Solving the Boltzmann equation on GPU's

Aldo FREZZOTTI[a], Gian Pietro GHIROLDI[a], Livio GIBELLI[*,a]

[a] *Politecnico di Milano, Dipartimento di Matematica, Piazza Leonardo da Vinci, 20133 Milano, Italy*

Abstract

We present algorithms specifically tailored for solving kinetic equations onto graphics processing units. Unlike particle methods, the proposed methods of solution are ideally suited for solving the unsteady low speed flows which typically occur in MEMS containing oscillating components. The efficiency of the algorithms is demonstrated by solving the two-dimensional low Mach number driven cavity flow of a monatomic gas. Computational results show that it is possible to cut down the computing time of the sequential codes up to two order of magnitudes. The algorithms can easily be extended to three-dimensional flows and to non-equilibrium flows of mixtures.

Key words: Gas micro flows, Boltzmann equation, BGKW equation, semi-regular method, regular method

1. Introduction

The correct description of rarefaction effects in gas micro flows in MEMS or microchannels often requires replacing the traditional hydrodynamic equations with the Boltzmann equation or model kinetic equations [1]. In many cases, the flow Mach number is small and one can linearize the kinetic equation. Numerical solutions of linearized kinetic model equations have been presented in several studies of micro flows [2, 3] and successfully compared with experimental data [4]. Applications of the full Boltzmann equation to micro flows gas have been limited by its greater complexity and the difficulty of extending the efficiency of DSMC schemes [5] to "slow" flows. Attempts have been made to extend DSMC in order to improve its capability to capture the small deviations from the equilibrium condition met in low Mach number flows [6, 7, 8]. However, in simulating high frequency unsteady flows, typical of microfluidics application to MEMS [9], the possibility of time averaging is lost or reduced. Acceptable accuracy can then be achieved by increasing the number of simulation particles or superposing several flow snapshots obtained from statistically independent simulations of the same flow; in both cases the computing effort is considerably increased.

In the present work, we solve the kinetic equations by regular and semi-regular methods. Such methods combine a finite difference discretization of the free streaming term with either a deterministic or a Monte Carlo evaluation of the collision integral [10, 11]. Unlike particle methods, they are not only better suited to solve unsteady flows, but can be also more easily translated into a parallel computer code to be executed on a Graphic Processing Unit (GPU) [12]. GPUs have been used to accelerate CPU critical applications such as simulations of hypersonic flows [13], magnetized plasma [14] and molecular dynamics [15]. However, so far few applications to kinetic theory of gases have been reported [16]. The aim of the paper is to describe efficient algorithms specifically tailored for solving kinetic equations onto GPUs using CUDA™ programming language [12]. The full non-linear form of the kinetic equations is used to explore the limits of the linearized approach. The efficiency of the algorithms is assessed by solving the low speed two-dimensional driven cavity flow since, in spite of its simple geometry, it contains most of the features which appear in more complicated problems described by kinetic equations. It is shown that it is possible to cut down the computing time of the sequential codes up to two order of magnitudes by a proper reformulation of the algorithm to be executed on a GPU. The rest of the paper is organized as follows. Section 2 is devoted to a concise description of the mathematical formulation of the two-dimensional driven cavity flow problem. In Section 3, a short outline of the semi-regular and regular methods of solution of the kinetic equations is given. In

[*] Corresponding author

Email address: `livio.gibelli@polimi.it` (Livio GIBELLI)

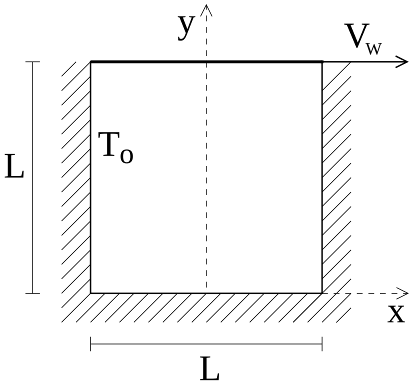

Figure 1: Flow configuration.

Section 4 the key aspects of the GPU hardware architecture and CUDA™ programming language are briefly described and some details of the implementation of the numerical methods are given. Results are discussed in Sections 5 and concluding remarks are presented in Section 6.

2. Mathematical formulation

A monatomic gas is confined in a two-dimensional square cavity with edge L. The flow is driven by a uniform translation of the top with velocity velocity $\mathbf{V}_{\rm w}$. Figure 1 illustrates the flow geometry. The gas flow is governed by the two-dimensional Boltzmann equation [19]

$$\frac{\partial f}{\partial t} + \mathbf{v} \cdot \nabla_{\mathbf{x}} f = C(f, f) \tag{1}$$

In Eq. (1), $f(\mathbf{x}, \mathbf{v}|t)$ denotes the distribution function of atomic velocity $\mathbf{v}$ at spatial location $\mathbf{x}$ and time t, whereas $C(f, f)$ gives the collisional rate of change of f at the phase space point $(\mathbf{x}, \mathbf{v})$ at time t. It is assumed that all the walls are isothermal with temperature T_0 and that the particles which strike the wall at $\mathbf{x}_{\rm w}$ are re-emitted at the same space location according to the Maxwell's scattering kernel with complete accommodation [19]

$$f(\mathbf{x}_{\rm w}, \mathbf{v}) = \frac{n_{\rm w}}{(2\pi R T_0)^{1/2}} \exp\left[-\frac{(\mathbf{v} - \mathbf{V}_{\rm w})^2}{2RT_0}\right] \tag{2}$$

where $\mathbf{V}_{\rm w}$ is the wall velocity and the wall gas density $n_{\rm w}(\mathbf{x}_{\rm w})$ is set so as to impose zero net flux across the wall

$$n_{\rm w} = \left(\frac{2\pi}{RT_0}\right)^{1/2} \int_{(\mathbf{v}-\mathbf{V}_{\rm W})\cdot\hat{\mathbf{n}}<0} (|\mathbf{v} - \mathbf{V}_{\rm w}| \cdot \hat{\mathbf{n}}) f \, d\mathbf{v} \tag{3}$$

where $\hat{\mathbf{n}}$ is the unit vector normal to the wall. The collision integral, $C(f, f)$, is a non-linear functional of f, whose precise structure depends on the assumed atomic interaction forces. For the dilute hard sphere gas considered here, the collision integral simplifies to

$$\begin{aligned} C(f, f) \quad = \quad & \frac{d^2}{2} \int [f(\mathbf{x}, \mathbf{v}^*|t) f(\mathbf{x}, \mathbf{v}_1^*|t) - \\ & f(\mathbf{x}, \mathbf{v}|t) f(\mathbf{x}, \mathbf{v}_1|t)] \, |\hat{\mathbf{k}} \circ \mathbf{v}_r| d\mathbf{v}_1 d^2\hat{\mathbf{k}} \end{aligned} \tag{4}$$

being d the hard sphere diameter and $\mathbf{v}^*$, $\mathbf{v}_1^*$ the precollisional velocities, obtained from $\mathbf{v}$, $\mathbf{v}_1$ and $\hat{\mathbf{k}}$ by the simple relationships

$$\mathbf{v}^* = \mathbf{v} + (\mathbf{v}_r \circ \hat{\mathbf{k}})\hat{\mathbf{k}}, \quad \mathbf{v}_1^* = \mathbf{v}_1 - (\mathbf{v}_r \circ \hat{\mathbf{k}})\hat{\mathbf{k}} \tag{5}$$

In Eq. (5), $\hat{\mathbf{k}}$ is the unit vector which gives the direction of the relative position of the colliding particles at the time of impact. Both from the theoretical and computational point of view, it is often convenient to use a simpler collision term. In the kinetic model proposed by Bhatnagar, Gross and Krook [17] and, independently, by Welander [18], the collision term takes the form:

$$C(f, f) = \nu(\Phi - f) \tag{6}$$

In Eq. (6) ν is the collision frequency, whereas $\Phi(\mathbf{x}, \mathbf{v}|t)$ is the local equilibrium Maxwellian distribution function given by the expression

$$\Phi = \frac{n}{(2\pi RT)^{3/2}} \exp\left[-\frac{(\mathbf{v} - \mathbf{V})^2}{2RT}\right] \tag{7}$$

If ν does not depend on the velocity $\mathbf{v}$, then conservation of mass, momentum and energy requires that $n(\mathbf{x}|t)$, $\mathbf{V}(\mathbf{x}|t)$ and $T(\mathbf{x}|t)$ in Eq. (7) coincide with the local values of density, bulk velocity and temperature obtained from f by the relationships

$$n(\mathbf{x}|t) \quad = \quad \int f(\mathbf{x}, \mathbf{v}|t) \, d\mathbf{v} \tag{8}$$

$$\mathbf{V}(\mathbf{x}|t) \quad = \quad \frac{1}{n} \int \mathbf{v} f(\mathbf{x}, \mathbf{v}|t) \, d\mathbf{v} \tag{9}$$

$$T(\mathbf{x}|t) \quad = \quad \frac{1}{3Rn} \int (\mathbf{v} - \mathbf{V})^2 f(\mathbf{x}, \mathbf{v}|t) \, d\mathbf{v} \tag{10}$$

being R the specific gas constant. The above expressions show that Eq. (6) is a strongly non-linear integro-differential equation, in spite of the linear appearance of its r.h.s.. As is well known, the BGKW model predicts an incorrect value of the Prandtl number in the hydrodynamic limit [19]. Hence, ν can be adjusted to obtain either the correct viscosity or heat conductivity, but not both. If the viscosity $\mu(T)$ is selected, then ν is obtained as nRT/μ.

3. Outline of the numerical methods

Eq. (1), with the collision integral given by Eq. (4) or Eq. (6), is numerically solved by a semi-regular and regular method, respectively. The spatial domain is a square whose sides have length L; each side is divided into N_s intervals of equal size to form N_s^2 equal square cells. The infinite three-dimensional velocity space is replaced by a rectangular box divided into $N_v = N_{v_x} \times N_{v_y} \times N_{v_z}$ cells of equal volume $\Delta\mathcal{V}$, N_{v_α} being the number of velocity nodes associated with the velocity component v_α. The size and position of the "velocity box" in the velocity space have to be properly chosen, in order to contain the significant part of f at any spatial position. The distribution function is assumed to be constant within each cell of the phase space. Hence, f is represented by the array $f_{\mathbf{i},\mathbf{j}}(t) = f(x(i_x), y(i_y), v_x(j_x), v_y(j_y), v_z(j_z)|t)$; $x(i_x), y(i_y), v_x(j_x), v_y(j_y), v_z(j_z)$ are the values of the spatial coordinates and velocity components in the center of the phase space cell $(\mathbf{i}, \mathbf{j})$, being $\mathbf{i} = (i_x, i_y)$ and $\mathbf{j} = (j_x, j_y, j_z)$.

The algorithm that advances $f^n_{\mathbf{i},\mathbf{j}} = f_{\mathbf{i},\mathbf{j}}(t_n)$ to $f^{n+1}_{\mathbf{i},\mathbf{j}} = f_{\mathbf{i},\mathbf{j}}(t_n + \Delta t)$ is constructed by time-splitting the evolution operator into a free streaming step, in which the r.h.s. of Eq. (1) is neglected, and a purely collisional step, in which spatial motion is frozen and only the effect of the r.h.s. is taken into account. More precisely, the distribution function $f^n_{\mathbf{i},\mathbf{j}}$ is advanced to $f^{n+1}_{\mathbf{i},\mathbf{j}}$ by computing an intermediate value from the free streaming equation

$$\frac{\partial f}{\partial t} + v_x \frac{\partial f}{\partial x} + v_y \frac{\partial f}{\partial y} = 0 \qquad (11)$$

Eq. (11) is solved by a simple first order explicit upwind conservative scheme. After completing the free streaming step, $f^{n+1}_{\mathbf{i},\mathbf{j}}$ is obtained by solving the homogeneous relaxation equation

$$\frac{\partial f}{\partial t} = C(f, f) \qquad (12)$$

The method of solution of Eq. (12) is different depending on the collision integral, Eq. (4) or Eq. (6). When the collision integral is given by Eq. (4), Eq. (12) is solved with a semi-regular method. At each spatial location $(x(i_x), y(i_y))$, Eq. (12) is integrated over the cell of the velocity space with center $\mathbf{j}$, $C_{\mathbf{j}}$,

$$\frac{dN_{\mathbf{i},\mathbf{j}}}{dt} = \int_{C_{\mathrm{j}}} C(f, f) d\mathbf{v} \qquad (13)$$

where $N_{\mathbf{i},\mathbf{j}}$ represents the expected number of atoms in the cell centered around the velocity node $\mathbf{j}$, i.e., $N_{\mathbf{i},\mathbf{j}}(t) = \Delta\mathcal{V} f^n_{\mathbf{i},\mathbf{j}}$. The integral in Eq. (13) can be transformed into an integral extended to the whole velocity domain $\mathcal{V}$ by introducing $\chi_{\mathbf{j}}$, the characteristic function of the cell $C_{\mathbf{j}}$

$$\frac{dN_{\mathbf{i},\mathbf{j}}}{dt} = \int_{\mathcal{V}} \chi_{\mathbf{j}} C(f, f) d\mathbf{v} \qquad (14)$$

Now, taking into account Eq. (4) to replace $C(f, f)$ and making use of some fundamental properties of the collision integral [19], Eq. (14) can be written in the following form

$$\begin{aligned} \frac{dN_{\mathbf{i},\mathbf{j}}}{dt} &= \frac{d^2}{4} \int_{\mathcal{V}\otimes\mathcal{V}} d\mathbf{v} d\mathbf{v}_1 \Phi_0(\mathbf{v}) \Phi_0(\mathbf{v}_1) \\ & \int_{-1}^{1} dk_z \int_0^{2\pi} d\phi \left[\chi_{\mathbf{j}}(\mathbf{v}^*) + \chi_{\mathbf{j}}(\mathbf{v}_1^*) - \chi_{\mathbf{j}}(\mathbf{v}) - \chi_{\mathbf{j}}(\mathbf{v}_1) \right] \\ & f(\mathbf{v}) f(\mathbf{v}_1) \Phi_0^{-1}(\mathbf{v}) \Phi_0^{-1}(\mathbf{v}_1) |\hat{\mathbf{k}} \cdot \mathbf{v}_r| \end{aligned} \qquad (15)$$

The eight-fold integral in Eq. (15) is calculated by a Monte Carlo quadrature method [20], since a regular quadrature formula would be too demanding in terms of computing time. The advantage of writing the rate of change of $N_{\mathbf{i},\mathbf{j}}$ in the above form is that the equilibrium Maxwellian distribution function may be considered a probability density function from which the velocity points are drawn to estimate the collision integral with lower variance. The Monte Carlo estimate of the rate of change of the particle number is then written as

$$\begin{aligned} \frac{dN_{\mathbf{i},\mathbf{j}}}{dt} &= \frac{n_0^2 d^2 \pi}{N_t} \sum_{l=1}^{N_t} \left[\chi_{\mathbf{j}}(\mathbf{v}_l^*) + \chi_{\mathbf{j}}(\mathbf{v}_{1l}^*) - \chi_{\mathbf{j}}(\mathbf{v}_l) - \chi_{\mathbf{j}}(\mathbf{v}_{1l}) \right] \\ & \frac{f(\mathbf{v}_l)}{\Phi_0(\mathbf{v})} \frac{f(\mathbf{v}_{1l})}{\Phi_0(\mathbf{v}_{1l})} |\hat{\mathbf{k}} \cdot \mathbf{v}_r| \end{aligned} \qquad (16)$$

Once the collision integral have been evaluated, the solution is advanced from the nth time level to the next according to the explicit scheme

$$f^{n+1}_{\mathbf{i},\mathbf{j}} = f^n_{\mathbf{i},\mathbf{j}} + Q^n_{\mathbf{i},\mathbf{j}} \Delta t \qquad (17)$$

where

$$Q^n_{\mathbf{i},\mathbf{j}} = \frac{1}{\Delta\mathcal{V}} \frac{d}{dt} N_{\mathbf{i},\mathbf{j}} \qquad (18)$$

Although memory demanding, the method outlined above produces accurate approximations of $f(\mathbf{x}, \mathbf{v}|t)$ which do not require time averaging to provide smooth macroscopic fields. A drawback of the technique is that, due to the discretization in the velocity space, mass, momentum and energy are not exactly conserved. The numerical error is usually small but tends to accumulate during the time evolution of the distribution function. The correction procedure proposed in Ref. [21] has been adopted to overcome this difficulty. At each time step the distribution function is corrected in the following way

$$\tilde{f}^{n+1}_{\mathbf{i},\mathbf{j}} = f^{n+1}_{\mathbf{i},\mathbf{j}} \left[1 + A + \mathbf{B} \cdot \mathbf{v} + C\mathbf{v}^2\right] \tag{19}$$

where the constants A, $\mathbf{B}$ and C are determined from the conditions

$$\int \psi(\mathbf{v}) \tilde{f}^{n+1}(\mathbf{v}) d\mathbf{v} = \int \psi(\mathbf{v}) f^{n}(\mathbf{v}) d\mathbf{v} \tag{20}$$

where $\psi(\mathbf{v}) = 1, \mathbf{v}, \mathbf{v}^2$.
The solution of the homogeneous relaxation equation (12) is much simpler when the collision integral is given by Eq. (6). Since n, $\mathbf{V}$ and T are conserved during homogeneous relaxation, Eq. (12) can be exactly solved to obtain

$$f^{n+1}_{\mathbf{i},\mathbf{j}} = \left[1 - \exp(-\nu_{\mathbf{i}}\Delta t)\right] \Phi_{\mathbf{i},\mathbf{j}} + \exp(-\nu_{\mathbf{i}}\Delta t) f^{n}_{\mathbf{i},\mathbf{j}} \tag{21}$$

in each cell $(\mathbf{i}, \mathbf{j})$ of the phase space. Since the density, bulk velocity and temperature obtained from the discretized Maxwellian distribution function $\Phi_{\mathbf{i},\mathbf{j}}$ are not exactly equal to $n_{\mathbf{i}}$, $\mathbf{V}_{\mathbf{i}}$ and $T_{\mathbf{i}}$, to ensure the exact conservation of mass momentum and energy, the correction procedure represented by Eqs. (19)-(20) should be used. Independently of the collision integral employed, the computational work associated to free streaming and relaxation sub-steps can be easily parallelized, each of them consisting of a number of independent threads.

4. CUDA

4.1. GPU and CUDA™ overview

NVIDIA GPU is built around a fully programmable processor array organized into a number of multiprocessor with a SIMD-like architecture[12], i.e. at any given clock cycle, each core of the multiprocessor executes the same instruction but operates on different data. CUDA™ is the high level programming language specifically created for developing applications on this platform.
A CUDA™ program is organized into a serial program which runs on the host CPU and one or more kernels which define the computation to be performed in parallel by a massive number of threads. Threads are organized into a three-level hierarchy. At the highest level, all threads form a grid; they all execute the same kernel function. Each grid consists of many different blocks which contain the same number of threads. A single multiprocessor can manage a number of blocks concurrently up to resource limits. Blocks are independent, meaning that a kernel must execute correctly no matter the order in which blocks are run. A multiprocessor executes a group of threads belonging to the active block, called warp. All threads of a warp execute the same instruction but operate on different data. If a kernel contains a branch and threads of the same warp follow different paths, then the different paths are executed sequentially (warp divergence) and the total run time is the sum of all the branches. Divergence and reconvergence are managed in hardware but may have a serious impact on performance. When the instruction has been executed, the multiprocessor moves to another warp. In this manner the execution of threads is not so much simultaneous as it is interleaved.
Each multiprocessors has a number of register which are dynamically partitioned among the threads running on it. Registers are memory spaces that are readable and writable only by the thread to which they are assigned. Threads of a single block are allowed to synchronize with each other and are available to share data through a high-speed shared memory. Threads from different blocks in the same grid may coordinate only via operations in a slower global memory space which is readable and writeble by all threads in a kernel as well as by the host. Shared memory can be accessed by threads within a block as quickly as accessing registers. Instead, reading from and writing to the global memory is particularly expensive, unless the access is coalesced [12]. Because of the interleaved warp execution, memory access latency are partially hidden, i.e., threads which have read their data can be performing computations while other warps running on the same multiprocessor are waiting for their data to come in from global memory. Note however that GPU global memory is still ten time faster than the main memory of recent CPUs.
Code optimization is a delicate task. In general, applications which require many arithmetic operations between memory read/write, and which minimize the number of out-of-order memory access, tend to perform better. Number of blocks and number of threads per block have to be chosen carefully. There should be at least as many blocks as there are multiprocessor in the device. Running only one block per multiprocessor can force the multiprocessor to idle during thread synchronization and device memory reads. By increasing the number of blocks, on the other hand, the amount of available shared memory for each block diminishes. Allocating more threads per block is better for efficient time slicing, but the more threads per block, the fewer registers are available per thread.

4.2. CUDA™ implementation

The code to numerically solve Eq. (1) is organized into a host program, which deals with all memory management and other setup tasks, and a number of kernels running on the GPU. One performs the streaming step and the others perform the collision step.

For each given cell of the velocity space, the streaming step involves the distribution function evaluated at different space locations. The key performance enhancing strategy is to allow threads to cooperate in the shared memory. The threads should thus grouped into as many blocks as the cells in the velocity space with a number of threads per block equals to the number of cells in the physical space. In practical applications, however, the number of cells in the physical space is greater than the maximum allowable number of threads per block. In order to fit into the device's resources, hence, the number of threads per block is set to a lower value which is chosen to maximize the utilization of registers and shared memory usage. When a block become active, each thread loads one element of the distribution function from global memory, stores it into shared memory and then update its value. To ensure non-overlapping access, threads are synchronized at the onset of both reading from and writing to the global memory. In order the access to the global memory to be coalesced, the discretized distribution function has been organized such that the value which refers to cells which are adjacent in the physical space are stored in contiguous memory locations. A random memory access would determine otherwise a performance bottleneck. Threads which update boundary points perform calculations which are slightly different to account for the incoming Maxwellian flux from the boundary of the domain. This leads to a thread divergence which determines some code inefficiency. However, testing shows that the performance loss is small.

The relaxation step in a cell of the phase space does not involve any information from nearby cell, whatever the collision operator employed, Eqs. (17) and (21). This naturally fits for GPUs.

The relaxation step for the Boltzmann equation is organized into two kernels. The first kernel computes the sequence of N_t velocity vectors $\mathbf{v}_l$ and $\mathbf{v}_{1l}$ by having a thread associate to each of the N_t samples. In the second kernel, there are as many threads as the number of cells in the physical space. The kernel updates the distribution function according to Eq. (17) and enforces the conservation properties by means of the correction procedure represented by Eqs. (19)-(20).

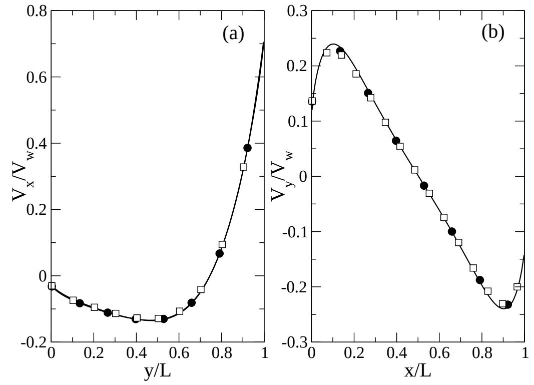

Figure 2: Profiles of (a) the horizontal component of the velocity on the vertical plane crossing the center of the cavity and (b) the vertical component of the velocity on the horizontal plane crossing the center of the top vortex. Solid lines: solution reported in Ref. [22]. Filled circles: parallel solution of the BGKW model equation. Unfilled squares: parallel solution of the Boltzmann equation.

Unlike the collision operator (4), the concurrent computation of (6) would be possible mapping each thread to a single cell in the phase space. According to Eq. (21), however, one must first calculate in each cell of the physical space the macroscopic quantities, Eqs. (8)-(10). In the attempt of reducing data transfers from and to the global memory, the computation of the macroscopic quantities and the collision step are then performed in the same kernel, by having a thread associated to each cell of the physical space. Although this choice reduce the overall number of threads, it is not quite limiting since for realistic three-dimensional problems, one would refine the physical grid more than the velocity grid.

5. Results and discussion

In Ref. [22], the cavity flow problem has been solved by assuming that $V_w \ll \sqrt{2RT_0}$ and thus Eq. (1) with the collision integral (6) has been linearized around the equilibrium state at rest. In order to reproduce these results, the dimensionless lid velocity is here set to 0.1. The gas is thus in a weakly non-equilibrium state and one can expect that the nonlinear results approach the linearized ones. The square cavity, $[0, \delta] \times [0, \delta]$, has been divided into 160×160 cells with uniform width. Here δ is the rarefaction parameter which is proportional to the inverse of the Knudsen number. In the following, δ will be based on the mean free path of the BGKW equation. In the Boltzmann results, the mean free path of the hard-sphere molecular system is converted to that of the BGKW equation through the viscosity. The cavity flow problem has been solved over a wide range of

	D			G		
δ	BHS	BGKW	Ref [22]	BHS	BGKW	Ref [22]
0.1	0.672	0.676	0.677	0.0970	0.0970	0.0974
1	0.623	0.625	0.628	0.103	0.103	0.104
10	0.392	0.392	0.413	0.142	0.141	0.145

Table 1: Drag coefficient, D, and reduced flow rate, G, versus the rarefaction parameter, δ

the rarefaction parameter. The computational grid in the physical space has been chosen to achieve the convergence of the results in the whole range of rarefaction parameter considered. The number of velocity cells has been set $N_{v_\alpha} = 20$ with $v_\alpha \in [-3, 3]$. Finally, the dimensionless time step has been varied in the range $10^{-4} - 10^{-2}$ depending on the value of the rarefaction parameter.

Figures 2a and 2b show the profiles of the horizontal component of the velocity, V_x, on the vertical plane crossing the center of a square cavity and the vertical component of the velocity, V_y, on the horizontal plane crossing the center of the top vortex, respectively, for $\delta = 10$. Solid lines are the numerical results reported in Ref. [22] whereas filled circles and unfilled squares are the results obtained by solving the BGKW model equation and the Boltzmann equation with the parallel codes, respectively. Although the parallel codes solve the nonlinear equations, for a sufficiently low velocity of the lid of the cavity there is an almost perfectly match with the linearized results presented in Ref. [22]. In order to proceed with a more detailed comparison, we introduce two overall quantities, namely the mean dimensionless shear stress along the moving plate, D, and the dimensionless flow rate of the main vortex, G. The former quantities is obtained by integrating the shear stress along the lid of the cavity, the latter by integrating the x-component of the velocity profile along the plane crossing the center of the cavity from the center of the top vortex up to the lid.

Table 1 compares the prediction of D and G obtained by solving Eq. (1) with the parallel codes and the values reported in Ref. [22], for different values of the rarefaction parameter. The agreement is good.

Figure 3 shows the drag coefficient versus the dimensionless time for both Boltzmann and BGKW equations. Although the noise due to the finiteness of the sample used to evaluate the collision integral, the semi-regular method is able to capture the transient of the solution. Smoother results may be obtained either by increasing N_t or by an appropriate time averaging.

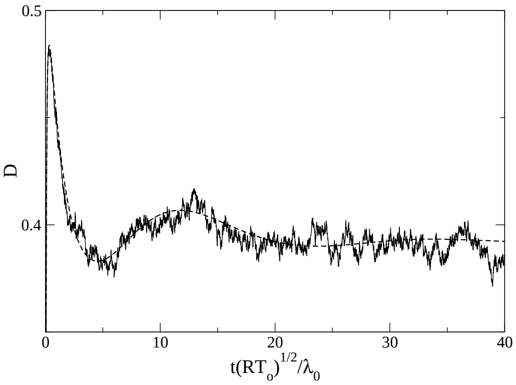

Figure 3: Drag coefficient versus dimensionless time. Solid line: Boltzmann solution with $N_t = 63488$. Dashed line: BGKW solution.

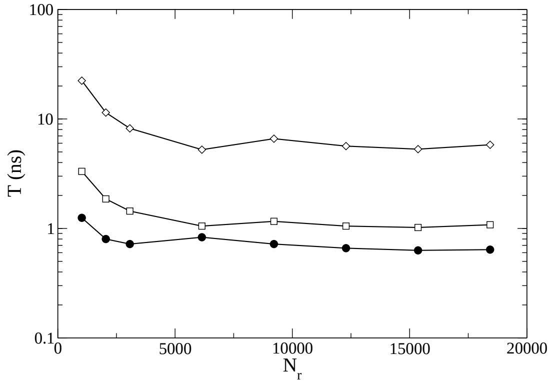

Figure 4: Time spent for processing one cell of the phase space versus the number cells used to discretized the physical space, N_r. Filled circles: BGKW solution. Unfilled squares: BHS solution with $N_t = 4096$. Unfilled diamonds: BHS solution with $N_t = 40960$. $N_{v_\alpha} = 16$.

Figure 4 shows the time spent for processing one cell of the the phase space at each time step, expressed in nanoseconds, versus the number of cells used to discretized the physical space, $N_r = N_s^2$. Filled circles are the results for the BGKW model equation whereas unfilled squares and diamonds are the results for the Boltzmann equation with $N_t = 4096$ and $N_t = 40960$, respectively. The codes fully utilize the GPU when the number of cells in the physical space is about 10000. For a greater number of cells, the total execution time increases linearly and hence the time spent for processing one cell of the phase space at each time step is nearly constant. The code which solves the Boltzmann equation is slower than the code which solves the BGKW model equation by a factor between 2 and 10, depending on the number N_t of velocities samples. According to the results reported in Ref. [16], the speed-up of the

BGKW code is greater than 100 and the same can be inferred for the Boltzmann code. The algorithms for the BGKW and Boltzmann equation, in fact, show the same degree of parallelism. Independently of the value of the rarefaction parameter, the BGKW solution takes 8 minutes whereas the Boltzmann solution needs 10 minutes for $\delta = 0.1$, 18 minutes for $\delta = 1$ and 75 minutes for $\delta = 10$. This is due to the fact that by increasing the rarefaction parameter, it is necessary to increase N_t.

6. Conclusion

The purpose of this paper was to develop algorithms to solve non-equilibrium low speed gas flows onto GPUs using NVIDIA CUDA™ programming model. The two-dimensional low Mach number driven cavity flow has been chosen as test problem. The Boltzmann equation has been solved with a semi-regular method based on the Monte Carlo evaluation of the collision integral, while the BGKW model equation has been solved by directly discretizing the collision integral with a finite difference scheme. These methods of solution are ideally suited for the parallel architecture provided by commercially available GPUs. The solutions of the linearized kinetic equations are obtained as the limiting solutions of the corresponding non-linear equations for a vanishing perturbation. Numerical experiments indicate that it is possible to cut down the computing time of the sequential codes up to two order of magnitudes. The algorithm described can easily be extended to three dimensions and to non-equilibrium flows of polyatomic gases and/or involving chemical reactions.

ACKNOWLEDGMENTS

Support received from **Fondazione Cariplo** within the framework of project *"Fenomeni dissipativi e di rottura in micro e nano sistemi elettromeccanici"*, and **Galileo Programme** of Università Italo-Francese within the framework of project MONUMENT (MOdellizzazione NUmerica in MEms e NanoTecnologie) is gratefully acknowledged. The authors wish to thank Professor Dimitris Valougeorgis for providing his numerical results.

References

[1] C. Cercignani, A. Frezzotti, S. Lorenzani, Using the kinetic equations for MEMS and NEMS, in *Advances in Multiphysics Simulation of MEMS and NEMS* edited by Aluru, C. Cercignani, A. Frangi and S. Mukherjee, Imperial College Press, London, 2008.

[2] S. Naris, D. Valougeorgis, The driven cavity flow over the whole range of the Knudsen number, *Phys. Fluids* **17**, 097106-12, 2005.

[3] I. Graur, F. Sharipov, Gas flow through an elliptical tube over the whole range of the gas rarefaction, *Eur. J. Mech. B/Fluids*, **27** (3), 335-345, 2008.

[4] S. Lorenzani, L. Gibelli, A. Frezzotti, A. Frangi, C. Cercignani, Kinetic approaches to gas flow in microchannels, *Nanoscale and Microscale Thermophysical Engineering*, **11**, 211-226, 2007.

[5] G. A. Bird, *Molecular Gas Dynamics and the Direct Simulation of Gas Flows*, Oxford University Press, 1994.

[6] J. Chun and D. L. Koch, A direct simulation Monte Carlo method for rarefied gas flows in the limit of small Mach number, *Phys. Fluids* **17**, 107107-14, 2005.

[7] T. M. M. Homolle, N. G. Hadjiconstantinou, A low-variance deviational simulation Monte Carlo for the Boltzmann equation, *J. Comput. Phys.* **226** (2), 2341-2358, 2007.

[8] W. Wagner, Deviational particle Monte Carlo for the Boltzmann equation, *Monte Carlo Methods and Applications*, **14** (3), 191-268, 2008.

[9] M. Gad-el-Hak, The fluid mechanics of microdevices - the Freeman Scholar Lecture, *J. Fluids Eng. (Trans. ASME)*, **121**, 5-33, 1999.

[10] A. Frezzotti, Numerical study of the strong evaporation of a binary mixture, *Fluid Dynamics Research*, **8** (5-6), 175-187, 1991.

[11] F. Tcheremissine, Direct numerical solution of the Boltzmann Equation, RGD24, AIP Conference proceeding, 762, 677-685, 2005.

[12] NVIDIA Corporation, *NVIDIA CUDA Programming Guide*, Jun. 2008. Version 2.0. http://www.nvidia.com/CUDA

[13] E. Elsen, P. LeGresley, E. Darve, Large calculation of the flow over a hypersonic vehicle using a GPU, *J. Comp. Phys.* **227**, 10148-10161, 2008.

[14] G. Stantchev, W. Dorland, N. Gumerov, Fast parallel Particle-To-Grid interpolation for plasma PIC simulations on the GPU, *J. Parallel Distrib. Comput.* **68**, 1339-1349, 2008.

[15] J. A. Anderson, C. D. Lorenz, A. Travesset, General purpose molecular dynamics simulations fully implemented on graphics processing units, *J. Comp. Phys.* **227**, 5342-5339, 2008.

[16] A. Frezzotti, G. P. Ghiroldi, L. Gibelli, Solving Kinetic Equations on GPUs I: Model Kinetic Equations, eprint arXiv:0903.4044

[17] P. L. Bhatnagar, E. P. Gross, M. Krook, A model for collision processes in gases. I. Small amplitude processes in charged and neutral one-component systems, *Phys. Rev.* **94**, 511-525, 1954.

[18] P. Welander, On temperature jump in a rarefied gas, *Arkiv fiir Fysik* 7 **44**, 507-553, 1954.

[19] C. Cercignani, *The Boltzmann Equation and Its Applications*, Springer-Verlag, New York, 1988.

[20] M. H. Kalos, P. A. Whitlock, *Monte Carlo Methods*, Wiley, New York, 1986.

[21] V. V. Aristov, F. G. Theremissine, *U.S.S.R. Comput. Math. Math. Phys.*, **20**, 208-225, 1980.

[22] S. Varoutis, D. Valougeorgis, F. Sharipov, Application of the integro-moment method to steady-state two-dimensional rerafeied gas flows subject to boundary induced discontinuities, *J. Comput. Phys.* **227**, 6272-6287, 2008.