Parallelization of an Adaptive Vlasov Solver

Olivier Hoenen[1], Michel Mehrenberger[2], and Éric Violard[1]

[1] Université Louis Pasteur, Laboratoire LSIIT, Groupe ICPS,
Boulevard Sébastien Brant, F-67400 Illkirch, France
{hoenen,violard}@icps.u-strasbg.fr
[2] Université Louis Pasteur, Laboratoire IRMA,
7 rue René Descartes, F-67084 Strasbourg, France
mehrenbe@math.u-strasbg.fr

Abstract. This paper presents an efficient parallel implementation of a Vlasov solver. Our implementation is based on an adaptive numerical scheme of resolution. The underlying numerical method uses a dyadic mesh which is particularly well suited to manage data locality. We have developed an adapted data distribution pattern based on a division of the computational domain into regions and integrated a load balancing mechanism which periodically redefines regions to follow the evolution of the adaptive mesh. Experimental results show the good efficiency of our code and confirm the adequacy of our implementation choices. This work is a part of the CALVI project[1].

1 Introduction

The Vlasov equation (see e.g. [9] for its mathematical expression) describes the evolution of a system of particles under the effects of self-consistent electromagnetic fields. Most Vlasov solvers in use today are based on the Particle In Cell method which consists in solving the Vlasov equation with a gridless particle method coupled with a grid based field solver (see e.g. [2]). For some problems in plasma physics or beam physics, particle methods are too noisy and it is of advantage to solve the Vlasov equation on a grid of phase space, i.e., the position and velocity space $(x, v) \in \mathbb{R}^d \times \mathbb{R}^d, d = 1, .., 3$. This has proven very efficient on uniform meshes in the two-dimensional phase space (for $d = 1$). However when the dimensionality increases the number of points on a uniform grid becomes too important for being performed on a single computer. So some parallelized versions had been developed (see e.g. [10], for $4D$ phase space Vlasov simulations) and it is essential to regain optimality by keeping only the 'necessary' grid points. Such adaptive methods have recently been developed, like in [8],[5],[3] where the authors use moving distribution function grids, interpolatory wavelets of Deslaurier and Dubuc or hierarchical biquadratic finite elements. We refer also to [4] for a summary of many Vlasov solvers.

[1] CALVI is a french INRIA project devoted to the numerical simulation of problems in Plasma Physics and beams propagation.

D. Kranzlmüller et al. (Eds.): EuroPVM/MPI 2004, LNCS 3241, pp. 430–435, 2004.

In this project, we had in mind to implement an efficient parallelized version of an adaptive Vlasov solver. So we have developed a code based on [3], where the underlying partitions of dyadic tensor-product cells offered a simple way to distribute data. After describing the numerical method in the adaptive context, we present a parallelization of this method and its mechanism of load balancing, and exhibit numerical results.

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 numerical solution at time $t^n = n\Delta t$ is represented by the approximate $f^n(a)$ of the solution at every nodes a of a *dyadic adaptive mesh* $\mathcal{M}^n$. A dyadic adaptive mesh forms a possibly non-uniform partition of the phase space: considering the unit square $[0,1]^2$ as the computational domain, each cell of the mesh identifies an elementary surface $[k\,2^{-j}, (k+1)\,2^{-j}] \times [l\,2^{-j}, (l+1)\,2^{-j}]$, where $k, l \in \mathbb{N}$, and $j \in \mathbb{N}$ is the level of the cell. We have $j_0 \leq j \leq J$, where j_0 and J stand for the coarsest and finest level of discretization. Each of the cell has 9 uniformly distributed nodes. Then, going to the next time step 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.

Some other methods represent the solution with less nodes and give a procedure to retrieve the other nodes by computation. This is done for example in the numerical methods which use a wavelet decomposition framework [5] and where the compression phase deletes nodes instead of cells. On the contrary, our method keeps all the nodes, this may loose adaptivity but improve data locality. Moreover, when the global movement of particles is known, our method can easily be extended to the concept of moving grid presented in [8].

3 Parallel Implementation

Extraction of Parallelism. The numerical method induces a data-parallel algorithm by considering the adaptive mesh as a parallel data structure whose elements are the cells of the mesh with their associated nodes and values. The parallelization then relies on distributing these elements among processors.

Data Distribution. 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.

Communications. We implement a specific communication scheme in order to overlap communications with computations during the prediction and evaluation phases: as the number and the source of messages is not known *a priori*, a special *end-of-send* message is used to stop the initialization of receives. Moreover, in order to minimize communications, we apply compression within the region limit only. So the compression phase do not require any communication in our implementation. This is an approximation of the numerical method since we eliminate less cells than in the original method, but it does not hazard convergence.

Data Structure. 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.

Load Balancing. As said previously, our load balancing mechanism consists in redefining regions. The new regions should have the following characteristics: 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 [6] to achieve this last requirement.

We model the global load and its localization onto the computational domain by a quad-tree [7] 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

Table 1. Elapsed time (s) and **speed-up** on a HP cluster

# procs	$J = 7$		$J = 8$		$J = 9$	
1	1089	**1**	1896	**1**	3202	**1**
2	543	**2**	937	**2**	1559	**2**
4	285	**3.82**	494	**3.83**	823	**3.89**
8	167	**6.52**	287	**6.6**	468	**6.84**
16	99	**11**	169	**11.21**	277	**11.55**

Table 2. Elapsed time (s) and **speed-up** on a SGI O3800

# procs	$J = 6$		$J = 7$		$J = 8$	
1	1055	**1**	1827	**1**	3074	**1**
2	527	**2**	908	**2**	1514	**2.03**
4	275	**3.83**	475	**3.84**	797	**3.86**
8	155	**6.8**	274	**6.66**	459	**6.70**
16	89	**11.85**	161	**11.34**	268	**11.47**
32	57	**18.5**	105	**17.4**	177	**17.37**

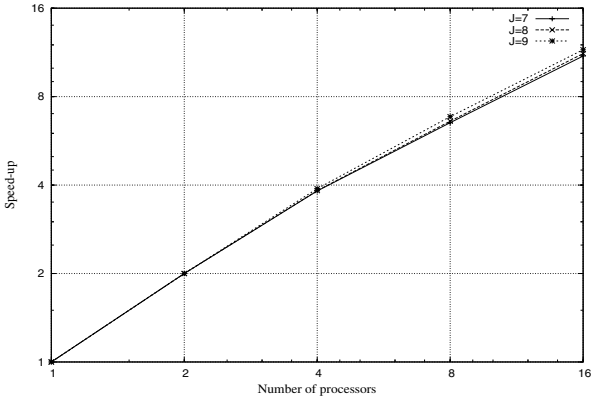

Fig. 1. Speed-up of the parallel code on a HP cluster

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

Our parallel code has been written in C++/MPI and tested (1) on a HP cluster, composed of 30 identical Itanium bi-processors nodes running at $1.3Ghz$, with

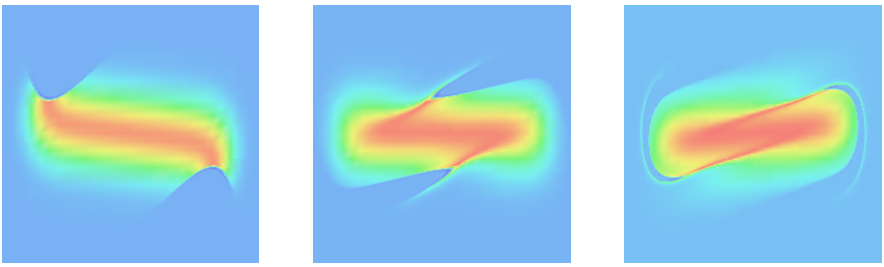

Fig. 2. Evolution of the particle beam in the phase space

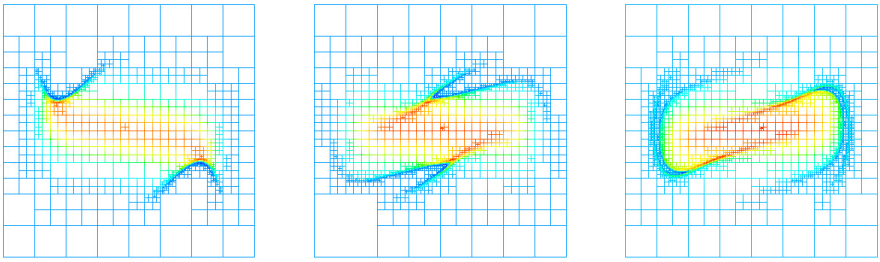

Fig. 3. Evolution of the dyadic mesh

$8GBytes$ of main memory and interconnected through a switched $200MBytes/s$ network and (2) on a SGI Origin 3800, composed of $R14k$ processors running at $500Mhz$, with $512MBytes$ of memory per node. Our test case is a 30 sec simulation (i.e. 160 iterations) of a semi-Gaussian beam in uniform applied electric field. The Vlasov equation is solved in a 2D phase space. We measured the wall-clock time for different values of the mesh finest level (J) and $j_0 = 3$. The results are reported on table 1 and 2.

Figure 1 shows the graphical representation on a log scale of the speed-up on the HP cluster. We observe that, for a fixed number of processors, the speed-up is approximatively constant as the level of details (J) increases which is a quite good property of our code. Table 1 and 2 show that the speed-up is approximately the same for two different parallel architectures.

5 Conclusion and Future Work

In this paper, we presented an efficient parallel implementation of a Vlasov solver using a numerical method based on a dyadic adaptive mesh. Numerical results show the good efficiency of our code for the $2D$ case. We still have some optimizations to implement – grouping multiple sends for example, and we are currently working on extending the load balancing mechanism for greater dimensions. Then, we plan to target the computational grid as execution environment, which will imply new scheduling and data locality constraints to deal with.

Fig. 4. Evolution of the regions for 8 processors

References

1. N. Besse, *Convergence of a semi-Lagrangian scheme for the one-dimensional Vlasov-Poisson system* , SIAM J. Numer. Anal., Vol 42, (2004), pp. 350-382.
2. C. K. Birdshall, A.B. Langdon, *Plasmaphysics via computer simulation* , McGraw-Hill, 1985.
3. M. Campos-Pinto, M. Mehrenberger, *Adaptive numerical resolution of the Vlasov equation* submitted in Numerical methods for hyperbolic and kinetic problems.
4. F. Filbet, *Numerical Methods for the Vlasov equation* ENUMATH'01 Proceedings.
5. M. Gutnic, Ioana Paun, E. Sonnendrücker, *Vlasov simulations on an adaptive phase-space grid* to appear in Comput. Phys. Comm.
6. J. K. Lawder, P. J. H. King, *Using Space-Filling Curves for Multi-dimensional Indexing*, Lecture Notes in Computer Science 1832 (2000).
7. A. Patra, J.T. Oden, *Problem decomposition for adaptive hp finite element methods*, Computing Systems in Eng., 6 (1995).
8. 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.
9. E. Sonnendrücker, J. Roche, P. Bertrand and A. Ghizzo *The Semi-Lagrangian Method for the Numerical Resolution of Vlasov Equations.* J. Comput. Phys. 149(1998), pp. 201-220.
10. E. Violard, F. Filbet *Parallelization of a Vlasov Solver by Communication Overlapping*, Proceedings PDPTA 2002 (2002).