Computers & Fluids 101 (2014) 114–125

Contents lists available at ScienceDirect

Computers & Fluids

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

Development of parallel direct simulation Monte Carlo method using a cut-cell Cartesian grid on a single graphics processor

M.-C. Lo [a], C.-C. Su [a], J.-S. Wu [a,*], F.-A. Kuo [a,b]

[a] Department of Mechanical Engineering, National Chiao Tung University, 1001 Ta-Hsueh Road, Hsinchu 30010, Taiwan
[b] National Center for High-Performance Computing, 7 R&D Road, Hsinchu 30076, Taiwan

ARTICLE INFO

Article history:
Received 13 December 2013
Received in revised form 21 May 2014
Accepted 3 June 2014
Available online 18 June 2014

Keywords:
DSMC
GPU
Cut-cell
Variable time-step
Transient adaptive sub-cell

ABSTRACT

This study developed a parallel two-dimensional direct simulation Monte Carlo (DSMC) method using a cut-cell Cartesian grid for treating geometrically complex objects using a single graphics processing unit (GPU). Transient adaptive sub-cell (TAS) and variable time-step (VTS) approaches were implemented to reduce computation time without a loss in accuracy. The proposed method was validated using two benchmarks: 2D hypersonic flow of nitrogen over a ramp and 2D hypersonic flow of argon around a cylinder using various free-stream Knudsen numbers. We also detailed the influence of TAS and VTS on computational accuracy and efficiency. Our results demonstrate the efficacy of using TAS in combination with VTS in reducing computation times by more than 10×. Compared to the throughput of a single core Intel CPU, the proposed approach using a single GPU enables a 13–35× increase in speed, which varies according to the size of the problem and type of GPU used in the simulation. Finally, the transition from regular reflection to Mach reflection for supersonic flow through a channel was simulated to demonstrate the efficacy of the proposed approach in reproducing flow fields in challenging problems.

1. Introduction

Since its invention by Bird [1] half a century ago, the direct simulation Monte Carlo (DSMC) method, which is based on particle collision kinetics, has become a common tool for the simulation of rarefied and non-equilibrium gas dynamics. It has been mathematically proven that the DSMC method is equivalent to solving the Boltzmann equation, should the number of simulation particles become sufficiently large [2,3]. The reason for not directly solving the integral–differential Boltzmann equation is the fact that it includes seven independent variables (time, positions, and velocities), which make this problem nearly intractable, even using the most powerful supercomputer. The problem is further exacerbated by the complex collision integral. Nonetheless, DSMC computation is far more demanding than solving the continuum Navier–Stokes equation, particularly when dealing with flow in the transitional or near-continuum regime. Thus, determining the means to reduce computational complexity remains an important research topic with numerous implications pertaining to rarefied gas dynamics.

In the past, it was common to apply physical domain decomposition using multiple CPUs in a distributed-memory machine (e.g., PC cluster) for the parallel computing of DSMC [e.g., 4–6]. In implementations using multiple instructions multiple data (MIMD), each processor works within its own domain using the standard DSMC method and communicates with other processors using a message passing interface (MPI) when particles move across the inter-processor boundaries. Graphics processing units (GPUs) have become an alternative computational platform for the parallel computing of scientific data, providing a high capability/price ratio when used within the single instruction multiple data (SIMD) paradigm. DSMC has a highly localized numerical scheme, which is a basic requirement for efficient computation on GPUs.

Very few studies have investigated DSMC simulation using GPU computing [7–9]. Su et al. [7] proposed a parallel two-dimensional DSMC method using a Cartesian structured grid on multiple GPUs for the simulation of rarefied gas dynamics. Compared to a single CPU core (Intel Xeon X5670, 2.93 GHz, 12 M Cache), they increased the speed of computation by 15× using a single GPU (Nvidia M2070, 6 GB DDR5 global memory) and by 185× using 16 GPUs in the computation of a large-scale near-continuum flow problem. These results demonstrated the impressive capability/price ratio of this approach. Despite the efficiency of Cartesian grids in tracking particles, treating flow problems using objects with a complex geometry can be exceedingly difficult. One alternative approach is to use an unstructured grid similar to that proposed by Wu and Lian [10] and Boyd [11]. However, two problems can arise

* Corresponding author. Tel.: +886 3 573 1693; fax: +886 3 611 0023.
E-mail address: chongsin@faculty.nctu.edu.tw (J.-S. Wu).

http://dx.doi.org/10.1016/j.compfluid.2014.06.003
0045-7930/© 2014 Elsevier Ltd. All rights reserved.

from the use of an unstructured grid. First, the efficiency of particle tracking is lower than that of a structured grid. Second porting a DSMC algorithm within a GPU environment using unstructured data can be very challenging. To overcome these difficulties, this study focuses on the cut-cell approach to treating geometrically complex objects in a Cartesian grid.

The quality of DSMC simulation strongly depends upon collision quality, which can be quantified using the "merit of collision", defined as the ratio of averaged binary collision distance to the local mean free path. Achieving a physically correct collision process requires that the merit of collision be less than unity [12]. Transient adaptive sub-cell (TAS) [13,14] and variable time-step (VTS) [15,16] approaches can be applied to improve collision quality and reduce the number of iterations required to reach a steady state. Thus, this study sought to apply the TAS and VTS algorithms in conjunction with the cut-cell approach for DSMC simulation using GPU computing. Two benchmark problems (hypersonic nitrogen flow past a ramp and hypersonic argon flow past a cylinder) were employed to evaluate the runtime efficiency associated with the modelling of rarefied gas dynamics and geometrically complex objects. The speedup of DSMC using various types of GPU cards is also investigated and compared. The high fidelity of the proposed single-GPU DSMC code was demonstrated by reproducing the transition from a regular reflection to a Mach reflection associated with supersonic flow through a channel.

2. Numerical method

2.1. Standard DSMC method

The direct simulation Monte Carlo method solves the Boltzmann equation based purely on particle collision kinetics statistically. One of the basic assumptions of the DSMC method is the decoupling of the movement and collisions of particles. The details of the DSMC method can be found in [1]. Briefly, the DSMC method involves initialization, particle movement, indexing, collisions, and sampling. In the initialization phase, the velocity of particles is sampled from an equilibrium Maxwell–Boltzmann distribution and the spatial position of the particles is randomly distributed in each cell. In the particle movement phase, all particles move according to their current phase-space states (3 positions and 3 velocities). The particles are relocated to their new spatial locations either through free flight or interaction with various types of walls (e.g., diffusive, specular, absorptive). Particles are removed when their new locations lie outside the computational domain. In the indexing phase, all particles are indexed with their resident cells to facilitate the efficient selection of particles during the collision phase. In the collision phase, two particles in each cell are randomly selected and the determination of whether they collide is probabilistic. In the event of a collision, the post-collision velocities are calculated according to the type of collision, such as elastic collision, translational–rotational energy transfer, translational–vibrational energy transfer, and reactive energy transfer. Prior to the selection of collision pairs in each cell during each time step, it is necessary that the maximal number of collision pairs be selected (e.g., No Time Counter, NTC [1]). In the sampling phase, the post-collision velocities are sampled (or accumulated) in order to calculate the macroscopic properties. With the exception of initialization, all of these procedures are repeated until the sample size is large enough.

2.2. Parallel DSMC on a single GPU

As shown in Fig. 1, this study adopted a nearly all-device (GPU) computational approach, in which all major procedures of the DSMC method, including particle movement, indexing, collision and sampling, are performed within the GPU. CUDA [17] is used to accelerate the DSMC-related simulation components as well as to transfer data between the CPU (host) memory and the GPU (device) global memory. Taking advantage of the forward architecture requires adaption of the original DSMC method to enable efficient all-device computation. This study adopted the algorithms proposed by Su et al. [7] for this function. Those developments briefly detailed in the following and the cut-cell approach, the TAS, and the VTS algorithms are outlined in a later section.

In the initialization phase, this study used the CUDA API function *cudaGetDeviceCount()* [18] to obtain the number of GPU devices available and *cudaSetDevice()* [18] to assign a GPU for computation (in the current study, only one GPU was employed). Input data and initial states were loaded into the memory on the host (CPU). We then used the CUDA API function *cudaMemcpy()* [18] to transfer the data (particle and cell) from host to the global memory of the device. The main procedures of the DSMC simulation (particle movement, indexing, collisions, and sampling), were then performed on the GPU. In the particle movement phase, *Np*/*Nthread* + 1 particles were tracked using a thread, in which *Np* is the total number of simulated particles and *Nthread* is the number of threads employed in the GPU device. Each thread reads/writes particle data from/to the global memory of the GPU device. The particle indexing phase of the computation is similar to the DSMC implementation in [1]. This study employed the Software Development Kit (SDK) of CUDA, *scanLargeArray* [18] to scan the data elements of large arrays contained within the global memory. This function was used to enable the efficient indexing of particles. In the collision phase, we employed a different parallelization philosophy in which all particle collisions within a cell are handled by a single thread, thereby allowing the efficient recollection of data since all of the data is coalesced. The speed of the sampling phase is increased by using the much faster shared memory in the GPU [17] for the temporary storage of sampling results. Upon the completion of sampling for several cells, the sampled data is transferred from the shared global memory.

2.3. Cut-cell approach

The cut-cell approach includes three kinds of computational cells for simulation, including fluid cells, solid cells, and cut cells, as shown in Fig. 2a. Fluid cells and solid cells are treated as usual in DSMC simulation; however, cut cells require special treatment. Fig. 2b illustrates the three basic types of cut cells in the two-dimensional domain, in which Types A, B, and C contain one, two, and three grid points in the solid body, respectively. The efficient implementation of the DSMC method requires an algorithm for the identification of cut cell type. This study employed the crossing number method [19], as outlined in the following.

If a solid body (or a polygon) is simple (i.e., no self-intersections), the crossing number method (also referred to as the even–odd rule) can be used to determine whether a point is included in the 2D solid body. This method assumes that a point is inside the polygon if an odd number of crossings occur with the edges of the polygon when a line is drawn from this point in an arbitrary direction; otherwise, it lies outside the polygon. In addition, the validity of this method was proven using the "*Jordan Curve Theorem*", which states that a simple closed curve divides a 2D plane into exactly 2 connected components: one bounded "inside" and one unbounded "outside". In the current study, we employed a more straightforward crossing number algorithm. This enables the selection of one horizontal ray (parallel to x-axis) and one vertical ray (parallel to y-axis), extending to the right and top of the grid, respectively. The use of these rays enables one to calculate the number and locations of the points intersecting with the solid bodies.

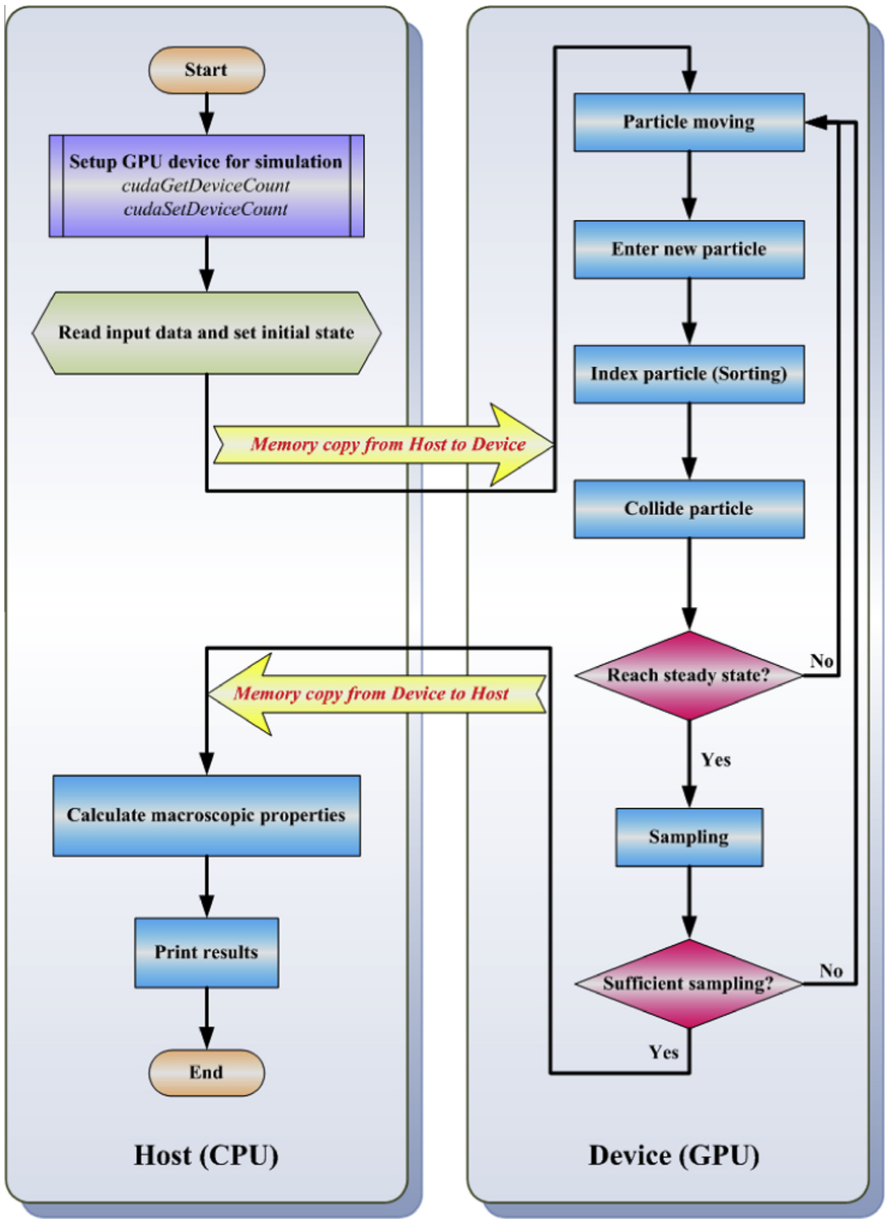


Fig. 1. Flowchart of DSMC running on a single GPU.

In actual applications, solid bodies in the flow domain are cut out of a background Cartesian grid with their boundaries represented using various types of cut cell. All of the cut cells must be identified and the volume of the corresponding fluid cells (area for the 2D case) must be calculated as follows:

1. The crossing number method is used to find the intersection points and determine whether the grid points in the vicinity of the solid bodies (potential cut cells) are inside the solid bodies. This procedure is detailed in Appendix A. In this step, the type of cut-cell is determined according to the number of grid points in the solid body (green[1] points), as shown in Fig. 2b.
2. Arrange the points intersecting the line segment, cell edges, and arbitrary grid points outside the solid body in a counter-clockwise fashion. Then calculate the slope of the line segment, which is consistent with and will be used by the particle tracking algorithm when considering the particle–wall interaction.
3. Calculate the fluid volumes of the cut cells.

A special treatment is required to handle particle interactions using the solid boundaries in the cut cells during DSMC simulation. When a particle collides with the wall, the reflected velocity is obtained according to the type of surface. At the same time, the coordinate transformation (depending upon the slope of the wall) must be linked directly to the slope of the boundary.

2.4. Variable time-step algorithm

In a typical supersonic/hypersonic flow problem, density often varies dramatically in the spatial domain. For the standard DMSC method using a Cartesian structured grid, a constant time-step (thus, a constant particle weight) is used throughout the

[1] For interpretation of color in Fig. 2, the reader is referred to the web version of this article.

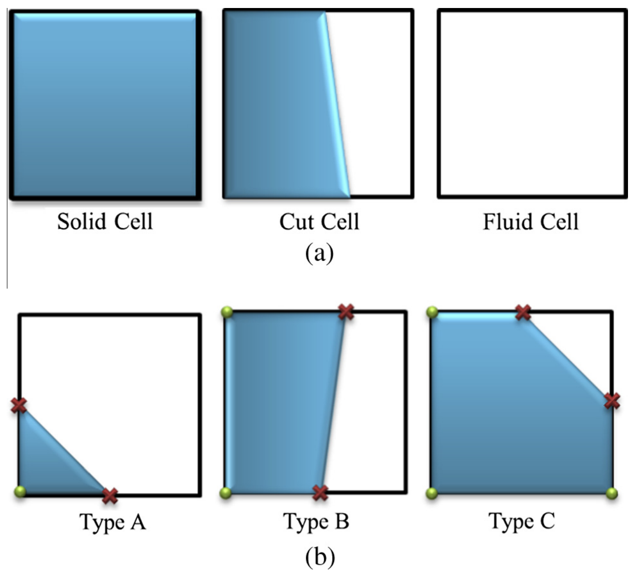


Fig. 2. Schematic diagram of computational cells: (a) cell types and (b) cut-cell types.

computation, resulting in wasted computational resources (a greater number of particles and longer times), particularly in the free-stream region. In addition, large statistical uncertainties regarding the properties of the sample may exist in the tiny cut-cells near the curved solid boundary due to the small number of particles within the cells. The variable time-step algorithm was proposed to overcome these problems by enforcing the conservation of flux (mass, momentum, and kinetic energy) for each moving particle when it crosses the interface between two neighboring cells [16]. This enforcement leads to the following form,

$$\frac{W_1 N_1 \Phi_1}{A\Delta t_1} = \frac{W_2 N_2 \Phi_2}{A\Delta t_2} \tag{1}$$

where W, N, Φ (=m, mv, $mv^2/2$ or other internal energy) and Δt are the particle weight, the number of simulated particles, the conserved flux quantities and the time-step size, respectively, with the subscripts used to identify the cell (1: source cell; 2: destination cell). A represents the area of the cell interface between two neighboring cells. It is obvious that there are some degrees of freedom in selecting these parameters, which satisfy Eq. (1). The best choice is to set $N_2 = 1$ (without particle cloning) and $\Phi_1 = \Phi_2$ (without changing the mass, momentum and energy of the moving particle) across the cell interface, such that $W_1/\Delta t_1 = W_2/\Delta t_2$ holds.

Using DSMC simulation in conjunction with VTS, we first used a constant time step with a small number of simulated particles ($\sim$1 particle per cell or even fewer) for a preliminary run over a short period of time. From this, we obtained a good estimate of the flow field, particularly with regard to density distribution. Based on the estimated density distribution, we then calculated a new local time step in each cell i as follows:

$$\Delta t_i = \frac{1}{2}\frac{\lambda_i}{\overline{V}_i + C_{mp_i}} \tag{2}$$

where $\lambda_i = 1/\sqrt{2}n_i\sigma$ is the local mean free path and $C_{mp_i} = \sqrt{2k_B T_i/m}$ is the most probable speed of the particles.

In formal DSMC simulation with a much larger number of particles (>10 particles per cell in the free stream), we scaled the local particle weight of cell i as follows:

$$W_i = W_\infty \frac{\Delta t_i}{\Delta t_\infty} \tag{3}$$

where subscript "∞" represents the free-stream condition. Once the time steps are assigned to all computational cells, the particles entering a new cell should be adapted to the size of the time step in the new destination cell when crossing the cell interface. In practice, the remaining time for a simulated particle in cell j should be rescaled according to the ratio of time-step sizes between the original (i) and destination (j) cells as follows:

$$\Delta t_{remaining,j} = (\Delta t_i - \Delta t_{moved,i})\frac{\Delta t_j}{\Delta t_i} \tag{4}$$

where $\Delta t_{moved,i}$ represents the time "moved" in cell i. The main advantage of implementing VTS is to improve the uniformity in particle distribution to enable the use of fewer particles with essentially the same statistical uncertainties across the computational domain. For example, in a tiny cut cell, one can reduce the time step (or particle weight) to increase the number of particles to improve the statistical sampling of the macroscopic data. Not only does this precisely satisfy the conservation of flux (mass, momentum, and energy) when a simulated particle moves across cell interface during simulation, but also it reduces tremendously the number of iterations required for the transition to reach a steady state. This also reduces the number of particles required for simulation resulting from variations in particle weight across the computational domain. Some specific example demonstrating the reduction of simulation particles and much smaller number of transient iterations to reach steady state by using the VTS scheme applied in unstructured-grid DSMC can be found in Fig. 7 of [16]. This makes it appropriate to use the VTS in a cut-cell grid system, which is one of the objectives in the current study.

2.5. Transient adaptive sub-cell algorithm

In the current study, we divided the computational domain into a Cartesian structured grid with uniform cells. This led to difficulties in adapting the cell to be smaller (e.g., 1/3–1/2) than the local mean free path everywhere within the computational domain, which is the basic requirement for maintaining collisions of good quality. Even though one can still adapt the cell using an octree data structure, similar to those presented by LeBeau [6] and Zhang and Schwartzentruber [20], structuring the data to enable efficient programming in a GPU environment can be challenging. To overcome this, we applied the transient adaptive sub-cell (TAS) algorithm [13,14] to improve collision quality without resorting to complex mesh adaptation. Merit of collision is a measure of collision quality in a DSMC simulation, defined as the ratio of the mean collision spacing (mcs) to the local mean free path (λ). Generally, the merit of collision (mcs/λ) should be less than unity for the DSMC simulation to be considered accurate [12,21].

In the TAS approach, the number of sub-cells for cell i is determined as follows:

$$N_i = \left[\int\left(\frac{\Delta x}{\lambda_i}\right) + 1\right]^2 \tag{5}$$

where int stands for the rounded integer number and Δx represents the size of the cell, which is a constant for all normal fluid cells and the square root of the cell area for cut cells. This design ensures that a greater number of sub-cells exist in regions of higher density, such as shock waves and the stagnation region in hypersonic flow. In the collision routine, collision events proceed cell by cell. For each binary collision event in each sampling cell, the first collision particle is selected randomly from all of the particles in the cell, indicating the sub-cell in which the first particle resides. The second collision particle is then selected by searching through other particles in the same sub-cell. In the event that the first particle is alone in the sub-cell, the nearest neighboring sub-cells are searched sequentially for a second collision particle. The TAS routine ensures that collision partners are closely spaced, even though the sampling cell is larger than the local mean free path. The TAS approach generally produces accurate DSMC solutions if the spatial resolution of the sample cells is high enough to resolve a change in the properties of macroscopic flow in the spatial domain.

3. Results and discussion

We compared the parallel performance of single-GPU DSMC code with that of single-CPU DSMC code by performing a subsonic lid-driven cavity flow problem. We then validated the cut-cell implementation on a single GPU using the VTS and the TAS algorithms to run two benchmarking test cases, including 2D hypersonic nitrogen flow ($M_\infty = 24.8$) over a compression ramp [22–24] and 2D hypersonic argon flow ($M_\infty = 10$) past a cylinder [12,25]. The simulated data were compared with previous simulations and experimental data wherever available. Finally, the code was used to simulate the transition of supersonic flow patterns from a regular reflection to a Mach reflection [26,27] within a channel via a slight change in ramp angle.

3.1. Speedup tests

This study performed simulations of subsonic ($M_\infty = 0.2$) argon lid-driven cavity flow [7] for speedup tests using a variety of GPU cards, which were also used to demonstrate the portability of the developed GPU code. We chose to resolve this problem without the use of the cut-cell approach mainly because the computational overhead resulting from the addition of cut cells is minimal (<1%), compared to that of a pure Cartesian grid. This is mainly due to the fact that in the test cases outlined later in the paper, the fraction of cut cells to normal fluid cells was generally less than 0.1%, and hence can be disregarded. Table 1 summarizes the speedup tests in two cases ($\mathrm{Kn}_\infty = 0.01$ with 10 million particles and $\mathrm{Kn}_\infty = 0.002$ with 30 million particles) running through 5000 time steps on a variety of GPU cards. The CPU processor used in this study was an Intel Xeon 5670 (2.93 GHz, 12 M Cache, 6.4 GT/s Intel QPI) with 48 GB RDIMM RAM. The tested GPU cards included a gaming card (Nvidia GTX-590 with 512 cores, 1.5 GB Global memory and a memory bandwidth of 327.7 GB/s), and two professional computing cards (Nvidia M2070 with 448 cores, 6 GB Global memory and a memory bandwidth of 144 GB/s and Nvidia K20 with 2496 cores, 5 GB Global memory and a memory bandwidth of 208 GB/s). Note that there is no test data for the case of $\mathrm{Kn}_\infty = 0.02$ using the GTX-590 because it ran out of memory. The results show that speedup ranged from 15.4× (M2070) to 31.6× (K20) for the large problem ($\mathrm{Kn}_\infty = 0.002$) and from 13.0× (M2070) to 29.7× (K20) for the small problem (Kn = 0.01). Based on a detailed breakdown of timing for various components of the DSMC code [7], we have determined that the bottleneck was the collision module and perhaps also the indexing module. Thus, the far higher speedup (45.75×) achieved in the indexing module using the K20 card deserves further investigation. For other modules such as moving and sampling, speedup could be as much as 40×–50× for the K20 card. Overall, if the size of the problem is sufficiently large, then a speedup of 30× can be easily achieved using the most advanced GPU cards, such as the Nvidia K20.

3.2. Validation 1: 2D hypersonic flow of nitrogen over a compression ramp

Fig. 3 presents a schematic diagram of the 2D hypersonic flow of nitrogen over a compression ramp. Distance L_1 between the leading edge of the flat plate and the corner of the ramp was 71.4 mm. Length L_2 was also 71.4 mm and the ramp angle was 35°. Nitrogen gas was considered under the following free-stream conditions: density (ρ_∞) of 1.401×10^{-4} kg/m^3, velocity (V_∞) of 1521 m/s, and temperature (T_∞) of 9.06 K. Based on L_1, the corresponding free-stream Mach number (M_∞), Knudsen number (Kn_∞), and Reynolds number (Re_∞) were 24.8, 0.0025, and 12,020, respectively. The wall was perfectly diffusive and the temperature of the wall was 403.2 K. The time step size (5×10^{-8} s in the free stream) in each cell was adjusted spatially based on the VTS algorithm, in addition to the TAS scheme. The total number of cells was 204,800 and the total number of particles was ~4,000,000. The fraction of cut cells to total cells was less than 0.1% (978/204,800) in this case. A typical simulation using 50,000 time steps required approximately 1.5 h running on a single GPU (Nvidia M2070).

Fig. 4 shows the spatial distributions of density (along with streamlines), temperature, pressure, and Mach number. The results show that an oblique shock generated from the leading edge impinged in the mid-section of the ramp surface and is reflected upward to the end of the ramp surface. These results also indicate that, at this low Knudsen number, a clear recirculation bubble formed near the turning corner between the flat plate and the ramp. The simulated maximum density at the impinging point of the oblique shock was 14.5× that of the free-stream value, which is in good agreement with the predictions of Moss et al. [22]. In addition, Fig. 5 compares the corresponding distributions of the merit of collision (MOC) with and without the inclusion of the

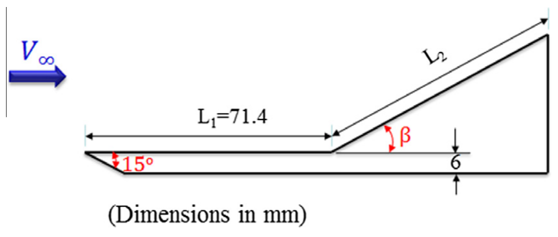


Fig. 3. Schematic diagram of 2D hypersonic flow of nitrogen over compression ramp.

Table 1
Speed analysis results compared to Intel Xeon X5670 CPU in two different cases ($\mathrm{Kn}_\infty = 0.01$ with 10 million particles and $\mathrm{Kn}_\infty = 0.002$ with 30 million particles) running the subsonic lid-driven cavity problem through 5000 iterations on various GPU cards.

		Total		Moving		Indexing		Collision		Sampling	
		Time (s)	Speedup	Time (s)	Speedup	Time (s)	Speedup	Time (s)	Speedup	Time (s)	Speedup
Case 1 (Kn = 0.01)	CPU	14174.5	1.00	5066.0	1.00	2169.4	1.00	4151.4	–	2780.4	1.00
	GPU-M2070	1088.7	13.02	266.2	19.03	239.8	9.05	430.5	9.64	145.3	19.13
	GPU-GTX590	736.6	19.24	202.4	25.03	190.7	11.38	250.6	16.57	86.7	32.08
	GPU-K20	477.7	29.67	147.4	34.38	47.4	45.75	207.5	20.01	64.7	42.98
Case2 (Kn = 0.002)	CPU	54359.3	1.00	23693.0	1.00	9392.8	1.00	12788.4	–	8412.0	1.00
	GPU-M2070	3520.4	15.44	861.5	27.50	976.6	9.62	1221.1	10.47	433.0	19.43
	GPU-K20	1720.7	31.59	471.9	50.20	432.3	21.73	583.1	21.93	187.8	44.79

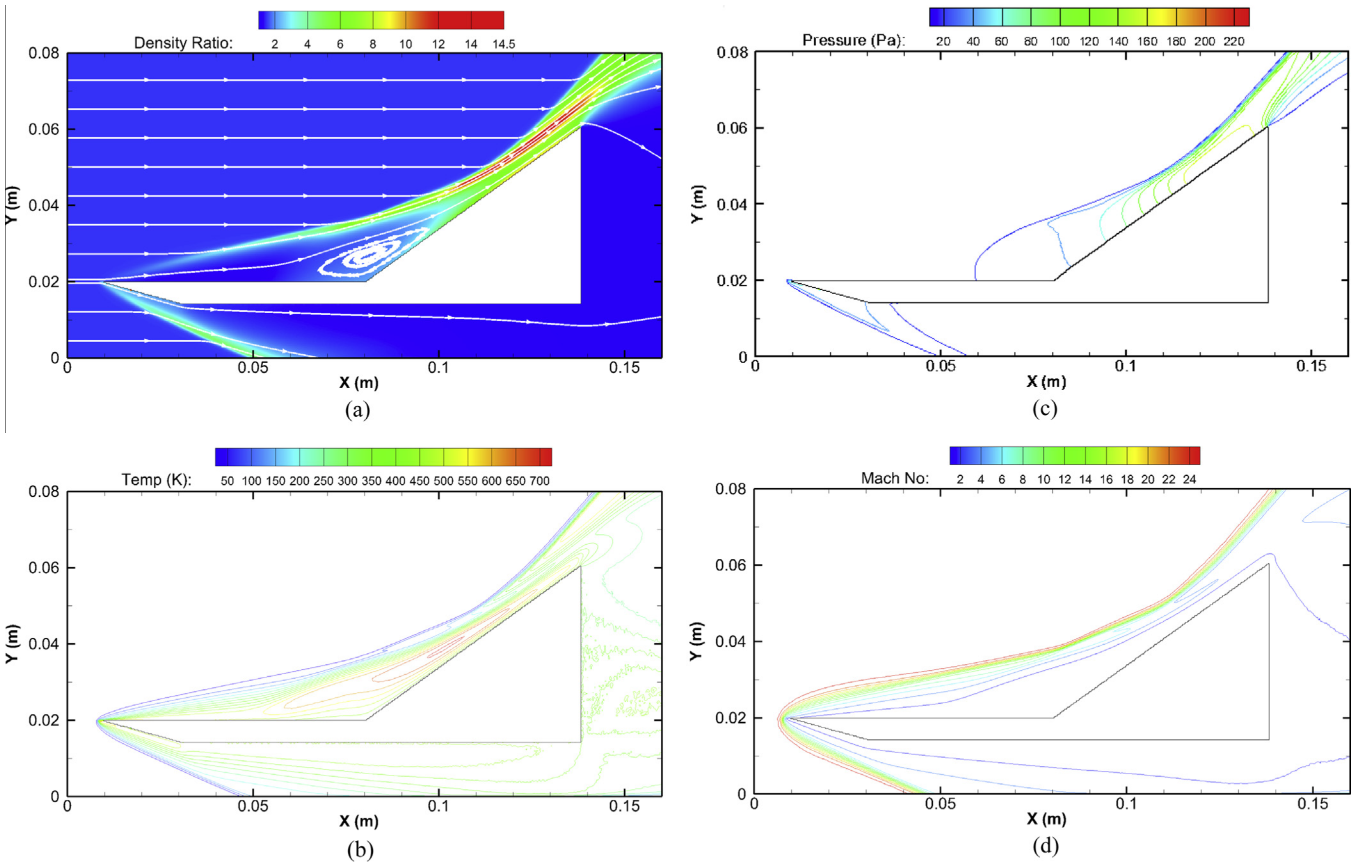


Fig. 4. Distributions of (a) density (along with streamlines), (b) temperature, (c) pressure and (d) Mach number (Kn_∞ = 0.0025, L_2 = 71.4 mm).

TAS and VTS algorithms. These results show that the MOCs were less than unity across the entire computational domain when the VTS and TAS algorithms were used.

Fig. 6 illustrates the distributions of pressure, shear stress, and heat transfer coefficients along the solid wall (L_1 = 71.4 mm, L_2 = 31.55 mm) at three different ramp angles (15°, 25° and 35°). Current flow conditions are briefly described as follows: VHS nitrogen gas, Mach number M_∞ = 22.8, density $\rho_\infty = 5.14 \times 10^{-5}$ kg/m^3, temperature T_∞ = 8.3 K, and a fully thermally accommodated and diffusive wall with T_w = 340 K. Based on L_1, the resulting Knudsen number Kn_∞ and Reynolds number Re_∞ were 0.0066 and 3,938.66, respectively. Note that the pressure and shear stress coefficients were normalized to the free-stream dynamic pressure, and the heat transfer coefficient was normalized to the free-stream dynamic heat flux. The results show that all of the simulated coefficients at the three angles used in this study were in excellent agreement with the simulation data predicted by Wu and Tseng [23] and Moss et al. [24]. Details of the physics were discussed in previous studies [23] and are therefore skipped here for brevity. One interesting point is that the shear stress coefficient becomes very small near the corner due to the formation of recirculation bubbles in this region.

These results demonstrate the efficacy of parallel DSMC using the cut-cell approach with the VTS and TAS algorithms on a single GPU or the problem with inclined straight solid boundaries.

3.3. Validation 2: 2D hypersonic flow of argon past a circular cylinder

Fig. 7 presents the second benchmarking test case of a sketch of 2D hypersonic flow past a cylinder under rarefied conditions. Note that the azimuthal angle began at the front stagnation point and ended at the rear stagnation point along the surface of the cylinder. The test conditions included: argon gas with a free-stream Mach number of 10, a free-stream temperature of 200 K, a cylinder diameter of 0.3048 m, a fixed cylinder wall temperature of 500 K, and a free-stream Knudsen number of 0.01, based on the diameter of the cylinder. The free-stream number density and velocity were 4.247×10^{20} particles/m^3 and 2,634.1 m/s, respectively. This case was first simulated by Lofthouse et al. [25] as a comparison between a Navier–Stokes equation solver and the MONACO DSMC code. The present study included a total of 48,000 cells and ~2,000,000 particles. The time step size, 5×10^{-7} s, was set to 1/2 of the mean collision time in the free stream for the cases with and without VTS algorithm. A typical simulation of 50,000 time steps required approximately two hours on a single GPU (Nvidia M2070). In the following, we compare the simulation results with and without the use of the VTS and the TAS algorithms.

Fig. 8 presents the distributions of simulated flow properties, including density, Mach number (along with streamlines), temperature, and pressure for the case of Kn_∞ = 0.01 using the VTS and TAS algorithms. Our results are in excellent agreement with those presented by Lofthouse et al. [25]. Bird [12] compared the results for drag across a cylinder and peak surface heat transfer flux for his own DS2 V program with other DSMC codes, including MONACO, SMILE, and DAC as well as the obsolete DS2G code. Table 2 summarizes the predicted total drag and peak heat transfer flux obtained using various DSMC codes, including the present single-GPU DSMC code. Our simulation results are in excellent agreement with those of other coding methods when the VTS and the TAS algorithms are employed. As expected, the results are seriously degraded when they are not used. It demonstrates that using TAS and VTS allows one to obtain accurate DSMC results on meshes

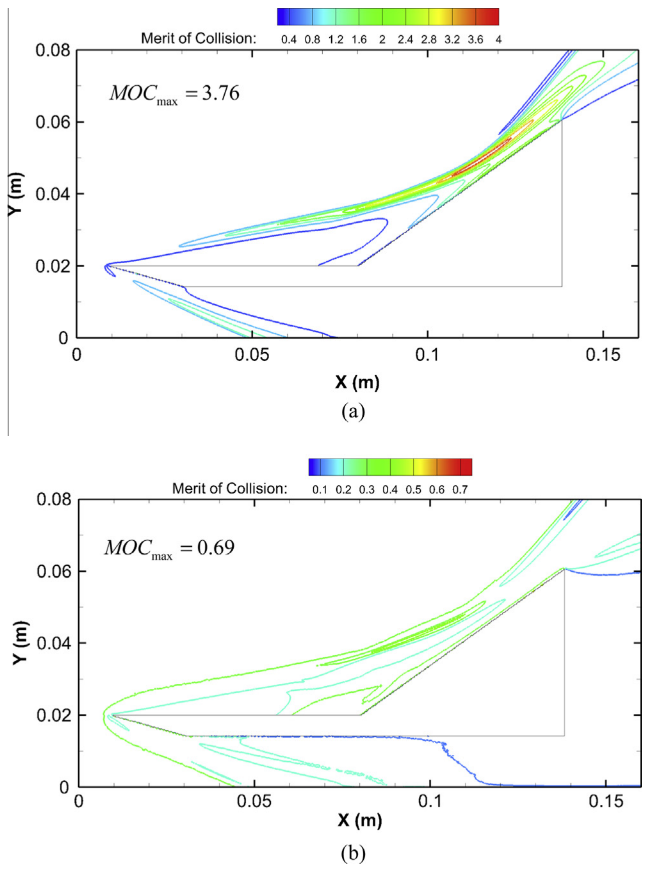


Fig. 5. Comparison of corresponding distributions of the merit of collision: (a) without and (b) with the use of the TAS and VTS algorithms ($Kn_\infty = 0.0025$, $L_2 = 71.4$ mm).

that are not refined to level of the local mean free path of the flow, and with potentially shorter transient times than the standard DSMC method.

Fig. 9 presents the distribution of predicted surface properties (pressure, shear stress, and heat transfer coefficients) as functions of azimuthal angle with various numbers of cells (48,000–3,072,000) without the use of VTS and the TAS algorithms. We also plotted data from Lofthouse et al. [25] using 34,770 unstructured cells for comparison. The number of particles in the simulations was increased proportionally, such that the average number of particles per cell was maintained at approximately 10.

Our results clearly show that the pressure surface coefficients in all the test cases with various numbers of cells are in strong agreement with previous simulations. However, large discrepancies from the previous data were observed in the predicted surface shear stress and heat transfer coefficients [25] in all cases, regardless of the fineness of the grid, particularly with regard to the shear stress coefficient. Table 3 summarizes the corresponding ratios of cell size to free-stream local mean free path, computational time, and maximal merit of collision in the computational domain for various numbers of cells. As can be seen, runtime increased from 1449 s (48,000 cells) to 93,824 s (3,072,000 cells), while the maximal merit of collision decreased from 11.78 (48,000 cells) to 2.62 (3,072,000 cells). The fact that the data based on three million cells was superior to that obtained using 48,000 cells can be attributed to far better collision quality in the former case, even though the former are far from being accurate. Fig. 10 presents the corresponding spatial distribution of merit of collision for cases of 48,000 and 3,072,000

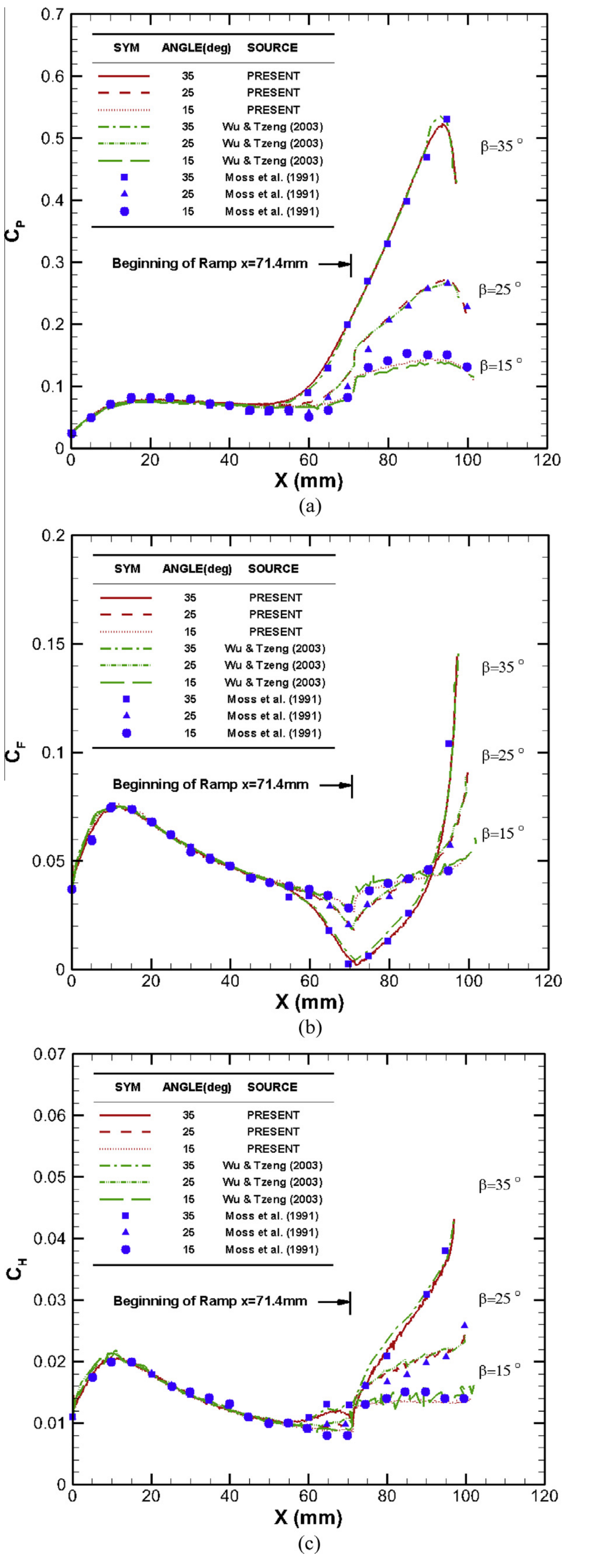


Fig. 6. Distributions of surface properties: (a) pressure coefficient; (b) shear stress coefficient; and (c) heat transfer coefficient as functions of distance along the ramp at three different ramp angles ($Kn_\infty = 0.0066$, $L_2 = 31.55$ mm).

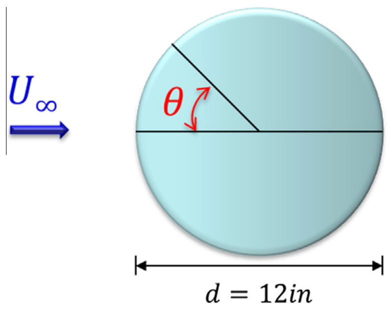


Fig. 7. Sketch of Mach 10 hypersonic flow over a cylinder.

cells. Clearly, the use of a more refined grid did improve the solution at the price of a severe drop in efficiency. Note the computational time of the case using 3,072,000 cells is approximately 60 times longer than that of the case using 48,000 cells.

Table 2
Comparisons of drag and peak heat flux on the surface of cylinder for M-10 hypersonic flow over a cylinder, as predicted using various DSMC codes.

	Drag (N)	Peak heat flux (W/m^2)
Present (standard)	41.83	65,162
Present (VTS + TAS)	39.89	38,657
DS2 V	39.76	38,400
DAC	39.71	38,500
SMILE	39.76	39,000
MONACO [24]	40.00	39,319

Fig. 11 presents the predicted surface properties (pressure, shear stress, and heat transfer coefficients) as functions of azimuthal angle using the standard approach and the VTS and VTS–TAS algorithms. As expected, all of the data obtained using the VTS and TAS algorithms with 48,000 cells are in good agreement with the results presented by Lofthouse et al. [25] with 34,770 unstructured cells. In fact, these results and are even better

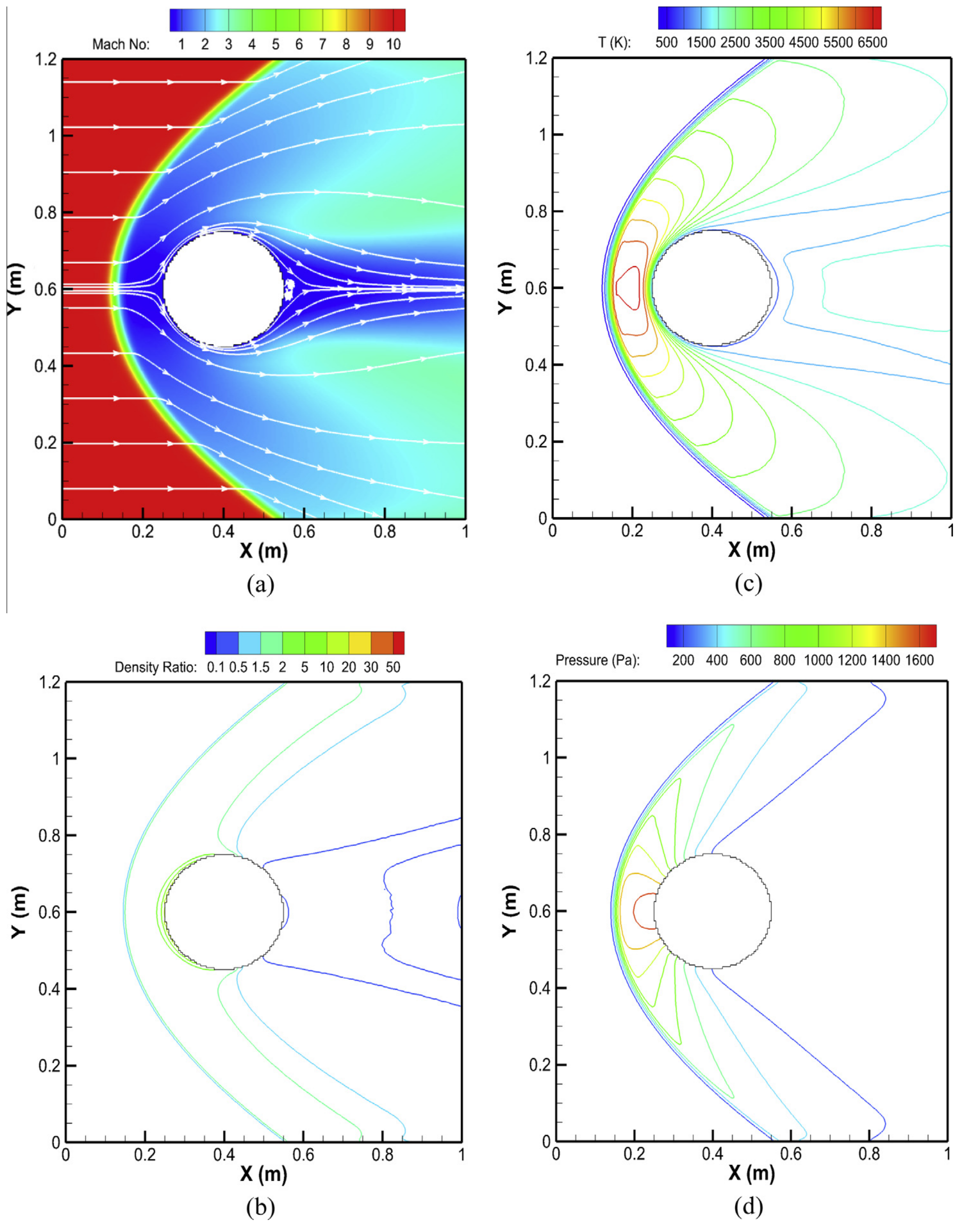
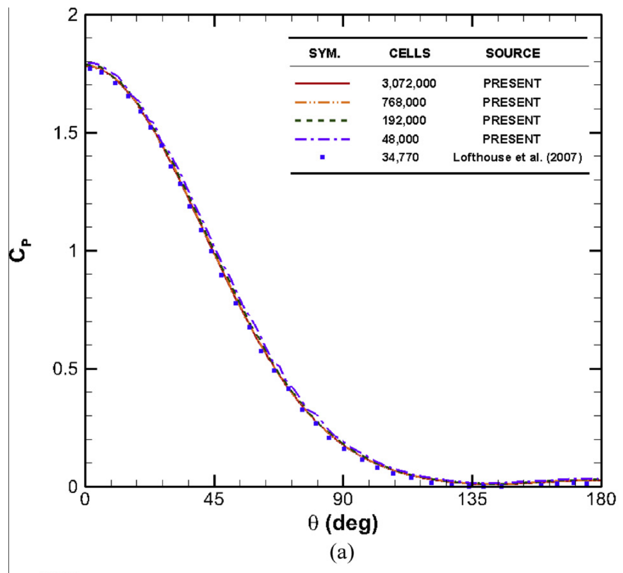
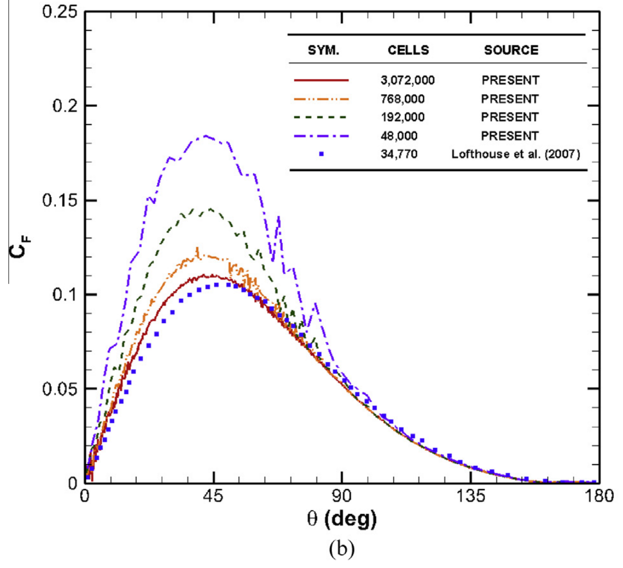


Fig. 8. Distributions of flow properties for hypersonic flow of argon past a cylinder with Kn$_\infty$ = 0.01: (a) Mach number (along streamlines); (b) density; (c) temperature and (d) pressure.





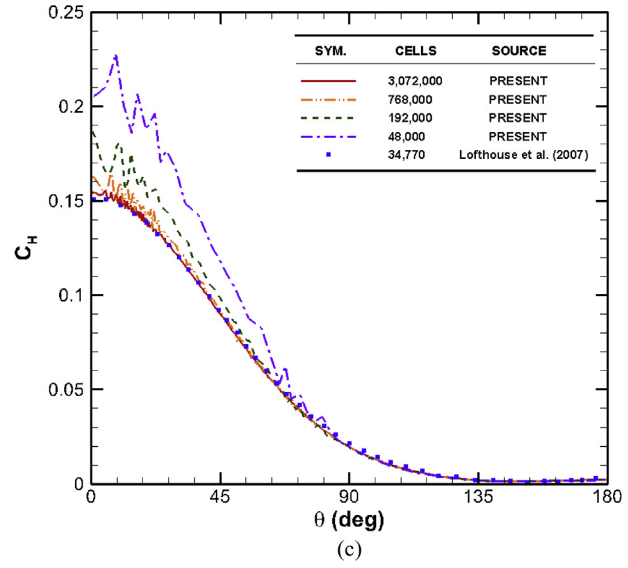
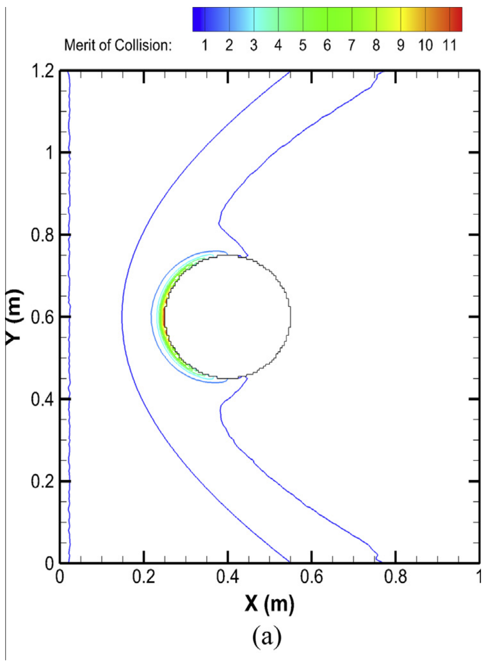


Fig. 9. Distributions of surface properties: (a) pressure coefficient; (b) shear stress coefficient and (c) heat transfer coefficient as functions of angle around the cylinder for flow past a cylinder using various simulation cells ($Kn_\infty = 0.01$).

Table 3
Comparisons of corresponding runtimes and maximum merits of collision for flow past a cylinder with various numbers of cells without the use of VTS or TAS algorithms (Kn_∞: 0.01, Initial particle number per cell: 10, total time steps: 50,000, sample time steps: 20,000).

Total cells	48,000	192,000	768,000	3,072,000
$\Delta x/\lambda_\infty$[a]	1.64	0.82	0.41	0.21
Simulation time (s)	1449	4908	21,616	93,824
$(MOC)_{max}$[b]	11.78	7.8	4.33	2.62

[a] Δx: cell size, λ_∞: free stream mean free path.
[b] $(MOC)_{max}$: maximum merit of collision.



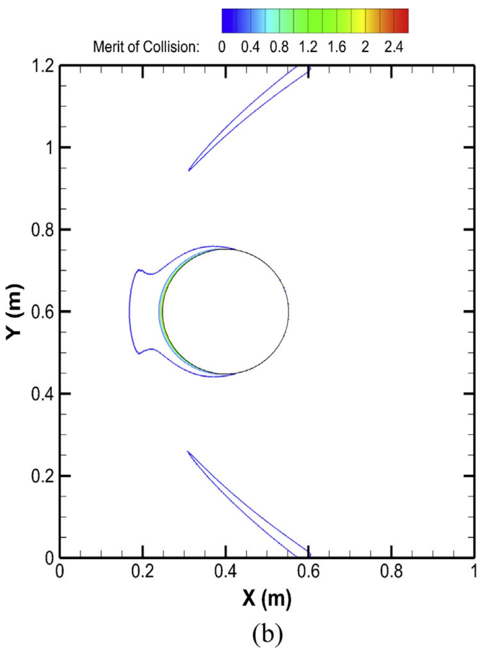
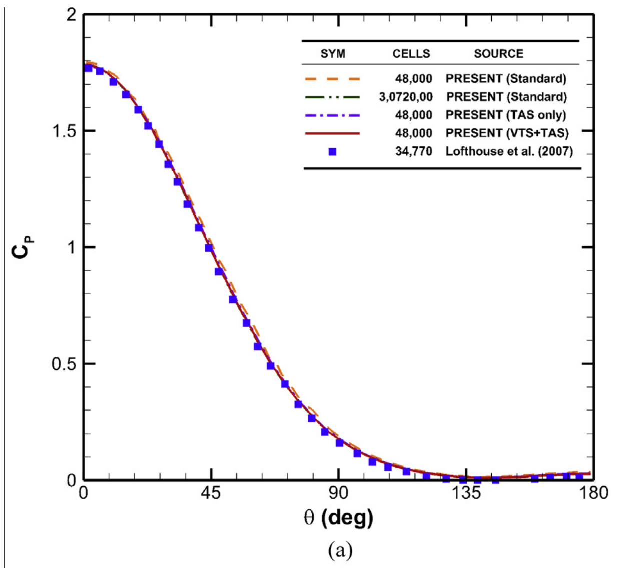
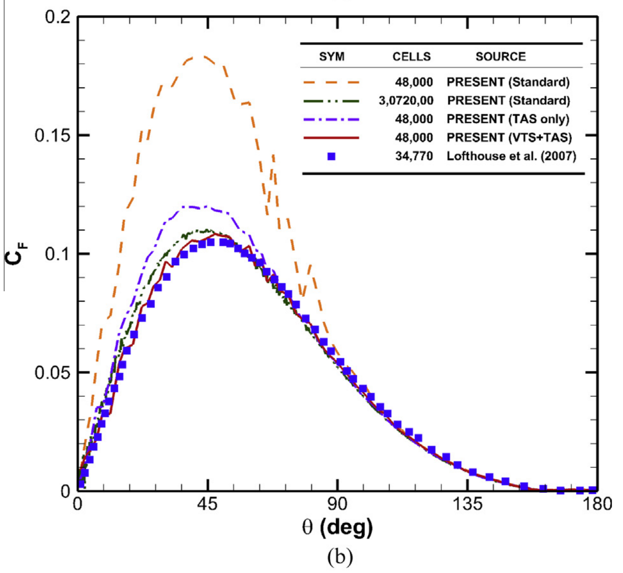


Fig. 10. Distribution of MOC for cases of (a) 48,000 and (b) 3,072,000 cells.

than those obtained without the use of the VTS and the TAS algorithms with more than three million cells. It should also be noted that the sole use of TAS improved the DSMC solution. However,





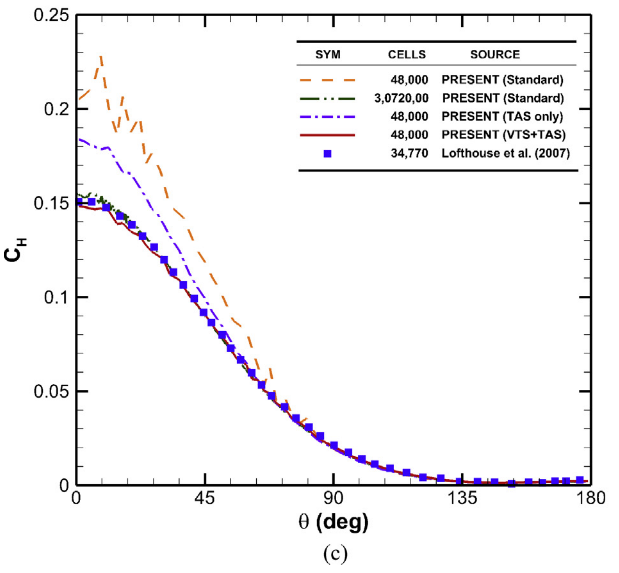


Fig. 11. Distributions of surface properties: (a) pressure coefficient; (b) shear stress coefficient and (c) heat transfer coefficient as functions of angle around the cylinder for flow past a cylinder with or without the use of TAS and VTS ($Kn_\infty = 0.01$).

Table 4
Comparisons of corresponding runtimes and maximum merits of collision for the flow past a cylinder with or without TAS and VTS (Kn_∞: 0.01, Particles per cell: 10, total time-step: 50,000, sample times: 20,000).

Approach	Standard	Standard	TAS only	TAS + VTS
Number of cells	48,000	3,0720,00	48,000	48,000
Simulation time (s)	1449	93,824	2080	6793
$(MOC)_{max}$[a]	11.78	2.62	1.68	1.02

[a] $(MOC)_{max}$: maximum merit of collision.

this did not occur to the degree achieved using the VTS and the TAS algorithms simultaneously, mainly because the time-steps in those small cut cells were too large and conflicted with the basic rules of DSMC. Conversely, when using the VTS algorithm in these cut cells; the time-steps were reduced to suit DSMC requirements. Table 4 summarizes the corresponding runtimes and maximal merits of collision presented in Fig. 11. The results show that the maximal merit of collision decreased dramatically from 11.78 (standard) to 1.02 (VTS + TAS) in the case of 48,000 cells, which explains the highly accurate surface data presented in Fig. 11. Nonetheless, this resulted in a penalty in computational time from 1449 s to 6793 s, representing an increase of approximately 4.5×. This time penalty was due primarily to the use of much smaller time-steps in the cut cells, which extended the time required for the simulation to reach a steady state. The case of three million cells without the use of the VTS or TAS algorithms was inferior (although close) to the case using only 48,000 cells; nonetheless, the time saving can be as great as 14×.

3.4. Application: transition from a regular reflection to a Mach reflection in the supersonic flow of nitrogen through a channel

To demonstrate the high fidelity of the developed parallel DSMC code, we have performed simulations for modeling the transition from a regular reflection to a Mach reflection of supersonic flow in a channel. Fig. 12 presents a schematic diagram of 2D supersonic flow past a pair of symmetric ramps attached to the channel walls. The test conditions include the following: nitrogen gas (specific heat ratio $\gamma = 1.4$) with an incoming Mach number of 4.96, a temperature of 365 K, a number density of 1.388×10^{21} particles/m^3, a ramp angle of θ, and a Knudsen number based on the free-stream conditions and the dimensions of the wedge ($L = w = 0.1$ m). The VHS model was used for collision kinetics and the wall presented a perfectly specular reflection. The total number of cells was 320,000 and the total number of particles was ~4,000,000. A typical simulation using 50,000 time steps required approximately 7.4 h running on a single GPU (Nvidia M2070).

Figs. 13 and 14 present the spatial distributions of the major flow properties incurred by changing the ramp angle from 27.5° and 27.85°. These results illustrate a clear transition from a regular

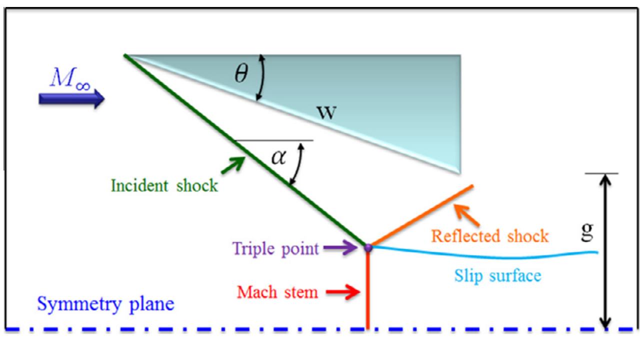


Fig. 12. Schematic diagram of 2D supersonic flow past a pair of symmetric ramps attached to the channel walls.

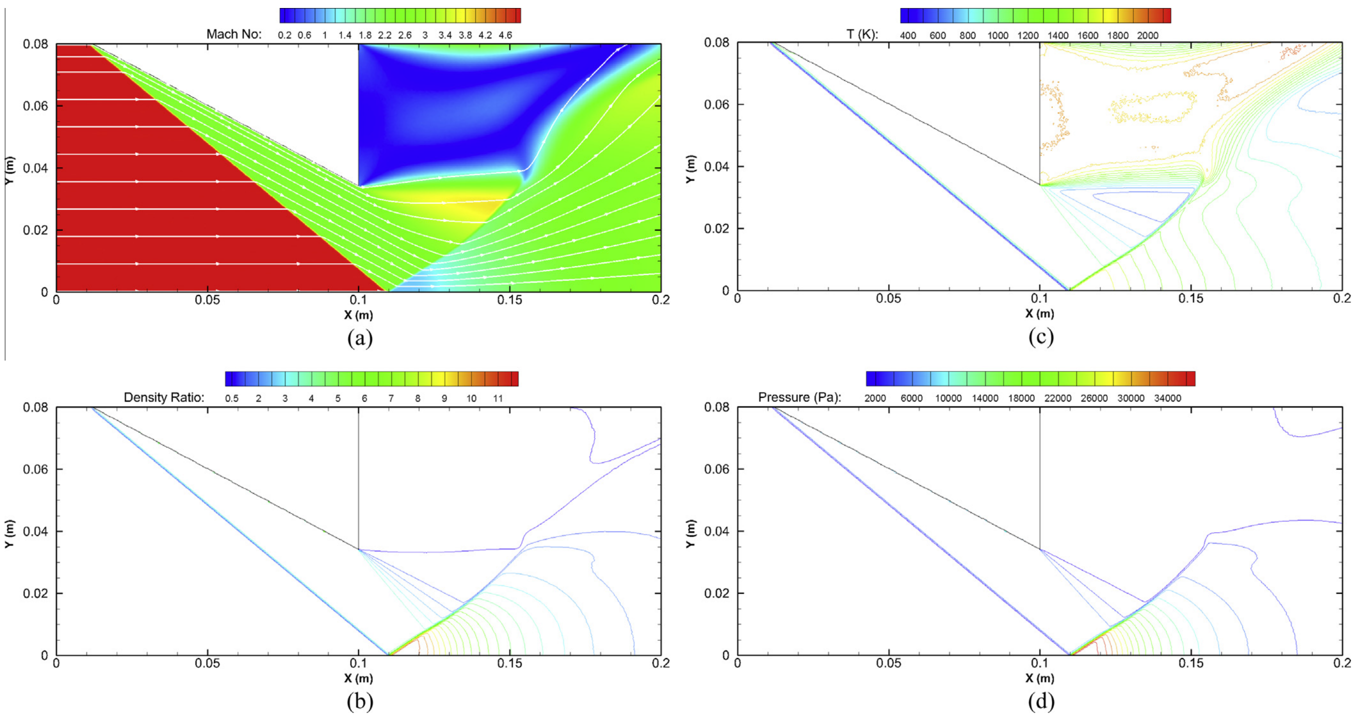


Fig. 13. Distributions of flow properties of supersonic flow past a pair of symmetric ramps attached to the channel walls at a ramp angle of 27.5°: (a) Mach number (along streamlines); (b) density; (c) temperature and (d) pressure.

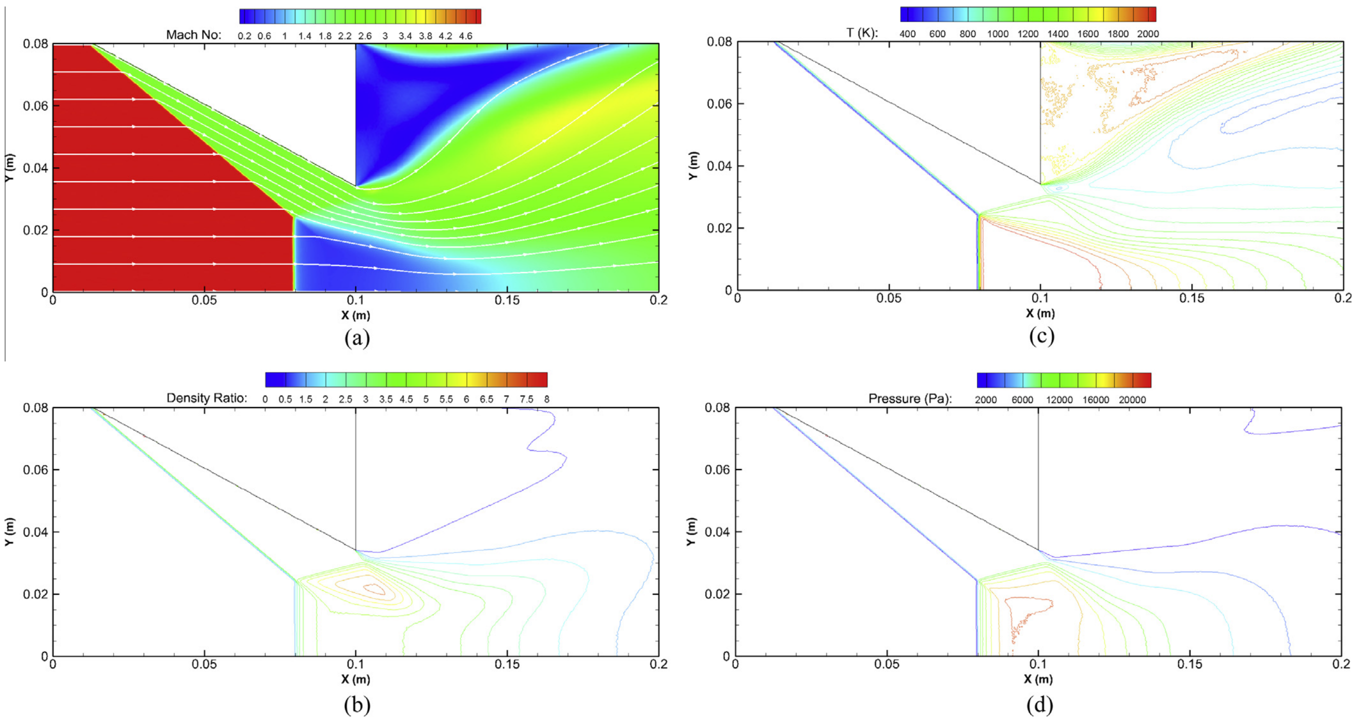


Fig. 14. Distributions of flow properties for supersonic flow past a pair of symmetric ramps attached to the channel walls at a ramp angle of 27.85°: (a) Mach number (along streamlines); (b) density; (c) temperature and (d) pressure.

reflection of impinging oblique shock originating from the corner of the ramp at the symmetric line with a ramp angle of 27.5° to a distinct Mach reflection pattern with a ramp angle of 27.85°. These results coincide with previous simulations and experiments [26,27]. In addition, the calculated Mach stem height was 0.024 m, which is in excellent agreement with 0.0235 m predicted by Ivanov et al. [26,27]. These simulations clearly demonstrate that the proposed parallel DSMC code using the cut-cell approach with the VTS and the TAS algorithms on a single GPU is capable of reproducing challenging fluid dynamics.

4. Conclusion

This study proposed and validated a cut-cell approach to the treatment of geometrically complex objects in a Cartesian-grid DSMC using the variable time-step (VTS) and transient adaptive subcell (TAS) algorithms on a single GPU. Simulation results demonstrate that the proposed parallel single-GPU DSMC code is capable of reproducing previous simulations using both the VTS and TAS algorithms. An accurate transition from regular reflection to Mach reflection in the supersonic flow of nitrogen within a channel was also predicted with high-fidelity using the single-GPU DSMC code. Simple speedup tests demonstrated – 30$\times$ speedup on the most advanced GPU card in cases of ten million particles or more. Efforts to extend the DSMC code to run on multiple GPUs, which is similar to the work by Su et al. [7], is currently underway and will be reported in the near future.

Acknowledgements

We would like to express our appreciation for the computing resources provided by the National Center for High-Performance Computing of Taiwan. The authors would also like to acknowledge the support obtained from the National Science Council (NSC) of Taiwan through Grant NSC-99-2221-E-009-056-MY2 and NSC-100-2627-E-009-001, and the National Space Organization (NSPO) of Taiwan through Grant NSPO-S-099128.

Appendix A. Crossing number test for a point in a polygon

```
// cn_PnPoly(): crossing number test for a point in a polygon
// Input:   P = test point
//      V[] = vertex points of a polygon V[n + 1] with V[n] = V[0]
// Return:  0 = outside, 1 = inside

int cn_PnPoly(Point P, Point* V, int n)
{
  int  cn = 0;  // the crossing number counter

// loop through all edges of the polygon
  for (int i = 0; i < n; i++) {  // edge from V[i] to V[i + 1]
    if (((V[i]·y <= P·y) && (V[i + 1]·y > P·y))  // an upward
  crossing
    || ((V[i]·y > P·y) && (V[i + 1]·y <= P·y))){  //a downward
  crossing
    // compute the actual edge-ray intersect x-coordinate
    float vt = (float)(P·y − V[i]·y)/(V[i + 1]·y − V[i]·y);
    float ixc = V[i]·x + vt* (V[i + 1]·x − V[i]·x);       // intersect
  x-coordinate
      if (P·x < ixc)
      cn = !cn;
    }
  }
  return cn;  // 0 if even (out), and 1 if odd (in)
}
```

References

[1] Bird GA. Molecular gas dynamics and the direct simulation of gas flows. Oxford Univ Press; 1994.
[2] Nanbu K. Direct simulation scheme derived from the Boltzmann equation I: mono component gases. J Phys Soc Jpn 1980;49(5):2042–9.
[3] Wagner W. A convergence proof for bird's direct simulation Monte Carlo method for the Boltzmann equation. J Stat Phys 1992;66:1011–44.
[4] Dietrich S, Boyd ID. Scalar and parallel optimized implementation of the direct simulation Monte Carlo method. J Comput Phys 1996;126:328–42.
[5] Ivanov M, Markelov G, Taylor S, Watts J. Parallel DSMC strategies for 3D computations. Proc Parallel CFD 1997;96:485–92.
[6] LeBeau GJ. A parallel implementation of the direct simulation Monte Carlo method. Comput Methods Appl Mech Eng 1999;174:319–37.
[7] Su C-C, Smith MR, Wu J-S, Hsieh C-W, Tseng K-C, Kuo F-A. Large-scale simulations on multiple graphics processing units (GPUs) for the direct simulation Monte Carlo method. J Comput Phys 2012;231:7932–58.
[8] Su C-C, Hsieh C-W, Smith MR, Jermy MC, Wu J-S. Parallel direct simulation Monte Carlo computation using CUDA on GPUs. In: Proceedings of the 27th international symposium on rarefied gas dynamics, Pacific Grove, California, USA; July 2010.
[9] Gladkov D, Tapia JJ, Alberts S, D'Souza RM. Graphics processing unit based direct simulation Monte Carlo. Simulation 2012;88(6):680–93.
[10] Wu J-S, Lian Y-Y. Parallel three-dimensional direct simulation Monte Carlo method and its applications. Comput Fluids 2003;32:1133–60.
[11] Boyd ID. Vectorization of a Monte Carlo simulation scheme for nonequilibrium gas dynamics. J Comput Phys 1991;96(2):411–27.
[12] Bird GA. Sophisticated versus simple DSMC. In: Proceedings of the 25th international symposium on rarefied gas dynamics (RGD25); 2006.
[13] Bird GA. The DS2V/3V program suite for DSMC calculations. In: Capitelli M, editor. Rarefied gas dynamics: proceedings of the 24th international symposium on rarefied gas dynamics, AIP conference proceedings, vol. 762. Melville, NY; 2005. p. 541–6.
[14] Su C-C, Tseng K-C, Cave HM, Wu J-S, Lian Y-Y, Kuo T-C, et al. Implementation of a transient adaptive sub-cell module for the parallel-DSMC code using unstructured grids. Comput Fluids 2010;39:1136–45.
[15] Kannenberg KC, Boyd ID. Strategies for efficient particle resolution in the direct simulation Monte Carlo method. J Comput Phys 2000;157:727–45.
[16] Wu J-S, Tseng K-C, Wu F-Y. Parallel three-dimensional DSMC method using mesh refinement and variable time-step scheme. J Comput Phys Commun 2004;162:166–87.
[17] NVIDIA corp. NVIDIA compute unified device architecture programming guide version 1.0; 2007.
[18] NVIDIA crop. NVIDIA CUDA C programming guide version 4.0; 2011.
[19] Eric H. Point in polygon strategies. In: Heckbert Paul, editor. Graphics gems IV. Academic Press; 1994. p. 24–46.
[20] Zhang C-L, Schwartzentruber TE. Robust cut-cell algorithms for DSMC implementations employing multi-level Cartesian grids. Comput Fluids 2012;69:122–35.
[21] Gallis MA, Torczynski JR, Rader DJ, Bird GA. Accuracy and convergence of a new DSMC algorithm. AIAA J 2008:2008–3913.
[22] Moss JN, Rault DF, Price JM. Direct Monte Carlo simulations of hypersonic viscous interactions including separation. In: Shizgal BD, Weaver DP, editors. Rarefied gas dynamics: space science and engineering; 1993. p. 209–20.
[23] Wu J-S, Tseng K-C. Parallel particle simulation of the near-continuum hypersonic flows over compression ramps. ASME J Fluids Eng 2003;125(1):181–8.
[24] Moss JN, Price JM, Chun CH. Hypersonic rarefied flow about a compression corner—DSMC simulation and experiment. In: 26th thermophysics conference, AIAA paper, no 91-1313; 1991.
[25] Lofthouse AJ, Boyd ID, Wright JM. Effects of continuum breakdown on hypersonic aerothermodynamics. Phys Fluids 2007;19:027105.
[26] Ivanov MS, Markelov GN, Kudryavtsev AN, Gimelshein SF. Numerical analysis of shock wave reflection transition in steady flows. AIAA J 1998;36(11):2079–86.
[27] Ivanov MS, Kudryavtsev AN, Nikiforov SB, Khotyanovsky DV, Pavlov AA. Experiments on shock wave reflection transition and hysteresis in low-noise wind tunnel. Phys Fluids 2003;15(6):1807–10.