

Parallelization of semi-Lagrangian Vlasov codes

O. COULAUD,¹ E. SONNENDRÜCKER,¹
E. DILLON,² P. BERTRAND³ and A. GHIZZO³

¹ IECN – Projet Numath, Université Henri Poincaré Nancy 1, 54506
Vandoeuvre-lès-Nancy, France

² INRIA Lorraine, Centre Charles Hermite, 54506 Vandoeuvre-lès-Nancy, France

³ LPMI, Université Henri Poincaré Nancy 1, 54506 Vandoeuvre-lès-Nancy, France

(Received 12 June 1998 and in revised form 12 September 1998)

We describe the parallel implementation of semi-Lagrangian Vlasov solvers, which are an alternative to particle-in-cell (PIC) simulations for the numerical investigation of the behaviour of charged particles in their self-consistent electromagnetic fields. The semi-Lagrangian method, which couples the Lagrangian and Eulerian points of view, is particularly interesting on parallel computers, since the solution is computed on grid points, the number of which remains constant in time on each processor, unlike the number of particles in PIC simulations, and thus greatly simplifies the parallelization process.

1. Introduction

Vlasov–Maxwell equations play a key role in plasma physics, since they describe the collective motion of a collisionless plasma with a wide range of applications.

The numerical resolution of kinetic equations, whose solution depends, in addition to the time, on three space variables and three velocity variables, is performed most of the time using particle-in-cell (PIC) methods, which enable us to get satisfying results with relatively few particles. However, for some applications, in particular when particles in the tail of the distribution play an important physical role or when particle noise is important, semi-Lagrangian methods that compute the solution on a grid may better describe the physics (see Ghizzo et al. 1990, 1993, 1996; Feix et al. 1994). Such methods are all the more interesting when using parallel computers, since unlike PIC methods, they are very scalable owing to their inherent parallelism, as we shall try to show in this paper.

In this paper, we shall first introduce the semi-Lagrangian method and discuss how it can be applied for different kinds of Vlasov equations. We shall then isolate two special cases that need different parallelization methods and illustrate those methods with two examples: the electrostatic two-dimensional (2D) Vlasov–Poisson model and the 2D guiding-centre Vlasov–Poisson model.

2. The semi-Lagrangian method

Let us first recall the principles of the semi-Lagrangian method (see Bermejo 1991; Staniforth et al. 1991) for the Vlasov equations; we refer the reader to Sonnendrücker et al. (1998) for more details.

All the types of Vlasov equations in which we are interested can be written in the following way:

$$\frac{\partial f}{\partial t} + U(X, t) \cdot \nabla_X f = 0, \quad (2.1)$$

where X stands for the phase-space coordinates and U is a divergence-free vector field having up to six components in the full three-dimensional (3D) case. For example, in the case of the 3D non-relativistic Vlasov equation,

$$X = (x, y, z, v_x, v_y, v_z),$$

$$U(X, t) = (v_x, v_y, v_z, E_x + v_y B_z - v_z B_y, E_y + v_z B_x - v_x B_z, E_z + v_x B_y - v_y B_x),$$

with all components of the electric and magnetic field depending on x, y, z and t .

Note that, since U is divergence-free, (2.1) can also be written in conservative form

$$\frac{\partial f}{\partial t} + \nabla_X [U(X, t)f] = 0. \quad (2.2)$$

Let us now introduce the characteristics of (2.1), which are the solutions of the dynamical system

$$\frac{dX}{dt} = U(X(t), t). \quad (2.3)$$

Let us denote by $X(t; x, s)$ the solution at time t whose value is x at time s . Taking $X(t)$ to be a solution of (2.3), we have

$$\frac{d}{dt}(f(X(t), t)) = \frac{\partial f}{\partial t} + \frac{dX}{dt} \cdot \nabla_X f = \frac{\partial f}{\partial t} + U(X(t), t) \cdot \nabla_X f = 0,$$

which means that f is constant along the characteristics. This can also be written as

$$f(X(t; x, s), t) = f(X(s; x, s), s) = f(x, s)$$

for any times t and s and phase-space coordinate x . It is this property that will be used in the semi-Lagrangian method to solve a discrete problem, which is defined by introducing a finite set of mesh points $(x_i)_{i=1, \dots, N}$, which may or may not be equally spaced. Then, given the value of the function f at the mesh points, at any given time step we obtain the new value at mesh point x_i using the fact that

$$f(x_i, t_n + \Delta t) = f(X(t_n - \Delta t; x_i, t_n + \Delta t), t_n - \Delta t).$$

For each mesh point x_i , f is computed in two steps:

1. Find the starting point of the characteristic ending at x_i , i.e. $X(t_n - \Delta t; x_i, t_n + \Delta t)$.
2. Compute $f(X(t_n - \Delta t; x_i, t_n + \Delta t), t_n - \Delta t)$ by interpolation, f being known only at mesh points at time $t_n - \Delta t$.

In order to deal with step 1, we need to introduce a time discretization of (2.3). Since in general, no information on the advection function U is known at any given time, we need to use a two-time-step scheme in order to remain second-order in time. The starting point of the characteristic is obtained, to second-order accuracy, by

$$\frac{x_i - X(t_n - \Delta t)}{2\Delta t} = U(X(t_n), t_n). \quad (2.4)$$

Writing $X(t_n) = \frac{1}{2}[X(t_n + \Delta t) + X(t_n - \Delta t)]$, there exists α_i such that $X(t_n) = x_i - \alpha_i$ and $X(t_n - \Delta t) = x_i - 2\alpha_i$. Then (2.4) becomes

$$\alpha_i = \Delta t U(x_i - \alpha_i, t_n) \quad (2.5)$$

which can be solved iteratively for the unknown α_i by writing

$$\alpha_i^{k+1} = \Delta t U(x_i - \alpha_i^k, t_n),$$

or using a Newton method. Note that U , known only on the mesh, is interpolated linearly at $x_i - \alpha_i^k$. Once α_i is known, $f(x_i - 2\alpha_i)$ is interpolated by a tensor product of cubic B-splines.

The method that we have just introduced deals with the most general case. However, in many cases, the process can be simplified by using an appropriate splitting of the Vlasov equation. The theory of splitting has been well studied for conservation laws, and preserves the second-order accuracy in time when applied properly. However, one has to be careful to perform the splitting such that the resulting equations can each be written in a conservative form corresponding to (2.2). This is only possible if the divergence of each separate term with respect to the corresponding advection variable does vanish; and thus it does not yield simple one-dimensional (1D) equations in all cases. We refer the reader to Sonnendrücker et al. 1998 for more details. On the other hand, even in cases where the splitting might be numerically possible, physical considerations might prescribe not to do it.

The non-relativistic electrostatic Vlasov equation, which in one dimension reads

$$\frac{\partial f}{\partial t} + v \frac{\partial f}{\partial x} + E(x, t) \frac{\partial f}{\partial v} = 0,$$

is an example where the splitting procedure can be applied. Indeed, it can be split into

$$\frac{\partial f}{\partial t} + v \frac{\partial f}{\partial x} = 0 \quad (2.6)$$

and

$$\frac{\partial f}{\partial t} + E(x, t) \frac{\partial f}{\partial v} = 0. \quad (2.7)$$

In this case, we have two 1D equations, the advection function v and E respectively being independent of x and v respectively. Thanks to this property, no iterations are needed in order to solve (2.5), and a one-time-step scheme can be used.

When the equations can be split, all the variables that do not appear in the derivatives, like v in (2.6) or x in (2.7), are really just parameters when solving the corresponding equation. Hence a trivial parallelization can be performed by distributing the computation on the processors according to the values of this parameter. It follows that there are really two distinct parallelization methodologies that need to be followed, depending on whether or not the equations can be split. We are now going to illustrate these methods with the following two examples: the 2D electrostatic Vlasov–Poisson model, where a splitting between space and velocity is performed; and the guiding-centre model, where no splitting can be performed.

3. Description of the models

The first model that we consider is the 2D electrostatic Vlasov equation

$$\frac{\partial f}{\partial t} + \mathbf{v} \cdot \nabla_x f + \mathbf{E}(\mathbf{x}, t) \cdot \nabla_v f = 0,$$

coupled with $\mathbf{E} = -\nabla\phi$, and Poisson's equation

$$\nabla^2\phi = 1 - \int f \, dv.$$

In this case, since \mathbf{v} is independent of \mathbf{x} and $\mathbf{E}(\mathbf{x}, t)$ is independent of \mathbf{v} , the above Vlasov equation can be written in the conservative form

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

and then split between space and velocity coordinates into

$$\frac{\partial f}{\partial t} + \mathbf{v} \cdot \nabla_{\mathbf{x}} f = 0 \quad (3.1)$$

and

$$\frac{\partial f}{\partial t} + \mathbf{E}(\mathbf{x}, t) \cdot \nabla_{\mathbf{v}} f = 0. \quad (3.2)$$

The second model that we consider is the guiding-centre Vlasov–Poisson model, which is an approximation of the full Vlasov equation valid in the presence of a large constant external magnetic field \mathbf{B}_0 . In this case, the average movement of the particles is an $\mathbf{E} \times \mathbf{B}_0$ drift. The equations read

$$\frac{\partial \rho}{\partial t} + \mathbf{v}_D \cdot \nabla_{\mathbf{x}} \rho = 0,$$

where

$$\mathbf{v}_D = \frac{\mathbf{E} \times \mathbf{B}_0}{B_0^2},$$

the electric field being given by Poisson's equation $-\nabla^2\phi = \rho$ with $\mathbf{E} = -\nabla\phi$. Here both components of \mathbf{v}_D depend a priori on the two space variables. Therefore the splitting cannot be justified theoretically, and will not be performed.

4. The specific semi-Lagrangian algorithms

4.1. The semi-Lagrangian method for the electrostatic Vlasov equation

As we saw in the previous section, the equation can be split into two 2D advections, with an advection field independent of the advection variable, namely

$$\frac{\partial f}{\partial t} + \mathbf{v} \cdot \nabla_{\mathbf{x}} f = 0 \quad (4.1)$$

and

$$\frac{\partial f}{\partial t} + \mathbf{E}(\mathbf{x}, t) \cdot \nabla_{\mathbf{v}} f = 0. \quad (4.2)$$

Here we could even split the equation into four 1D advections. However, this would have no influence on the parallelization strategy. Hence we shall not consider this possibility in the sequel.

In this case, the feet of the characteristics can be computed explicitly. The displacement from the mesh points is the same everywhere, namely $\mathbf{v}\Delta t$ for the advection (4.1) over a time step Δt , and $\mathbf{E}\Delta t$ for the advection (4.2) over a time step Δt . Only the second step of the algorithm involves effective computation: the distribution function at the previous time step is interpolated by cubic splines (a 2D tensor product of cubic B-splines in the each case).

4.2. The semi-Lagrangian method for the guiding-centre model

Here, since no splitting can be performed, a full 2D scheme is necessary. The advection function $\mathbf{v}_D(x, t)$ depends on position and time. Therefore we need to use a two-time-step scheme in order to remain second order in time. The algorithm that we described in Sec. 2 does not simplify. Let us give it explicitly for this specific case. The characteristics are the solutions of the differential system

$$\frac{dX}{dt} = v_{Dx}(x, y, t), \tag{4.3a}$$

$$\frac{dY}{dt} = v_{Dy}(x, y, t). \tag{4.3b}$$

Then, applying the algorithm described in Sec. 2, for each mesh point (x_i, y_j) , ρ is computed in two steps:

1. Find the starting point of the characteristic ending at (x_i, y_j) , i.e. $(X(t_n - \Delta t; x_i, y_j, t_n + \Delta t), Y(t_n - \Delta t; x_i, y_j, t_n + \Delta t))$, which is done by computing, for each (i, j) , the displacements $(\alpha_{ij}, \beta_{ij})$ from the mesh point (x_i, y_j) , by solving iteratively the nonlinear system

$$\begin{aligned} \alpha_{ij} &= \Delta t v_{Dx}(x_i - \alpha_{ij}, y_j - \beta_{ij}, t_n), \\ \beta_{ij} &= \Delta t v_{Dy}(x_i - \alpha_{ij}, y_j - \beta_{ij}, t_n). \end{aligned}$$

Concerning the parallelization, once \mathbf{v}_D is known, the treatment of all the mesh points can be performed concurrently. Moreover, the data involved in the computation are the values of \mathbf{v}_D at neighbouring mesh points. So the problem is essentially local, involving only interprocessor communication for mesh points close to a boundary of the decomposition.

2. Compute $\rho(X(t_n - \Delta t; x_i, y_j, t_n + \Delta t), Y(t_n - \Delta t; x_i, y_j, t_n + \Delta t), t_n - \Delta t)$ by a tensor product cubic B-spline interpolation, ρ being known only at mesh points at time $t_n - \Delta t$.

Let us describe here more precisely the tensor-product cubic B-spline interpolation procedure, since in this case it will have a direct influence on the parallelization strategy. The first step consists in computing the coefficients $\eta_{\nu\kappa}$ of the cubic spline interpolation function $s(x, y)$ given by

$$s(x, y) = \sum_{-2 \leq \nu \leq N_x - 1} \left[\sum_{-2 \leq \kappa \leq N_y - 1} \eta_{\nu\kappa} B_{3\nu}(x) B_{3\kappa}(y) \right].$$

The spline s must satisfy the interpolation conditions

$$s(x_i, y_j) = \rho(x_i, y_j, t_n - \Delta t)$$

for $i = 1, \dots, N_x, j = 1, \dots, N_y$, and the two boundary conditions of the function ρ in each direction, which in our case are periodic in the x direction and natural in the y direction.

In order to compute the coefficients $\eta_{\nu\kappa}$, we first solve the N_y one-dimensional interpolation problems

$$s(x, y_j) = \sum_{-2 \leq \nu \leq N_x - 1} \gamma_\nu^j B_{3\nu}(x) \quad \text{for } j = 1, \dots, N_y,$$

each satisfying the N_x interpolation conditions $s(x_i, y_j) = \rho(x_i, y_j, t_n - \Delta t)$ and the periodic boundary conditions, where we denote

$$\gamma_\nu(y) = \sum_{-2 \leq \kappa \leq N_y - 1} \eta_{\nu\kappa} B_{3\kappa}(y)$$

and

$$\gamma_\nu^j = \gamma_\nu(y_j).$$

Using these interpolation and boundary conditions, we are brought to solve N_y linear systems, one for each value of j , involving the same $(N_x + 2)$ -dimensional tridiagonal matrix corresponding to the one-dimensional B-spline interpolation, which gives us the γ_ν^j .

Then we obtain $\eta_{\nu\kappa}$ by solving the $N_x + 2$ interpolation problems

$$\gamma_\nu(y) = \sum_{-2 \leq \kappa \leq N_y - 1} \eta_{\nu\kappa} B_{3\kappa}(y) \quad \text{for } \nu = -2, \dots, N_x - 1,$$

verifying the N_y interpolation conditions $\gamma_\nu(y_j) = \gamma_\nu^j$ and natural boundary conditions. Using these interpolation and boundary conditions, we have to solve $N_x + 2$ linear systems, one for each value of ν , involving the same $(N_y + 2)$ -dimensional tridiagonal matrix corresponding to the one-dimensional B-spline interpolation, in order to obtain the $\eta_{\nu\kappa}$, which are the required coefficients.

Both steps described above involve a set of identical computations, namely tridiagonal solves with different right-hand sides. Hence, taken separately, these steps are perfectly parallel in nature. However, assuming a distributed architecture, there needs to be a data redistribution between the two steps.

Finally, once the B-spline coefficients $\eta_{\nu\kappa}$ for all ν and κ have been computed, the value of ρ at the origin of the characteristics is taken to be the value of the B-spline $s(x_i - 2\alpha_{ij}, y_j - 2\beta_{ij})$. This procedure is essentially local, since it involves only points at the neighbourhood of the point being considered.

5. The field solves

The problems with which we deal involve a doubly periodic domain for the 2D electrostatic model and a domain periodic in one direction and bounded in the other for the guiding-centre model. The algorithms used for the solves are as follows.

For the first geometry (periodic in both directions) a 2D fast Fourier transform (FFT) enables us to compute the electric field components directly.

For the second geometry (periodic in x and bounded in y), we want to compute the electric fields to fourth-order accuracy. This is done using the procedure first introduced by Knorr et al. (1980) which we shall recall here in order to get some insight into the intrinsic parallelism of the method. We first need a fourth-order Poisson solve, which consists of the following steps:

1. Perform a FFT in x , which yields for each discrete mode a 1D problem in the y direction:

$$-\frac{\partial^2}{\partial y^2} \phi_k(y) + k^2 \phi_k = \rho_k.$$

2. Use of a fourth-order scheme for the resulting 1D problem in the y direction:

$$-\frac{12}{(\Delta y)^2}(\phi_{k,j+1} - 2\phi_{k,j} + \phi_{k,j-1}) = (\rho_{k,j+1} + 10\rho_{k,j} + \rho_{k,j-1}) - k^2(\phi_{k,j+1} + 10\phi_{k,j} + \phi_{k,j-1}).$$

Gathering the identical terms, this yields

$$(1 - c_k)\phi_{k,j+1} - (2 + 10c_k)\phi_{k,j} + (1 - c_k)\phi_{k,j-1} = \frac{(\Delta y)^2}{12}(\rho_{k,j+1} + 10\rho_{k,j} + \rho_{k,j-1}),$$

where $c_k = \frac{1}{12}\Delta y^2 k^2$. This holds for $j = 2, \dots, N_y - 1$. In addition, we have $\phi_{k,1} = \phi_{k,N_y} = 0$ for vanishing Dirichlet boundary conditions. This gives us for each mode k a tridiagonal system, which needs to be solved.

3. Perform an inverse FFT in x to obtain the result.

From a computational point of view, the first step consists of N_y simultaneous independent FFT, and the second step consists of the resolution of $\frac{1}{2}N_x + 1$ tridiagonal systems, which can be done concurrently. The third step consists of N_y simultaneous independent inverse FFTs. Hence each of the three steps can be performed completely in parallel; however, steps 1 and 3 involve data stored in the rows of the matrix $(\rho_{i,j})$, whereas step 2 involves data stored in the columns of the matrix $(\rho_{i,j})$. So, assuming a distributed memory architecture, this means that the matrix $(\rho_{i,j})$ needs to be redistributed on the processors between the steps.

Once the potential ϕ is obtained to fourth-order accuracy, the electric field can be computed, still to fourth-order accuracy, by simply using the following fourth-order-accurate 1D scheme for computing a derivative:

$$f'_{i+1} + 4f'_i + f'_{i-1} = \frac{3}{\Delta x}(f_{i+1} - f_{i-1}).$$

Thanks to the periodicity of the problem in the x direction, we can use this scheme in the x direction as $E_x = -\partial_x \phi$. This yields, for $j = 1, \dots, N_y$ and $i = 2, \dots, N_x - 1$,

$$E_{xi+1,j} + 4E_{xi,j} + E_{xi-1,j} = \frac{3}{\Delta x}(\phi_{i+1,j} - \phi_{i-1,j}).$$

Moreover, because of the periodicity, we have

$$E_{x1,j} + 4E_{xN_x-1,j} + E_{xN_x-2,j} = \frac{3}{\Delta x}(\phi_{1,j} - \phi_{N_x-2,j}),$$

$$E_{x2,j} + 4E_{x1,j} + E_{xN_x-1,j} = \frac{3}{\Delta x}(\phi_{2,j} - \phi_{N_x-1,j}).$$

The same scheme can also be used in the y direction in order to compute $E_y = -\partial_y \phi$ away from the boundary.

However, in order to keep fourth-order accuracy, a special treatment is necessary at the boundary: as for the Poisson solver, we can transform the problem in a set of 1D problems using the Fourier transform. Then the following 1D formula is accurate to fourth order:

$$f_{i+1} - f_i = \frac{\Delta x}{2}(f'_{i+1} + f'_i) - \frac{\Delta x^2}{12}(f''_{i+1} - f''_i).$$

Applying this formula for $f = \phi_k$, $f' = -E_{y_k}$ and $f'' = k^2\phi_k - \rho_k$, yields

$$E_{k,1}^y + E_{k,2}^y = - \left[\frac{2}{\Delta y} (1 + c_k)(\phi_{k,2} - \phi_{k,1}) + \frac{\Delta y}{6} (\rho_{k,2} - \rho_{k,1}) \right],$$

$$E_{k,n}^y + E_{k,n-1}^y = - \left[\frac{2}{\Delta y} (1 + c_k)(\phi_{k,n} - \phi_{k,n-1}) + \frac{\Delta y}{6} (\rho_{k,n} - \rho_{k,n-1}) \right].$$

Then, using an inverse Fourier transform,

$$E_{i,1}^y + E_{i,2}^y = - \frac{2}{\Delta y} \{ (\phi_{i,2} - \phi_{i,1}) + \mathcal{F}^{-1} [c_k(\phi_{k,2} - \phi_{k,1})] \} + \frac{\Delta y}{6} (\rho_{i,2} - \rho_{i,1}),$$

$$E_{i,n}^y + E_{i,n-1}^y = - \frac{2}{\Delta y} \{ (\phi_{i,n} - \phi_{i,n-1}) + \mathcal{F}^{-1} [c_k(\phi_{k,n} - \phi_{k,n-1})] \} + \frac{\Delta y}{6} (\rho_{i,n} - \rho_{k,n-1}),$$

where $c_k = \frac{1}{12}k^2(\Delta y)^2$. These two equations complete the system, which remains fourth order.

As for the Poisson solves, there is an inherent parallelism in the computation of the electric field, which consists for both components in the resolution of a set of tridiagonal systems. However, here again the distribution of the data on the processors needs to be different for the computation of E_x and E_y .

6. The parallel algorithms

6.1. The guiding-centre model

Going through the different parts of the algorithm, we can explicitly describe the different kinds of computation that have to be done. These are, for the density advance in time, fixed-point iterations over the displacements from the mesh points, tridiagonal solvers for computing the cubic B-spline coefficients (a whole set in each direction, since we are using a tensor product interpolant), and evaluation of the interpolated spline at the feet of the characteristics. On the other hand, for the field solve the computations to be performed are multiple 1D FFTs and inverse FFT, as well as multiple tridiagonal solvers.

We note that a few stages are completely local: finding the origin of the characteristics, and computing the spline values, which only involves the mesh points in the neighbourhood of the one being computed. And most stages (FFT, tridiagonal solvers) involve whole lines of the domain, but unfortunately not always in the same direction.

Normally, for a 2D problem like ours, a global 2D decomposition is optimal, since it has the smallest interprocessor boundary and involves no data redistribution on the processors. However, tridiagonal solves and FFTs are well known not to have very good scalability when performed on multiple processors. On the other hand, we saw that, for each stage of the algorithm, there is a 1D band decomposition for which there is a natural optimal parallelism. Therefore our final choice was to take several distinct decompositions, even though this would involve global communication. One primary decomposition in 1D bands in the x direction would be used most of the time, the global data being stored according to this decomposition, and several secondary decompositions, not always the same, would be used where required. That is in our specific case: the tridiagonal solvers for Poisson, for E_y and one direction of the spline interpolation.

Using these considerations, we start with a 1D band distribution along the x direction and proceed at each time step through the following stages:

1. Compute the new electric field. For this, we first perform a Poisson solve, which consists of the following:
 - (a) a multiple 1D FFT for ρ on each processor, the data being distributed evenly according to their index j ;
 - (b) a data redistribution – the data are now distributed across the processors according to their mode number;
 - (c) multiple tridiagonal solves on each processor;
 - (d) a data redistribution to go back to the initial distribution according to the index j ;
 - (e) a multiple inverse FFT.

Then we compute the derivatives in each direction of the electric field, for which the steps are as follows:

- (a) multiple tridiagonal solves on each processor, where the data are distributed according to the index j for computing the derivative with respect to x ;
 - (b) a data redistribution, so that the distribution is along the index i ;
 - (c) multiple tridiagonal solves on each processor, where the data are distributed according to the index i for computing the derivative with respect to y ;
 - (d) a data redistribution, to get back to the original distribution for the density advance.
2. Advance the charge density ρ , which is performed as follows:
 - (a) Compute the feet of the characteristics: this needs information from the neighbourhood; hence it involves interprocessor communication for mesh points at the edge of the decomposition.
 - (b) Compute the spline coefficients in the x direction. This involves multiple tridiagonal solves on each processor.
 - (c) Redistribute data so that the distribution is along the index i .
 - (d) Compute the tensor product spline coefficients. This involves multiple tridiagonal solves on each processor, which can be performed locally thanks to the actual data distribution.
 - (e) Redistribute data to go back to the original distribution for the beginning of the next time step.
 - (f) Evaluate splines at the feet of the characteristics: this needs information from the neighbourhood, and thus a slight amount of local interprocessor communication.

6.2. The electrostatic model

Let us now go through the steps of the algorithm for the electrostatic model in the same way. Thanks to the splitting method that we use, the semi-Lagrangian method is applied once for a 2D physical-space advection, with the velocity coordinate \mathbf{v} as a parameter, and once for a 2D velocity-space advection, with the physical space coordinate \mathbf{x} as a parameter. Thus, if the data are distributed in each case according to the parameter, there will be no communication at all – not even the almost local communications that occurred in the previous example. The computations in this case, which are fully local, involve multiple tridiagonal solves for the cubic spline

interpolation, and explicit computations for the spline evaluations. For the field solve, we need a 2D FFT which is also best achieved by multiple 1D FFTs involving data redistribution.

Finally, in this case for each time step, starting with a 1D band distribution along the v_x direction, the algorithm reads as follows:

1. perform a spatial shift over $\frac{1}{2}\Delta t$; that is, apply the simplified semi-Lagrangian algorithm for (3.1) over a time step of $\frac{1}{2}\Delta t$.
2. Redistribute the distribution function in order to get a 1D band distribution along the x direction. Then integrate to obtain ρ .
3. Compute the electrical field:
 - (a) perform a multiple 1D FFT along the y direction for the electric field;
 - (b) redistribute the field data along the y direction;
 - (c) perform a multiple 1D FFT along the x direction for the electric field;
 - (d) redistribute the distribution the electric field in order to get a 1D band distribution along the y direction.
4. redistribute the distribution function in order to get a 1D band distribution along the v_x direction.
5. Perform a velocity shift over Δt ; that is, apply the simplified semi-Lagrangian algorithm for (3.2) over a time step of Δt .
6. Perform a second spatial shift over $\frac{1}{2}\Delta t$.

7. The implementations

7.1. A shared memory implementation of the guiding-centre model

The easiest way to implement the parallel algorithm that we have described for the guiding-centre model is to use the shared distributed memory concept available on Silicon Graphics' Origin 2000. This concept means that, even though the memory of the computer is physically distributed, there exists a software system giving access to the whole memory. When programming on such a platform, one still needs to take care of the data distribution on the processors, but the communications are handled by the system through compiler directives or system calls. Let us now give a few more details.

The parallel implementation uses just one parallel loop over the number of processors which appears after the initialization part of the code and is ended at the end of the computation. This is implemented using the `!$DOACROSS` directive. The arrays through which the communication will be carried are declared as shared at the beginning of this parallel loop; all the others are local to each processor. Then a matrix transpose is implemented by simple array copy:

```
do i=1,n
  a(i,j)=b(j,i)
end do
```

where `b` is a shared array. Moreover, the almost-local communications that appear in the computation of the feet of the characteristics and the spline evaluations are transparent through the use of shared arrays for the drift velocity and the spline coefficients.

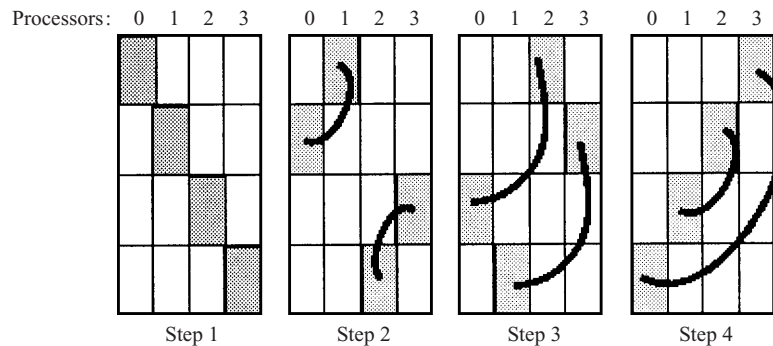


Figure 1. Successive steps of the transposition of a matrix distributed by bands on four processors.

Before each global data redistribution, there is a call to the system routine `mp_barrier` for synchronization.

7.2. A message-passing implementation of the electrostatic model

In this case, we noticed that with our parallel algorithm, the only communications are the row–column exchanges involved in the global data redistribution. No other communication is necessary. The implementation that we used in this case was the MPI message-passing library.

The most obvious approach for implementing the data redistribution consists in having the processors send all at once the rows of the arrays to the processors that need them for further computations, and then have them all receive their data in the appropriate columns. This worked fine on Cray’s T3E, but overflowed the Origin 2000’s message-passing buffers. Indeed, this communication scheme requires each processor to bufferize the rows to be sent. As a consequence, the number of required MPI buffers increases very rapidly with the number of processors, leading to deadlock on the Origin 2000: each processor blocks because all of its local buffers are full, but still need to execute ‘send’ actions before actually starting to receive columns – freeing by the way other processors’ buffers.

For this reason, a more refined algorithm was needed. The main idea that guided the design of the new algorithm for this row–column exchange was to avoid buffering most of the time. To do this, we chose to refine the algorithm into a sequence of steps (not only ‘bulk send followed by bulk receive’), where each step would only require one MPI buffer, whatever the total number of processors. Thanks to this, it is possible to reduce the memory needs, and, most importantly, to avoid deadlock, even if the architecture has only a few MPI buffers available.

The algorithm designed for this row–column exchange is to be applied recursively by each processor on their local band. Its main advantage is that it ensures that, at each step of the global exchange, each processor will only perform one point-to-point communication. Consequently, all congestions and bufferizations can be avoided during communications.

Let us see how the recursive algorithm works on an example. For this example, we shall distribute the global matrix by bands on four processors. Figure 1 shows the successive steps performed in parallel on the matrix.

At each step, the global matrix is virtually split into 16 blocks, so that each processor will only have one partner to which to send its own data.

Table 1. Speed-up for the whole code of the guiding-centre model for a 1024×1024 grid.

Number of processors, p	1	2	4	8	16
Time (s)	608	324	163	85	47
Speed-up $S(p)$	1	1.88	3.73	7.15	12.94
Efficacy $E(p)$	1	0.94	0.93	0.89	0.80

The first step does not include any communication, since this step only involves the data stored on the diagonal of the global matrix, which means that this block does not need to be moved.

The three next steps then consist in point-to-point exchanges between processors. It might be noticed that, at each step, each processor only has one send–receive operation to complete, so that it only needs one buffer for the exchange.

Since our 2D Vlasov–Poisson application also required the use of 2D FFT over the global matrix, we chose to extend this row–column global exchange to a true matrix transposition in order to use it for a 2D parallel FFT. Indeed, the extension to the previous algorithm was to actually transpose the data during the exchange. This transposition was performed, on the fly, using MPI-derived datatype.

To perform the parallel 2D FFT over a matrix distributed in band-on processors, we can use the following algorithm :

1. Each processor computes a multiple real-to-complex 1D FFT on its local columns.
2. The result of this first multiple complex-to-real 1D FFT is globally transposed, to swap directions.
3. Each processor computes a new complex-to-real 1D-FFT on these new columns.
4. The result is transposed once again, to get each direction in the right way.

Multiple 1D FFTs are available from the *sycl* library on CRAY and SGI platforms. We implemented the row–column exchange algorithm with ‘on the fly’ transposition thanks to MPI derived data types, to get a true transposition algorithm.

8. The parallel speed-ups

To study the performance of the parallel algorithms, we introduce the speed-up $S(p)$ and the efficacy $E(p)$, defined as follows:

$$S(p) = \frac{\text{time on one processor}}{\text{time on } p \text{ processors}}, \quad E(p) = \frac{S(p)}{p}.$$

Owing to our parallel algorithms, the speed-up of the whole code is driven by the speed-up of data transposition to exchange the data distributions of the density function f . The performances of the parallel algorithms are summarized in Tables 1 and 2. For the guiding-centre model, $S(p)$ is given for the whole code, whereas for the electrostatic code, $S(p)$ is only for the transposition, the rest of the algorithm being perfectly parallel.

As we have shown in this paper, the semi-Lagrangian algorithm can be applied such that most steps are perfectly parallel. In fact, in the case of the electrostatic

Table 2. Speed-up of a 1024×1024 complex matrix transposition using a message-passing algorithm based on MPI.

Number of processors p	8	16	32
Speed-up $S(p)$	7.53	11.1	16.75
Efficacy $E(p)$	0.94	0.69	0.52

Vlasov–Poisson model, where the feet of the characteristics are known, all stages of the computation are perfectly parallel, provided that data redistributions are performed at some points. Moreover, we have found a way to perform the data redistribution in an efficient manner that enables us to have good parallel efficiency.

To explain the loss of efficacy in Table 2, we model the time of one communication by $t_{\text{com}} = \tau + LT_B$, where τ is the latency, L the length of the message, and T_B the bandwidth. With this definition, the ideal time spent in communication for our algorithm to exchange data is $t_{\text{rout}} = (p - 1)t_{\text{com}}$, which can be rewritten as $t_{\text{rout}} = t_s + L_m T_B$, where $t_s = O(p)\tau$ and $L_m = O(N^2/p)$. For the configuration of our Origin 2000, the measured τ (by a ping-pong) is $18 \mu\text{s}$. Since τ is important here, for a large number of processors the dominant term in t_{rout} is t_s , which explains the loss of speed-up in the transposition. Following some recent improvement by SGI on both MPI and the system, the latency is now $4 \mu\text{s}$.

In the shared memory model and the distributed one, loss of speed-up is due to migration of data between processes. In the shared memory model, the hardware is used directly (via the compiler) to move data; whereas in the MPI code, data are moved explicitly by `mpi-send` and `mpi-receive` functions.

9. Conclusion

Parallelization of Vlasov codes is an essential task, since the numerical solution of Vlasov equations is very time- and memory-consuming. The two special models discussed above have been chosen to reflect the specific difficulties associated with the parallelization process of more general Vlasov codes. The performances obtained in these examples are very satisfying. Let us also mention that the examples that we have developed here were essentially classical problems picked to test the numerical algorithm and benchmark the codes. More complex and realistic problems related to laser–plasma interaction are addressed in Bégué et al. 1998.

Building on the methods that we have introduced here, we are now ready to develop a Fortran 90 module library implementing the different advection types that are needed. Assembling these modules will then enable us to treat many problems occurring in plasma physics using the semi-Lagrangian methodology.

References

- Bégué, M. L., Ghizzo, A., Bertrand, P., Sonnendrücker, E. and Coulaud, O. 1998 Two-dimensional semi-Lagrangian Vlasov simulations of laser–plasma interaction in the relativistic regime. Submitted to *J. Plasma Phys.*
- Bermejo, R. 1991 Analysis of an algorithm for the Galerkin-characteristic method. *Numer. Math.* **60**, 163–194.

- Feix, M. R., Bertrand, P. and Ghizzo, A. 1994 Eulerian codes for the Vlasov equation. In: *Kinetic Theory and Computing* (ed. B. Perthame), pp. 45–81. Advances in Mathematics for Applied Sciences, Vol. 22. World Scientific, Singapore.
- Ghizzo, A., Bertrand, P., Shoucri, M., Johnston, T. W., Filjakow, E. and Feix, M.R. 1990 A Vlasov code for the numerical simulation of stimulated Raman scattering. *J. Comput. Phys.* **90**, 431–457.
- Ghizzo, A., Bertrand, P., Shoucri, M., Filjakow, E. and Feix, M. R. 1993 An Eulerian code for the study of the drift-kinetic Vlasov equation. *J. Comput. Phys.* **108**, 105–121.
- Ghizzo, A., Bertrand, P., Begue, M. L., Johnston, T. W. and Shoucri, M. 1996 A Hilbert–Vlasov code for the study of high frequency plasma beatwave accelerator. *IEEE Trans. Plasma Sci.* **24**, 370–378.
- Knorr, G., Joyce, G. and Marcus, A. J. 1980 Fourth-order Poisson solver for the simulation of bounded plasmas. *J. Comput. Phys.* **38**, 227–236.
- Sonnendrücker, E., Roche, J., Bertrand, P. and Ghizzo, A. 1999 The semi-Lagrangian method for the numerical resolution of Vlasov equations. *J. Comput. Phys.* **149**, 201–220.
- Staniforth, A. and Côté, J. 1991 Semi-Lagrangian integration schemes for atmospheric model – a review. *Mon. Weather Rev.* **119**, 2206–2222.