Domain decomposition performance on ELMFIRE plasma simulation code

Francisco Ogando[1,2], **S. Janhunen**[2]. **J. Heikkinen**[3], **T. Kiviniemi**[2], **S. Leerink**[2] *and* **M. Nora**[2]

1) *UNED, Spain*
2) *TKK-EURATOM Tekes Association, Finland*
3) *VTT-EURATOM Tekes Association, Finland*

ABSTRACT: *ELMFIRE is a gyrokinetic full-f plasma simulation code, based on a particle-in-cell algorithm and parallelized with MPI. The coupled calculation of electrostatic field and plasma particle dynamics leads to huge memory demands that can be splitted among processors by the introduction of Domain Decomposition. This technique modifies the MPI-process topology with influence to collective operations. This paper shows the performance change due to the new algorithm, as well as the extension of capabilities in the CSC louhi cluster.*

KEYWORDS: Gyrokinetics, domain decomposition, plasma simulation

1 Introduction

Plasma turbulence has shown to be responsible for the so called anomalous transport that compromises the magnetic confinement of fusion plasmas. This anomalous transport takes values hundreds of times higher than predicted by the neoclassical theory. The study of the dynamics of turbulent plasmas has become a key issue in order to control the generation of turbulence in a plasma and its influence on the degradation of confinement.

With that purpose the ELMFIRE code was developed as a joint project between VTT and TKK. It is a Monte Carlo full-f nonlinear gyrokinetic plasma simulation code in five-dimensional toroidal geometry. The code includes several physical features like multiple different species, collisions between them and neutral particle ionization. The movement equations take into consideration the existing drifts in tokamak plasmas.

The gyrokinetic model simplifies the description of motion of charged particle motion by averaging it over the quick gyration around the magnetic fields lines. This produces a reduction of phase space to five dimensions and thus reducing the computational requirements for its solving. The full-f formulation of the problem includes solving the whole five dimensional particle distributions, opposite to the delta-f technique, which for further simplification considers perturbations from local Maxwellian distributions. However the full-f technique included in ELMFIRE produces accurate results even in the presence of strong gradients or perturbations in the plasma.

2 ELMFIRE overview

ELMFIRE[2] is a full-f gyrokinetic plasma simulation code developed as a cooperation between Helsinki University of Technology (TKK) and VTT Technical Research Centre of Finland. The code is based on a particle-in-cell (PIC) algorithm, which calculates the dynamics of a set (10^7-10^9) of markers, each marker corresponding to about 10^{10} particle gyrocenters. The markers are followed in a toroidal geometry inside a given magnetic field background and a self-consistent electric field. Since ELMFIRE is a PIC code, from now on markers will be referred to as particles, taking into account the difference with actual plasma particles. Particles are associated with a selected main ionic species (normally hydrogen or deuterium), electrons and optionally higher-Z impurities (e.g. Oxygen). The treatment of electrons may be either using the PIC algorithm or with an adiabatic model, which assumes the electron density n_e = $<n_i>$*exp(eΦ/kT), $<n_i>$ being a local average electron density. Collisions between species may be included in the PIC algorithm. As the code is computationally highly demanding in both memory and CPU-time usage, the code has been parallelized using the MPI[3] (Message Passing Interface) standard.

ELMFIRE calculates plasma dynamics with a quasineutrality condition that forces ion and electron charges to neutralize each other every time step over all system cells. This is justified considering the plasma frequency and characteristic times. Fields and particles are evolved consistently in time so that plasma neutrality is kept at any moment. The calculation of motion, decomposed as mentioned in the introduction, is performed partially explicitly and partially implicitly (consistently with the electric field advanced in time), depending on the sensitivity of the given motion on the equation. The electrostatic field is then calculated using the gyrokinetic Poisson equation, considering implicitly the previously mentioned drifts

$$\nabla^2 \Phi + \frac{q^2}{mB\varepsilon_0} \int \left[(\Phi - \langle \Phi \rangle) \frac{\partial \langle f \rangle}{\partial \mu} - \frac{m}{q\Omega} \langle f \rangle \nabla_p^2 \langle \Phi \rangle \right] dv = \frac{-1}{\varepsilon_0} (q n_i - e n_e)$$

where **Φ** stands for the electrostatic potential, *q*, *m* the charge and mass of the ionic species, **B** the magnetic field, ε_0 the vacuum susceptibility, <·> a gyroaveraging operatorwith Larmor radius, *f* the particle distribution function, **Ω** the ion gyrofrequency and $n_{i,e}$ charge densities. As the full-f algorithm follows accurately arbitrarily strong plasma perturbations, the particle distribution function *f* has to be sampled every time step in order to compute the coefficients of the gyrokinetic Poisson equation. This procedure is different from the delta-f approach, which requires the construction of the Poisson equation just once from the assumed particle background which does not evolve in time. This is why delta-f codes never had to solve the problem which is the motivation of this article. It is the construction and resolution of this implicit problem that leads to a numerically heavy process of collecting data from particles placed almost randomly across the toroidal domain. The optimization of this collection process is the target of the work presented in this article.

2.1 Calculation sequence

A simulation run in ELMFIRE starts with the initialization of the particle coordinates according to initial temperature and density profiles, which may evolve in time. Using the gyrokinetic model, five phase-space coordinates are stored for each particle. Particles optionally may be unequally weighed (equivalent to different number of actual plasma particles), in which case it is neccessary to store the particle weight as a sixth parameter. Particles can be intuitively initialized using a given density profile and local temperature Maxwellian velocity distributions consistent with a given temperature profile.

After initialization, and as a first step in every time step, particles are propagated explicitly using a Runge-Kutta algorithm but without consideration of either ion polarization or electron parallel acceleration. The electric field is considered to be zero before the first time step. This particle propagation process demands most of the CPU-time during a run, effectively limiting the number of particles a single MPI process may treat in order to achieve reasonable calculation times. Using current processor technology a single processor may well handle around 10^6 particles within a reasonable time. This amount of particle coordinates presents no problem regarding memory consumption.

However the number of particles has to be selected considering the acceptable level of numerical noise which, as it was mentioned before, is considerably higher than with the delta-f method. This value depends on the problem itself but 2000 particles per cell is a good estimate.

An optional collision operator calculates the effect of binary collisions on particle positions. Particle collisions produce a significant effect on gyrocenter positions, contributing to heat transport.

Once all particles have been partially propagated, without the consideration of either ion polarization drift or electron parallel acceleration, the electric field is calculated from the estimated final positions of the particles. This estimation is made consistently with the electrostatic field under calculation.
The electrostatic potential is calculated by solving the gyrokinetic Poisson equation on a 3D grid. Cell ordering in the matrix is done along poloidal, radial, and toroidal directions. The grid size is directly related to global memory requirements, hence available memory effectively limits the grid size that can be used. The refinement in the toroidal direction is lower than the refinements in the other directions since the cells are field-oriented, following the particle streaming direction. The particle charge density is projected onto the grid to calculate the rhs source term for the Poisson equation.

This charge deposition procedure can be time time-consuming, but it is performed simultaneously to the construction of the density influence DI matrix, using common interpolated values. The projection of densities given the gyrocenter position is performed by sampling four positions of the gyro orbit, as proposed by Lee [1]. This process does not pose a limit, since the array of cell charges is much smaller than the DI matrix.

The electrostatic potential to be solved will influence the final position of the particles, and therefore also the densities on the RHS of the equation. This dependence is partially linearized in order to move some of these terms to the left hand side. Effectively (by neglecting the effect of the Laplace operator in the equation because of the quasineutrality condition) the advanced E-field is calculated so that the plasma final state is electrically neutral.

The construction of the density influence matrix representing the influence of advanced E-field on the cell densities is a CPU-time and memory demanding process. The matrix is constructed from every particle, whose position can in principle be considered random inside the calculation system. The ratio of particles to cells is directly related to the numerical noise of the PIC algorithm, and is kept at values around 2000. This means that the matrix elements are filled in random order and with many contributions to the same element. Even particles stored and handled by different MPI processes may contribute to the same matrix element. These contributions have to be interchanged between the processes and the matrix has to be split in order to let a parallel algebra package invert it.

The resulting matrix is sparse since it represents the effect of cell-averaged potential values onto the cell-averaged charge density. The charge density depends on final particle positions, which are influenced by both ion polarization drift and electron acceleration. Both displacements have a spatially limited dependence on the E-field, that is, the charge density has spatially limited dependence on potential values. This locality has a characteristic length of several local ion gyration (Larmor) radii. Since the numerical implementation of the gyrokinetic theory becomes easily impractical with cell sizes lower than the local Larmor radius, we can estimate the $\Phi \rightarrow n$ interaction locality within several (~10) cells around in the poloidal plane, and one in the toroidal direction. Due to the cell ordering this matrix structure produces a multidiagonal sparse matrix, where diagonals correspond to non-zero interactions with neighboring poloidal, radial and toroidal cells. Boundary conditions can be set to either a fixed potential or a fixed electric field.

This whole matrix has to be inverted in order to solve the electrostatic potential. Some codes reduce the problem to a bidimensional one, since the fluctuations along the field lines are much lower than across them. However, since those toroidal differences have a strong influence on electron parallel dynamics, ELMFIRE solves the complete 3D problem for every timestep, with implicit consideration of electron parallel acceleration.

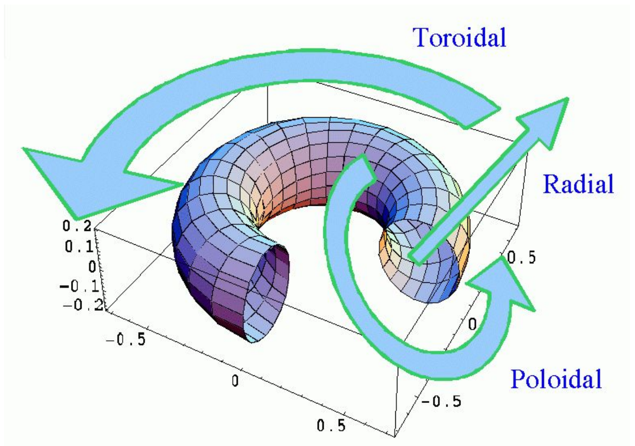

Figure 1: Toroidal geometry description

3 Construction of the linear problem matrix

As it has been already mentioned, the matrix associated to the gyrokinetic Poisson equation is not dense, but may contain around 500 non-zero values in every row. This matrix has to be constructed from the particle data at every timestep, and every particle produces contributions to several (~16) matrix elements. That means that most matrix elements will be updated many times during matrix construction, which justifies the need of a special storage structure optimized for element location and updating. The used sparse algebra packages (PESSL, PETSc) are effective for adding matrix elements, but the update of an already included element is not processed with an acceptable performance.

3.1 Block-based storage

The first implementation of a matrix structure made use of the physics involved in the matrix construction. The influence of the potential on some cells on a certain cell density (that is the nonzero coefficients) comes from the implementation of the following physical processes.

- The implicit consideration of the ionic polarization drift. This process takes into account that actual particles rotate around the gyrocenter with a approximately circular trajectory perpendicular to the magnetic field line with Larmor radius. This efectively relates every cell's density to the the potential of cells lying on the same toroidal plane at a distance lower than twice the Larmor radius which can be calculated analytically. This distance may correspond to up to 15 cell lenghts. As the polarization process require a big amount of cells for its stencil, it is considered that the particle is situated toroidally in the coordinate of the center of the cell, so that this process is restricted to a given plane, not coupling different toroidal planes (which will take much more memory). This is an approximation which has so far proven not to invalidate the solution.

- The implicit consideration of electron acceleration component parallel to the magnetic field line involves calculating a derivative following that direction. This makes use of the neighboring cells along the toroidal direction.

Coming from these two descriptions one can see that the non-zero matrix coefficients will come from surrounding cells, but located in a somehow special structure of diagonals which may add up to 500 or more. For this reason the matrix structure was constructed following the actual cell distribution around every given cell, with the pattern shown in figure 2.

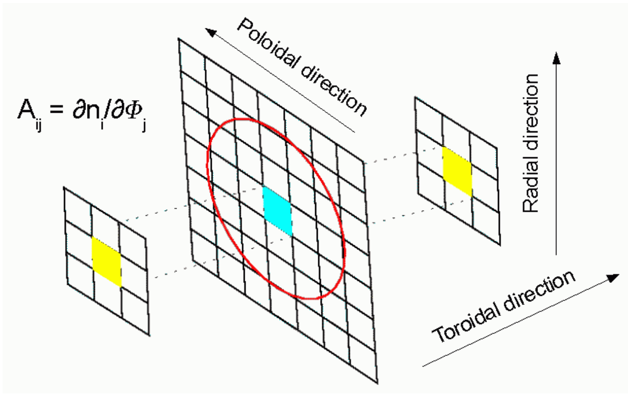

Figure 2: Non-zero coefficient stencil around a given cell

This figure represents, for a given cell density under consideration (that is, a matrix row), three toroidal planes with the estimated non-zero matrix coefficients in each of them. The central plane contains in the center the cell corresponding to the matrix row. Surrounding this cell in the central plane there is a block of cells whose potential may be linked to the row-cell density through the explained ion polarization process.

The neighboring planes, containing much less cells (precisely a block of 9) include the effect of derivatives along the toroidal direction, which are implicitly calculated with a two-point stencil. This stencil may take the forward and central planes, or the central and backward planes, depending on the particle position relative to the central plane.

This pattern of matrix row (which may contain over 500 cells) is repeated for every system cell in all processes, since any given particle in every process may fill one of the estimated non-zero positions. The search operation is very fast, since it is based on matrix coordinated for all cells. Once every process has gathered its contribution to the matrix, the resulting matrix is summed and splitted among processes using a single `MPI_REDUCE_SCATTER` instruction which is effective but requires high network usage since the matrix storage may be about 500 MB in size.

This method has two main disadvantages:

1. The Larmor radius of gyration, which is directly related to the block sizes, is different for different radial positions and/or particle velocity. In order to maintain the matricial search structure a single conservative radius has to be selected, producing efectively an overestimation of non-zero values. During a normal run, about 25% of the estimated non-zeros are actually filled.
2. Since particles are distributed almost randomly in the whole system for every process, all of them have to keep in memory the estimated non-zero positions for the whole torus.

Because of the previous undesirable features of this scheme, there is a considerable amount of misused memory, and the system behaves poorly when scaling to higher number of processors, since in any case all of them will have to maintain data from the whole system.

During many years, the supercomputers availability to the ELMFIRE group was moderate enough so that scalability was not a big issue and memory consumption was not a bottleneck for the simulations performed. However, the available computing power at CSC has been growing exponentially and the access to multihundred or even thousand processor runs have introduced the need for finer optimization of the code.

3.2 Domain decomposition

Mainly trying to overcome the second undesirable feature of the block-based storage (detailed in previous section) the domain decomposition method was studied for implementation into ELMFIRE. Through this method, the system is somehow divided into subsystems (a.k.a. domains) in such way that the processes of a certain domain only keeps data about itself.

The first consideration in order to implement domain decomposition is the kind of system division to consider. The main concern for this task lies in the domain boundary conditions, that is, the data that a certain domain needs to know about the surrounding ones. As it is shown in figure 2, the non-zero values of the matrix span several cells in the poloidal and radial directions, while only one cell along the toroidal. This is the reason why the toroidal onedimensional decomposition has been implemented as a first option.

If somehow all the particles handled by a certain process are always kept in the domain it belongs to, the matrix coefficients produced by those particles can be also bounded to certain matrix rows and columns. As the matrix A_{ij} contains the effect of potential values of a cell *j* on the density of a cell *i*, this text is going to follow the notation of i-cells and j-cells referring to those defining matrix row and column respectively. An i-cell is a cell whose density is going to be affected by a certain particle (corresponding to a matrix row), while a j-cell (related to a certain i-cell) is a cell, whose potential is going to affect the particle trajectory in a way that will affect the density of the corresponding i-cell (corresponding to a matrix coefficient A_{ij}).

The main advantage of domain decomposition is that, while the traditional storage method forced to store data from all i-cells, the process of keeping all handled particles in a certain domain will restrict the space of possible i-cells affected by those particles.

Considering again the physical processes described in section 3.1, both the polarization drift and the electron parallel acceleration will require the already described cell stencil for every considered cell. The main difference now is that there is no need anymore to keep that stencil for all the cells of the system.

Since the polarization drift is considered assuming the particle centered in the domain cell, it will affect only i-cells belonging to that domain. That is, the polarization process will only affect i-cells lying in the given domain.

The implicit treatment electron parallel acceleration does not consider that the particle is centered toroidally in the cell, so a given particle may produce density variations on cells of the neighboring domains (part of the particle may lay outside its domain). This is the most accurate consideration that will (in the future) be applied to the polarization drift, and in this case it is enforced since the stencil for this electron process is much smaller than the one of polarization drift. As it was previously detailed, the stencil is only the needed for calculating a derivative along the toroidal direction in a 3D grid.

The stencil resulting of taking into account both processes will contain the original one plus the possible effect of a particle on the neighboring domain's density through the limited stencil of electron parallel acceleration. The overall resulting stencil is represented in figure 3.

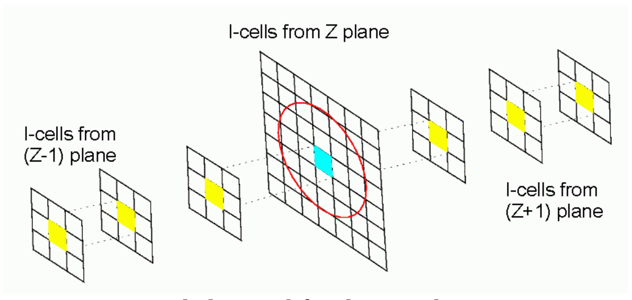

Figure 3: Extended stencil for domain decomposition

Despite the stencil being somehow bigger, it is only stored in every process for the group of cells belonging to its domain. That is, assuming an optimal division of one toroidal cell per domain, the reduction of stored matrix positions may be several times lower. The search procedure is still based on matricial lookup, which is the fastest possible. Now there is, however an extra lookup step, to decide to what of the i-cell blocks the coefficient belongs. Basically the matrix construction time is very similar to the original one.

However, the combination of all matrix components is different from the original algorithm, since the stored parts of the matrix in every process may be different, and in some cases overlapping. The combination procedure in decomposed into two different operations.

1. Inside every domain processes are internally ordered. Two processes are called correspondent in they share the same internal order in two different domains. The first operation sends and receives data from the correspondent processes of neighboring domains. The exchanged data relates to the stored i-cells that do no belong the the proper domain but to the neighboring ones. After this exchange operation (which is simultaneously done through a `MPI_SENDRECV` call) all processes store data related only to its own domain.
2. At this stage all processes belonging to the same domain store data related to the same cells (all the cells of their domain), so that they now add up all local contributions inside every domain and split it accordingly to a matrix row division to all processes. This operation is simultaneously done by means of a `MPI_REDUCE_SCATTER` call, but using a different communicator in every domain.

After these two operation, every process holds exactly the same data that it would have got using the original algorithm, being ready to feed this data to the proper algebra package (PESSL, PETSc...).

4 Test performance

In order to test the performance of the new domain decomposition method some representative runs have been performed in louhi, the most powerful machine of CSC. Louhi is a Cray-XT4 supercomputer consisting of 1012 nodes of 1 dual-core AMD Opteron processors. The overall sustained performance is around 10 Tflops.

The sample run is representative in the sense that it is taken from a series of test runs on the DIII-D tokamak, that is, a real tokamak machine. The simulation parameters, however, have been chosen to be favorable to the domain decomposition method. The benefits from domain decomposition are enhanced in the following cases:

- When the linear problem matrix takes much memory space. That is, when the potential grid has many cells.
- When the system is solved with high toroidal resolution, that is when the system can be divided in as many domains as possible.

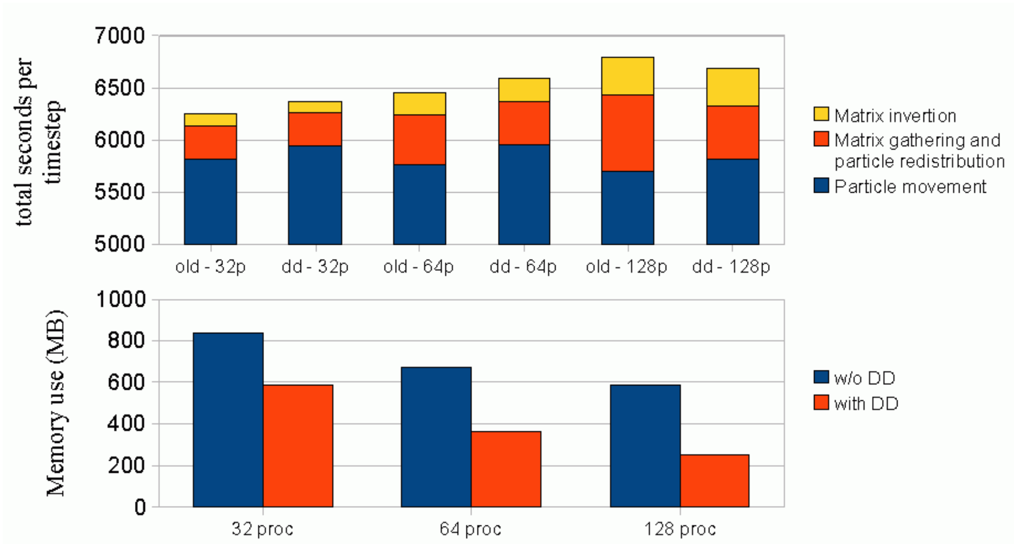

Figure 4: Sample test at louhi: comparison results for different processor number

The simulation makes use of 107 million particles and 160 thousand cells in the potential grid with 16 toroidal levels. Several runs have been performed with varying number of processors, in the sense of a scalability hard test. The results of the test are shown in figure 4.

The upper histograms show, for a timestep, the total time (cpu*s) comparing the traditional method (named as “old” and the domain decomposition (named as “dd”), with a variety of processor number in the range 32-128. Please note that the figure has not a lower range of zero in the Y axis, in order to get better detail of the subdivision in different operations.

The lower part of the figure shows the maximum resident memory consumption, measured in the code at runtime for several code operations. Maximum memory use takes place for with the traditional method for matrix construction, while for domain decomposition it is during the process of finding what particles have to be moved to another process (and the matrix storage is active at the same time). Two main conclusions can be extracted from the graph:

The domain decomposition method do not introduce significative overheads regarding CPU-time.

The maximum memory use during code operation is reduced strongly, even more in the case with high processor number, when less particles are assigned to each process (and most memory is taken by the matrix).

For the case of 128 processors (where still there are many particles handled by each process) the memory use is less than half the original one.

5 Conclusions

A well established method like the domain decomposition has been successfully developed and implemented into the ELMFIRE code for the GK-Poisson matrix construction. This method has shown to reduce the memory demands to less than 50% of the original values, while not introducing overheads regarding computing time. This method will provide the code an immediate extension of capabilities, specially important in supercomputers designed to hold limited memory per node (which is actually the current trend). Also specific parallel-mode analysis, with high resolution along the toroidal direction, are now extended by a much higher factor, depending on the number of toroidal domains that can be defined in the system.

Future work on this matter include the study of domain decomposition along the other directions, and the joint development of storage structure which minimizes the allocation of null elements of the matrix [4].

6 Acknowledgements

The facilities of CSC (Finnish IT Center for Science) have been used for this work. This project has received funding from the European Commision and support from the CIEMAT-EURATOM association.

7 References

[1] W.W. Lee, Gyrokinetic particle simulation model, J. Comput. Phys **72** (1987) 243-269

[2] J.A. Heikkinen, S.J. Janhunen, T.P. Kiviniemi and F. Ogando, *Full-f gyrokinetic method for particle simulation of tokamak transport*, Journal Comput. Phys. **227** (2008) 5582-5609.

[3] William Gropp, Ewing Lusk and Anthony Skjellum, Using MPI: Portable Parallel Programming with the Message-Passing Interface MIT Press 1994

[4] A. Signell, et al, Scalable plasma simulation with ELMFIRE using efficient data structures for process communicationAccepted for publication in CPC (2008).