C. R. Mecanique 342 (2014) 619–635

Contents lists available at ScienceDirect

Comptes Rendus Mecanique

www.sciencedirect.com

Theoretical and numerical approaches for Vlasov–Maxwell equations

Reduced Vlasov–Maxwell simulations

Philippe Helluy [a,*], Laurent Navoret [a], Nhung Pham [a], Anaïs Crestetto [b]

[a] IRMA, Université de Strasbourg & Inria TONUS, 7, rue René-Descartes, 67084 Strasbourg cedex, France
[b] LMJL, Université de Nantes, 2, rue de la Houssinière, BP 92208, 44322 Nantes cedex 3, France

ARTICLE INFO

Article history:
Received 25 March 2014
Accepted 8 June 2014
Available online 6 August 2014

Keywords:
Vlasov–Maxwell
Model reduction
Particle-In-Cell
Discontinuous Galerkin
GPU

ABSTRACT

In this paper we review two different numerical methods for Vlasov–Maxwell simulations. The first method is based on a coupling between a Discontinuous Galerkin (DG) Maxwell solver and a Particle-In-Cell (PIC) Vlasov solver. The second method only uses a DG approach for the Vlasov and Maxwell equations. The Vlasov equation is first reduced to a space-only hyperbolic system thanks to the finite-element method. The two numerical methods are implemented using OpenCL in order to achieve high performance on recent Graphic Processing Units (GPU).

0. Introduction

The Maxwell–Vlasov system is a fundamental model in physics. It can be applied to plasma simulations, charged particles beam, astrophysics, etc. The unknowns are the electromagnetic field, solution to the Maxwell equations and the distribution function, solution to the Vlasov equation. The two systems of equations are coupled because the motion of particles generates an electric current at the right-hand side of the Maxwell equations, while the electromagnetic field accelerates the particles in the Vlasov equation. The Maxwell equations are a system of linear hyperbolic equations. Today, they are routinely solved in industrial applications with commercial software. Several numerical methods exist: finite difference, finite elements. In this paper, we will use a more and more popular method for solving the Maxwell equations: the Discontinuous Galerkin (DG) finite element approach. This method is presented in many works. We refer for instance to Helluy [1], Bourdel et al. [2], Cohen et al. [3], Klöckner et al. [4], Crestetto [5].

The Vlasov equation is a rather simple transport equation, but set in a six-dimensional (x, v) space-velocity phase space. This leads to very heavy computations. The most popular method for solving the Vlasov equation is thus the Particle-In-Cell (PIC) method of Birdsall and Langdon [6], because this is one of the less expensive. It consists in distributing random particles with random velocities in the computational domain. The particles are then pushed by the electromagnetic field. They are also deposited on the Maxwell finite-element mesh in order to generate a current in the right-hand side of the Maxwell equations.

The PIC method is very easy to implement. However it is subject to numerical noise. It leads also to other issues: smoothing, charge conservation errors, energy conservation errors. We will also see in this paper that the PIC method is rather difficult to parallelize.

Therefore, while they are certainly more memory consuming, Eulerian approaches, which solve the Vlasov equation directly on a phase-space grid, are more and more investigated.

* Corresponding author.
E-mail addresses: philippe.helluy@unistra.fr (P. Helluy), laurent.navoret@math.unistra.fr (L. Navoret), pham@math.unistra.fr (N. Pham), anais.crestetto@univ-nantes.fr (A. Crestetto).

http://dx.doi.org/10.1016/j.crme.2014.06.008
1631-0721/

In this paper, we first review some aspects of the DG and PIC methods, which are very important for obtaining a robust and precise approximation. We then describe a parallelization of the full Vlasov–Maxwell coupling on recent Graphic Processing Units (GPU) with the OpenCL framework.

We will see that the DG method is very well adapted to parallelization. The PIC method is more difficult to efficiently parallelize. Therefore, we will present a recent approach, the reduction method, which allows approximating the Vlasov equation also by a DG solver.

1. Vlasov–Maxwell equations

1.1. Maxwell equations

In this paper, we consider the two-dimensional Maxwell equations in Transverse Magnetic (TM) mode. The unknowns depend on the space variable $x = (x_1, x_2) \in \mathbb{R}^2$ and the time $t \geq 0$. They are the electric field

$$E = (E_1, E_2, 0)^{\mathrm{T}}$$

and the magnetic field

$$H = (0, 0, H_3)^{\mathrm{T}}$$

The current

$$j = (j_1, j_2, 0)$$

is supposed to be given.

We then write the unknowns and the source term in a vector form

$$W = (E_1, E_2, H_3)^{\mathrm{T}}, \qquad S = (-j_1, -j_2, 0)^{\mathrm{T}}$$

In this way, the Maxwell equations read as a linear first-order hyperbolic system

$$\partial_t W + A^i \partial_i W = S \tag{1}$$

where we use the Einstein convention (sum on repeated indices)

$$A^i \partial_i = A^1 \partial_1 + A^2 \partial_2$$

and the notation

$$\partial_i = \frac{\partial}{\partial x_i}$$

In the first-order differential system (1), the matrices A^i are given by

$$A^1 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}, \qquad A^2 = \begin{pmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{pmatrix}$$

(the equations are written under a dimensionless form where the speed of light $c = 1$). Let now $n = (n_1, n_2)^{\mathrm{T}} \in \mathbb{R}^2$. We can also define the flux of the Maxwell equations:

$$f(W, n) = A^i n_i W$$

If we define

$$n^1 = (1, 0)^{\mathrm{T}}, \qquad n^2 = (0, 1)^{\mathrm{T}}$$

the Maxwell equations can also be written under the more general conservative form:

$$\partial_t W + \partial_i f\left(W, n^i\right) = S \tag{2}$$

1.2. Vlasov equation

We consider now the motion of N particles of mass m and charge q in the electromagnetic field. The particles are labeled by an index k, $1 \leq k \leq N$. The position of particle k at time t is $x^k(t)$ and its velocity is

$$\dot{x}^k(t) = \frac{\mathrm{d}}{\mathrm{d}t} x^k(t)$$

For $x = (x_1, x_2) \in \mathbb{R}^2$ and $v = (v_1, v_2) \in \mathbb{R}^2$, the distribution function of particles is defined by

$$f(x, v, t) = \sum_k \omega_k \delta\big(x - x^k(t)\big)\delta\big(v - \dot{x}^k(t)\big)$$

where δ denotes the Dirac measure on $\mathbb{R}^2$ and ω_k is the weight of particle k. The electric current generated by the particles motion is given by

$$j(x, t) = \int_v f(x, v, t) v \,\mathrm{d}v = \sum_k \omega_k \delta\big(x - x^k(t)\big)\dot{x}^k(t) \tag{3}$$

The particle acceleration is given by the relativistic equation of motion

$$\dot{x} = v, \qquad \ddot{x} = \mu \frac{q}{m}(E + v \wedge H) \tag{4}$$

with

$$E = (E_1, E_2, 0)^{\mathrm{T}}, \qquad H = (0, 0, H_3)^{\mathrm{T}}$$

$$\mu = \mu(v) = (1 - v \cdot v)^{1/2}(Id - v \otimes v) \tag{5}$$

where $Id - v \otimes v$ denotes the projection to the orthogonal space to v. We can also write the acceleration with the notation

$$\ddot{x} = \mu(\dot{x}) a(x, \dot{x}, t), \qquad a(x, v, t) = \frac{q}{m}\big(E_1(x, t) + v_2 H_3(x, t), E_2(x, t) - v_1 H_3(x, t)\big)$$

It is then possible to prove that, in the weak sense, the distribution function satisfies the relativistic Vlasov equation written under a conservative form

$$\partial_t f + \nabla_x \cdot (f v) + \big(\mu(v) a\big) \cdot \nabla_v f = 0 \tag{6}$$

When the particle velocity is small compared to the speed of light $c = 1$, we can use the Galilean approximation

$$\mu(v) \simeq 1 \tag{7}$$

In addition, we introduce the admissible velocity disk

$$D = \big\{v \in \mathbb{R}^2, v \cdot v < 1\big\}$$

Let us also observe that on the boundary of the velocity disk, we have:

$$v \in \partial D \quad \Rightarrow \quad \mu(v) = 0$$

This is a nice property. Indeed, the Vlasov equation (6) is a transport equation written in the (x, v) velocity phase space. The phase space transport velocity is

$$V = (v, \mu a) \in \mathbb{R}^4$$

The component of V in the velocity direction v is the acceleration μa, which vanishes on ∂D. Thus we will have no boundary condition to apply on ∂D, or more precisely, if at the initial time $f = 0$ on ∂D, then this property is maintained at all times.

1.3. Divergence cleaning

The charge $\rho(x,t)$ is defined by

$$\rho(x,t) = \int_v f(x,v,t)\mathrm{d}v$$

Integrating the Vlasov equation with respect to v, we obtain the charge conservation:

$$\partial_t \rho + \nabla_x \cdot j = 0 \tag{8}$$

If at the initial time $t=0$ the Gauss law is satisfied

$$\nabla_x \cdot E(x,t=0) = \rho(x,t=0)$$

then, using the charge conservation (8) and the Maxwell equations (1) we deduce the Gauss law at all times

$$\nabla_x \cdot E = \rho \tag{9}$$

The Gauss law is thus a consequence of the Vlasov–Maxwell equations. However, depending on the numerical scheme, it might not be well satisfied by the numerical approximation. A practical tool for improving the numerical accuracy is to use a divergence cleaning technique of Munz et al. [7]. The divergence cleaning consists in considering an additional artificial unknown $\phi(x,t)$ in the Maxwell equations, which satisfies:

$$\partial_t \phi + \chi \nabla_x \cdot E = \chi \rho$$

The constant parameter $\chi > 0$ represents the speed at which the divergence errors are propagated to the boundaries of the computational domain. In addition, the time derivative of the electric field $\partial_t E$ is replaced by $\partial_t E + \chi \nabla_x \phi$ in the Maxwell equation. We thus obtain a new vector of unknowns

$$W = (E_1, E_2, H_3, \phi)^{\mathrm{T}}$$

The divergence cleaning model still reads as a hyperbolic system (1), but the matrices are now

$$A^1 = \begin{pmatrix} 0 & 0 & 0 & \chi \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ \chi & 0 & 0 & 0 \end{pmatrix}, \qquad A^2 = \begin{pmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & \chi \\ -1 & 0 & 0 & 0 \\ 0 & \chi & 0 & 0 \end{pmatrix}$$

and the source term becomes

$$S = (-j_1, -j_2, 0, \chi\rho)^{\mathrm{T}} \tag{10}$$

An important feature of this extended Maxwell system is that we recover exactly the Maxwell equations when ϕ is constant. Thus, with adequate boundary conditions, we introduce only a numerical stabilization of the divergence errors without additional numerical errors, even for small values of χ. For more details on the divergence cleaning system, we refer to Munz et al. [7], Crestetto and Helluy [8], Crestetto [5], Fornet et al. [9].

1.4. Boundary conditions

The Vlasov–Maxwell equations (2), (6) need to be supplemented by conditions at the boundary of the computational domain $\Omega \times D$.

1.4.1. Maxwell boundary conditions

Stable boundary conditions for the classic Maxwell equations have been extensively studied. These conditions are properly generalized to the divergence cleaning model in a few papers: Fornet et al. [9], Crestetto and Helluy [8]. We recall here briefly the general theory.

We consider local boundary conditions in the form

$$M(n)\big(W - W^{\mathrm{inc}}\big) = 0 \tag{11}$$

where $n = (n_1, n_2, 0)$ is the normal outward vector on $\partial\Omega$ and W^{inc} a given incident boundary electromagnetic field (which can be zero). The boundary condition has to be chosen in such a way that the Maxwell operator in space x is maximal positive. Stability conditions are given by the Lax–Philips theory (Lax and Phillips [10], Petkov and Stoyanov [11], Bourdel et al. [2], Helluy [1], Crestetto and Helluy [8], Fornet et al. [9]) which requires:

- $\frac{1}{2}A^i n_i + M(n) \geq 0$.
- $\dim\ker A^i n_i^- = \dim\ker M(n)$.

It is possible to prove that the following boundary conditions (and associated M matrices) satisfy the Lax–Philips stability conditions:

- generalized "metal":

$$n \times E = 0, \qquad \phi = \lambda E \cdot n, \quad \lambda \geq 0 \tag{12}$$

- generalized "Silver-Müller", $M = -A^i n_i^-$:

$$\begin{aligned} &H_3 - n_1 E_2 + n_2 E_1 = H_3^{\text{inc}} - n_1 E_2^{\text{inc}} + n_2 E_1^{\text{inc}} \\ &E \cdot n - \phi = E^{\text{inc}} \cdot n \end{aligned} \tag{13}$$

The generalized metal condition is compatible with the original Maxwell system only when $\lambda = 0$ (because we need to have $\phi = 0$). However, a small $\lambda > 0$ can be interesting for numerical reasons, because it introduces a slight energy damping.

We would like to emphasize that the respect of the Lax–Philips stability condition is absolutely crucial for obtaining stable and precise numerical results, especially when the DG solver is coupled to a PIC solver.

1.4.2. Vlasov boundary conditions

As seen in Section 1.2, no boundary condition is required on the boundary $\Omega \times \partial D$ of the velocity domain. We thus consider the case $x \in \partial\Omega$ and $v \in D$. The outward normal vector on $\partial\Omega$ is still noted $n(x)$.

Inflow condition It is natural to impose the value of the distribution function f only at inflow (Johnson and Pitkäranta [12]). Introducing the notations

$$\alpha^+ = \max(\alpha, 0), \qquad \alpha^- = \min(\alpha, 0)$$

the inflow condition can be written

$$\big(v \cdot n(x)\big)^- f(x, v, t) = (v \cdot n)^- f_0(x, v, t)$$

in such a way that when $v \cdot n > 0$, no condition is imposed on f.

Child–Langmuir condition A more subtle boundary condition is the Child–Langmuir current condition. This condition is useful at a particle-emitting boundary because it allows creating just the quantity of charges that cancels the normal component of the electric field. For the moment we do not know how to express in a rigorous mathematical way the Child–Langmuir condition. We just describe the practical computation.

The electrons are emitted at the cathode if the normal electric field is strong enough, until it cancels (Child–Langmuir law).

More precisely, we use the following algorithm for a discretization cell L that touches the cathode boundary Γ_C:

- if $E \cdot n < 0$ on $\partial L \cap \Gamma_C$, then compute

$$\delta_L = \rho_L - \int_{\partial L \setminus \Gamma_C} E \cdot n$$

  where $\rho_L = \sum_{x^k \in L} \omega_k$ (charge in the cell L).
- if $\delta_L < 0$, create n_e random particles in the cell L with weights δ_L / n_e.

2. Discontinuous Galerkin method

2.1. Weak upwind DG formulation

The Discontinuous Galerkin (in short DG) approximation is a more and more popular method for approximating hyperbolic systems of conservation laws (see, among many others, Bourdel et al. [2], Crestetto [5], Crestetto and Helluy [8], Cohen et al. [3], Klöckner et al. [4], Lesaint and Raviart [13]).

We consider a mesh of the domain Ω. In each cell L, the fields $W(x,t)$ are approximated by a second-order polynomial in x. We denote by $P_2(\mathbb{R}^2)$ a linear space of second-order polym.Mesh(c12, c13, c0, c15, c16, c17, c4, c18, c19, c20, c9, c8); nomial in $x = (x_1, x_2)$. In practice, we use $P_2(\mathbb{R}^2) = \text{span}\{1, x_1, x_2, x_1 \cdot x_2, x_1^2, x_2^2, x_1^2 \cdot x_2, x_1 \cdot x_2^2\}$ because with this choice, we have $\dim P_2(\mathbb{R}^2) = 8$, which is well suited to GPU optimizations. Then:

$$W_L(x,t) = \sum_j w_{L,j}(t)\psi_{L,j}(x), \quad \{\psi_{L,j}\} \text{ basis of } P_2\big(\mathbb{R}^2\big)^4$$

The DG upwind weak formulation (Lesaint and Raviart [13], Johnson and Pitkäranta [12], Helluy [1]) consists in finding the basis components $w_{L,j}(t)$ in each cell L such that for all test function $\psi_L \in P_2(\mathbb{R}^2)^3$

$$\int_L \partial_t W_L \cdot \psi_L - \int_L W_L \cdot A^i \partial_i \psi_L + \int_{\partial L\cap\Omega} \big(A^i n_i^+ W_L + A^i n_i^- W_R\big)\cdot\psi_L + \int_{\partial L\cap\Omega} \big(M + A^i n_i\big) W_L \cdot \psi_L = \int_L S\cdot\psi_L + \int_{\partial L\cap\Omega} M W^{\text{inc}}\cdot\psi_L \tag{14}$$

where we denote by n the normal vector on ∂L oriented from the cell L to the neighboring cells R and

$$x^+ = \max(0,x), \qquad x^- = \min(0,x)$$

The DG formulation is a generalization of the finite volume method. It relies on the standard upwind numerical flux for linear hyperbolic systems:

$$f(W_L, W_R, n) = A^i n_i^+ W_L + A^i n_i^- W_R$$

Finally, (14) is equivalent to a system of ordinary differential equations for the $w_{L,j}(t)$.

We do not give all the details of the implementations, but for our application, the main lines are:

- the cells L are quadratic curved "quadrilaterals";
- the components of the basis functions $\psi_{L,j}$ are orthonormal polynomials on the cell L when the cell is a parallelogram: we use a modal basis defined directly in the physical space. We do not rely on a reference element. Thanks to this choice we will not have to invert a geometric transformation for computing the fields at the particles;
- we use a high-order numerical integration (16 Gauss–Legendre quadrature points in the cells and 4 points on each edge).

For more details, we refer to Crestetto and Helluy [8].

2.2. GPU parallelization

The DG method can be parallelized efficiently on a Graphic Processing Unit (GPU) (Klöckner et al. [4]). GPUs are recent computing devices that have proven to be very efficient for performing computations on data that can be regularly organized into the GPU memory.

GPUs are not as easy to program as classic processors. CUDA is a well known environment for programming NVIDIA GPUs. OpenCL is another framework for programming various multicore devices, including GPUs or CPUs of several vendors. OpenCL means "Open Computing Language". It includes a library of C functions, called from the host, in order to drive the GPU and a C-like language for writing the programs that will be executed on the processors of the multicore accelerator. The specification is managed by the Khronos Group [14].

OpenCL proposes a rather general abstraction that works well for various multicore SIMD hardware. Very schematically, an OpenCL device possesses a few gigabytes of global memory and is made of a few tens of Computing Units (CU). Each CU contains a few processors called Processing Elements (PE), and a small cache memory of a few kilobytes. The same program, called a kernel, can be executed on all the Processing Elements at the same time. The PEs have a very fast access to the cache memory of their CU. The PEs have also an efficient access to the GPU global memory if they read or write to adjoining memory locations. For non-regular computations, a classic strategy is thus first to fetch a tile of data into the CU cache, then to perform the computations with fast access to the cache. When the computations are finished, they are copied back, in a regular way to the global memory. A special behavior of OpenCL devices makes the GPU programming rather complicated: if two processors try to write at the same memory location at the same time, only one will succeed... This has to be kept in mind, for instance in the flux collecting algorithm or when computing the current created by the particles in the same cell L.

We have written an OpenCL implementation of the previous DG formulation. Our programming strategy is described in details in Crestetto and Helluy [8]. We just recall here the principal points:

- initialization: we compute and invert the local mass matrices on the CPU. We send (all) the data to the GPU;
- first pass of each time step: we associate with each Gauss point of each edge one processor. We compute the flux at the Gauss points and store it into global memory;
- second pass: we associate with each basis function of each element a processor. We compute the time derivative of the $w_{L,j}$ using the DG weak formulation, the previously computed fluxes and the stored inverted mass matrices. The separation into two passes with parallelism redistribution allows avoiding concurrent writing operations;
- time integration: we use a simple second order Runge–Kutta scheme.

According to some benchmark that we have performed, we observed a spectacular efficiency of the GPU implementation. Compared to a single core implementation on a traditional CPU, we have observed that the GPU implementation is 50–100 times faster (Crestetto and Helluy [8]).

3. PIC method

3.1. Generalities on the PIC method

The Particle-In-Cell method is a very natural method (Birdsall and Langdon [6]) for approximating the Maxwell–Vlasov system (1), (6) because it starts directly from the particle interpretation of the Vlasov equation (4). The idea is simply to move the particles with a standard ordinary differential equation solver. This requires the computation of the electromagnetic field at the particle positions. We have thus to know at each time step to which element belongs each particle. Reversely, the motion of the particles produces an electric current measure given by (3).

The action of the fields over the particles is rather easy to parallelize.

The action of the particles on the current at the right-hand side of the Maxwell equations is less obvious to program. Indeed, a naive parallelization would lead to concurrent memory writing. Recent GPU OpenCL drivers allow memory locking and atomic operations, but it is not recommended to use these features because they generally lead to a dramatic drop of performance. Therefore, we prefer to adopt a more sophisticated sorting technique (presented for instance in Aubert et al. [15]) in order to achieve faster computations.

Let us mention that we have employed a very simple PIC method. Many authors have observed much better precision of the PIC algorithm when the particles are smoothed. Smoothing is considered to be very important for stability, precision and charge conservation, especially when the electromagnetic field is computed by a discontinuous approximation. Unfortunately, smoothing also requires atomic operations and thus an even more complicated GPU implementation. Therefore, we have decided to test the rough PIC method anyway. Surprisingly, we have been able to obtain rather precise results. The condition for obtaining these results is to use a high divergence cleaning parameter $\chi \simeq 10$. We will discuss the consequences of this choice in Section 4.

3.2. GPU implementation

We give some details on the GPU implementation for one time step. More information is given in Crestetto and Helluy [8].

3.2.1. Particles motion

- Emission: we use the algorithm described in the last paragraph of Section 1.4.2. The random positions are given by independent van der Corput sequences.
- Particle acceleration: at each time step we associate one processor with each particle. We move the particle and find its new cell location. Our algorithm works on an unstructured grid, but for efficiency, we assume that during one time step the particle cannot cross more than one cell layer. This condition imposes a CFL condition that is not constraining compared to the DG solver CFL condition.

3.2.2. Current

This is the most subtle part of the GPU algorithm because we have to avoid concurrent memory write operations. This difficulty has been already addressed by several authors (see, for instance, Aubert et al. [15]). The most efficient solutions generally rely on a particles sorting pass at each time iteration.

- We thus first sort the list of particles according to their cell numbers. For this sorting, we use a GPU-optimized radix sort algorithm of Helluy [16]. Then, it is easy to know how many particles are in each cell.
- We can also sort the cells list according to the number of particles inside the cells (optional).
- We associate with each cell a processor. Then for each cell it is possible to loop on its particles in order to compute their contributions to the current:

$$\int_L S \cdot \psi_L = \sum_{x^k \in L} \omega_k \big(-\dot{x}_1^k(t), -\dot{x}_2^k(t), 0, \chi\big)^{\mathrm{T}} \cdot \psi_L\big(x^k(t)\big)$$

- Thanks to the second optional sorting, neighboring processors treat approximately the same number of particles and in this way they do not wait too much for each other. In some computations, this can increase the efficiency.

4. GPU numerical experiments

GPU programming is complex and time consuming. We thus expect at least high speedups of the implementation. For measuring the efficiency, we have compared in Crestetto and Helluy [8] a sequential and a GPU implementation of the DG

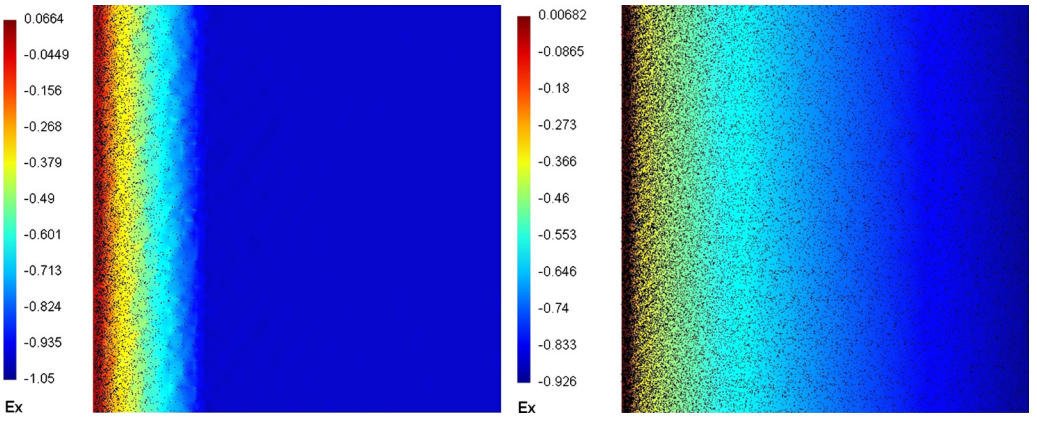

Fig. 1. (Color online.) Planar diode case: E_x at times $t = 1$ with 8918 particles (left), and $t = 5$ with 44133 particles (right), 1024 elements.

Maxwell solver. In this case, without particles, we have observed speedups of the order of 50–100. When we couple the Vlasov and the PIC solvers, we have still good speedups of the order 5–10, but we clearly loose one order of magnitude in the computational time. This can be explained by several reasons:

- particle sorting is time consuming and requires non-coalescing memory accesses;
- the particles are sorted with an indirection array. The current computation thus also requires random memory accesses;
- particles sorting and current evaluation take approximately the same time. While the particle solver is called only every ten iterations of the DG solver, the DG solver represents only 15% of the computation time. Recall that we do not use particles smoothing and that we need high values of χ for performing a good divergence cleaning (typically, we take $\chi \simeq 10$). Finally, this is not so expensive, because the DG solver is much cheaper than the PIC solver.

We now present rapidly several numerical experiments.

4.1. Child–Langmuir current

In our first example, we try to compute numerically a stationary solution that can be expressed analytically. We will see that a high value of χ is necessary. If χ is too small, the scheme does not correct the divergence errors efficiently. Maybe that a smoothing of the particles would permit to diminish χ.

We consider a planar diode $\Omega = [0, L_x] \times [0, L_y]$ with a cathode $C = \{x = 0\} \times [0, L_y]$ and an exit boundary $A = \{x = L_x\} \times [0, L_y]$. We consider a metal boundary condition (19) at the anode $x = 0$ and we apply the exact solution with an inhomogeneous Silver–Müller condition at $x = L_x$. At this point, the "incident" field is defined by

$$(E_1, E_2, H_3, \psi)^{\text{inc}} = (-1, 0, 0, 0)^{\text{T}} \tag{15}$$

We apply periodic conditions at $y = 0$ and $y = L_y$.

We represent E_x on Fig. 1 at times $t = 1$ and $t = 5$. χ is taken equal to 5 and we move the particles every 25 time steps in order to let the divergence correction act. Smaller values of χ give inaccurate results (Crestetto [5]). We see the emission (there is no particle at $t = 0$) and the motion of electrons from the cathode to the anode. We can also remark that on the cathode $E_x = E \cdot n \approx 0$, which is the searched Child–Langmuir condition.

For such a planar diode in Cartesian geometry, the Child–Langmuir current J_{CL} on the anode for a given potential drop V_0 between the cathode and the anode verifies

$$J_{\text{CL}}(L_x) = \frac{4}{9{L_x}^2}\sqrt{\frac{2|q|}{m}}V_0^{\frac{3}{2}} \tag{16}$$

where

$$J_{\text{CL}} = \int_A j \cdot n \tag{17}$$

and

$$V_0 = V(x = L_x, y_0) - V(x = 0, y_0) = \int_0^{L_x} \partial_x V(x, y_0)\mathrm{d}x = -\int_0^{L_x} E_x(x, y_0)\mathrm{d}x \tag{18}$$

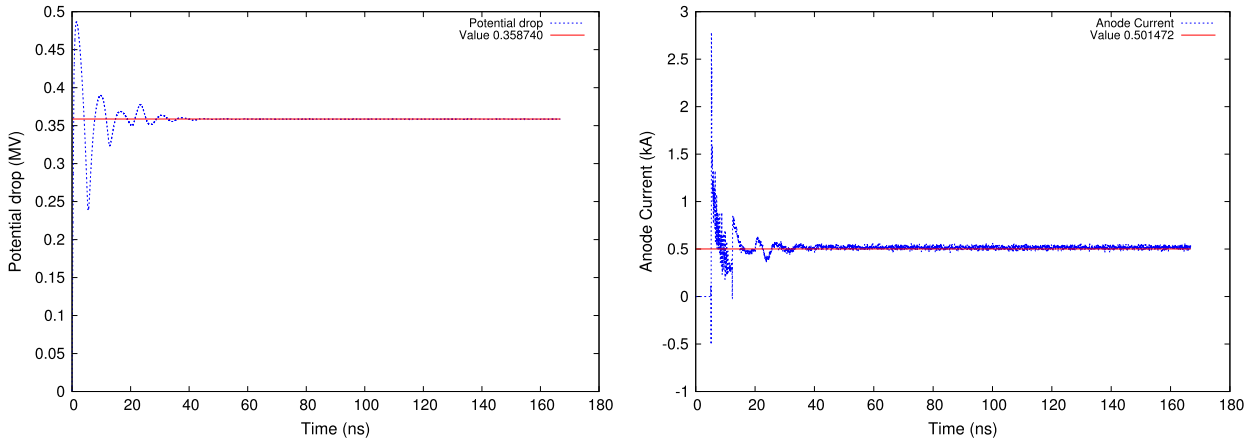

Fig. 2. (Color online.) Child–Langmuir law: potential drop (left) and Child–Langmuir current (right), computed values (blue) compared to theoretical ones (red).

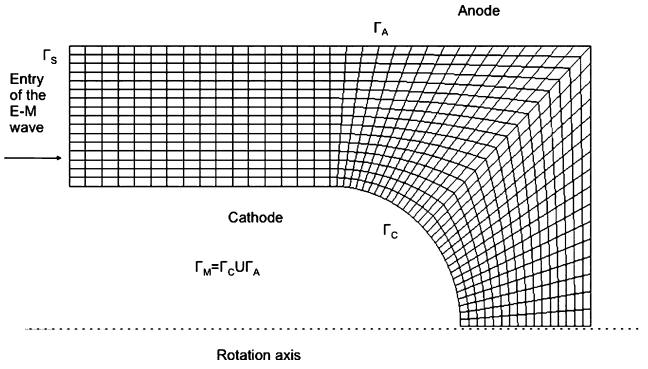

Fig. 3. Diode geometry.

$y_0 \in [0, L_y]$ and V denoting the scalar potential such that $E = -\nabla_x V$. The computed potential drop is given as a function of time on Fig. 2(left). It tends to the value denoted by V_0. The corresponding theoretical current J_{CL} is also given on Fig. 2(right) and compared to the computed one.

4.2. X-ray generator

With our code, we have also been able to simulate an electrons emitting diode. This device is used for producing X-rays, when the electrons hit the anode. The axisymmetric geometry of the diode and the mesh of the computational domain are represented in Fig. 3. At time $t = 0$, an electromagnetic wave is entering at the left of the computational domain. At this boundary Γ_s we apply an inhomogeneous Silver–Müller boundary condition. At the cathode and the anode, we apply metal boundary conditions. The electrons are emitted at the cathode. The rotational symmetry implies additional geometric terms in the Maxwell equations and in the particles weights (see Crestetto and Helluy [8]). In addition there is no boundary condition to apply on the rotation axis, because the numerical flux in the Galerkin Discontinuous method simply cancels (it is multiplied by the distance to the axis).

We represent the radial component of the electric field and the particles at time $t = 0.22$ on Fig. 4.

This numerical simulation has been awarded a prize at the AMD OpenCL competition in 2011. See http://developer.amd.com/events/amd-opencl-coding-competition-2/.

5. Reduced modeling

The conclusion of our Vlasov–Maxwell PIC–DG experience is that it is finally possible to achieve interesting accelerations on a GPU. However, the implementation is complex. While the PIC sequential implementation is straightforward, its parallelization requires particles sorting, random memory accesses, etc. And in the end, most of the GPU time is spent in the PIC algorithm, while the DG solver is very efficient.

We would thus like to present another approach where we use a unified Eulerian DG solver for the Maxwell and the Vlasov equations. When coding our DG solver, we have tried to be as generic as possible. For instance the physical model is

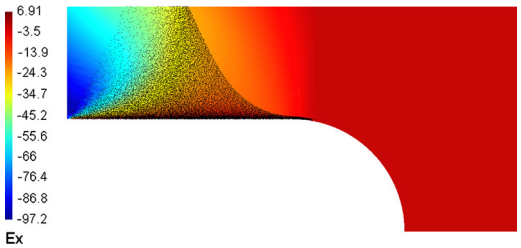

Fig. 4. (Color online.) Electron emission at dimensionless time $t = 0.22$. The length of the computational domain $L = 0.4$ and the dimensionless speed of light $c = 1$.

explicit only in a few parts of the code: in the numerical flux, the source terms and the boundary terms. But generally, the whole DG algorithm is not aware of the underlying physical model. We will show how to rewrite the Vlasov equation in order to obtain an augmented hyperbolic system, written only in (x, t) which can thus be solved by the generic DG solver. The resulting model is called the reduced Vlasov–Maxwell model.

5.1. Velocity expansion

The Vlasov equation is a transport equation written in the (x, v) space. The objective of reduced modeling is to rewrite the Vlasov equation in order to obtain a hyperbolic system of conservation laws, but set only in the x space. In this way it is possible to reuse a generic DG solver. For this purpose, we consider a finite number of continuous basis functions depending on the velocity

$$v \in D \mapsto \varphi_i(v), \quad i = 1 \cdots P$$

We suppose that

$$v \in \partial D \quad \Rightarrow \quad \varphi_i(v) = 0 \tag{19}$$

We expand the distribution function in this basis

$$f(x, v, t) \simeq \sum_{j=1}^{P} f^j(x, t)\varphi_j(v) = f^j \varphi_j$$

We insert this representation into the Vlasov equation (6), multiply by φ_i and integrate on the velocity domain D. We also integrate by parts the acceleration term, and using (19), we obtain (Helluy et al. [17]):

$$\int_v \varphi_i \varphi_j \partial_t f^j + \int_v \varphi_i \varphi_j v^k \partial_k f^j - \int_v \mu a \cdot \nabla_v \varphi_i \varphi_j f^j = 0$$

We can then define the following $P \times P$ matrices

$$M_{i,j} = \int_v \varphi_i \varphi_j, \qquad A^k_{i,j} = \int_v \varphi_i \varphi_j v^k, \qquad B_{i,j} = -\int_v \mu a \cdot \nabla_v \varphi_i \varphi_j \tag{20}$$

and the Vlasov equation can be rewritten in the reduced form

$$M \partial_t w + A^k \partial_k w + B w = 0 \tag{21}$$

or also

$$\partial_t w + M^{-1} A^k \partial_k w + M^{-1} B w = 0 \tag{22}$$

where

$$w = \left(f^1, \cdots, f^P\right)^{\mathrm{T}} \tag{23}$$

The form (22) is called the reduced Vlasov equation. It is a first-order hyperbolic system of conservation laws (Helluy et al. [17]) that can be solved by a standard DG solver. However, for practical reasons it is important to provide an efficient choice of basis functions φ_i. A good choice ensures a small number of basis functions P and that the matrices M, A^k and B are sparse. We detail now such a basis and also an adequate choice of numerical quadrature that will lead to diagonal matrices M and A^k.

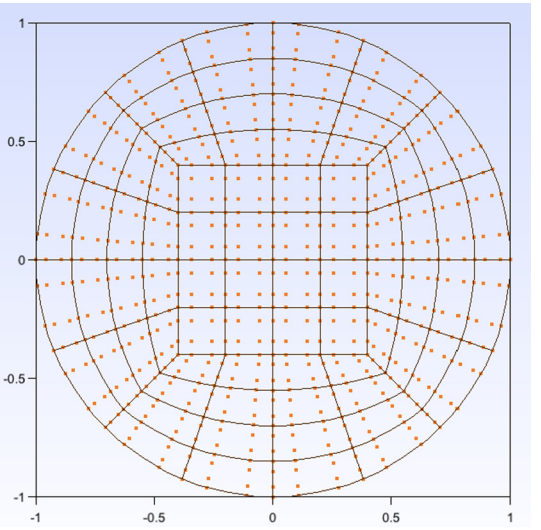

Fig. 5. (Color online.) A mesh of the velocity disk D with degree $d = 3$, $K = 80$ elements (in black), $P_D = 745$ Gauss–Lobatto nodes (red points) and $P = 697$ interior nodes. The distribution function cancels on the boundary and is thus computed only at the interior velocity points. Such a mesh leads to very heavy computations because we have to solve in space and time a hyperbolic system with $P = 697$ (Vlasov) $+ 3$ (Maxwell) $= 700$ components.

5.2. Finite-element basis with nodal integration

We consider a nodal finite element interpolation in the velocity space with curved "quadrilaterals". In addition, the nodal points will coincide with Gauss–Lobatto quadrature points. In this way we obtain several interesting properties of the basis functions. Let us give now more details.

We choose first a degree $d \geq 1$ of polynomial approximation. We can associate with this degree $d + 1$ Gauss–Lobatto points on the interval $[0, 1]$, $\xi_1 = 0 < \xi_2 < \cdots < \xi_{d+1} = 1$. We consider also integration weights $\omega_1 \cdots \omega_d$. The Gauss–Lobatto integration rule

$$\int_0^1 Q(\xi)\mathrm{d}\xi \simeq \sum_{i=1}^{d+1} \omega_i Q(\xi_i)$$

is then exact if Q is a polynomial of degree $\leq 2d - 1$. We also consider the Lagrange polynomials L_i associated with the Gauss–Lobatto subdivision. The polynomial L_i, $i = 1 \cdots d + 1$ is of degree d and satisfies

$$L_i(\xi_j) = \delta_{ij}$$

where δ_{ij} is the Kronecker delta.

We construct now a mesh of the velocity disk D. The mesh is made of nodes V_k, $k = 1 \cdots P_D$, $P_D > P$. Each node V_k for $k = 1 \cdots P$ is associated with a basis function φ_k. We also suppose that the nodes $V_{P+1} \cdots V_{P_D}$ are on the boundary ∂D in such a way that they are not associated with basis functions φ_k. The nodes are associated with elements Λ_k, $k = 1 \cdots K$. Each element is a curved "quadrilateral" and owns $(d + 1)^2$ nodes. An example of such a mesh with degree $d = 3$, $K = 80$ elements, $P_D = 745$ nodes and $P = 697$ interior nodes is given in Fig. 5.

As it is traditional in the finite element method, we consider a local and a global numbering of the nodes of element Λ_k. The local node l, $l = 1 \cdots (d + 1)^2$ of element Λ_k is also noted

$$V_{k,l} = V_{\kappa(k,l)}$$

where $\kappa(k, l)$ is the $K \times (d+1)^2$ connectivity array of the finite-element mesh. Each element Λ_k is obtained from a geometrical transformation τ_k that maps the square $\hat{\Lambda} =]0, 1[\times]0, 1[$ on element Λ_k. The geometrical transformation is defined as follows. First, we consider $(d + 1)^2$ reference nodes

$$\hat{V}_l = (\xi_i, \xi_j), \quad \text{with } l = (i - 1)(d + 1) + j,\ 1 \leq i,\ j \leq d + 1$$

The reference nodes are the two-dimensional Gauss–Lobatto points of the reference element. Reference node $\hat{V}_l$ is also associated with an integration weight:

$$\hat{\omega}_l = \omega_i \omega_j \quad \text{with } l = (i - 1)(d + 1) + j,\ 1 \leq i,\ j \leq d + 1 \tag{24}$$

With each reference node, we associate a reference basis function, which is a tensor product of Lagrange polynomials

$$\hat{\varphi}_l(\xi,\eta) = L_i(\xi)L_j(\eta), \quad \text{with } l=(i-1)(d+1)+j,\ 1\le i,\ j\le d+1$$

We can check that

$$\hat{\varphi}_l(\hat{V}_m) = \delta_{lm}$$

Then, the geometric transformation is defined by

$$\tau_k(\xi,\eta) = \sum_{l=1}^{(d+1)^2} \hat{\varphi}_l(\xi,\eta) V_{k,l}$$

in such a way that

$$\tau_k(\hat{V}_l) = V_{k,l}$$

We also suppose that the node V_i are defined in such a way that τ_k is invertible and also a direct transformation:

$$\det \tau_k' > 0$$

We have now all the pieces to construct the basis functions. Let $i = 1\cdots P$ and $v \in D$, then necessarily, $v \in \Lambda_k$ for some $k = 1\cdots K$.[1] We then have two possibilities.

1. Node V_i is not a node of element Λ_k then

$$\varphi_i(v) = 0 \tag{25}$$

2. Node V_i is a node of element Λ_k. It means that $i = \kappa(k,l)$ for some $l = 1\cdots(d+1)^2$. Then

$$\varphi_i(v) = \hat{\varphi}_l(\hat{v}), \quad \text{with } v = \tau_k(\hat{v}) \tag{26}$$

In this case, we can also compute the gradient of the basis function

$$\nabla_v \varphi_i(v) = \left(\tau_k'(\hat{v})^{\mathrm{T}}\right)^{-1} \nabla_{\hat{v}} \hat{\varphi}_l(\hat{v}), \quad v = \tau_k(\hat{v}) \tag{27}$$

From the previous definitions, we obtain basis functions that are continuous on D. In addition, they satisfy the interpolation property:

$$\varphi_i(V_j) = \delta_{ij}, \quad 1\le i,j\le P$$

This interpolation property ensures that the components of w in (23) are simply approximations of the distribution function at the Gauss–Lobatto points V_i:

$$f^i(x,t) \simeq f(x,V_i,t)$$

We can thus also use the convention that on the boundary nodes

$$f^i(x,t) = 0 \quad \text{if } P+1 \le i \le P_D$$

For computing the matrices in (20), we use the Gauss–Lobatto integration rule. For a given function h defined on D the integral is first split into elementary integrals

$$\int_D h(v)\mathrm{d}v = \sum_{k=1}^{K} \int_{\Lambda_k} h(v)\mathrm{d}v$$

and then (using definition (24))

[1] We assume that the elements Λ_k are disjoint open sets and that $\overline{\bigcup \Lambda_k} = \overline{D}$. Of course, this cannot be exactly true because τ_k is a polynomial transformation. We neglect in our presentation the error made in the approximation of D.

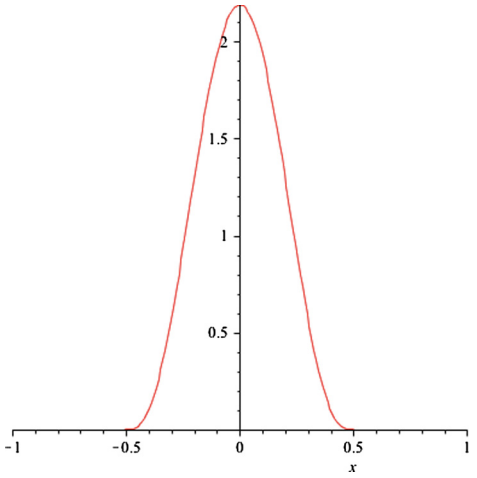

Fig. 6. The function q.

$$\begin{aligned}\int_{\Lambda_k} h(v)\mathrm{d}v &= \int_{\hat{\Lambda}} h\big(\tau_k(\xi,\eta)\big)\det \tau_k'(\xi,\eta)\\ &\simeq \sum_{l=1}^{(d+1)^2} \hat{\omega}_l \det \tau_k'(\hat{V}_l) h\big(\tau_k(\hat{V}_l)\big)\\ &= \sum_{l=1}^{(d+1)^2} \omega_{k,l} h(V_{k,l}) \end{aligned} \tag{28}$$

Using the quadrature rule (28) and formula (27) for computing the gradient of the basis function we can practically compute the matrices in (20). Our choice of integration points does not ensure exact integration for M, A^k, and B. However, in the sequel, we use the same notation for the exact and approximate matrices. With our choice of quadrature points, we obtain that M and A^k are diagonal matrices. More precisely, we have:

$$M_{ii} = \sum_{i=\kappa(k,l)} \omega_{k,l} \tag{29}$$

and

$$A^k_{ii} = M_{ii} V^k_i, \quad V_i = \big(V^1_i, V^2_i\big)$$

These computations show that the components of the vector w satisfy a set of coupled transport equations:

$$\partial_t f^i + V_i \cdot \nabla_x f^i + \Sigma^i(w) = 0, \quad i = 1 \cdots P \tag{30}$$

In the vector form, the coupling source term is given by:

$$\Sigma(w) = M^{-1} B w$$

More precisely, after expanding the computations, the coupling source term becomes

$$\Sigma^i(w) = \frac{-1}{M_{ii}} \sum_{\overline{\Lambda_k} \ni V_i} \sum_{l=1}^{(d+1)^2} \omega_{k,l} f^{\kappa(k,l)} \mu(V_{k,l}) a(\cdot, V_{k,l}, \cdot) \cdot \nabla_v \varphi_i(V_{k,l}) \tag{31}$$

where we recall that the gradient of the basis function φ_i is given by (27).

Remark. In practice, we can compute M_{ii} and Σ^i efficiently by a classic finite-element assembly procedure: we loop on the elements, compute the elementary contributions and distribute them into the global (M_{ii}) or (Σ^i) vectors.

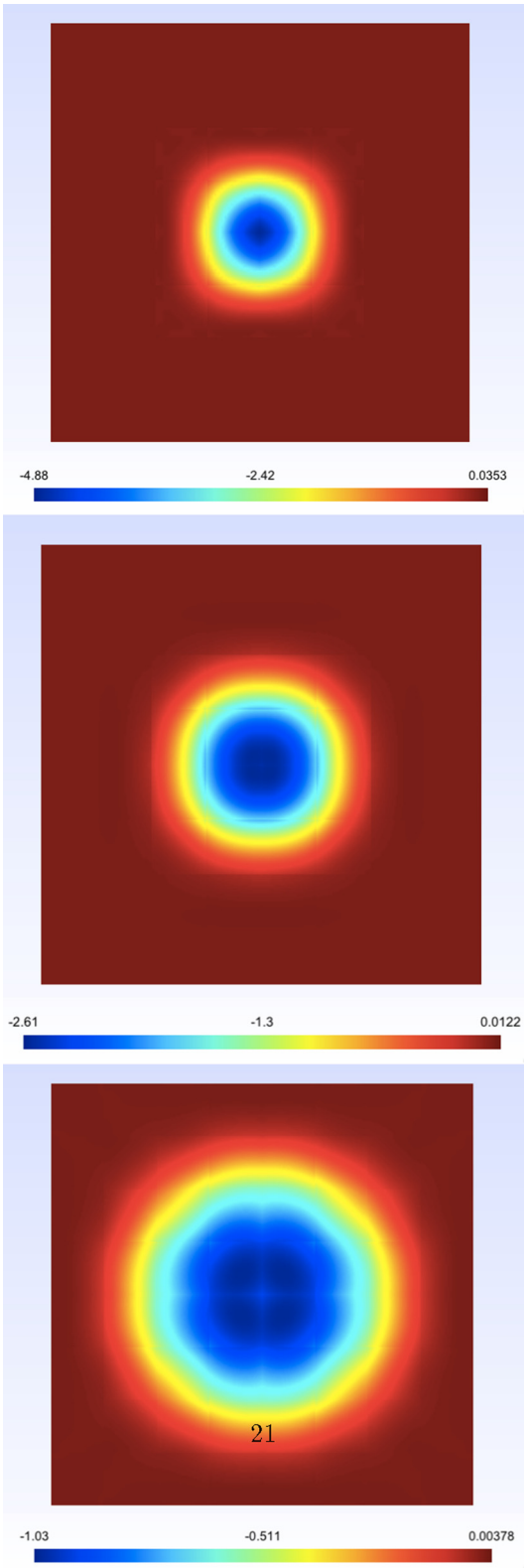

Fig. 7. (Color online.) Evolution of the charge in the computational domain, $t = 0$ (top), $t = 0.5$ (middle), $t = 1.0$ (bottom).

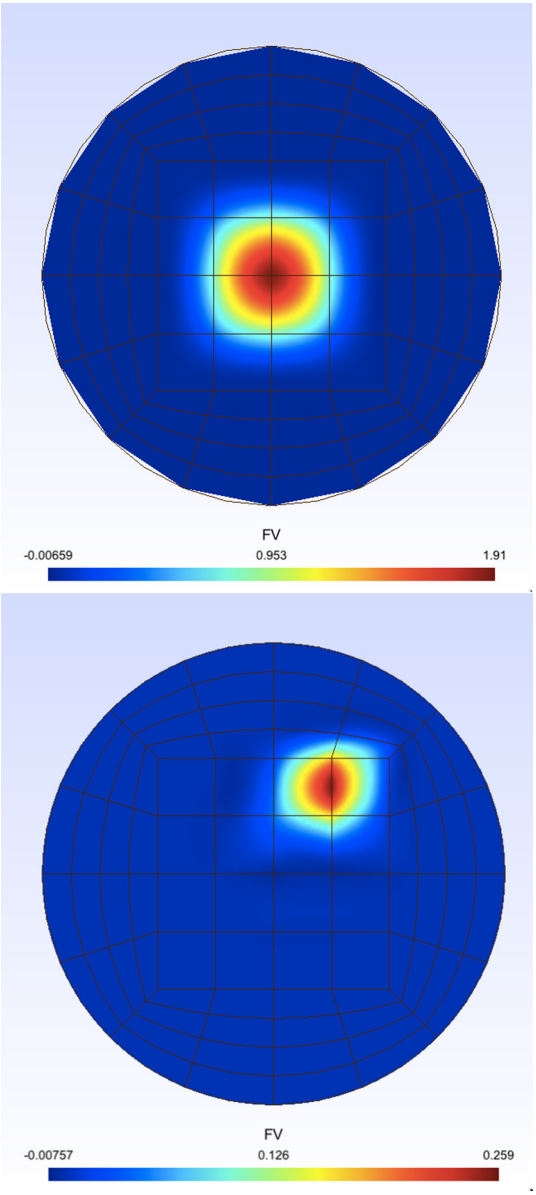

Fig. 8. (Color online.) Evolution of the distribution function $f(0.37, 0.5, v_1, v_2, t)$ in the computational domain, $t = 0$ (top), $t = 0.5$ (bottom). The apparent polygonal shape of the mesh boundary is due to a post-processor bug in the first picture.

5.3. Unified expression of the reduced Vlasov–Maxwell model

We are now in a position to write in a unified way the Maxwell and reduced Vlasov system. We extend the original vector W of (1) in the following way:

$$W = \left(E_1, E_2, H_3, \phi, f^1 \cdots f^P\right)^{\mathrm{T}}$$

We then obtain a hyperbolic system in the form (1) where the new matrices A^k are block diagonal. The blocks are constructed with the matrices A^k of Section 1.3 and Section 5.2. The source term of the new system is also assembled from the source terms (10) and (5) of Section 1.3 and Section 5.2:

$$S(W) = \left(j_1, j_2, 0, \chi\rho, \Sigma^{\mathrm{T}}\right)^{\mathrm{T}} \tag{32}$$

For computing the current and the charge here, we again use the Gauss–Lobatto quadrature (28) in the velocity space

$$\rho = \sum_{k,l} \omega_{k,l} f^{\kappa(k,l)}, \qquad j = \sum_{k,l} \omega_{k,l} f^{\kappa(k,l)} V_{k,l} \tag{33}$$

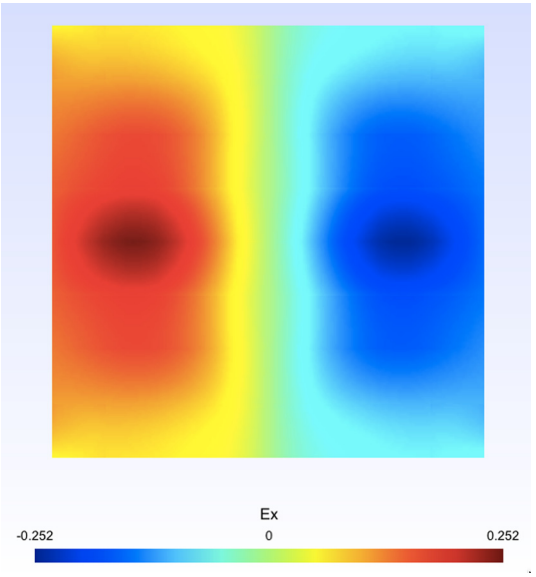

Fig. 9. (Color online.) x_1 component of the electric field at time $t = 1.0$.

In this formalism, it is very easy to adapt a generic Discontinuous Galerkin solver for handling the reduced Vlasov–Maxwell model. We just have to modify the numerical flux and source functions. We have seen in (30) that the reduced Vlasov equation is a set of coupled transport equations with velocities V_i. We use a standard upwind numerical flux for the transport equations. The source term (31) is computed with the assembly procedure and superimposed with the Maxwell source term from (33) and (10).

5.4. Preliminary numerical results

In this section, we present preliminary Vlasov–Maxwell numerical results obtained with the reduced approach. We have not yet implemented the Child–Langmuir boundary condition. Therefore we only present a very simple and academic test case. We consider a cloud of charged particles in the center of a square domain Ω. Because the particles repel each other, the cloud will expand with time. We plot the evolution of the total charge in the domain at different times. We also represent the distribution function in the velocity space at a given point $(x_1, x_2) = (0.37, 0.5)$. The initial distribution function is

$$f(x_1, x_2, v_1, v_2, 0) = -q(x_1)q(x_2)q(v_1)q(v_2),$$

where the function q, represented in Fig. 6, has its support in $[-1/2, 1/2]$ and satisfies $\int_{r=-\infty}^{\infty} q(r)\mathrm{d}r = 1$.

We suppose that the initial electromagnetic field vanishes. This initial condition is not physical, because the Gauss law is not satisfied: $\nabla \cdot E \neq \rho$. Therefore, we take a divergence correction parameter $\chi = 4$. On the boundary of Ω, we apply homogeneous Silver–Müller conditions. Finally, we take $\mu = 1$: our computation is non-relativistic. The simulation time is short enough so that the exact distribution function vanishes on the boundary of the velocity disk, even at the final time $T = 1$.

We use a mesh of Ω with $8 \times 8 = 64$ cells. The velocity mesh is of order $d = 2$. It contains 305 Gauss–Lobatto nodes.

We plot the charge evolution in Fig. 7.

We also plot the distribution function at point $(x_1, x_2) = (0.37, 0.5)$ at different times on Fig. 8.

Finally, we plot the x_1-component of the electric field at time $t = 1$ in Fig. 9.

Conclusion

In this work, we have presented two numerical schemes for approximating the Vlasov–Maxwell system. The first scheme is a coupling between an upwind DG solver for the Maxwell equations with a PIC solver for the Vlasov equation. We have reviewed some practical aspects of a robust and precise implementation of the whole procedure: high-order polynomials, upwind flux, stable boundary conditions, divergence cleaning. We have also implemented the algorithm on GPU, which requires a sorting of the particles list at each time step. We obtained interesting speedups, but we also observe that the PIC method is the most expensive part of the computation. Therefore we propose another fully Eulerian approach. Thanks to a decomposition of the distribution function on velocity basis functions, we obtain a reduced Vlasov model, which appears to be a hyperbolic system of conservation laws written only in the (x, t) space. We can thus adapt very easily our DG solver to the reduced model. We presented preliminary numerical results. Our next step will be to implement more physical boundary conditions and test the reduced approach on emitting diode test cases.

References

[1] P. Helluy, Résolution numérique des équations de Maxwell harmoniques par une méthode d'éléments finis discontinus, PhD thesis, Sup'aéro, 1994, http://tel.archives-ouvertes.fr/tel-00657828.

[2] F. Bourdel, P.A. Mazet, P. Helluy, Resolution of the non-stationary or harmonic Maxwell equations by a discontinuous finite element method. Application to an E.M.I. (electromagnetic impulse) case, in: Proceedings of the 10th International Conference on Computing Methods in Applied Sciences and Engineering, 1992, pp. 405–422.

[3] G. Cohen, X. Ferrieres, S. Pernet, A spatial high-order hexahedral discontinuous Galerkin method to solve Maxwell's equations in time domain, J. Comput. Phys. 217 (2) (2006) 340–363.

[4] A. Klöckner, T. Warburton, J. Bridge, J.S. Hesthaven, Nodal discontinuous Galerkin methods on graphics processors, J. Comput. Phys. (ISSN 0021-9991) 228 (21) (2009) 7863–7882, http://dx.doi.org/10.1016/j.jcp.2009.06.041.

[5] A. Crestetto, Optimisation de méthodes numériques pour la physique des plasmas. Application aux faisceaux de particules chargées, PhD thesis, Université de Strasbourg, 2012.

[6] C.K. Birdsall, A.B. Langdon, Plasma Physics via Computer Simulation, Series in Plasma Physics, Institute of Physics (IOP), 1991.

[7] C.-D. Munz, P. Omnes, R. Schneider, E. Sonnendrücker, U. Voß, Divergence correction techniques for Maxwell solvers based on a hyperbolic model, J. Comput. Phys. 161 (2) (2000) 484–511.

[8] A. Crestetto, P. Helluy, Resolution of the Vlasov–Maxwell system by PI discontinuous Galerkin method on GPU with OpenCL, in: CEMRACS'11: Multiscale Coupling of Complex Models in Scientific Computing, in: ESAIM Proc., vol. 38, EDP Sci., Les Ulis, 2012, pp. 257–274.

[9] B. Fornet, V. Mouysset, Á. Rodríguez-Arós, Mathematical study of a hyperbolic regularization to ensure Gauss's law conservation in Maxwell–Vlasov applications, Math. Models Methods Appl. Sci. (ISSN 0218-2025) 22 (4) (2012) 1150020, 28 pp., http://dx.doi.org/10.1142/S0218202511500205.

[10] P.D. Lax, R.S. Phillips, Local boundary conditions for dissipative symmetric linear differential operators, Commun. Pure Appl. Math. 13 (3) (1960) 427–455.

[11] V.M. Petkov, L.N. Stoyanov, Geometry of Reflecting Rays and Inverse Spectral Problems, Pure and Applied Mathematics (New York), John Wiley & Sons Ltd., Chichester, ISBN 0-471-93174-8, 1992.

[12] C. Johnson, J. Pitkäranta, An analysis of the discontinuous Galerkin method for a scalar hyperbolic equation, Math. Comput. (ISSN 0025-5718) 46 (173) (1986) 1–26, http://dx.doi.org/10.2307/2008211.

[13] P. Lesaint, P.-A. Raviart, On a finite element method for solving the neutron transport equation, in: Mathematical Aspects of Finite Elements in Partial Differential Equations (Proc. Sympos., Math. Res. Center, Univ. Wisconsin, Madison, Wis., 1974), vol. 33, Math. Res. Center, Univ. of Wisconsin–Madison, Academic Press, New York, 1974, pp. 89–123.

[14] The Khronos Group Inc., OpenCL documentation, 2013, http://www.khronos.org/.

[15] D. Aubert, M. Amini, R. David, A Particle-Mesh Integrator for Galactic Dynamics Powered by GPGPUs, Lecture Notes in Computer Science, vol. 5544, 2009, pp. 874–883.

[16] P. Helluy, A portable implementation of the radix sort algorithm in OpenCL, 2011, http://hal.archives-ouvertes.fr/hal-00596730/PDF/ocl-radix-sort.pdf, http://hal.archives-ouvertes.fr/hal-00596730.

[17] P. Helluy, N. Pham, A. Crestetto, Space-only hyperbolic approximation of the Vlasov equation, in: ESAIM: Proceedings, vol. 43, 2013, pp. 17–36, http://dx.doi.org/10.1051/proc/201343002, http://hal.archives-ouvertes.fr/hal-00797974.