Adaptive 2-D Vlasov Simulation of Particle Beams*

M. Gutnic†, M. Mehrenberger*, E. Sonnendrücker*, IRMA, Strasbourg and INRIA-CALVI Project
O. Hoenen‡, G. Latu†, E. Violard†, LSIIT Strasbourg and INRIA-CALVI Project, France.

Abstract

This paper presents our progress for the solution of the 4D Vlasov equation on a grid of the phase space, using two adaptive methods. We briefly recall the principle of the two methods and then particularly focus on computer science features - as data structures or parallelization - for the efficient implementation of the methods. Some relevant numerical results are presented.

INTRODUCTION

In order to address the noise problems occuring in Particle-In-Cell (PIC) simulations of intense particle beams, we have been investigating numerical methods based on the solution of the Vlasov equation on a grid of phase-space. However, especially for high intensity beam simulations in periodic or alternating gradient focusing fields, where particles are localized in phase space, adaptive strategies are required to get computationally efficient codes based on this method. To this aim, we have been developing fully adaptive techniques based on a hierarchical decomposition of the solution of the nonrelativistic four-dimensional Vlasov equation

$$\frac{\partial f}{\partial t} + \mathbf{v} \cdot \nabla_{\mathbf{x}} f + (\mathbf{E} + \mathbf{v} \times \mathbf{B}) \cdot \nabla_{\mathbf{v}} f = 0, \qquad (1)$$

where $\mathbf{x} = (x, y)$, $\mathbf{v} = (v_x, v_y)$ and with initial condition $f(0, \mathbf{x}, \mathbf{v}) = f_0(\mathbf{x}, \mathbf{v})$. The self-consistent electric field $\mathbf{E}$ is computed here from Poisson's equations

$$\begin{aligned} -\Delta\phi &= \rho(t,\mathbf{x}) = \int_{\mathbf{R}^d} f(t,\mathbf{x},\mathbf{v}) d\mathbf{v}, \\ E(t,\mathbf{x}) &= -\nabla_{\mathbf{x}}\phi(t,\mathbf{x}), \end{aligned} \qquad (2)$$

whereas the magnetic field is considered external and known. For the normalization of the paraxial model used in particle beam applications presented in this paper, we refer to [3].

We present here two adaptive methods for the solution of this model. The two methods are based on the semi-Lagrangian method which consists in computing directly the distribution function of particles f on a Cartesian grid of the phase space. Using the conservation property of f along the particle trajectories of the Vlasov equation, these characteristics are first solved backward at each time step and then, the distribution function is interpolated at the feet of the characteristics (see [13] for more details on the semi-Lagrangian method).

The first method we have implemented is based on quadratic finite element interpolation as was first introduced in [2] and describes a new efficient parallel version of an adaptive Vlasov solver, which is also presented in [8, 10]. It turns out that adaptive numerical methods are often difficult to parallelize, because they introduce dependencies 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. Data structure and parallel implementation, as well as relevant numerical results are presented in the second section.

In the second method, the distribution function is decomposed in a wavelet basis at different levels such that the coefficients called details are small whereas the function is regular. Therefore, according to a prediction procedure and a given threshold, only the significant details are computed. The distribution function can then be determined on the corresponding adaptive grid (see [5, 6] for more details on the method). Data structure and implementation features, as well as relevant numerical results are presented in the last section.

QUADRATIC INTERPOLATION

We consider here the extension in $4D$ of a parallel adaptive Vlasov solver presented in the $2D$ case in [8, 10]. The solution is represented in a hierarchical way: each cell of level $j \in \mathbb{N}$, which is a $4D$-cube of size 2^{-j} can be recursively divided into 16 cells of the same size and of level $j+1$. A function is here represented locally by quadratic interpolation with the 81 equidistributed nodes of a cell (the analog of the 9 nodes in $2D$), and the solution at time t^n is then given by an adaptive mesh consisting of a dyadic partition of cells and the level j of a given cell will vary from a coarse level j_0 to a fine level j_1. A central issue in this context is designing an efficient data structure to represent and handle the adaptive mesh; it is crucial that the used data-structure minimizes memory usage and access time.

Data structure

Elements of our data-structure are nodes and cells. In order to locate the nodes and the cells, we use indexing. For sake of optimization, an index is an integer whose binary

* Work partially supported by CEA, Bruyères-le-Châtel, France, contract ref.: 4600116049/P6H34.

† gutnic,mehrenbe,sonnen@math.u-strasbg.fr
‡ hoenen,latu,violard@icps.u-strasbg.fr

representation is otained by concatening strings of bits. A node index is composed with $d = 4$ strings of $j_1 + 1$ bits, each string representing the coordinate of the node along one dimension. Cell indexing reflects the cell hierarchy: a cell index is composed with j_1 strings of 4 bits, each string identifying one amongst the 16 descendants at each level of the hierarchy.

We have considered two types of data structure for storing the nodes and the cells: hash-tables and two-level arrays. The first structure is classically used [9]. We have used here the `hash_map` data structure of the Standard Template Library (STL) with perfect hash-functions avoiding collisions. Moreover, in order to improve locality, the hash function tends to associate close hashes with elements that are close in the computational domain. The second structure is an array where each element is a value or an array itself. More precisely, we have an array that stores all the nodes of coarse level j_0 and for each parent cell of level j_0 containing at least one cell of level $j \geq j_0$, we have an array which represents a uniform fine grid that stores all the nodes of the existing child cells.

Parallel implementation

The parallelization of the solver mainly relies on the distribution of the data-structure among processors. Each region corresponding to a processor should approximatively contain the same amount of cells and these cells must be connected in order to balance the workload and reduce communications. To this purpose, we use the Hilbert's space filling curve (see Figure 1 and also [11]) which is extented here to the $4D$ dyadic mesh. Therefore each cell possesses two neighbours: the previous and the next one along the Hilbert's curve, and a connected region owned by a processor corresponds to a portion of the curve. Workload balancing reduces then to redistributing data onto an homogeneous processor ring.

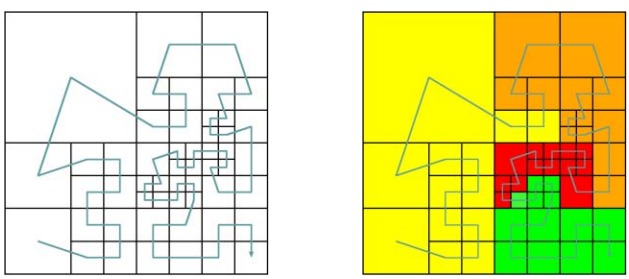

Figure 1: Hilbert curve in the $2D$ case.

Numerical results

The code was implemented in C++/MPI. Simulations have been performed on a HP cluster with 30 itanium bi-processors nodes running at 1.3 GHz and interconnected through a $2Gbits/s$ network. We consider a proton beam of current intensity $0.01A$, energy $1Mev$ and emittance $5 \cdot 10^{-4} m \cdot rad$. The initial distribution is given by the following gaussian distribution

$$f_0(x, y, v_x, v_y) = \frac{1}{(2\pi)^2} \exp\left(-\frac{x^2 + y^2 + v_x^2 + v_y^2}{2}\right).$$

The time step is $\Delta t = 0.00493$ so that one period corresponds to 64 iterations. The computational domain is $[-6, 6]^4$ with 64 points per direction.

In this simulation, only 2.6 percent of cells are used in the adaptive case, for a relative error with the uniform case (in the $x - v_x$ projection, see Figure 3) less than 3 percent. The results of the performances are shown on Table 1. We see the benefit of using the two-array data-structure; the speed-up is quite good (see Figure 2). Moreover, a comparison between uniform and adaptive simulation is shown on Figure 3.

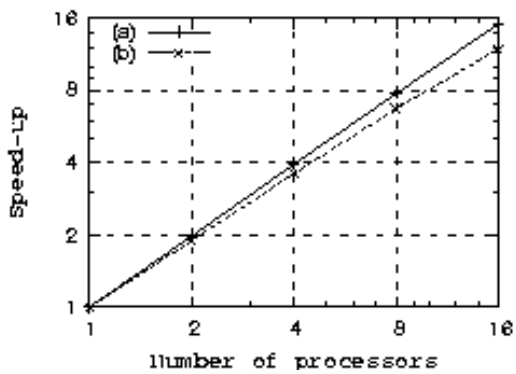

Figure 2: Performance of the parallel codes: hash-table (a), 2-level array with $j_0 = 3$ (b).

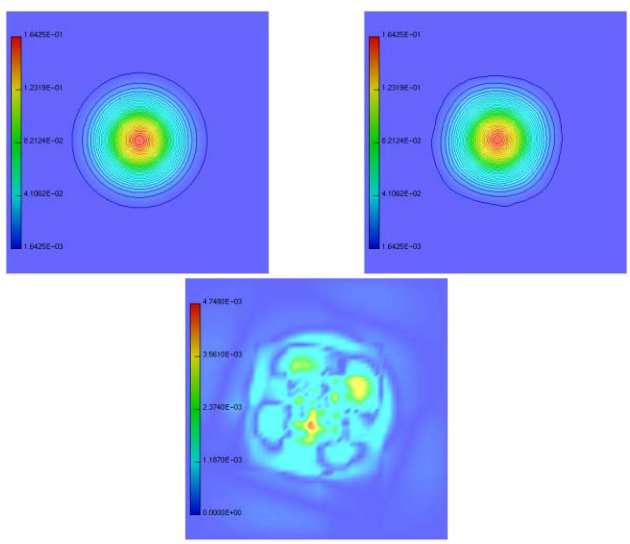

Figure 3: Projection $x - v_x$ of the Gaussian distribution after one period: uniform case (top-left), adaptive case (top-right). Absolute difference between uniform and adaptive case (bottom)

Table 1: Runtime and memory usage for 1 period.

	hash-table	2-level array		
		$j_0 = 2$	$j_0 = 3$	$j_0 = 4$
runtime (s)	1276	1088	824	888
memory use (KB)	2036	366	923	1049

WAVELET BASED INTERPOLATION

In order to develop the code efficiently in 4D, we introduce a hierarchical procedure based in particular on a hierarchical sparse data structure, which enabled us not only to reduce considerably the computation time but also the required memory.

Adaptive data structure

We have implemented a wavelet compression scheme that takes as input a 4D distribution function. Wavelet compression is a well-known approach to data reduction. Thus lossy compression can be achieved by only storing the important coefficients. The basic principle is thresholding small coefficients to zero. For basis functions with good interpolating properties, many coefficients can be dropped without degrading data quality.

For our four-dimensional data we construct the basis functions as the tensor product of one-dimensional wavelet and scaling functions. This corresponds to the different combinations of the scaling function along the coordinate directions. Let $\mathcal{I}_4 = \{\alpha = (\alpha_1, \alpha_2, \alpha_3, \alpha_4) \in \{0,1\}^4\}$, the 4D hierarchical approximation from a coarser level j_0 to a finer level j_1 of the distribution function f reads

$$\begin{aligned} f(\mathbf{x},\mathbf{v}) &= \sum_{\kappa}^{j_1} c_\kappa^{j_0,0}\, \phi_\kappa^{j_0,0}(\mathbf{x},\mathbf{v}) \\ &+ \sum_{j=j_0}^{j_1} \sum_{\kappa} \sum_{\substack{\alpha\in\mathcal{I} \\ \alpha\neq 0}} d_\kappa^{j,\alpha}\, \phi_\kappa^{j,\alpha}(\mathbf{x},\mathbf{v}), \end{aligned} \tag{3}$$

where $\phi_\kappa^{j,\alpha}(\mathbf{x},\mathbf{v}) = \varphi_{k_1}^{j,\alpha_1}(x)\,\varphi_{k_2}^{j,\alpha_2}(y)\,\varphi_{k_3}^{j,\alpha_3}(v_x)\,\varphi_{k_4}^{j,\alpha_4}(v_y)$ with $\varphi_k^{j,0}(\cdot) = \varphi(2^j\cdot - k)$ and $\varphi_k^{j,1}(\cdot) = \varphi(2^j(2\cdot -1) - k)$ for any k and j, φ denoting the scaling function at the coarsest level j_0. Compression is then performed by keeping only the significant details $d_\kappa^{j,\alpha}$, also called wavelet coefficients, in (3).

To build the corresponding adaptive phase-space grid and organize the 4D wavelet coefficients, we use a hierarchical data structure named a tree. The wavelet coefficients could be stored in a space-partioning hexadeca-tree structure [1]. The design goal of this tree is to support both a quick access to wavelet coefficient at each scale (especially for the finest scale that contains usually much of the non-zero coefficients) and to have a compact memory representation. On analogy with binary-tree, quad-tree and octree that are used to partition 1D, 2D and 3D spaces, we use a hexadeca-tree to store the 4D wavelet decomposition. Each node of the tree, at level j_0, contains the distribution value $c_\kappa^{j0,0}$ (a double precision real number) and links towards its 16 children nodes responsible for details $d_\kappa^{j0,\alpha}$ and its descendants. Each node, at level $j > j_0$ of the tree, owns the detail $d_\kappa^{j,\alpha}$ and has links towards its descendant nodes $d_{2\kappa+\alpha}^{j+1,\beta}$ with $\beta \in \mathcal{I}$.

To construct the initial tree in our simulation, a wavelet transform is performed. First, we detect the collection of wavelet coefficients that should be stored depending on a threshold choosen by the user. Then, we check the coherency of the tree, adding wavelet coefficients where needed in order to get a well-built wavelet decomposition. This produces a 4D hierarchy that corresponds to a complete hexadeca-tree. Each node of the tree represents a 4D subvolume at a given resolution. We reduce the memory usage of the data-structure by pruning the tree when a child does not exist. Because these nodes are not needed to reconstruct the original function, they can be removed from the tree.

Implementation

The adaptivity allows us to significantly reduce the memory size needed as well as the algorithmic complexity of computation. Nevertheless, the choice of an appropriate data structure is a keypoint for optimize computation time and a compact representation in memory. To this purpose, we used initially a sparse data structure based on two levels of dense arrays (not the hierarchical structure previously introduced). In this structure, the first array contains all the grid points up to some intermediate level j_{limit}. The second array which is allocated where needed contains all the grid points from this level j_{limit} up to the finest level j_1. Finally, all grid points can be accessed with at most one indirection pointer. In the 2D case, the computing time decreased by a factor of 3 with the use of this structure (see [12]). This sparse data structure is also very useful in the purpose of parallelization thanks to data locality (see [4] for more details). In spite of very good performances for test cases on grid sizes up to 128^4, this kind of data structure wastes memory on bigger test cases because arrays of the second level contains few non-zero coefficients and many zeros.

To keep the memory occupancy at a minimum, another data structure was used. The direct usage of hexadeca-trees leads to the problem of many indirection pointers to access coefficient at the finest level. Then, we choose an alternate implementation of the hexadeca-tree to reduce the number of indirection to retrieve any coefficient. A tree containing nodes with 256 children were used. Each node encapsulates two levels of the hexadeca-tree. The node structure is dynamic depending on the existence of non-zero coefficients and children. This new data structure allows us to reduce the number of indirection to cross in order to read or write a wavelet coefficient. At the same time, the representation is compact and is adaptive depending on the sparsity of the data to store.

Numerical results

The code was implemented in C/OpenMP. Simulations have been performed on an IBM cluster with 11 SMP nodes with 16 processors per node running at 1.5Ghz and interconnected through a Federation Switch. We consider a Potassium beam with current intensity 40 mA and energy 1 MeV in alternating gradient lattice. The initial distribution is given by the following gaussian distribution

$$f_0(\mathbf{x},\mathbf{v}) = \exp\left(-\frac{x^2+y^2+v_x^2+v_y^2}{2}\right).$$

The time step is $\Delta t = 0.000464$, so that one period corresponds to 128 iterations. The computational domain is $[-5.1,5.1]^2 \times [-24,24]^2$ with 128 points per direction.

In Figure 4, we show that the number of points is less than 10 percent, with a relative error compared to the uniform solution obtained by a cubic spline interpolation less than 0.5 percent (see 5).

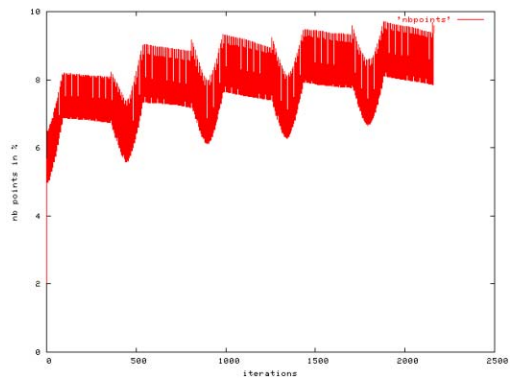

Figure 4: Number of points kept in the adaptive simulation.

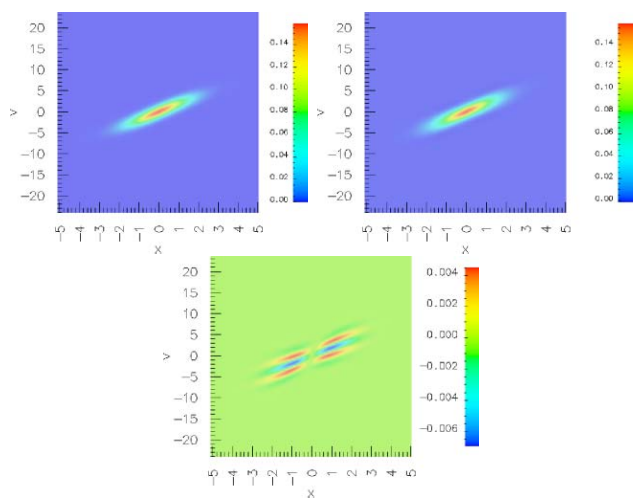

Figure 5: Projection $x - v_x$ of the Gaussian distribution after two and a half period: uniform case (top-left), adaptive case (top-right). Absolute difference between uniform and adaptive case (bottom).

Moreover, we have computed the X_{rms} (Root Mean Square) quantity given by the square root of

$$\int_{\mathbf{R}^4} x^2 f(\mathbf{x},\mathbf{v})\, d\mathbf{x}\, d\mathbf{v},$$

which can be done directly from formula (3) (see [7]). Figure 6 compares the behaviour in time of the X_{rms} quantity for the ideal KV-distribution and in the adaptive case for 128 and 256 points in each direction. Finally, the efficiency of parallelization is shown in Table 2 which gives the computational time as well as the speedup for the adaptive method.

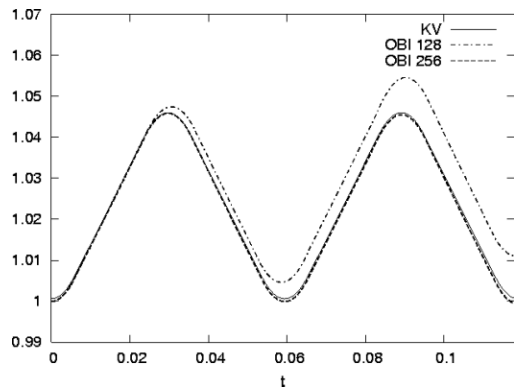

Figure 6: X_{rms} quantity for the ideal KV-distribution and in the adaptive cases for 128^4 and 256^4 points.

Table 2: Computational time and speedup for 1 iteration of the adaptive simulation.

Numbers of processors	1	2	4	8
Time (in s.)	408	206	105	55
Speedup	1	1.98	3.88	7.41

CONCLUSION

We have been developing for a few years now adaptive grid based method for the numerical simulation of intense particle beams. Both strategies we pursued, hierarchical quadratic finite element interpolants and interpolating wavelet based interpolants, are now working in the four-dimensional phase space enabling us to perform realistic simulations of long beams in the transverse plane. In order to be efficient, such methods need sophisticated data structures and optimization on parallel computers. This being done they can allow very precise noise free computations. Our next step will be to benchmark these codes in realistic accelerator configurations and compare their features to those of PIC codes.

Aknowledgement The authors are particularly grateful to Jean-Louis Lemaire of C.E.A. (Bruyères-le-Châtel, France) for the fruitful discussions and the support given by C.E.A.

REFERENCES

[1] L. AHRENBERG, I. IHRKE, M. MAGNOR, *Volumetric Reconstruction, Compression and Rendering of Natural Phenomena from Multi-Video Data* Volume graphics 2005 : Eurographics/IEEE VGTC workshop proceedings; Fourth International Workshop on Volume Graphics, pp. 83–90 (2005).

[2] M. CAMPOS PINTO, M. MEHRENBERGER, *Adaptive numerical resolution of the Vlasov equation*, in Numerical Methods for Hyperbolic and Kinetic Problems, Proceedings of Cemracs 2003, IRMA Lect. Math. Theor. Phys., **7**, Eur. Math. Soc., Zrich, S. Cordier, T. Goudon, M. Gutnic, E. Sonnendrcker eds, European Mathematical Society, pp. 43–58 (2005).

[3] F. FILBET, E. SONNENDRÜCKER, *Modeling and numerical simulation of space charge dominated beams in the paraxial approximation*, Math. Mod. Meth. App. Sc., **16**, pp.1-29, (2006).

[4] M. GUTNIC, M. HAEFELE, G. LATU, *A parallel Vlasov solver using a wavelet based adaptive mesh refinement*, in 2005 International Conference on Parallel Processing (ICPP'2005), 7th Workshop on High Perf. Scientific and Engineering Computing, IEEE Computer Society Press, pp. 181–188 (2005).

[5] M. GUTNIC, M. HAEFELE, I. PAUN, E. SONNENDRÜCKER, *Vlasov simulation on an adaptive phase space grid*, Comput. Phys. Comm., **164**, pp. 214–219 (2004).

[6] M. GUTNIC, M. HAEFELE, E. SONNENDRÜCKER, *Moments conservation in adaptive Vlasov solver*, Nuclear Instruments and Methods in Physics Research Section A, Proceedings of the 8th International Computational Accelerator Physics Conference - ICAP 2004, **558, Issue 1**, pp. 159–162 (2006).

[7] M. GUTNIC, G. LATU, E. SONNENDRÜCKER, *Adaptive two-dimensional Vlasov simulation of heavy ion beams*, to appear in HIF 2006 proceedings.

[8] O. HOENEN, M. MEHRENBERGER, E. VIOLARD, *Parallelization of an Adaptive Vlasov Solver*, Lecture Notes in Computer Science, **3241**, pp. 430–436 (2004).

[9] D. KNUTH, *The art of computer programming: Sorting and Searching*, Addison-Wesley, **3** (1973).

[10] M. MEHRENBERGER, E. VIOLARD, O. HOENEN, M. CAMPOS PINTO AND E. SONNENDRÜCKER, *A Parallel Adaptive Vlasov Solver Based on Hierarchical Finite Element Interpolation, Moments conservation in adaptive Vlasov solver*, Nuclear Instruments and Methods in Physics Research Section A, Proceedings of the 8th International Computational Accelerator Physics Conference - ICAP 2004, **558, Issue 1**, pp. 188–192 (2006).

[11] H. SAGAN, *Space Filling Curves*, Springer (1994).

[12] E. SONNENDRCKER, M. GUTNIC, M. HAEFELE, G. LATU, *Vlasov simulations of beams and halo*, PAC 2005 proceedings (CD-ROM).

[13] E. SONNENDRÜCKER, J. ROCHE, P. BERTRAND, A. GHIZZO, *The semi-Lagrangian method for the numerical resolution of the Vlasov equations*, J. Comput. Phys., **149**, pp. 201–220 (1999).