ELSEVIER

Parallel Computing 27 (2001) 295–314

PARALLEL COMPUTING

www.elsevier.com/locate/parco

Applications

Parallel PIC plasma simulation through particle decomposition techniques

B. Di Martino [a], S. Briguglio [b,*], G. Vlad [b], P. Sguazzero [c]

[a] *Dipartimento di Informatica e Sistemistica, University of Naples "Federico II", Naples, Italy*
[b] *Associazione Euratom-ENEA sulla Fusione, Centro Ricerche Frascati, C.P. 65, Via Enrico Fermi 45, I-00044 Frascati, Rome, Italy*
[c] *IBM, Via Shanghai 53, I-00144 Rome, Italy*

Received 15 September 1998; received in revised form 19 January 2000; accepted 29 May 2000

Abstract

Parallelization of a particle-in-cell (PIC) code has been accomplished through a "particle decomposition" technique instead of the more usual "domain decomposition" one. The adopted technique requires a moderate effort in porting the code in parallel form and results in intrinsic load balancing and modest inter-processor communication. The resulting data parallel implementation has been carried out within the High Performance Fortran (HPF) framework, and tested on the IBM SP parallel system. The performance tests obtained confirm the hypothesis of high effectiveness of the strategy, if targeted towards moderately parallel architectures. Optimal use of resources is also discussed with reference to a specific physics problem. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Plasma simulation; Particle-in-cell simulation; Particle decomposition; High Performance Fortran; High Performance Fortran extrinsic procedure

1. Introduction

One of the main purposes of the theoretical investigation in the field of magnetic-confinement fusion is represented by the comprehension of the transport mechanisms which are responsible for the confinement degradation of the plasma.

* Corresponding author. Fax: +6-9400-5400.
E-mail address: briguglio@frascati.enea.it (S. Briguglio).

It is generally recognized that such mechanisms can be traced back to turbulent phenomena associated to the fluctuations that develop in plasmas due to the presence of equilibrium inhomogeneities.

A relevant role in determining the stability of the plasma with respect to these fluctuations, as well as their saturation level and, eventually, the magnitude of the transport coefficients is played by kinetic effects, like the resonant wave–particle interaction. These effects depend on the details of the particle distribution function or, more precisely, of its deviation from the equilibrium one; thus, they require the direct solution of Boltzmann equation, and cannot be studied in the framework of a fluid magnetohydrodynamic (MHD) description of the plasma. Such a description, indeed, corresponds to taking a set of velocity-space momenta of Boltzmann equation and closing them with suited approximation of the unknown transport coefficients.

The task of exactly solving Boltzmann equation for a particle population which evolves in the presence of external fields and particle–particle interactions can be greatly simplified, for a plasma, because the latter are dominated by long-range effects. This is true as long as the number of particles in a Debye sphere is much larger than one ($n\lambda_D^3 \gg 1$, with λ_D being the Debye length and n the particle density), and it means that the collision term, related to short-range interactions, can be neglected in the Boltzmann equation, which then reduces to the Vlasov one. Long-range interactions maintain, instead, their role in this equation, under the form of smooth electromagnetic fields. An accurate description of these interactions can then be obtained by particle-in-cell (PIC) simulation techniques [2], which consist in following the continuous phase-space evolution of a “macro-particle” (or simulation-particle) population, each of them representing a cloud of non-mutually interacting physical particles. The mutual interaction between each pair of simulation particles is taken into account by means of an electromagnetic field, computed only at the points of a discrete spatial grid and then interpolated at each particle position. By identifying the charge and the mass of each simulation particle with those of the whole cloud, and imposing that such a particle moves as its physical counterpart, all the relevant parameters (Debye length, Larmor radius, etc.) of the simulation plasma coincide with the corresponding parameters of the physical plasma, notwithstanding that the simulation-particle density is much lower than the physical-particle one.

Mediating particle–particle interactions by fields reduces the order of magnitude of the number of operations needed for each time step from $O(N_{\text{part}}^2)$ (typical of a N-body calculation) to $O(N_{\text{part}})$, with N_{part} being the number of simulation particles. Moreover, the discreteness of the field computation and the consequent artificial cutoff imposed on the short-distance interactions, far from negatively affecting the overall calculation (such interactions are indeed expected to be physically irrelevant), allows for satisfying the *physical* condition of dominating long-range interactions even if the corresponding requirement, $n\lambda_D^3 \gg 1$, cannot be fulfilled by the *simulation* plasma. In fact, it can be shown that the latter requirement is replaced by the less stringent one, $nL_c^3 \gg 1$, with L_c being the typical spacing of the grid points. In other words, the presence of a discrete grid avoids that short-range interactions become spuriously dominant in the numerical simulation, due to the relatively small number of simulation particles.

Between the two extreme simulation approaches represented by MHD and PIC codes – the former being suited for the investigation of macroscopic plasma behavior, the latter for the exploration of kinetic effects – several intermediate (hybrid) techniques have been developed in order to address "mixed" problems, in which a particular plasma component (e.g., a particle species) exhibits relevant kinetic behavior, while the other components do not appreciably depart from fluid-like dynamics.

A relevant example of these kind of mixed problems is represented by the investigation on linear and non-linear behavior of the shear-Alfvén modes in tokamaks. Shear-Alfvén waves are transverse waves, which propagate along the magnetic-field lines with group velocity equal to the Alfvén velocity $v_A \equiv B/\sqrt{4\pi n_i m_i}$, where B is the magnitude of the magnetic field and n_i and m_i are the bulk-ion density and mass, respectively. The displacement of the fluid element and the perturbation of magnetic field oscillate in phase, behaving as a massive elastic string under tension and causing field-line bending. Such modes can be destabilized, in plasmas close to ignition conditions, by the resonant exchange of energy with the energetic α particles produced, e.g., by fusion reactions, whose confinement properties can, in turn, be strongly affected by the non-linear interaction with the modes themselves. Due to the high values of the phase velocity of these modes (of the order of the Alfvén velocity), the details of the distribution function of electrons and bulk ions (deuterons and tritons) can be neglected, and these two components play a role in determining Alfvén modes behavior, only affecting the fluid properties of the system.

On the basis of theoretical studies, the most unstable Alfvén modes are expected to be those characterized by wavelengths of the order of few hundredths of the typical size of the macroscopic system. The need for adequate spatial resolution then impose to handle spatial grids with hundreds thousand cells. Such a requirement, together with the above condition of large number of simulation particles per cell, brings to face simulations characterized by millions particles, beyond the limits of the computational resources offered by today single-processor computers. It is then a high-priority task that of implementing efficient distributed versions of existing numerical codes, in order to adequately exploit parallel architectures.

Several contributions exist, in literature, on this issue [1,9,12,25,28]. They are all based on the *domain decomposition* strategy [15,25]: different portions of the physical domain and of the corresponding grid are assigned to different processors, together with the particles that reside on them. In particular, Ref. [12] and, especially, Ref. [9] compare the results obtained by the parallelized code on different architectures; Ref. [28] discusses the benefits of the object-oriented approach to the parallel PIC simulation; Ref. [1] presents the results of the implementation of parallel PIC codes in High Performance Fortran [21] (HPF) and compares them with those obtained in a message passing framework. A common feature of these experiences is given by the need, for general plasma simulations, of a dynamic load balancing, associated to particle migration from one portion of the domain to another one. Such a load balancing can make the parallel implementation of a serial code very complicate, besides introducing extra computational loads.

Aim of this paper is to present a different parallelization strategy, based on the condition $nL_c^3 \gg 1$ and applied to a hybrid MHD-PIC code (but, in fact, applicable to any PIC code). Different from the *domain decomposition* approach, such a strategy resorts to *particle decomposition*: the whole spatial grid is assigned to each processor, which however takes care only of a subset of the particle population. Partial contributions to the total pressure at the grid points, which is required to update the electromagnetic fields, are then communicated among processors and summed together. This strategy implies a *data parallel* execution scheme for the implementing code. Thus, the approach presents, over the domain-decomposition strategy, the advantages of (1) intrinsic load balancing and very modest amount of inter-processor communication, and (2) moderate effort in porting existing sequential programs into parallel form: the HPF parallel model is well suited for this strategy, and the parallelization of a sequential Fortran 77/Fortran 90 code can be carried out quite straightforwardly by translating it into a HPF program.

On the other side, the strategy comes out to be efficient only as long as the condition of a large average number of particles per cell is satisfied even on the single processor. In the opposite case, indeed, the effort required to calculate and communicate grid quantities becomes comparable with, or even greater than, the effort needed for particle calculation. Although it cannot then represent the optimal solution for the ambitious task of an indefinitely scalable spatial resolution, the particle-decomposition method is in practice particularly suited for code porting on moderately parallel architectures (up to few tens of processors, each of whom capable to support the entire spatial grid).

The paper is organized as follows. Section 2 introduces the mathematical model for the investigation of Alfvén modes. Section 3 describes the essential features of the hybrid MHD-gyrokinetic code (HMGC), an initial-value code implemented to simulate the behavior of these modes and the parallelization strategy based on particle decomposition. The particular implementation in the HPF framework is presented in detail in Section 4. Speed-up factors obtained by the parallel version of the code are reported in Section 5, while Section 6 contains remarks concerning the validity of the proposed decomposition. Conclusions are drawn in Section 7.

2. The mathematical model for hybrid fluid-particle simulation of Alfvén modes

An adequate model for the investigation of Alfvén-mode stability and saturation is yielded by the following resistive-MHD equations, containing a driving term related to the energetic-ion population [30]:

$$\frac{\mathrm{d}\varrho}{\mathrm{d}t} = -\varrho\nabla\cdot\mathbf{v},$$

$$\varrho\frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = -\nabla\mathbf{P} - \nabla\cdot\mathbf{\Pi}_H + \frac{1}{c}\mathbf{J}\times\mathbf{B},$$

$$\frac{\partial\mathbf{B}}{\partial t} = -c\nabla\times\mathbf{E},$$

$$\frac{\mathrm{d}P}{\mathrm{d}t} = -\hat{\gamma}P\nabla\cdot\mathbf{v},$$

$$\begin{aligned} \mathbf{E} &= \eta \mathbf{J} - \frac{1}{c} \mathbf{v} \times \mathbf{B}, \\ \mathbf{J} &= \frac{c}{4\pi} \nabla \times \mathbf{B}, \\ \nabla \cdot \mathbf{B} &= 0. \end{aligned} \tag{1}$$

In the above equations, $\mathbf{v}$ is the fluid velocity, $\mathbf{J}$ the plasma current, $\mathbf{B}$ the magnetic field, $\mathbf{E}$ the electric field, ϱ and P are, respectively, the mass density and the scalar pressure of the bulk plasma, $\hat{\gamma}$ the ratio of the specific heats, η the resistivity, c the speed of light, and $\mathrm{d}/\mathrm{d}t \equiv \partial/\partial t + \mathbf{v} \cdot \nabla$. The term $\mathbf{\Pi}_H$ is the pressure tensor of the energetic (hot) ions; it can be expressed in terms of the corresponding distribution function ($\mathbf{\Pi}_H \equiv m_H \int \mathrm{d}^3 v \mathbf{v}\mathbf{v} f_H$, with m_H being the energetic-ion mass), to be determined by solving Vlasov equation.

Two important simplifications can be introduced in the above model, by taking into account the particular situation of interest in the field of tokamak plasmas for nuclear-fusion research [27]. Tokamak devices are characterized by toroidal configurations with major radius R_0 much larger than the minor radius a, and intense magnetic field. The first simplification is then obtained [32] by expanding Eqs. (1) in powers of the small inverse aspect ratio $\epsilon \equiv a/R_0$. At the order ϵ^3, neglecting the bulk-plasma pressure and the time dependence of the density, and introducing a cylindrical-coordinate system (R, Z, φ), Eqs. (1) reduce, in terms of the poloidal-magnetic-field stream function ψ ($\mathbf{B}_\mathrm{p} \equiv R_0 \nabla\psi \times \nabla\varphi$) and the scalar potential ϕ, to the following form [22,33]:

$$\begin{aligned} \frac{\partial \psi}{\partial t} &= -\frac{cR^2}{R_0 B_0} \nabla\psi \times \nabla\varphi \cdot \nabla\phi - \frac{c}{R_0} \frac{\partial \phi}{\partial \varphi} + \eta \frac{c^2}{4\pi} \Delta^* \psi + \mathrm{O}\left(\epsilon^4 v_A B_\varphi\right), \\ \hat{\varrho} &\left(\frac{\mathrm{D}}{\mathrm{D}t} - \frac{2c}{R_0 B_0} \frac{\partial \phi}{\partial Z} \right) \nabla_\perp^2 \phi + \nabla\hat{\varrho} \cdot \left(\frac{\mathrm{D}}{\mathrm{D}t} - \frac{c}{R_0 B_0} \frac{\partial \phi}{\partial Z} \right) \nabla\phi \\ &= -\frac{B_0}{4\pi c} \mathbf{B} \cdot \nabla \Delta^* \psi - \frac{B_0}{c R_0} \nabla \cdot \left(R^2 \nabla \cdot \mathbf{\Pi}_H \times \nabla\varphi \right) + \mathrm{O}\left(\epsilon^4 \varrho \frac{v_A^2}{a^2} \right), \end{aligned} \tag{2}$$

where

$$\hat{\varrho} = \frac{R^2}{R_0^2} \varrho, \quad \frac{\mathrm{D}}{\mathrm{D}t} = \frac{\partial}{\partial t} + \mathbf{v}_\perp \cdot \nabla, \quad \mathbf{v}_\perp \approx -\frac{cR^2}{R_0 B_0} \nabla\phi \times \nabla\varphi,$$

$$\nabla_\perp^2 \equiv \frac{1}{R} \frac{\partial}{\partial R} R \frac{\partial}{\partial R} + \frac{\partial^2}{\partial Z^2},$$

the Grad–Shafranov operator Δ^* is defined by

$$\Delta^* \equiv R \frac{\partial}{\partial R} \frac{1}{R} \frac{\partial}{\partial R} + \frac{\partial^2}{\partial Z^2},$$

B_0 is the vacuum magnetic-field amplitude at $R = R_0$ and the subscript $\perp$ denotes components perpendicular to $\nabla\varphi$.

The second simplification concerns the determination of the energetic-ion distribution function, needed to compute the pressure tensor and to close the set of reduced MHD equations (2). It is worth [23,24] to solve Vlasov equation in the gyrocenter-coordinate system $\overline{Z} \equiv (\overline{\mathbf{R}}, \overline{M}, \overline{U}, \overline{\theta})$, where $\overline{\mathbf{R}}$ is the gyrocenter position, $\overline{M}$ the magnetic momentum, $\overline{U}$ the parallel velocity (i.e., the velocity along the magnetic-field line) and $\overline{\theta}$ is the gyrophase. This corresponds to averaging the single-particle equations of motion over the fast Larmor precession and allows one to retain the relevant finite-Larmor-radius effects (the time scales of the phenomena of interest being much longer than the precession period) without resolving the details of the gyromotion. Such a choice is particularly suited for numerical particle pushing, as the numerical-stability constraint on the time-step size comes out to be much less severe than in the unaveraged case.

The hot-particle pressure tensor assumes, in terms of the gyrocenter-coordinate distribution function, the form

$$\begin{aligned} \mathbf{\Pi}_H(t, \mathbf{x}) = & \frac{1}{m_H^2} \int \mathrm{d}^6 \overline{Z} D_{z_c \to \overline{Z}} \overline{F}_H(t, \overline{\mathbf{R}}, \overline{M}, \overline{U}) \\ & \times \left[\frac{\Omega_H \overline{M}}{m_H} \mathbf{I} + \hat{\mathbf{b}}\hat{\mathbf{b}} \left(\overline{U}^2 - \frac{\Omega_H \overline{M}}{m_H} \right) \right] \delta(\mathbf{x} - \overline{\mathbf{R}}), \end{aligned} \tag{3}$$

where $\mathbf{I}$ is the identity tensor ($I_{ij} \equiv \delta_{ij}$), $\overline{F}_H(t, \overline{\mathbf{R}}, \overline{M}, \overline{U})$ the energetic-particle distribution function in gyrocenter coordinates, and $D_{z_c \to \overline{Z}}$ is the Jacobian of the transformation from canonical to gyrocenter coordinates.

Vlasov equation can be written as

$$\frac{\mathrm{d}\overline{F}_H}{\mathrm{d}t} = 0, \tag{4}$$

with

$$\frac{\mathrm{d}}{\mathrm{d}t} \equiv \frac{\partial}{\partial t} + \frac{\mathrm{d}\overline{Z}^i}{\mathrm{d}t} \frac{\partial}{\partial \overline{Z}^i},$$

with $\mathrm{d}\overline{Z}^i/\mathrm{d}t$ given by the following equations of motion [4,10,18,19]:

$$\begin{aligned} \frac{\mathrm{d}\overline{\mathbf{R}}}{\mathrm{d}t} &= \overline{U}\hat{\mathbf{b}} + \frac{e_H}{m_H \Omega_H} \hat{\mathbf{b}} \times \nabla\phi - \frac{\overline{U}}{m_H \Omega_H} \hat{\mathbf{b}} \\ & \qquad \times \nabla a_\parallel + \left[\frac{\overline{M}}{m_H} + \frac{\overline{U}}{\Omega_H} \left(\overline{U} + \frac{a_\parallel}{m_H} \right) \right] \hat{\mathbf{b}} \times \nabla \ln B, \\ \frac{\mathrm{d}\overline{M}}{\mathrm{d}t} &= 0, \\ \frac{\mathrm{d}\overline{U}}{\mathrm{d}t} &= \frac{1}{m_H} \hat{\mathbf{b}} \cdot \left\{ \left[\frac{e_H}{\Omega_H} \left(\overline{U} + \frac{a_\parallel}{m_H} \right) \nabla\phi + \frac{\overline{M}}{m_H} \nabla a_\parallel \right] \times \nabla \ln B + \frac{e_H}{m_H \Omega_H} \nabla a_\parallel \times \nabla\phi \right\} \\ & \qquad - \frac{\Omega_H \overline{M}}{m_H} \hat{\mathbf{b}} \cdot \nabla \ln B. \end{aligned} \tag{5}$$

Here e_H and $\Omega_H \equiv e_H B / m_H c$ are, respectively, the energetic-ion charge and Larmor frequency; $\hat{\mathbf{b}} \equiv \mathbf{B}/B$ is the unit vector of the equilibrium magnetic field. The fluctuating potential $a_\parallel$ is related to the stream function ψ through the relationship

$$a_{\|} = \frac{e_H}{c}\frac{R_0}{R}\psi. \tag{6}$$

Note that the magnetic momentum $\overline{M}$ is exactly conserved in this coordinate system and that, correspondingly, neither $\overline{F}_H$ nor the equations of motion contain any dependence on the gyrophase $\overline{\theta}$.

Particle-simulation approach to the solution of Vlasov equation, Eq. (4), consists in representing any phase-space function $G(t,\overline{Z})$ by its discretized form,

$$G(t,\overline{Z}) \equiv \int \mathrm{d}^6\overline{Z'}G(t,\overline{Z'})\delta(\overline{Z}-\overline{Z'}) \approx \sum_l \Delta_l G(t,\overline{Z}_l)\delta(\overline{Z}-\overline{Z}_l), \tag{7}$$

where Δ_l is the volume element around the phase-space marker $\overline{Z}_l$, and assuming that each marker evolves in time according to the gyrocenter equations of motion, Eqs. (5). Such markers can then be interpreted as the phase-space coordinates of a set of N_{part} "particles", and $G(t,\overline{Z})$ can be approximated by

$$G(t,\overline{Z}) \approx \sum_{l=1}^{N_{\text{part}}} \Delta_l(t)G(t,\overline{Z}_l(t))\delta(\overline{Z}-\overline{Z}_l(t)). \tag{8}$$

The time-variation of the volume element $\Delta_l(t)$ is then given by

$$\frac{\mathrm{d}\Delta_l}{\mathrm{d}t} = \Delta_l(t)\left(\frac{\partial}{\partial\overline{Z^i}}\frac{\mathrm{d}\overline{Z^i}}{\mathrm{d}t}\right)_{t,\overline{Z}_l(t)}. \tag{9}$$

For the purpose of calculating the pressure-tensor components, Eq. (3), it is worth to represent directly the quantity $D_{z_c\to\overline{Z}}\overline{F}_H$ according to its discretized form

$$D_{z_c\to\overline{Z}}(t,\overline{Z})\overline{F}_H(t,\overline{Z}) \approx \sum_{l=1}^{N_{\text{part}}} \overline{w}_l(t)\delta(\overline{Z}-\overline{Z}_l(t)), \tag{10}$$

with the weight factor $\overline{w}_l$ defined by

$$\overline{w}_l(t) \equiv \Delta_l(t)D_{z_c\to\overline{Z}}(t,\overline{Z}_l(t))\overline{F}_H(t,\overline{Z}_l(t)). \tag{11}$$

From Eqs. (4), (9) and from the Liouville theorem,

$$\frac{\partial}{\partial t}D_{z_c\to\overline{Z}} + \frac{\partial}{\partial\overline{Z^i}}\left(D_{z_c\to\overline{Z}}\frac{\mathrm{d}\overline{Z^i}}{\mathrm{d}t}\right) = 0, \tag{12}$$

it is immediate to show that

$$\frac{\mathrm{d}\overline{w}_l}{\mathrm{d}t} = 0. \tag{13}$$

3. A particle decomposition strategy for the parallelization of the hybrid MHD-gyrokinetic code

A hybrid MHD-gyrokinetic initial-value code has been developed [4] to integrate the coupled sets of reduced MHD equations for the fluctuating fields, Eqs. (2), and gyrokinetic equations of motion for the collision-free energetic ions, Eqs. (5).

It is an approximatively 16,000-lines F77 code, distributed over more than 40 procedures. The core computation performs a typical PIC plasma simulation. At a very coarse grain level, the structure of the computation can be described as follows.

The physical domain is represented by a three-dimensional toroidal grid, for which quasi-cylindrical coordinates are adopted: the minor radius of the torus, r, and the poloidal and toroidal angles, ϑ and φ, respectively.

Phase-space coordinates and weights for the simulation particles are initially determined in such a way to yield a prescribed (e.g., Maxwellian) distribution function.

At each time step:

1. a field solver computes the fluctuating electromagnetic fields at the N_{grid} grid points of the toroidal domain;
2. phase-space coordinates and weights of the particles are then evolved (*particle pushing*) by using the fluctuating fields interpolated at each particle position, according to Eqs. (5), (9) and (13);
3. the pressure-tensor components at the grid points are updated, in order to close the MHD equations for the next time step, by distributing the contribution of each particle among the vertices of the cell the particle belongs to.

The field solver for the $\mathrm{O}(\epsilon^3)$ reduced MHD equations [33,34] is based on a previous $\mathrm{O}(\epsilon^2)$ version [3] and uses finite differences in the minor radius direction and Fourier expansion in the poloidal and toroidal directions.

Particle pushing is performed by integrating Eqs. (5), (9) and (13) by a second-order Runge–Kutta method, more accurate ($\mathrm{O}(\Delta t^2)$ is properly retained), though more expensive, than the standard $\mathrm{O}(\Delta t)$ Euler method. Field values at each particle position are obtained by trilinear interpolation of the fields at the vertices of the cell the particle belongs to. A corresponding trilinear weight function is adopted in order to distribute the contribution of each particle to the pressure-tensor components among the vertices of the cell.

Several quantities (both scalar and vectorial) – e.g., the number of particles that hit the wall ($r = a$) and are considered lost – are also calculated cumulating particle contributions in time, while performing the main computation.

The high resolution, both in physical and in velocity space, needed for realistic, accurate and noise-free simulations, has justified a large effort, in the last years, in implementing parallel versions of PIC codes. The most widely explored approach [25,15] consists in decomposing the physical domain among the different processors, assigning to each of them only the portion of particle population that resides into the subdomain of pertinence. At each time step, particles that have migrated from a subdomain to another must be withdrawn from the original processor and assigned to the new one.

The most important merit of such a method is represented by the intrinsic scalability of the physical-space resolution with increasing number of processors: keeping the resolution in velocity space fixed, indeed, the maximum number of cells that a single processor can handle is determined by its Random Access Memory (RAM) resources; adding further processors then allows for adding new cells to the physical domain and hence, once the whole-domain size has been fixed, for increasing the spatial resolution of the simulation. Unfortunately, this positive feature is balanced by at least the two following negative elements. First, parallelization of an existing serial code is not straightforward: tasks as updating each-processor population after particle pushing, for example, are peculiar of the parallel version and make it completely different from the original serial one. Second, serious load-balancing problems can take place: in principle, the whole-particle population could be assigned, at a certain time step, to the same processor, causing the advantages of parallelization to vanish or requiring complicate dynamical redefinition of the subdomains.

On the basis of the previous considerations, it can be advantageous, in several practical cases, to adopt an alternative approach to parallelization, consisting in distributing statically the particle population among the processors, while assigning the whole domain to each processor. The respective merits of this "particle" *data decomposition* are exactly complementary to those of the domain decomposition: the differences between the serial and the parallel version are expected to be very contained and load balancing is automatically ensured (except for the typically small amount of lost particles); on the opposite side, the physical-space resolution is limited by the RAM resources of the single processor, and increasing the number of processors only allows for increasing the velocity-space resolution.

Anyway, if the number of particles assigned to each processor is much larger than the number of cells (the domain size), the single-node memory occupation of a replicated domain becomes negligible with respect to the memory occupation of a particle decomposition. This condition is equivalent to require that the number of particles per cell must be sensibly larger than the number of processors. Thus, if the target architecture presents a moderate number of nodes, not exceeding the average number of particles per cell, the benefits of a domain decomposition on the memory occupation become negligible with respect to a particle decomposition. A large number of particles per cell represents anyhow, as stated above, the condition for the accuracy of the simulation (interactions dominated by long-range effects); thus, such a limitation on the number of nodes is typically not too severe.

The condition of large number of particles per cell, which is needed for both the accuracy of the simulation and the effectiveness of the particle-decomposition strategy, plays a role also in the definition of a strategy for the *work distribution*. In fact, such a condition means that the work related to the on-grid field-computation *first phase* is negligible with respect to the one needed for the other two phases (particle pushing and pressure-tensor updating), which involve scans over the particle population. The computational load is thus of the order $O(N_{\mathrm{part}})$, with N_{part} being the number of simulation particles; it is then worth to distribute the second- and third-phase work, and to replicate the first-phase one, consistently with the replication of the domain.

Distributing work for the second and third phases means distributing the computations that access (read and/or modify) the data structures related to the particles.

The *second phase* is thus inherently parallel, with no communication required by non-local accesses (except for the computation of the time-cumulating quantities; see the following): pushing of each particle is performed by updating the quantities related to the particle, making use of the previous-step computed values for that particle only (i.e., with no dependence on other-particle quantities), and of the replicated-field values interpolated at the particle position; all these quantities are locally available.

The *third phase* presents two strictly linked problems:

1. the quantities to be updated (the pressure-tensor components at the grid points) are replicated, and thus must be kept consistent among the processors; and
2. each quantity takes contribution from a number of particles that can reside on different nodes.

The solution relies on the associative and distributive properties of the updating laws for the pressure terms, with respect to the contributions given by every single particle: the computation for each update is split among the nodes into partial computations, involving the contribution of the local particles only, and the partial results are then reduced into global results, which are broadcasted to all the nodes.

In addition, all the quantities that cumulate with particles and time present these same properties; thus we can apply the same strategy.

A parallel flow chart, representing the above three phases, is shown in Fig. 1.

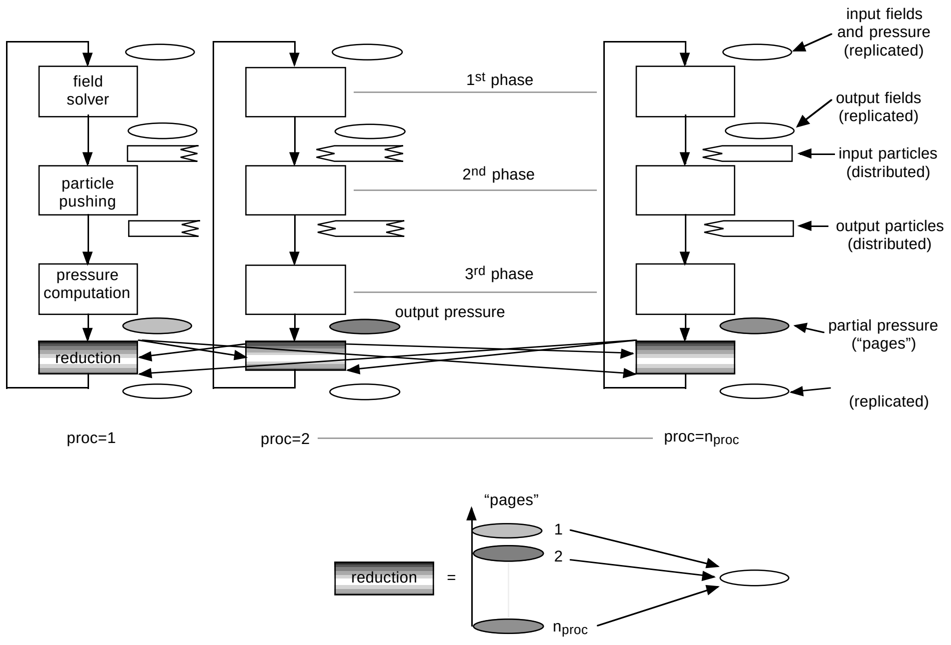

Fig. 1. Parallel flow chart representing the three phases of the work distribution strategy adopted within the particle decomposition approach. The work related to the first phase (the on-grid field updating) is replicated and does not need any communication. The particle pushing phase does not require any communication, apart from that required for quantities that cumulate (neglected in this figure); all the data needed to evolve particles coordinates are indeed locally available. In the third phase, pressure updating is split into partial computations involving only the contribution of local particles. Partial results are then reduced into global results, which are broadcasted to all the nodes.

4. Implementation of the parallelization strategy in HPF

The particle decomposition parallelization strategy, both for data and work distribution, could be implemented in the framework of several approaches to programming on distributed memory architectures. Such approaches include low-level languages and libraries, like the *message passing* environments (e.g., PVM [16] and MPI [26]), and high-level ones, like the HPF [20,21,31] language, PCN [13], Fortran M [14], P^3L [6] and SCL [8]. [1] The former are based on sets of primitives for collective data movement, used to rearrange data among processes, or for collective computation operations, like reductions. The use of these primitives allows for optimizing communication performance, by exploiting the knowledge of the hardware and interconnection structure of the target architecture. The low level of abstraction and decoupling from the characteristics of the target architectures makes the programming in the message passing paradigm more difficult, and affects the portability of the produced codes. On the opposite side, high-level environments adopt the approach to the programming of parallel architectures based on the automatic transformation of sequential code, possibly augmented with parallelization directives, into explicitly parallel code, by means of *parallelizing compilers*.

In this paper, we consider the implementation of the strategy depicted in the previous section in the HPF standard. [2] In particular, HPF directives for (cyclic) data distribution can be applied to all the data structures related to the particle quantities, and the HPF `INDEPENDENT` directive can be used to distribute the above-described second- and third-phase loop iterations over the particles.

The scheme to handle with the "inhibitors of parallelism" within the loops over the particles (the pressure-tensor components and the quantities that cumulate with particles and time), can be implemented in HPF by restructuring the code in the following way:

- The data structures [scalars and (multi-dimensional) arrays] that store the values of these quantities are replaced, within the bodies of the distributed loops, by

[1] Parallel environments have been developed also for shared memory architectures. For the low-level multithread programming model a standard library exists, designed by POSIX (an acronym for Portable Operating System Interface – a set of committees in the IEEE concerned with creating an API common to all Unix systems) and commonly referred as *Pthreads* [11], which is currently supported on most operating systems. No such a standard library exists for the high-level models, though a role of de facto standard is being played by the OpenMP [29] language, which is the counterpart of HPF for the shared memory architectural model: it augments C and Fortran with a set of directives to express task and data parallelism, synchronization and control of scope on access of data, and work balancing.

[2] With HPF, the programmer is only responsible for defining, by means of (high level) directives, how the data are to be distributed to the processors. The HPF compiler is responsible for producing code to distribute the array elements on the available processors. The message-passing code produced by the compiler, according to the Single Program Multiple Data (SPMD) [7] execution model instructs each processor to update the subset of the array elements which are stored in the local memory, possibly by getting values, corresponding to non-local accesses, owned by the other processors via communication primitives. HPF provides a standardized set of extensions to Fortran 90 (F90) and Fortran directives to enable the execution of a sequential program on (distributed memory) parallel computer systems.

corresponding data structures augmented by one dimension; their rank must be equal to or greater than the number of available processors.

- These data structures are distributed, along the added dimension, over the processors; each of the distributed "pages" will store the partial computations of the quantities, which include the contributions of the particles that are local to each processor.
- At each iteration of the loops over the particles, the contribution of the corresponding particle to an element of the quantities under consideration is added to the appropriate element of the distributed page.
- At the end of the iterations, the temporary data structures are reduced along the added and distributed dimension, and the results are assigned to the corresponding original data structures. This is implemented with the use of HPF intrinsic reduction functions such as `SUM`.

The only need for communication is related to these reductions and the subsequent broadcasts, and thus it is embedded in the execution of the intrinsic functions. If the underlying HPF compiler supports the implementation of highly optimized versions of the HPF intrinsic procedures for distributed parameters, these communications are performed as vectorized and collective minimum-cost communications.

A problem arose when implementing the depicted strategy. The computation for the selection of the distributed pages described above, even though not complex, is anyway a non-linear function of the loop index of the distributed loops. This, together with the presence of indirect references (through the elements of an array) to the elements of each page, represented a problem because the target HPF compiler, being able to perform data-dependence analysis for array elements, actually performed unrequested checks about the assertion of independence of the loop iterations, provided with the `INDEPENDENT` directive. In those cases the compiler did not distribute the loop iterations, because the non-linearity and the indirect character of the indexing expressions prevent any state-of-the-art dependence test from proving the actual independence of the loop iterations, and would make worst-case assumption of dependence.

This problem was anyway quite easily bypassed with the help of the HPF extrinsic procedures `HPF_LOCAL`. [3]

[3] HPF programs may call non-HPF subprograms as *extrinsic procedures* [21]. This allows the programmer to use non-Fortran language facilities, handle problems that are not efficiently addressed by HPF, hand-tune critical kernels or to call optimized libraries. An extrinsic procedure can be defined as explicit SPMD code by specifying the local procedure code that is to execute on each processor. HPF provides a mechanism for defining local procedures in a subset of HPF that excludes only data mapping directives, which are not relevant to local code. If a subprogram definition or interface uses the extrinsic-kind keyword `HPF_LOCAL`, then an HPF compiler should assume that the subprogram is coded as a local procedure. All distributed HPF arrays passed as arguments by the caller to the (global) extrinsic procedure interface, are logically divided into pieces; the local procedure executing on a particular physical processor sees an array containing just those elements of the global array that are mapped to that physical processor. A call to an extrinsic procedure results in a separate invocation of a local procedure on each processor. The execution of an extrinsic procedure consists of the concurrent execution of a local procedure on each executing processor. Each local procedure may terminate at any time by executing a RETURN statement. However, the extrinsic procedure as a whole terminates only after every local procedure has terminated.

We used the extrinsic mechanism to achieve the same effect as the `INDEPENDENT` directive, i.e., the distribution of the execution of loop iterations over the processors, when this latter could not be enabled due to the complex and/or indirect references to distributed arrays within the yet independent loop iterations. To this purpose, the loops over the particles were restructured to become the bodies of the extrinsic procedures.

As an example of the depicted strategy, we consider the following skeletonized F90 code excerpt, where each element of the three-dimensional array `pressure` is updated by the contribution of the particles falling in the neighbors of the corresponding grid point:

```
 pressure = 0.
 do l=1,n_part
   weight_l = weight(l)
   r_l      = r(l)
   theta_l  = theta(l)
   phi_l    = phi(l)
   j_r      = f_1(r_l)
   j_theta  = f_2(theta_l)
   j_phi    = f_3(phi_l)
   pressure(j_r,j_theta,j_phi) =
&       pressure(j_r,j_theta,j_phi) +
&       f_4(weight_l,r_l,theta_l,phi_l)
 enddo
```

Here `f_1`, `f_2`, `f_3` and `f_4` are suited non-linear functions of the particle weight and/or phase-space coordinates. This excerpt represents, very schematically, the update of the pressure-tensor components, by trilinear weighting of the contribution of each particle among the vertices of the cells. The computation of the array `pressure` is anyway representative of any computation of all the quantities that inhibit the parallel execution of the loops over the particles.

The code, restructured according to the above guidelines, looks like the following:

```
                    INTERFACE
                    EXTRINSIC(HPF_LOCAL)

    &subroutine extr_pressure(weight,r,theta,phi,
    &                           pressure_par)
     real*8, dimension(:), intent(in) :: weight,r,theta,phi
     real*8, dimension(:,:,:,:), intent(out) :: pressure_par
!HPF$ DISTRIBUTE (CYCLIC) :: weight,r,theta,phi
!HPF$ ALIGN WITH weight(:) :: pressure_par(*,*,*,:)
     end subroutine extr_pressure
     END INTERFACE
```

```
      ...
      real*8, dimension (0:n_r,n_theta,n_phi,
     &                   number_of_processors()):: pressure_par
      real*8, dimension (n_part) ::  weight,r,theta,phi
!HPF$ DISTRIBUTE (CYCLIC) :: weight,r,theta,phi
!HPF$ ALIGN WITH weight(:) :: pressure_par(*,*,*,:)
      nprocs = number_of_processors()
      ...
      call extr_pressure(weight,r,theta,phi,pressure_par)
      pressure(0:n_r,n_theta,n_phi) =
     &      SUM(pressure_par(0:n_r,n_theta,n_phi,:),dim=4)
      ...

      EXTRINSIC(HPF_LOCAL)
     &subroutine extr_pressure(weight,r,theta,phi,
     &                         pressure_par)
      real*8, dimension(:), intent(in) :: weight,r,theta,phi
      real*8, dimension(:,:,:,:), intent(out) :: pressure_par

      pressure_par = 0.
      do l=1, UBOUND(weight,dim=1)
        weight_l = weight(l)
        r_l      = r(l)
        theta_l  = theta(l)
        phi_l    = phi(l)
        j_r      = f_1(r_l)
        j_theta  = f_2(theta_l)
        j_phi    = f_3(phi_l)
        pressure_par(j_r,j_theta,j_phi,1)=
     &          pressure_par(j_r,j_theta,j_phi,1) +
     &          f_4(weight_l,r_l,theta_l,phi_l)
      enddo
      end subroutine extr_pressure
```

Here each local procedure executing on a given processor sees the portion of the arrays related to the particles (`weight`, `r`, `theta` and `phi`), and the page of the "partial results" array `pressure_par` assigned to that processor. It executes only the set of loop iterations that access the particles local to the processor [`L=1,UBOUND(weight, dim=1)`], and updates the page of `pressure_par` [`pressure_par(j_r, j_theta, j_phi,1)`] assigned to it. At the end of the execution of the local extrinsic procedure, all the partial updates of the components of `pressure_par` are collected in the global-HPF-index-space `pressure_par`. Then the reduction of the (distributed) pages of `pressure_par` in `pressure` (replicated) is very easily performed with the use of the HPF intrinsic function `SUM`.

5. Results

In this section, we present the results obtained from the described HPF parallel version of HMGC making use of the extrinsic mechanism. The code was compiled with the IBM *xlhpf* compiler [17] – an optimized native compiler for IBM SP systems – under the -*O3* and -*qhot* options, and tested on an IBM Scalable Power system (SP) at the ENEA Research Centre of Frascati, Rome.

The results presented here refer to executions on a spatial grid with $N_r = 64$ points in the radial direction, $N_\vartheta = 32$ points in the poloidal direction and $N_\varphi = 16$ points in the toroidal one; we performed several executions by varying both the number of processors n_{proc} (ranging from $n_{\text{proc}} = 2$ to $n_{\text{proc}} = 14$) and the average number of particle per cell $N_{\text{ppc}} \equiv N_{\text{part}}/N_{\text{grid}}$ (ranging from $N_{\text{ppc}} = 1$ to $N_{\text{ppc}} = 256$).

The cpu-times per time step are reported in Table 1, where the results relative to the sequential code, obtained by compiling the source code with the *xlhpf* compiler under the -*qnohpf* option, are also reported. The use of the -*qstrict* option does not give any appreciable difference.

These results are reported in graphical form in Figs. 2–4, under different views. Fig. 2 shows the scaling of the speed-up with respect to the number of processors for different values of the average number of particles per cell. The curve corresponding to efficiency (the ratio between the speed-up and number of processors) equal to one is also reported for comparison. The speed-up scales well with the number of processors, for all the considered cases. Both the scalability and the absolute value of the speed-up improve with increasing N_{ppc}. Note that for the largest simulations, superlinear results are obtained, which can be possibly traced back to memory and/or cache effects and compiler options.

These effects are better shown in Fig. 3, which displays the scaling of the efficiency with N_{ppc}, for different values of the number of processors. We can observe that the efficiency tends to reach its ideal value at larger values of N_{ppc} the greater the number of processors is. This feature is due to the fact that, increasing the number of processors, the average number of particles per cell assigned to each processor decreases, and the importance of the grid-related calculation tends to overcome the particle-related one. In fact, the scattered results shown in Fig. 3 approximatively fall on a "universal" curve, when plotted against $N_{\text{ppc}}/n_{\text{proc}}$, as shown in Fig. 4. The efficiency falls below its ideal value for $N_{\text{ppc}}/n_{\text{proc}}$ lower than a certain threshold value, α_{eff}, which shows a very weak dependence on the specific value of n_{proc} considered.

Table 1
Cpu-times per time step

N_{ppc}	Cpu-time (s)				
	Sequential	$n_{\text{proc}} = 2$	$n_{\text{proc}} = 4$	$n_{\text{proc}} = 8$	$n_{\text{proc}} = 14$
1	1.52	1.35	1.09	0.96	0.93
4	5.32	3.06	1.93	1.41	1.15
16	23.01	11.40	5.63	3.22	2.05
64	90.40	42.66	22.14	11.75	5.61
256	319.70	166.96	85.02	43.75	19.81

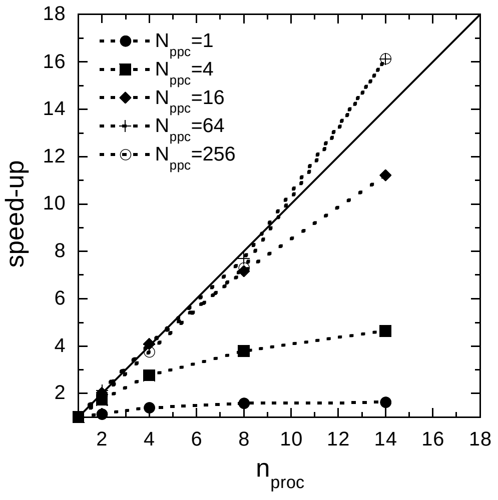

Fig. 2. Scaling of the speed-up with respect to the number of processors n_{proc} for different values of the average number of particles per cell N_{ppc}. The curve corresponding to efficiency equal to one (solid line) is also reported for comparison.

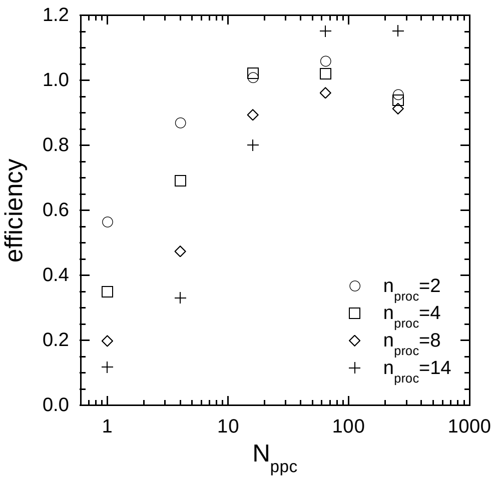

Fig. 3. Scaling of the efficiency with the average number of particles per cell N_{ppc}, at different values of the number of processors n_{proc}.

6. Validity of the particle-decomposition approach

Because of the particular choice in distributing the arrays on the different processors, there is an intrinsic limit in the applicability of the particle-decomposition approach: if M_0 is the size of the RAM resources of the single computational node, the condition

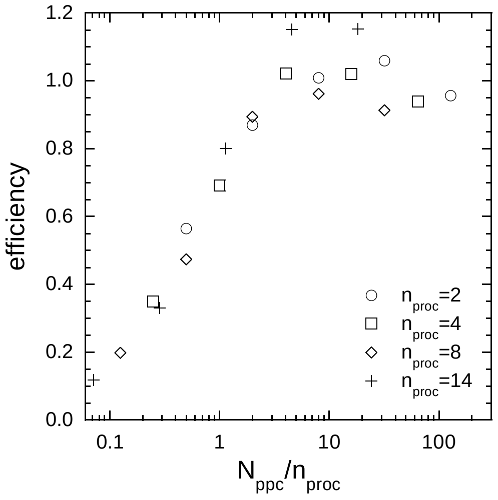

Fig. 4. Scaling of the efficiency with the average number of particles per cell per processor $N_{\mathrm{ppc}}/n_{\mathrm{proc}}$.

$$N_{\mathrm{grid}}m_{\mathrm{grid}} + N_{\mathrm{part}}m_{\mathrm{part}} \leqslant M_0 \tag{14}$$

has to be satisfied in order to avoid *memory paging* during the simulation. Here m_{grid} and m_{part} are the amounts of memory needed to store the field arrays on each grid point and, respectively, the coordinates and the weight for each particle. This condition puts an upper limit, α_{M_0} on the achievable value of $N_{\mathrm{ppc}}/n_{\mathrm{proc}}$:

$$\alpha_{M_0} = \frac{1}{m_{\mathrm{part}}}\left(\frac{M_0}{N_{\mathrm{grid}}} - m_{\mathrm{grid}}\right). \tag{15}$$

The necessary condition for obtaining efficient particle-decomposition parallel simulation is then given by $\alpha_{\mathrm{eff}} < \alpha_{M_0}$. Under such a condition, which depends on the spatial resolution required by the particular problem considered (i.e., on the number of grid points N_{grid}), a range of $N_{\mathrm{ppc}}/n_{\mathrm{proc}}$ values exists, in which efficient parallel simulations can be obtained. The specific choice for N_{ppc} and n_{proc} can be done on the basis of the following considerations.

Fig. 5 reports the findings of HMGC simulations corresponding to different values of N_{ppc}. In particular, the linear growth rate γ of an energetic particle mode [5], normalized to the inverse characteristic time of Alfvén-mode propagation, is shown. Such a quantity corresponds to the rate at which a small electromagnetic perturbation to a given equilibrium grows in time. In the present case, an equilibrium characterized by a ratio between the kinetic pressure of a destabilizing energetic population and the equilibrium magnetic pressure equal to 0.03 has been considered. It can be observed that, by increasing N_{ppc} (and thus the resolution in the velocity space), the growth rate converges to a limiting (true) value. Defining conventionally the accuracy of the simulation as the relative deviation from this value, the requirement of a given accuracy fixes the minimum needed resolution, i.e., the

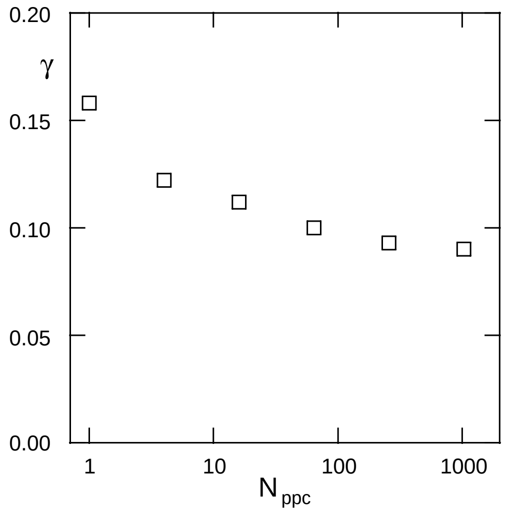

Fig. 5. Linear growth rate of an energetic particle mode normalized to the inverse characteristic time of Alfvén-mode propagation versus the average number of particles per cell $N_{\rm ppc}$.

minimum $N_{\rm ppc}$ value, $N_{\rm ppc}^{\rm acc}$. Note that the determination of such a value may depend strongly on the different physical scenarios considered.

Once fixed $N_{\rm ppc} \approx N_{\rm ppc}^{\rm acc}$, minimum and maximum values of $n_{\rm proc}$ are defined, corresponding, respectively, to the memory and the high-efficiency constraint:

$$n_{\rm proc}^{\rm min} \equiv \min\left[\frac{N_{\rm ppc}^{\rm acc}}{\alpha_{M_0}}, 1\right]$$

and

$$n_{\rm proc}^{\rm max} \equiv {\rm int}\left[\frac{N_{\rm ppc}^{\rm acc}}{\alpha_{\rm eff}}\right],$$

with int[x] being the integer part of the real number x. The choice $n_{\rm proc} = n_{\rm proc}^{\rm min}$ corresponds to minimizing the hardware-resource allocation; the opposite choice, $n_{\rm proc} = n_{\rm proc}^{\rm max}$ corresponds to minimizing the simulation time.

In the particular case presented in this paper, an accuracy level of 5% in the determination of the growth rate of the Alfvén mode corresponds to $N_{\rm ppc}^{\rm acc} \approx 120$. Moreover, from Fig. 4, we see that $\alpha_{\rm eff} \approx 3.5$. Finally, the memory-related parameters are given by $N_{\rm grid} = 32{,}768$, $M_0 = 512$, $m_{\rm grid} = 9.03 \times 10^{-5}$, $m_{\rm part} = 5.62 \times 10^{-5}$, with M_0, $m_{\rm grid}$ and $m_{\rm part}$ expressed in megabytes. This implies that $\alpha_{M_0} \approx 276.4$ (the condition for the existence of a high-efficiency range of $N_{\rm ppc}/n_{\rm proc}$ is largely satisfied). From such values we obtain $n_{\rm proc}^{\rm min} = 1$ and $n_{\rm proc}^{\rm max} = 34$: for $1 < n_{\rm proc} < 34$, it is then worth adopting the particle-decomposition parallel approach in simulating the considered case.

7. Conclusions

In the present paper, we have described a paralellization strategy for PIC codes, applied to the HMGC, developed to study the Alfvénic turbulence in magnetically confined plasmas. Such a strategy, which relies on the particle decomposition, instead of the more usual domain decomposition, has been implemented within the HPF framework, and seems particularly attractive if targeted towards moderately parallel architectures. Its main advantages are represented by a perfect load balancing among processors, a modest amount of inter-processor communications and a moderate effort in porting the code in the parallel form.

A very high efficiency can be expected in the limit of high average number of particles per cell ($N_{\rm ppc}$) per processor. In fact, we get efficiency close to unity for $N_{\rm ppc}/n_{\rm proc} \gtrsim 3.5$. Thus, once the accuracy level needed for the particular physics problem investigated (and, hence, the average number of particles per cell required) is fixed, an upper bound, $n_{\rm proc}^{\rm max}$, on the number of processors to be used in the simulation is put by the requirement of high efficiency. A lower bound, $n_{\rm proc}^{\rm min}$, on the same number is posed by the necessity of avoiding *memory paging*. The particle-decomposition approach to parallel PIC plasma simulation is suited and efficient if and only if the condition $n_{\rm proc}^{\rm min} < n_{\rm proc} < n_{\rm proc}^{\rm max}$ is satisfied.

References

[1] E. Akarsu, K. Dincer, T. Haupt, G.C. Fox, Particle-in-cell simulation codes in high performance Fortran, in: Proceedings of SuperComputing '96 (IEEE, 1996). http://www.supercomp.org/sc96/proceedings/SC96PROC/AKARSU/INDEX.HTM.

[2] C.K. Birdsall, A.B. Langdon, Plasma Physics via Computer Simulation, McGraw-Hill, New York, 1985.

[3] A. Bondeson, Simulations of tokamak disruptions including self-consistent temperature evolution, Nucl. Fusion 26 (1986) 929–940.

[4] S. Briguglio, G. Vlad, F. Zonca, C. Kar, Hybrid magnetohydrodynamic-gyrokinetic simulation of toroidal Alfvén modes, Phys. Plasmas 2 (1995) 3711–3723.

[5] L. Chen, Theory of magnetohydrodynamic instabilities excited by energetic particles in tokamaks, Phys. Plasmas 1 (1994) 1519–1522.

[6] M. Danelutto, R. Di Meglio, S. Orlando, S. Pelagatti, M. Vanneschi, The P^3L Language: an introduction, Technical Report HPL-PSC-91-29, Hewlett–Packard Laboratories, Pisa Science Centre, December 1991.

[7] F. Darema, D.A. George, V.A. Norton, G.F. Pfister, A single program multiple data computational model for EPEX/Fortran, Parallel Comput. 7 (1) (1988) 11–24.

[8] J. Darlington, A.J. Field, P.G. Harrison, P.H.J. Kelly, D.W.N. Sharp, Q. Wu, R.L. Whie, Parallel programming using skeleton functions, in: PARLE'93, Lecture Notes in Computer Science, vol. 694, Springer, Berlin, 1993, pp. 146–160.

[9] V.K. Decyk, Skeleton PIC codes for parallel computers, Comput. Phys. Comm. 87 (1995) 87–94.

[10] A.M. Dimits, L.L. Lo Destro, D.H.E. Dubin, Gyroaveraged equations for both the gyrokinetic and drift-kinetic regimes, Phys. Fluids B 4 (1992) 274–277.

[11] Extension for threads (1003.1c-1995), in: Information Technology – Portable Operating System Interface (POSIX) – Part 1: System Application: Program Interface (API), IEEE/ANSI Std 1003.1, 1996 Edition (ISBN 1-55937-573-6).

[12] R.D. Ferraro, P. Liewer, V.K. Decyk, Dynamic load balancing for a 2D concurrent plasma PIC code, J. Comput. Phys. 109 (1993) 329–341.
[13] I. Foster, R. Olson, S. Tuecke, Productive parallel programming: the PCN approach, Sci. Programming 1 (1) (1992) 51–66.
[14] I. Foster, Compositional parallel programming languages, ACM Trans. Prog. Lang. Syst. 18 (4) (1996) 454–476.
[15] G.C. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, D. Walker, Solving Problems on Concurrent Processors, Prentice-Hall, Englewood Cliffs, NJ, 1988.
[16] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderam, PVM: Parallel Virtual Machine – A Users Guide and Tutorial for Network, MIT Press, Cambridge, MA, 1994.
[17] M. Gupta, S. Midkiff, E. Schonberg, V. Seshadri, D. Shields, K.Y. Wang, W.M. Ching, T. Ngo, A HPF Compiler for the IBM SP2, in: Proceedings of SuperComputing '95, ACM, 1995.
[18] T.S. Hahm, Nonlinear gyrokinetic equations for tokamak microturbulence, Phys. Fluids 31 (1988) 2670–2673.
[19] T.S. Hahm, W.W. Lee, A. Brizard, Nonlinear gyrokinetic theory for finite-beta plasmas, Phys. Fluids B 31 (1988) 1940–1948.
[20] High Performance Fortran Forum, High Performance Fortran Language Specification, Sci. Programming 2(1&2) (1993) 1–170.
[21] High Performance Fortran Forum, High Performance Fortran Language Specification, Version 2.0, Rice University, 1997.
[22] R. Izzo, D.A. Monticello, W. Park, J. Manickam, H.R. Strauss, R. Grimm, K. McGuire, Effects of toroidicity on resistive tearing modes, Phys. Fluids 26 (1983) 2240–2246.
[23] W.W. Lee, Gyrokinetic approach in particle simulations, Phys. Fluids 26 (1983) 556–562.
[24] W.W. Lee, Gyrokinetic particle simulation model, J. Comput. Phys. 72 (1987) 243–269.
[25] P.C. Liewer, V.K. Decyk, A general concurrent algorithm for plasma particle-in-cell codes, J. Comput. Phys. 85 (1989) 302–322.
[26] Message Passing Interface Forum, Document for a Standard Message-Passing Interface, Technical Report No. CS-93-214, University of Tennessee, 1994.
[27] K. Miyamoto, Plasma Physics for Nuclear Fusion, MIT Press, Cambridge, MA, 1976.
[28] C.D. Norton, B.K. Szymanski, V.K. Decyk, Object oriented parallel computation for plasma simulation, Comm. ACM 38 (10) (1995) 88–100.
[29] OpenMP Architecture Review Board, OpenMP Fortran Application Program Interface, Version 1.0, October 1997.
[30] W. Park, S. Parker, H. Biglari, M. Chance, L. Chen, C.Z. Cheng, T.S. Hahm, W.W. Lee, R. Kulsrud, D. Monticello, L. Sugiyama, R. White, Three-dimensional hybrid gyrokinetic-magnetohydrodynamics simulation, Phys. Fluids B 4 (1992) 2033–2037.
[31] H. Richardson, High Performance Fortran: history, overview and current developments, Technical Report TMC-261, Thinking Machines Corporation, 1996.
[32] H.R. Strauss, Nonlinear, three-dimensional magnetohydrodynamics of noncircular tokamaks, Phys. Fluids 20 (1977) 134–140.
[33] G. Vlad, S. Briguglio, C. Kar, F. Zonca, F. Romanelli, Linear and nonlinear stability of toroidal Alfvén eigenmodes, in: E. Sindoni, J. Vaclavik (Eds.), Proceedings of the Joint Varenna-Lausanne International Workshop '92 (Editrice Compositori Società Italiana di Fisica, Bologna, 1992) on Theory of Fusion Plasmas, pp. 361–382.
[34] G. Vlad, C. Kar, F. Zonca, F. Romanelli, Linear and nonlinear dynamics of Alfvén eigenmodes in tokamaks, Phys. Plasmas 2 (1995) 418–441.