AIP Conference Proceedings

Fast deterministic solution of the full Boltzmann equation on graphics processing units

Cite as: AIP Conference Proceedings **2132**, 060001 (2019); https://doi.org/10.1063/1.5119541
Published Online: 05 August 2019

Shashank Jaiswal, Jingwei Hu, and Alina A. Alexeenko

View Online

Export Citation

AIP | Conference Proceedings

Get 30% off all print proceedings!

Enter Promotion Code PDF30 at checkout

AIP Publishing

© 2019 Author(s).

Fast deterministic solution of the full Boltzmann equation on Graphics Processing Units

Shashank Jaiswal[1], Jingwei Hu[2] and Alina A. Alexeenko[1,a)]

[1]*School of Aeronautics & Astronautics, Purdue University, West Lafayette, IN 47907, USA*
[2]*Department of Mathematics, Purdue University, West Lafayette, IN 47907, USA*

[a)]Corresponding author: alexeenk@purdue.edu

Abstract. The Boltzmann equation, a six-dimensional integro-differential equation, governs the fluid flow behavior at molecular level for a wide range of physical phenomena, including shocks, turbulence, diffusion, and non-equilibrium chemistry which are beyond the reach of continuum fluid flow modelling based on the Navier-Stokes equations. Despite Boltzmann equation's wide applicability, its deterministic solution presents a huge computational challenge, and has been so far tractable only in simplified forms. We implement the Discontinuous Galerkin Fast Spectral (DGFS) method (Jaiswal, Alexeenko, and Hu 2019 [1]) for solving the full Boltzmann equation on streaming multi-processors. The proposed method is flexible and robust allowing: a) arbitrary unstructured geometries, b) control of spatial accuracy using high-order polynomial approximation without compromising simulation stability, c) exponential error convergence (spectral accuracy) in velocity space, and d) compact nature of DG as well as collision operator thus minimizing communication and maximizing parallel efficiency. The DG operators (for instance gradient, curl, etc), as well as the collision operator is applied in an element-local way, with flux-based element-to-element coupling. It is this locality that equips DGFS with strong parallel performance on streaming multi-processors. In the present work, we describe, devise and implement DGFS for General-Purpose Graphics Processing Units (GP-GPU). We consider the simulations of 0D spatially homogeneous, and rarefied 1D Fourier heat transfer. A speedup of approximately 10–100x, and parallel efficiency of 0.95 is demonstrated on multi-CPU/multi-GPU architectures. It is this speedup that now allows researchers to solve problems within a day that would otherwise take months on traditional CPUs. The key optimizations and techniques used to achieve these GPU performance results have been highlighted.

Introduction

For numerical simulations of flow behavior, the Navier-Stokes (NS) equations [2] are widely used. For aerospace applications in particular, NS have been used in simulating flow phenomenon in early 90's space-shuttle missions [3], as well as the recent deep-space Space Launch System (SLS) launch vehicles [4]. Yet, the Navier-Stokes fails when it comes to simulation of flow in low density environment, for instance, the atmospheric re-entry at high altitudes, as well as the flows in micro-geometries such as blood vessels.

Perhaps, more fundamental than Navier-Stokes, is the Boltzmann equation [5] which models the molecule-molecule interaction. In theory, under non-equilibrium conditions, Navier-Stokes equations can be derived as the asymptotic limit of the Boltzmann equation [6, 7]. The Boltzmann equation, a six-dimensional integro-differential equation, governs the fluid flow behavior for a wide range of physical phenomena, including shocks, turbulence, diffusion, and non-equilibrium chemistry which are beyond the reach of Navier-Stokes equations. Yet, the numerical solution of this equation presents a huge computational challenge even on today's petascale clusters. The equation is often deliberately simplified for engineering applications, primarily due to its high dimensionality (the six-dimensional nature: $\sim O(N^6)$ computations, with N being the number of points in each direction of velocity mesh); and the complicated collision operator.

The Boltzmann equation is a *phase-space* equation which describes molecular interaction statistically. A point in phase-space is described by three position coordinates (physical-space), and three velocity coordinates (velocity-space) resulting in six-dimensional solution space. In velocity-space, to determine how molecules collide, and what happens after collision (pre and post-collision states), we are required to solve a collision integral. A direct algorithm will require $O(N^6)$ computations for the solution of collision integral [8]. In the present work, we implement the

31st International Symposium on Rarefied Gas Dynamics
AIP Conf. Proc. 2132, 060001-1–060001-14; https://doi.org/10.1063/1.5119541
Published by AIP Publishing. 978-0-7354-1874-5/$30.00

recently-introduced fast-spectral method [9] for solving *full* Boltzmann equation. The method has computational complexity of $O(MN^4 \log N)$, where $M \ll N^2$ is number of points on half-sphere (details on complexity in section a)), and exhibits spectral convergence behavior. In particular, the method is applicable for general molecular interactions. More than > 99% of the computation-time is spent on the solution of this collision integral, which in turn presents the possibility of two-order-of-magnitude speedup in computation time by accelerating solvers for this step.

In addition to the solution of collision integral in velocity-space, we are required to advect the Boltzmann equation in physical space. To this end, we use discontinuous Galerkin (DG) [10] method. The DG method is amenable to high-order spatial and temporal accuracy, needed for robust and flexible numerics. Moreover, the method requires minimal processor-to-processor communication. To be more specific, the communication *strictly* takes place at the processor boundaries, and usually requires a single *collective* near-neighbor data-exchange at each timestep[1]. This equips the method with excellent nearly-linear scaling characteristics on large-scale clusters, as well as heterogeneous architectures (see [11, 12] for instance).

Related work

The Boltzmann equation has so far been tractable only in simplified forms such as linearized-Boltzmann (LB) [13], Bhatnagar-Gross-Krook (BGK) [14], and ellipsoidal Bhatnagar-Gross-Krook (ES-BGK) [15]. Implementations of DG method for solving the BGK and ES-BGK equations can be found in [16] for 0D/1D, and [17] for 2D flow problems. Low-order finite-volume based serial implementations of the Boltzmann equation for anisotropic scattering can be found in [18]. Without being exhaustive, we refer to [1, 19, 20] for a comprehensive review. To the best of our knowledge, on GPU, DG discretization in physical space coupled with fast spectral method in velocity space for general molecular interactions, hasn't been applied for solving the full Boltzmann equation.

Contribution

In this work, our contribution is four folds:

- First high-order spatially and temporally accurate Boltzmann equation solver applicable for *general* molecular interaction on single/multiple GPUs.
- Discussion of the fundamental ideas behind the algorithm and implementation which makes it amenable to excellent nearly-linear scaling on large clusters.
- Verification and Benchmarking of the method/implementation for standard rarefied flows.

Paper organization

In the section that follows, we give a brief overview of the Boltzmann equation, as well as details of Galerkin projection needed for high-order spatially-and-temporally accurate solution of partial differential equations. It discusses the key ideas of the DG method in the process. The *collision operator* section describes the fast Fourier spectral method, including the psuedo-code of the algorithm. Benchmark tests and results are discussed in the penultimate section. Concluding remarks and future work are presented in the last section.

The Boltzmann equation

The non-dimensional Boltzmann equation for a single-species, monatomic gas without external forces can be written as

$$\frac{\partial f}{\partial t} + \mathbf{c} \cdot \nabla_{\mathbf{x}} f = \frac{1}{\mathrm{Kn}} Q(f, f), \quad t \geq 0, \; \mathbf{x} \in \Omega_x, \; \mathbf{c} \in \mathbb{R}^3, \tag{1}$$

where $f = f(t, \mathbf{x}, \mathbf{c})$ is the one-particle probability density function (PDF) of time t, position $\mathbf{x}$, and particle velocity $\mathbf{c}$. $Q(f, f)$ is the collision operator describing the binary collisions among particles, and acts only in the velocity space:

$$Q(f, f)(\mathbf{c}) = \int_{\mathbb{R}^3} \int_{S^2} \mathcal{B}(\mathbf{c} - \mathbf{c}_*, \sigma)[f(\mathbf{c}')f(\mathbf{c}'_*) - f(\mathbf{c})f(\mathbf{c}_*)] \, \mathrm{d}\sigma \, \mathrm{d}\mathbf{c}_*, \tag{2}$$

[1]In particular for spectral/DG 3-D finite-element codes, for instance Nek5000, the GSLIB (`https://github.com/gslib`) implementation of Gather for *collective* near-neighbor data exchange have been shown to scale extremely-well beyond thousand-cores.

where $(\mathbf{c}, \mathbf{c}_*)$ and $(\mathbf{c}', \mathbf{c}'_*)$ denote the pre and post collision velocity pairs, which are related through momentum and energy conservation as

$$\mathbf{c}' = \frac{\mathbf{c} + \mathbf{c}_*}{2} + \frac{|\mathbf{c} - \mathbf{c}_*|}{2}\sigma, \quad \mathbf{c}'_* = \frac{\mathbf{c} + \mathbf{c}_*}{2} - \frac{|\mathbf{c} - \mathbf{c}_*|}{2}\sigma, \tag{3}$$

with the vector σ varying over the unit sphere $\mathcal{S}^2$. The quantity $\mathcal{B}$ (≥ 0) is the collision kernel depending only on $|\mathbf{c} - \mathbf{c}_*|$ and the scattering angle χ (angle between $\mathbf{c} - \mathbf{c}_*$ and $\mathbf{c}' - \mathbf{c}'_*$), and can be expressed as

$$\mathcal{B}(\mathbf{c} - \mathbf{c}_*, \sigma) = B(|\mathbf{c} - \mathbf{c}_*|, \cos\chi), \quad \cos\chi = \frac{\sigma \cdot (\mathbf{c} - \mathbf{c}_*)}{|\mathbf{c} - \mathbf{c}_*|}. \tag{4}$$

In the widely-used variable soft sphere (VSS) model, the function B and the Knudsen number Kn are given as

$$B = b_{\omega,\alpha}\, |\mathbf{c} - \mathbf{c}_*|^{2(1-\omega)}\, (1 + \cos\chi)^{\alpha-1}, \quad b_{\omega,\alpha} = \frac{\alpha}{2^{2-\omega+\alpha}\,\Gamma(2.5-\omega)\,\pi}, \quad \mathrm{Kn} = \frac{1}{\sqrt{2}\,\pi\, n_0\, d_{\mathrm{ref}}^2\, (T_{\mathrm{ref}}/T_0)^{\omega-0.5}\, H_0}. \tag{5}$$

Here Γ is the Gamma function, ω, α, d_{ref} and T_{ref} are, respectively, the viscosity index, scattering parameter, reference diameter, and reference temperature [21]. Additionally, H_0, T_0, and n_0 denote the characteristic length, characteristic temperature, and characteristic number density used for non-dimensionalization (see [1] for more details). Note that $0.5 \leq \omega \leq 1$, $1 \leq \alpha \leq 2$. In particular, when $\alpha = 1$, one recovers the variable hard sphere (VHS) model, wherein $\omega = 0.5$ corresponds to hard spheres and $\omega = 1$ corresponds to Maxwell molecules.

The discontinuous Galerkin (DG) formulation

For simplicity, we assume that the Boltzmann equation (1) is posed in the 1D domain $x \in [L, R] = \Omega_x$, subject to some initial condition

$$f(0, x, \mathbf{c}) = f_0(x, \mathbf{c}), \tag{6}$$

and boundary conditions at the left (L) and right (R) boundaries

$$f(t, L, \mathbf{c}) = f_L(t, \mathbf{c}), \quad c_1 > 0; \quad f(t, R, \mathbf{c}) = f_R(t, \mathbf{c}), \quad c_1 < 0, \tag{7}$$

where c_1 is the first component of $\mathbf{c}$.

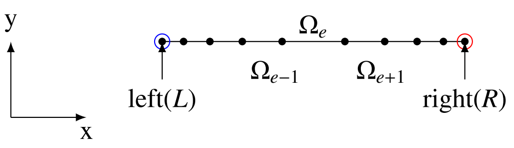

FIGURE 1. Illustration of 1D mesh.

Our objective is to find $f(t, \mathbf{x}, \mathbf{c})$. We start by partitioning the interval Ω_x into N_e non-overlapping elements as illustrated in Fig. 1 such that

$$\Omega \approx \bigcup_{e=1}^{N_e} \Omega_e, \quad e = \{1, \ldots, N_e\}; \quad \Omega_i \cap \Omega_j = \emptyset, \ \forall\, i \neq j, \quad i, j = \{1, \ldots, N_e\}. \tag{8}$$

An arbitrary element e is then defined in the interval $\Omega_e = [x_e^l, x_e^r]$, with x_e^l, x_e^r being the left and right ends of the element. In each element, we approximate the solution by a polynomial of order N_p (we leave velocity $\mathbf{c}$ and time t to be continuous as of now)

$$f_e(t, x, \mathbf{c}) = \sum_{l=1}^{K} \mathcal{F}_l^e(t,\, \mathbf{c})\, \phi_l^e(x), \quad x \in \Omega_e = [x_e^l, x_e^r], \tag{9}$$

where ϕ_l^e are the polynomial basis functions supported in Ω_e, $\mathcal{F}_l^e$ are the coefficients for the element e that need to be determined, and $K = N_p + 1$ is the total number of terms in the expansion. The reader is referred to [10] for details on *optimal* polynomial basis.

Substituting (9) into the equation (1) and conducting the standard Galerkin projection, we obtain

$$\sum_{l=1}^{K} \mathcal{M}_{ml} \frac{\partial}{\partial t} \mathcal{F}_l^e - c_1 \sum_{l=1}^{K} \mathcal{S}_{ml} \mathcal{F}_l^e = - \int_{\partial \Omega_e} \hat{n}_e \cdot (c_1 f_e)^* \phi_m^e \, \mathrm{d}x + \frac{1}{\mathrm{Kn}} \sum_{l_1, l_2 = 1}^{K} \mathcal{H}_{m\, l_1\, l_2} Q(\mathcal{F}_{l_1}^e, \mathcal{F}_{l_2}^e), \quad 1 \leq m \leq K, \tag{10}$$

where the matrices $\mathcal{M}_{ml}$, $\mathcal{S}_{ml}$, and tensor $\mathcal{H}_{m\, l_1\, l_2}$ are defined as

$$\mathcal{M}_{ml} = \int_{\Omega_e} \phi_m^e(x)\, \phi_l^e(x) \, \mathrm{d}x, \quad \mathcal{S}_{ml} = \int_{\Omega_e} \phi_l^e(x) \, \frac{\partial}{\partial x} \phi_m^e(x) \, \mathrm{d}x, \quad \mathcal{H}_{m\, l_1\, l_2} = \int_{\Omega_e} \phi_m^e(x)\, \phi_{l_1}^e(x)\, \phi_{l_2}^e(x) \, \mathrm{d}x. \tag{11}$$

These integrals can be evaluated exactly using the Gauss quadrature.

The first term on the right side of equation (10) is commonly referred as *numerical flux*, which is important for: a) exchange of information between elements which are side-by-side, and b) enforcing boundary conditions as in (7). Specifically, $\hat{n}_e$ denotes the outward pointing normal of element e, and

$$(c_1 f_e)^* = \{\{c_1 f_e\}\} + |c_1| \frac{1-a}{2} [[f_e]], \quad 0 \leq a \leq 1, \tag{12}$$

with

$$\{\{c_1 f_e\}\} = \frac{c_1 f_e^- + c_1 f_e^+}{2}, \quad [[f_e]] = \hat{n}_e^- f_e^- + \hat{n}_e^+ f_e^+. \tag{13}$$

Here, the superscript '+' denotes the information from *exterior* of the element, and '-' denotes the information from the *interior* of the element. For example: For the element Ω_e, at the left-edge, the '+' information is the one contributed by element Ω_{e-1}, and '-' is the information contributed by the element Ω_e itself. Similarly at the right-edge of the element Ω_e, the '+' information is the one contributed by element Ω_{e+1}, and '-' is the information contributed by the element Ω_e itself. Note that for the first-element $\Omega_{e=1}$, the '+' information is contributed by the left-boundary condition, and for the last-element $\Omega_{e=N_e}$, the '+' information is contributed by the right-boundary condition. Thus, the numerical flux automatically incorporates and enforces the boundary conditions. The role of the flux is to guarantee stability of the formulation by mimicking the flow of information in the underlying partial differential equation. In particular for scalar linear advection with constant advection velocity (as in our problem), the upwind flux ($a = 0$ in (12)) provides a highly accurate solution to the Riemann problem. Details about other numerical fluxes can be found in [2, 10] for instance.

The equation (10) can be evolved in time using the explicit Strong Stability Preserving Runge-Kutta (SSP-RK) method [22] provided one properly discretizes the velocity space and evaluates the collision operator, which we shall describe in the next section.

The Fourier spectral (FS) method for the collision operator

To discretize the velocity space, we employ a finite difference method, i.e., each velocity dimension is discretized uniformly with N points in a chosen interval $[-L, L]$. Given the function values $\mathcal{F}_{l_1}^e$, $\mathcal{F}_{l_2}^e$ at N^3 velocity grid points, the fast Fourier spectral method is called as a black box solver to return the values $Q(\mathcal{F}_{l_1}^e, \mathcal{F}_{l_2}^e)$ on the same grid with $O(MN^4 \log N)$ complexity, where M is the number of discretization points on the sphere and $M \ll N$. For completeness, we summarize the main steps of the method below. More details can be found in [1, 9].

In the nutshell, given $f(\mathbf{c})$ and $g(\mathbf{c})$, to evaluate $Q(f, g)(\mathbf{c})$, one proceeds as follows.

- Replace the collision kernel by its symmetrized version:

$$B_{\mathrm{sym}}(|\mathbf{c} - \mathbf{c}_*|, \cos\chi) = \frac{B(|\mathbf{c} - \mathbf{c}_*|, \cos\chi) + B(|\mathbf{c} - \mathbf{c}_*|, -\cos\chi)}{2}. \tag{14}$$

- Change the variable $\mathbf{c}_*$ to the relative velocity $\mathbf{c}_r = \mathbf{c} - \mathbf{c}_*$:

$$Q(f, g)(\mathbf{c}) = \int_{\mathbb{R}^3} \int_{\mathcal{S}^2} B_{\mathrm{sym}}(|\mathbf{c}_r|, \sigma \cdot \hat{\mathbf{c}}_r) [f(\mathbf{c}') g(\mathbf{c}'_*) - f(\mathbf{c}) g(\mathbf{c} - \mathbf{c}_r)] \, \mathrm{d}\sigma \, \mathrm{d}\mathbf{c}_r, \tag{15}$$

where $\hat{\mathbf{c}}_r$ is the unit vector along $\mathbf{c}_r$, and

$$\mathbf{c}' = \mathbf{c} - \frac{\mathbf{c}_r}{2} + \frac{|\mathbf{c}_r|}{2} \sigma, \quad \mathbf{c}'_* = \mathbf{c} - \frac{\mathbf{c}_r}{2} - \frac{|\mathbf{c}_r|}{2} \sigma. \tag{16}$$

- Determine the extent of velocity domain $D_L = [-L, L]^3$, and periodically extend f, g to $\mathbb{R}^3$. Truncate the integral in $\mathbf{c}_r$ to a ball B_R with $R = \frac{4}{3+\sqrt{2}}L$.
- Approximate f, g by truncated Fourier series

$$f^N(\mathbf{c}) = \sum_{k=-N/2}^{N/2-1} \hat{f}_k e^{i\frac{\pi}{L}k\cdot\mathbf{c}}, \quad g^N(\mathbf{c}) = \sum_{k=-N/2}^{N/2-1} \hat{g}_k e^{i\frac{\pi}{L}k\cdot\mathbf{c}}. \tag{17}$$

  Note here k is a three-dimensional index.
- Substitute f^N, g^N into (15), and perform the Galerkin projection

$$\hat{Q}_k := \frac{1}{(2L)^3}\int_{D_L} Q(f^N, g^N)(\mathbf{c}) e^{-i\frac{\pi}{L}k\cdot\mathbf{c}}\,\mathrm{d}\mathbf{c} = \sum_{\substack{l,m=-N/2\\ l+m=k}}^{N/2-1} [G(l,m) - G(m,m)]\hat{f}_l\hat{g}_m, \quad k = -N/2, \dots, N/2-1, \tag{18}$$

  where the kernel mode G is given by

$$G(l,m) = \int_{B_R}\int_{\mathcal{S}^2} B_{\text{sym}}(|\mathbf{c}_r|, \sigma\cdot\hat{\mathbf{c}}_r)\; e^{-i\frac{\pi}{L}\frac{l+m}{2}\cdot\mathbf{c}_r + i\frac{\pi}{L}|\mathbf{c}_r|\frac{l-m}{2}\cdot\sigma}\,\mathrm{d}\sigma\,\mathrm{d}\mathbf{c}_r. \tag{19}$$

It is clear that the direct evaluation of $\hat{Q}_k$ (for all k) would require $O(N^6)$ complexity. But if we can find a low-rank, separated expansion of $G(l,m)$ as

$$G(l,m) \approx \sum_{r=1}^{R} \alpha_r(l+m)\,\beta_r(l)\,\gamma_r(m), \tag{20}$$

then the gain term (positive part) of $\hat{Q}_k$ can be rearranged as

$$\hat{Q}_k^+ = \sum_{r=1}^{R} \alpha_r(k) \sum_{\substack{l,\,m=-N/2\\ l+m=k}}^{N/2-1} \left(\beta_r(l)\hat{f}_l\right)\,(\gamma_r(m)\hat{g}_m), \tag{21}$$

which is a convolution of two functions $\beta_r(l)\hat{f}_l$ and $\gamma_r(m)\hat{g}_m$, hence can be computed via fast Fourier transform (FFT) in $O(RN^3\log N)$ operations. Note that the loss term (negative part) of $\hat{Q}_k$ is readily a convolution and can be computed via FFT in $O(N^3\log N)$ operations.

In order to find the approximation in (20), we simplify (19) as (using the symmetry of the kernel)

$$G(l,m) = 2\int_0^R \int_{\mathcal{S}^{2+}} F(l+m, \rho, \sigma)\cos\left(\frac{\pi}{L}\rho\frac{l-m}{2}\cdot\sigma\right)\mathrm{d}\sigma\,\mathrm{d}\rho, \tag{22}$$

where $\mathcal{S}^{2+}$ denotes the half sphere, and

$$F(l+m, \rho, \sigma) := 2\rho^2\int_{\mathcal{S}^{2+}} B_{\text{sym}}(\rho, \sigma\cdot\hat{\mathbf{c}}_r)\cos\left(\frac{\pi}{L}\rho\frac{l+m}{2}\cdot\hat{\mathbf{c}}_r\right)\mathrm{d}\hat{\mathbf{c}}_r. \tag{23}$$

Now using the fact that $\cos(\alpha-\beta) = \cos\alpha\cos\beta + \sin\alpha\sin\beta$, if we approximate the integral in (22) by a quadrature, we obtain

$$G(l,m) \approx 2\sum_{\rho,\sigma} w_\rho w_\sigma F(l+m,\rho,\sigma)\left[\cos\left(\frac{\pi}{L}\rho\frac{l}{2}\cdot\sigma\right)\cos\left(\frac{\pi}{L}\rho\frac{m}{2}\cdot\sigma\right) + \sin\left(\frac{\pi}{L}\rho\frac{l}{2}\cdot\sigma\right)\sin\left(\frac{\pi}{L}\rho\frac{m}{2}\cdot\sigma\right)\right], \tag{24}$$

which is exactly in the desired form (20).

The aforementioned method is applicable for general molecular interactions. For simplicity, In the present work, we will focus on VHS kernel, for which F does not depend on σ, and is given by a closed-form analytical expression

$$F(l+m,\rho) = 4\,\pi\,b_{\omega,\alpha}\,\rho^{\gamma+2}\,\text{Sinc}\left(\frac{\pi}{L}\rho\frac{|l+m|}{2}\right) \tag{25}$$

where $\mathrm{Sinc}(x) = \sin(x)/x$, $\gamma = 2(1-\omega)$ is the collision-parameter (see eqn. (5)).

Further, we note that $G(m,m)$ in Eq. (18) has an analytical form for VHS molecular interactions, which can be expressed as

$$G(m,m) = 16\,\pi^2\, b_{\omega,\alpha} \int_0^R \rho^{\gamma+2}\, \mathrm{Sinc}\Big(\frac{\pi}{L}\rho|m|\Big)\mathrm{d}\rho \tag{26}$$

This can be approximated using a quadrature as

$$G(m,m) \approx \sum_{\rho} 16\,\pi^2\, b_{\omega,\alpha}\, w_\rho\, \rho^{\gamma+2}\, \mathrm{Sinc}\Big(\frac{\pi}{L}\rho|m|\Big) \tag{27}$$

The collision operator algorithm

As simple as the formulations in the last sections appear, there is often a leap between mathematical formulations and an actual implementation of the algorithms. Note that Q collision-procedure will be called most number of times in a simulation. Therefore, our motive is to avoid spurious computation for every timestep even if it means sacrificing some memory. We first outline the procedure for pre-computing variables that can be stored for reuse during the course of the simulation.

- First, we precompute $(\pi/L\ \rho\ l/2 \cdot \sigma)$. We use Gauss-Lobatto-Jacobi quadrature (with $\alpha = 0, \beta = 0$) for integration. So ρ, the GLQ zeros, is an array of size N. Additionally, we use *spherical design* [23] quadrature on sphere. So, σ, the spherical-quadrature zeros, is an array of size M. l as previously defined is the 3-D velocity-space index. So l is an array of size N^3. Hence $(\pi/L\ \rho\ l/2 \cdot \sigma)$ is precomputed and stored as a $N \times M \times N^3$ flattened row-major array a_{pqr}. This is described in steps 1–10 of Algo. (1).
- Second, we compute $F(l+m,\rho)$ as per Eq. (25). Note that $k = l+m$ is velocity-space index of size N^3. ρ, the GLQ zeros, is an array of size N. Since both $l+m$ and ρ do not change with time, the term $F(l+m,\rho)$ is precomputed and stored as a $N \times N^3$ flattened row-major array b_{pr}. This is described in step 12 of Algo. (1).
- Third, we precompute $G(m,m)$ as per Eq. (27). The output is stored as a N^3 flattened row-major array c_r. This is described in step 13 of Algo. (1).

Next we outline the procedure for computing Q. Recall that our motive is to compute (18)

- First, we compute the forward Fourier transform of $\mathcal{F}^e_{l_1}$, and $\mathcal{F}^e_{l_2}$ to obtain $\hat{f}_l$ and $\hat{g}_m$ respectively. This is described in step 1 of Algo. (2).
- Second, we compute $G(l,m)$ as in Eq. (24). Recall that $(\pi/L\,\rho\,l/2 \cdot \sigma)$ has been already precomputed and stored as a_{pqr}. Also recall that $F(l+m,\rho)$ has been precomputed and stored as b_{pr}. These can be reused to compute $G(l,m)$. This is described in step 2–17 of Algo. (2).
- Third, we perform convolutions to compute $\hat{Q}_k$ as in Eq. (18). Recall that $G(m,m)$ has been already precomputed and stored as c_r, and can be reused here. Finally, we perform inverse Fourier transform to obtain final Q. This is described in step 18 of Algo. (2).

It is to be emphasized that, although we select the widely-used collision-kernel (Equation: 5) for demonstration (see [21] for this choice of B), the algorithm procedure Algo (2) would still be applicable for all the possible choices of B in *practical* CFD computations. The pre-computation, however, needs to be adapted *accordingly* but only *slightly* for different choices of B.

The elegance of Collision-Algorithm (2) lies in its use of linear scalar algebraic operations to solve the complex collision-operator. This is generally critical for performance on GPU architectures (see [24] for details on memory layout, cache behavior, etc). It is this use of FFT and linear algebraic operations helps us to obtain excellent nearly-linear scaling characteristics on large-scale clusters needed for performing billion floating-point operations at every timestep (see efficiency Table 3).

There are five *fundamental* CUDA-kernels in our implementation as described below. A convex combination of these are used for implementing Algo. (2).

1. *FFT:* Performs *batched* forward Fourier transform. This is used in steps 1 and 9 of Algo. (2).
2. *IFFT:* Performs *batched* inverse Fourier transform. This is used in steps 8 and 18 of Algo. (2).

Algorithm 1: Pre-computation for Collision-Algorithm[9]

```
Input: Number of points in each-direction of velocity mesh $N$, number of points on half-sphere $M$, spherical
    quadrature weight $w_\sigma$, spherical quadrature-points $\sigma$ (vector-field size: $M$), Gauss quadrature-weights $w_\rho$ (size:
    $N$), Gauss quadrature-points $\rho$ (size: $N$), collision parameter $\gamma$, size of velocity mesh $L$
Output: a,b,c
    Declare:
      a (size: $MN \times N^3$), b (size: $N \times N^3$), c (size: $N^3$)
      $l$ (vector-field size: $N^3$), v (size: $N$)
 1: for $p = 0$ to $N - 1$ do
 2:   $v_p$ = p - (p ≥ N/2) × N
 3: end for
    // See octave function: [v0,v1,v2]=ndgrid(v)
 4: $l \leftarrow$ ndgrid(v)
    // Subscript p,q,r on symbols denote array-index
 5: for $p = 1$ to $N$ do
 6:   for $q = 1$ to $M$ do
 7:     for $r = 1$ to $N^3$ do
 8:       $a_{pqr} \leftarrow \pi/L \times \rho_p/2 \times (l_r \cdot \sigma_q)$
          // ( · ) denotes vector dot-product
 9:     end for
10:   end for
      // sinc(x) = sin(x)/x
      // |·| denotes vector-magnitude
11:   for $r = 1$ to $N^3$ do
12:     $b_{pr} \leftarrow b_{\omega,\alpha}\, \rho_p^{\gamma+2} \times 4\pi \times \mathrm{sinc}(\pi/L \times \rho_p/2 \times |l_r|)$
13:     $c_r \leftarrow c_r + 16\pi^2\, b_{\omega,\alpha}\, (w_\rho)_p \times \rho_p^{\gamma+2} \times \mathrm{sinc}(\pi/L \times \rho_p \times |l_r|)$
14:   end for
15: end for
16: return a,b,c
```

3. *SAXPY:* This performs the following algebraic operation

$$Z = \alpha \times X + Y \tag{28}$$

where X, Y, Z are arrays, and α is scalar. This is used in step 10 of Algo. (2).

4. *FXXY:* This performs the following algebraic operation

$$Z = \texttt{unaryOp}(X) \times Y \tag{29}$$

where X, Y, Z are arrays, and `unaryOp` is some unary-function. This is used, for instance, in steps 4–7 of Algo. (2). These types of cuda-kernels can be generated at compile-time either through templates or macro-substitution.

5. *SUM-REDUCE:* Computes the reduced-sum using divide-and-conquer tree-reduction.

$$\alpha = \sum_{i=1}^{\mathrm{length}(X)} X_i \tag{30}$$

where X is an array, and α is the reduced scalar.

The loops can be unrolled for explicit-vectorization (vectorized load/stores) hint to the compiler. We mention that DG-operators, for instance gradient, consumes less than 1% of total computing time in Boltzmann equation (see [12] for details GPU implementation of DG operators).

Algorithm 2: Collision-Algorithm Pseudo-code [9]

```
Input: Number of points in each-direction of velocity mesh $N$, Distribution-functions $\mathcal{F}^e_{l_1}$ and $\mathcal{F}^e_{l_2}$ (size: $N^3$), number
    of points on half-sphere $M$, spherical quadrature weight $w_\sigma$, Gauss quadrature-weights $w_\rho$ (size: $N$),
    precomputed variable a (size: $MN \times N^3$), precomputed variable b (size: $N \times N^3$), precomputed variable c (size:
    $N^3$)
Output: Q
    Declare:
      {t1,...,t6}, Q, QGs (each size: $N^3$), QG (size: $MN^4$)
 1: Compute forward FFT:
      FTf ← fft($\mathcal{F}^e_{l_1}$)
      FTg ← fft($\mathcal{F}^e_{l_2}$)
    /* These loops can be collapsed using omp pragma, albeit with additional memory-needs */
    // Subscript p,q on symbols denote array-index
    // Inner-most loop $r \in \{1,\ldots,N^3\}$ has been ignored
 2: for $p = 1$ to $N$ do
 3:   for $q = 1$ to $M$ do
 4:     t1 ← cos(a$_{pq}$) × FTf
        // Note: These are array-operations over $N^3$
 5:     t2 ← cos(a$_{pq}$) × FTg
 6:     t3 ← sin(a$_{pq}$) × FTf
 7:     t4 ← sin(a$_{pq}$) × FTg
        // ifft denotes inverse FFT
 8:     t5 ← ifft(t1)×ifft(t2) + ifft(t3)×ifft(t4)
 9:     t6 ← fft(t5)
10:     QG$_{pq}$ = $2 \times (w_\rho)_p \times w_\sigma \times$ b$_p$ × t6
11:   end for
12: end for
13: for $p = 1$ to $N$ do
14:   for $q = 1$ to $M$ do
15:     QGs += QG$_{pq}$
16:   end for
17: end for
    // real returns real part of complex number
18: Q = real( ifft(QGs) - $\mathcal{F}^e_{l_1}$*ifft(c × FTg) )
19: return Q
```

Benchmark Tests and Results

Bobylev-Krook-Wu (BKW) [25, 26], and Fourier heat transfer [27] problems are solved using the currently developed solver. BKW provides exact solution to Boltzmann equation when the convective-part is absent. This is useful for verification, computational-efficiency, and accuracy benchmarks. The second test-case i.e., Fourier heat-transfer, is a standard rarefied gas flow problem in which a gas is trapped between two walls at different temperatures. The gas-motion takes place as a result of this temperature-difference.

Hardware configuration

Serial and MPI-parallellized implementations are run on a 15-node cluster. Each node is equipped with two 12-core Intel Xeon Gold 6126 CPU, and three Tesla-P100 GPU. The operating system used is 64-bit CentOS 7.4.1708 (Core). GPU-parallellized implementations of solver are run on the same machine with NVIDIA Tesla-P100 GPU accompanying CUDA driver 8.0 and CUDA runtime 8.0. The GPU has 10752 CUDA cores, 16GB device memory, and compute capability of 6.0. The solver has been written in C++/CUDA and is compiled using OpenMPI 1.8.5, g++ 5.2.0, and nvcc 8.0.61 compiler with third level optimization flag. The benchmark-test are run using four solvers:

a) CPU/serial variant, b) CPU/MPI/parallel variant, c) GPU/CUDA/serial variant, and d) GPU/CUDA/MPI/parallel variant using CUDA-aware MPI. All the simulations are done with double precision floating point values.

BKW analytical solution

The accuracy and efficiency of implementation is evaluated by considering the standard benchmark case of Bobylev-Krook-Wu (BKW) [25, 26] for Maxwell-molecules. BKW provides an exact solution to the the spatially homogeneous Boltzmann equation for constant collision kernel. Equation (1), in the present case, simplifies to:

$$\frac{\partial f}{\partial t} = Q(f, f), \quad t > 0, \ v \in \mathbb{R}^3 \tag{31}$$

with the analytical solution expressed as

$$f(t, v) = \frac{1}{2 * (2\pi K(t))^{3/2}} \exp\left(-\frac{v^2}{2K(t)}\right)\left(\frac{5K(t) - 3}{K(t)} + \frac{1 - K(t)}{K^2(t)} v^2\right) \tag{32}$$

where $K(t) = 1 - \exp(-t/6)$. The initial time t_0 must be greater than $6 \ln(2.5) \approx 5.495$ for f to be positive. An arbitrary time of $t = 5.5$ has been picked in the present work. The distribution function f given in Eq. (32) must satisfy Eq. (31). Hence, the time-derivative of f yields

$$Q(f) = \frac{\partial f}{\partial t} = K'\left(-\frac{3}{2K} + \frac{v^2}{2K^2}\right) f + \left[\frac{1}{2(2\pi K)^{3/2}} \exp\left(-\frac{v^2}{2K}\right)\left(\frac{3}{K^2} + \frac{K - 2}{K^3} v^2\right)\right] K' \tag{33}$$

where $K'(t) = \exp(-t/6)/6$. By setting $\omega = 1$, $\alpha = 1$, $\mathrm{Kn}^{-1} = \Gamma(1.5)$ in the equation (5) and $\gamma = 0$, $b_{\omega,\alpha} = 1$ in equations (25–27), one can compute the constant kernel BKW solution.

In principle, to quantify the numerical error resulting from fast spectral approximation, one can compare the results obtained from the direct collision-operator evaluation procedure which requires $O(N^6)$ computations. Such comparisons have been already provided in [9]. In particular, tables 1-2 of [9] compare the accuracy and efficiency of the fast-spectral and the direct-spectral method. We refrain from repeating those results in the present work.

In the present section, we compare the L_∞ error between the analytical solution (33), and the numerical solution obtained using Algo. (2). Table 1 depicts the effect of velocity mesh-size on the error norm, and the speedup comparison between the CPU and GPU variants of the code. It is observed that the error *exponentially* decreases with increase in number of points in the velocity domain. Overall, since the error is on order of 10^{-8}, it indeed verifies that the Collision-Algorithm (2) provides the solution to Boltzmann Collision-operator. While the errors are on order of 10^{-8}, the *actual* error-norm value is dependent on several factors: a) the truncation of the infinitely-long velocity mesh to a finite $[-L, L]^3$ interval, b) the size $[-L, L]^3$ of the finite velocity-mesh, c) number of points in the velocity-mesh, d) choice of selected Gaussian quadrature, e) choice of spherical quadrature, f) finite precision round-off errors. The convergence behavior as depicted in the table, is well-known characteristics of Fourier spectral method [8, 9]. Table 1 also depicts the timings for single-evaluation of collision kernel in double precision. On a single-GPU, the code achieves a speedup of roughly 60-100x over the serial version of the code. For a fair comparison of speedup (CPU vs GPU), the multi-GPU/multi-CPU benchmarks are described in the next section.

Fourier heat transfer

For general Boltzmann equation (1), analytical solutions do not exist. Therefore, we compare our results with widely-accepted direct simulation Monte Carlo (DSMC) [21] method. We want to emphasize that DSMC is a stochastic-method for solution of the N-particle master kinetic equation which *converges* to the Boltzmann equation in the limit of infinite number of particles [28].

In the current test case, the coordinates are chosen such that the walls are parallel to the y direction and x is the direction perpendicular to the plates similar to one shown in Fig. (1). The two parallel walls have been set $H = 1\times10^{-3}$ meter apart, where left and right walls are fixed to $\mathbf{u}_w = (0\,0\,0)\, ms^{-1}$. The left wall i.e, cold junction temperature is fixed at $T_w = 263$ K while the right wall i.e, hot junction temperature is fixed at 283 K. The reference (T_{ref}) temperature is taken as 273 K. The Knudsen numbers are fixed corresponding to the pressure of 4 Pa, and 8 Pa respectively [2].

[2]In rarefied gas-dynamics, two widely-used definitions of Knudsen number exist. The first definition is by Cercignani [5] which yields Knudsen numbers of 1.582 and 0.791 respectively for the present case. The second definition is by Bird[21] (Eq. 5 in present work) which yields Knudsen numbers of 1.22 and 0.61 respectively.

TABLE 1. Weak scaling. The timings for collision kernel evaluation are averaged for 90 runs. Error is defined as maximum of absolute difference between $Q_{\text{numerical}}$ and Q_{exact} normalized by Q_{exact} i.e., $||Q_{\text{numerical}} - Q_{\text{exact}}||/||Q_{\text{exact}}||_{L_\infty}$

Velocity Mesh	Time (s)		Speedup	Error L_∞
	CPU	GPU		
$20 \times 20 \times 20$	0.1060	0.0052	20.38	0.00016982210
$24 \times 24 \times 24$	0.2681	0.0072	37.24	2.418682338e-05
$32 \times 32 \times 32$	0.8197	0.0151	54.28	5.234900906e-08
$36 \times 36 \times 36$	1.4470	0.0226	64.03	1.878435925e-08
$48 \times 48 \times 48$	4.8395	0.0647	74.80	1.875531057e-08
$64 \times 64 \times 64$	17.562	0.1932	90.90	1.876041262e-08
$72 \times 72 \times 72$	26.355	0.3064	86.02	1.875840274e-08

Argon with $\omega = 0.81$, $\alpha = 1$, and $d_{\text{ref}} = 4.17 \times 10^{-10}\, m$ is taken as working gas. We refer the reader to [27] for DSMC simulation parameters.

Verification

Figure (2) illustrates the temperature profile along the domain for different Knudsen numbers. The results are compared against the DSMC simulations, where our method/implementation: a) matches well with DSMC, b) captures the *non-linear* nature of temperature profiles in the near wall region, i.e., the Knudsen layer. The small discrepancies, in the results, are primarily due to: a) statistical fluctuations inherent to the Monte Carlo methods, b) practical limitations on number of particles used in DSMC simulations.

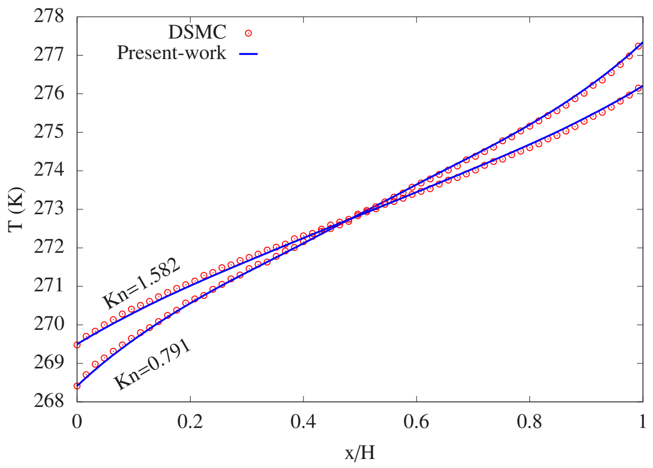

FIGURE 2. Variation of Temperature along the domain for $Kn = 0.791$, and 1.582 using Variable-Hard-Sphere collision model for Argon molecules obtained with DSMC and DGFS. The physical space consists of 20 cells and polynomial order of 3, and velocity space consists of 32^3 points.

Solver-Performance

The simulations are carried out on 18 different test-cases with varying element-count (N_e), polynomial-approximation order ($N_p = K - 1$), and velocity-space sizes (N). The spatial elements are distributed to different processors using the well-known domain-decomposition strategy. Each of these simulations involve at-least 100 billion floating-point operations at each timestep (recall computational-complexity: $N_e K^2 LMN^4 \log(N)$). Speed up obtained with multi-GPU/multi-CPU based solver is presented in Table (2). The CPU/serial implementation is computationally too expen-

TABLE 2. Performance of the GPU based solver for Fourier heat-transfer test case. Work units (±10 seconds) represent the total simulation time for first 10 timesteps. The phase-space is defined using a convenient triplet notation $\mathbf{N_e/K/N^3}$, which corresponds to $\mathbf{N}_e$ elements in physical space, **K** order DG (equivalently $N_p = K - 1$ order polynomial for 1-D domain), and $\mathbf{N}^3$ points in velocity space. n**P** ($n > 1$) denotes CPU/MPI/parallel execution on n cores shared equally across ($n/24$) nodes. Similarly, **1G** denotes GPU/CUDA/serial variant invocation on single GPU. n**G** ($n > 1$) denotes GPU/CUDA/MPI/parallel execution on n GPUs shared equally across ($n/3$) nodes. Speedup is defined as ratio **A/B**, where **A** and **B** are execution-times on designated number of CPUs/GPUs. **N/A** denotes the test-cases that are prohibitively expensive on designated number of CPUs/GPUs.

Phase space	Work Units (s)								Speedup			
	CPU			GPU								
	24P	48P	96P	1G	3G	6G	12G	24G	24P/1G	96P/3G	48P/96P	6G/12G
$384/3/20^3$	402.35	201.99	101.85	353.42	124.67	62.59	31.56	15.85	1.14	0.82	1.98	1.98
$384/3/32^3$	2936.38	1470.74	748.16	1019.56	344.07	172.37	86.62	43.51	2.88	2.17	1.97	1.99
$384/3/48^3$	N/A	9163.40	5156.78	4468.61	1493.02	746.75	374.12	187.92	N/A	3.45	1.78	1.99
$384/5/20^3$	1140.08	572.09	286.16	970.19	341.70	171.55	86.56	43.98	1.18	0.84	1.99	1.98
$384/5/32^3$	8192.15	4103.20	2056.43	2835.96	945.45	473.04	236.97	118.53	2.89	2.18	1.99	1.99
$384/5/48^3$	N/A	25071.90	12814.90	N/A	4141.67	2072.62	1036.71	518.59	N/A	3.09	1.96	1.99
$768/3/20^3$	818.49	409.39	205.84	713.97	245.03	125.68	63.75	31.98	1.15	0.84	1.99	1.97
$768/3/32^3$	5933.40	2970.49	1492.67	2054.55	687.74	344.17	173.30	86.92	2.89	2.17	1.99	1.99
$768/3/48^3$	N/A	18333.10	10454.30	8940.79	2987.43	1494.21	748.28	375.01	N/A	3.50	1.75	1.99
$768/5/20^3$	2273.04	1154.99	578.94	1937.42	682.22	345.72	172.87	86.90	1.17	0.85	1.99	1.99
$768/5/32^3$	16508.70	8309.91	4158.42	5659.84	1891.02	946.20	473.28	237.73	2.92	2.20	1.99	1.99
$768/5/48^3$	N/A	51016.50	25725.50	N/A	8287.70	4146.89	2074.31	1037.84	N/A	3.10	1.98	1.99
$1536/3/20^3$	1629.47	819.28	421.58	1464.46	490.44	248.93	126.53	63.62	1.11	0.86	1.94	1.97
$1536/3/32^3$	11903.40	5957.46	3028.59	4124.51	1380.95	690.78	345.60	172.93	2.89	2.19	1.97	1.99
$1536/3/48^3$	N/A	36734.01	18449.30	17883.99	5967.74	2986.42	1495.52	748.79	N/A	3.09	1.99	1.99
$1536/5/20^3$	4603.09	2357.83	1186.03	3931.94	1334.29	680.51	344.22	172.63	1.17	0.89	1.99	1.98
$1536/5/32^3$	32668.90	16406.30	8492.44	11336.84	3781.34	1892.39	947.18	474.32	2.88	2.25	1.93	1.99
$1536/5/48^3$	N/A	N/A	53277.60	N/A	16571.46	8289.56	4148.44	2078.09	N/A	3.22	N/A	1.99

TABLE 3. Parallel efficiency of the GPU/MPI implementation. Velocity mesh is fixed to $N^3 = 20^3$ with $M = 6$, and DG-order to $K = 3$. Here g and N_e denote the GPU count, and number of elements in physical-space respectively.

	$g = 1$ (1 core)	$g = 3$ (1 node)	$g = 6$ (2 nodes)	$g = 12$ (4 nodes)	$g = 24$ (8 nodes)
$N_e = 384$	1.00	0.9449	0.9411	0.9332	0.9290
$N_e = 768$	1.00	0.9713	0.9468	0.9333	0.9302
$N_e = 1536$	1.00	0.9953	0.9805	0.9645	0.9591

TABLE 4. Performance on billion grid-point phase spaces. The notations remain the same as in Table 2. Work units (± 10 seconds) represent the total simulation time for first 5 timesteps.

Phase space	Work Units (s)		Speedup
	30G (10 nodes)	42G (14 nodes)	30G/42G
$40320/3/20^3$	658.47	470.96	1.398
$40320/3/32^3$	1815.13	1296.63	1.399
$40320/3/48^3$	7844.28	5605.33	1.399

sive for the large size of phase-space considered in the benchmarks, and therefore have been omitted. As evident from the table, the acceleration due to GPU parallelization increases with increase in the size of computational grid. More specifically, the increase in N_e and K have small-effect on overall speedup which suggests that DG-operators (for instance derivative, time-evolution) are rather computationally-inexpensive operations. On the other hand, increase in velocity-grid drastically increases the speedup by a factor of 1.5–2. Increasing the velocity-mesh further, should increase the speedup as was observed in BKW-case in Tab. (1). On a single GPU, the implementation achieves a speedup of roughly 2–3x over the MPI-parallelized code running on 24 cores (1 node). On three GPUs of a single node, the implementation achieves a speedup of 0.8–4x over the MPI-parallelized code running on 96 cores (4 nodes). The strong scaling behavior for this case has been depicted in Fig (3).

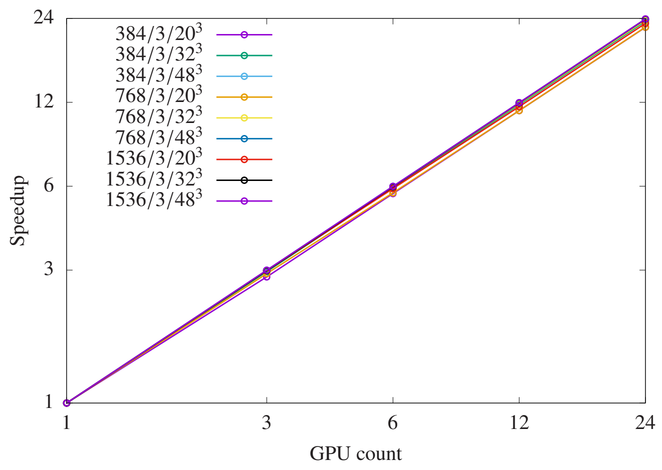

FIGURE 3. Strong scaling for Fourier heat transfer test case. Both the axes are on logarithmic scale.

Table (3) presents the parallel efficiency of the GPU/MPI implementation. The methodology presented in the current work exhibits excellent nearly-linear scaling characteristics. The effects of off-node communication do exist, but in the present case, are small. The off-node communication should be more apparent in simulations that use large number of nodes. We want to emphasize that the amount of computations is very high in our simulations. These simulations are more memory-bound and less I/O bound. The use of a “communication-free” collision integral that consumes more than 99% of total computational time, and “minimal-communication” DG method, both ensure that the I/O overhead is low. An important key observation is that the efficiency can be maintained provided we have enough work on each processor – one of the characteristics of scalable algorithms.

We further increased the size of phase space to billion grid points. Table (4) shows the performance of the solver on 14 and 12 nodes. The largest case requires nearly 100 trillion floating point calculations at every timestep. The speedup obtained is found to be proportional to the ratio of number of GPUs employed in computation (42/30 = 1.4).

Similar speedups and scaling behavior is observed for 2D/3D domains consisting of triangles, tetrahedrons, quadrilaterals, hexahedrons, prisms, and pyramids. Due to relatively large size of the present manuscript, they have been omitted here. A speedup of this magnitude, now enables us (and fellow researchers) to solve problems within a day that would otherwise take months on traditional CPU architectures. More importantly, It opens up the opportunity to explore family of Boltzmann equations, for instance, Vlasov-Fokker-Plank-Maxwell equations needed in plasma modelling, or Quantum-Boltzmann in semiconductor modelling, or Acoustic-Boltzmann, all of which still remain relatively unexplored for general 2D/3D problems.

Conclusions

We have presented a high-order discontinuous Galerkin fast spectral formulation for solving full Boltzmann equation on GPUs. The DG-type formulation employed in the present work has advantage of having high-order accuracy at the element-level, and its element-local compact nature (and that of our collision algorithm) enables effective parallelization on massively parallel architectures. For verification and benchmarks, we carry out spatially homogeneous BKW, and Fourier heat-transfer simulations. Speedups on order of 10–100x, and parallel efficiency close to 0.95 are observed on a multi-node multi-CPU/multi-GPU system. An important key observation is that the efficiency can be maintained provided we have enough work on each processor–one of the characteristics of scalable algorithms.

While the speedup observed in the present work are promising, our experience shows that even heavily tuned codes can be further improved. We want to emphasize that the multi-CPU (CPU/MPI) test-cases in the present work have been run on Intel Xeon-Gold. The higher-end Xeon-Phi series can further improve the observed speedup. It would be interesting to see how the method performs beyond thousand cores. Extending the implementation to general mixed grids coupled with adaptivity in physical and velocity spaces, is an interesting direction as well. Yet another future direction, would be extending the implementation for family of Boltzmann equations, for instance, Vlasov-Fokker-Plank equations for plasma modelling.

Acknowledgment

J. Hu’s research was supported by NSF grant DMS-1620250 and NSF CAREER grant DMS-1654152.

REFERENCES

[1] S. Jaiswal, A. Alexeenko, and J. Hu, Journal of Computational Physics **378**, 178–208 (2019).
[2] R. H. Pletcher, J. C. Tannehill, and D. Anderson, *Computational fluid mechanics and heat transfer* (CRC Press, 2012).
[3] J. Slotnick, M. Kandula, and P. Buning, “Navier-stokes simulation of the space shuttle launch vehicle flight transonic flowfield using a large scale chimera grid system,” in *12th Applied Aerodynamics Conference* (1994) p. 1860.
[4] S. E. Rogers, D. J. Dalle, and W. M. Chan, “CFD simulations of the space launch system ascent aerodynamics and booster separation,” in *53rd AIAA Aerospace Sciences Meeting* (2015) p. 0778.
[5] C. Cercignani, *The Boltzmann Equation and Its Applications* (Springer-Verlag, New York, 1988).
[6] C. Bardos, F. Golse, and D. Levermore, Journal of statistical physics **63**, 323–344 (1991).
[7] F. Bouchut, F. Golse, and M. Pulvirenti, *Kinetic equations and asymptotic theory* (Elsevier, 2000).
[8] L. Pareschi and G. Russo, SIAM J. Numer. Anal. **37**, 1217–1245 (2000).
[9] I. Gamba, J. Haack, C. Hauck, and J. Hu, SIAM J. Sci. Comput. **39**, B658–B674 (2017).
[10] J. S. Hesthaven and T. Warburton, *Nodal discontinuous Galerkin methods: algorithms, analysis, and applications* (Springer Science & Business Media, 2007).

[11] R. Biswas, K. D. Devine, and J. E. Flaherty, Applied Numerical Mathematics **14**, 255–283 (1994).
[12] A. Klöckner, T. Warburton, J. Bridge, and J. S. Hesthaven, Journal of Computational Physics **228**, 7863–7882 (2009).
[13] E. P. Gross and E. A. Jackson, The physics of fluids **2**, 432–441 (1959).
[14] P. L. Bhatnagar, E. P. Gross, and M. Krook, Physical review **94**, p. 511 (1954).
[15] L. H. Holway Jr, Physics of Fluids (1958-1988) **9**, 1658–1673 (1966).
[16] A. Alexeenko, C. Galitzine, and A. Alekseenko, "High-order discontinuous galerkin method for boltzmann model equations," in *40th Thermophysics Conference* (2008) p. 4256.
[17] W. Su, A. A. Alexeenko, and G. Cai, Computers & Fluids **109**, 123–136 (2015).
[18] L. Wu, C. White, T. J. Scanlon, J. M. Reese, and Y. Zhang, Journal of Computational Physics **250**, 27–52 (2013).
[19] L. Mieussens, "A survey of deterministic solvers for rarefied flows," in *Proceedings of the 29th International Symposium on Rarefied Gas Dynamics, AIP Conf. Proc*, Vol. 1628 (2014), pp. 943–951.
[20] G. Dimarco and L. Pareschi, Acta Numer. **23**, 369–520 (2014).
[21] G. A. Bird, *Molecular Gas Dynamics and the Direct Simulation of Gas Flows* (Clarendon Press, Oxford, 1994).
[22] S. Gottlieb, D. Ketcheson, and C.-W. Shu, *Strong Stability Preserving Runge-Kutta and Multistep Time Discretizations* (World Scientific, 2011).
[23] R. S. Womersley, in *Contemporary Computational Mathematics-A Celebration of the 80th Birthday of Ian Sloan* (Springer, 2018), pp. 1243–1285.
[24] A. R. Brodtkorb, T. R. Hagen, and M. L. Sætra, Journal of Parallel and Distributed Computing **73**, 4–13 (2013).
[25] A. Bobylev, "Exact solutions of the boltzmann equation," in *Akademiia Nauk SSSR Doklady*, Vol. 225 (1975), pp. 1296–1299.
[26] M. Krook and T. T. Wu, The Physics of Fluids **20**, 1589–1595 (1977).
[27] M. Gallis, D. Rader, and J. Torczynski, Physics of Fluids **14**, 4290–4301 (2002).
[28] A. Alexeenko and S. Gimelshein, in *The Handbook of Fluid Dynamics*, edited by R. Jonhson (CRC Press Boca Raton, FL, 2016), pp. 39:1–40.