Solving Kinetic Equations on GPU's

Aldo Frezzotti, Gian Pietro Ghiroldi and Livio Gibelli

Contents

*Dipartimento di Matematica del Politecnico di Milano
Piazza Leonardo da Vinci 32, 20133 Milano, Italy

1 Introduction

Non-equilibrium gas flows are met in several different physical situations ranging from the re-entry of spacecraft in upper planetary atmospheres to fluid-structure interaction in small-scale devices Cercignani (1988); Bird (1994). The correct description of nonequilibrium effects requires replacing the traditional hydrodynamic equations with the Boltzmann equation which, in the absence of assigned external force fields, reads

$$\frac{\partial f}{\partial t} + \boldsymbol{v} \cdot \nabla_{\boldsymbol{r}} f = \mathcal{C}(f, f) \tag{1}$$

In Eq. (1), the distribution function $f(\boldsymbol{r}, \boldsymbol{v}|t)$ is the atomic number density at the single atom phase space point $(\boldsymbol{r}, \boldsymbol{v})$ at time t. The symbols $\boldsymbol{r}$ and $\boldsymbol{v}$ denote atom position and velocity, respectively. The left hand side of Eq. (1) represents the rate of change of f due to the indipendent motion of gas atoms. Effects of collisions are accounted for by the source term $\mathcal{C}(f, f)$ which is a non-linear functional of f whose precise structure depends on the assumed atomic interaction forces. Obtaining numerical solutions of Eq. (1) for realistic flow conditions is a challenging task because it has the form of a non-linear integro-differential equation in which the unknown function, f, depends on seven variables. Numerical methods used to solve Eq. (1) can be roughly divided into three groups:

(a) Particle methods

(b) Semi-regular methods

(c) Regular methods

Methods in group (a) originate from the Direct Simulation Monte Carlo (DSMC) scheme proposed by G.A. Bird Bird (1994). They are by far the most popular and widely used simulation methods in rarefied gas dynamics. The distribution function is represented by a number of mathematical particles which move in the computational domain and collide according to stochastic rules derived from Boltzmann equation. Macroscopic flow properties are usually obtained by time averaging particle properties. If the averaging time is long enough, then accurate flow simulations can be obtained by a relatively small number of particles. The method can be easily extended to deal with mixtures of chemically reacting polyatomic species Bird (1994) and to dense fluids Frezzotti et al. (2005). Although DSMC (in its traditional implementation) is to be recommended in simulating most of rarefied gas flows, it is not well suited to the simulation of low Mach number or unsteady 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 Homolle and Hadjiconstantinou (2007); Wagner (2008). However, in simulating high frequency unsteady flows, typical of microfluidics application to MEMS, 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.

Methods in groups (b) and (c) adopt similar strategies in discretizing the distribution function on a regular grid in the phase space and in using finite difference schemes to

approximate the streaming term. However, they differ in the way the collision integral is evaluated. In semi-regular methods $\mathcal{C}(f,f)$ is computed by Monte Carlo or quasi Monte Carlo quadrature methods Frezzotti (1991); F. (2005) whereas deterministic integration schemes are used in regular methods Aristov (2001). Whatever method is chosen to compute the collision term, the adoption of a grid in the phase space considerably limits the applicability of methods (b) and (c) to problems where particular symmetries reduce the number of spatial and velocity variables. As a matter of fact, a spatially three-dimensional problem would require a memory demanding six-dimensional phase space grid. Extensions to polyatomic gases are possible Frezzotti (2007) but the necessity to store additional variables associated with internal degrees of freedom further limits the applications to multi-dimensional flows. Therefore, until now the direct solution of the Boltzmann equation by semi-regular or regular methods has not been considered a viable alternative to DSMC for simulating realistic flows, not even for low speed and/or unsteady flows. The availability of low cost Graphics Processing Units (GPUs) has changed the situation. Although GPUs have been originally developed for graphics rendering, they have became general purpose desktop supercomputers capable of delivering teraflops peak performance at the price of conventional workstations. Mapping efficiently an algorithm on the SIMD-like architecture of the GPUs, however, is a difficult task which often requires the algorithm to be revised or even redesigned to both balance the hardware structure benefits and meet the implementation requirements. For instance, preliminary tests, performed within the framework of the research work described here, have shown that the standard form of DSMC is not efficiently ported on GPU's because of their SIMD architecture. On the other hand, we have shown in Ref. Frezzotti et al. (2010), that a regular method of solution of the BGKW kinetic model equation is ideally suited for GPUs. The main aim of the present paper is to translate efficiently a semi-regular method of solution of the full non-linear Boltzmann equation into a parallel code to be executed on a GPU. The efficiency of the algorithm is assessed by solving the classical two-dimensional driven cavity flow. It is shown that it is possible to cut down the computing time of the sequential codes of two order of magnitudes. This paper is organized as follows. Sections 2 and 3 are devoted to a concise description of the mathematical model and the adopted numerical method. In Section 4 the key aspects of the GPU hardware architecture and CUDA™ programming model are briefly described and implementation details are provided. Sections 5 is devoted to the description of the test problem and the discussion of the results. Concluding remarks are presented in Section 6.

2   Mathematical Formulation

The hard-sphere model is a good approximation for simple fluids, that is fluids whose properties are largely determined by harshly repulsive short range forces. The hard-sphere Boltzmann collision integral reads

$$\mathcal{C}(f,f) = \frac{\sigma^2}{2} \int (f^* f_1^* - f f_1) |\hat{\boldsymbol{k}} \cdot \boldsymbol{v}_r| d\boldsymbol{v}_1 d^2 \hat{\boldsymbol{k}} \qquad (2)$$

In Eq. (2), σ is the hard sphere diameter, $\boldsymbol{v}_r = \boldsymbol{v} - \boldsymbol{v}_1$ is the relative velocity between two colliding atoms and $f^* = f(\boldsymbol{r}, \boldsymbol{v}^*|t), f_1^* = f(\boldsymbol{r}, \boldsymbol{v}_1^*|t), f_1 = f(\boldsymbol{r}, \boldsymbol{v}_1|t)$. Here and in the remainder of the paper, integration extends over the whole velocity space. Similarly, the solid angle integration is over the surface of the unit sphere, whose points are associated with the unit vector $\hat{\boldsymbol{k}}$. The pre-collisional velocities, $(\boldsymbol{v}^*, \boldsymbol{v}_1^*)$, are obtained from the post-collision velocities, $(\boldsymbol{v}, \boldsymbol{v}_1)$, and the unit vector on the sphere, $\hat{\boldsymbol{k}}$, by the relationships

$$\boldsymbol{v}^* = \boldsymbol{v} + \left(\boldsymbol{v}_r \cdot \hat{\boldsymbol{k}} \right) \hat{\boldsymbol{k}} \qquad (3)$$

$$\boldsymbol{v}_1^* = \boldsymbol{v}_1 - \left(\boldsymbol{v}_r \cdot \hat{\boldsymbol{k}} \right) \hat{\boldsymbol{k}} \qquad (4)$$

In view of the applications to the study of low Mach flows, Refs. Homolle and Hadjiconstantinou (2007); Baker and Hadjiconstantinou (2008) will be followed to rewrite Eqs. (1) and (2) in terms of the deviational part of the distribution function, $h(\boldsymbol{r}, \boldsymbol{v}|t)$, defined as

$$f = \Phi_0 \left(1 + \epsilon h \right) \qquad (5)$$

where ϵ is a parameter that measures the deviation from equilibrium conditions and $\Phi_0(\boldsymbol{r}, \boldsymbol{v}|t)$ is the Maxwellian at equilibrium with uniform and constant density n_0 and temperature T_0, i.e.,

$$\Phi_0 = \frac{n_0}{(2\pi R T_0)^{3/2}} \exp \left(-\frac{\boldsymbol{v}^2}{2 R T_0} \right) \qquad (6)$$

The physical rationale behind this formulation is a proper rescaling of the (small) deviation from equilibrium to reduce the variance in the Monte Carlo evaluation of the collision integral and thus to capture arbitrarily small deviations from equilibrium with a computational cost which is independent of the magnitude of the deviation. By substituting Eq. (5) into Eq. (1), we obtain

$$\frac{\partial h}{\partial t} + \boldsymbol{v} \cdot \nabla_{\boldsymbol{r}} h = \mathcal{Q}(h, h) \qquad (7)$$

where the collision integral takes the form

$$\mathcal{Q}(h, h) = \frac{d^2}{2} \int \Phi_0 \, \Phi_{0,1} \left[h^* + h_1^* - h - h_1 + \epsilon \left(h^* h_1^* - h h_1 \right) \right] |\hat{\boldsymbol{k}} \cdot \boldsymbol{v}_r| d\boldsymbol{v}_1 d^2 \hat{\boldsymbol{k}} \qquad (8)$$

Eq. (8) has been obtained by using the property $\Phi_0^* \, \Phi_{01}^* = \Phi_0 \, \Phi_{01}$ Cercignani (1988).
For later reference, we here report the expressions of dimensionless perturbed density,

velocity, temperature and stress tensor

$$\rho = \frac{n - n_0}{n_0}\frac{1}{\epsilon} = \int \Phi_0\, h\, d\boldsymbol{v} \tag{9}$$

$$\boldsymbol{u} = \frac{\boldsymbol{V}}{\sqrt{2RT_0}}\frac{1}{\epsilon} = \frac{1}{1 + \epsilon\,\rho}\int \Phi_0\, h\, \boldsymbol{v}\, d\boldsymbol{v} \tag{10}$$

$$\theta = \frac{T - T_0}{T_0}\frac{1}{\epsilon} = \frac{1}{1 + \epsilon\,\rho}\left(\frac{1}{3}\int \Phi_0\, h\, \boldsymbol{v}^2\, d\boldsymbol{v} - \rho\right) - \frac{\epsilon}{3}\boldsymbol{u}^2 \tag{11}$$

$$\Pi_{ij} = \frac{p_{ij} - p_0\delta_{ij}}{p_0}\frac{1}{\epsilon} = \int \Phi_0\, h\, v_i v_j\, d\boldsymbol{v} - \epsilon\, u_i\, u_j - \epsilon^2\, \rho\, u_i\, u_j \tag{12}$$

where $p_0 = n_0 R T_0$. At the boundaries, Maxwell's completely diffuse boundary condition is assumed. Accordingly, the distribution function of atoms emerging from walls is given by the following expression

$$\Phi_0 + \epsilon\,\Phi_0\, h = n_w\,\Phi_w \qquad (\boldsymbol{v} - \boldsymbol{V}_w)\cdot\hat{\boldsymbol{n}} > 0 \tag{13}$$

In Eq. (13), $\hat{\boldsymbol{n}}$ is the inward normal and Φ_w is the normalized wall Maxwellian distribution function

$$\Phi_w(\boldsymbol{r}, \boldsymbol{v}) = \frac{1}{(2\pi RT_w)^{3/2}}\exp\left[-\frac{(\boldsymbol{v} - \boldsymbol{V}_w)^2}{2RT_w}\right] \tag{14}$$

where $\boldsymbol{V}_w$ the wall velocity and T_w the wall temperature. The wall density n_w is determined by imposing zero net mass flux at any boundary point

$$n_w\int_{\boldsymbol{c}\cdot\hat{\boldsymbol{n}} > 0}|\boldsymbol{c}\cdot\hat{\boldsymbol{n}}|\Phi_w d\boldsymbol{v} = \int_{\boldsymbol{c}\cdot\hat{\boldsymbol{n}} < 0}|\boldsymbol{c}\cdot\hat{\boldsymbol{n}}|\Phi_0 d\boldsymbol{v} + \epsilon\int_{\boldsymbol{c}\cdot\hat{\boldsymbol{n}} < 0}|\boldsymbol{c}\cdot\hat{\boldsymbol{n}}|\Phi_0\, h d\boldsymbol{v} \tag{15}$$

where $\boldsymbol{c} = \boldsymbol{v} - \boldsymbol{V}_w$. It is worth noticing that when the perturbation is sufficiently small, i.e., $\epsilon \to 0$, Eq. (7) reduces to the linearized Boltzmann equation and Eqs. (9)-(12) to the linearized expression of the macroscopic quantities. The formulation in terms of the deviational part of the distribution function, however, is not restricted to a vanishing perturbation but it is valid in the non-linear case as well.

3   Outline of the numerical method

The method of solution adopted to solve Eq. (7) is a semi-regular method in which a finite difference discretization are used to evaluate the free streaming term on the left hand side while the collision integral on the right hand side is computed by a Monte Carlo technique. The three-dimensional physical space is divided into $N_r = N_x \times N_y \times N_z$ parallelepipedal cells. Likewise, the three-dimensional velocity space is replaced by a parallelepipedal box divided into $N_v = N_{v_x} \times N_{v_y} \times N_{v_z}$ cells. 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 h at any spatial position. The distribution function is assumed to be constant within each cell of the phase space. Hence, h is represented by the array $h_{\boldsymbol{i},\boldsymbol{j}}(t) = h(x(i_x), y(i_y), z(i_z), v_x(j_x), v_y(j_y), v_z(j_z)|t)$; $x(i_x), y(i_y), z(i_z)$ and $v_x(j_x), v_y(j_y), v_z(j_z)$ are the values of the spatial coordinates and velocity components in the center of the phase space cell corresponding to the indexes $\boldsymbol{i} = (i_x, i_y, i_z)$ and $\boldsymbol{j} = (j_x, j_y, j_z)$.

The algorithm that advances $h_{\boldsymbol{i},\boldsymbol{j}}^n = h_{\boldsymbol{i},\boldsymbol{j}}(t_n)$ to $h_{\boldsymbol{i},\boldsymbol{j}}^{n+1} = h_{\boldsymbol{i},\boldsymbol{j}}(t_n + \Delta t)$ is constructed by time-splitting the evolution operator into a free streaming step, in which the right hand side of Eq. (7) is neglected, and a purely collisional step, in which spatial motion is frozen and only the effect of the collision operator is taken into account. More precisely, the distribution function $h_{\boldsymbol{i},\boldsymbol{j}}^n$ is advanced to $h_{\boldsymbol{i},\boldsymbol{j}}^{n+1}$ by computing an intermediate value, $\tilde{h}_{\boldsymbol{i},\boldsymbol{j}}^{n+1}$, from the free streaming equation

$$\frac{\partial h}{\partial t} + \boldsymbol{v} \cdot \nabla_{\boldsymbol{r}} h = 0 \tag{16}$$

When solving Eq. (16), boundary conditions have to be taken into account. Eq. (16) is discretized by a simple first order explicit upwind conservative scheme. For later reference, we here report the difference scheme in the two dimensional case with $v_x > 0$ and $v_y > 0$

$$\tilde{h}_{i_x,i_y;\boldsymbol{j}}^{n+1} = (1 - \mathrm{Cu}_x - \mathrm{Cu}_y) h_{i_x,i_y;\boldsymbol{j}}^n + \mathrm{Cu}_x\, h_{i_x-1,i_y;bj}^n + \mathrm{Cu}_y\, h_{i_x,i_y-1;\boldsymbol{j}}^n \tag{17}$$

In Eq. (17), $\mathrm{Cu}_x = v_x(j_x)\Delta t/\Delta x$ and $\mathrm{Cu}_y = v_y(j_y)\Delta t/\Delta y$ are the Courant numbers in the x and y directions, respectively.

After completing the free streaming step, $h_{\boldsymbol{i},\boldsymbol{j}}^{n+1}$ is obtained by solving the homogeneous relaxation equation

$$\frac{\partial h}{\partial t} = \mathcal{Q}(h, h) \tag{18}$$

where $\mathcal{Q}(h, h)$ is given by Eq. (8). In order to be solved, Eq. (18) is first integrated over the cell of the velocity space $C_{\boldsymbol{j}}$

$$\frac{dN_{\boldsymbol{i},\boldsymbol{j}}}{dt} = \int_{C_{\boldsymbol{j}}} \mathcal{Q}(h, h) d\boldsymbol{v} \tag{19}$$

where $N_{\boldsymbol{i},\boldsymbol{j}}$ represents the deviation of the number of particles with position $\boldsymbol{r}_i$ in the velocity cell centered around the velocity node $\boldsymbol{j}$ with respect to its mean value at equilibrium, i.e., $N_{\boldsymbol{i},\boldsymbol{j}} \simeq \Delta\mathcal{V}_{\boldsymbol{j}}\, \Phi_{0,\boldsymbol{j}}\, h_{\boldsymbol{i},\boldsymbol{j}}$ with $\Delta\mathcal{V}_{\boldsymbol{j}}$ the volume of the velocity cell $C_{\boldsymbol{j}}$. The integral in Eq. (19) is then transformed into an integral extended to the whole velocity domain $\mathcal{V}$

$$\frac{dN_{\boldsymbol{i},\boldsymbol{j}}}{dt} = \int_{\mathcal{V}} \chi_{\boldsymbol{j}}\, \mathcal{Q}(h, h)\, d\boldsymbol{v} \tag{20}$$

where $\chi_{\boldsymbol{j}}$ is the characteristic function of the cell $C_{\boldsymbol{j}}$

$$\chi_j(\boldsymbol{v}) = \begin{cases} 1 & \boldsymbol{v} \in C_{\boldsymbol{j}} \\ 0 & \boldsymbol{v} \notin C_{\boldsymbol{j}} \end{cases} \tag{21}$$

Making use of some fundamental properties of the collision integral Cercignani (1988), Eq. (20) can be written in the following form

$$\frac{dN_{\boldsymbol{i},\boldsymbol{j}}}{dt} = \frac{d^2}{4} \int d\boldsymbol{v}\, d\boldsymbol{v}_1\, \Phi_0(\boldsymbol{v})\, \Phi_0(\boldsymbol{v}_1) \int_{-1}^{1} dk_z \int_{0}^{2\pi} d\phi$$

$$[\chi_{\boldsymbol{j}}(\boldsymbol{v}^*) + \chi_{\boldsymbol{j}}(\boldsymbol{v}_1^*) - \chi_{\boldsymbol{j}}(\boldsymbol{v}) - \chi_{\boldsymbol{j}}(\boldsymbol{v}_1)]\,[h(\boldsymbol{v}) + h(\boldsymbol{v}_1) + \epsilon h(\boldsymbol{v}) h(\boldsymbol{v}_1)]\, |\hat{\boldsymbol{k}} \cdot \boldsymbol{v}_r| \quad (22)$$

The eight-fold integral in Eq. (22) is calculated by a Monte Carlo quadrature method, since a regular quadrature formula would be too demanding in term of computing time. The advantage of writing the rate of change of $N_{\boldsymbol{i},\boldsymbol{j}}$ in the above form is that the gaussian distribution function Φ_0 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 integral on the right hand side of Eq. (22) gives

$$\frac{dN_{\boldsymbol{i},\boldsymbol{j}}}{dt} = \frac{n_0^2 d^2 \pi}{N_t} \sum_{l=1}^{N_t} [\chi_{\boldsymbol{j}}(\boldsymbol{v}_l^*) + \chi_{\boldsymbol{j}}(\boldsymbol{v}_{1l}^*) - \chi_{\boldsymbol{j}}(\boldsymbol{v}_l) - \chi_{\boldsymbol{j}}(\boldsymbol{v}_{1l})]$$

$$[h(\boldsymbol{v}_l) + h(\boldsymbol{v}_{1l}) + \epsilon\, h(\boldsymbol{v}_l)\, h(\boldsymbol{v}_{1l})]\, |\hat{\boldsymbol{k}} \cdot \boldsymbol{v}_r| \quad (23)$$

where N_t is the number of velocity samples Frezzotti (1991). It is worth noticing that the same set of collisions can be used to evaluate the collision integral at different space locations. Once the collision integral have been evaluated, the solution is advanced from the n-th time level to the next according to the explicit scheme

$$h_{\boldsymbol{i},\boldsymbol{j}}^{n+1} = \tilde{h}_{\boldsymbol{i},\boldsymbol{j}}^{n+1} + \mathcal{Q}_{\boldsymbol{i},\boldsymbol{j}}^{n}\, \Delta t \tag{24}$$

where

$$\mathcal{Q}_{\boldsymbol{i},\boldsymbol{j}}^{n} = \frac{1}{\Delta \mathcal{V}_{\boldsymbol{j}}\, \Phi_{\boldsymbol{j}}} \frac{dN_{\boldsymbol{i},\boldsymbol{j}}}{dt} \tag{25}$$

Although memory demanding, the method outlined above produces accurate approximations of the solution 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, 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. Aristov and G. (1980) has been adopted to overcome this difficulty. At each time step the full distribution function is corrected in the following way

$$\Phi_{0,\boldsymbol{j}}\big(1 + \epsilon\, h_{\boldsymbol{i},\boldsymbol{j}}^{n+1}\big) = \Phi_{0,\boldsymbol{j}}\big(1 + \epsilon\, \tilde{h}_{\boldsymbol{i},\boldsymbol{j}}^{n+1}\big)\,\big[1 + A + \boldsymbol{B} \cdot \boldsymbol{v} + C\boldsymbol{v}^2\big] \tag{26}$$

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

$$\int \psi(\boldsymbol{v})\, \Phi(\boldsymbol{v})\, h^{n+1}(\boldsymbol{v})\, d\boldsymbol{v} = \int \psi(\boldsymbol{v})\, \Phi(\boldsymbol{v})\, \tilde{h}^{n+1}(\boldsymbol{v})\, d\boldsymbol{v} \tag{27}$$

being $\psi(\boldsymbol{v}) = 1, \boldsymbol{v}, \boldsymbol{v}^2$. The correction procedure given by Eq. (26) involves the full distribution function and not its deviational part in order the linear system (27) to be well conditioned.

4   GPU Implementation

4.1   GPU and CUDA™ overview

NVIDIA®GPU is built around a fully programmable processors array organized into a number of multiprocessors with a SIMD-like architecture NVIDIA Corporation (2008), 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 the 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 beloging 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 re-convergence are managed in hardware but may have a serious impact on performances. When the instruction has been executed, the multiprocessor moves to another warp. In this manner the execution of threads is interleaved rather than simultaneous.

Each multiprocessor has a number of registers 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 writable 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. On the contrary, I/O operations involving global memory are particularly expensive, unless access is coalesced NVIDIA Corporation (2008). Because of the interleaved warp execution, memory access latency is 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 multiprocessors 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.

The computations that are shown below, have been performed on a commercially available GPU GeForce GTX 260 produced by NVIDIA$^{®}$ using CUDA$^{™}$ version 2.0. The GTX 260 GPU model consists of 24 streaming multiprocessors with 8 streaming processors (SP) each for a total of 192 units. Each SP is clocked at 1.242 GHz and performs up to 3 floating point operation (FLOP) per clock cycle, yielding a peak theoretical performance of 715.4 GFLOPs ($192 \times 1.242 \times 3$). Each group of SP shares one 16 kB of fast per-block shared memory while the GPU has 896 MB of device memory with a memory bandwidth of 111.9 GB/s. The graphic processing unit has been hosted by a personal computer equipped with 4 GB of main memory and an Intel$^{®}$ Core Duo Quad Q9300 CPU, running at 2.5 GHz. The host machine has also been used to run the sequential version of the program to obtain the speed-up data. The host code has been compiled using the gcc/g++ compiler with optimization option "-03".

4.2   Implementation details

The code to numerically solve Eq. (7) is organized into a host program, which deals with all memory management and other setup tasks, and three kernels running on the GPU which perform the streaming and the collision steps. In the following, we report and discuss the pseudo-codes of each kernel. Because of their different impact on the code performance, we distinguish the slow global memory reads, $\Leftarrow$, and writes, $\Rightarrow$, from the fast reads, $\leftarrow$, and writes, $\rightarrow$, from local registers and shared memory.

Algorithm (1) reports the pseudocode of the two dimensional streaming kernel. The one-dimensional case has been discussed in Ref. Frezzotti et al. (2010) whereas the extension to three-dimensional geometries is straightforward. Moreover, for clarity of presentation, the pseudo-code of the streaming kernel refers to one cell of the velocity space with $v_x > 0$ and $v_y > 0$. The other cases can be handled analogously. As shown by Eq. (17), for each cell of the velocity domain, the streaming step involves the distribution function evaluated at different space locations. Similarly to the one dimensional case, the key performance enhancing strategy is to allow threads to cooperate in the shared memory. In order to fit into the device's resources, blocks are composed by a two dimensional grid of threads with dimension $B_x \times B_y$ having each thread associated with one cell of the physical space. When a block become active, each thread loads one element of the distribution function from global memory, stores it into shared memory (line 5), updates its value according to Eqs. (17) (line 21) and then saves it back to the global memory (line 22). This procedure is repeated sequentially $(N_x/B_x - 1) \times (N_y/B_y - 1)$ times. To ensure non-overlapping access, threads are synchronized at the onset of writing to the global memory (lines 20). In order to obtain a coalesced access to the global memory, values of the discretized distribution function of cells which are adjacent in the physical space are stored in contiguous memory locations. However, not all the threads in a block can read data in a coalescent manner. In fact, in order to update $h_{i,j}$ the values of the distribution function of two upwind neighboring nodes, often referred to as "halo" nodes Micikevivius (2009), are required. The halos on one physical direction can be read in with coalesced access (line 13-19) while the others have to be read in with non-coalesced access (line 6-12). Threads which update boundary points perform calculations which are slightly different to account for the incoming Maxwellian flux from the boundaries of the domain (lines 8 and 15).

The relaxation step is organized into two kernels whose pseudo-codes are listed in Algorithms (2) and (3). The first kernel computes the sequence of N_t collisions used in the Monte Carlo evaluation of the collision integral. The second kernel updates the discretized distribution function, executes the correction procedure and computes the macroscopic quantities of interest as well.

Algorithm (2) reports the pseudo-code of the sampling kernel. Here, there are as many threads as the number of the collision samples, N_t. Firstly, each thread generates the pre-collisional velocities $\boldsymbol{v}$ and $\boldsymbol{v}_1$ by sampling the maxwellian distribution function with the Box-Muller algorithm (line 1-2) and the unit vector $\hat{\boldsymbol{k}}$ by sampling the uniform distribution on the unit sphere (line 3). Afterwards, the post-collisional velocities are evaluated (line 4-6) and the index of the velocity cells to which they belong are calculated and stored in the vectors I, I_1, I^* and I_1^* defined in the global memory (lines 7-11). Finally, for each velocity cell, the values to be added and subtracted to these velocity cells are calculated (lines 11-14) and stored (lines 15-18) in the vectors C, C_1, C^* and C_1^* defined in the global memory. It is important to note that in order to maximize the performance all the accesses to the global memory are done in a coalesced manner NVIDIA Corporation (2008).

To update the discretized distribution function in a cell of the physical space according to Eq. (24), no information from nearby space cells is required. This naturally fits for the GPU, where one may define as many threads as the number of cells in the physical space. Moreover, by having one thread for each cell of the physical space, potentially dangerous conflicts between threads are avoided and the accesses to the global memory may be coalesced. Firstly, each thread updates the discretized distribution function according to Eq. (23), then executes the correction procedure to enforce the conservation of momentum and energy, Eq. (26), and finally compute the macroscopic quantities of interest. Algorithm (3) shows the pseudo-code of the relaxation kernel. The main part of the algorithm is in between the lines 7 and 21 where the collision integral is evaluated according to the Monte Carlo method, Eq. (23). Lines from 1 to 6 and from 22 to 30 evaluate the required moments of the distribution function before and after the collision step, respectively. These moments are used in Eq. (27) to obtain the constants A, $\boldsymbol{B}$ and C. The last loop over the velocity space (lines 32-39) corrects the distribution function according to Eq. (26) and compute the macroscopic quantities of interest.

5   Test case: driven cavity flow

5.1   Formulation of the problem

The driven cavity flow is a classical benchmark problem. In spite of its simple geometry, in fact, it contains most of the features of more complicated problems described by kinetic equations Varoutis and Sharipov (2008). In the following, we restrict to the spatially two-dimensional case. We thus consider a monatomic rarefied gas contained in a square enclosure with lenght L. All the walls are kept at uniform and constant temperature T_0. Initially, the gas is supposed to be in equilibrium with density n_0 and temperature T_0. The flow is driven by the uniform translation of the lid of the cavity with velocity V_w. We describe the dynamics of the gas by Eq. (7) and assume that atoms which strike the walls are re-emitted according to the Maxwell's scattering kernel with complete accomodation, Eq. (13).

We choose the reference mean free path λ_0 to define a dimensionless position as x/λ_0, being

$$\lambda_0 = \mu_0/p_0(\pi R T_0/2)^{1/2}$$

and μ_0 is the viscosity of the hard sphere gas Chapman and Cowling (1990). Likewise the characteristic time is given by λ_0/V_0, where $V_0 = (2RT_0)^{1/2}$. The cavity flow problem has been solved for three values of the rarefaction parameter $\delta = 0.1, 1, 10$, being δ the reciprocal of the Knudsen number $\delta = L/\lambda_0$. Since the proposed method of solution is particularly effective in capturing small deviations from equilibrium, we set the lid velocity to $V_w/V_0 = 0.01$. The computations described in below, hence, refer to very low Mach number driven cavity flows. The square cavity has been divided into $N_r = N_x \times N_y$ uniform square cells, with $N_x = N_y$. Likewise the velocity space has been divided into $N_v = N_{v_x} \times N_{v_y} \times N_{v_z}$ with $N_{v_x} = N_{v_y} = N_{v_z}$. Since the deviation form equilibrium is supposed to be small, a cubic velocity space has been constructed by distributing the velocity nodes along each velocity component in the interval $[-3V_0, 3V_0]$. In order to achieve a faster convergence of the solutions in the velocity space, the lenghts of the cells are uniformly stretched with a progression ratio r_v according to the relations $\Delta v_\alpha(j_\alpha) = r_v \Delta v_\alpha(j_\alpha - 1)$, being the smaller cells located at the origin of the velocity space. More precisely, it has been chosen $N_v = 8000$ and $r_v = 0.973$ for $\delta = 0.1$ and $\delta = 1$ whereas $N_v = 5832$ and $r_v = 0.986$ for $\delta = 10$ The number of collisions used in the Monte Carlo evaluation of the collision integral have been varied with the rarefaction parameter. In particular, it has been set $N_t = 1024$ for $\delta = 0.1$, $N_t = 6144$ for $\delta = 1$, $N_t = 8192$ for $\delta = 10$. Finally, the time step has been determined by requiring that $\mathrm{Cu}_x = \mathrm{Cu}_y = 0.5$.

5.2   Results and discussion

In this section, we first carry out a convergence analysis of the method in the physical space and then we investigate the parallel performances of the code.

In order to establish the convergence rate we compute two global flowfield properties, namely the mean dimensionless shear stress on the moving wall, D, and the dimensionless flow rate of the main vortex, G. The two quantities are defined as

$$D = \frac{1}{\delta} \int_0^\delta \Pi_{xy}(x, L|t)\, dx, \qquad G = \frac{1}{\delta} \int_0^\delta |u_x(L/2, y|t)|\, dy \qquad (28)$$
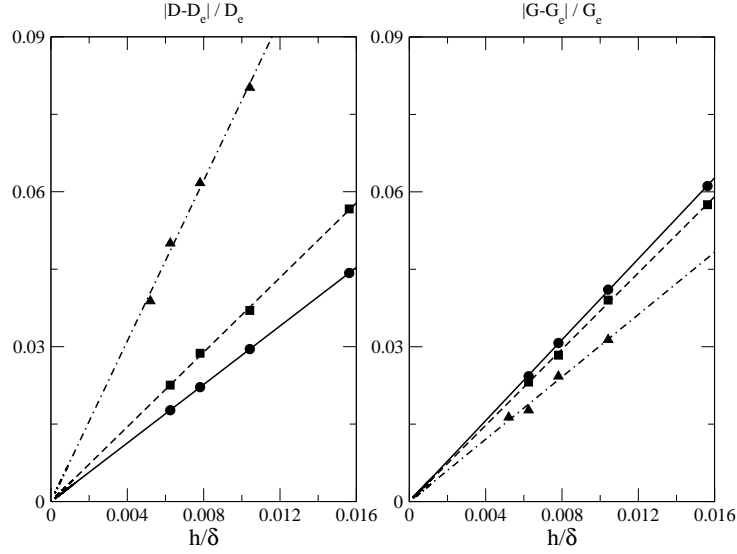
Figure 1: Absolute relative error on (a) drag coeffi-
cient and (b) mean flow rate for $\delta = 0.1$ (circles), $\delta = 1$
(squares) and $\delta = 10$ (triangles) versus the size h/δ of
the physical grid. Lines are the least-mean square fit
of the results. $N_v = 8000$, $r_v = 0.973$, $N_t = 1024$ for
$\delta = 0.1$; $N_v = 8000$, $r_v = 0.973$, $N_t = 6144$ for $\delta = 1$;
$N_v = 5832$, $r_v = 0.986$, $N_t = 8192$ for $\delta = 10$.

The absolute relative error in the stationary values of D and G are shown in Figs 1a and
1b, respectively, versus the spatial grid size, $h/\delta = 1/N_r$, and for $\delta = 0.1$ (circles), $\delta = 1$
(squares) and $\delta = 10$ (triangles). The exact values of D and G, which are referred to as D_e
and G_e, have been extrapolated from the linear fit of the results when $h \to 0$. The linear
behaviour of the absolute relative errors demonstrates that the results are in the asymp-
totic range of convergence and the method is first order accurate Salas (2006). The finest
physical grid size provides predictions which are accurate only within few percent. More
precisely, the largest error in D is of the order of 4% and is attained at $\delta = 10$ whereas
the one in G is 2% at $\delta = 0.1$. The error is mainly due to the physical and velocity
discretizations. As a matter of fact the statistical error associated with the finite sample
size used in the Monte Carlo evaluation of the collision integral does not affect the results
significantly. The standard deviation of D and G with respect to their averaged values in
stationary conditions, in fact, is negligible small. For instance, the standard deviation of
D at $\delta = 1$ from its long-term mean value is less than 0.05%. The grid resolution in the
physical and the velocity domains are the more accurate discretization compatible with
the GPU global memory constraint, i.e., $N_v = 8000$ and $N_r = 25600$ for $\delta = 0.1, 1$ and
$N_v = 5832$ and $N_r = 36864$ for $\delta = 10$. Therefore, in order to improve the accuracy of the
numerical solutions, we have adopted a nonuniform grid in the physical space. The lenghts
of the physical cells are uniformly stretched according to the relations $\Delta x_{i_x} = r_x \Delta x_{i_x - 1}$
and $\Delta y_{i_y} = r_y \Delta y_{i_y - 1}$ with progression ratios that depend on the rarefaction parameter,
$r_x = r_y = 0.990$ for $\delta = 10$ and $r_x = r_y = 0.996$ otherwise. The smaller cells are located
close to the upper corners of the cavity where severe gradients are anticipated. All the
results which follow have been obtained with this discretization. Table 1 compares the
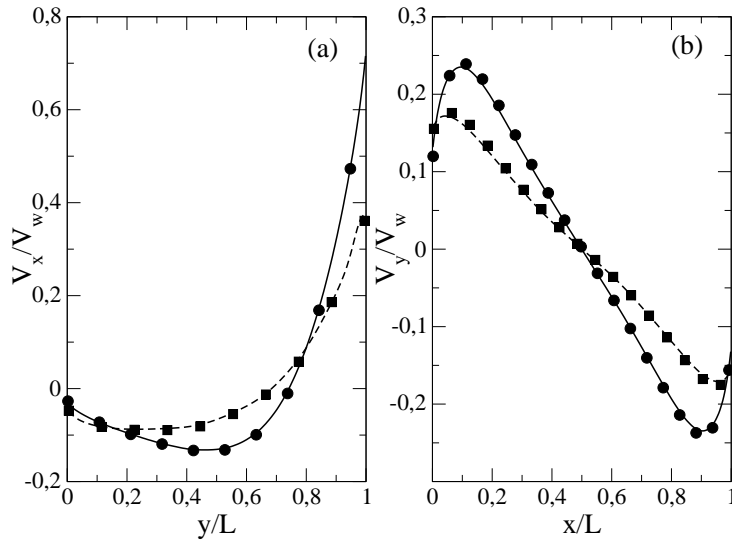
Figure 2: Profiles of the dimensionless (a) horizontal mean velocity along the vertical line crossing the center of the cavity, and (b) vertical mean velocity component along the horizontal line crossing the center of the main vortex. Solid and dashed lines: numerical solutions obtained by solving Eq. (7) with the semi-regular method for $\delta = 10$ and $\delta = 0.1$, respectively. Circles and squares: numerical solutions reported in Ref. Varoutis and Sharipov (2008) for $\delta = 10$ and $\delta = 0.1$, respectively. $N_v = 8000$, $r_v = 0.9994$, $N_t = 1024$, $N_r = 25600$, $r_x = r_y = 0.996$ for $\delta = 0.1$; $N_v = 5832$, $r_v = 0.970$, $N_t = 8192$, $N_r = 36864$, $r_x = r_y = 0.990$ for $\delta = 10$.

predictions of the mean stationary values of D and G with the extrapolated exact values, D_e and G_e, and the results reported in Ref. Varoutis and Sharipov (2008) where the linearized BGKW equation has been solved with a discrete velocity method. The accuracy of the numerical solution of the non-linear Boltzmann equation can be estimated to be within 2%. The agreement with the BGKW results is good, which is not unexpected. Since the velocity of the lid is small, in fact, the gas is in a weakly nonequilibrium state and the solution of the non-linear Boltzmann equation approaches the solution of the linearized BGKW equation. Figures 2a and 2b show the profiles of the dimensionless horizontal component of the velocity, V_x/V_w, along the vertical line crossing the center of the cavity and the dimensionless vertical component of the velocity, V_y/V_w, along the horizontal line crossing the center of the main vortex which forms in the cavity, respectively. Lines are the solutions of the non-linear Boltzmann equation, whereas symbols are the results reported in Ref. Varoutis and Sharipov (2008). The results refer to two different values of the rarefaction parameters: $\delta = 0.1$ (dashed lines and squares) and $\delta = 10$ (solid lines and circles). The agreement is very good.

Although the convergence analysis has been performed by referring to quantities evaluated in stationary conditions, it is important to point out that the proposed method of

δ	D	D_e	D^*	G	G_e	G^*
0.1	0.6712	0.6815	0.676-0.678	0.0955	0.0977	0.0973-0.0976
1	0.6266	0.6389	0.625-0.631	0.1017	0.1039	0.104-0.105
10	0.4096	0.4176	0.412-0.415	0.1427	0.1451	0.145-0.145

Table 1: Drag coefficient, D, and mean flow rate, G, versus the rarefaction parameter, δ. D_e and G_e represent the estimated exact values; D^* and G^* are reference values from Varoutis and Sharipov (2008).

solution provides accurate results in the unsteady regime as well. As an example, Figure 3 shows the evolution of D during the simulation for $\delta = 10$. It is important to stress that a similar result would be difficult to obtain with a particle method where computationally expensive ensemble averanging are needed to provide smooth macroscopic fields. The performance of the GPU implementation is compared against the single-threaded version running on the CPU by computing the speed-up factor $S = T_{\text{CPU}}/T_{\text{GPU}}$, where T_{CPU} and T_{GPU} are the times used by the CPU and GPU, respectively. Times are measured after initial setup, and do not include the time required to transfer data between the disjoint CPU and GPU memory spaces.

We analyse separately the streaming and the collision step, the latter comprising both the sampling and the collision kernel. Figure 4 reports the obtained speed-up data as a function of the number of spatial grid points N_r. The curve refers to $\delta = 1$. The speed-up grows rapidly with N_r and than levels up at about 450 if N_r approximately exceeds 10^4. This behavior is the result of the parallel set up of the collision step in N_r independent threads one for each cell of the physical space. As discussed below, the collision step absorbs most of the computational resources and its execution strongly affects the overall performances. As shown by the speed-up curve the GPU power is not fully exploited till the number of concurrent threads reaches a threshold. Beyond, the speed-up saturates and the computing time approximately behaves as a linear function of N_r. This behaviour is similar to the one reported in Refs. Elsen et al. (2008); Anderson et al. (2008). Figure 5 shows the relative time which is spent on the streaming step (dark bar) and on the collision step (light bar) as well as the total execution time (numbers over the bars) for $\delta = 1$. As expected, the collision step is more time consuming than the streaming step which takes at most 36% of overall computing time.

A strongly simplified evaluation of ideal performances of the streaming and collision step can be carry out as follows. A single application of the upwind scheme requires the execution of 11 floating point operations and 2.3 accesses to the global memory. The GPU delivers 715.4 GFLOPs but the transfer rate to/from the main memory is limited to 111.9 GB/s. Since in the case of streaming the ratio of number of floating point operations to the number of bytes accessed is low (11 : 9.2), it is reasonable to obtain the number of floating point operation per second from the transfer rate alone. Hence, the ideal number of GFLOPs can be obtained by assuming that 11 floating point operations will be executed in the time required to transfer 9.2 bytes from the main memory. Accordingly, this simple argument yields an ideal performance of the streaming step of 133 GFLOPs. A similar performance analysis can be applied to the collision step which encompasses both the sampling and collision kernel. In order to update the distribution function and com-
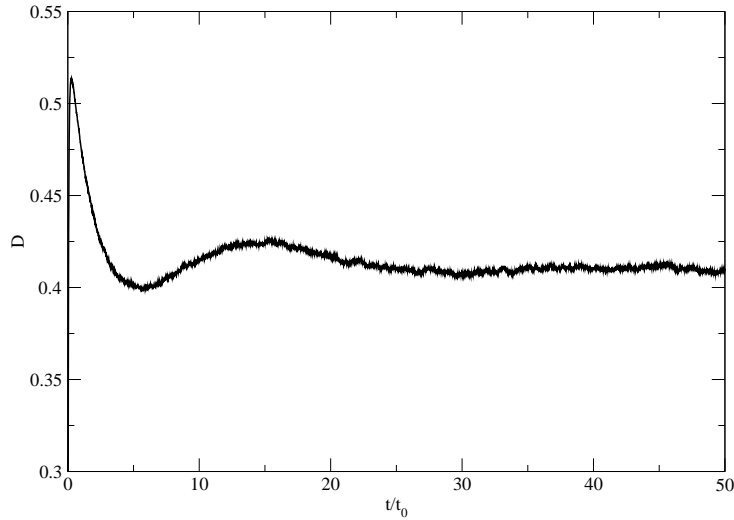
Figure 3: Drag coefficient over the moving wall, D,
versus dimensionless time, t/t_0. $\delta = 1$.

pute the macroscopic quantities of interest, the number of FLOPs that the GPU executes at any given time step and for each cell in the physical space is the sum of two contributions. The first is proportional to the number of velocity cells, N_v, and the second one is proportional to the number of collisions, N_t, used to evaluate the collision integral. The resulting number of FLOPs for each time step are of the order of $N_x\,N_y(80N_v + 12N_t)$. Likewise the number of bytes accesses to the global memory per time step is of the order of $N_x\,N_y(8N_{v\alpha}^3 + 64N_t)$ Arguments similar to those above lead to estimate an ideal performace of the collision step of about 174.7 GFLOP/s.

Timing the execution of the separate kernels and counting the number of associated floating point operations provides the real performance. The results are reported in Fig. 6 where GFLOPs are shown as a function of the number of grid points, N_r. Solid line with circles, dashed line with squares and dot-dashed line with triangles are the measured performances of the streaming step, the collision step and the overall code. respectively. It is possible to note that the performance of the streaming step grows with N_r and quickly levels at about 30 GFLOPs, approximately one third of the estimated ideal performance. The difference can be justified by observing that the real CUDA implementation of the finite difference scheme is not free from thread divergence and ancillary tasks whose effects can be evaluated with difficulty Micikevivius (2009). The collision step performance closely patterns the speed-up behavior, that is it rapidly grows in the range $N_r < 10^4$ and then levels up at about 140 GFLOP/s, reasonably close to the theoretical estimate. The collision kernel performs better than the streaming kernel due to its higher FLOP to memory operation ratio which, in turn, allows a more efficient use of GPU computing power. The absence of thread divergence is also a feature which positively affects performances.
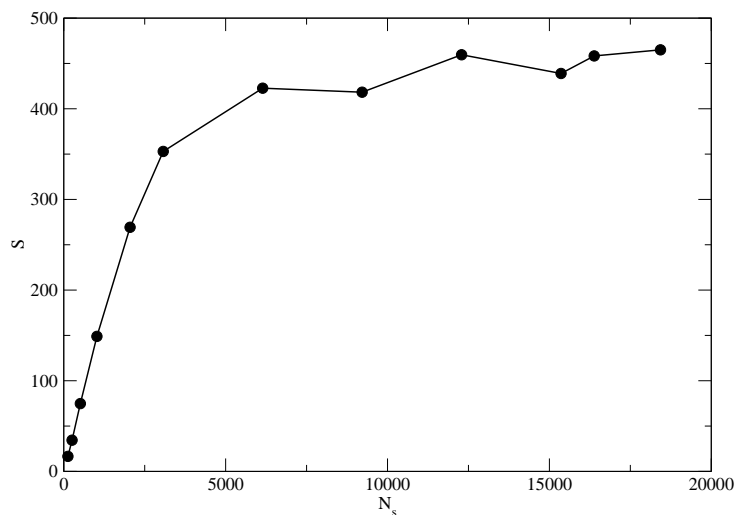
Figure 4: Overall speed-up, S, versus number of cells in the physical space, N_r; $\delta = 1$, $N_v = 8000$, $N_t = 6144$, $N_r = 25600$
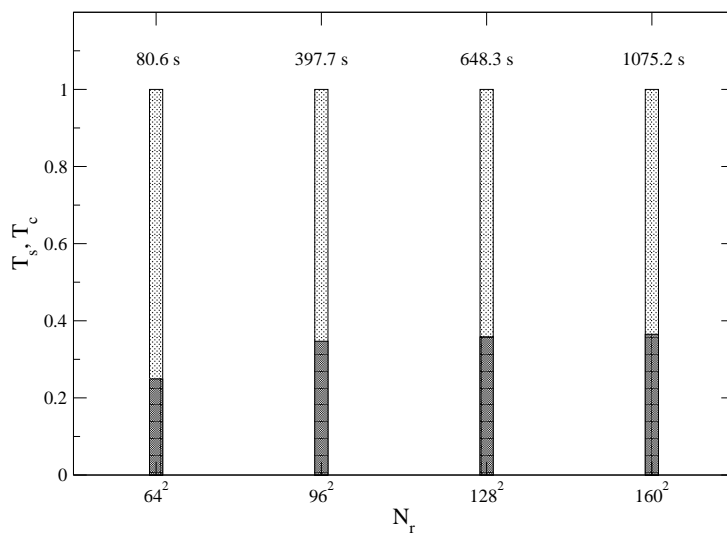

Figure 5: Relative time spent on the streaming step (dark bar) and on collision step (light bar). The numbers above the bars refer to the total execution time expressed in seconds. $\delta = 1$, $N_v = 8000$, $N_t = 6144$, $N_r = 25600$
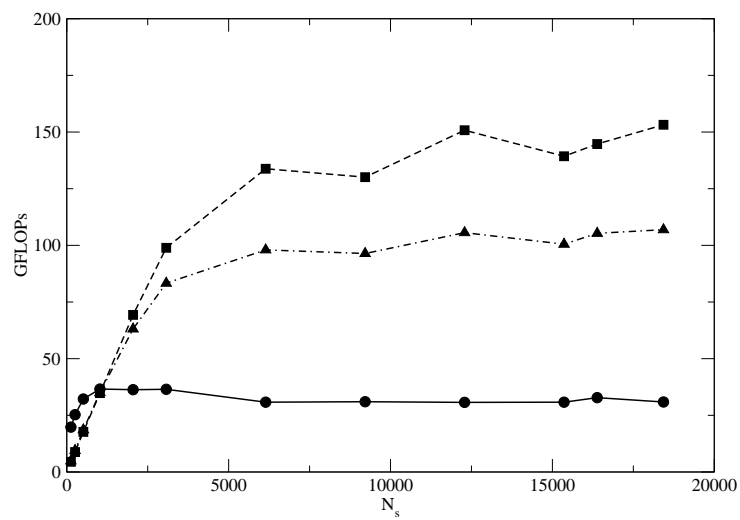
Figure 6: GFLOP/s versus the number of cells in the physical space, N_r. Solid line with circles: streaming step; dashed line with squares: collision step; dot and dashed line with triangles: overall code. $\delta = 1$, $N_v = 8000$, $N_t = 6144$, $N_r = 25600$.

6   Conclusions

In this notes we have described the possibility of exploiting the computational power of modern GPUs to simulate nonequilibrium rarefied gas flows. The full nonlinear Boltzmann equation has been solved by means of a semi-regular method which combines a finite difference discretization of the free-streaming term with a Monte Carlo evaluation of the collision integral. This method of solution is ideally suited for the SIMD-like architecture provided by the commercially available GPUs. The two dimensional driven cavity flow has been used as a benchmark problem. The results lead to concluding that the porting of the sequential code onto GPUs allows a reduction of the computing time of two orders of magnitude, being the observed speed-up as high as 400. Although the test problem examined here has clearly shown that the size of physical memory is the main obstacle toward the application to complex two or three-dimensional flows, the numerical method described above can be reformulated as a less memory particle scheme in many ways. Hence, the present work and results are a first step toward the construction of a more flexible and efficient method for the numerical solution of kinetic equations.

7   Acknowledgments

References

Anderson, J. A., Lorenz, C. D., and Travesset, A. (2008). General purpose molecular dynamics simulations fully implemented on graphics processing units. *J. Comp. Phys.*, 227:5342–5359.

Aristov, A. A. (2001). *Direct Methods for Solving the Boltzmann Equation and Study of Nonequilibrium Flows.* Springer-Verlag, New York.

Aristov, V. V. and G., T. F. (1980). *U.S.S.R. Comput. Math. Phys.*, 20:208–225.

Baker, L. L. and Hadjiconstantinou, N. G. (2008). Variance-reduced monte carlo solutions of the boltzmann equation for low-speed gas flows: A discontinuous galerkin formulation. *Int. J. Numer. Meth. Fluids*, 58:381–402.

Bird, G. A. (1994). *Molecular Gas Dynamics and the Direct Simulation of Gas Flows.* Oxford University Press, Oxford.

Cercignani, C. (1988). *The Boltzmann Equation and Its Applications.* Springer-Verlag, New York.

Chapman, S. and Cowling, T. G. (1990). *The mathematical theory of non-uniform gases.* Cambridge University Press.

Elsen, E., LeGresley, P., and Darve, E. (2008). Large calculation of the flow over a hypersonic vehicle using a gpu. *J. Comp. Phys.*, 227:10148–10161.

F., T. (2005). Direct numerical solution of the boltzmann equation. In *Proceedings of the 24th International Symposium on Rarefied Gas Dynamics*, volume 762 of *AIP Conference Proceedings*, pages 677–685. AIP.

Frezzotti, A. (1991). Numerical study of the strong evaporation of a binary mixture. *Fluid Dynamics Research*, 8:175–187.

Frezzotti, A. (2007). A numerical investigation of the steady evaporation of a polyatomic gas. *Eur. J. Mech. B: Fluids*, 26:93–104.

Frezzotti, A., Gibelli, L., and Lorenzani, S. (2005). Mean field kinetic theory description of evaporation of a fluid into vacuum. *Phys. Fluids*, 17:012102–12.

Frezzotti, A., P., G. G., and Gibelli, L. (2010). Solving model kinetic equations on gpus. submitted.

Homolle, T. M. M. and Hadjiconstantinou, N. G. (2007). A low-variance deviational simulation monte carlo for the boltzmann equation. *J. Comput. Phys.*, 226:2341–2358.

Micikevivius, P. (2009). 3D finite difference computation on GPUs using CUDA. In *Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*, volume 383 of *ACM International Conference Proceeding Series*, pages 79–84. ACM.

NVIDIA Corporation (2008). *NVIDIA CUDA Programming Guide.* http://www.nvidia.com/CUDA, version 2.0 edition.

Salas, M. D. (2006). Some observations on grid convergence. *Comp. & Fluids*, 35:688–692.

Varoutis, S.and Valougeorgis, D. and Sharipov, F. (2008). Application of the integro-moment method to steady-state two-dimensional rarefied gas flows subject to boundary induced discontinuities. *J. Comput. Phys.*, 227:6272–6287.

Wagner, W. (2008). Deviational particle monte carlo for the boltzmann equation. *Monte Carlo Methods and Applications*, 14:191–268.

8   Appendix: CUDA pseudo-codes

Algorithm 1 GPU pseudo-code of the two-dimensional streaming kernel

---

Require: Cu_x is Courant number along x direction
Require: Cu_y is Courant number along y direction
Require: B_x is the number of threads in x direction
Require: B_y is the number of threads in y direction
Require: t_x is the index of the thread in x direction
Require: t_y is the index of the thread in y direction
Require: f_{sh} is a matrix of dimensions $(B_x + 1) \times (B_y + 1)$ in the shared memory

  1: **for** $I_{by} = N_y/B_y - 1$ to 0 **do**
  2:     **for** $I_{bx} = N_x/B_x - 1$ to 0 **do**
  3:         $i_x \leftarrow t_x + B_x I_{bx}$
  4:         $i_y \leftarrow t_y + B_y I_{by}$
  5:         $f_{\mathrm{sh}}(t_x + 1, t_y + 1) \Leftarrow f^n_{i_x,i_y;\boldsymbol{j}}$
  6:         **if** $t_y == 0$ **then**
  7:             **if** $i_y - 1 < 0$ **then**
  8:                 $f_{\mathrm{sh}}(t_x, t_y) \leftarrow$ boundaryFlux
  9:             **else**
10:                 $f_{\mathrm{sh}}(t_x, t_y) \Leftarrow f^n_{i_x,i_y-1,\boldsymbol{j}}$
11:             **end if**
12:         **end if**
13:         **if** $t_x == 0$ **then**
14:             **if** $i_x - 1 < 0$ **then**
15:                 $f_{\mathrm{sh}}(t_x, t_y) \leftarrow$ boundaryFlux
16:             **else**
17:                 $f_{\mathrm{sh}}(t_x, t_y) \Leftarrow f^n_{i_x-1,i_y,\boldsymbol{j}}$
18:             **end if**
19:         **end if**
20:         syncthreads
21:         $f_{\mathrm{rg}} \leftarrow (1 - \mathrm{Cu}_x - \mathrm{Cu}_y)f_{\mathrm{sh}}(t_x + 1, t_y + 1) + \mathrm{Cu}_y\, f_{\mathrm{sh}}(t_x, t_y + 1) + \mathrm{Cu}_x\, f_{\mathrm{sh}}(t_x + 1, t_y)$
22:         $f_{\mathrm{rg}} \Rightarrow \tilde{f}^{n+1}_{i_x,i_y,\boldsymbol{j}}$
23:         $I_{bx} \leftarrow I_{bx} - 1$
24:     **end for**
25:     $I_{by} \leftarrow I_{by} - 1$
26: **end for**

---

8 APPENDIX: CUDA PSEUDO-CODES

Algorithm 2 GPU pseudo-code of the sampling kernel

Require: i is the global index of thread

1: $\boldsymbol{v} \leftarrow$ BoxMulller
2: $\boldsymbol{v}_1 \leftarrow$ BoxMulller
3: $\hat{\boldsymbol{k}} \leftarrow$ UnitSphere
4: $\boldsymbol{v}_r \leftarrow \boldsymbol{v}_1 - \boldsymbol{v}$
5: $\boldsymbol{v}^* \leftarrow \boldsymbol{v} + (\boldsymbol{v}_r \cdot \hat{\boldsymbol{k}})\hat{\boldsymbol{k}}$
6: $\boldsymbol{v}_1^* \leftarrow \boldsymbol{v}_1 - (\boldsymbol{v}_r \cdot \hat{\boldsymbol{k}})\hat{\boldsymbol{k}}$
7: cells$(\boldsymbol{v}) \Rightarrow I(i)$
8: cells$(\boldsymbol{v}_1) \Rightarrow I_1(i)$
9: cells$(\boldsymbol{v}^*) \Rightarrow I^*(i)$
10: cells$(\boldsymbol{v}_1^*) \Rightarrow I_1^*(i)$
11: $g_{\boldsymbol{j}} \leftarrow dt\pi d^2 n_0^2 (\Delta \mathcal{V}_{\boldsymbol{j}} \Phi_{\boldsymbol{j}})^{-1} |\boldsymbol{v}_r \cdot \hat{\boldsymbol{k}}|$
12: $g_{\boldsymbol{j}_1} \leftarrow dt\pi d^2 n_0^2 (\Delta \mathcal{V}_{\boldsymbol{j}_1} \Phi_{\boldsymbol{j}_1})^{-1} |\boldsymbol{v}_r \cdot \hat{\boldsymbol{k}}|$
13: $g_{\boldsymbol{j}^*} \leftarrow dt\pi d^2 n_0^2 (\Delta \mathcal{V}_{\boldsymbol{j}^*} \Phi_{\boldsymbol{j}^*})^{-1} |\boldsymbol{v}_r \cdot \hat{\boldsymbol{k}}|$
14: $g_{\boldsymbol{j}_1^*} \leftarrow dt\pi d^2 n_0^2 (\Delta \mathcal{V}_{\boldsymbol{j}_1^*} \Phi_{\boldsymbol{j}_1^*})^{-1} |\boldsymbol{v}_r \cdot \hat{\boldsymbol{k}}|$
15: $g_{\boldsymbol{j}} \Rightarrow C(i)$
16: $g_{\boldsymbol{j}_1} \Rightarrow C_1(i)$
17: $g_{\boldsymbol{j}^*} \Rightarrow C^*(i)$
18: $g_{\boldsymbol{j}_1^*} \Rightarrow C_1^*(i)$

Algorithm 3 GPU pseudo-code of the collision kernel

Require: i is the global x index of the thread inside the grid
Require: j is the global y index of the thread inside the grid

1: **for all $\boldsymbol{j}$ do**
2:    $h \Leftarrow \tilde{h}_{i,\boldsymbol{j}}^{n+1}$
3:    $\rho \leftarrow \rho + \Phi_{0,\boldsymbol{j}}\, h$
4:    $\boldsymbol{u} \leftarrow \boldsymbol{u} + \boldsymbol{v_j}\, \Phi_{0,\boldsymbol{j}}\, h$
5:    $e \leftarrow e + |\boldsymbol{v_j}|^2\, \Phi_{0,\boldsymbol{j}}\, h$
6: **end for**
7: **for $m = 1$ to N_t do**
8:    $h \Leftarrow \tilde{h}_{i,I(m)}^{n+1}$
9:    $h_1 \Leftarrow \tilde{h}_{i,I_1(m)}^{n+1}$
10:    $h^* \Leftarrow \tilde{h}_{i,I^*(m)}^{n+1}$
11:    $h_1^* \Leftarrow \tilde{h}_{i,I_1^*(m)}^{n+1}$
12:    $g \leftarrow h + h_1 + \epsilon\, h\, h_1$
13:    $h \leftarrow h - C(m)\, g$
14:    $h_1 \leftarrow h_1 - C_1(m)\, g$
15:    $h^* \leftarrow h^* + C^*(m)\, g$
16:    $h_1^* \leftarrow h_1^* + C_1^*(m)\, g$
17:    $h \Rightarrow \tilde{h}_{i,I(m)}^{n+1}$
18:    $h_1 \Rightarrow \tilde{h}_{i,I_1(m)}^{n+1}$
19:    $h^* \Rightarrow \tilde{h}_{i,I^*(m)}^{n+1}$
20:    $h_1^* \Rightarrow \tilde{h}_{i,I_1^*(m)}^{n+1}$
21: **end for**
22: **for all $\boldsymbol{j}$ do**
23:    $h \Leftarrow \tilde{h}_{i,\boldsymbol{j}}^{n+1}$
24:    $a_{11} \leftarrow a_{11} + \Phi_{0,\boldsymbol{j}}\, h$
25:    $a_{12} \leftarrow a_{12} + \boldsymbol{v_j}\, \Phi_{0,\boldsymbol{j}}\, h$
26:    $a_{13} \leftarrow a_{13} + |\boldsymbol{v_j}|^2\, \Phi_{0,\boldsymbol{j}}\, h$
27:    $\vdots$
28:    // others moments of the distribution function
29:    $\vdots$
30: **end for**
31: $[A, \vec{B}, C] = \text{linearSolver}(n, \boldsymbol{u}, e, a_{11}, a_{12}, a_{13}, \ldots)$
32: **for all $\boldsymbol{j}$ do**
33:    $h \Leftarrow \tilde{h}_{i,\boldsymbol{j}}^{n+1}$
34:    $h \leftarrow 1/\epsilon((1 + \epsilon h)(1 + A + \vec{B} \cdot \boldsymbol{v_j} + C\boldsymbol{v_j}^2 - 1)$
35:    $h \Rightarrow h_{i,\boldsymbol{j}}^{n+1}$
36:    $\vdots$
37:    // compute macroscopic quantities
38:    $\vdots$
39: **end for**