ELSEVIER

Future Generation Computer Systems 16 (2000) 541–552

FGCS
FUTURE GENERATION COMPUTER SYSTEMS

Parallelization of plasma simulation codes: gridless finite size particle versus particle in cell approach

S. Briguglio [a,*], G. Vlad [a], B. Di Martino [b], G. Fogaccia [a]

[a] *Associazione Euratom-ENEA sulla Fusione, C.R. Frascati, C.P. 65-I-00044-Frascati, Rome, Italy*
[b] *Dipartimento di Ingegneria dell'Informazione, Second University of Naples, Naples, Italy*

Accepted 15 October 1999

Abstract

The main features of gridless finite-size-particle codes are examined and compared with those of particle-in-cell ones, from the point of view of the performances that can be obtained, with respect to both the spatial-resolution level and the efficiency of parallel particle simulations. The particle-decomposition parallelization strategy is discussed, and its implementation in the framework of the High Performance Fortran paradigm is presented. It is shown that such codes are particularly suited for particle-decomposition parallelization on distributed-memory architectures, as they present a strong reduction, in comparison with particle-in-cell codes, of the memory and computational offsets related to storing and updating the replicated fluctuating-field arrays. ©2000 Elsevier Science B.V. All rights reserved.

Keywords: Plasma; Particle simulation; Particle-in-cell; Finite-size-particle; Parallelization; Distributed memory; HPF

1. Introduction

Particle simulation codes [1] seem to be the most suited tool for the investigation of turbulent plasma behaviour. Particle simulation indeed consists in evolving, according to the equations of motion, the phase-space coordinates of a particle population in the fluctuating electromagnetic fields selfconsistently computed, at each time step, in terms of the contribution yielded by the particles themselves (e.g., pressure perturbation), and allows one to fully retain all the relevant kinetic effects.

Aim of particle simulation is, of course, that of obtaining from the numerical plasma the same behaviour of the physical one. The main problem in reaching such a target is represented by the impossibility, with today numerical resources, of simulating a number of particles comparable with those typical of the real plasmas. This problem can be partially bypassed by considering each simulation particle as a cluster ("macroparticle") of very many physical particles; mutual interactions between macroparticles are retained, while those between particles inside the same cluster are neglected. By identifying the charge and the mass of each simulation particle with those of the whole cluster (but still imposing that such a simulation particle moves as a physical one), all the relevant parameters (Debye length, Larmor radius, etc.) of the simulation plasma coincide with the corresponding parameters of the physical plasma, notwithstanding

* Corresponding author.
E-mail addresses: briguglio@frascati.enea.it (S. Briguglio), vlad@frascati.enea.it (G. Vlad), dimartin@grid.unina.it (B. Di Martino), fogaccia@frascati.enea.it (G. Fogaccia).

PII: S0167-739X(99)00125-9

that the simulation-particle density is much lower than the physical-particle one [2].

Further problems, however, arise, related to the fact that physical plasmas are characterized by the *plasma condition*, i.e., by the presence of a large number of particles in a Debye sphere ($n_0\lambda_D^3 \gg 1$, with n_0 being the particle density and λ_D the Debye length). It is apparent that, once the equality between simulation and physical scales has been ensured, it holds in particular for λ_D, and the plasma condition is no longer satisfied by the simulation plasma, because of its very low density. This implies that the set of numerical particles is, by itself, too collisional to be considered a plasma: short range interactions between particles dominate over the long range ones, and physical behaviours are not well reproduced.

To overcome such a difficulty, at least two methods have been devised: the *particle-in-cell* (PIC) and the *finite-size-particle* (FSP) ones. The former [3] consists in computing the electromagnetic fields only at the points of a discrete spatial grid, then interpolating them at the (continuous) particle positions in order to perform particle pushing. Short-range interactions are then cut off for mutual distances shorter than the typical spacing — L_c — between grid points, whilst the relevant long-range interactions are not significantly affected. It is immediate to show that the PIC particle ensemble behaves as a plasma under the much more relaxed condition $n_0 L_c^3 \gg 1$ (with $L_c \gg \lambda_D$).

The latter method [4] consists in smoothing the singular particle contribution to the density, $\delta(\mathbf{x} - \mathbf{x}_l)$ (with $\mathbf{x}_l$ being the lth particle position), by replacing the δ function by a shaped regular (e.g., Gaussian) one, $S(\mathbf{x} - \mathbf{x}_l)$. This introduces a smoothing of the short-range interactions analogous to the PIC one, and if L_s is the width of the function S, the plasma condition is replaced by $n_0 L_s^3 \gg 1$. Such a width can be interpreted as the typical size of the numerical particles (or, more appropriately, *charge clouds*). The two methods are expected to yield the same qualitative findings if $L_c \approx L_s$, but they can result in very different computational requirements.

In this paper we compare the PIC and the FSP approach to particle simulation, with particular regard to the parallelization on distributed-memory architectures of the corresponding numerical codes. The paper is organized as follows. In Section 2, the problem of the computational loads associated to the two methods is considered, and the main strategies developed for the parallelization of the corresponding codes are qualitatively discussed. The main features of the gridless FSP version of the hybrid MHD-gyrokinetic particle simulation code [5] (HMGC) are reported in Section 3. Section 4 describes the implementation of the high performance Fortran (HPF) parallelization of the FSP code. A comparison between parallelization performances obtained by PIC and gridless FSP versions of HMGC is reported in Section 5, and a brief summary of the main ideas discussed in the paper is presented in Section 6.

2. PIC and FSP computational loads

Let us assume, for the sake of simplicity, that both the PIC and the FSP method resort to the Fourier transform in order to solve the equations for the electromagnetic fields. We can approximately estimate the number of operations that must be performed, for each time step, in a PIC simulation as

$$O^{\mathrm{PIC}} \approx f(N_{\mathrm{harm}}) + n_{\mathrm{FT}} \times N_{\mathrm{harm}} \times N_{\mathrm{cell}} + n_{\mathrm{int}} \times N_{\mathrm{part}}, \quad (1)$$

where N_{harm} is the number of Fourier harmonics retained in the simulation; $f(N_{\mathrm{harm}})$ an increasing function of N_{harm} whose precise dependence is determined by the structure of the problem and by the algorithm used by the field solver; n_{FT} the number of operations needed to compute each addendum in the sum required by the Fourier transform (and anti-transform);[1] N_{cell} is the number of cells of the spatial grid; n_{int} the number of operations required for the field interpolation from the grid to the particle position (and for the pressure assignment from each particle to the grid) and N_{part} is the number of simulation particles. Analogously, the memory required by the simulation can be written as

$$M^{\mathrm{PIC}} \approx m_{\mathrm{harm}} \times N_{\mathrm{harm}} + m_{\mathrm{cell}} \times N_{\mathrm{cell}} + m_{\mathrm{part}} \times N_{\mathrm{part}}, \quad (2)$$

[1] We refer to general situations in which the fast Fourier transform (FFT) algorithm is not recommended (e.g., because the Fourier space is not densely filled). Our conclusions, however, would maintain their qualitative validity even in the FFT framework.

where m_{harm}, m_{cell} and m_{part} are the amounts of memory needed to store, respectively, a single harmonic of the complete set of Fourier-space fields, the real-space fields at each grid point and the phase-space coordinates for each particle.

In the FSP case, only two terms appear, due to the fact that the fields are transformed from the Fourier space to the real one (and viceversa), directly at the particle positions

$$O^{\text{FSP}} \approx f\,(N_{\text{harm}}) + n_{\text{FT}} \times N_{\text{harm}} \times N_{\text{part}}, \tag{3}$$

$$M^{\text{FSP}} \approx m_{\text{harm}} \times N_{\text{harm}} + m_{\text{part}} \times N_{\text{part}}. \tag{4}$$

Taking into account that, both in Eqs. (1) and (3), the quantity $f\,(N_{\text{harm}})$ is typically negligible in comparison with the terms proportional to N_{part}, and that, for PIC codes, $N_{\text{ppc}} \equiv N_{\text{part}}/N_{\text{cell}} \approx n_0 L_{\text{c}}^3 \gg 1$, it can be easily seen that, as far as $n_{\text{FT}} \times N_{\text{harm}} \gg n_{\text{int}}$ and the calculation has to be performed by a single processor (as it was the case before the development of parallel computers), the gridless FSP method is more expensive than the PIC one, without presenting any significant advantage in terms of memory requests. This motivates the large diffusion of PIC codes and the poor consideration in which gridless FSP simulation was taken in the past.

Two distinct reasons could, however, justify a different trend: the first one is represented by the interest in simplified simulations in which only very few modes are evolved. This interest is motivated, first of all, by the relevance of linear simulations, which allow one to predict whether the plasma is stable or unstable with respect to particular modes and hence may be focused on the evolution of the most unstable modes alone; furthermore, it is related to the fact that the usually weak non-linear coupling between different modes restricts even the non-linear mode spectrum to a limited number of significant harmonics. It is then apparent that, in such a *few-harmonic* framework, the condition $n_{\text{FT}} \times N_{\text{harm}} \gg n_{\text{int}}$ can be violated or, at least, significantly weakened, because the set of modes over which the sums must be performed when transforming back and forth from the real space to the Fourier one is drastically reduced.

The second reason has to do with the utilization of parallel computers. The need for high phase-space resolution and consequently for large number of particles has indeed motivated a large effort, in the last years, in porting particle codes on such computers. In this paper, we restrict our analysis to the case of distributed-memory architectures.

Let us start considering standard PIC codes. Two main different techniques have been developed in parallelizing such codes. The first approach is based on the *domain decomposition* [6]: different portions of the physical domain are assigned to different processors, together with the particles that reside on them. In this way, both the number of operations per time step and the memory space required to each computational node are reduced (in an average sense) by a factor approximately equal to the number of processors, n_{proc}. Neglecting corrections due to inter-processor communication, these two quantities are indeed given by the following expressions:

$$O^{\text{PIC}}_{\text{d.d.}} \approx f\,(N_{\text{harm}}) + \frac{1}{n_{\text{proc}}} \left(n_{\text{FT}} \times N_{\text{harm}} \times N_{\text{cell}} + n_{\text{int}} \times N_{\text{part}}\right), \tag{5}$$

$$M^{\text{PIC}}_{\text{d.d.}} \approx m_{\text{harm}} \times N_{\text{harm}} + \frac{1}{n_{\text{proc}}} \left(m_{\text{cell}} \times N_{\text{cell}} + m_{\text{part}} \times N_{\text{part}}\right). \tag{6}$$

The main advantage of the scheme is apparent from Eq. (6) and consists in the almost linear scaling of the attainable physical-space resolution (more precisely, the maximum number of spatial cells) with the number of processors. Inter-processor communication are, however, necessary to transfer particle data from one computational node to another when some particle moves from one portion of the grid to a different one; this particle migration can result in severe load-balancing problems and the above expressions, which state that such method is very efficient, can be, in practice, invalidated. In order to avoid such problems, a dynamical redistribution of grid and particle quantities is required, which makes the parallel implementation of a PIC code very complicate.

An alternative approach to the parallelization of PIC codes is based on *particle decomposition* [7]. It consists in statically distributing the particle population among processors, while replicating the data relative to grid quantities. Before updating the electromagnetic fields, at each time step, partial contributions to particle pressure coming from different portions of the

population must be summed together. It is apparent that load balancing is automatically enforced, because no particle has to be transferred from one processor to another. As a consequence, the implementation of parallelization is, in principle, almost straightforward. On the opposite side, grid calculations do not take advantage, with regard both to the number of operations and the memory requests, from such a parallelization, because each processor has to handle the whole spatial domain. Neglecting the corrections due to the need for communicating partial-pressure data, the computational load on each node is indeed given, in this case, by

$$O_{\text{p.d.}}^{\text{PIC}} \approx f\,(N_{\text{harm}}) + n_{\text{FT}} \times N_{\text{harm}} \times N_{\text{cell}} + n_{\text{int}} \times \frac{N_{\text{part}}}{n_{\text{proc}}}, \tag{7}$$

$$M_{\text{p.d.}}^{\text{PIC}} \approx m_{\text{harm}} \times N_{\text{harm}} + m_{\text{cell}} \times N_{\text{cell}} + m_{\text{part}} \times \frac{N_{\text{part}}}{n_{\text{proc}}}. \tag{8}$$

Particle decomposition comes out to be very efficient as far as the computational load related to the particle population dominates, for each processor, the one related to the grid. This condition corresponds to require a large average number of particles per cell on each processor, and it is surely satisfied for moderately parallel machines. However, for $n_{\text{proc}} \geq N_{\text{ppc}}$, it breaks down, and the computational effort of each processor is mainly devoted to the replicated calculation of grid quantities. Moreover, even neglecting efficiency problems, the implementation of the code on a very large number of processors does not allow one to reach high spatial-resolution levels, due to the offset in the memory requests represented by the grid data: the highest spatial resolution that can be reached in a simulation without requiring *memory paging* depends on the Random Access Memory resources of the single computational node, whereas increasing the number of processors only allows to increase the average number of particles per cell and hence the velocity-space resolution.

The existence, in the framework of the more appealing particle-decomposition approach, of bottle-necks in efficiency and performance associated to grid quantities induces one to bypass the introduction of a spatial grid, so resorting to the gridless FSP simulation. A particle-decomposition approach to the parallelization of a gridless FSP code would indeed bring to the following computational loads on each node:

$$O_{\text{p.d.}}^{\text{FSP}} \approx f\,(N_{\text{harm}}) + n_{\text{FT}} \times N_{\text{harm}} \times \frac{N_{\text{part}}}{n_{\text{proc}}}, \tag{9}$$

$$M_{\text{p.d.}}^{\text{FSP}} \approx m_{\text{harm}} \times N_{\text{harm}} + m_{\text{part}} \times \frac{N_{\text{part}}}{n_{\text{proc}}}. \tag{10}$$

The superiority of the FSP method in a parallel- simulation framework is apparent for two reasons. The first one is that, though FSP simulations require back and forth Fourier transforms being executed for each particle, these calculations are distributed among the processors, unlike the replicated PIC grid calculations.

The second reason corresponds to the fact that, unlike the particle decomposition PIC case, massively parallel simulations can yield significant benefits as far as the number of modes, N_{harm}, retained in the simulation is relatively small, in spite of the high mode numbers considered (high spatial resolution). The *few-harmonic* condition, indeed, strongly reduces the replicated terms in the right-hand side of Eqs. (9) and (10), and preserves both large parallelization efficiency and the ideal scaling of the maximum allowed spatial resolution with n_{proc} even for a very large number of processors. At the same time, of course, the positive feature of an automatic load balancing, typical of the particle-decomposition parallelization, is preserved.

3. The FSP hybrid MHD-gyrokinetic code

In this section, we present the main features of a parallel gridless FSP version of HMGC, a code for the investigation of Alfvénic turbulence in magnetically confined plasmas, previously implemented in a PIC version [5]. The code solves the coupled sets of MHD equations for the fluctuating electromagnetic fields and gyrokinetic equations of motion for collision-free energetic ions. The physical domain is represented by a three-dimensional toroidal grid, for which quasi-cylindrical coordinates are adopted (see Fig. 1): the minor-radius coordinate, r, and the poloidal and toroidal angles, ϑ and φ, respectively. The field solver uses finite differences in the minor

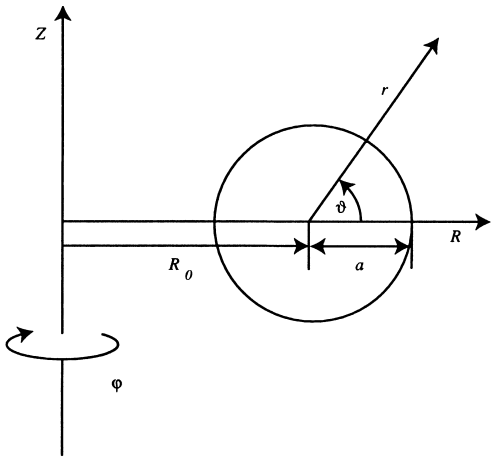


Fig. 1. Toroidal coordinate system (r, ϑ, φ) for a magnetically confined plasma with major radius R_0 and minor radius a.

radius direction and Fourier expansion in the poloidal and toroidal directions. Several quantities (both scalar and vectorial) — e.g., the number of particles that hit the internal wall of the toroidal chamber at $r = a$, with a being the minor radius of the torus, and are considered lost — are also calculated, cumulating particle contributions in time, while performing the main computation.

The FSP version of the code eliminates the grid in the ϑ and φ directions, and introduces finite sizes for particles along the same directions. The corresponding Fourier variables are $k_\vartheta \equiv m/r$ and $k_\varphi \equiv n/R$, where m and n are, respectively, the poloidal and the toroidal number, and R is the major-radius coordinate of the torus. The radial grid is instead maintained (so that this FSP code is not a "gridless" one, in an absolute sense), because of the impossibility of adopting Fourier expansion in that direction.

The Alfvén modes, which constitute the main object of the investigation performed by HMGC, are typically characterized [8], for a given value of the toroidal number n, by the coupling of several poloidal harmonics, with sharper radial structure the higher the number n is, and poloidal numbers such that $nq(0) - 1/2 \leq m \leq nq(a) + 1/2$ (here $q(r) \equiv r B_\varphi / R_0 B_\vartheta$ is the so-called *safety factor*, R_0 is the major radius of the torus, B_φ the toroidal magnetic field component and B_ϑ is the poloidal one). As a consequence, both poloidal and radial resolution requests (linearly) increase with increasing n, as it happens — of course — for toroidal resolution. In principle, memory resources of each processor could still be saturated by the offset associated to the first term in Eq. (10). Anyway, such an offset is in fact negligible, because its mode-number dependence, for a number of retained harmonics proportional to n, is quadratic (both m_{harm}, which takes into account the radial dependence of each mode, and N_{harm} scale linearly with n). In the PIC case, instead, the offset is dominated by the term $m_{\text{cell}} \times N_{\text{cell}}$, which scales as n^3. More precisely, with the level of precision adopted in our simulations to describe the spatial structure of the modes, the memory (in Megabytes) needed to store the field arrays for the FSP and the PIC version of HMGC, respectively, scales with the toroidal number n according to the following expressions:

$$M_{\text{field}}^{\text{FSP}} \approx 0.023 n^2, \tag{11}$$

$$M_{\text{field}}^{\text{PIC}} \approx 0.37 n^3. \tag{12}$$

Here we consider linear simulations with modes characterized by a single n, and with $q(0) = 1.1$ and $q(a) = 1.9$. It can be appreciated that the PIC approach is in fact penalized both by the n dependence and the proportionality coefficient. As such arrays are replicated on the different processors, the saturation of memory space (and the need for *memory paging*) is reached, in the PIC case, at lower values of n and thus at lower space resolution. The whole memory needed to store the distributed arrays associated to particle quantities — corresponding to the last terms in Eqs. (8) and (10) — scales instead, for both versions of HMGC, according to the following expression:

$$M_{\text{part}} \approx 0.23 N_{\text{ppc}} \times n^3, \tag{13}$$

with the cubic dependence on n being associated to the "number of cells", $N_{\text{part}}/N_{\text{ppc}}$. In the FSP case, N_{cell} and hence N_{ppc} have, by themselves, no meaning. However, we can interpret N_{cell} as the number of cells that would be necessary to accurately describe, in the PIC framework, the same modes we investigate by the gridless code. Remembering that the condition for obtaining the same accuracy by the two methods is given by $L_{\text{s}} \approx L_{\text{c}}$, and that in PIC simulations $N_{\text{cell}} \propto L_{\text{c}}^{-3}$, we can estimate, for FSP simulations, $N_{\text{cell}} \propto L_{\text{s}}^{-3}$. We can then still formally refer to N_{ppc} as to an average number of particles "per cell", and look

at N_{part} as the product of certain levels of spatial and velocity-space resolution (measured by N_{cell} and N_{ppc}, respectively).

In Ref. [9], the PIC version of the code has been applied to the investigation of the linear stability and the non-linear saturation of Alfvén modes in Tokamaks, in the presence of an energetic-ion population. The main results obtained in that analysis can be summarized as follows: as the energetic-ion pressure increases above a certain threshold, depending on the toroidal number n, the fast-growing energetic particle mode (EPM) becomes unstable [8]. The EPM saturates because of a sudden displacement of a large part of the energetic particles along the minor radius. Such a displacement can, in principle, greatly enhance the energetic-particle losses, and makes the determination of the threshold for EPM destabilization an important issue in the theoretical research on reactor-relevant plasmas.

Fig. 2 shows the growth rate of the EPMs obtained by PIC and FSP HMGC simulations at different values of n and β_H (the ratio between the energetic-particle pressure and the magnetic one). The case of a plasma with aspect ratio $R_0/a = 10$ and energetic-particle Larmor radius ρ_H such that $\rho_H/a = 0.01$ is considered here. A density profile of the form $\exp[-\left(r^2/L_n^2\right)^{\alpha_n}]$ has been assumed for the energetic ions, with $a^2/L_n^2 = 2$ and $\alpha_n = 2$. Slight quantitative differences between the results obtained by the two methods can be traced back to the different algorithms adopted, but a substantial agreement is found between the two approaches.

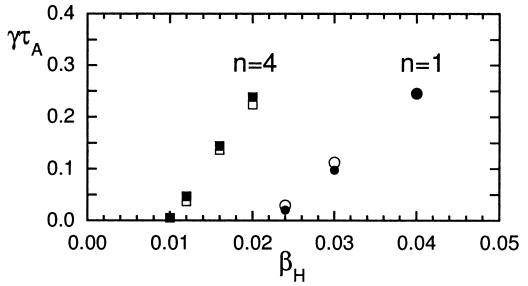


Fig. 2. Growth rate of energetic particle modes, normalized to the inverse of the Alfvén time τ_A, obtained, at different values of n and β_H, by PIC (full symbols) and FSP (empty symbols) HMGC simulations.

4. Implementation of the parallelization strategy in HPF

The FSP HMGC has been parallelized, according to the particle decomposition strategy depicted in Section 2, within the high performance Fortran framework [10]. The analogous strategy for the parallelization of the corresponding PIC version is presented in Ref. [7].

High performance Fortran (HPF) is a programming language designed to support the data parallel [2] programming style. HPF eliminates the complex, error-prone task of explicitly programming how, where and when to pass messages between processors on distributed memory machines. The underlying HPF compiler is responsible for producing code to distribute the array elements on the available processors. The single program multiple data (SPMD) [11] message-passing code produced by the compiler 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. Compiler directives such as `PROCESSORS`, `ALIGN`, `DISTRIBUTE` and `TEMPLATE` are introduced to control the alignment and distribution of array elements on the (abstract) processors; directives such as `INDEPENDENT` and language constructs such as `FORALL` are introduced to express data parallelism. An extended set of intrinsic functions and a standard library provide parallel functionality at a high level of abstraction. `EXTRINSIC` procedures standardize the interface with other languages and sequential or parallel execution schemes, and suited directives can be used to address sequence and storage association issues. See Refs. [10,12,13] for an extended overview of the language and its features.

The strategy depicted in Section 2, both for data and work distribution, can be implemented, quite straight-

[2] In this model, the same operation is performed on many data elements by many processors simultaneously. Global data structures are distributed uniformly, and processor power follows data. Thus a single program controls the distribution of, and operations on, data distributed across all processors.

forwardly, in the framework of the HPF paradigm. 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 loop iterations over the particles, which implement the computations needed for the particle pushing and the pressure-tensor Fourier components updating phase. The underlying HPF compiler will distribute those iterations by following the "owner-computes rule" applied to the distributed data.

The updating of the pressure-tensor Fourier components presents two strictly linked problems: (1) these quantities are replicated, and thus must be kept consistent among the processors, and (2) each quantity takes contribution from particles that reside on different nodes. The strategy adopted to solve this problem 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 the same properties; we can then apply the same strategy to the task of computing such quantities.

The scheme to handle with these "inhibitors of parallelism" within the loops over the particles (the pressure-tensor Fourier 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 — which store the values of these quantities, are replaced, within the bodies of the distributed loops, by 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 by using HPF intrinsic reduction functions such as `SUM`.

As an example, we consider the following skeletonized F90 code excerpt, where each element of the two-dimensional array `p` is updated by the contribution of the particles falling in the neighbours of the corresponding radial grid point.

```
p=0.
do l=1,n_part
  weight_l=weight(l)
  r_l    =r(l)
  j_r    =f_1(r_l)
  do i=1,n_mode
    p(i,j_r)=p(i,j_r)
    +f_2(weight_l,r_l,i)
  enddo
enddo
```

where `f_1` and `f_2` are suited non-linear functions of the particle weight and/or radial coordinate. This excerpt represents, very schematically, the updating of the pressure-tensor Fourier components. More generally, the computation of the array `p` is representative of the computation of any quantity that inhibits the parallel execution of the loops over the particles.

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

```
real*8, dimension (1:n_mode,0:n_r,
&    number_of_processors()):: p_par
real*8, dimension (n_part) :: weight, r
!HPF$ DISTRIBUTE (CYCLIC) :: weight, r
!HPF$ ALIGN WITH weight(:) :: *
&    p_par(*,*,:)
     n_proc=number_of_processors()
     p_par=0.
 !HPF$ INDEPENDENT,
&    NEW(weight_l,r_l,j_r,i_proc)
     do l=1,n_part
       weight_l=weight(l)
       r_l  =r(l)
       j_r  =f_1(r_l)
       if (mod(l,n_proc)=0) then
&       i_proc=n_proc
```

```
      else i_proc=mod(l,n_proc)
      endif
      do i=1,n_mode
        p_par(i,j_r,i_proc)=
&         p_par(i,j_r,i_proc)
          + f_2(weight_l,r_l,i)
      enddo
    enddo
    p(1:n_mode,0:n_r)=
&  SUM(p_par(1:n_mode,0:n_r,:),dim=3)
```

The pages of the structure `p_par` are distributed to the processors. The code restructuring within the loop is limited to the computation (`if ...`) of the page of `p_par` that is local to the particle considered in that iteration, and the updating of the proper element belonging to that page.

The reduction of the (distributed) pages of `p_par` in `p` (replicated) is very easily performed by using the HPF intrinsic function `SUM`. The only need for communication is related to this reduction and the subsequent broadcast, and thus it is embedded in the execution of the intrinsic function. 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 ones.

The computation for the selection of the page of `p_par` local to each particle, even though not complex, is anyway a non-linear function of the loop index (`l`). This, together with the presence of indirect references (through the elements of an array) to the elements of each page of `p_par`, could represent a problem if the target HPF compiler is able to perform data-dependence analysis for array elements, and actually performs an unrequested check about the assertion of independence of the loop iterations, provided by the user with the `INDEPENDENT` directive. In this case, such a compiler would 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 can be anyway quite easily bypassed with the help of the HPF extrinsic procedures `HPF_LOCAL`.

HPF programs may call non-HPF subprograms as *extrinsic procedures* [13]. 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 has to be executed 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 a 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.

In our case, we use 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 cannot 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 become the bodies of the extrinsic procedures, as exemplified, for the case under examination, in the following:

```
  INTERFACE
  EXTRINSIC(HPF_LOCAL)
&subroutine extr_pressure
&(weight,r,p_par)
 real*8, dimension(:), intent(in)
& :: weight,r
 real*8, dimension(:,:,:),
& intent(out) :: p_par
```

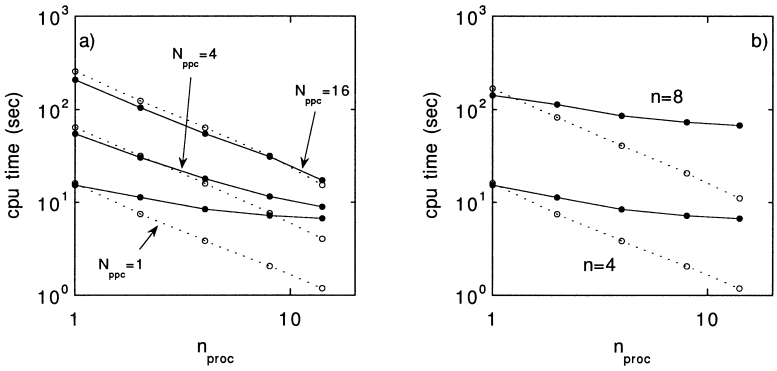


Fig. 3. CPU *user time* per time step versus n_{proc}. Results obtained by FSP (empty symbols) and PIC (full symbols) simulations with fixed mode number ($n = 4$) and different values of N_{ppc} (a), or fixed velocity-space resolution ($N_{\text{ppc}} = 1$) and different values of n (b) are compared.

```
!HPF$ DISTRIBUTE (CYCLIC) :: weight,r
!HPF$ ALIGN WITH weight(:) ::
& p_par(*,*,:)
      end subroutine extr_pressure
      END INTERFACE
      ...
      call extr_pressure(weight,r,
     &    p_par)
      p(1:n_mode,0:n_r)=
     &   SUM(p_par(1:n_mode,0:n_r,:),
     & dim=3)
      ...
      EXTRINSIC(HPF_LOCAL)
     & subroutine extr_pressure
     & (weight,r,p_par)
      real*8, dimension(:), intent(in)
     & :: weight,r
      real*8, dimension(:,:,:),
     & intent(out) :: p_par
        p_par=0.
        do l=1, UBOUND(weight,dim=1)
        weight_l=weight(l)
        r_l      =r(l)
        j_r      =f_1(r_l)
        do i=1,n_mode
           p_par(i,j_r,1)=
        &  p_par(i,j_r,1)
        &  +f_2(weight_l,r_l,i)
          enddo
        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`) and the page of the "partial results" array `p_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 `p_par` (`p_par(1:n_mode,0:n_r,1)`) assigned to it. At the end of the execution of the local extrinsic procedure, all the partial updates of the components of `p` are collected in the global-HPF-index-space `p_par`. The, `p_par` is reduced to `p` by using the `SUM` intrinsic function, as seen before.

5. Parallelization results

The parallel version of the FSP HMGC has been tested on a 16-node IBM SP parallel system, by using the IBM *xlhpf* compiler [14] (an optimized native compiler for IBM SP systems).

Fig. 3 shows the central processor unit (CPU) *user time* required by each simulation time step versus the number of processors. Results obtained by FSP (empty symbols) and PIC (full symbols) simulations with fixed mode number ($n = 4$) and different values of N_{ppc} (a), or fixed velocity-space resolution ($N_{\text{ppc}} = 1$) and different values of n (b) are compared. It can be seen that the FSP approach becomes more advantageous when the number of processors exceeds a cer-

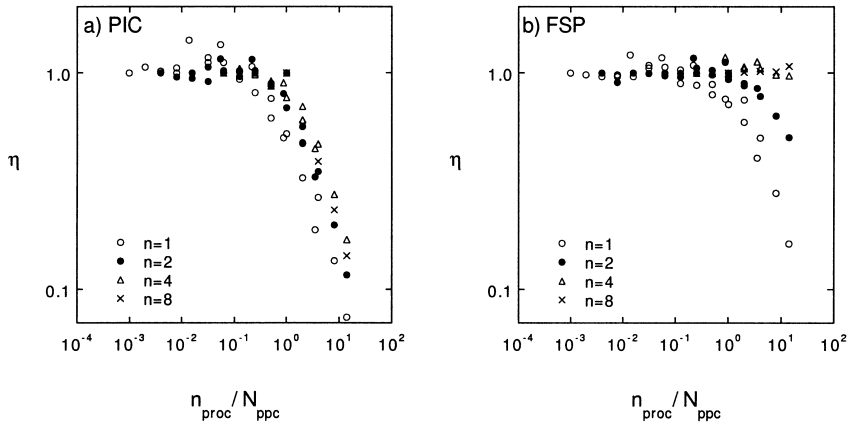


Fig. 4. Efficiency η, defined as the speed-up factor divided by the number of processors, versus $n_{\mathrm{proc}}/N_{\mathrm{ppc}}$. Simulations retaining modes with a single toroidal number are considered. The results obtained by PIC HMGC (a) are compared with those obtained by FSP HMGC (b), for several different choices of n.

tain threshold value, which increases with N_{ppc} and more weakly with n. From Eqs. (7) and (9), we expect that a gridless FSP code performs more efficiently than a PIC one for

$$n_{\mathrm{FT}} \times N_{\mathrm{harm}} \times N_{\mathrm{cell}} + n_{\mathrm{int}} \times \frac{N_{\mathrm{part}}}{n_{\mathrm{proc}}} \gtrsim n_{\mathrm{FT}} \times N_{\mathrm{harm}} \times \frac{N_{\mathrm{part}}}{n_{\mathrm{proc}}}, \tag{14}$$

or

$$n_{\mathrm{proc}} \gtrsim N_{\mathrm{ppc}} \left(1 - \frac{n_{\mathrm{int}}}{n_{\mathrm{FT}} \times N_{\mathrm{harm}}}\right). \tag{15}$$

Considering that $N_{\mathrm{harm}} \propto n$, the results shown in Fig. 3 appears to be in a qualitative agreement with such conclusions.

The efficiency η, defined as the speed-up factor divided by the number of processors, is plotted, in Fig. 4, versus $n_{\mathrm{proc}}/N_{\mathrm{ppc}}$. Simulations retaining modes with a single toroidal number n are considered here. The results obtained by FSP HMGC are compared with those obtained by PIC HMGC for several cases, corresponding to different choices of n. It is possible to note that, for each value of n, the efficiency essentially maintains its ideal value ($\eta \approx 1$) up to a certain threshold value in $n_{\mathrm{proc}}/N_{\mathrm{ppc}}$. Such a value comes out to be higher in the FSP case than in the PIC one, as we expect from the arguments developed in Section 2, because of the lesser amount of replicated calculations that characterizes the FSP code. [3]

The most relevant features, however, is represented by the fact that, while for the PIC code the threshold value exhibits a negligible dependence on n, for the FSP one such a dependence is positive and much stronger. This fact can be understood by considering that the threshold value is approximately obtained by imposing that the replicated calculations exceed the distributed ones. For the PIC case, from Eq. (7), such a condition can be written as

$$f\,(N_{\mathrm{harm}}) + n_{\mathrm{FT}} \times N_{\mathrm{harm}} \times N_{\mathrm{cell}} \gtrsim n_{\mathrm{int}} \times \frac{N_{\mathrm{part}}}{n_{\mathrm{proc}}}. \tag{16}$$

Taking into account how the different terms scale with n (see Section 3), this yields, for the threshold, the following approximate dependence:

$$\left.\frac{n_{\mathrm{proc}}}{N_{\mathrm{ppc}}}\right|_{\mathrm{th}}^{\mathrm{PIC}} \propto \frac{n^3}{f(n) + \alpha n^4}, \tag{17}$$

with α being a coefficient depending on n_{FT} and other parameters related to the specific simulations consid-

[3] 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.

ered. For the FSP code, the corresponding condition is obtained from Eq. (9) and reads

$$f\,(N_{\mathrm{harm}}) \gtrsim n_{\mathrm{FT}} \times N_{\mathrm{harm}} \times \frac{N_{\mathrm{part}}}{n_{\mathrm{proc}}}. \tag{18}$$

The threshold is then given by

$$\left.\frac{n_{\mathrm{proc}}}{N_{\mathrm{ppc}}}\right|_{\mathrm{th}}^{\mathrm{FSP}} \propto \frac{n^4}{f(n)}. \tag{19}$$

From Eqs. (17) and (19), we get for the ratio between the above threshold values,

$$\frac{\left.\frac{n_{\mathrm{proc}}}{N_{\mathrm{ppc}}}\right|_{\mathrm{th}}^{\mathrm{FSP}}}{\left.\frac{n_{\mathrm{proc}}}{N_{\mathrm{ppc}}}\right|_{\mathrm{th}}^{\mathrm{PIC}}} \propto n\left[1 + \frac{\alpha n^4}{f(n)}\right]. \tag{20}$$

Our results show how, for simulations characterized by high resolution in the real space (high n values) as well as in the velocity space (high N_{ppc} values), the particle-decomposition parallelization of FSP codes, different from the same parallelization of PIC codes, is very efficient even for massively parallel architectures.

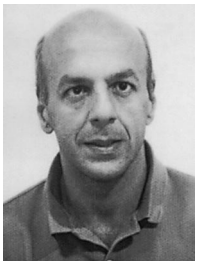
6. Conclusions

In the present paper, we have shown that, in the framework of the *gridless finite-size-particle* approach, highly efficient parallel particle simulations can be obtained. In particular, the parallelization of the FSP codes comes out to be more efficient, as far as few harmonics of the fluctuating fields are retained in the simulation, than the one obtained in the framework of the standard *particle-in-cell* approach. The comparison of the two methods, applied to a specific particle simulation code, show that a very good agreement exists under the condition that the size of the simulation particles in the FSP method is of the same order of the spacing of grid points in the PIC one. In the *few-harmonic* limit, the FSP method allows one to overcome the difficulties associated to the *particle-decomposition* parallelization. In spite of the high real-space resolution required by such simulations, indeed, the memory required by the storage of replicated arrays is, in this case, drastically reduced in comparison with PIC simulations. Further benefits are given by the enhanced distributed character of the calculations needed to compute the fields acting on each particle; this makes the gridless FSP method more efficient than the PIC one in the framework of parallel computation on distributed-memory architectures, although it comes out to be typically heavier in the serial limit. On the other hand, all the advantages that make the *particle-decomposition* approach to parallelization of PIC codes more convenient than the *domain-decomposition* one are preserved in the FSP case: perfect load balancing, simplified and reduced inter-processor communication, easy implementation of the parallel version of originally serial codes. Almost ideal efficiency is obtained in gridless FSP simulations even in the limit of high number of computational nodes. This is particularly true in the case of high resolution both in real and in velocity space, because the relative weight of the distributed calculation becomes higher in such a limit.

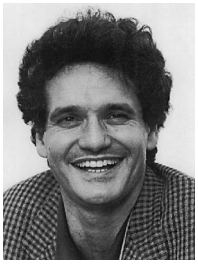
References

[1] C.K. Birdsall, A.B. Langdon, Plasma Physics via Computer Simulation, McGraw-Hill, New York, 1985.
[2] A. Sestero, Basic interactions in real plasmas and in the plasmas of numerical experiments, Il Nuovo Cimento B 9 (1972) 222–232.
[3] R.W. Hockney, Computer simulation of anomalous plasma diffusion and numerical solution of Poisson's equation, Phys. Fluids 9 (1966) 1826–1835.
[4] A.B. Langdon, C.K. Birdsall, Theory of plasma simulation using finite-size particles, Phys. Fluids 13 (1970) 2115–2122.
[5] S. Briguglio, G. Vlad, F. Zonca, C. Kar, Hybrid magnetohydrodynamic-gyrokinetic simulation of toroidal Alfvén modes, Phys. Plasmas 2 (1995) 3711–3723.
[6] P.C. Liewer, V.K. Decyk, A general concurrent algorithm for plasma particle-in-cell codes, J. Comput. Phys. 85 (1989) 302–322.
[7] B. Di Martino, S. Briguglio, G. Vlad, P. Sguazzero, Parallel plasma simulation in high performance Fortran, in: High Performance Computing and Networking, Springer, Berlin, 1998, pp. 203–212.
[8] L. Chen, F. Zonca, Theory of shear Alfvén waves in toroidal plasmas, Phys. Scripta T60 (1995) 81–90.
[9] S. Briguglio, F. Zonca, G. Vlad, Hybrid magnetohydrodynamic-particle simulation of linear and non-linear evolution of Alfvén modes in tokamaks, Phys. Plasmas 5 (1998) 3287–3301.
[10] H. Richardson, High Performance Fortran: history, overview and current developments, Technical Reports TMC-261, Thinking Machines Corporation, 1996.

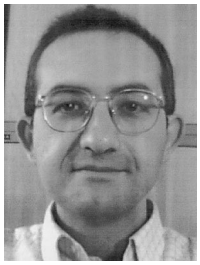
[11] F. Darema et al., A single program multiple data computational model for EPEX/Fortran, Parallel Comput. 7 (1) (1988) 11–24.
[12] High Performance Fortran Forum, High performance Fortran language specification, Sci. Programming 2 (1/2) (1993) 1–170.
[13] High Performance Fortran Forum, High Performance Fortran Language Specification, Version 2.0, Rice University, 1997.
[14] 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 the SuperComputing'95, ACM, 1995.



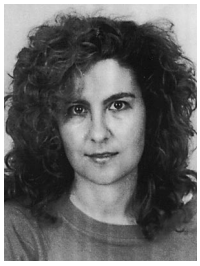
Sergio Briguglio received a degree in Physics in 1980, and M.S. degree in Fusion Engineering in 1983, both from Padua University. Since 1983 he is a Research Staff Member in the Fusion Division at the ENEA (the Italian National Agency for New Technology, Energy and the Environment) Research Centre in Frascati, Rome. He has served as a Visiting Physicist at the UKAEA and JET Laboratories, Culham, and the Princeton Plasma Physics Laboratory. His research interests include microinstabilities, anomalous transport, Alfvén modes and related energetic particle dynamics in magnetically confined plasmas, particle simulation techniques, traffic simulation.



Gregorio Vlad received a degree in Physics in 1981 and Ph.D. degree in Physics in 1987, from the University of Rome, Italy. From 1984 he is a permanent staff Researcher at ENEA Fusion Division. He has served as a Visiting Physicist in several institutions (MIT, USA; JET-Culham, UK; Chalmers University, Sweden; CRPP-EPFL, CH; Max-Planck-Institut für Plasmaphysik, Garching, Germany). His research interests include theory and numerical simulation of magnetohydrodynamic stability, Alfvén modes in the presence of energetic particles, energy transport in thermonuclear plasmas. He is the author of more than 90 publications in international journals and conferences.



Beniamino Di Martino received the M.S. degree in Physics and Ph.D. in Computer Science, both from University of Naples, Italy, in 1992 and 1996, respectively. In 1994 he joined the Institute for Software Technology and Parallel Systems at the University of Vienna, Austria, where he has served as a researcher till 1998. He is currently Assistant Professor at the Electronic Engineering Faculty, Second University of Naples. He was consultant for IBM Semea and ENEA in Rome, Italy. He is the author of more than 30 publications in international journals and conferences. His research interests include Compiler Techniques for High Performance Computing, Parallel Computing and Architectures, Automated Program Analysis and Transformation.



Giuliana Fogaccia was born in Rome in 1968. In 1991, she received a degree in Physics, magna cum laude, and the Ph.D. degree in Physics from the University of Rome, Italy. In 1992, she obtained a fellowship at the ENEA Research Centre of Frascati. She is now working as a researcher at the ENEA Fusion Division. Her research interests include the field of fusion plasma theory (kinetic theory, magnetohydrodynamic theory, turbulence, dynamical systems) and simulation techniques (Lattice Boltzmann methods).