Gysela 5D, a gyrokinetic semilagrangian parallel code

Guillaume Latu, Nicolas Crouseilles, Virginie Grandgirard and Eric Sonnendrucker
latu@labri.fr

Abstract

We are interested in solving the Vlasov equation used to describe collective effects in plasmas. This non-linear partial differential equation coupled with a field equation describes the time evolution of the particle distribution in phase space. In this paper, we focuses on a recently developed 5D parallel numerical application dedicated to gyrokinetic simulation of Takamak systems. We got a multi-level parallelized application that achieves a good scalability up to hundreds of processors on a cluster of SMP nodes.

1 INTRODUCTION

In the quest for a new source of energy, understanding plasma behavior is one of the most challenging problems to overcome. Plasma are obtained in different facilities, in particular in tokamak reactors. A kinetic description could be used to model phenomena that take place inside the tokamaks. These plasma are well described by the Vlasov equation, coupled with quasi-neutralilty equation[?]. Vlasov equation characterizes the evolution of particle distribution in time and space according to the electro-magnetic fields. The distribution function $f(\overrightarrow{x}, \overrightarrow{v}, t)$ represents the particle density at a time t and $(\overrightarrow{x}, \overrightarrow{v})$ in phase space. To describe the most general case, one need $(\overrightarrow{x}, \overrightarrow{v}) \in \mathbb{R}^d \times \mathbb{R}^d$ with $d = 3$. Finding an approximation of this non-linear partial differential Vlasov equation enables the simulation of new accelerator and tokamak designs to validate them before building the devices. For a more fundamental purpose, finding such an approximate solution makes it possible for physicists to represent the behavior of different physical parameter sets.

The numerical resolution of Vlasov type equations, the solution of which depends on 6 variables plus time, is performed most of the time using particle methods (Particle In Cell methods) where the plasma is approached by a finite number of macro-particles [BL91]. Even if these methods give satisfying results with relatively few particles, for some applications however, it is well known that the numerical noise inherent to the particle methods becomes too significant. Consequently, methods which discretize the Vlasov equation on a phase space grid have been proposed (see [SRBG99, FS06]) for plasma and beam physics applications. Among these Eulerian methods, we are interested in the implementation of the semi-Lagrangian method; it consists in updating the values of the distribution function at the nodes of the grid by following the characteristics ending at these nodes backward and interpolating the value at the bottom of the characteristics from the known values at the previous time step. The computation is done by integrating the characteristic curves backward at each time step and interpolating the value at the feet of the characteristics thanks to interpolation techniques (Lagrange or Hermite or cubic splines for example). In fact, a time splitting procedure [CK76] is added. One simulation time step is split into several successive substeps: displacements in seperate dimensions. With this scheme, the interpolations are mainly performed in one or two dimensions instead of interpolations in space having more than 3 dimensions. The computation cost is then drastically reduced.

The present work describes a parallel implementation of the semi-Lagrangian method by using a cubic splines interpolation method and a parallelization of a quasi neutrality solver. The model underlying the simuator uses a rewritten Vlasov equation that uses a distribution function in 5D plus time [?]. In this context of a 5D model, coupled with a splitting scheme, the cubic spline interpolation seems to be a good compromise between accuracy and simplicity. Nevertheless, the standard method does not provide the locality of the reconstruction since all the values of the distribution function for a given 2D section are necessary to reconstruct f in each cell of this 2D section. To overcome this problem of strong dependencies, we propose a solution that allows us to interpolate on quasi independant small 2D patches. Thus, we decompose 2D domains into patches, each patch being devoted to a set of processor. One patch computes its own local cubic spline coefficients by solving reduced linear systems. Some adapted boundary conditions are imposed at the interface of the patches to obtain a $\mathcal{C}^1$ global resolution which is close to the sequential resolution that uses classical global splines. Moreover, thanks to a restrictive condition on the time step, the inter-processor communications are only done between logically adjacent set of processors, which enables us to obtain competitive results from a scalability point of view.

The major drawback of Eulerian methods using uniform meshes is that memory consumption and computation cost increase drastically when the number of dimensions d, and grid size increase. That explains why parallelism is required. A previous parallel simulator were designed [?]. Nevertheless, this first approach implied several global transpositions of the distribution function (the main 5D data) per time step because of classical splines. The amount of communication induced were prohibitive from 16 processors to a greater number of processors. This early solution does not scale well and then were upgraded by the solution exposed in this paper.

The present work focuses on the parallelization of a realistic semi-lagrangian code which considers a full tokamak system. The simulator is scalable and is able to use hundreds of processors efficiently. The section 2 focuses on the numerical scheme. Section 3 describes sequential algorithms of the simulator, and their associated algorithmic costs. Section 4 depicts the parallelization. Section 5 deals with performance analysis and is followed by a conclusion. These current researches are performed in an interdisciplinary approach together with Fusion department of CEA (DRFC) and with the INRIA CALVI project, partly located at Strasbourg 1 University, France.

2 NUMERICAL SCHEME

The 5D model considers the following variables to discretize the distribution function : r and θ are the polar coordinates in the shortest cross-section of the torus (called poloïdal section), φ refers to the angle in the largest cross-section of the torus, $v_\parallel$ is the velocity along the φ dimension, μ the magnetic momentum acts as a parameter that represents the poloïdal component of the velocity. The time evolution of the guiding-center 5D gyroaveraged distribution function $\bar{f}_t(r,\theta,\varphi,v_\parallel,\mu)$ is governed by the gyrokinetic Vlasov equation :

$$\frac{\partial \bar{f}}{\partial t} + \frac{\mathrm{d}r}{\mathrm{d}t}\frac{\partial \bar{f}}{\partial r} + \frac{\mathrm{d}\theta}{\mathrm{d}t}\frac{\partial \bar{f}}{\partial \theta} + \frac{\mathrm{d}\varphi}{\mathrm{d}t}\frac{\partial \bar{f}}{\partial \varphi} + \frac{\mathrm{d}v_\parallel}{\mathrm{d}t}\frac{\partial \bar{f}}{\partial v_\parallel} = 0 \qquad (1)$$

The electric quasineutrality provides the self-consistency of the problem, coupling the electric potential $\Phi_t(r,\theta,\varphi)$ (which plays a major role in the $\frac{\mathrm{d}v_\parallel}{\mathrm{d}t}\frac{\partial \bar{f}}{\partial v_\parallel}$ term) to $\bar{f}$. The Φ function is found in solving the equation (with $\nabla_\perp = (\partial_r, \frac{1}{r}\partial_\theta)$).

$$-\frac{1}{n_0(r)}\nabla_\perp \cdot \left[\frac{n_0(r)}{B_0\,\omega_c}\nabla_\perp \Phi\right] + \frac{e}{T_e(r)}\left[\Phi - \langle\Phi\rangle\right] = \bar{A}(r,\theta,\varphi) \qquad (2)$$

with

$$\begin{aligned}&\bar{A}(r,\theta,\varphi) = \\ &\tfrac{2\pi}{m_i\, n_0(r)} \int d\mu B(r,\theta) J_0(k_\perp \rho_c) \int dv_\parallel (\bar{f} - \bar{f}_{eq})\end{aligned} \qquad (3)$$

The larmor radius corresponds to the notation ρ_c. The ion charge is $e_i = Z_i\, e$, and the ion mass m_i. The magnetic configuration is a circular concentric tokamak configuration with B_0 the value of magnetic field at $r =$??. The time is normalized to the inverse of the ion cyclotronic frequency $\omega_c = e_i\, B_0/m_i$.

Equations (1) and (2) are solved successively at each time step thanks to an explicit method. One deduces Φ_t from integral computations on $\bar{f}_t$ followed by the solving of equation (2). Then the electrostatic field $E_t(r,\theta,\phi) = -\nabla\Phi_t(r,\theta,\varphi)$ is found. For the resolution of equation (1), we use a backward semi-lagrangian method to compute $\bar{f}_{t+dt}$ from $\bar{f}_t$ and E_t (in fact we use too the values of f_{t-dt} in order to get a time scheme of second order). The principle of the semi-lagrangian method is to compute the value of the distribution function $\bar{f}_{t+dt}$ on a grid of the phase space using the property that $\bar{f}$ is constant along particular phase space curves called characteristics. So, the computation consist in computing displacements of grid points of $\bar{f}_{t+dt}$ thanks to the characteristics, followed by an interpolation on the grid $\bar{f}_t$ (or $\bar{f}_{t-dt}$).

3 SEQUENTIAL ANALYSIS

3.1 Global algorithm

The Vlasov equation (1) is solved by splitting it into the three advection equations:

$$\begin{aligned}\partial_t \bar{f} + \overrightarrow{v_{GC}} \cdot \overrightarrow{\nabla_\perp} \bar{f} &= 0, & &(\hat{r\theta} \ \text{ operator})\\ \partial_t \bar{f} + v_\parallel \partial_\varphi \bar{f} &= 0, & &(\hat{\varphi} \ \text{ operator})\\ \partial_t \bar{f} + \dot{v_\parallel} \partial_{v_\parallel} \bar{f} &= 0. & &(\hat{v_\parallel} \ \text{ operator})\end{aligned}$$

Each advection consists in applying a shift operator. A splitting of Strang [?] is employed to keep a scheme of second order accuracy. The sequence we choose is $(\hat{v_\parallel}/2, \hat{\varphi}/2, \hat{r\theta}, \hat{\varphi}/2, \hat{v_\parallel}/2)$, where the factor 1/2 corresponds to a shift over a reduced time step $dt/2$.

Algorithm 1: One time step in GYSELA code

```
   // Vlasov solver
   Input  : $f_{t-dt}$ and $\Phi_t$
   Output: $f_{t+dt}$
 1 1D splitting, operator $\frac{\hat{v_\parallel}}{2}$ on $f_{t-dt}$ (using $\Phi_t$)
 2 1D splitting, operator $\frac{\hat{\varphi}}{2}$
 3 2D splitting, operator $\hat{r\theta}$ (using $\Phi_t$)
 4 1D splitting, operator $\frac{\hat{\varphi}}{2}$
 5 1D splitting, operator $\frac{\hat{v_\parallel}}{2}$ (using $\Phi_t$)
   // Field solver
   Input  : $f_{t+dt}$
   Output: $\Phi_{t+dt}$
 6 Compute $A_{t+dt}$, integrals, gyroaverage on $f_{t+dt}$
 7 Compute $\Phi_{t+dt}$, with systems inversion on $A_{t+dt}$
```

Algorithm 1 focuses on one time step of the simulator. At time step t, we will present key points of the different computations and their associated complexities.
The algorithm manipulates two types of data structures: the 5D data f_{t-dt}, f_t, f_{t+dt}, and the 3D data Φ and A. The sizes of those structures are parameterized by the discretization along the different dimensions. Let $\#_r$, $\#_\theta$, $\#_\varphi$, $\#_{v_\parallel}$, $\#_\mu$ be respectively the number of points in each dimensions r, θ, φ, $v_\parallel$, μ. The size of 5D data are $(\#_r\,\#_\theta\,\#_\varphi\,\#_{v_\parallel}\,\#_\mu)$, and for 3D data we have a size of $(\#_r\,\#_\theta\,\#_\varphi)$.
The Vlasov solver is composed of 5 splitting substeps. A substep requires the computation of the shift of each grid point, and then an interpolation using f values. The algorithmic complexity of each substep is in $\Theta(\#_r\,\#_\theta\,\#_\varphi\,\#_{v_\parallel}\,\#_\mu)$. At the line 5 of the algorithm, a traversal of data f_{t+dt} is needed and the complexity is too in $\Theta(\#_r\,\#_\theta\,\#_\varphi\,\#_{v_\parallel}\,\#_\mu)$. The computation at line 6 implies a cost of $\Theta(\#_r\,\#_\theta\,\#_\varphi\, log(\#_\theta)\, log(\#_\varphi))$ because 2D Fast Fourier Transforms are included in the solver.
All parts of the simulator consume significant processor time. Even if the Vlasov solver concentrates the major part of execution time, we have to consider the parallelization of every part to get an eventually scalable program. Concerning the Vlasov solver, a domain decomposition of the phase space and consequently of the 5D data structures will provide independent computation.

4 PARALLEL ALGORITHM

4.1 Domain decomposition

Concerning the Vlasov solver, the variable μ acts as a parameter. Then we give the responsability of each value of μ to a given set of processors. Within a set, a 2D domain decomposition allows us to attribute to each processor a subdomain in (r, θ) dimensions. For a given local (μ, r, θ) tuple, a processor stores all values $\varphi = *$ and $v_\parallel = *$. This data distribution leads to a straightforward parallelization of all parts of the Vlasov solver, excluding the $\hat{r\theta}$ operator part. We will see in the sequel of the paper how to overcome the problem raised by this operator. The algorithm 2 introduces the computation distribution. In this algorithm, communications between processors are only required at lines 13, 25 and 26.

Algorithm 2: One time step in parallel GYSELA

```
    // Vlasov solver
 1  for $\mu, r, \theta$ in local subdomain do in parallel
 2  |   forall $\varphi, v_\parallel$ do
 3  |   |   1D splitting, operator $\frac{\hat{v_\parallel}}{2}$
 4  |   end
 5  end
 6  for $\mu, r, \theta$ in local subdomain do in parallel
 7  |   forall $\varphi, v_\parallel$ do
 8  |   |   1D splitting, operator $\frac{\hat{\varphi}}{2}$
 9  |   end
10  end
11  for $\mu, r, \theta$ in local subdomain do in parallel
12  |   forall $\varphi, v_\parallel$ do
13  |   |   2D splitting, parallel operator $\hat{r\theta}$
14  |   end
15  end
16  for $\mu, r, \theta$ in local subdomain do in parallel
17  |   forall $\varphi, v_\parallel$ do
18  |   |   1D splitting, operator $\frac{\hat{\varphi}}{2}$
19  |   end
20  end
21  for $\mu, r, \theta$ in local subdomain do in parallel
22  |   forall $\varphi, v_\parallel$ do
23  |   |   1D splitting, operator $\frac{\hat{v_\parallel}}{2}$
24  |   end
25  end
    // Field solver
26  Compute in parallel and broadcast $A_{t+dt}$
27  Compute in parallel and broadcast $\Phi_{t+dt}$
```

4.2 Local spline interpolation

In this section, we present our interpolation technique based on a cubic spline method described in one dimension [CLS06]. With a 2D tensor product of this spline method, interpolations on a 2D subdomain is achievable. In order to apply the $\hat{r\theta}$ operator, we use practically this 2D extension. Nevertheless, we explain here, only the 1D case to simplify the explanations.

Let us consider a function f which is defined on a global domain $[x_{\min}, x_{\mathrm{Max}}] \subset \mathbb{R}$. This domain is decomposed on several sub-domains called generically $[x_{m_p}, x_{M_p-1}]$; each sub-domain will be devoted to the processor p. In the following, we will use the notation $x_i = x_{m_p} + ih$, where h is the cell size: $h = (x_{M_p} - x_{m_p})/K$ and K the number of cells on a sub-domain ($K \in \mathbb{N}$).

Let us now restrict the study of $f : x \mapsto f(x)$ on the interval $[x_{m_p}, x_{M_p}]$ with $M_p = m_p + K$. The projection s of f onto the cubic spline basis reads

$$f(x) \simeq s(x) = \sum_{\nu=-1}^{K+1} \eta_\nu B_\nu(x),$$

where B_ν is the cubic B-spline (see [SRBG99]). The interpolating spline s is uniquely determined by $(K+1)$ interpolating conditions

$$f(x_i) = s(x_i), \quad \forall i = m_p, ..., M_p, \tag{4}$$

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

$$f'(x_{m_p}) \simeq s'(x_{m_p}), \quad f'(x_{M_p}) \simeq s'(x_{M_p}). \tag{5}$$

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 (4) yields

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

On the other hand, we have $B'_{i\pm1}(x_i) = \pm 1/(2h)$, and $B'(x_i) = 0$. Thus the Hermite boundary conditions (5) become

$$\begin{aligned} f'(x_{m_p}) &\simeq s'(x_{m_p}) = -\frac{1}{2h}\ \eta_{m_p-1} + \frac{1}{2h}\ \eta_{m_p+1}, \\ f'(x_{M_p}) &\simeq s'(x_{M_p}) = -\frac{1}{2h}\ \eta_{M_p-1} + \frac{1}{2h}\ \eta_{M_p+1}. \end{aligned}$$

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

$$F = \left[f'(x_{m_p}), f(x_{m_p}), ..., f(x_{M_p}), f'(x_{M_p})\right]^T,$$

and A denotes the following *fixed* matrix

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

A classical LU algorithm is used to solve the linear system.

Approximation of the interface derivatives

In order to get accurate numerical simulations (in a sense that we recover in the best possible way the sequential results), one has to take care of the approximation of the derivatives at the interface of the sub-domains. Various approximations have been implemented but in order to recover the approximation of these interface derivatives obtained by a classical global cubic splines interpolation, we derive a new formula to evaluate $f'(x_{m_p})$ and $f'(x_{M_p})$. Accurate numerical results are obtained with:

$$\begin{aligned} f'_{left}(x_i) &= \textstyle\sum_{j=1}^{j=10} \tilde{\gamma}_j^+ f_{i+j}, \\ f'_{right}(x_i) &= \textstyle\sum_{j=-10}^{j=-1} \tilde{\gamma}_j^- f_{i+j}, \\ f'(x_i) &= f'_{left}(x_i) + f'_{right}(x_i). \end{aligned} \tag{6}$$

We refer the reader to [CLS06] for the details of the obtention of approximation (6) and values of coefficients γ_j^- and γ_j^+.

Properties

In our simulator we expect to interpolate on the interval $[x_{m_p-1}, x_{M_p}]$ instead of $[x_{m_p}, x_{M_p}]$. In order to extend the interpolation capability on one processor, we compute an extra η_{m_p-2} coefficient . We pose the property $m_{p_i} = M_{p_j}$ for p_i and p_j two adjacent processors that shares the grid point $x_{m_{p_i}}$. Finally, with these modifications, each processor has the responsability to modify the values of grid points in the interval $[x_{m_p}, x_{M_p-1}]$, and have the capability to interpolate onto the extended interval $[x_{m_p-1}, x_{M_p}]$. In the 2D splitting phase, where the local splines are used, it means that the shift of one single grid point on the border of a subdomain must not exceed the elementary cell width. this constraint is the main drawback of the method, it is not possible to consider big shift (in r or θ) during a single time step in the 2D splitting phase.

The computation of the η_{m_p-2} coefficient implies a modification of the right hand side term F. We choose to add an term $f(x_{m_p-1})$ to F in order to deduce the

η_{m_p-2} value.

Communication pattern

In the parallel implementation of the local spline method, communications are needed between adjacent processors to build the right hand side term F. On a local processor, the known values are $(f(x_i))_{i\in[x_{m_p},x_{M_p-1}]}$. For processors located at the borders of the global domain ($x_{\min} = x_{m_p}$ or $x_{\text{Max}} = x_{M_p}$), boundary conditions (compact or periodic) are considered to retrieve the values of f needed outside of the domain. Herefter, we enumerate the data that lacks on the local processor to get F (excluding the specific problems that arose at the global domain boundaries):

1. Values of $f(x_{m_p-1})$ and $f'_{left}(m_p)$ are received from a first neighboring processor.

2. Values of $f(x_{M_p})$ and $f'_{right}(M_p)$ are received from a second neighboring processor.

3. The quantities $f'_{right}(m_p)$ and $f'_{left}(M_p)$ are computed on the local processor.

Concerning the item 3, we choose practically a large enough K to have only local calculations to compute $f'_{right}(m_p)$ and $f'_{left}(M_p)$. Experimentally, we have determined a lower bound on K ($K_{\min} = 32$), that leads to a relatively small overhead and has good numerical stability for the local spline interpolation.
In the case of a 2D interpolation (like in the 2D splitting phase), the F term is a matrix instead of a vector. The assembly of F requires communications with the 8 neighboring processors (instead of only 2 processors in the one dimensional case). On one processor and for a 2D patch of size $K_1 \times K_2$, the number of double precision real numbers to receive is $4\,(K_1 + K_2 + 4)$. This amount of communication could be compared to the interpolation cost of the $K_1 \times K_2$ points for the patch, which is $\Theta(K_1\,K_2)$.

Limitation

Numerical experiments with the local spline method for the 2D splitting on physical test cases have shown a bottleneck. The shifts in direction θ are often too large and above the limit we fixed (the width of one cell). It were not feasable to keep this configuration, so we were compelled to remove completely the θ parallelization in the algorithm (2).

4.3 Field solver

The quasi-neutrality solver includes three computation parts. First, integrals are computed to get $\int(\bar{f} - \bar{f}_{eq})\,dv_{\|}$. Second, 1D discrete fourier transforms in φ of the resulting function are performed (in the equation (3)). Tridiagonal systems are solved to apply $J_0(k_\perp \rho_c)$ operator. Then, we get a gyroaveraged quantity $\bar{A}$ after an other integration in $d\mu$.

Third, the function Φ is found in computing 2D discrete fourier transforms in (θ,φ) of $\bar{A}$, followed by the solving of tridiagonal systems. Thus, the equation (2) is finally solved.

Hereafter, an algorithm that proposes a simple formulation for parallelizing the first and second computation parts corresponding to the $\bar{A}$ calculation. The final part, that performs the Φ computation, is redondant on each processor.

Algorithm 3: Partial parallelization of QN solver

```
   Input  : $f(r = block, \theta = block, \varphi = *, v_{\|} = *, \mu)$
   Output: $\Phi(r = *, \theta = *, \varphi = *)$
   // Part 1
1  Integration in $dv_{\|}$ of $f$ (parallel in $r, \theta, \mu$)
2  **Send** local data $\int f\, dv_{\|}(r = block, \theta = block, \varphi = *, \mu)$
3  **Redistribution & Synchronization**
4  **Receive** block $\int f\, dv_{\|}(r = *, \theta = *, \varphi = block, \mu = *)$
   // Part 2
5  Application of operator $J_0$ (parallel in $\varphi$).
6  Integrals in $d\mu$ to get $\bar{A}$ (parallel in $\varphi$).
7  **Send** block $\bar{A}(r = *, \theta = *, \varphi = block)$
8  **Broadcast of $\bar{A}$ / Synchronization**
9  **Receive** global data $\bar{A}(r = *, \theta = *, \varphi = *)$
   // Part 3
10 2D FFT on dimensions $(\theta,\varphi)$ (not parallel),
11 Solving quasi neutrality equation (not parallel),
12 Inverse FFT 2D on dim. $(\theta,\varphi)$ (not parallel)
```

This relatively simple parallelization allows us to keep the same global scheme to compute the 3D structure Φ as in the sequential previous version. The first and second parts of the algorithm lead to a good load balancing. The bad point is the redistribution of data needed line 3 in order to apply afterwards the J_0 operator. This type of communication could not be avoided, becauseforf the numerical method used to apply J_0, we expect data covering global domain in variables r and θ.

The global algorithm has the constraint that each processor must know the data Φ to perform the splittings in the Vlasov solver. So, the big amount of communications, caused by the broadcast of line 8, would be difficult to remove. Another problem raised by this procedure is the redondant computation in part 2 on all processors.

5 PERFORMANCE ANALYSIS

5.1 Efficiency of the parallelization

Numerical experiments were performed on a cluster of IBM 16-core nodes located at Bordeaux, France. Each node hosts Power5 processors and offers 27GB of shared memory. Performances for one of the smallest 5D test case are presented in table 1 ($\#_r = 128, \#_\theta = 128, \#_\varphi = 32, \#_{v_\|} = 32, \#_\mu{=}16$). The number of μ values is 16. For this configuration, our parallel program uses 16 processors at the minimum. So, the relative efficiency shown in table 1 considers as a reference the execution times on 16 processors of a single 16-way node. Let us recall that we are limited to 2D patch of width $K_{\min} = 32$ at the minimum (see subsection 4.2) and we do not have parallelization on variable θ; so, the (r, θ) domain could be decomposed up to only $proc_r = 4$ subdomains.

The 1D splittings are perfectly parallel and scalable, because no overhead in computation nor in communication is needed. However, the 2D splitting requires a communication step to transmit boundary coefficient and derivatives. Furthermore, the 2D interpolation on small patches induces a computation overhead in comparison to the global spline method. These two facts explain why the efficiency decays whenever $proc_r$ increases. About the field solver, the part 1 is very scalable because computation is equally distributed onto processors. The part 2 is scalable up to 32 processors only, because the parallelization is on variable φ and this test case consider few values for φ: $\#_\varphi = 32$. Concerning the part 3 which is redondantly computed in all processes, there is no scalability to expect. The communication overhead of the field solver is not predominant compared to computation times of the field solver. But, it would not be useful to find a better parallelization of part 3, if it induces a significant increase of communications.

5.2 Hybrid approach

The designed parallel simulator achieves good performances. Nevertheless, the limitation on the maximum number of processors that can be used, requires that we investigate other possible levels of parallelism. A refinement of the MPI parallelization would require a fair amount of code restructuring and would imply new communication schemes. However, the MPI and OpenMP programming models can be combined into a hybrid paradigm to exploit levels of parallelism at a finer grain, without heavy code manipulation. The hybrid approach is suitable for clusters of SMP nodes where MPI provides communication capability across nodes and OpenMP exploits loop level parallelism within a node.

We add several parallels loop in all parts of the algorithm 2. A parallelization in variable φ is adequate for the 1D splitting in $v_\|$, the 2D splitting in (r, θ) and in the first two parts of the Field solver. A parallel loop in θ allows a simple formulation in the 1D splitting in φ. Different parallel loops are successively used in the third part of the Field solver to get a qualitative OpenMP code.

In order to measure the scalability of the new hybrid simulator on a medium-sized parallel machine, we presents hereafter a 4D test case with $\#_\mu = 1$. The variable μ corresponds to the coarser level of parallelism. So, if we imagine running the same test case in a 5D configuration (with $\#_\mu = 16$) on 16 times more processors, we should observe very similar scalability performances.

In table 2, a 4D test case illustrates the competitive performance of the hybrid approach ($\#_r 256, \#_\theta = 256, \#_\varphi = 128, \#_{v_\|} = 64, \#_\mu{=}1$). Each part shows a good speedup factor, excluding the only one that does not benefits from OpenMP : the communication phases of the Field solver.

6 CONCLUSION

We described the parallelization of a numerical simulator that solves a 5D Vlasov system. Multiple levels of parallelism are used by combining message passing and OpenMP parallelization. Amost every steps of the original algorithm benefits from this hybrid approach. The scalability is really impressive on a cluster of SMP nodes.

References

[BL91] C.K. Birdsall and A.B. Langdon. *Plasma Physics via Computer Simulation.* Institute of Physics Publishing, Bristol and Philadelphia, 1991.

[CK76] C.Z. Cheng and Georg Knorr. The integration of the vlasov equation in configuration space. *J. Comput Phys.*, 22:330, 1976.

[CLS06] N. Crouseilles, G. Latu, and E. Sonnendrücker. Hermite spline interpolation on patches for a parallel solving of the vlasov-poisson equation. Technical Report 5926, Research report INRIA, 2006. http://hal.inria.fr/inria-00078455/en/.

[FS06] F. Filbet and E. Sonnendrücker. Modeling and numerical simulation of space charge dominated beams in the paraxial approximation. In *M3AS*, volume 16, pages 763–784, 2006.

[SRBG99] E. Sonnendrücker, J. Roche, P. Bertrand, and A. Ghizzo. The semi-lagrangian method for the numerical resolution of the vlasov equations. *J. Comput. Phys.*, 149:201–220, 1999.

	Time	Efficiency	Time	Efficiency	Time	Efficiency
Nb. processors	**16** ($proc_r = 1$)		**32** ($proc_r = 2$)		**64** ($proc_r = 4$)	
Advections						
advection 1D (φ)	8.69	100	4.28	101	2.11	103
advection 1D ($v_{//}$)	9.76	100	4.90	100	2.43	100
advection 2D (r, θ)	21.90	100	11.21	98	5.95	92
Field solver						
field solver (parts 1,2)	1.04	100	0.54	97	0.27	96
field solver (part 3)	0.20	100	0.20	50	0.20	25
field solver (comm.)	0.03	100	0.06	27	0.08	10

Table 1. Relative efficiency and computation time in seconds for a single time step of a small test case $\#_r = 128, \#_\theta = 128, \#_\varphi = 32, \#_{v_\parallel} = 32, \#_{\mu=}16$ ($proc_r$ the number of processors to compute one μ)

	Time	Efficiency	Time	Efficiency	Time	Efficiency	Time	Efficiency	Time	Efficiency
Nb. processors	**4** ($nbt = 1$)		**8** ($nbt = 2$)		**16** ($nbt = 4$)		**32** ($nbt = 8$)		**64** ($nbt = 16$)	
Advections										
advection 1D (φ)	83.30	100	41.51	100	21.82	95	10.80	96	5.38	97
advection 1D ($v_{//}$)	80.88	100	39.38	103	20.00	101	10.12	100	5.07	100
advection 2D (r, θ)	187.72	100	94.70	99	47.61	99	24.06	98	12.32	95
Field solver										
Field solver (parts 1,2)	7.03	100	3.62	97	1.87	94	1.00	88	0.56	79
Field solver (part 3)	6.15	100	3.19	96	2.00	77	0.90	86	0.44	88
Field solver (comm.)	0.09	100	0.08	60	0.08	29	0.09	13	0.11	5

Table 2. Relative efficiency and computation time in seconds for a single time step of a medium test case $\#_r = 256, \#_\theta = 256, \#_\varphi = 128, \#_{v_\parallel} = 64, \#_{\mu=}1$ ($proc_r = 4$ the number of MPI processes and nbt the number of threads within each MPI process)