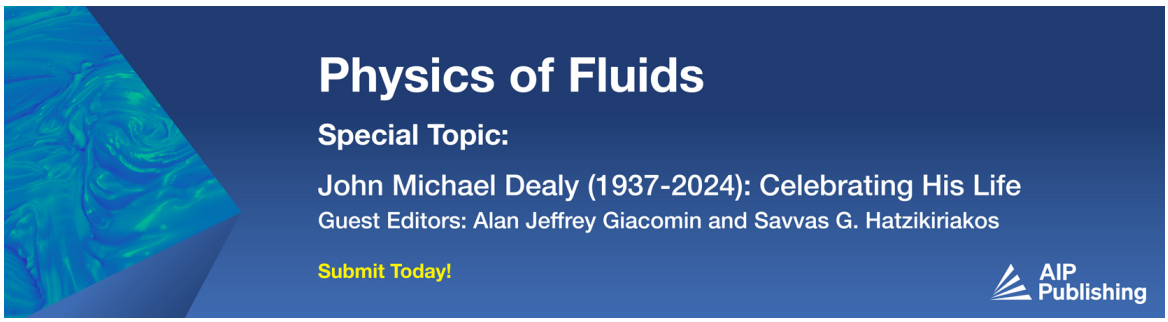
RESEARCH ARTICLE | AUGUST 01 2019

Direct simulation Monte Carlo on petaflop supercomputers and beyond

Special Collection: Direct Simulation Monte Carlo — The Legacy of Graeme A. Bird

S. J. Plimpton; S. G. Moore; A. Borner; A. K. Stagg ; T. P. Koehler; J. R. Torczynski ; M. A. Gallis

Physics of Fluids 31, 086101 (2019)
https://doi.org/10.1063/1.5108534



Physics of Fluids ARTICLE scitation.org/journal/phf

Direct simulation Monte Carlo on petaflop supercomputers and beyond

Cite as: Phys. Fluids 31, 086101 (2019); doi: 10.1063/1.5108534
Submitted: 30 April 2019 • Accepted: 2 July 2019 •
Published Online: 1 August 2019

S. J. Plimpton,[1,a)] S. G. Moore,[1] A. Borner,[2] A. K. Stagg,[1] T. P. Koehler,[1] J. R. Torczynski,[1] and M. A. Gallis[1,a)]

AFFILIATIONS

[1]Sandia National Laboratories, Albuquerque, New Mexico 87185, USA
[2]Science and Technology Corporation at NASA Ames Research Center, Moffett Field, California 94035, USA

Note: This paper is part of the special issue on Direct Simulation Monte Carlo—The Legacy of Graeme A. Bird.
[a)]**Electronic addresses:** sjplimp@sandia.gov and magalli@sandia.gov

ABSTRACT

The gold-standard definition of the Direct Simulation Monte Carlo (DSMC) method is given in the 1994 book by Bird [*Molecular Gas Dynamics and the Direct Simulation of Gas Flows* (Clarendon Press, Oxford, UK, 1994)], which refined his pioneering earlier papers in which he first formulated the method. In the intervening 25 years, DSMC has become the method of choice for modeling rarefied gas dynamics in a variety of scenarios. The chief barrier to applying DSMC to more dense or even continuum flows is its computational expense compared to continuum computational fluid dynamics methods. The dramatic (nearly billion-fold) increase in speed of the largest supercomputers over the last 30 years has thus been a key enabling factor in using DSMC to model a richer variety of flows, due to the method's inherent parallelism. We have developed the open-source SPARTA DSMC code with the goal of running DSMC efficiently on the largest machines, both current and future. It is largely an implementation of Bird's 1994 formulation. Here, we describe algorithms used in SPARTA to enable DSMC to operate in parallel at the scale of many billions of particles or grid cells, or with billions of surface elements. We give a few examples of the kinds of fundamental physics questions and engineering applications that DSMC can address at these scales.

I. INTRODUCTION

Bird invented the Direct Simulation Monte Carlo (DSMC) algorithm for modeling rarefied gas dynamics in a seminal series of papers and books.[5–7] Interestingly, his papers in the 1960s were followed in the 1970s by Monte Carlo algorithms based on related principles but developed independently and applied to very different systems. These include the stochastic simulation algorithm (SSA) of Gillespie for stochastic modeling of biochemical networks[31,32] and the kinetic Monte Carlo method known as the n-fold way or BKL algorithm (for its three authors) used in statistical physics.[10]

Algorithmically, the basic DSMC method has retained its canonical formulation since its introduction more than 50 years ago.[5,6] In his 1994 book,[7] Bird refined the original algorithm, introducing new molecular models, meshing techniques, and enhanced energy exchange and chemistry models, which improved physical fidelity. In 2013, Bird[8] introduced a set of practical enhancements aimed at improving computational performance.

The limited computational power available when the DSMC method was first introduced has influenced both the development and the perception of the method ever since. In contrast with continuum methods, DSMC appeared to employ overly simplistic models (heuristic in nature) and to lack a solid mathematical foundation. DSMC was considered to be computationally slow and thus too limited compared to its continuum competitors.

Over this time span, validation against experimental measurements and the introduction of improved procedures have alleviated many of these conceptual concerns. DSMC was proved to produce solutions in agreement with the Boltzmann equation (BE) for simple monatomic, nonreacting gases.[62] Subsequent theoretical work and numerical comparisons for simple gases have demonstrated that DSMC is in fact better at describing the nonequilibrium behavior[30] of a gas than established continuum methods.

Since it includes thermal fluctuations and deviations from equilibrium distributions usually omitted from continuum formulations, DSMC is richer in physical content than most of its continuum counterparts and thus is capable of delivering higher-physical-fidelity simulations. As a result, DSMC has been applied to areas far beyond rarefied gas dynamics, including hydrodynamic instabilities[24,25] and turbulence modeling.[26,27] However, the fundamental molecular nature of the method, which gives DSMC a physical advantage, does indeed make it computationally demanding, which in a practical sense limits its applicability.

Given this history, it is not surprising that DSMC did not become widely appreciated or used until computing power caught up with Bird's vision in the 1980s. Since that time, the power of the largest computers available for scientific simulation has increased dramatically due to both continuous advances in processor hardware (Moore's law) and the advent of massively parallel supercomputers based on cheap commodity components. Initially, parallel machines used single or few-core CPUs (central processing units); in the last decade, accelerator chips such as GPUs (graphics processing units) and many-core CPUs (e.g., the Intel KNL processor) have been leveraged for scientific use, including in the largest parallel machines.

To quantify, in 1988, the fastest supercomputer (as measured by the LINPACK benchmark[48]) was an 8-processor Cray YMP, which could factor a dense matrix at the rate of 2.1 gigaflops (10^9 64-bit floating point operations per second). The fastest current machine computes the same benchmark (for a much larger matrix) at 122 petaflops (10^{15}), and exascale machines (10^{18}) are planned for deployment in the early 2020s both in the US and internationally. Thus, the increase in available scientific computing power on the largest machines will have grown by a factor of half a billion in a little more than 30 years. While these performance numbers are for dense-matrix operations, many scientific computational methods, including DSMC, have benefited from similar factors of speed increase due to Moore's law and massive parallelism.

The fundamental DSMC algorithm is inherently parallel. In each time step, particles advect independently and may experience collisions with surface elements representing the surfaces of complex objects embedded in the gas flow. Particles are then grouped by grid cell, and pairwise collisions and chemical reactions are computed independently within each grid cell. Thus, the advection step can be parallelized over particles, and the collision-chemistry step can be parallelized over grid cells. However, balancing these computations across large numbers of processors can be challenging due to temporal and spatial variations of particle densities in the flow. For example, densities can vary by orders of magnitude for hypersonic flows over objects that include both the compression region and the wake.

Several notable parallel DSMC codes have been developed over the last 30 years to leverage the increasing computational power of parallel machines. The ICARUS code[3] was developed in the 1990s for 2D and 2D axisymmetric models and ran scalably on teraflop (10^{12}) machines with 1000s of single-core CPUs. It used a concatenated grid that was a union of regular-grid tiles that were body-fitted to object surfaces by a simple analytical formulation. The DAC code[44] was developed at NASA in the 2000s and was the first to run 3D models on the large-scale parallel machines of that era. It used hierarchical rectilinear grids with up to 3 levels of refinement and embedded arbitrary triangulated surfaces into the grid via a cut/split methodology, whereby one or more triangles cut through an orthogonal grid cell and might split it into more than one disjoint flow volume. The SMILE code[36,37] was developed in the same time frame at the Russian Academy of Sciences. It models 2D or 3D flows using a rectilinear grid that can dynamically adapt to flow properties. Kashkovsky, one of the SMILE developers, has also worked more recently on adapting DSMC algorithms for GPUs, including parallelization methods for GPU clusters,[40] a topic we discuss in Sec. II H.

In the 2010s, the Schwartzentruber group at the University of Minnesota developed their parallel Molecular Gas Dynamics Software (MGDS) code.[54,65] Similar to DAC, MGDS is a 3D code using 3-level hierarchical rectilinear grids with cut/split cells for embedding triangulated surfaces. MGDS has been used as a platform to develop new, efficient algorithms for performing cut/split computations[65] (now also used in DAC) and for distributing very large surface-element models across processors to enable modeling of porous heat-shield materials exposed to gas flows.[58] Also in the 2010s, the dsmcFoam code (now dsmcFoam+)[56,64] was implemented within the OpenFOAM finite-element fluid dynamics framework.[12] It uses the body-fitted unstructured finite element meshes provided by OpenFOAM for continuum fluid dynamics as DSMC grid cells for grouping particles for collisions and chemistry. dsmcFoam is the only one of these several parallel codes which has been made freely available as open-source software.

In 2012, we began the development of SPARTA,[13,29] a new open-source DSMC code, with two goals. The first goal was to implement capabilities similar to those found in DAC and MGDS but in a manner that would enable them to run efficiently on the largest parallel machines, with tens of thousands or more nodes and millions of cores. Recently, this has included the development of DSMC kernels that can run on GPUs and many-core CPUs (Intel KNL) via Cuda and OpenMP threading. As we highlight in this paper, SPARTA has been used to run the largest DSMC models of which we are aware, with up to one hundred billion particles and billions of grid cells. SPARTA now supports models with billions of surface elements as well. The second goal was to make it easy to add new capabilities to SPARTA (e.g., different gas/surface collision/chemistry models or boundary conditions) so that the code can be used as an open research platform for users and developers to implement and test new models.

In brief, SPARTA allows for 2D, 2D axisymmetric, and 3D models. It uses a many-level hierarchical rectilinear grid with cut/split embedding of surface elements (triangles in 3D, line segments in 2D, or 2D axisymmetric). It also implements two kinds of surface elements, either explicit elements as in DAC and MGDS or implicit elements. The latter are defined by level-set values on grid corner points, which can be input from an experimental voxelated image. A Marching Cubes algorithm is used to infer triangles in each grid cell. As we explain in Sec. II, this approach offers an alternative methodology for modeling porous materials and ablation processes compared to the methodology described in Ref. 58.

It is worth noting that various approximations have been developed since the 1994 formulation of DSMC[7] to lessen the computational cost of particular kinds of simulations by reducing the number of particles or grid cells needed. Examples include the BGK-DSMC method[28] and its linearized collision model and the "sophisticated

DSMC" variant,[8] which selects nearest-neighbor collisions and uses variable-size time stepping (subcycling) in regions of higher density.

We have largely eschewed these methods in SPARTA in the interest of accuracy for modeling fundamental hydrodynamics instabilities and turbulence effects. Instead, we have focused on enabling the canonical 1994 Bird algorithm to run as scalably as possible on large machines. The kinds of problems we highlight in this paper were outside the scope of the original DSMC algorithm and its focus on rarefied flows. However, due to advances in computing power and the insights gained in the last 50 years of DSMC research, it appears that DSMC can now not only model these phenomena but also provide physical insight.

The remainder of this paper is organized as follows: In Sec. II, we discuss the parallel design and algorithms used in SPARTA that enable it to run efficiently on large machines using MPI (the standard message passing interface library). These include algorithms for on-the-fly load-balancing, grid adaptation, and image creation, all of which we have found useful when running large simulations. In Sec. III, we highlight SPARTA performance on different platforms with up to millions of CPU cores and on GPUs and Intel KNL processors. Finally, in Sec. IV, we give brief descriptions of several physics and engineering simulations for which being able to run DSMC models at large scale has yielded new scientific insights. In Sec. V, we offer a few comments on the state of current petascale simulation and prospects for phenomena that could potentially be modeled with DSMC on coming exascale machines.

II. SPARTA ALGORITHMS AND DESIGN

A serial or parallel DSMC simulation involves three kinds of data: particles, grid cells, and (optionally) surface elements. Surface elements tile the surfaces of objects embedded in the gas flow and act as impermeable boundaries to particles. Such objects must be "watertight" in the sense that they enclose a volume which particles cannot enter. SPARTA defines the surfaces of 2D or 2D axisymmetric objects with a collection of line segments (truncated cones in the axisymmetric sense) and the surfaces of 3D objects with a collection of triangles.

SPARTA uses a rectilinear hierarchical grid. Rectilinear means that all grid cells are axis-aligned, as rectangles in 2D or rectangular parallelepipeds (bricks) in 3D. The hierarchy is defined as follows. The simulation box itself is the root or zeroth level; it is conceptually a single large grid cell. The first level of the hierarchy is a regular grid of $A_1 \times B_1 \times C_1$ cells. Any individual first-level grid cell can be refined further with an $A_2 \times B_2 \times C_2$ regular subgrid to produce second-level cells. This process continues recursively to a depth of N levels. The user defines appropriate $\{A_i, B_i, C_i\}$ values for each level. A typical gridding procedure defines $\{A_1, B_1, C_1\}$ as large values to create a coarse grid over the entire simulation domain. The remaining $\{A_i, B_i, C_i\}$ values are small (e.g., 2 or 3) to allow incremental refinement of the coarse grid where needed within the simulation domain. The adaptive grid strategy in SPARTA is described in Subsection II E.

Cells within the hierarchy that are refined are parent cells; they store only minimal information. Cells with no further refinement are child cells. They store information that enables particles to advect through their volumes, including surface elements that intersect them and indices to neighboring grid cells. Each cell has an integer ID which is the concatenation of the bits for its parent's ID with bits that index the child from 1 to M within the $M = A_i \times B_i \times C_i$ subgrid cells of the parent. The only limit on N, the number of levels in the hierarchy, is that IDs must fit within a 64-bit integer. For example, a 3D octree, where all $\{A_i, B_i, C_i\} = 2$, can have up to 15 levels. A 2D quadtree can have up to 21 levels. An example of a grid with 5 levels is shown in Fig. 1. In a 2D slice of the 3D grid, the first-level grid is 25×25. The other 4 levels are refined as 2×2 subgrids within specific cells.

As illustrated in Fig. 1, the purpose of the hierarchical grid is twofold. First and foremost, it allows for variable-sized grid cells to group similar numbers of particles even when the spatial density of particles varies dramatically (e.g., on the upwind side of a spacecraft in a flow as compared to the downwind side). This enables the number of pairwise collisions performed in each grid cell to be no larger than is needed to generate accurate statistics. Likewise, gas-flow properties which have large spatial gradients and are measured by tallying grid-cell statistics can be tracked at the appropriate resolution. Second, when particles move from one grid cell to the next and the two adjacent cells are of different sizes (at different levels), the hierarchical nature of the grid makes it easy to find which grid cell the particle has moved into. Note that DSMC has no requirement that the change in grid cell size be at most one level for adjacent cells. The recursive nature of the hierarchy means that identifying which child cell a particle is in requires at most N recursive steps from the root cell to the child cell. Typically, it takes only 1–2 steps depending on the change in size between adjacent cells.

For parallel simulations, SPARTA partitions grid and particle data across processors based on child cells. Each child cell is assigned to one processor along with the particles in that cell. This assignment can be done such that each processor owns a clump of child cells and the processor's subdomain is compact with minimal surface-to-volume ratio. Figure 2 shows a 2D analog of Fig. 1 for flow around

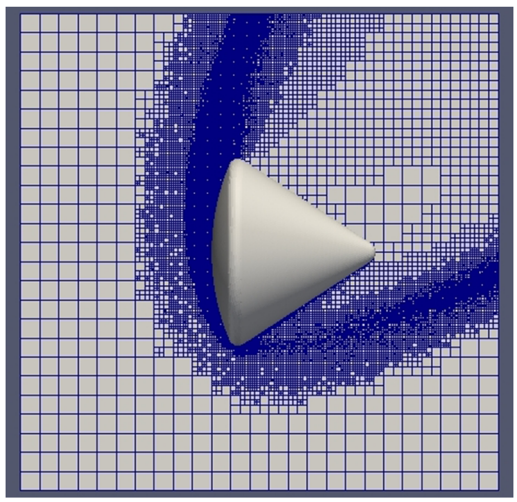


FIG. 1. A 2D slice of a 3D hierarchical grid around the Apollo space capsule. The grid has 5 levels of refinement that track the spatially varying particle density resulting from gas flow from the bottom left to the top right of the simulation domain.

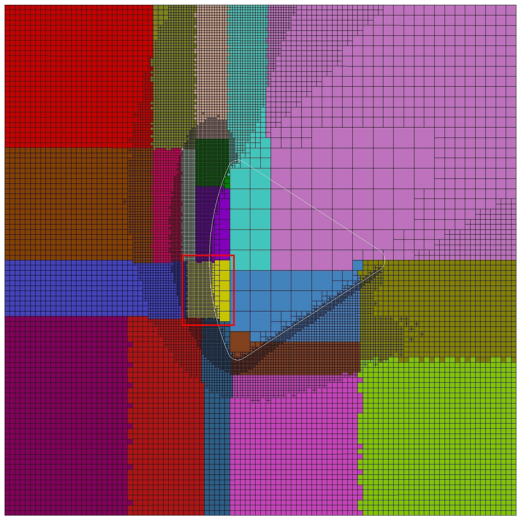


FIG. 2. A 5-level 2D hierarchical grid around the Apollo space capsule, outlined in white. This grid was partitioned via recursive coordinate bisectioning to assign equal numbers of child grid cells to each of 20 processors (colored subdomains). The red rectangle is a bounding box around the gold processor's subdomain extended by a cutoff distance. Grid cells of other colors overlapping the bounding box are stored as ghost cells by the gold processor.

the Apollo space capsule. As before, the flow is from lower left to upper right; grid cells are refined where the particle density is largest. The 2D representation shows that any large grid cell which is partially outside the capsule is a candidate for refinement. The colors represent clumps of child grid cells assigned to each of 20 processors. Details of the load-balancing method which produces this kind of grid partitioning are discussed in Subsection II C.

The red rectangle in Fig. 2 is a bounding box around the grid cells owned by the gold processor, augmented by a cutoff distance. Each processor also owns ghost cells, which are copies of grid cells owned by other processors that overlap its extended bounding box; there are 7 such neighbor processors of the gold processor. The purpose of the clumped partitioning and the storing of extra ghost cells is to minimize communication of particles between processors. At each time step, the majority of particles a processor advects remain within grid cells it owns; only particles near the surface of its clumped domain will potentially move to grid cells owned by other processors. Since the processor owns ghost copies of those cells, it can advect those particles to their final positions. At the end of the time step, only the relatively few particles that end up in ghost grid cells need to be communicated to their new owning processors. More details of this communication procedure are described in Subsection II B.

Each child cell stores indices into a list of surface elements that intersect its volume. In SPARTA, explicit surface elements are those for which an input file of element geometry is provided (line-segment end points for 2D, triangle corner points for 3D). A single explicit element may intersect many grid cells, each of which stores its index. For problems with small surface-element counts (less than a few million), each processor can store the entire list of surface elements and their geometric data. Note that, depending on the surface resolution needed, even huge, complex objects can be represented with far less than a million triangles (the Mir space station has been modeled with 50,000 triangles in DSMC simulations), so this storage mode is often adequate. For problems with much larger counts of surface elements, SPARTA also has an option to distribute surfaces so that each processor stores only the subset of elements which intersect its owned and ghost grid cells. In this mode, a surface element that intersects multiple grid cells will be stored only by the processors which own one or more of those cells. As explained above, the reason to store surfaces that intersect ghost cells is to enable processors to track particle movement, including collisions with surfaces, to the end of the time step.

When the surface elements of arbitrary-shaped objects are embedded in a rectilinear grid, the surface elements that intersect an individual grid cell reduce (or cut) the portion of its volume accessible to particles. This flow volume must be calculated for each cell for use as a parameter when gas-phase collisions are performed. In some cases, the set of intersecting surfaces can split the cell into two or more disjoint subvolumes. This must also be detected so that particles in the cell can be assigned to the correct subvolume, again for the purpose of grouping particles to perform gas-phase collisions.

Similar to the MGDS and DAC codes,[44,54] SPARTA implements the cut/split algorithm of Zhang and Schwartzentruber[65] to perform these calculations in 3D for triangles intersecting a rectangular parallelepiped (brick). In 2D, for multiple line segments intersecting a rectangle, an adaptation of the Weiler-Atherton algorithm is used.[63] We note that the intersection of a set of triangles with the face of a brick is conceptually similar to the 2D cut/split problem. The SPARTA implementation of the 3D cut/split algorithm leverages this connection by invoking the 2D algorithm 6 times (once per brick face) to simplify the coding for the 3D case.

Figure 3 diagrams four different topologies for surface-element/grid-cell intersections in 2D. The upper left panel shows

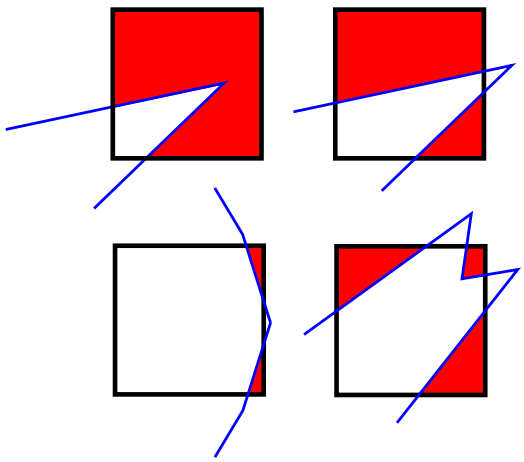


FIG. 3. Different kinds of intersections between a 2D surface composed of blue line segments and a single black 2D grid cell. Upper left: a sharp-corner surface cuts the cell, resulting in reduced flow volume (in red). Upper right: the same surface splits a cell, resulting in two disjoint flow subvolumes. Lower left: a gently curved surface can still split a cell into two disjoint subvolumes. Lower right: a more complex surface can split a cell into more than two disjoint subvolumes.

a cut cell with no splits. The reduced flow volume (exterior to the object embedded in the flow) is shaded in red. The surface elements in the upper right panel are identical, but their positions relative to the grid cell split it into two disjoint subvolumes (subareas in 2D), each shaded in red. While it may seem that only an extreme geometry can produce split cells, the lower left panel illustrates that this can occur even for gently curved objects due to discretizing the surface into line segments (or triangles). The lower right panel is an example of a split with three disjoint subvolumes. SPARTA is coded to detect an arbitrary number of split subvolumes per grid cell.

SPARTA also supports an implicit surface representation in which the geometry of individual elements is inferred from an input file containing a 2D or 3D array of scalar values. This input can be pixel or voxel values from a 2D or 3D experimental image of a porous material, such as heat-shield insulation for a spacecraft. Such values are often 1-byte integers (0–255), representing gray-scale in an image. References 9, 11, 19, 20, and 50 give more details on how microtomographic images can used to computationally characterize porous materials for heat-shield applications, including for DSMC.

When reading a 3D file, SPARTA maps the 3D array to a 3D region of uniform-size child grid cells. Each grid cell is assigned 8 values from the array, associated spatially with its 8 corner points. A threshold value is selected (e.g., 112.5) which conceptually partitions the volume of the collection of grid cells into two subvolumes, one for the interior of the porous material and the other for the exterior where gas particles can flow. Note that, depending on the threshold and corner-point values, either or both of these volumes may be noncontiguous.

A triangulation of the level-set surface that divides the two partitions is computed by the Marching Cubes algorithm,[49] invoked on each grid cell. The implementation of Marching Cubes in SPARTA includes topological and robustness enhancements from Refs. 15 and 45. The threshold and 8 corner-point values define a set of up to 12 triangles wholly contained within the grid cell, with vertices and some edges which lie on the faces of the cell. Adjacent grid cells compute triangles with consistent vertices and edges. This is because two cells which share a face also share 4 identical corner-point values for the face. This means that the collection of Marching Cube triangles from all of the grid cells will represent contiguous surfaces of one or more watertight objects, so the DSMC algorithm can correctly advect particles through the porous material. Note that the Marching Cubes algorithm is inherently parallel. Each processor performs the computation for the child cells it owns and stores the resulting triangles. Figure 4 shows an example of a triangulated model of a porous material generated by this procedure.

SPARTA supports an analogous representation of 2D implicit surfaces. A 2D array is read from a file, 4 corner-point values are assigned to each grid cell, and a Marching Squares algorithm[51] is invoked to compute a set of up to 2 line segments contained in each grid cell.

The motivation for modeling porous materials with implicit surface elements is twofold. First, it enables generation of a material model matched to experimental data (images) which characterize real materials of different kinds, both structurally and quantitatively (e.g., a porosity metric). Second, the inference of surface elements by corner-point values is a natural means to simulate ablation by implementation of a model which correlates the energetics

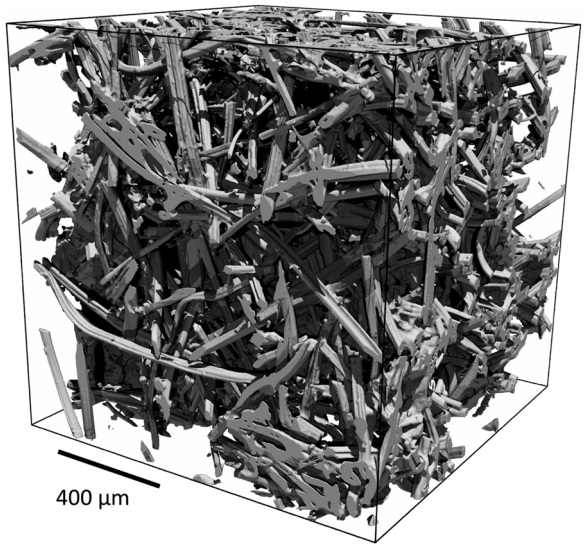


FIG. 4. A ParaView visualization of the triangulated surfaces generated from a sample of FiberForm™ by the Marching Cubes (MC) algorithm. FiberForm is a porous carbon fiber material which has been adopted as a substrate for NASA's flagship Thermal Protection Material PICA. A 3D microtomographic image was taken of a 0.52-mm^3 cube of material with 0.65-μm voxel edge size; the volume fraction of physical material is 14.4%. Image voxels were then mapped to 800^3 grid cells within SPARTA, and the MC algorithm was used to produce 57.4M (million) triangles which represent the surface of the material. Only 6% of the grid cells contain triangles due to the high porosity and small thickness of the fibers.

or chemistry of gas-surface collisions in each grid cell with decrements of its corner-point values. The set of altered surface elements can then be regenerated periodically as a simulation proceeds to incorporate changes in material structure in a spatially inhomogeneous manner. Work is currently in progress to add such models to SPARTA.

In Subsections II A–II H, we discuss a few additional SPARTA algorithms and features in more detail, with an emphasis on how they are implemented to enable large-scale parallelism. They also work well on smaller parallel clusters or desktop machines or even on a single processor:

- memory costs,
- time stepping in parallel with irregular communication,
- load balancing via recursive coordinate bisectioning,
- tallying of surface-element statistics in parallel,
- on-the-fly grid adaptivity,
- on-the-fly image creation,
- modular code design, and
- implementation for GPU and Intel KNL processors.

A. Memory costs

The memory cost for SPARTA to store each of the three DSMC data types discussed above is roughly 100 bytes/particle, 200 bytes/grid cell, and 150 bytes/surface element. A typical problem without surface elements has 20 particles/grid cell for adequate collision statistics. A terabyte (TB) of memory can thus store 450M (million) grid cells with their particles. For implicit surfaces, there

can be as many as 12 triangles per grid cell (say, 6 on average). A TB of memory can store 325M grid cells with their particles and surface elements. For problems where more particles per grid cell are desired, these numbers are reduced accordingly. There are also additional memory costs for each grid cell to store indices for the surface elements that intersect the cell (4 bytes/index) and for each processor to store ghost cells and their surfaces as well as communication buffers to exchange data between processors. However, these generally incur only small additional memory overheads if the simulation data can be evenly distributed across processors.

To put these memory costs in context, a current-generation CPU-based cluster at Sandia with dual-socket Intel Broadwell CPUs (36 physical cores) has 128 gigabytes (GB) of memory per node, or 1 TB for every 8 nodes; a 1024-node cluster has 128 TB. On the older IBM Blue Gene Q platform (where many of the simulations described in Sec. IV were run), each 16-core node has 16 GB of memory. A single rack (1024 nodes) thus has 16 TB of memory; the full Sequoia machine with 96 racks has 1.5 petabytes (PB).

These numbers indicate that large parallel machines can easily store the data needed for DSMC simulations with tens to hundreds of billions of particles, as well as billions of grid cells and/or surface elements. Typically, the DSMC method and SPARTA are CPU-limited, not memory-limited, when determining how large a problem can be effectively modeled.

B. Time stepping in parallel with irregular communication

With the described distribution of data across processors, the per-time step computational tasks for a parallel DSMC simulation are conceptually simple.

1. Create: create particles at simulation-box or surface boundaries.
2. Move: advect each particle; surface collisions and chemistry may occur.
3. Communicate: migrate particles to new owning processors.
4. Sort: group particles by grid cell.
5. Collide: perform gas-phase collisions and chemistry within each grid cell.
6. Stats: tally values on a particle, grid-cell, or surface-element basis.

The Create step is optional; particles may be inserted by each processor that owns grid cells which touch simulation-box faces or intersect surfaces that emit particles.

For many problems, the Move step is the most computationally expensive. Each particle advects (independent of all others) for a straight-line distance determined by its velocity and the time step. In SPARTA, this is implemented as one large loop over all particles a processor owns. A particle's advection path is ray-traced from one grid cell to the next. If a grid cell it moves through is intersected by surfaces, the advection path is checked to see which surface element (if any) the particle collides with first. A surface collision changes the particle velocity and, if surface chemistry is enabled, may delete the particle, changes its species, or add a new particle. As mentioned above, because each processor owns copies of nearby ghost cells and ghost surface elements, it can advect particles through cells owned by another processor. At the end of the Move step, the majority of particles are still in cells owned by the same processors because of the clumped distribution illustrated in Fig. 2.

At the end of the Move step, a per-grid-cell weighting factor can also optionally be applied. Particles are cloned or deleted depending on the ratio of weighting factors for the two grid cells in which the particle's advection began and ended. Cloning and deletion decisions for fractional ratios use random numbers. Weighting factors can be particularly useful for 2D axisymmetric problems to ensure adequate particle counts near the axisymmetric axis, where the 3D swept volumes of 2D axisymmetric cells are small.

In the Communicate step, particles which ended their advection in a ghost grid cell owned by another processor are sent to that processor. This is a local communication operation in which each processor sends messages to a few neighbor processors, but it is also irregular due to the nature of the partitioning illustrated in Fig. 2 (as contrasted with a regular 2D or 3D grid of processors communicating with their neighbors).

Each time grid cells are assigned to processors (e.g., initially or on a load-balancing time step as discussed in Subsection II C), ghost cells are acquired anew from neighboring processors. Each processor thus knows the short list of neighbor processors to which it may need to send particles. In the first stage of the Communicate step, each processor copies those particles into a send buffer and creates an index list for each neighbor of which particles to send it. It then compresses its own particle list to remove the particles it is losing. In the second stage, each processor sends a message to each neighbor with the count of particles it will receive. Each processor can then allocate new memory (if needed) for its incoming particles and tell MPI where to append each packet of particles to its list of owned particles. In the third stage, each processor sends a list of particles to each of its neighbors (if the count is nonzero). It then waits for all its incoming particles to arrive before proceeding.

If the ghost cutoff distance is sufficiently large, then the Communicate step is done only once per time step. If not, a (small) number of particles may advect to grid cells beyond processor's ghost cells. In this case, the particle is sent to the processor owning the last ghost cell, along with the remaining distance it needs to advect and a flag indicating it is still active. The receiving processor continues its advection. In this case, the Move and Communicate steps are looped over (within a single time step) until all particles have completed their advection.

The purpose of the Sort step is to create a list of particles in each grid cell for use by the subsequent Collide step. In SPARTA, each processor stores a single large list of particles for all its owned grid cells. Because particles continuously move to new cells or are added and deleted, the list rapidly becomes unordered with respect to grid cells. The Sort creates a linked list for each grid cell of the particles it contains. This requires one integer vector of length M = number of grid cells, which stores the index of the first particle in the cell, and a second integer vector of length N = number of particles, where each particle stores the index of the next particle in the same cell. These two vectors can be created by a single loop over all the particles. An option is provided to reorder the particles in memory so as to group particles by grid cell, which can offer some improvement in cache performance. On CPUs, the extra cost for reordering is often not worthwhile. On GPUs, it can be, as explained in Subsection II H.

For many problems, the Collide step is the second most-expensive step after the Move step. If the particle/cell count is high for desired statistical accuracy, this step may dominate since the number of collision attempts in a grid cell is quadratic in the number of particles. In each grid cell (independent of all others), collisions are attempted between randomly selected pairs of particles and performed if a collision probability exceeds a random number. Collisions change the velocity of each particle. If gas-phase chemistry is enabled, a collision may delete one of the particles, change one or both of their species, or create a new particle.

In the final Stats step, requested statistics are tallied for the collection of particles and individual grid cells or surface elements. These can be temporally or spatially averaged and output in various ways. For surface elements, the tallying due to collisions of individual particles with the element is performed during the Move step when the collisions occur. For explicit surfaces, a single surface element can intersect many grid cells. Thus, the contributions from multiple grid cells, possibly owned by multiple processors, need to be combined to produce the result for the entire element. This operation is discussed in Subsection II D.

C. Load balancing via recursive coordinate bisectioning

The motivation for load balancing is to distribute equal amounts of computation to each processor while minimizing the volume of communication between processors.

SPARTA uses the recursive coordinate bisectioning (RCB) method[4] for this purpose because it can track large spatial variations in particle density and is quick and highly parallel to compute. It can be invoked before or after runs (e.g., after equilibration and before a statistics-gathering run). Alternatively, it can be invoked periodically during a run if particle densities are changing spatially or temporally. An imbalance threshold can be set to trigger the rebalancing only when necessary. Imbalance is defined as the maximum "computational weight" on any processor divided by the average weight across all processors; a value of 1.0 is thus perfect balance.

The granularity for partitioning computational work in SPARTA is a child grid cell and the particles it contains. When RCB is invoked, each processor contributes the center point and a weight value for each of its owned grid cells. RCB partitions the global collection of cells to assign each processor a compact clump of cells with aggregate weight that is (nearly) the same for every processor. If weight values are set to unity for every cell, each processor is assigned the same number of grid cells. If the weights are the numbers of particles per cell, each processor is assigned the same number of particles. Another weighting option is based on the CPU time each processor tallies for preceding time steps. The time spent on the Move, Sort, and Collide operations is summed, which excludes the Communicate operation and any idle time that results from load imbalance in the current partitioning. This summed time is then apportioned to each grid cell as a weight based on the number of particles in that cell divided by the processor's total particle count.

The way the RCB algorithm works is illustrated in Fig. 2 for the case of assigning equal numbers of child cells to each of 20 processors. In this example, the first-level cut is made with a horizontal cut at a position in the y dimension that nearly bisects the capsule. It puts half the grid cells below the cut and half above. Similarly, 10 of the processors are assigned below the cut and 10 above. The placement of the cut is determined iteratively (similar to a binary search in a sorted list), by computing the sum of weights on each side of the cut and adjusting its position for the next iteration.

This procedure is repeated recursively until each processor is assigned a single rectangular-shaped subdomain. At every level, each cut is made in a coordinate direction (x, y, or z) chosen to keep the aspect ratio of the two resulting subdomains as near unity as possible (square in 2D and cubic in 3D). In Fig. 2, the second-level cuts on both sides of the first-level cut are vertical lines. Below the first cut, the vertical cut is between the dark blue and pink processors (at the bottom edge of the domain). Above the first cut, the vertical cut is between the green and off-white processors (at the top edge of the domain). Both the vertical cuts assign 5 processors to their left and right. For non-power-of-2 processor counts, whenever an odd number of processors (say 13) is assigned to a subdomain at some level of the recursion, the cut position at the next level is adjusted accordingly (6/13 of the weight on one side and 7/13 on the other).

The RCB operation completes after $\log_2 P$ levels of recursion, where P is the processor count (number of MPI tasks). At each level, one MPI_Allreduce operation per iteration is needed within subsets of processors owning a subdomain to iteratively find the position of the next cut. After the cut is made, each processor in the subset sends a message to a partner processor on the other side of the cut containing grid-cell center point and weight values that belong on the partner's side of the new cut.

After the RCB operation completes, every grid cell is assigned to a processor. The data and particles for cells assigned to new processors, including their particles and the surface elements they intersect, must be communicated. If the new partitioning is dramatically different from the old, nearly all cell/particle/surface data may be communicated in this operation. Because this involves packing and unpacking data to and from message buffers, SPARTA has an option to perform this large-scale data communication in stages, using buffers of limited size.

D. Tallying of surface-element statistics in parallel

As mentioned at the end of Subsection II B, summing surface collision statistics for explicit surface elements requires parallel communication. This is because a single surface element may overlap grid cells owned by many processors. SPARTA formulates this summation as a rendezvous operation.[55] When surface elements are read in, each element is assigned to and stored uniquely by a single processor in a round-robin fashion. Processors are assigned every Pth element, where P is the number of processors. This storage is in addition to each processor owning a copy of surface geometric data for elements which intersect its owned and ghost grid cells.

When surface statistics are tallied, each processor sends the partial statistics it has accumulated for each surface element that intersects its grid cells to the processor which owns it uniquely. The contributions from all processors thus rendezvous at the owning processor of each element, which can sum them to a final value for further time averaging or output.

Unlike the local irregular communication performed to migrate particles to new owning processors each time step, the rendezvous operation is global; each processor communicates with many arbitrary processors, possibly even all other processors. The total volume of communication is proportional to $M \times N$, where N is the number of surface elements and M is the average number of grid cells each element intersects. The round-robin assignment of surface elements ensures that the communication is roughly load-balanced and can use the full communication bandwidth of a parallel machine. The MPI_Alltoallv() method is used for this rendezvous operation; it is typically well-optimized on parallel machines with large processor counts to work effectively at scale.

E. On-the-fly grid adaptivity

The hierarchical grid SPARTA used to track particle advection and group particles for collision was described at the beginning of Sec. II. SPARTA allows the grid to be adapted either between runs or on-the-fly during a run as flow parameters or particle densities change spatially. Individual child grid cells can be refined to the next level in the hierarchy (e.g., 1 cell becomes $2 \times 2 \times 2 = 8$ smaller grid cells. Or a cluster of child cells in the same parent cell can be coarsened into a single new child cell replacing the parent (e.g., $2 \times 2 \times 2$ child cells become 1 larger child cell). The criterion for refinement or coarsening a grid cell can be particle count or a grid-cell-based quantity that SPARTA tallies statistics for, either on the current time step or in a time-averaged sense (e.g., the mean free path λ between molecular collisions, which is a function of the number density and the temperature of the particles in the cell). In addition, threshold values are specified which will trigger refinement or coarsening. SPARTA also allows for grid adaptation to be considered only for a subregion of the simulation box, which can be useful if spatial flow properties are known *a priori*.

The refinement algorithm requires minimal communication between processors. Each processor loops over the child grid cells it owns and determines whether to refine them or not. If a cell is refined, the processor creates new child cells, assigns each of the old cell's particles to the appropriate new cell, computes the list of surface-element intersections for each new cell from the list of old cell intersections, and then converts the old cell to a new parent cell. Each processor must then discard its ghost cells and reacquire them from neighboring processors since processor's ghost cells may have been refined by another processor.

The coarsening algorithm requires more communication. Consider the set of child cells which belong to the same parent cell. The decision to coarsen the parent so that it becomes a new child cell requires information from each child cell in the set (e.g., their particle count). The child cells in the set may be owned by different processors. This information needs to be gathered to a single processor which can make the decision to coarsen or not. If yes, the single processor becomes the owner of the new child cell and assigns all the particles and surface elements from the original set of child cells to the new cell. Finally, it sends a message to the owners of each child cell in the set, so they know whether to delete their child cells or not. To maintain a clumped distribution of child cells, as in Fig. 2, a processor which owns a child cell at (or near) the geometric center of the original set is chosen as the single processor the others in the set communicate with.

F. On-the-fly image creation

When running simulations for many time steps with billions of particles, grid cells, or surfaces, the volume of data to output can be immense. Postprocessing and visualization can incur additional large computational and I/O costs. As an alternative option, SPARTA allows for creation of on-the-fly snapshots (JPEG or PNG files) of a simulation. Examples are shown in Fig. 8 and in Refs. 24 and 25.

These images are created in parallel in the following manner. Each processor renders the fraction of objects it owns into an empty JPEG (or PNG) buffer the size of the desired image (e.g., 1024 × 1024). Particles are rendered as small spheres, grid cells as rectangles (2D) or rectangular parallelepipeds (3D), and surface elements as cylinders (2D line segments) or triangles (3D). The first time a pixel is drawn into the buffer, a per-pixel depth value is also stored in a separate buffer, which represents the depth (positive or negative) of that pixel (on an object's surface) with respect to the image plane. Anytime a new pixel is computed, its depth value is compared to the stored value for that pixel. If the new pixel is in front of the old one, it replaces the pixel in the JPEG buffer, and the stored depth is updated. If it is behind the old one, the new pixel is discarded.

Once each processor has rendered an image of its objects, the image buffers are pairwise merged into a final image in $\log_2(P)$ number of iterations, where P = the number of processors. At each iteration, half the processors with unmerged images send their image buffer and depth buffer to partner processors which perform the merges. The pixels of both images are looped over, and the in-front pixel and its depth value are kept in the new image. After the last merge, the final image contains all the simulation data; one processor writes it to disk. This operation scales well to large parallel machines because the initial rendering is perfectly parallel (assuming equal amounts of simulation data/processor), the iteration count is logarithmic in P, and the communication per processor needed at each iteration is the size of a single image.

G. Modular code design

One of the goals of SPARTA design as an open-source code was to make it easy for developers and users to contribute code for new features. This is done via styles SPARTA defines, each of which has two parts. The first is a C++ virtual parent class which specifies the API (application program interface) by which the rest of SPARTA uses the style. The second part is multiple child classes, each of which inherits the interface and implements a specific version of the style. An advantage of this code structure is that implementing a new feature often requires only a few functions to be written in a single new class, without the code author needing to know too many details about the rest of the code. Ideally, the new class will then work seamlessly with the rest of SPARTA, with minimal chance of it inducing bugs in the remainder of the code.

Defined styles include collision models (one for gas-phase and one for surfaces), chemistry models (likewise for gas-phase or surface), diagnostic calculations, and output formats. Examples of diagnostic calculations are global quantities, such as gas temperature, or per-particle, per-grid cell, or per-surface-element energies, fluxes, and moments.

There is also a very flexible "fix" style which allows customization of the SPARTA time step by inserting calculations defined in

the child class at specific points in the time step. An example is child classes which emit particles from simulation box faces or surfaces at the beginning of each time step. Another is time-averaging classes which invoke diagnostic styles every few time steps at the end of time steps for grid cells or surfaces and accumulate their values for occasional output. Likewise, classes which perform dynamic load balancing and adaptive gridding, as described in Subsections II E and II F also operate at the end of time steps. An ambipolar model for ionized gases was added to SPARTA via this style, leveraging an option it provides to define additional per-particle data which are carried along with particles as they migrate to new processors. In the ambipolar case, a flag indicating the particle is an ionized electron and storage for its velocity vector is defined. These per-particle values are accessed and modified by gas-phase and surface collision models with ambipolar options.

Finally, there is a style which allows definition of new input script commands which can be invoked before, between, or after time stepping runs. As an example, commands to read/write grid or surface topologies from/to files are defined in this manner.

Current SPARTA is ~115K (thousand) lines of code. Roughly 1/3 of these lines implement core SPARTA functionalities; the remaining 2/3 are added child style classes. We expect that the latter fraction to grow as new capabilities is added to the code.

H. Implementation for multicore and GPU processors

In the past decade, GPUs and many-threaded CPUs (such as the Intel KNL processor) have become widely used for scientific computing, both on the desktop and increasingly in large-scale clusters and supercomputers. Fully exploiting their capabilities for a DSMC code requires exposing fine-grain parallelism in its fundamental operations, in addition to the coarse-grained distribution of work across processors, which can be handled effectively via MPI as described earlier in this section.

In SPARTA, we use the Kokkos library[18,47] to define the fine-grain parallelism inherent in the per-time step computations outlined in Subsection II B. Kokkos is an open-source library that provides abstractions to map a single source-code implementation of an application kernel (such as particle advection or per-grid-cell collisions in DSMC) onto different back-end languages such as CUDA or OpenMP for specific hardware (GPUs or KNLs). This choice is made at compile time and also allows for the layout of data structures like 2D arrays to be changed (e.g., row-order vs column-order) without altering the application source code, to optimize performance for different hardware. It thus allows computational kernels in SPARTA to run efficiently on different node-level hardware (GPUs, KNLs, or CPUs) in tandem with the interprocess or internode level parallelism provided by MPI.

The modularity provided by styles, explained in Subsection II G, enabled us to write new Kokkos versions of the algorithms for particle advection and particle sorting (one thread per particle) and for gas-phase collisions (one thread per grid cell). Similarly, Kokkos variants of different diagnostic and fix styles have also been added incrementally to the code. Operations that occur only occasionally, such as load balancing and grid adaptation, are still performed on the host CPU using (nonthreaded) MPI-style parallelism. On a GPU system, this requires moving grid and particle data from the GPU memory to the CPU memory and then back after the operation is complete, which incurs some performance overhead.

Particle sorting by grid cell is an example where a different data structure and a different algorithm are needed to enable threaded parallelism. A 2D array of grid cells vs particle indices is created to store the particles in each grid cell, along with a 1D vector with counts of particles/cell. Each per-particle thread inserts an index into the 2D array and updates the count. Since particles assigned to different threads may reside in the same grid cell, thread atomics are required to ensure safe updates of the array and vector. The resulting 2D array is used by the gas-phase collision algorithm; each per-grid-cell thread is assigned a row (or column depending on the hardware) of the array to access the particles in its cell.

Other unique characteristics of the DSMC method that can impact threaded performance, either on GPUs or KNLs, are worth noting. We found that periodically reordering the particle list to make particles in the same grid cell be contiguous in memory can improve performance on GPUs. This is generally not the case on traditional CPUs.

When chemical reactions occur in a grid cell during gas-phase collisions, new particles can be created, which need to be immediately considered as candidates for new collisions. If too many particles are created, the size of data structures like the sorted grid/particle array can be exceeded. On a GPU, a data structure cannot be reallocated on-the-fly while it is being accessed in a thread-parallel manner. This thus requires reinvoking the collision algorithm (for that time step) using a larger data structure. Over-allocating such data structures with extra space can reduce the number of times this happens, but that increases memory costs and can degrade performance as well.

Finally, the complexity of the code required to advect a particle can impact the performance of a threaded version of the algorithm (one thread per particle). This is due to two effects. The first is the branching used to handle different possible trajectory paths. One particle may cross through multiple cells (e.g., near a corner); another may stay in the same cell. For each cell crossed, a variable number of surface elements need to be checked for possible collisions. Surface collisions may occur for one particle but not for another. Second, the diversity of trajectory paths also means that the computational cost per particle (and thus thread) can be very different. Likewise, for gas-phase pairwise collisions, there is a more modest degree of branching in the algorithms for various collision models that determine whether a collision takes place and how the different kinds of energies (translational, vibrational, and rotational) are repartitioned by the collision.

III. PERFORMANCE

The benchmark problem used here to assess performance is a uniform hard-sphere gas in a closed isothermal box. Molecular collisions in the gas as well as with the walls lead the gas, regardless of its initial state, to eventually reach thermodynamic equilibrium. Since mass, momentum, and energy are exactly conserved in every DSMC collision, this benchmark allows for gas-phase collision models and specular and diffuse boundary conditions to be verified, as well as code performance to be evaluated.

For these benchmark runs, argon gas at a temperature of 273.15 K and a number density of 7.1×10^{22} m^{-3} are simulated with

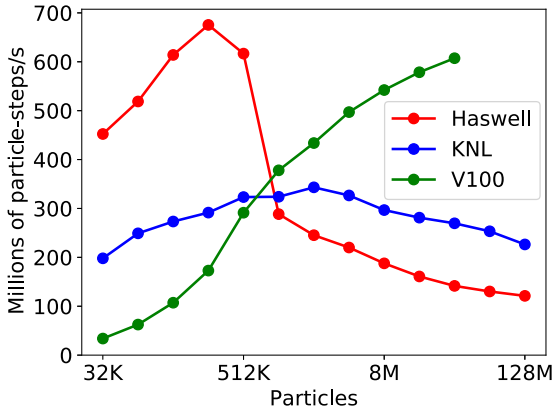


FIG. 5. Performance for the collision benchmark running on a single node or GPU of different hardware as a function of number of particles.

an average of 10 particles per grid cell. The system is first run for a short time with a large time step to disorder the particles within each processor's list (to avoid misleading performance due to preferential initial ordering on CPUs). The time step is then reduced to 7.0×10^{-9} s, and benchmark timings are measured. For GPU runs, the particle list is reordered in memory every 10 time steps.

Figure 5 shows performance for SPARTA as a function of particle count when running on a single node of three different kinds of node hardwares: a dual-socket Intel Haswell CPU, a many-threaded Intel KNL processor, and a single NVIDIA Volta (V100) GPU. The CPU has 32 physical cores and can run up to 64 MPI tasks (2x hyperthreading); the KNL has 68 physical cores and can run with up to 272 MPI tasks (4x hyperthreading), though 64 cores were used in these runs. For the Haswell and KNL nodes, different configurations of MPI ranks and OpenMP threads were tested to find which was fastest for a particular problem size.

The x-axis of Fig. 5 is on a log scale from 32K (thousand) to 128M (million = 1024 ∗ 1000 in this case) particles (3.2K to 12.8M grid cells). The y-axis is performed in millions of particle-steps per CPU second. Thus a KNL performance of 300M particle-steps/s for a model with 512K particles is running at a rate of 586 time steps/s. Note that this normalization of the y-axis performance means that perfect scalability for any curve would be a horizontal line.

The Haswell CPU has a strong performance maximum at 256K particles due to cache effects. Performance drops for larger problems when grid and particle data no longer fit in cache. The KNL performance is more flat, independent of problem size. The GPU performance increases with problem size; too small a problem does not generate enough work to fully utilize the available hardware threads. For this benchmark, Kokkos keeps all particle and grid data on the GPU for the duration of the run; there are no overhead costs to move data between the host CPU and GPU.

Figure 6 shows strong scaling of SPARTA on several parallel platforms with different node hardware. The Sequoia machine at Lawrence Livermore National Laboratory (LLNL) is an IBM Blue Gene Q (BG/Q) platform with 96K nodes, each with 16 physical cores which allow for up to 4x hyperthreading (64 MPI tasks per node). The Trinity machine at Los Alamos National Laboratory (LANL) has two partitions: one with 9400 dual-socket Haswell nodes and the other with 10 000 Intel KNL nodes. The Sierra machine at LLNL has 4320 nodes, each of which has 4 NVIDIA Volta (V100) GPUs and a dual-socket IBM Power 9 CPU.

Problem sizes for the collision benchmark range from 1M to 8B particles (100K to 800M grid cells) for all the machines except Sequoia, where the largest test has 1T (trillion) particles (100B grid cells). Each curve on each plot is the strong scaling for a particular problem size across a range of node counts. The y-axis on all the plots is millions of particle-time steps/s/node. The per-node normalization means that, as in Fig. 5, perfect scalability would be horizontal lines on these plots.

The BG/Q, Haswell, and KNL plots show similar trends. For a particular problem size, the performance (per node) increases for smaller node counts due to cache effects as the problem size per node decreases. As more and more nodes are used, the problem size per node shrinks enough that the cost of interprocessor communication degrades the per-node performance, though the aggregate performance across all nodes is still increasing.

For these 3 machines, as in the single-node runs, the impact of hyperthreading (more MPI tasks per core) and OpenMP threading (via Kokkos on the Haswell and KNL nodes, at most 2 threads/MPI task on Haswell, and 8 threads/MPI task on KNL) was tested in separate runs; the fastest results are shown in the 3 plots. For large problems on the Haswell system, use of all-MPI hyperthreading (64 MPI tasks/node) was almost always faster. On KNL, 4–8 threads/MPI task was typically best.

For large particle-per-node counts on BG/Q, runs with 64 MPI tasks/node (4× hyperthreading) were about 2.5× faster than using a single MPI task/core. However, the large problems could not be run on a large fraction of the machine with hyperthreading enabled due to the internal system memory overhead of using many millions of MPI tasks. Thus the 32K-node data points are for runs with 2× hyperthreading and the 64K and 96K (full machine) data points are for a single MPI/task per core, or 1.6M MPI tasks on the full machine. The performance at these data points drops more than it would if hyperthreading were possible.

The V100 GPU plot is different. Performance (per GPU) for all the strong scaling problems decreases with increasing GPU count (the aggregate performance still increases). Unlike Fig. 5, a portion of the data on each GPU must now be communicated each time step (via MPI) to other GPUs as particles move into grid cells owned by different GPUs. However, the performance trend is primarily due to the shrinking problem size per GPU when strong scaling. As Fig. 5 indicates, the smaller the problem becomes on a single GPU, the lower the GPU performance. The rightmost data point of 2048 GPUs is 512 nodes on the Sierra machine.

These plots can also be used to infer SPARTA's weak scaling performance, where the problem size grows proportionally with the node or GPU count. The fact that very large problems (curves on the right side of each plot) run at nearly the same speed (in particle-steps/s/node) as much smaller problems (left side curves of each plot) indicates excellent weak scaling.

We note that the per-node performance (y-axis) of current hardware (Haswell, KNL, and GPU) is significantly faster than the older BG/Q nodes. However, the Sequoia machine has many more nodes than even the largest current parallel machines. Thus, it has been an excellent platform for testing the parallel scalability of the

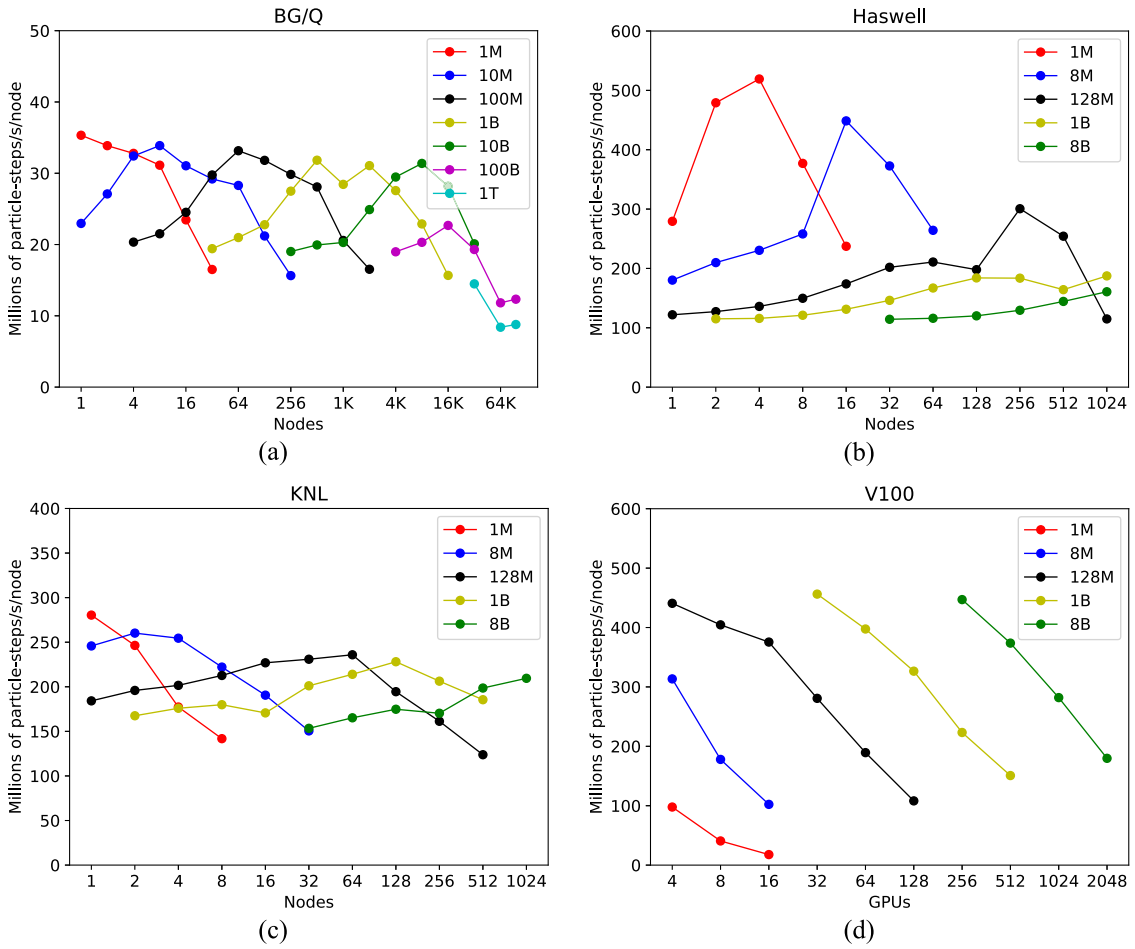


FIG. 6. Strong scaling of SPARTA for several different problem sizes run on varying numbers of nodes. (a) Sequoia Blue Gene Q machine with IBM nodes, (b) Haswell CPU partition of the Trinity machine, (c) Intel KNL partition of the Trinity machine, and (d) Sierra machine with 4 Volta (V100) GPUs per node.

various DSMC algorithms implemented in SPARTA, showing that the code can run reasonably efficiently on millions of MPI tasks. Similar to Sequoia, we have also run production problems scalably on the full Trinity machine using both KNL and Haswell nodes together (19K nodes and 1.2M MPI ranks).

Finally, Figs. 5 and 6 indicate that for a model of a given size (grid cells and particles), the question of how many nodes or even what kind of hardware (e.g. CPU vs GPU) is optimal to run on, is not a simple one. For DSMC models, at least in SPARTA, performance can be a strong function of problem size per node or GPU. Short benchmark runs like those shown in these plots are the most accurate way to estimate parallel efficiency or run times for a large simulation.

IV. RESULTS

In this section, we present three DSMC simulations performed with SPARTA that illustrate both the scale and the scope of models which can be run to observe and analyze fundamental fluid-flow phenomena. The test cases range from traditional hypersonic flow fields to canonical hydrodynamic instabilities, including their onset and growth. The computational power offered by modern platforms and the scalability of SPARTA for high-density/low-altitude hypersonic flows combine to enable the effects of nonequilibrium on flow fields to be investigated.

In previous work not discussed here, DSMC was shown to be capable of reproducing some of the hallmarks of turbulent flow, namely, the Kolmogorov −5/3 law[26] and the law of the wall.[27] SPARTA was also used to simulate hydrodynamic instabilities, namely, 2D Richtmyer-Meshkov[24] and Rayleigh-Taylor[25] instabilities. The simulations of 3D turbulent flows employed up to 10B (billion) grid cells and 100B particles were run on a third of the LLNL Sequoia BG/Q machine discussed in Sec. III for up to 1500 h. The simulations of 2D hydrodynamic instability flows used roughly one order of magnitude less cells and particles and required 30–60 h of the same computational resources.

A. Flow over a 25°–55° biconic

Axisymmetric flow over a 25°–55° biconic has been used as a standard benchmark for multiple computational and experimental

studies.[34,35,41,53] Here, the aim of this test case is twofold. The first is to demonstrate the validity of the code by comparing with experimental data. The second is to demonstrate that mesh adaptation can provide such significant savings that this test case can be simulated on current desktop machines.

All of the experiments are conducted at Reynolds numbers low enough to ensure that the flow remains laminar both upstream and downstream of the recirculation zone. The flow field involves complicated shock/shock and shock/boundary-layer interactions, and these interactions create steep flow gradients and high surface-heating loads. Thus, the flow characteristics are challenging to predict. As a result, this case assesses the validity of SPARTA and illustrates the kinds of models that can be run on current desktop machines.

This test case contains high-speed low-density regions, low-speed high-density regions, laminar expanding regions, and a laminar recirculation zone. Thus, a DSMC simulation of the system includes a full spectrum of spatial and temporal variations. The flow conditions are for the Mach-15.6 nitrogen flow experiments performed by Holden *et al.*[35] The accommodation coefficient of nitrogen on the stainless-steel surface is taken to be 0.87, as suggested by Trott *et al.*[61] Nitrogen is simulated using the Variable Soft Sphere (VSS) collision model with Bird's suggested parameters.[7]

Unlike previous studies, which focused predominantly on the front area of the cone,[34,53] the computational domain here includes an extensive wake region, which introduces further complexity. Simultaneously simulating the forebody and wake regions increases computational requirements substantially because the density ratio between these regions exceeds two orders of magnitude. Figures 7(b) and 7(d) illustrate the hierarchical computational grid used for the simulations.

The flow is simulated using axisymmetric boundary conditions. The initial computational grid contains only 253 × 166 cells, but areas close to the surface and in the shock layers are refined up to 10 times by a factor of 2 in both coordinate directions at each level for a total of 1.22M (million) grid cells. The initial resolution can be seen in the cells in the wake. The simulations were performed on a desktop computer with a dual-socket Intel Haswell CPU (32 cores, 64 GB memory) using 340M particles and a time step of 2.5 ns. Individual simulations took ~12 h to run.

The flow field, including the wake, reaches steady state by 5 ms of physical time although the region around the forebody reaches steady state after only 1.5 ms. After reaching steady state, the simulation is averaged over the subsequent 1.5 ms of physical time to acquire adequate number of independent samples. Figure 7(a) shows the temperature profile of the flow field. A detached shock develops

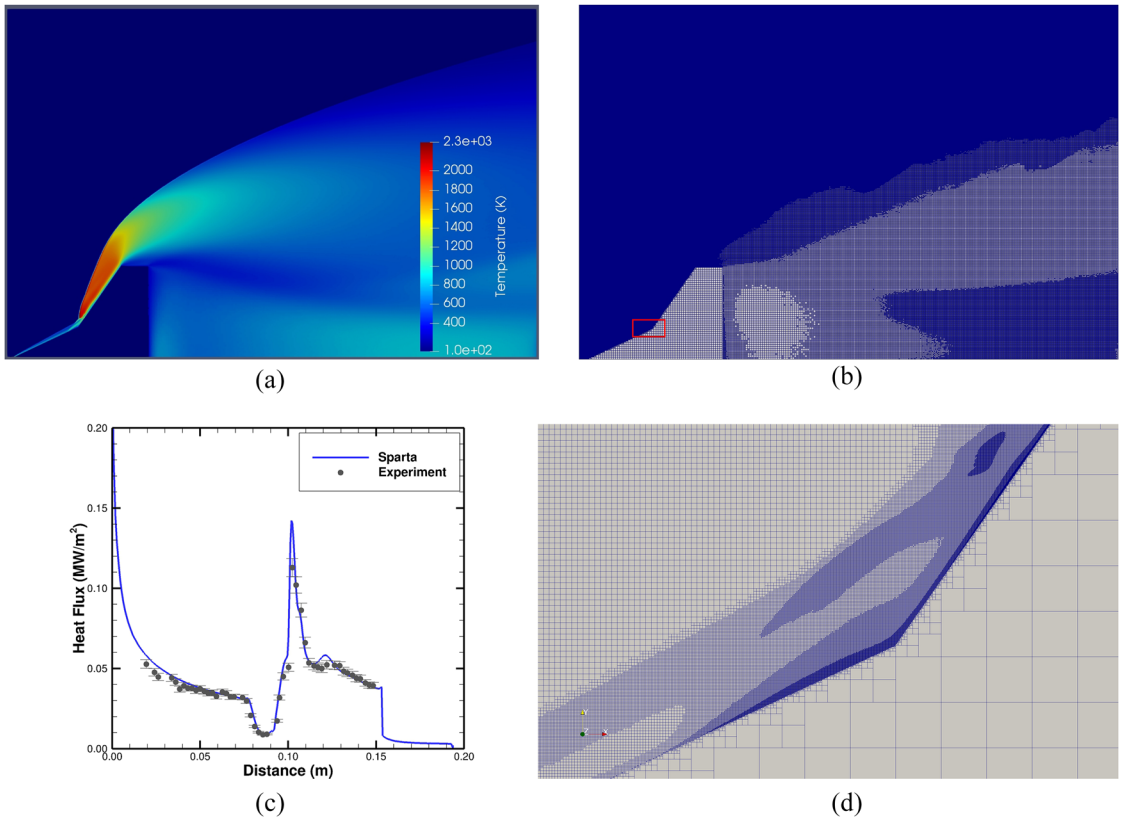


FIG. 7. Simulation results for axisymmetric flow over a 25°–55° biconic. (a) Translational temperature profile, (b) computational mesh for the entire domain with 11 hierarchical levels, (c) comparison of surface-heat-flux values from DSMC simulation and experiment, and (d) closeup of the computational mesh within the red rectangle of (b).

at the leading edge of the 25° cone and intersects a second, strong oblique shock that develops in front of the 55° cone. This interaction (an Edney Type-V[33] shock-shock interaction) creates a laminar ($\mathrm{Re} = 137\,000\ \mathrm{m}^{-1}$) recirculation zone in the region where the two cones intersect. The flow density increases to about 30 times its freestream value in front of the 25° cone and 50 times the freestream value in front of the 55° cone. The peak temperature is 2050 K and is located in the area of the shock-shock interaction. Downstream of the shock-shock interaction, the temperature gradually decreases until the end of the 55° cone, beyond which a 90° expansion rapidly cools the flow. In the wake behind the biconic, a toroidal eddy is observed, which occupies the roughly circular region of different mesh density seen in Fig. 7(b).

Figure 7(c) shows the DSMC heat-flux values on the surface of the biconic along with the corresponding experimental measurements of Holden *et al.*,[35] which have 5% error bars. Comparing the DSMC and experimental heat-flux values indicates that the DSMC simulation reproduces the measured heat flux with good accuracy at all locations on the surface. The heat flux on the rearward-facing surface of the biconic is also captured in this simulation and is much lower than the heat flux on the forward-facing surface.

B. 3D Rayleigh-Taylor instability

In this example, SPARTA is used to simulate baroclinic instabilities triggered by naturally occurring thermal fluctuations in gases, which are typically ignored in continuum fluid mechanics. Specifically, the Rayleigh-Taylor instability (RTI) at the interface between two gases is simulated.

The RTI occurs at the interface between two fluids of different densities subjected to a common acceleration normal to the interface. Small interfacial perturbations evolve into mushroom-like structures with wavelength λ and amplitude a, both of which increase as time progress. Taylor[59] conducted the original theoretical study of the RTI, and Lewis[46] conducted the companion experimental study. Since then, many sophisticated theoretical, numerical, and experimental studies have been conducted to quantify the behavior of the RTI. Extensive reviews of recent research on the RTI can be found in Refs. 1, 42, 57, 66, and 67.

Although the RTI has been modeled successfully with continuum hydrodynamic methods, RTI flow fields have also been simulated recently using molecular methods on large supercomputers. The applicability of molecular methods to instability simulations has been investigated by Kadau *et al.*,[38,39] Barber *et al.*,[2] and Mościński *et al.*[52] Their results suggest that, for liquids, molecular dynamics (MD) approaches can qualitatively and quantitatively describe the development of the RTI. In their results, the three stages of the instability are clearly observed as time progresses. The small initial perturbations evolve into larger spikes and bubbles, some showing turbulent shedding of vortices and eventually breaking completely apart. Kadau *et al.*[38,39] further showed good agreement with experimental data for spike positions and velocities.

Recently, it has been demonstrated that DSMC can simulate 2D baroclinic instabilities for the Richtmyer-Meshkov[24] and Rayleigh-Taylor instabilities.[25] Starting from a single-mode-perturbed interface, DSMC simulations have produced results in agreement with expectations from simulations, theory, and experiments.

For miscible fluids subject to a constant acceleration, the RTI can be described as having three fundamentally different phenomenological stages of mixing: linear, nonlinear, and turbulent. In the first stage, small initial perturbations on the interface grow exponentially. In this stage, the growth dynamics can be described by two length scales: the perturbation amplitude a in the acceleration direction and the perturbation wavelength λ, which is related to the mode of fastest growth (i.e., the most unstable wavelength λ_m). In the second stage, larger coherent structures appear as smaller disturbances interact and merge. These interactions occur when the perturbation amplitude becomes comparable to the perturbation wavelength. In this stage, the amplitude growth follows a power law with time. In the third stage, secondary instabilities such as the Kelvin-Helmholtz instability develop and eventually break up the coherent structures, which results in turbulent and chaotic mixing of the fluids. As the RTI develops further during this stage, the coherent structures begin to break down, and a self-similar turbulent mixing layer evolves.

The existence of a self-similar regime for the RTI was first discussed by Fermi and von Neumann.[21] In the turbulent regime, experimental data suggest that the bubble and spike growth rates are both self-similar and can be described by a relatively simple model,

$$a^{b,s} = \alpha^{b,s} A g t^2. \tag{1}$$

Here, A is the Atwood number for fluids with heavy and light densities $\rho_H > \rho_L$,

$$A = \frac{\rho_H - \rho_L}{\rho_H + \rho_L}. \tag{2}$$

Typically, α^b is a constant, usually between 0.03 and 0.07, but α^s depends on A. For the self-similar solution to be applicable, the instability amplitude must be greater than the viscous and diffusion scales. In practical terms, this means that the total instability amplitude $h = a^b + a^s$ must satisfy the condition,

$$\lambda_m \ll h \ll L, \tag{3}$$

where L is the size of the physical domain.

The details of the evolution of the RTI clearly depend on the properties of the two fluids. By broadening the density transition between the gases, viscosity, and diffusivity inhibit small-wavelength perturbations from growing, thereby allowing a particular wavelength (the most unstable wavelength λ_m) to emerge and grow more rapidly than all other wavelengths.[43] Diffusion can also reduce the effective buoyant forces, thus slowing the mixing process.[17]

As a result, for an initially flat interface, the most unstable wavelength has the following form:[57]

$$\lambda_m \approx 4\pi \left(\nu^2/Ag\right)^{1/3}, \tag{4}$$

where g is the gravitational acceleration and ν is the mean kinematic viscosity of the two fluids. Other factors that influence the growth of the instability include the dimensionality of the flow and boundary effects.

It is common to interpret the results in terms of length and time scales based on gravity and the initial perturbation wavelength. A typical time scale associated with the growth of the instability is $\tau = 1/\sqrt{Agk}$, where the wavenumber is related to the wavelength according to $k = 2\pi/\lambda$.

Experiments have confirmed that linear theory correctly accounts for the early-time evolution of the instability amplitude.[66] However, at later times, when $ka \approx 1$, the flow is more complicated and requires a nonlinear stability approach or an empirical analysis. The fluids mix at different rates on either side of the interface: the amplitude of the bubbles, the coherent structures that penetrate from the light fluid up into the heavy fluid, grows more slowly than the amplitude of the spikes, the coherent structures that penetrate down from the heavy fluid into the light fluid.

Here, in an attempt to better understand the physics of turbulent mixing, SPARTA is used to simulate the RTI in a 3D millimeter-scale domain for gases at atmospheric pressure. These simulations were performed on a third of the LLNL Sequoia machine; the largest simulations required 90 h.

The gases are dilute and obey the perfect-gas equation of state and are taken to have constant values of the specific heat ratio γ. These approximations are reasonable for the modest temperatures and pressures considered here. The initial conditions consist of argon (Ar), the heavier gas, on top of helium (He), the lighter gas. Initially, both gases are at 273.15 K and have density distributions corresponding to hydrostatic equilibrium throughout the domain, and the lighter gas has a pressure of 101 325 Pa at the bottom of the domain. The gases are subject to a constant gravitational acceleration of $g = 10^8$ m/s^2 in the downward direction. Due to the small length scales considered, the reason for using a large value for g is to enable the instability to develop within a reasonable number of time steps. The analysis that follows treats time as a dimensionless parameter; the behavior of the instability would be similar for an appropriately rescaled simulation.

The computational domain is a 3D box with a square cross section of 1 mm × 1 mm and a height of 4 mm. The interface is located at $z = 2$ mm. Periodic boundary conditions are used in the small dimensions (plane of the interface); the top and bottom of the long dimension are treated as diffuse reflective boundaries. The domain is gridded with 1.6B (billion) cubical cells with a side length of 50 nm, which is less than a mean free path at the interface for the gases at this pressure. The domain is initialized in hydrostatic equilibrium with an average of 12.5 particles per cell, for a total of 20B particles. Due to the initial density gradient, most of the particles are initially concentrated in the lower half of the domain. Pairwise collisions of particles are performed using the hard-sphere model with parameter values suggested by Gallis *et al.*[30] A time step of 0.1 ns is used, which is less than both the mean collision time and the mean transit time of particles in a cell. The instability growth is initially linear but ultimately becomes nonlinear.

Snapshots of the density as a function of time for a single large run are shown in Fig. 8. The initial mixing layer grows due to diffusion until an instability at the most unstable wavelength emerges. Although the interface is macroscopically flat initially, it is perturbed microscopically on the length scale of interparticle separations. As a result, in the absence of a macroscopic perturbation, molecular fluctuations become important to the development and growth of the instability. Perturbations with wavelengths near the most unstable wavelength grow the most quickly. However, if no unstable wavelengths are produced, the interface grows only by diffusion. The interface grows as the emerging bubbles and spikes become longer, begin interacting with each other, and eventually merge, which results in chaotic mixing.

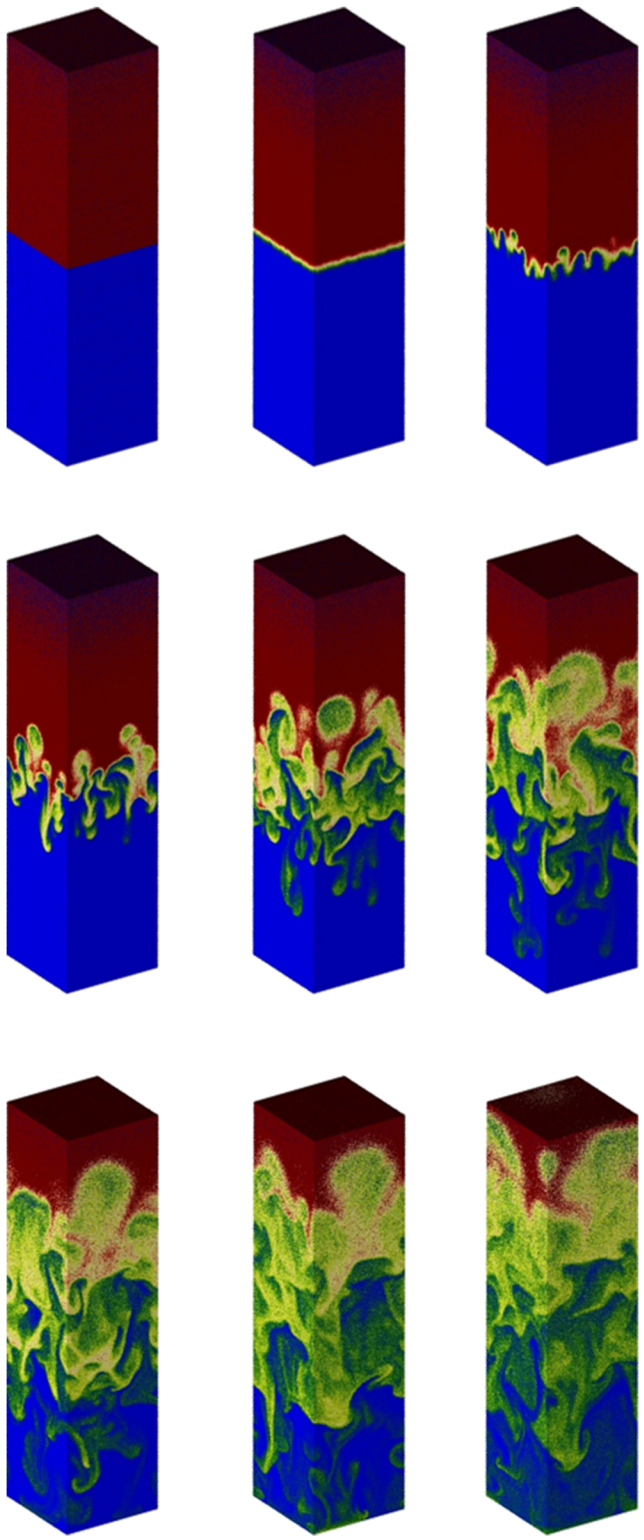


FIG. 8. Density profiles from a 3D DSMC RTI simulation advancing in time from left to right and then from top to bottom (3 μs between frames). Blue and red represent 10% and 90% of maximum density; other colors are intermediate densities.

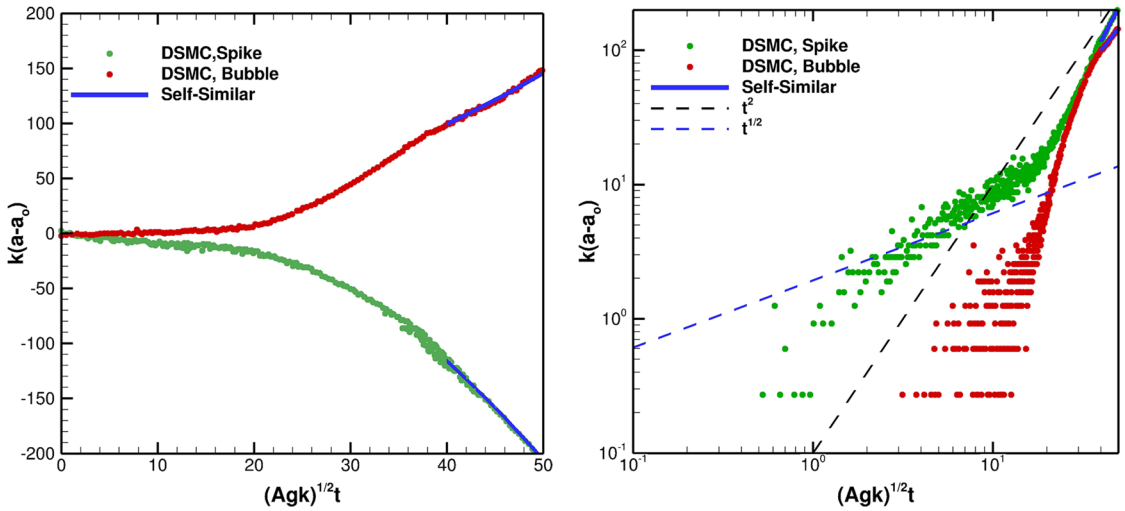


FIG. 9. Amplitude growth of bubbles and spikes for initially flat interfaces. The left plot is linear on the x and y axes; the right plot shows the same data on logarithmic scales.

Figure 9 plots the bubble and spike amplitudes as functions of time for a smaller RTI simulation where high-resolution image data were generated and archived to disk. These quantities are defined as the distances between the initial position of the interface and the highest point of helium penetration into argon (bubbles) and the lowest point of argon penetration into helium (spikes). The distances on the y-axis are normalized by the most unstable wavelength for the Ar/He mixture. The solid blue curves are quadratic fits in the self-similar regime. For this case, the self-similarity coefficients are $\alpha^b = 0.0515$ and $\alpha^s = 0.1000$, which are in reasonable agreement with experimental observations.[66]

This agreement, common among particle techniques that account for thermal fluctuations,[2,38,39,52] suggests that thermal fluctuations can have significant effects on the growth of the RTI mixing layer. The ability of DSMC to reproduce these fluctuations and their effects does not appear to be affected by the statistical nature of DSMC.

The right graphic in Figure 9 plots the absolute values of the same data as the left graphic in log-log format to better illustrate when the system transitions from the diffusive to self-similar regime. At early times, the growth is proportional to $t^{1/2}$, which is characteristic of growth by diffusion. At late times, the growth becomes proportional to t^2, as expected in the self-similar regime. The transition between these regimes occurs around $t\sqrt{Agk} \approx 40$, which this smaller simulation flow field just barely reaches.

The data for the plots of Fig. 9 are from high-resolution image snapshots created on-the-fly as discussed in Subsection II F. Individual particles were rendered in two colors for helium and argon. For this simulation, 285 snapshots were generated, one per 1000 time steps for an xz slice of the 3D domain, with thickness one tenth the width of the y dimension of the simulation box dimension. The snapshots were examined one-by-one to extract two data points per snapshot for high/low bubble/spike amplitudes. The spacing of the data points in the lower half of the right graphic in Figure 9 is due to the pixel-size resolution of the images.

C. Kármán vortex street

The effect of infinitesimal flow perturbations on the stability of a flow can also be investigated in the context of another canonical of fluid dynamics flow, namely, 2D flow past a transverse cylinder. This flow passes through a sequence of flow regimes of increasing complexity as the Reynolds number increases.[16,60] For $Re < 1$, the flow pattern is steady and symmetric. For $1 < Re < 40$, steady vortices form and remain attached to the trailing end of the cylinder. As the Reynolds number is increased, these vortices grow in size but remain steady and symmetric. For $Re > 40$, a Kelvin-Helmholtz instability appears in the separated boundary layer and propagates upstream, and vortices begin to shed from the trailing end of the cylinder in a regular periodic manner.

Even though the boundary conditions are steady, this instability arises spontaneously and can be described as an Andronov-Hopf bifurcation[23] that destroys symmetry and makes the flow time-periodic. In this regime, the flow remains laminar, but symmetry is lost. This flow pattern is the well-known Kármán vortex street of alternating vortices. As a vortex is being shed on one side of the cylinder, another vortex is formed on the other side. Upon release, the vortices convect downstream and gradually dissipate by viscous action. The Kármán vortex street persists up to $Re = 200$, above which value three-dimensional instabilities develop. At roughly $Re = 400$, low levels of turbulence start to appear within the vortices. At higher values, fully developed turbulence appears in the wake.

The vortex-shedding frequency of the Kármán vortex street has been extensively studied experimentally, numerically, and analytically.[23,60] As a result, accurate closed-form correlations exist for the Strouhal number $Sr = fD/U$ as a function of Reynolds number $Re = \rho UD/\mu$, where f is the shedding frequency, U is the freestream velocity, D is the cylinder diameter, and ρ and μ are the density and the viscosity of the fluid, respectively.[22]

The onset of the Kármán vortex street can be sensitive to disturbances and thus difficult to simulate with continuum codes, for

which disturbances must be artificially introduced. The stochastic nature of DSMC, due to the randomized nature of pairwise collisions, can excite instabilities directly. However, an open question is whether a DSMC simulation will produce the correct vortex shedding frequency.

Here, SPARTA is used to simulate 2D flow past a cylinder at Re = 100, at which value a Kármán vortex street is expected. The gas is argon, and the hard-sphere (HS) collision model is used. The freestream temperature is T = 273.15 K, which yields a viscosity of $\mu = 2.117 \times 10^{-5}$ Pa s and a sound speed of c = 307.9 m/s. The freestream Mach number is set to $Ma = U/c = 0.1$ to simulate nearly incompressible behavior, which yields a streamwise velocity of U = 30.79 m/s. The cylinder has a diameter of D = 1 m and has unity accommodation on its surface at the temperature T = 273.15 K. The freestream density is selected so that the Reynolds number is Re = 100, which yields a freestream density of $\rho = 6.876 \times 10^{-5}$ kg/m^3, a freestream pressure of p = 3.91 Pa, and a freestream mean free path of λ = 0.001 62 m. Thus, the cylinder Knudsen number is $Kn = \lambda/D = 0.001\,62$, so velocity slip at the cylinder surface is small.

The computational domain is rectangular and extends 5 cylinder diameters upstream, 10 diameters downstream, and 5 diameters in the spanwise direction (laterally) on both sides of the cylinder. The boundary conditions are freestream inflow on the upstream boundary, outflow on the downstream boundary, and periodic in the vertical dimension. The domain is meshed with 720M (million) square cells (32 863 × 21 908) whose side length is 1/4 of a molecular mean free path. The cylinder perimeter is discretized with 10 000 surface elements to ensure a smooth surface (a discretization with only 1000 surface elements yields almost identical results). Each cell contains ~100 particles, for a total of 72B particles in the domain. The time step $\Delta t = 1.4824\ \mu$s is set to be 1/3 of the molecular mean collision time. A series of simulations was run on a third of the LLNL Sequoia machine. Each 30-h run covered approximately two periods of vortex shedding; successive runs were restarted from the previous simulation's end point.

Figure 10 shows the streamwise velocity component at 8 equally spaced times covering one shedding cycle after the initial transient has decayed and shedding is well established. The expected Kármán vortex street is clearly observed downstream of the cylinder.

Figure 11 plots the spanwise component of the force per unit length on the cylinder due to the flowing gas. This quantity is determined by summing the momentum changes of molecules that reflect from the cylinder surface over time windows that are short

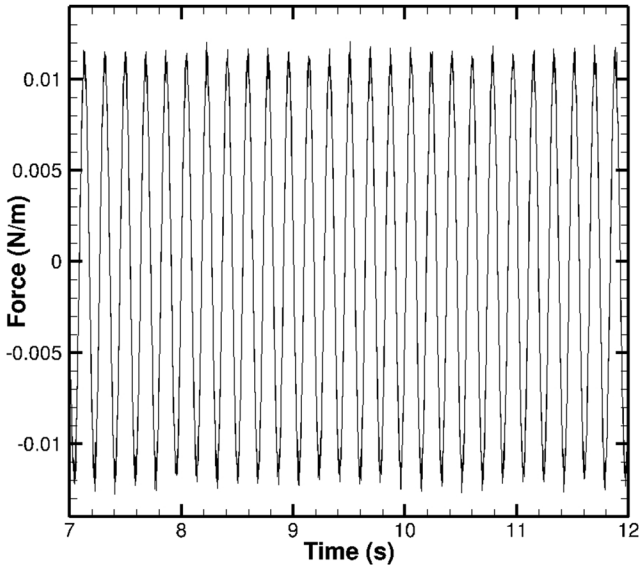


FIG. 11. DSMC spanwise force per unit length as a function of time at Re = 100.

compared to the shedding period. The periodic nature of the flow is clearly evident. Since many cycles can be observed, the shedding frequency and hence the Strouhal number can be determined accurately as f = 5.4 Hz and Sr = 0.175, respectively.

This value for the Strouhal number is about 6% higher than the value 0.165 obtained from the closed-form correlation[22] mentioned above. The slightly larger DSMC value is not related to any difference between molecular gas dynamics and continuum fluid dynamics. Instead, the difference results from the finite spanwise width of the computational domain; the correlation is for an infinite spanwise width. As evidence, we used the COMSOL Multiphysics code[14] to perform well-resolved incompressible Navier-Stokes simulations for several domains with a range of finite widths and found the same value for the Strouhal number as DSMC when the domain widths of the two simulations are the same.

This good agreement between DSMC and Navier-Stokes simulations strongly supports the assertion that the DSMC method is capable of accurately representing vortex shedding from solid surfaces, an important process in many turbulent flows. Moreover, this

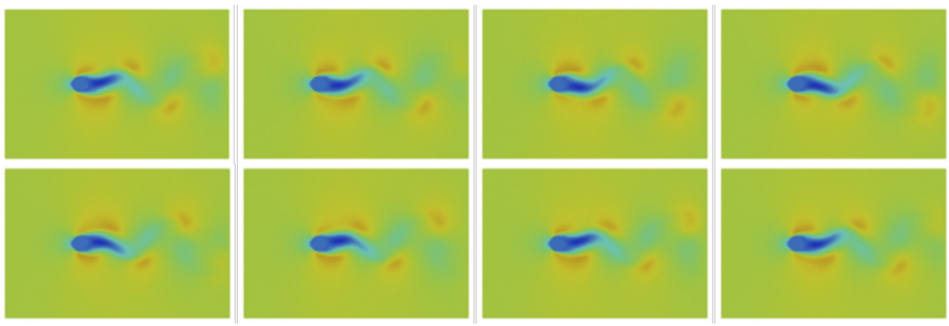


FIG. 10. Streamwise velocity component for Re = 100 at equally spaced time intervals over one shedding cycle. The Kármán vortex street is clearly seen.

process can be modeled in DSMC without the need to artificially perturb the system. This agreement suggests that statistical fluctuations, inherently present in DSMC simulations, do not affect the ability of DSMC to accurately predict such instabilities.

V. CONCLUSIONS

The Direct Simulation Monte Carlo (DSMC) method was introduced in the 1960s by Bird. His original motivation was to compute re-entry flow fields at transitional conditions between the free-molecular and continuum regimes, for which quantitative results could not be obtained from more traditional continuum approaches based on solving partial differential equations. Since then, the fundamental physics-based nature of the DSMC algorithm in conjunction with the ever-increasing power of modern computational platforms has allowed DSMC to model a wide range of problems outside its original regime of applicability.

In this paper, we have discussed parallel algorithms that have enabled our SPARTA code to run large-scale DSMC models on these large machines. We have illustrated that questions regarding the roles of thermal fluctuations and strong thermal nonequilibrium in the onset, development, and decay of turbulence and hydrodynamic instabilities can now be studied via DSMC at the molecular scale. Similarly, interesting aspects of hydrodynamic flows in the near-continuum regime can now also be modeled with DSMC. As computing power progresses in the next few years to the exascale, one area that could leverage the physical fidelity of the DSMC method and is thus potentially ripe for investigation is nonequilibrium effects on hydrodynamic flows when strong gradients are present.

We note that the goal of such DSMC simulations is not to replace continuum approaches in the study of hydrodynamic phenomena. Indeed, for many continuum and near-continuum systems, DSMC will never compete with continuum methods due to their accuracy and computational efficiency. Rather, the goal is to understand hydrodynamic phenomena with the intent of identifying and quantifying the role of molecular and stochastic effects, such as thermal fluctuations, and thermal and chemical nonequilibrium, such as induced by strong shocks. Subsequently, these effects can be included (e.g., as subgrid physics models) in continuum approaches, thereby improving their accuracy and extending their range of applicability for nonequilibrium flows.

ACKNOWLEDGMENTS

The authors acknowledge contributions to the Kokkos work in SPARTA by Tim Fuller and Dan Ibanez, and to the on-the-fly imaging algorithm by Nathan Fabian, all from Sandia. This work was supported by the Exascale Computing Project (ECP) (No. 17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration (NNSA) and by the Advanced Simulation and Computing (ASC) program of the NNSA. A.B. acknowledges funding from the NASA Entry Systems Modeling Project.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under Contract No. DE-NA0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

REFERENCES

[1]S. I. Abarzhi, "Review of theoretical modelling approaches of Rayleigh-Taylor instabilities and turbulent mixing," Philos. Trans. R. Soc., A **368**, 1809–1828 (2010).
[2]J. L. Barber, K. Kadau, T. C. Germann, P. S. Lomdahl, B. L. Holian, and B. J. Alder, "Atomistic simulation of the Rayleigh-Taylor instability," J. Phys.: Conf. Ser. **46**, 58–62 (2006).
[3]T. J. Bartel, S. J. Plimpton, and C. R. Justiz, "Direct Monte Carlo simulation of ionized rarefied flows on large MIMD parallel supercomputers," in *Proceedings of the 18th International Symposium on Rarefied Gas Dynamics* (AIAA, Vancouver, Canada, 1992), Vol. A94-30156, pp. 155–165.
[4]M. J. Berger and S. H. Bokhari, "A partitioning strategy for nonuniform problems on multiprocessors," IEEE Trans. Comput. **C-36**(5), 570–580 (1987).
[5]G. A. Bird, "Approach to translational equilibrium in a rigid sphere gas," Phys. Fluids **6**, 1518 (1963).
[6]G. A. Bird, *Molecular Gas Dynamics* (Clarendon Press, Oxford, UK, 1976).
[7]G. A. Bird, *Molecular Gas Dynamics and the Direct Simulation of Gas Flows* (Clarendon Press, Oxford, UK, 1994).
[8]G. A. Bird, The DSMC Method, Version 1.1, 2013.
[9]A. Borner, F. Panerai, and N. N. Mansour, "High temperature permeability of fibrous materials using direct simulation Monte Carlo," Int. J. Heat Mass Transfer **106**, 1318–1326 (2017).
[10]A. B. Bortz, M. H. Kalos, and J. L. Lebowitz, "New algorithm for Monte Carlo simulation of Ising spin systems," J. Comput. Phys. **17**(1), 10–18 (1975).
[11]See http://software.nasa.gov/software/ARC-17920-1 for Porous Microstructure Analysis (PuMA) code page on the NASA software website.
[12]See https://www.openfoam.com for OpenFoam CFD code website.
[13]See http://sparta.sandia.gov for SPARTA DSMC code website.
[14]COMSOL AB, COMSOL Multiphysics User's Guide, COMSOL AB, Stockholm, Sweden, 2008.
[15]L. Custodio, T. Etiene, S. Pesco, and C. Silva, "Practical considerations on marching cubes 33 topological correctness," Comput. Graphics **37**(7), 840–850 (2013).
[16]P. A. Davidson, *Turbulence* (Oxford University Press, 2015).
[17]R. E. Duff, F. H. Harlow, and C. W. Hirt, "Effects of diffusion on interface instability between gases," Phys. Fluids **5**(4), 417–425 (1962).
[18]H. C. Edwards, C. R. Trott, and D. Sunderland, "Kokkos: Enabling manycore performance portability through polymorphic memory access patterns," J. Parallel Distrib. Comput. **74**(12), 3202–3216 (2014).
[19]J. C. Ferguson, F. Panerai, A. Borner, and N. N. Mansour, "PuMA: The porous microstructure analysis software," SoftwareX **7**, 81–87 (2018).
[20]J. C. Ferguson, F. Panerai, F. Lachaud, A. Martin, S. Bailey, and N. N. Mansour, "Modeling the oxidation of low-density carbon fiber material based on micro-tomography," Carbon **96**, 57–65 (2016).
[21]E. Fermi and J. von Neumann, "Taylor instability of incompressible liquids. Part 1. Taylor instability of an incompressible liquid. Part 2. Taylor instability at the boundary of two incompressible liquids," Technical Report AECU-2979, Los Alamos Scientific Laboratory, Los Alamos, NM, 1953.
[22]U. Fey, M. König, and H. Eckelmann, "A new Strouhal-Reynolds-number relationship for the circular cylinder in the range $47 < Re < 2 \times 10^5$," Phys. Fluids **10**(7), 1547–1549 (1998).
[23]U. Frisch, *Turbulence* (Cambridge University Press, 1995).
[24]M. A. Gallis, T. P. Koehler, J. R. Torczynski, and S. J. Plimpton, "Direct simulation Monte Carlo investigation of the Richtmyer-Meshkov instability," Phys. Fluids **27**(8), 084105 (2015).

[25]M. A. Gallis, T. P. Koehler, J. R. Torczynski, and S. J. Plimpton, "Direct simulation Monte Carlo investigation of the Rayleigh-Taylor instability," Phys. Rev. Fluids **1**(4), 043403 (2016).
[26]M. A. Gallis, T. P. Koehler, J. R. Torczynski, S. J. Plimpton, and G. Papadakis, "Molecular-level simulations of turbulence and its decay," Phys. Rev. Lett. **118**(6), 064501 (2017).
[27]M. A. Gallis, T. P. Koehler, J. R. Torczynski, S. J. Plimpton, and G. Papadakis, "Gas-kinetic simulation of sustained turbulence in minimal Couette flow," Phys. Rev. Fluids **3**(7), 071402(R) (2018).
[28]M. A. Gallis and J. R. Torczynski, "Investigation of the ellipsoidal-statistical Bhatnagar-Gross-Krook kinetic model applied to gas-phase transport of heat and tangential momentum between parallel walls," Phys. Fluids **23**(3), 030601 (2011).
[29]M. A. Gallis, J. R. Torczynski, S. J. Plimpton, D. J. Rader, and T. Koehler, "Direct simulation Monte Carlo: The quest for speed," in *Proceedings of the 29th International Symposium on Rarefied Gas Dynamics* (AIP, Xi'an, China, 2014), Vol. 1628, pp. 27–36.
[30]M. A. Gallis, J. R. Torczynski, and D. J. Rader, "Molecular gas dynamics observations of Chapman-Enskog behavior and departures therefrom in nonequilibrium gases," Phys. Rev. E **69**(4), 042201 (2004).
[31]D. T. Gillespie, "A general method for numerically simulating the stochastic time evolution of coupled chemical reactions," J. Chem. Phys. **22**, 403–434 (1976).
[32]D. T. Gillespie, "Exact stochastic simulation of coupled chemical reactions," J. Phys. Chem. **81**(25), 2340–2361 (1977).
[33]P. A. Gnoffo, "CFD validation studies for hypersonic flow prediction," AIAA Paper 2001-1025, 2001.
[34]J. K. Harvey, "A review of a validation exercise on the use of the DSMC method to compute viscous/inviscid interactions in hypersonic flow," AIAA Paper 2003-3643, 2003.
[35]M. S. Holden, T. P. Wadhams, G. V. Candler, and J. K. Harvey, "Measurements in regions of low density laminar shock wave/boundary layer interaction in hypervelocity flows and comparison with Navier-Stokes predictions," AIAA Paper 2003-1131, 2003.
[36]M. S. Ivanov, S. F. Gimelshein, and G. N. Markelov, "Statistical simulation of the transition between regular and Mach reflection in steady flows," Comput. Math. Appl. **35**, 113–125 (1998).
[37]M. S. Ivanov, A. V. Kashkovsky, S. F. Gimelshein, G. N. Markelov, A. A. Alexeenko, Y. A. Bondar, G. A. Zhukova, S. B. Nikiforov, and P. V. Vaschenkov, "SMILE system for 2D/3D DSMC computations," in *Proceedings of the 25th International Symposium on Rarefied Gas Dynamics* (AIP, St. Petersburg, Russian Federation, 2006).
[38]K. Kadau, J. L. Barber, T. C. Germann, B. L. Holian, and B. J. Alder, "Atomistic methods in fluid simulation," Philos. Trans. R. Soc., A **368**, 1547–1560 (2010).
[39]K. Kadau, T. C. Germann, N. G. Hadjiconstantinou, P. S. Lomdahl, G. Dimonte, B. L. Holian, and B. J. Alder, "Nanohydrodynamics simulations: An atomistic view of the Rayleigh-Taylor instability," Proc. Natl. Acad. Sci. U. S. A. **101**(16), 5851–5855 (2004).
[40]A. Kashkovsky, "3D DSMC computations on a heterogeneous CPU-GPU cluster with a large number of GPUs," AIP Conf. Proc. **1628**, 192 (2014).
[41]A. G. Klothakis, I. K. Nikolos, T. P. Koehler, M. A. Gallis, and S. J. Plimpton, "Validation simulations of the DSMC code SPARTA," in *Proceedings of the 30th International Symposium on Rarefied Gas Dynamics* (AIP, 2016), Vol. 1786, p. 050016.
[42]F. Kull, "Theory of Rayleigh-Taylor instability," Phys. Rep. **206**(5), 197–325 (1991).
[43]D. Layzer, "On the instability of superposed fluids in a gravitational field," Astrophys. J. **122**(1), 1–12 (1955).
[44]G. J. LeBeau and F. E. Lumpkin III, "Application highlights of the DSMC analysis code (DAC) software for simulating rarefied flows," Comput. Methods Appl. Mech. Eng. **191**, 595–609 (2001).
[45]T. Lewiner, H. Lopes, A. Vieira, and G. Tavares, "Efficient implementation of marching cubes' cases with topological guarantees," J. Graphics Tools **8**(2), 1–15 (2003).
[46]D. J. Lewis, "The instability of liquid surfaces when accelerated in a direction perpendicular to their planes. II," Proc. R. Soc. London, Ser. A **202**(1068), 81–96 (1950).
[47]See https://github.com/kokkos/kokkos for Kokkos library website.
[48]See http://www.top500.org for Top 500 list website.
[49]W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," ACM SIGGRAPH Comput. Graphics **21**, 163–169 (1987).
[50]N. N. Mansour, F. Panerai, A. Martin, D. Y. Parkinson, A. MacDowell, T. Fast, G. Vignoles, and J. Lachaud, "A new approach to light-weight ablators analysis: From micro-tomography measurements to statistical analysis and modeling," in *44th AIAA Thermophysics Conference* (AIAA, 2013), p. 2768.
[51]C. Maple, "Geometric design and space planning using the marching squares and marching cube algorithms," in Proceedings of the 2003 International Conference on Geometric Modeling and Graphics, 2003.
[52]J. Mościński, W. Alda, M. Bubak, W. Dzwinel, J. Kitowski, M. Pogoda, and D. A. Yuen, *Molecular Dynamics Simulations of Rayleigh-Taylor Instability*, edited by D. Stauffer (World Scientific Publishing Company, Singapore, 1997).
[53]J. N. Moss and G. A. Bird, "Direct simulation Monte Carlo simulations of hypersonic flows with shock interactions," AIAA J. **43**, 2565–2573 (2005).
[54]I. Nompelis and T. E. Schwartzentruber, "Strategies for parallelization of the DSMC method," AIAA Paper 2013-1204, 2013.
[55]S. J. Plimpton, B. Hendrickson, and J. R. Stewart, "A parallel rendezvous algorithm for interpolation between multiple grids," J. Parallel Distrib. Comput. **64**, 266–276 (2004).
[56]T. J. Scanlon, E. Roohi, C. White, M. Darbandi, and J. M. Reese, "An open source, parallel DSMC code for rarefied gas flows in arbitrary geometries," Comput. Fluids **39**, 2078–2089 (2010).
[57]D. H. Sharp, "An overview of Rayleigh-Taylor instability," Physica D **12**(1-3), 3–18 (1984).
[58]E. C. Stern, S. Poovathingal, I. Nompelis, T. E. Schwartzentruber, and G. V. Candler, "Nonequilibrium flow through porous thermal protection materials, Part I: Numerical methods," J. Comput. Phys. **380**, 408–426 (2019).
[59]G. I. Taylor, "The instability of liquid surfaces when accelerated in a direction perpendicular to their planes," Proc. R. Soc. London, Ser. A **201**(1065), 192–196 (1950).
[60]D. J. Tritton, *Physical Fluid Dynamics* (Oxford Science Publications; Clarendon Press, Oxford, 1988).
[61]W. M. Trott, D. J. Rader, J. N. Castaneda, J. R. Torczynski, and M. A. Gallis, "Measurements of gas-surface accommodation," Rev. Sci. Instrum. **82**(3), 035120 (2011).
[62]W. Wagner, "A convergence proof for Bird's direct simulation Monte Carlo method for the Boltzmann equation," J. Stat. Phys. **66**(3-4), 1011 (1992).
[63]K. Weiler and P. Atherton, "Hidden surface removal using polygon area sorting," ACM SIGGRAPH Comput. Graphics **11**, 214–222 (1977).
[64]C. White, M. K. Borg, T. J. Scanlon, S. M. Longshaw, B. John, D. R. Emerson, and J. M. Reese, "dsmcFoam+: An OpenFOAM based direct simulation Monte Carlo solver," Comput. Phys. Commun. **224**, 22–43 (2018).
[65]C. Zhang and T. E. Schwartzentruber, "Robust cut-cell algorithms for DSMC implementations employing multi-level Cartesian grids," Comput. Fluids **69**, 122–135 (2012).
[66]Y. Zhou, "Rayleigh-Taylor and Richtmyer-Meshkov instability induced flow, turbulence, and mixing. I," Phys. Rep. **720-722**, 1–136 (2017).
[67]Y. Zhou, "Rayleigh-Taylor and Richtmyer-Meshkov instability induced flow, turbulence, and mixing. II," Phys. Rep. **723-725**, 1–160 (2017).