A Parallel Adaptive Vlasov Solver Based on Hierarchical Finite Element Interpolation

M. Mehrenberger [a,1], E. Violard [b,1], O. Hoenen [b,1],
M. Campos Pinto [c] and E. Sonnendrücker [a,1]

[a] *IRMA, Université Louis Pasteur, Strasbourg, France*
[b] *LSIIT, Université Louis Pasteur, Strasbourg, France*
[c] *LJLL, Université Pierre et Marie Curie, Paris, France*

Abstract

We present a parallel adaptive scheme for the Vlasov equation. Our method is based on a way of reducing dependencies between data, thanks to a hierarchical finite element interpolation approach. A specific data distribution pattern yields an efficient implementation. Numerical results are exhibited for a classical beam simulation in the $1D$ phase space.

PACS 29.27.a 52.65-y MSC 65Y05 82D10

Key words: Adaptive; Parallel; Vlasov; Simulation; Beam; Plasma

1 Introduction

Thanks to the rapid increase of computing power in recent years, simulations of plasmas and particle beams based on direct solution of the Vlasov equation on a multi-dimensional phase-space grid are becoming attractive as an alternative to Particle-In-Cell (PIC) simulations. Their strength lies essentially in the fact that they are noiseless and that all parts of phase space, including the tail of the distribution, are equally well resolved. Their major drawback is that, for inhomogeneous systems, many of the grid points (where no particles are present) are wasted. This is especially the case for beam simulations where the beam moves rapidly through the phase space (due to varying alternating-gradient focusing forces, for example). In order to overcome this problem, adaptive methods can be used [2,1].

[1] supported by CALVI, INRIA Lorraine project, Nancy, France

In this paper, we present an adaptive method based on bi-quadratic finite element interpolation as was first introduced in [1] and describe a new efficient parallel version of an adaptive Vlasov solver, which is also described in [3]. It turns out that adaptive numerical methods are often difficult to parallelize, because they introduce dependancies between data at different grid levels and it is then difficult to manage data locality. We have designed here a numerical method well fitted for parallelization where the underlying partitions of dyadic tensor-product cells offer a simple way to distribute data. In fact, with such a strategy, each data essentially depends on the neighbour data of the same level. After describing the numerical method in the adaptive context, we present its parallelization and exhibit an application to a classical beam simulation.

2 An adaptive resolution scheme for the Vlasov equation

Here is a brief description of the numerical method of resolution. We refer the reader to [1] for a more detailed presentation. For sake of conciseness, we give the scheme for a 2-dimensional phase space, but it generalizes to higher dimensions.

The adaptive method is based on a classical semi-Lagrangian method which takes benefit of the conservation of the distribution function along particle trajectories. This method requires to be able to localize a point in the phase-space and uses an interpolation operator. Therefore, we use a dyadic cutting of the phase space together with a finite stencil for the interpolation.
We represent our solution in a hierarchical way. Each mesh cell can be recursively subdivided into four subcells of same size. A function is approximated at a given level by a biquadratic interpolation using its value at nine nodes: the four vertices of the cell, the four midpoints of the edges and the midpoint of the cell. The biquadratic interpolation is a tensor product of quadratic 1D interpolation. Hence the interpolation procedure in 2D (or more dimensions) can be derived easily from the 1D procedure. When dealing, with the fine grid, instead of interpolating directly using the basis functions on the fine grid, we consider the three basis functions at the coarse grid and we add two others of the fine level. The basis functions for one coarse cell and its two subcells are represented in Figure 1.

The solution at time t^n is given by an adaptive mesh $\mathcal{M}^n$ consisting of a dyadic partition of cells and the level j of a given cell will vary from a coarser level j_0 to a finer level J. Going from time step t^n to t^{n+1} consists in three steps:

(1) **Prediction** of $\mathcal{M}^{n+1}$: for each cell $\alpha \in \mathcal{M}^n$, denoting j its level, compute its center c_α and the forward advected point $\mathcal{A}(c_\alpha)$ by following the characteristics of the Vlasov equation (see [1] for more details about characteristics and advection operator $\mathcal{A}$). Then add to $\mathcal{M}^{n+1}$ the unique cell $\bar{\alpha}$ of level j which fits at that place in $\mathcal{M}^{n+1}$ and all the necessary cells so that $\mathcal{M}^{n+1}$ is a dyadic adaptive mesh. Last, if $j < J$, refine $\bar{\alpha}$ of one level, that is, replace it by the 4 cells of level $j+1$ which cover the same surface.

(2) **Evaluation** : for each node a of $\mathcal{M}^{n+1}$, compute the backward advected point $\mathcal{A}^{-1}(a)$ and set $f^{n+1}(a)$ to $f^n(\mathcal{A}^{-1}(a))$: the evaluation $f^n(c)$ of the solution at any point $c \in [0,1]^2$ is obtained by searching the unique cell α of the adaptive mesh $\mathcal{M}^n$ where the point is located, using the values at the nodes of that cell and computing the local biquadratic interpolation on that cell, say $I(c, \alpha, f^n(c))$.

(3) **Compression** of $\mathcal{M}^{n+1}$: from $j = J-1$ to j_0, replace 4 cells of level $j+1$ by a cell α of level j (do the converse of refining α) when the norm of the differences $f^{n+1}(a) - I(a, \alpha, f^n(a))$, for all node a of α, is small enough.

3 Parallel implementation

The computational domain is subdivided into *regions*. A *region* is a surface of the computational domain which is defined by an union of mesh cells. Regions are allocated to processors so that each processor owns and computes the mesh cells and nodes which are included in its own *region*. As the mesh adapts to the evolution in time of the physics, the number of cells within a region change and it is then necessary to include a load balancing mechanism which then consists in redefining regions for each processor. In order to minimize communications, we apply compression within the region limit only. So the compression phase does not require any communication in our implementation. This is an approximation of the numerical method since we eliminate fewer cells than in the original method, but it does not hazard convergence. Each processor owns a local representation of the mesh. The mesh is represented by two hash tables: the cell hash table stores a set of cells which forms a partition of the whole computational domain and associates each cell with its owner identity. The node hash table stores the value at each node within the region. This representation allows cells and nodes to be accessed in constant time while minimizing the memory usage.

As said previously, our load balancing mechanism consists in redefining regions for each processor, the number of cells in each region should be approximatively the same and each region should have a "good shape" to improve the compression. Moreover, every region should be connex in order to reduce the volume of communications. We use the Hilbert's curve [4] to achieve this last requirement.

We model the global load and its localization onto the computational domain by a quad-tree [5] whose nodes are weighted by the number of leaves in the subtree. Each leaf of this quad-tree identifies one cell of the mesh and the level of a leaf in the tree is the level of the corresponding cell in the mesh.

We then build the new regions by partitioning the quad-tree. Each region is the union of the cells corresponding to the leaves of each part of the quad-tree.

To obtain a good partition, we browse the quad-tree starting from its root to its leaves, and try to make a cut as soon as possible. A part, say $\mathcal{P}$, of the partition is such that $(1-\lambda) * I \leq \|\mathcal{P}\| \leq (1+\lambda) * I$, where $\|\mathcal{P}\|$ is the number of leaves of the

part, I equals to the total number of cells divided by the number of processors, and $\lambda \in [0,1]$ is an error factor that permits a certain degree of liberty for finding good parts.

We use this method at initialization, and a less expensive version to update regions at runtime without penalizing performance.

4 Numerical results

In order to assess the benefits of the adaptive solver we computed the transverse evolution of a semi-Gaussian beam in a uniform focusing channel. For such a beam, the initial distribution function reads

$$f(r,v) = \frac{1}{\pi a^2 \sqrt{2\pi} b} e^{-\frac{1}{2}(v^2/b^2)} \text{ if } r < a,$$

and $f(r,v) = 0$ else. Here, $a = 4/\sqrt{15}$, $b = 1/(2 * \sqrt{15})$ and the time step is $1/32^{\text{th}}$ of period, that is $\Delta t = 2\pi/32$.

We have made a simulation of 5 periods (160 iterations). Our parallel code has been written in C++/MPI and tested (1) on a HP cluster, formed by 30 identical Itanium bi-processors nodes cadenced at $1.3Ghz$ with $8GBytes$ of main memory and interconnected through a switched $200MBytes/s$ and (2) on a SGI Origin 3800, composed of $R14k$ running at $500Mhz$ with $512MBytes$ per node. The results are reported on figures 2 and 3 On figure 2, the finest level is $J = 8$, which corresponds to an underlying fine grid of 512×512 points. We observe that the adaptive grid follows very well the evolution of the fine structures. Figure 3 (left) shows the graphical representation of the speed-up on a logarithmic scale. We observe that the speed-up is approximatively constant as the level of details increases. Figure 3 (right) shows that the wall-clock time keeps decreasing as the number of processors increases up to 64 processors.

5 Conclusion and future work

In this paper, we have presented a Vlasov solver based on a hierarchical finite element interpolation. The inherent good localization of the cells has permitted to give an efficient parallel implementation. We have also shown numerical results which prove that our code works very well in the $2D$ phase space case. A $4D$ parallel code is being developed.

References

[1] M. Campos Pinto, M. Mehrenberger, *Adaptive numerical resolution of the Vlasov equation*, to appear in CEMRACS 2003 proceedings, IRMA series in Mathematics and theoretical Physics (de Gruyter).

[2] M. Gutnic, M. Haefele, I. Paun, E. Sonnendrücker, *Vlasov simulations on an adaptive phase-space grid* to appear in Comput. Phys. Comm.

[3] O. Hoenen, M. Mehrenberger, E. Violard. *Parallelization of an Adaptive Vlasov Solver* to appear in ParSim04 (Special Session of EuroPVM/MPI 2004).

[4] J. K. Lawder, P. J. H. King, *Using Space-Filling Curves for Multi-dimensional Indexing*, Lecture Notes in Computer Science 1832 (2000).

[5] A. Patra, J.T. Oden, *Problem decomposition for adaptive hp finite element methods*, Computing Systems in Eng., 6 (1995).

[6] E. Sonnendrücker, F.Filbet, A. Friedman, E. Oudet, J.L. Vay *Vlasov simulation of beams on a moving phase-space grid* to appear in Comput. Phys. Comm.

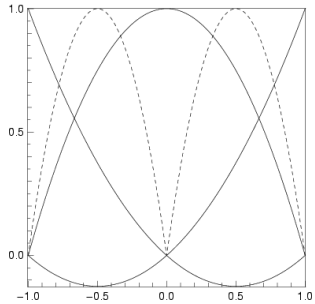

Fig. 1. Hierarchical quadratic basis functions. Basis functions on coarse cell (solid line). Additional basis functions for fine cells (dashed line).

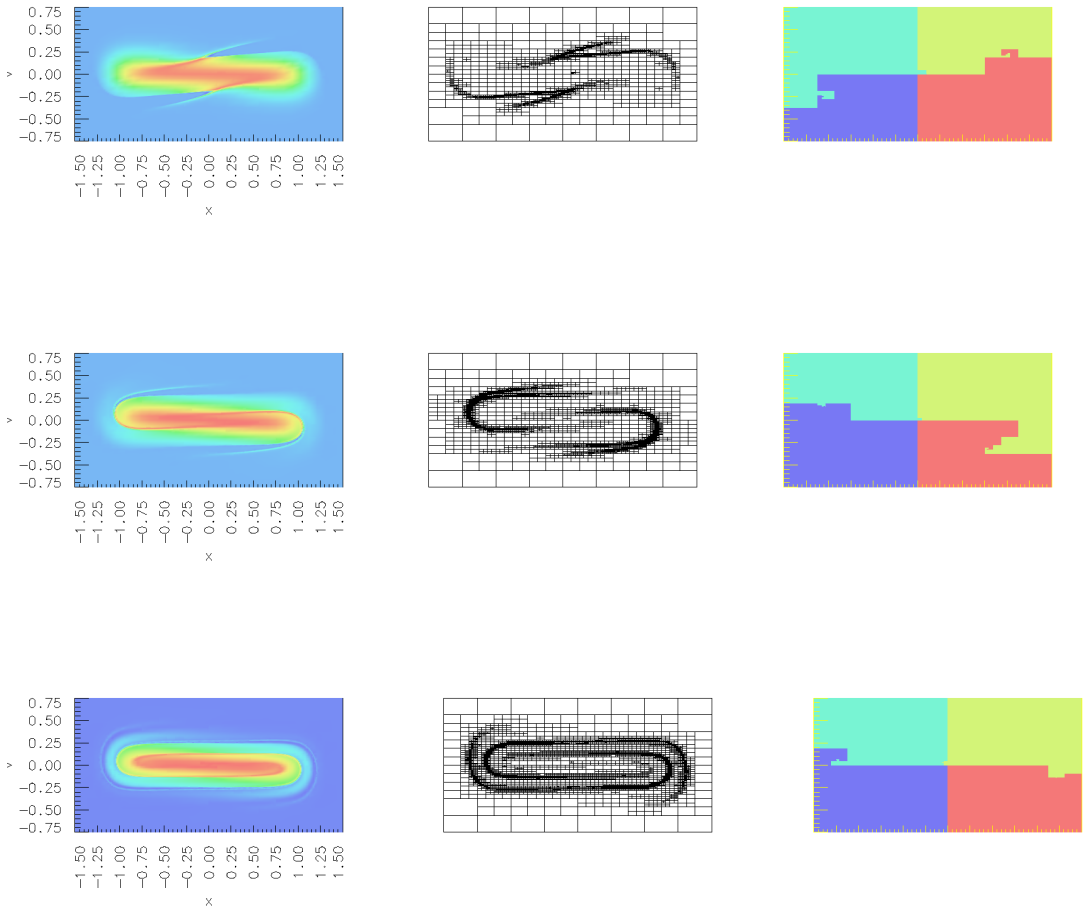

Fig. 2. evolution of the distribution function together with the grid and the regions distribution of processors after one half, one and two periods (resp. 32, 64 and 128 iterations)

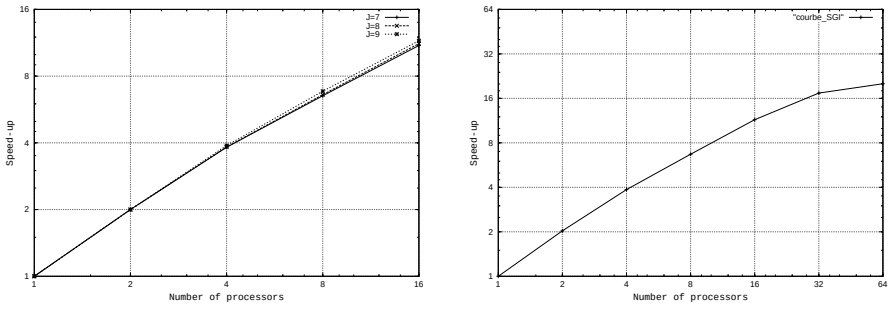

Fig. 3. speed-up on a HP cluster (left) and on a SGI O3800 (right)