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.