Journal of Computational Physics 228 (2009) 1429–1446

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

A parallel Vlasov solver based on local cubic spline interpolation on patches

Nicolas Crouseilles [a,c,*], Guillaume Latu [b], Eric Sonnendrücker [c]

[a] INRIA-Nancy-Grand Est, CALVI Project, France
[b] LSIIT-Strasbourg et INRIA-Nancy-Grand Est, CALVI Project, France
[c] IRMA-Strasbourg et INRIA-Nancy-Grand Est, CALVI Project, France

ARTICLE INFO

Article history:
Received 21 November 2006
Received in revised form 30 September 2008
Accepted 29 October 2008
Available online 17 November 2008

Keywords:
Vlasov equation
Semi-Lagrangian method
Numerical methods
Parallelism

ABSTRACT

A method for computing the numerical solution of Vlasov type equations on massively parallel computers is presented. In contrast with Particle In Cell methods which are known to be noisy, the method is based on a semi-Lagrangian algorithm that approaches the Vlasov equation on a grid of phase space. As this kind of method requires a huge computational effort, the simulations are carried out on parallel machines. To that purpose, we present a local cubic splines interpolation method based on a domain decomposition, e.g. devoted to a processor. Hermite boundary conditions between the domains, using ad hoc reconstruction of the derivatives, provide a good approximation of the global solution. The method is applied on various physical configurations which show the ability of the numerical scheme.

1. Introduction

To model the evolution of charged particles at the kinetic level, the Vlasov equation is often studied, in particular in the context of collisionless plasmas or space charge dominated beams. The coupling with Maxwell or Poisson equations has to be taken into account to compute self-consistent forces. The unknown f is a distribution function of particles in phase space which depends on the time $t \geqslant 0$, the physical space $x \in \mathbb{R}^d$ and the velocity $v \in \mathbb{R}^d$, where d is the dimension $d = 1, 2, 3$. The Vlasov equation is a nonlinear partial differential equation, whose analytical solution is available in a few simplified linear cases, but the nonlinear regime, including the most interesting physical phenomena, must be investigated numerically.

In this context, Particle In Cell (PIC) codes have been considered up to now as the most effective approach to simulate plasmas in the framework of the kinetic theory. Indeed, high dimensional simulations can be performed using this approach with a relatively small computational cost (see [3,6,20]). However, it is known that this method suffers from some serious drawbacks stemming from statistical numerical noise particularly when a detailed structure of the distribution is needed (when particles in the tail of the distribution function play an important role in the investigated physics, or when the influence of density fluctuations which are at the origin of instabilities are studied).

On the other hand, another approach that numerically solves the Vlasov equation is the Eulerian method; this kind of method uses a computational grid of the whole phase space and the time integration of the distribution function is carried out at each computational grid point. Various techniques have been investigated and we refer the reader to [10–12,15,16,23,26,27,30,32,33] for plasma physics applications, and to [1,31] for other applications. Among these Eulerian methods, the semi-Lagrangian method consists in computing directly the distribution function on a Cartesian grid of the phase space, by integrating the characteristic curves backward at each time step and interpolating the value at the feet of

* Corresponding author. Address: 7, rue René Descartes, 67084, Strasbourg cedex.
E-mail address: crouseil@math.u-strasbg.fr (N. Crouseilles).

0021-9991/$ - see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2008.10.041

the characteristics by some interpolation techniques (Lagrange, Hermite or cubic splines for example). We refer the reader to [30] for more details on the semi-Lagrangian method and to [12] for a comparison of Eulerian solvers dedicated to the Vlasov equation.

Eulerian methods have proven their efficiency on uniform meshes in two-dimensional phase space. However when higher dimensional problems need to be solved, as these methods cover the whole phase space with grid points, the memory storage and computation time rapidly increase since a minimum of points are required in each direction to represent the physics correctly. To overcome this problem, some efficient parallelized versions of the code have been implemented to simulate high dimensional problems. For example, in [7], a time splitting procedure enables the authors to parallelize the algorithm, but a global transpose between each split step induces an excessive communication cost when a large number of processors was used (see [7,14]). Apart from this transpose, that can be overlapped with computations in favorable cases, there is no communication between processors. However, when heterogeneous grids and several hundreds or more processors are targeted (see [17,21]), a global transpose involves a huge amount of data being transfered and this can become very inefficient because of intense network traffic. For these reasons, we develop in this paper a local spline interpolation technique to solve Vlasov type equations which enables the *efficient* use of a large number of processors, and consequently to decrease the runtimes.

The numerical method is based on the semi-Lagrangian approach by using the cubic spline interpolation operator. In order to validate the method, we have designed the parallel software LOSS (LOcal Splines Simulator). Even if cubic spline interpolation seems to be a good compromise between accuracy (small diffusivity) and simplicity, it does not provide the locality of the reconstruction since all the values of the distribution function are used for the reconstruction in each cell. To overcome this problem of global dependency, we decompose the phase space domain into patches, each patch being devoted to one processor. One patch computes its own local cubic spline coefficients by solving reduced linear systems; Hermite boundary conditions are imposed at the boundary of the patches to reconstruct a smooth global numerical solution and the needed derivative are computed in such a way that they are close to those given by the global spline interpolant. Hence this new interpolation method benefits from the advantages of the cubic splines approach yielding almost the same solution with the benefit of more efficient parallelization.

In fact, our strategy consists in getting a parallel version of the code, the results of which are as close as possible to the results of the sequential version. Even if the methodology remains slightly different from the sequential case (essentially due to local solutions of the cubic spline coefficients *versus* global solution), our main efforts consist in recovering in the best possible way the global solution. Thanks to an adapted treatment of the Hermite boundary conditions, the obtained numerical results are then in a good agreement with those obtained with the sequential version of the code. But the numerical method introduces data-processing problems since some communications between processors have to be managed in a suitable way; indeed, as particles can leave the subdomain, their information must be forwarded to the appropriate processor that controls the subdomain in which the particles now reside. Such interprocessor communications would involve a relatively large amount of data exchange, but a condition on the time step allows us to control the shifts so that the communications are only done between adjacent processors. Hence, this communication scheme enables us to obtain competitive results from a scalability point of view. Let us mention that even if a uniform grid is used here, the methodology could be extended to sets of lines which are not equally spaced (adaptive meshes for example).

Hence, this methodology enables to perform high dimensional Vlasov simulations on massively parallel computers, without degradation of the numerical results compared to the standard solvers. More precisely, to simulate realistic physics problems, since a large number of grid points is required, Eulerian methods become very costly. Using this approach, the data structure is efficiently decomposed into a large number of processors so that the memory requirements and computational cost can be handled.

The paper is organized as follows: first, we recall the main steps of the semi-Lagrangian method applied to the Vlasov equation. Next, we propose the Hermite spline interpolation on patches before illustrating the efficiency of the method by presenting several numerical and performance results.

2. The Vlasov equation and the semi-Lagrangian method

The evolution of the particle distribution function $f(t,x,v)$ in phase space, where $(x,v) \in \mathbb{R}^d \times \mathbb{R}^d$ with $d = 1,2,3$ is the phase space position and t the time, is given by the following Vlasov equation which is written in dimensionless units

$$\frac{\partial f}{\partial t} + v \cdot \nabla_x f + F(t,x,v) \cdot \nabla_v f = 0, \tag{2.1}$$

where the force field $F(t,x,v)$ can be coupled to the distribution function f through the Maxwell or Poisson equations. This nonlinear system features conservation properties that should be maintained as accurately as possible by the numerical scheme: the mass, momentum and total energy, together with the L^p norms ($1 \leqslant p \leqslant +\infty$) are preserved in time.

On the other hand, we can define the characteristic curves of the Vlasov Eq. (2.1) as the solutions of the following first order differential system

$$\begin{cases} \frac{dX}{dt}(t;s,x,v) = V(t;s,x,v), \\ \frac{dV}{dt}(t;s,x,v) = F(t,X(t;s,x,v), \quad V(t;s,x,v)), \end{cases} \tag{2.2}$$

with the initial conditions

$$X(s;s,x,v) = x, \quad V(s;s,x,v) = v.$$

We denote by $(X(t;s,x,v), V(t;s,x,v))$ the position in phase space at the time t, of a particle which was in (x,v) at time s. The functions $t \to (X(t;s,x,v), V(t;s,x,v))$ are the characteristic curves solution of (2.2). Then, the solution of the Vlasov Eq. (2.1) is given by

$$f(t,x,v) = f(s, X(s;t,x,v), \quad V(s;t,x,v)) \tag{2.3}$$

$$= f_0(X(0;t,x,v), V(0;t,x,v)), \quad (x,v) \in \mathbb{R}^d \times \mathbb{R}^d, \quad t \geqslant 0, \tag{2.4}$$

where f_0 is a given initial condition of the Vlasov equation. This equality means that the distribution function f is constant along the characteristic curves which is the basis of the semi-Lagrangian method. Replacing s by t^n and t by t^{n+1} in (2.3), and denoting $X^n = X(t^n, t^{n+1}, x, v)$ and $V^n = V(t^n, t^{n+1}, x, v)$ we have

$$f(t^{n+1}, x, v) = f(t^n, X^n, V^n).$$

For each point of the phase space grid (x,v), the distribution function is updated thanks to the two following steps

1. Find the starting point of the characteristic curves ending at (x,v), i.e. $X^n = X(t^n, t^{n+1}, x, v)$ and $V^n = V(t^n, t^{n+1}, x, v)$ by solving (2.2).
2. Compute $f(t^n, X^n, V^n)$ by interpolation, f being known only at mesh points at time t^n.

As discussed in [30], step 1 must be performed carefully by introducing a time discretization of (2.2). But when a splitting procedure can be adopted, the resolution of step 1 becomes straightforward since the advection field may not depend on the variable to be advected.

3. The local spline interpolation

In this section, we present our interpolation technique based on a cubic spline method (see [4,19,30]). Even if the cubic spline approach is quite standard for solving Vlasov equations, it remains a global method since it requires the values of the distribution function on the whole domain or on big two-dimensional domains for two-dimensional advections, which is an inconvenient from a parallelization point of view. Our approach avoids this globality. Indeed, we decompose the phase space into several patches, each patch being devoted to one processor. The strategy is based on adapted boundary conditions which allow a $\mathcal{C}^1$ reconstructed solution on the global phase space domain even on the patches boundaries.

We first present the interpolation on one patch in an unidimensional context before focusing on the two-dimensional case.

3.1. The local spline interpolation in one dimension

Let us consider a function f which is defined on a global domain $[x_{\min}, x_{\max}] \subset \mathbb{R}$. This domain is decomposed into several subdomains called generically $[x_0, x_N]$; each subdomain will be devoted to a processor. In the following, we will use the notation $x_i = x_0 + ih$, where h is the mesh size: $h = (x_N - x_0)/(N+1)$.

Let us now restrict the study of $f: x \mapsto f(x)$ on an interval $[x_0, x_N]$, $N \in \mathbb{N}$, where x_0 and x_N are to be chosen, according to the domain decomposition. The projection s of f onto the cubic spline basis reads

$$f(x) \simeq s(x) = \sum_{v=-1}^{N+1} \eta_v B_v(x),$$

where B_v is the cubic B-spline

$$B_v(x) = \frac{1}{6h^3} \begin{cases} (x - x_{v-2})^3 & \text{if } x_{v-2} \leqslant x \leqslant x_{v-1}, \\ h^3 + 3h^2(x - x_{v-1}) + 3h(x - x_{v-1})^2 & \\ -3(x - x_{v-1})^3 & \text{if } x_{v-1} \leqslant x \leqslant x_v, \\ h^3 + 3h^2(x_{v+1} - x) + 3h(x_{v+1} - x)^2 & \\ -3(x_{v+1} - x)^3 & \text{if } x_v \leqslant x \leqslant x_{v+1}, \\ (x_{v+2} - x)^3 & \text{if } x_{v+1} \leqslant x \leqslant x_{v+2}, \\ 0 & \text{otherwise.} \end{cases} \tag{3.1}$$

Let us remark that f and s coincide when f is a polynomial of degree less than 3 for example. The interpolating spline s is uniquely determined by $(N+1)$ interpolating conditions

$$f(x_i) = s(x_i) \quad \forall i = 0, \ldots, N, \tag{3.2}$$

and the Hermite boundary conditions at both ends of the interval in order to obtain a $\mathcal{C}^1$ global approximation

$$f'(x_0) = s'(x_0), \quad f'(x_N) = s'(x_N). \tag{3.3}$$

The only cubic B-spline not vanishing at point x_i are $B_{i\pm1}(x_i) = 1/6$ and $B_i(x_i) = 2/3$. Hence (3.2) yields

$$f(x_i) = \frac{1}{6}\eta_{i-1} + \frac{2}{3}\eta_i + \frac{1}{6}\eta_{i+1}, \quad i = 0, \dots, N. \tag{3.4}$$

On the other hand, we have $B'_{i\pm1}(x_i) = \pm 1/(2h)$, and $B'(x_i) = 0$. Thus the Hermite boundary conditions (3.3) become

$$f'(x_0) = s'(x_0) = -1/(2h)\eta_{-1} + 1/(2h)\eta_1, \tag{3.5}$$

and

$$f'(x_N) = s'(x_N) = -1/(2h)\eta_{N-1} + 1/(2h)\eta_{N+1}.$$

Finally, $\eta = (\eta_{-1}, \dots \eta_{N+1})^T$ is the solution of the $(N+3) \times (N+3)$ system $A\eta = F$, where F is the following vector and

$$F = [f'(x_0), f(x_0), \dots, f(x_N), f'(x_N)]^T. \tag{3.6}$$

and A denotes the following matrix

$$A = \frac{1}{6}\begin{pmatrix} -3/h & 0 & 3/h & 0 & \cdots & 0 \\ 1 & 4 & 1 & 0 & & \vdots \\ 0 & 1 & 4 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & 0 & 1 & 4 & 1 \\ 0 & 0 & 0 & -3/h & 0 & 3/h \end{pmatrix}. \tag{3.7}$$

Let us precise that $f\prime$ is generally not known, and from a numerical point of view, the derivative of f has to be approximated adequately. We will focus on this in the sequel of the paper.

Solution of the linear system $A\eta = F$

The matrix A of the linear system has a special structure. Its LU decomposition is of the following form

$$L = \begin{pmatrix} 1 & 0 & 0 & \cdots & \cdots & 0 \\ -h/3 & 1 & 0 & \ddots & & \vdots \\ 0 & l_1 & 1 & \ddots & & \vdots \\ 0 & 0 & \ddots & \ddots & 0 & \vdots \\ \vdots & \ddots & \ddots & l_N & 1 & 0 \\ 0 & \cdots & 0 & -(3l_N)/h & (3l_{N+1})/h & 1 \end{pmatrix},$$

and

$$U = \frac{1}{6}\begin{pmatrix} -3/h & 0 & 3/h & 0 & \cdots & 0 \\ 0 & d_1 & 2 & 0 & & \vdots \\ 0 & 0 & d_2 & 1 & \ddots & \vdots \\ 0 & 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & d_{N+1} & 1 \\ 0 & \cdots & 0 & 0 & 0 & (3d_{N+2})/h \end{pmatrix},$$

where l_i and d_i can be computed from the following relations

$$d_1 = 4, \quad l_1 = 1/4, \quad d_2 = 4 - 2l_1 = 7/2,$$

from $i = 2, N$

$$l_i = 1/d_i,$$
$$d_{i+1} = 4 - l_i,$$

and

$$l_{N+1} = \frac{1}{d_N d_{N+1}},$$
$$d_{N+2} = 1 - l_{N+1}.$$

The LU decomposition of A can then be computed once for all. At each time step, a spline interpolant needs to be computed solving $LU\eta = F$ in two steps: the solution of $L\varphi = F$, and then, the solution of $L\eta = \varphi$.

3.2. The local spline interpolation in two dimensions

In a two-dimensional space, f is projected on a cubic spline basis $\forall (x,y) \in [x_0, x_{N_x}] \times [y_0, y_{N_y}]$ as follows:

$$f(x,y) \simeq s(x,y) = \sum_{v=-1}^{N_x+1} \sum_{\beta=-1}^{N_y+1} \eta_{v,\beta} B_v(x) B_\beta(y). \tag{3.8}$$

The same notations as in the previous section are used and we have to compute the coefficients $\eta_{v,\beta}$. To that purpose, we first solve the $N_y + 1$ following systems

$$s(x, y_j) = \sum_{v=-1}^{N_x+1} \gamma_v(y_j) B_v(x) \quad \forall j = 0, \ldots, N_y, \tag{3.9}$$

where

$$\gamma_v(y_j) = [\gamma_{-1}(y_j), \gamma_0(y_j), \ldots, \gamma_v(y_j), \ldots, \gamma_{N_x}(y_j), \gamma_{N_x+1}(y_j)]^T.$$

Each of the $N_y + 1$ systems (3.9) satisfies the $N_x + 1$ interpolation conditions (at fixed j)

$$f(x_i, y_j) = s(x_i, y_j), \quad i = 0, \ldots, N_x,$$

and the Hermite boundary conditions in the x-direction

$$\frac{\partial f}{\partial x}(x_0, y_j) = \frac{\partial s}{\partial x}(x_0, y_j), \frac{\partial f}{\partial x}(x_{N_x}, y_j) = \frac{\partial s}{\partial x}(x_{N_x}, y_j).$$

We have denoted by

$$\gamma_v(y_j) = \sum_{\beta=-1}^{N_y+1} \eta_{v,\beta} B_\beta(y_j). \tag{3.10}$$

We have been brought to solve $N_y + 1$ linear systems $A\gamma_v(y_j) = F(y_j)$, one for each value of j, involving the $(N_x + 3) \times (N_x + 3)$ matrix (3.7) and a $(N_x + 3)$ vector similar to (3.6) evaluated in y_j. Following the same procedure used previously (*via* the LU decomposition), we then obtain the $(N_x + 3)$ vector of unknown $\gamma_v(y_j)$, for $j = 0, \ldots, N_y$

$$\gamma_v(y_j) = [\gamma_{-1}(y_j), \gamma_0(y_j), \ldots, \gamma_v(y_j), \ldots, \gamma_{N_x}(y_j), \gamma_{N_x+1}(y_j)]^T.$$

The second step consists in the solution for each $v = -1, .., N_x + 1$ of a one-dimensional problem given by (3.10). However, the left hand side of this system is only known for values of y_j, $j = 0, \ldots, N_y$ (*i.e.* it is a vector of $N_y + 1$ components) whereas the right hand side is a $(N_y + 3)$ vector. Some boundary conditions are necessary to close the system. Hermite boundary conditions are imposed for the first and last component of the vector (which corresponds to $j = -1$ and $j = N_y$), that is to say, we have to compute $\gamma'_v(y_0)$ and $\gamma'_v(y_{N_y})$, $\forall v = -1, \ldots, N_x + 1$. To achieve this task, we solve two systems: we first take the derivative of (3.10) with respect to the y variable, and then evaluate it in $y_j = y_0$ and $y_j = y_{N_y}$. The Hermite boundary conditions have to be adapted to this particular case. Consequently, we have to solve the two following systems (associated to $j = 0$ and $j = N_y$): $A\gamma'_v(y_j) = \partial_y f(x, y_j)$, where A is the matrix (3.7),

$$\gamma'_v(y_j) = [\gamma'_{-1}(y_j), \ldots, \gamma'_v(y_j), \ldots, \gamma'_{N_x+1}(y_j)]^T,$$

and the right hand side writes

$$\partial_y f(x, y_j) = [\partial_{xy} f(x_0, y_j), \partial_y f(x_0, y_j), \ldots, \partial_y f(x_i, y_j), \ldots, \partial_y f(x_{N_x}, y_j), \partial_{xy} f(x_{N_x}, y_j)]^T.$$

Once we have computed $\gamma'_v(y_0)$ and $\gamma'_v(y_{N_y})$ for all $v = -1, \ldots, N_x + 1$, we solve the system (3.10) which writes here $A\eta_{v,\beta} = \Gamma_{v\beta}$. The matrix A is given by (3.7) and the right hand side $\Gamma_{v\beta}$ reads

$$\Gamma_{v\beta} = [\gamma'_v(y_0), \gamma_v(y_0, \ldots, \gamma_v(y_{N_y}), \gamma'_v(y_{N_y})]^T,$$

for each value of $v = -1, \ldots, N_x + 1$.

Once the spline coefficients $\eta_{v,\beta}$ have been computed for all v and β, the value of f at the origin of the characteristics (X^n, V^n) (determined following the semi-Lagrangian method described in Section 2) is taken to be the value of the spline $s(X^n, V^n)$.

If (X^n, V^n) belongs to $[x_i, x_{i+1}] \times [y_j, y_{j+1}]$, the approximation of the function $f(X^n, V^n)$ is given by

$$s(X^n, V^n) = \sum_{v=i-1}^{i+2} \left(\sum_{\beta=j-1}^{j+2} \eta_{v,\beta} B_v(X^n) B_\beta(V^n) \right),$$

where B_v and B_β are computed by (3.1). To get $s(X^n, V^n)$ for all mesh points, it requires $\mathcal{O}(N_x N_y)$ floating-points operations.

In summary, we have to solve

$(N_y + 1)$ systems of size $(N_x + 3) \times (N_x + 3)$ (to get $\gamma_v(y_j) \quad \forall j = 0, \ldots, N_y$),
2 systems of size $(N_x + 3) \times (N_x + 3)$ (to get $\gamma'_v(y_0)$ and $\gamma'_v(y_{N_y})$),
$(N_x + 3)$ systems of size $(N_y + 3) \times (N_y + 3)$ (to get$\eta_{v,\beta}$).

From a computational cost point of view, the solution of a linear system of size N_x using the *LU* decomposition needs $\mathcal{O}(N_x)$ operations. Such a solution has to be done N_y times for the x-direction; the same is true for the y-direction. Finally, the two-dimensional interpolation of $N_x \times N_y$ points leads to $\mathcal{O}(N_x N_y)$ operations.

3.3. Towards an efficient parallelization

In order to get accurate numerical simulations, one has to take care of boundary conditions for each local LU solve. Indeed, our strategy consists in being as close as possible to the corresponding sequential version. Hence, from a decomposition of the global domain into several patches, each processor being devoted to a patch, one wants that our local solution of cubic spline coefficients recovers in the best way a usual solution on the global domain. To that purpose, some efforts have to be done to approximate the derivatives of f in a particular way with respect to x and y. The points where derivatives must be computed are shared between two processors since x_0 and x_N are both beginning and end of subdomains (x_N of the target processor corresponds to x_0 of the adjacent processor). Hence, these derivatives of f join adjacent subdomains and play an important role in the quality of the numerical results (see Fig. 4 in Section 5).

Different ways have been explored to obtain the derivatives: finite differences of different orders (centered and uncentered), an ad-hoc cubic spline approximation. In order to reconstruct a smooth approximation (let's say $\mathcal{C}^1$ on the global domain), the cubic spline approximation has been chosen. Indeed, we remark that even in regions where f is smooth enough, a finite differences approximation remains quite different from a cubic spline approximation given by (3.5). Hence, as we want to reconstruct the distribution function *via* a cubic spline approximation, the first line of the linear system the matrix of which is given by (3.7) can introduce some numerical errors that propagate into the rest of the system; in the numerical experimentations we have performed, the final results are damaged, especially when one observes the mass conservation. Indeed, the finite differences approximation leads to some variations of the mass conservation which is an inconvenient for the long time behaviour of the numerical solution. Moreover, when uncentered finite differences are used, unsymmetry is imposed at the interface of the subdomains leading to unstable results for the two-stream instability test case for example (see 5.1.3). On the contrary, the approximation of the derivatives using cubic splines enables us to obtain a robust code with a relatively small number of discretization points.

By constructing an approximation of the derivatives using the cubic spline coefficients and the equalities (3.4) and (3.5), we manage to overcome this kind of error (see Fig. 4(b) in Section 5). Moreover, the final global reconstructed numerical solution is consistent with a numerical solution which is computed through a sequential solution. Let us explain it in the following in the one-dimensional case (the multi-dimensional case can be deduced straightforwardly). First, relations (3.5) and (3.4) enable us to write

$$\begin{aligned} s'(x_i) &= \frac{1}{2h}(\eta_{i+1} - \eta_{i-1}), \\ &= \frac{1}{2h}\left(\frac{3}{2}f_{i+1} - \frac{1}{4}\eta_i - \frac{1}{4}\eta_{i+2} - \frac{3}{2}f_{i-1} + \frac{1}{4}\eta_{i-2} + \frac{1}{4}\eta_i\right), \\ &= \frac{3}{4h}(f_{i+1} - f_{i-1}) + \frac{1}{8h}(\eta_{i-2} - \eta_{i+2}), \end{aligned} \tag{3.11}$$

to obtain the following equality

$$s'(x_i) = \frac{3}{4h}(f_{i+1} - f_{i-1}) - \frac{1}{4}(s'(x_{i-1}) + s'(x_{i+1})). \tag{3.12}$$

If we inject (3.11) in (3.12) to compute $s'(x_{i\pm1})$, we obtain

$$\begin{aligned} s'(x_i) &= \frac{3}{4h}(f_{i+1} - f_{i-1}) - \frac{1}{4}\left(\frac{3}{4h}(f_{i+2} - f_{i-2}) + \frac{1}{8h}(\eta_{i-3} - \eta_{i+1} + \eta_{i-1} - \eta_{i+3})\right), \\ &= \frac{3}{4h}(f_{i+1} - f_{i-1}) - \frac{1}{4}\left(\frac{3}{4h}(f_{i+2} - f_{i-2})\right) - \frac{1}{16}(2s'(x_i) + s'(x_{i-2}) + s'(x_{i+2})), \end{aligned}$$

then we get another expression for the derivative of s

$$s'(x_i) = \frac{6}{7h}(f_{i+1} - f_{i-1}) - \frac{3}{14h}(f_{i+2} - f_{i-2}) + \frac{1}{14}(s'(x_{i+2}) - s'(x_{i-2})). \tag{3.13}$$

Thanks to (3.13), the evaluation of $s'(x_{i+2})$ and $s'(x_{i-2})$ leads to the following new approximation of $s'(x_i)$

$$\begin{aligned}\alpha s'(x_i) = {} & \tfrac{6}{7h}(f_{i+1} - f_{i-1}) - \tfrac{3}{14h}(f_{i+2} - f_{i-2}) + \tfrac{6}{98h}(f_{i+3} - f_{i+1} + f_{i-1} - f_{i-3}) \\ & - \tfrac{3}{14^2 h}(f_{i+4} - f_{i-4}) + \tfrac{1}{14^2}(s'(x_{i+4}) - s'(x_{i-4})).\end{aligned} \tag{3.14}$$

with $\alpha = (1 - 2/14^2)$. A last iteration allows us to obtain a high order approximation of the derivative of s

$$\alpha s'(x_i) = \sum_{j=-8}^{j=8} \omega_j f_{i+j} + \frac{1}{\alpha 14^2}(s'(x_{i+8}) + s'(x_{i-8})) + \frac{2}{\alpha 14^2} s'(x_i),$$

to obtain

$$\left(1 - \frac{2}{14^2} - \frac{2}{(1 - 2/14^2)14^2}\right) s'(x_i) = \sum_{j=-8}^{j=8} \omega_j f_{i+j} + \frac{1}{\alpha 14^2}(s'(x_{i+8}) + s'(x_{i-8})), \tag{3.15}$$

where the derivatives $s'(x_{i+8})$ and $s'(x_{i-8})$ are evaluated thanks to a finite differences approximation of order 4. For example, $s'(x_{i+8})$ is approximated by

$$s'(x_{i+8}) = (-f(x_{i+10}) + 8f(x_{i+9}) - 8f(x_{i+7}) + f(x_{i+6}))/(12h)$$

where h is the discretization step. Even if this choice does introduce some error in the final evaluation of $s'(x_i)$, this error becomes smaller as the number of terms used becomes higher because of the small coefficient it is multiplied by. The final approximation of $s'(x_i)$ then reads

$$\begin{aligned} s'(x_i) &= \sum_{j=-10}^{j=10} \tilde{\omega}_j f_{i+j}, \\ &= \sum_{j=-10}^{j=-1} \tilde{\omega}_j^- f_{i+j} + \sum_{j=1}^{j=10} \tilde{\omega}_j^+ f_{i+j}, \end{aligned} \tag{3.16}$$

since the coefficient $\tilde{\omega}_0$ is null here. Let us note that ω_j^- and ω_j^+ are computed once for all. More iterations could also be done, but formula (3.16) gives satisfying results.

The coefficients $\tilde{\omega}_j, j = -10, \ldots, 10$ are summarized in Table 1. The $\tilde{\omega}_j^+$ coefficients are given from the following relation: $\tilde{\omega}_j^+ = -\tilde{\omega}_j^-$.

4. Parallelization of the computations

In order to perform a parallelization of the interpolation step, data and computation have to be distributed onto processors. A classical technique of domain decomposition is used here to split the phase space into subdomains. Thus, a single processor works on local data and shares information located on the borders of its subdomain with adjacent processors. The set of values exchanged with the eight processors in the neighborhood of a given processor is named the *ghost* area (at the west, north-west, north, north-east, east, south-east, south and south-west). This area is needed because each processor has to know information belonging to others, in order to build the right hand side matrix of Section 3.2. Values of function f and some kind of derivatives are stored in the ghost zone in order to manage this step. From a parallel performance point of view, the number of values transmitted between processors must be minimal. So the ghost zone should be chosen as small as possible.

Indeed, on the patch, only points (x_i, y_j) for $i = 0, \ldots, N_x - 1$ and $j = 0, \ldots, N_y - 1$ are known, and the interpolation step requires the knowledge of values on the patches borders; moreover we have to evaluate the derivative in (x_0, y_j) and (x_{N_x}, y_j), for all j, (x_i, y_0) and (x_i, y_{N_y}) for all i, which requires (see the previous section) a linear combination of 21 points.

The knowledge of these points enables us to build and solve the LU systems, and to interpolate on $[x_0, x_{N_x}] \times [y_0, y_{N_y}]$. But we have to take into account the advected points that come out of the targeted patch. As mentioned in the Section 1, we impose a restriction on the time step to enforce the displacement to be lower than the cell size. Hence, the interpolation area

Table 1
Coefficients for the approximation of the derivatives.

$\tilde{\omega}_{-10}^-$	$\tilde{\omega}_{-9}^-$	$\tilde{\omega}_{-8}^-$	$\tilde{\omega}_{-7}^-$	$\tilde{\omega}_{-6}^-$
0.2214309755E−5	−1.771447804E−5	7.971515119E−5	−3.011461267E−4	1.113797807E−3
$\tilde{\omega}_{-5}^-$	$\tilde{\omega}_{-4}^-$	$\tilde{\omega}_{-3}^-$	$\tilde{\omega}_{-2}^-$	$\tilde{\omega}_{-1}^-$
−4.145187862E−3	0.01546473933	−0.05771376946	0.2153903385	−0.8038475846

becomes $[x_0-\Delta x, x_{N_x}+\Delta x]\times[y_0-\Delta y, y_{N_y}+\Delta y]$ (where Δx and Δy denote the discretization steps) and additional cubic spline coefficients have to be computed.

To that purpose, the solution of the linear systems described in Section 3.2 takes into account the following augmented right hand side matrix (the derivatives are approximated thanks to (3.16))

$$\begin{pmatrix} f(-1,-1) & \partial_y f(-1,0) & f(-1,0) & \cdots & \partial_y f(-1,N_y) & f(-1,N_y+1) \\ \partial_x f(0,-1) & \partial^2_{xy} f(0,0) & \partial_x f(0,0) & \cdots & \partial^2_{xy} f(0,Ny) & \partial_x f(0,N_y+1) \\ f(0,-1) & \partial_y f(0,0) & f(0,0) & \cdots & \partial_y f(0,N_y) & f(0,N_y+1) \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ f(N_x,-1) & \partial_y f(N_x,0) & f(N_x,0) & \cdots & \partial_y f(N_x,Ny) & f(N_x,N_y+1) \\ \partial_x f(N_x,-1) & \partial^2_{xy} f(N_x,0) & \partial_x f(N_x,0) & \cdots & \partial^2_{xy} f(N_x,N_y) & \partial_x f(N_x,N_y+1) \\ f(N_x+1,-1) & \partial_y f(N_x+1,0) & f(N_x+1,0) & \cdots & \partial_y f(N_x+1,N_y) & f(N_x+1,N_y+1) \end{pmatrix}. \tag{4.1}$$

The solution in the x-direction is done for all j, which gives us the temporary spline coefficients $\gamma_v(y_j)$, $v=-1,..,N_x$ and $j=-2,\ldots,N_y+1$. The coefficients corresponding to $v=-2$ and $v=N_x+1$ are deduced from (3.4) applied for $i=-1$ and $i=N_x$.

In the same way, the solution in the y-direction is done for all $v=-2,\ldots,N_x+1$, and gives the coefficients $\eta_{v,\beta}$ for $\beta=-1,\ldots,N_y+1$; the boundary values $\eta_{v,-2}$ and η_{v,N_y+1} are obtained from (3.4)

The target processor has to gather all points needed to compose the matrix (4.1). To that purpose, as the values of the distribution function are known at (x_i,y_j) for $i=0,\ldots,N_x-1$ and $j=0,\ldots,N_y-1$, the local ghost zone needs to receive from others processors

- $f(-1,j)$ for $j=0,\ldots,N_y-1$,
- $f(i,-1)$ for $i=0,\ldots,N_x-1$,
- $f(N_x:N_x+1,j)$ for $j=0,\ldots,N_y-1$,
- $f(i,N_y:N_y+1)$ for $i=0,\ldots,N_x-1$,
- $f(-1,-1)$,
- $f(-1,N_y:N_y+1)$,
- $f(N_x:N_x+1,-1)$,
- $f(N_x:N_x+1,N_y:N_y+1)$,
- some weighted summations of 10 points which are computed on the neighboring processors to evaluate all derivatives.

5. Numerical simulations

In this section, some numerical results are described to demonstrate the accuracy and efficiency of our method. Especially, we should emphasize the scalability of the method comparing sequential and parallel simulations and testing it on various problems that occur in plasma physics: the Landau damping and the two-stream instability test cases in two dimensions of the phase space. We then extend the scheme to the four-dimensional phase space that consists of two spatial directions x, y and their velocity directions v_x, v_y. A subsection is devoted to performance analysis and comparison with the previous approach.

5.1. Two-dimensional phase space test cases

In order to highlight the numerical features of the methods, we shall first consider simple cases in two-dimensional phase space composed of space x and velocity v_x. We numerically solve the Vlasov equation

$$\frac{\partial f}{\partial t}+v_x\frac{\partial f}{\partial x}+E_x(t,x)\frac{\partial f}{\partial v_x}=0, \tag{5.1}$$

coupled with the one-dimensional Poisson equation

$$\frac{\partial E_x}{\partial x}=\int_{\mathbb{R}} f(t,x,v_x)dv_x-1. \tag{5.2}$$

We use a cartesian mesh to represent the $x-v_x$ phase space with the computational domain $(x,v_x)\in[0,L]\times[-v_{max},v_{max}]$, where L is the spatial length and v_{max} is the cutoff velocity. The number of mesh points used in the x and v_x directions is designated by N_x and N_v, respectively.

5.1.1. Linear Landau damping

The first example is the linear Landau damping (see [11,12,26,27]). The initial condition associated to the scaled Vlasov–Poisson Eqs. (5.1) and (5.2) has the following form

$$f_0(x, v_x) = \frac{1}{\sqrt{2\pi}} \exp(-v_x^2/2)(1 + \alpha \cos(kx)), \quad (x, v_x) \in [0, L] \times \mathbb{R}, \tag{5.3}$$

where k is the wave number and $\alpha = 0.001$ is the amplitude of the perturbation. We choose $v_{max} = 6$, $L = 2\pi/k$ and the following numerical parameters: $N_x = 64$, $N_v = 64$. The boundary conditions for the distribution function are periodic in the space variable and vanishing in the velocity direction. The final time is $T = 60\omega_p^{-1}$, with ω_p the plasma frequency.

In this test, we are interested in the evolution of the square root of the electric energy approximated by

$$\mathcal{E}_h(t) = \left(\sum_i (E_x)_i^2(t) \Delta x \right)^{1/2}, \tag{5.4}$$

where Δx is the space step. According to Landau's theory, the amplitude of $\mathcal{E}_h(t)$ is expected to be exponentially decreasing.

On Fig. 1, we represent the evolution of $\log(\mathcal{E}_h(t))$ in the parallel case, for two different values of the wave number: $k = 0.3$ and $k = 0.5$. The phase space domain is decomposed into 4 patches of the same size 32×32 points, so that the global domain involves 64×64 points; moreover, Hermite boundary conditions are imposed at the boundary of each patch using the approximation (3.16). We observe that $\mathcal{E}_h(t)$ is exponentially decreasing, and the damping rate becomes larger when k increases, as predicted by the Landau theory. The numerical damping rates are $\gamma = 0.0127$ for $k = 0.3$, and $\gamma = 0.154$ for $k = 0.5$, which are very similar to the predicted values available in the literature (see [2,11,22]).

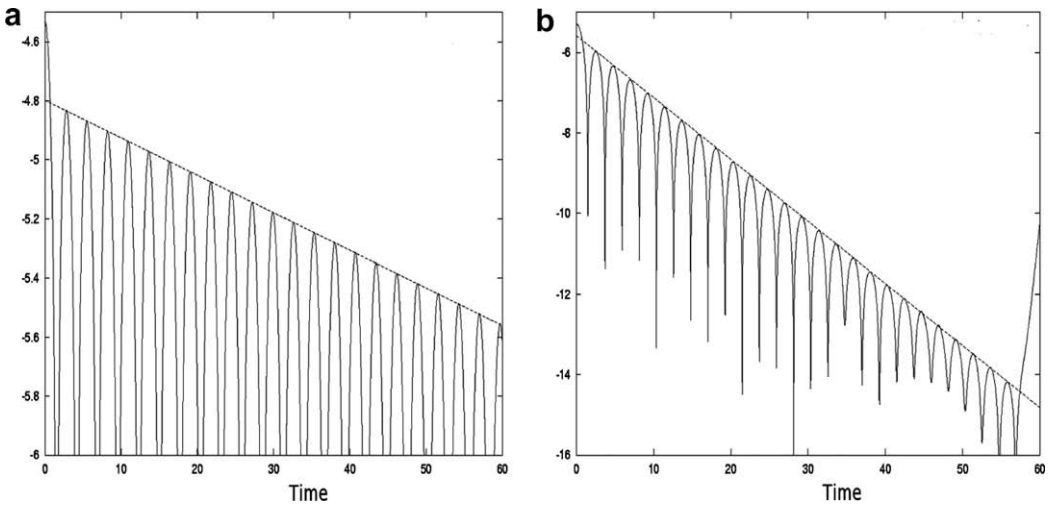

Fig. 1. Electric energy as a function of time for the linear Landau damping in the parallel case, with $N_x = N_v = 64$ points: (a) $k = 0.3$ and (b) $k = 0.5$.

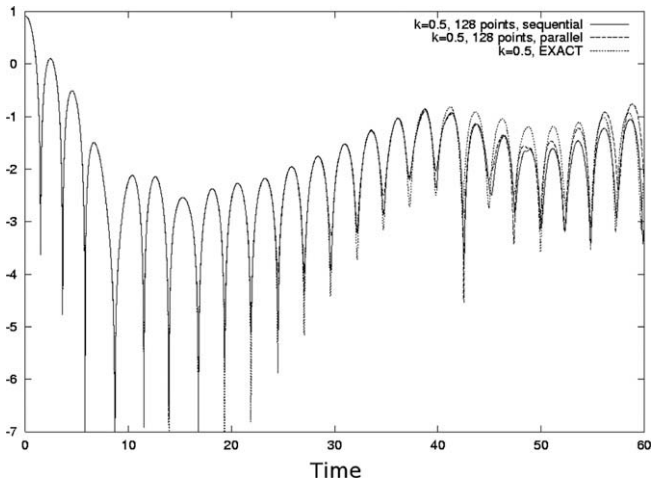

Fig. 2. Electric energy as a function of time for the strong Landau damping. Comparison between the sequential and the parallel case. $k = 0.5$ and $N_x = N_v = 128$. An almost "exact" solution (512×1024 points) is plotted for comparison.

5.1.2. Strong Landau damping

When the initial amplitude in (5.3) of the perturbation increases, it turns into strong Landau damping which has been computed by many authors [5,11,12,23,25–27]. The initial condition, given by (5.3) considers here $\alpha = 0.5$, $k = 0.5$. Moreover $v_{max} = 6.5$, and the number of cells will be equal to $N_x = N_v = 128$.

We are also interested in the evolution of $\log(\mathcal{E}_h(t))$ (where $\mathcal{E}_h(t)$ is given by (5.4)) as a function of time. The linear theory of the previous test can not be applied in this case since the nonlinear effects have to be taken into account.

On Fig. 2, we compare the evolution of the logarithm of the electric energy between the sequential case and the parallel case. We notice that the amplitude of the electric energy is first exponentially decreasing in time, and oscillates around a constant for larger times for the two simulations. Moreover, the electric energy reaches its minimum at $t \approx 15\omega_p^{-1}$. It grows exponentially until $t \approx 40\omega_p^{-1}$ and then saturates. We then remark that even until large times, the two curves (associated to the sequential and the parallel cases) are very similar, and only at $t \simeq 50\omega_p^{-1}$, they become slightly different. As in the linear case, the sequential and the parallel case present quite good results compared to results available in the literature.

Fig. 3 shows the time development of the spatially integrated distribution function $F(v_x)$ as a function of the velocity v_x

$$F(v_x) = \int_0^{2\pi/k} f(x, v_x)dx,$$

in the parallel case. We observe that particles whose kinetic energy is smaller than the potential energy are trapped by electrostatic waves around the phase velocity $v_{ph} = \omega/k \approx 2.64$, where small bumps appear preceded by small holes in the region $2 < v_x < 2.5$ until $t \approx 25\omega_p^{-1}$. These numerical results are in very good agreement with those obtained by [12,26].

Moreover, to emphasize the influence of the approximation of the derivative on the results, we plot on Fig. 4 the evolution in time of the total relative mass. Comparisons between the parallel and sequential case are presented in two different contexts; on Fig. 4(a), all the derivatives are approximated through the following fourth order finite differences operator

$$s'(x_i) \simeq \frac{-f(x_{i-2}) + 8f(x_{i-1}) - 8f(x_{i+1}) + f(x_{i+2})}{12h}, \tag{5.5}$$

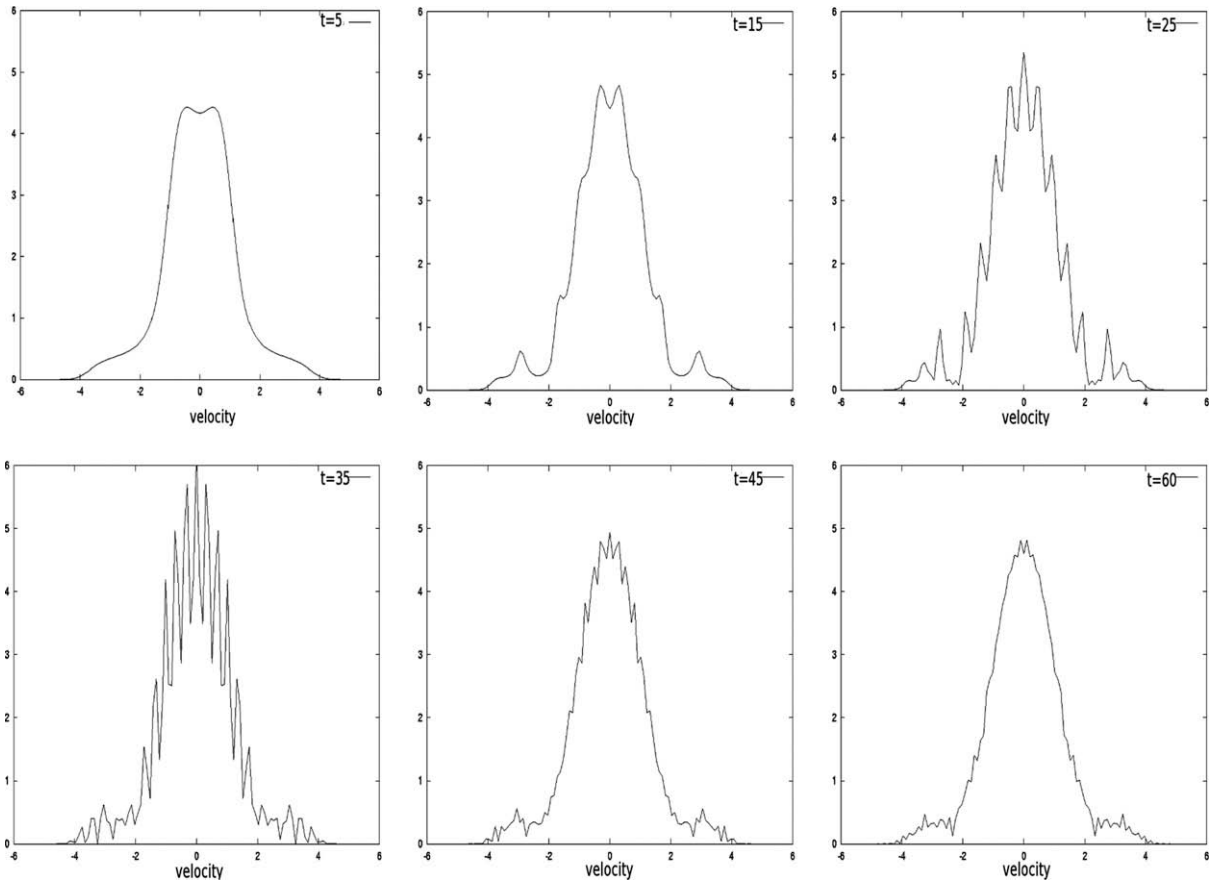

Fig. 3. Time development of the spatially integrated distribution function for the strong Landau damping. Parallel case. $N_x = N_v = 128$. The figures correspond to $t = 5, 15, 25, 35, 45, 60\omega_p^{-1}$.

where h is the step corresponding to the derivative direction. On Fig. 4(b), the derivatives are replaced by the formula (3.16). We can observe that the finite differences approximation (5.5) is not well suited for the parallel implementation since the total mass presents some important oscillations from $t \simeq 20\omega_p^{-1}$, these fluctuations becoming greater when time increases. On the contrary, the use of cubic spline derivative approximation with 21 points leads to a mass conservation which is very similar to the mass conservation occuring in the sequential case. The use of higher order finite differences operators does not improve the results since they converge towards a different value from that given by the derivative computed using cubic splines approximation. Let us remark that the use of finite differences or cubic spline approximation does not affect the mass conservation in the sequential case.

5.1.3. Two-stream instability

We consider the symmetric two-stream instability with the following initial condition

$$f(0,x,v_x) = \frac{1}{\sqrt{2\pi}} v_x^2 \exp(-v_x^2/2)(1+\alpha\cos(kx)), \quad (x,v) \in [0,2\pi/k] \times \mathbb{R},$$

where $\alpha = 0.05$, $k = 0.5$, and $v_{max} = 9$. The number of mesh points is $N_x = 128$ in space and $N_v = 128$ in velocity to get a good accuracy. The final time is $T = 500\omega_p^{-1}$. For more details on this test, we refer the reader to [2,3,8,12,26].

Fig. 5 show the time development of the distribution function in phase space, in the parallel case (4 patches equally decompose the phase space domain). From time $t \simeq 10\omega_p^{-1}$ to $t \simeq 20\omega_p^{-1}$, the instability grows rapidly and a hole structure appears. After $t \simeq 20\omega_p^{-1}$, the trapped particles oscillate in the electrostatic potential and the vortex rotates periodically. These remarks are in good agreement with the results available in [2,12,26].

Moreover, the inherent precision of the cubic spline interpolation allows to follow thin filaments developed by the solution; even if the methodology which enables the parallelization is slightly different from the sequential version, we observe that the parallelization does not affect the precision due to the spline interpolation.

5.2. Four-dimensional phase space test cases

In this section, we extend the scheme to the four-dimensional phase space. Several tests applied to physical configurations have been implemented. The semi-Lagrangian method presented in Section 2 is used in the different situations we will encounter in the sequel, together with the local spline interpolation. A cartesian grid represents the phase space with the computational domain $(x,y,v_x,v_y) \in [0,L_x] \times [0,L_y] \times [-v_{max},v_{max}]^2$. The number of mesh points is always designated by N_x for the two-dimensional space direction, and N_v for the two-dimensional velocity direction.

5.2.1. Two-dimensional Landau damping

This first test corresponds to the numerical solution of the four-dimensional Vlasov–Poisson equation

$$\frac{\partial f}{\partial t} + \mathbf{v}\cdot\nabla_{\mathbf{x}} f + E(t,\mathbf{x})\cdot\nabla_{\mathbf{v}} f = 0, \tag{5.6}$$

with $\mathbf{x} = (x,y)$ and $\mathbf{v} = (v_x,v_y)$, and where the electric field $E(t,\mathbf{x})$ solves the two-dimensional Poisson equation

$$\nabla_{\mathbf{x}}\cdot E = \int_{\mathbb{R}^2} f(t,\mathbf{x},\mathbf{v})d\mathbf{v} - 1. \tag{5.7}$$

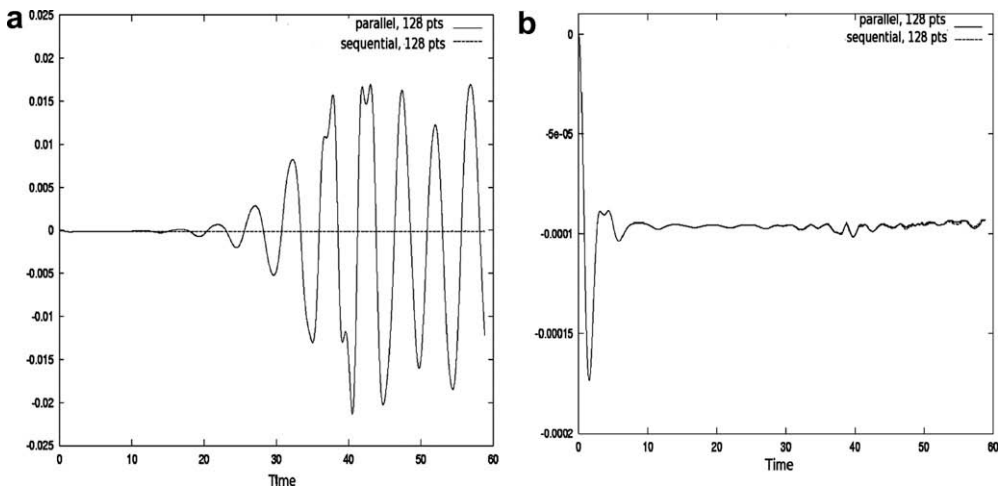

Fig. 4. Comparison between the sequential and parallel case for the total relative mass conservation as a function of time, for the strong Landau damping: (a) fourth order finite differences approximation (5.5) and (b) cubic spline approximation with 21 points (3.16).

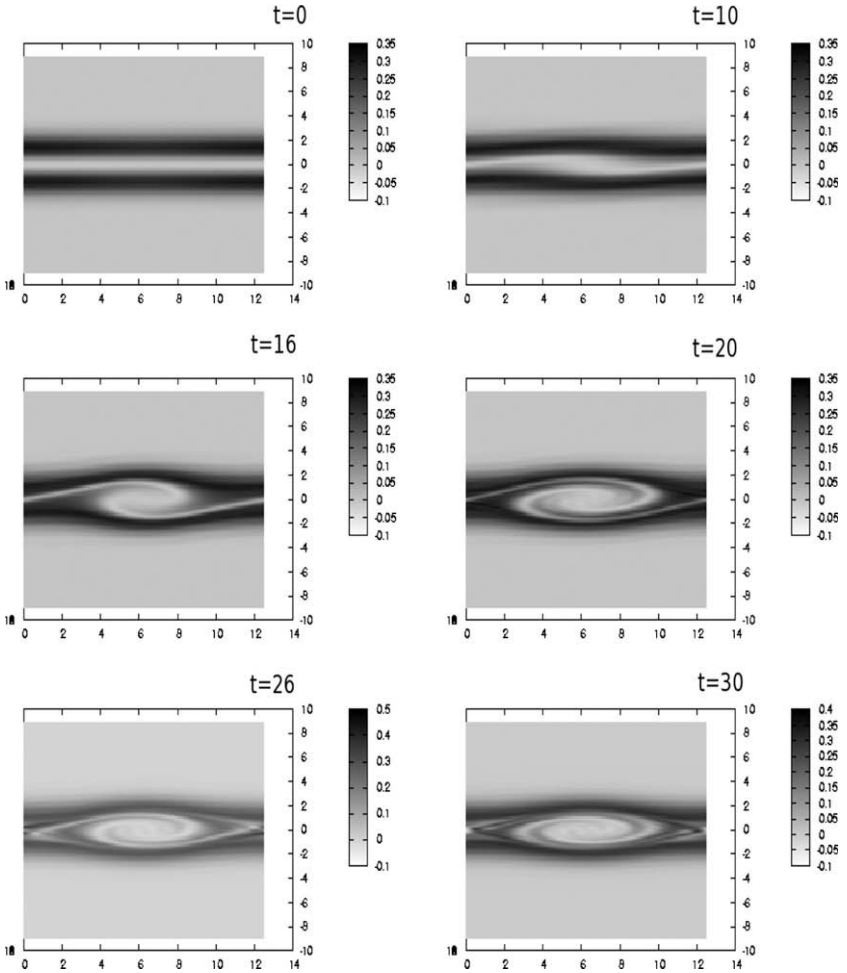

Fig. 5. Time evolution of the distribution function f in the phase space, with $N_x = N_v = 128$ in the parallel case (four processors are used), for the two-stream instability test. The figures correspond to $t = 0, 10, 16, 20, 26, 30\omega_p^{-1}$.

The associated initial condition is set to

$$f_0(x, y, v_x, v_y) = \frac{1}{2\pi} \exp\left(-\frac{v_x^2 + v_y^2}{2}\right)(1 + \alpha \cos(k_x x)\cos(k_y y)), \tag{5.8}$$

with $\alpha = 0.05$ for the linear configuration whereas $\alpha = 0.5$ is considered for the strong Landau damping. The velocity space is truncated to $v_{max} = 6$, the wave numbers are $k_x = k_y = 0.5$, and the length of the periodic box in the physical space is $L_x = L_y = 4\pi$. Finally, the four-dimensional grid contains $N_x = 32^2$ points for the spatial direction and $N_v = 128^2$ points for the velocity direction; from a parallel performance point of view, we impose a restriction on the time step: the displacements in a direction have to be smaller than the cell size corresponding to this direction. In this test, this CFL type condition depends only on the $x - y$ displacement since the amplitude of the electric field (and consequently of the displacements in the v_x and v_y directions) is not very important; then the time step must satisfy the following CFL condition: $\Delta t < \Delta x / v_{max}$.

The numerical simulation of Landau damping test cases in the four-dimensional phase space is quite difficult since the number of grid points is strongly limited by computer memory; hence, examples of simulations are not frequent in the literature (see [12,26]). Indeed, it requires the use of high order schemes and of parallel computations.

On the one hand, we plot on Fig. 6 the time evolution of the electric energy $\mathcal{E}(t)$

$$\mathcal{E}(t) = \left[\sum_{i,j}\left((E_x)_{i,j}^2 + (E_y)_{i,j}^2\right)\right]^{1/2},$$

as a function of time obtained by the semi-Lagrangian method using local cubic spline interpolation, in the linear context. We can observe the exponential decay of the amplitude of $\mathcal{E}(t)$ with $\gamma = 0.4$ (damping rate of the oscillations) and $\omega = 1.676$ (frequency of the oscillations) which is in a good agreement with the theoritical values $\gamma = -0.394$ and $\omega = 1.682$.

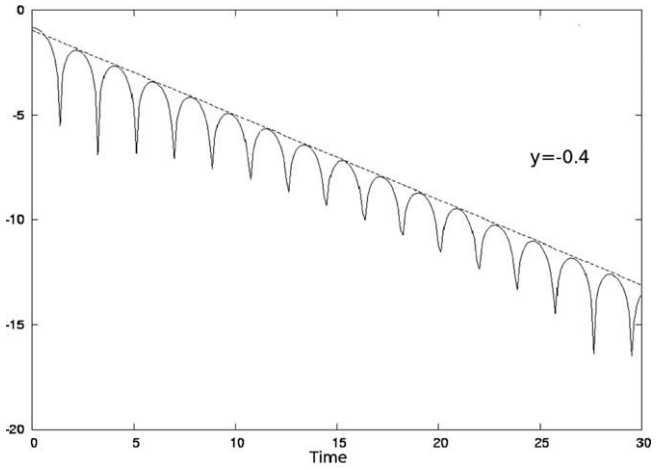

Fig. 6. Time evolution of the electric energy for the linear Landau damping solving the four-dimensional Vlasov–Poisson equation. $k_x = k_y = 0.5$, $N_x = 32^2$, $N_v = 128^2$ and eight processors are used.

On the other hand, for the nonlinear case plotted on Fig. 7, we recover a damping coefficient which is in a good agreement with [12], and we also notice that the amplitude of the electric energy is oscillating around a constant for large times (as in [12]). This test has been performed using $N_x = 32^2$ points in spatial direction and $N_v = 128^2$ in the velocity direction. The two-dimensional velocity space is decomposed into eight subdomains (*i.e.* eight processors). We can observe on Fig. 8 that the method inherits the accuracy of the cubic splines interpolation; indeed, the L^p norms has a reasonable behaviour and the total energy presents small oscillations, the amplitude of which does not exceed a few percents.

5.2.2. Beam focusing problems

We now consider the evolution of a RMS matched Gaussian beam in a focusing channel. In this case, an appropriate model is the paraxial model which can be derived from the six-dimensional Vlasov equation (see [9,13]). This model is much simpler than the full Vlasov–Maxwell system since it includes only four dimensions; it can be written, for all $\mathbf{x} = (x, y)$ and $\mathbf{v} = (v_x, v_y)$

$$\frac{\partial f}{\partial t} + \mathbf{v} \cdot \nabla_{\mathbf{x}} f + (E_{self}(t, \mathbf{x}) + E_{appl}(t, \mathbf{x}) + (\mathbf{v}, v_b)^T \times B_{appl}(t, \mathbf{x}, z)) \cdot \nabla_{\mathbf{v}} f = 0, \tag{5.9}$$

where E_{self} is the self-consistent electric field given by the Poisson equation, v_b is the propagating velocity of the beam along the z-axis and E_{appl} (resp. B_{appl}) is an external electric (resp. magnetic) field to focalize the beam.

Then, we intend to solve the four-dimensional model (5.9). For the external forces, three different types are mostly considered in accelerators for modeling purposes: uniform focusing by a continuous field, periodic focusing and alternating

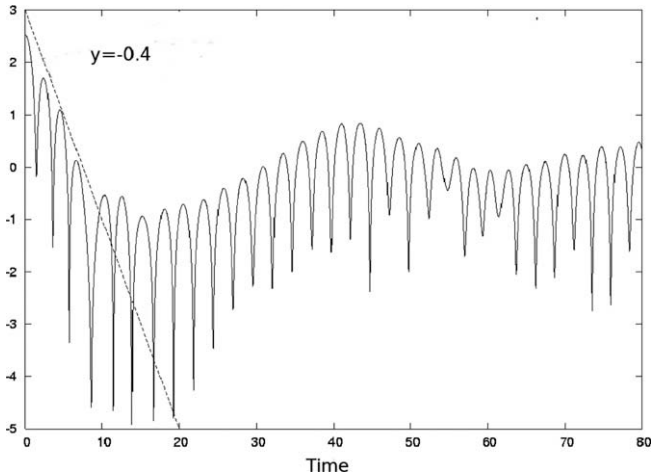

Fig. 7. Time evolution of the electric energy for the nonlinear Landau damping solving the four-dimensional Vlasov–Poisson equation. $k_x = k_y = 0.5$, $N_x = 32^2$, $N_v = 128^2$ and eight processors are used.

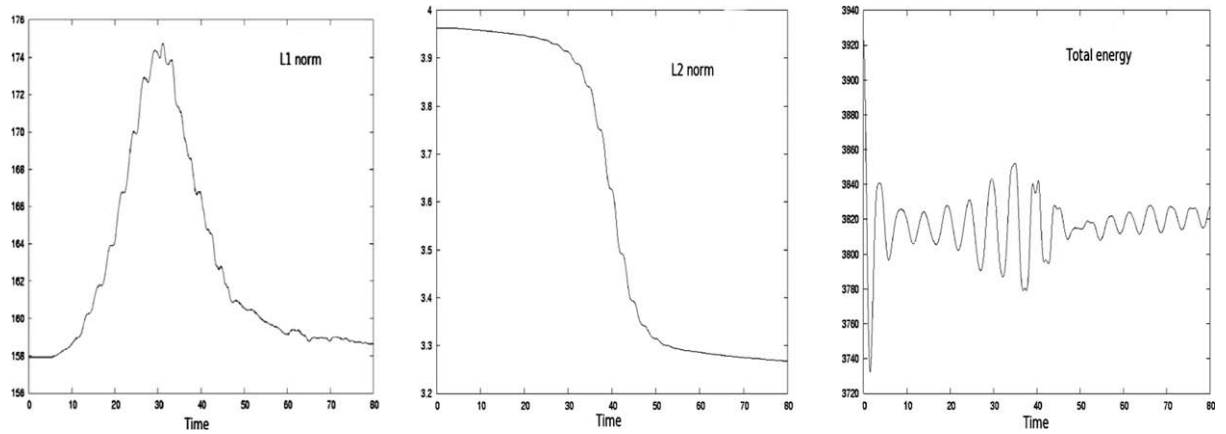

Fig. 8. Time evolution of the L^p norms ($p = 1, 2$) and of the total energy for the nonlinear Landau damping solving the four-dimensional Vlasov–Poisson equation. $k_x = k_y = 0.5$, $N_x = 32^2$, $N_v = 128^2$ and eight processors are used.

gradient focusing. In this section, we consider an alternating gradient focusing configuration, taking into account a Gaussian initial condition of the form

$$f_0(x, y, v_x, v_y) = \frac{1}{(2\pi)^2} \exp\left(-\frac{x^2 + y^2 + v_x^2 + v_y^2}{2}\right). \tag{5.10}$$

Many papers have been devoted to focusing beam problems, but essentially using PIC codes; we mention here some works where simulations are performed using Vlasov simulations [13,28,29].

We present a beam of ionized potassium particles focused with an alternating gradient method: the applied electric field is given by

$$E_{app}(t, \mathbf{x}) = \begin{pmatrix} +k_0(t)x \\ -k_0(t)y \end{pmatrix},$$

where for $t \in [0, T], T = 1m$, $k_0(t)$ is positive, null and negative. The initial condition is a Gaussian distribution function and the physical parameters are the following: the current I is 40 mA, the energy of the beam is equal to 1 MeV, and the emittance is 50π mm mrad.

For the numerical parameters, we choose $v_{max} = 32$, $L_x = L_y = [-6, 6]$, and $N_x = N_v = 128$. The time step is equal to $dt = 4.64 \times 10^{-4}$, whereas the simulations are performed on two periods.

We present the evolution of the RMS quantities of the Gaussian beam; in Fig. 9, we also plot the RMS quantities of the equivalent KV beam for comparison. As expected, the RMS quantities of the Gaussian beam are not exactly periodic, but they remain close to the ones corresponding to the periodic KV beam (it is especially the case for the x_{rms} quantity for example).

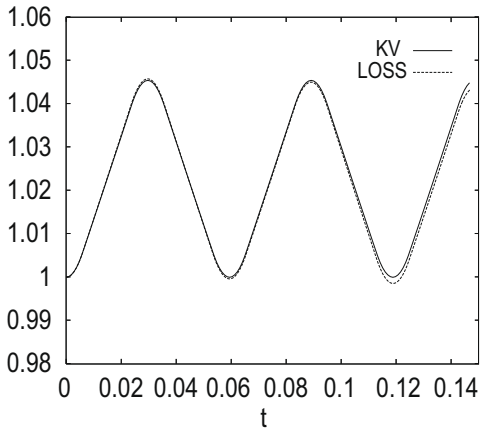

Fig. 9. Time evolution of the X_{rms} quantity obtained from the numerical solution of the paraxial model, with a alternating gradient electric field, and its associated KV beam.

Moreover, on Figs. 10 and 11, we plotted the snapshots of the projection onto the $x - v_x$ and the $y - v_y$ planes of the distribution function. We can observe that, as for the KV beam, the axial phase space is not elliptical because the beam is matched to nonlinear forces. The beam is focused in one direction and defocused in the other.

5.3. Performance analysis

We comment in this section the execution times of the code. In Fig. 12, we present some speedup results relative to the 2*D* case. The experiments were conducted on two parallel computers: a cluster (the IBM machine belongs to the M3PEC group, Bordeaux 1 University) of 16 IBM nodes (16-way Power5 processors at 1.9 Ghz), and a shared memory SGI machine (located at Montpellier, France, at the computing center CINES) of 512 processors (Origin 3800 with 500 Mhz processors). The results are quite good since they are super-linear due to so-called cache effects. Let us remark that the diagnostics are not taken into account in these results. Moreover, the experiments dictate the fact that a minimum number of points has to be considered (for small patches, the overhead in computations to estimate the derivatives becomes too large).

In Fig. 13 some speedup results relative to the 4*D* case for two parallel machines are presented, up to a large number of processors. As expected regarding the 2*D* results, the results are good. The communication–computation overlap that has been implemented along with cache effects enables us to get very satisfying speedup results.

Moreover, on Table 2, we present the performances of the LOSS code (in its 4D version) towards the SLV2D code (see [7]). In this latter code, a similar semi-Lagrangian method using cubic splines interpolation is used; but a global transposition between the space advection and the velocity one is performed, which involves a huge amount of communication when

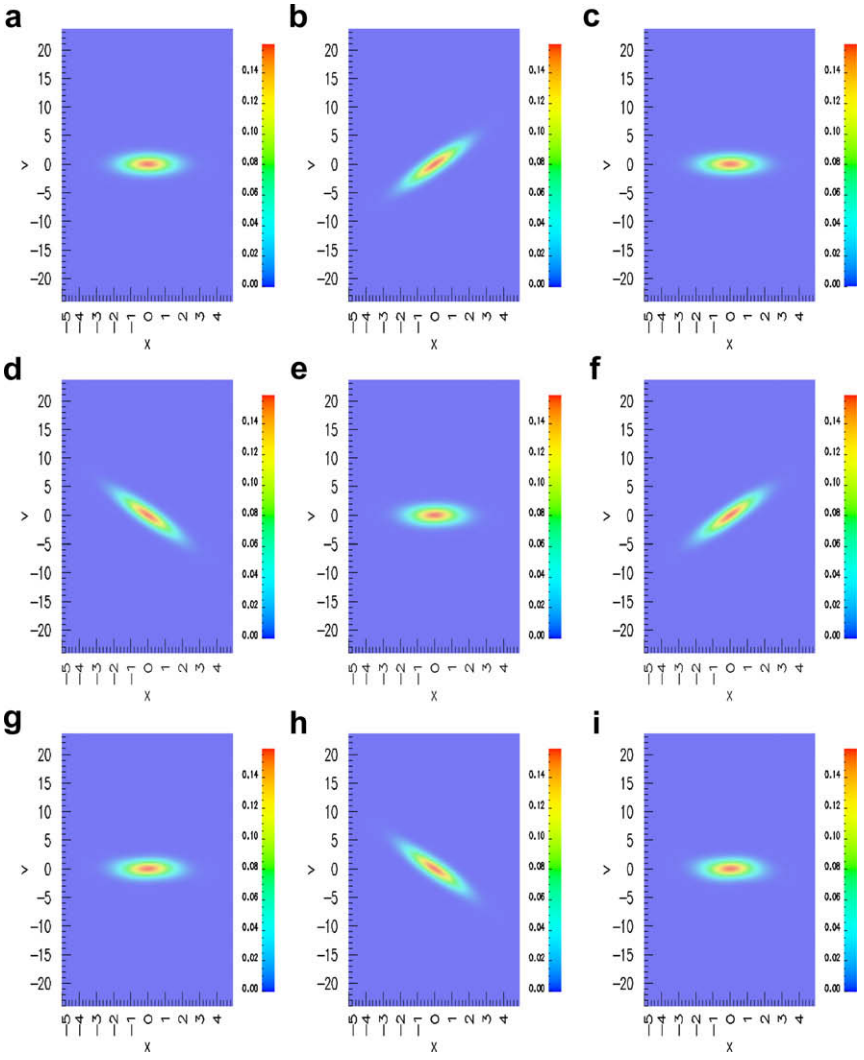

Fig. 10. Time evolution of $x - v_x$ projection of the distribution function: (a) $t = 0$, (b) $t = T/4$, (c) $t = T/2$, (d) $t = 3T/4$, (e) $t = T$, (f) $t = 5T/4$, (g) $t = 3T/2$, (h) $t = 7T/4$, (i) $t = 2T$, with $T = 0.05931$.

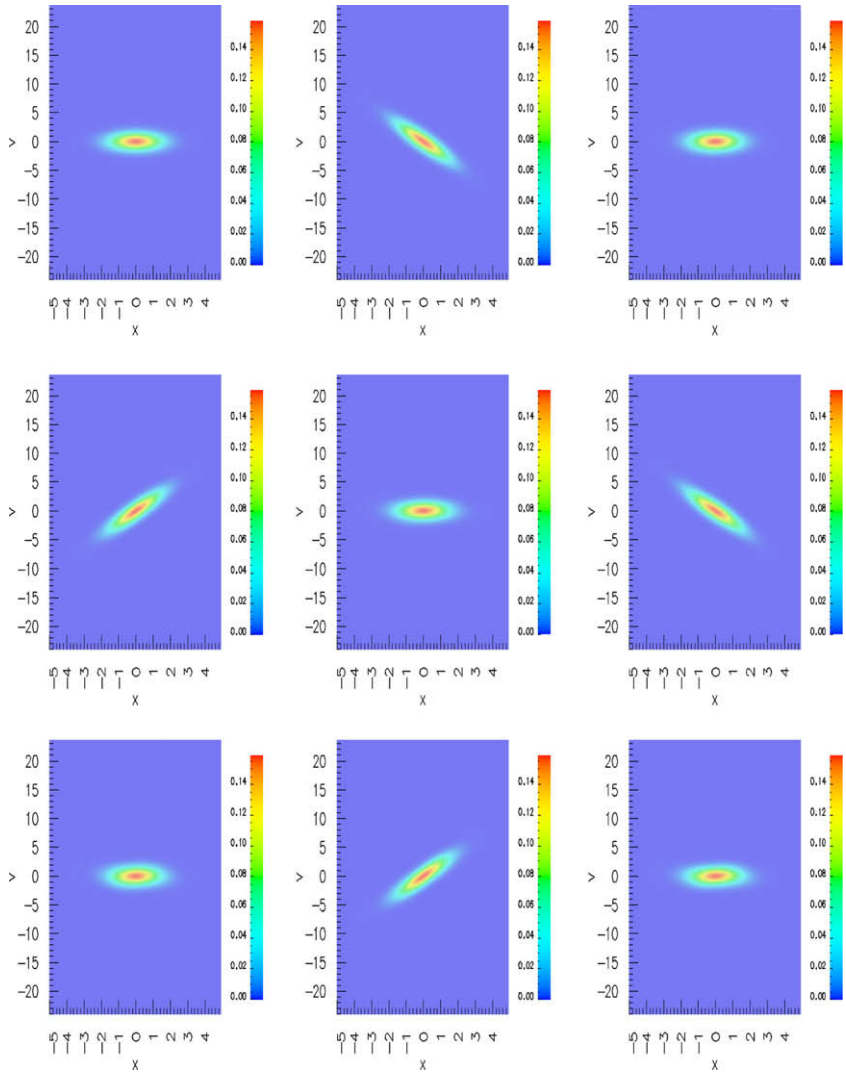

Fig. 11. Time evolution of $y - v_y$ projection of the distribution function: (a) $t = 0$, (b) $t = T$, (d) $t = 3T/4$, (e) $t = T$, (f) $t = 5T/4$, (g) $t = 3T/2$, (h) $t = 7T/4$, (i) $t = 2T$, with $T = 0.05931$.

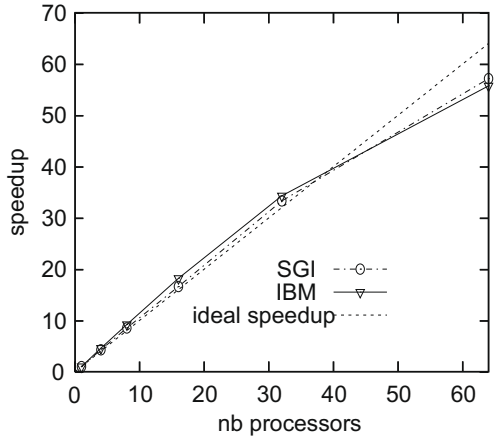

Fig. 12. Speedup for the two-stream instability on a shared memory SGI machine and on a cluster of 16 IBM nodes. The results corresponds to 512×512 points in the phase space; the simulation is stopped after 300 iterations.

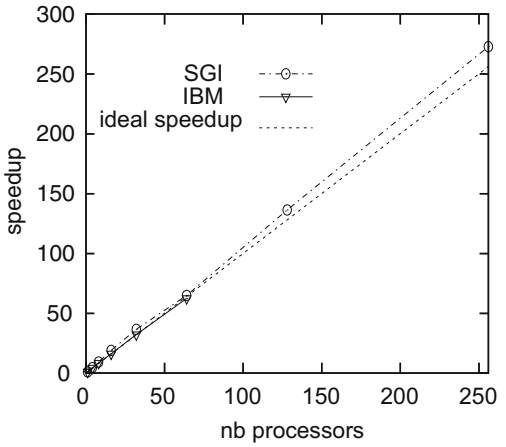

Fig. 13. Speedup for the four-dimensional case on a shared memory SGI machine and on a cluster of 16 IBM nodes. The results corresponds to $512 \times 512 \times 32 \times 32$ points in the phase space; the simulation is stopped after eight iterations.

Table 2
Computation time for one time step in the four-dimensional case for SLV2D and LOSS on a grid5000 machine with a 256^4 domain and 32 nodes (128 processors).

	SLV2D	LOSS4D
Time transpose/comm. (s)	41.0	13.9
Time (in s) v advection	4.0	2.2
Time (in s) x advection	4.0	4.2
Time (in s) field solver	0.9	0.5
Time (in s) total	49.9	20.8

the number of processors becomes important. Thus, the two codes have almost the same algorithmic complexity, the main difference lies in the communication pattern. Let us mention that the LOSS code is able to combine the MPI and OpenMP programming models (see [24]) to exploit levels of parallelism at a very fine grain. Hence, one can use a larger number of processors.

Let P be the number of processors. Also, let be TH the number of threads used in the LOSS code (induced by OpenMP parallelization). Moreover, we define by N_v (resp. N_x) the total number of points in the velocity (resp. spatial) direction and we assume that $N_{v_x} = N_{v_y}$ so that $\sqrt{N_v} = N_{v_x} = N_{v_y}$. At each time step of SLV2D, the whole 4D distribution function is sent twice on the machine network. During the transposition of an advection, each processor has to send (respectively received) around $C_{slv} = (\lceil \sqrt{N_v}/P \rceil \sqrt{N_v}) N_x$ floating-point values. For the LOSS code, only exchange of boundary values of the patches are performed between logically adjacent processors. Let pm be the perimeter of one patch taken in the $v_x - v_y$ directions. The number of floating-point values sent (respectively, received) during an advection for one processor is $C_{loss} = (3pm/TH)N_x$. For typical big runs, we suppose that $TH \geq s4$ and $pm = 128$ (for patches of size 32×32); thus, the inequality $3pm/H < 128$ is always satisfied. In such configurations, for all test cases involving more than 128^4 grid points, we have $(\lceil \sqrt{N_v}/P \rceil \sqrt{N_v}) \geq 128$. Then, it induces that $C_{loss} < C_{slv}$, so that the LOSS code save communications compared to SLV2D.

We show performance results for a test case of 256^4 points in the phase space using 128 processors. This kind of test can not be performed using a smaller number of processors due to memory limitation. On the machine we used, each node of four processors hosts 4GB of RAM memory. During one run, the LOSS code takes 3.4GB per node, whereas the SLV2D code takes 2.4GB. In Table 2, we focus on the execution time for one typical time step. The transposition/communication step is three times longer for the SLV2D code than for the LOSS code. Furthermore, the advection step in v is faster with the LOSS code because of cache effects. Hence, the patches are considered one after the other, and each patch of 32×32 points fit into the cache. It induces temporal locality leading to fast access to memory. With this example, we want to show that the LOSS code get shorter execution times than the SLV2D code on quite big test cases.

6. Conclusion

In this paper, we introduced a local semi-Lagrangian method which has been applied to simulate four-dimensional Vlasov type equations. The methodology presents a good behaviour when it is tested on plasma or beam configurations, even when the number of processors is important. Hence, an important number of points can be taken into account to well describe the distribution function. This kind of method is interesting in order to benchmark numerical results obtained by PIC codes. For

example, the method has been implemented with success in the GYSELA code [18], which enables massively parallel gyrokinetic simulations. This kind of decomposition is also feasible using 3 or 4 dimensional patches so that a larger number of dimensions can be parallelized. It should become conceivable to solve six-dimensional problems on today's supercomputers.

References

[1] R. Bermejo, Analysis of an algorithm for the Galerkin-characteristic method, Numer. Math. 60 (1991) 163–194.
[2] N. Besse, E. Sonnendrücker, Semi-Lagrangian schemes for the Vlasov equation on an unstructured mesh of phase space, J. Comput. Phys. 191 (2003) 341–376.
[3] C.K. Birdsall, A.B. Langdon, Plasma Physics via Computer Simulation, Institute of Physics Publishing, Bristol and Philadelphia, 1991.
[4] C. DeBoor, A Practical Guide to Splines, Springer Verlag, New York, 1978.
[5] C.Z. Cheng, G. Knorr, The integration of the Vlasov equation in configuration space, J. Comput. Phys. 22 (1976) 330.
[6] G.-H. Cottet, P. Koumoutsakos, Vortex Methods – Theory and Practice, Cambridge-University Press, 2000.
[7] O. Coulaud, E. Sonnendrücker, E. Dillon, P. Bertrand, A. Ghizzo, Parallelization of semi-Lagrangian Vlasov codes, J. Plasma Phys. 61 (1999) 435–448.
[8] N. Crouseilles, G. Latu, E. Sonnendrücker, Hermite Spline interpolation on patches for parallelly solving of the Vlasov–Poisson equation, Int. J. Appl. Math. Comput. Sci. 17 (2007) 101–115.
[9] P. Degond, P.-A. Raviart, On the paraxial approximation of the stationary Vlasov–Maxwell system, Math. Mod. Meth. Appl. Sci. 3 (1993) 513–562.
[10] M.R. Feix, P. Bertrand, A. Ghizzo, in: B. Perthame (Ed.), Series on Advances in Mathematics for Applied Science: Kinetic Theory and Computing, 1994.
[11] F. Filbet, E. Sonnendrücker, P. Bertrand, Conservative numerical schemes for the Vlasov equation, J. Comput. Phys. 172 (2001) 166–187.
[12] F. Filbet, E. Sonnendrücker, Comparison of Eulerian Vlasov solvers, Comput. Phys. Comm. 151 (2003) 247–266.
[13] F. Filbet, E. Sonnendrücker, Modeling and numerical simulation of space charge dominated beams in the paraxial approximation, Math. Mod. Meth. Appl. Sci. 16 (2006) 1–29.
[14] F. Filbet, E. Violard, Parallelization of a Vlasov solver by communication overlapping, in: Proceedings PDPTA 2002, 2002.
[15] A. Ghizzo, P. Bertrand, M.L. Begue, T.W. Johnston, M. Shoucri, A Hilbert–Vlasov code for the study of high-frequency plasma beatwave accelerator, IEEE Trans. Plasma Sci. 24 (1996) 370.
[16] A. Ghizzo, P. Bertrand, M. Shoucri, T.W. Johnston, E. Filjakow, M.R. Feix, A Vlasov code for the numerical simulation of stimulated Raman scattering, J. Comput. Phys. 90 (1990) 431–457.
[17] V. Grandgirard, M. Brunetti, P. Bertrand, N. Besse, X. Garbet, P. Ghendrih, G. Manfredi, Y. Sarazin, O. Sauter, E. Sonnendrücker, J. Vaclavik, L. Villard, A drift-kinetic semi-Lagrangian 4D code for ion turbulence simulation, J. Comput. Phys. 217 (2006) 395–423.
[18] V. Grandgirard, Y. Sarazin, X. Garbet, G. Dif-Pradalier, P. Ghendrih, N. Crouseilles, G. Latu, E. Sonnendrücker, N. Besse, P. Bertrand, A full-f global gyrokinetic semi-Lagrangian code for ITG turbulence simulations, in: Proceedings of Theory of Fusion Plasmas, Varenna, 2006.
[19] G. Hämmerlin, K.H. Hoffmann, Numerical Mathematics, Springer Verlag, New York, 1991.
[20] R.W. Hockney, S.W. Eastwood, Computer Simulation Using Particles, Hilger Bristol, UK, 1988.
[21] C.C. Kim, S.E. Parker, Massively parallel three-dimensional toroïdal gyrokinetic flux-tube turbulence simulation, J. Comput. Phys. 161 (2000) 589–604.
[22] C.J. McKinstrie, R.E. Giacone, E.A. Startsev, Accurate formulas for the Landau damping rates of electrostatic waves, Phys. Plasmas 6 (1999) 463–466.
[23] A.J. Klimas, A method for overcoming the velocity space filamentation problem in collisionless plasma model solutions, J. Comput. Phys. 68 (1987) 202–226.
[24] G. Latu, N. Crouseilles, V. Grandgirard, E. Sonnendrücker, Gyrokinetic semi-Lagrangian parallel simulation using a Hybrid OpenMP/MPI Programming, in: Recent Advances in PVM and MPI, LNCS, vol. 4757, Springer, 2007, pp. 356–364.
[25] G. Manfredi, Long time behaviour of strong linear Landau damping, Phys. Rev. Lett. 79 (1997).
[26] T. Nakamura, T. Yabe, Cubic interpolated propagation scheme for solving the hyper-dimensional Vlasov–Poisson equation in phase space, Comput. Phys. Comm. 120 (1999) 122–154.
[27] M. Shoucri, G. Knorr, Numerical integration of the Vlasov equation, J. Comput. Phys. 14 (1974) 84–92.
[28] E. Sonnendrücker, J.J. Barnard, A. Friedman, D.P. Grote, S.M. Lund, Simulation of heavy ion beams with a semi-Lagrangian Vlasov solver, Nucl. Inst. Meth. Phys. Res. 464 (2001) 470–476.
[29] E. Sonnendrücker, F. Filbet, A. Friedman, E. Oudet, J.L. Vay, Vlasov simulation of beams on a moving phase space grid, Comput. Phys. Comm. 164 (2004) 390–395.
[30] E. Sonnendrücker, J. Roche, P. Bertrand, A. Ghizzo, The semi-Lagrangian method for the numerical resolution of the Vlasov equations, J. Comput. Phys. 149 (1999) 201–220.
[31] A. Staniforth, J. Coté, Semi-Lagrangian integration schemes for atmospheric models – a review, Mon. Weather Rev. 119 (1991) 2206–2223.
[32] S.I. Zaki, L.R. Gardner, T.J.M. Boyd, A finite element code for the simulation of one-dimensional Vlasov plasmas I. Theory, J. Comput. Phys. 79 (1988) 184–199.
[33] S.I. Zaki, L.R. Gardner, T.J.M. Boyd, A finite element code for the simulation of one-dimensional Vlasov plasmas II. Applications, J. Comput. Phys. 79 (1988) 200–208.