Lattice Group Models: GPU Acceleration and Numerics

Stefan Brechtken

Institute of Mathematics, Ilmenau University of Technology, Weimarer Straße 25, 98693 Ilmenau, DE

Abstract. *Lattice group models* (LGpM) are kinetic models on integer lattices derived from the automorphism group of the lattice. In the last decades it was too expensive to simulate large systems (100 - 1000 velocities in a 2D or 3D model), with complex physical two or three dimensional domains, on normal computers or clusters within an acceptable amount of time. That changed due to the fast growth of the computing power of modern processing units. We briefly introduce lattice group models and then describe the main parallelization and optimization strategies for an efficient implementation on NVIDIA® graphics cards and INTEL® processors. Afterwards we present the achieved speedups and discuss some results from numerical simulations of physical phenomena (shock waves, Knudsen pump, Karman vortex street) in two and three dimensions at high, transitional and low Knudsen numbers.

Keywords: Boltzmann equation, LGpM, parallel computing, GPU, GP-GPU, CUDA, SSE
PACS: 02.60.-x, 47.45.-n, 47.40.-x, 47.32.ck,

INTRODUCTION

Most numerical computations of the Boltzmann equation are based on probabilistic Monte Carlo techniques, which lead to *direct simulation Monte Carlo methods* (DSMC), or special collision models such as *Bhatnagar-Gross-Krook* (BGK) together with the discretization of the velocity space, which lead to *lattice Boltzmann methods* (LBM). Due to the low computational costs and the wide field of applications these approaches have proven their potential in computational physics. But they can have problems in situations where high Mach numbers or compressible fluids occur and one needs to avoid numerical fluctuations which originate in the use of random sequences.

We use an alternative approach based on the so called *lattice group model* (LGpM) that was introduced by Babovsky in [1, 2]. These LGpMs can generally be considered as *discrete velocity models* (DVM) and are comparable to DVMs that were the subject of the review article of Platkowski and Illner [3] and were further developed by several authors. Panferov and Heintz [4] stated in 1999 that the resulting numerical schemes were computationally too expensive for computers of this time to simulate real, space inhomogeneous problems. Because of that, other authors like Buet [5] developed acceleration schemes that traded computational costs for precision and significantly reduced the random noise of Monte-Carlo techniques.
Because of the development of computing technology and the introduction of graphics processors as usable co-processors, this situation (of insufficient computational resources) changed. Especially *graphics processing units* (GPUs) are used with success in a wide range of scientific applications to accelerate the computations, for example in fluid flow (Boltzmann and Navier Stokes) simulations [6, 7, 8, 9, 10]. So we developed and implemented an LGpM based solver for one, two, and three dimensional problems with a variable discretization of the velocity space (ranging from 9 to 963 points in the two or three dimensional velocity space) on GPUs and CPUs. In the next sections we will take a brief look at the used numerical methods and the results.

MATHEMATICAL APPROACH

With an equidistant discretization $\mathfrak{V}$ of a ball $B_R(0)$ around zero in the velocity space $\mathbb{R}^n_{\mathbf{v}}$, $n = 2,3$, that can easily be obtained via

$$\mathfrak{V} := \overline{\mathfrak{V}} \cap B_R(0),\ \overline{\mathfrak{V}} := \left\{ \mathbf{v} | v_j \in \Delta v \cdot \mathbb{Z},\ j = 1,\ldots,n,\ \mathbf{v} \in R^n_{\mathbf{v}} \right\},\ \Delta v \in \mathbb{R}^+ ,$$

we can write the standard form (as in [3]) of the discrete Boltzmann equation as

$$(\partial_t + v_i \cdot \nabla_{\mathbf{x}}) f_i = \alpha \sum_{j,k,l \in M_{\mathfrak{V}}} A^{k,l}_{i,j} (f_k f_l - f_i f_j) =: J_i[f,f],\ i \in M_{\mathfrak{V}} , \tag{1}$$

where $M_{\mathfrak{V}}$ is the index set of $\mathfrak{V}$, i.e., $M_{\mathfrak{V}} := \{1,\dots,|\mathfrak{V}|\}$, and therefore $\mathbf{v}_i$ is a distinct element of $\mathfrak{V}$. The parameter α is proportional to the desired mean free path of a particle and A is an operator that describes possible particle interactions. In [11] Babovsky described how we can obtain this operator for an arbitrary equidistant discretization of the velocity space. If we apply this to our discretization we have to introduce the superset $\mathfrak{C} \supset \mathfrak{V}$ of all center points of potentially interacting particles. Because we are looking on an equidistant discretization we can simply define $\mathfrak{C}$ as an equidistant discretization of the same region with the discretization parameter $\frac{\Delta v}{2}$:

$$\mathfrak{C} = \left\{ \mathbf{v} | v_j \in \frac{\Delta v}{2} \cdot \mathbb{Z}, j = 1,\dots,n, \mathbf{v} \in R^n_{\mathbf{v}} \right\} \cap B_R(0) \; .$$

The last ingredient is the orthonormal group $\mathfrak{O}$ of $\overline{\mathfrak{V}}$, i.e. the set of all reflections and rotations around zero leaving $\overline{\mathfrak{V}}$ invariant. With this preliminaries we can define the set

$$M_i := \left\{ (i,j,k,l) \;\middle|\; \begin{array}{l} (\mathbf{v}_i, \mathbf{w}_j, \mathbf{v}_k, \mathbf{w}_l) \in \mathfrak{V}^4 \\ \mathbf{w}_j = \mathbf{c} - (\mathbf{v}_i - \mathbf{c}), \\ \mathbf{v}_k = \mathbf{c} + o(\mathbf{v}_i - \mathbf{c}), \\ \mathbf{w}_l = \mathbf{c} - o(v_i - \mathbf{c}), \\ \mathbf{c} \in \mathfrak{C}, o \in \mathfrak{O} \end{array} \right\}$$

that contains all collision relations (particle interactions) on the velocity grid that belong to $\mathbf{v}_i$. And now we can describe the operator A as an indicator function over $M^4_{\mathfrak{V}}$, i.e.

$$A : M^4_{\mathfrak{V}} \mapsto \{0,1\}, A^{k,l}_{i,j} = \mathbb{1}_{M_i}(i,j,k,l) \; .$$

The computational complexity of the calculation of the right hand side of (1) is directly proportional to the number of collision relations in the set $\bigcap_{i \in M_{\mathfrak{V}}} M_i$ and this number is generally growing quadratically with the number of velocities $|\mathfrak{V}|$ i.e. with the radius R of the ball B. Table 1 contains the sizes of $|\mathfrak{V}|$ and the corresponding number $|\bigcap_{i \in M_{\mathfrak{V}}} M_i|$ of collision relations.

TABLE 1. size of the velocity space over induced collision relations

$\lvert\mathfrak{V}\rvert$	27	57	123	251	485	949
$\lvert\bigcap_{i \in M_{\mathfrak{V}}} M_i\rvert$	288	1'140	8'988	61'932	255'108	1'062'576

NUMERICAL METHODS

Because the calculation of **J** (the right hand side of (1)) has high computational costs and because this was our first attempt in parallelization, we wanted to use simple algorithms with a minimal number of calculations of **J**. This led to a first order (with minimal modifications second order) operator splitting method for (1) that gives the two equations

$$\partial_t f_i(\mathbf{x},t) = -v_i \cdot \nabla_{\mathbf{x}} f_i(\mathbf{x},t) \tag{2}$$

$$\partial_t f_i(\mathbf{x},t) = \alpha \sum_{j,k,l \in M_{\mathfrak{V}}} A^{k,l}_{i,j} \left(f_k(\mathbf{x},t) f_l(\mathbf{x},t) - f_i(\mathbf{x},t) f_j(\mathbf{x},t) \right) , \tag{3}$$

that have to be solved consecutively. We now discretize a bounded domain Ω of the position space $\mathbb{R}^n_{\mathbf{x}},\ n = 1,2,3$ in an equidistant manner,

$$\mathfrak{X} := \left\{ \mathbf{x} | x_j \in \Delta x \cdot \mathbb{Z}, j = 1,\dots,n, \mathbf{x} \in \Omega \subset \mathbb{R}^n_{\mathbf{x}} \right\} \; , \Delta x \in \mathbb{R}^+ \; .$$

The collision equation (3) is independent of the position variable $\mathbf{x}$, thus giving a straightforward way of parallelization, because this equation can be numerically solved in parallel for every point in the position space $\mathfrak{X}$. We solve the transport equation (2) with a first order finite difference (upwind) scheme over the equidistant position space, thus getting a Courant-Friedrich-Levy condition based stability bound for the transport time step Δt_T :

$$\Delta t_T \leq \frac{\Delta x}{\max\{c \in \mathbb{R}^+ | c = \|\mathbf{v}\|_\infty, \mathbf{v} \in \mathfrak{V}\}} \; . \tag{4}$$

And we solve the collision equation (3) with an adaptive second order explicit Runge-Kutta method. For such a method we generally get a stability bound around

$$\Delta t_k \leq \frac{1}{|\lambda_{\max}(t)|}, \tag{5}$$

where λ_{max} is the biggest eigenvalue of the linearization of the ODE (3) at time t with respect to its absolute value. In our numerical experiments it turned out, that (4) only posed a real restriction at high and transitional Knudsen numbers, whereas (5) dominated cases with low Knudsen numbers. Because we found out that phenomena like vortex shedding of a compressible mono atomic gas can be simulated with our model (and with only the above equations, no other modifications are needed), we recently got interested in low Knudsen numbers and will implement an implicit scheme in the near future for further investigations alongside a time adaptive error aware discretization of the position space $\mathfrak{X}(t)$. We also implemented the three different wall interactions of specular, inverse, and diffuse (temperature) reflection. These reflections can be used in any convex combination.

PARALLELIZATION

Our parallelization approach followed four main steps, and since we parallelized for CPUs and GPUs, the first two steps applied to both of them. Since the collision equation (3) can be solved for every point in the position space without dependencies to other points, the first two steps simply divide the position space into subspaces that can be dealt with in parallel.

(i) At first we divide the position space in n subspaces with minimal circumference. Here n corresponds to the number of CPU threads or the number of GPUs we want to use for our calculations. We choose subspaces with minimal circumference, because we want to minimize the costly communication between the involved processors.

(ii) Now, if we look at one subspace, we further divide it into m-point blocks. On modern CPUs m is equal to four or eight, because modern CPUs can do simple operations like $*, +, -$ in parallel on four (if the CPU is capable of the *Streaming SIMD Extensions* (SSE)) or eight (if the CPU has *Advanced Vector Extensions* (AVX)) floats in parallel. On the GPU side this partition corresponds to the fact, that a GPU consists of a number of *multiprocessors* (MPs) that are working mainly independent of each other. So these m-point blocks get distributed to the MPs and every MP is working on up to two of these blocks in parallel. For GPUs the m must be chosen very carefully to fully utilize the capabilities of a GPU, in fact m is a function of the hardware resources of the GPU like available registers, available shared memory and the maximal amount of threads per multiprocessor.

The last two parallelization steps are only necessary to suit the special hardware structure of NVIDIA® GPUs.

(iii) Because of the extremely limited resources (shared memory) of an MP, it is simply not possible to realize an efficient algorithm such that one of the *scalar processors* (SPs) that the MP consists of (a MP has 8 - 192 of them) calculates the right hand side ($\mathbf{J}$) of (3) for one point in the position space. Because of this we need another level of parallelization for the smallest operations, which determine the runtime of our algorithm. In our case these operations are clearly given by the number of collision relations (see Table 1). Because of this we use all SPs of one MP to calculate $\mathbf{J}$ for one point in the position space. Since we do this we can also use the fact, that one collision relation $(i,j,k,l) \in M_i$ has three permutations that occur within M_j, M_k, M_l. That can be used to quarter the necessary calculations. So the SPs typically do the following critical computation for the calculation of $\mathbf{J}$:

(a) Before the calculation begins, set $\mathbf{J} = 0$.

(b) The collision relations get distributed to the SPs and they calculate:

$$F := f_k f_l - f_i f_j \quad \Rightarrow \quad \begin{cases} J_i \leftarrow J_i + F \\ J_j \leftarrow J_j + F \\ J_k \leftarrow J_k - F \\ J_l \leftarrow J_l - F \end{cases}$$

(iv) The last parallelization step simply optimizes the memory bandwidth usage on the GPU. In this context we only highlight optimization strategies that may not be obvious to a GPU programmer. The set of pairwise distinct (with respect to the relation that is given by the three mentioned permutations) collision relations $\overline{\left|\bigcap_{i \in M_{\mathfrak{Y}}} M_i\right|}$ (which only has a quarter of the elements of $\left|\bigcap_{i \in M_{\mathfrak{Y}}} M_i\right|$) must be accessed very often, and it does not fit into the shared memory. When one SP accesses an element of this set, it has to read four consecutive 32 bit memory regions (representing integers). Because of this we convert it to texture memory with a special int4 data type (that consists of four integers corresponding to (i,j,k,l) of a collision relation) to get the advantage of faster, cached reads and to minimize the necessary transfer operations. This greatly decreased the necessary global memory bandwidth. The second step to optimize the memory bandwidth is to order the collision relations (i,j,k,l) (under usage of their possible permutations) in such a way, that the so called shared memory access collisions can

mainly be avoided. This greatly increases the shared memory throughput in our simulations and lead to 33% lower computation times.

For the following experiments and CPU / GPU comparisons we used the GCC compiler version 4.5.3 with the optimization flags *-march=native -O3* for the CPU part and the CUDA toolkit version 4.1 with the optimization flags *-O3 -arch sm_20* for the GPU part of the program. All calculations were done in float precision. Beside the main program, which does the calculations, we developed a *graphical user interface* (GUI) that uses OpenGL for a visualization and the interactive creation of position spaces and discretization of the velocity space. It is possible to create three dimensional position spaces with complex objects inside it. The GUI also possesses the capability of data inspection of the macroscopic values, automatic picture and video creation, calculation of streak and stream lines and three dimensional visualization via visualization of the macroscopic values on planes in the position space. The used hardware consists of a Core I7 960 CPU and three Geforce GTX 580 GPUs. For the speed comparison between CPU and GPU we used simulations on only one of the GPUs for the sake of comparability. All three GPUs were used for the numerical experiments in the next section. We used the problem of a three dimensional, Mach one gas flow through a pipe with Knudsen numbers in the range [0.04,0.2] and position space grids that contain between 65536 and 1048576 points. In Figure 1 we can see a comparison of four different versions of the algorithm in such a way that we see the speedup factor over the size of the discretized velocity space. At first we see, that an SSE parallelized version, that only uses one core of the CPU, of our algorithm is around three times as fast as a not parallelized version. If we use the SSE together with the utilization of all four cores and hyper-threading (eight virtual cores, which can be utilized through the usage of 16 threads) on this CPU we get an additional speedup of around 6.5 thus giving us a total speedup of around 20 for an efficient CPU parallelized implementation. As we can see the GPU is another six to ten times faster than the whole CPU (at least at such “small” velocity spaces with up to 123 velocities).

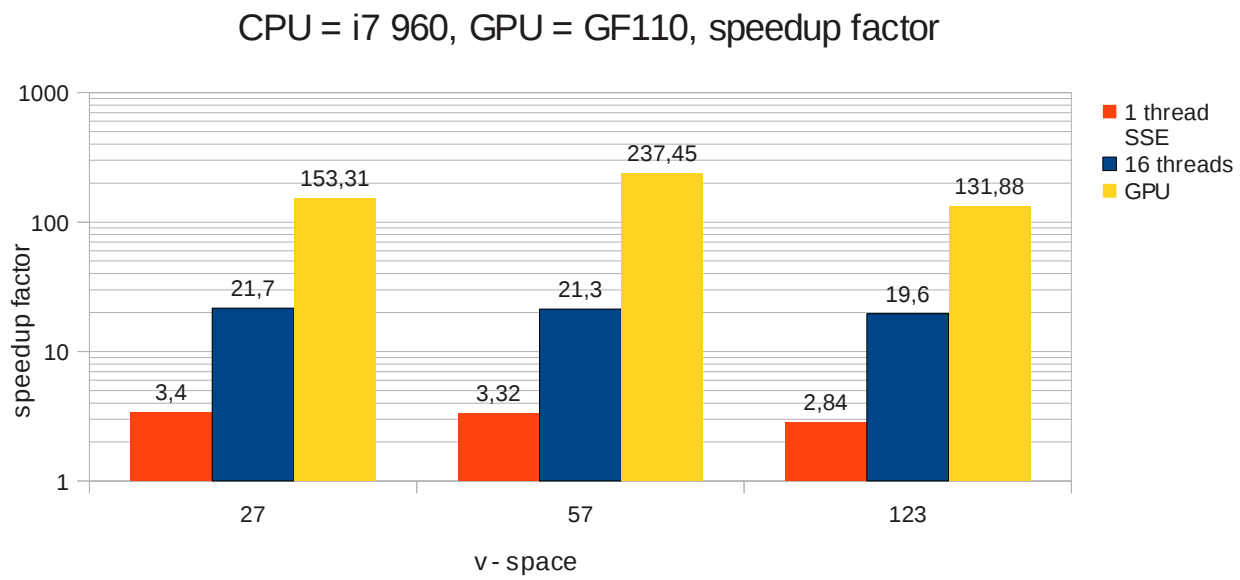

FIGURE 1. speed comparison of not parallelized CPU vs SSE CPU vs parallelized SSE CPU vs GPU version

An important thing we have to mention here is that a GPU implementation is much more complicated than a simple (not parallelized) CPU implementation and because of this the development of such programs simply need more time. So a GPU implementation should always be compared with a parallelized and optimized CPU implementation that was developed with a comparable or at least an adequate amount of time. Unfortunately only a fraction of the scientific GPU programmers is following this idea and because of this there are always publications in which the authors claim to get speedups of around two magnitudes or more compared to recent CPUs. As one can easily see, parallelization on one Intel® processor already gives speedups of more than one magnitude. Because of this we look at a realistic comparison of GPUs and CPUs, that means we compare a parallelized and optimized CPU implementation that uses SSE with an optimized (regarding memory access restrictions and algorithm redesigning to fit into the Single Instruction Multiple Data approach) GPU implementation (see Figure 2). As we can see the GPU implementation is much better on big velocity spaces. That is due to parallelization steps (iii) and (iv) and the fact, that level one and level two cache misses

increase significantly for such large velocity spaces on the CPU (because the necessary data for one position space point simply don't fit anymore in these caches). The main parallelization on the GPU is focused on a huge amount of collision relations, where shared memory collisions can be avoided through reordering of the collision relations and where every SP has to calculate many operations that correspond to collision relations. Since our aim was to create an efficient implementation for big velocity spaces, we are satisfied with these results. An implementation that focuses on small velocity spaces would probably look very different. If we want to compare the GPU with a single threaded not optimized (in the context of parallelization) CPU version we can look at the speedup factor at $|\mathfrak{V}| = 251$ and multiply it with the CPU speedup factor of 20 giving a total speedup of around 600 (which could maybe be reduced by optimizing the CPU version with respect to the usage of the CPU cache).

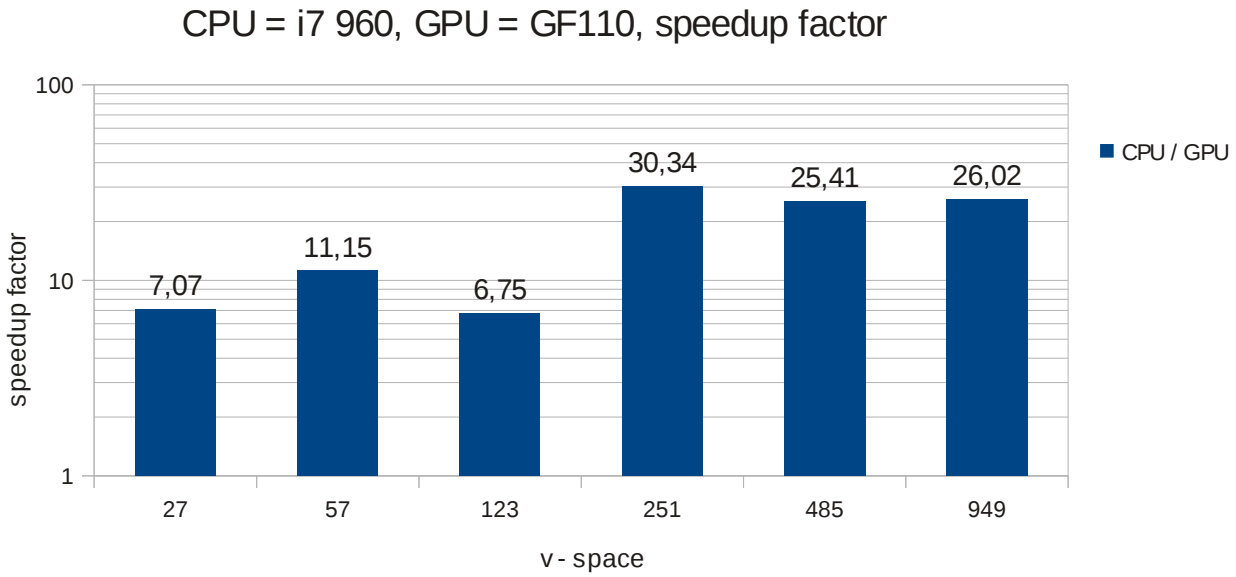

FIGURE 2. speed comparison of fully parallelized CPU vs GPU version

EXPERIMENTS

We will now get a first impression of which physical phenomena can be investigated through this approach. The titles of the following pictures have the same structure and consist of comma-separated informations:

$$\overbrace{\text{2D261}}^{(i)},\ \overbrace{512\times 512,\ 512}^{(ii)},\ \overbrace{\text{scaling = cons.}}^{(iii)},\ \overbrace{\text{calc-time} = 0:2:53:34}^{(iv)},\ \overbrace{\text{sim-time} = 1403.76\mu\text{s}}^{(v)}$$

(i) This is the dimensionality of the velocity space and number of velocities.

(ii) This is the number of points in the position space in each direction (x,y,z) and number of time steps.

(iii) If the ratio between discretization points in x and y direction corresponds to the hight / length ratio of the picture we call it a constrained scaling otherwise it's an unconstrained scaling.

(iv) This is the time that was needed to finish the simulation in days:hours:minutes:seconds.

(v) This is the simulation time (time within the simulation) at which the picture was taken.

All simulations have these initial parameters: pressure 1.43587 Pa, temperature 273.15 K, density $1.83348 \cdot 10^{-5} \frac{\text{kg}}{\text{m}^3}$.

Knudsen Pump

A well known rarefied gas effect is thermal creep flow, a flow induced by a gradient of the wall temperature. In the case of moderate to large Knudsen numbers this effect can be used to create the so called Knudsen pump cf. [11]. We

now choose a Knudsen number of 0.25 on a two dimensional 512 x 512 grid in the position space with 261 velocities in the velocity space. We have two reservoirs of gas, one on the left and one on the right. These reservoirs initially have the same pressure and they are connected via a canal with linear decreasing (increasing) temperatures inside (resp. outside) the rectangular ditches at the lower canal wall. All other walls in this arrangement possess specular reflection. We would anticipate a one-sided flow through the channel. Figure 3 shows the expected result of an induced flow through the connecting canal. The flow has been visualized by the calculation of lines that fit onto the impulse vector in each point (also known as stream lines).

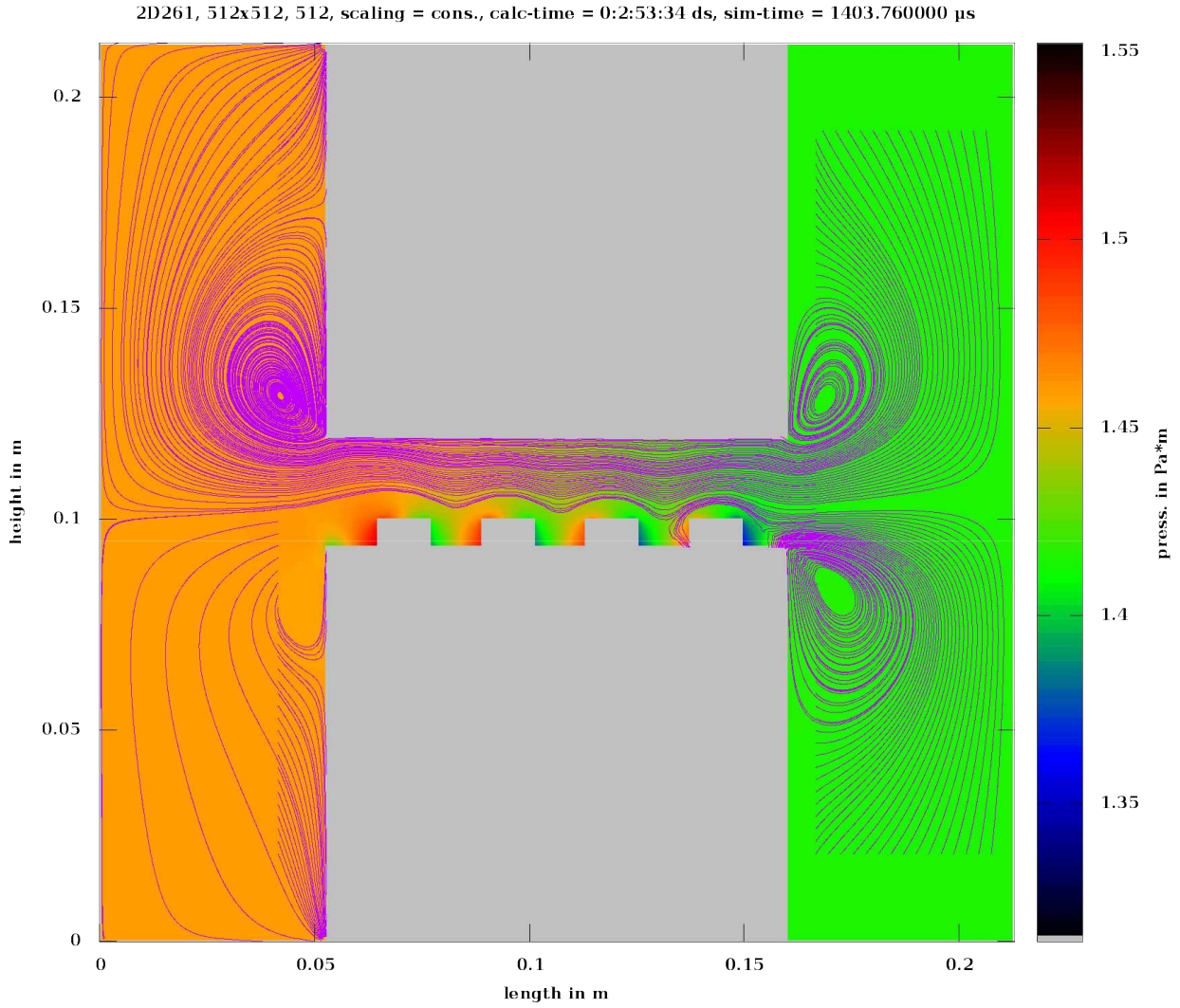

FIGURE 3. final picture of a simulation of a Knudsen pump with stream lines

Kàrmàn vortex street

Another well known effect of fluid dynamics is the Kàrmàn vortex street that appears when fluids are flowing around an obstacle. We did a three dimensional simulation with 960×480 points in the position space (periodic in the z direction), 147 points in the velocity space, a Knudsen number of 0.002 and a Raynolds number around 500. The position space consists of two infinite long walls in the x,z plane at the upper and lower end with specular reflection. So we are looking at one x,y plane. On the left side we have "input" points from where a gas with Mach two originates. On the right hand side we have "output" points where the gas can disappear. We placed three obstacles with inverse reflection in the left side of the canal (cylinders). Figure 4 shows the expected result of a vortex street within the canal.

The vortexes have been visualized by the calculation of the movement of virtual particles that have been inserted at the left side of the channel (also known as streak lines).

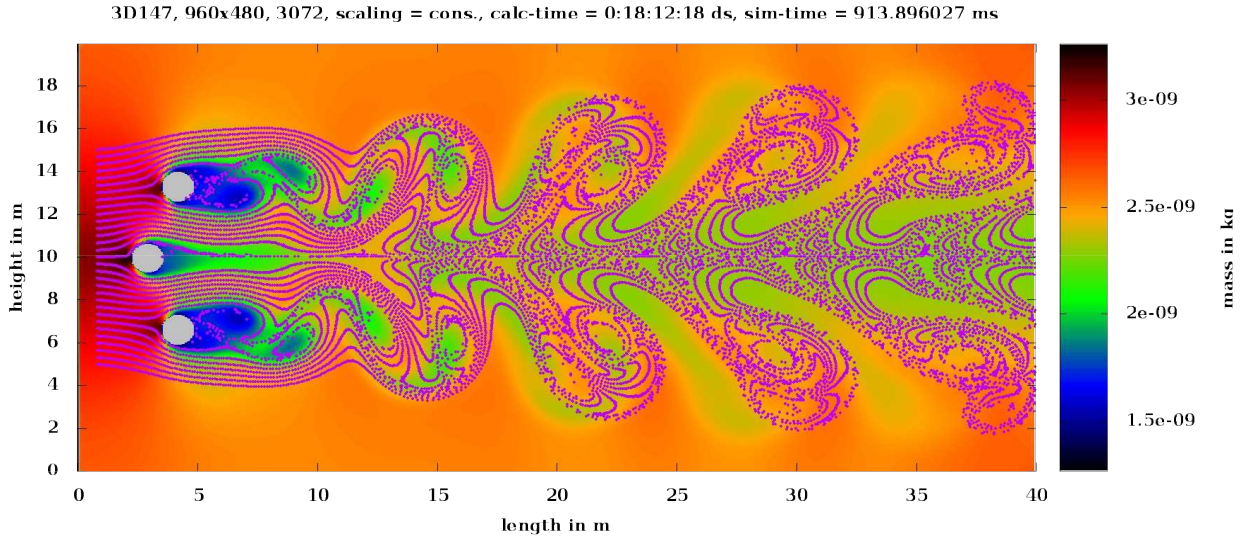

FIGURE 4. final picture of a simulation of a vortex street with streak lines

Supersonic flow

Supersonic flows are gas flows with an average particle velocity $\overline{\mathbf{v}}$ between Mach 1.2 and Mach 5. We have simulated a typical test case of a gas flow between two infinite long plates at Mach two with a triangular obstacle on one canal wall. The simulation is three dimensional in the velocity and in the position space. So we have two infinite long walls in the x,z plane and periodic conditions in the z direction i.e. we are looking at one x,y plane. The walls have specular border conditions and the obstacle has inverse border conditions. At the left hand side we have "input" points from which a steady gas flow at Mach four originates. At the right hand side of the canal we have "output" points where the mass disappears (of course with some special border conditions). The Knudsen number for this experiment was 0.000625. As we can see in Figure 5 the simulation reveals a clear shock front originating from the obstacle. The shock structures reflect the triangular shape of the obstacle in the canal. This is in good agreement with the expected result. In this example we can see that our simulation needed more than two days to solve this problem with such a small Knudsen number. In the same situation a conventional gas dynamics algorithm would need a few minutes or less. We included this simulation to show that we can do simulations in Knudsen regimes that overlap with the typical regimes of conventional schemes, thus offering the possibility to develop hybrid schemes that use our model in regions with high Knudsen numbers.

CONCLUDING REMARKS AND FUTURE PERSPECTIVES

We have demonstrated that the development in computing technology within the last ten years led to processors that are powerful enough to handle algorithms that arise from direct deterministic approaches to the Boltzmann equation (like the used LGpMs). It is now possible to investigate phenomena in relatively complex position spaces through the Boltzmann equation and this model in the transitional and continuum regime. But here one has to consider that the underlying equations were developed with the idea of a rarefied gas in which the binary collisions of particles prevail and that in the continuum regime the cell size is typically larger than the mean free path (what could lead to unphysical solutions or artificial effects). This model is not intended to replace or compete with well-established numerical simulation tools like LBM or DSMC methods, because our approach is very expensive and slow in terms of computing time compared to those and there is a number of real gas effects which are not easy to implement in LGpMs. However, LGpMs present an alternative approach to gas flows that is able to describe complex physical phenomena like the vortex street on the basis of the Boltzmann equation and without the introduction of any other equations or models. It will be a near future aim to use the developed model to investigate some physical effects like

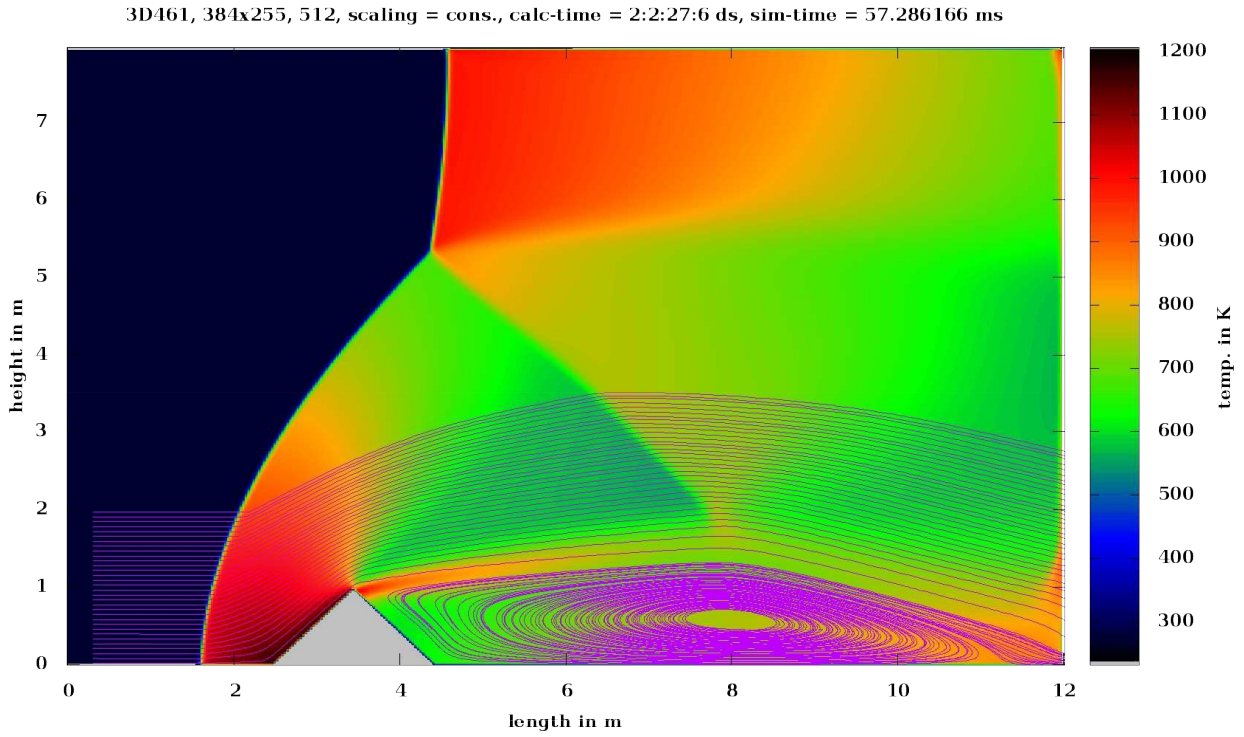

FIGURE 5. final picture of a simulation of a supersonic flow between two plates

the transition from flows without vortex shedding to flows with vortex shedding (and maybe transition to turbulent behavior), Knudsen pumps and even Rayleigh - Bènard convection (that has successfully been simulated). Because of this we will implement a position space adaptive solver for equation (2), in the hope that this could lead to a much more detailed dynamic in the vortex streets. And we will implement an implicit solver for (3), so that we can simulate lower Knudsen numbers. Another near future aim is the comparison of the obtained results with results originating from other models and even physical experiments.

REFERENCES

1. H. Babovsky, "Kinetic models on orthogonal groups and the simulation of the Boltzmann equation," in *Proceedings of 26th International Symposium on Rarefied Gas Dynamics*, edited by T. Abe, AIP Conference Proceedings 1084, 2009, pp. 415–420.
2. H. Babovsky, *Comput. Math. Appl.* **58**, 791–804 (2009).
3. T. Płatkowski, and R. Illner, *SIAM Review* **30**, 213–255 (1988).
4. V. A. Panferov, and A. G. Heintz, *Math. Methods Appl. Sci* **25**, 571–593 (1999).
5. C. Buet, *Transport Theory and Statistical Physics* **25**, 33–60 (1996).
6. P. Bailey, J. Myre, S. D. C. Walsh, D. J. Lilja, and M. O. Saar, "Accelerating Lattice Boltzmann Fluid Flow Simulations Using Graphics Processors," in *Proceedings of the 2009 International Conference on Parallel Processing*, ICPP '09, IEEE Computer Society, Washington, DC, USA, 2009, pp. 550–557, ISBN 978-0-7695-3802-0.
7. A. Frezzotti, G. Ghiroldi, and L. Gibelli, "Direct solution of the Boltzmann equation for a binary mixture on GPUs," in *American Institute of Physics Conference Series*, 2011, vol. 1333, pp. 884–889.
8. E. Malkov, and M. Ivanov, "Parallelization of algorithms for solving the Boltzmann equation for GPU-based computations," in *American Institute of Physics Conference Series*, 2011, vol. 1333, pp. 946–951.
9. I. Kampolis, X. Trompoukis, V. Asouti, and K. Giannakoglou, *Computer Methods in Applied Mechanics and Engineering* **199**, 712 – 722 (2010).
10. Y. Zhao, *The Visual Computer* **24**, 323–333 (2008), 10.1007/s00371-007-0191-y.
11. H. Babovsky, "Kinetic Lattice Group Models: Structure and Numerics," in *American Institute of Physics Conference Series*, 2011, vol. 1333 of *American Institute of Physics Conference Series*, pp. 63–68.