Load-Balancing for a Block-Based Parallel Adaptive 4D Vlasov Solver *

Olivier Hoenen and Eric Violard

LSIIT-ICPS, University Louis Pasteur - Strasbourg, France
{hoenen,violard}@icps.u-strasbg.fr

Abstract. This work is devoted to the numerical resolution of the 4D Vlasov equation using an adaptive mesh of phase space. We previously proposed a parallel algorithm designed for distributed memory architectures. The underlying numerical scheme makes possible a parallelization using a block-based mesh partitioning. Efficiency of this algorithm relies on maintaining a good load balance during the whole simulation. In this paper, we propose a dynamic load balancing mechanism based on a relevant cost metric and a geometric partitioning algorithm. This mechanism is deeply integrated into the parallel algorithm in order to minimize overhead. Performance measurements on a PC cluster show the good quality of our load balancing and confirm the pertinence of our approach.

Keywords: Parallel numerical algorithm, Vlasov equation, Adaptive method, Load balancing.

1 Introduction

The Vlasov equation is a non-linear partial differential equation that describes the evolution in time of charged particles under the effects of external and self-consistent 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 which has 6 dimensions in the real case (one dimension of velocity for each dimension of position).

Amongst the numerical methods for solving the Vlasov equation, recent Eulerian methods (see [6, 5]) based on the semi-Lagrangian scheme [11] are of great interest to get an accurate description of the physics. These methods have proven their efficiency on uniform meshes in two dimensional phase space. But when the dimensionality increases, the number of points on a uniform grid becomes very important which makes numerical simulations challenging.

Two approaches have been investigated to simulate four dimensional problems: adaptive methods and parallel algorithms. Adaptive methods decrease computational cost drastically by keeping only a subset of all grid points. Some

* This work was partially supported by a grant from Alsace Region and is part of the french INRIA project CALVI (http://www-math.u-strasbg.fr/calvi).

semi-Lagrangian adaptive methods have been developed, like in [10] and [2, 7] where the authors use a moving grid or a multi-resolution analysis based on interpolating wavelets. But, few works use both approaches. A main difficulty in developing a time-dependent highly adaptive solver is achieving a good load balancing. At our knowledge, existing parallelized version of these adaptive methods are essentially designed for shared memory architectures [8, 4].

In the present work, we investigate another adaptive method for solving the four dimensional Vlasov equation. This scheme is based on a hierarchical finite element decomposition [3] and on a convenient time splitting. It has been designed for targeting distributed memory architectures. This scheme and its parallelization have been introduced in [9]. This previous work tackles load balancing through a static partitioning based on initial function values. In this paper, we address the crucial problem of maintaining a good load balancing during the whole simulation. In order to solve this problem, we developed a suited dynamic load-balancing mechanism based on a geometric partitioning algorithm.

The paper is organized as follows: next section recalls our numerical scheme and parallel algorithm. Section 3 presents our load balancing mechanism and its integration into the algorithm. Section 4 reports performances obtained on a PC cluster before concluding.

2 The parallel adaptive solver

Evolution of particles in the phase space is given by a distribution function $f((\mathbf{x}, \mathbf{v}), t)$ where $(\mathbf{x}, \mathbf{v}) \in \mathbb{R}^d \times \mathbb{R}^d, d = 1, ..., 3$. Value of this function is given by the normalized Vlasov equation,

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

where the self-consistent electric field $\boldsymbol{E}(\mathbf{x}, t)$ is computed using the Poisson's equation from the charge density $\rho(\mathbf{x}, t)$

$$\rho(\mathbf{x}, t) = \int_{\mathbb{R}^d} f(\mathbf{x}, \mathbf{v}) d\mathbf{v} \tag{2}$$

In this work we consider the reduced case of a four dimensional phase space ($d = 2$) where $\mathbf{x} = (x, y) \in \mathbb{R}^2$ and $\mathbf{v} = (v_x, v_y) \in \mathbb{R}^2$. We refer the reader to [9] for more details about this numerical resolution scheme and its parallelization.

2.1 Numerical scheme

Our adaptive mesh is a structured *dyadic mesh*, i.e., a hierarchical mesh with at most J levels, where each cell at level j is a 4-cube that may be subdivided into 16 equal-sized cells at level $j+1$. Coarsest cells (at level 0) belongs to an 4D uniform grid that we call the *coarse grid*.

Our resolution scheme is based on the splitting in time of the Vlasov equation into three transport equations in x, in y and in $\mathbf{v}$. Time is discretized and at

each time step $\Delta t = t^{n+1} - t^n$, the three transport equations are solved in turn. The solution of a transport equation is used as initial condition for solving the following one.

For any axis direction z, we solve a transport equation in z by using a semi-Lagrangian scheme based on the conservation property [11] of solution:

$$f((\mathbf{x}, \mathbf{v}), t^{n+1}) = f(\mathcal{A}^z_{\Delta t}(\mathbf{x}, \mathbf{v}), t^n) \tag{3}$$

where $\mathcal{A}^z_{\Delta t}$ is called *advection operator*. This advection operator defines the particles motion along z direction. We have $\mathcal{A}^x_{\Delta t}(\mathbf{x}, \mathbf{v}) = ((x - v_x.\Delta t, y), \mathbf{v})$, $\mathcal{A}^y_{\Delta t}(\mathbf{x}, \mathbf{v}) = ((x, y - v_y.\Delta t), \mathbf{v})$ and $\mathcal{A}^{\mathbf{v}}_{\Delta t}(\mathbf{x}, \mathbf{v}) = (\mathbf{x}, \mathbf{v} - E(\mathbf{x}, t^n).\Delta t)$. The procedure to solve a transport equation in z is called *advection*. It finds a representation (on a dyadic mesh) of the solution at time t^{n+1} from a known representation of the solution at time t^n in three steps:

1. (prediction) We build a new dyadic mesh with enough nodes to get an accurate representation of solution at time t^{n+1}.
2. (valuation) We compute the values at the new mesh nodes from conservation property: each value at point $(\mathbf{x}, \mathbf{v})$ is obtained by a biquadratic Lagrange interpolation from the values at the old mesh nodes close to point $\mathcal{A}^z_{\Delta t}(\mathbf{x}, \mathbf{v})$.
3. (compression) We coarsen some new mesh cells to remove unnecessary nodes that were improperly created during prediction step.

The electric field E is computed on a uniform grid at the finest level J of the 2D position space. We first integrate f to get the discrete charge density ρ at each point of this uniform grid. This is achieved by computing the contribution of every cell of the dyadic mesh and then by computing the sum of all these contributions. Then, we solve the Poisson's equation by using Fourier transforms. Since the value of E is required for performing the advection in $\mathbf{v}$, the computation of E precedes this advection within each time-step iteration.

Figure 1 summarizes our resolution scheme and shows the successive operations within each time-step iteration. For sake of conciseness, specific treatments for diagnostics have been omitted.

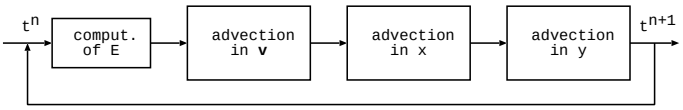

Fig. 1. The time-step iteration of our resolution scheme.

2.2 Parallel algorithm

An important property of our numerical scheme is that each step of an advection (prediction, valuation and compression) can be performed locally within a region of the phase space independently of the others. Therefore, given any partitioning of the phase space, an advection can be performed by applying the

same treatment to every partition independently. This is the base of our data-parallel algorithm. Each processor is assigned to one partition. It holds in its local memory all the data within this partition, i.e., the corresponding parts of the old and new meshes. According to the classical owner computes rule, it is in charge of all the computations local to this partition. We define a *partition* as an union of *blocks*, where a block is defined as the 4D cube-shaped phase space area corresponding to a cell of the coarse grid. Our partitioning aims at reducing communication volume and is based on the following dependency analysis.

The data dependencies are defined by the advection operators: value at point $(\mathbf{x}, \mathbf{v})$ depends on the values at a few points close to point $\mathcal{A}^z_{\Delta t}(\mathbf{x}, \mathbf{v})$. Given the advection operator only acts on one particular axis (x, y or $\mathbf{v}$) while letting the other coordinates unchanged, the treatment of any block only requires data within some blocks along the same axis. For example, let us consider a block of integer coordinates $(i_x, i_y, i_{v_x}, i_{v_y})$ within phase space. During any advection in x, the treatment of this block only requires data within some blocks having the same coordinates i_y, i_{v_x}, i_{v_y}.

The block dependencies during an x-advection (similarly during an y-advection) are predictable and linear. Figure 2 (left) shows their projection on the plane (x, v_x) for two distinct values of Δt. Parameters $[xmin, xmax] \times [vxmin, vxmax]$ in the figure define the borders of the domain on which we solve the Vlasov equation. Blocks in light grey are ones needed to compute blocks in dark grey. We observe that provided some assumptions on simulation parameters (for $xmin = -xmax$, $vxmin = -vxmax$, and $\Delta t < \frac{blocksize}{vxmax}$, see the right part of figure 2), the treatment of any block only needs 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. On the other hand, any $\mathbf{v}$-advection exhibits irregular and unpre-

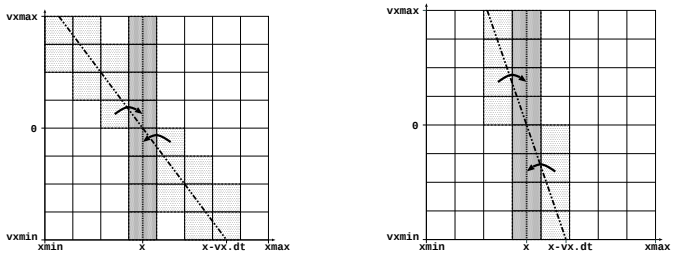

Fig. 2. Dependencies during an advection in x.

dictable block dependencies because the advection operator is determined by the self consistent electrostatic field E which depends itself on the values of f. These block dependencies may induce costly communications. Therefore, the partitions we consider are such that all blocks having the same coordinates i_x, i_y in position space (for any coordinates i_{v_x}, i_{v_y} in velocity dimensions) are contained in the same partition. With such a partitioning, any $\mathbf{v}$-advection induces no communication. Let us call *slice* of coordinates (i_x, i_y), the set of all the blocks having the same coordinate i_x, i_y in position dimensions. Each partition is thus a group of slices.

The remaining communications during x- and y-advection are implemented by using *ghost cells* to replicate needed remote data. We also use overlapping of communications with computations to reduce the overhead for updating the ghost cells. This update operation is performed one advection ahead in order to maximize computations/communications overlap. Figure 3 shows the location of this update operation within each time step. The computation of electric field E requires one all-to-all communication of ρ contributions.

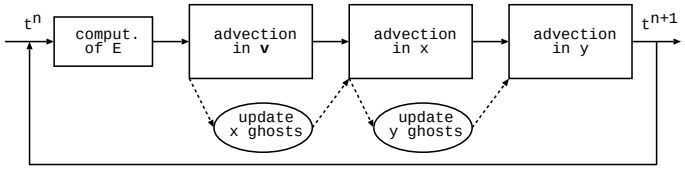

Fig. 3. The time-step iteration of our parallel algorithm including communications/computations overlapping.

3 Load-balancing mechanism

Our mechanism is based on three issues: a cost metric to measure the load of a partition and detect imbalance, a partitioning algorithm and a procedure to map new partitions to processors.

3.1 Imbalance detection

Our imbalance detection relies on measuring the cost of any partition. Ideally, this cost takes both computations and communications into account [12]. In the scope of this work, we decide to neglect the communications cost. We do this approximation because communications are overlapped with computations and thus their cost is partially hidden. Therefore, the chosen cost metric is time spent in computing one block. The measure is taken dynamically using the processor real time clock. This metric is well adapted to homogeneous architectures where all processors are cadenced at the same frequency. By definition, the cost of any partition is the sum of the costs of all the slices that are contained in this partition. The cost of any slice is the sum of the costs of all the blocks in this slice. Load imbalance is detected when the difference between the greatest partition cost and the lowest one is over a given threshold.

3.2 Partitioning algorithm

Our partitioning algorithm aims to build partitions having approximately the same cost. We propose a heuristic which is based on the well-known Recursive Coordinate Bisection (RCB) [1]. The RCB technique is a geometric-based partitioning method, which has the advantages of being simple and well suited to

Cartesian meshes. The geometric domain that we consider in our variant, is a 2D uniform grid of the position space. Each cell (i, j) of this grid identifies a slice and therefore a sub-domain made of several cells defines a partition. As in the classical algorithm, a divide-and-conquer approach is taken. The domain is first cut in largest dimension to yield two sub-domains. Cuts are then made recursively in the new sub-domains until we have as many sub-domains as processors. Each cut is made so that the two partitions corresponding to sub-domains have approximately the same cost. In the case of our parallel algorithm, we do not want to subdivide a cell: the cut can only be made between adjoining cells.

If we use a straight line cut, then the cost of the two resulting partitions may be too different. We propose an inexpensive optimization with the aim at reducing the difference between the costs of resulting partitions. This optimization consists in using a *zig zag cut* rather than a straight line cut This is shown in figure 4: on the left, the set of cells is optimally bisected with a straight line cut between two columns. On the right, the optimal bisection (the dotted straight line) passes through a column of cells. In this case we use a zigzag cut: the orthognal segment is chosen to better divide the cost. Our RCB variant is given

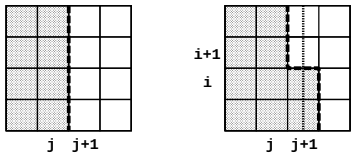

Fig. 4. Bisection of a set of slices.

in algorithm 1. It uses three local variables : *rank*, the rank of the processor associated to the current partition, *part*, the current partition that corresponds to a sub-domain of the 2D uniform grid, and *nbis*, the number of cuts that have been already made. It could happen that our algorithm generates an empty partition or new partitions such that the imbalance is not better than the previous one. In that case, the new partitioning is considered as not valid and the current one remains unchanged.

3.3 Partitions re-mapping on processors

The mapping of the new partitioning onto processors involves communication of slices amongst processors, which may cause a significant overhead. In order to reduce this overhead and not to waste the gain of a better load balance, this redistribution and the other issues of load balancing are mixed up with the successive operations within a time-step iteration. Thus communications can be overlapped with computations which would not be possible if re-mapping was performed in a seperate phase. When a load balancing occurs, the time-step iteration given in figure 3 is slightly modified as shown in figure 5. Load balancing issues (in grey) are integrated into the computation of E and two advection steps. Migration of slices is overlapped with **v**-advection computations.

Algorithm 1: Recursive partitioning function

Data: *nproc*, the number of processors (assumed to be a power of 2)
Input: $rank, part, nbis$ (initially $rank = 0$, $part$ is the whole 2D domain, $nbis = 0$)
if $2^{nbis} = nproc$ **then**
  maps partition $part$ to processor $rank$
else
  compute $(H \times W)$, the height and width extent of the current partition $part$
  if $H > W$ **then**
    switch dimensions x and y
  let $w(n, m)$ be the cost of slice (n, m) if $(n, m) \in part$ (0 otherwise)
  compute cost $w[n]$ of every column n of partition $part$ ($w[n] = \sum_m w(m, n)$)
  compute cost sum of the whole partition $part$
  find column $j = \max\{k \mid S_k \leq sum/2\}$ with $S_k = \sum_{n \leq k} w[n]$
  if $S_j = sum/2$ **then**
    the straight line cut is between columns j and $j + 1$
  else
    the cut pass through the column $j + 1$
    find row i which minimizes $|sum/2 - (S_j + s_i)|$ with $s_i = \sum_{m \leq i} w(m, j)$
    the cut is between rows i and $i + 1$
  let $part_1$ and $part_2$ be the subsets of $part$ defined by bisection
  call partitioning function with $rank$,$part_1$,$nbis + 1$
  call partitioning function with $rank + nproc/(2^{nbis+1})$, $part_2$,$nbis + 1$

Let us describe these changes in details. The cost of each partition is communicated during computation of electric field E within the same message as ρ contribution. These costs are used to perform the imbalance test. If the workload is not sufficiently balanced, then all processors exchange the cost of their slices. This does not penalize solver performance since processors are already synchronized. Then, on each processor, the partitioning algorithm is used to compute new partitions and then, a re-mapping of partitions is planned. For any given processor, say P, let us note P_{old}, its old partition and P_{new}, its new one. During **v**-advection, processor P computes all the (local) blocks of P_{old}, sends locally computed blocks belonging to the difference $P_{old} \setminus P_{new}$ and receives remote computed blocks belonging to $P_{new} \setminus P_{old}$ (including ghosts cells in x for P_{new}).

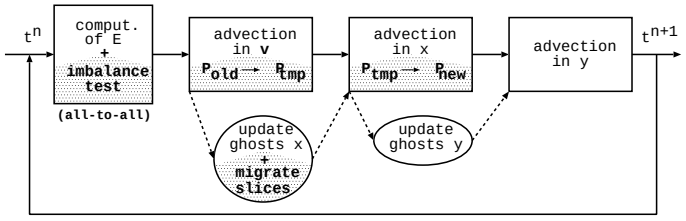

Fig. 5. The time-step iteration including load balancing.

In order to temporarily store both computed and received data, a new memory space is allocated. Its extent corresponds to the union $P_{tmp} = P_{old} \cup P_{new}$. During x-advection, the only change made for load balancing issues is to read needed data from the temporary memory space. At the end of the x-advection step, the temporary memory space becomes useless and thus is deallocated. Then the time-step iteration returns to a normal state.

4 Performance measurements

Our code has been written in C with calls to MPI. Our test case is the uniform magnetic focusing of a semi-Gaussian beam of protons. A detailed description of this test case is given in [9]. We perform 75 time steps. Coordinates $\mathbf{x}$ and $\mathbf{v}$ live in $[-6.5, 6.5]^2$. Phase space is split into 16×16 slices of 8×8 blocks each and $J = 2$, which corresponds to maximum grid of $128 \times 128 \times 64 \times 64$ points. Simulation starts with a partitioning determined from the initial distribution function. We test our code on a cluster of Opteron 2.4 GHz bi-processor nodes with 4 GB RAM each, connected through a Myrinet network. Each node holds 2 Myrinet interfaces to achieve a theoretical bandwidth of 495 MB/s.

Impact of load balancing. Figure 6 shows the workload imbalance at each time step of the simulation launched on 16 processors for 3 different load balancing (LB) strategies. We approximate imbalance as the difference between the greatest and the lowest partition cost and a mark indicates each balancing step. Imbalance is given for a simulation without any dynamic load balancing, with dynamic LB based on straight cuts (sDLB) and with zigzag cuts (zDLB). With zDLB, we can observe that the imbalance cost always is under 1 second.

Figure 7 shows the impact of re-mapping on communications. Light grey columns represent the number of sent blocks, and dark grey ones represent the number of blocks that are waited at the `MPI_Waitall` barrier. We can observe

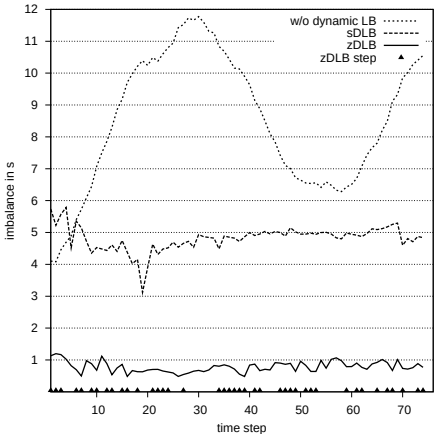

Fig. 6. Imbalance evolution with different dynamic load balancing strategies.

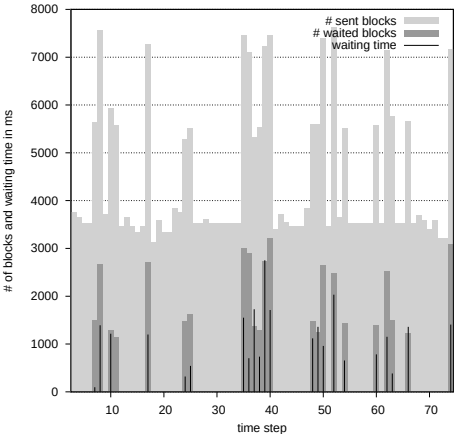

Fig. 7. Impact of re-mapping on communications.

that most of the time, all sent blocks are received before the waiting barrier. But for some re-mapping there is a strong increasing of the number of tansferred blocks. When the number of blocks to be sent is over the number of blocks to be computed, communication time can not be entirely overlapped with computation, thus there are some waiting times. This overhead is a drawback of our partitioning algorithm that does not take communication cost into account.

Performance and speedup. We compare the impact of our dynamic load balancing algorithm upon wall-clock time. We run the same simulation with 4 different LB strategies: zDLB, sDLB, and static LB with equal-sized areas (eSLB) and partitioning of a bounding box (bSLB) based on the initial distribution function (as defined in [9]). Figure 8 shows the wall-clock time of the code in each case for an increasing number of processors. We can see that zigzag cut partitioning always achieves better performance than straight cut one.

Figure 9 shows speedup and efficiency obtained up to 32 processors. We also made a run with 32×32 slices instead of 16×16. Speedup is quite good with zigzag cuts, since it takes into account the computation of the electric field which is still sequential and also the initial step whose imbalance can be penalizing. We can see that a better load balancing is obtained when more slices are used. With 32×32 slices we reach an efficiency over 92% which is quite good considering mesh adaptation and load balancing occur every time steps.

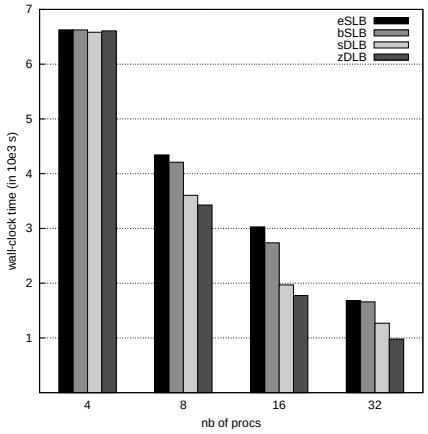

Fig. 8. Wall-clock time with different load balancing strategies.

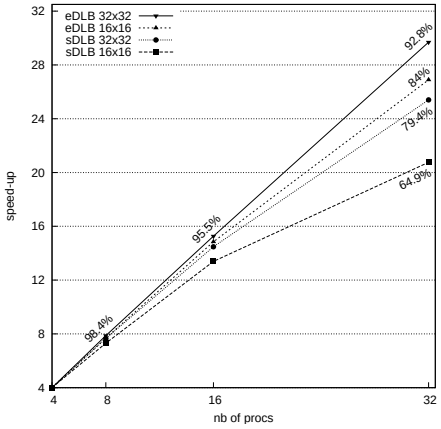

Fig. 9. Speedup and efficiency for 16×16 and 32×32 slices.

5 Conclusion

We provided our parallel algorithm presented in [9] with a mechanism to balance the workload. This dynamic load balancing mechanism is based on an improved recursive bisections and gives satisfying results up to 32 processors. This shows that an adaptive 4D Vlasov solver can achieve satisfying efficiency on distributed memory architectures in comparison with shared memory implementations.

Our mechanism may be improved on several points. We have seen that increasing the number of slices gives better load balance. However, this implies to use a smaller Δt because its value is bounded by the size of a block by assumptions on simulation parameters. If we want to use the same Δt then we could increase the number of ghost cells in x and y. This would increase the number of messages but not the volume of transferred data. We also could implement other partitioning algorithms (for example to take communication costs into account) and investigate their advantages for different test cases.

Future works will be devoted to scale the code on more processors. With the multiplication of architectures based on SMP nodes and multicore processors it becomes interesting to use both message passing and shared memory paradigm. A straightforward idea is to distribute blocks within a slice onto several processors sharing the same memory space.

References

1. M. J. Berger and S. H. Bokhari. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Transactions on Computers*, 36(5):570–580, 1987.
2. N. Besse, F.Filbet, M. Gutnic, I. Paun, and E. Sonnendrücker. An adaptive numerical method for the Vlasov equation based on a multiresolution analysis. In *Numerical Mathematics and Advanced Applications*, pages 437–446, 2001.
3. 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.
4. N. Crouseilles, M. Gutnic, G. Latu, and E. Sonnendrücker. Comparison of two eulerian solvers for the four dimensional vlasov equation (parts i and ii). In *Proc. of the 2nd international conference of Vlasovia*, volume 13(1) of *Communications in Nonlinear Science and Numerical Simulation*, pages 88–99, 2008.
5. F. Filbet. Numerical methods for the Vlasov equation. In *Numerical Mathematics and Advanced Applications*, 2001.
6. F. Filbet, E. Sonnendrücker, and P. Bertrand. Conservative numerical schemes for the Vlasov equation. *J. Comput. Phys.*, 172:166–187, 2000.
7. M. Gutnic, M. Haefele, I. Paun, and E. Sonnendrücker. Vlasov simulations on an adaptive phase-space grid. *Comput. Phys. Commun.*, 164:214–219, dec 2004.
8. M. Haefele, G. Latu, and M. Gutnic. A parallel Vlasov solver using a wavelet based adaptive mesh refinement. In *Proceedings of the 2005 International Conference on Parallel Processing Workshops (ICPPW'05)*, pages 181–188, 2005.
9. O. Hoenen and E. Violard. A block-based parallel adaptive scheme for solving the 4d vlasov equation. to appear in proceedings of the 7th Parallel Processing and Applied Mathematics Conference, Gdańsk, Poland, September 2007.
10. 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.
11. E. Sonnendrücker, J. Roche, P. Bertrand, and Alain Ghizzo. The semi-Lagrangian method for the numerical resolution of the Vlasov equation. *J. Comput. Phys.*, 149:201–220, 1999.
12. J. D. Teresco, K. D. Devine, and J. E. Flaherty. *Numerical Solution of Partial Differential Equations on Parallel Computers*, chapter Partitioning and Dynamic Load Balancing for the Numerical Solution of Partial Differential Equations. Springer-Verlag, 2005.