# A partly matrix-free solver for the gyrokinetic field equation in three-dimensional geometry

R.Kleiber

*Max-Planck-Institut für Plasmaphysik, EURATOM Association*
*17491 Greifswald, Germany*

R.Hatzky

*Max-Planck-Institut für Plasmaphysik, EURATOM Association*
*85748 Garching, Germany*

**Abstract**

In the case of adiabatic electrons the gyrokinetic field equation for the electrostatic potential includes an averaging operator acting on flux surfaces. For realistic three-dimensional configurations, as e.g. in stellarator devices, the discretisation of this integro-differential equation leads to very large nearly dense matrices (full matrix approach) which typically cannot be stored in computer memory explicitly. A low memory consuming partly matrix-free approach, based on a preconditioned iterative matrix solver, has been developed where the Helmholtz part of the field equation is used in a matrix formulation while the averaging term is treated matrix-free. For matrices which could still be stored in memory explicitly, it is demonstrated that this approach is also much faster than the full matrix approach.

*Keywords:* gyrokinetics, stellarators, simulation

## 1. Introduction

The gyrokinetic description [1] has become a standard for the simulation of micro instabilities and turbulence in hot magnetised plasmas (e.g. in fusion devices). This description, in its simplest form, consists of a kinetic equation for the ion particle distribution function and a time-independent field equation determining the electrostatic potential $\phi$ from the density $n$ which, in turn, is calculated as a moment of the distribution function. Numerically the equations can be solved globally (i.e. in the full plasma volume) by continuum or particle-in-cell (PIC) methods: In the former approach the kinetic equation is treated as a partial differential equation and is solved using e.g. finite differences while in the PIC approach it is discretised using a particle Monte-Carlo method. For electrostatic simulations one usually employs the approximation of so-called adiabatic electrons. In this case the field equation contains a term which is the

average of the potential over a flux surface. For linear simulations of high mode number perturbations this term has no effect and can be neglected. As soon as small mode numbers get involved (e.g. due to an inverse cascade in nonlinear simulations) it gets very important since it controls the zonal flow behaviour. While the field equation without this term is of the Helmholtz type and can be solved with moderate numerical effort, it becomes a much more complicated integro-differential equation when the averaging term is included. However, for axisymmetric systems (e.g. tokamaks) the numerical effort can be reduced significantly by making use of the symmetry by transforming to Fourier space. Unfortunately, for non-axisymmetric systems (e.g. stellarator configurations) this approach leads to no improvement and it becomes necessary to develop an elaborate solver in order to pave the way to global turbulence simulations.
In the following we first state the problem in a general way without reference to gyrokinetics, then introduce the new solving strategy, its implementation and finally present an application using the stellarator PIC code EUTERPE [2].

## 2. Equations

In the following we assume generalised toroidal coordinates $x^1 = s$, $x^2 = \vartheta$, $x^3 = \varphi$ with a given metric $g^{ij}$ and Jacobian $\sqrt{g}$ (we further define $\mathrm{d}V = \sqrt{g}\,\mathrm{d}s\,\mathrm{d}\vartheta\,\mathrm{d}\varphi$ and $\mathrm{d}A = \sqrt{g}\,\mathrm{d}\vartheta\,\mathrm{d}\varphi$). Here $s \in [0,1]$ is a radial coordinate and $\vartheta, \varphi \in [0, 2\pi]$ are angle-like coordinates in the poloidal and toroidal direction, respectively. Such a coordinate system is very often used for describing toroidal magnetic systems [3]. The equation to be solved in the toroidal domain is given by

$$-\nabla_\perp \cdot \rho^2 \, \nabla_\perp \phi + \phi - \langle \phi \rangle = n \tag{1}$$

together with the Dirichlet boundary condition $\phi = 0$ at the outer boundary ($s = 1$). Here $n$ and $\rho$ are functions of the spacial variables. $\rho$ is non-zero and of the order $10^{-2}$ for a typical gyrokinetic application. The operator $\nabla_\perp \cdot \rho^2 \, \nabla_\perp$ in the first term on the left hand side of Eq. (1) acts in the $(s, \vartheta)$-plane and is given by

$$\nabla_\perp \cdot \rho^2 \, \nabla_\perp \phi \;=\; \sum_{i,j=1}^{2} \frac{1}{\sqrt{g}} \frac{\partial}{\partial x^i} \left( \rho^2 \sqrt{g} g^{ij} \frac{\partial \phi}{\partial x^j} \right). \tag{2}$$

The self-adjoint operator $\langle \phi \rangle$ is defined as the flux surface average of $\phi$ over the toroidal surface $s = \text{const.}$:

$$\langle \phi \rangle(s) \stackrel{\text{def}}{=} \frac{1}{N(s)} \int_{s=\text{const.}} \phi \, \mathrm{d}A \quad \text{with} \quad N(s) \stackrel{\text{def}}{=} \int_{s=\text{const.}} \mathrm{d}A. \tag{3}$$

Note, that due to the averaging Eq. (1) is an integro-differential equation with a time-independent symmetric positive semi-definite operator (consisting of a Helmholtz and an averaging part) on the left-hand side.

## 3. Numerical discretisation

Tensor products of B-splines (of order $\alpha$) [4] $\Lambda_\nu \stackrel{\text{def}}{=} \Lambda_i(s)\,\Lambda_j(\vartheta)\,\Lambda_k(\varphi)$ (where $\nu=(i,j,k)$ denotes a multi index) are used for a finite element discretisation of the equation on a regular grid with $N_s, N_\vartheta, N_\varphi$ grid points for the corresponding coordinate. Multiplication of Eq. (1) with $\Lambda_\nu$ and subsequent integration over the whole space gives, after using the spline representation $\phi = \sum_{\nu'} \phi_{\nu'}\Lambda_{\nu'}$, the weak formulation

$$\sum_{\nu'} \phi_{\nu'} \int \left[ \rho^2 \sum_{i,j=1}^{2} g^{ij} \frac{\partial \Lambda_\nu}{\partial x^i} \frac{\partial \Lambda_{\nu'}}{\partial x^j} + \Lambda_\nu \Lambda_{\nu'} \right] \mathrm{d}V - \int \Lambda_\nu \langle\phi\rangle \,\mathrm{d}V = \int \Lambda_\nu n \,\mathrm{d}V. \quad (4)$$

This can be written more concisely as

$$\sum_{\nu'} H_{\nu\nu'}\phi_{\nu'} - M_\nu(\phi_{\nu'}) = b_\nu \quad (5)$$

with $H_{\nu\nu'}$ the matrix representation of the Helmholtz operator and

$$M_\nu(\phi) \stackrel{\text{def}}{=} \int \Lambda_\nu \langle\phi\rangle \,\mathrm{d}V \quad (6)$$

the averaging operator. Using the matrix representation of $M_\nu(\phi)$

$$M_{\nu\nu'} \stackrel{\text{def}}{=} \int \Lambda_\nu \langle\Lambda_{\nu'}\rangle \mathrm{d}V \quad (7)$$

one arrives at the matrix representation of Eq. (1)

$$\sum_{\nu'} (H_{\nu\nu'} - M_{\nu\nu'})\phi_{\nu'} = b_\nu. \quad (8)$$

## 4. Numerical solution

For the efficient numerical solution of Eq. (8) the above problem needs to be parallelised. In the following a domain decomposition with $n_\mathrm{D}$ domains in the $\varphi$ direction is assumed [the implementation is done using the Message-Passing-Interface (MPI) library].
It is important to estimate the size of the matrices in Eq. (8): $H_{\nu\nu'}$ results from a differential operator and its filling factor (ratio of non-zero elements to total number of elements) can be estimated as $f_\mathrm{H} \sim (N_s N_\vartheta N_\varphi)^{-1}$. $M_{\nu\nu'}$, on the other hand, results from a nonlocal integral operator and has the structure of a block-banded matrix of band width $2\alpha+1$ where each block is a full matrix of size $N_\vartheta N_\varphi$. So, its filling factor is given by $f_\mathrm{M} \sim N_s^{-1}$. Consequently, $H_{\nu\nu'}$ is a sparse matrix but $M_{\nu\nu'}$ is not. Even for small cases (where typical values for the number of grid points are $N_s = N_\vartheta = N_\varphi = 64$) the matrix $M_{\nu\nu'}$ contains too many elements to be stored in computer memory explicitly, while for

$H_{\nu\nu'}$ this is feasible. Nevertheless, since for standard three-dimensional problems the dimension of $H_{\nu\nu'}$ is too large ($N_s N_\vartheta N_\varphi$), direct methods for solving the equation (e.g. by parallel sparse LU decomposition) are not efficient and preconditioned iterative methods must be used instead. For this we employ the PETSc library [5] which offers a convenient framework for choosing different parallel solvers and preconditioners. In the following we use a conjugated gradient (CG) method (see e.g. [6]) together with a block Jacobi preconditioner (which applies an ILU(0) to each block). The blocks are determined by the toroidal domain decomposition, so there are $n_\mathrm{D}$ blocks.
A considerable simplification of the numerical effort (as e.g. in PIC codes specialised in tokamak configurations) is possible if the physical system is axisymmetric, i.e. the metric coefficients and $\rho$ are independent of $\varphi$. A Fourier transformation of Eq. (1) with respect to $\varphi$ then leads to a decomposition into $N_\varphi$ two-dimensional equations where only the equation corresponding to the $n = 0$ Fourier component contains the averaging operator. The resulting matrix is small enough to fit into the memory, and the matrix equation can be solved by standard direct methods [7]. For three-dimensional systems a Fourier transformation does not give any advantage since it does not decouple the equations.
We have therefore developed a new method for solving Eq. (1): The entries of $H_{\nu\nu'}$ are time consuming to calculate but, due to its small filling factor, this matrix fits easily into the memory and can consequently be pre-computed and stored. Instead of using the matrix $M_{\nu\nu'}$ we employ a matrix-free formulation where the operator $M_\nu$ is applied to $\phi$. In the PETSc framework this can be conveniently implemented by using a matrix shell [5]. By this mechanism the solver calls a user supplied matrix-vector routine whenever it needs to multiply the matrix declared as a matrix shell with a vector. So, in order to solve a matrix equation $\sum_{\nu'} A_{\nu\nu'}\phi_{\nu'} = b_\nu$ the CG-routine delivers at each iteration step a vector $\tilde{\phi}_{\nu'}$ and the user routine has to return the result of its multiplication with $A_{\nu\nu'}$. Using the representation from Eq. (5) this results in taking the matrix-vector product of $H_{\nu\nu'}$ with $\tilde{\phi}_{\nu'}$ and applying the operator $M_\nu$ to $\tilde{\phi}_{\nu'}$. This method relies on the fact that the implementation of $M_\nu$ does not need much memory and that its application can be done in an efficient way.
The averaging operator can be written as

$$M_\nu(\phi) = \int N^{-1}\Lambda_i G_{jk} \sum_{i'} \Lambda_{i'} \sum_{j',k'} G_{j'k'}\, \phi_{i'j'k'}\, \mathrm{d}s \tag{9}$$

where the quantities

$$G_{jk}(s) \stackrel{\mathrm{def}}{=} \int \Lambda_j \Lambda_k \mathrm{d}A \tag{10}$$

have been introduced. The integral in Eq. (9) is discretised using a Gauss-Legendre integration rule with $n_\mathrm{g}$ integration points in each of the $N_s$ $s$-grid cells. In gyrokinetic PIC simulations it is convenient to use B-splines of order $\alpha \leq 3$ (see e.g. [7, 8]). Then, choosing $n_\mathrm{g} = 4$ is sufficient since it gives an exact integration for the product of two splines. The quantities $G_{jk}(s)$ involve surface integrals and are as time consuming to calculate as $H_{\nu\nu'}$. The important point

is that their memory consumption is relatively low: it is $O(n_{\mathrm{g}} N_s N_\vartheta N_\varphi)$, which is just $n_{\mathrm{g}}$ times the size of the solution vector $\phi_\nu$. Consequently, $G_{jk}(s)$ can be pre-computed and stored. In Eq. (9) the sum over $i'$ only involves those indices where a finite overlap between the splines with index $i'$ and $i$ exists. Otherwise the integral is zero since B-splines of order $\alpha$ have a finite support of $\alpha + 1$ grid cells. Thus this sum involves only $2\alpha + 1$ elements.
The most expensive part is the sum over $j'$ and $k'$ which consists of $N_\vartheta N_\varphi$ elements. Due to the domain decomposition in $\phi$, calculating this sum also involves parallel communication since the index $k'$ is distributed over the cores. Using a sum over the $n_{\mathrm{D}}$ parallel domains, the sum over $j'$ and $k'$ can be rewritten as $\sum_{l=1,n_{\mathrm{D}}} L_l(i', s)$, which is easily implemented by the MPI command MPI_ALLREDUCE. Here

$$L_l(i', s) = \sum_{k' \in D_l} \sum_{j'} G_{j'k'}(s)\, \phi_{i'j'k'} \tag{11}$$

and $D_l$ denotes the set of spline indices inside the domain $l$. So, the quantity $L_l(i', s)$ needs to be communicated but since it contains only $n_{\mathrm{g}} N_s (2\alpha + 1)$ elements the communication overhead is kept small.
For faster convergence iterative methods need a preconditioner. In case of the matrix formulation Eq. (8) a preconditioner could be derived from $H_{\nu\nu'} + M_{\nu\nu'}$. In the partly matrix-free formulation just described it would be possible to build a block Jacobi preconditioner by using the action of the operator $M_\nu$ on the unit vectors to construct the block diagonal part of $M_{\nu\nu'}$. Nevertheless, this is not useful if one wants to avoid large matrices. The most convenient choice is to derive the preconditioner from the sparse matrix $H_{\nu\nu'}$ alone, although this may not be optimal regarding the speed of convergence.

## 5. Results

The described method was implemented in the gyrokinetic code EUTERPE [2], which is a gyrokinetic $\delta f$ particle-in-cell code simulating the full three-dimensional toroidal equilibrium domain of a fusion device, e.g. a stellarator. At each time step the particle density $n$ is calculated from the particles using B-splines in the charge assignment process. This is followed by solving the field equation for the electrostatic potential, which then acts back on the particles.
For the equilibrium data of the Large Helical Device (LHD) stellarator configuration, several runs were done on the HPC-FF Linux cluster (Intel Xeon processors with a clock rate of 2.93 GHz) at the Jülich Supercomputing Centre (JSC) to investigate two cases: the partly matrix-free solver and, for the matrix formulation of the problem (Eq. (8)), the CG method. The results for these runs are displayed in Table 1. For all runs the above described block Jacobi preconditioner (with ILU(0)) has been used. A relative convergence tolerance of $10^{-5}$ for the CG method was assumed.
For building the preconditioner there are two choices: it can be derived either from $H_{\nu\nu'} + M_{\nu\nu'}$ or from $H_{\nu\nu'}$. These choices we denote by $P_{\mathrm{H+M}}$ and $P_{\mathrm{H}}$, respectively. Due to the sparseness of $H$ the preconditioner $P_{\mathrm{H}}$ needs a relatively

small amount of memory. In contrast, using the $P_{\mathrm{H+M}}$ preconditioner is only practicable for small grid sizes where it is possible to store the diagonal blocks of $M_{\nu\nu'}$ explicitly.
For reference, column three in Table 1 shows the performance of solving the field equation without the averaging term $\langle\phi\rangle$. This needs few iterations and is relatively fast. Adding the averaging term (see column four) leads to a strong increase in the number of required iterations showing that the condition of the problem gets worse. Since building the $P_{\mathrm{H+M}}$ preconditioner is no longer possible for larger matrices we chose instead only $P_{\mathrm{H}}$ (see column five). It can be seen that using the $P_{\mathrm{H}}$ preconditioner leads to a slight increase in the number of iterations and the computing time. We conclude that using $P_{\mathrm{H}}$ for large matrices is thus an economic choice.
Results for the partly matrix-free solver (using the $P_{\mathrm{H}}$ preconditioner) described in the previous section are displayed in the last column. As expected, this solver needs the same number of iterations as the non matrix-free solver with the $P_{\mathrm{H}}$ preconditioner (column five) but, especially for larger grid sizes, it needs much less time. Performing the explicit sum in Eq. (9) is thereby much faster than a matrix vector multiplication with the matrix $M_{\nu\nu'}$. The reason for this is that the summation takes advantage of the structure of the problem in an appropriate way while, on the other hand, the conventional solver uses the general purpose sparse matrix-vector product of the PETSc framework for a non-sparse matrix. Additionally, the communication overhead is minimised and the cache usage is improved.
In physical applications done with the code EUTERPE it is normally most effective to use one core for each toroidal grid cell, i.e. $n_{\mathrm{D}} = N_\varphi$. Additionally, the physics involved usually requires to have the same grid resolution in $\vartheta$ and $\varphi$, i.e. $N_\vartheta = N_\varphi$. Consequently, in order to investigate the scaling of the solver under realistic conditions, we increased the number of cores from 8 to 128 and simultaneously the number of grid points in each angular direction (the number of grid points in $s$ was held fixed at $N_s = 64$). In Figure 1 the computing time for using the partly matrix-free solver is shown (solid line). For comparison the curve for solving the problem without the averaging term is also shown (dashed line) since it shows the scaling of the underlying preconditioned CG solver (in both cases $P_{\mathrm{H}}$ was used). The scaling of both curves is similar and, as expected, nearly linear. But solving the problem with the averaging term is slower by approximately a factor of 4.6.
We also performed a weak scaling where only the grid size in the parallelised direction ($N_\varphi$) together with the number of cores was increased from 8 to 128 but the other dimensions were held fixed at $N_s = 64, N_\vartheta = 128$. In Figure 2 the resulting executing time for one iteration is shown. Note, that the jump in the curves from 8 to 16 cores is caused by the architecture of the HPC-FF machine: eight cores are located within a node. Again both curves scale similarly with a relatively weak dependency on the number of cores which shows that the communication overhead is acceptable.

In order to check the solver for a realistic three-dimensional problem we performed the so-called Rosenbluth-Hinton test [9]. In this test the evolution

of an initially specified electrostatic field is followed in time. Here the term $\langle\phi\rangle$ in Eq. (1) is essential: If it were neglected the field would just decay to zero. If, however, this term is taken into account the field performs damped oscillations with the frequency of the geodesic acoustic mode (GAM) [10] and finally settles down to a non-zero level (residual flow). For this problem the solver is a crucial building block in the typical flowchart of a PIC code. The result of a simulation for the LHD stellarator is shown in Figure 3. In this run (about three hours on 512 cores), which uses $64 \cdot 10^6$ particles and a grid size of $N_s = 32, N_\vartheta = N_\varphi = 64$, typically 67 solver iterations were necessary. The frequency of the GAM oscillation agrees very well with the analytical estimate (more details about the physics involved can be found in [11, 12]). Simulations like this become only possible with the partly matrix-free solver.

## 6. Conclusions

For gyrokinetic simulations with adiabatic electrons the equation for the electrostatic field contains an averaging operator which results in a quite cumbersome integro-differential equation. If such an equation is discretised, e.g. by finite elements using B-splines, the resulting matrix is relatively dense. Already for medium size three-dimensional problems, such a matrix cannot be stored explicitly in the main memory of today's parallel computers. Hence, the memory consuming implementation of the averaging term of the potential equation has to be treated in a matrix-free way while the other terms can be used in a matrix representation. This approach is used in a partly matrix-free solver based on a preconditioned iterative method employing the PETSc library. In addition to its small memory consumption the proposed solver is much faster (already by a factor of approximately 20 for relatively small cases) than a solver relying on an explicitly stored matrix. Its speed-up is more pronounced the larger the problem size becomes. The reason is that in its implementation the structure of the operators has been taken into account in an appropriate way and the communication overhead is reduced to a minimum. For up to 128 cores it has been demonstrated that the solver shows a very good scaling behaviour. This solver allows global simulations of zonal flows to be carried out in three dimensional stellarator configurations, which has been impossible before. Further work could be done on the reduction of the required number of iterations of the solver. Especially, low memory matrix-free preconditioners would be of interest.

## Acknowledgements

We would like to thank A. Könies for useful hints regarding PETSc and M. Borchardt for clarifying discussions.

## References


[1] T. S. Hahm, Nonlinear gyrokinetic equations for tokamak microturbulence, Phys. Fluids 31 (1988) 2670.

[2] V. Kornilov, R. Kleiber, R. Hatzky, L. Villard, G. Jost, Gyrokinetic global three-dimensional simulations of linear ion-temperature-gradient modes in Wendelstein 7-X, Phys. Plasmas 11 (2004) 3196.

[3] W. D. D'Haeseleer, W. N. Hitchon, J. D. Callen, Flux Coordinates and Magnetic Field Structure, Springer, 1991.

[4] C. DeBoor, A Practical Guide to Splines, Springer, 1978.

[5] S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, H. Zhang, PETSc users manual, Tech. Rep. ANL-95/11 - Revision 3.0.0, Argonne National Laboratory, http://www.mcs.anl.gov/petsc (2008).

[6] Y. Saad, Iterative Methods for Sparse Linear Systems, PWS, 1996.

[7] S. Jolliet, A. Bottino, P. Angelino, R. Hatzky, T. M. Tran, B. F. Mcmillan, O. Sauter, K. Appert, Y. Idomura, L. Villard, A global collisionless PIC code in magnetic coordinates, Comp. Phys. Commun. 177 (2007) 409.

[8] M. Fivaz, S. Brunner, G. de Ridder, O. Sauter, T. M. Tran, J. Vaclavik, L. Villard, K. Appert, Finite element approach to global gyrokinetic Particle-In-Cell simulations using magnetic coordinates, Comp. Phys. Commun. 111 (1998) 27.

[9] M. N. Rosenbluth, F. L. Hinton, Poloidal flow driven by ion-temperature-gradient turbulence in tokamaks, Phys. Rev. Lett. 80 (1998) 724.

[10] N. Winsor, J. L. Johnson, J. M. Dawson, Geodesic acoustic waves in hydromagnetic systems, Phys. Fluids 11 (1968) 2448.

[11] R. Kleiber, R. Hatzky, A. Mishchenko, Simulation of residual zonal flow levels in stellarators including a radial electric field, Contributions to Plasma Physics 50 (2010) 766.

[12] P. Helander, A. Mishchenko, R. Kleiber, P. Xanthopoulos, Oscillations of zonal flows in stellarators, Plasma Phys. Controlled Fusion 53 (2011) 054006.

# Captions

Table 1: Time (in seconds) and number of iterations for different grid sizes and number of cores on the HPC-FF machine at JSC. Column three: problem without averaging term. Column four and five: problem with averaging term using a matrix formulation for $M$. Column six: problem with averaging term using the partly matrix-free solver. The type of preconditioner, if derived from $H$ or $H + M$, is denoted by $P_{\mathrm{H}}$ or $P_{\mathrm{H+M}}$.

Figure 1: Computing time as a function of the number of cores. Solid line: problem with averaging term, partly matrix-free solver. Dashed line: problem without averaging term, matrix solver.

Figure 2: Computing time per iteration as a function of the number of cores (weak scaling). Solid line: problem with averaging term, partly matrix-free solver. Dashed line: problem without averaging term, matrix solver.

Figure 3: Time evolution of the normalised radial electric field at position $s = 0.5$ for the Rosenbluth-Hinton test in the LHD stellarator configuration.

| grid size $N_s \times N_\vartheta \times N_\varphi$ | cores | H $P_{\mathrm{H}}$ time [s] (iter) | H+M $P_{\mathrm{H+M}}$ time [s] (iter) | H+M $P_{\mathrm{H}}$ time [s] (iter) | H+M, matrix-free $P_{\mathrm{H}}$ time [s] (iter) |
|---|---|---|---|---|---|
| $8^3$ | 8 | 1.7E-3 (17) | 6.3E-3 (43) | 7.5E-3 (52) | 6.0E-3 (52) |
| $16^3$ | 16 | 2.8E-3 (17) | 7.3E-2 (59) | 7.9E-2 (66) | 1.2E-2 (66) |
| $32^3$ | 32 | 9.8E-3 (17) | 1.0 (59) | 1.1 (71) | 4.5E-2 (71) |

Table 1

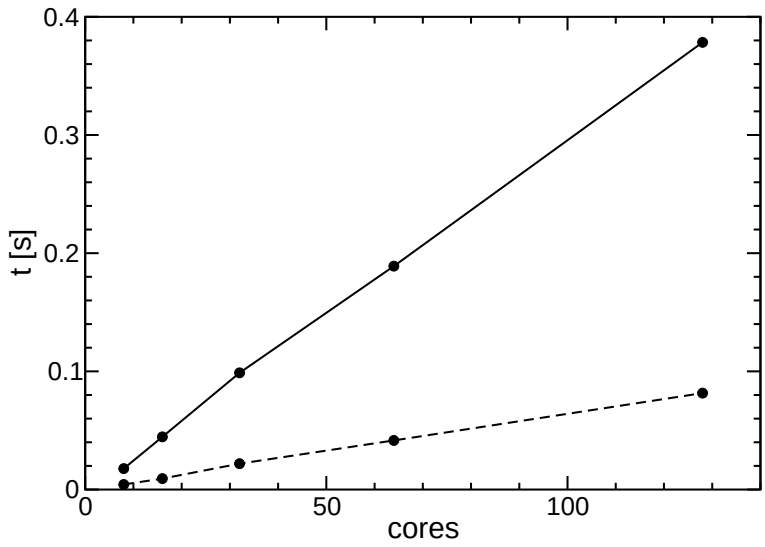

Figure 1

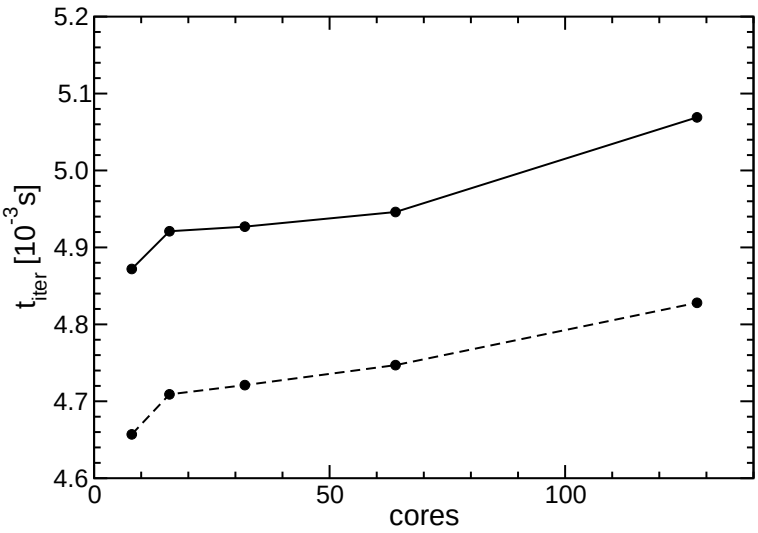

Figure 2

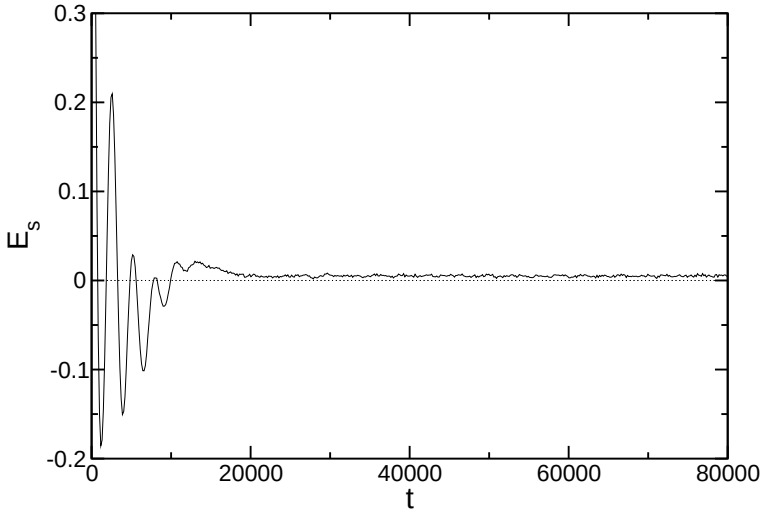

Figure 3