GPU acceleration of an iterative scheme for gas-kinetic model equations with memory reduction techniques

Lianhua Zhu[a,*], Peng Wang[a], Songze Chen[b], Zhaoli Guo[b], Yonghao Zhang[a]

[a]*James Weir Fluids Laboratory, Department of Mechanical and Aerospace Engineering, University of Strathclyde, Glasgow G1 1XJ, UK*
[b]*State Key Laboratory of Coal Combustion, School of Energy and Power, Huazhong University of Science and Technology, Wuhan, 430074, China*

Abstract

This paper presents a Graphics Processing Unit (GPU) acceleration of an iteration-based discrete velocity method (DVM) for gas-kinetic model equations. Unlike the previous GPU parallelization of explicit kinetic schemes, this work is based on a fast converging iterative scheme. The memory reduction techniques previously proposed for DVM are applied for GPU computing, enabling full three-dimensional (3D) solutions of kinetic model equations in the contemporary GPUs usually with a limited memory capacity that otherwise would need terabytes of memory. The GPU algorithm is validated against the direct simulation Monte Carlo (DSMC) simulation of the 3D lid-driven cavity flow and the supersonic rarefied gas flow past a cube with the phase-space grid points up to 0.7 trillion. The computing performance profiling on three models of GPUs shows that the two main kernel functions can utilize 56% $\sim$ 79% of the GPU computing and memory resources. The performance of the GPU algorithm is compared with a typical parallel CPU implementation of the same algorithm using the Message Passing Interface (MPI). The comparison shows that the GPU program on K40 and K80 achieves 1.2 $\sim$ 2.8 and 1.2 $\sim$ 2.4 speedups for the 3D lid-driven cavity flow, respectively, compared with the MPI parallelized CPU program running on 96 CPU cores.

Keywords: GPU, CUDA, Discrete Velocity Method, Gas-Kinetic Equation, High Performance Computing

1. Introduction

Rarefied gas flows appear in wide ranges of application areas such as the space vehicles re-entery, vacuum sciences, and Microelectromechanical systems.

The dominant numerical method for rarefied gas flows remains to be the Direct Simulation Monte-Carlo (DSMC) method [1, 2] which can be viewed as a stochastic approach to solving the Boltzmann equation. However, with the significantly improved computing power nowadays, the deterministic methods solving the Boltzmann equation or its model equations are gaining popularity [3, 4]. These deterministic approaches include the discrete ordinate method (DOM) and the discrete velocity method (DVM), which were proposed earlier [5, 6] but have largely been limited to one- or two-dimensional (2D) simulations. Compared with the DSMC method, these deterministic methods are free from statistical noise, thus they are particularly preferable for low speed flows [3, 7–11]. More importantly, the deterministic approach has greater flexibility in designing efficient numerical schemes, e.g., asymptotic-preserving schemes [12–20], implicit schemes [21–26] and high-order schemes [27–30].

However, even with advanced numerical algorithms and simplified kinetic models, direct simulations of practical 3D problems using DVM are still computationally expensive regarding both the floating-point operations and the memory requirement, due to the sheer number of grid points in the phase space including the physical space and the molecular velocity space. For practical simulations, especially those involving high speed flows, massively parallel supercomputing facilities are indispensable [31, 22, 32–37]. Various techniques have been proposed to reduce the grid points, e.g., the adaptive local refinement on the spatial and velocity grids [38–42]. However, efficient parallelization on such non-regular grids can be a challenge especially for dynamic adaptive mesh refinement (AMR), where sophisticated load-balancing techniques are necessary. We note that for the lattice Boltzmann method (LBM), a special DVM scheme with minimum discrete velocity set, the link-wise artificial compressibility method (LWACM) has been proposed as an alternative approach that can eliminate the need to store the discrete distribution function [43, 44]. But this approach is specially designed for continuum flows and deals with only macroscopic variables instead of molecular distribution functions. In this article, we mainly discuss the computational challenges of the general DVM simulation of rarefied gas flows.

The enormous computing cost of deterministic solvers for the gas-kinetic equations makes the adoption of emerg-

*Corresponding author
Email addresses: `l.zhu@strath.ac.uk` (Lianhua Zhu), `yonghao.zhang@strath.ac.uk` (Yonghao Zhang)

ing heterogeneous parallel computing platforms such as GPU and Intel Phi co-processors appealing. Programs executed on such platforms are written with specialized programming frameworks such as CUDA and OpenCL. Computationally intensive parts of the algorithms are offloaded to the accelerators which have onboard memory, and the computed results are copied back to the CPU memory. GPU, in particular, has achieved great success in the grid-based LBM [45–51] and particle-based DSMC method [52–54]. For more general DVM simulations, GPUs have not been fully exploited [55–61]. Frezzotti et al. reported a parallelization of a BGK-equation solver using an explicit DVM scheme and evaluated its performance on an Nvidia GTX-260 GPU [55]. The reported speedup is over 300 compared with a serial program on a contemporary CPU for the 2D lid-driven cavity flow with 3D velocity grid configuration (we use the abbreviation like 2D3V for 2D physical space and 3D molecular velocity space hereinafter). However, the maximum grid size was limited to $160^2 \times 20^3$ as there was only 896 MB GPU memory. Later, they extended the implementation to the full Boltzmann equation, achieving a speedup over 400 [56] on the same device with the grid size up to $192^2 \times 18^3$. Kloss et al. reported a GPU accelerated solver for the full Boltzmann equation with a conservative projection method and achieved a speedup up of 150 with a maximum grid size of $160^2 \times 20^3$ [58]. Overstays and Crocke presented a similar GPU implementation to solve the Boltzmann equation and achieved an overall speedup around 50 on a low-end GPU [61] with the maximum grid size being $128 \times 64 \times 18 \times 18 \times 9$. Zabelok et al. demonstrated a GPU acceleration of the 3D3V DVM module in their Unified Flow Solver (UFS) with a grid of 20000×32^3 on a single GPU with the help of AMR in 3D physical space. Their reported speedup is $20 \sim 50$ [60]. The above literature reveals that except for Zabelok's work [60], all of the implementations of GPU accelerated DVM were restricted to 2D problems, and the primary reason is the limited memory size on a single GPU.

Even though the global memory capacities on contemporary GPU devices have been growing, they are still relatively small ($4 \sim 16$ GB) compared with the CPU nodes and offer much fewer FLOPS. The memory size restriction can be technically mitigated with multi-GPU implementations, but the high-volume data transfer between multiple GPU devices may lead to new difficulties [62]. Another common feature of the above implementations is that they all use explicit discretization schemes for the convection term, which makes the stencil computations more suitable than an implicit/iterative scheme on GPUs because each computation stencil evolves only the old-time-step distribution function of neighbor grid nodes. The stencil computations of all spatial nodes in a time step can be executed concurrently independent of each other. However, for steady-state rarefied gas flows, we usually prefer implicit or iterative DVM schemes; unlike explicit schemes, the time steps in implicit schemes are not restricted by the Courant–Friedrichs–Lewy (CFL) condition [21, 63, 23, 25]. In iterative schemes for the steady-state gas kinetic equation, the boundary information can propagate quickly into the whole domain, achieving fast convergence [64, 26]. However, their parallelization is hindered by the data dependence in the physical space in a single time step (iteration), especially on massively fine-grained parallel computing devices like GPUs.

Here, we propose a GPU acceleration algorithm to solve a Boltzmann model equation [65] with memory reduction techniques that are well known to the rarefied gas flow community but have never been applied on GPUs [66]. The aim is to address the two issues discussed above, i.e., the GPU memory size barrier and the difficulty in parallelizing the iterative/implicit scheme on a GPU. In this algorithm, the memory reduction technique in the velocity space is applied to reduce the number of discrete distribution functions to be stored. The storage in the physical space is further reduced from 3D to essentially 2D by allocating storage space of only two slices of spatial cells in the first two dimensions, and reuse the storage space repeatedly when marching along the third dimension.

With these two techniques, the required memory space can be significantly reduced. In addition, the data dependence in the physical space can be avoided by parallelizing in the molecular velocity space with each GPU thread mapped to one discrete velocity, which is different from all the previous implementations [55, 60] on GPUs. We note this parallelization strategy has been routinely used in the conventional CPU platform.

Combination of the memory reduction techniques and the GPU parallelization enables full 3D solution of kinetic model equations on a single GPU. This approach is expected to greatly accelerate the DVM simulations of high-speed flow where more discrete velocities are needed or the low-speed flows such as thermally induced flow where the DSMC method is very inefficient.

The remaining of the paper is organized as follows. In Sec. 2, we introduce the Shakhov model equation [65] to demonstrate our algorithm and explain how the memory occupation can be dramatically reduced in both the molecular velocity and physical spaces. In Sec. 3 we give a simple introduction to the GPU programming paradigm using the CUDA framework and present the algorithms. In Sec. 4, the implemented GPU programs are verified with the lid-driven cubic cavity flow and the supersonic flow past a cube. In Sec. 5, the parallel performance of the GPU programs is profiled on several different GPU models, and the speedup is compared with an MPI parallelization. The performance bottlenecks at various conditions are also identified. Section 6 concludes the article with our main findings and discussions for further development.

2. The numerical method

2.1. The Shakhov model equation

The parallel implementation is based on an iterative scheme for the steady-state Boltzmann model equation. In this study, we use the Shakhov model [65] to demonstrate our implementation which is also applicable to other model equations. The governing equation of the velocity distribution function $f(\boldsymbol{x}, \boldsymbol{\xi})$ reads as

$$\boldsymbol{\xi} \cdot \boldsymbol{\nabla} f = -\frac{f - f^S}{\tau}, \tag{1}$$

where τ is the relaxation time and is related to the local viscosity μ and pressure p by $\tau = \mu/p$. f^S is defined as

$$f^S = f^M \left[1 + (1 - \mathrm{Pr})\frac{\boldsymbol{c} \cdot \boldsymbol{q}}{5pRT}\left(\frac{c^2}{RT} - 5\right)\right], \tag{2}$$

with

$$f^M = \frac{\rho}{(2\pi RT)^{3/2}} \exp\left(-\frac{c^2}{2RT}\right), \tag{3}$$

where Pr is the Prandtl number and $\boldsymbol{c} = \boldsymbol{\xi} - \boldsymbol{U}$ is the peculiar velocity with $\boldsymbol{U}$ being the hydrodynamic velocity; c is the magnitude of $\boldsymbol{c}$. For a monatomic gas, Pr equals to 2/3. The macroscopic variables such as the density ρ, velocity $\boldsymbol{U}$, temperature T, and heat flux $\boldsymbol{q}$ can be calculated from the moments of the distribution function,

$$\rho = \int f \mathrm{d}\boldsymbol{\xi}, \tag{4}$$

$$\rho \boldsymbol{U} = \int \boldsymbol{\xi} f \mathrm{d}\boldsymbol{\xi}, \tag{5}$$

$$\rho E = \frac{1}{2}\int \xi^2 \mathrm{d}\boldsymbol{\xi}, \tag{6}$$

$$\boldsymbol{q} = \frac{1}{2}\int \boldsymbol{c} c^2 f \mathrm{d}\boldsymbol{\xi}, \tag{7}$$

where $\rho E = 1/2\rho U^2 + C_v T$ is the total energy with C_v being the heat capacity [$(3/2)R$ for monatomic gases]. The pressure is related to the density and temperature by $p = \rho RT$.

In DVM, the molecular velocity space is first discretized with a chosen 3D velocity grid $\{\boldsymbol{\xi}_\alpha | \alpha = 1, 2, ..., M\}$. The discrete form of the governing equation is then expressed as

$$\boldsymbol{\xi}_\alpha \cdot \boldsymbol{\nabla} f_\alpha = -\frac{1}{\tau}[f_\alpha - f_\alpha^S], \tag{8}$$

where f_α is the distribution function of $\boldsymbol{\xi}_\alpha$ and f_α^S is the Shakhov-corrected equilibrium. The macroscopic variables are evaluated by taking numerical integrations of f_α as follows,

$$\rho = \sum_\alpha w_\alpha f_\alpha, \tag{9}$$

$$\rho \boldsymbol{U} = \sum_\alpha w_\alpha \boldsymbol{\xi}_\alpha f_\alpha, \tag{10}$$

$$\rho E = \frac{1}{2}\sum_\alpha w_\alpha (\xi_{\alpha,x}^2 + \xi_{\alpha,y}^2 + \xi_{\alpha,z}^2) f_\alpha, \tag{11}$$

$$\boldsymbol{q} = \frac{1}{2}\sum_\alpha w_\alpha \boldsymbol{c}_\alpha \left(c_{\alpha,x}^2 + c_{\alpha,y}^2 + c_{\alpha,z}^2\right) f_\alpha, \tag{12}$$

where w_α are the coefficients for the numerical quadratures.

2.2. Iteration scheme with memory reduction technique

Here we use an iteration scheme to solve the discrete-velocity version of the governing equation, i.e. Eq. (8). In this scheme we only need to store the distribution function of one discrete velocity, thus dramatically reducing the memory requirement from 6D to 3D for a full 3D problem. In the following, we first introduce the iterative scheme and then discuss its favorable parallelization approaches.

Equation (8) is solved using the following scheme,

$$\boldsymbol{\xi}_\alpha \cdot \boldsymbol{\nabla} f_\alpha^{n+1} = -\frac{1}{\tau^n}[f_\alpha^{n+1} - f_\alpha^{S,n}], \tag{13}$$

where n denotes the index of iteration steps. The f_α^S calculated from macro variables is treated explicitly. Given $f_\alpha^{S,n}$ or equivalently the moments at the nth iteration step, and assuming the spatial gradient is evaluated with an upwind scheme, the f_α^{n+1} can be updated by a spatial sweep sequentially along the characteristic direction for each discrete velocity [66, 64]. In the original work of Ref. [64], the authors proposed to rewrite the above formula in a delta form and use a central scheme to attain 2nd-order accuracy, while introducing an inner sub-iteration loop in each iteration to recover the old distribution function. Introducing the inner sub-iteration loop increases the overall computing time, and the more complex stencil computation makes the method more difficult to parallelize. In this work, we employ a 1st-order upwind scheme for the gradient discretization; so the above difficulty is avoided albeit with the expense of lower accuracy. The accuracy of the 1st-order scheme will be formally evaluated in Sec. 4.

At the beginning of the iteration, $f_\alpha^{S,0}$ is set to the equilibrium state based on the initial (usually uniform) macroscopic fields. Then the spatial field of the distribution function for each discrete velocity is updated in a spatial sweeping manner on a Cartesian grid. The computational stencil is determined by the signs of the discrete velocity components. For example, assuming we are currently processing the discrete velocity $\boldsymbol{\xi}_\alpha$ with $\xi_{\alpha,\{x,y,z\}} > 0$, when the spatial sweep approaches to an inner cell with spatial index (i, j, k), the distribution function $f_{\alpha,i,j,k}^{n+1}$ is updated from the following 1st-order upwind discretization formula:

$$\xi_{\alpha,x}\left(\frac{f^{n+1}_{\alpha,i,j,k}-f^{n+1}_{\alpha,i-1,j,k}}{\Delta x_i}\right)+\xi_{\alpha,y}\left(\frac{f^{n+1}_{\alpha,i,j,k}-f^{n+1}_{\alpha,i,j-1,k}}{\Delta y_j}\right)$$
$$+\xi_{\alpha,z}\left(\frac{f^{n+1}_{\alpha,i,j,k}-f^{n+1}_{\alpha,i,j,k-1}}{\Delta z_k}\right)=-\frac{1}{\tau^n_{i,j,k}}\left(f^{n+1}_{\alpha,i,j,k}-f^{S,n}_{\alpha,i,j,k}\right), \tag{14}$$

in which $f^{n+1}_{i-1,j,k}, f^{n+1}_{i,j-1,k}, f^{n+1}_{i,j,k-1}$ are already known because they are information on the upwind cells and have been calculated before we process the cell (i, j, k).

Because the same sweeping procedure for each discrete velocity can be executed individually without interaction with the other discrete velocities, we can allocate the memory space for a single discrete velocity and process all the discrete velocities sequentially using the same storage space. Once the whole field of the distribution function of a specific discrete velocity is updated, its contributions are added up to the moments of the new iteration step. In this way, the same memory space is used repeatedly for all the discrete velocities. After all discrete velocities have been processed, the updating of new moments is also completed. The iteration is repeated until it converges by the following definition,

$$\epsilon^{n+1} \equiv \frac{\sqrt{\sum_i \left[(M_i^{n+1})^2-(M_i^n)^2\right]}}{\sum_i (M_i^n)^2} < \epsilon_c, \tag{15}$$

where $M \in \{\rho, \rho\boldsymbol{U}, \rho E\}$, ϵ_c is the convergence criterion, and the summations take over all of the cells. Using double-precision floating-point calculations, the ϵ^n can decrease to zero by machine precision as the iteration continues. But we stopped the iterations by setting $\epsilon_c = 1 \times 10^{-9}$, because the changes in the macro variable fields with the number of iterations are already sufficiently small under this setting. Boundary conditions are processed when the sweeping starts or approaches the domain boundaries. The overall algorithm of the iterative scheme is illustrated in Algorithm 1.

By analyzing the data dependence in the algorithm, we can find the best parallelization strategy. Clearly, due to the directional spatial sweep, each stencil computation depends on its upwind-cell information; thus, it is not straightforward to parallelize the sweeping in the physical space. In the traditional multi-core CPU approach, the classical domain decomposition in the physical space (multi-block parallelization) can partially solve this problem where the deterioration of convergence caused by the desynchronization between blocks is negligible due to the coarse granularity of the decomposition [22, 37]. However, with the fine-grained GPU architecture, the desynchronization can dramatically deteriorate the convergence rate. One possible strategy is to decompose the domain among the diagonal sweeping wavefront, but it requires careful consideration for load balance and the parallelism is limited by the relative small wavefront size [67–69]. As

Algorithm 1 The algorithm of the iterative scheme

```
Allocate the $f^{n+1}$ and $f^n$ fields for a single discrete velocity.
Initialize $\rho^n$, $\boldsymbol{U}^n$, $T^n$ as uniform fields.
$\epsilon^{n+1} \leftarrow 1.0$.
while $\epsilon^{n+1} < \epsilon_c$ do
    Initialize $\rho^{n+1}$, $(\boldsymbol{\rho}U)^{n+1}$, $(\rho E)^{n+1}$ as
        zero moments fields.
    for $\boldsymbol{\xi}_\alpha \in \{\boldsymbol{\xi}_\alpha | \xi_{\alpha,\{x,y,z\}} > 0\}$ do
        for $k \leftarrow 1,\mathtt{NZ}$; $j \leftarrow 1,\mathtt{NY}$; $i \leftarrow 1,\mathtt{NX}$ do
            Handling boundary condition when any of
                {i, j, k} starts from boundaries.
            Calculate $f^{S,n}_\alpha$ from Eq. (2).
            Update $f^{n+1}_{\alpha,i,j,k}$ by Eq. (14).
            Add $f^{n+1}_{\alpha,i,j,k}$'s contribution to moments.
                $\rho^{n+1}$, $(\rho U)^{n+1}$, $(\rho E)^{n+1}$ by Eq. (9).
            Handling boundary condition when any of
                {i, j, k} reaches boundaries.
        end for
    end for
    for $\boldsymbol{\xi}_\alpha \in \{\boldsymbol{\xi}_\alpha | \xi_{\alpha,\{x,y\}} > 0, \xi_{\alpha,z} < 0\}$ do
        ...                                   ▷ Spatial sweep.
    end for
    ...              ▷ Other 6 octants of discrete velocities.
    Calculate $\boldsymbol{U}^{n+1}$, $T^{n+1}$ from $\rho^{n+1}$, $(\rho U)^{n+1}$,
        $(\rho E)^{n+1}$.
    Calculate $\epsilon^{n+1}$ by Eq. (15).
    $n \leftarrow n + 1$.
end while
```

the sweeping of each discrete velocity is independent of each other, the parallel computing in the molecular velocity space is much simpler. The communication only occurs at the stage of moments calculations. The parallelization in the molecular velocity space with Message Passing Interface (MPI) is straightforward in the same way as reported in the literature including Refs. [3, 31, 66, 64, 70] et al. In these MPI implementations, each MPI process handles a subset of discrete velocities, and the MPI communications occur only during the calculation of the moments by global reductions of the partial moments pre-calculated on each MPI process. The total memory will increase with the number of parallel processes or threads. For MPI with distributed memory (multi-node) parallelization, the total memory requirement will not be an issue. However, for GPU parallelization with several thousands of tiny threads, the total memory occupation can be considerable and may exceed the memory capacity. We will show a memory reduction technique in the following section to tackle this problem.

3. GPU implementations

3.1. GPU programming introduction

GPU enables massively parallel computing supported by the advances in both hardware and software. A single GPU board nowadays contains thousands of light-weight streaming processors (SPs or cores) grouped into dozens of streaming multiprocessors (SMs) and it can provide several teraflops computing power. The GPU hardware also comes with a hierarchical system of memories, registers, and caches to enable high-bandwidth and low-latency feeding of data to the processors. Meanwhile, there are application programming interface (API) models that support general purpose computing on GPU devices by extensions of the popular programming language such as C++. The most widely adopted GPU programming API model is Nvidia's CUDA, the others include the OpenCL and DirectCompute. Since in this work, we implement our algorithm using CUDA, here we briefly explain the related concepts in the CUDA environment. Note that the algorithms presented here can also be implemented in other frameworks like OpenCL in principle since the programming metrologies in the different frameworks are similar [71] and we do not use any exclusive CUDA feature.

CUDA employs a hierarchical organization of threads. The fine-grained computing tasks are mapped to a logical grid of threads. The thread grid is divided into thread blocks, each contains the same number of threads and are dynamically scheduled to the SMs. The threads execute special functions called *kernels*. To realize the full potential of GPU, the data and thread organization and kernel function implementations have to follow a few principles, including:

- keeping memory accesses coalesced in a half-thread warp (consecutive 16 threads);
- avoiding frequent data transfer between the CPU and GPU ends;
- using the on-chip shared memory to make the threads in a thread block work together effectively;
- limiting the usage of registers and shared memory in the kernels to ensure that enough thread blocks can reside on SMs (quantified as SM occupancy commonly).

The major floating-point operations in the iterative scheme described in Sec. 2 come from the spatial-sweep and moment-evaluation procedures, i.e. Eqs. (14) and (9) respectively. For each procedure, we implement an individual kernel function because the two procedures have different data dependence patterns. The spatial-sweep procedure has data dependence only in the physical space while the moment-evaluation procedure only in the velocity space. The two kernels are explained in detail in Sec. 3.2 and Sec. 3.3.

3.2. Spatial-sweep kernel with further memory reduction

As explained above, the spatial-sweep procedure is intrinsically sequential in the physical space while each discrete velocity is independent of the other discrete velocities in the molecular velocity space. Therefore, for the sweeping procedure, we map each thread to one discrete velocity. The thread grid size can be set as a fraction of the total number of discrete velocities as we can process the discrete velocities with a batch-by-batch manner (see Fig. 1 and the main iteration procedure in Algorithm 2). But it should still be sufficiently large to make sure there are enough thread blocks to populate the SMs. One natural choice of the batch size is an octant of the discrete velocity grid as there are eight sweeping directions from the eight corners of the domain. The required memory is proportional to the thread grid size. For modern GPU containing thousands of cores, this requirement will exceed the global memory capacity. Here we take Nvidia's Tesla K40 GPU as an example, which has 2880 cores and 12GB memory. When simulating a problem with the physical grid size of 128^3 using single-precision float, the modest estimation of the storage space exceeds $128^3 \times 2880 \times 4/2^{30} = 22.5\text{GB} > 12\text{GB}$. Therefore, it is necessary to reduce the memory occupation further.

To this end, we employ the idea presented in Ref. [66] to store the distribution function of only two slices of cells (as illustrated in Fig. 2). We note that when updating the newer-step distribution function using a 1st-order upwind scheme, all stencil points reside in the current slice and the upwind slice [refer to Eq. (14)]. The storage space is reused when marching along the Z-direction. Such a configuration reduces memory consumption dramatically. Take the simulation with an N^3-by-M^3 phase-space grid for example. The modest estimation of the storage in a single-precision conventional implementation would be $4N^3M^3$ bytes. With the memory reduction techniques, and assuming the velocity grid batch size is B (refer to Fig. 1), the memory occupation will be $4 \times 2N^2B$ bytes, where the "2" comes from the "two" slices of cells. In this work, we choose B to be $(M/2)^3$ as illustrated in Fig. 1. Thus, the memory reduction is $4N$ times. The batch size B can be even smaller for larger velocity grid size then the reduction can be even more significant. The layout of the thread grid is illustrated in Fig. 3 and the kernel function implementation is detailed in Algorithm 3.

3.3. Moment-evaluation kernel

Each time after the distribution functions of the cells in one slice are updated, the moment-evaluation kernel adds up their contributions to the temporary moment variables. As the summations are local operations in the physical space, we map each thread block to one cell in the 2D slice. All the threads in the same block work collaboratively using the shared memory, to sum up the distribution functions. The layouts of the thread grids and blocks are illustrated in Fig. 4. The moment-evaluation kernel

Algorithm 2 Main iteration procedure

```
function ITERATION
    for $d \leftarrow 0, 7$ do                                    ▷ Loop through the 8 discrete velocity groups
        Setting parameters for the $d$th velocity group
        if $d > 3$ then         ▷ For the velocity groups above the $\xi_z = 0$ plane in the molecular velocity space
            for $k \leftarrow 1$, NZ do                                         ▷ Sweeping along the $Z^+$ direction
                SWEEP_KERNEL($d$, $k$, fSliceA, fSliceB, momentsOld, ...)
                MOMENT_KERNEL($d$, $k$, fSliceA, fSliceB, momentsNew, ...)
            end for
        else                    ▷ For the velocity groups below the $\xi_z = 0$ plane in the molecular velocity space
            for $k \leftarrow$ NZ, 1 do                                         ▷ Sweeping along the $Z^-$ direction
                SWEEP_KERNEL($d$, $k$, fSliceA, fSliceB, momentsOld, ...)
                MOMENT_KERNEL($d$, $k$, fSliceA, fSliceB, momentsNew, ...)
            end for
        end if
    end for
    SWAP_MOMENT_KERNEL(momentsOld, momentsNew )                        ▷ Swapping the moments variables
end function
```

Algorithm 3 Spatial-sweep kernel function, using the lid-driven cubic cavity flow as an example.

```
function SWEEP_KERNEL($d$, $k$, fSliceA, fSliceB, momentsOld, ...)
    tid ← threadId.x                                                                        ▷ Thread id.
    Define shared memory variables for moments densL[NX2], vxL[NX2], vyL[NX2], vzL[NX2], ...
    switch $d$ do                                        ▷ Different entry points for the 8 discrete velocity groups.
        case 0                              ▷ For the discrete velocity group 0, i.e., with $\xi_x < 0$, $\xi_y < 0$, $\xi_z < 0$
            if k==NZ then                                                           ▷ The slice near the boundary
                for $j \leftarrow 1$, NY do
                    for $i \leftarrow 1$, NX do
                        Update fSliceA[j][i][tid] as Maxwell boundary
                    end for
                end for
            end if
            for $i \leftarrow 1$, NX do                                                     ▷ The shaft near the boundary
                Update fSliceB[NY+1][i][tid] as Maxwell boundary
            end for
            for $j \leftarrow 1$, NY do
                if tid $<$ NX $+ 2$ then
                    Collaboratively load the moment variables from momentsOld to densL[], vxL[], ... .
                end if
                Update fSliceB[j][NX+1] as Maxwell boundary                                 ▷ The cell near the boundary
                for $i \leftarrow$ NX, 1 do                                                      ▷ Stencil computation
                    Calculate equilibrium distribution function feq using the moments in the shared memory.
                    Calculate geometric, discrete velocity and $\tau$ related coefficients as a, b, c, d, e
                    fSliceB[j][i][tid] = (- a*fSliceB[j][i+1][tid]] - b*fSliceB[j+1][i][tid]
                                         - c*fSliceA[j][i][tid] + d[i]*feq ) * e
                end for
            end for
        case 1                              ▷ For the discrete velocity group 1, i.e., with $\xi_x > 0$, $\xi_y < 0$, $\xi_z < 0$
            Do similar loops as case 0.
        ...                                                               ▷ For the discrete velocity groups 2 → 6.
        case 7                              ▷ For the discrete velocity group 7, i.e., with $\xi_x > 0$, $\xi_y > 0$, $\xi_z > 0$
            Do similar loops as case 0.
end function
```

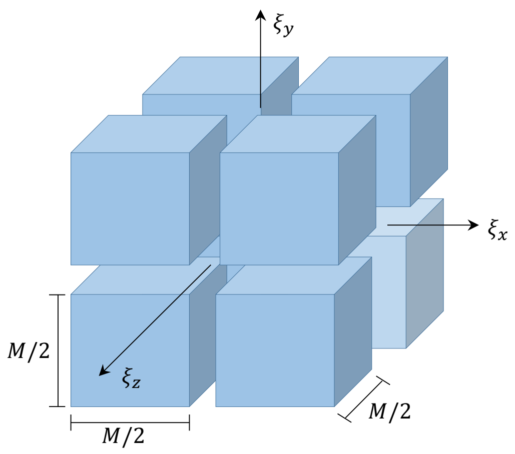


Figure 1: The discrete velocity grid is divided into smaller subsets such that distribution functions of each subset can be held on the GPU global memory. In this work, we divided them into eight octants as shown in this figure (M is the number of velocity points in each dimension). This is a natural choice since in each of the octants all the three components ξ_x, ξ_y and ξ_z of the discrete velocities have the same signs and they share the same spatial-sweep direction. For a larger velocity grid where an octant of the discrete velocity cannot fit into the GPU global memory, the division needs to be finer.

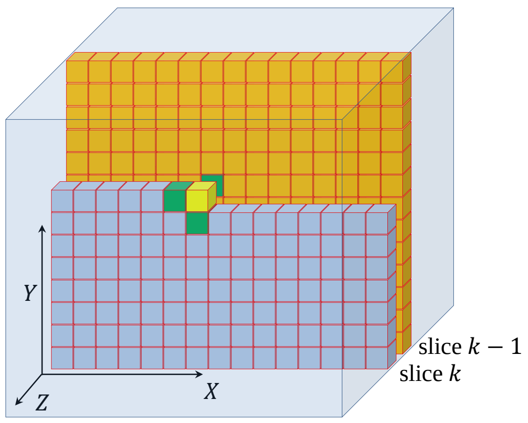


Figure 2: Illustration of the spatial-sweep storing the distribution functions of only two slices. The sweeping direction is $(0,0,0) \to (X^+, Y^+, Z^+)$. The highlighted cell in yellow color is the current cell being updated and the green cells are the upwind stencil cells (colorful online). The two slices are marching along the $0 \to Z^+$ direction.

is outlined in Algorithm 4, where the reduction operation using the shared memory is given in detail.

3.4. Boundary condition

We use the Maxwell diffuse boundary condition for the solid walls. The free-stream boundary condition is used for the external boundary in the supersonic flow case. The boundary condition treatments are embedded in the spatial-sweep and moment-evaluation procedures. In the Maxwell diffuse boundary, the velocity distribution function of the emitting (reflected) particles is:

$$f_{\alpha,w}^{n+1} = \frac{\rho_w^n}{(2\pi R T_w)^{3/2}} \exp\left[-\frac{\xi_\alpha^2}{2RT_w}\right], \tag{16}$$

where ρ_w^n is the density determined by non-penetration condition:

$$\rho_w^n = -\sqrt{\frac{2\pi}{RT_w}} \sum_{\boldsymbol{\xi}_\alpha \cdot \boldsymbol{n}_w < 0} f_\alpha^n \boldsymbol{\xi}_\alpha \cdot \boldsymbol{n}_w. \tag{17}$$

Note that ρ_w^n is calculated using the last step (n) value of the distribution function, because it is impossible to get the distribution functions of other discrete velocities at the newer step in the iterative scheme. This mismatch will induce a total mass change in internal flows. As a remedy, we uniformly scale the density field such that the total mass is unchanged. For the free streaming boundary, the distribution function of the particles entering the computational domain is set to be the equilibrium distribution based on the free stream gas state. When the spatial sweep reaches a solid wall, the density fluxes of the outgoing particles are accumulated during the moment-evaluation procedure.

4. Validation

In this section, we validate our GPU algorithm implementation using the canonical lid-driven cubic cavity (LDCC) flow and the supersonic gas flow past a cube. We first ensure the GPU programs give identical results as the CPU versions implementing the same scheme. Then we compare the results of our GPU programs with a highly accurate 2nd-order solution for the LDCC flow. The DSMC solutions as extra references are also obtained for the two test cases using the open source dsmcFoam solver [72]. We use the DSMC method for validation because this method has been recognized as a reliable benchmark tool for validating other numerical methods solving rarefied gas flow. In addition, using the open source DSMC solver, the results can be reproduced easily. However, we should note that because the DSMC method solves the full Boltzmann equation while our DVM program solves the Shakhov model equation. As a result, systematic minor differences among their solutions are expected, especially for the temperature fields in the transition regime [73, 19, 13, 16].

The gas media in following simulations is argon. Its viscosity changes with the temperature as $\mu = \mu_{\text{ref}}(T/T_{\text{ref}})^\omega$

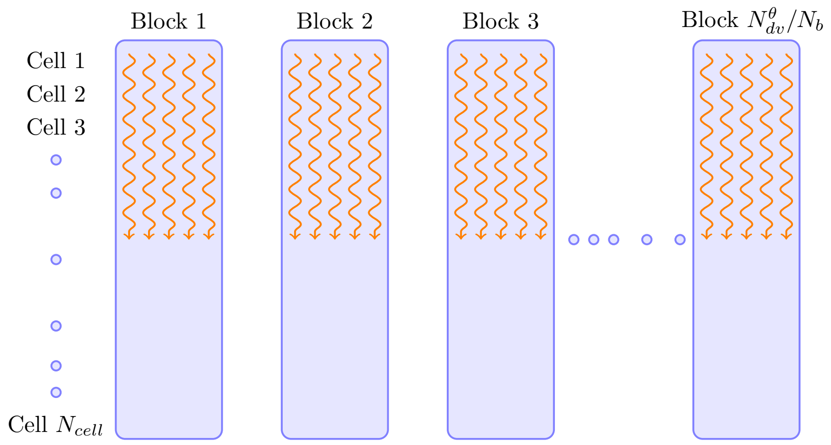


Figure 3: The layout of the spatial-sweep kernel, in which, N_{cell} is the number of the cells in a 2D slice of the XY plane (see Fig. 2), i.e., $N_{cell} = N_x \times N_y$, N_{dv}^{θ} ($\theta = 0, 1, \ldots, 7$) is the number of discrete velocities in the θth batch, and N_b is the size of the thread blocks.

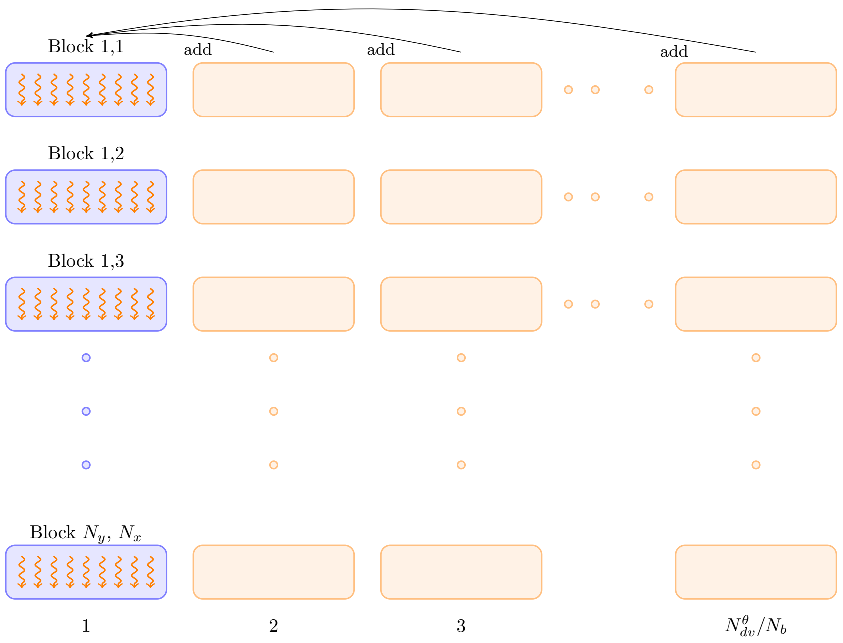


Figure 4: The layout of the moment-evaluation kernel. Each thread block is mapped to one cell in the 2D slice (see Fig. 2). The threads in each block collaboratively add up the distribution function of all discrete velocities of the currently processed batch to the temporary moments. N_{dv}^{θ} ($\theta = 0, 1, \ldots, 7$) is the number of discrete velocities in the θth batch. N_b is the size of the thread blocks.

Algorithm 4 Moment-evaluation kernel function for the cavity flow.

```
function MOMENT_KERNEL($d$, k, fSliceA, fSliceB, momentsNew, ...)
    tid ← threadId.x
    i ← blockIdx.x                                        ▷ Physical cell index in $X$ direction.
    j ← blockIdx.y                                        ▷ Physical cell index in $Y$ direction.
    Allocate shared memory for moments as mom[8][BLK_SIZE] and initialize to zeros
    for $b \leftarrow 1$, dvGrpSize/BLK_SIZE do           ▷ Add BLK_SIZE discrete velocities per time.
        dvId = b * BLK_SIZE + tid
        fTmp = fSliceB[j][i][dvId]
        fSliceA[j][i][dvId] = fTmp
        Accumulate fTmp's contribution to mom[0][tid], ..., mom[7][tid]
    end for
    for $s \leftarrow$ BLK_SIZE/2, 0 do                  ▷ Parallel reduction with sequential addressing.
        if tid < $s$ then
            for m← 0, 7 do
                mom[m][tid] += mom[m][tid+s]
                Synchronize threads
            end for
        end if
    end for
    if is thread 0 then       ▷ Only one thread writes the data from the shared memory to the global memory.
        momentsNew ← mom[][0]
    end if
    if i, j or k near boundary then
        Update solid boundary information     ▷ Extrapolate the wall densities needed for the next iteration step.
    end if
end function
```

with $\omega = 0.81$, which is consistent with the variable hard sphere model in the DSMC method at $T_{\text{ref}} = 273.15$ K. The reference viscosity μ_{ref} is calculated as

$$\mu_{\text{ref}} = \frac{5\sqrt{\pi}}{8}\frac{p_{\text{ref}}\lambda_{\text{ref}}}{\sqrt{2RT_{\text{ref}}}}, \tag{18}$$

where p_{ref} and λ_{ref} are the reference pressure and mean-free path, respectively.

4.1. Lid-driven cubic cavity flow

The lid-driven cubic cavity flow is illustrated in Fig. 5. The inner walls are maintained at a uniform temperature of $T_w = 273.15K$. The lid moves in the X direction with a velocity of $U_w = 0.1\sqrt{2RT_w}$. The Knudsen number is 1.0 based on the initial constant pressure and the side length of the cavity. The DSMC simulation uses the uniform 64^3 grid. An average of 20 particles are initialized in each cell. The DSMC simulation is run without sampling in the initial 20,000 steps and then run with sampling for a further 6 million steps to output the results.

The 1st-order scheme enables GPU computation with much larger grid sizes but sacrifices the accuracy compared with the commonly used 2nd-order schemes. Therefore, it is desirable to evaluate the relative advantages of the GPU parallelized 1st-order scheme. In the following, we will examine accuracy and computational cost of the 1st-order scheme compared with a 2-order scheme, together with the

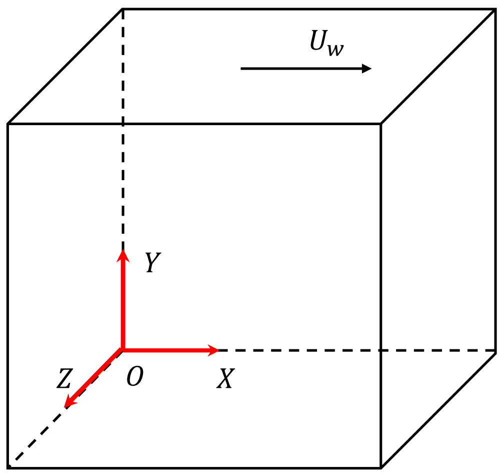


Figure 5: Illustration of the 3D lid-driven cubic cavity flow. The lid moves in the X direction with the velocity U_w. All the walls are maintained at the uniform temperature of T_w.

comparison with the DSMC solution. To this end, we have also implemented the 2nd-order central scheme of Eq. 13 into a CPU solver, in addition to the GPU accelerated with a 1st-order solver (see Appendix A).

The velocity grid used in the 1st-order simulations is the tensor product of the half-range Gauss-Hermite (hrGH) quadrature nodes with a size of 28^3. While in the 2nd-solution, a velocity grid based on the special-coordinate system is used instead (see Appendix A). The uniform physical grids with sizes of 32^3, 64^3, 96^3 and 128^3 are used in both the 1st- and 2nd-order simulations.

4.1.1. Accuracy of the 1st-order solutions

We first analyze the relative accuracy of the 1st-order solutions. The contours of the 1st-order solutions on the XOY symmetric plane are shown in Fig. 6 together with the 2nd-order reference solutions (see Appendix A) and the DSMC solutions. The reference DVM solutions agree well with the DSMC solutions, especially in terms of the velocity field. We can see the converging trend of the 1st-order solutions to the reference solutions as the physical grid refines. Figure 7 shows the temperature contours obtained with the 1st-order scheme on the 128^3 grid, together with the DSMC solution, from which we can observe the overall good agreements between them despite the large statistical noise in the DSMC solution. The quantified scaled average deviations (Appendix A) of the 1st-order solutions from the reference solution are listed in Table 1. The data reveal that as the grid refines, the 1st-order solutions converge to the reference solutions gradually but with much slower rates than the 2nd-order solutions as shown in Table A.6. The absolute deviations of the 1st-order solutions are much larger than those of the 2nd-order solutions using the same grids.

Table 1: The average deviations of the 1st-order solutions from the reference solutions [see the definition of $E(\Psi, N)$ in Eq. (A.2)].

Ψ	$E(\Psi, N)$			
	$N = 32$	$N = 64$	$N = 96$	$N = 128$
ρ	7.82E-04	4.25E-04	2.93E-04	2.27E-04
T	1.31E-04	7.97E-05	6.12E-05	5.23E-05
U_x	1.07E-02	5.47E-03	3.63E-03	2.73E-03
U_y	6.92E-03	3.67E-03	2.51E-03	1.93E-03

4.1.2. Relative cost of 1st- and 2nd-order solutions

Now we compare the relative cost of the 1st- and 2nd-order schemes. More thorough performance analysis of the GPU algorithm will be given in Sec. 5.

Table 2 lists the wall time to obtain the solutions on various grids. The GPU program implementing the 1st-order scheme runs on a Tesla K40 GPU. The 2nd-order scheme is implicated as an MPI parallelized CPU program which runs 96 cores on a high-performance computing (HPC) facility (ARCHER, the UK's national HPC service) with each computing node equipped with two 12-core Intel Xeon E5-2697 v2 (Ivy Bridge) CPUs. All calculations are in double precision. To make the comparison simple, both the 1st- and 2nd-order solutions use the 28^3-point hrGH velocity grid.

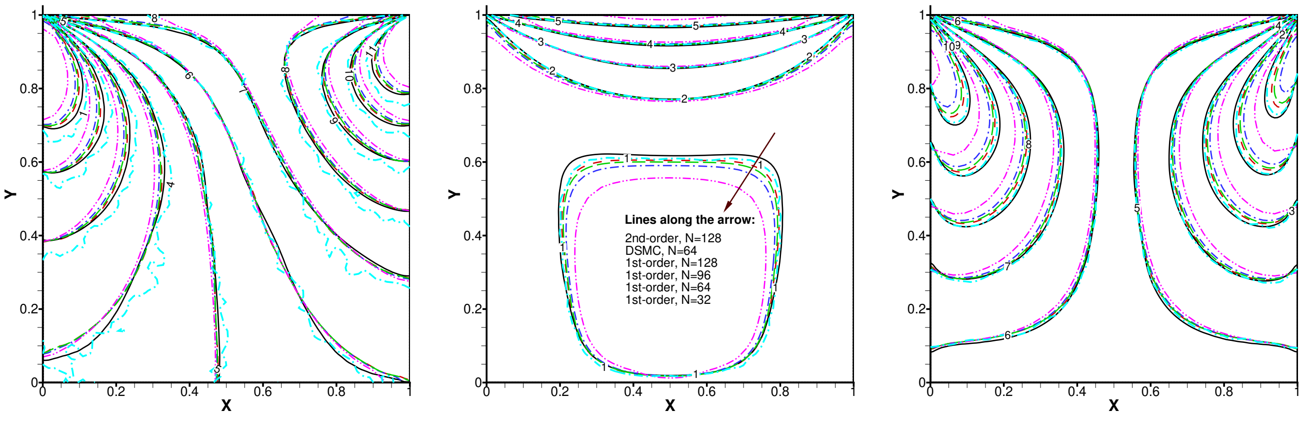
The wall time to obtain the DSMC solution on the 64^3 uniform grid with 128 CPU cores is 141 hours. The table reveals that the 2nd-order scheme requires significantly more iteration steps to converge than the 1st-order scheme. With the same physical grid, the GPU parallelized 1st-order solver gives converged solution in a significantly shorter ($\sim 1/20$) time. Considering the 1st-order scheme needs a finer physical grid to achieve similar accuracy, we compare the wall time to obtain the 1st-order solution on the 128^3 grid and the 2nd-order solutions on the 32^3, 64^3 grids. The comparison shows that the GPU parallelized 1st-order scheme needs shorter wall time than the 2nd-order scheme with the 64^3 grid but longer than that with the 32^3 grid.

4.1.3. Comparison of the GPU performance with literature

Lastly, we compare our GPU algorithm's performance with that of Ref. [55], where the 2D lid-driven cavity flow is solved by a GPU parallelized explicit scheme of the BGK equation. In their study, the grid configuration close to this study is the one with a 64^2 physical grid and a 20^2 velocity grid. For this case, the wall time used by their GPU program is 75.7s. If we extrapolate this time to the 3D case linearly with respect to the number of grid points in the phase space, and considering that there is no need to use an additional distribution function for 3D flows, it would need about 35 hours for our grid configuration ($64^3 \times 28^3$). The much higher computing power of the K40 GPU explains only partially why the current implementation is much faster. The primary reason is due to the adoption of the iterative scheme in this study which is more efficient than the explicit scheme in Ref. [55].

Table 2: Comparison of the computing time of various cases. The 1st- and 2nd-order schemes are implemented as a GPU and an MPI parallelized CPU program, respectively. The wall time is measured by running the corresponding program with a K40 GPU and 96 CPU cores, respectively. The last row indicates the total wall time to obtain the DSMC solution with 128 CPU cores. All calculations are in double precision.

Physical grid	Velocity grid	Spatial order	Iteration steps	Wall time
32^3	hrGH 28^3	2nd	369	423s
64^3	hrGH 28^3	2nd	524	4890s
32^3	hrGH 28^3	1st	42	28s
64^3	hrGH 28^3	1st	42	214s
128^3	hrGH 28^3	1st	42	1688s
64^3	—	DSMC	—	141 hours



(a) (b) (c)

Figure 6: Comparison of the temperature and velocity contours on the XOY symmetric plane of the 3D cavity using the 1st-order DVM scheme on various grids, the 2nd-order scheme on the $N = 128$ grid, and the DSMC method on the $N = 64$ grid. (a) The contours of the non-dimensional temperature T/T_w; the contour levels are from 0.996 to 1.006 with a step of 0.001. (b) The contours of the U_x velocity, non-dimensioqnalized by $\sqrt{2RT_w}$; the contour levels are from -0.005 to 0.035 with a step of 0.01. (c) The contours of the U_y velocity, non-dimensioqnalized by $\sqrt{2RT_w}$; the contour levels are from -0.018 to 0.018 with a step of 0.004. The contour levels are labeled in each subfigure. The line legends for all the three subfigures are drawn in the subfigure (b).

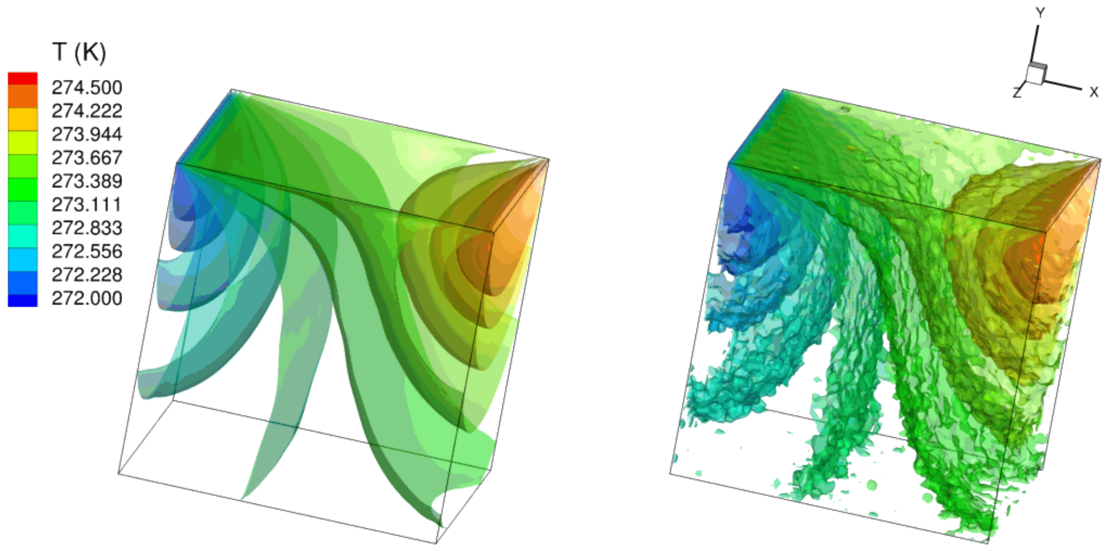


Figure 7: Comparison of the temperature iso-surfaces predicted by the GPU accelerated 1st-order DVM and the DSMC method. Physical grid size in the DVM is 128^3 while in DSMC is 64^3. The velocity grid in the DVM solver is 28^3 half-range Gauss-Hermite quadrature points.

4.2. Rarefied gas flow past a cube

The second case is a supersonic rarefied gas flow past a cube with Ma $= 2$ and Kn $= 1$. The cube size is 1^3 while the computational domain size is $14 \times 12 \times 12$. The center of the cube is located at $(0, 0, 0)$. The computation domain is illustrated in Fig. 8. The surface of the cube is maintained at 273K. In the DSMC simulation, we simulate only a quadrant of the domain due to the two-fold symmetry. Even though using symmetry boundaries in the DVM is possible, implementing them in the current memory-reduced DVM is not straightforward [70]; thus, we simulate the full domain here to avoid further complicating the algorithm presented above. The full physical grid size is $181 \times 181 \times 191$. The cell size expands with a cell-by-cell ratio of 1.02 in the front and lateral sides of the cube and 1.03 at the rear of the cube. The velocity grid size is uniform with 48^3 points in the range of $[-4\sqrt{2RT_w}, 4\sqrt{2RT_w}]^3$ and a trapezoidal rule is used to calculate the moments. The DVM simulation takes approximately 20 hours with 41 iteration steps on the Tesla K40 GPU. In the DSMC simulation, each cell has 50 particles on average and the time step size is 2.0×10^{-7}s. The sampling begins from 1,000 steps and continues for 68,000 steps which take 128.5 hours on 128 CPU cores.

The temperature iso-surfaces around the cube are presented in Fig. 9, from which an overall agreement between the DVM results and the DSMC solutions can be found. Due to the high Knudsen number, the bowl-shaped shockwave in front of the cube is very thick. The maximum temperature in the shock wave region is around 550K, while the lowest temperature appears at the rear of the cube due to the blockage effect on the gas molecules by the cube. Figure 10 shows the detailed comparisons of the temperature, density and velocities distributions on the symmetry plane of the computation domain. The overall good agreement, especially near the wall surface region, indicates that the DVM prediction is satisfactory. From these contour plots, we can see the temperature field in the shockwave region increases as early as $x = -3$ and the high-temperature region is restricted to a relatively small region in front of the cube. The temperature contours lines far away from the cube show a relatively larger difference between the DVM and DSMC solutions. The main reason is the larger statistical noise in the temperature field of the DSMC solution. The earlier rise of the temperature in the DVM solution along the stagnation line in the front of the shockwave is also a well-known problem of the Shakhov model equation and had been identified in the studies on shockwave structures [16, 18].

5. Performance profiling

In this section, we analyze the parallel computing efficiency and scalability of our GPU algorithm. We also compare the speedups with an MPI-CPU parallel implementation of the same iterative scheme. All the testings are based on the 3D lid-driven cavity flow with single-precision computations. The GPU cards used in the platform include a Tesla K40 GPU, a Tesla K80 GPU and a Quadro M2000 GPU. The major specifications of the three GPU models are listed in Table 3. The M2000 is based on a newer architecture called Maxwell while the K40 and K80 are based on the older Kepler architecture. The M2000 has fewer streaming cores, but the multiprocessors can operate at a much higher frequency than the K40/K80. The commercial K80 card actually contains two GK201 GPUs, while in this study we only use a single GPU on it and all the data listed in Table 3 are the values of a single GK201 GPU. The GPU program is developed with CUDA C++ and is compiled using the Nvidia CUDA Toolkit (version 9.2) without the aggressive optimization flags such as `-prec-div=false`, `-prec-sqrt=false` or `-use_fast_math`. The host compiler is Intel's C++ compiler (version 15.0).

The CPU program implementing the same 1st-order scheme is parallelized with MPI and written in C++. Note that this CPU program is different from the 2nd-order one used in Appendix A. The MPI-CPU program is compiled with the same compiler and Intel MPI library with compiling flag `-xHost` which enable advanced arithmetical instructions such as AVX2 and FMA. The program runs on an in-house cluster equipped with a 56Gbps InfiniBand network. Each computing node comprises two Xeon E5-2680v3 (Haswell) @2.5GHz CPU. The MPI-CPU program uses the parallelization method as described at the end of Sec. 2.

5.1. Speedups and comparison with the MPI parallelization

Firstly, we investigate the overall speedups of the GPU algorithm on different GPU devices. The average wall time for a single iteration step of the cavity flow case with the various grid size of physical and molecular velocity space is measured on different platforms. The results are presented in Table 4, and the corresponding speedups against the single-core program are shown in Fig. 11. The measured GPU global memory consumptions are also listed in Table 4 to demonstrate the advantage of our memory reduction techniques. For the two largest grids, i.e., $64^3 \times 128^3$ and $64^3 \times 128^3$, the global memory occupations are over 4 GB and therefore unable to run on the M2000. It should be remarked that if we do not use the memory reduction techniques above, the most conservative estimation (storing only one float variable on each grid point in the phase space) of the memory occupation can be as high as $128^3 \times 64^3 \times 4/10^{12} = 2.2$ TB.

Several interesting patterns can be observed from the table and chart. First, when the velocity grid is 64^3 or larger, both the K40 and K80 achieve speedups around 190. It is much higher than the cases with the 32^3 velocity grid which is only around 100. This contrast can be easily explained. As there are over two thousand cores on K80/K40, when simulating the cases with the 32^3 velocity grid, the thread grid size of the `sweepSlice` kernel is only $16^3 = 4096$, meaning that there is simply not

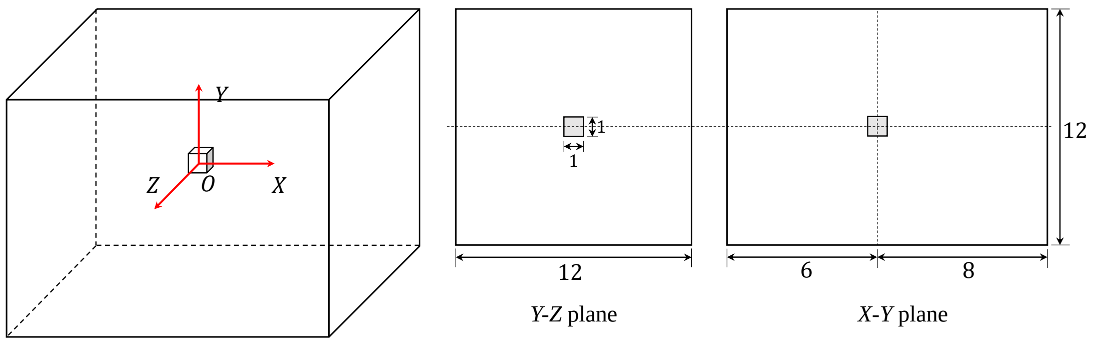


Figure 8: Schematic diagram of the rarefied gas flow past a cube.

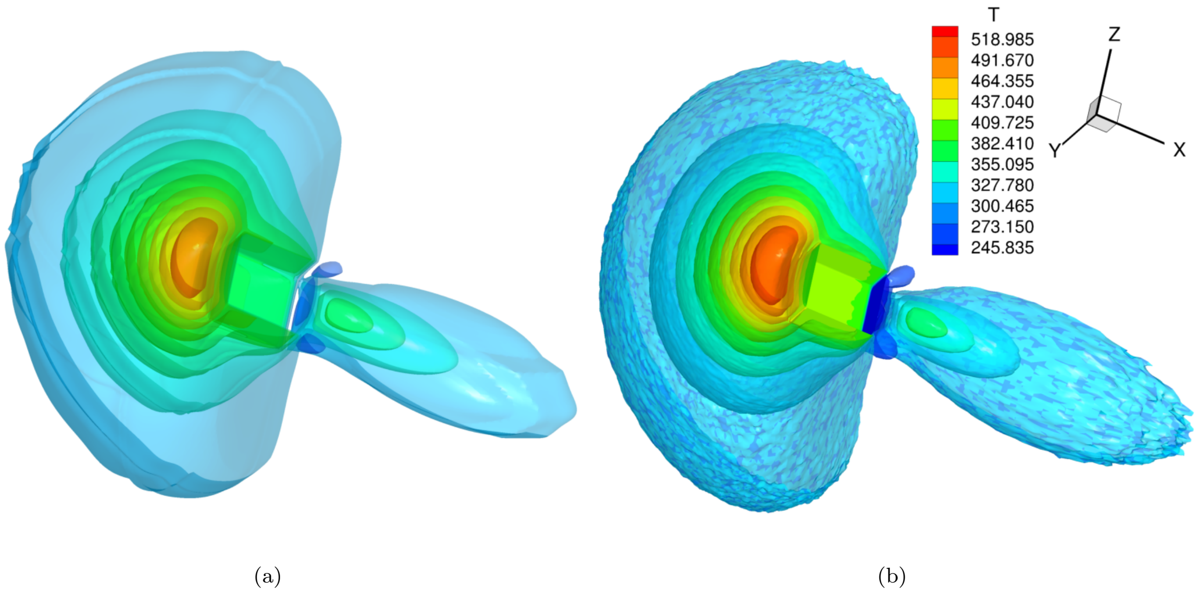


Figure 9: Temperature iso-surfaces predicted by (a) the GPU accelerated DVM simulation and (b) the DSMC simulation.

enough parallelism for the GPUs to fulfill their computing potential. Second, comparing the speedups among the three GPUs, we can find that the K40 is about 20% faster than the K80, this is reasonable as the number of cores and memory bandwidth of the K40 are relatively higher than those of the K80, while their other configurations are almost the same. The M2000 performs better than the other two with the 32^3 velocity grid even though it has much fewer streaming processors and smaller global memory bandwidth. This means that for these cases, the computing power and high memory bandwidth on K40/K80 is not fully used. Lastly, we consider the performance of the MPI-CPU program. When running with 96 CPU cores, the MPI parallelization can achieve a speedup around 70, which means the strong scaling parallel efficiency is around 73%, and is typical for the collective communications of the `MPI_Allreduce` when reducing a large chunk of data. We note that with more CPU cores, the efficiency of the MPI-CPU program with the velocity space decomposition strategy will deteriorate quickly [74].

5.2. Kernel performance analysis

We now analyze the two major kernels's performance, i.e. `sweepSlice` and `momentSlice` (Algorithms 3 and 4) using the lid-driven cavity flow as an example. The Nvidia visual profiler (nvvp) [75] is employed to measure various runtime metrics of the kernels. The general metrics are shown in Table 5 for grid sizes of $64^3\times64^3$, $128^3\times64^3$ and $64^3\times128^3$. These metrics indicate how efficient the various GPU resources are being used by the kernel functions, and reveal the performance limiters, e.g. memory bandwidth band, compute band or instruction/memory latencies.

Table 5 shows for the 64^3 velocity grid, the `sweepSlice` kernel takes slight longer time than the `momentSlice` kernel. While the situation reverses for the 128^3 velocity grid. This means the `momentSlice` kernel does not scale equally well as the `sweepSlice` kernel.

The SM occupancy of the `sweepSlice` kernel is around 61% on the M2000/K40 and 96% on the K80. The relative lower occupancy on M2000 and K40 is due to the fewer registers available on their SM (64K as opposed to

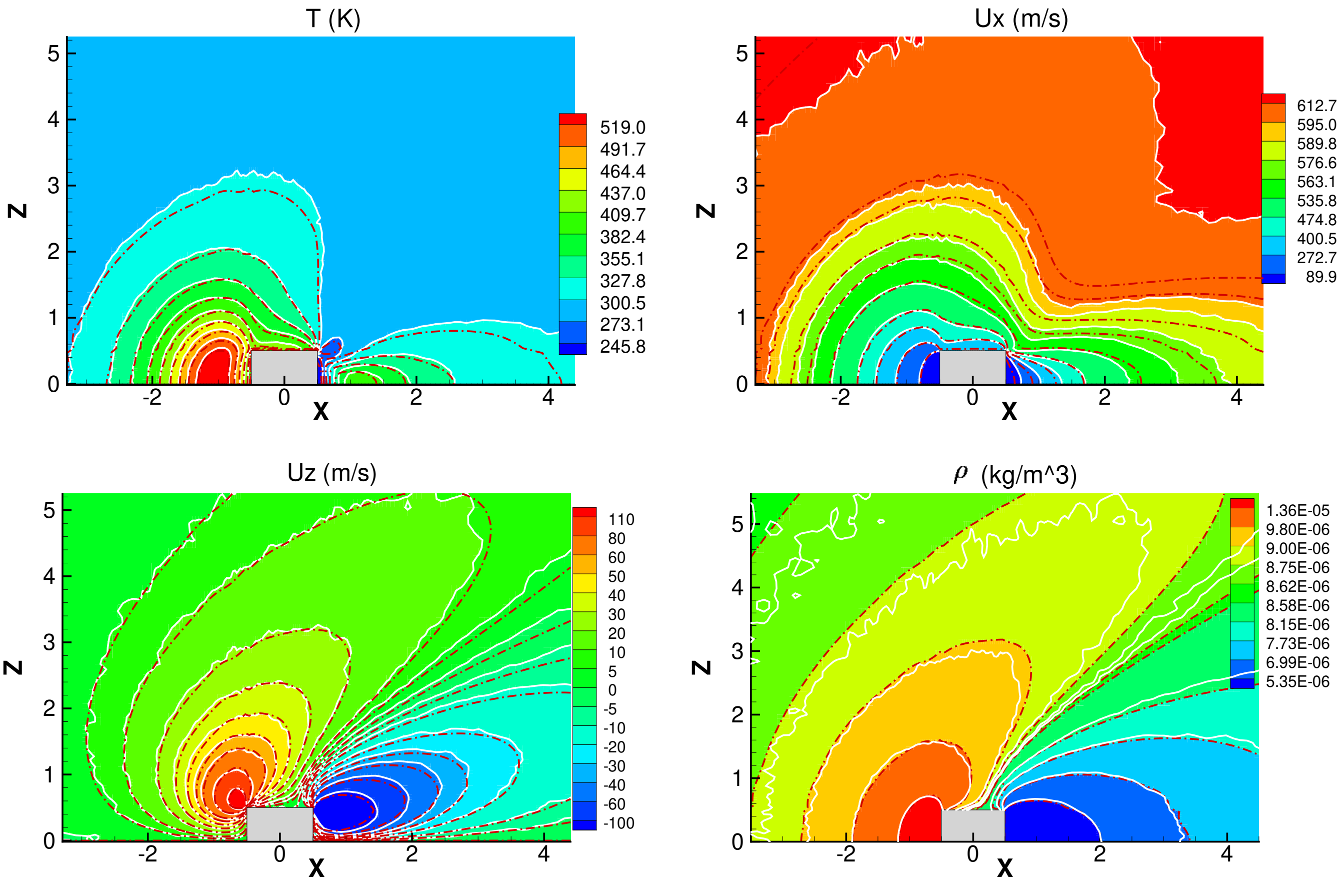


Figure 10: Comparison of various fields on the X-Z symmetric plane of the flow domain: (a) temperature, (b) X-component velocity U_x, (c) Z-component velocity U_z and (d) density ρ. In each of the plots, the dashed red lines denote the DVM result, and the solid white lines with colored background represent the DSMC solution.

128K on K80, see Table 3). The profiling shows that the GPU performance on M2000 and K40 is limited by the instruction and memory fetching latencies, but when we restrict the kernel to use fewer registers via compiler flag, the achieved occupancy can be increased to 69% but the overall computing time is longer due to the increased instruction and memory dependencies. The `sweepSlice` kernel performance on K80 is bounded by the computing, i.e., the instruction executing speed. A further investigation using the profiler reveals that both the utilizations of the load/store and the arithmetic function units on the K80 reach around 75%. In the `momentSlice` kernel, each thread requires only 32 registers but each thread block needs 8208 bytes of shared memory. On the K40, the SM occupancy is limited by the shared memory size which only has 48 KB per SM. While on the M2000 and K80, which have more shared memory (see Table 3), the SMs are almost fully occupied. On K40 and K80, the performance is limited by shared memory bandwidths, which are measured at 1.5 TB/s and 1.3 TB/s respectively, close to the hardware limits.

We also investigate the effect of workload on the kernel performance with larger grid sizes: $64^3 \times 128^3$ and $128^3 \times 64^3$. It should be noted that a velocity grid size as large as 128^3 is too expensive for most practical applications. In practice, the large velocity grid size can be avoided using more sophisticated velocity grids such as non-uniform or adaptive velocity grids, and conservative moment-evaluation procedures. Here we use the large velocity grid as an extreme case to explore the upper limit of the kernels' performance under the condition of sufficiently abundant parallelism. For example, with the 128^3 velocity grid size, each SM on the K40 can be populated with at least 68 thread blocks, which are enough to saturate the SMs and minimize the trail effect (some SMs get one thread block less than others due to the round-robin scheduling policy and the number of SM not dividing the total number of thread blocks).

Table 5 shows the overall GPU computing and memory utilization levels of the two kernel functions on the three GPUs are from 56% to 79%, by considering the main performance limiter for each kernel function using each GPU.

Table 3: Specifications of the three different GPUs used in the performance evaluation. For all the GPUs, the GPU auto-boost feature is turned off and the fixed SM frequencies are sustainable in all the workloads. The measured global memory accessing bandwidth is obtained from the `bandwidthTest` program in the NVIDIA_CUDA_SAMPLES. * The parameters for K80 GPU card are for a one of its two GK201 GPUs inside.

GPU Model	Number of SP	Device memory	SM frequency	Bandwidth theoretical	Peak single precision	Max shm per SM	Max reg per SM
Quadro M2000	768	4 GB	1088 MHz	106 GB/s	1.786 TFLOPS	96 KB	64 K
Tesla K40	2880	12 GB	745 MHz	288 GB/s	4.29 TFLOPS	48 KB	64 K
Tesla K80*	2496	12 GB	745 MHz	240 GB/s	4.11 TFLOPS	112 KB	128 K

Table 4: Average computing time (s) for a single iteration step and GPU global memory occupations on different platforms for the lid-driven cavity flow case. All profile runnings use single precision. The last two cases on the M2000 GPU are not available due to its insufficient global memory capacity.

Physical	Velocity	CPU MPI Program		GPU Program			GPU
grid N	grid M	1 core	96 cores	M2000	K40	K80	memory
32^3	32^3	53.50 s	0.68 s	0.48 s	0.55 s	0.57 s	75 MB
32^3	64^3	424.56 s	4.92 s	3.29 s	2.15 s	2.52 s	347 MB
64^3	32^3	429.32 s	6.20 s	3.77 s	4.33 s	4.42 s	206 MB
64^3	64^3	3097.39 s	43.29 s	26.33 s	16.57 s	19.80 s	1205 MB
64^3	128^3	24366.40 s	358.41 s	—	127.56 s	132.84 s	9242 MB
128^3	64^3	25020.61 s	371.02 s	—	131.04 s	156.47 s	4703 MB

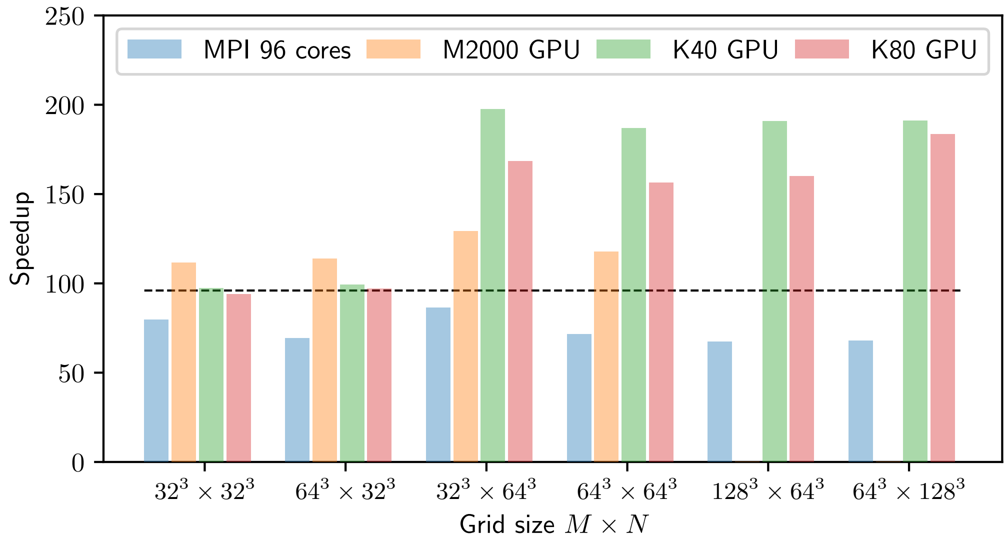


Figure 11: Speedup for the lid-driven cavity flow case with various grid sizes based on the data in Table 4. The dashed black line indicates the theoretical speedup (96) of the CPU program using 96 cores versus using one core. The data of the M2000 GPU on the two largest grid sizes are not available due to insufficient global memory.

Table 5: Utilization metrics of GPU resource for the `sweepSlice` and `momentSlice` kernels. The data are retrieved from Nvidia Visual Profiler (nvvp). The test case is the lid-driven cubic cavity flow with the grid size of $64^3 \times 64^3$. The *latency* in the table stands for instruction and memory latencies. The relative runtime is the ratio the kernel execution time to the overall computing time for a single iteration step.

Kernel	GPU model	Achieved occupancy	Compute utilization	Memory utilization	Device memory throughput	Performance limiter	Relative runtime
	Physical grid size 64^3, velocity grid size 64^3.						
`sweepSlice`	M2000	60.1%	64%	55%	53.4 GB/s	latency	58.0%
	K40	61.6%	65%	38%	112.5 GB/s	latency	51.6%
	K80	95.5%	78%	46%	90.4 GB/s	compute	53.9%
`momentSlice`	M2000	99.7%	42%	56%	52.5 GB/s	latency	42.0%
	K40	61.3%	30%	78%	84.2 GB/s	memory	48.4%
	K80	98.4%	31%	76%	73.7 GB/s	memory	46.1%
	Physical grid size: 128^3, velocity grid size: 64^3.						
`sweepSlice`	K40	61.6%	59%	47%	128.8 GB/s	latency	51.4%
	K80	95.6%	79%	46%	91.6 GB/s	compute	53.6%
`momentSlice`	K40	61.4%	31%	75%	84.0 GB/s	memory	48.6%
	K80	98.4%	30%	76%	73.8 GB/s	memory	46.4%
	Physical grid size: 64^3, velocity grid size: 128^3.						
`sweepSlice`	K40	62.4%	58%	48%	143.1 GB/s	latency	48.0%
	K80	98.7%	78%	58%	139.4 GB/s	compute	43.1%
`momentSlice`	K40	62.1%	29%	78%	163.3 GB/s	memory	52.0%
	K80	99.4%	30%	77%	157.8 GB/s	memory	56.9%

6. Conclusion and discussions

In summary, an efficient memory-reduced DVM for kinetic model equations has been developed to enhance GPU acceleration. Different from the previously reported GPU accelerations of kinetic equation that use explicit schemes, the current implementation is based on a fast converging iterative scheme. The memory reduction techniques in both the molecular velocity and the physical spaces reduce memory requirements significantly from terabytes to gigabytes, enabling a full 3D simulation on a single GPU. The test cases with up to 0.7 trillion phase-space grid points demonstrated the capability of the proposed GPU algorithm in simulating large-scale 3D flows on a single GPU. The performance profiling on different GPUs shows that our kernel function implementations have good utilization levels of the GPU resources. Depending on the GPU devices and phase-space grid sizes, the speedups for the lid-driven cavity flow range from 1.2 to 2.8 over the 96-core running MPI CPU program implementing the same scheme. Owing to the adoption of the iterative scheme, our implementation for steady state flows is more efficient than those using explicit schemes.

The performance of the current implementation benefits from a large velocity grid as 64^3 or more, where there is enough parallelism to realize the full potential of high-end Tesla-series GPUs with thousands of cores. However, the current trend in the latest released GPUs is that the number of cores is increasing even more, e.g., 5120 on the V100 GPU released in 2017. In such a case, to fully utilize improved computing power in the future, more parallelism in the algorithm should be applied other than using only the molecular-velocity-space parallelization. Another possible improvement in the future is to extend the 1st-order upwind scheme to 2nd-order by storing three consecutive slices of cell's distribution functions thus the 2nd-order upwind stencil points are available for each grid point in the physical space during the spatial-sweep procedure. Using a high-order scheme can reduce the physical grid points significantly as has been demonstrated during the grid-independence study for the lid-driven cavity flow.

Acknowledgments

This project leading to this paper has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement number 793007. Financial support from the UK Engineering and Physical Sciences Research Council (EPSRC) under Grant No. EP/M021475/1 is gratefully acknowledged. S. Chen acknowledges the financial support from National Science Foundation of China (Grants No. 91530319). Z. Guo acknowledges the financial support from National Science Foundation of China (Grants No. 11702223). Computing time during the program development & testing on the ARCHER is provided by the UK

Consortium on Mesoscale Engineering Sciences (EPSRC Grant Nos. EP/L00030X/1 and EP/R029598/1). L. Zhu thanks Dr. Minh-Tuan Ho from University of Strathclyde for discussions of the relation between the numerical quadrature and the ray effect.

Appendix A. The second order reference solution

A 2nd-order scheme has been implemented on the CPU platform, to provide highly accurate reference solution when evaluating accuracy of the 1st-order scheme in Sec. 4. The 2nd-order scheme is based on the "delta" form of Eq. (13),

$$\boldsymbol{\xi}_\alpha \cdot \boldsymbol{\nabla} \Delta f_\alpha^{n+1} + \frac{1}{\tau^n} \Delta f_\alpha^{n+1} = -\boldsymbol{\xi}_\alpha \cdot \boldsymbol{\nabla} f^n - \frac{1}{\tau^n}[f_\alpha^n - f_\alpha^{S,n}], \tag{A.1}$$

where $\Delta f_\alpha^{n+1} \equiv f_\alpha^{n+1} - f_\alpha^n$. The gradient terms on the left- and right-hand sides (LHS and RHS) are discretized using the 1st-order upwind and 2nd-order central schemes, respectively. The 2nd-order central-scheme-based solution is attained as the iteration converges and the LHS approaches to zero [64]. Because the f_α^n term at the RHS of Eq. (A.1) is needed when solving Δf_α^{n+1}, the distribution functions of all discrete velocities are stored. This requirement leads to huge memory consumptions; thus, we only implemented it on the CPU platform.

The higher-order scheme and finer physical grids in the reference solutions magnify the ray effect which is caused by the discontinuous velocity boundary condition at the leading and trailing edges of the moving lid [76–79]. The ray effect appears in high-Kn flows as discontinuities in the distribution function and non-smoothness in the macroscopic fields. Unlike in the 1st-order scheme, here we use a spherical-coordinate-based discrete velocity grid [79], which is found to produce much weaker ray effect than the tensor product based quadratures in the Cartesian-coordinate. The spherical-coordinate-based velocity grid we used is the tensor product of

- the 4 positive nodes of the 8th-order hrGH quadrature as radial nodes;
- 160 uniformly distributed polar angles in $[0, 2\pi]$ on the XOY plane;
- the inverse cosine values of the 12th-order Gauss-Legendre quadrature nodes as azimuthal angles.

The grid independence of the 2nd-order solutions is verified by calculating the following scaled average deviations of the coarser-grid solutions from the reference solutions on the 128^3 grid,

$$E(\Psi, N) = \frac{1}{N^3 \Psi_0} \sum_{i,j,k=1}^{N} \left| \Psi_{i,j,k}(128^3 \to N^3) - \Psi_{i,j,k}(N) \right|, \tag{A.2}$$

in which $\Psi = \rho, T, U_x, U_y$; the corresponding scales are $\Psi_0 = p_{\text{ref}}/(RT_w), T_w, U_w, U_w$; N^3 is the grid size, i.e. 32^3, 64^3 or 96^3; $\Psi_{i,j,k}(N)$ is the solution of cell (i, j, k) on the N^3 grid, and $\Psi_{i,j,k}(128^3 \to N^3)$ is the linearly interpolated solution on the grid N^3 from the reference solution. The deviations calculated from the coarser-grid solutions are listed in Table A.6, where we can see the quick convergences of the solutions as the grid is refined. The deviations of the 96^3-grid solutions from the references (on the 128^3) ones are less than 0.23%. Therefore, the solutions on the 128^3 grid can be confidently used as the reference solutions for the 1st-order solution in Sec. 4.1.

Table A.6: The scaled average deviations of the 2nd-order solutions from the reference solutions on the 128^3 grid [see the definition of the scaled average deviation in Eq. (A.2)].

Ψ	$E(\Psi, N)$		
	$N = 32$	$N = 64$	$N = 96$
ρ	1.17E-04	3.98E-05	1.32E-05
T	3.72E-05	1.29E-05	4.46E-06
U_x	2.17E-03	7.06E-04	2.28E-04
U_y	1.38E-03	4.66E-04	1.53E-04

Reference

References

[1] G. A. Bird, Molecular Gas Dynamics and the Direct Simulation of Gas Flows, Clarendon Press, 1994.

[2] G. A. Bird, The DSMC Method, CreateSpace Independent Publishing Platform, 2013.

[3] V. V. Aristov, Direct Methods for Solving the Boltzmann Equation and Study of Nonequilibrium Flows, Springer Science & Business Media, 2001.

[4] L. Mieussens, A survey of deterministic solvers for rarefied flows (Invited), in: AIP Conference Proceedings, Vol. 1628, 2014, pp. 943–951. doi:10.1063/1.4902695.

[5] J. E. Broadwell, Study of rarefied shear flow by the discrete velocity method, Journal of Fluid Mechanics 19 (1964) 401–414. doi:10.1017/S0022112064000817.

[6] J. Y. Yang, J. C. Huang, Rarefied flow computations using nonlinear model Boltzmann equations, Journal of Computational Physics 120 (1995) 323–339. doi:10.1006/jcph.1995.1168.

[7] Y. Sone, Molecular Gas Dynamics: Theory, Techniques, and Applications, Springer Science & Business Media., 2007.

[8] L. Wu, J. M. Reese, Y. Zhang, Solving the Boltzmann equation deterministically by the fast spectral method: Application to gas microflows, Journal of Fluid Mechanics 746 (2014) 53–84. doi:10.1017/jfm.2014.79.

[9] J. Meng, Y. Zhang, N. G. Hadjiconstantinou, G. A. Radtke, X. Shan, Lattice ellipsoidal statistical BGK model for thermal non-equilibrium flows, Journal of Fluid Mechanics 718 (2013) 347–370. doi:10.1017/jfm.2012.616.

[10] J.-C. Huang, K. Xu, P. Yu, A Unified Gas-Kinetic Scheme for Continuum and Rarefied Flows III: Microflow Simulations, Communications in Computational Physics 14 (2013) 1147–1173. doi:10.4208/cicp.190912.080213a.

[11] L. Wu, M. T. Ho, L. Germanou, X.-J. Gu, C. Liu, K. Xu, Y. Zhang, On the apparent permeability of porous media in rarefied gas flows, Journal of Fluid Mechanics 822 (2017) 398–417. doi:10.1017/jfm.2017.300.

[12] S. Jin, Asymptotic preserving (AP) schemes for multiscale kinetic and hyperbolic equations: A review, Lecture Notes for Summer School on "Methods and Models of Kinetic Theory"(M&MKT), Porto Ercole (Grosseto, Italy) (2010) 177–216.

[13] K. Xu, J.-C. Huang, A unified gas-kinetic scheme for continuum and rarefied flows, Journal of Computational Physics 229 (2010) 7747–7764. doi:10.1016/j.jcp.2010.06.032.

[14] L. Mieussens, On the asymptotic preserving property of the unified gas kinetic scheme for the diffusion limit of linear kinetic models, Journal of Computational Physics 253 (2013) 138–156. doi:10.1016/j.jcp.2013.07.002.

[15] Z. Guo, K. Xu, R. Wang, Discrete unified gas kinetic scheme for all Knudsen number flows: Low-speed isothermal case, Physical Review E 88 (2013) 033305. doi:10.1103/PhysRevE.88.033305.

[16] Z. Guo, R. Wang, K. Xu, Discrete unified gas kinetic scheme for all Knudsen number flows. II. Thermal compressible case, Physical Review E 91 (2015) 033313. doi:10.1103/PhysRevE.91.033313.

[17] S. Liu, P. Yu, K. Xu, C. Zhong, Unified gas-kinetic scheme for diatomic molecular simulations in all flow regimes, Journal of Computational Physics 259 (2014) 96–113. doi:10.1016/j.jcp.2013.11.030.

[18] K. Xu, Direct Modeling for Computational Fluid Dynamics: Construction and Application of Unified Gas-Kinetic Schemes, Advances in Computational Fluid Dynamics, World Scientific Publishing, 2015.

[19] L. Zhu, Z. Guo, K. Xu, Discrete unified gas kinetic scheme on unstructured meshes, Computers & Fluids 127 (2016) 211–225. doi:10.1016/j.compfluid.2016.01.006.

[20] L. Zhu, Z. Guo, Application of discrete unified gas kinetic scheme to thermally induced nonequilibrium flows, Computers & Fluidsdoi:10.1016/j.compfluid.2017.09.019.

[21] L. Mieussens, Discrete velocity model and implicit scheme for the bgk equation of rarefied gas dynamics, Mathematical Models and Methods in Applied Sciences 10 (2000) 1121–1149. doi:10.1142/S0218202500000562.

[22] V. Titarev, M. Dumbser, S. Utyuzhnikov, Construction and comparison of parallel implicit kinetic solvers in three spatial dimensions, Journal of Computational Physics 256 (2014) 17–33. doi:10.1016/j.jcp.2013.08.051.

[23] Y. Zhu, C. Zhong, K. Xu, Implicit unified gas-kinetic scheme for steady state solutions in all flow regimes, Journal of Computational Physics 315 (2016) 16–38. doi:10.1016/j.jcp.2016.03.038.

[24] Y. Zhu, C. Zhong, K. Xu, Unified gas-kinetic scheme with multigrid convergence for rarefied flow study, Physics of Fluids 29 (2017) 096102. doi:10.1063/1.4994020.

[25] L. M. Yang, C. Shu, W. M. Yang, J. Wu, An implicit scheme with memory reduction technique for steady state solutions of DVBE in all flow regimes, Physics of Fluids 30 (2018) 040901. doi:10.1063/1.5008479.

[26] P. Wang, M. T. Ho, L. Wu, Z. Guo, Y. Zhang, A comparative study of discrete velocity methods for low-speed rarefied gas flows, Computers & Fluids 161 (2018) 33–46. doi:10.1016/j.compfluid.2017.11.006.

[27] F. Filbet, G. Russo, High order numerical methods for the space non-homogeneous Boltzmann equation, Journal of Computational Physics 186 (2003) 457–480. doi:10.1016/S0021-9991(03)00065-2.

[28] A. Alexeenko, C. Galitzine, A. Alekseenko, High-Order Discontinuous Galerkin Method for Boltzmann Model Equations, in: 40th Thermophysics Conference, American Institute of Aeronautics and Astronautics, Seattle, Washington, 2008, p. 4256. doi:10.2514/6.2008-4256.

[29] C. Wu, B. Shi, C. Shu, Z. Chen, Third-order discrete unified gas kinetic scheme for continuum and rarefied flows: Low-speed isothermal case, Physical Review E 97 (2018) 023306. doi:10.1103/PhysRevE.97.023306.

[30] W. Su, P. Wang, Y. Zhang, L. Wu, A high-order hybridizable discontinuous Galerkin method with fast convergence to steady-state solutions of the gas kinetic equation, Journal of Computational Physics 376 (2019) 973–991. doi:10.1016/j.jcp.2018.08.050.

[31] Z.-H. Li, H.-X. Zhang, Gas-kinetic numerical studies of three-dimensional complex flows on spacecraft re-entry, Journal of Computational Physics 228 (2009) 1116–1138. doi:10.1016/j.jcp.2008.10.013.

[32] Z.-H. Li, A.-P. Peng, H.-X. Zhang, J.-Y. Yang, Rarefied gas flow simulations using high-order gas-kinetic unified algorithms for Boltzmann model equations, Progress in Aerospace Sciences 74 (2015) 81–113. doi:10.1016/j.paerosci.2014.12.002.

[33] G. Dimarco, R. Loubère, J. Narski, Towards an ultra efficient kinetic scheme. Part III: High-performance-computing, Journal of Computational Physics 284 (2015) 22–39. doi:10.1016/j.jcp.2014.12.023.

[34] S. Li, Q. Li, S. Fu, J. Xu, The high performance parallel algorithm for Unified Gas-Kinetic Scheme, in: AIP Conference Proceedings, Victoria, BC, Canada, 2016, p. 180007. doi:10.1063/1.4967676.

[35] V. A. Titarev, Application of model kinetic equations to hypersonic rarefied gas flows, Computers & Fluids 169 (2018) 62–70. doi:10.1016/j.compfluid.2017.06.019.

[36] G. Dimarco, R. Loubère, J. Narski, T. Rey, An efficient numerical method for solving the Boltzmann equation in multidimensions, Journal of Computational Physics 353 (2018) 46–81. doi:10.1016/j.jcp.2017.10.010.

[37] M. T. Ho, L. Zhu, L. Wu, P. Wang, Z. Guo, Z.-H. Li, Y. Zhang, A multi-level parallel solver for rarefied gas flows in porous media, Computer Physics Communications 234 (2019) 14–25. doi:10.1016/j.cpc.2018.08.009.

[38] V. I. Kolobov, R. R. Arslanbekov, V. V. Aristov, A. A. Frolova, S. A. Zabelok, Unified solver for rarefied and continuum flows with adaptive mesh and algorithm refinement, Journal of Computational Physics 223 (2007) 589–608. doi:10.1016/j.jcp.2006.09.021.

[39] S. Chen, K. Xu, C. Lee, Q. Cai, A unified gas kinetic scheme with moving mesh and velocity space adaptation, Journal of Computational Physics 231 (2012) 6643–6664. doi:10.1016/j.jcp.2012.05.019.

[40] C. Baranger, J. Claudel, N. Hérouard, L. Mieussens, Locally refined discrete velocity grids for stationary rarefied flow simulations, Journal of Computational Physics 257, Part A (2014) 572–593. doi:10.1016/j.jcp.2013.10.014.

[41] S. Brull, L. Mieussens, Local discrete velocity grids for deterministic rarefied flow simulations, Journal of Computational Physics 266 (2014) 22–46. doi:10.1016/j.jcp.2014.01.050.

[42] S. Zabelok, R. Arslanbekov, V. Kolobov, Adaptive kinetic-fluid solvers for heterogeneous computing architectures, Journal of Computational Physics 303 (2015) 455–469. doi:10.1016/j.jcp.2015.10.003.

[43] P. Asinari, T. Ohwada, E. Chiavazzo, A. F. Di Rienzo, Link-wise artificial compressibility method, Journal of Computational Physics 231 (2012) 5109–5143. doi:10.1016/j.jcp.2012.04.027.

[44] C. Obrecht, P. Asinari, F. Kuznik, J.-J. Roux, Thermal link-wise artificial compressibility method: GPU implementation and validation of a double-population model, Computers & Mathematics with Applications 72 (2016) 375–385. doi:10.1016/j.camwa.2015.05.022.

[45] Z. Fan, F. Qiu, A. Kaufman, S. Yoakum-Stover, GPU Cluster for High Performance Computing, in: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing, SC '04, IEEE Computer Society, Washington, DC, USA, 2004, pp. 47–. doi:10.1109/SC.2004.26.

[46] J. Tölke, M. Krafczyk, TeraFLOP computing on a desktop PC with GPUs for 3D CFD, International Journal of Computational Fluid Dynamics 22 (2008) 443–456. doi:10.1080/10618560802238275.

[47] C. Obrecht, F. Kuznik, B. Tourancheau, J.-J. Roux, A new approach to the lattice Boltzmann method for graphics processing units, Computers & Mathematics with Applications 61 (2011) 3628–3638. doi:10.1016/j.camwa.2010.01.054.

[48] J. E. McClure, J. F. Prins, C. T. Miller, A novel heterogeneous algorithm to simulate multiphase flow in porous media on multicore CPU–GPU systems, Computer Physics Communications 185 (2014) 1865–1874. doi:10.1016/j.cpc.2014.03.012.

[49] A. Xu, L. Shi, T. S. Zhao, Accelerated lattice Boltzmann simulation using GPU and OpenACC with data management, International Journal of Heat and Mass Transfer 109 (2017) 577–588. doi:10.1016/j.ijheatmasstransfer.2017.02.032.

[50] A. R. G. Harwood, A. J. Revell, Interactive flow simulation using Tegra-powered mobile devices, Advances in Engineering Software 115 (2018) 363–373. doi:10.1016/j.advengsoft.2017.10.005.

[51] A. R. G. Harwood, P. Wenisch, A. J. Revell, A Real-Time Modelling and Simulation Platform for Virtual Engineering Design and Analysis, in: Proceedings of 6th European Conference on Computational Mechanics (ECCM 6) and 7th European Conference on Computational Fluid Dynamics (ECFD 7), Glasgow, 2018, pp. 11–15.

[52] C. C. Su, M. R. Smith, F. A. Kuo, J. S. Wu, C. W. Hsieh, K. C. Tseng, Large-scale simulations on multiple Graphics Processing Units (GPUs) for the direct simulation Monte Carlo method, Journal of Computational Physics 231 (2012) 7932–7958. doi:10.1016/j.jcp.2012.07.038.

[53] M. J. Goldsworthy, A GPU–CUDA based direct simulation Monte Carlo algorithm for real gas flows, Computers & Fluids 94 (2014) 58–68. doi:10.1016/j.compfluid.2014.01.033.

[54] R. Jambunathan, D. A. Levin, CHAOS: An octree-based PIC-DSMC code for modeling of electron kinetic properties in a plasma plume using MPI-CUDA parallelization, Journal of Computational Physics 373 (2018) 571–604. doi:10.1016/j.jcp.2018.07.005.

[55] A. Frezzotti, G. P. Ghiroldi, L. Gibelli, Solving model kinetic equations on GPUs, Computers & Fluids 50 (2011) 136–146. doi:10.1016/j.compfluid.2011.07.004.

[56] A. Frezzotti, G. P. Ghiroldi, L. Gibelli, Solving the Boltzmann equation on GPUs, Computer Physics Communications 182 (2011) 2445–2453. doi:10.1016/j.cpc.2011.07.002.

[57] A. Frezzotti, G. P. Ghiroldi, L. Gibelli, Direct solution of the Boltzmann equation for a binary mixture on GPUs, in: AIP Conference Proceedings, Vol. 1333, 2011, pp. 884–889. doi:10.1063/1.3562757.

[58] Y. Y. Kloss, P. V. Shuvalov, F. G. Tcheremissine, Solving Boltzmann equation on GPU, Procedia Computer Science 1 (2010) 1083–1091. doi:j.procs.2010.04.120.

[59] V. V. Aristov, A. A. Frolova, S. A. Zabelok, V. I. Kolobov, R. R. Arslanbekov, Acceleration of Deterministic Boltzmann Solver with Graphics Processing Units, in: AIP Conference Proceedings, Vol. 1333, Pacific Grove, California, (USA), 2011, pp. 867–872. doi:10.1063/1.3562754.

[60] S. A. Zabelok, V. I. Kolobov, R. R. Arslanbekov, GPU Accelerated Kinetic Solvers for Rarefied Gas Dynamics, in: AIP Conference Proceedings, Vol. 1501, 2012, pp. 429–434. doi:10.1063/1.4769562.

[61] O. Rovenskaya, G. Croce, Numerical investigation of the effect of boundary conditions for a highly rarefied gas flow using the GPU accelerated Boltzmann solver, Computers & Fluids 110 (2015) 77–87. doi:10.1016/j.compfluid.2014.10.015.

[62] S. Zabelok, R. Arslanbekov, V. Kolobov, Multi-GPU Kinetic Solvers using MPI and CUDA, in: AIP Conference Proceedings, Vol. 1628, 2014, pp. 539–546. doi:10.1063/1.4902640.

[63] V. A. Titarev, Implicit high-order method for calculating rarefied gas flow in a planar microchannel, Journal of Computational Physics 231 (2012) 109–134. doi:http://dx.doi.org/10.1016/j.jcp.2011.08.030.

[64] S. Chen, C. Zhang, L. Zhu, Z. Guo, A unified implicit scheme for kinetic model equations. Part I. Memory reduction technique, Science Bulletin 62 (2016) 119–129. doi:10.1016/j.scib.2016.12.010.

[65] E. M. Shakhov, Generalization of the Krook kinetic relaxation equation, Fluid Dynamics 3 (1968) 95–96. doi:10.1007/BF01029546.

[66] S. Pantazis, D. Valougeorgis, Rarefied gas flow through a cylindrical tube due to a small pressure difference, European Journal of Mechanics - B/Fluids 38 (2013) 114–127. doi:10.1016/j.euromechflu.2012.10.006.

[67] H. Wasserman, A. Hoisie, A. Hoisie, O. Lubeck, O. Lubeck, Performance and Scalability Analysis of Teraflop-Scale Parallel Architectures using Multidimensional Wavefront Applications, International Journal of High Performance Computing Applications 14 (2000) 330–346. doi:0.1177/109434200001400405.

[68] S. Moustafa, I. Dutka-Malen, L. Plagne, A. Ponçot, P. Ramet, Shared memory parallelism for 3D Cartesian discrete ordinates solver, Annals of Nuclear Energy 82 (2015) 179–187. doi:10.1016/j.anucene.2014.08.034.

[69] T. Deakin, S. McIntosh-Smith, M. Martineau, W. Gaudin, An improved parallelism scheme for deterministic discrete ordinates transport, International Journal of High Performance Computing Applications 32 (2016) 555–569. doi:10.1177/1094342016668978.

[70] L. Zhu, X. Yang, Z. Guo, Thermally induced rarefied gas flow in a three-dimensional enclosure with square cross-section, Physical Review Fluids 2 (2017) 123402. doi:10.1103/PhysRevFluids.2.123402.

[71] D. B. Kirk, W.-m. W. Hwu, Programming Massively Parallel Processors: A Hands-on Approach, 3rd Edition, Morgan Kaufmann, 2016.

[72] T. J. Scanlon, E. Roohi, C. White, M. Darbandi, J. M. Reese, An open source, parallel DSMC code for rarefied gas flows in arbitrary geometries, Computers & Fluids 39 (2010) 2078–2089. doi:10.1016/j.compfluid.2010.07.014.

[73] J. C. Huang, K. Xu, P. B. Yu, A unified gas-kinetic scheme for continuum and rarefied flows II: Multi-dimensional cases, Communications in Computational Physics 12 (2012) 662–690. doi:10.4208/cicp.030511.220911a.

[74] L. Zhu, S. Chen, Z. Guo, dugksFoam: An open source OpenFOAM solver for the Boltzmann model equation, Computer Physics Communications 213 (2016) 155–164. doi:10.1016/j.cpc.2016.11.010.

[75] N. Corporation, Profiler User's Guide v9.2 (2018).
URL https://docs.nvidia.com/cuda/profiler-users-guide/index.html

[76] K. Aoki, C. Bardos, C. Dogbe, F. Golse, A note on the propagation of boundary induced discontinuities in kinetic theory, Mathematical Models and Methods in Applied Sciences 11 (2001) 1581–1595. doi:10.1142/S0218202501001483.

[77] K. Aoki, S. Takata, H. Aikawa, F. Golse, A rarefied gas flow caused by a discontinuous wall temperature, Physics of Fluids 13 (2001) 2645–2661. doi:10.1063/1.1389283.

[78] S. Naris, D. Valougeorgis, The driven cavity flow over the whole range of the Knudsen number, Physics of Fluids 17 (2005) 097106. doi:10.1063/1.2047549.

[79] M. T. Ho, J. Li, L. Wu, J. M. Reese, Y. Zhang, A comparative study of the DSBGK and DVM methods for low-speed rarefied gas flows, Computers & Fluids 181 (2019) 143–159. doi:10.1016/j.compfluid.2019.01.019.