A Parallel Vlasov solver using a Wavelet based Adaptive Mesh Refinement

Matthieu Haefele, Guillaume Latu and Michael Gutnic
INRIA CALVI project (`http://math.u-strasbg.fr/calvi`)
LSIIT, UMR CNRS 7005 & IRMA, UMR CNRS 7501
Université Louis Pasteur, 7 rue Descartes Strasbourg, France
{haefele | latu}@lsiit.u-strasbg.fr, gutnic@math.u-strasbg.fr

Reference: Paper accepted in the 7th Workshop on High Performance Scientific and Engineering Computing (ICPP-HPSEC-05).

Abstract

We are interested in solving the Vlasov equation used to describe collective effects in plasmas. This non-linear partial differential equation coupled with Maxwell equation describes the time evolution of the particle distribution in phase space. The numerical solution of the full three-dimensional Vlasov-Maxwell system represents a considerable challenge due to the huge size of the problem. A numerical method based on wavelet transform enables to compute the distribution function on an adaptive mesh from a regular discretization of the phase space. In this paper, we evaluate the costs of this recently developed adaptive scheme applied on a reduced one-dimensional model, and its parallelization. We got a fine grain parallel application that achieves a good scalability up to 64 processors on a shared memory architecture.

1 INTRODUCTION

In the quest for a new source of energy, understanding plasma behavior is one of the most challenging problems to overcome. Plasma can be considered as the fourth state of matter and exists at huge temperature conditions (10^4 K or more). These conditions are reached in different facilities, in particular in tokamak reactors. A kinetic description is used to model such phenomenon. The plasma is governed by the Vlasov equation coupled with Poisson or Maxwell equations to evaluate the self-consistent fields generated by the particles. Vlasov equation

$$\frac{\partial f}{\partial t} + \vec{v}\,.\nabla_x f + (E + \vec{v} \times B)\,.\nabla_v f = 0, \qquad (1)$$

characterizes the evolution of particle distribution in time and space according to the electro-magnetic fields $E(\vec{x},t)$ and $B(\vec{x},t)$. The distribution function $f(\vec{x},\vec{v},t)$ represents the particle density at a time t and a point $(\vec{x},\vec{v})$ in phase space. In phase space, a point is characterized by a position vector $\vec{x}$ and a velocity vector $\vec{v}$. To describe numerous cases, one need $(\vec{x},\vec{v}) \in \mathbb{R}^d \times \mathbb{R}^d$ with $d = 3$.

Finding an approximation of this non-linear partial differential equation enables the simulation of new accelerator and tokamak designs to validate them before building the devices. For a more fundamental purpose, finding such an approximate solution makes it possible for physicists to represent the behavior of different physical parameter sets.

Two major kinds of numerical methods exist to find an approximate solution of the Vlasov equation. The Particle In Cell method (PIC) [1] follows a large number of particles ($\simeq 10^9$) initially distributed randomly in phase space, whereas the semi-langrangian method approximates the particle distribution function on a uniform mesh of the phase space [10]. Although mathematicians have already studied wavelet transforms and adaptive numerical scheme, coupling wavelets and non-linear partial differential equations approximation represents a great deal of interest. The wavelet decomposition gives a sparse representation and a natural criterion to perform local grid refinements. The parallelization of such method [9, 5] is interesting in order to deal with applications that manipulate large data with possibly many dimensions. The present work focuses on the parallelization of an adaptive semi-lagrangian code which considers a single physical dimension. This work corresponds to a first step in the design of a new

parallel Vlasov solver.

These current researches are performed in an interdisciplinary approach within the INRIA CALVI project: physics and mathematics with Nancy 1 University, mathematics and computer science with Strasbourg 1 University.

The section 2 focuses on the numerical scheme. Section 3 describes sequential algorithms of the simulator, their costs and the data structures we used. Section 4 depicts the parallelization. Section 5 deals with performance analysis and is followed by the results and a conclusion in section 6 and 7.

2 NUMERICAL SCHEME

In the present work, we consider a reduced model for only one physical dimension ($d = 1$), corresponding to x and v such as $(x, v) \in \mathbb{R}^2$. Moreover, the self consistent magnetic field is neglected because v is considered to be small in this particular case. The reduced 1D Vlasov-Poisson system then reads:

$$\frac{\partial f}{\partial t} + v \,.\frac{\partial f}{\partial x} + E(x,t) \,.\frac{\partial f}{\partial v} = 0\,, \qquad (2)$$

$$\frac{dE}{dx} = \frac{1}{\varepsilon_0}\rho(x,t) = \frac{q}{\varepsilon_0}\int f(x,v,t)dv\,, \qquad (3)$$

where ρ is the charge density, q the charge of a particle and ε_0 the vacuum permittivity.

Equations (2) and (3) are solved successively at each time step. Equation (3) gives the self-consistent electrostatic field $E(x,t)$ generated by particles. It is evaluated with two integral computations of f. Our work focuses on the resolution of equation (2) using an adaptive backward semi-lagrangian method.

The principle of the semi-lagrangian method is to compute the value of the distribution function f on a grid of the phase space using the property that f is constant along particular phase space curves called characteristics (Fig. 1).

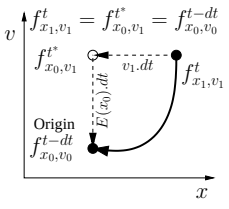

Figure 1. Characteristic curve and backward advection in time

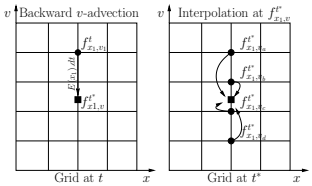

Figure 2. Semi-lagrangian method

For each grid points at time t, characteristics are followed backward in time. Then, value of f at a given grid point is interpolated at the reached point (origin of the characteristic) using values of f in the neighbourhood at the previous time step $t - dt$.

In fact, a time splitting procedure [2] is used. One simulation time step is then split into two steps: a displacement in v-direction, followed by a displacement in x-direction. First, the semi-lagrangian procedure is used to compute an intermediate distribution function f^{t^*}: at constant x, each grid point is shifted with quantity $E(x,t).dt$ in the v-direction. After this backward v-advection, the value at the reached point is evaluated by using points in its neighbourhood and a 1D interpolation (Fig. 1 and 2). Then, a similar backward advection is done with quantity $v.dt$ in the x-direction followed by a 1D interpolation.

The major drawback of Vlasov methods using uniform meshes is that memory consumption and computation cost increase drastically when the number of dimensions d, and grid size increase. An adaptive algorithm was developed to address this problem. Adaptivity is then performed by using a wavelet framework [3, 4]. The main idea is to represent the distribution function at different levels of resolution j (with $j = 0, \ldots, L_{max}$) using a wavelet decomposition. The function is then represented by a coarse mesh ($j = 0$) and a set of details for each level of refinement ($j = 1, \ldots, L_{max}$). In short, these details characterize the information loss induced by the coarse representation of the distribution function at level $j = 0$. In our interpolets framework, the detail is equal to the difference between the real value of the distribution function and an interpolated value. The latter is given at a point at level j by the Lagrange polynomial built from values at coarser level $j - 1$ (Fig. 3). Therefore, we have a natural criterion of compression since the detail is small where the function is regular. Compression of the wavelet decomposition is then performed by removing grid points with detail smaller than a given threshold ϵ.

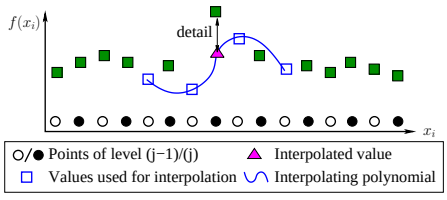

Figure 3. Computation of a detail in a 1D wavelet decomposition at level j

This adaptive framework needs two more steps in the numerical scheme. At each time-splitting step t, a set of points A_t is built from the wavelet decompo-

sition C_{t-dt} obtained after compression (we will refer to this step as the prediction step). The distribution function F_t is then computed on this sparse structure A_t using the backward semi-lagrangian method. Then the wavelet decomposition of the new distribution function F_t is computed and compressed providing the new adaptive grid C_t. Note that the wavelet transform algorithm requires a well formed tree of wavelet coefficients on which the distribution value is known (Fig. 4) that will be taken into account after the prediction step. In our case, the 2D wavelet decomposition is the tensor product of two 1D decompositions in x and v. More details on this numerical scheme can be found in [6].

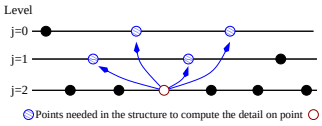

Figure 4. Tree structure of wavelet coefficients

1. Read input parameters;
2. Compute initial conditions: electrostatic field, wavelet decomposition of the distribution function;
3. For all time steps t required;
   - 3A. Splitting in v-direction;
     - 3A.1 Prediction step in v-direction.;$(C^x_{t-dt} \rightarrow A^v_{t^*})$
     - 3A.2 Make a well formed tree;$(A^v_{t^*} \rightarrow A^v_{t^*})$
     - 3A.3 Inverse wavelet transform and (C^x_{t-dt}) backward advection in v-direction.;$(A^v_{t^*} \rightarrow F^v_{t^*})$
     - 3A.4 Adaptive wavelet transform;$(F^v_{t^*} \rightarrow C^v_{t^*})$
   - 3B. Splitting in x-direction;
     - 3B.1 Prediction step in x-direction.;$(C^v_{t^*} \rightarrow A^x_t)$
     - 3B.2 Make a well formed tree;$(A^x_t \rightarrow A^x_t)$
     - 3B.3 Inverse wavelet transform and $(C^v_{t^*})$ backward advection in x-direction.;$(A^x_t \rightarrow F^x_t)$
     - 3B.4 Adaptive wavelet transform;$(F^x_t \rightarrow C^x_t)$
   - 3C. Compute the field with the wavelet coefficients;

   End for

Figure 5. Global algorithm of the simulator

3 SEQUENTIAL ANALYSIS

3.1 Algorithms and Complexities

A first sequential simulator, implementing this adaptive numerical scheme was developed. Let assume that the discretization is a $N_x \times N_v$ grid, where N_x and N_v are taken equal to N for the sake of simplicity.

This algorithm allows to advect effectively only a percentage p of the whole uniform discretization of the phase space. For small values of p ($p \ll 1$), we reduce the total computation cost by advecting only $p\,N^2$ points instead of N^2 [6]. Nevertheless, the management of wavelet coefficients for each advection represents an overhead.

Figure 5 shows the global algorithm of the simulator. For one time step t, we will present key points of the different steps and their associated complexities. We will refer to: $A^v_{t^*}$ (and A^x_t) as the set of points where we want to compute the wavelet coefficients for the splitting in v (respectively for the splitting in x), $C^v_{t^*}$ (and C^x_t) as the set of wavelet coefficients, $F^v_{t^*}$ (and F^x_t) as the set of points where the distribution function f is known.

The prediction step (3A.1) considers each wavelet coefficients of the set C^x_{t-dt} generated at previous time step to create a list of advected points $A^v_{t^*}$ (translated at constant x) where we want to compute the wavelet coefficient at time t^*. Automatically, eight points are added around each translated point in order to capture new physical phenomena that could appear locally.

The following step (3A.2) adds points to $A^v_{t^*}$ in order to have a usable data structure for the next adaptive wavelet transform step (3A.4). Indeed, a well formed tree of wavelet coefficients will be needed (see Fig. 4). For each point at a given level j, several points are added in its neighbourhood at level j and $j-1$. The position and the number of these higher level points are determined by a specific filter which characterizes the used wavelet.

The step (3A.3) reconstructs the whole distribution function at previous time step $t - dt$ using wavelet coefficients C^x_{t-dt}. The principle of the algorithm is a traversal of each level from the coarsest to the finest in order to compute iteratively the values of the distribution. The complexity of this algorithm in memory accesses and in computation is several times N^2.
This algorithm was transformed to get a block decomposition in order to benefit from cache memory effects. The blocks we consider, consist in a set of rows or columns of the phase space (see section 4 for more details on block decomposition). This method induces an overhead due to redundancy in the computation of points located at the boundaries of the blocks. Nevertheless, this method is interesting because the block size could be chosen to fit in the cache memory. Then, wavelet coefficients are read only a few times, so the number of effective memory accesses decreases thanks

to the cache use. Thus, the complexity in memory readings is reduced to $\Theta(\#C^x_{t-dt})$, whereas in computation we keep the complexity $\Theta(N^2)$.
For each computed block, the semi-lagrangian principle is applied: every points P in $A^v_{t^*}$ are translated backward in the v-direction to find the origin of the characteristic O. As the distribution values at P and O are identical, P is set to the value of O which is interpolated using the whole distribution function at $t - dt$. The computed values are then stored in $F^v_{t^*}$. Complexity in writing is in $\Theta(\#A^v_{t^*})$.

The next step (3A.4) considers traversals from finest level L_{max} down to the level 0 for all points in $F^v_{t^*}$. For each point of level j, the detail is evaluated by convolving the wavelet filter with a small set of points of level $j - 1$. Details greater than ϵ are stored in $C^v_{t^*}$. Complexity in computation and memory accesses is $\Theta(\#F^v_{t^*})$, *i.e.* the number of details computed at time t^* for the splitting in v-direction.

The splitting in x-direction (step 3B) is almost identical to the previous splitting. The only difference consists in translating points in the other dimension.

In the final step (3C), we integrate the distribution function F^x_t over v-dimension, to evaluate for each x the charge density ρ (Equation 3). To reduce the computation cost, wavelet coefficients are used to derive this integration directly. This specific algorithm implies level traversals of C^x_t, starting from the finest points to the coarsest ones. The electrical field is then obtained by computing numerically the integral of ρ. The complexity in memory accesses and computation for this step is in $\Theta(\#C^x_t + N)$.

3.2 Data Structures

In the past, the code used hash-tables [5, 9] to keep the wavelet coefficients. There was one hash-table for all points belonging to the same level of refinement. This storing method had several advantages like its low cost in memory consumption. Except for prediction step, traversals of entire level are needed in all steps of each splitting, which is easy to process by browsing a single hash-table. Moreover, in prediction steps 3A.1 and 3B.1, random accesses in writing could be generated in creating advected points. These accesses are cheap in hash-tables.

Nevertheless, considering our algorithms, we note drawbacks for these data structures. In each algorithm of the splitting, for one given point at level j we often have to read or write points in its neighbourhood at level $j - 1$ or $j + 1$ (because of the wavelet representation). As points of different levels are stored in different hash-tables, a local region in the phase-space can be spread out in memory when the number of points becomes large. So we do not benefit from spatial locality that cache memory could offer. Finally, it would be efficient to access quickly to values of coarse points, that are the most read data of the program.

Hash-tables were replaced with a sparse data structure that has better properties in term of averaged access time (in reading and writing) for the algorithms we use. We choose to have a dense storage of the coarser levels from level 0 to $L_{threshold}-1$ in a two-dimensional array, that we call *coarse matrix*. The finest levels from $L_{threshold}$ to L_{max} are contained in blocks, that we call *fine blocks*. An additional table of pointer indirections is used to access points in these fine blocks. This table represents an overhead in term of memory consumption. Moreover, coarse matrix and fine blocks will contain frequently zero values ; so, the management of the sparsity is not optimal. But the cost to read or write a coarse element in the sparse structure consists in just one memory access. And it costs two memory accesses to read an element in fine blocks due to the indirection. The writing of such an element needs in average more than two accesses, because the first time a fine block is written, the indirection pointer must be set to reference an allocated memory block.

This sparse structure with one level of indirection enables both to take into account the sparsity and to have a fast access to one element. For small or large number of points, the traversal algorithm always benefits from spatial locality. So, traversals of points of level j are effective even if points of level $j + 1$ or $j - 1$, located in the neighbourhood, are used. By representing the coarse matrix, and the table of indirection pointers with 2D arrays, the parallelization will lead to map parts of these arrays onto processors.

By removing hash-tables from the application, the execution time of simulations has been reduced for all our physical test cases. For large grids that implies between 1 200 000 and 3 000 000 points, the execution time is divided in average by a factor three on the machine described in subsection 5.1.

4 PARALLEL ALGORITHM

A profiling of the application (see Table 1 for one processor) indicates that all steps of the simulator consume significant processor time. Even if the backward advection step with the inverse wavelet transform concentrates 58% of execution time, we have to consider the parallelization of all steps to get an eventually scalable program. Algorithms of this simulator could potentially exploit medium-grain and fine-grain parallelism. Inside each procedure, a domain decomposition

of the phase space and of sparse data structures could provide independent computation. Nevertheless, to work on a given subdomain, the wavelet based method implies a large use of coarsest level points. These points are as far from each other in phase space as their level is coarse. So, most of our algorithms have memory access patterns that cover almost the whole phase space. Moreover, our procedures involve numerous and complex data shape dependencies between data values, and the grain of parallelism is then not coarse. For these reasons, a good approach for parallelization is a programming model that does not need explicit communication. We choose to develop the parallel solution within the OpenMP programming paradigm [8]. So, the target parallel machine is a shared memory architecture.

4.1 Partitioning

Parallel computations over a large number of wavelet coefficients presents an interesting challenge. A critical issue concerns the management for the details of the last two or three finest levels. For costly test cases with a large number of details, they induce over 90% of the memory acesses and computation costs. Our data structure is designed in order to take into account the sparsity of these details: no fine block is allocated in region where details are under the fixed threshold ϵ. And a traversal of the whole phase space skips regions where no fine block is allocated. At run-time, we do consider that there is exactly one thread per used processor, which does computations on column-wise blocks of the phase space (sets of x-columns). As most of our algorithms need a traversal of the whole phase space, the same parallelization strategy is often applied. To perform a traversal of the phase space $(x, v) \in \mathbb{R}^2$, the loop in variable x is parallelized. Spatial locality is then maximized because elements are accessed in sequential order for the loop in variable v. Performing such a parallelization comes to decompose the phase space domain into several blocks whose largest dimension is N (Fig. 6). For the moment, the for-loop in x is distributed with a uniform block-cyclic data distribution onto the processors. We expect to refine that data and computation distribution in the near future to enhance the load imbalance. Some procedures need a traversal of the phase space one level of refinement after the other. In this case, the parallelized x-loop is nested inside a j-loop, and a synchronization between every processors is required at each iteration of j.

By parallelizing the x-loop, the prediction step (3A.1, 3B.1) distributes column-wise blocks of the phase space to the threads. Threads translate and add points in a sparse data structure $A^v_{t^*}$ (or A^x_t). Eventually two threads could try to add a point at a same location where fine block is not allocated in the indirection table. The use of OpenMP lock primitives is needed to prevent these threads to set the indirection pointer simultaneously with two different fine blocks.
A level by level traversal is required for the construction of the well formed tree. Thus, the data locality on the processor is improved because outputs generated by the previous prediction step are used again.
For the backward advection steps (3A.3, 3B.3), the explicit block algorithm already described were parallelized. Data written locally on processors in the previous step may be used again here. For the splitting in v-direction, step (3A.3) implies a parallelization in variable x which preserves data locality for points in $A^v_{t^*}$. But for the splitting in x-direction, step (3B.3) has to be associated with a parallel for-loop in variable v. Indeed, values needed by the interpolation have to be on the same processor because of data dependencies with other points on the same v-row. So, there is an implicit transposition (done automatically thanks to shared memory) of the distributed data A^x_t. Finally, results of the interpolation are written in the $F^v_{t^*}$ (or F^x_t) data structure in parallel. These writings could be done concurrently because each processor writes values inside its own block.
Adaptive wavelet transform (3A.4, 3B.4) and field computation (3C) require level by level traversals. These traversals are performed as described above.

To summarize, for all steps except inverse wavelet transform: the x-loop (often nested inside the j-loop) is parallelized, and a j-loop implies a synchronization at each iteration.

4.2 Data Placement

OpenMP lacks the capability for data placement among processors and threads for achieving data locality [7]. In general, the absence of such a mechanism leads to poor cache memory reuse and costly remote memory accesses. In our environment (described in subsection 5.1), the strategy of first-touch page placement is used. So, the thread that first references a virtual address causes that address to be mapped to a page on the node where the thread runs.

To control data locality, the strategy consists in having exactly the same parallel x-loop for all steps except inverse wavelet transform. In such case, most data readings and writings are performed locally. Indeed, each thread is responsible for the computations on a fixed set of x values. An example of such a distribu-

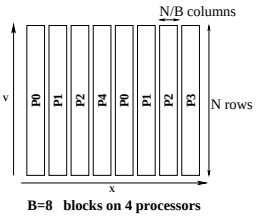

Figure 6. Block distribution for B blocks of x-values

tion of the phase space is sketched in Figure 6 for $B = 8$ blocks and 4 processors for the parallelization of the x-loop. Thus, the associated x-columns of sparse data structures A, F and C are initially stored by the operating system on the node where the thread runs. During all parallel steps, only a small percentage of readings concerns data mapped on an other node. These accesses occur mainly for points of coarsest levels in the sparse structures and they imply crossing the boundary of the local memory domain.
As said previously, we chose a block-cyclic mapping of x values onto the threads. As complexities of most algorithms are linear in the number of points in the sparse structure, there are some blocks which generate large computation cost and blocks implying small number of operations. That leads necessarily to load imbalance. On the one hand, small blocks (large B value) manage a load balancing automatically, because computation of costly regions is dispatched on several threads. But small blocks induce a communication overhead, because many cache memory readings are converted into distant memory access on other nodes. On the other hand, big blocks (small B value) improve the spatial locality as our algorithms often use data in a relatively large neighbourhood, but increase the load imbalance. We have to find a balance to obtain an optimal block size.

4.3 Optimization

In the inverse wavelet transform step (coupled with backward advection), the considered blocks in the algorithm are either x-wise or v-wise depending on the splitting direction. It is coded in a SPMD style and the management of computation on the boundaries of the block, copy of input values in a block and copy of the results have to be done explicitly. x-wise blocks are kept throughout the splitting in v-direction in order to maximize data locality. For the splitting in x-direction, the v-wise blocks lead to numerous communications between nodes with an All-To-All pattern.

For simplicity of the analysis in the next section, we will assume the number B of blocks is identical for the two splittings and during the entire simulation. The parameter B plays a major role in achievable performance. With a very few blocks, the inverse wavelet transform will be quicker, but the load imbalance will be degraded and the parallelism will be limited to B processors. Depending on the cache size of the machine and the phase space size, an optimal value for B could be found.

5 PERFORMANCE ANALYSIS

5.1 Numerical Experiments

The performance analysis presented in this section has been carried out on two SGI machines of 512 and 256 processors at CINES[1]. The SGI Origin 3800 is a scalable shared memory multiprocessor system. In terms of parallel computer architecture, this machine is a cc-NUMA architecture (cache coherent Non Uniform Memory Access). With a frequency equal to 500 MHz, the theoretical peak performance is 1 GFLOPS per processor if the two independent floating-point units are busy. The level 1 instruction and data caches have a size of 32 KB, whereas the secondary unified instruction/data cache size is 8 MB. In all tests, the number of processors was chosen equal to the number of processors.

5.2 Scalability

For a representative test case, we present efficiency and computation time in seconds at one particular time step whithin a simulation of 1000 time steps. The Table 1 shows the time usage of the time step $t = 540$ including performances of each part of the global algorithm described previously. The adaptive algorithm starts with a distribution function of $2^8 \times 2^8$ points at level $j=0$, with $L_{max}=4$ levels of refinement, and with $L_{threshold}$ set to 1. We took a small number of levels of refinement coupled with a large grid of points at level $j=0$. Thus, the number of synchronization in several algorithms decreases. But, sparsity is then reduced in the data structure.

Phase space is then a $2^{12} \times 2^{12}$ mesh composed of a $2^9 \times 2^9$ coarse mesh and a set of fine blocks (each fine block contains $2^{2L_{max}-2L_{threshold}}=64$ points). For this machine, experiments showed that a number of blocks $B = 16$, $B = 32$ and $B = 64$ are efficients and similar in performance. That induces respectively a

[1]Centre Informatique National de l'Enseignement Supérieur (France).

block of 9.8 MB, 5.7 MB, and 3.6 MB intensively used in the inverse wavelet transform step (3B.3). These block sizes mean a good usage of L2 cache of 8 MB. In the following results, we give execution times for $B=64$ because it allows us to use up to $P=64$ processors.

The number of details greater than the ϵ threshold is 1.2×10^6 for the splitting in x and 1.8×10^6 for the splitting in v. This explains partly the differences in execution times between the two splittings, because complexites grow linearly with the number of non-zero coefficients.

The splitting in v-direction scales very well up to 64 processors with only one block per thread. Efficiency is greater than 70% for all associated steps. Load imbalance is the only reason that explains the parallel overhead. The relatively large computation times make the few necessary synchronizations negligible, even for 64 processors. A good cache reuse of the set of x-rows on a single processor explains efficiencies upper than 100% in the wavelet transform step.

Concerning the splitting in x-direction, efficiencies for the backward advection and the wavelet transform are not very good. For this test case and this time step, wavelet coefficients of finest levels are gathered in about 50% of the v-columns (see the dispersion of wavelet coefficients on Fig. 8). This implies idle processors and then a large load imbalance. Furthermore, this step consumes memory bandwidth to exchange details between nodes because of the parallelization of the v-loop and the implicit transposition.

Efficiency and computation time for the complete test case (1000 time steps) are given in Table 1. The application has a good scalability. With 32 processors (respectively 64 processors), we achieve a speed-up of 25 (respectively 42). On another architecture too (IBM NH2 16-way nodes with Power3 processors), the application provides a sustained speed-up of 14.6 on 16 processors for the entire simulation.

6 PHYSICAL RESULTS

In order to test our adaptive scheme, we have chosen the standard two-stream instability case. In this model, two streams of charged particles encounter each other in the physical space with opposite velocities (see [1] page 94 for more details). When evolving in time, a perturbation occurs that grows exponentially. In the phase space, this perturbation corresponds to a vortex creation at the center of the distribution function.

During the simulation of this case, the adaptivity is performed with interpolets that conserves the density and the interpolation in the backward advection is computed within the wavelet space. These two key

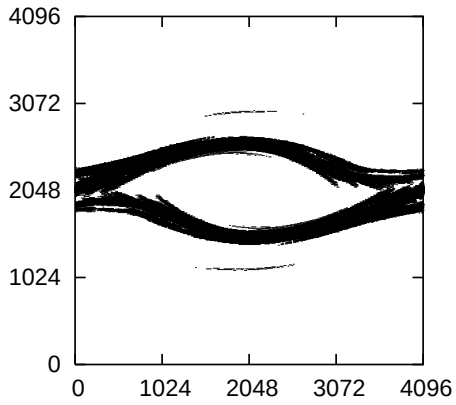

Figure 7. Details contained in fine blocks at $t = 540$

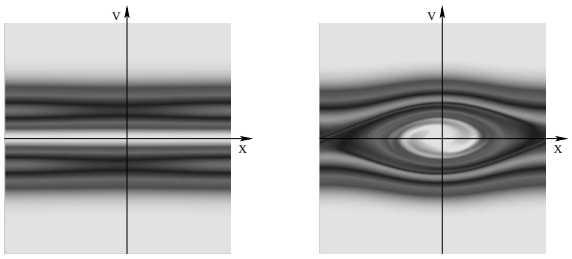

Figure 8. Distribution function at $t = 0$ **and** $t = 540$

points ensure an accurate computation of the numerical solution. Although no analytical solution is known for this test case, the qualitative behaviour is well known by physicists and applied mathematicians. Our simulation reproduces a correct behaviour for our physicist colleagues of INRIA-Calvi project

Moreover, Figure 8 shows a large number of points in the adaptive grid. So, this case is relevant to test the parallelization of the adaptative scheme.

7 CONCLUSION

We described the parallelization at a medium-grain level of an adaptive numerical simulator that solves the 1D Vlasov-Poisson system. Because access patterns to sparse data are complex and very large, a shared memory architecture was targeted for this application. Almost every steps of the original algorithm were parallelized and we obtained that each computing thread worked on a block of local data most of the time. The scalability is quite good but strongly linked with the sparse representation of physical phenomena. In adapt-

ing the block size for each thread, we could find a better data distribution to reduce the load imbalance. But a difficulty comes from the lack of facilities for data placement with OpenMP.

Future work will also consists in enlarging the spectrum of numerical physical experiments. Physical phenomena evolve naturally in three dimensions, so the full 3D Vlasov-Maxwell system has to be simulated for most realistic physical cases. One of our first objective is to extend the described algorithm in two dimensions to obtain a 2D adaptive Vlasov solver. An other work will consist in coupling this 2D solver with an electromagnetic field solver based on Maxwell equations instead of Poisson equation.

References

[1] C. Birdsall and A. Langdon. *Plasma Physics via Computer Simulation*. Mc Graw Hill, 1985.

[2] C. Cheng and G. Knorr. The integration of the vlasov equation in configuration space. *J. Comput Phys.*, 22:330, 1976.

[3] A. Cohen, S. Kaber, S. Mueller, and M. Postel. Fully adaptive multiresolution finite volume schemes for conservation laws. *Math. Comp.*, 72:183–225, 2003.

[4] M. Griebel and F. Koster. Adaptive wavelet solvers for the unsteady incompressible Navier-Stokes equations. In *Advances in Mathematical Fluid Mechanics*, pages 67–118. Springer, 2000.

[5] M. Griebel and G. Zumbusch. Parallel multigrid in an adaptive PDE solver based on hashing. In *ParCo '97*, pages 589–599. Elsevier, 1998.

[6] M. Gutnic, M. Haefele, I. Paun, and E. Sonnendrücker. Vlasov simulations on an adaptive phase-space grid. *Comput. Phys. Commun.*, 164:214–219, 2004.

[7] A. Marowka, Z. Lui, and B. Chapman. OpenMP-Oriented Applications for Distributed Shared Memory Architectures. *Concurrency and Computation: Practice and Experience*, 16(4):371–384, 2004.

[8] OpenMP Architecture Review Board, OpenMP Specifications, 2004. `http://www.openmp.org`.

[9] M. Parashar and J. C. Browne. Systems engineering for high performance computing software: The HDDA/DAGH infrastructure for implementation of parallel structured adaptive mesh refinement. In *IMA Vol. 117: Structured Adaptive Mesh Refinement Grid Methods*, pages 1–18. Springer, 2000.

[10] E. Sonnendrücker, J. Roche, P. Bertrand, and A. Ghizzo. The semi-Lagrangian method for the numerical resolution of the Vlasov equation. *J. Comput. Phys.*, 149(2):201–220, 1999.

	Time	Efficiency	Time	Efficiency	Time	Efficiency	Time	Efficiency
Nb. processors	**1**		**16**		**32**		**64**	
3A splitting in v-direction								
3A.1 Prediction	0.7659	100	0.0523	92	0.0313	76	0.0172	70
3A.2 Well formed tree	1.4023	100	0.0908	96	0.0474	93	0.0289	76
3A.3 Backward Advection	5.1605	100	0.3494	92	0.1858	87	0.0954	85
3A.4 Wavelet transform	0.6782	100	0.0393	108	0.0208	102	0.0110	96
3B splitting in x-direction								
3B.1 Prediction	1.1447	100	0.0786	91	0.0452	79	0.0244	73
3B.2 Well formed tree	2.4335	100	0.1595	95	0.0876	87	0.0467	81
3B.3 Backward Advection	4.9924	100	0.3746	83	0.2182	71	0.1532	51
3B.4 Wavelet transform	0.8598	100	0.0602	89	0.0382	70	0.0250	54
3C Compute Field	0.2523	100	0.0163	97	0.0107	73	0.0056	70
Execution time (step $t = 540$)	17.6903	100	1.2217	91	0.6862	81	0.4083	68
Execution time (entire simulation)	15539	100	1094	89	618	79	368	66

Table 1. Efficiency and computation time in seconds for the time step at $t = 540$