Progress in Mesh-Free Plasma Simulation with Parallel Tree Codes

Paul Gibbon*, Robert Speck*, Benjamin Berberich*, Anupam Karmakar*†, Lukas Arnold * and Martin Mašek ‡
*Institute for Advanced Simulation, Jülich Supercomputing Centre
Forschungszentrum Jülich GmbH, 52425 Jülich, Germany
Email: {p.gibbon, r.speck, b.berberich, a.karmakar, l.arnold}@fz-juelich.de
†ExtreMe Matter Institute EMMI, GSI Helmholtzzentrum für Schwerionenforschung GmbH, Planckstraße 1, 64291 Darmstadt, Germany
‡Institute of Physics, Academy of Sciences of the Czech Republic, Prague, Czech Republic
Email: masekm@fzu.cz

***Abstract*—Recent developments in mesh-free plasma modelling using parallel tree codes are described, covering algorithmic and performance issues, and how to apply this technique to multi-dimensional electrostatic plasma problems. Examples are given of simulations of ion acceleration by high intensity lasers, as well as more recent applications of this technique to vortex-fluids and edge-plasma modelling in tokamaks.**

I. Introduction

Nearly all kinetic plasma modelling over the past four decades has utilized a spatial mesh to mediate the interplay between plasma particles and their associated electric and magnetic fields. These models have proved highly successful, as exemplified by the now routine use of three-dimensional, electromagnetic particle-in-cell (PIC) codes for modelling nonlinear wave-particle phenomena in all areas of plasma physics. Despite this success, the presence of a grid ultimately places restrictions on the spatial resolution, dynamic range or geometry which can be handled by PIC codes, especially in three dimensions. These difficulties can be tackled to some extent with adaptive mesh refinement, but usually at the expense of reduced parallelism. Recently a versatile, mesh-free plasma simulation paradigm has been introduced which overcomes some of these limitations for certain classes of problem. Inspired by the N-body tree algorithms originally conceived to speed up gravitational problems in astrophysics, this approach reverts to first principles by performing direct pair-wise summation of forces acting on the particles and subsequently integrating their trajectories in a Lagrangian, 'molecular dynamics' fashion.

Following the first application of this technique to strongly coupled plasmas [1], tree codes have since been used to model atomic clusters [2], two-dimensional bounded plasmas [3] and electrostatic sheaths in tokamaks [4]. Despite their reduced physics content compared to fully electromagnetic particle-in-cell codes, mesh-free tree codes already offer completely new possibilities in plasma simulation, particularly where collisions are important; for modelling complex geometries including strong density gradients; or for mass-limited systems in which artificial boundaries would normally compromise the simulation's validity (for example atomic clusters). In the Plasma Simulation Laboratory at Jülich Supercomputing Centre we are developing this technique further to included self-generated magnetic fields, and have been using a parallel electrostatic tree code to study three-dimensional phenomena in laser and particle beam interactions with various types of complex plasma target [5]–[8].

Exploiting this algorithm on modern parallel computer architectures is a significant challenge in computational science: our current electrostatic version (PEPC-E) is currently capable of performing dynamic simulations with 256 million particles on 8192 cores of the new JSC BlueGene/P system In this paper a brief introduction to the technique of mesh-free plasma simulation based on heirarchical tree codes is given, pointing out its merits and potential pitfalls. Recent applications in the areas of laser-plasma interactions and magnetic fusion will be presented. Finally, prospects for future magnetoinductive, radiation free simulations including slowly varying electric and magnetic fields will also be addressed.

II. Plasma Tree Codes: the Basics

Algorithmically speaking, a plasma tree code is no different to the many Newtonian gravity N-body solvers used in astrophysics, which are nearly all based on some for of the Barnes-Hut hierarchical tree algorithm [9].

Briefly summarized: the electrostatic force-sum on each particle is computed by systematically replacing more distant charges by multipole expansions of charge groups, thus reducing the standard $O(N^2)$ direct sum to an $O(N \log N)$ complexity at the price of a small, controllable error [10]. For most plasma physics applications there is no need to compute potentials and forces to higher accuracy than the error incurred by time-integration, which can be anywhere between 10^{-4} for a high-order Runge-Kutta scheme, to around 1% for the simple 2nd-order Leap-Frog method. Compared to most PIC codes, which neglect short-range contributions completely, this type of force-summation is more expensive, but orders of magnitude more accurate.

Domain decomposition: At first sight, the hierarchical data structure required in a tree code to manage the multipole information would seem to rule out parallelism altogether. In fact,

several schemes for parallel tree codes have been proposed and implemented, including virtual shared-memory versions [11], and distributed memory schemes using geometrical domain decomposition methods [12], [13]. The scheme adopted here follows the one devised by Salmon and Warren [14], [15], who practically reinvented the BH algorithm by scrapping memory pointers for bookkeeping in favour of a set of universal binary keys to represent particle and tree-node coordinates alike.

The basic idea is to convert the coordinate triple of each particle into a single, unique 64-bit integer key. The keys do not *replace* the coordinates, but as we shall see, provide a natural and rapid means of sorting the particles and building up the tree structure around them. Given its key and owner, locating any node in the tree is reduced to an $O(1)$ operation.

In the present code, PEPC, the keys are constructed from the binary interleave operation:

$$k = p + \sum_{j=0}^{nb-1} 8^j (4 \times \mathtt{bit}(i_z, j) + 2 \times \mathtt{bit}(i_y, j) + \mathtt{bit}(i_x, j))$$

The function `bit()` selects the jth bit of the integer coordinate component (i_x, i_y, i_z), which are computed from:

$$i_x = x/s, \text{etc.},$$

where

$$s = L/2^{\mathtt{nlevels}}$$

and L is the simulation box length; `nlevels` the maximum refinement level. The latter obviously depends on the machine precision, and for a 64-bit machine, we can have 21 bits per coordinate (or `nlevels=20`) plus a place-holder bit:

$$p = 2^{63}.$$

This procedure yields a space-filling curve following the so-called Morton or Z-ordering, a 2-dimensional example of which is shown in Fig. 1 below.

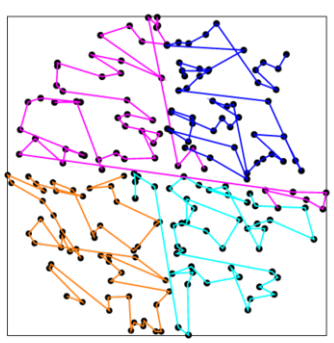


Fig. 1. 2-dimensional Morton (Z-) ordering of 200 simulation particles, equally shared among 4 processor domains.

The simulation particles are then sorted according to the list of binary keys generated above. The fully parallel sort currently implemented is a recursive adaptation of the PSRS (parallel sort by regular sampling) algorithm originally proposed in Ref. [16]. Since the distribution of keys depends sensitively on the geometry of the system simulated—that is, whether the particles are initially arranged in a cube, sphere or more complex object—regular sampling tends to produce highly imbalanced particle numbers across the processors. To compensate this effect, we instead use weighted sampling, which allows for the actual distribution of keys along the whole space-filling curve (Fig. 1).

A big advantage of binary coordinate ordering over standard addressing techniques in tree codes is that the hierarchical structure is recovered automatically. Keys of parent and neighbour cells are obtained by simple bit operations, so that the average access-time for any particle or node in the tree is $O(1)$ instead of the usual $O(\log N)$. The obvious drawback is that the number of possible keys, $2^{63} \simeq 10^{19}$ on a 64-bit machine, vastly exceeds the memory available, typically $\sim 10^5 - 10^6$ locations per processor. This mismatch is resolved by using a hashing function to map the key onto a physical address in memory, for example:

$$\mathtt{address} = \mathtt{k\ AND}\ (2^h - 1), \tag{1}$$

where h is the number of bits available for the address. This address then acts as a pointer to the particle or multipole properties. In case two or more keys give the same address (a 'collision'), a linked-list is constructed to resolve it. Clearly a high occurence of collisions will ultimately degrade performance; however, as Warren & Salmon pointed out [15], the distribution of particles and nodes between many processors with their own address-spaces helps to reduce their number to a negligible level.

Domain decomposition is then reduced to the almost trivial task of cutting out equal portions of the sorted list and allocating these to the processors. An decomposition example for 200 particles divided amoung 4 processors is also seen in Fig. 1. Note that with this scheme, load balancing can be easily introduced by biasing the key-list segments according to the number of interactions computed for each particle in the force summation during the previous iteration.

Construction of local trees: Once a set of particles has been allocated to a particular processor, and their associated properties (mass, charge, velocity etc.) have been fetched from their original location, one can immediately begin to construct the local trees. This can be done very efficiently because the particle keys implicitly contain the necessary information on all their ancestor nodes up to the root. The parent of a particle or twig-node is simply found by a 3-bit shift operation:

$$k_{\text{parent}} = \mathtt{RIGHTSHIFT(k,3)} \tag{2}$$

Likewise, if a node's children are numbered from 0 to 7 (in a 3D oct-tree), their keys can be obtained by the inverse operation:

$$k_{\text{child}} = \mathtt{LEFTSHIFT(k,3)\ OR\ child(0-7)}, \tag{3}$$

The local sorted list of particle keys would thus provide a natural starting point for determining their parent nodes if we

knew how they were distributed. In a dynamic application we cannot assume anything about their distribution, however, so instead we start from the highest (coarsest) level and work down to the leaves. As in a sequential algorithm [10], all particles are initially attached to the root, in this case a cube encompassing the whole simulation region. Next, the region is subdivided into 8 sub-boxes, and the particles re-attached accordingly. A sub-box containing exactly one particle is defined as a leaf; a box with 2 or more constitutes a twig and empty boxes are discarded. This procedure is continued at the next highest level until each particle sits in its own box. For highly clustered distributions, it may become necessary to relax this requirement, otherwise simulations with more than a few million particles may result in identical key assignments.

Each new leaf or twig node created this way is added to the local hash-table via the same hash function (1) as the particles. Collisions are again dealt with via a simple linked list. In principle this function can be refined to improve the distribution of hash-table addresses in memory space: the sharing of keys across a number of processors keeps the collision count down to tolerable levels.

Global branch nodes: At their coarsest level, the local trees will contain 'incomplete' twig nodes; that is, nodes which cross domain boundaries. Information from neighbouring domains is therefore needed to complete them. To facilitate the exchange of information (and later multipole moments) between processors, a set of local 'branch' nodes is defined first, comprising the minimum number of *complete* twig and leaf nodes covering the whole local domain—Fig.2.

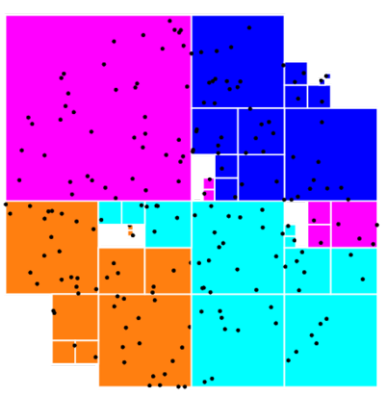
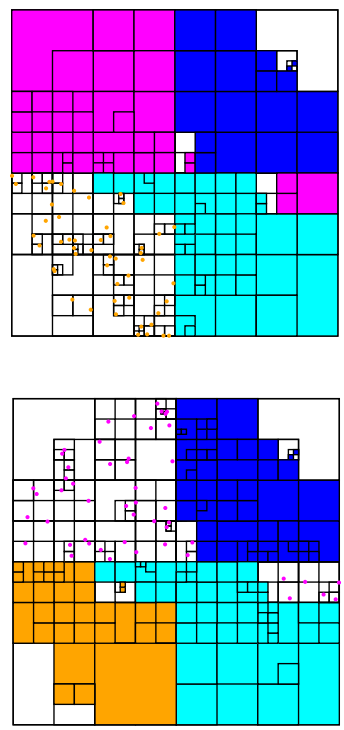


Fig. 2. Branch nodes belonging to 4 processor domains.

This set of branch nodes is then broadcast to all other processors, so that each one subsequently knows where to request any missing non-local particle or multipole information. For example, a branch's child nodes can immediately be found from a byte code stored with the hash-table entry, the first 8 bits of which declare which children exist at the next refinement level. Applying the operation (3) yields each (still non-local) child key. A branch's hash-entry will also contain the total number of particles contained beneath it, so that the top level nodes above can now be filled in up to the root. At this point the local trees comprise 3 types of node: i) twig or leaf nodes covering the local domain, ii) branch nodes and iii) top level twig nodes, each covering the whole simulation region—Fig.3. Leaf node entries contain a pointer to the actual particle coordinates, charge and mass, as well as a globally unique label for tracking purposes. Twig nodes, including the special branch nodes, contain pointers to the multipole moments of their associated charge distributions, together with some flags indicating the status of non-local child nodes (in particular, whether a local copy already exists).



a)

b)

Fig. 3. Local trees for a) processor 0 and b) processor 2 prior to tree-walk. The shaded boxes represent the branch nodes gathered from all remote processors.

Construction of multipole moments: Once the basic tree structure is in place, it is a straightforward matter to accumulate multipole moments for each node from the leaves up. Once again, this procedure is considerably simplified by sorting the keys for the twig-nodes contained within the list of local branch nodes. Twig nodes with the highest keys will, by definition, have the highest refinement levels:

$$\texttt{level} = \frac{\log(\texttt{key})}{\log 8} \tag{4}$$

This means that multipole moments at higher levels can be successively shifted up to their parent levels using simple displacement vectors as described in [10]. This procedure is continued by working through the sorted list of twigs in reverse order up to the local branch nodes, which then contain the *complete* multipole information for the local domain. This information is then broadcast to all other processors, so that the remaining top-level nodes can be filled in using the above

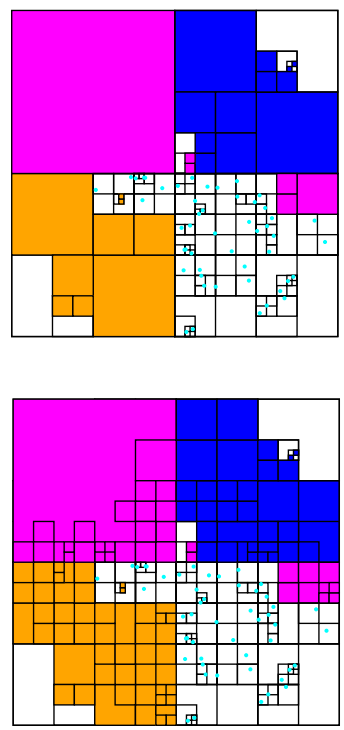
Fig. 4. Parallel tree-walk algorithm for determining interaction lists for a batch of particles.

```
while any defer list still > 0 do
  while any particle not finished walk do
    find next node on particles' tree-walks
    if MAC OK then
      put node on interaction list
      walk-key = next-node
    else if MAC not OK for local node then
      subdivide: walk-key = first-child
    else if MAC not OK for non-local node then
      walk-key = next-node
      put particle on 'defer' list
      put node on request list
    end if
    remove finished particles from walk list
  end while
  gather request lists for non-local nodes from all processors and discard duplicates
  for all remote processors do
    initiate receive buffer for incoming child data
    send off requests for remote child data
  end for
  for all remote processors do
    test for incoming request
    package and ship back child multipole data to processor that requested it
  end for
  for all requests do
    if data has arrived for requested node then
      create new hash-table entries for each child
    end if
  end for
  copy particle defer lists to new walk lists for next pass through tree
end while
```

shifting rules. At the end of this procedure, each processor has the complete multipole expansion for the entire simulation region contained in the root node.

Tree traversal: building interaction lists : By far the most important and algorithmically demanding part of a parallel tree code is the tree-traversal, which in the present asynchronous implementation requests multipole information 'on the fly' from non-local processor domains. Rather than performing complete traversals for one particle at a time, as many 'simultaneous' traversals are made as possible, thus minimizing the duplication incurred when the same non-local multipole node is requested many times and ii) maximising the communication bandwidth by accumulating many nodes before shipment. In practice, this means creating interaction lists for batches of 200–1000 particles at a time before actually computing their forces. The routine `tree_walk`, which finds the interaction list for each batch has the structure depicted in Fig. 4.

In the first half of this routine, traversals are made through the local tree using the familiar divide-and-conquer strategy common to sequential tree codes [17]. The multipole acceptance criterion (MAC) determines whether to accept or subdivide local nodes as usual, but also provides for a third possibility: the subdivision of a *non*-local node for which child data is not yet available. This is then placed on a special 'request list' to be processed in the 2nd half of the routine when all particles have completed their traversals as far as they can with the available node data. Each processor then compiles a lists of nodes it needs child data from, and sends them to the owners of the parent nodes. In the first pass, these will just be the branch nodes. On receipt of a request list, a processor packages and ships back the multipole data for the children. The use of non-blocking SENDS and RECEIVES for the multipole information allows some overlap of communication with the creation of new hash-table entries locally. At the end of all the traversals, each processor's local tree contains all the nodes required to compute the forces on its own particles. The nodes fetched during the traversals actually take up most of the space in the local hash-table, as Fig. 5 illustrates.



a)

b)

Fig. 5. Tree for processor 1, the domain in bottom right quadrant: a) before and b) after traversals for all locally held particles.

Force summation : Once an interaction list has been found for a particle, it is a straightforward task to compute its force and/or potential. Separation of the actual force sum from the tree traversal has the advantage that this floating-point-intensive routine can be hardware-optimised. Also, the physics and algorithm are kept naturally apart, so that additional forces (for example, short-range components or magnetic fields)

and/or boundary conditions (for example, corrections from a periodic Ewald summation) can be added with relative ease. In the present implementation, forces are computed for each batch of interaction lists returned from the tree-walk routine. One subtlety which arises here is that even if overall load-balancing has been arranged during the domain decomposition, it is not necessarily guaranteed for each batch of particles (which may comprise only 1/100 of the total number on each processor). To redress this problem, the batch size N_b for each processor is determined individually, so that the integral

$$\sum_{p=1}^{N_b} N_{\text{int}}(p) \tag{5}$$

is the same, and each processor computes the same number of interaction pairs during each pass.

III. Algorithm Scaling and Performance

Initialise particle properties $\boldsymbol{r}_i, \boldsymbol{v}_i, q_i, m_i$	N/P
Key construction: $(x_i, y_i, z_i) \rightarrow k_i$	N/P
Sort keys: $k_1, k_2, ...k_N$	$N/P \log N$
Domain decomp.: $k_1, ..k_n$; $k_{n+1}, ..k_{2n}$; ...; $k_{N-n}...k_N$	N/P
Construct branch nodes	P $\log N/P$
Fill in top level local tree nodes	$\log P$
Build multipole moments	$\log N/P$
Construct interaction lists (tree traversal)	$N/P \log N$
Compute forces and potential	$N/P \log N$
Update particle velocities and positions	N/P

TABLE I
Algorithmic scaling of major routines in PEPC. The symbols N and P represent the total number of particles and processors respectively, and $n = N/P$.

The overall algorithm is depicted together with the theoretical scaling of each major routine in Table I. We see that in principle, all of the above routines can be performed in parallel, and thus require a computational effort $O(N/P)$, give or take a slowly varying logarithmic factor. Single-timestep benchmarks with this new code broadly confirm the theoretical scalings indicated in Table I. As expected, most of the time is spent in the tree-traversal and force-summation routines: the total overhead incurred by the tree construction (which includes the steps 2.3, 2.3 and 2.4 described previously) is around 3%, although this figure excludes tree-nodes copied locally during the traversal—Table II.

The PEPC code is written in a generic fashion without the usage of external libraries. This results in excellent portability. In the PRACE benchmarking framework PEPC was run on four different computer architectures, namely: IBM Blue Gene/P (jugene), IBM Power6 (huygens), Cray XT5 (louhi), Intel Nehalem (juropa). As a test case we used $5 \cdot 10^7$ particles and a cubic, homogeneous initial distribution. The scaling behavior is shown in figure 6, where the time needed for one inner loop step is shown as a function of the peak performance of the used partition. As shown, PEPC is able to utilize the

Routine/ No. CPUs	8	16	64
Domain decomposition	0.2	0.24	0.33
Tree building	2.3	2.3	2.7
Tree traversal	32.9	36.1	40.8
Force summation	64.4	61.2	55.7

TABLE II
Breakdown of relative computational effort (percentage of wall-clock time spent in each routine) in the parallel tree code for a test case with 100k particles and 8, 16 and 64 processors respectively on a PC cluster

given partition performance independently of the architecture. Only the Intel Nehalem architecture shows a significantly better performance. PEPC's overall scaling behavior is also impressive for a tree code, although the current version is only able to use up to approximately 8192 cores. But with these the code is capable of simulating more than 10^8 particles.

Three different areas have been identified as challenges for further development of the code: The inevitable need for global communication in Barnes-Hut tree codes, a strong memory dependency of the number of cores and a reliable load balancing scheme [18]. These problems have to be well analysed in order to make resonable use of PEPC on larger partitions. While the second topic is a question of data structure and organisation the two others focus on communication patterns and their impact on the performance. It is clear that although the tree code is an intrinsically adaptive algorithm which performs very well even with very inhomogeneuous particle distributions, its performance depends strongly on a sophisticated load balancing scheme, which is closely linked to the domain decomposition method. The need for non-local communication is an inevitable consequence of the elliptic nature of the Poisson equation.

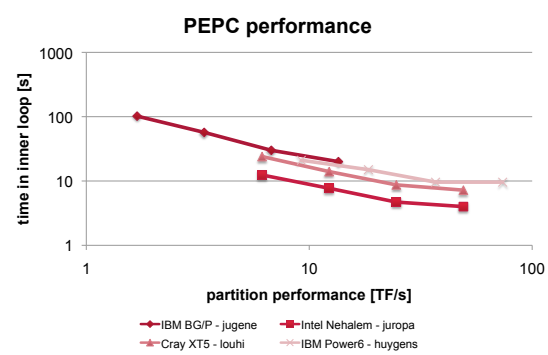


Fig. 6. Performance of PEPC on various HPC architectures, normalised to the theoretical peak performance of multi-core partition.

IV. Collisional Modelling with Mesh-Free Codes

An earlier incarnation of this code was used to perform molecular dynamics (MD) calculations of nonlinear inverse Bremsstrahlung absorption in strongly coupled plasmas [1],

[17]. In the present context, PEPC is used in the macroscopic sense, using quasi-particles to trace the dynamics of phase-space elements just as in PIC simulation. The analogy with PIC stops there, however: first, the present tree code does away with both spatial grid completely; second, both electron-ion and electron-electron collisions can be implicitly included in a natural, adjustable manner, drawing on the theoretical framework for 'finite-size-particle-' (FSP) simulation set out 30 years ago by Langdon, Okuda and Birdsall [19], [20].

According to this theory, the collisionality of a plasma comprising finite-sized clouds with radius ε is typically reduced by orders of magnitude compared to a 'real' plasma comprising point particles, so that the number of particles in a Debye sphere, $N_D = \frac{4\pi}{3} n \lambda_D^3$, is effectively replaced by the parameter $N_c = \frac{4\pi}{3} n \varepsilon^3$, where n, and λ_D are the number density and Debye length respectively. Okuda and Birdsall [20] expressed this attenuation effect quantitatively by evaluating the scattering cross-section σ_{cloud} for charge clouds numerically, and then plotting the ratio $\sigma_{\text{cloud}}/\sigma_{\text{point}}$ as a function of the cloud radius and N_D.

In the large cloud limit, $\varepsilon/\lambda_D \gg 1$ (the regime of interest for the present study), the curves in Fig. 7 of Ref. [20] can be fitted to better than 20% by a conveniently simple expression:

$$\frac{\sigma_{\text{cloud}}}{\sigma_{\text{point}}} = \frac{1}{3 \ln \Lambda} \left(\frac{\lambda_D}{\varepsilon} \right)^2, \tag{6}$$

where $\Lambda = 9 N_D$. Applying the usual definition $\nu = \overline{nv}\sigma$, one can thus write down an effective collision frequency for cloud charges:

$$\frac{\nu_c}{\omega_p} \simeq \frac{Z}{30 N_D} \left(\frac{\lambda_D}{\varepsilon} \right)^2 = \frac{Z}{30 N_c} \left(\frac{\varepsilon}{\lambda_D} \right). \tag{7}$$

This expression is also displayed graphically in Fig. 7. In obtaining this fit, we have also made use of the usual expression for point particles [21], [22]:

$$\frac{\nu_{ei}}{\omega_p} = \frac{1}{3} \left(\frac{\pi}{32} \right)^{1/2} \frac{Z \ln \Lambda}{N_D} \simeq \frac{Z \ln \Lambda}{10 N_D}, \tag{8}$$

where ω_p is the plasma frequency.

In PIC codes the particle size is usually equivalent to the grid spacing Δ, which must be kept $\leq \lambda_D$ to avoid aliasing instabilities, usually manifesting themselves as numerical heating [23]. To map the particle densities smoothly onto the grid, it is also desirable to have as many particles per cell as possible, or $N_c \gg 1$ (although 3-dimensional simulations are often performed with as few as 2–3 electrons/ions per cell). This combination means that PIC codes are typically operated in the bottom-left, 'collisionless' corner of Fig. 7, with $\nu_c/\omega_p \leq 10^{-2}$.

By contrast, the gridless FSP approach gives us the freedom to set up a simulation within a much larger area of the parameter space depicted in Fig. 7. The most sensitive parameter here is N_c, or equivalently $\varepsilon/\overline{a}$, the ratio of the cloud size to the average interparticle spacing. Even at modest densities, adjusting $\varepsilon/\overline{a}$ enables us to emulate both hot, collisionless plasmas where PIC simulation is valid; or the cold, collisional

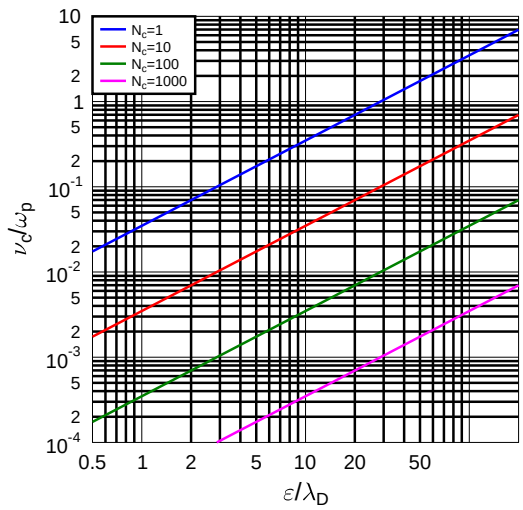


Fig. 7. (color online) Normalized electron-ion collision frequency for finite-sized particles as a function of cloud radius for various values of N_c.

state normally treated with hydrodynamic or Fokker-Planck modelling. Choosing a large ε/λ_D (or low temperature) will wash out details on the cold electron Debye-length scale (such as Langmuir wave dispersion), but this is of minor concern for problems where the physics is dominated by large (micron-) scale charge separation effects.

In practical terms, the finite cloud size is introduced via a smoothed Coulomb potential, which in PEPC takes the form:

$$\boldsymbol{\Phi}(\boldsymbol{r}) = \frac{q\boldsymbol{r}}{(r^2 + \varepsilon^2)^{1/2}}, \tag{9}$$

with the corresponding equation of motion for each particle i:

$$m_i \frac{d\boldsymbol{u}_i}{dt} = \frac{1}{3} q_i \sum_{i \neq j} \frac{q_j \boldsymbol{r}_{ij}}{(r_{ij}^2 + \varepsilon^2)^{3/2}} + q_i \boldsymbol{E}^p(\boldsymbol{r}_i), \tag{10}$$

where $\boldsymbol{r}_{ij} = \boldsymbol{r}_i - \boldsymbol{r}_j$ is the separation between particles i and j; $\boldsymbol{u}_i = \gamma \boldsymbol{v}_i$ is its proper velocity with relativistic factor $\gamma = (1 + |\boldsymbol{u}|^2/c^2)^{1/2}$; $\boldsymbol{E}^p$ is an external field arising from the laser (see Sec. III).

In Eqs. 9 and 10 the variables t, r, v, q, m, Φ and E have been normalized to $1/\omega_p$, c/ω_p, $N_p e, N_p m_e, m_e c^2/e$ and $m_e \omega_p c/e$ respectively, where

$$N_p = \frac{4\pi}{3} n_e \left(\frac{c}{\omega_p} \right)^3 \tag{11}$$

is a dimensionless constant representing the number of physical charges contained within a simulation particle.

V. Numerical Plasma Preparation

To set up a FSP simulation with PEPC, the electrons and ions are first brought into a homogeneous, equilibrium configuration according to the target geometry, density n_0 and electron temperature T_e [5]. For example, a foil target might

be set up with dimensions $L_x L_y L_z = 5 \times 12 \times 12 \quad \mu\text{m}^3$, comprising 3.2×10^6 electrons and ions, giving $\overline{a} = 0.7c/\omega_p$. The electron and ion densities would be set to $n_i = n_e = n_0 = (4 \rightarrow 10)\ n_c$, where n_c is the critical density corresponding to the laser frequency ω, related by: $\omega^2 = 4\pi e^2 n_c/m_e$, where e and m_e are the electronic charge and mass respectively. The ions are given a charge $Z = 1$, mass $m_i = 1836\ m_e$ and initial temperature $T_i = 0$. Unlike in PIC codes, the electron temperature is not artificially constrained to some value (typically several keV) determined by the mesh size, but can be varied from a few 10s of eV upwards to control the initial target resistivity. For lower temperatures, however, the timestep (typically $0.1 - 0.5\ \omega_p^{-1}$) is generally reduced in order to resolve the collision dynamics and ensure reasonable energy conservation.

VI. Laser-Proton Acceleration

One of the hot topics of laser-matter interactions is ion acceleration. Current experiments have made dramatic progress in producing multi-MeV beams of light ions and protons which may eventually be used in tumor therapy. Before this goal can be realised however, a number of challenges have to be overcome regarding the beam quality. Simulations are essential here for exploring novel target configurations in order to provide experimental guidance. Microstructured targets have been proposed to reduce the ion beam energy bandwidth and emittance. Our simulations show that a pure proton microdot target does not by itself result in a quasimonoenergetic proton beam: in face, such a beam can only be produced with a very lightly doped target, in qualitative agreement with one-dimensional theory [7] – Fig.8 The simulations suggest that beam quality in current experiments [24] could be dramatically improved by choosing microdot compositions with a 5–10 times lower proton fraction. Further investigations have further quantified these findings, setting lower limits on the useful proton fraction, beyond which an energy filtering scheme becomes more effective [25].

VII. Heating and Ion Acceleration in Nanostructured Foils

Nanostructure surfaces are especially promising as highly absorbing targets for high-peak-power sub-picosecond laser-matter interaction. Efficient hot electron, fast ion, and thermonuclear neutron production with moderate laser intensity have already been reported, but theoretical investigations on the use of porous targets for these purposes are still scarce. In a recent study using PEPC, a new phenomenon of Coulomb *implosion* has been identified [8]. The implosion effect is caused by hot electrons circulating inside the shells, drawing ions inwards, where they eventually meet in the centre – Fig. 9.

Under the same irradiation conditions, a single shell simply blows apart, and does not exhibit the symmetric collapse observed in the foam lattice simulation here. In this case, some hot electrons circulate inside the shell, but most are dragged outside, leading to a net force on the ions directed radially outwards. (A fully stripped ion shell will, by Gauss' Law, have zero electric field inside). This implies that the laser heating is strongly modified by the regular lattice structure, which in turn radically alters the ion dynamics. These findings have recently been corroborated by 2D PIC simulations in which the electron heating and laser propagation is treated self-consistently with a fully electromagnetic field solver – Fig. 10.

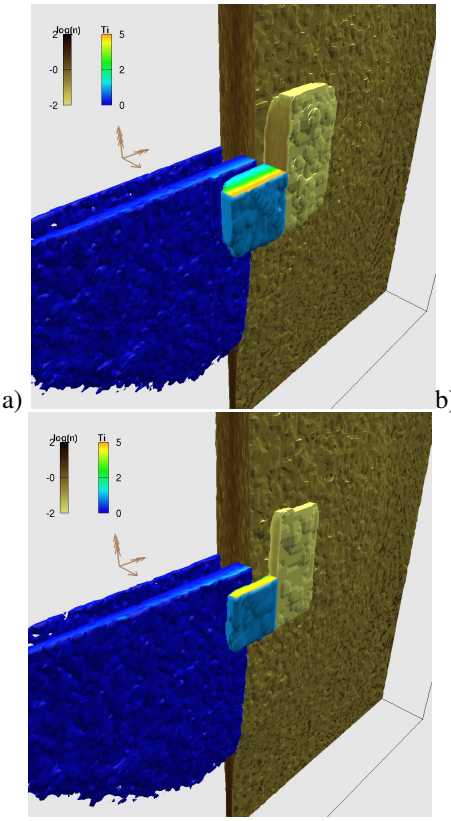


Fig. 8. Laser acceleration of proton microdot (central feature). Energy spread of the protons can be reduced by decreasing the relative proton density – here 50% (a) and 5% (b) respectively.

Should this nano-implosion phenomenon scale to higher, relativistic intensities, it might also have potential as a compact neutron source. A recent comparison between atomic clusters and aerogels [26] suggests that the latter are capable of yielding 10 times as many neutrons for the same laser energy because of the higher kinetic energy imparted to deuterons contained within the aerogel lattice. The present study indicates that density enhancements created by heating *regular* porous lattice structures should result in even higher neutron yields.

VIII. Mesh-free Modelling of Tokamak Edge Physics

The group 'plasma edge simulations for fusion plasmas' in IEF-4 (Prof. Reiter) develops and applies 2- and 3-dimensional computer simulation codes for interpretative and predictive studies of physics close to the plasma container wall (vacuum chamber). This domain is characterised by a complex

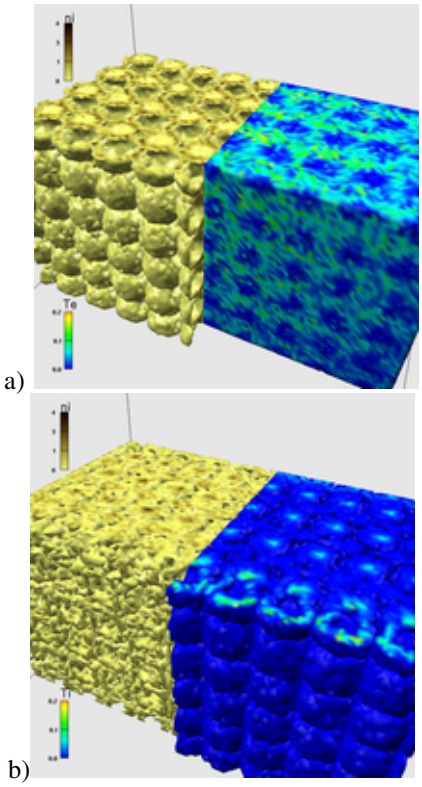


Fig. 9. Coulomb implosion of thin shells initially arranged in a bcc foam matrix with ionized electrons confined to the inner shell surfaces – a). Acceleration of ions off the inner surfaces leads to uniform convergence and peaked ion temperatures (hot spots) at the shell centres – b).

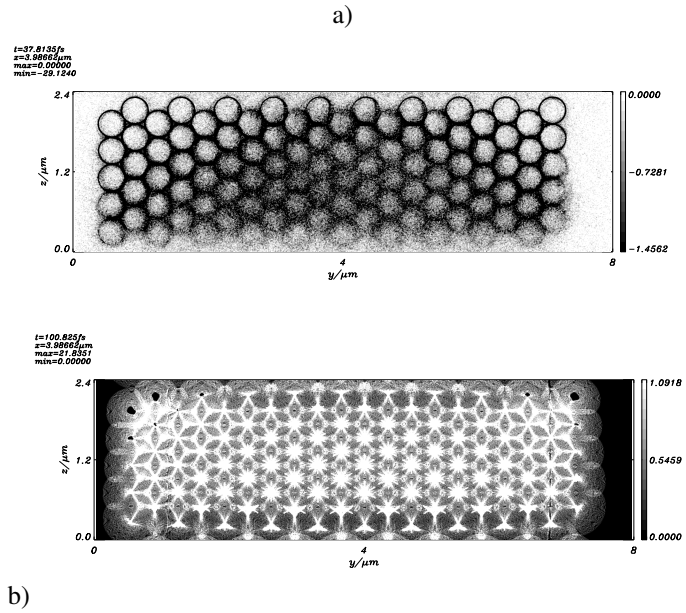


Fig. 10. 2D PIC simulation of foam array irradiated by high intensity pulse (incident from the bottom) with similar parameters to those in the tree-code simulations. a) Electron density at a few fs, b) ion density after 100 fs.

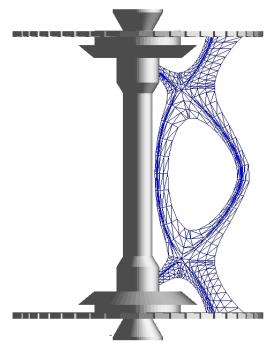


Fig. 11. *Example of a triangular mesh fitted in the vessel of the MAST-tokamak.*

interaction of plasma-chemical and turbulent processes. The models contain both empirical and first-principles based modules. The long term goal of model development is step by step replacement of the *ad hoc* by texitab initio models, aided by increasing HPC resources based on the parallel computing paradigm. The self-consistent electrical field, obtained from the position of charged test particles, or, in fluid approximations, of charged fluid parcels, is one such component.

The mesh-free method offered by the tree code PEPC-B developed at JSC is a promising candidate either to provide self-consistent fields in tandem with existing modelling tools, or as a stand-alone (radiation-free) code for fusion plasma simulations. Currently this is being jointly developed by JSC and IEF-4 to model plasma edge phenomena such as the potentially catastrophic edge-localised modes (ELMs).

In a first step interfaces have been developed to provide PEPC-B with fusion relevant magnetic fields. For this purpose equilibria fields similar to those used in the transport code EIRENE [27] have been introduced. To define the external magnetic field PEPC-B assigns vector data according to a 2d triangular mesh (VIII), assuming toroidal symmetry in the tokamak. This feature in principle enables PEPC-B to run full tokamak core simulations. Next, the field data provided from external sources have to be smoothed and adjusted to guarantee $div(\vec{B}) = 0$.

A further addition is the inclusion of a collision module based on the Monte-Carlo model of Takizuka and Abe [28]. This permits collisions between the injected tracer particles and the plasma backround to be taken into account. An example application is the modelling of impurities such as C^+ ions occurring in fusion plasmas after sputtering or gas puff scenarios in a realistic plasma environment. A current priority is to simulate gas puff experiments recently performed on TEXTOR and to see how these findings scale to larger machines such as ITER.

IX. Darwin Model: Route to Gridless Magnetoinductive Plasma Simulation

A strategic goal with this project is to extend the fast summation algorithm deployed in PEPC to include self-generated

magnetic fields. This 'Darwin' or magnetoinductive approach has been pursued within the particle-in-cell paradigm for some time [29]. A mesh-free Darwin model would open up a large range of new applications of mesh-free plasma simulation, not least in the tokamak modelling scenarios described above, and also in particle beam transport in dense plasmas. The model incorporates slowly varying mag- netic fields by neglecting the transversal part of displacement current in Ampère's law. This transforms the hyperbolic equation system into a set of elliptic equations, which can again by solved by a fast summation algorithm.

$$\begin{aligned}
\phi_i(\boldsymbol{r}_i) &= \sum_{j\neq i} \frac{q_j}{|\boldsymbol{r}_i - \boldsymbol{r}_j|} \\
\boldsymbol{E}_i^l(\boldsymbol{r}_i) &= \sum \frac{q_j \boldsymbol{r}_{ij}}{r_{ij}^3} \\
\boldsymbol{A}_i(\boldsymbol{r}_i) &= \frac{1}{2c}\sum \frac{q_j \boldsymbol{v}_j}{r_{ij}} + \frac{1}{2c}\sum \frac{(q_j \boldsymbol{v}_j \cdot \boldsymbol{r}_{ij})\boldsymbol{r}_{ij}}{r_{ij}^3} \qquad (12) \\
\boldsymbol{B}_i(\boldsymbol{r}_i) &= \nabla\times\boldsymbol{A}_i = \frac{1}{c}\sum \frac{q_j \boldsymbol{v}_j \times \boldsymbol{r}_{ij}}{r_{ij}^3} \\
\boldsymbol{E}_i^t(\boldsymbol{r}_i) &= -\frac{1}{2c^2}\sum \frac{q_j \dot{\boldsymbol{v}}_j}{r_{ij}} - \frac{1}{2c^2}\sum \frac{(q_j \dot{\boldsymbol{v}}_j \cdot \boldsymbol{r}_{ij})\boldsymbol{r}_{ij}}{r_{ij}^3}
\end{aligned}$$

On the other hand, numerical difficulties arising from this approximation have already been reported by previous authors who implemented the Darwin model within PIC or Vlasov codes. The standard scheme known from the fully electromagnetic codes used for the calculation of time derivative of the vector potential causes a violent numerical instability destroying the whole run in a few time-steps. One of the possible solutions to this problem is to express the quantities in terms of Hamiltonian generalized variables, which avoids the time derivative of the vector potential in the the equation of motion.

Our magnetoinductive model employs a multipole expansion of the Darwin field equation, modified to account for finite-sized particles and evaluated within the PEPC tree algorithm framework. First tests of this model have been made on charged particle beam evolution in vacuum. Details will be given in a separate publication.

X. Conclusion

To summarize, we have described recent activities concerning mesh-free plasma modelling at JSC. Most of this work involves application and further development of the mesh-free code PEPC, but recently we have also started using fully electromagnetic PIC codes to model laser-plasma interaction processes. Future work will include rigorous comparisons between the PIC and mesh-free approaches, in order to more clearly identify the advantages of the latter. The mesh-free Darwin or magnetoinductive approach offers completely new possibilities in many areas of plama physics from magnetic fusion to space physics. A major challenge for the usage of the mesh-free technique on contemporary HPC systems is to improve its parallel scalability. Efforts are underway to improve the current limit of 8192 cores on BG/P to at least 64k cores.

Acknowledgment

This work was supported by the Alliance Program of the Helmholtz Association (HA216/EMMI). Simulations were performed with computing resources granted by the VSR of the Research Centre Jülich unter project JZAM04.

References

[1] S. Pfalzner and P. Gibbon, "Direct calculation of inverse-bremsstrahlung absorption in strongly coupled, nonlinearly driven laser plasmas." *Phys. Rev. E*, vol. 57, pp. 4698–4705, 1998.

[2] U. Saalmann and J.-M. Rost, "Ionization of clusters in intense laser pulses through collective electron dynamics," *Phys. Rev. Lett.*, vol. 91, p. 223401, 2003.

[3] A. J. Christieb, R. Krasny, J. P. Verboncoeur, J. W. Emhoff, and I. D. Boyd, "Grid-free plasma simulation techniques," *IEEE Trans. Plasma Sci.*, vol. 34, pp. 149–165, 2006.

[4] K. Matyash, R. Schneider, R. Sydora, and F. Taccogna, "Application of a grid-free kinetic model to the collsionless sheath," *Contrib. Plasma Phys.*, vol. 48, pp. 116–120, 2008.

[5] P. Gibbon, F. N. Beg, R. G. Evans, E. L. Clark, , and M. Zepf, "Tree code simulations of proton acceleration from laser-irradiated wire targets," *Phys. Plasmas*, vol. 11, pp. 4032–4040, 2004.

[6] P. Gibbon, "Resistively enhanced proton acceleration via high-intensity laser interactions with cold foil targets," *Phys. Rev. E*, vol. 72, no. 026411, pp. 1–8, 2005.

[7] A. P. L. Robinson and P. Gibbon, "Production of proton beams with narrow-band energy spectra from laser-irradiated ultrathin foils," *Phys. Rev. E*, vol. 75, no. 1, p. 015401, 2007.

[8] P. Gibbon and O. N. Rosmej, "Stability of nanostructure targets irradiated by high intensity laser pulses," *Plasma Phys. Contr. Fus.*, vol. 49, pp. 1873–1883, 2007.

[9] J. Barnes and P. Hut, "A hierarchical $O(N \log N)$ force-calculation algorithm," *Nature*, vol. 324, pp. 446–449, 1986.

[10] S. Pfalzner and P. Gibbon, *Many Body Tree Methods in Physics*. New York: Cambridge University Press, 1996, iSBN 0-521-49564-4 (hardback); ISBN 0-521-01916-8 (paperback).

[11] U. Becciani, V. Antonuccio-Delogu, and M. Gambera, "A modified parallel tree code for n-body simulation of the large-scale structure of the universe," *J. Comp. Phys.*, vol. 163, pp. 118–132, 2000.

[12] J. Dubinski, "A parallel tree code," *New Astronomy*, vol. 1, pp. 133–147, 1996.

[13] H. Yahagi, M. Mori, and Y. Yoshii, "The forest method as a new parallel tree method with the sectional voronoi tessellation," *Ap. J. Supp.*, vol. 124, pp. 1–9, 1999.

[14] M. S. Warren and J. K. Salmon, "A parallel hashed oct-tree n-body algorithm," in *Supercomputing '93*. Los Alamitos: IEEE Comp. Soc., 1993, pp. 12–21.

[15] ——, "A portable parallel particle program," *Comp. Phys. Commun.*, vol. 87, no. 266–290, 1995.

[16] H. Shi and J. Schaeffer, "Parallel sorting by regular sampling," *J. Par. Dist. Comp.*, vol. 14, pp. 361–372, 1992.

[17] S. Pfalzner and P. Gibbon, "A hierarchical tree code for dense plasma simulation," *Comp. Phys. Commun.*, vol. 79, pp. 24–38, 1994.

[18] R. Speck, P. Gibbon, and M. Hoffmann, "Efficiency and scalability of the paralle Barnes-Hut tree code PEPC," in *Parallel Computing (ParCo)*, 2009.

[19] A. B. Langdon and C. K. Birdsall, "Theory of plasma simulation using finite-size particles," *Phys. Fluids*, vol. 13, pp. 2115–2122, 1970.

[20] H. Okuda and C. K. Birdsall, "Collisions in a plasma of finite-size particles," *Phys. Fluids*, vol. 13, pp. 2123–2134, 1970.

[21] J. D. Huba, *NRL Plasma Formulary*. Washington DC: Naval Research Laboratory, 1994.
[22] W. L. Kruer, *The Physics of Laser Plasma Interactions*. New York: Addison-Wesley, 1988.
[23] C. K. Birdsall and A. B. Langdon, *Plasma Physics via Computer Simulation*. New York: McGraw-Hill, 1985.
[24] S. M. Pfotenhauer, O. Jäckel, A. Sachtleben, J. Polz, W. Ziegler, H.-P. Schlenvoigt, K.-U. Amthor, M. C. Kaluza, K. W. D. Ledingham, R. Sauerbrey, P. Gibbon, A. P. L. Robinson, and H. Schwoerer, "Spectral shaping of laser generated proton beams," *New J. Phys.*, vol. 10, p. 033034, 2008.
[25] A. P. L. Robinson, P. Gibbon, M. Zepf, S. Kar, R. G. Evans, and C. Bellei, "Relativistically correct hole-boring and ion acceleration by circularly polarized laser pulses," *Plasma Phys. and Cont. Fus.*, vol. 51, p. 024004, 2009.
[26] V. P. Krainov and M. B. Smirnov, "Nuclear fusion induced by a super-intense ultrashort laser pulse in a deuterated glass aerogel," *JETP*, vol. 93, pp. 485–490, 2001.
[27] D. Reiter, M. Baelsmans, and P.Börner, "The eirene and b2-eirene codes," *Fusion Science and Technology*, vol. 47, p. 172, 2005.
[28] T. Takizuka and H. Abe, "A binary collision model for plasma simulation with a particle code," *J. Comp. Phys.*, 1977.
[29] D. W. Hewett, "Elimination of electromagnetic radiation in plasma simulation: the Darwin or magnetoinductive approximation," *Space. Sci. Rev.*, vol. 42, pp. 29–40, 1985.