DIPARTIMENTO DI MATEMATICA
"Francesco Brioschi"
POLITECNICO DI MILANO

Solving Model Kinetic Equations on GPUs

Frezzotti, A.; Ghiroldi, G.P.; Gibelli, L.

Collezione dei *Quaderni di Dipartimento*, numero **QDD 105**
Inserito negli *Archivi Digitali di Dipartimento* in data 20-7-2011

Piazza Leonardo da Vinci, 32 - 20133 Milano (Italy)

Solving Model Kinetic Equations on GPUs

A. Frezzotti, G. P. Ghiroldi, L. Gibelli *

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

Abstract

We present an algorithm specifically tailored for solving model kinetic equations onto Graphics Processing Units (GPUs). The efficiency of the algorithm is demonstrated by solving the one-dimensional shock wave structure problem and the two-dimensional low Mach number driven cavity flow. Computational results show that it is possible to cut down the computing time of the sequential codes of two order of magnitude. The algorithm can be easily extended to more general collision models.

Key words: Microflows, BGKW equation, regular methods, parallel algorithms, Graphics Processing Units, CUDA™ programming model
PACS: 02.70.Bf, 47.45.Ab, 51.10.+y

1 Introduction

A recent trend emerging in computational physics stems from the availability of low cost general purpose Graphics Processing Units (GPUs). GPUs have been used to accelerate CPU critical applications such as simulations of hypersonic flows [1], magnetized plasma [2] and molecular dynamics [3]. However, no applications to kinetic theory of gases seem to have been considered yet. Kinetic theory of gases deals with non-equilibrium gas flows which 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 [4,5]. The dynamics of dilute (or rarefied) gas flows is governed by the Boltzmann equation [6] which takes the form

* Corresponding author.
Email addresses: aldo.frezzotti@polimi.it (A. Frezzotti), gian.ghiroldi@mail.polimi.it (G. P. Ghiroldi), livio.gibelli@polimi.it (L. Gibelli).

$$\frac{\partial f}{\partial t} + \boldsymbol{v} \circ \nabla_{\boldsymbol{x}} f + \frac{1}{m} \nabla_{\boldsymbol{v}} \circ (\boldsymbol{F} f) = \mathcal{C}(f, f) \tag{1}$$

$$\mathcal{C}(f, f) = \int [f(\boldsymbol{x}, \boldsymbol{v}^*|t) f(\boldsymbol{x}, \boldsymbol{v}_1^*|t) - f(\boldsymbol{x}, \boldsymbol{v}|t) f(\boldsymbol{x}, \boldsymbol{v}_1|t)] \, \sigma(\|\boldsymbol{v}_r\|, \hat{\boldsymbol{k}} \circ \boldsymbol{v}_r) \|\boldsymbol{v}_r\| d\boldsymbol{v}_1 d^2\hat{\boldsymbol{k}} \tag{2}$$

when written for a gas composed by a single monatomic species whose atoms have mass m. In Eqs. (1,2), $f(\boldsymbol{x}, \boldsymbol{v}|t)$ denotes the distribution function of atomic velocities $\boldsymbol{v}$ at spatial location $\boldsymbol{x}$ and time t, $\boldsymbol{F}(\boldsymbol{x}, \boldsymbol{v}|t)$ is an assigned external force field, whereas $\mathcal{C}(f, f)$ gives the collisional rate of change of f at the phase space point $(\boldsymbol{x}, \boldsymbol{v})$ at time t. As is clear from Eq. (2), $\mathcal{C}(f, f)$ is a non-linear functional of f, whose precise structure depends on the assumed atomic interaction forces through the differential cross section $\sigma(\|\boldsymbol{v}_r\|, \hat{\boldsymbol{k}} \circ \boldsymbol{v}_r)$. The dynamics of binary encounters determines σ as a function of the modulus $\|\boldsymbol{v}_r\|$ of the relative velocity $\boldsymbol{v}_r = \boldsymbol{v}_1 - \boldsymbol{v}$ of two colliding atoms and of the orientation of the unit impact vector $\hat{\boldsymbol{k}}$ with respect to $\boldsymbol{v}_r$ [7]. Moreover, since momentum and energy are conserved, the pre-collisional velocities $\boldsymbol{v}^*$ and $\boldsymbol{v}_1^*$ are obtained from $\boldsymbol{v}, \boldsymbol{v}_1$ and $\hat{\boldsymbol{k}}$ through the relationships

$$\boldsymbol{v}^* = \boldsymbol{v} + (\boldsymbol{v}_r \circ \hat{\boldsymbol{k}}) \hat{\boldsymbol{k}} \tag{3}$$

$$\boldsymbol{v}_1^* = \boldsymbol{v}_1 - (\boldsymbol{v}_r \circ \hat{\boldsymbol{k}}) \hat{\boldsymbol{k}} \tag{4}$$

For illustration purposes, it is worth mentioning that the collision integral $\mathcal{C}(f, f)$ simplifies to

$$\mathcal{C}(f, f) = \frac{d^2}{2} \int [f(\boldsymbol{x}, \boldsymbol{v}^*|t) f(\boldsymbol{x}, \boldsymbol{v}_1^*|t) - f(\boldsymbol{x}, \boldsymbol{v}|t) f(\boldsymbol{x}, \boldsymbol{v}_1|t)] \, |\hat{\boldsymbol{k}} \circ \boldsymbol{v}_r| d\boldsymbol{v}_1 d^2\hat{\boldsymbol{k}} \tag{5}$$

for a dilute gas of hard spheres of diameter d.

Obtaining numerical solutions of the Boltzmann equation for realistic flow conditions is a challenging task because the unknown function depends, in principle, on seven variables. Moreover, the computation of $\mathcal{C}(f, f)$ requires the approximate evaluation of a fivefold integral. Numerical methods for rarefied gas dynamics studies 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 [8]. 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 Eqs. (1,2). The method can be easily extended to deal with mixtures of

chemically reacting polyatomic species [8] and to dense fluids [9]. Macroscopic flow properties are usually obtained by time averaging particle properties. In steady flows simulations, the averaging time can be long enough to obtain accurate flow simulations by a relatively small number of particles. 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 [10,11]. However, in simulating high frequency unsteady flows, typical of microfluidics application to MEMS [4], 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 on the l.h.s of Eq. (1). 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 [12,13] whereas deterministic integration schemes are used in regular methods [14]. 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 [15] but the necessity to store additional variables associated with internal degrees of freedom further limits the applications to multi-dimensional flows. In spite of the drawbacks listed above, the direct solution of the Boltzmann equation by semi-regular or regular methods is a valid alternative to particle schemes in studying unsteady or low speed flows. Actually, when the deviation from equilibrium is small a limited number of grid points in the velocity space is sufficient to provide accurate and noise free approximations of f, therefore simulations of multi-dimensional flows are feasible on modern personal computers in a wide range of Knudsen numbers [16].

An important feature of kinetic equations for dilute gases is the locality of the collision term; the collisional rate of change $\mathcal{C}(f, f)$ at the spatial location $\boldsymbol{x}$ is completely determined by $f(\boldsymbol{x}, \boldsymbol{v}|t)$. Hence, the time consuming evaluation of the collision integral can be concurrently executed at each spatial grid point on parallel computers. As shown below, the numerical algorithm associated with regular or semi-regular methods is ideally suited for the parallel architecture provided by commercially available GPUs. Hence, the aim of the paper is to describe an efficient algorithm specifically tailored for solving kinetic equations onto GPUs using CUDA™ programming model [17]. The efficiency of the algorithm is assessed by solving the classical one-dimensional shock wave

structure and a low speed 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 magnitude by a proper reformulation of the algorithm to be executed on a GPU.
In order to make the algorithm development easier, the computations presented here have been performed by replacing $\mathcal{C}(f,f)$ with its simpler BGKW approximation [6]. This choice eliminates the intricacies connected with the numerical evaluation of the Boltzmann collision integral and allows easier identification of bottlenecks and optimization strategies. As will be shown in a separate paper, the full Boltzmann equation can be solved within the same general algorithmic framework by adopting a Monte Carlo quadrature method.
This paper is organized as follows. Section 2 is devoted to a concise description of the mathematical model and the adopted numerical method. In Section 3 the key aspects of the GPU hardware architecture and the adopted parallelization strategies are briefly described. Sections 4 and 5 are devoted to the description of the test problems and the discussion of the results. Concluding remarks are presented in Section 6.

2 Theoretical and numerical background

2.1 Mathematical formulation

Both from the theoretical and computational point of view, it is often convenient to replace the full Boltzmann equation with a model equation having a simplified collision term. In the kinetic model proposed by Bhatnagar Gross and Krook [18] and independently by Welander [19], $\mathcal{C}(f,f)$ is replaced by the expression $\nu\,(\Phi - f)$. Accordingly, Eq. (1) is turned into the following kinetic equation:

$$\frac{\partial f}{\partial t} + \boldsymbol{v} \circ \nabla_{\boldsymbol{x}} f + \frac{1}{m} \nabla_{\boldsymbol{v}} \circ (\boldsymbol{F} f) = \nu\,(\Phi - f) \tag{6}$$

In Eq. (6) ν is the collision frequency, whereas Φ is the local equilibrium Maxwellian distribution function given by the expression

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

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

$$n = \int f d\boldsymbol{v} \qquad \mathbf{V} = \frac{1}{n} \int f \boldsymbol{v} d\boldsymbol{v} \qquad T = \frac{1}{3Rn} \int f(\boldsymbol{v} - \mathbf{V})^2 d\boldsymbol{v} \tag{8}$$

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 right hand side.
As is well known, the BGKW model predicts an incorrect value of the Prandtl number in the hydrodynamic limit [6]. Hence, ν can be adjusted to obtain either the correct viscosity or heat conductivity, but not both. If viscosity is selected, then ν is given by the following expression:

$$\nu = \frac{nRT}{\mu} \tag{9}$$

being $\mu(T)$ the gas viscosity. Although this paper deals only with the BGKW kinetic model equation, it should be observed that the algorithm described below can be extended to deal with more general kinetic equations by introducing minimal modifications. For instance, the right hand side of Eq. (6) could be replaced by the ellipsoidal statistical model [6], which retains the simplicity of the BGKW kinetic model but, at the same time, provides the correct transport properties in the hydrodynamic limit.

2.2 Outline of the numerical method

In view of the exploratory nature of the present work, Eq. (6) has been solved by a simple numerical method which will be illustrated on a spatially one-dimensional problem. The extension to two or three-dimensional geometries is straightforward.
In absence of external forces and in one-dimensional slab geometry, Eq. (6) takes the form:

$$\frac{\partial f}{\partial t} + v_x \frac{\partial f}{\partial x} = \nu \left(\Phi - f\right) \tag{10}$$

where x is the single spatial coordinate and v_x the x-component of the velocity vector $\boldsymbol{v} = (v_x, v_y, v_z)$. The spatial domain is a finite interval of the real axis, divided into N_x cells of equal size Δx. 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 $f_{i,\boldsymbol{j}}(t) = f(x(i), v_x(j_x), v_y(j_y), v_z(j_z)|t)$, being $x(i), v_x(j_x), v_y(j_y), v_z(j_z)$ the values of the spatial coordinate and velocity components in the center of the phase space cell $(i, \boldsymbol{j})$ and $\boldsymbol{j} = (j_x, j_y, j_z)$.
The algorithm that advances $f_{i,\boldsymbol{j}}(t)$ to $f_{i,\boldsymbol{j}}(t + \Delta t)$ is constructed by time-splitting the evolution operator into a free streaming step, in which the r.h.s. of Eq. (10) 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 $f^n_{i,\boldsymbol{j}} = f_{i,\boldsymbol{j}}(t_n)$ at time level t_n is advanced to its value $f^{n+1}_{i,\boldsymbol{j}} = f_{i,\boldsymbol{j}}(t_{n+1})$ at time level $t_{n+1} = t_n + \Delta t$ by first computing an intermediate value $\tilde{f}^{n+1}_{i,\boldsymbol{j}}$ from the free streaming equation

$$\frac{\partial f}{\partial t} + v_x \frac{\partial f}{\partial x} = 0 \tag{11}$$

Eq. (11) is solved by a simple first order explicit upwind scheme

$$\tilde{f}^{n+1}_{i,\boldsymbol{j}} = \begin{cases} \left(1 - \dfrac{v_x(j_x)\Delta t}{\Delta x}\right) f^n_{i,\boldsymbol{j}} + \dfrac{v_x(j_x)\Delta t}{\Delta x} f^n_{i-1,\boldsymbol{j}} & v_x(j_x) \geq 0 \\ \left(1 + \dfrac{v_x(j_x)\Delta t}{\Delta x}\right) f^n_{i,\boldsymbol{j}} - \dfrac{v_x(j_x)\Delta t}{\Delta x} f^n_{i+1,\boldsymbol{j}} & v_x(j_x) < 0 \end{cases} \tag{12}$$

where the absolute value of the Courant number $C = v_x(j_x)\Delta t/\Delta x$ must be less than or equal to 1 to ensure the stability of the numerical method. In the vast majority of applications, the largest velocities in the velocity space grid, associated with the tail of f, limit the time step to a value considerably smaller than the one dictated by accuracy requirements. Such limitation can be easily removed by noting that the exact solution of Eq. (11) is

$$f(x, \boldsymbol{v}, t + \Delta t) = f(x - v_x \Delta t, \boldsymbol{v}, t) \tag{13}$$

Thus, for each molecular velocity and for, $v_x(j_x)$, the value of the distribution function in the cell $(i, \boldsymbol{j})$ of the phase space can be obtained by first translating the distribution function by a number of cells equal to the integer part of the Courant number, $[C]$, and then applying expressions (12) for the residual time step advancement.
After completing the free streaming step, macroscopic variables n_i, $\boldsymbol{V}_i$ and T_i are computed at each spatial grid point and $f^{n+1}_{i,\boldsymbol{j}}$ is finally obtained by solving the homogeneous relaxation equation

$$\frac{\partial f}{\partial t} = \nu(\Phi - f) \tag{14}$$

Since n, $\boldsymbol{V}$ and T are conserved during homogeneous relaxation, Eq. (14) can be exactly solved to obtain

$$f^{n+1}_{i,\boldsymbol{j}} = [1 - \exp(-\tilde{\nu}_i \Delta t)]\, \tilde{\Phi}_{i,\boldsymbol{j}} + \exp(-\tilde{\nu}_i \Delta t) \tilde{f}^{n+1}_{i,\boldsymbol{j}} \tag{15}$$

in each cell $(i, \boldsymbol{j})$ of the phase space. It should be observed that the density, bulk velocity and temperature obtained from the discretized Maxwellian distribution function $\tilde{\Phi}_{i,\boldsymbol{j}}$ are not exactly equal to $\tilde{n}_i$, $\tilde{\boldsymbol{V}}_i$ and $\tilde{T}_i$. To ensure exact conservation of mass momentum and energy, the discretized $\tilde{\Phi}_{i,\boldsymbol{j}}$ should be computed from Eq. (7) by using effective values $\bar{n}_i$, $\bar{\boldsymbol{V}}_i$ and $\bar{T}_i$ which are obtained by requiring that the moments of the discretized Maxwellian coincide

with $\tilde{n}_i$, $\tilde{\boldsymbol{V}}_i$ and $\tilde{T}_i$ [12]. However, numerical tests have shown that calculating effective local values $\bar{n}_i$, $\bar{\boldsymbol{V}}_i$ and $\bar{T}_i$ to force exact conservation of collisional invariants did not affect appreciably the solutions of the problems described below, not even for coarse velocity space grids.
Since neither the streaming step nor the collision step are subject to stability constraints, the choice of the time step is determined only by considerations related to the accuracy of the numerical solution. As a rule of thumb, Δt has been determined by requiring that, at each spatial node, the distance traveled in one time step by atoms flying at the mean velocity V is less than the mean free path $\lambda = \sqrt{2RT}/\nu$, the natural scale of spatial gradients. The condition stated above leads to the inequality $\nu\Delta t < \sqrt{2RT}/V$. It should also be observed that a second time step limitation comes from the relaxation step since $\nu\Delta t$ should be less than one to avoid distorting relaxation processes. Since the ratio $\sqrt{2RT}/V$ is essentially the reciprocal of the Mach number, it is clear that the time step is limited by the first condition in high speed flows, whereas the second one dictates the time step to be used in low speed flows. Finally, it worth mentioning that the described numerical method has been designed for unsteady flow computations. The algorithm will be tested on steady problems just because accurate numerical solutions were already available. However, it is possible to consider extending the parallel GPU implementations that will be discussed in the next section to either fully implicit time advancing schemes or iterative schemes, based on the steady form of the kinetic equation, which could obtain steady solution by smaller effort.

3 Parallelization strategies in CUDA™

Splitting the time evolution of f into a free streaming and a collision step allows breaking each one into a large number of independent calculations which can be concurrently executed. As a matter of fact, each atomic group with velocity $v_x(j_x)$ is independently transported according to Eq. (11), during the free streaming. Similarly, the local structure of the collision operator allows its concurrent evaluation at each spatial grid node. The intrinsic parallelism of the algorithm can be exploited by the development of a program to be executed on a NVIDIA®GPU consisting of a set of multiprocessors with a SIMD-like architecture. During each clock cycle, each core of the multiprocessor array executes the same instruction but operates on different data. The program is organized into a serial program which runs on the host CPU and one or more kernels which define the computation to be concurrently performed. Kernels are executed by threads, i.e. by program units which can be independently executed and assigned to GPU individual processors for concurrent execution. Threads are organized into a two-level grid and block hierarchy which mirrors GPU architecture. One may think of a grid as the GPU

itself, a block as multi-processor of the GPU and a thread as a single core in the multi-processor. GPU memory has also a hierarchical structure whose understanding is of fundamental importance for code optimization. Actually, data transfer from and to the global memory is the rate limiting step in most applications particularly when the memory access are not synchronized with threads activity (non-coalescent). [17]. Each thread may access private, low latency memory registers. Threads belonging to the same block are allowed to synchronize with each other and to share data through a shared memory which is as fast as registers. However, threads from different blocks may coordinate only via operations in the slower global memory. Threads concurrently execute the same instructions on different data unless a conditional statement in the code leads them to follow different execution paths. In this case thread divergence occurs and the different branches are executed sequentially. Then, the total run time is the sum of all the branches. Threads divergence and re-convergence are managed in hardware but have a negative impact on performances. Figure 1 summarizes the organization of the grid of threads and the relationship between threads and available memory spaces.
Following the algorithm structure described above, the code is organized into a host program, which deals with all memory management and other setup tasks, and two kernels running on the GPU. The first kernel performs the streaming step whereas the second kernel performs the collision step.
Figure 2 illustrates the parallelization strategy used for implementing the streaming kernel. Each block of the streaming kernel is associated with a velocity $v_x(j_x)$ and is composed by a one dimensional grid of threads of dimension N_s, having each thread associated with one cell of the physical space. When a block becomes active, each thread loads one element of the distribution function from global memory, stores it into shared memory, updates its value according to the CFL-free upwind scheme and then saves it back to the global memory. This procedure is repeated sequentially N_x/N_s times. In order to obtain a coalesced access to the global memory and hence maximize the memory bandwidth [17], values of the discretized distribution function in spatially adjacent cells are stored in contiguous memory locations.
Figure 3 illustrates the parallelization strategy used for implementing the collision kernel. The collision or relaxation step is concurrently performed on the whole spatial domain by a kernel whose threads are associated with a spatial cell. Each thread operates on the distribution function at a given spatial location by computing first the local values of density, velocity and temperature which define the local Maxwellian, then the distribution function is updated according to Eq. (15). It is worth observing that alternative parallelization schemes are possible. For instance, the relaxation step could have been organized by assigning a block of threads to each spatial cell, having each thread associated to a a phase space cell where f changes according to Eq. (15). However, the choice presented here seems to possess a few advantages. Actually, the concurrent execution of relaxation in the phase space can be performed only after the evaluation of macroscopic quantities at each spatial location.

Such evaluation is more naturally performed by the adopted scheme which avoids using parallel reduction algorithms [1] by associating threads to spatial cells. Therefore, it seems reasonable to keep the same thread association and perform the relaxation step within the kernel computing macroscopic variables. Such strategy also reduces the number of data transfer from and to the global memory.
The computations described 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 "-O3".

4 Shock wave

4.1 Formulation of the problem

The propagation of a planar shock wave is a classical application of kinetic equations and it is a rather natural choice as a benchmark problem because of the considerable number of previous studies [20,21]. Moreover, the numerical treatment of the problem is made particularly simple by the boundary conditions which assign prescribed equilibrium states to the upstream and downstream distribution function. In the wave front reference frame, the stationary flow field is assumed to be governed by the one-dimensional steady BGKW equation

$$v_x \frac{\partial f}{\partial x} = \nu(\Phi - f) \tag{16}$$

x being the spatial coordinate which spans the direction normal to the (planar) wave front. It is further assumed that, far from the wave front, the distribution function $f(x, \boldsymbol{v})$ satisfies the boundary conditions

$$\lim_{x \to \mp\infty} f(x, \boldsymbol{v}) = \Phi^{\mp}(\boldsymbol{v}) = \frac{n^{\mp}}{(2\pi R T^{\mp})^{3/2}} \exp\left[-\frac{(v_x - V^{\mp})^2 + v_y^2 + v_z^2}{2RT^{\mp}}\right] \tag{17}$$

where $n^{\mp}$, $V^{\mp}$ and $T^{\mp}$ are the upstream and downstream values of number density, velocity and temperature, respectively. The parameters of the equilibrium states specified by Eq. (17) are connected by the Rankine-Hugoniot relationships

$$\frac{V^-}{V^+} = \frac{n^+}{n^-} = \frac{4(M^-)^2}{(M^-)^2+3} \qquad \frac{T^+}{T^-} = \frac{[5(M^-)^2-1]\,[(M^-)^2+3]}{16(M^-)^2} \tag{18}$$

In Eqs. (18) M^- denotes the upstream infinity Mach number defined as

$$M^- = \frac{V^-}{(\gamma R T^-)^{1/2}} \tag{19}$$

being $\gamma = 5/3$ the specific heat ratio of a monatomic gas.
The numerical scheme described in Section 2.2 has been adopted to obtain approximate solutions of Eq. (16) with boundary conditions (17) as long-time limit of solutions of Eq. (10) with identical boundary conditions and initial condition

$$f(x, \boldsymbol{v}|0) = \begin{cases} \Phi^-(\boldsymbol{v}) & x < 0 \\ \Phi^+(\boldsymbol{v}) & x > 0 \end{cases} \tag{20}$$

The computations reported have been carried out for both a weak, $M^- = 1.5$, and a medium strength, $M^- = 3$, shock wave. The collision frequency has been obtained from Eq. (9), assuming that the viscosity is given by the expression

$$\mu(T) = \mu_0 \left(\frac{T}{T_0}\right)^{0.74} \tag{21}$$

In Eq. (21), T_0 is a reference temperature and μ_0 is the value of the viscosity at the reference temperature. The temperature exponent has been set equal to 0.74 to match the computational conditions of Ref. [20] whose results have been used to asses the accuracy of the calculations presented here. A non-dimensional form of the Eq. (10) has been adopted in actual computations by normalizing velocity $\boldsymbol{v}$ to $\sqrt{2RT^-}$, time t to $\tau^- = 1/\nu^-$ and spatial coordinate x to the mean free path $\lambda^- = \sqrt{2RT^-}\tau^-$. The reference value ν^- for the collision frequency has been obtained by setting $T_0 = T^-$ in Eq. (21). The infinite physical space has been replaced by the finite interval $[-L/2, L/2]$ which has been divided into N_x identical cells of width $\Delta x = L/N_x$. The non-dimensional size L of the spatial domain has been set equal to 70, varying N_x between 128 and 18432. The cell number N_x has been increased well above the limit imposed by accuracy in order to investigate the GPU performances as a function of computational load. Similarly, the velocity space has been replaced by a parallelepiped in which each velocity component v_α varies in a finite interval, divided into N_{v_α} equal cells. Both position and dimension of the parallelepiped in the velocity space as well as the number of velocity cells vary with the chosen Mach number. In case of the weak shock wave, $M^- = 1.5$, the same number of grid points has been used for the three normalized velocity

components by setting $N_{v_\alpha} = 16$, with $v_x \in [-5, 7]$ and $v_y, v_z \in [-6, 6]$. In case of the $M^- = 3$ shock wave, the grid point setting has been changed to $N_{v_\alpha} = 30$, with $v_x \in [-10, 12]$ and $v_y, v_z \in [-11, 11]$. Finally, the normalized time step Δt has been set equal to 0.05. Before describing the algorithm implementation and discussing the results, it is worth observing that, for *spatially* one and two-dimensional problems, the dimensionality of the *velocity* space associated with kinetic model equations having the structure of Eq. (6) can be accordingly reduced to one and two, respectively [22]. The standard form of the simple reduction method leads to a system of two coupled kinetic equations which govern the behavior of two distribution functions defined on the reduced velocity space and keep the same form of the original BGKW equation. Application of the projection method allows a considerable reduction of memory demand but destroys compatibility with the solver of the full Boltzmann equation which always requires a three-dimensional velocity space. Hence, a full three-dimensional velocity space has been used in most of the calculations presented here and the application of the reduction method has been limited to a few cases (described below) where the memory saving allows a spatial resolution level which would not otherwise possible, because of the GPU memory limits.

4.2 Results and discussion

In this section, we first validate the code by solving the plane shock structure problem and then we evaluate its performance by comparing the GPU and CPU execution times.

Figures 4a and 4b show the velocity and temperature profiles versus the x coordinate. Solid and dashed lines are the results from the numerical solution of Eq. (16) for $M^- = 1.5$ and $M^- = 3$, respectively. Solid circles and squares are the results presented in Ref. [20] for $M^- = 1.5$ and $M^- = 3$, respectively. The agreement is good and provides a validation of the numerical code.

The performance of the GPU implementation is compared against the sequential version running on the single core of 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, e.g., after file I/O, and do not include the time required to transfer data between the disjoint CPU and GPU memory spaces. Figure 5 reports the obtained speed-up data as a function of the number of spatial grid points N_x. The curve marked by circles refers to the case $M^- = 1.5$ and $N_{v_\alpha} = 16$, whereas squares mark the speed-up curve for $M^- = 3$ and $N_{v_\alpha} = 30$. The maximal value of N_x is different for the two curves because the memory storage, proportional to $N_x N_v$, is limited by the size of the GPU memory pool. A comparison of the two curves in the common range $N_x < 6000$ shows that speed-up grows rapidly and it is mainly determined by N_x, the overall storage having a smaller effect on

performances. In the $M^- = 1.5$ case it is possible to further increase N_x and show that the speed-up levels up at about 600 if N_x approximately exceeds 10^4. This behavior is the result of the parallel set up of the collision step in N_x independent threads assigned to cells 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 of the $M^- = 1.5$ case, 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_x, as reported in Refs. [1,3].
Figures 6a and 6b show the relative time spent to perform the streaming kernel, $T_{\rm s}$, and the collision kernel, $T_{\rm c}$, versus the number of cells in the physical space, for $M^- = 1.5$ and $M^- = 3$, respectively. As expected, the collision kernel is more time consuming than the streaming kernel which takes at most 35% of overall computing time. Moreover, the relative time does not appear to depend appreciably on the number of cells in the velocity space.
The increasing importance of streaming shown by Figures 6 recommends a closer examination of the algorithm efficiency, in view of the application to the two-dimensional problem discussed in the next session. A strongly simplified evaluation of ideal performances of the streaming step can be obtained by observing that a single application of Eq. (12) requires the execution of four floating point operations (the Courant number is computed once at the beginning of the loop) and two accesses to the global memory (one for loading the distribution function to the shared memory and one for updating its value). As mentioned above, 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 $(4 : 8)$, it is reasonable to obtain the number of floating point operation per second (FLOPs) from the transfer rate alone. Hence, the ideal number of FLOPs can be obtained by assuming that four floating point operations will be executed in the time required to transfer eight bytes from the main memory. Accordingly, this simple argument yields an ideal performance of 56 GFLOPs. A similar performance analysis can be performed on the collision kernel whose threads compute macroscopic fields in each spatial cell and perform the local homogeneous relaxation step according to Eq. (15). The computation of density, velocity, temperature and a few selected stress tensor components in addition to the distribution function update requires about 30 floating point operations and 3 float accesses on the main memory, for each element of the distribution function array. Again, assuming that ideal performances can be estimated from memory bandwidth, the resulting ideal performance amounts to 280 GFLOPs.
Timing the execution of the separate kernels and counting the number of associated floating point operations provides the real kernels performance, reported in Fig. 7 where GFLOPs are shown as a function of N_x in the $M^- = 1.5$ case. Solid line with circles, dashed line with squares and dot-dashed line with

triangles are the measured performances of streaming kernel, collision kernel and overall code, respectively. It is possible to note that the performance of the streaming kernel grows with N_x and quickly levels at about 30 GFLOPs, approximately one half of the estimated ideal performance. The difference can be justified by observing that the real CUDA™ implementation of the finite difference scheme [23] is not free from thread divergence and ancillary tasks whose effects can be evaluated with difficulty. The collision kernel performance closely patterns the speed-up behavior: it rapidly grows in the range $N_x < 10^4$, then it reaches a peak value of about 175 GFLOPs. After the peak, the GFLOPs rate oscillates around 150 GFLOPs. The main reason why the collision kernel performs better that the streaming kernel is its higher FLOP to memory operation ratio which allows a more efficient use of GPU computing power. The absence of thread divergence is also a feature which positively affects performances.

5 Driven cavity

The driven cavity flow is a classical spatially two-dimensional benchmark problem which, in spite of its simple geometry, contains most of the features of more complicated problems described by kinetic equations. In particular, the handling of the streaming step is made more complicated by the presence of impermeable walls. Since regular and semi-regular schemes are particularly effective in capturing small deviations from equilibrium, the computations described in this section refer to very low Mach number driven cavity flows. The results are compared with previous investigations of the same flow geometry where low Mach rarefied gas flows, governed by the BGKW kinetic model, have been studied adopting the linearized form of the model and varying the degree of rarefaction [16]. As shown below, the results of linearized theory can be obtained as a particular application of the full non-linear BGKW equation by the numerical method described in Section 2.

5.1 Formulation of the problem

A monatomic gas is confined in the two-dimensional square cavity

$$\mathcal{C} = \{(x, y) : 0 < x < L, 0 < y < L\}$$

The flow is driven by a uniform translation of the top with velocity $V_w \hat{\boldsymbol{e}}_x$, being $\hat{\boldsymbol{e}}_x$ a unit vector parallel to x direction. The gas flow is governed by the

two-dimensional steady BGKW equation

$$v_x \frac{\partial f}{\partial x} + v_y \frac{\partial f}{\partial y} = \nu(\Phi - f) \tag{22}$$

It is further assumed that all the walls are kept at uniform and constant temperature T_w and that the gas atoms which strike the walls are re-emitted according to the Maxwell's scattering kernel with complete accommodation

$$f(\overline{\boldsymbol{x}}, \boldsymbol{v}) = \frac{n_w(\overline{\boldsymbol{x}})}{(2\pi R T_w)^{3/2}} \exp\left\{-\frac{[\boldsymbol{v} - \boldsymbol{V}_w(\overline{\boldsymbol{x}})]^2}{2RT_w}\right\}, \quad (\boldsymbol{v} - \boldsymbol{V}_w) \circ \hat{\boldsymbol{n}} > 0 \tag{23}$$

where $\overline{\boldsymbol{x}}$ is a point of the boundary, $\hat{\boldsymbol{n}}(\overline{\boldsymbol{x}})$ the inward normal at $\overline{\boldsymbol{x}}$, $\mathbf{V}_w(\overline{\boldsymbol{x}})$ is the wall velocity, different from zero only on the top wall, T_w the wall temperature and $n_w(\overline{\boldsymbol{x}})$ the wall density which is determined by impinging mass flux through the following relationship

$$n_w(\overline{\boldsymbol{x}}) = \left(\frac{2\pi}{RT_w}\right)^{1/2} \int_{(\boldsymbol{v}-\boldsymbol{V}_w)\circ\hat{\boldsymbol{n}}<0} |(\boldsymbol{v} - \boldsymbol{V}_w) \circ \hat{\boldsymbol{n}}| f \, d\boldsymbol{v} \tag{24}$$

which ensures zero net mass flux at boundary points.
The straightforward two-dimensional extension of the numerical scheme described in Section 2.2 has been adopted to obtain approximate solutions of Eq. (22) with boundary conditions (23) as long time limit of the unsteady problem. It is assumed that the initial gas state, at time $t = 0$, is described by the uniform equilibrium Maxwellian

$$f(\boldsymbol{x}, \boldsymbol{v}|0) = \Phi_0(\boldsymbol{v}) = \frac{n_0}{(2\pi R T_0)^{3/2}} \exp\left(-\frac{\boldsymbol{v}^2}{2RT_0}\right) \tag{25}$$

being n_0 and $T_0 = T_w$ the initial values of the uniform density and temperature, respectively. The computation stops when a certain global flow quantity ϵ_s falls below a fixed threshold, i.e., 10^{-4} for the computations described in the following section. ϵ_s is obtained as the largest relative error defined as $|(\psi(\boldsymbol{x}, t) - \psi(\boldsymbol{x}, t - t_s))/\psi(\boldsymbol{x}, t - t_s)|$ in the computational domain, where $\psi(\boldsymbol{x}, t)$ represents the density, temperature and velocity fields. The time t_s between two successive snapshots is set equal to $L/\sqrt{2RT_w}$. The non-dimensional form of the governing equation is easily obtained by adopting the reference mean free time $\tau_0 = 1/\nu_0$ and mean free path $\lambda_0 = \sqrt{2RT_0}\tau_0$ as time and length units, respectively. It is also immediately seen that the problem solutions depend on the dimensionless wall velocity $V_w/\sqrt{2RT_0}$ and the rarefaction parameter $\delta = L/\lambda_0$ which is the reciprocal value of the Knudsen number $\mathrm{Kn} = \lambda_0/L$. As mentioned above, in Ref. [16], cavity flows have been studied by the linearized BGKW equation, assuming that $V_w \ll \sqrt{2RT_0}$. In order to reproduce these results by the full non linear model equation considered here, the dimensionless lid velocity has been set equal to 0.01. The gas is thus in a weakly

non-equilibrium state and the non-linear results approach the linearized ones. The cavity problem has been solved for three values of the rarefaction parameter $\delta = 0.1, 1, 10$. Two series of computations have been performed. The first one is based on a computer code version, called V1, which uses a fully three-dimensional grid in the velocity space. This code version is compatible with the full Boltzmann equation solver which keeps the same streaming kernel but replaces the simplified BGKW collision model with the Monte Carlo evaluation of the collision integral in Eq. (1). Although more general the code is quite memory demanding and, for some flow conditions, the grid refinement process was stopped by the GPU memory limits, before satisfactory agreement with the reference solutions was reached. Hence a second series of computations has been performed by using the BGKW specific code, called V2, based on the reduction method which eliminates the z velocity component and allows finer grids in the reduced phase space. This second series of computations has the only aim of assessing accuracy. Actually, the parallel performances of the code have been investigated using the first and more general version, not the reduced one. As in the case of the shock problem, a sequential version of the algorithm, using the three-dimensional velocity space, has been run on the CPU to obtain speed-up data.

5.2 Results and discussion

The square cavity, $[0, \delta] \times [0, \delta]$, has been divided into $N_x \times N_y$ identical cells. Since the deviation form equilibrium is small, a cubic velocity space has been constructed by varying each normalized velocity component v_α in the interval $[-3, 3]$. The velocity box has been divided into a number of identical cubic cells, by evenly distributing N_{v_α} velocity nodes along each velocity component. Finally, the time step has been varied in the range $10^{-4} - 10^{-2}$ depending on the rarefaction parameter.
Figures 8a and 8b show the profiles of the normalized velocity component, V_x/V_w, along the vertical line $x/L = 1/2$ and the normalized velocity component, V_y/V_w along the horizontal line crossing the center of the main vortex which forms in the cavity, respectively. The lines represent the results of the V1 code whereas the symbols are the results reported in Ref. [16]. Each figure reports the results for two different values of the rarefaction parameters: $\delta = 0.1$ (dashed lines and squares) and $\delta = 10$ (solid lines and circles). The results of the $V1$ parallel code have been obtained by setting $N_x = N_y = 160$ and $N_{v_\alpha} = 20$. The total memory occupation amounts to about 819Mb which is very close to the GPU physical memory limit. The grid resolution is sufficient to obtain good agreement with the reference solutions velocity fields. In order to obtain a more detailed comparison, we introduce two global flowfield properties, namely the mean dimensionless shear stress, D, on the moving wall and the dimensionless flow rate, G, of the main vortex. The two quantities are

defined as

$$D = \frac{\int_0^L p_{xy}(x, L)\, dx}{L p_0 \frac{V_w}{\sqrt{2RT_0}}}, \qquad G = \frac{\int_0^L |v_x(L/2, y)|\, dy}{L V_w} \tag{26}$$

being p_0 the pressure in the reference initial state.
Table 1 compares the predictions of D and G obtained by solving Eq. (22) by the $V1$ parallel code with the values reported in Ref. [16] where the linearized BGKW equation has been solved by a discrete velocity method. Although the overall agreement is good for $\delta = 0.1$, slightly larger discrepancies are observed for $\delta = 1.0$ and $\delta = 10$, the largest deviation being found for D at $\delta = 10$. The version $V2$ of the code has then been used to refine the phase space grid and investigate the convergence properties of the numerical scheme. The D and G values obtained by grid refinements are also reported in Table 1 and marked by an asterisk. The data for $\delta = 1.0$ have been obtained by setting $N_x = N_y = 416$ and $N_{v_\alpha} = 24$ whereas the grid setup $N_x = N_y = 544$ and $N_{v_\alpha} = 18$ has been used for $\delta = 10$. The comparison shows that the grid refinement brings the results closer to the reference solution. The correction on G is quite modest since the coarser grid used by the $V1$ code was sufficient to obtain a good approximation of the velocity field. However, a finer grid is required for an accurate evaluation of the stress distribution on the moving wall.
Grid size effects on the profile of the xy component of the stress tensor have been evaluated by means of the V2 code and are shown in Figure 9. As a part of the validation process, the total mass variation in the cavity during the simulations has been computed. In fact, as mentioned in Section 2.2, the non-conservative approximation of the Maxwellian distribution function could cause a variation of the total mass inside the domain. Numerical tests have shown that the V1 code with the reference discretization reported above produce percentage mass errors per time step equal to $2.8 \times 10^{-6}\%$ and $11.7 \times 10^{-6}\%$ for the streaming and collision step, respectively. The overall percentage mass error at the end of the simulation is about 0.059%. The error can be further reduced by increasing the number of cells in the velocity space or by using a local correction procedure [12] whose computational cost is comparable with the parallel evaluation of macroscopic variables.
As mentioned in Section 4.2, the performance of the GPU implementation is compared against the sequential version running on the single core of the CPU by computing the speed-up factor. The results for the $V1$ code are summarized in Figures 10 and 11. The former shows the behavior of the speed-up ratio S as a function of the number of spatial cells $N_s = N_x \times N_y$. The speed-up behavior is similar to the one observed in the one-dimensional test case and it can be justified by means of the same arguments given in Section 4.3. A more detailed account of the code parallel performances for different grid setups is given in Table 2 which shows that the number of grid nodes in the velocity space does not appreciably affect parallel performances because of the threads organization described in Section 4.3. It should be also observed that the maximum

speed-up achieved by the two-dimensional code is about 340, almost one half of the one-dimensional code peak performances. The reason for the speed-up worsening can be more clearly understood by the separate analysis of the performances of the collision and streaming kernels presented in Figure 11 which shows the measured GFLOPs rates associated with the collision and streaming kernels in addition to the overall code GFLOPs rate. As expected, the collision kernel keeps the same performance level reached in the one dimensional case (see Figure 7). However, the streaming kernel performance drops from about 34 GFLOPs to 22.5 GFLOPs. This is due to the greater number of both divergent threads and non-coalescent memory accesses in the present case with respect to the one-dimensional problem. The instruction overhead increases as well, thus leading to a further reduction of the estimated ideal performance. As a result of its poorer performances, the streaming kernel absorbs about 65% of the total execution time and reduces overall performances accordingly. As is evident from its definition, the speed-up factor also depends on the execution time of the sequential code. The results reported in Figures 5 and 10 show that the measured speed-up may exceed the number of GPU cores whose working frequency is considerably lower than the CPU clock frequency of the host computer. One may thus wonder whether the reported high speed-up ratios are due to a poor optimization of the sequential codes and not to a real gain in computational time of their parallel counterparts. As mentioned above, CPU-based codes developed here have not been optimized beyond the level of standard optimization options made available by an open source compiler. A crude estimate of the sequential code performances can be obtained along the same lines as for the GPU-based code. The main difference is that the time the CPU requires to execute floating point operations is no longer negligible when compared with the time required to transfer data to/from the main memory. Accordingly, it has to be taken into account in the estimate of the ideal number of GFLOPs. The CPU maximum bandwidth and the nominal computing performance per single core have been respectively set equal to 12.8 GB/s and 20 GFLOPs, as obtained from Ref. [24] for the CPU model used in the numerical tests. The CPU GFLOPs rate has been obtained as

$$\text{GFLOPs} = \frac{N_{op}}{T_{op}} \tag{27}$$

where N_{op} and T_{op} are the number of operations and the computing time necessary to complete a full cycle which includes the advection step, the computation of the moments and the homogeneous relaxation step. Adopting a simplified view which ignores the complex CPU internal data flow and the presence of a memory cache, T_{op} can be obtained by summing the time to transfer N_{data} to the time to perform N_{op} operations on the transferred data

$$T_{op} = \frac{N_{data}}{12.8 \text{ GB/s}} + \frac{N_{op}}{20 \text{ GFLOPs}} \tag{28}$$

Then, the ideal CPU GFLOPs rate can be estimated as

$$\text{GFLOPs} = \frac{1}{\dfrac{N_{data}}{N_{op}}\dfrac{1}{12.8\ \text{GB/s}} + \dfrac{1}{20\ \text{GFLOPs}}} \tag{29}$$

The algorithm analysis shows that the ratio N_{data}/N_{op} can be assigned a value of 0.51 bytes/FLOP. Therefore, the sequential code may ideally deliver 11.1 GFLOPs, in marked contrast with the effective measured performance which amounts to about 0.34 GFLOPs. The reasons for the large discrepancy between ideal and effective CPU performances are not clear. However, although the CPU ideal GFLOPs rate might have been overestimated, this simple analysis suggests that the high speed-up factors may be also explained by insufficient optimization of the sequential code. Nevertheless, we point out that the sequential code performances reported here are comparable to those of similar codes described in literature. For instance, in Ref. [16] the numerical solution of the cavity flow problem is obtained in about 3 hours. The speed-up factor evaluated on the basis of this execution time would then drop to 265. Therefore, speed-up data may change in favor of CPU-based codes by using more efficient compilers or exploiting parallel programming tools on multicore machines. In the latter case the speed-up factor will be ideally divided by the number of cores, provided that the sequential code is properly redesigned to achieve high parallel efficiency. At present, however, GPU acceleration seems to offer speed-up ratios which will safely remain beyond the capabilities of any optimization strategy on the CPU version of the class of algorithms considered here.

6 Conclusions

The aim of this paper is to explore the possibility of exploiting the computational power of modern GPUs to solve kinetic equations by regular and semi-regular numerical methods. Two benchmark problems have been studied by adopting the Bhatnagar-Gross-Krook-Welander (BGKW) kinetic model for the collision term in combination with a simple finite difference scheme. The results lead to concluding that, adopting a particularly simple form of the numerical scheme (rectangular phase space cells, uniform and fixed grid size) the porting of the sequential code onto GPUs allows a reduction of the computing time of two orders of magnitude. This is a quite encouraging result in view of the applications of this class of methods to steady and unsteady low Mach number flows, since it also applies to the full Boltzmann equation version of the scheme, as it will be reported in a forthcoming paper. However, the test problems examined here have clearly shown that the size of physical memory, not the number crunching capability, is the main obstacle toward the applica-

tion to complex two or three-dimensional flows. The memory constraint can be simply alleviated by using multi-GPU hardware. However, a number of new issues related to the application development process and resource management should be addressed, since programming a multi-GPU application is similar to programming an application on multicore hardware [17]. Therefore, further progress in this direction will require the adoption of several strategies. The first one is to develop adaptive grids both in the physical and velocity space as well as more accurate differencing schemes. The second one is the development of hybrid CPU-GPU codes in which the computational domain is split into a number of sub-domains which are sequentially submitted to the GPU. The data transfer overhead decreases the overall speed-up, but a proper design of the algorithm can still obtain good performances.

Acknowledgment

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

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

[3] J. A. Anderson, C. D. Lorenz, A. Travesset, "General purpose molecular dynamics simulations fully implemented on graphics processing units", J. Comp. Phys. 227 (2008) 5342-5359.

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

[5] S. Lorenzani, L. Gibelli, A. Frezzotti, A. Frangi, C. Cercignani, "Kinetic approach to gas flows in microchannels", Nanoscale and Microscale Thermophysical Engineering 11 (2007) 211-226.

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

[7] S. Chapman, T. G. Cowling, The mathematical theory of non-uniform gases, Cambridge University Press, 1990.

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

[9] A. Frezzotti, L. Gibelli, S. Lorenzani, "Mean field kinetic theory description of evaporation of a fluid into vacuum", Phys. Fluids 17 (2005) 012102-12.

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

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

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

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

[14] V. V. Aristov, Direct Methods for Solving the Boltzmann Equation and Study of Nonequilibrium Flows, Springer-Verlag, New York, 2001.

[15] A. Frezzotti, "A numerical investigation of the steady evaporation of a polyatomic gas", Eur. J. Mech. B: Fluids 26 (2007) 93-104.

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

[17] NVIDIA Corporation, "NVIDIA CUDA C Programming Guide", March 2011. Version 4.0. http://www.nvidia.com/object/cuda_home_new.html

[18] 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 (1954) 511-525.

[19] P. Welander, "On temperature jump in a rarefied gas", Arkiv fiir Fysik 7 44 (1954) 507-553.

[20] H. W. Liepmann, R. Narasimha, M. T. Chahine, "Structure of a plane shock layer", Phys. Fluids 5 (1962) 1313-1324.

[21] P. Kowalczyk, A. Palczewski, G. Russo, Z. Walenta, "Numerical solutions of the Boltzmann equation: comparison of different algorithms", Eur. J. Mech. B: Fluids 27 (2008) 62-74.

[22] C. K.Chu, "Kinetic-theoretic description of the formation of a shock wave", Phys. Fluids 8 (1965) 12-22.

[23] P. Micikevivius, "3D finite difference computation on GPUs using CUDA", ACM International Conference Proceeding Series, 383 (2009) 79-84.

[24] Intel® microprocessor export compliance metrics.
http://www.intel.com/support/processors/sb/CS-023143.htm

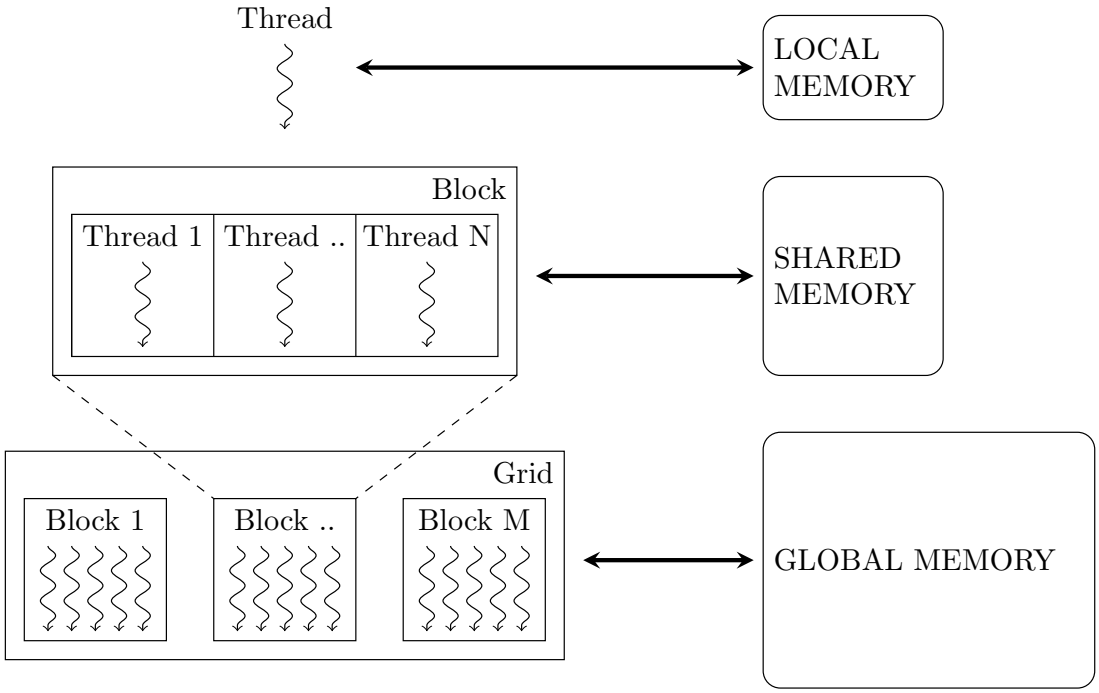

Fig. 1. Organization of grid of threads and relationship between threads and available memory spaces.

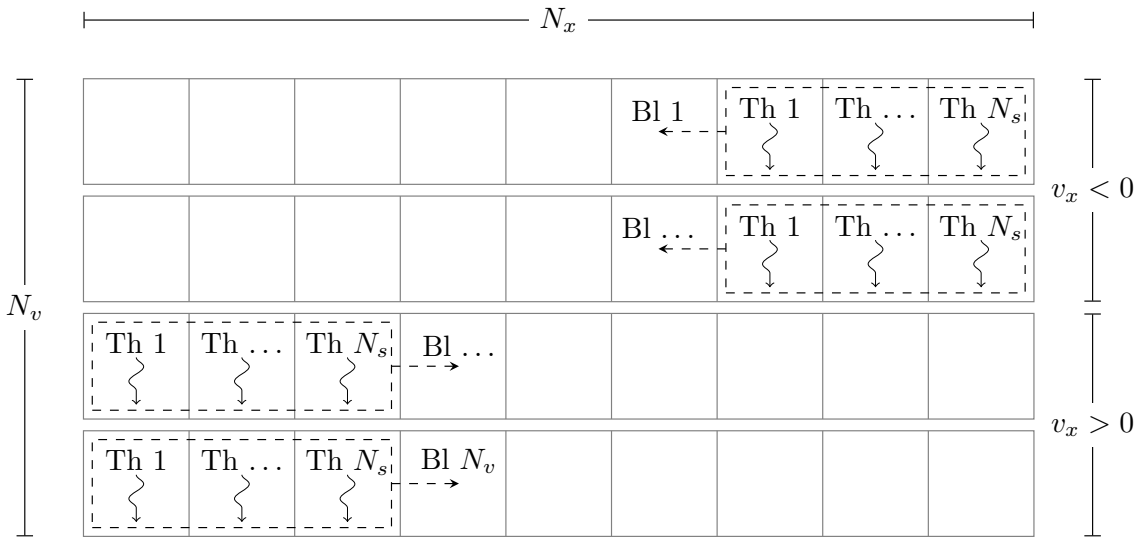

Fig. 2. Parallelization strategy of the streaming kernel.

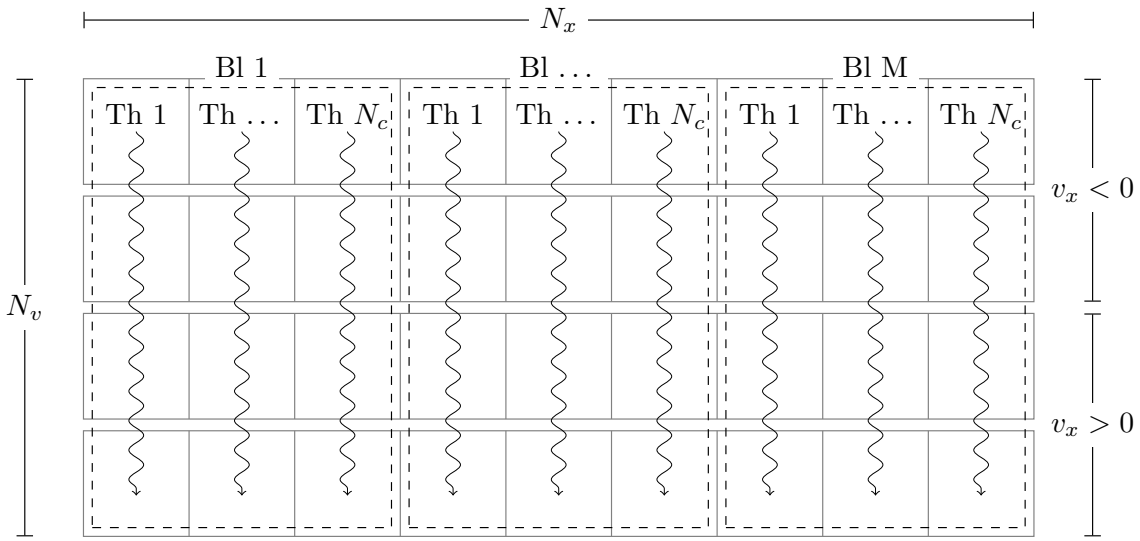

Fig. 3. Parallelization strategy of the collision kernel.

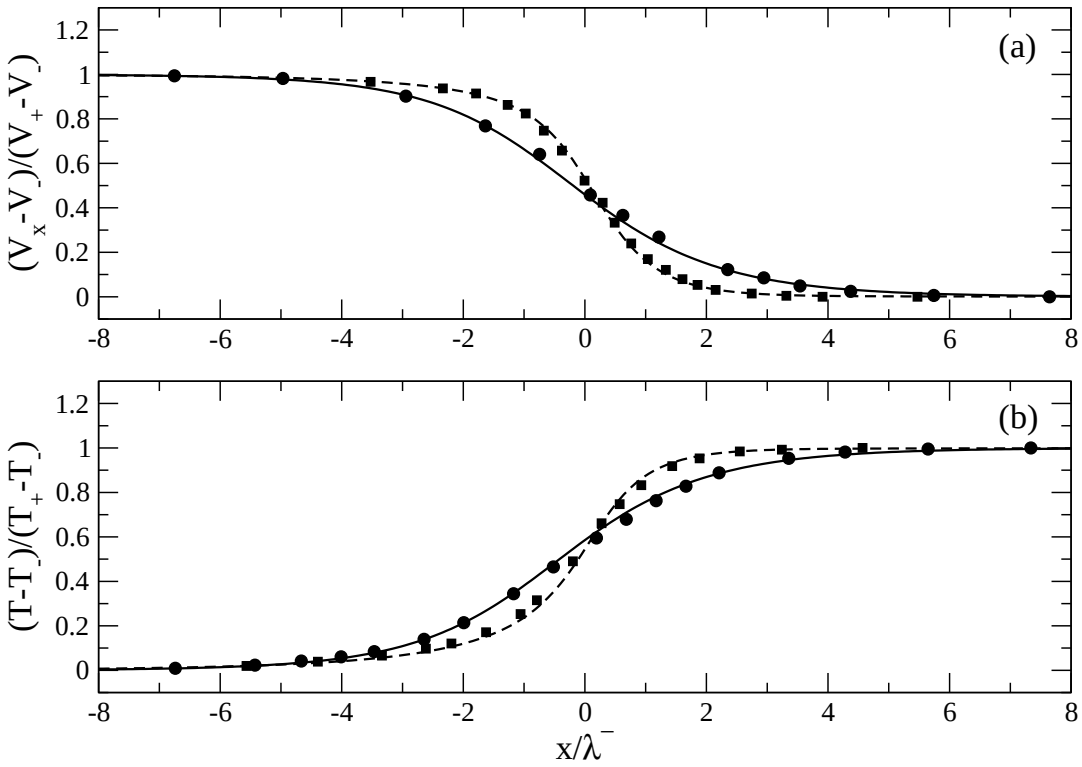

Fig. 4. Dimensionless (a) mean velocity and (b) temperature profiles versus the dimensionless x coordinate. Solid and dashed lines are the results obtained with the parallel code for $M^- = 1.5$ and $M^- = 3$, respectively. Solid circles and squares are the results presented in Ref. [20] for $M^- = 1.5$ and $M^- = 3$, respectively. $N_x = 1024$, $N_{v_\alpha} = 16$ ($M^- = 1.5$) and $N_{v_\alpha} = 30$ ($M^- = 3$).

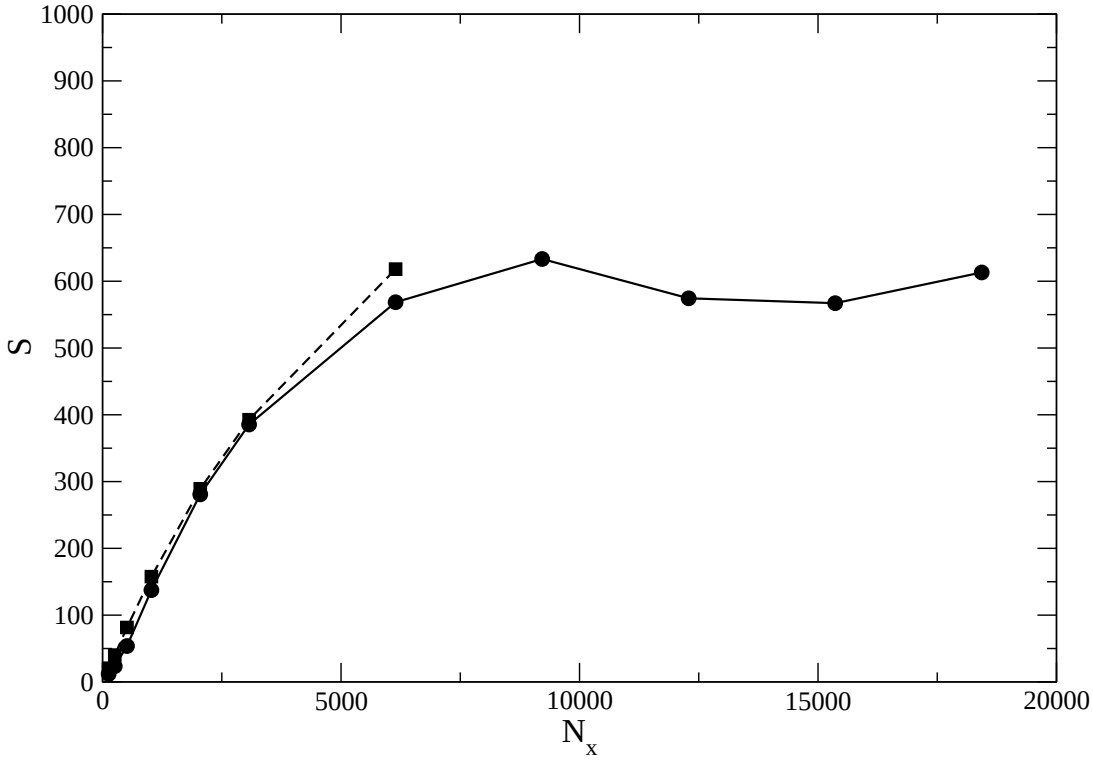

Fig. 5. Overall speed-up, S, versus the number of cells in the physical space, N_x. Solid line with circles and dashed line with squares are the results obtained with the parallel code for $M^- = 1.5$ and $M^- = 3$, respectively. $N_{v_\alpha} = 16$ ($M^- = 1.5$) and $N_{v_\alpha} = 30$ ($M^- = 3$).

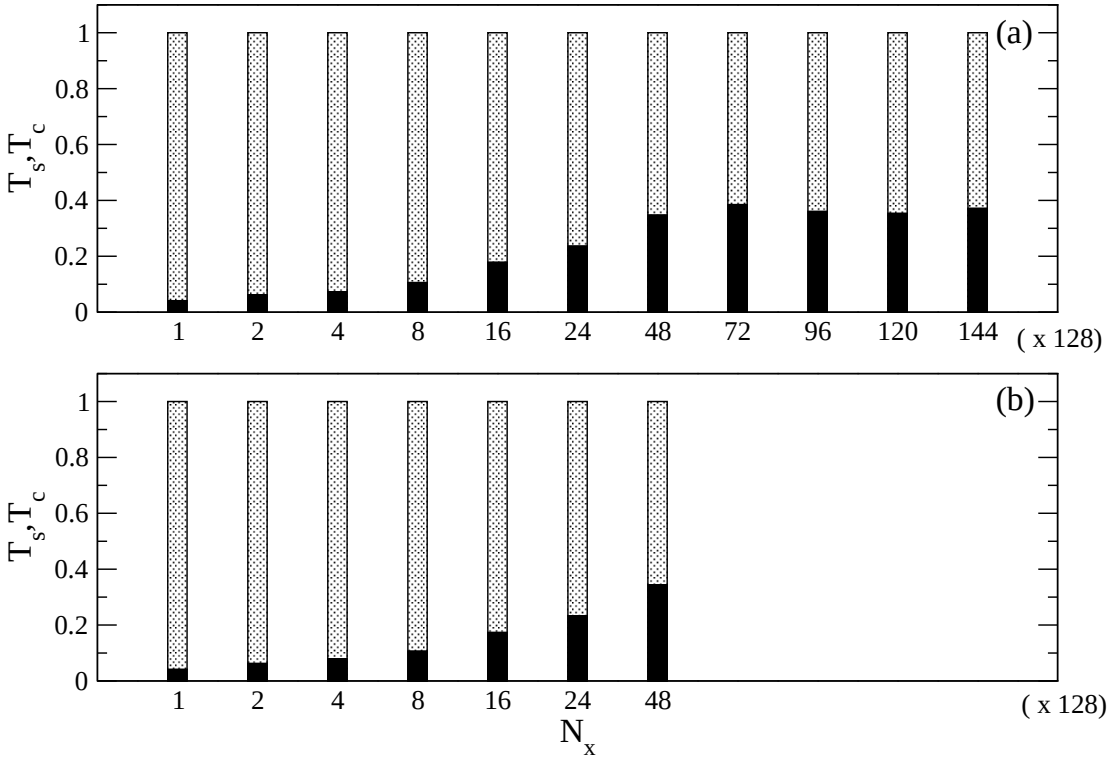

Fig. 6. Relative time spent on the streaming and collision kernel for (a) $M^- = 1.5$ case and (b) $M^- = 3$ case. Solid bar: streaming kernel; pattern bar: collision kernel. $N_{v_\alpha} = 16$ ($M^- = 1.5$) and $N_{v_\alpha} = 30$ ($M^- = 3$).

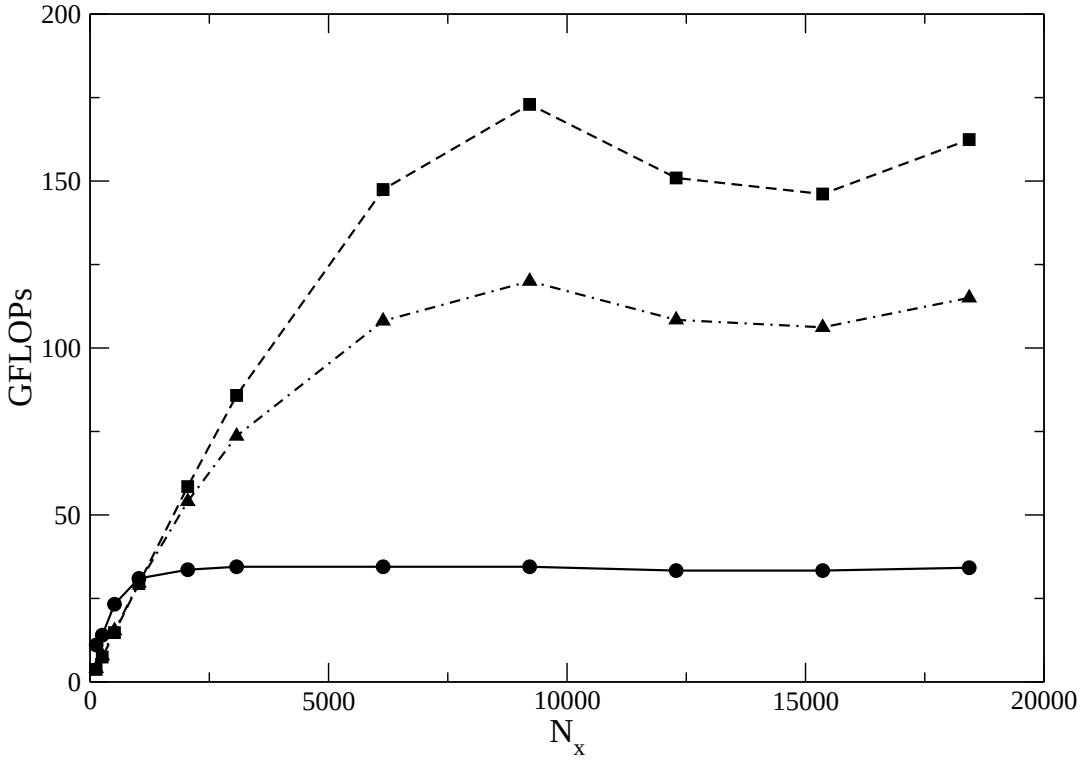

Fig. 7. GFLOPs versus the number of cells in the physical space, N_x. Solid line with circles: streaming kernel; dashed line with squares: collision kernel; dot and dashed line with triangles: overall code. $M^- = 1.5$, $N_{v_\alpha} = 16$.

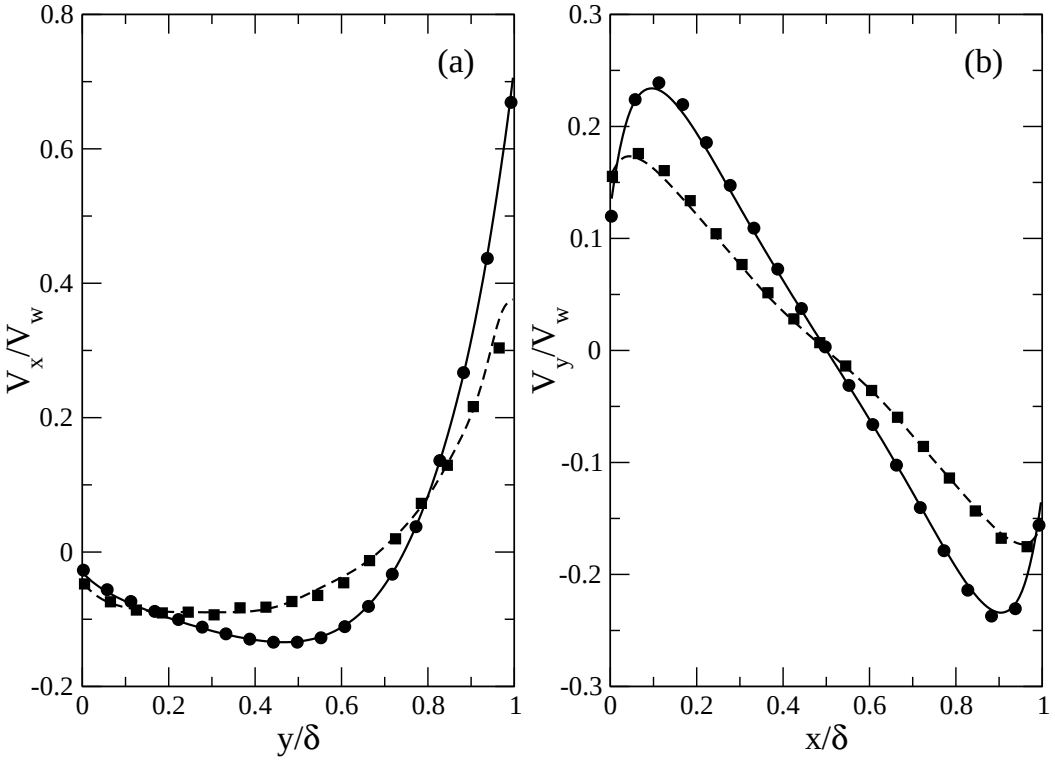

Fig. 8. Profiles of the dimensionless (a) horizontal component of the velocity on the vertical plane crossing the center of the cavity and (b) vertical component of the velocity on the horizontal plane crossing the center of the top vortex. Solid and dashed lines: numerical solutions obtained with the parallel code for $\delta = 10$ and $\delta = 0.1$ respectively; solid circles and solid squares: numerical solutions reported in Ref. [16] for $\delta = 10$ and $\delta = 0.1$ respectively. $V_w/\sqrt{2RT_w} = 0.01$, $N_x = N_y = 16$, $N_{v_\alpha} = 20$.

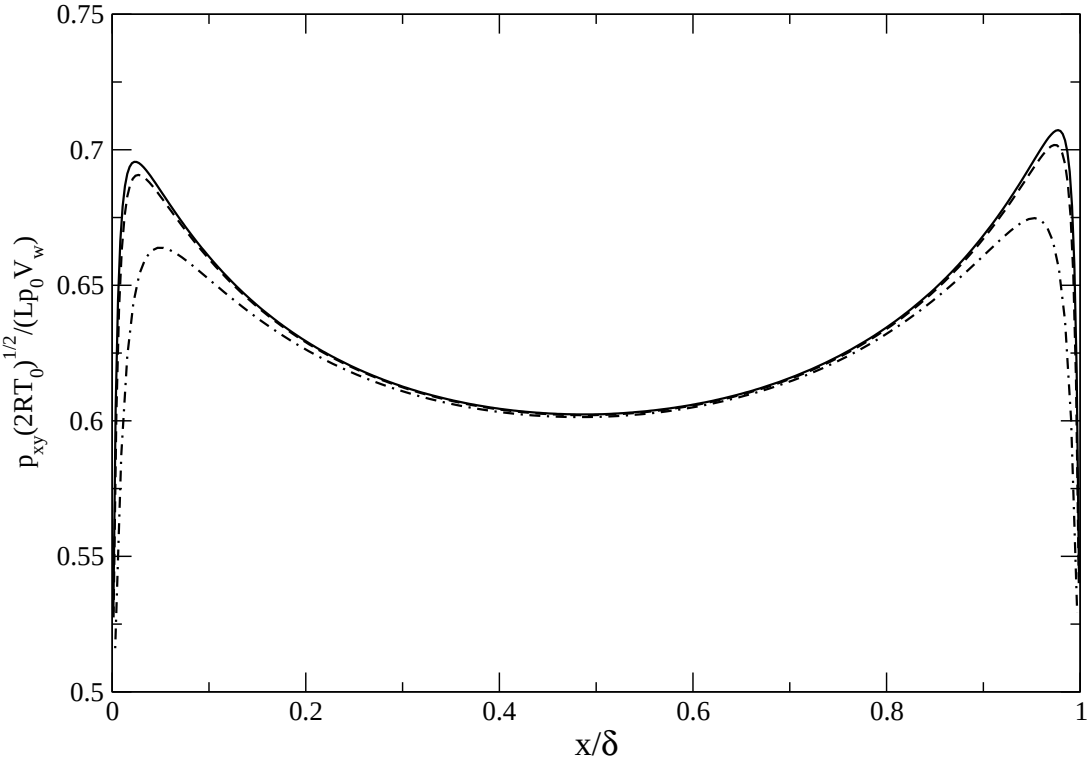

Fig. 9. Dimensionless xy component of the stress tensor along the moving wall versus the dimensionless x coordinate. Dashed-dotted line: $N_{v_\alpha} = 20$ and $N_x = N_y = 160$; dashed line: $N_{v_\alpha} = 24$ and $N_x = N_y = 384$; solid line: $N_{v_\alpha} = 24$ and $N_x = N_y = 416$. $\delta = 1$, $V_w/\sqrt{2RT_w} = 0.01$.

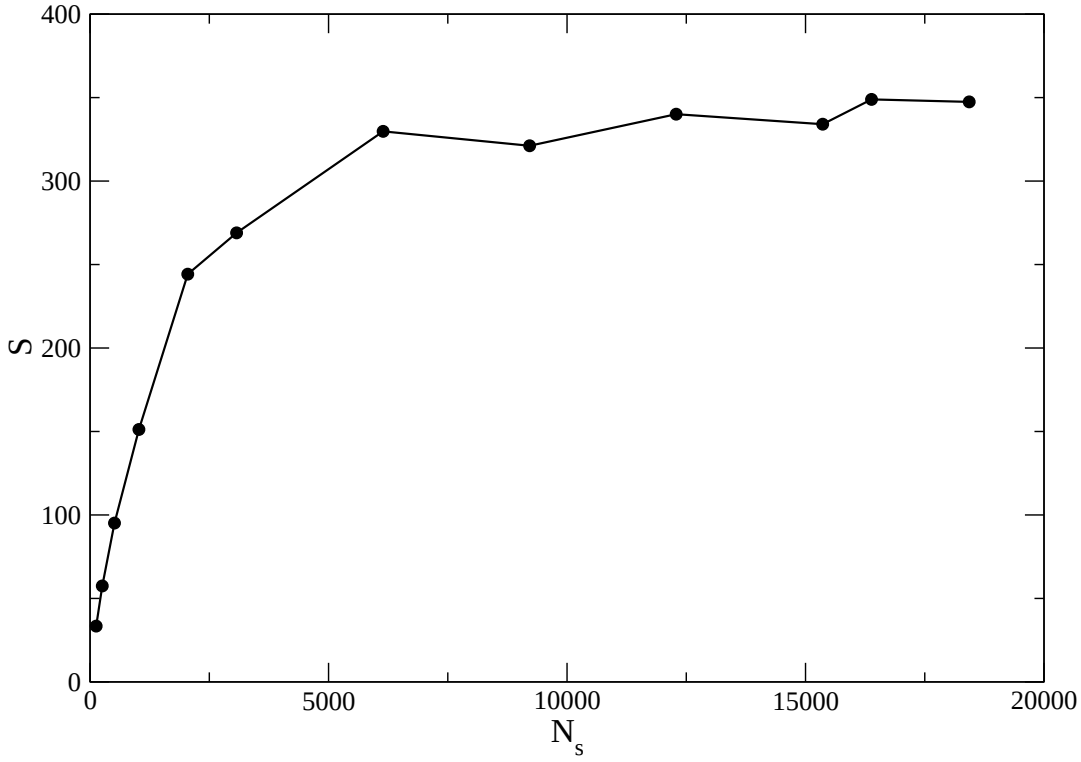

Fig. 10. Overall speed-up, S, versus the number of cells in the physical space, N_s. $\delta = 1$, $V_w/\sqrt{2RT_w} = 0.01$, $N_{v_\alpha} = 20$.

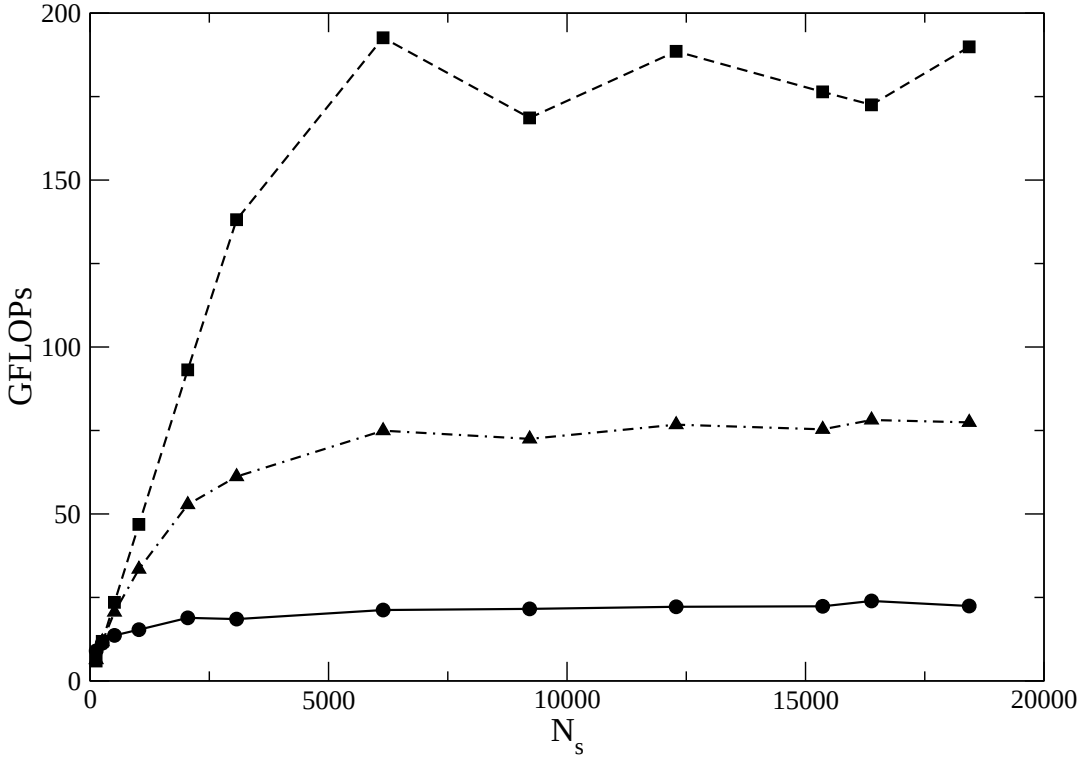

Fig. 11. GFLOPs versus the number of cells in the physical space, N_s. Solid line with circles: streaming kernel; dashed line with squares: collision kernel; dot and dashed line with triangles: overall code. $\delta = 1$, $V_w/\sqrt{2RT_w} = 0.01$, $N_{v_\alpha} = 20$.

δ	D	D (Ref. [16])	G	G (Ref. [16])
0.1	0.675	0.676-0.678	0.0975	0.0973-0.0976
1	0.624 0.631*	0.625-0.631	0.103 0.104*	0.104-0.105
10	0.393 0.411*	0.412-0.415	0.143 0.145*	0.145-0.145

Table 1
Drag coefficient, D, and flow rate of the main vortex, G, versus the rarefaction parameter, δ, obtained with the V1 code; $Nx = N_y = 160$, $N_{v_\alpha} = 20$. Starred data are the results obtained with the V2 code; $N_x = N_y = 416$, $N_{v_\alpha} = 24$ for $\delta = 1$ and $N_x = N_y = 544$, $N_{v_\alpha} = 18$ for $\delta = 10$. $V_w/\sqrt{2RT_w} = 0.01$.

$N_x = N_y$	N_{v_α}	Time (s)	speed-up	GFLOPs
160	14	139	331	74.5
160	16	207	331	74.7
160	18	295	331	74.7
64	20	75.7	281	63.8
96	20	151	319	71.9
128	20	254	340	76.1
160	20	404	331	74.7
416*	24*	825	-	-

Table 2
Execution time (in seconds), speed-up and GFLOPs versus the number of cells in the physical space, $N_x = N_y$, and the number of cells in the velocity space, N_{v_α}, obtained with the V1 code. Starred data are the results obtained with the V2 code. $V_w/\sqrt{2RT_w} = 0.01$.