve computational fluid dynamics (CFD) problems. It has become more complex over the years since its initial release, but its basic principle remains 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 a framework that can be used to build new applications.

dsmcFoam has been released with OpenFOAM since version 1.7. It was initially developed by OpenCFD Ltd, in collaboration with Scanlon et al. [6], as an extension to the molecular dynamics solver *mdFoam* developed by Macpherson et al. [7–11]. The core DSMC functionality of dsmcFoam has since remained largely unchanged and can be found in the current release of OpenFOAM.

This article concerns a branch of dsmcFoam that has undergone significant development since the original was accepted into OpenFOAM, and is referred to henceforth as dsmcFoam+. The executable is named dsmcFoamPlus in order to conform with long-standing OpenFOAM naming conventions using only alphabetic characters. This solver is a direct continuation and development of the original dsmcFoam, branched following its release with OpenFOAM-1.7. It provides an enhanced set of DSMC capabilities when compared to dsmcFoam [12–18], as well as new capabilities in terms of parallel performance through the introduction of dynamic load balancing. Table 1 gives an overview of the differences between dsmcFoam and dsmcFoam+. In addition to these key feature differences, dsmcFoam+ includes a greatly extended suite of boundary conditions and macroscopic property measurement tools compared to dsmcFoam.

In basing the core functionality of dsmcFoam+ around standard OpenFOAM libraries, the code is able to make use of powerful meshing facilities, parallelised Lagrangian/mesh-tracking algorithms, templated particle classes, and pre- and post-processing methods. As an example of this, DSMC particles 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 particles), as well as visualising results via ParaView [19]. Access to the mesh processing capabilities of OpenFOAM provides a powerful tool to define DSMC simulations with complex mesh structures, which are often a requirement for three-dimensional, realistic geometries.

A key strength of dsmcFoam+ comes from its strict adherence to OpenFOAM coding practices, meaning that its design is almost entirely modular in the form of C++ classes. As an example, if it is desirable to add a new binary particle collision formulation then all that needs to be done is to create a copy of the existing derived binary collision classes (which are themselves specialisms of the base binary collision class) and then modify the algorithm that it implements in order to calculate the new routine. Once this new class is included in the application's compilation, it automatically becomes available as a valid selection for a case. In order to use the new collision model, the user then only needs to change the textual description in the appropriate configuration file.

2. Direct simulation Monte Carlo with dsmcFoam+

2.1. Background

Rarefied gas dynamics is the study of gas flows in which the molecular mean free path, λ, is not negligible with respect to the characteristic length scale, L, of the system under consideration. The degree of rarefaction is defined by the Knudsen number,

$$Kn = \frac{\lambda}{L}. \tag{1}$$

In the limit of zero Knudsen number, inter-molecular collisions dominate and the gas is in perfect thermodynamic equilibrium. But, as the Knudsen number increases, molecular collisions become less frequent until the free-molecular limit is reached where inter-molecular collisions are very unlikely. Since it is inter-molecular collisions and gas–surface interactions that drive a system towards thermodynamic equilibrium, it is clear that non-equilibrium effects become dominant with increasing Knudsen number. The different Knudsen number regimes are illustrated in Fig. 1.

A description of each regime and its appropriate numerical approach can be summarised as [20]:

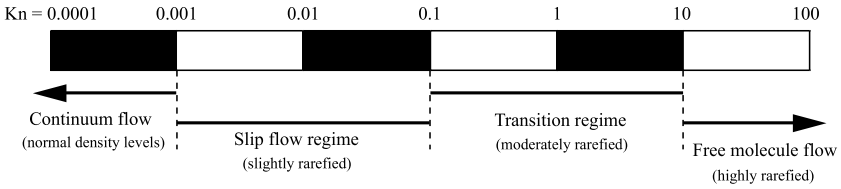


Fig. 1. Knudsen number regimes, adapted from Ref. [20].

- $Kn \rightarrow 0$: inviscid flow (Euler fluid equations);
- $Kn \leq 0.001$: continuum regime (Navier–Stokes–Fourier fluid equations);
- $0.001 \leq Kn \leq 0.1$: slip regime (Navier–Stokes–Fourier fluid equations with velocity-slip and temperature-jump boundary conditions);
- $0.1 \leq Kn \leq 10$: transition regime (Boltzmann equation or particle methods such as DSMC);
- $Kn \geq 10$: free-molecular regime (collisionless Boltzmann equation or particle methods such as DSMC).

2.1.1. Continuum regime

For the continuum flow regime, the familiar Navier–Stokes–Fourier (NSF) continuum-fluid equations provide an excellent approximation for gas flows that are very close to equilibrium. It is assumed that local macro-properties can be described as averages over elements that are large compared to the microscopic structure of the fluid, but small enough with respect to the macroscopic phenomena to permit the use of differential calculus. Inter-molecular collisions are dominant at low Knudsen number, i.e. there are enough molecular collisions occurring for local thermodynamic equilibrium to be reached in a very short time compared to the macroscopic time scale. In the limit of zero Knudsen number, the NSF equations can be reduced to the inviscid Euler equations because molecular diffusion can be ignored, and so the transport terms in the continuum momentum and energy equations are negligible [20].

2.1.2. Slip regime

As the Knudsen number becomes significant, molecule–surface interactions become less frequent and regions of non-equilibrium start to appear near surfaces. This can be observed from a macroscopic point of view as the gas velocity and temperature (U_g and T_g) at a surface not obtaining the same values as the surface itself (U_s and T_s). These phenomena are known as velocity-slip and temperature-jump, respectively. If this type of non-equilibrium is in the flow, the range of validity of the NSF equations can be extended to the slip regime by applying Maxwell's velocity-slip [21] and Von Smoluchowski's temperature-jump [22] boundary conditions.

2.1.3. Transition and free-molecular regimes

In the transition and free-molecular regimes, non-equilibrium effects dominate and a solution to the Boltzmann equation must be sought, as the assumption of linear constitutive relations in the NSF equations is no longer valid. Examples include flows in micro-scale geometries (e.g. micro-channels) [23], and hypersonic flows, particularly at low ambient pressures [24]. In these cases, the Navier–Stokes–Fourier (NSF) equations fail to give accurate predictions of the flow behaviour because the assumptions of a continuum fluid and local equilibrium break down. Recourse to the Boltzmann equation must be sought for an accurate description of the flow behaviour. The Boltzmann equation for a single-species, monatomic, non-reacting gas takes the form:

$$\frac{\partial (nf)}{\partial t} + \mathbf{c}\frac{\partial (nf)}{\partial \mathbf{r}} + \mathbf{F}\frac{\partial (nf)}{\partial \mathbf{c}} = J\left(f, f^*\right), \tag{2}$$

where nf is the product of the number density and the normalised molecular velocity distribution function, $\mathbf{r}$ and $\mathbf{c}$ are the position and velocity vectors of a molecule, respectively, $\mathbf{F}$ is an external force, $J\left(f, f^*\right)$ is a non-linear integral that describes the binary collisions, and the superscript $*$ represent post-collision properties. The collision integral takes the form:

$$J\left(f, f^*\right) = \int_{-\infty}^{\infty}\int_{0}^{4\pi} n^2\left(f^* f_1^* - f f_1\right) c_r \sigma_T d\Omega d\mathbf{c}_1, \tag{3}$$

where f and f_1 are the velocity distribution function at $\mathbf{c}$ and $\mathbf{c}_1$ respectively, c_r is the relative speed of two colliding molecules, σ_T is the molecular cross-section, and Ω is the solid collision angle.

In 1992, a mathematical proof that the DSMC method provides a solution to the Boltzmann equation for a monatomic gas in the limiting case of an infinite number of particles was published [25], removing any remaining ambiguity over how DSMC relates to the Boltzmann equation. The DSMC method is comprehensively described in Bird's 1994 and 2013 monographs [1,26].

2.2. Initialisation

A DSMC simulation performed using dsmcFoam+ begins with a set of pre-located particles. This process is achieved using pre-processing tools and involves specifying the positions of the domain extremities, the velocities of the particles, and the type of particles (i.e. species, mass, internal energy parameters). The user specifies macroscopic values of temperature, velocity, and density that are then used to insert particles with energies and positions that return, on average, these macroscopic parameters.

dsmcFoam+ comes with a number of pre-processing utilities that enable the user to prepare a DSMC case, such as basic initialisation by filling the entire domain with particles based on a user-defined temperature, velocity, and density, or different regions of the mesh can be assigned different values. These tools are built within OpenFOAM, so they are as extensible as the main dsmcFoam+ solver, and implementing new initialisation models is a relatively simple task.

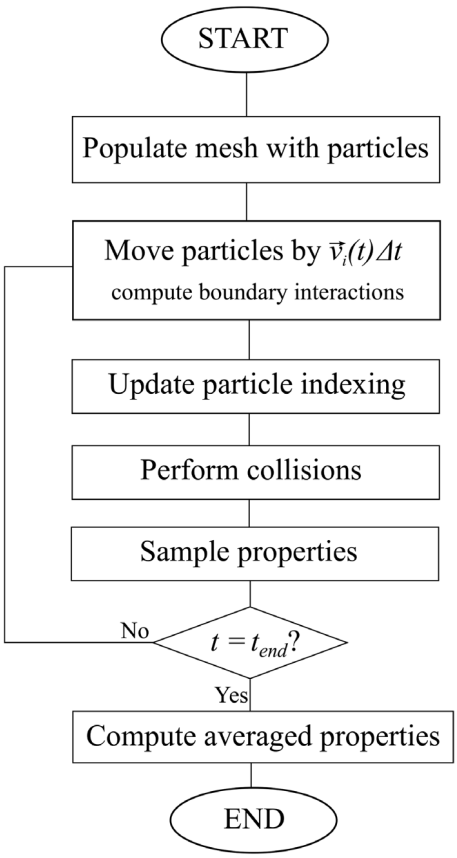


Fig. 2. Flow chart of the basic DSMC time-integration scheme.

2.3. Algorithmic overview

dsmcFoam+ implements an explicit time-stepping scheme in order to move particles in time and space within the computational domain. The basic algorithm that all DSMC solvers follow, including dsmcFoam+, can be described for a single DSMC time-step, $t \rightarrow t + \Delta t$, as follows, and shown in Fig. 2 :

Step 1 Update the position of all N particles in the system using OpenFOAM's inbuilt particle tracking algorithm [27], as discussed in Section 2.4, which handles motion of particles across faces of the mesh (and also deals with boundaries). In its mathematical form, the move step for the ith particle is given by:

$$\vec{r}_i\,(t + \Delta t) = \vec{r}_i\,(t) + \vec{v}_i\,(t)\,\Delta t = \vec{r}_i\,(t) + \Delta\vec{r}_i \tag{4}$$

Step 2 Update the list of particles in each computational cell to prepare for the collision routine.
Step 3 Compute the number of collisions to attempt in each computational cell. The numerical implementation of this step is described in Section 2.5.
Step 4 Perform the collisions according to the user-defined binary collision model.
Step 5 Sample the particle positions, velocities, internal energies, etc. that are required to return the desired macroscopic values.
Step 6 Go back to **Step 1**, where now $t = t + \Delta t$ and repeat again until the simulation end-time t_{end} has been reached.

2.4. Movement: Particle tracking

The particle tracking algorithm [27] on which dsmcFoam+ is based is an existing OpenFOAM functionality for the discrete motion of particles within a mesh, with the two important objectives of knowing when particles switch cells in the mesh and dealing with particles as they interact with boundaries. To facilitate this, a robust way of tracking particles within a mesh is required. Rather than applying $\Delta\vec{r}_i$ at once to a particle, the number of faces (of the mesh) that the particle potentially crosses are first calculated. The particle 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 the particle's motion. If the face is internal, then there is nothing additional to do, and the particle proceeds to its next position. However, if the face is part of a boundary, an action will be applied (midstream of its move step). For example, if the boundary is a specular wall, the particle's normal velocity component switches direction on contact with the boundary and the particle continues the rest of its trajectory within the domain. This has been found to be a robust algorithm that ensures particles never stray outside the domain; it also enables dsmcFoam+ to deal with various complex boundaries that would not be possible otherwise.

2.5. Binary collisions

After all of the particles have moved and been tracked to their new positions in the mesh, it is necessary to re-index them before beginning the collision routine, because the collision and sampling routines depend on information about each cell's current occupancy. Collisions are then performed in a probabilistic manner, which sets DSMC apart from deterministic techniques such as Molecular Dynamics. There are various methods for ensuring that the correct number of collisions are performed to remain consistent with analytical theory [28,29], but the main method used in dsmcFoam+ is Bird's no-time-counter (NTC) scheme [1,30]. The sub-cell technique [1], or more modern transient-adaptive sub-cell techniques [31], are used to promote near-neighbour collisions. The probability P_{coll} of particle i colliding with particle j within a cell is

$$P_{coll}\left[i,j\right] = \frac{|\mathbf{c}_i - \mathbf{c}_j|}{\sum_{m=1}^{N}\sum_{n=1}^{m-1}|\mathbf{c}_m - \mathbf{c}_n|}, \tag{5}$$

where N is the instantaneous number of DSMC particles in the cell. This would be computationally expensive to calculate for each pair and so an acceptance–rejection scheme is used to choose which pairs collide. The NTC method is that

$$\frac{1}{2V_C}F_N N\left(N-1\right)\left(\sigma_T c_r\right)_{max}\Delta t \tag{6}$$

pairs are selected from a cell at a given time step, where V_C is the cell volume, $(\sigma_T c_r)_{max}$ is the maximum product of collision cross-section and relative speed of all possible particle pairs in the cell, Δt is the time step size, and F_N is the number of real atoms or molecules that each DSMC particle represents. Particle i is chosen at random from all particles in the cell and j is chosen from the same sub-cell to ensure near-neighbour collisions. Each collision pair ij is then tested using the acceptance–rejection method [1,32], i.e. the collision is accepted if

$$\frac{\left(\sigma_T c_r\right)_{ij}}{\left(\sigma_T c_r\right)_{max}} > R_f, \tag{7}$$

where R_f is a random number uniformly chosen in [0, 1].

Once a particle pair has been selected for collision, the particles must be 'collided'. If a chemical reaction model is to be used, the selected particles are tested for possible reactions at this stage. In dsmcFoam+, the quantum-kinetic (QK) chemical reaction framework is implemented, allowing for dissociation, exchange, and ionisation reactions to take place. Details of this model and its implementation can be found in Refs. [12,33].

Collisions are simulated by resetting the velocities of both partners in the collision pair; their positions are not altered. First, consider elastic collisions, which occur between particles that are not exchanging rotational or vibrational energy. Linear momentum is conserved by ensuring that the centre of mass velocity, $\mathbf{c}_{cm}$, remains constant, i.e.

$$\mathbf{c}_{cm} = \frac{m_i\mathbf{c}_i + m_j\mathbf{c}_j}{m_i + m_j} = \frac{m_i\mathbf{c}_i^* + m_j\mathbf{c}_j^*}{m_i + m_j} = \mathbf{c}_{cm}^*, \tag{8}$$

and energy is conserved by keeping the magnitude of the relative velocity constant, i.e.

$$c_r = |\mathbf{c}_i - \mathbf{c}_j| = |\mathbf{c}_i^* - \mathbf{c}_j^*| = c_r^*, \tag{9}$$

where the superscript $*$ denotes post-collision properties. Using Eqs. (8) and (9), along with an equation for the scattering angles, it is possible to solve for $\mathbf{c}_r^*$. In a variable hard sphere gas, the scattering angles θ and ϕ are uniformly distributed over a unit sphere. The azimuthal angle ϕ is uniformly distributed between 0 and 2π,

$$\phi = 2\pi R_f. \tag{10}$$

The elevation angle θ is uniformly distributed in the interval $[-1, 1]$ and calculated through

$$\cos\theta = 2R_f - 1 \qquad \text{and} \qquad \sin\theta = \sqrt{1-\cos^2\theta}. \tag{11}$$

The three components of the post-collision relative velocity are defined as

$$\mathbf{c}_r^* = c_r^*\left[\left(\cos\theta\right)\hat{\mathbf{x}} + \left(\sin\theta\cos\phi\right)\hat{\mathbf{y}} + \left(\sin\theta\sin\phi\right)\hat{\mathbf{z}}\right], \tag{12}$$

and the post-collision velocities become

$$\begin{aligned}\mathbf{c}_i^* &= \mathbf{c}_{cm}^* + \left(\frac{m_j}{m_i+m_j}\right)\mathbf{c}_r^*,\\ \mathbf{c}_j^* &= \mathbf{c}_{cm}^* - \left(\frac{m_i}{m_i+m_j}\right)\mathbf{c}_r^*.\end{aligned} \tag{13}$$

In the case of diatomic molecules with rotational energy, inelastic collisions must take place in order to exchange energy between the translational and rotational modes. In DSMC, the most common method of exchanging rotational energy is the phenomenological Larsen–Borgnakke model [34]. To capture a realistic rate of rotational relaxation, this model only treats a certain fraction of collisions as inelastic. In dsmcFoam+, a constant rotational relaxation probability Z_{rot} is used, which the user chooses by setting the *rotationalRelaxationCollisionNumber* number field in the *[case]/constant/dsmcProperties* file. When a collision is to take place, it is first checked for rotational relaxation, which is accepted if

$$\frac{1}{Z_{rot}} > R_f, \tag{14}$$

and the particle is assigned a new rotational energy. In order to conserve energy, the total translational energy available to the particle pair is decreased accordingly and a new post-collision relative speed c_r^* is calculated as

$$c_r^* = \sqrt{\frac{2\varepsilon_{tr}}{m_r}}, \tag{15}$$

where ε_{tr} is the total translational energy available to the collision pair after the adjustment for rotational relaxation has taken place, and m_r is the reduced mass of the collision pair. The remainder of the collision process proceeds as from Eq. (10).

dsmcFoam+ uses the quantum Larsen–Borgnakke model to exchange energy between the vibrational and electronic modes to the translational mode. Details of the Larsen–Borgnakke procedures for the vibrational and electronic modes can be found in [35] and [33], respectively.

2.6. Sampling

In most cases, the aim of any DSMC simulation is to recover macroscopic gas flow properties. In order to do so, it is necessary to sample the particle properties after the collisions have been processed, and then use these time-averaged particle properties to calculate the macroscopic fields. For example, the number density in a computational cell is calculated as

$$n = \frac{F_N \overline{N}}{V_C}, \tag{16}$$

where $\overline{N}$ is the time-averaged number of DSMC particles in the cell during the measurement interval, and V_C is the cell volume. A comprehensive overview of the measurements required to return many macroscopic fields can be found in Bird's 2013 monograph [26].

If the flow is steady, a simulation is allowed to reach its steady state and then properties are measured over a large enough sample size to reduce the statistical error to an acceptable level, which can be estimated using the relationships given by Hadjiconstantinou et al. [36]. For a transient problem, the simulation must be repeated enough times to provide a large enough sample, and the results can then be presented as an ensemble average.

2.7. Software implementation

Problems solved by applications like dsmcFoam+ tend to fall under the category of *N-body* and are often described as "embarrassingly parallel" in nature. However in the case of a complex DSMC problem, the best software implementation is not always clear. The primary reason for this is that while many problems have very little data that need to be stored per discrete body in the system (e.g. particle position), the amount of data needed per particle for a DSMC simulation can be far greater (e.g. velocity, rotational energy, vibrational energy, electronic energy, etc.). More importantly, it is often necessary for some DSMC particles in a system to store different sets of properties to others (e.g. atoms do not need to store rotational and vibrational energy, but molecules do); in other words, DSMC 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 DSMC scenarios that new particles can be added or removed from a simulation during the run-time of a problem, meaning the data storage requirements of the application can vary significantly during the course of a problem's evolution. The end result is that the most flexible and easily considered software implementation uses a linked-list type of data structure to create an array-of-structures (or AoS). This provides an easily expanded or contracted data structure and allows each structure stored to be customised for each body in the system. However, this solution may also provide 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 (or SoA) were used.

In dsmcFoam+ the majority of the data stored in memory is encapsulated in a single *dsmcCloud* class, which is an extension of a base OpenFOAM Lagrangian class, *Cloud*. When instantiated, the *dsmcCloud* class creates a doubly-linked list that stores pointers to instantiations of a mutable class known as the *parcel* (as in a collection of particles) class, which itself is an extension of the base OpenFOAM *Particle* class. Although this design provides flexibility and ease of concept, it does not guarantee memory contiguity, which is the primary source of the application's cache misses.

It is not entirely clear whether it is better to sacrifice run-time performance to enable easier programming by using an AoS design (the current situation) or whether design complexity should be increased to reduce run-time by converting this to SoA. This is a key area for the software's future development, however it may be that hybrid approaches seen in other *N-body* application areas [37] provide the most suitable balance between usability and computational performance for future versions of the code. However it is worth noting that this may then mean deviating from building upon core OpenFOAM libraries, as the double-linked list AoS design is inherent to those.

2.8. Parallel processing

dsmcFoam+ provides an MPI based domain-decomposition method for performing parallel DSMC simulations. The mesh that represents the entire domain to be simulated is first split up, or decomposed, into as many sub-parts as there are processor cores to be executed upon. Domain-decomposition is well-known and fundamental to OpenFOAM application 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 on one CPU, scalability of the code tends to be both problem-specific and limited to the order of hundreds of MPI ranks in many cases. This limitation is primarily due to the way MPI parallelism is implemented within OpenFOAM. Improving parallelism within dsmcFoam+ is an important future task as high performance computing (HPC) heads towards a fine-grained parallelism model. Currently, however, the parallel performance of dsmcFoam+ is sufficient to make it a useful tool for simulating complex DSMC problems in the order of a few tens of millions of particles over reasonable time periods.

The major challenges in carrying out a large-scale dsmcFoam+ simulation revolve around mesh generation and particle initialisation. To create large meshes with OpenFOAM, a successful strategy is to start with a coarse mesh using the blockMesh utility and then create

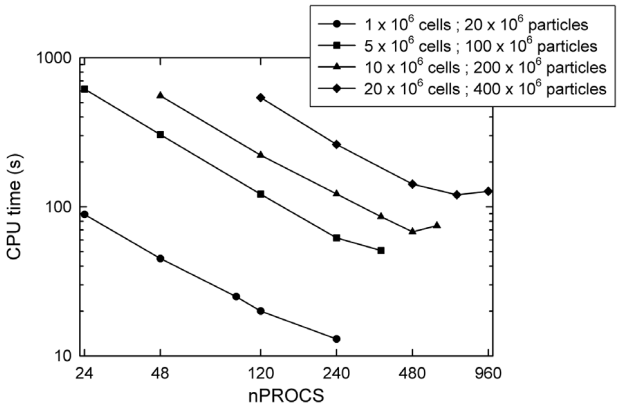


Fig. 3. Parallel performance of dsmcFoam+ for a lid-driven cavity case, showing the parallel speed-up for both strong and weak scaling.

the final mesh using the snappyHexMesh tool to refine the mesh in parallel. Other mesh refinement strategies, such as refineMesh could also possibly be utilised, but ultimately they need to be run in parallel across multiple nodes to have access to enough memory. Even more important than mesh generation is the particle initialisation phase. For particle numbers in the order of several hundred millions and more, initialisation must be performed in parallel using X processors with a *mpirun -np X dsmcInitialise –parallel* command. For this to work, domain decomposition using the decomposePar executable has to be performed before the case is populated with particles using the dsmcInitialise executable.

The parallel performance and scalability of the dsmcFoam+ code has been assessed on the ARCHER high performance computing system [38]. ARCHER is the UK's national supercomputing service based around a Cray XC30 supercomputer, and was rated number 61 in the top 500 list of supercomputers published in November 2016. ARCHER compute nodes contain two 2.7 GHz, twelve core ES-2697 V2 (Ivy Bridge) processors, and have 64 GB of memory shared between the two processors. A weak scalability study has been carried out using a lid-driven cavity test case, considering 4 different cases of grid-size and particle numbers, i.e. 1 million cells with 20 million particles; 5 million cells with 100 million particles; 10 million cells with 200 million particles; and 20 million cells with 400 million particles. The Scotch domain decomposition method [39] is followed for this scalability study, in which all domains have approximately the same number of cells. The variation of wall-clock time taken for all cases is shown in Fig. 3 for up to 960 cores, n_{PROCS}. The code performs reasonably well for both increase in load and increase in number of MPI tasks for all the cases considered. The computational time is measured using the in-built OpenFOAM timing functionality and is the total time for the simulation to run to completion (this study was performed with a slightly older version of dsmcFoam+).

Run-time load balancing has also been introduced to dsmcFoam+, which the user controls in the *[case]/system/balanceParDict* and *[case]/system/loadBalanceDict* files. This enables the code to measure the maximum current level of parallel load imbalance, L_{max}. Since the move function (which performs the particle movements described in Section 2.4) carries the largest computational overhead in dsmcFoam+, it is most efficient to ensure that each core has roughly the same number of particles N_{proc} to balance their loads. The ideal number of particles on each core N_{ideal} is then N/n_{PROCS}, where N is the total number of DSMC particles in the simulation and n_{PROCS} is the number of processors the simulation is running on. At each write interval, the number of particles on each processor N_{proc} is compared to N_{ideal}. If any of these fall outside a user-set tolerance (e.g. $0.9N_{ideal} < N_{proc} < 1.1N_{ideal}$) then the simulation is temporarily halted while the domain decomposition is performed again to re-balance the load. Following this, the simulation automatically starts again from where it was stopped, using a bash script as shown below, where the user changes the final time directory (0.0004 in this example) to match that in their *[case]/system/controlDict* file:

```
#!/bin/sh
cd ${0%/*} || exit 1    # run from this directory

while :
do
    ### Check that the final time directory does not exist ###
    if [ ! -d "processor0/0.0004" ]
    then
        echo "Directory processor0/0.0004 DOES NOT exist, "\
             "restart from the latest time."
        mpirun -np 8 dsmcFoamPlus -parallel
    else
        echo "Directory processor0/0.0004 DOES exist, "\
             "killing the script."
        break ### exit the loop
    fi
done

exit 0
```

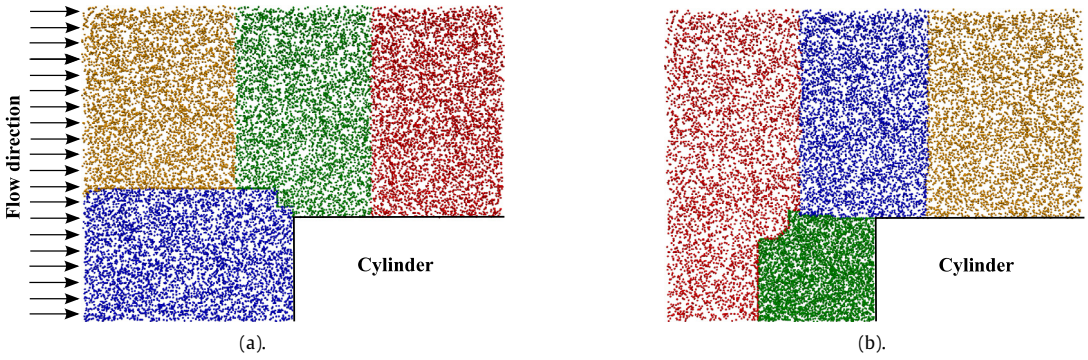


Fig. 4. Final particle distributions in the cylinder case, coloured by processor number. (a) Simulation without load balancing, and (b) simulation with load balancing.

Bird's case of axially-symmetric flow past a blunt-nosed cylinder (see [1], page 374) was used to demonstrate the effectiveness of the dynamic load balancing. This problem considers a flow of argon with a number density of 1×10^{21} m^{-3}, a temperature of 100 K, and a velocity of 1000 m/s (i.e. Mach number of 5.37). The cylinder has a radius of 0.1 m and the domain extends 0.2 m upstream and downstream of the cylinder's flat face, and 0.3 m radially. A constant DSMC cell size of 2.5×10^{-4} m in the axial and radial directions was used, with a wedge angle of 5° in the symmetry direction. The cylinder surface was set at a constant temperature of 300 K and fully diffuse reflections were modelled. The time step was 8×10^{-8} and each DSMC particle represented 1×10^{10} argon atoms. The radial weighting factor of 1000.

First, Scotch decomposition [39] was considered and the simulation performed on 4 cores of a desktop workstation equipped with a dual threaded quad-core Intel® Core™ i7–4770 CPU for 5000 time steps. Steady state was achieved after 1500 time steps, with around 2.9 million DSMC simulator particles in the domain. During the run, dsmcFoam+ reported a maximum load imbalance of 44%, but load balancing was not activated; 3:59:15 hrs:mins:secs were required to complete the simulation. The simulation was then repeated, this time allowing a maximum imbalance of 5% before the load was rebalanced. This resulted in a reduced run time, including the time required to recompose and re-decompose the simulation, of 2:46:25 hrs:mins:secs. This is a speed-up of 30%, demonstrating the effectiveness of this technique for reducing run times on simulations involving density gradients that evolve as the simulation proceeds. Fig. 4 compares the particle distributions, coloured by the processor that they belonged to when the simulation was complete, between the cases with and without load balancing. It is clear that the load balancing code has identified the stagnation region in front of the cylinder and concentrated processor power at this location, which has helped to reduce the computational time required to complete the simulation.

2.9. Extensible design

An important feature of dsmcFoam+ is its inherent ability to be extended. This capability was one of the key reasons OpenFOAM was originally selected as the base development framework. dsmcFoam+ 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. The result is that the typical design for all of the application'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.

Should it be desirable to add extra functionality to dsmcFoam+ then all that is required is to copy an appropriate specialised class, give it a new name, update the functionality, and then add the new class to the compilation list as per any other OpenFOAM addition. Once dsmcFoam+ 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 DSMC codes may strive to provide this level of extensibility, the strict adherence of OpenFOAM (and therefore dsmcFoam+) to a fully object-oriented programming model means that any new addition made will perform as well as the code it is built upon. It also ensures a level of standardisation in any new addition and simplifies the process of adding new functionality.

The extensibility of dsmcFoam+ is also notable in the way cases are defined. Because the standard OpenFOAM case model is adopted, cases are specified as a series of input files, known as "dictionaries". These are plain text files that adopt a standardised format in which new variables can be defined by adding new text and a value associated with the name. Dictionary parsing is done within dsmcFoam+ using the basic OpenFOAM provided 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).

3. Downloading and installing dsmcFoam+

This article does not provide detailed instructions to download and build dsmcFoam+, as this process does not differ from building a standard implementation of OpenFOAM. OpenFOAM, along with the latest dsmcFoam+ software can be downloaded through the associated CPC library entry or directly from a Git repository [5] for the latest version. Once downloaded, building the software for a chosen platform is best achieved by following the detailed instructions included within the *doc/MicroNanoFlows* folder within the main repository directory [5], referring back to platform-specific instructions for the current build of OpenFOAM that dsmcFoam+ is released alongside.

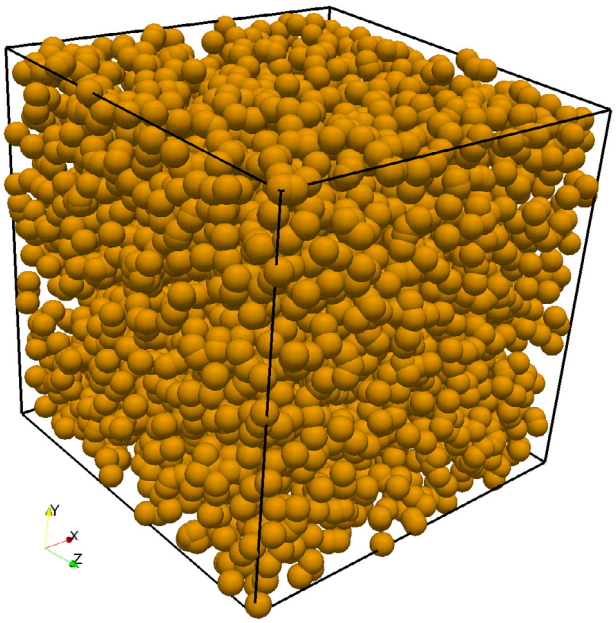


Fig. 5. DSMC simulation of argon in a cube of side length L, with specular walls applied at all boundaries.

4. Using dsmcFoam+

For those familiar with using an application within the OpenFOAM suite, usage of dsmcFoam+ will be relatively straightforward. However, in order to provide an illustrative example, the following section describes the process of defining, initialising, running (both in serial and parallel), and finally post-processing a simple case, consisting of a periodic cubic domain filled with argon at a temperature of 300 K and a molecular number density of 3×10^{20} m^{-3}.

This DSMC simulation is a canonical *NVT* ensemble, which means that it has a constant number of particles N, a constant volume V, and a constant temperature T. The case can be seen in Fig. 5; the domain consists of a periodic cubic box of side length $L = 0.6$ m.

4.1. Case definition

Using dsmcFoam+ begins by creating a new DSMC case. As with other OpenFOAM applications, a dsmcFoam+ case has a typical file structure, therefore it is defined by first creating a new folder of 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 which control most of the running parameters for the DSMC case (i.e. time-step control, case initialisation parameters, boundary conditions, etc.), while the latter contains details of the physical domain that is used to create and populate the underlying OpenFOAM mesh with particles. While there are some files within this structure that are specific to dsmcFoam+, they are named using typical nomenclature and are designed to be self-descriptive.

Alongside the example case described here, tutorial cases are also supplied within the software repository and can be found within the *tutorials/discreteMethods/dsmcFoamPlus* folder.

4.1.1. Mesh creation

Simulations performed using dsmcFoam+ are defined in a three-dimensional Cartesian co-ordinate system and are most commonly processed using the blockMesh application (which is a base application within the OpenFOAM suite). While it is entirely possible, and sometimes desirable for complex three-dimensional geometries, to use one of the more advanced meshing tools that OpenFOAM provides, it is often sufficient to use blockMesh; therefore this will be described here.

The mesh is then used by a dsmcFoam+ simulation for a number of purposes: (a) to initially place particles; (b) to provide a cell-list algorithm for performing binary particle collisions; (c) to decompose a domain for parallel execution; and (d) to resolve macroscopic field properties from the underlying DSMC simulation.

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

```
/*--------------------------------------------------------------------------*\
| =========                 |                                                |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox          |
|  \\    /   O peration     | Version:  2.4.0-MNF                            |
|   \\  /    A nd           | Web:      http://www.openfoam.org              |
|    \\/     M anipulation  |                                                |
\*--------------------------------------------------------------------------*/

FoamFile
 {
     version         2.4;
     format          ascii;

     root            "";
     case            "";
     instance        "";
     local           "";

     class           dictionary;
     object          blockMeshDict;
 }

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 1;

vertices
(
     (0 0 0)
     (0.6 0 0)
     (0.6 0.6 0)
     (0 0.6 0)
     (0 0 0.6)
     (0.6 0 0.6)
     (0.6 0.6 0.6)
     (0 0.6 0.6)
);

blocks
(
     hex (0 1 2 3 4 5 6 7) dsmcZone (60 60 60) simpleGrading (1 1 1)
);

boundary
(
     walls
     {
         type wall;
         faces
         (
             (1 2 6 5)
             (0 4 7 3)
             (2 3 7 6)
             (0 1 5 4)
             (4 5 6 7)
             (0 3 2 1)
         );
     }
);

mergePatchPairs
( );

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

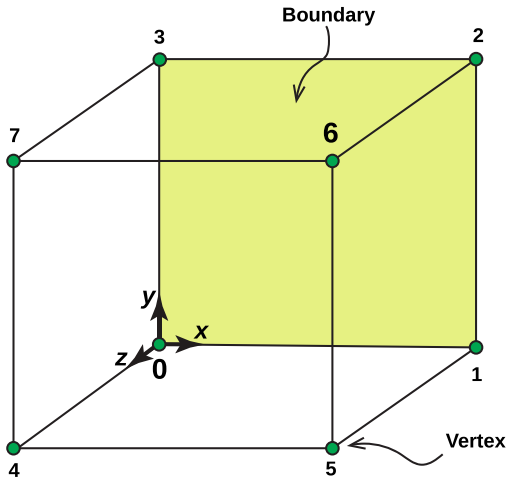


Fig. 6. Block structure of the cubic domain for the example case.

Referring to the example above, lines 1–24 are standard OpenFOAM dictionary header files. They 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 are all that is required. These vertices are numbered 0 to 7 and their physical locations are depicted in Fig. 6. The value **convertToMeters** on line 25 allows the domain to be scaled, but in this case no scaling is applied and the vertex locations are given in metres.

Blocks are defined within the **blocks** region; in this case, there is only one block that can be seen on lines 39–42, where the vertex order is first defined followed by the keyword **dsmcZone** and then **simpleGrading**. The entry **dsmcZone** defines the number of cells 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 a DSMC simulation that uses the cell-list algorithm, because ideally the cell size should be significantly smaller than the expected local molecular mean free path. The **simpleGrading** parameter allows cells to be scaled in size in one or more directions, which can be useful if there is an obvious compression region where the mean free path will be smaller, particularly in hypersonic flows.

The boundary entries shown on lines 44–58 indicate the *patches* that form the domain's boundaries. In this example of a periodic cube there is only one boundary, which is defined as a specular wall surface. Each patch contains the same two parameters:

1. **patchName**, which defines the type of the patch; this can be any viable OpenFOAM input, however it is most common to use either *patch* (a generic boundary), or *wall*;
2. the **faces** of a patch are defined by the vertices, e.g. here, the first face of the cubic domain is defined as being created by connecting vertex numbers one, two, six, and five, where the vertex numbering starts at zero.

Once the boundaries have been defined geometrically, it is also necessary to link the boundaries to the correct DSMC functionality, this is done within the file *[case]/system/boundariesDict* and for this example:

```
18  dsmcPatchBoundaries
19  (
20      boundary
21      {
22          patchBoundaryProperties
23          {
24              patchName   walls;
25          }
26
27          boundaryModel   dsmcSpecularWallPatch;
28
29          dsmcSpecularWallPatchProperties
30          {
31          }
32      }
33  );
34
35  dsmcCyclicBoundaries
36  (
37  );
38
39  dsmcGeneralBoundaries
```

```
40  (
41  );
42
43  // ************************************************************************* //
```

Each boundary has its parameters defined: **boundaryModel**, which specifies the type of model dsmcFoam+ will use, in this case the *dsmcSpecularWallPatch* class has been selected. Other models exist within the application, such as *dsmcDiffuseWallPatch* class, and new ones can be added by extending its capabilities. The **patchName** links the names of the patches within the *blockMeshDict* with the DSMC boundary models chosen.

Once these files are defined, the mesh and boundary data can be generated by running the OpenFOAM application blockMesh in the base of the *[case]* folder.

4.1.2. Particle properties

Constant particle properties can be set for each gas species involved in a simulation; these are defined in the dictionary file located at *[case]/constant/dsmcProperties*. An example of this file is:

```
19  // General Properties
20  // ~~~~~~~~~~~~~~~~~~
21
22
23  nEquivalentParticles            6.48e9;
24
25  // Axisymmetric Properties
26  // ~~~~~~~~~~~~~~~~~~
27
28  axisymmetricSimulation      false;
29  radialExtentOfDomain        0.03;
30  maxRadialWeightingFactor    1000.0;
31
32
33  // Binary Collision Model
34  // ~~~~~~~~~~~~~~~~~~~~~~
35
36  BinaryCollisionModel            LarsenBorgnakkeVariableHardSphere;
37
38  LarsenBorgnakkeVariableHardSphereCoeffs
39  {
40      Tref                                    273;
41      rotationalRelaxationCollisionNumber     5.0;
42      electronicRelaxationCollisionNumber     500.0;
43  }
44
45  // Collision Partner Selection Model
46  // ~~~~~~~~~~~~~~~~~~~~~~
47
48  collisionPartnerSelectionModel          noTimeCounter;
49
50
51  // Molecular species
52  // ~~~~~~~~~~~~~~~~~
53
54  typeIdList                      (Ar);
55
56  moleculeProperties
57  {
58      Ar
59      {
60          mass                                    66.3e-27;
61          diameter                                4.17e-10;
62          rotationalDegreesOfFreedom              0;
63          vibrationalModes                        0;
64          omega                                   0.81;
65          alpha                                   1.0;
66          characteristicVibrationalTemperature    (0);
67          dissociationTemperature                 (0);
68          ionisationTemperature                   0;
69          charDissQuantumLevel                    (0);
```

```
70          Zref                                (0);
71          referenceTempForZref                (0);
72          numberOfElectronicLevels            1;
73          degeneracyList                      (1);
74          electronicEnergyList                (0);
75      }
76  }
```

Line 23 defines how many real gas atoms or molecules each DSMC particle represents. Properties for axi-symmmetric simulations are set on lines 28–30, but this example does not utilise this feature, hence line 28 is set `false`. The binary collision model to be used when two particles are accepted for collision is defined on line 36, with the user-defined properties for this model set on lines 38–43. The model to decide how many collisions to attempt in each computational cell is defined on line 48; in this case, Bird's no-time-counter model is used but others, such as Stefanov's simplified Bernoulli trials [29], are also available in dsmcFoam+.

Line 54 shows the definition of a new species of molecule in the form of a **typeIdList**. Each species should be given a unique name within this list (in this case there is only one, for argon). The index of each unique name within this list is used within dsmcFoam+ as a unique identifying number in order to keep track of different species types, in this case *Ar* is given an ID of 0.

The physical properties of all species of molecule are defined between lines 56–76. Note, the properties of each species must be defined in the order they are listed in the **typeIdList**.

4.1.3. Time control

The correct choice of time-step is fundamental to achieving a reliable DSMC simulation. A time-step that is too small will make the simulation too computationally expensive, while a time-step that is too large may cause an unrealistic transfer of mass, momentum, and energy during the collision routine, and cause particles to travel an unreasonable distance during the move function.

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     dsmcFoamPlus;
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         0.001; //Stops simulation when time has reached this value
26
27  deltaT          1e-06; //Size of DSMC time-step
28
29  writeControl    runTime;
30
31  writeInterval   1e-03; //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  10; //Floating point precision to write
38
39  writeCompression uncompressed; //Files can be compressed upon write
40
41  timeFormat      general;
42
43  timePrecision   6;
44
45  runTimeModifiable yes;
46
47  adjustTimeStep  no;
48
49  nTerminalOutputs 10;
```

While the majority of this file is self-explanatory, there are a few important points to note. The **nTerminalOutputs** option is unique to dsmcFoam+ and allows the user to set how often the output of the number of particles in the system, the number of collisions taking place, etc. is written to the terminal window. Writing this information to terminal every single iteration can take an appreciable amount of time, which can be reduced by only writing this information (in this case) every ten iterations. The majority of the time-control functionality offered by dsmcFoam+ is built upon that provided by OpenFOAM, therefore capabilities like the ability to restart a simulation are inherent within the application. This functionality can be an important tool when considering multiple scenarios for a DSMC case as it enables an initial period to be simulated once (e.g. the flow approaches to an initial steady state) and then multiple alternative scenarios (e.g. with

and without chemical reactions) to be restarted following the same basic initial state. The nature of this functionality is best described in the OpenFOAM user-guide, but it involves setting the value of **startTime** equal to an existing time folder containing a valid set of DSMC data. 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 of **endTime**, however it is important to remember that a DSMC simulation has a significant computational cost. Therefore, while it is normally desirable for the simulated time to be as long as possible, it is sensible to minimise the value of **endTime** to as few time-steps as are needed for an acceptable level of statistical scatter to be achieved.

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 if possible as it will introduce significant processing overhead; 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 have a significant impact on the final size of the case directory, especially in a typical DSMC scenario where the simulation may be run for hundreds of thousands of time-steps. Often the important values obtained from the simulation are time-averages, and not exact particle 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 3 time-steps written remain on disk. For performance reasons, however, even when disk space is not an issue, the output frequency should still be controlled. Setting the value of **purgeWrite** equal to 0 will ensure all historical data will remain and the file-system overhead of deleting directories will not be incurred. An important additional point is that the value of **writeInterval** should be an exact integer factor of the time-step size **deltaT**.

A useful piece of functionality provided by dsmcFoam+ is the ability to modify the running state of an active DSMC simulation by editing the case's controlDict file, so 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 required (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.

4.2. Case initialisation

Once all of the dictionary files to define a new case are in place and the underlying mesh has been created, as per Section 4.1.1, it is then necessary to create an initial state for the DSMC simulation to evolve from. This is done using a pre-processing tool called dsmcInitialise, which, like dsmcFoam+ itself, is designed to be extensible to allow new initialisation algorithms to be implemented.

For the example case presented here, the goal is to insert N argon particles into the cubic domain in order to satisfy a target number density of $n = 3 \times 10^{20}$ m^{-3} atoms, target temperature of 300 K, and target macroscopic velocity of 0 m/s.

dsmcInitialise is controlled using a dictionary file located at *[case]/system/dsmcInitialiseDict*, the file for the example case is:

```
19  configurations
20  (
21      configuration
22      {
23          type                    dsmcMeshFill;
24
25          numberDensities
26          {
27              Ar            3e20;
28          };
29
30          translationalTemperature     300;
31          rotationalTemperature        0;
32          vibrationalTemperature       0;
33          electronicTemperature        0;
34          velocity        (0 0 0);
35      }
36  );
```

As the example presented only has one species (argon, Ar), there is only one configuration entry in the above file, which refers to the correct molecules according to the pre-defined value **typeID** (see Section 4.1.2). Should multiple species be involved, then one entry for each would be required. In this case the initialisation algorithm being used is the *dsmcMeshFill* class, which fills the entire computational domain with particles with properties that will, on average, recover the desired density, temperature, and velocity.

When initialising a case, the main rule to follow is to ensure that there are enough DSMC particles in each computational cell to enable accurate collision statistics to be recovered. For the no-time-counter method, this requires at least 20 particles per cell. In this example test case, many more particles than that are used, but the following calculation of the expected number of particles in each cell is generally used to help decide the value of **nEquivalentParticles** in Section 4.1.2 that should be set. In this example, the cell volume is uniform at 1×10^{-6} m^3. A molecule number density of 3×10^{20} m^{-3} and **nEquivalentParticles** of 6.48×10^9 gives a theoretical total of 46, 296.3 particles per cell. The dsmcInitialise tool will only create an integer number of particles, and uses a random fraction to decide if 46,296 or 46,297 particles will be created in each cell.

Once all case dictionaries are in place, the application is called from the base of the case directory; this will generate an initial time folder equal to the value of the variable **startTime** in the dictionary *[case]/system/controlDict*.

4.3. Running dsmcFoam+

Once all dictionary files have been created and the case initialised with the appropriate pre-processing tools, as described in Sections 4.1.1 and 4.2, it is then possible to start the DSMC calculations in serial straight away. As the name suggests, this will perform all execution using a single dsmcFoam+ process.

While a serial run may be suitable for a small DSMC simulation, it is usually preferable to make use of multiple processors wherever possible. As dsmcFoam+ utilises MPI based domain decomposition, this can be accomplished on a workstation with a few CPU cores or on a large distributed memory system over hundreds or even thousands of cores. In order to execute a dsmcFoam+ based 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 — some its own, others provided by external libraries. 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 contained with the initialisation folder created in Section 4.2.

Once the domain decomposition has been achieved, a parallel run of dsmcFoam+ 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 dsmcFoam+, like other OpenFOAM applications, must be informed that it is to run in parallel. This is done using a new switch passed through at the point of execution; an example of running 256 MPI ranks using a typical environment would be:

```
mpirun -np 256 dsmcFoamPlus -parallel
```

4.4. Post-processing results

One of the powerful features of building dsmcFoam+ from OpenFOAM is the post-processing and visualisation of results. By default, OpenFOAM provides a wrapper around the visualisation environment ParaView [19] (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 while data exists (i.e. it is possible to view an initialised DSMC case before any DSMC calculations are performed) by simply executing paraFoam in the base of the case folder. While it is beyond the scope of this document to provide full details of how to utilise ParaView to best effect, detailed instructions can be found within the documentation provided in the software release.

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.

If results have been generated by a parallel run of dsmcFoam+, then 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 was executed in the base of the case folder before reconstructPar was executed then this would be invalid).

4.5. Source code structure

As dsmcFoam+ 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 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 on the part of the reader in terms of how OpenFOAM organises its applications and classes.

As with all OpenFOAM applications, the source code for dsmcFoam+ 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, dsmcFoam+ can be found in *applications/discreteMethods/dsmc/dsmcFoamPlus* while associated processing applications can be found in *applications/utilities/preProcessing/dsmc*. Underlying classes can be found in *src/lagrangian/dsmc*.

Detailed documentation is also included with the repository, providing both installation instructions for a general workstation with a POSIX environment as well as in-depth technical detail. These can be found in the *doc* folder, located at *doc/MicroNanoFlows*.

Finally, tutorial example cases are also provided, these can be found within the *tutorials* folder, located at *tutorials/discreteMethods*.

5. Validation

In this section, four examples that validate dsmcFoam+ against analytical solutions and other DSMC codes are presented.

5.1. Collision rates

It is of fundamental importance to verify that the no-time-counter (NTC) implementation within dsmcFoam+ returns collision rates that are in agreement with analytical solutions. Here, two cases are tested in order to validate that dsmcFoam+ calculates the correct

Table 2
Number densities for the collision rate test simulations.

Case	n_{O_2} (m^{-3})	n_{N_2} (m^{-3})
I	1×10^{19}	0
II	1×10^{19}	1×10^{19}

Table 3
Molecular properties for the collision rate test simulations.

	O_2	N_2
T_{ref} (K)	273	273
d_{ref} (m)	4.07×10^{-10}	4.17×10^{-10}
m (kg)	53.12×10^{-27}	46.5×10^{-27}
ω	0.77	0.74

Table 4
Simulated collision rates compared with analytical values for an oxygen–nitrogen mixture.

Temperature (K)	Simulated/analytical rate	% Error
3,000	1.0000	0.0035
5,000	0.9998	0.0225
10,000	0.9995	0.0462
15,000	0.9994	0.0610
20,000	0.9993	0.0693
25,000	0.9995	0.0467
30,000	0.9995	0.0522

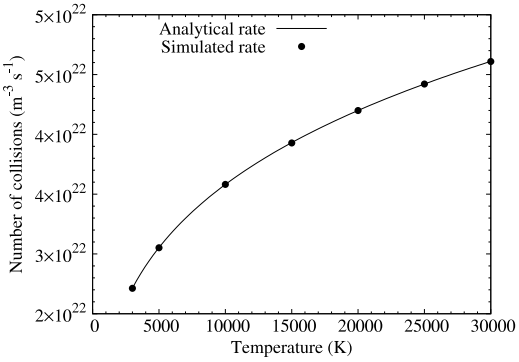


Fig. 7. Collision rate variation with temperature for Case I, pure oxygen. dsmcFoam+ results are compared to the analytical solution of Eq. (17).

collision rates. In the following simulations, the particles do not move and they do not collide; however, the probability of them colliding is still calculated and if the collision is 'accepted', a counter is updated, enabling collision rates to be measured for an equilibrium system with no gradients.

Both of the cases are of a single-cell simple adiabatic box. The cell is cubic with a volume of 1.97×10^{-5} m^3, and the time step employed for all cases is 1×10^{-7} s. Each DSMC simulator particle represents 2×10^8 real gas molecules. The molecule number densities and gas species (oxygen and nitrogen) for each case are outlined in Table 2, with the molecular properties being given in Table 3. Case I contains around 1 million DSMC particles and Case II has around 2 million. The number of collisions is counted for a total of 1000 time steps in all cases.

We compare the equilibrium collision rates $\left(N_{pq}\right)_0$ calculated by dsmcFoam+ to the exact analytical expression as given in Equation (4.78) of Bird [1], i.e.:

$$\left(N_{pq}\right)_0 = 2\pi^{\frac{1}{2}} (d_{\text{ref}})_{pq}^2 n_p n_q \{T/(T_{\text{ref}})_{pq}\}^{1-\omega_{pq}} \{2k_B(T_{\text{ref}})_{pq}/m_r\}^{\frac{1}{2}}, \tag{17}$$

where p and q represent different species and the subscript 0 denotes equilibrium quantities, d_{ref}, T_{ref} and ω are the variable hard sphere parameters for reference diameter, reference temperature and viscosity exponent, respectively, n is the number density, k_B is the Boltzmann constant, T is temperature, and m_r is reduced mass. Eq. (17) returns the number of collisions per unit volume per second, although it double counts because it calculates collisions between pq and qp; therefore, if a same species collision calculation is being performed it is necessary to include a factor of a half in order to achieve the correct rate.

Fig. 7 shows the analytical rate for Case I, pure oxygen, compared with the results from the dsmcFoam+ simulation. Excellent agreement is found between the simulated and analytical rates, indicating that the no-time-counter model implemented in dsmcFoam+ returns collision rates that are in agreement with theory for an equilibrium, single species gas.

Fig. 8 shows the simulated dsmcFoam+ results for Case II compared to the analytical solution for the gas mixture. Table 4 shows the percentage error in the measured simulation collision rate relative to the analytical solution, all of which are a fraction of a percent. The

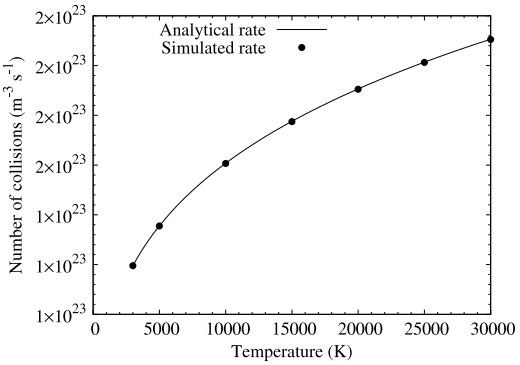


Fig. 8. Collision rate variation with temperature for Case II, 50% nitrogen, 50% oxygen. dsmcFoam+ results are compared to the analytical solution of Eq. (18).

analytical solution accounts for the three possible types of collisions in this gas mixture of nitrogen and oxygen, and is calculated as:

$$\begin{aligned}\left(N_{N_2O_2}\right)_0 = {} & \pi^{\frac{1}{2}}(d_{\text{ref}})^2_{O_2} n_{O_2} n_{O_2} \{T/(T_{\text{ref}})_{O_2}\}^{1-\omega_{O_2}} \{2k_B(T_{\text{ref}})_{O_2}/m_{r_{O_2}}\}^{\frac{1}{2}} \\ & + \pi^{\frac{1}{2}}(d_{\text{ref}})^2_{N_2} n_{N_2} n_{N_2} \{T/(T_{\text{ref}})_{N_2}\}^{1-\omega_{N_2}} \{2k_B(T_{\text{ref}})_{N_2}/m_{r_{N_2}}\}^{\frac{1}{2}} \\ & + \left[2\pi^{\frac{1}{2}}(d_{\text{ref}})^2_{N_2O_2} n_{N_2} n_{O_2} \{T/(T_{\text{ref}})_{N_2O_2}\}^{1-\omega_{N_2O_2}} \right. \\ & \left. \{2k_B(T_{\text{ref}})_{N_2O_2}/m_{r_{N_2O_2}}\}^{\frac{1}{2}}\right]. \end{aligned} \tag{18}$$

5.2. Free-molecular flow over a cylinder

In this section, dsmcFoam+ results for free-molecular flow past a stationary cylinder are compared to the corresponding analytical solutions. A range of flows are considered, varying from high subsonic to supersonic flow conditions. The DSMC simulations have been carried out in two-dimensions and all gas–surface interactions are assumed to be fully diffuse. A variable hard sphere model corresponding to the argon atom is used as the working gas. The boundary conditions at the inflow and outflow depend on the flow speed under consideration, and are implemented by injecting particles into the domain with a Maxwellian velocity distribution corresponding to the flow conditions at each external boundary [1]. For supersonic cases, the mean flow parameters of the injected particles at the inflow are set to correspond directly to the free-stream conditions. At supersonic outflow boundaries, "vacuum" conditions are imposed since no particles will have thermal velocities great enough to enter the computational domain from outside. For the subsonic cases, the particles injected at the inflow are based on both the free-stream velocity and density conditions whereas at the outflow boundaries, the particle properties are determined from the interior flow solution based on the characteristic boundary condition [40,41].

The drag coefficient, C_D, in the DSMC simulations is computed as

$$C_D = \frac{F_D}{\frac{1}{2}\rho A U_\infty^2}, \tag{19}$$

where F_D is the total drag force on the cylinder, ρ is the reference density, A is the projected area of the cylinder normal to the flow direction, and U_∞ is the free-stream velocity. The corresponding free-molecular analytical solution (for diffusive reflection) given by Schaaf and Chambré [42] is

$$C_{D,\text{diffuse}} = \frac{\sqrt{\pi}}{S} e^{-\left(S^2/2\right)} \left[\left(S^2 + \frac{3}{2}\right) I_0\left(\frac{S^2}{2}\right) + \left(S^2 + \frac{1}{2}\right) I_1\left(\frac{S^2}{2}\right)\right] + \frac{\pi^{3/2}}{4S_W}, \tag{20}$$

where I_0 and I_1 are the modified Bessel functions of the first kind, and S is the molecular-speed ratio. This can be expressed in terms of the Mach number Ma and the specific heat ratio γ by

$$S = \frac{U_\infty}{\sqrt{2RT_\infty}} = \sqrt{\frac{\gamma}{2}} Ma, \tag{21}$$

where R is the specific gas constant and T_∞ is the free-stream temperature, and S_W is the molecular speed-ratio calculated using the cylinder's wall temperature. In the present study, the wall temperature is assumed to be the isentropic stagnation temperature T_0 calculated as

$$T_0 = T_\infty \left(1 + \frac{\gamma - 1}{2} Ma^2\right). \tag{22}$$

The Knudsen number is based on the cylinder diameter, and the molecular mean free path is taken as the free-stream variable hard sphere value. $Kn \approx 50$ is considered the free-molecular limit, and molecular speed ratios in the range $0.4 < S < 3$ are investigated. Fig. 9 shows that the results for the cylinder drag coefficient from dsmcFoam+ are in excellent agreement with Eq. (20) for all of the flow speeds we considered.

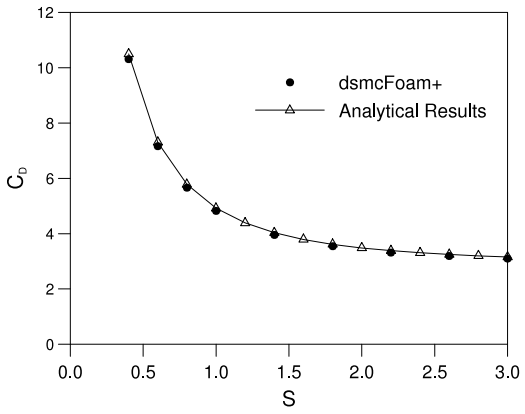


Fig. 9. Validation of the computed drag coefficient for flow past a stationary cylinder against the free-molecular analytical solution [42].

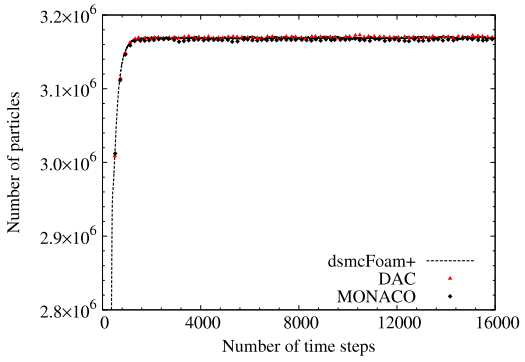


Fig. 10. Approach to steady-state in the flat-plate hypersonic flow simulation. Comparison of results from dsmcFoam+, DAC, and MONACO.

5.3. Hypersonic 3D flow over a flat plate

In this test case, results from dsmcFoam+ are compared to those from two well-established DSMC codes, DAC and MONACO. Padilla [2] has previously compared DAC and MONACO for a 3D flow over a flat plate, which was considered experimentally by Allègre et al. [43]. Here, the aim is to demonstrate that the measurement tools return high quality, time-averaged data that can be post-processed in a manner that suits multiple applications. This case is available as one of the DSMC tutorial simulations in the *~/OpenFOAM/OpenFOAM-2.4.0-MNF/tutorials/discreteMethods/dsmcFoamPlus/hypersonicFlatPlate* directory, once the repository [5] has been installed.

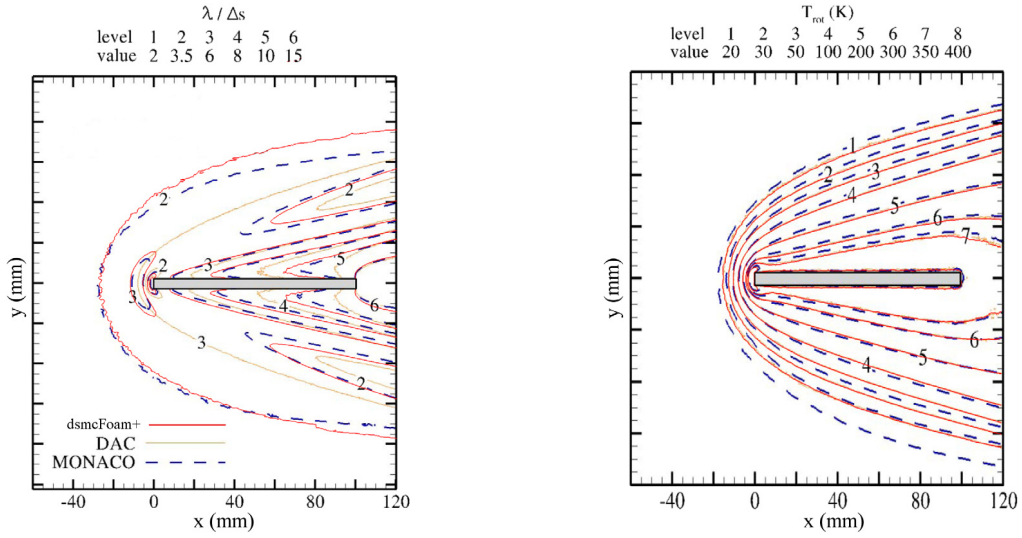
A flat plate, 100 mm in length, 5 mm in depth, 1 mm in width, is simulated within a domain that measures 180 × 205 × 1 mm. A structured mesh of cubic cells with an edge length Δs of 0.5 mm is used to ensure the dsmcFoam+ simulation is consistent with the previous numerical work. The *dsmcVolFields* measurement class is used to return macroscopic fields that can be viewed in Paraview, before being manipulated and having selections of data, e.g. properties on a specific surface, exported in a format for plotting. Particle properties are measured every second time step to remain consistent with the previous DAC and MONACO work.

The working gas is variable hard sphere nitrogen, with a reference diameter of 4.17×10^{-10} m and viscosity index ω of 0.74 at a reference temperature of 273 K. The free-stream velocity, temperature, and molecular number density are 1503 m/s, 13.32 K (i.e. Mach 20), and 3.716×10^{20} m^{-3}, respectively. The plate surface temperature is 290 K and all gas–surface interactions are modelled as diffuse reflections. Each DSMC simulator particle represents 4.645×10^9 real molecules and a time step of 3.102×10^{-7} s is used. Rotational energy exchange is modelled using the Larsen-Borgnakke approach [34], and dsmcFoam+ uses a constant relaxation probability of 0.2 (which is identical to DAC). Initially, there are no particles in the system, and the simulation is run for a total of 16,000 time steps.

Fig. 10 shows the number of particles in the system as a function of time step for all three codes — all show an almost identical approach to steady-state from the initial vacuum condition, and have the same number of particles at steady-state. dsmcFoam+ matches the DAC results slightly better because both codes use a constant rotational relaxation probability, while MONACO has this as a function of local temperature, leading to slightly different results.

After 8000 time steps, the time averaging is activated in dsmcFoam+. Another key feature of OpenFOAM is the run-time modifiable capability, meaning that while a simulation is running it is possible to simply turn the averaging on by changing all of *resetAtOutput* options to *off* in the *fieldPropertiesDict*. If this option is set to *on* the accumulated information is reset at each write interval, meaning that the displayed results are only averaged for the period covered during the last write interval.

Fig. 11(a) compares contours of a time-averaged macroscopic quantity, i.e. rotational temperature, obtained from dsmcFoam+, with results from DAC and MONACO. The dsmcFoam+ results are in better agreement with those from DAC, which is expected since both of these codes use a constant rate of rotational energy relaxation, whilst MONACO has this rate as a function of local temperature. Fig. 11(b) compares the ratio of the local mean free path to the cell size, i.e. $\lambda/\Delta s$, measured from dsmcFoam+, DAC, and MONACO. The dsmcFoam+ results are in closer agreement to those from MONACO for this property, although it is evident that the MONACO results are



(a) Contours of mean free path to cell size ratio.

(b) Contours of rotational temperature.

Fig. 11. DSMC simulations of hypersonic flow over a flat plate. Contours of ratio of the mean free path to cell size (left), and rotational temperature (right). Comparison of results from dsmcFoam+, DAC, and MONACO.

Table 5
Drag forces on the simulated flat plate; comparison of results from dsmcFoam+, DAC, MONACO.

	$D_{pressure}$ (N)	$D_{skin\ friction}$ (N)	D_{total} (N)
DAC	1.911×10^{-4}	2.472×10^{-4}	4.383×10^{-4}
MONACO	1.916×10^{-4}	2.461×10^{-4}	4.377×10^{-4}
dsmcFoam+	1.904×10^{-4}	2.477×10^{-4}	4.381×10^{-4}
error of dsmcFoam+ relative to DAC	0.366%	0.202%	0.046%
error of dsmcFoam+ relative to MONACO	0.626%	0.65%	0.091%

not symmetrical around the top and bottom of the flat plate (this is likely a display error in the original paper where the DAC and MONACO results were published, and not an issue with the MONACO code; it is a well-established and validated solver).

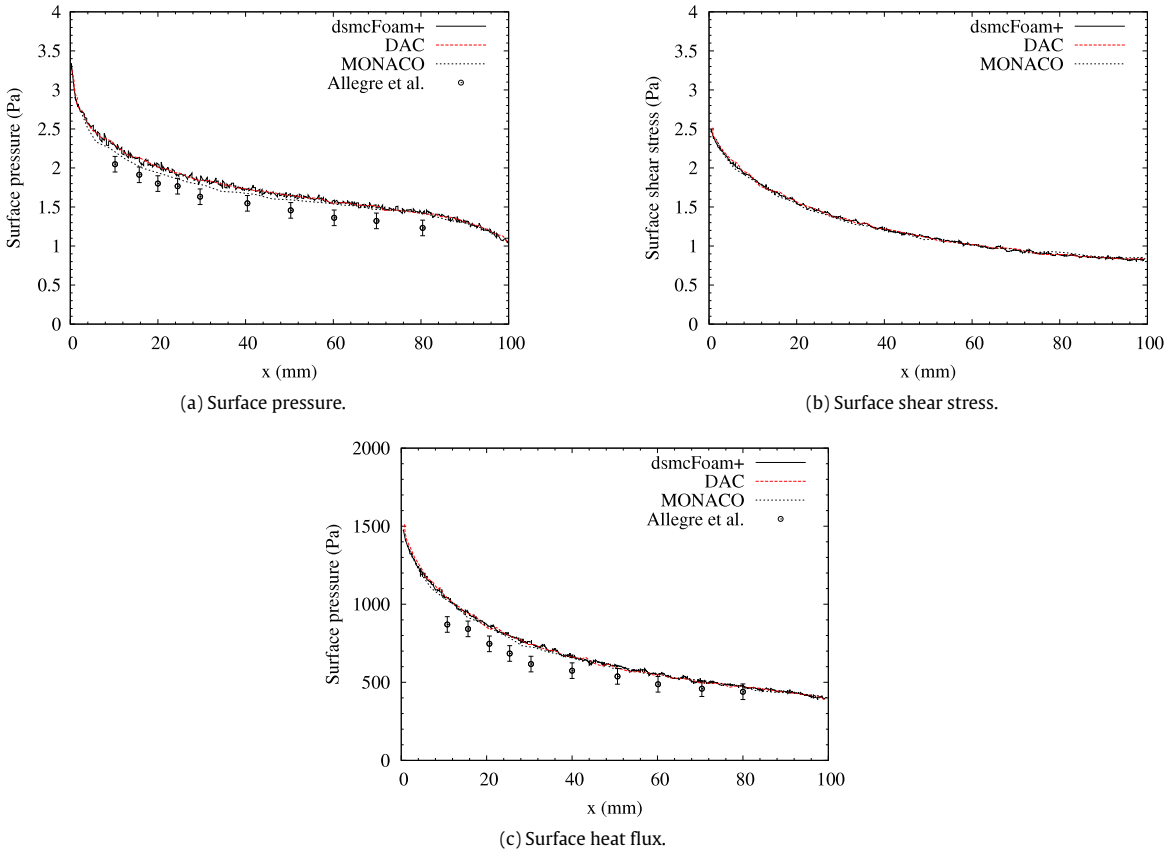
The *dsmcVolFields* measurement class is designed to detect the surface normal vector of each face on the boundaries and convert the measured force density into the pressure and shear stress components. Although a simple plate geometry has been used here, *dsmcVolFields* is implemented in a general manner such that the shear stress and pressure in complex geometries will also be calculated in a consistent manner. Fig. 12 shows the surface pressure, shear stress, and heat flux measured by the three DSMC codes, with experimental results also provided for the pressure and heat flux [43]. All of the numerical results are in excellent agreement with each other, although the pressure and heat flux are being slightly over-predicted (when compared to experiment) by all three codes.

Table 5 lists the measured drag forces, split into the pressure and skin friction components, predicted by each of the three DSMC codes. The skin friction drag is larger than the pressure drag because of the relatively large surface area of the plate in comparison to its frontal area. All three codes predict very similar forces on the plate, with all the individual force component results from dsmcFoam+ falling within 1% of those from both DAC and MONACO.

Padilla [2] also reports the computational time taken for this simulation, on a workstation with a 64 bit AMD Opteron™ Processor 244 processor having 1.8 GHz clock rate and 4 GB memory, to be 9:42:18 hrs:mins:secs and 8:45:10 hrs:mins:secs for DAC and MONACO, respectively. In comparison, the same simulation using dsmcFoam+ takes 24:35:46 hrs:mins:secs when performed on a desktop workstation with a quad-core Intel® Core™ i7–4770 CPU having 3.40 GHz clock rate and 19 GB of memory, having been compiled with GCC 4.8.4 using the 'O3' and 'march=native' optimisation flags. Code profiling of dsmcFoam+ using the Oprofile suite [44] has been performed for 200 time steps at the steady state condition. The results show that the simulation spends 72.96% of the computational time in functions related to the movement of particles, 7.75% on particle indexing, 6.88% performing measurements required to return the macroscopic fields, and 5.51% on collisions. The most expensive sub-routine reported is the *trackToFace* function at 44.77% of the total compute time; this is part of the move function and is responsible for moving particles between faces on the mesh. For this same problem, it has been reported [2] that DAC spends 35% of its computational time on the move function, and MONACO 48%. The relative burden of particle movement is significantly higher in dsmcFoam+, making it a clear area where performance enhancements should be sought in the future.

5.4. Microscale Poiseuille flow

Macroscopic fluxes due to particles crossing faces of the mesh can be measured with dsmcFoam+, and this is particularly useful for measuring mass flow rates. The particle tracking algorithm in OpenFOAM [27] is used to keep track of particles as they cross mesh faces during a time step. In OpenFOAM, two neighbour cells P and Q are connected by a common face f and the face normal vector $\mathbf{n}_f$ is always defined as being positive in the direction of owner cell P to neighbour cell Q, as illustrated in Fig. 13.



(a) Surface pressure.

(b) Surface shear stress.

(c) Surface heat flux.

Fig. 12. DSMC simulations of hypersonic flow over a flat plate. Plots of (a) pressure, (b) shear stress, and (c) heat flux along the top surface of the flat plate. Comparison of results from dsmcFoam+, DAC, MONACO, and experimental results where available [43].

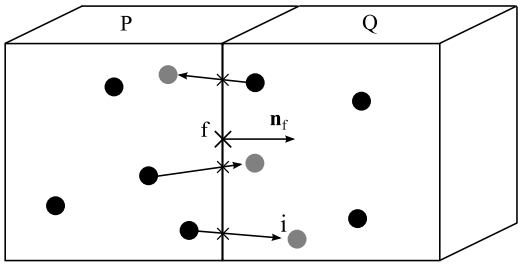


Fig. 13. Schematic for sampling fluxes across faces.

The direction that a particle i crosses face f can be defined as $sgn(\mathbf{c}_i \cdot \mathbf{n}_f)$, which equals 1 if $\mathbf{c}_i \cdot \mathbf{n}_f > 0$, or -1 if $\mathbf{c}_i \cdot \mathbf{n}_f < 0$. A particle that moves on the plane of face f will have $\mathbf{c}_i \cdot \mathbf{n}_f = 0$ and so $sgn(\mathbf{c}_i \cdot \mathbf{n}_f) = 0$ for this case. This is measured and summed for all particles crossing all faces at each time step. The average mass flow rate crossing face f is then defined as:

$$\langle \dot{m}_f \rangle = \frac{1}{t_{av}} \sum_{i}^{\Delta N_f(t \to t_n)} F_N m_i \, sgn(\mathbf{c}_i \cdot \mathbf{n}_f), \tag{23}$$

with

$$sgn\,(x) = \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 1, \end{cases} \tag{24}$$

where t_{av} is the physical time that the mass flow rate is averaged for, F_N is the number of real atoms or molecules that each DSMC particle represents, m_i is the molecular mass, and $\Delta N_f(t \to t_n)$ is the total number of computational particles that cross face f during the time period $t \to t + t_{av}$. The mass flux can then be calculated as $\langle \dot{m}_f \rangle / A_f$, where A_f is the area of face f. A group of faces can be joined and the mass flux measured over their entire area. Results are only written to disk for the user-defined faces that are selected.

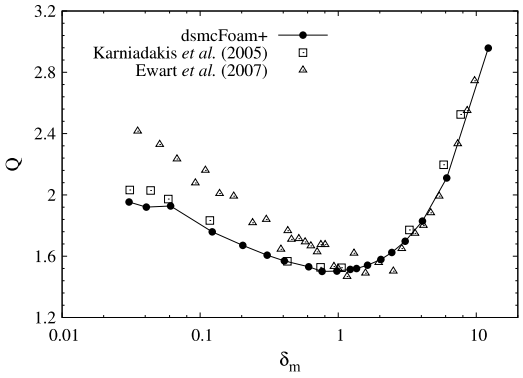


Fig. 14. Normalised mass flow rate in 2D planar Poiseuille flow, showing the Knudsen minimum phenomenon.

Currently, only a mass flux measurement tool has been implemented in dsmcFoam+, called *dsmcMassFluxSurface*, although the fluxal properties measurement framework could be extended to include momentum flux and energy flux measurement tools as future work, as described in Ref. [8] for molecular dynamics simulations.

In order to demonstrate the measurement of mass flux, a series of pressure-driven planar Poiseuille flows are simulated over a large range of Knudsen number. The mass flow rate $\dot{m}$ is measured and normalised as follows:

$$Q = \frac{\dot{m}L\sqrt{2RT}}{h^2 w\,(P_{in} - P_{out})}, \tag{25}$$

where L, h and w are the length, height and width of the planar flow geometry, respectively, and R, T, P_{in}, and P_{out} are the specific gas constant, the isothermal temperature of the gas in the channel, inlet pressure, and outlet pressure, respectively. The *dsmcMassFluxSurface* class is used in the *fieldPropertiesDict* to measure the mass flux. Since this counts a very large number of particles crossing a face, the major advantage of this measurement tool is that the statistical error in the measured mass fluxes are significantly smaller than if the mass flux was calculated in post-processing from the macroscopic values of velocity and density.

A rarefaction parameter δ_m is defined through the average of the inlet and outlet Knudsen numbers (based on the variable hard sphere mean free path and the channel height h) Kn_m in each case as:

$$\delta_m = \frac{\sqrt{\pi}}{2Kn_m}. \tag{26}$$

The inlet to outlet pressure ratio in all our cases is 3, and the aspect ratio of the planar Poiseuille geometries considered is 20.

The dsmcFoam+ results are compared to previous DSMC results [45] in Fig. 14. Experimental results from Ewart et al. [46] are also plotted for comparison. The DSMC results are in good agreement with one another, and agreement with the experimental data is excellent at low Kn and reasonable at high Kn. It has previously been noted [47] that the asymptotic value that Q obtains is proportional to $\ln(L/h)$; since the experimental work was performed on geometries with a very large aspect ratio ($L/h = 1000$), it is expected that the DSMC results for aspect ratio 20 will therefore not match exactly. Unfortunately, it is not possible to simulate an aspect ratio of 1000 using DSMC, as the velocities are too low for a converged solution to be obtained in a practical time scale. The famous Knudsen paradox [48] can clearly be observed, where the normalised mass flow rate has a minimum at around $Kn = 1$.

A pressure-driven micro-Poiseuille case is available as one of the DSMC tutorials in the *tutorials/discreteMethods/dsmcFoamPlus/microScaleTestCase* directory, once the repository [5] has been installed. This tutorial demonstrates how to set up mass flux measurements.

6. Conclusions and future work

dsmcFoam+ has been developed in order to investigate problems in rarefied gas dynamics using the direct simulation Monte Carlo method. It is designed entirely within the OpenFOAM software framework and is a highly-extensible and fully object-oriented C++ based approach. The code is released under the same license as the OpenFOAM base, at a publicly-available software repository [5] that includes documentation and example cases. dsmcFoam+ is parallelised using an MPI-based domain-decomposition approach built upon the parallel capability provided by OpenFOAM. To date, the code has been used to solve a wide variety of DSMC problems; it has been shown to be robust and reliable in its results, and typically scales well over a few hundred processors.

dsmcFoam+ is intended to be a useful research platform for those needing DSMC for their simulation and design work. It is straightforward to build new cases, with OpenFOAM's mesh based processing forming the basis of domain description. Extension capability is inherent to dsmcFoam+ and all of its associated utilities, and typically involves the creation of a single new C++ class; nearly all aspects of the DSMC functionality can be extended in this way. It is our hope that dsmcFoam+ will be used, and its capabilities extended by different research and industrial groups from different disciplines.

The development of dsmcFoam+ is ongoing by the authors, with chemical reactions involving charged species and further development of axisymmetric capability being the current priorities in improving the code. Optimisations and bug fixes are often applied to the repository; however, for the future, two major non-algorithmic developments need to be improving the code's serial and then parallel performance, first through memory design optimisation and then through hybridisation of the parallelisation strategies away from pure MPI.

Acknowledgements

Results were obtained using the EPSRC-funded ARCHIE-WeSt High Performance Computer (www.archie-west.ac.uk, EP/KO38427/1). The authors thank the UK's Engineering and Physical Sciences Research Council (EPSRC) for funding under grant nos. EP/K000586/1, EP/K038621/1, and EP/N016602/1. This work used the ARCHER UK National Supercomputing Service (http://www.archer.ac.uk).

References

[1] G.A. Bird, Molecular Gas Dynamics and the Direct Simulation of Gas Flows, Oxford Science Publications, Oxford University Press Inc, New York, 1994.
[2] J.F. Padilla, Comparison of DAC and MONACO DSMC Codes with Flat Plate Simulation, Tech. Rep. NASA/TM-2010–216835, NASA, 2010.
[3] SPARTA Direct Simulation Monte Carlo (DSMC) Simulator, Available online: http://sparta.sandia.gov/.
[4] H. Weller, G. Tabor, H. Jasak, C. Fureby, Comput. Phys. 12 (1998) 620–631.
[5] M. Borg, S. Longshaw, S. Ramisetti, J. Zhang, D. Lockerby, D. Emerson, J. Reese, mdFOAM plus OpenFOAM, 2017, https://github.com/MicroNanoFlows/OpenFOAM-2.4.0-MNF.
[6] T.J. Scanlon, E. Roohi, C. White, M. Darbandi, J.M. Reese, Comput. Fluids 39 (10) (2010) 2078–2089.
[7] G.B. Macpherson, Molecular Dynamics Simulation in Arbitrary Geometries for Nanoscale Fluid Mechanics (Ph.D. thesis), University of Strathclyde, Glasgow, 2008.
[8] M.K. Borg, Hybrid Molecular-Continuum Modelling of Nanoscale Flows (Ph.D. thesis), University of Strathclyde, Glasgow, 2010.
[9] G.B. Macpherson, M.K. Borg, J.M. Reese, Molecular Simulation 33 (15) (2007) 1199–1212.
[10] G.B. Macpherson, J.M. Reese, Mol. Simul. 34 (1) (2008) 97–115.
[11] M.K. Borg, G.B. Macpherson, J.M. Reese, Mol. Simul. 36 (10) (2010) 745–757.
[12] T.J. Scanlon, C. White, M.K. Borg, R.C. Palharini, E. Farbar, I.D. Boyd, J.M. Reese, R.E. Brown, AIAA J. 53 (2015) 1670–1680.
[13] C. White, M.K. Borg, T.J. Scanlon, J.M. Reese, Comput. & Fluids 71 (2013) 261–271.
[14] C. White, C. Colombo, T.J. Scanlon, C.R. McInnes, J.M. Reese, Adv. Space Res. 51 (11) (2013) 2112–2124.
[15] N. Dongari, C. White, T.J. Scanlon, Y. Zhang, J.M. Reese, Phys. Fluids 25 (5) (2013) 052003.
[16] R.C. Palharini, C. White, T.J. Scanlon, R.E. Brown, M.K. Borg, J.M. Reese, Comput. & Fluids 120 (2015) 140–157.
[17] C. White, T.J. Scanlon, R.E. Brown, J. Spacecr. Rockets 53 (1) (2016) 134–142.
[18] B. John, X.J. Gu, R.W. Barber, D.R. Emerson, AIAA J. 54 (2016) 1670–1681.
[19] J. Ahrens, B. Geveci, C. Law, C.D. Hansen, C.R. Johnson, ParaView: An End-User Tool for Large-Data Visualization, Academic Press, 2005.
[20] M. Gad-el-Hak, J. Fluids Eng. 121 (1) (1999) 5–33.
[21] J.C. Maxwell, Philos. Trans. R. Soc. Part 1 170 (1879) 231–256.
[22] M.V. Smoluchowski, Ann. Phy. Chem. 64 (1898) 101–130.
[23] A.A. Alexeenko, S.F. Gimelshein, D.A. Levin, J. Microelectromech. Syst. 14 (4) (2005) 847–856.
[24] A.J. Lofthouse, L.C. Scalabrin, I.D. Boyd, J. Thermophys. Heat Transfer 22 (1) (2008) 38–49.
[25] W. Wagner, J. Stat. Phys. 66 (1992) 1011–1044.
[26] G.A. Bird, The DSMC Method, CreateSpace Independent Publishing Platform, 2013.
[27] G.B. Macpherson, N. Nordin, H.G. Weller, Commun. Numer. Methods. Eng. 25 (3) (2009) 263–273.
[28] M.S. Ivanov, S.V. Rogasinsky, Russ. J. Numer. Anal. Math. Model. 3 (6) (1988) 453–465.
[29] S.K. Stefanov, SIAM J. Sci. Comput. 33 (2) (2011) 677–702.
[30] G.A. Bird, Progr. Astronaut. Aeronaut. 118 (1989) 211–226.
[31] C.C. Su, K.C. Tseng, H.M. Cave, J.S. Wu, Y.Y. Lian, T.C. Kuo, M.C. Jermy, Comput. & Fluids 39 (7) (2010) 1136–1145.
[32] A.L. Garcia, Numerical Methods for Physics, Prentice-Hall, Inc, Upper Saddle River, New Jersey, 2000.
[33] D. Liechty, Extension of a Kinetic Approach to Chemical Reactions to Electronic Energy Levels and Reactions Involving Charged Species with Application to DSMC Simulations (Ph.D. thesis), University of Maryland, 2013.
[34] C. Borgnakke, P.S. Larsen, J. Comput. Phys. 18 (4) (1975) 405–420.
[35] G.A. Bird, AIP Conf. Proc. 1084 (25) (2008) 245–250.
[36] N.G. Hadjiconstantinou, A.L. Garcia, M.Z. Bazant, G. He, J. Comput. Phys. 187 (1) (2003) 274–297.
[37] R. Vacondio, S. Longshaw, S. Siso, L. Mason, B. Rogers, Proc. 11th Smoothed Particle Hydrodynamics European Research Interest Community Conference, SPHERIC 2016, Vol. 11, 2016.
[38] ARCHER: the UK National Supercomputing Service, Available online: http://www.archer.ac.uk/.
[39] F. Pellegrini, J. Roman, Proceedings of High-Performance Computing and Networking: International Conference and Exhibition HPCN EUROPE, 1996, pp. 493–498.
[40] Q. Sun, I.D. Boyd, AIAA J. 42 (6) (2004) 1066–1072.
[41] C. Hirsch, Numerical Computation of Internal and External Flows-II, Wiley, West Sussex, England, UK, 1991.
[42] S.A. Schaaf, P.L. Chambré, Flow of Rarefied Gases, Princeton University Press, 1961.
[43] J. Allègre, A.C.M. Raffin, L. Gottesdiener, Progr. Astronaut. Aeronaut. 160 (1992) 285–295.
[44] J. Levon, P. Elie, Oprofile: A system profiler for linux, 2017, Available online: http://oprofile.sourceforge.net.
[45] G. Karniadakis, A. Beskok, N. Aluru, Microflows and Nanoflows: Fundamentals and Simulation, Springer Science+Business Media, Inc., New York, 2005.
[46] T. Ewart, P. Perrier, I.A. Graur, J.G. Méolans, J. Fluid Mech. 584 (2007) 337–356.
[47] N. Dongari, F. Durst, S. Chakraborty, Microfluidics Nanofluidics 9 (2010) 831–846.
[48] W. Steckelmacher, Rep. Progr. Phys. 49 (10) (1999) 1083–1107.