Scalable Quasineutral solver for gyrokinetic simulation

Guillaume Latu, Virginie Grandgirard, Nicolas Crouseilles, Guilhem Dif-Pradalier

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Scalable Quasineutral solver for gyrokinetic simulation

G. Latu — V. Grandgirard — N. Crouseilles — G-Dif. Pradalier

N° 7611

Mai 2011

Domaine 1

ISSN 0249-6399 ISRN INRIA/RR--7611--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
NANCY - GRAND EST

Scalable Quasineutral solver for gyrokinetic simulation

G. Latu * †, V. Grandgirard *, N. Crouseilles †, G-Dif. Pradalier‡

Domaine : Mathématiques appliquées, calcul et simulation
Équipes-Projets CALVI

Rapport de recherche n° 7611 — Mai 2011 — 15 pages

Abstract: Modeling turbulent transport is a major goal in order to predict confinement issues in a tokamak plasma. The gyrokinetic framework considers a computational domain in five dimensions to look at kinetic issues in a plasma. Gyrokinetic simulations lead to huge computational needs. Up to now, the gyrokinetic code GYSELA performed large simulations using a few thousands of cores. The work proposed here improves GYSELA onto two points: memory scalability and execution time. The new solution allows the GYSELA code to scale well up to 64k cores.

Key-words: Quasineutrality solver, Gyrokinetics, MPI, OpenMP

* CEA Cadarache, 13108 Saint-Paul-les-Durance Cedex
† INRIA Nancy-Grand Est & Université de Strasbourg, 7 rue Descartes, 67000 Strasbourg
‡ UCSD, La Jolla, CA 92093, USA

Centre de recherche INRIA Nancy – Grand Est
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy
Téléphone : +33 3 83 59 30 00 — Télécopie : +33 3 83 27 83 19

Solveur Quasi-neutre extensible pour les simulations gyrocinétiques

Résumé : La modélisation du transport turbulent est un point clef pour prédire les propriétés de confinement d'un plasma de fusion. La théorie gyrocinétique propose une description à 5 dimensions permettant de calculer et comprendre les effets cinétiques dans un plasma. Les simulations gyrocinétiques conduisent à des coûts en calcul réellement prohibitifs. Jusqu'à maintenant, le code gyrocinétique GYSELA réalisait de grosses simulations en utilisant quelques milliers de cœurs de calcul. Le travail proposé ici améliore GYSELA sur deux aspects: l'extensibilité mémoire et le temps d'exécution. La nouvelle solution permet au code GYSELA d'être opérationnel et scalable jusque 64k cœurs au moins.

Mots-clés : Solveur quasi-neutre, Gyrocinétique, MPI, OpenMP

1 Introduction

To have access to a kinetic description of the plasma dynamics inside a tokamak, one usually needs to solve the Vlasov equation nonlinearly coupled to Maxwell equations. Then, a parallel code adressing the issue of modelling tokamak plasma turbulence needs to couple a parallel Vlasov solver with a parallel field solver. On the first hand, a Vlasov solver moves the plasma particles forward in time. On the other hand, the role of the field solver is to give the electromagnetic fields generated by a given particles setting in phase space. In this paper, we focus on the study of the field solver embedded in a gyrokinetic code, namely GYSELA. An algorithm is presented here that has excellent performance up to a few thousands of cores. An adapted communication scheme is introduced to optimize the communication costs for exchanging information between the Semi-Lagrangian Vlasov solver and the field solver (the two main parts of the code). Using 64k cores, the field solver tends towards an asymptotic time cost. Nevertheless, its small computation cost compared to other parts of the code, allows the GYSELA code to get good performance up to 64k core with 78% of relative efficiency. As a consequence, a well parallelized solution is finally obtained. In the last sections, performance is evaluated both in term of execution time and in term of memory scalability.

2 Gyrokinetic model

2.1 Parallel solving of Vlasov equation

Our gyrokinetic model considers as a main unknown a distribution function $\bar{f}$ that represents the density of ions at a given phase space position. This function depends on time and 5 other dimensions. First, r and θ are the polar coordinates in the shortest cross-section of the torus (called poloïdal section), while φ refers to the angle in the largest cross-section of the torus. Second, the velocity space is also discretized: $v_\parallel$ is the velocity along the magnetic field lines (one has $v_\parallel = \mathrm{d}\varphi/\mathrm{d}t$), and μ the magnetic moment corresponds to the action variable associated with the gyrophase. The time evolution of the guiding-center 5D gyroaveraged distribution function $\bar{f}_t(r, \theta, \varphi, v_\parallel, \mu)$ is governed by the so-called gyrokinetic equation [1, 3]:

$$\frac{\partial \bar{f}}{\partial t} + \frac{\mathrm{d}r}{\mathrm{d}t}\frac{\partial \bar{f}}{\partial r} + \frac{\mathrm{d}\theta}{\mathrm{d}t}\frac{\partial \bar{f}}{\partial \theta} + \frac{\mathrm{d}\varphi}{\mathrm{d}t}\frac{\partial \bar{f}}{\partial \varphi} + \frac{\mathrm{d}v_\parallel}{\mathrm{d}t}\frac{\partial \bar{f}}{\partial v_\parallel} = 0. \tag{1}$$

In this Vlasov gyrokinetic equation, μ acts as a parameter because it is an adiabatic motion invariant. Let us denote by N_μ the number of μ values, we have N_μ independant Eq. 1 to solve at each time step. The function $\bar{f}$ is periodic along θ and φ. Vanishing perturbations are imposed at the boundaries in the non-periodic directions r and $v_\parallel$.

GYSELA is a global nonlinear electrostatic code which solves the gyrokinetic equations in a five dimension phase space with a semi-Lagrangian method [1, 2]. The Semi-Lagrangian time integration technique [8] couples the Lagrangian and Eulerian points of view. The main advantage offered by the semi-Lagrangian technique is that time steps are not restricted by the CFL condition. We combine this scheme with a second order in time Strang splitting method. The resolution of the Vlasov Eq. (1) is not the topic of this paper and we refer

the reader to [1, 4] for detailed descriptions. We will only recall a few issues concerning the parallel domain decomposition used by this Vlasov solver.

Large data structures are used in Vlasov and Field solver: the 5D data $\bar{f}$, and the electric potential Φ which is a 3D data (and also some of derivatives of the gyroaverage of Φ along spatial dimensions as we will see). The sizes of these structures are parametrized by the discretization along the dimensions. Let N_r, N_θ, N_φ, $\mathrm{N}_{v_\parallel}$, N_μ be respectively the number of points in each dimension r, θ, φ, $v_\parallel$, μ. The size of 5D and 3D data are $(\mathrm{N}_r\,\mathrm{N}_\theta\,\mathrm{N}_\varphi\,\mathrm{N}_{v_\parallel}\mathrm{N}_\mu)$ and $(\mathrm{N}_r\,\mathrm{N}_\theta\,\mathrm{N}_\varphi)$.

In the Vlasov solver, as μ acts as a parameter, we give the responsibility of each value of μ to a given set of MPI processes [4] (a MPI communicator). We fixed that there are always N_μ sets of processes, such as only one μ value is attributed to each communicator. Within each set, a 2D domain decomposition allows us to attribute to each MPI Process a subdomain in (r, θ) dimensions. Thus, a MPI process is then responsible for the storage of the subdomain defined by

$$\bar{f}(r = [i_{start}, i_{end}], \theta = [j_{start}, j_{end}], \varphi = *, v_\parallel = *, \mu = \mu_{value}) \ .$$

The parallel decomposition used in the Vlasov solver is initially set up thanks to the locally owned parameters list $(i_{start}, i_{end}, j_{start}, j_{end}, \mu_{value})$. They are derived from a classical block decomposition of the r domain of size $\mathrm{N_r}$ into $\mathrm{p_r}$ pieces, and of the θ domain of size N_θ into p_θ subdomains. It ends up that the numbers of MPI process used during one run is equal to $\mathrm{p_r} \times \mathrm{p}_\theta \times \mathrm{N}_\mu$. Inside each MPI process, OpenMP threads give access to fine-grained parallelism.

2.2 Quasineutrality equation

The quasi-neutrality equation and parallel Ampere's law close the self-consistent gyrokinetic Vlasov-Maxwell system. However, in an electrostatic code as GYSELA, the field solver reduces to the numerical resolution of the quasi-neutrality Poisson equation (3) (see [3]). It requires the solving of a 3-dimensional equation. In the form we are working on, this equation involves a non local term, the average of Φ along (θ, φ) dimensions, which penalizes the parallellization [5, 6].

One of the possible ways to numerically treat the quasi-neutrality equation consists in the use of Fast Fourier Transform, other choices can be to use multigrid method or a direct solver for example. Even if the FFT approach is not adapted to general geometries [7], if one has a periodic direction it remains a fast, simple and accurate method. Hence, we propose here a new Quasineutral solver based on FFT.

In tokamak configurations, the plasma quasineutrality (denoted QN) approximation is currently assumed [1, 3]. This leads to $n_i = n_e$ where n_i (resp. n_e) is the ionic (resp. electronic) density. On the one side, electron inertia is ignored, which means that an adiabatic response of electrons are supposed. On the other side, the ionic density splits into two parts. Using the notation $\nabla_\perp = (\partial_r, \frac{1}{r}\partial_\theta)$, the so-called linearized polarization density n_{pol} writes

$$n_{pol}(r, \theta, \varphi) = -\nabla_\perp \cdot \left[\frac{n_0(r)}{B_0} \nabla_\perp \Phi(r, \theta, \varphi) \right],$$

where n_0 is the equilibrium density, B_0 the magnetic field at the magnetic axis. Second, the guiding-center density n_{Gi} is

$$n_{Gi}(r, \theta, \varphi) = 2\pi \int B(r, \theta) d\mu \int dv_{//} J_0(k_\perp \sqrt{2\mu}) \bar{f}(r, \theta, \varphi, v_{//}, \mu), \qquad (2)$$

where B is the magnetic field, J_0 is the Bessel function and $k_\perp$ is the transverse component of the wave vector and $T_e(r)$ the electronic temperature. Hence, the QN equation can be written in dimensionless variables

$$-\frac{1}{n_0(r)}\nabla_\perp \cdot \left[\frac{n_0(r)}{B_0}\nabla_\perp \Phi(r,\theta,\varphi)\right] + \frac{1}{T_e(r)}\left[\Phi(r,\theta,\varphi) - \langle\Phi\rangle_{\theta,\varphi}(r)\right] = \tilde{\rho}(r,\theta,\varphi) \tag{3}$$

where the definition of $\tilde{\rho}$ is given by

$$\tilde{\rho}(r,\theta,\varphi) = \frac{2\pi}{n_0(r)}\int B(r,\theta)d\mu \int dv_{//} J_0(k_\perp\sqrt{2\mu})(\bar{f} - \bar{f}_{eq})(r,\theta,\varphi,v_{//},\mu). \tag{4}$$

In this last equation, $\bar{f}_{eq}$ denotes an electronic local Maxwellian equilibrium, and $\langle\rangle_{\theta,\varphi}$ the average onto the variables θ,φ.

Our QN solver includes two computation parts. First, the function $\tilde{\rho}$ is derived taking as input function $\bar{f}$ coming from Vlasov solver. Specific methods [6] are used to evaluate the gyroaverage operator J_0 on $(\bar{f} - \bar{f_{eq}})$ in Eq. (4). Second, the 3D potential Φ is found in computing discrete Fourier transforms of $\tilde{\rho}$, followed by solving of tridiagonal systems and inverse Fourier transforms. For this step, several approaches have been foreseen [5, 6]. The novel algorithm presented in this paper retains components of [5], while improving its performances.

In the following, we will refer to some 3D data structures as *3D field data*. They are produced and distributed over the parallel machine just after the QN solver, once we know the electric potential Φ. These field data set, namely:

$$\Phi, \frac{\partial J_0(k_\perp\sqrt{2\mu})\Phi}{\partial r}, \frac{\partial J_0(k_\perp\sqrt{2\mu})\Phi}{\partial \theta}, \frac{\partial J_0(k_\perp\sqrt{2\mu})\Phi}{\partial \varphi} \ ,$$

are distributed on processes in a way that exclusively depends on the parallel domain decomposition chosen in the Vlasov solver. Indeed, they are inputs for the gyrokinetic Vlasov Eq. (1), and they play a major role in terms $\frac{\mathrm{d}r}{\mathrm{d}t}, \frac{\mathrm{d}\theta}{\mathrm{d}t}, \frac{\mathrm{d}\varphi}{\mathrm{d}t}, \frac{\mathrm{d}v_\parallel}{\mathrm{d}t}$ not detailed here. In this paper, we will use [1] the domain decomposition in (r,θ) described in [4]. So we need that the subdomain owned by each MPI process has the form $(r = [i_{start}, i_{end}], \theta = [j_{start}, j_{end}], \varphi = *)$ for the 3D field data. The computation of 3D field data will be discussed in section 3.3.

3 Scalable algorithm for the QN solver

3.1 1D Fourier transforms method

The method that follows considers only 1D FFTs in θ dimension and uncouple hardly all computations in the φ direction. This allows for loop parallelization along this direction. The equation (3) averaged on (θ,φ) gives :

$$-\frac{\partial^2 \langle\Phi\rangle_{\theta,\varphi}(r)}{\partial r^2} - [\frac{1}{r} + \frac{1}{n_0(r)}\frac{\partial n_0(r)}{\partial r}]\frac{\partial \langle\Phi\rangle_{\theta,\varphi}(r)}{\partial r} = \langle\tilde{\rho}\rangle_{\theta,\varphi}(r) \tag{5}$$

[1] The 3D field data can be split across processes in another way in the GYSELA code. Another Vlasov solver is available that uses transposition of distribution function. In this Vlasov solver some 3D data are distributed along φ dimension. The technique shown here is also applied in this different setting, but it generates few more communications and is a little bit more complex.

A Fourier transform in θ direction gives:

$$\begin{array}{ll}\Phi(r,\theta,\varphi) = & \sum_u \hat{\Phi}^u(r,\varphi)e^{i\,u\,\theta} \\ \tilde{\rho}(r,\theta,\varphi) = & \sum_u \hat{\rho}^u(r,\varphi)e^{i\,u\,\theta}\end{array} \tag{6}$$

The equation (3) could be rewritten as:

for $u>0$:

$$-\frac{\partial^2\hat{\Phi}^u(r,\varphi)}{\partial r^2} - [\frac{1}{r}+\frac{1}{n_0(r)}\frac{\partial n_0(r)}{\partial r}]\frac{\partial\hat{\Phi}^u(r,\varphi)}{\partial r} + \frac{u^2}{r^2}\hat{\Phi}^u(r,\varphi) + \frac{\hat{\Phi}^u(r,\varphi)}{Z_i\,T_e(r)} = \hat{\rho}^u(r,\varphi) \tag{7}$$

for $u=0$:

$$\frac{\partial^2\langle\Phi\rangle_\theta(r,\varphi)}{\partial r^2} - [\frac{1}{r}+\frac{1}{n_0(r)}\frac{\partial n_0(r)}{\partial r}]\frac{\partial\langle\Phi\rangle_\theta(r,\varphi)}{\partial r} + \frac{\langle\Phi\rangle_\theta(r,\varphi)-\langle\Phi\rangle_{\theta,\varphi}(r)}{Z_i\,T_e(r)} = \langle\tilde{\rho}\rangle_\theta(r,\varphi) \tag{8}$$

The equation (5) allows one to directly find out the value of $\langle\Phi\rangle_{\theta,\varphi}(r)$ from the input data $\langle\tilde{\rho}\rangle_{\theta,\varphi}(r)$. Let us define the function $\Upsilon(r,\theta,\varphi)$ as $\Phi(r,\theta,\varphi) - \langle\Phi\rangle_{\theta,\varphi}(r)$. Substracting equation (5) to equation (8) leads to

$$-\frac{\partial^2\langle\Upsilon\rangle_\theta(r,\varphi)}{\partial r^2} - [\frac{1}{r}+\frac{1}{n_0(r)}\frac{\partial n_0(r)}{\partial r}]\frac{\partial\langle\Upsilon\rangle_\theta(r,\varphi)}{\partial r} + \frac{\langle\Upsilon\rangle_\theta(r,\varphi)}{Z_i\,T_e(r)} = \langle\tilde{\rho}\rangle_\theta(r,\varphi) - \langle\rho\rangle_{\theta,\varphi}(r) \tag{9}$$

Let us notice that $\hat{\Phi}^0(r,\varphi)=\langle\Upsilon\rangle_\theta(r,\varphi)+\langle\Phi\rangle_{\theta,\varphi}(r)$. So, the solving of equations (5) and (9) allows one to compute $\langle\Phi\rangle_{\theta,\varphi}(r)$,$\langle\Upsilon\rangle_\theta(r,\varphi)$ and $\hat{\Phi}^0(r,\varphi)$ from the quantities $\langle\tilde{\rho}\rangle_\theta(r,\varphi)$ and $\langle\tilde{\rho}\rangle_{\theta,\varphi}(r)$.

Then, the equation (7) is sufficient to compute $\hat{\Phi}^{u>0}(r,\varphi)$ from $\tilde{\rho}$. The different equations are solved using a LU decomposition precomputed once. Moreover, variable φ acts as a parameter in equation (7), allowing computations to be parallelized.

3.2 Data distribution issues

We assume two main hypothesis concerning the data distribution in the QN solver: 1) at the beginning of QN solver each process knows the values of a subdomain $\bar{f}(r=[i_{start},i_{end}],\theta=[j_{start},j_{end}],\varphi=*,v_{//}=*,\mu=\mu_{value})$ (output of the Vlasov solver), distributed over processes. In earlier works [5, 6], we used to simplify the problem of data dependancies in broadcasting the entire Φ data structure to all processes and to compute in each MPI process the derivatives of gyroaveraged electric potential redondantly. Nevertheless, this strategy leads to a bottleneck for large platforms (typically more than 4k cores). Indeed, the broadcast involves a communication amount that grows linearly with the number of processes, and the sequential nature of derivatives computation becomes also problematic. These two overheads are unnecessary, even they simplify the implementation of various diagnostics and reduces also the complexity of data management. In the version presented here, only a small subdomain of Φ is sent to each process at the end of the QN solver. Also, a distributed algorithm computes the derivatives of $J_0(k_\perp\sqrt{2\mu})\Phi$, as you will see in Algo. 2 because these quantites are inputs of Vlasov solver. Therefore, the domain decomposition for this 3D field derivatives must match the needs of the Vlasov solver.

3.3 Parallel algorithm descriptions

The algorithm presented in this paragraph describes a scalable parallel algorithm of the QN solver[2].

Algorithm 1: Parallel algorithm for the QN solver

Input : local block
$\bar{f}(r=[i_{start},i_{end}],\theta=[j_{start},j_{end}],\varphi=*,v_{//}=*,\mu=\mu_{value})$

(task 1*)*
Computation : $\tilde{\rho}_1$ by integration in $dv_{//}$ of $\bar{f}$
(parallelization in μ, r, θ)
Send local data $\tilde{\rho}_1(r=[i_{start},i_{end}]),\theta=[j_{start},j_{end}],\varphi=*,\mu=\mu_{value})$
Redistribute $\tilde{\rho}_1$ / **Synchronization**
Receive block $\tilde{\rho}_1(r=*,\theta=*,\varphi=[s^{\varphi}_{start},s^{\varphi}_{end}],\mu=[s^{\mu}_{start},s^{\mu}_{end}])$

(task 2*)*
for $\varphi=[s^{\varphi}_{start},s^{\varphi}_{end}]$ and $\mu=[s^{\mu}_{start},s^{\mu}_{end}]$ **do**
  (parallelization in μ,φ)
  Computation : from $\tilde{\rho}_1$ at one φ, computes $\tilde{\rho}_2$ applying $J_0(k_\perp\sqrt{2\mu})$
  (Fourier transform in θ, Solving of LU systems in r)
  $\forall\mu\in[s^{\mu}_{start},s^{\mu}_{end}]$
  Computation : $\tilde{\rho}_3$ for a given φ by integration in $d\mu$ of $\tilde{\rho}_2$
end
if $[s^{\mu}_{start},s^{\mu}_{end}]\neq[0,N_\mu-1]$ **then**
  Send local data $\tilde{\rho}_3(r=*,\theta=*,\varphi=[s^{\varphi}_{start},s^{\varphi}_{end}])$
  Reduce Sum $\tilde{\rho}\mathrel{+}=\tilde{\rho}_3$ / **Synchronization**
  Receive summed block $\tilde{\rho}(r=*,\theta=*,\varphi=[g_{start},g_{end}])$
end

(task 3*)*
for $\varphi=[g_{start},g_{end}]$ **do**
  (parallelization in φ)
  Computation : accumulation of $\tilde{\rho}$ values to get $\langle\tilde{\rho}\rangle_\theta(r=*,\varphi)$
end
Send local data $\langle\tilde{\rho}\rangle_{\theta(r=*,\varphi=[g_{start},g_{end}])}$
Broadcast of $\langle\tilde{\rho}\rangle_\theta$ / **Synchronization**
Receive $\langle\tilde{\rho}\rangle_{\theta(r=*,\varphi=*)}$

(task 4*)*
Computation : Solving of LU system to find $\langle\Phi\rangle_{\theta,\varphi}$ from $\langle\tilde{\rho}\rangle_\theta$, eq. (5)
for $\varphi=[g_{start},g_{end}]$ **do**
  (parallelization in φ)
  Computation : 1D FFTs of $\tilde{\rho}$ on dimension (θ)
  Computation : Solving of LU systems for $\hat{\Phi}$ modes ($\forall u>0$), eq. (7)
  Computation : Solving of LU system for $\langle\Upsilon\rangle_{\theta(r=*,\varphi)}$, eq. (9)
  Computation : Adding $\langle\Phi\rangle_{\theta,\varphi}$ to $\langle\Upsilon\rangle_{\theta(r=*,\varphi)}$ gives $\hat{\Phi}^0{}_{(r=*,\varphi)}$
  Computation : inverse 1D FFTs on $\hat{\Phi}^0$ and $\hat{\Phi}^{u>0}$ to get $\Phi_{(r=*,\theta=*,\varphi)}$
end
Send local data $\Phi(r=*,\theta=*,\varphi=[g_{start},g_{end}])$ N_μ times
Broadcast of values / **Synchronization**
Receive global data $\Phi(r=*,\theta=*,\varphi=[q_{start},q_{end}])$
Outputs : $\Phi(r=*,\theta=*,\varphi=[q_{start},q_{end}])$

This algorithm improves previous ones [1, 5, 6] in introducing a better work distribution, and also in reducing the final communication. At the end of the

[2] This formulation is not yet valid in toroïdal setting, cyclindrical geometry is used here.

solver, we distribute the electric potential Φ among processes, instead of broadcasting it.
The main idea of the algorithm is to get $\langle\Phi\rangle_{\theta,\varphi}$ for solving eq. (9), and then to uncouple computations of $\hat{\Phi}^u$ along φ direction in the Poisson solver (task 4 in the following list). Finally, each process send each locally computed $\Phi(r,\theta,\varphi)$ value to process that is responsible for it. The computation sequence is:

- Task 1: integrate $\bar{f}$ over $v_\parallel$ direction
- Task 2: compute righ-hand side $\tilde{\rho}$ in summing over μ direction
- Task 3: perform averages $\langle\tilde{\rho}\rangle_\theta(r{=}*,\varphi{=}*)$ and $\langle\tilde{\rho}\rangle_{\theta,\varphi}(r{=}*)$
- Task 4:
  - get $\langle\Phi\rangle_{\theta,\varphi}$ from $\langle\tilde{\rho}\rangle_{\theta,\varphi}$ thanks to eq. (5)
  - For each φ value
    - $\diamond$ FFT in θ on $\tilde{\rho}$
    - $\diamond$ derive $\hat{\Phi}^u$ modes $(\forall u > 0)$ with eq. (7)
    - $\diamond$ compute $\langle\Upsilon\rangle_\theta(r{=}*,\varphi)$ with eq. (9)
    - $\diamond$ add $\langle\Phi\rangle_{\theta,\varphi}{+}\langle\Upsilon\rangle_\theta(r{=}*,\varphi)$ to get $\hat{\Phi}^0(r{=}*,\varphi)$
    - $\diamond$ inverse FFT in θ on $\hat{\Phi}$

The presented algorithm 1 has some parameters: the mappings s, g and q that are detailed in the next subsection. They characterizes the effective data and computation distributions on the parallel machine.

The Algo. 2 follows immediatly the QN solver. It applies the gyroaverage on Φ and then computes its derivatives along spatial dimensions. These 3D fields (named A_1, A_2, A_3 in the algorithm) are inputs of the Vlasov solver. Then, they are redistributed in a communication steps in order to match the mapping needed by the Vlasov solver. In the *task* 2, derivatives along φ direction are computed; these computations have been delayed because it is much easier to derive them having access to all values along φ direction.

Algorithm 2: Parallel algorithm to get derivatives of the potential

Input : local block $\Phi(r=*,\theta=*,\varphi=[q_{start},q_{end}])$

(task 1*)*
for $\varphi=[q_{start},q_{end}]$ and $\mu=\mu_{value}$ **do**
  (parallelization in μ,φ)
  Computation : $A_0(r=*,\theta=*,\varphi) = J_0(k_\perp\sqrt{2\mu})\,\Phi(r=*,\theta=*,\varphi)$
  $A_1(r=*,\theta=*,\varphi) = \frac{\partial A_0(r=*,\theta=*,\varphi)}{\partial r}$
  $A_2(r=*,\theta=*,\varphi) = \frac{\partial A_0(r=*,\theta=*,\varphi)}{\partial \theta}$
end
Send local data $A_0|A_1|A_2(r=*,\theta=*,\varphi=[q_{start},q_{end}])$
Redistribute $A_0|A_1|A_2$ **inside** μ **communicator/ Synchronization**
Receive blocks $A_0|A_1|A_2(r=[i_{start},i_{end}],\theta=[j_{start},j_{end}],\varphi=*)$

(task 2*)*
for $r=[i_{start},i_{end}]$ and $\theta=[j_{start},j_{end}]$ and $\mu=\mu_{value}$ **do**
  (parallelization in μ,r,θ)
  Computation : $A_3(r,\theta,\varphi=*) = \frac{\partial A_0(r,\theta,\varphi=*)}{\partial\varphi}$
end

Outputs : $A_1|A_2|A_3(r=[i_{start},i_{end}],\theta=[j_{start},j_{end}],\varphi=*)$

3.4 Mapping functions

The two presented algorithms use three different mappings to distribute computations and data on the parallel machine. These mappings concerns φ and μ variables and are illustrated in Figures 1 and 2 for a large testbed and a small one respectively. Let #C be the number of cores used for a simulation run and #P be the number of MPI process. The number of threads #T per MPI process is fixed (usually it corresponds to the number of cores inside a SMP node), so we have #C = #P #T.

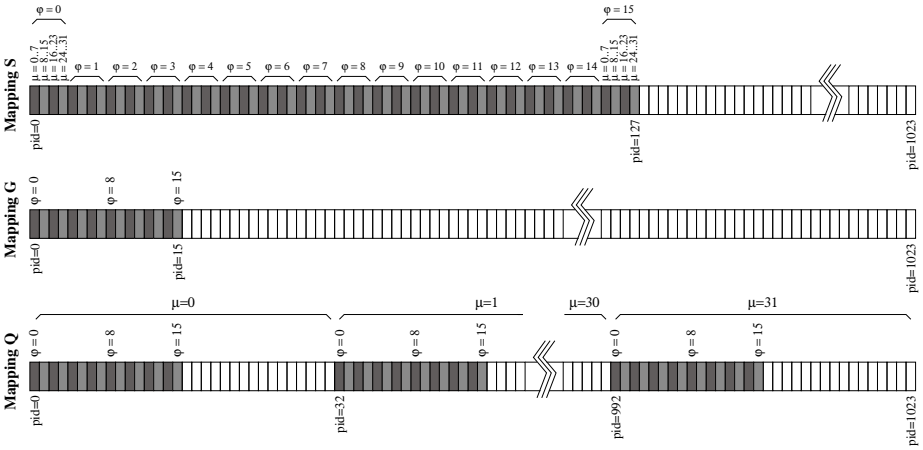

Figure 1: Mappings of φ and μ variables ($N_\mu = 32,\ N_\varphi = 16$) on processes for a large testbed (#P = 1024)

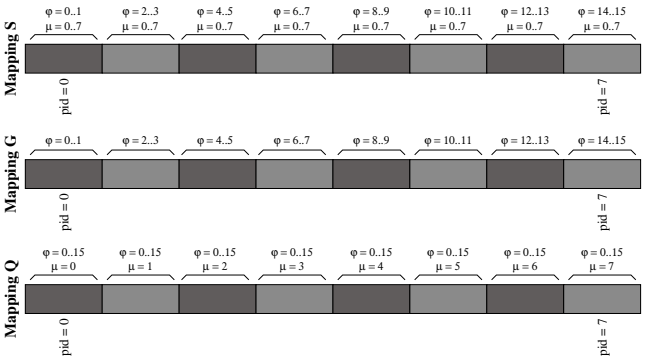

Figure 2: Mappings of φ and μ variables ($N_\mu = 8,\ N_\varphi = 16$) on processes for a small testbed (#P = 8)

Each rectangle on the Figures 1 and 2 represents a MPI process. Processes filled in dark gray or light gray have computations to perform, whereas the white color denotes processes that are idle. These mappings also implicitly prescribe how communication schemes exchange data during the execution of the two algorithms. We give here a brief description of these mappings:

- **Mapping S** - It defines the ranges $\varphi \in [s^{\varphi}_{start}, s^{\varphi}_{end}]$ and $\mu \in [s^{\mu}_{start}, s^{\mu}_{end}]$. In the task 2 of QN solver, we use this mapping to distribute the computation of the gyroaverage J_0. The maximal parallelism is then obtained whenever each core has at most one gyroaverage operator to apply. We have considered in the example shown in Fig. 1 that each MPI process hosts $\#\mathrm{T} = 8$ threads, so that it can deal with 8 gyroaveraging simultanously ($s^{\mu}_{end} - s^{\mu}_{start} + 1 = 8$). This distribution is computed in establishing a block distribution of domain $[0, \mathrm{N}_{\mu} - 1] \times [0, \mathrm{N}_{\varphi} - 1]$ on $\#\mathrm{C}$ cores.

- **Mapping G** - It defines the range $[g_{start}, g_{end}]$ for the φ variable. A simple block decomposition is used along φ dimension. For a large number of cores, this distribution gives to processes 0 to $\mathrm{N}_{\varphi} - 1$ the responsability to compute the N_{φ} slices of Φ data structure (task 4 of the QN solver). For a small number of cores (see Fig. 2), this mapping is identical to S one ($[s^{\mu}_{start}, s^{\mu}_{end}] = [0, N_{\mu} - 1]$). In such case, we save the communication step in task 2 of QN solver.

- **Mapping Q** - The mapping Q defines the range $\varphi \in [q_{start}, q_{end}]$ in each MPI process. We use inside each μ communicator a block decomposition along φ dimension. It is designed to carry out the computation of the gyroaverage of Φ, together with the computation of its derivatives (Algo. 2). These calculations depend on the value of μ. It is cost effective to perform them inside a μ communicator: we need to only locally redistribute data inside the μ communicator at the end of Algo. 2 in order to prepare the input 3D data fields for the Vlasov solver.

3.5 Communication costs analysis

Let us have a look on communication costs associated with the QN solver (Algo 1). From line 6 to line 8, a 4D data $\tilde{\rho}_1$ is redistributed (a global transposition that involves all MPI processes). The amount of communication represents the exchange of $\mathrm{N}_r\mathrm{N}_{\theta}\,\mathrm{N}_{\varphi}\mathrm{N}_{\mu}$ floats. For the lines 18 to 20, the amount of communication is strictly lower to $\mathrm{N}_r\mathrm{N}_{\theta}\,\mathrm{N}_{\varphi}\mathrm{N}_{\mu}$ floats, and tightly depends on the mapping S. The broadcast of lines 28 to 30 corresponds to a smaller communication cost of $\mathrm{N}_r\mathrm{N}_{\varphi}\min(\#\mathrm{C}, \mathrm{N}_{\varphi})$. The final communication of lines 42-44 sends N_{μ} times each 2D slice of Φ locally computed. It has a cost similar to the first one with $\mathrm{N}_r\mathrm{N}_{\theta}\,\mathrm{N}_{\varphi}\mathrm{N}_{\mu}$ floats to send, but without the need of data transposition (so less memory copies).

The computation of derivatives imply also communications. In Algo 2, a communication step transpose three 3D data inside each μ communicator. The overall cost is $3\,\mathrm{N}_r\,\mathrm{N}_{\theta}\,\mathrm{N}_{\varphi}\,\mathrm{N}_{\mu}$ floats, but only local communications inside μ communicators are realized.

4 Performance analysis

4.1 Parallel computing scalability

Timing measurements have been performed on CRAY-XT5 Jaguar machine[3] (Department of Energy's, Oak Ridge, USA). This machine has 18 688 XT5 nodes hosting dual hex-core AMD Opteron 2435 processors and 16 GB of memory. The Table 1 reports timing of the QN solver extracted from GYSELA runs. The smallest test case was run from 256 cores to 4096 cores. The parameters that has been are the following (small case):

$$N_r = 128, N_\theta = 256, N_\varphi = 128, N_{v\parallel} = 64, N_\mu = 32 \ .$$

In Table 2, timings for a bigger test case are presented. Its size is

$$N_r = 512, N_\theta = 512, N_\varphi = 128, N_{v\parallel} = 128, N_\mu = 32 \ .$$

For this second case, the parallel testbed were composed of 4k cores to 64k cores. In tables, the `io1`, `io2`, `io3`, `io4` steps states for communications associated with task 1, task 2, task 3, task 4 respectively. Computation costs, `comp1`, `comp2`, `comp3`, `comp4` stands for computations relative to task 1, task 2, task 3, task 4 respectively.

Note that the Vlasov solver of the GYSELA code uses a parallelization based on a domain decomposition along dimensions μ, r and θ. The number of processes #P is given by the product of N_μ the number of μ values with $\mathrm{p_r} \times \mathrm{p}_\theta$ the number of blocks along r and θ dimensions ($\#\mathrm{P} = \mathrm{N}_\mu \times \mathrm{p_r} \times \mathrm{p}_\theta$). The number of μ values in the considered cases is $\mathrm{N}_\mu = 32$, then GYSELA requires a minimum of 32 nodes to run.

General observations The OpenMP paradigm is used in addition to MPI parallelization (#T threads). All computations costs are lowered thanks to this fine grain parallelization. The main idea for this OpenMP parallelization has been to target φ loops. This approach is efficient for the computation task 1 of QN solver (integrals in $v_\parallel$). But in the tasks 3,4, this strategy competes with the MPI parallelization that uses also variable φ for the domain decomposition. Thus, above N_φ cores, no parallelization gain is expected. This fact is not the hardest constraint up to now: communication costs are the critical overhead, much more than computation distribution (as you see in Table 2). A recent improvement has been to add a parallelization along μ direction in task 2. Even if this change adds a communication step that can be avoided (`io2`), it is worthwhile on large platforms. We benefit from this extra level of parallelism, as soon as communication of task 2 (`io2`) remains low. Notably, we see that `comp2` scales beyond $N_\varphi = 128$ cores in Table 2.

Comments for the small case In Table 1, the communication costs for exchanging $\tilde{\rho}_1$ values (`io1` - task 1) is reduced along with the involved number of nodes. This is explained by the fact that the overall available network bandwidth increases with larger number of nodes, while the total amount of data exchanged remains the same. The communication cost associated with `io3` is

[3]Many thanks to C.S. Chang for giving us acces to Jaguar, project reference FUS022.

Nb. cores	256	1k	4k
Nb. nodes	32	128	512
QN solver			
comp1	2300 ms	580 ms	100 ms
io1	300 ms	160 ms	90 ms
comp2	170 ms	43 ms	13 ms
io2	0 ms	0 ms	8 ms
comp3	0 ms	0 ms	2 ms
io3	17 ms	42 ms	43 ms
comp4	4 ms	3 ms	4 ms
io4	100 ms	40 ms	35 ms
Total time	2900 ms	870 ms	300 ms
Relative eff.	100%	83%	60%

Table 1: Time measurements for one call to the QN solver - Small case

Nb. cores	4k	16k	64k
Nb. nodes	512	2k	8k
QN solver			
comp1	2200 ms	570 ms	95 ms
io1	450 ms	300 ms	470 ms
comp2	320 ms	180 ms	180 ms
io2	30 ms	60 ms	70 ms
comp3	3 ms	2 ms	3 ms
io3	150 ms	140 ms	130 ms
comp4	30 ms	30 ms	30 ms
io4	180 ms	100 ms	65 ms
Total time	3400 ms	1400 ms	1000 ms
Relative eff.	100%	61%	21%

Table 2: Time measurements for one call to the QN solver - Big case

mainly composed of synchronization of nodes and broadcasting the 2D data slice $\langle\tilde{\rho}\rangle_\theta(r=*,\varphi=*)$. The `io4` communication involves a selective send of parts of the electric potential Φ to each node; and one can note the same decreasing behaviour depending on number of cores also observed in `io1` .

The `comp1` calculation is one of the biggest CPU consumer of the QN solver; it scales well with the number of cores, combining MPI and OpenMP parallelizations. The `comp3` is negligible time and `comp4` is a small computation step, time measurements are nearly constant for all number of cores shown. In fact N_φ cores is the upper bound of the parallel decomposition for these steps. Then, between 1 and N_φ cores the speedup increases, whereas the speedup and computation time are constant above this limit (N_φ=128 cores for the cases shown here). The relative efficiency for the overall QN solver is 60% at 4k cores which is a good result for this computational problem that synchronizes and redistribute information between all processesx.

Comments for the big case The relative speedups shown in Table 2 considers as a reference the execution times on 4k cores, in order to have access to enough memory. Communication costs are larger than in the small test case. Badly `comp2` and `comp4` do not scale well. Only `comp1` and `io4` parts behave as we wish. In future works, we expect improving the scalability of this algorithm in lowering parallel overheads. The reduction of communication costs is one candidate (we foresee to use a compression method). An alternative can be to find a way to better distribute computations in the `comp2`, `comp3`, `comp4` parts. Even if improvements can be found, a good property of this solver is that execution time globally decreases along with the number of cores, and does not explode at all (for example due to growing communication costs). We see in Fig. 6 that this cheap cost of the QN solver brings to GYSELA code a good overall scalability. GYSELA reaches 78% of relative efficiency at 64k cores. In Fig. 3 and 4, timings for short runs of GYSELA are presented. The field solver and computation and derivatives of the gyroaveraged Φ are very low compared to Vlasov solver and diagnostics costs. Then, their limited scalability at very large number of cores, does not impact significantly the scalability of the overall GYSELA code.

Also, Let us remark the excellent scalability of GYSELA for the small case (Fig. 5) with a 97% of overall relative efficiency at 4k cores.

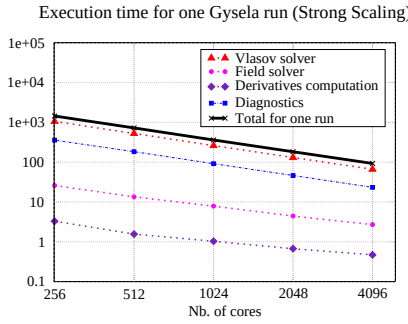

Figure 3: Small case - GYSELA timings

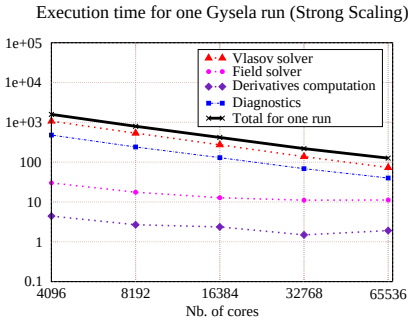

Figure 4: Big case - GYSELA timings

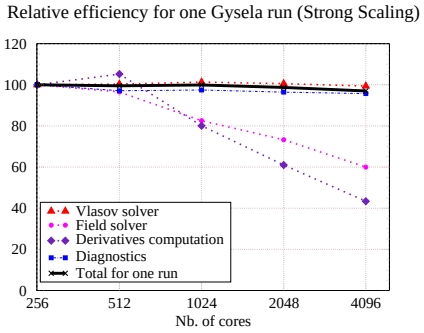

Figure 5: Small case, GYSELA efficiency

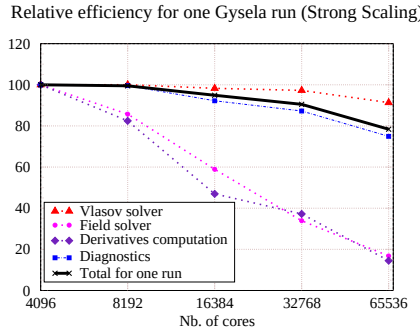

Figure 6: Big case, GYSELA efficiency

4.2 Memory scalability

The algorithm 1 proposed for the Poisson solver avoids the caveat of the final Φ broadcast to all MPI processes. The computation of the derivatives in spatial directions of the gyroaveraged potential has also been parallelized, compared to the previous version of GYSELA. It follows that we avoid the storage of complete 3D data structures in memory on each node. We end up with a pair of algorithms for Poisson and computation of the derivatives that only requires distributed field 3D data structures. Each node owns only one MPI process that himself hosts OpenMP threads. All threads inside of a single MPI process share these 3D field data.

The Table 3 shows the memory consumption for growing number of cores for a given problem size (strong scaling). The parameter N_μ remains constant while both p_r and p_θ are increased. The main result concerns the overall memory consumption that has diminished between the previous and the present version. But, the memory scalability is also greatly improved: each time the number of cores is doubling, the memory occupancy decreases better than previously.

Nb. cores	4k	8k	16k	32k	64k
Nb. nodes	512	1k	2k	4k	8k
Previous version					
4D data struct.	6.82	3.45	1.80	0.93	0.49
3D data struct.	2.79	2.75	2.73	2.72	2.72
2D data struct.	1.83	1.82	1.81	1.81	1.81
1D data struct.	0.09	0.04	0.02	0.01	0.01
All data struct.	11.55	8.07	6.38	5.48	5.02
Present version					
4D data struct.	6.82	3.45	1.80	0.92	0.49
3D data struct.	1.07	0.82	0.58	0.39	0.24
2D data struct.	1.05	1.04	1.03	1.03	1.03
1D data struct.	0.55	0.52	0.51	0.51	0.51
All data struct.	9.50	5.83	3.93	2.86	2.27

Table 3: Memory consumption (in GB) on each node by GYSELA, big case

At 64k cores, there is more than a factor 2 of difference between the memory consumption of the two GYSELA versions.

In addition to the splitting of 3D data structure previously described, other modifications have occured in the new version. A set of 3D and 2D data buffers have been removed and replaced by a unique large 1D array. It explains why the new version exhibits a larger amount of memory of 1D data while 3D and 2D data are drastically reduced compared to previous version. Another benefit: limitation concerning the minimal number of cores required to run a given test case. Further work can be done to cut down on remaining 2D and 1D data storage. But it will imply to change existing algorithms to new ones that saves memory.

5 Conclusion

We describe the parallelization of a quasineutral Poisson solver used into a full-f gyrokinetic 5D simulator [4]. The parallel performance of the numerical solving method is demonstrated. It achieves a good parallel computation scalability up to 64k cores combining several levels of parallelism and an hybrid OpenMP/MPI approach. The coupling of the quasineutral solver and the Vlasov code has been improved a lot compared to previous results [1, 4, 5, 6]. The modifications result also in savings in the memory occupancy, which is a big issue when physicists wish to run very large case.

References

[1] V. Grandgirard, M. Brunetti, P. Bertrand, N. Besse, X. Garbet, P. Ghendrih, G. Manfredi, Y. Sarazin, O. Sauter, E. Sonnendrucker, J. Vaclavik, and L. Villard. A drift-kinetic semi-lagrangian 4d code for ion turbulence simulation. *Journal of Computational Physics*, 217(2):395 – 423, 2006.

[4] Acknowledgments: This work was partially suported by an ANR GYPSI contract.

[2] V. Grandgirard, Y. Sarazin, X. Garbet, G. Dif-Pradalier, Ph. Ghendrih, N. Crouseilles, G. Latu, E. Sonnendrucker, N. Besse, and P. Bertrand. Computing ITG turbulence with a full-f semi-Lagrangian code. *Communications in Nonlinear Science and Numerical Simulation*, 13(1):81 – 87, 2008. Vlasovia 2006: The Second International Workshop on the Theory and Applications of the Vlasov Equation.

[3] T. S. Hahm. Nonlinear gyrokinetic equations for tokamak microturbulence. *Physics of Fluids*, 31(9):2670–2673, 1988.

[4] G. Latu, N. Crouseilles, V. Grandgirard, and E. Sonnendrucker. Gyrokinetic semi-Lagrangian parallel simulation using a hybrid OpenMP/MPI programming. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 4757 of *Lecture Notes in Computer Science*, pages 356–364. Springer, 2007.

[5] Guillaume Latu, Nicolas Crouseilles, and Virginie Grandgirard. Parallel bottleneck in the Quasineutrality solver embedded in GYSELA. Research Report RR-7595, INRIA, 04 2011.

[6] Guillaume Latu, Virginie Grandgirard, Nicolas Crouseilles, Radoin Belaouar, and Eric Sonnendrucker. Some parallel algorithms for the Quasineutrality solver of GYSELA. Research Report RR-7591, INRIA, 04 2011.

[7] Z. Lin and W. W. Lee. Method for solving the gyrokinetic Poisson equation in general geometry. *Phys. Rev. E*, 52(5):5646–5652, Nov 1995.

[8] Eric Sonnendrucker, Jean Roche, Pierre Bertrand, and Alain Ghizzo. The semi-Lagrangian method for the numerical resolution of the Vlasov equation. *Journal of Computational Physics*, 149(2):201 – 220, 1999.

Centre de recherche INRIA Nancy – Grand Est
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399