Parallelization of the Numerical Lyapunov Calculation for the Fermi-Pasta-Ulam Chain

Fabrice Rastello, Thierry Dauxois

Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON n° 8512

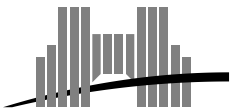
CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

Parallelization of the Numerical Lyapunov Calculation for the Fermi-Pasta-Ulam Chain.

Fabrice RASTELLO
Thierry DAUXOIS

November 2001

Research Report N° RR2001-42



École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : lip@ens-lyon.fr

INRIA

Parallelization of the Numerical Lyapunov Calculation for the Fermi-Pasta-Ulam Chain.

Fabrice RASTELLO
Thierry DAUXOIS

November 2001

Abstract

In this paper, we present an efficient and simple solution to the parallelization of discrete integration programs of ordinary differential equations (ODE). The main technique used is known as loop tiling. To avoid the overhead due to code complexity and border effects, we introduce redundant tasks and we use non parallelepiped tiles. Thanks both to cache reuse ($\times 4.3$) and coarse granularity ($\times 24.5$) , the speedup using 25 processors over the non-tiled sequential implementation is larger than 106.
We also present the draft of a fuzzy methodology to optimize the tile size and we illustrate it using real measurements for the communication cost and the execution time. In particular, we observe that the model of communication latencies over a Myrinet network is not as simple as is usually reported.
We apply this solution to study the Lyapunov exponents of the Fermi-Pasta-Ulam (FPU) chain and in particular the dependence of the maximum Lyapunov exponents as a function of the length of the chain.

Keywords: Hierarchical tiling, redundant tasks, parallelism, scalability, cache optimization, locality, tiles shape, communication overhead, heterogeneous ressource, dynamical system theory, Fermi-Pasta-Ulam chain, Lyapunov instability analysis, phase space properties.

Résumé

Nous présentons dans ce rapport une solution à la fois simple et efficace à la parallélisation d'un programme d'intégration d'équations différentielles ordinaires (ODE). La technique utilisée est basée sur la méthode de pavage qui permet de favoriser la localité à la fois temporelle et spatiale des accès aux données. Cette implémentation, testée sur une pile de 25 processeurs permet une exécution plus de 106 fois plus rapide que le programme séquentiel initial. Cette solution est appliquée à l'étude des exposants de Lyapunov de la chaîne de Fermi-Pasta-Ulam en fonction de la longeur de la chaîne.

Mots-clés: Pavage, tâches redondantes, parallélisme, optimisation de cache, localité des accès mémoire, forme de tuiles, théorie des systèmes dynamiques, chaîne d'oscilateurs de Fermi-Past-Ulam, Analyse d'instabilité de Lyapunov, propriétés de l'espace des phases.

1 Introduction

The goal of this paper is to present a practical implementation of tiling methods to a real-life problem, namely the study of Lyapunov exponents of the Fermi-Pasta-Ulam (FPU) chain. This article can be considered from two angles: a physical and a computational point of view.

Lyapunov calculation for the Fermi-Pasta-Ulam Chain Without underestimating the numerous experimental and theoretical contributions, it is important to recall the importance of computers in the development of the dynamical system theory. The pionnerring work [18] by Fermi, Pasta and Ulam in Los Alamos on the first computer, the Maniac, has led to the new concept of soliton; they correspond to localized waves with the astonishing property to collide with other solitary waves remaining unchanged. Two decades later, Lorenz realized [16] the chaotic behavior of his atmospherical model thanks to his computer, again: this lead to the so-called "butterfly effect". These are the two first examples of many others.

Following these seminal works, many theoretical and numerical studies have been devoted to the characterization of chaos in high-dimensional dynamical systems. Nevertheless, several fundamental issues are not understood. In particular, the relation between Lyapunov instability analysis and phase space properties like diffusion of orbits, relaxation to equilibrium states, spatial development of instabilities remains to be clarified.

In this paper we present a numerical study to compute the Lyapunov exponents of the Fermi-Pasta-Ulam (FPU) chain. The FPU model has played for dynamical system theory a similar role to the Ising model in statistical mechanics. Let us just quote some major studies and discoveries motivated by the FPU numerical experiment: the introduction of the concept of soliton, the prediction of the transition to large-scale chaos by the resonance overlap criterion, the use of KAM perturbation theory and Nekhoroshev stability estimates, the numerical detection of the strong stochasticity threshold (see Ref. [20] for a review).

Besides, there are very few examples of analytical calculations of the Lyapunov exponents in the chaotic component corresponding to energy equipartition and that's why it is very important to have reliable and fast algorithm to measure them numerically. This is the main goal of this paper where we implement the idea of tiling to this typical problem of dynamical system theory.

Tiling the iteration space The basic idea of tiling, also known as loop blocking, is to group elemental computation points into tiles that will be viewed as computational units. Hence, the *communication* time (which is usually proportional to the surface of the tile) decreases while the *computation* time (which is proportional to the volume of the tile) increases. Tiling is also a widely used technique to increase the granularity of computations and the locality of data references. The program that we aim to parallelize belongs to a class of algorithms that has been extensively studied, namely Successive Over Relaxation methods (SOR) or Alternate Direction Implicit (ADI) integration. Many differential equations are expressed in implicit form and a large part of the literature is focused on achieving parallelism across the method and on the research of stable integration schemes based on direction splitting to achieve parallelism across the system. However, the McLachlan-Atela's integration scheme which we consider here is explicit and the problem is embarrassingly parallel across the space. Unfortunately, to the best of our knowledge, existing implementations perform partitioning across the space direction but not across the time direction. This leads to the solution presented in Figure 4 where space locality is enhanced but time locality is not: cache reuse is not fully optimized. Nevertheless, existing literature on tiling [1, 15, 7, 4, 6, 29] provides a solution to this simple problem: a diamond-shaped blocking as illustrated in Figure 5. The main drawback of this solution is the complexity of its implementation. In the concern of our study, there is an additional difficulty: the presence of high frequency synchronization barriers (see Section 3.1). Hence classical methods should be adapted using hierarchical tiling: this solution is illustrated in Figure 6. We present a solution which is simpler to implement yet still very efficient: we use redundant computational tasks (see Figure 8). This solution is particularly adapted to explicit integration schemes, but can also be applied to general multidimensional SOR programs with local dependences (like heat propagation or molecular dynamics), in addition to many image processing algorithms.

Also, automatic partitioning is complex since it should take into account both the volume and frequency of communications, in addition to data reuse, code complexity, etc. In this article we give the draft of a

fuzzy methodology decomposed into different successive phases, sorted in decreasing importance order. This methodology is illustrated using our practical optimization problem. Finally, we are led to testing the validity of usual assumptions made for tiling optimizations, like the formulae used to express computation and communication times. In particular, we claim that linear communication models designed for small-scale message passing with perfectly synchronized communications [27], becomes strongly incorrect when communications are performed among data based computations.

This report is decomposed into three major parts. The physical problem is presented in Part 2. The numerical method of tiling and all the details are presented in Part 3. The results obtained on the original problem are emphasized in Part 4.
Then the computational part is organized in the following way: the code is presented in Section 3.1, then its modelisation into task graph is given in Section 3.2. Different partitioning solutions are discussed in Section 3.3, then different steps to find optimal parameters are described in Section 3.4. Finally, Section 3.5 provides performance results, together with a short discussion on heterogeneous platforms.

2 The physical problem

In the framework of one-dimensional lattices, Livi, Politi and Ruffo [22] have shown that the Lyapunov exponents are only functions of the energy density and not of the length of the chain. This major result was recently questioned because of some simulations results presented by Searles et al. [30]. It was however a very difficult task to confirm or infirm the reported logarithmic dependence, without the help of very fast algorithms. Another important question nowadays is the behavior of the largest maximal Lyapunov exponent in the very low energy limit. In this completely inaccessible region with usual algorithms, where the chaotic layers are distincts, the speed of convergence toward energy equipartition was studied, starting from particular initial conditions. However, this equipartition predicted by equilibrium statistical mechanics is reached very slowly and one needs again efficient tools to give a final answer to the Fermi-Pasta-Ulam paradox, discovered in the fifties.

2.1 The Fermi-Pasta-Ulam chain

Denoting by $x_i(t)$ (respectively $v_i(t)$) the position (respectively the velocity) of the i-th atom ($i \in [0, H-1]$) at time t, the equations of motion of the β-FPU chain read

$$\left\{ \begin{array}{rcl} \dot{x}_i & = & v_i \\ \dot{v}_i & = & x_{i+1} + x_{i-1} - 2x_i + \beta \ \left[(x_{i+1} - x_i)^3 - (x_i - x_{i-1})^3 \right] \end{array} \right. \tag{1}$$

where $\beta = 0.1$ for historical reason. We have chosen fixed boundary conditions but periodic ones would have led to extremely minor changes.

As usual, in numerical simulations one does not study the real Hamiltonian system, but a discrete time version that approximates the time continuous dynamics. It is therefore essential for long time simulations to use an appropriate symplectic integration scheme in order to preserve as far as possible the Hamiltonian structure of the problem. We adopt the best 4th-order symplectic algorithm, the McLachlan-Atela's algorithm [23], with a time step $dt = 0.01$; this choice allows us to obtain an energy conservation with an excellent accuracy (8 digits).

We follow the approach proposed by Fermi-Pasta-Ulam in Ref. [18], i.e. to consider the stability of one normal mode of the harmonic part. However, contrary to them, we have performed simulations adopting as initial condition the highest frequency mode, the so called π-mode, which corresponds to the following zig-zag pattern for x_i

$$x_i = (-1)^i \ a \quad \text{and} \quad v_i = 0 \tag{2}$$

where a is its amplitude.

Since most of the normal modes of the harmonic part of the Hamiltonian are no longer solutions of the full Hamiltonian, energy initially fed into one single mode will be shared on later times among other

modes. The transient time, called equipartition time since the system evolves toward energy equipartition, is strongly dependent on the energy density and diverges in the low regime: this divergence is at the origin of the FPU paradox. However, the π-mode being an exact solution, we have added a small amount of noise (of order 10^{-14}) on the velocities to destabilize this initial state. As already shown in Ref. [11], the π-mode is modulationally unstable above a critical energy E_c, that can be analytically derived [13].

Let us recall briefly now the definition of the Lyapunov exponents and their associated numerical calculation, before presenting the estimation of the N_{lyap} few largest Lyapunov exponents λ_j $(1 \leq j \leq N_{lyap})$ as a function of the energy density and as a function of the number of site H for a given energy density.

2.2 Lyapunov Exponents

Lyapunov exponents play an important role in the theory of both Hamiltonian and dissipative dynamical systems. They provide a computable, quantitative measure of the degree of stochasticity for a trajectory. In addition, there is a close link between the Lyapunov exponent and other measurements of randomness such as Kolmogorov entropy or fractal dimension.

The procedure for computing the Lyapunov exponents was developed by Benettin *et al.* [17]. The largest one, λ_1, is given by the rate of exponential growth in the separation of two infinitesimally displaced trajectories. To evaluate this growth we study the evolution of vector [1] $\begin{pmatrix} \delta x^1 \\ \delta v^1 \end{pmatrix}$, tangent to $\begin{pmatrix} x \\ v \end{pmatrix}$, and defined as follow:

$$\left\{ \begin{array}{rcl} \delta \dot{x}_i^1 & = & \delta v_i^1 \\ \delta \dot{v}_i^1 & = & \left[1 + 3\beta(x_{i+1} - x_i)^2\right] \delta x_{i+1}^1 + \left[1 + 3\beta(x_{i-1} - x_i)^2\right] \delta x_{i-1}^1 \\ & & \qquad - \left[2 + 3\beta\left[(x_{i+1} - x_i)^2 + (x_{i-1} - x_i)^2\right]\right] \delta x_i^1 \end{array} \right. \tag{3}$$

Denoting by $\|\delta_1(t)\| = \left\| \begin{pmatrix} \delta x^1(t) \\ \delta v^1(t) \end{pmatrix} \right\|$ the Euclidean norm at time t between both trajectories $\begin{pmatrix} x \\ v \end{pmatrix}$ and $\begin{pmatrix} x + \delta x^1 \\ v + \delta v^1 \end{pmatrix}$, the largest Lyapunov is defined as

$$\lambda_1 = \lim_{t \to \infty} \frac{1}{t} \ln \frac{\|\delta_1(t)\|}{\|\delta_1(0)\|}. \tag{4}$$

One important difficulty is located in the fact that if the norm increases exponentially with time t, this leads to overflow and other computation errors. To circumvent this problem, we choose a small fixed iteration interval n_{ortho} and we renormalize $\delta_1(t)$ to unity every $n_{ortho} \times dt$ time interval. Thus the computation scheme is the following:

Program 1 *Computation scheme of the largest Lyapunov exponent* λ_1

$lambda[1] = 0$
do $t = 0, T : dt$
  evolution of the orbit $\begin{pmatrix} x \\ v \end{pmatrix}$ and of the perturbation $\begin{pmatrix} \delta x^1 \\ \delta v^1 \end{pmatrix}$.
  if $\frac{t}{dt} \equiv 0\ [n_{ortho}]$
    $lambda[1] = lambda[1] + \ln \left\| \begin{pmatrix} \delta x^1 \\ \delta v^1 \end{pmatrix} \right\|$
    $\begin{pmatrix} \delta x^1 \\ \delta v^1 \end{pmatrix} = \dfrac{\begin{pmatrix} \delta x^1 \\ \delta v^1 \end{pmatrix}}{\left\| \begin{pmatrix} \delta x^1 \\ \delta v^1 \end{pmatrix} \right\|}$

[1] δx^j (respectively $\delta v^j, x, v$) represents the column vector made of scalars δx_i^j (respectively $\delta v_i^j, x_i, v_i$).

and we finally use the remarkable mathematical property that, if $n_{ortho}dt$ is not too large:

$$\lambda_1 = \lim_{T\to\infty} \frac{lambda[1]}{T} \tag{5}$$

Unfortunately, there is no *a priori* condition for determining neither the number of iterations T that must be performed, nor a good estimate for the renormalizing time $n_{ortho}dt$. The latter should be chosen as small as possible, whereas the Gram-Schmidt orthonormalization procedure is extremely time consuming, and T should be chosen of course as big as possible.

Usually, we are interested in more than one Lyapunov exponent ($N_{lyap} > 1$), and a further difficulty appears. As the tangent vectors evolve, the angle between any two vectors ($\begin{pmatrix} \delta x^j \\ \delta v^j \end{pmatrix}$, $1 \leq j \leq N_{lyap}$) generally becomes too small for numerical computations. Thus in addition to renormalizing after each time interval $n_{ortho}dt$, we must replace the evolved vectors $\begin{pmatrix} \delta x^j \\ \delta v^j \end{pmatrix}_{1\leq j\leq N_{lyap}}$ by a new set of orthonormal vectors, which must be chosen to span the same subspace as the evolved set. This procedure can be systematized using a Gram-Schmidt algorithm, which gives directly the different Lyapunov exponents [17].

In summary, we need to implement the numerical integration of the evolution of the orbit represented by Eq. (1) and the evolution of the tangent space to compute the Lyapunov exponents. For a chain with H sites, the computation time for obtaining the first N_{lyap} Lyapunov is

$$O\left((N_{lyap}+1) * 2H * 4 * T\right) \tag{6}$$

since the symplectic integrator is of order four. Finally, every time $n_{ortho}dt$, we need to renormalize and orthonormalize the perturbation using the Gram-Schmidt algorithm. Let us present in the following section why the tiling procedure is particularly adapted to this tasks.

3 Computational strategy

In this part, we present our strategy for implementing the code onto a distributed memory parallel machine. For clarity, we will first present the initial program. Then, we will expose different possible strategies for parallelizing the code. Finally, after describing the parallel support we used, we will give performance results.

3.1 The sequential program

Let us present the sequential program, its main loop with its dependencies. It is made up of three consecutive phases: after the initialization,

- During the first phase ($0 \leq t \leq t_1$), the system evolves.
- During the second phase ($t_1 < t \leq t_2$), the system still evolves, in addition to the tangent space that is orthonormalized every $n_{ortho}dt$ iterations time.
- The third phase ($t_2 < t \leq t_3 = T$) is identical to the second one but the calculation of the Lyapunov exponents is additionally performed.

Program 2 *The sequential code*

Initialization
do $t = 0, T : dt$
  if $t > t_1$ {phase 2 or 3}
    do $k = 0, 3$ {McLachlan-Atela's algorithm uses 4 iterations for each time step }
      doall $j \in \{1, \ldots, N_{lyap}\}$
        dovect $i \in \{0, \ldots, H-1\}$
          $\delta v_i^j = \delta v_i^j + c_k \times g_k(x_{i-1}, x_i, x_{i+1}, \delta x_{i-1}^j, \delta x_i^j, \delta x_{i+1}^j)$
          $\delta x_i^j = \delta x_i^j + d_k \times \delta v_i^j$
        } Evolution of tangent space
    if $\frac{t}{dt} \equiv 0 \ [n_{ortho}]$
      Gram_Schmidt $\begin{pmatrix} \delta x^1 & \ldots & \delta x^N \\ \delta v^1 & \ldots & \delta v^N \end{pmatrix}$
      if $t > t_2$ {phase 3} then updates the N_{lyap} Lyapunov's exponents
  do $k = 0, 3$
    dovect $i \in \{0, \ldots, H-1\}$
      $v_i = v_i + c_k \times f_k(x_{i-1}, x_i, x_{i+1})$
      $x_i = x_i + d_k \times v_i$
    } Evolution of the system

Where,

- H, represents the number of particles.
- N_{lyap}, represents the number of Lyapunov's exponents.
- c_k and d_k are coefficients of McLachlan-Atela's algorithm [23].
- f and g are functions obtained from the equations of motion (1) and their derivated forms.

Note that in practical, $t_3 \gg (t_1 \,\&\, t_2)$, i.e. the main time is spent on phase 3. In addition, as the time for updating Lyapunov's exponents is negligible, phase 2 and phase 3 can be considered identical. Consequently, we will focus our attention on phase 2 only.

3.2 The task graph

A task graph is an oriented graph, where vertices represent tasks, and edges represent dependences. There is a dependence between two tasks, say $Task_1$ and $Task_2$, whenever the computation of $Task_2$ requires the results of the computation $Task_1$. That means, that $Task_2$ must begin necessarily after the total completion of $Task_1$. For the sake of simplicity, only transitive reductions of graphs are represented: two tasks are in dependence if and only if there exists a path between there corresponding vertices.

As the task graph is rather complex, its representation is split into two parts.

- The sketch of Figure 1 represents only dependences related to the system evolution $\begin{pmatrix} x \\ v \end{pmatrix}$. The calculation of one particle is represented by an unfilled circle.
- The graph of Figure 2 represents only dependences related to the tangent space evolution $\begin{pmatrix} \delta x^j \\ \delta v^j \end{pmatrix}$. The calculation of one perturbation is represented by a cross. The four phases of McLachlan-Atela's algorithm for the system's evolution are not represented.
- The superposition of those two graphs gives the task graph of our program, approximatively represented in Figure 3.

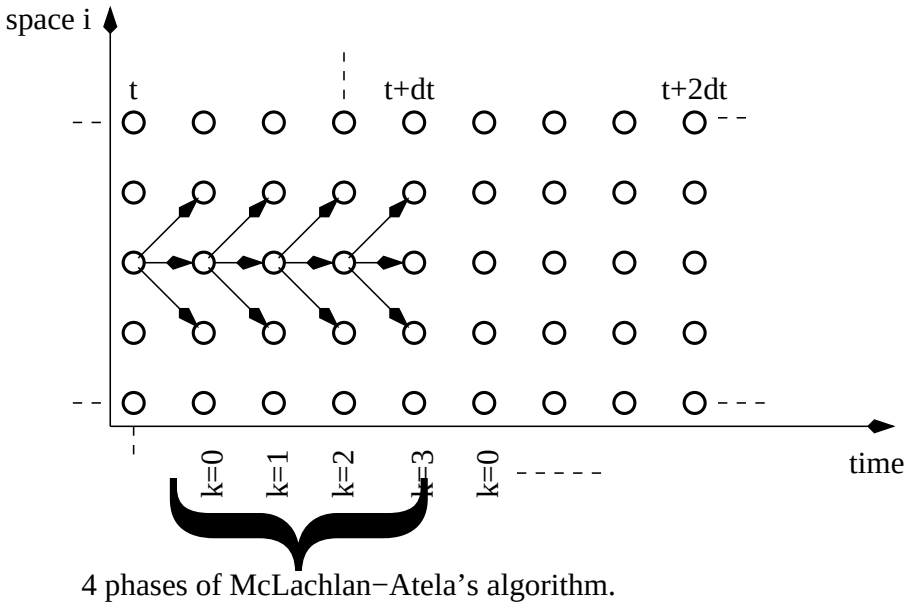


Figure 1: Tasks graph related to the system evolution. An unfilled circle represents the calculation of x_i and v_i. Reduced dependences are $(1,1)^t$, $(1,0)^t$ and $(1,-1)^t$.

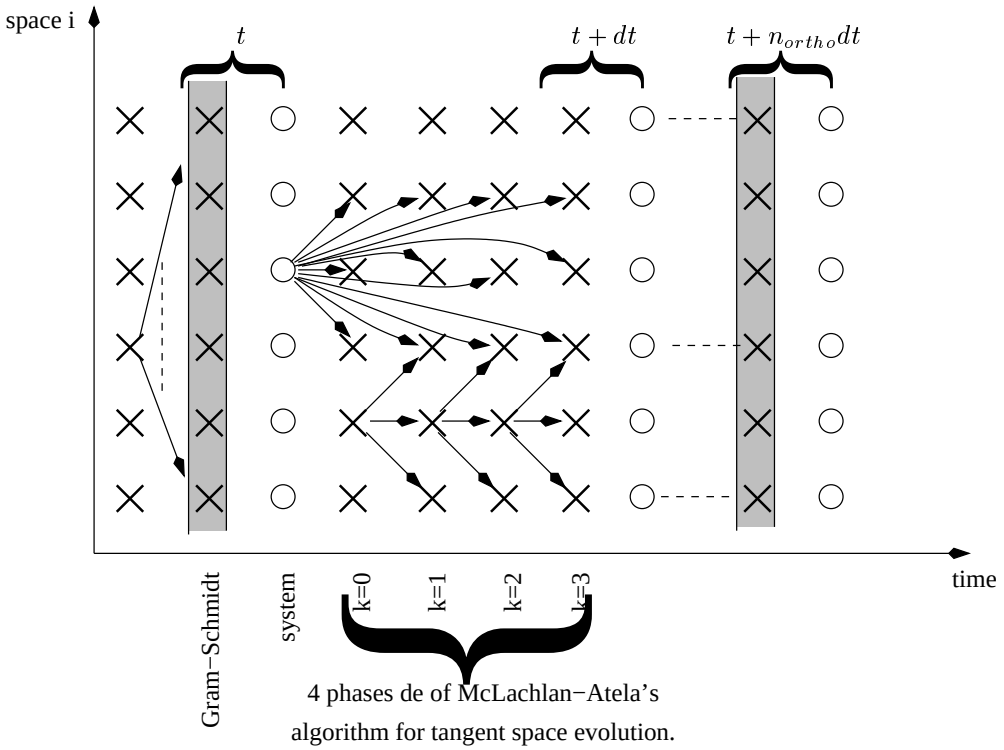


Figure 2: Tasks graph related to the tangent space evolution. An unfilled circle represents the calculation of x_i and v_i. A cross represents the calculation of δx_i and δv_i. Reduced dependences are $(1,1)^t$, $(1,0)^t$ and $(1,-1)^t$. Orthonormalization phase behaves like a synchronization barrier.

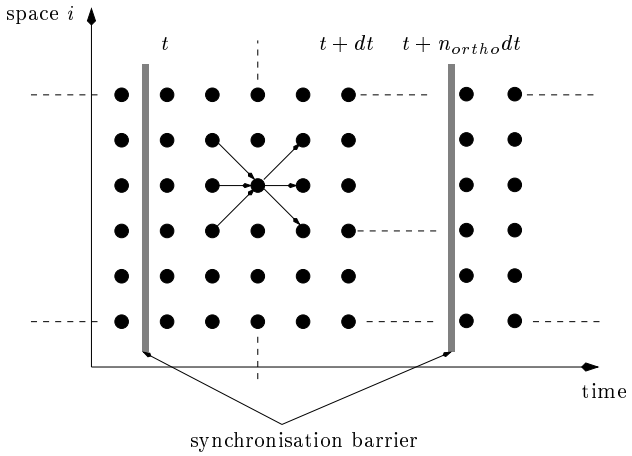


Figure 3: Reduced tasks graph of our program: both sub-graphs (system and tangent space) have been superposed. Reduced dependences are then $(1,1)^t$, $(1,0)^t$ et $(1,-1)^t$. To avoid the use of an additional temporary array, the four phases of the perturbations evaluation for time $t+dt$ that depends on the value of the system at time t are considered as atomic.

3.3 The parallelization: the strategies

3.3.1 On the importance of tiling the iteration space

For programs that deal with a large volume of data, *tiling* optimization technique (even on sequential) should be applied to speed-up the execution time.
Tiling is a widely used compiler technique to increase the *granularity of computations* and the *locality of data references*. This technique was originally restricted to perfect loop nests with uniform dependencies, as defined by Banerjee [2], but has been extended to sets of fully permutable loops [12, 21, 31]. The basic idea of tiling, also known as *loop blocking*, is to group elemental computation points into tiles that will be viewed as computational units. The larger the tiles, the more efficient the computations performed using state-of-the-art processors with pipelined arithmetic units and a multilevel memory hierarchy. Another advantage of tiling is the decrease in communication time (which is proportional to the surface of the tile) relative to the computation time (which is proportional to the volume of the tile). Hence, the naive approach represented in Fig. 4, that corresponds to schedule the tasks in the initial order and simply use the existent parallelism in the i-direction, would lead to catastrophic performances (see part 3.4).

Tiling has been studied by several groups and in different contexts. Rather than providing a detailed motivation for tiling, we refer the reader to the papers by Calland, Dongarra, and Robert [7] and by Högsted, Carter and Ferrante [19], which provide a review of the existing literature. Most of the work amounts to partitioning the iteration space of a uniform loop nest into tiles whose shape and size are optimized according to some criteria (such as the communication-to-computation ratio). Once the tile shape and size are defined, it remains to distribute the tiles to physical processors and to compute the final scheduling.

In our case, since the Gram-Schmidt ortho-normalization can be seen as a synchronization barrier, the domain to be tiled is a wide strip (usually $n_{orth} \lesssim 25$) of width $4 \times n_{orth}$. Considering that $h = H/P$ might be large compared to n_{ortho}, the domain should be partitioned into parallelogram-shaped tiles. Themselves should be sub-tiled into 2 triangles and parallelograms (see Fig. 6). As we will see (see part 3.3.2), this method is quite complex and might involve an overhead in the main calculation kernel. Hence we propose an alternative solution (the one we choose to implement), using dummy-tasks technique, which is still asymptotically optimal. The next parts are devoted to the description of those two solutions.

3.3.2 The parallelogram solution

For large iteration domains, classical tiling results [4, 6, 15, 29] set that tile's borders should be along the dependence cone. In our example, reduced dependences are $(1,1)^t$ and $(1,-1)^t$, so tiles should be diamond-

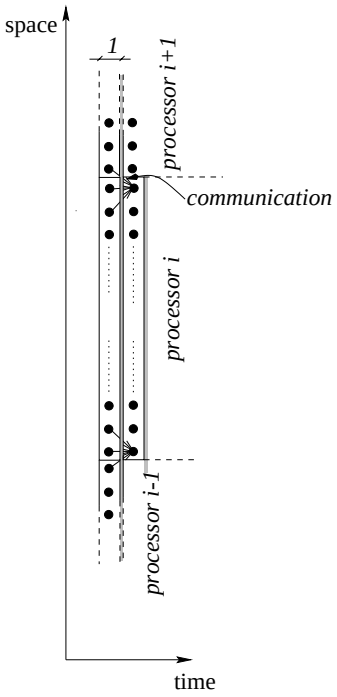


Figure 4: The naive approach would consist on partitioning the iteration space onto P horizontal strips. After each step (k or t incremented), the neighboring processors communicate the data useful for the next step. In other words, it corresponds to distributing the `doall(j)-dovect(i)` nested loops over the processors.

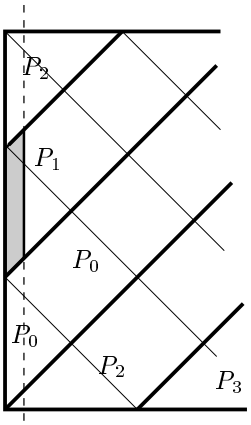


Figure 5: When the iteration-space is large, optimal tiling is made of diamond-shaped tiles. The presence of orthonormalization phase every $4n_{ortho}$ steps (represented by dash-line) explains the parallelogram shapes of tiles represented in Figure 6.

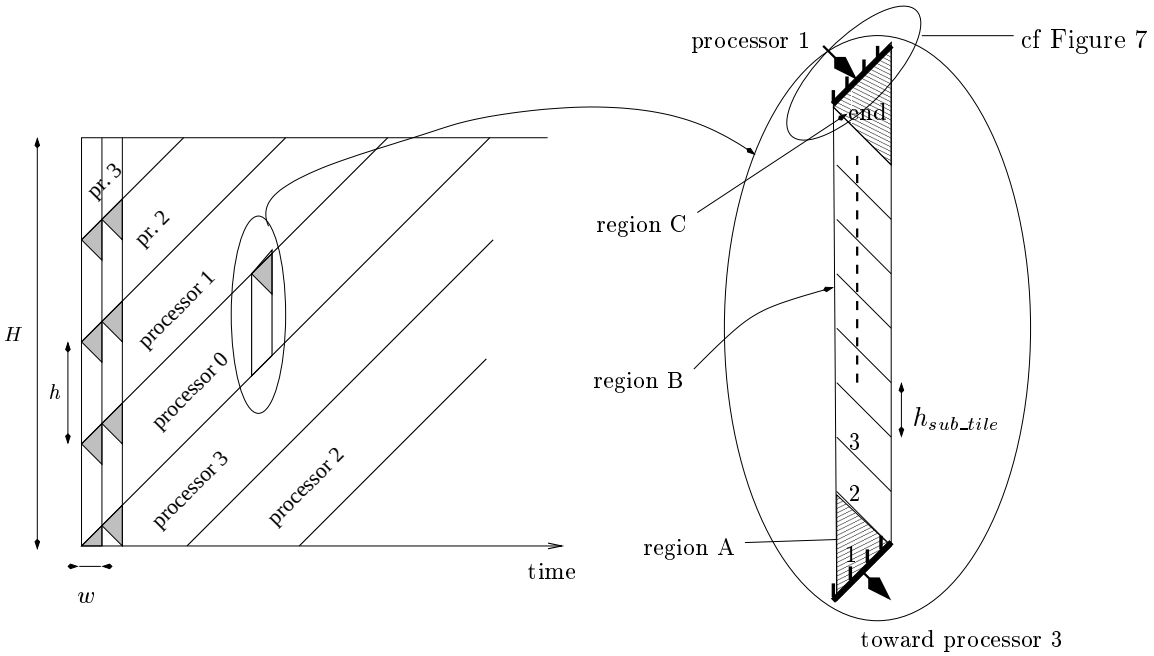


Figure 6: The parallelogram solution: The iteration space is partitioned into strips along the diagonal direction. Then strips are distributed cyclically over the processors. Hence, between each ortho-normalization, each processor takes care of parallelogram-shaped tile. Because of dependences, the computation of region C requires results from the computation of region A of the above processor. Hence, consecutively (but for the processors on the border), region A is performed, then data useful for region C are communicated between each neighboring processors, and finally regions B & C are computed from the bottom to the top. Eventually, if the domain is large, region B is tiled itself to ensure locality of data references.

shaped as represented in Figure 5. But the presence of the orthonormalization phase every $4n_{ortho}$ steps, shows non-atomic parallelogram-shaped tiles. That corresponds to the solution presented in Figure 6: each processor takes care of a diagonal strip and deal with an array of size $h = H/P$. The main problem with this solution is, as illustrated in Figure 7, the complexity of communicated data set. Indeed, the presence of conditional branches inside the loop, partially due to borders effect, would imply a non-negligible overhead.

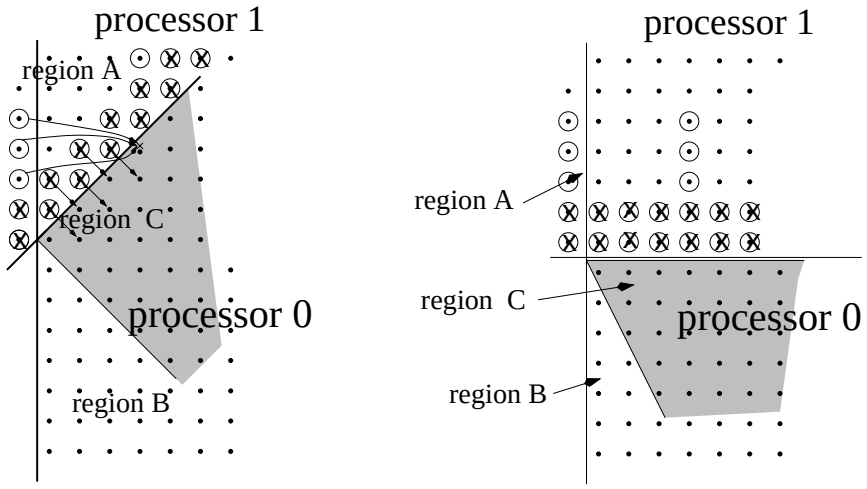


Figure 7: Part of the task graph of figure 6 before and after skewing of the iteration space. The data, necessary for the computation of region C by processor i, are represented with a circle for the main system (x_i & v_i) and with a cross for the perturbations (δx_i^j & δv_i^j).

Hence, even with loop optimizations like loop peeling, complexity of the generated code involves a consequent overhead in the execution time of the main kernel. Consequently, we looked for an other solution, extremely simple, and asymptotically optimal.

3.3.3 Second approach : rectangular tiling

As explained in Fig. 8, this solution consists on simply distributing the iteration space on horizontal strips over the processors. Hence, the computational unit of given processor between two communication steps is initially a rectangle of height $h = H/P$ and of width say w (w is a multiple of 4 and a divisor of $4n_{ortho}$).

However, because of dependences, each processor should do some redundant work. More precisely, it should perform (but for the processors on the border) $w(w-1)$ more tasks for each tile. Hence, while w increases, the communication cost overhead will decrease, and the main kernel of computation will be faster (because of locality of data references). On the other hand, while w increases, the amount of work also increases. A trade-off should be found to define the optimal parameters of the algorithm. That will be the goal of section 3.4.

3.4 Optimal parameters

In the existing literature, most of the work amounts to partitioning the iteration space of a uniform loop nest into tiles whose shape and size are optimized according to some criteria (such as the communication-to-computation ratio). Shape is usually constrained by the set of dependences, and size is determined to minimize the latency implied by a coarse grain computation. Classical approaches aim to solve analytically this optimization problem entirely all at once. Instead, we propose to consider the different criteria separately and treat them in decreasing importance order:

1. **Sequential execution time**: even on a sequential program, data reuse or absence of conditional branches can have a *huge* impact on the performances. Hence our first goal is to optimize the critical computational kernel performed by each processor. In this context:

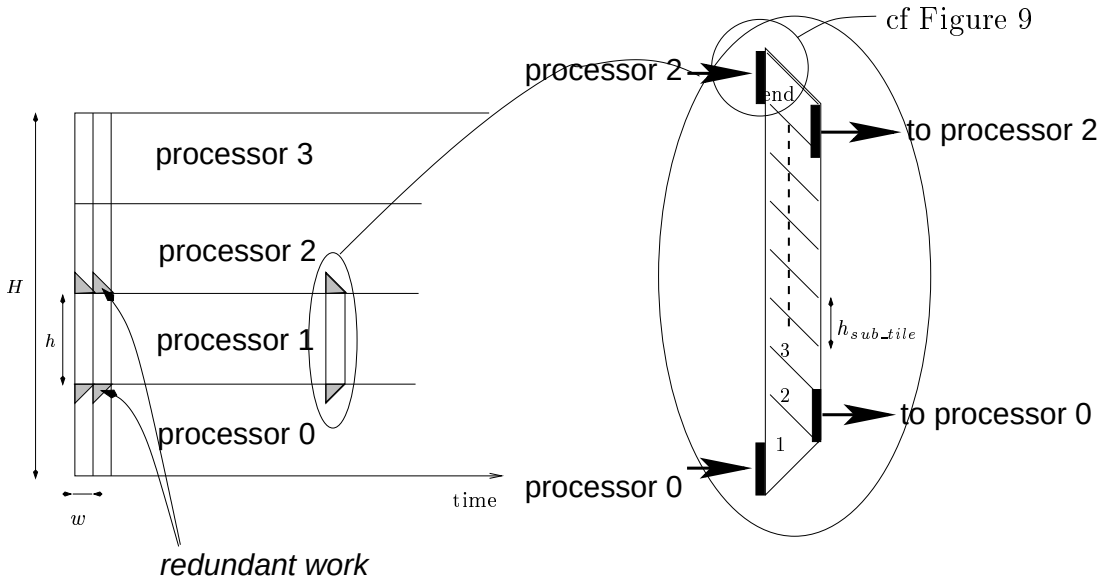


Figure 8: Rectangular tiling solution: the iteration space is partitioned into P horizontal strips that are distributed over the P processors. Hence, between each ortho-normalization, each processor takes care of a rectangle. Because of dependences the computation of each rectangle requires the results from the computation of two triangles on the top and on the bottom. The choice here consists on computing those triangles by the processor (say i) itself. That corresponds to do *redundant work* because those two triangles are also computed by the neighbored processors on top (processor $i+1$) and on bottom (processor $i-1$). Hence, before each step of computation, each processor should get from its neighboring processors the data (a vector on the top and on the bottom) useful for its computation. Then, parallelepiped-shaped tile can be tiled itself into a triangle and parallelogram-shaped sub-tiles of height h_{sous_tuile} to ensure locality. Sub-tiles are executed from the bottom to the top.

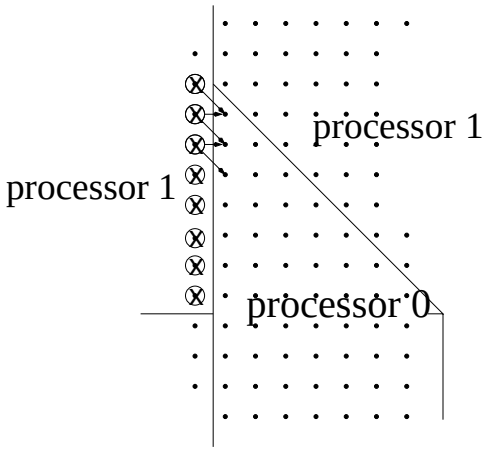


Figure 9: Part of the task graph of Figure 8. The data necessary for the computation are represented with a circle for the main system (x_i & v_i), and with a cross for the perturbations (δx_i^j & δv_i^j).

- For data based computation, tiling should be performed at least across one direction.
- If possible, rectangular tiling should be preferred to trapezoidal ones: borders effects are minimized, loop bounds are simple.

As an example, in our implementation, we fixed tiles width to be greater than 20, and sub-tiles height to be lower than 100. This lead to a speedup over than 4.

2. **Express parallelism**: too large tiles can involve subsequent latencies, and tiling could kill the parallelism.
   - Many applications contain recurrences of length 1 over each direction. In this context, tiles should not be too large and then distributed cyclically or in a more sophisticated way [10] over the processors.
   - Rescheduling tasks inside tiles in addition to pipelined communications can sometimes restore the parallelism [28].
   - Finally, introducing redundant tasks as it is done here can also restore the parallelism and is sometimes better than skewing the iteration space.
3. **Communication overhead**: even for perfectly synchronized applications, communication overhead can be critical. It is not the case of our application here where, in practical cases, H is large enough so that communication time becomes negligible compared to computation time. However,
   - In many applications, communication volume per computation volume decreases with the size of the tiles [29].
   - It is usually reported that for small-scale message passing, the communication latency is a linear function of the size of the messages. This is not true on the parallel support we used, but we can suppose that on some well configured distributed platforms, it might be better to send one time $2l$ bytes instead of two times l bytes (because of the startup). Then, coarse granularity should lead to better performances.

We call our approach a "fuzzy methodology" because the goal of each phase is not to minimize an analytical formula and to provide an exact result ("tiles size should be 3.567×0.45"). Instead, we need a result that is less constraint-full as possible. This idea is well illustrated by Figure 16.

3.4.1 The parallel support

The platform is made of 2 Piles of PC interconnected with a Myrinet network.

The first one, called PoPC have the following characteristics:

- 12 identical nodes (popc0-11):
  - Processors: Pentium Pro 200Mhz (cache L2: 256Ko),
  - Memory: 64Mo per node,
  - chipset SCSI adaptec AIC-7880 (Ultra narrow 20Mo/s),
  - Ethernet DEC21140,
  - Myrinet LANAI4.1 256Kbytes,
  - Hard drive SCSI 2Go per node, but for popc0 (4.3Go),
- 2 x Switch Myrinet double M2M-DUAL-SW8/SAN (ie 4 crossbars 8x8),
- Hub Ethernet 10Mb/s 16 ports.

The second one, called Pom have the following characteristics:

- 16 identical nodes:

  - Processors: PowerPC 604e 200Mhz (cache L1:32Ko+32Ko, cache L2: 256Ko),
  - Memory: 64Mo per node,
  - chipset SCSI ncr53c825a (fast narrow 10Mo/s),
  - Ethernet DEC21140,
  - Myrinet PMC LANAI4.1 512Kbytes,
  - Diskless but for pom0 and pom8 (4Go SCSI),
- Switch Myrinet octuple: M2M-OCT-SW8
- Switch Myrinet octuple: M2M-OCT-SW8
- Hub Ethernet 10Mb/s 24 ports.

Those parallel resources are part of several other platforms from the Laboratory for High Performance Computing (see `http://www.ens-lyon.fr/LHPC/ANGLAIS/choix.html` for more details). The optimized communication interfaces are developed by the "High Speed Networks and Parallel Applications" team from the LHPC: BIP, IP-BIP and MPI-BIP are available [26].

The measurements presented in this section have been performed on Pom.

3.4.2 The cache effect

The goal of this section is to show the importance of tiling on mono-processor, and to evaluate the optimal height for sub-tiles (h_{sub_tile}). For this purpose, we have performed performance measurements on a rectangular iteration space of size $100 \times H$, executed on a mono-processor without any orthonormalization phase.

As one can see on Figure 10, execution on a non-tiled program shows two levels of cache: there are three steps $H < H_{L1} = 300$, $500 < H < 3000$ and $H > 5000$ corresponding respectively to the cache L1, the cache L2 and the memory.

Because of the parallelogram-shape of the sub-tiles, the width w has to be taken into account in the choice of a good height h_{sub_tile}. Hence, we aim that $w + h_{sub\_tile} < H_{L1} = 300$. Since $w \leq 4n_{ortho} \lesssim 100$, we have chosen $h_{sub\_tile} = 100$.

We then performed tiling of our sequential program with parallelogram tiles of size $w \times h_{sub\_tile}$: we can remark a slight overhead (for $H > 3000$) corresponding to the initial load of data in the memory at the beginning of each tile. This overhead might be slightly (not exactly because tiles are not rectangular-shaped but parallelogram-shaped...) inversely proportional to the width of the tile (see Figure 11). We can remark on Figure 11, that even for the tiled version of our program the memory access has an impact for values of $H > 3000$. In fact, hierarchical tiling would be necessary using "super-tiles" of size 1000×1000, which is not possible due to orthonormalizations every $4n_{ortho} \lesssim 100$ iterations.

For the rest of the experiments, in the case $\frac{H}{P} > 200$, the iteration space will be automatically tiled with tiles of size $w_{sub\_tile} \times h_{sub\_tile} = w \times 100$.

3.4.3 Optimal tile size on a mono-processor

We have seen the impact of tiling on the computation time. Because of the size of cache L1, a good value for the tiles height is $h_{sub\_tile} = 100$. Experiments presented in this section show that a simple model (computation time=calculation time + memory access) provides a good approximation. However, we will see in the next section that the impact of communications per column is smaller than 200 times the average execution of one task. More formally,

$$\frac{\text{communication time for one column}}{\text{average computation time of one column}} < \frac{200}{h} = \frac{200P}{H}$$

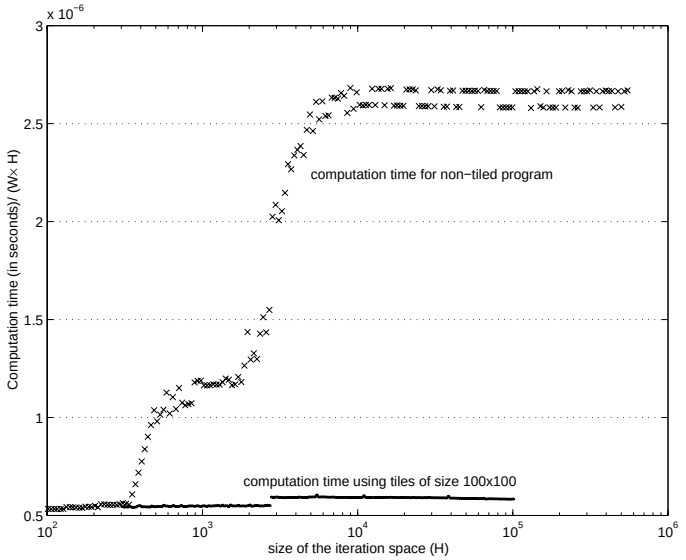


Figure 10: Time in seconds to perform one computational unit. The number of iterations is fixed to 100. The number of particles varies from 10^2 to 10^6. For domains of size $500 < H < 3000$ the tiling speeds-up the computation of a ratio more than 2. For domains of size $H > 5000$, the speed-up is more than 4.3.

Consequently, for domain size $H \geq 4000P$, communications will be negligible (5 %).

Hence, for locality purpose, tiles should be as large as possible (see figure 11). However, as we increase w, the amount of dummy tasks ($w \times (w-1)$) are increased and one should find the best trade-off between providing locality and computing less dummy tasks. Let us formalize this idea.

Let us denote by τ_{L2} (respectively τ_{Mem}) the average time necessary to fetch a data from cache L2 to cache L1 (respectively from the memory to cache L2); let us also denote by τ_{calc} the average calculation time of one task. Then, the computation time of a rectangle of size $h \times w$ is given by the formula (where $\delta_{h>3000}$ is 1 if $h > 3000$ and 0 otherwise).

$$T_{calc}(h,w) \simeq hw \times \left((\tau_{L2} + \delta_{h>3000}\tau_{Mem}) \times \frac{1}{w} + \tau_{calc} \right).$$

That can be experimentally observed on Figure 11. Consequently the execution time of parallelepiped tile (rectangle of size $h \times w$ plus two triangles of height w) is

$$T_{calc}(h,w) \simeq hw \times \left((\tau_{L2} + \delta_{h>3000}\tau_{Mem}) \times \left(\frac{1}{w} + \frac{2}{h} \right) + \left(1 + \frac{1}{h}(w-1) \right) \tau_{calc} \right)$$

That can be experimentally observed on Figure 12 for values $w \gtrsim 15$. For $w \lesssim 15$, a more precise model of memory access than the simple formula $(\tau_{L2} + \delta_{h>3000}\tau_{Mem})$ would be necessary. But, as we will see on the next section, for h values for which w should be chosen as small, the communication overhead has a non-negligible impact. In this context, it becomes hazardous to formulate analytically the average execution time.

3.4.4 The communication overhead

In this section, we study the impact of communications on the computation time. Because our study deals with small-scale message passing, we might use a simple model [27] for communication made of two components: the start-up β, and the communication itself $l\tau$ proportional to its size l. Usually, $\beta \gg \tau$, which motivates tiling because it increases the granularity of computation. Here, for a domain of size $W = 4T/dt$, tiled with tiles of size $w \times h = w \times H/P$, there are W/w groups of communications of size proportional to

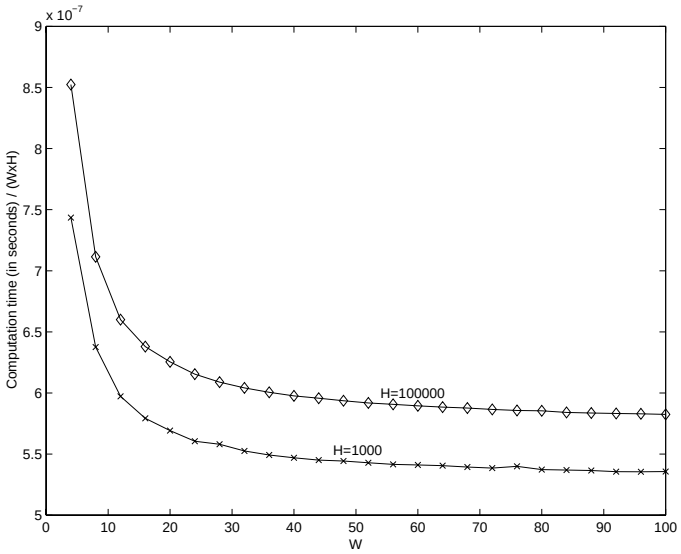


Figure 11: Influence of the width w on the locality of data references for the tiled sequential program: the iteration space is of size $w \times h$, and tiles of size $w_{sub\_tile} \times h_{sub\_tile} = w \times 100$. For $h = 10^3$, all data remain in cache L1 or L2. For $h = 10^5$, the impact of tiling is more important because the data are initially in the memory, and should be fetched from memory to cache L2.

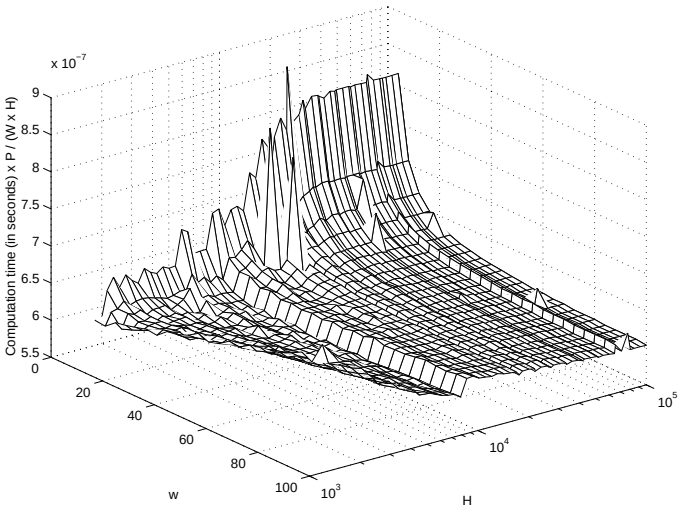


Figure 12: Influence of the width w on the total computation time (without any communications) versus the domain size H for parallel program executed on three processors.

w. If we normalize by W, we obtain a communication cost per column of $\frac{\beta}{w} + \tau$. However, Fig. 13 do not confirm this approach:

- We can observe two different phases: in practical, for communications larger than 960 bytes ($w \geq 20$), messages are divided into sub-messages (by the communication interface) which involves an overhead. This can be observed on the curves by a step between $w = 16$ and $w = 20$.
- The communication overhead is very sensitive to the amount of data used by the program (see figure 14):
  1. Indeed, the communication interface context might be removed from the cache by the computations between each communication phase. That would explains why the communication overhead (computation time with communications minus computation time without communications) is larger than the "*pure communication time*" (the program executed without any calculation).
  2. When the amount of data per processor is just below a transitional period ($h = \frac{H}{P} \approx 2000$), the communication might slow down the computation part, by increasing the amount of data.

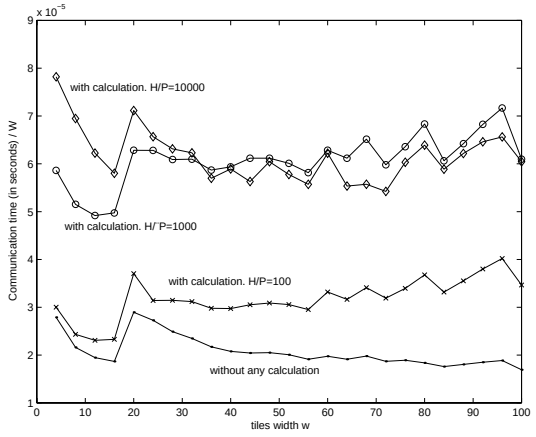


Figure 13: Evaluation of the communication overhead. Two tests are performed: one ("*without calculations*") corresponds simply to execute the program in which calculations have been commented (only communications are performed). The second ("*with calculations*") corresponds to the difference between the program with communication and without communications (for this last case, even if the program result is wrong, we have checked that no exception occurs..). It evaluates the actual communication overhead. Since, the communication overhead depends on the value of $\frac{H}{P}$ (see figure 14), we have performed this last test with different domain sizes: $h = \frac{H}{P} \in \{10^2, 10^3, 10^4\}$.

3.4.5 The optimal parameters

From the previous remarks, we can summarize the general behavior of the program to the following points:

- For a small domain size $\frac{H}{P}$, the "redundant-work" overhead is very significant. Hence, w should be smaller than 20. In that context, larger the tiles, smaller the communication overhead, faster the calculation time (because of locality), and larger the redundant work overhead. Here, communication time and calculation time are predominant. The best tiles width is then $w = 16$ (see figure 15).
- For very large domains, either the "redundant-work" overhead and the communication overhead are negligible. Hence, the best tiles width is $w = 4n_{ortho} = 100$ (see figure 15).
- Between these two extreme cases, it is difficult to define a formal solution because of the irregularity of the communication overhead; the best solution might consist on testing all the possibilities (only 25 possibilities for w) with fixed $W = 4\frac{T}{dt} = 500$ (see figure 16).

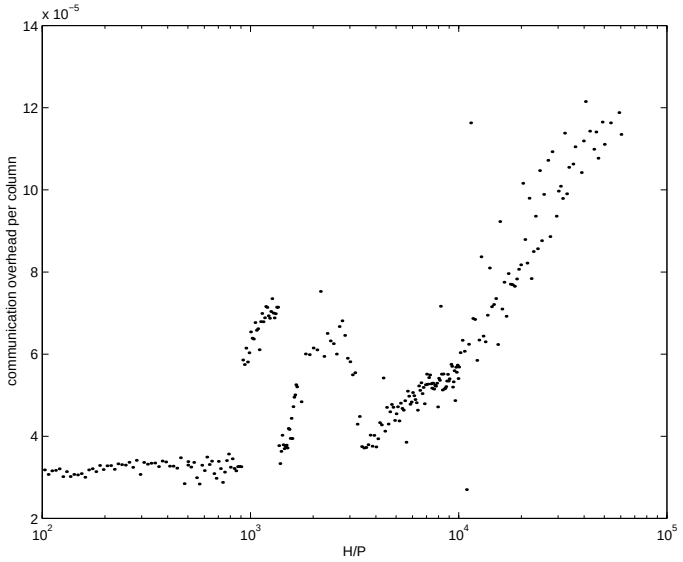


Figure 14: Influence of the amount of data per processor on the communication overhead. $P = 6$, $\frac{H}{P}$ varies from 10^2 to 10^5, and $w = 32$.

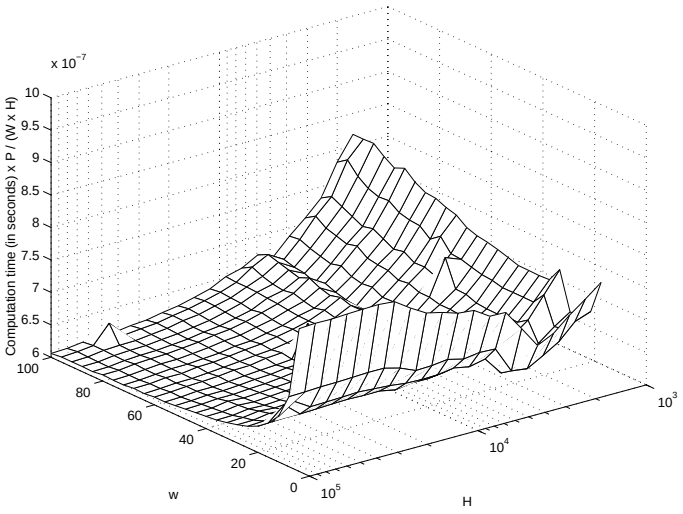


Figure 15: Influence of the width (w) on the total computation time versus H. Iteration domain size is $w \times H$; $h = \frac{H}{P}$; $P = 6$.

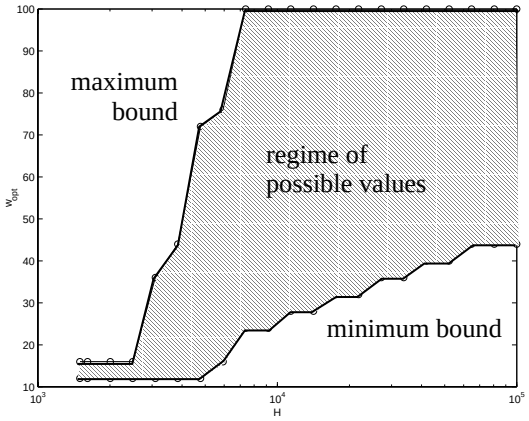


Figure 16: Optimal values of w versus H on $P = 6$ processors. Let us denote by $t_{calc}(H, w)$ the normalized computation time. For a given value of domain height H, we denote by w_{opt} the optimal tile's width: $t_{calc}(H, w_{opt}) = \min_w t_{calc}(H, w)$. Then, by allowing an error of 2%, the minimum bound corresponds to $\min\{w,\ t_{calc}(H, w) < t_{calc}(H, w_{opt}) \times 1.02\}$ and the maximum bound to $\max\{w,\ t_{calc}(H, w) < t_{calc}(H, w_{opt}) \times 1.02\}$.

3.5 Performance results

In this section, we present performance results obtained on the clusters described in Paragraph 3.4.1. Unfortunately, because no reservation system is available on this machine, it is extremely difficult to get all the processors together in a dedicated mode. Hence, we had to dynamically balance the loads on processors during the execution: the height h_j of the tile computed by processor P_j was chosen to verify the constraint $\forall j,\ T_{calc}(h_j, w, P_j) \simeq Constant$. The imperfection of the system (miss of reactivity) explains the irregularity observed on the curves. Moreover, because we were interested mainly in large domain size, we have restricted the experiments to a tiles' width of $w = 100$. Instead, smaller values would have provided better results on small iteration domains (cf Figure 16). However, results are still very good. Two experiments have been performed:

- one on 11 homogeneous processors of the Pom,
- the other one on 25 heterogeneous processors: 14 processors from Pom with an average cycle time $t_{seq}(pom) = 3.99 \times 10^{-7}$; 11 processors from PoPC with an average cycle time $t_{seq}(popc) = 7.09 \times 10^{-7}$.

For each of this experiments, H varies from 4000 to 10^6. The iteration space have a size of $W \times H = 4\frac{T}{dt} \times H = 500 \times H$. We have reported,

1. the average execution time of one task $t_{calc} = \frac{T_{exe\_para}}{WH}$.
2. the speedup of the parallel execution.

3.5.1 Experiences on 11 homogeneous processors

Results are reported on figures 17 and 18. Two curves are represented on Figure 17:

- the *average execution time per task*, corresponds to the total computation time divided by the size of the iteration domain $W \times H$.
- the *optimal execution time* (horizontal line) is defined as follow: for a large set of domain size, sequential execution time per task is measured. Among all those values, we took the minimum one and divided it by 11.

Figure 18 represents the speedup for our parallel execution: for a given number of particles H, the program is executed both in sequential and in parallel. The speedup is the ratio of those two values. For this example, a good speedup should be around 11.

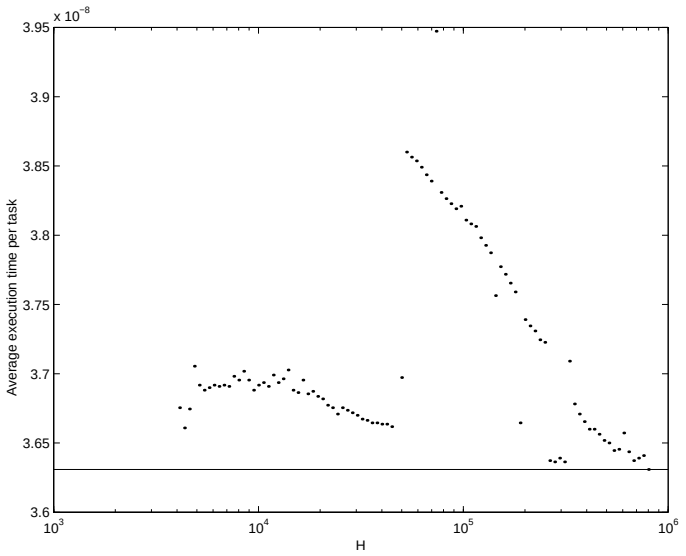


Figure 17: Average execution time per task on 11 processors from the Pom, for a problem size varying between $H = 4000$ and $H = 10^6$. The horizontal line represents the optimal execution time as defined in the text.

Execution time curve (Figure 17) presents a step (around $H = 4000$) that corresponds to the cache limit. Because sequential execution time curve is roughly identical but scaled by a factor of $P = 11$, the ratio between sequential execution time and parallel one represented in Figure 18 contains two bumps (that exceed the "optimal" value 11).

3.5.2 Experiences on 25 heterogeneous processors

Let us suppose that we use all the available resources simultaneously, i.e. processors from PoPC in addition to processors from Pom. In that case, nor the processors, neither the network are homogeneous. Hence, the work should be load balanced, and the heterogeneity of the network should be taken into account. If not, the execution time of the program will depend only on the slowest parts of the system (see [5, 3] for a reflection on load balancing problems).

For clarity, let us consider a toy example : suppose we have two parallel machines c_0 and c_1 made respectively of 11 processors from PoPC and 14 processors from Pom. Suppose the network between those two clusters is 10 times slower then the network inside each cluster.

A solution consists on considering each cluster has a processor of cycle time $t_{seq}(c_0) = t_{para}(popc) \simeq \frac{t_{seq}(popc)}{11}$ and $t_{seq}(c_1) = t_{para}(pom) \simeq \frac{t_{seq}(pom)}{14}$. Then, tile the iteration space consequently as done for an homogeneous network. Finally, each tile is distributed over the processors of each cluster. Figure 19 represents this strategy on this example.

Of course, this method is recursive, and p_i could represent itself a cluster of processors etc...

The results for the homogeneous problem clarifies therefore that this solution is useful only if the communication overhead is not negligible compared to the computation time. It might be the case, when running the application on the computational grid (see [24] for a discussion on meta-computing issues). In such case, n_{ortho} should be large enough to compensate the communication overhead...

Here, we aim at executing the program on very large domains, and the communication overhead between two clusters is not large enough to justify the use of a hierarchical tiling method (see [8] for a discussion

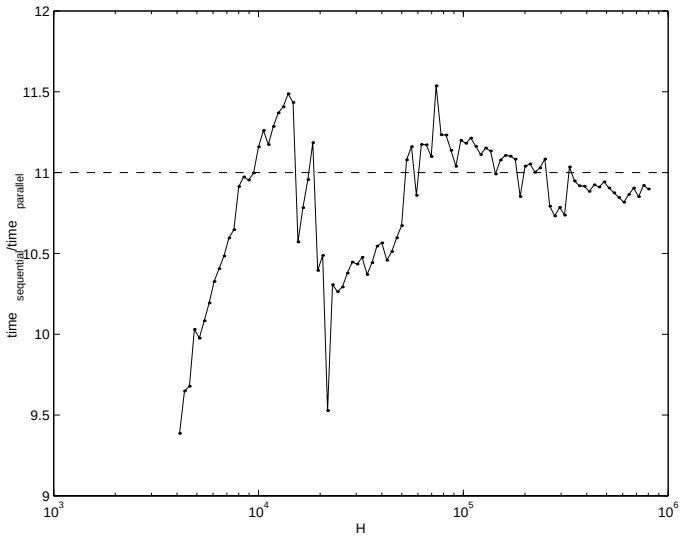


Figure 18: Speedup for parallel execution on 11 processors from Pom versus sequential execution.

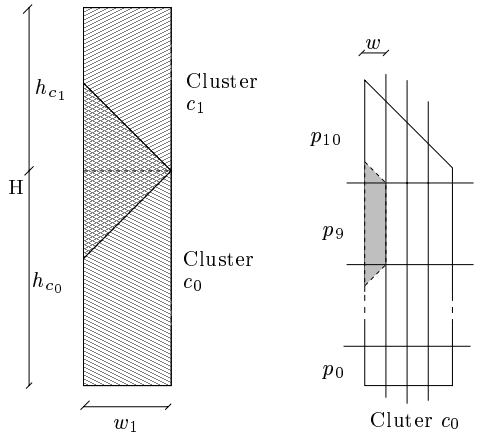


Figure 19: Hierarchical tiling for two clusters of processors. Here $h_{c_1} = \frac{t_{seq}(c_0)}{t_{seq}(c_0)+t_{seq}(c_1)}$ and $h_{c_0} = \frac{t_{seq}(c_1)}{t_{seq}(c_0)+t_{seq}(c_1)}$.

on hierarchical tiling). Hence, we chose to implement our program just as if the network was homogeneous. Results are reported in figures 20 and 21.

Two curves are represented in Figure 20:

- The average execution time per task corresponds to the total execution time divided by the iteration domain size $W \times H = 500 \times H$.
- the *extrapolated optimal execution time* (horizontal line) is evaluated as follow: the minimum (over all possible domain size) average sequential execution time per tile is measured on one node from the Pom (denoted by $t_{seq}(pom)$), and on one node from the PoPC (denoted by $t_{seq}(popc)$). The average sequential time is then extrapolated by the formula

$$\frac{1}{\frac{14}{t_{seq}(pom)} + \frac{11}{t_{seq}(popc)}} \simeq 1.97 \times 10^{-8}$$

.

Figure 21 represents the speedup for our parallel execution versus the extrapolated sequential execution.

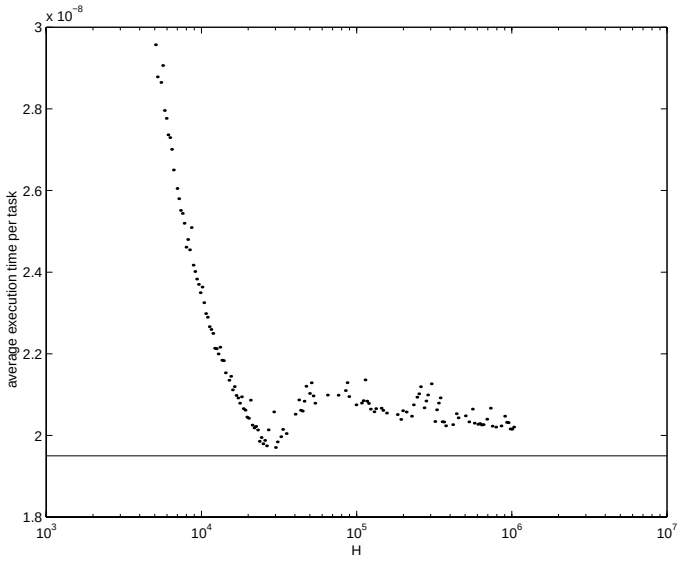


Figure 20: Execution time on 25 processors, 11 from the PoPC and 14 from the Pom. Domain size varies from $H = 5000$ to $H = 10^6$. The horizontal line represents the extrapolated optimal execution time as defined in the text.

As in the case of Figure 17, the execution time curve (Figure 20) contains a step (around $H = 3000$) that corresponds to the the cache limit. For small domain size, the speedup is worst than on 11 processors due to the communication time more costly between PoPC and Pom than between two nodes of the Pom.

4 Results

In this article, taking advantage of the code developed and presented in Section 3, we would like to briefly test the following issue: Is there a dependence of the maximum Lyapunov exponent on the length of the chain N. In an old but well-known article [22], Livi, Pettini and Ruffo have established the fact that the Lyapunov are dependent only on the density of energy and not on the size of the chain N. This fact was very recently disputed by Searles, Evans and Isbister [30]. More precisely they argued that the largest Lyapunov exponent diverges *logarithmically* with the number of particles in the system. They considered the Fermi-Pasta-Ulam chain but also other systems in two and three dimensions. This is of course a crucial question since most

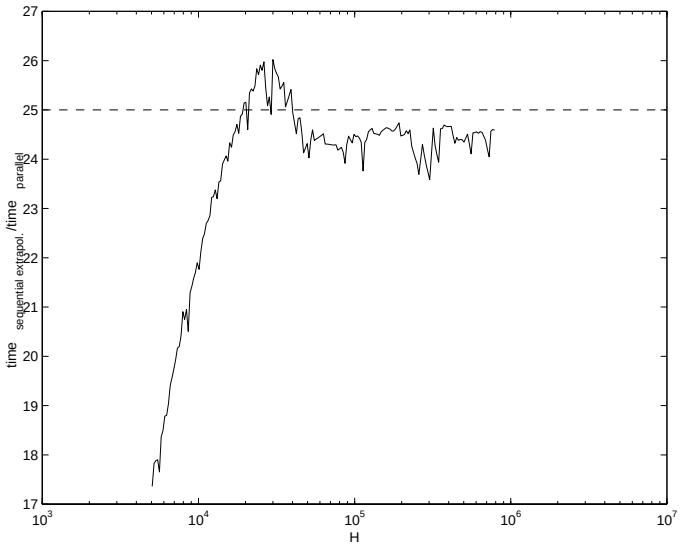


Figure 21: Speedup for our parallel execution on 25 processors versus the extrapolated sequential execution.

works on the FPU chain are considering, of course, a *finite* number of sites but the aim is to draw conclusions in the thermodynamic limit ($N \to \infty$).

Their main conclusion is that Livi *et al.* were using too small chains to note this weak divergence and thanks to the powerful computers of today, they claimed to be able to check the disagreement. However the logarithmic divergence was very weak and their results were disputed, that's why we have decided to check it with our code, which allows us to attain longer chains and longer simulations time.

As shown by Fig. 22, we have computed the Lyapunov exponents over very long time of order $t \simeq 10^6$, which means more than 10^8 timesteps. We made simulations for different sizes of the chains from 10^2 to 10^6.

The first important but usual remark is that the Lyapunov should be computed only after a "sufficiently long time". If the largest Lyapunov exponent is a measure of the greatest dynamical instability, it takes time for the eigenvector to seek out the direction of instability. Also, too early measures of the values of δ_1 would disappear only as $1/t$ in the the computation of λ_1, defined by Eq. (5).

The second point is that Searles et al. used the Gear predictor-corrector (4th order) algorithm which is not symplectic, but we see here that the best 4th order symplectic algorithm confirms their finding: there is presumably a very small but existing dependence on the length of the chain N, at least in the energy and length domains considered.

This is an unexpected result that should be confirmed by considering higher values of the energy density, clearly above the so-called stochasticity threshold, which could explain this very weak divergence with N. Higher values of the energy density will lead to higher Lyapunov exponent and therefore to a faster convergence toward equilibrium. As the maximal time would be smaller, this would allow, of course, to test longer chains. Work along this line is in progress thanks to this implementation.

5 Conclusion

In this paper, we have applied tiling techniques to a practical problem of dynamical system theory, namely the numerical Lyapunov calculation. This work gave the opportunity to check the veracity of models often used for tiling optimizations. In particular communication cost models that are a simple function of the communicated data volume l (like $\beta + l\tau$) have been shown inaccurate. The major objective of this article was to present a simple-to-implement yet very efficient solution for tiling the iteration space (cyclic and non-cyclic) of explicit integration schemes of differential equations. This solution can be applied on all algorithms that present a recurrence on all but one directions, which is the case, in particular, of some SOR and DSP

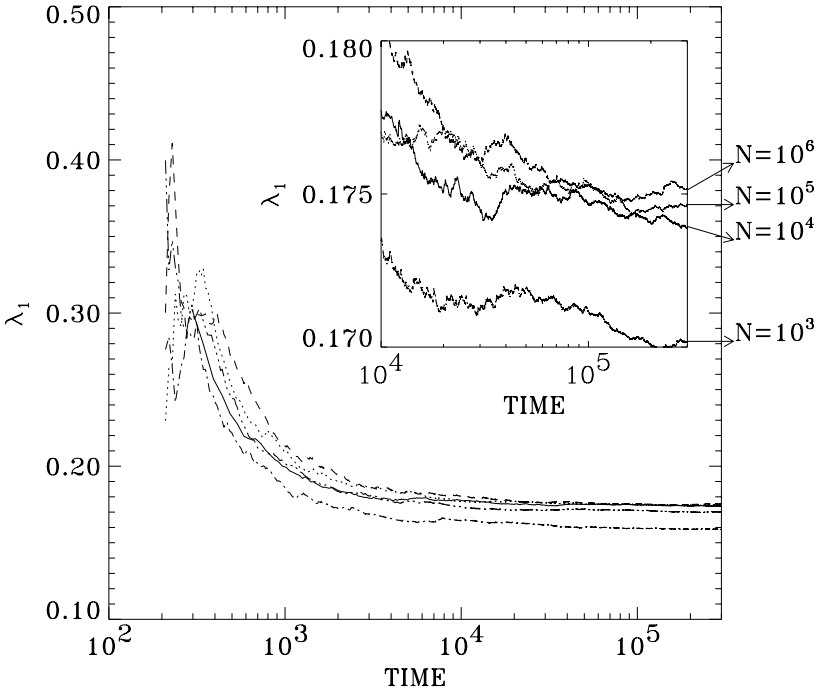


Figure 22: Evolution of the Lyapunov exponents in the case of an energy per particle of $e = 19.53$ for different lengths N of the chains. The curves corresponds from top to bottom to $N = 10^6$, 10^5, 10^4, 10^3 and 10^2. The inset shows more precisely the three largest cases.

algorithms. The optimization of the tile size has been decomposed into different successive phases, sorted in decreasing importance order, and which have been studied in this article. Our approach differs from classical ones that usually aim to solve tiling optimization problem in a single step. We strongly believe that our "fuzzy" approach should be used by future automatic partitioning algorithms. Finally, except maybe for the orthonormalization which is quite specific to our physical study, our implementation is scalable and well-suited for massively parallel platforms.

However, taking a physical point of view, this implementation confirmed the logarithmic divergence of the largest Lyapunov exponent with the number of particles in the system. Also this implementation is the suitable tool to study the low density energy regime of the Fermi-Pasta-Ulam chain. Indeed, in the last decade, the behavior of the largest Lyapunov exponent as a function of the energy density was recognized as a key element to understand the Fermi-Pasta-Ulam paradox and the transition to the statistical mechanical behavior. In particular, different authors have reported the existence of the so-called *stochasticity threshold*. Above this critical energy density [25], the transition toward energy equipartition is fast whereas below (this was indeed the case for the original article by FPU [18]), the dynamics seems to be in contradiction with this statistical mechanics theorem.

The question whether there is no transition or an extremely slow transition was debated extensively in the literature and the need of a clear demonstration is still present. Recently, using a nice Riemannian geometrical approach [9], Casetti, Livi and Pettini have obtained an analytical expression in clear agreement with available numerical data; another more phenomenological approach gave also excellent results [13]. However, as these methods are not completely rigorous, extensive and precise measures at very low energy densities are still importantly requested. Work along these lines is in progress.

Acknowledgments The authors thank Marie Rastello, Stefano Ruffo and Alessandro Torcini for very helpful discussions, Hervé Gilquin for its very efficient assistance when using the Sun-Enterprise of PSMN [14]. Finally, the authors would also like to thanks Michael Winter for his work on parallelepiped-shaped tiles.

References

[1] R. Andonov, S. Rajopadhye, and N. Yanev. Optimal orthogonal tiling. In *Europar'98*, pages 480–490, September 1998.

[2] Utpal Banerjee. An introduction to a formal theory of dependence analysis. *The Journal of Supercomputing*, 2:133–149, 1988.

[3] Vincent Boudet, Fabrice Rastello, and Yves Robert. A proposal for a heterogeneous cluster ScaLAPACK (dense linear solvers). In Hamid R. Arabnia, editor, *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, pages 1285–1291. CSREA Press, 1999. Extended version available as LIP Technical Report RR-99-17.

[4] Pierre Boulet, Alain Darte, Tanguy Risset, and Yves Robert. (Pen)-ultimate tiling? *Integration, the VLSI Journal*, 17:33–51, 1994.

[5] Pierre Boulet, Jack Dongarra, Fabrice Rastello, Yves Robert, and Frederic Vivien. Algorithmic issues on heterogeneous computing platforms. *Parallel Processing Letters*, 9(2):197–213, 1999. Extended version available as LIP Technical Report RR-98-49.

[6] Pierre-Yves Calland and Tanguy Risset. Precise tiling for uniform loop nests. In P. Cappello et al., editors, *Application Specific Array Processors ASAP 95*, pages 330–337. IEEE Computer Society Press, 1995.

[7] P.Y. Calland, J. Dongarra, and Y. Robert. Tiling with limited resources. In L. Thiele, J. Fortes, K. Vissers, V. Taylor, T. Noll, and J. Teich, editors, *Application Specific Systems, Achitectures, and Processors, ASAP'97*, pages 229–238. IEEE Computer Society Press, 1997. Extended version available on the WEB at http://www.ens-lyon.fr/~yrobert.

[8] L. Carter, J. Ferrante, S. F. Hummel, B. Alpern, and K.S. Gatlin. Hierarchical tiling: a methodology for high performance. Technical Report CS-96-508, University of California at San Diego, San Diego, CA, 1996. Available at http://www.cse.ucsd.edu/~carter.

[9] L. Casetti, R. Livi, and M. Pettini. Gaussian model for chaotic instability of hamiltonian flows. *Physical Review Letters*, (74):375–378, 1996.

[10] Daniel Chavarra-Miranda, Alain Darte, Robert Fowler, and John Mellor-Crummey. Generalized multipartitioning. In *Second Annual Los Alamos Computer Science Institute (LACSI) Sy mposisum*, Santa Fe, NM, October 2001.

[11] T. Cretegny, T. Dauxois, S. Ruffo, and A. Torcini. Localization and equipartition of energy in the β-fpu chain : chaotic breathers. *Physica D*, (121):109–126, 1998.

[12] Alain Darte, Georges-André Silber, and Frédéric Vivien. Combining retiming and scheduling techniques for loop parallelization and loop tiling. *Parallel Processing Letters*, 7(4):379–392, 1997.

[13] T. Dauxois, S. Ruffo, and A. Torcini. Modulational estimate for the maximal lyapunov exponent in fermi-pasta-ulam chain. *Physical Review E*, (56):R6229–R6232, 1997.

[14] Pôle Scientifique de Modélisation Numérique. Psmn. World Wide Web document, URL: `http://psmn.ens-lyon.fr`.

[15] Frederic Desprez, Jack Dongarra, Fabrice Rastello, and Yves Robert. Determining the idle time of a tiling: new results. *Journal of Information Science and Engineering*, 14:167–190, 1998.

[16] N. Lorenz E. Deterministic nonperiodic flow. *Journal of athmospheric Science*, (20):130, 1963.

[17] G. Benettin et al. Lyapunov characteristic exponents for smooth dynamical systems and for hamiltonian systems; a method for computing all of them. *Meccanica*, (9):21, March 1980.

[18] E. Fermi, J. Pasta, and S. Ulam. Science laboratory report no. la-1940 (1955), unpublished. *University of Chicago Press*, 2:978, 1965.

[19] K. Högstedt, L. Carter, and J. Ferrante. Determining the idle time of a tiling. In *Principles of Programming Languages*, pages 160–173. ACM Press, 1997. Extended version available as Technical Report UCSD-CS96-489, and on the WEB at http://www.cse.ucsd.edu/∼carter.

[20] A. J. Lichtenberg and M. A. Lieberman. *Regular and Chaotic Dynamics*. Springer Verlag, Berlin, 1992.

[21] Amy W. Lim and Monica S. Lam. Maximizing parallelism and minimizing synchronization with affine transforms. In *Proceedings of the 24th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 201–214. ACM Press, January 1997.

[22] R. Livi, A. Politi, and S. Ruffo. Distribution of characteristic exponents in the thermodynamic limit. *Journal of Physics A*, (19):2033–2040, 1986.

[23] R. I. MacLachlan and P. Atela. The accuracy of symplectic integrators. *Nonlinearity*, (5):541–562, 1992.

[24] Jean-Francois Mehaut and Yves Robert. Algorithms and tools for (distributed) heterogeneous computing: A prospective report. Technical Report RR-1999-36, LIP, ENS Lyon, August 1999.

[25] M. Pettini and M. Landolfi. Relaxation properties and ergodicity breaking in nonlinear hamiltonian dynamics. *Physical Review A*, (41):768–783, 1990.

[26] Loïc Prylli and Bernard Tourancheau. BIP: a new protocol designed for high performance networking on Myrinet. In *1st Workshop on Personal Computer based Networks Of Workstations (PC-NOW '98)*, volume 1388 of *Lect. Notes in Comp. Science*, pages 472–485. Held in conjunction with IPPS/SPDP 1998. IEEE, Springer-Verlag, April 1998.

[27] Loïc Prylli, Bernard Tourancheau, and Roland Westrelin. Modeling of a high speed network to maximize throughput performance: the experience of bip over myrinet. In Hamid R. Arabnia, editor, *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'98)*, volume II, pages 341–349. CSREA Press.

[28] Fabrice Rastello, Amit Rao, and Santosh Pande. Optimal task scheduling to minimize inter-tile latencies. In *International Conference on Parallel Processing (ICPP'98)*, pages 172–179. IEEE Computer Society Press, 1998.

[29] Fabrice Rastello and Yves Robert. Automatic partitioning of parallel loops with parallelepiped-shaped tiles. *IEEE Transaction on Parallel Distributed Systems*, 2001. to be published.

[30] D. J. Searles, D. J. Evans, and D. J. Isbister. The number dependence of the maximum lyapunov exponent. *Physica A*, (240):96–107, 1997.

[31] Michael E. Wolf and Monica S. Lam. A loop transformation theory and an algorithm to maximize parallelism. *IEEE Trans. Parallel Distributed Systems*, 2(4):452–471, October 1991.