ESAIM: PROCEEDINGS, December 2013, Vol. 43, p. 274-294
S. Descombes, B. Dussoubs, S. Faure, L. Gouarin, V. Louvet, M. Massot, V. Miele, Editors

OPTIMIZING THE PARALLEL SCHEME OF THE POISSON SOLVER FOR THE REDUCED KINETIC CODE TERESA *

Thomas Cartier-Michaud[1], Philippe Ghendrih, Virginie Grandgirard and Guillaume Latu

Abstract. The parallelization performance of the TERESA Code for Trapped Element REduction in Semi lagrangian Approach is analyzed. TERESA is a kinetic code in four dimensions ($4D$), two in "real" space and two "energy" coordinates. It addresses the turbulent evolution of the distribution of ions governed by fluctuations of the electric potential. The numerical scheme is split into four steps: the $4D$ advection of the distribution function with the Vlasov equation, the computation of the charge density, the calculation of the electric potential using the quasineutrality asymptotic limit of the Poisson equation and finally, the two high frequency averages of the electric potential in order to compute the advection field. Starting from an initial standard parallelization scheme we find that the Poisson solver accounts for most of the execution time of TERESA (up to 90% at aimed meshes) and that its scaling performance is poor. Two improvements of the Poisson solver have been implemented and tested. These show that performance levelling are due to interference phenomena when addressing computation resources as well as communication overheads when the loading of the CPUs with MPI processes / OpenMP threads is not optimized. Tests on two computers, Rheticus et Helios, with very similar architecture, allows one to better diagnose the interference phenomena. These developments lead to reduction by up to a factor 100 of the execution time and extends the scalability properties up to 1024 cores for large meshes cases.

* *We acknowledge suggestions and help from Julien Bigot, Chantal Passeron, Fabien Rozar, Patrick Tamain and Olivier Thomine*

[1] CEA Cadarache, IRFM, F-13108 Saint-Paul-lez-Durance, France

Article published online by EDP Sciences and available at http://www.esaim-proc.org or http://dx.doi.org/10.1051/proc/201343017

Résumé. Ce papier décrit l'optimisation de la parallelisation du code TERESA pour Trapped Element REduction in Semi lagrangian Approach. TERESA est un code cinétique en 4 dimensions, 2 coordonnées dans l'espace "réel" et 2 coordonées dans l'espace des "énergies". Il permet de calculer l'évolution de la fonction de distribution d'ions en présence d'une turbulence induite par les fluctuations du potentiel électrique. L'espace $6D$ de l'espace des phases est réduit par une double moyenne sur les phases les plus rapides des trajectoires des ions. En conséquence, les variables d'énergie sont des constantes du mouvement. Dans cet espace des phases réduit, l'évolution des particules est déterminée par le potentiel électrique moyenné qui dépend alors des 4 dimensions de l'espace des phase réduit. Dans le schéma numérique, l'évolution de la fonction de distribution est décomposée en 4 étapes : l'advection de la fonction de distribution en $4D$ via l'équation de Vlasov, le calcul des densités de particules déterminant la densité de charge, le calcul du potentiel électrique par l'équation de Poisson compte tenu de la densité de charges, et enfin la moyenne du potentiel électrique sur les hautes fréquences afin de calculer le champ d'advection. A partir d'un premier schéma de parallélisation on trouve que le solveur de Poisson représente l'essentiel du temps de calcul (jusqu'à 90% pour les maillages visés) et que sa scalabilité est mauvaise. Deux schémas améliorés de ce solveur sont développés et testés. Ces études font apparaître d'autres sources de perte de performance du code, des interférences lors de l'accès aux ressources et des surcoût de communication lorsque la charge des CPU en process MPI / thread OpenMP n'est pas optimisée. L'utilisation des deux machines, Rheticus et Helios, semblables dans leur architecture, permet de mieux cerner des phénomènes d'interférrence. L'optimisation de TERESA permet de gagner, sur des gros maillages ciblés, un facteur jusqu'à 100 sur le temps d'exécution et d'étendre la scalabilité jusqu'à 1024 coeurs.

1. Introduction

The theoretical investigation of plasma turbulence in fusion reactors is a very challenging issue. With the increasing computation capability, kinetic simulations of plasma turbulence have been undertaken. However, despite the reduction of the problem from $6D$ to $5D$, in the so-called gyrokinetic framework, the numerical effort remains very important in terms of mesh sizes and computation time. Furthermore, for global simulations aiming at long time convergence towards statistically stable thermodynamics, the simulation of gyrokinetic electrons with present computers remains marginal if not impossible. Physicists and applied mathematicians are thus highly interested in a code allowing one to address the physics of plasma turbulence and to test numerical schemes at reduced dimensions and cost. A further reduction of the ion motion on the bounce frequency of the trapped particles provides a means to reduce the phase space of the system to $4D$. The TERESA code, for Trapped Element REduction in Semi lagrangian Approach, has been developed to address these issues. It takes benefit of earlier versions such as TIM3D [2] and TIM4D [4], but the code structure has been updated to be as similar as possible as that of the $5D$ GYSELA code [5,6]. We present here the analysis of the code performance and the tests and improvements that have allowed us to significantly improve the performance level of TERESA.

The paper is organized as follows. The set of equations used in the TERESA code is presented in Section 2. The implementation of the equations in the code is described in Section 3. The analysis of the code performance based on a standard parallelization scheme is addressed in Section 4. Modification of the code and tests addressing the distribution of the workload are detailed in Section 5. Hardware limitations are discussed in Section 6. Finally discussion and conclusion are found in Section 7.

2. Overview of the system

We proceed here to the description of the equations that are actually implemented in the code. Extending the gyro-average procedure to the average on the particle bounce motions in the well of the magnetic field introduces a second invariant so that the energy space can be described in terms of three invariants, namely the

particle energy E, the particle trapping parameter κ and the magnetic moment. The average on the gyrophase introduces an average of a displacement operator, isotropic in the plane orthogonal to the magnetic field, such that the characteristic distance scales like the Larmor radius ρ_0. The average over the bounce frequency introduces another scale, the so-called banana width of the trajectory δ_b. The displacement that is then averaged out is restricted to the radial coordinate ψ. In the limit $\delta_b \gg \rho_0$ the averaging procedure takes the form of an average in the plane orthogonal to the magnetic field with the scale δ_b in the coordinate ψ and the scale ρ_0 in α, the angle coordinate in the magnetic surface.

The TERESA code is dedicated to solving this reduced kinetic equation where two angles are averaged out assuming that the turbulence of interest is at low frequency compared to the pulsation associated to these fast angles. The phase space for this reduced kinetic equation is therefore $4D$. This $4D$ phase space is split into a 2D space of constant of motions and a $2D$ space of conjugate variables where phase space convection takes place. Conveniently, one chooses as constants of motion E ($E \in [0, \infty[$), the energy of the particles normalized by the characteristic thermal energy T_0, and the parameter κ that measures the degree of trapping with highly trapped particles at $\kappa \to 0$ and the transition between barely trapped particles and passing particles at $\kappa \to 1$ ($\kappa \in [0, 1[$). For convenience, the set of variables E, κ will be called energy coordinates in the following while the two other variables will be referred to as space variables. Indeed, one finds that the two remaining variables, the angle α and the action ψ, localize the particle in space, α being akin to the symmetry angle of the tokamak magnetic geometry and ψ is a label of a magnetic surface, hence constant on the magnetic surface. This variable can be viewed as an effective radius when extending actual geometries from nested concentric torii. We shall consider here the normalized poloidal flux function that varies from zero on the magnetic axis to 1 at the outer boundary, defined as the last closed magnetic surface.

The code then evolves the single particle distribution function averaged on the two fast angles $\bar{f}(\alpha, \psi, \kappa, E)$. The Vlasov evolution equation for $\bar{f}$ can be written as:

$$\partial_t \bar{f} - \left[\bar{\Phi}, \bar{f}\ \right] + E\omega_d \frac{\partial}{\partial \alpha} \bar{f} = 0 \tag{1}$$

In this expression $[.,.]$ stands for the Poisson Bracket: $[\bar{\Phi}, \bar{f}] = \partial_\psi \bar{\Phi} \partial_\alpha \bar{f} - \partial_\alpha \bar{\Phi} \partial_\psi \bar{f}$, and Φ is the normalized electric potential. Given the choice for the variable ψ and the normalization of time to the inverse of the ion Larmor frequency, one finds Φ in terms of the electric potential U combined to characteristic parameters:

$$\Phi = \frac{eU}{T_e} \tag{2}$$

The electric potential is of order one in $\rho_* = \rho_0/a$, from the choice of the normalization, with ρ_0 the reference Larmor radius and a the plasma minor radius. The potential $\bar{\Phi}$ that appears in the Vlasov equation (1) is the fast angle averaged electric potential and ω_d the drift frequency at the characteristic energy T_0. Regarding the space variables, one considers $0 \leq \alpha \leq 2\pi$ and $0 \leq \psi \leq 1$. The cost of the $4D$ reduction is that $\bar{\Phi}$, the fast angle averaged electric potential, is a $4D$ function depending on the space and energy variables. The fast angle averaging process is performed on the $2D$ electric potential determined by the Maxwell-Gauss equation in the quasineutral limit, namely:

$$C_e(\Phi - \varepsilon_e \langle \Phi \rangle_\alpha) - C_i \bar{\Delta} \Phi = \frac{1}{\sqrt{\pi} n_{eq} f_p} \int_0^{1^-} d\kappa \int_0^{+\infty} dE\ \ \mathcal{J}\bar{f} - 1 \tag{3}$$

In this expression, the operator $\mathcal{J}$ stands for the averaging operator on the fast angles and $\bar{\Delta}$ is defined as:

$$\bar{\Delta}\Phi = \frac{\partial^2}{\partial \alpha^2} + \bar{\delta}_b^2 \frac{\partial^2}{\partial \psi^2} \tag{4}$$

The parameter $\bar{\delta}_b$ is the ratio of the reference banana width which is defined as $\delta_b = \rho_0 q_0/\sqrt{\varepsilon}$ and the reference Larmor radius ρ_0, $\bar{\delta}_b = q_0/\sqrt{\varepsilon}$, where q_0 is the reference safety factor and ε the inverse aspect ratio. As a consequence one finds that the difference in weighing coefficient between the two spatial directions is of a factor up to 5 for $q \approx 3$ and $\varepsilon \geq 1/\sqrt{3}$. This drives therefore a significant asymmetry between the two directions in space. The parameter C_e depends on the parameter $\tau = T_0/T_e$, the ratio of the reference ion temperature to the electron temperature and the fraction of trapped ions f_p of the distribution function solution of the Vlasov equation at vanishing electric potential.

$$C_e = \frac{T_0}{T_e} \tag{5}$$

$$C_i = \rho_*^2 \left(a \frac{\partial \psi}{\partial r} \right)^2 \tag{6}$$

The parameter ε_e is introduced because the flux surface average that must be taken into account to describe the adiabatic electrons departs from the α-average of the bounce averaged trapped ions. This parameter is particularly important when considering the α-average of the Poisson equation (3):

$$\left(C_e(1-\varepsilon_e) - C_i \bar{\delta}_b^2 \frac{\partial^2}{\partial \psi^2} \right) \langle \Phi \rangle_\alpha = \frac{1}{\sqrt{\pi} n_{eq} f_p} \int_0^{1^-} d\kappa \int_0^{+\infty} dE \left\langle \mathcal{J} \bar{f} \right\rangle_\alpha - 1 \tag{7}$$

The right hand side acts as a source $\mathcal{S}_z$ for $\langle \Phi \rangle_\alpha$, so that the zonal flow $\langle \partial_\psi \Phi \rangle_\alpha$ that plays a strong role in turbulence self-organization, is then determined by:

$$\langle \partial_\psi \Phi \rangle_\alpha \approx \frac{i\ k_\psi \mathcal{S}_z}{C_e\,(1-\varepsilon_e) - C_i \left(k_\psi \bar{\delta}_b \right)^2} \tag{8}$$

One then finds that in the regime $(1-\varepsilon_e) \ll (k_\psi \bar{\delta}_b)^2$ the shear of the zonal flow will be maximized and conversely for $(1-\varepsilon_e) \gg (k_\psi \bar{\delta}_b)^2$ the impact of the zonal flow on the turbulence self-regulation will be significantly reduced. The physics in the limit $\varepsilon_e \to 1^-$ will thus be quite different from that for other values of ε_e where the effect of zonal flows is reduced since the electron adiabatic response filters out the response of $\langle \partial_\psi \Phi \rangle_\alpha$ at the smallest radial scales $k_\psi \bar{\delta}_b \approx 1$.

3. Mathematical discretization of TERESA model

3.1. **Phase space mesh**

The TERESA code is developed as a sister code of the 5D gyrokinetic GYSELA code. It is therefore a semi-Lagrangian code, where the characteristics are followed backward in time to determine the distribution function on a grid. The latter is regular in space coordinates (α, ψ) since turbulence is essentially isotropic. While operators are $2D$ real space operators, and are therefore sensitive to the real space mesh size, the mesh of the energy coordinates is defined to ensure the highest precision in the calculation of the integrals in quasi-neutrality equation (3). This is particularly important since errors in this integral induce spurious electric fields that modify the turbulent behavior.

The real space mesh is governed by two sets of scales, on the one hand, the size of the system, on the other hand the small scales where dissipation occurs. Since the fast frequency averaging operators determine a cut-off on these small scales, we consider that the Larmor radius ρ in the α direction and the banana width δ_b in the radial direction ψ, determine the relevant mesh sizes for the real space mesh. The typical banana width in units of poloidal flux is $\delta_b^{\psi_{pol}} = R_0 B_0 \rho \sqrt{\varepsilon}$. We consider that the maximum poloidal flux can be written as: $S_\psi \Psi$, where S_ψ is a constant depending on the magnetic shear and Ψ the range of magnetic surfaces, so that the

reference scale in the normalized ψ_{pol} units is $\bar{\delta}_b^{\psi_{pol}} = q_a\rho_*/(S_\psi\sqrt{\varepsilon})$, where $q_a = B_0a^2/\Psi$ is an effective safety factor. Defining N_s the number of points to mesh the banana width, one then finds that the number of points required in the ψ direction is:

$$N_\psi = N_s \frac{S_\psi\sqrt{\varepsilon}\Delta_\psi}{q_a}\frac{1}{\rho_*} \tag{9}$$

The interval in ψ that we consider for the simulation will extend from typically $\psi \approx 0.2$ to $\psi \approx 1$, hence yielding $\Delta_\psi \approx 0.8$. This interval is chosen so that it is possible to consider a constant magnetic shear region. In the α-direction the reference length is the Larmor radius ρ that must be compared to $2\pi R_0$ (full-torus) to determine the mesh size:

$$N_\alpha = N_s \frac{2\pi}{\varepsilon}\frac{1}{\rho_*} \tag{10}$$

This determines the scaling of the real space grid size $N_\alpha \times N_\psi$ and the aspect ratio of the grid N_α/N_ψ.

$$N_\alpha \times N_\psi = \frac{2\pi S_\psi\Delta_\psi}{q_a\sqrt{\varepsilon}}\left(\frac{Ns}{\rho_*}\right)^2 \tag{11}$$

$$\frac{N_\alpha}{N_\psi} = \frac{2\pi q_a}{S_\psi\Delta_\psi\varepsilon^{3/2}} \tag{12}$$

This ratio is found to be large, typically 250 for realistic simulations. Since fast Fourier transforms are used in TERESA, one thus requires that $N_\alpha = 2^n + 1$ where n is an integer and where an extra point has been added for efficient spline calculations and spacial integration.

Regarding the energy variables one can readily observe that the κ-integral is not very sensitive to the value of κ. However, since the drift frequency is nearly constant (κ independent) and only varies significantly at the boundary between trapped and passing particles $\kappa \to 1^-$, the number of points in κ will play a role with respect to an accurate description of this class of particles. Should this variation be considered as important, one can remap κ to $K = \kappa^2$ to increase the description in the vicinity of 1. Regarding the energy, typical distribution functions will be of the form $\exp(-E/T_0)$, hence the bulk of the distribution function is localized at $E \approx 0$. In order to increase the accuracy of the energy integration required to compute the particle density in the Poisson equation, it is then interesting to map the energy grid according to the velocity V such that $V = \sqrt{2E}$.

3.2. **Semi Lagrangian scheme for the Vlasov advection**

The semi-Lagrangian scheme method is based on the invariance of the distribution function along the characteristics $X(t; x, s)$ solution of (13) at time t given the value x at time s.

$$\frac{dX(t)}{dt} = \mathcal{V}(X(t), t) \tag{13}$$

where positions are phase space positions and $\mathcal{V}$ is the advection field that governs the evolution of the distribution function in the Vlasov equation.

$$\frac{d}{dt}f(X(t),t) = \partial_t f + \frac{dX}{dt}.\nabla f = \partial_t f + \mathcal{V}(X(t),t).\nabla f = 0 \tag{14}$$

Equation (14) allows one to determine f at different times by following the characteristics.

$$\begin{aligned} f(X(t+\delta t;x,t),t+\delta t) &= f(X(t;x,t)+\delta x,t+\delta t) && (15)\\ &= f(x,t) && (16)\\ \delta x &= \int_t^{t+\delta t} \mathcal{V}(X(s;x,t),s)ds && (17)\end{aligned}$$

The Vlasov advection is achieved in two steps, the first step determines the foot of the characteristics, namely the position in phase space at time t, from which stems the characteristics reaching a given grid point at time $t+\delta t$. This calculation is achieved by solving a fixed point problem with a Newton algorithm [8] parametred to achieve an order two accuracy. The second step is performed by interpolating f at the the foot of the characteristic with bi-cubic splines [1, 3]. The splines are necessarily chosen periodic along α while they are chosen natural along ψ.

3.3. **Poisson solver for the quasi-neutrality equation**

The quasi-neutrality equation takes the form of a Poisson equation which is significantly easier to solve when taking into account the periodicity in α-dimension to use Fourier mode decomposition in α and finite differences along ψ.

$$\Phi(\alpha,\psi)=\sum_m \hat{\Phi}_m(\psi)e^{-im\alpha} \qquad ; \qquad \bar{\bar{n}}(\alpha,\psi)=\sum_m \hat{\bar{\bar{n}}}_m(\psi)e^{-im\alpha} \tag{18}$$

Hence, equation (3) applied to the orthogonal mode m is:

$$\Big\{C_e\Big(1-\delta_{m,0}\varepsilon_e\Big)-C_i\Big(-m^2+\bar{\delta}_b^2\partial_\psi^2\Big)\Big\}\hat{\Phi}_m(\psi) = \hat{\bar{\bar{n}}}_m(\psi) \tag{19}$$

In order to get an order four approximation of the second order derivative in ψ, this equations leads to solve $(N_\alpha-1)/2+1$ independent pentadiagonal linear systems of size N_ψ instead of a unique one 9-diagonal linear system of size $N_\alpha \times N_\psi$. This decomposition can then be used to implement a parallelization of these operations. Boundary conditions along ψ are designed so that no particle can be advected outside the box, hence $\partial_\alpha\Phi=0$ at the boundaries. Moreover Φ is defined so that its boundary conditions are the Dirichlet condition $\Phi_m(\psi_{in})=\Phi_m(\psi_{out})=0$ at the two chosen boundaries ψ_{in} and ψ_{out}. Boundaries are computed at second order.

3.4. **Gyro-Bounce average operator**

The gyro-bounce average operator $\mathcal{J}_{\rho_{c0},\delta_{b0}}$ has an analytical expression if Fourier space in terms of Bessel functions J_0. However, this operators is applied in real space and in particular in the ψ-direction that is requires some care to be addressed in Fourier space since it is not periodic. Furthermore, the averaging operator depends on the energy variables so that the averaged of the electric potential on the two high frequencies is a $4D$ field in phase space. For this operator we consider a Pade approximation [7] that exhibits the appropriate width for the filtering but tends to over-damp the small scales.

$$\mathcal{J}_{\rho_{c0},\delta_{b0}}f=(1-\frac{E}{4}\delta_{b0}^2\partial_\psi^2)^{-1}(1-\frac{E}{4}\rho_{c0}^2\partial_\alpha^2)^{-1}f=\bar{f} \tag{20}$$

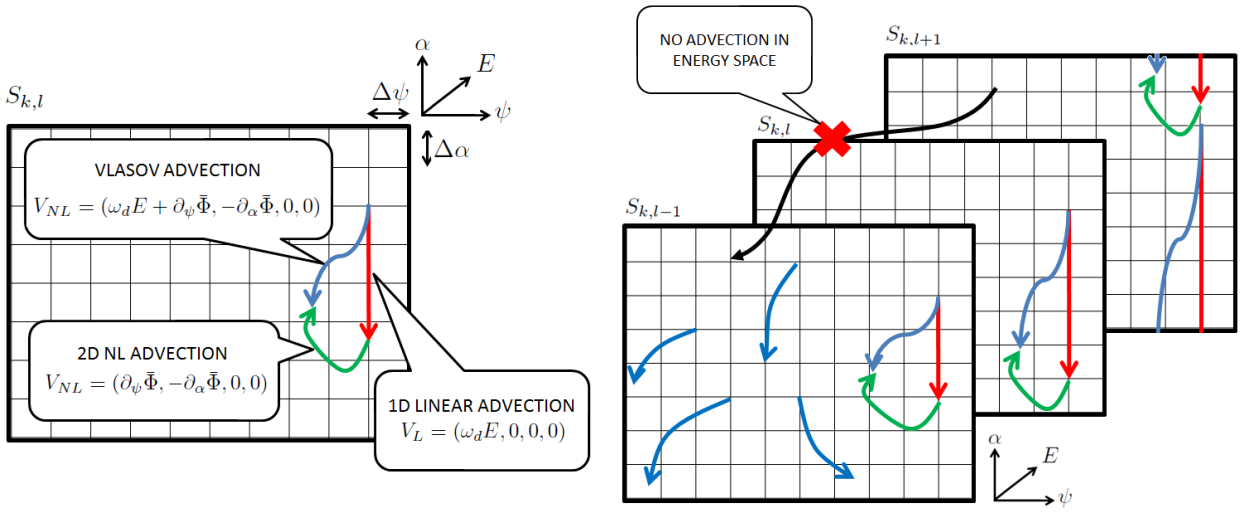
This operator is split into two second order $1D$-derivative. Groups of independent $1D$ systems are then solved instead of a unique $2D$ problem. Second order derivatives are computed by finite differences up to second order with Dirichlet boundary conditions for the ψ direction and periodic boundary conditions along α.

4. TERESA Parallelization scheme

The TERESA parallelization is based on a hybrid MPI / OpenMP scheme. This allows one to benefit of the computing capability of the HPC at the cost of a finer grain tunning of the parallelization. Each of the standard N_{MPI} MPI porcesses is completed by N_{OpenMP} threads. This allows one to access more cores ($N_{core} = N_{MPI} \times N_{OpenMP}$) and, by proper balancing, it increases computing performance.

4.1. TERESA workflow and domain decomposition

The parallelization scheme of TERESA system is constrained by the number of points required to mesh the unknown $\bar{f}_{4D}$ which can be very large compared to the available memory per core from 512 MBytes to 4 GBytes. The data must be distributed with minimum redundancy over MPI processes, themselves distributed within and over several nodes. As already discussed, the $4D$ phase space can be decomposed between conjugate variables ($\alpha \times \psi$), which actually determine the location in real space of the trapped particles, and the $2D$ space of invariants ($\kappa \times E$). The first two dimensions are used to define $2D$ sub-domains $S_{k,\ell}$ indexed by k, the coordinate respective to the trapping parameter κ and by ℓ the coordinate respective to the energy E. Thus the energy space is used to build a parallel domain decomposition. Sub-domains are constrained by $\cup_{k,\ell} S_{k,\ell} = \bar{f}_{4D}$ and $\cap S_{k,\ell} S_{i,j} = \emptyset$ if $(k,\ell) \neq (i,j)$. This decomposition is convenient because differential operators are include in real space and those operators are costly to distribute over several MPI processes. The phase space advection exemplifies this decomposition: particles are trapped in the $S_{k,\ell}$ sub-domains because the energy space ($\kappa \times E$) is invariant, position of a particle in ($\kappa \times E$) can not evolve (see Fig 1). It is to be underlined that the advection depends on all $4D$ variables but only acts on the $2D$ ($\alpha \times \psi$) variables (see equation 1). The computations related to advection are local to the process owing $S_{k,\ell}$.



(A) Motion decomposition within one subdomain $S_{k,\ell}$. (B) α periodicity and no advection in Energy space.

FIGURE 1. Decomposition of a variable $\bar{f}_{3D}$ in a union of $S_{k,\ell}$ sub domains (same principle for $\bar{f}_{4D}$ in $4D$). Particle advections are bounded to $S_{k,\ell}$ sub domains with periodic α boundaries.

If we choose a smaller sub-domain than $S_{k,\ell}$, it would require extra MPI communications because we can not build a decomposition which assures particles would not be advected from one sub-domain to another one (see Fig. 1b). For almost all real space meshes that will be needed, typically with less than 2^{13} grid points per dimension, our phase space decomposition is sufficient. The drawback is this distribution therefore faces a upper limit on the number of MPI processes determined by the number of sub-domains $S_{k,\ell}$, $\max(N_{MPI}) = N_\kappa \times N_E$,

where N_κ and N_E are the number of points meshing the κ and E dimensions. When less MPI processes than points in energy space are used, sub-domains of each MPI processes are enlarged as unions of $S_{k,\ell}$. The TERESA

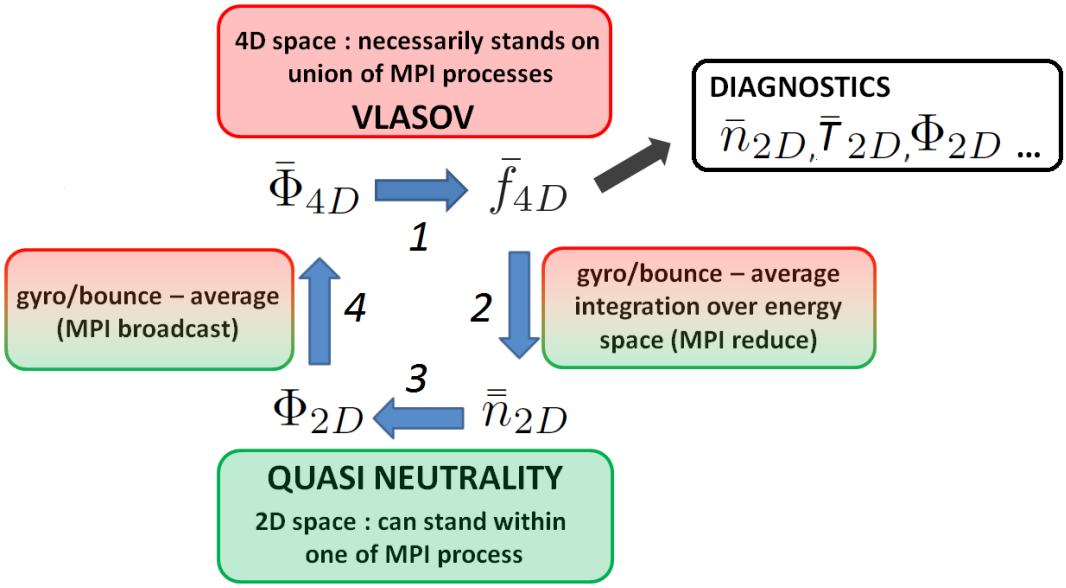


FIGURE 2. Four steps workflow of TERESA system

workflow is organized in four algorithmic steps, see Fig. 2. We distinguish two computational blocks in two different spaces : the Vlasov advection in the $4D$ phase space (step 1) and the solving of the quasineutrality condition, the so-called Poisson operator, in the $2D$ real space (step 3). Two projection-like operators bridge those two blocks: $4D \to 2D$ the energy space integration of $\bar{f}_{4D}$ (step 2) is projection in real space yielding the particle density, $2D \to 4D$ corresponds to the gyro-bounce average of Φ_{2D} (step 4), which yields the $4D$ advection potential $\nabla_{\alpha,\psi}\bar{\Phi}_{4D}$ since the gyro-bounce averaging process depends on the energy variables.

(1) The discretized Vlasov equation evolves $\bar{f}_{4D}(t)$ into $\bar{f}_{4D}(t+\Delta t)$, using $\nabla_{\alpha,\psi}\bar{\Phi}_{4D}$ as advection field. It is performed independently on each sub-domain $S_{k,\ell}$ distributed over the MPI processes. Advection is based on two subroutines. First, the origins of the characteristics are localized with the fixed point Newton algorithm. Second, the value of $\bar{f}_{4D}(t+\Delta t)$ at this location is interpolated with bi-cubic splines. Each step is performed independently for the $N_\alpha \times N_\psi$ grid points in real space. The computation of the spline coefficients is done once in each sub-domain $S_{k,\ell}$ and is not parallelized. This could be improved by using local cubic spline.

(2) The $4D$ reduction into $2D$, including an energy integration, require an "MPI reduce" process. Note that the gyro-bounce average, alike the Vlasov advection, is performed independently on each sub-domain $S_{k,\ell}$ by MPI processes.

(3) The quasi-neutrality equation yields the electrical potential Φ_{2D}. This step takes place in the $2D$ space described by $N_\alpha \times N_\psi$ points, and thus fits on one MPI process (within one node).

(4) The gyro-bounce advection potential $\bar{\Phi}_{4D}$ is distributed over MPI processes like $\bar{f}_{4D}$. This step thus requires an "MPI broadcast" process, opposite to the "MPI reduce" of step 2. Once Φ_{2D} is distributed

on the $S_{k,\ell}$ sub-domains, the gyro-bounce average is computed.

A fifth step in the TERESA workflow, which is not part of the main loop, is the diagnostic step. At present these are mainly based on fluid projection of $\bar{f}_{4D}$. It implies integrating the distribution function on the energy space, weighted by Hermite polynomials. Communication between all MPI processes is unavoidable. We have not focused on the optimization of the diagnostic step, because it does not require large computation time or complex dedicated routines. A bottleneck remains, apart from the intrinsic communication time, the I/O time to write on the disks. As diagnostics are not computed at each temporal integration step, the overall computational cost of diagnostics remains small compared to other steps.

4.2. **Performance analysis**

4.2.1. *Timer implementation and performance indicators*

In order to determine performance scalings, we need to precisely time each MPI process in each routine. An MPI barrier is set before starting any timer, processes are then synchronized when they enter a routine. Note that there is no need for explicit barriers in the algorithm of TERESA, only one implicit barrier is used at the end of step 2 of the workflow. The timing data provides a diagnostic of the performance of the various steps in the workflow. However, since the computers have a variable work load they provide a fluctuating environment to the codes being tested. This is particularly sensitive regarding the communication time. Each step is thus called several times to provide a statistical data base.

Given N_i, the number of cores used in simulation "i", T_i the execution time and the computation cost C_i, readily defined as $C_i = N_i \times T_i$, we then determine the four following standard indicators:

- *Speed-Up* $SU_{i/j}$ between two simulations i and j: it quantifies the acceleration in execution time: $SU_{i/j} = T_i/T_j$.
- *Relative Efficiency* S_{N_i/N_j} between two simulations i and j: it is defined as the ratio of the computation costs, $S_{N_i/N_j} = C_i/C_j = (N_i/N_j)SU_{i/j}$. In practice it provides a comparison to an ideal case where the computation cost does not depend on the number of processes hence $S^*_{N_i/N_j} = 1$. Code optimization then targets a scalability value as close as possible to 1. It is important to note that this indicator can exceed 1 whenever hardware optimization is available (cache effects).
- *Routine computational weight* R^x_i of routine x, $R^x_i = T^x_i/T_i$ where T^x_i is the time spent in routine x during simulation i. R^x_i identifies the weight of routine x on the overall simulation. Its variation with the number of cores is a clear indicator of its performance in terms of parallelization.
- *Speed-up gain* $G_i(R^x_i, SU)$ of simulation i achieved by accelerating one routine x by a factor SU : $G_i(R^x_i, SU) = T_i/(T_i - T^x_i + T^x_i/SU)$

4.2.2. *Key technical features of the computers used for TERESA optimization*

The two computers used to address the optimization of TERESA is an international computing facility machine, *Helios*, and a local cluster *Rheticus*. The technical features of these two computers are the following.

- *Rheticus* at the *Meso centre* of *Aix-Marseille* university (France) (Fig. 3) :
  - 96 nodes of 2 Intel X5675 CPU (6 cores @ 3.06 Ghz each) $\Rightarrow$ 1152 cores.
  - 12 MBytes cache L3 and 12 GBytes of RAM per CPU
  - 32 GBytes/s peak RAM bandwidth per CPU, 25.6 GBytes/s QPI bandwidth between CPUs
  - network between nodes: Infiniband QDR.
- *Helios* at the *IFERC center* in Rokkasho (Japan) (Fig. 4) :
  - 4410 nodes of 2 Intel E5-2680 CPU (8 cores @ 2.7 Ghz each) $\Rightarrow$ 70560 cores.
  - 20 MBytes cache L3 and 32 GBytes of RAM per CPU
  - $51,2$ GBytes/s peak RAM bandwidth per CPU, 32 GBytes/s QPI bandwidth between CPUs
  - network between nodes : Infiniband QDR.

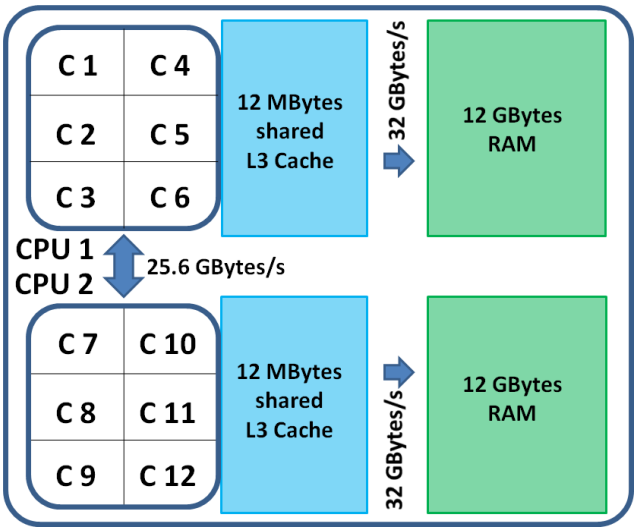


FIGURE 3. Architecture of a Rheticus node.

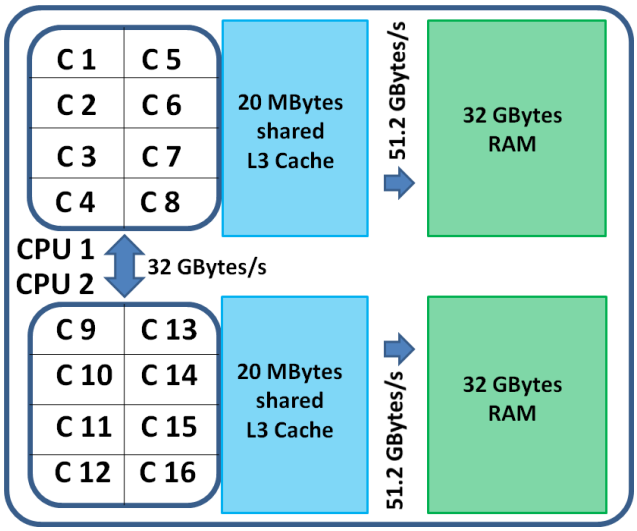


FIGURE 4. Architecture of a Helios node.

The architecture of these two computers being comparable, albeit at lower performance level for the local computer, the development of TERESA has been achieved on Rheticus while Helios runs where used to complete the tests at larger number of cores.

4.3. **TERESA Bottleneck: the Poisson solver**

At a given stage of TERESA development a systematic scaling analysis has been performed to determine the code parallelization performance, and consequently steps to be undertaken to improve this performance level. Since parallelization performance depends on the mesh size, four different meshes, labeled by their relevance for a particular investigation of the physics, have been tested. They are ordered by increasing communication requirement and decreasing number of MPI processes.

(1) Case *3D High Kinetic Resolution* is characterized by a reduced resolution in real space 129×64 ($N_\alpha \times N_\psi$), and large resolution in energy 1×65536 ($N_\kappa \times N_E$). The size of MPI messages is defined by the real space sub-mesh: $8 \times N_\alpha \times N_\psi = 64$ KBytes. The energy resolution determines the maximum number of 65536 MPI processes. The total number of grid points is 5.4×10^8 (or 4.0 GBytes). The overall memory requirement remains smaller than the available RAM on one Helios node.

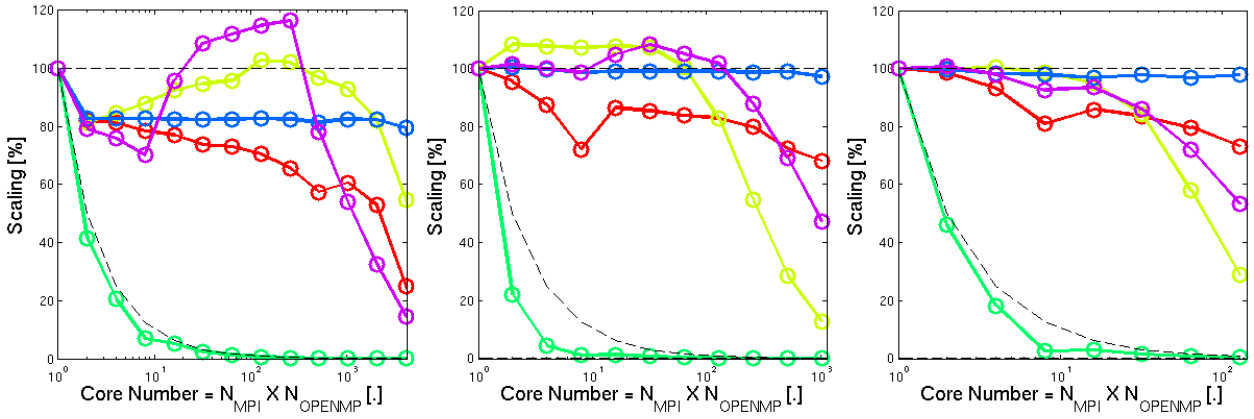
(2) Case *4D Medium Resolution* is characterized by a real space resolution of 513×512, hence MPI messages of 2 MBytes, and an energy resolution of 32×256 leading to a maximum of 8192 MPI processes. The total number of grid points is 2.2×10^9 (or 16.0 *GBytes*). The overall memory requirement requires four nodes of Helios.

(3) Case *3D High Resolution* characterized by a high resolution in real space 1025×768, hence MPI messages of 6 MBytes, high resolution in energy 1×1024 leading to a maximum of 1024 MPI processes. The total number of grid points is 8.1×10^8 (or 6.0 *GBytes*). The overall memory requirement remains smaller than one Helios node.

(4) Case *3D High Space Resolution* characterized by a high resolution in real space 2049×2048, hence MPI messages of 32 MBytes, and a standard resolution in energy 1×128 leading to a maximum of 128 MPI processes. The total number of grid points is 5.4×10^8 (or 4.0 *GBytes*). The overall memory requirement remains smaller than one Helios node.

Maximum overall footprint of TERESA code is about $\times 10$ size of $\bar{f}_{4D}$ while using 1 MPI process. It does not scale efficiently but, for example, mesh 2 stands on 4 nodes (64 cores) while it can possibly use up to 8192 nodes (131072 cores). At the moment, no mechanism of footprint reduction are undertaken as this limit never occurred expect to determine the lowest number of cores we can use on a specified mesh while scaling.

4.3.1. *Steps in TERESA workflow with sufficient parallelization performance*



(A) Helios case (1): 3D High Kinetic Resolution $129 \times 64 \times 1 \times 65536$

(B) Helios case (3) 3D High Resolution $1025 \times 1024 \times 1 \times 1024$

(C) Helios case (4) 3D High Space Resolution $2049 \times 2048 \times 1 \times 128$

FIGURE 5. Relative Efficiency investigation on Helios. Red : Vlasov advection (step 1 in Figure 2); Yellow : Gyro-Bounce + Integration (step 2); Green : Quasi Neutrality Equation (step 3); Blue : Broadcast + Gyro-Bounce (step 4) and Purple : Diagnostics. The scalability $S_{N_{core}/1}$ is plotted versus the number of cores N_{core}, the dashed black curve $1/N_{core}$ is the case without scalability while the horizontal line at $S_{N_{core}/1} = 100$ is the ideal case.

The step 1, Vlasov advection, is observed to decay rather gradually with the number of cores. This result is probably connected to the cost of initializing the large amount of intermediate variables used at that step. As for other steps, non monotonic behavior is observed in the 8 - 32 N_{core} range. This question will be further addressed in Section 6.1. Up to 1024 cores the relative efficiency remains above the 60 % range which is quite a good result. The relative efficiency of the diagnostic step is found to exhibit a non-monotonic behavior with more pronounced drop as the number of cores increases. Their is clearly room for improvement for this step. However, since the contribution of this diagnostic step in the computation cost is always small, this has not been undertaken since the actual impact on the code performance would have been moderate. The step 4, based only on the gyro-bounce operator in this version, is observed to exhibit excellent relative efficiency. It can be expected since gyro-bounce operator is based on independent calculations. The step 2, also based on the gyro-bounce operator shows a sudden drop at highest number of cores. The increasing cost of communications are responsible for this behavior. The step 3 is even less efficient than a routine which would use only one core. The case 2 is used to identify spurious inhomogeneity in MPI decomposition of the $2D$ energy space. It allowed us to check that less than a 2% variation of execution occures when changing the data distribution.

4.3.2. *The Poisson solver bottleneck*

The time spent in the Poisson solver step normalized by the total computation time is plotted on Fig. 6 for Helios cases 1, 3 and 4 versus the number of cores. One can see that this ratio tends to increase rather sharply with the number of cores and reaches large values in the 80% to 90% range, except for case 1 with the smallest sub-domain $N_\alpha \times N_\psi$. This behavior is thus very sensitive to the size ($N_\alpha \times N_\psi$) of the real space sub-domain.

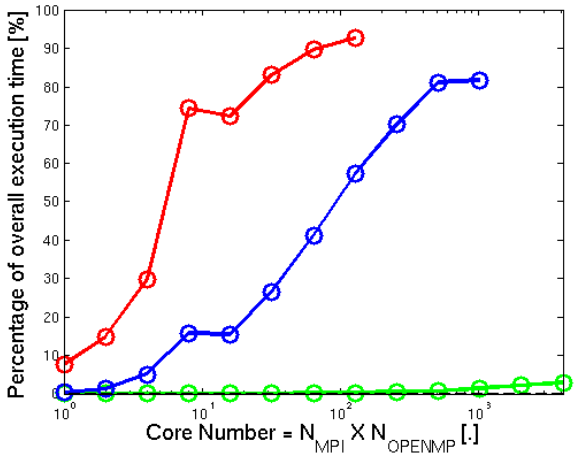


FIGURE 6. Ratio of the run time of the Poisson solver normalized by the overall TERESA run time on Helios. Green, case 1: $129 \times 64 \times 1 \times 65536$ mesh size; Blue, case 3: $1025 \times 768 \times 1 \times 1024$ mesh size; and Red, case 4: $2049 \times 2048 \times 1 \times 128$ mesh size.

This result clearly indicates that the Poisson solver exhibits the poorest relative efficiency and that it is the most time consuming step when the number of cores is increased beyond some 128 cores.

5. Advanced MPI / OpenMP for the real space sub-domain

5.1. Poisson solvers theoretical schemes

5.1.1. *Sequential computation of the Poisson solver done by all cores*

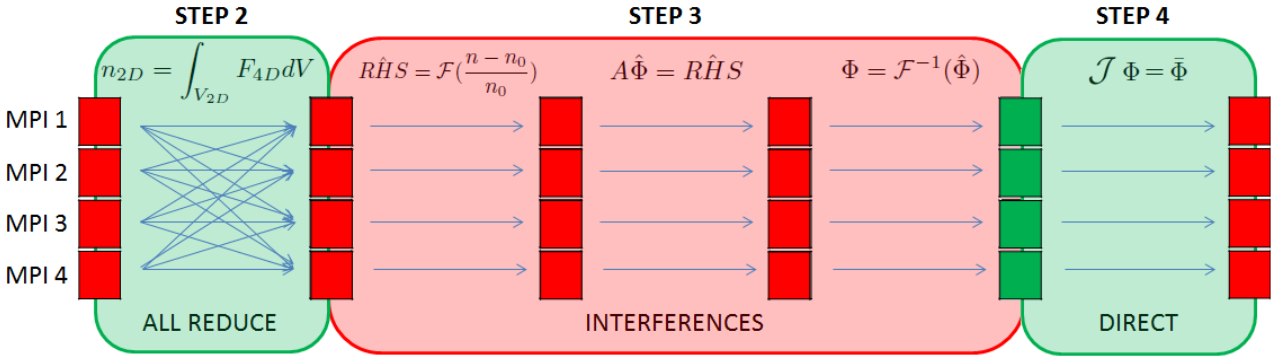


FIGURE 7. Sequential computation of the Poisson solver done by all cores. The bottleneck is due to interferences between the various MPI processes performing redundant calculation. Red squares = working MPI processes, green squares = idle MPI processes.

The first version of the Poisson solver in the TERESA code, Fig. 6 and Fig. 7, is the easiest to implement. The step 2 in the workflow, is the energy integration. It is performed by parts, first locally and then distributed to all MPI processes at the same time to be completed. Then, the same Poisson solver, step 3 of the workflow, is solved independently and synchronously by each core. The computation time by each core then depends

on the size of the real space sub-mesh. When the real space resolution is low, the Poisson solver has an overall computational cost that remains smaller than the energy integration, step 2. This implementation takes benefit from the optimized communication scheme provided by the "all reduce" MPI collective routine. Every processor hosting redundant calculations on different cores will exhibit a reduced efficiency as soon as common resources are addressed, RAM bandwidth and L3 cache, and govern interferences between the various calculations. This effect is expected to increase with the size of the real space sub-domain as readily observed in Fig. 6.

5.1.2. *Sequential computation of the Poisson solver done by a single master core*

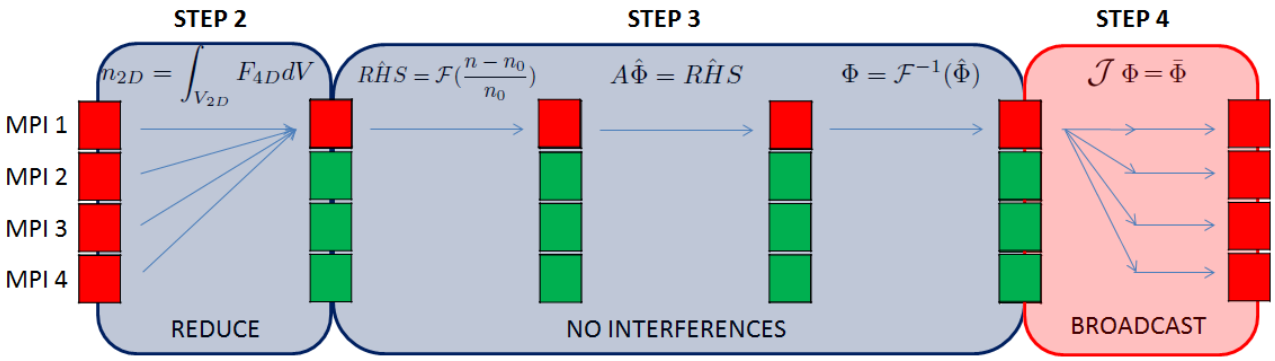


FIGURE 8. Sequential computation of the Poisson solver done by a single master core. The bottleneck is the workload of the master CPU doing the job.

This sequential computation of the Poisson solver done by a single master core (Fig. 8) avoids redundant calculations at the cost of broadcasting the result at the end of the calculation. Many MPI processes are idle which underlines that the use of computing power is not optimal (but hardly worse than having the MPI processes duplicating a calculation).

5.1.3. *Parallelized computation of the Poisson solver performed by a sub-group of cores*

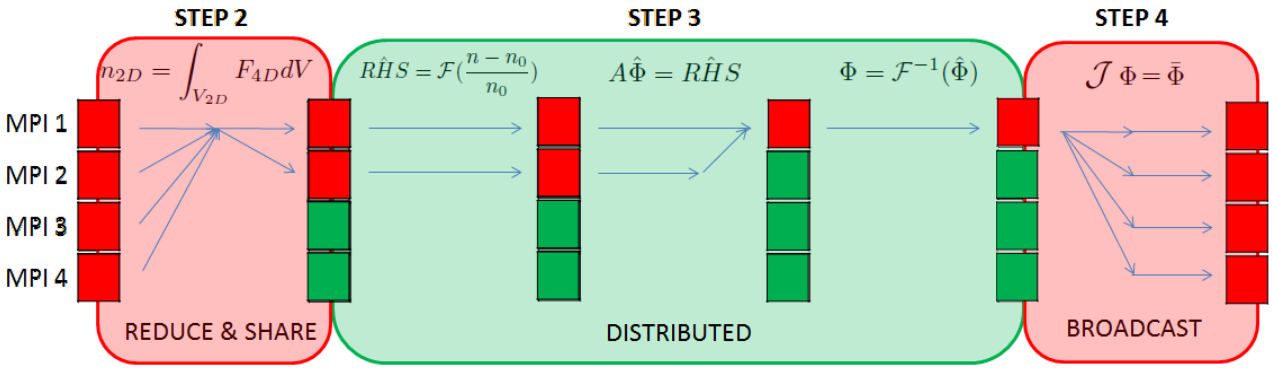
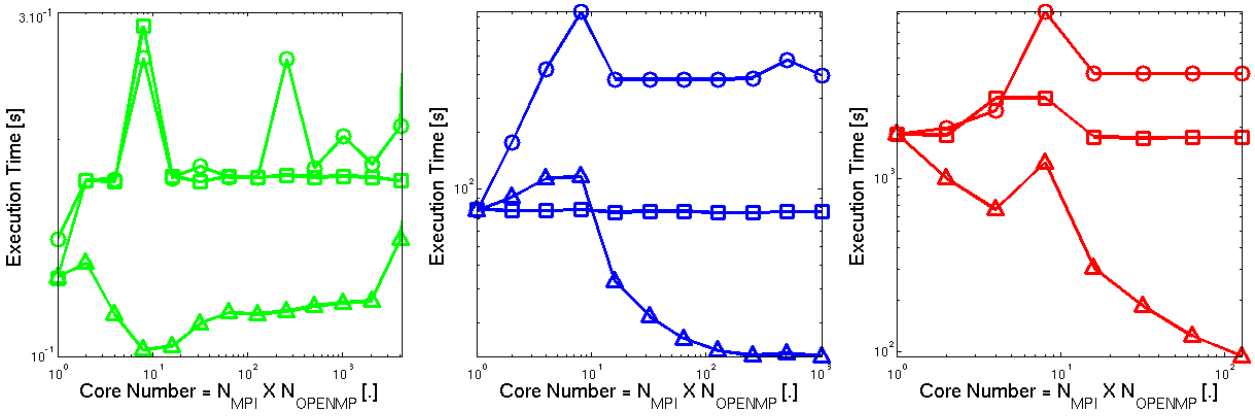


FIGURE 9. Parallelized computation of the Poisson solver performed by a sub-group of cores.

The parallelized computation of the Poisson solver performed by a sub-group of cores, Fig. 9, is in line with the previous step. However allowing the parallel Poisson solver to be split in independent linear systems shared

on several cores via OpenMP parallelization reintroduces memory access concurrency between the various calculations sharing common resources L3 cache and RAM. An other limitation of this solution is the number of cores on a given node: 12 cores on Rheticus and 16 cores on Helios. In such a scheme the speed-up that can be achieved is at most a factor 16 gain which is considerably smaller than the number of independent problems that must be solved, typically $(N_\alpha - 1)/2$ ($\gg 16$). We thus adopt an advanced MPI scheme splitting the workload over up to $\max(N_{PS}) = (N_\alpha - 1)/2$ MPI processes by dispatching groups of $((N_\alpha - 1)/2)/N_{PS}$ linear systems on MPI processes. Whenever it is possible to choose N_{PS} different nodes in the MPI scheme, one can avoid memory bandwidth limitation due to concurrent threads on the same node. In all cases we add an overhead in communications which will be sensitive to the way the N_{PS} new MPI processes are distributed.

5.2. **Evaluation of advanced MPI Poisson solvers**



(A) Case (1) $129 \times 64 \times 1 \times 65536$ (B) Case (3) $1025 \times 1024 \times 1 \times 1024$ (C) Case (4) $2049 \times 2048 \times 1 \times 128$

FIGURE 10. Computation time of the three versions of the Poisson solver on three different meshes. First version (part. 5.1.1): circle (○), second version (part. 5.1.2) : square (□) and third version (part. 5.1.3) : triangle (△)

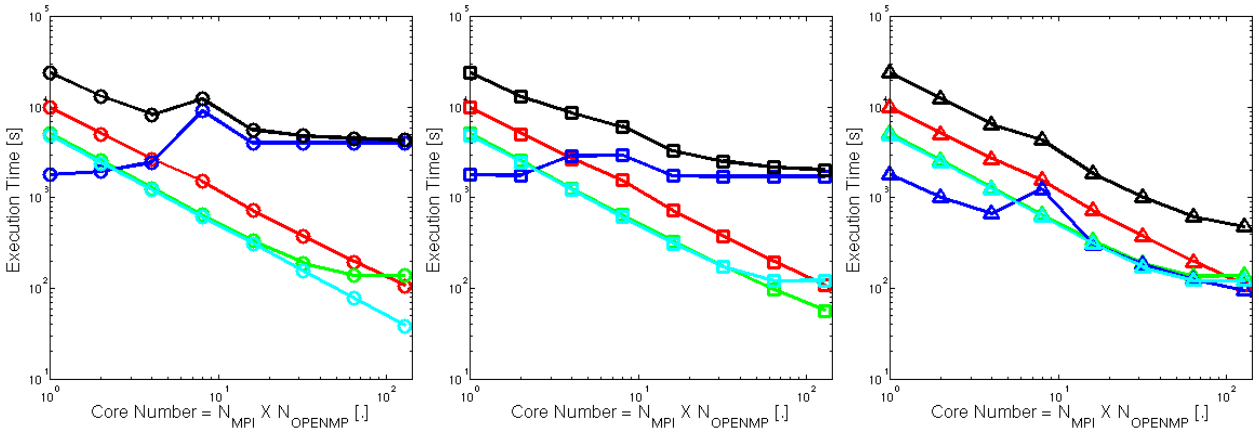
Let us first consider the Helios case 1 with the mesh $129 \times 64 \times 1 \times 65536$, Fig. 10a, which is characterized by the smallest real space sub-domain so that the Poisson solver does not govern the parallelization behavior and such that interference between various processes targeting the same resources is limited. One can notice that the first and second version of the Poisson solver exhibit similar execution time. A common peak at 8 cores, also found in other results of this Section will be discussed in Section 6.1. It is to be stressed that in that case increasing the number of cores does not lead to any gain in computation time of the Poisson solver, but rather to an increase. The third version of Poisson solver is slightly faster than the first two. Its execution time decreases when increasing the number of cores from 1 to 8 cores, it then gradually increases until 64 cores where it reaches a plateau up to 2048 cores. This plateau is determined by the maximum number of MPI processes used by the Poisson solver, namely : $N_{PS} = (N_\alpha - 1)/2 = 64$. No strong benefit from the point of view of the Poisson solver is obtained in case 1. However, considering that the Poisson solver weighs less than a few percents (Fig. 6), the overall effect stays within the fluctuation error bars. For case 1, the energy integration, the sum of 65536 slices of $2D$ sub-domains, is the major time consuming step.

For the Helios case 3 with the mesh $1025 \times 1024 \times 1 \times 1024$, Fig. 10b, the execution time of the first version of the Poisson solver increases from 1 core and peaks at 8 cores where it is $\times 10$ slower than the reference case with one core. Such a positive slope can also be observed in Fig. 10a but much stronger for this mesh. It is

due to the interference phenomenon within one CPU (see 6.1). After this peak, the computation times reaches a plateau 5 times slower than the reference one core value. The second version is constant over the whole range of number of cores. The third version starts with a positive slope, sharing workload is not sufficient to counter the interference effect. Once the load is sufficiently distributed, the interference effects are reduced and the execution time decreases. A plateau $\times 8$ faster than the reference one core is reached at 512 cores ($(N_\alpha - 1)/2$). However, due to the increased communication cost, the total execution time of TERESA is similar with the second and third Poisson solvers.

With the mesh $2049 \times 2048 \times 1 \times 128$ for the Helios case 4, Fig. 10c, the first version of the Poisson solver reaches a peak at 8 cores with a factor $\times 4.5$ slowing down, before a plateau ($\times 2$ slower). The second version has a comparable behavior than with the Helios case 3. The third version of the Poisson solver exhibits a monotonic decrease of the execution time, except at the 8 cores peak. With 128 cores, the third version is $\times 40$ faster than the first version.

The analysis of the performance of the different Poisson solvers shows that an important improvement is obtained with the third version. It is restricted to cases with a high resolution in real space and standard resolution in energy space, typical of electron simulations. This could prove to be interesting should one couple TERESA to GYSELA, where TERESA would handle trapped electrons and GYSELA the whole ion population. This would require a rather fine grid in real space and a comparable mesh in energy as used presently in GYSELA. For other uses of TERESA, in particular dedicated to kinetic simulations with very precise mesh in energy, the benefit appears to be rather weak since the cost in communications required by the energy integration step of the workflow (step 2) tends to dwarf all other effects. It is to be noted however that the strong scaling criteria used here is particularly severe since all cases that have been tested can run on a single core. Since the various numerical steps that are dispatched on the different cores are rather simple and fast, it is difficult to compensate the expensive communication cost with modest gains in run time.



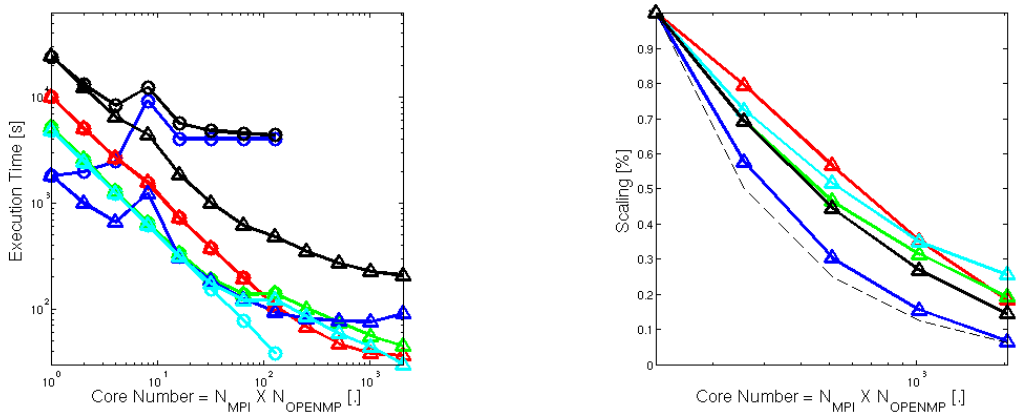
(A) First version. (B) Second version. (C) Third version.

FIGURE 11. Red : computation time of Vlasov advection (step 1); Green : integration over the energy space (step 2); Blue : Poisson solver (step 3); Cyan : computation and broadcast of $\bar{\Phi}$ (step 4); and the total of these steps in Black. Simulations have been performed on Helios using case 4 ($2049 \times 2048 \times 1 \times 128$).

5.3. **Further parallelized workload sharing with OpenMP**

Already introduced in the Section 4.1, two important limitations of the decomposition in sub-domains $S_{k,\ell}$ are, on the one hand, we have not distributed the work along the real space dimensions, on the other hand, when we use 1 core per MPI process, we are drastically constrained on the maximal number of used cores. In order to overcome those two limits, we added N_{OpenMP} OpenMP threads in real space operators to control real space workload and to increase total number of cores by changing the relation $N_{cores} = N_{MPI} \leq N_E \times N_\kappa$ into $N_{cores} = N_{MPI} \times N_{OpenMP} \leq N_E \times N_\kappa \times N_{core/node}$. Thus, the limit on the number of MPI processes is not bounded to cores but to nodes : $max(N_{core}) = max(N_{MPI}) \times max(N_{OpenMP}) = N_\kappa \times N_E \times N_{core/node}$.

Mathematical operators are often applied with independent inputs with typically $(N_\alpha - 1)/2$ or N_α or N_ψ or even $N_\alpha \times N_\psi$ independent data structures, thus we naturally introduce OpenMP directives in each main routine. Furthermore $min(N_\psi, (N_\alpha - 1)/2) \gg 16$ and one could therefore use the 16 cores of the Helios nodes to perform N_{OpenMP} OpenMP threads within a given MPI process. For example in $3D$ simulations, with a single value of κ, an $N_\kappa \times N_E = 1 \times 128$ mesh exhibits suitable numerical precision properties regarding energy space integration. On Helios ($N_{core/node} = 16$), using OpenMP increases the core number from $N_\kappa \times N_E \times 1 = 128$ to $N_\kappa \times N_E \times 16 = 2048$.



(A) First Poisson solver circles (○), third Poisson solver triangles (△). For a number of cores exceeding 128 cores, OpenMP is used.

(B) Focus on the relative efficiency of OpenMP threads from 128 MPI processes, the maximum possible of the mesh 4.

FIGURE 12. Study time of main steps and total time of TERESA core. Red: Vlasov advection (step 1), Green: energy space integration (step 2), Blue: Poisson solver (step 3), Cyan: computation and broadcast of $\bar{\Phi}$ (step 4) and Black : TERESA complete workflow. Calculation performed on Helios with mesh 4: $2049 \times 2048 \times 1 \times 128$

The effective speed-up associated to this advanced parallelization will depend on the number of mathematical operators used in the real space sub-domain that can be distributed with OpenMP. It appears that most of the costly routines are compatible (including the advection step of the Vlasov operator). At present, the main limitation is therefore the computation of the spline coefficients which is a complex matter to distribute on several cores. On the mesh 4, we can not use more than 128 MPI processes, so using OpenMP threads is relevant. Creating and killing OpenMP threads have a cost. In order to achieve the best scaling, each OpenMP threads must execute a task as big as possible compared to the threads handling cost. This requires a trade-off between using more cores and loading the OpenMP threads. The parallelization performance of the Poisson solver is found to improve with OpenMP even if this routine does not include OpenMP instructions. This effect

is due to suppression of interference phenomenon investigated in Section 6.1. Unfortunately, we can see Fig. 12b that for now, relative efficiency decreases quickly. One should not use more than 4 OpenMP threads by MPI processes, for a $\times 2$ speed-up, in order to not waste resources.

6. Hardware limits : concurrent accesses to memory resources

All the performance indicators have shown non trivial behaviors of TERESA code when scaling within one node. When scaling on several nodes, one can neglect those singularities, focuses on behaviors at the entire cluster scale. Nevertheless, understanding machine at the scales of a CPU and a node can lead to important speed-up. As the TERESA code is tough to be light and design to use a medium number of cores $(32 - 1024)$, we search to understand its behavior at all scales. Here is developed an interpretation based on the study of TERESA source code and the well known hardware limits of Helios and Rheticus like node architecture. No incompatibility is found between interpretation and results. Only the instrumentation of hardware monitor in TERESA code can validate the following interpretation. This work is presently conduct.

6.1. **Limitation introduced by interference effects when addressing the L3 cache**

The analysis of the performance of the first Poisson solver is interesting because it shows that increasing the number of computing units does not govern a simple behavior. By scaling three series of meshes using a single node, we investigate two interference phenomena. The first one is the increasing execution time when using from 1 up to $N_{core/node}/2$ cores, namely the number of cores in each CPU (each node being made of two CPUs). To point out this phenonmenon we use a series of cases with increasing N_ψ at fixed N_α, in order to evaluate the weight of the pentadiagonal solver used in the ψ direction of the Poisson solver. A second series has shown that increasing N_α at fixed N_ψ has no non linear impact on the execution time.

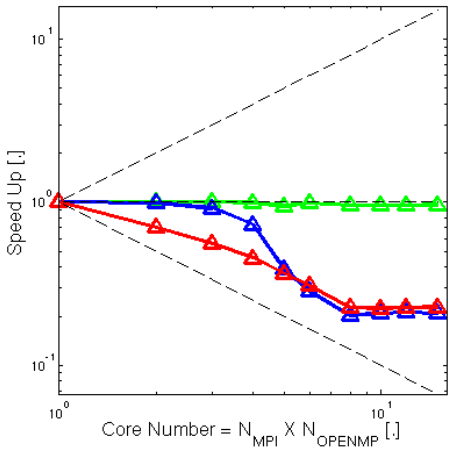


Figure 13. Speed-up of the first Poisson core solver on Helios. Real mesh : (129×256) in green, (129×512) in blue and (129×1024) in red.

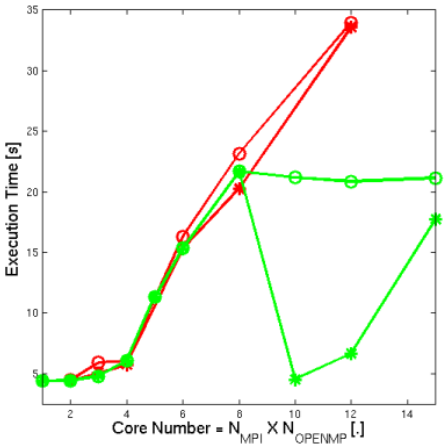


Figure 14. Run time of the first Poisson core solver on mesh (129×512) : Helios in green and Rheticus in red, slowest MPI process "o", fastest MPI process "$*$".

On Fig. 13 are plotted speed-ups versus the number of cores for different values of N_ψ. Starting from the smallest mesh, we can see that the speed-up is constant and equal to $\times 1$. The second mesh is worse and exhibits

a decrease from 1 to 8 cores where the loss in speed-up levels off at a factor $\times 1/5$. The third mesh is comparable to the second regarding the plateau effect at 8 cores or more, but the drop from one to 8 cores is more gradual. The overall increase in execution time on Rheticus is similar to that on Helios (Fig. 14). However, the effect already appears for the smallest mesh while the speed-up on Helios is constant. This non linear impact of N_ψ is characteristic of a cache effect. Indeed, as long as the data used to solve one mode in the Poisson solver can be stored in the L3 cache, the solver is fast. When too many MPI processes share the L3 cache, their overall memory requirement will exceed that of the L3 cache and thus RAM access will take place, slowing down the computation.

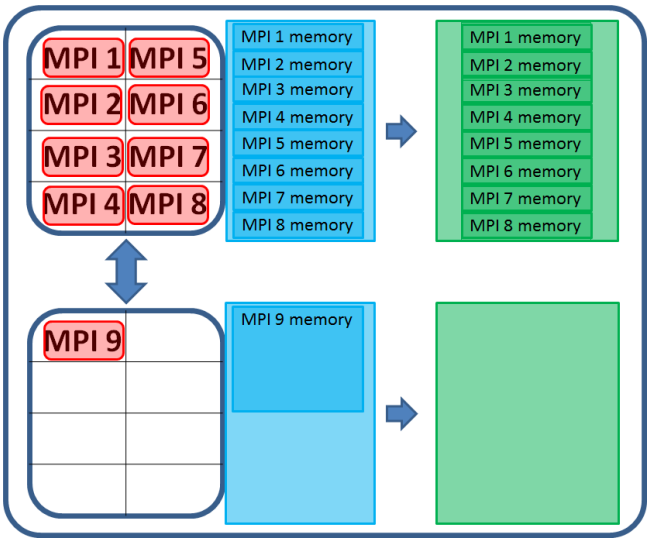


FIGURE 15. Case with strong imbalance: the eight first MPI processes load a single CPU, sharing the L3 cache. The ninth MPI process is alone on the second CPU. This configuration drives a gap between the fastest, the ninth MPI process, and the slowest, the eight first MPI processes

On Rheticus, 6 cores share a 12 MBytes L3 cache while on Helios 20 MBytes L3 cache are shared by 8 cores, see Fig. 3 and 4. We observe no drop in speed-up on Helios on the 129×256 case but the observed drop occurs at roughly 5 cores for the 129×512 case and at 2 cores for the 129×1024 case. Solving one mode require to store $8 \times (N_\psi(2 \times N_\psi + 10) + 16384)$ Bytes for assembling matrices and solving the problem. That is to say 1.15 MBytes, 4.16 MBytes and 16.20 MBytes for the three meshes tested in Fig 13. It is consistent with execution times on the three meshes : first mesh, Helios L3 memory is large enough but on Rheticus the critical L3 memory level is reached before 12 cores, a slow down clearly appears. On the second mesh, critical level is reach at 5 cores and on the last mesh at 2 cores. Finally, since Helios has a 51.6 GBytes/s RAM bandwidth per CPU compared to 32 GBytes/s on Rheticus, the slowing down effect, when it occurs, should be a factor 1.6 less important on Helios than on Rheticus. Conversely, when increasing N_α at fixed N_ψ, there is nearly no effect of the grid size from the smallest run at $N_\alpha \times N_\psi = 129 \times 256$ up to the largest at 8197×256. It confirms that the amount of memory required by this calculation and transferred to the cache does not depend on N_α. The elementary brick of the computation is one Fourier mode in α along ψ 19. Considering Helios runs, one finds that in a given range of cores there is a gap between the slowest, open circles "o", and fastest, stars "$*$", MPI process, see Fig. 14. The threshold for this gap is at nine cores where it is maximum. It then gradually decreases when increasing the number of cores and disappears when $N_{core} = N_{core/node} = 16$. This corresponds to the structure of Helios nodes made of two CPUs with eight cores each. If the operating system is not optimized to distribute the MPI processes then the eight first processes are loaded on a single CPU,

sharing the same L3 cache and memory bandwidth, Fig. 15. When adding a further process, the ninth, it will be loaded on the second CPU where it will benefit of the whole resources and therefore experience a much faster execution time. On Rheticus this effect has not been observed very likely due to an optimized distribution of the MPI processes by the operating system which tends to avoid aggregation of processes on a unique CPU.

6.2. **Limitation due to the QuickPath Interconnect (QPI) bus**

A second interference phenomenon is the peak execution time at $N_{core/node}/2$. To address this point, let us consider the N_E-scaling (N_E, the number of points in energy dimension) with different numbers of cores reported on Fig. 17. It is characterized by a strong discontinuity between $N_E = 8000$ and $N_E = 10000$ although the quasi neutrality computing scheme does not depend on the energy dimension. The execution time then exhibits a factor two increase. However, the 16 cores case (black curve) is not affected while the 8 cores case is the most affected, being the slowest due to cache effect. This behavior is characteristic of some MPI processes which have to access RAM but cannot access the RAM of their own CPU, because it is already fully allocated. The RAM required by the process is then obtained via a second CPU through a QPI bus (16). As a consequence, even with no explicit communications required at the parallelization level, communications via the QPI bus do occur and therefore can contribute to slowing the code.

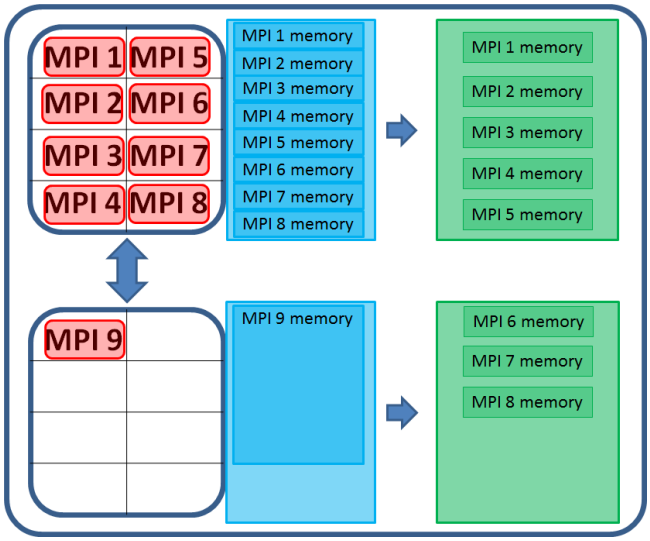


FIGURE 16. Eight MPI processes are assigned to a single CPU, sharing the L3 cache. MPI processes six, seven and eight access the RAM of the second CPU via a QPI bus.

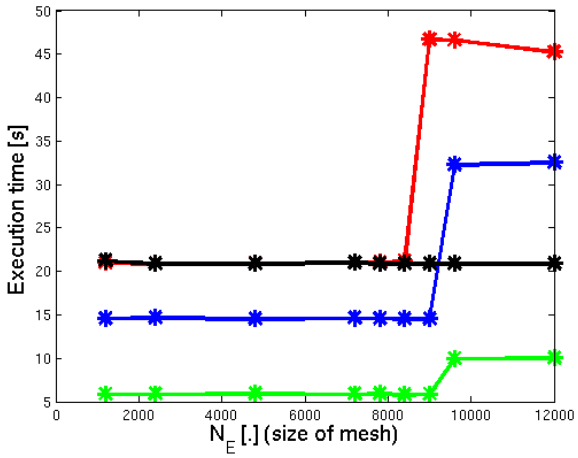


FIGURE 17. Execution time on Helios of the first Poisson core solver with increasing N_E : Green, 4 cores; Blue 6 cores; Red, 8 cores; Black, 16 cores.

7. Discussion and conclusion

In this paper we have analyzed means to improve the parallelization performance of the TERESA code. When testing the code, one found that a specific step, the Poisson solver, in the code workflow could weigh up to 90 % of the computation time with very poor relative efficiency when increasing the number of cores. Alternative algorithm have been developed to identify the issues at hand and to improve the code relative efficiency. We have found that various interference phenomena occured within the Poisson solver. These interferences are slowing down effects that appear when various processes target the same computation and memory resources

which governs a reduced efficiency. We have thus developped several schemes using both MPI and OpenMP parallelization to avoid this competition phenomena. These are tuned to take advantage of the possibilities offered by the computer architecture. Controlling the workload generated by the computer operating system also proved efficient to reduce the run time.

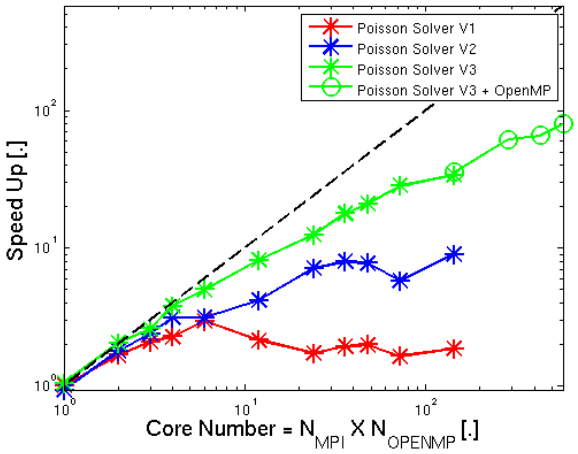


FIGURE 18. Mesh $1025\times1024\times1\times144$ on Rheticus.

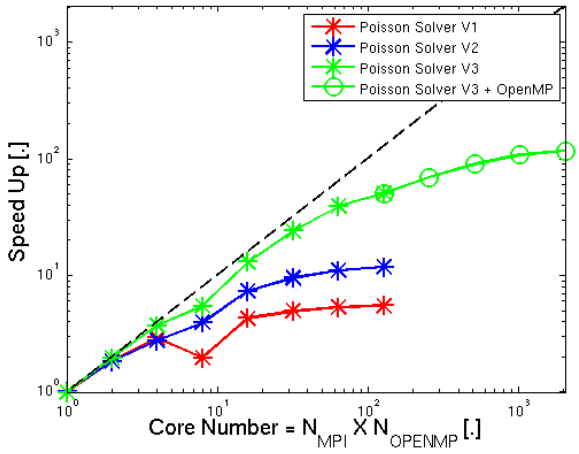


FIGURE 19. Mesh $2049\times2048\times1\times128$ on Helios.

Using two different computer systems also proved quite valuable to determine the computation properties governed by various facets of the computer architecture. On the less demanding mesh ready to simulate electrons, TERESA is now $\times20$ faster at equivalent computing cost or $\times5$ faster at $\times5$ less computational cost. On the local cluster Rheticus, Fig. 18, the code relative efficiency was upgraded from 6 cores to 288 cores and the speed-up from a factor $\times3$ to a factor $\times60$. On the international cluster Helios, Fig. 19, the initial performance was leveled-off at 32 cores and at a factor $\times5$ speed-up. Improving the Poisson solver alone opened the way to using 128 cores together with a factor $\times50$ reduction in computation time. Use of OpenMP has allowed us to further increase the performance level of TERESA with a little more than a factor $\times100$ improvement in computation time up to 1024 cores.

Having significantly improved the TERESA parallelization performance opens the way to using the code with several objectives. From the applied mathematics point of view, we provide a kinetic code that contains the key features of the sister code GYSELA, regarding both the operators that are used, in particular the gyro-average and the spline interpolation for the Vlasov advection, as well as advanced parallelization. TERESA code can then be used as a test bed for applied maths, changing various numerical routines and checking these in a complex, but less demanding, parallelized environment. The code can therefore be used as workhorse in the development of GYSELA. From the physics point of view, several issues will be addressed with the code, some requiring a significant amount of data to reach statistical equilibrium or high resolution in energy space.

REFERENCES

[1] N. Besse and M. Mehrenberger. Convergence of classes of high-order semi-lagrangian schemes for the vlasov-poisson system. *Mathematics of Computation*, 77(61):93–123, 2008.

[2] G. Darmet, Ph. Ghendrih, Y. Sarazin, X. Garbet, and V. Grandgirard. Intermittency in flux driven kinetic simulations of trapped ion turbulence. *Communications in Nonlinear Science and Numerical Simulation*, 13(1):53–58, 2008. Vlasovia 2006: The Second International Workshop on the Theory and Applications of the Vlasov Equation.

[3] C. DeBoor. *A practical guide to splines*. Springer-Verlag, New York, applied mathematical sciences 27 edition, 2001.

[4] G. Depret, X. Garbet, P. Bertrand, and A. Ghizzo. Trapped-ion driven turbulence in tokamak plasmas. *Plasma Phys. Control. Fusion*, 23(42):949–971, 2000.

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

[6] V. Grandgirard, Y. Sarazin, X. Garbet, G. Dif-Pradalier, Ph. Ghendrih, N. Crouseilles, G. Latu, E. Sonnendrücker, 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.

[7] Y. Sarazin, V. Grandgirard, E. Fleurence, X. Garbet, Ph. Ghendrih, P. Bertrand, and G. Depret. Kinetic features of interchange turbulence. *Plasma Phys. Control. Fusion*, 47(10):1817–1840, 2005.

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