A Block-Based Parallel Adaptive Scheme for Solving the 4D Vlasov Equation

Olivier Hoenen* and Eric Violard

Laboratoire LSIIT - ICPS, UMR CNRS 7005
Université Louis Pasteur, Strasbourg
{hoenen,violard}@icps.u-strasbg.fr

Abstract. We present a parallel algorithm for solving the 4D Vlasov equation. Our algorithm is designed for distributed memory architectures. It uses an adaptive numerical method which reduces computational cost. This adaptive method is a semi-Lagrangian scheme based on hierarchical finite elements. It involves a local interpolation operator. Our algorithm handles both irregular data dependencies and the big amount of data by distributing data into blocks. Performance measurements on a PC cluster's confirm the pertinence of our approach. This work is a part of the CALVI project [1].
Key words: Parallel numerics, 4D Vlasov equation, Adaptive method.

1 Introduction

The Vlasov equation can describe the evolution in time of charged particles under the effects of electro-magnetic fields. It is used to model important phenomena in plasma physics such as controlled thermonuclear fusion. This equation is defined in the phase space, i.e., the position and velocity space which has 6 dimensions in the real case (one dimension of velocity for each dimension of position). Methods discretizing the Vlasov equation on a mesh of phase space has been proposed to get an accurate description of the physics. Due to the high number of dimensions of the equation domain, solving this equation with such a numerical method yields a very large computational problem.

In order to reduce this computational cost, adaptive methods have been developed. Some of these methods are based on the semi-Lagrangian schemes [1, 2]. We developed such an adaptive method and its parallel implementation for the 2D Vlasov equation [3]. But a higher number of dimensions brings new challenges. On an other hand, some parallel implementations of 4D adaptive Vlasov solver exist but they are essentially designed for shared memory architectures. Therefore new parallel adaptive schemes have to be developed for distributed memory machines.

* supported by a grant from Alsace Region
[1] CALVI is a french INRIA project devoted to the numerical simulation of problems in Plasma Physics and beams propagation.

This paper presents an appropriate block-based semi-Lagrangian scheme and its parallelization. This scheme is based on a hierarchical finite element decomposition [4] which involves a local interpolation operator. This method yields an efficient parallelization based on data distribution into blocks.

The paper is organized as follows. Section 2 presents our adaptive semi-Lagrangian scheme. Section 3 presents in details its parallel implementation. Section 4 shows our experimental results before concluding.

2 An adaptive numerical scheme

We consider the four dimensional Vlasov equation

$$\frac{\partial f}{\partial t} + \mathbf{v}\frac{\partial f}{\partial \mathbf{x}} + E(t,\mathbf{x})\frac{\partial f}{\partial \mathbf{v}} = 0. \tag{1}$$

whose unknown $f(t,(\mathbf{x},\mathbf{v}))$ represents the distribution of particles at time t, where $\mathbf{x} = (x,y) \in \mathbb{R}^2$ and $\mathbf{v} = (v_x, v_y) \in \mathbb{R}^2$ are coordinates of a point in the phase space, and $E(t,\mathbf{x})$ is the, so called, self consistent electrostatic field, which is generated by particles. The equation is coupled with the Poisson's equation which gives the electric field E.

Our resolution scheme is based on the splitting of the equation (introduced in [5]) into a succession of three transport equations of the form

$$\frac{\partial f}{\partial t} + U(t,\mathbf{z})\frac{\partial f}{\partial \mathbf{z}} = 0\ . \tag{2}$$

whose resolution, so called *advection in* $\mathbf{z}$ (where $\mathbf{z}$ stands for phase space coordinates), is performed by using a semi-Lagrangian scheme. This scheme uses the property of conservation of the unknown along the characteristic curves, i.e.,

Property 1. $f(t^{n+1},(\mathbf{x},\mathbf{v})) = f(t^n, \mathcal{A}_{\mathbf{z}}^{\Delta t}(\mathbf{x},\mathbf{v}))$ where $\Delta t = t^{n+1} - t^n$ is the time step and $\mathcal{A}_{\mathbf{z}}^{\Delta t}$, so called *advection operator*, is a one-to-one correspondence between points of the phase space (see [6] for more details).

Our numerical method therefore boils down to perform three successive advections on a phase space mesh : one in x, one in y and one in $\mathbf{v}$. These advections are defined by the advection operators:

$$\begin{array}{l} \mathcal{A}_x^{\Delta t}(\mathbf{x},\mathbf{v}) = ((x - v_x\Delta t\ ,\ y)\ ,\ \mathbf{v}), \\ \mathcal{A}_y^{\Delta t}(\mathbf{x},\mathbf{v}) = ((x\ ,\ y - v_y\Delta t)\ ,\ \mathbf{v}), \\ \mathcal{A}_{\mathbf{v}}^{\Delta t}(\mathbf{x},\mathbf{v}) = (\mathbf{x}\ ,\ \mathbf{v} - E(t,\mathbf{x})\Delta t)\ . \end{array} \tag{3}$$

We use a *dyadic* structured adaptive mesh, i.e., a mesh whose each cell belongs to an uniform grid of 2^j cells per dimension. The integer j is called the level of the cell. In this 4-dimensional dyadic mesh, each cell of level j, say α, is a 4-cube and can be refined into 2^4 smaller cells of level $j+1$. These smaller cells are called the *daughters* of α and, by analogy, cell α is called their *mother*. We will denote J, the finest level and j_0, the coarsest level of our mesh cells.

At any time step $t^n = n\Delta t$, the solution f is represented by a dyadic mesh $\mathcal{M}^n$ and a function $\mathcal{F}^n$ which gives the value of f at every point $(\mathbf{x}, \mathbf{v})$ corresponding to a node of $\mathcal{M}^n$. Each cell has 3^4 equally spaced nodes and we use a biquadratic Lagrange interpolation to reconstruct the value of f at any point $(\mathbf{x}, \mathbf{v})$ within a cell.

Let us now present one advection in $\mathbf{z}$, i.e., our numerical scheme for solving one transport equation. It gives the new representation $(\mathcal{M}^{n+1}, \mathcal{F}^{n+1})$ from a known old one $(\mathcal{M}^n, \mathcal{F}^n)$. It consists in the following procedure, for every cell α of the uniform grid of level j_0 and denoting $\mathcal{M}_\alpha^{n+1}$ the part of $\mathcal{M}^{n+1}$ which is contained in α:

1. **Mesh prediction**: Let us note j_β, the level of any cell β; C_β, the point of the phase space corresponding to the center of β; β', the cell of $\mathcal{M}^n$ which contains the advected point $\mathcal{A}_{\mathbf{z}}^{\Delta t}(C_\beta)$ and $j_{\beta'}$, the level of β'. Then recursively refine each cell β of $\mathcal{M}_\alpha^{n+1}$ such that $j_{\beta'} \geq j_\beta$.
2. **Computation of values**: For every node of $\mathcal{M}_\alpha^{n+1}$, let N be its corresponding point in the phase space and α', the cell of $\mathcal{M}^n$ which contains the advected point $\mathcal{A}_{\mathbf{z}}^{\Delta t}(N)$. By conservation property 1, set $\mathcal{F}^{n+1}(N)$ to the interpolated value at point $\mathcal{A}_{\mathbf{z}}^{\Delta t}(N)$ by using the values of $\mathcal{F}^n$ at nodes of α'.
3. **Mesh compression**: For every group of 2^4 daughter cells in $\mathcal{M}_\alpha^{n+1}$, compare every value at their nodes with the value obtained by interpolation by using the values at mother's nodes. If the L1-norm of the difference is lower than a given threshold ϵ, then replace the daughter cells with their mother into $\mathcal{M}_\alpha^{n+1}$.

Figure 1 shows the time loop of our resolution scheme. Notice that for sake of conciseness, the diagnostic step has not been reported. As the advection in $\mathbf{v}$ step uses the electric field E, it is preceded by the computation of E. In order to compute E, the Poisson's equation is discretized onto an uniform grid of the position space $(\mathbf{x})$ with $2^{J+1}+1$ points per dimension. The computation of E then consists in a summation of values onto the 2D position space.

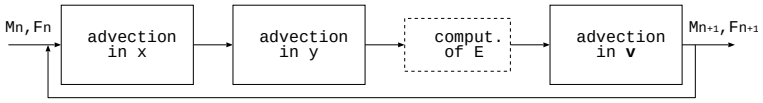

Fig. 1. The time loop of our resolution scheme.

3 Parallelization

In our numerical scheme an advection consists in applying the same treatment to every parts $\mathcal{M}_\alpha^{n+1}$ of the dyadic mesh in any order. In the following we call *block* all the data used to describe such a part. Our parallel implementation is based on the distribution of blocks amongst processors. We first present the data structure which will represent the dyadic mesh. We propose a partitioning of the

mesh which will determine how blocks are distributed. Then we show how to optimize communications.

3.1 Data structure

Data structure has a great impact on the efficiency of an adaptive method implementation. In our case, due to the domain high number of dimensions and the big amount of data it represents, we search a good tradeoff between data access cost and memory usage. In our algorithm, the access to any value is defined by the advection operator that uses an absolute location within the phase space. Therefore, random access is critical to our algorithm (as shown in [7]). Thus, we use arrays to store values. Moreover, we use pointers to handle mesh sparsity and we use dynamically-allocated arrays to minimize memory usage.

Our data structure is composed with four-dimensional arrays and has two level. The first level is made of two arrays: one $[2^{j_0+1}]^4$-sized array of double which stores the values at nodes of the cell of level j_0 and one $[2^{j_0}]^4$-sized array of pointers whose elements are in one-to-one correspondence with cells of the uniform grid of level j_0. Each pointer either is NULL meaning that the corresponding cell belongs to the mesh, or points to an array of second level. An array of second level stores the values of all the nodes which are located within the corresponding part of the mesh. The undefined value is stored in the array when the corresponding node does not exist in the mesh. The second level is thus made of some $[2^{J-j_0+1}]^4$-sized arrays of double. Moreover since the mesh adapts at each time step, the arrays of second level are dynamically-allocated arrays. A data block either is stored into the array of first level if it describes a single cell of level j_0, or is stored into an array of second level.

3.2 Mesh partitioning

A good mesh partitioning is one which both reduces the cost of communications, and balances the computational load [8]. In order to meet the first issue, we use the following properties of our numerical method :

- During any advection in $\mathbf{v}$, the treatment of any block, say B, only requires data within blocks having the same coordinates in position space as B. Symmetrically, during any advection in $\mathbf{x}$, the treatment of any block B only requires data within blocks having the same coordinates in velocity space as B.
- Any $\mathbf{v}$-advection exhibits irregular and unpredictable data dependencies because these dependencies are defined by the self consistent electrostatic field. On an other hand, the data dependencies in any $\mathbf{x}$-advection are predictable as they are only defined by $\mathbf{v}$ and dt.

According to these properties, if we partition the mesh along the position dimensions, then the advections in $\mathbf{v}$ will induce no communication. Moreover, the communications which will occur during the advections in $\mathbf{x}$ are predictable and

linear. Hence, we choose to reduce the partitioning problem to partitioning the 2D uniform grid of level j_0 of the position space. This means that each partition is defined by an 2D area made of some cells of this uniform grid. The partition contains all the mesh cells of the 4D phase space whose projection onto the position space is included in this area. As said previously the induced communications are linear, therefore we only consider connex areas to form our partitions. More precisely, we choose to define our partitions from some rectangular areas, which simplifies the communications scheme.

Now to achieve the second requirement that is load balancing we build approximately balanced areas for any given test case. Notice that our approach in this work is to perform a static partitioning. Future work will be directed at modifying partitions dynamically to follow the evolution in time of the physics more closely. As said previously, each area will be a 2D rectangular one. Numerous techniques tends to build such areas by minimizing the ratio between height and width and reduce the amount of border elements in each direction that usually induces communications (see [9] as an example). Besides these techniques, our partitioning uses some knowledge on the considered test case. More precisely, it uses a bounding box, say $\mathcal{B}$, which approximates the shape of the beam of particles. It is assumed that most of computational load is carried by the mesh elements contained within bounding box $\mathcal{B}$. The other parameters used in the building of our partitions are P, the number of processors and j_0, the level of the uniform grid. Notice that level j_0 defines the coarse grid and determines the number of discretisation points per surface unit of the physics domain. We therefore consider it as a parameter and not as an unknown because we do not want to influence the solution accuracy.

The partitioning problem then reduces to find P rectangles composed of coarse cells such that these rectangles do not overlap and every rectangle approximatively covers the same extent of surface of $\mathcal{B}$. Our partitioning problem is an optimization one which is minimizing the greatest area amongst the P surface extents of $\mathcal{B}$ which are covered by each of the P rectangles. This optimization problem can be expressed as an 0-1 integer linear programming problem. It is thus in general NP-hard, and as such, it is considered unlikely that there exists an efficient algorithm for solving it. Instead, we propose an heuristic which consists in recursively splitting the bounding box area into 2 approximatively equal parts. It is thus assumed that the number of processors is a power of 2.

3.3 Obtaining regular communications

The communications are generated by the data dependences which are induced by the advection operator: the treatment of a node corresponding to the point (x, y, v_x, v_y) of the phase space requires the values of the cell containing the advected point. As we partition the mesh along the (x, y)-axis (Cf. section 3.2), the communications only occur during a x- or y-advection and the data dependences are defined by the linear advection operators $x \mapsto x - v_x \Delta t$ and $y \mapsto y - v_y \Delta t$. Figure 2 (left) shows these data dependences for the x-advection in the Cartesian (x, v_x)-plane : the data on the oblique line are needed to compute the data

on the vertical line. Figure 2 (right) shows which data blocks are required to compute the data blocks on a column. In this particular case, we observe that

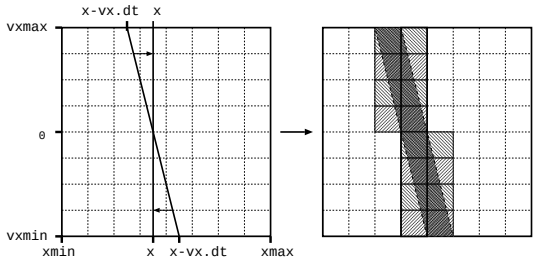

Fig. 2. Data dependencies and blocks required for updating one column of blocks.

the computation of any data blocks only requires the data within two blocks : the block itself and a neighboring block at the left or at the right depending on the v_x sign. It is a suitable situation where the communication volume is minimum. This situation occurs when the test case parameters satisfy a particular condition. Our algorithm works on the assumption that this condition holds. This condition can be expressed by the following equation

$$\frac{x_{max} - x_{min}}{2^{j_0}} \geq \max(\ |v_{xmin}|\ ,\ |v_{xmax}|\ )\Delta t\ . \tag{4}$$

3.4 Communication overlapping

Conceptually, in the application of partitioning, each partition is assigned to a processor. In this section, we consider a lower level of abstraction and discuss how data have to be effectively distributed across processors in order to reach the best performance. To allow easy access to boundary data on neighboring processors, the data structure is extended to include *ghost cells* on each processor. Ghost cells are remote data blocks replicated in local memory to reduce access time (see [10]). The data structure have allocated memory for some boundary blocks

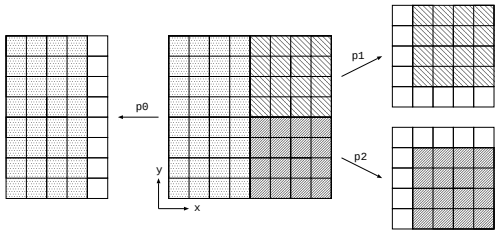

Fig. 3. Extension of global ordinary data structure to local regions, i.e., data structures with replicated blocks (for 3 processors).

that can be filled in with data from the adjoining processors. This is illustrated in

figure 3. In the following, the region assigned to a processor that is formed from the partition extended with replicated blocks will be refered to as *local region*. Each advection in our numerical scheme then consists in an "update operation" which computes data within the local region and fills in some of replicated blocks with updated data.

Several algorithms for this update operation can be investigated. Their efficiency mostly depends on the characteristics of the computer system and on the size or dimensionality of the problem (see [11] for more details). Our implementation is described in algorithm 1. This algorithm tends to maximize the overlapping of communication and computation.

Algorithm 1: Update of one local region

```
Input: A, one local region
Output: B, the updated region
begin
    init recv of replicated blocks of B from adjoining processors
    foreach border blocks of B do
        compute it from blocks of A
        init its send to the neighboring processor
    end
    compute inner blocks of B (from inner and border blocks of A)
    wait all send/recv
end
```

We use two data structures we call A and B. Data structure A stores the data before the operation and B stores the resulting data. Since our numerical scheme consists in a succession of advections, the output data is used as the input by the next operation. The role of each structure changes each time the operation is performed. This change of role is achieved by pointer assignments. In addition, the local region is virtually split into three disjunctive parts : the *replicated blocks*, the *border blocks* and *inner blocks*. These parts are defined by the direction (x, y or $\mathbf{v} = (v_x, v_y)$) of the next advection. They are shown on figure 4. The algorithm works on the assumption that all required data of A

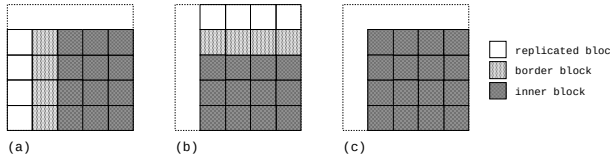

Fig. 4. The local region p2 in Fig.3 split into three kinds of blocks: (a) when the next update is a x-advection, (b) when the next update is a y-advection, (c) when the next update is a $\mathbf{v}$-advection (no replicated or border block).

are initially available. In particular, it means that the replicated blocks have been filled in with data at the first time step. The algorithm first initiates the non-blocking receive of all these blocks in B. Then, it computes each border

block of B and initiates its send as soon as it is computed. Communications are overlapped with computation of inner blocks and next border blocks of B.

4 Implementation and performance measurements

Our code has been written in C and use the portable implementation MPICH [12]. In order to reduce memory copies, we defined two MPI data-types that describe the storage of data within the cells which are received or sent.

A difficulty here is that data blocks may have two different memory mapping because a data block describes one single or more cells, as said in section 3.1. Because of the adaptive mesh, it is not possible to predict which kind of blocks will be sent or received. Therefore we use two MPI data-types : `MPI_Block1` and `MPI_Block2`, one for each block kind. These two types are identical except that the extent of type `MPI_Block2` is greater than the extent of type `MPI_Block1` in memory. Each of these two types is a structure with the same fields and the same offset for each of these fields. We use type `MPI_Blocks2` to initiate a block receive. Therefore data are received in place whatever the kind the block has. Last, a call to function `MPI_Get_count` is used to determine the block type.

For our experiments, we use a PC cluster's equipped with 17 dual core processors with 2 GB of RAM each. Processors are Athlon 4800+ cadenced at $2.4\,GHz$ and connected through an Gigabit Ethernet. Each core is seen as an independent MPI processor unit. Our test case is the uniform magnetic focus-

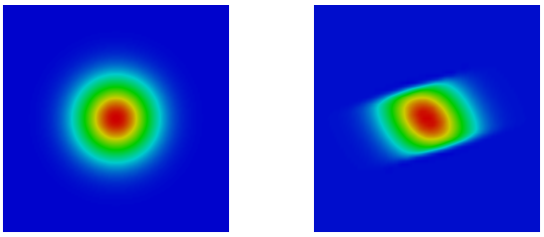

Fig. 5. Representation of the $x-y$ projection (left) and $x-v_x$ projection (right) of the distribution function at time step 20

ing of a semi-Gaussian beam of protons. The emittance of this proton beam is $2.36\times10^{-5}\,\pi\,m\,rad$, its current is equal to $100\,mA$, and its energy is $5\,MeV$. The initialization parameters are computed by solving the envelope equation of the equivalent KV beam (see [13]). These physical parameters give a tune depression of 0.7. The length of the period is equal to $S = 26.6292\,m$. We perform 75 time steps (with $\Delta t = 0.0841\,s$). The particles beam is given by a semi-Gaussian distribution function

$$f_0(\mathbf{x},\mathbf{v}) = \begin{cases} \frac{1}{8\pi^2} e^{(-\frac{{v_x}^2+{v_y}^2}{2})} & \text{if} \quad x^2+y^2<6 \\ 0 & \text{else} \end{cases} . \tag{5}$$

where $\mathbf{x}$ and $\mathbf{v}$ live in $[-6,6]^2$. The number of points is equal to 128 in each direction of position space and 64 in each direction of velocity space. We use

$j_0 = 3$ and $J = 5$. Figure 5 shows projection of the distribution function.

Impact of load balancing. We compare two partitioning of the mesh. The first one uses our heuristics to build balanced areas with a bounding box set to the square $[-3.5, 3.5]^2$. The second one splits the domain into equally sized areas such that the difference between width and height is minimum. Figure 6 (right) shows the wall-clock time of the code in each of these two cases. Since the **x**-projection of the distribution function is round in shape and forms a disk of center $(0, 0)$, the second partitioning gives well-balanced areas for 4 processors. Therefore our heuristics cannot enhance performance in that case. Otherwise simulations are between 15% and 30% faster with our heuristics.

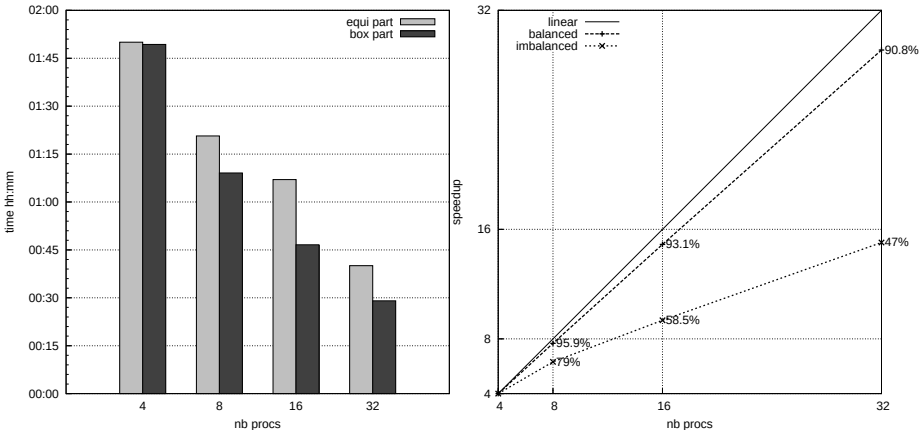

Fig. 6. Performances of our code on a PC cluster

Performance and speedup. We observe that the performance improves until 32 processors. But the speedup is under-linear due to load imbalance arising from the beam evolution in time. Figure 6 (left) shows the speedup and efficiency obtained when the workload is well-balanced throughout the execution (setting the compression threshold $\epsilon = 0$) in comparison with an unbalanced execution. Note that the version of the code is not fully optimized, particularly the interpolation operation cost can be reduce. We observe a slight loss of efficiency. It is not due to communications because they are totally overlapped by computation: we checked that in this experiment all messages are received before the wait barrier is reached. Thus, it can be explained by the computation of field E that is sequential.

5 Conclusion

In this paper, we proposed a new adaptive numerical scheme and its parallel implementation for solving the 4D Vlasov equation. On the contrary to the previous method [14], each part of the predicted mesh can be computed independently

of each other, at each time step. Thus, our parallel implementation based on mesh partitioning induces low overhead. Moreover, we made some assumptions on the physical parameters in order to ensure a regular communication pattern. Experiments on a PC cluster shown a good scalability provided the computational workload is well balanced throughout the whole simulation. They shown that a static partitioning alone is not enough to obtain the good scalability and performance even in the advantageous case of the focusing beam. Future works then consist in integrating to the code a mechanism that updates partitioning in order to follow the evolution of the physics. Then we intend to compare the scalability of our solver for distributed memory machines with the codes which target shared memory machines.

References

1. M. Gutnic, M. Haefele, and G. Latu. A parallel Vlasov solver using a wavelet based adaptive mesh refinement. In *7th Workshop on High Perf. Scientific and Engineering Computing (ICPP'2005)*, pages 181–188, 2005.
2. E. Sonnendrücker, F. Filbet, A. Friedman, E. Oudet, and J.L. Vay. Vlasov simulation of beams with a moving grid. *Comput. Phys. Commun*, 164:390, 2004.
3. O. Hoenen, M. Mehrenberger, and E. Violard. Parallelization of an adaptive Vlasov solver. In *11th European PVM/MPI Users' Group Conference (EuroPVM/MPI), ParSim Session*, volume 3241 of *LNCS*, pages 430–435. Springer-Verlag, 2004.
4. M. Campos-Pinto and M. Mehrenberger. Adaptive numerical resolution of the Vlasov equation. In *Numerical Methods for Hyperbolic and Kinetic Problems, CEMRACS*, pages 43–58, 2004.
5. C. Z. Cheng and G. Knorr. The integration of the Vlasov equation in configuration space. *J. of Comput. Phys.*, 22:330–351, nov 1976.
6. E. Sonnendrücker et al. The semi-Lagrangian method for the numerical resolution of the Vlasov equation. *J. Comput. Phys.*, 149:201–220, 1999.
7. O. Hoenen and E. Violard. An efficient data structure for an adaptive Vlasov solver. Technical Report 06-02, LSIIT–ICPS, February 2006.
8. A. Boukerche and C. Tropper. A static partitioning and mapping algorithm for conservative parallel simulations. In *8th workshop on Parallel and Distributed Simulation (PADS'94)*, pages 164–172, 1994.
9. M. J. Berger and S. H. Bokhari. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Transactions on Computers*, 36(5):570–580, 1987.
10. C. Ding and Y. He. A ghost cell expansion method for reducing communications in solving PDE problems. In *Supercomputing'01 (CDROM)*, 2001.
11. B. Palmer and J. Nieplocha. Efficient algorithms for ghost cell updates on two classes of mpp architectures. In *Parallel and Distributed Computing and Systems (PDCS 2002)*, pages 192–197, 2002.
12. W. Gropp et al. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, 1996.
13. F. Filbet and E. Sonnendrücker. Modeling and numerical simulation of space charge dominated beams in the paraxial approximation. Technical Report RR-5547, INRIA Lorraine, 2005.
14. M. Mehrenberger et al. A parallel adaptive Vlasov solver based on hierarical finite element interpolation. *Nucl. Inst. and Meth. in Phys. Res.*, 558(A):188–191, 2006.