A PARALLEL PIC CODE FOR A HALL THRUSTER SIMULATION

I. Vieira, L. Gargaté, R. Fonseca, L. Silva and J. T. Mendonça

GoLP – Centro de Física de Plasmas
Instituto Superior Técnico
Av. Rovisco Pais, 1049-001
Lisboa -Portugal

Abstract
Fluid codes are widely used to model Hall thrusters because of their simplicity and low computational demands. However, they have not been successful in describing all the physics of Hall thrusters. On the other hand, PIC codes (Particle In Cell) are powerful for simulating microscopic phenomena. Their drawback is the requirement in the number of particles in order to achieve good results. This implies stronger computational resources and longer runs.
We have been developing a parallel PIC code for simulating a Hall thruster, able to run on a cluster of computers. This code is able to simulate a two-dimensional model of a Hall thruster with more than 100 millions particles and 1000 x 1000 cells on the cluster installed at IST. This paper describes the architecture of the code and its implementation on our cluster.

1. Introduction

Hall thruster technology is becoming more and more interesting because of the fuel mass saving that is possible to achieve in comparison with conventional chemical thrusters. However, although several thrusters have flown with success, not all the physics involved is yet completely understood. Several fluid codes have been developed but they have been always unable to describe all the phenomena. On the other hand PIC (Particle In Cell) codes are very useful for simulating microscopic effects. Some of them have been developed for Hall Thruster simulation [5] but the need of high computational resources is their main drawback. One has also to care about the huge difference between thruster discharge chamber plasma density and plume density. Full simulation of both zones needs substantial computational resources or an irregular mesh or even dynamic load balancing which requires more complicated codes.
At GoLP, the EP2 cluster of 40 Macintosh processors allows parallel processing of very large (up to 200 millions particles and 4 millions cells). We have been developing a fully kinetic two-dimensional Hall thruster code in order to understand the hidden physics and to test thruster performances optimisation.
This paper describes the selected models and the implemented algorithms, designed in a way to provide the means to perform parallel simulation of a Hall thruster on our cluster.

2. Models

The physical problem we are trying to model is the plume divergence of a Hall Thruster and the possibility to deflect this plume in some determined angle as explained in [1]. The simulation program is developed in C++ object oriented.
The simulation program is based on the electrostatic Particle In Cell (PIC) paradigm. We have to solve Poisson's equation on a discrete 2 dimensional mesh, and push both ions and electrons on a self-consisting fashion for such codes. Both the field solver and the particle dynamics are constrained by unusual boundary conditions.
Poisson's equation can be solved on a two dimensional mesh by using the finite difference method. This is done using the numerical algorithms described below. The field solver includes both the external electric field applied between the hollow cathode and the anode on a hall thruster, and the self consistent field due to the particles positions in the simulation zone.

The magnetic field present in a hall thruster, mostly radial, is also modeled. (Its simulation is done in exactly the same way as for the electrical field except for that it is not self-consistent with the currents within the plasma as it is expected that these currents are small and do not alter the externally imposed magnetic field).
The species present in the simulation of the system described in [1] should be neutral Xenon – a Xenon background fluid, ions – Xenon ions (with charge + |e| or + 2 |e| since these are the most common species present in the plasma) and electrons. The first species, the neutrals, is treated as a background fluid. This is acceptable, as the neutrals do not feel the influence of the fields present and do not contribute for the overall thruster force.
Both ions and electrons are simulated using PIC techniques. Namely, each particle stands for a pre-determined number of real particles, or super particle [2]. Particles, also have to obey strict boundary conditions. A typical simulation box for the problem we are solving can be seen on figure 1. It can be observed that only half of the hall thruster is present. This is because a cylindrical symmetry is assumed.

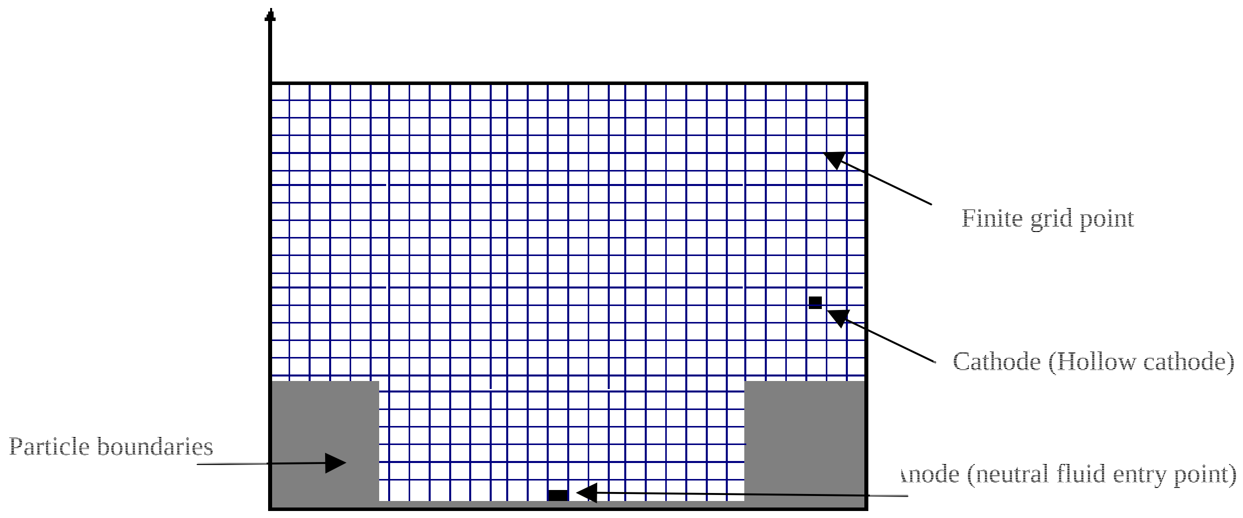

Fig. 1 – Simulation area for a Hall thruster

2.1 Particle Dynamics

The simulation includes several features that usually occur in a Hall thruster. The dynamics of the particles is evolved by pushing the particles using a simple leapfrog scheme [2]. After being pushed, boundary conditions are tested, and ionization processes which enable us to create new ion–electron pairs are also done. Finally, there is also a possibility to inject electrons from the hollow cathode.
The physical model used in pushing the particles is non-relativistic, a reasonable approximation since the thermal velocities of ions and electrons and therefore their velocities distribution is far from the light velocity. A finite difference method is used to numerically compute the velocities and positions of each individual particle. The computation of position and velocity is through the leap frog (position and velocity calculation offset by dt/2), and the field solver is time centered with the position.
Furthermore it should be mentioned that the implementation of this scheme is done to allow for change of push method without any effort. Also, the push method is applied to ions and electrons with different time steps in order to reduce simulation time – this is due, of course, to the enormous difference of masses between Xenon ions and electrons, a technique known as particle subcycling [10]

2.2 Collisions

Two types of collisions have been included: electron-neutral collisions and collisions with the boundaries. These last collisions have been included for both particle species: electrons and ions. Different kinds of models are possible: 1) The wall reflects particles as a perfect mirror; 2) The wall reflects particles as a non perfect mirror (particles loose some of their energy); 3) The wall reflects particles as a diffusion surface

(particles loose some of their energy and are reflected on random direction); 4) The wall reflects particles as an absorbing surface (particles loose most of their energy and a secondary electron is created from a thermal bath).
Although the code design allows the implementation of all these models, the second model is the one that has been implemented so far.
Regarding the collisions of electrons with neutrals, we have used a Monte Carlo algorithm that is updated at each iteration, in order to provide realistic statistical collisions. Electrons loose energy after the collision and a secondary electron is created. Ionization is accompanied by the creation of an ion.

2.3 Charge Deposition and Force interpolation

In order to avoid numerical problems (e.g. non-physical self-heating, non-physical noise), the same algorithm for the charge deposition into the mesh and for the calculation of the force into the super-particle has been used. The NGP (Nearest Grid Point) method is used, which is appropriate for fast computation. Although other algorithms are possible to implement, we will first assess by means of an evaluation campaign, if the method used is suitable for Hall-thruster simulation.

2.4 Electric and Magnetic Field Solver

The field solvers have been developed in order to easily allow the inclusion of electromagnetic boundary conditions. Only electrostatic and magnetostatic fields are considered but the last one is not self-consistent. This means that the magnetic field has to be calculated only at the beginning of the simulation for the whole mesh. An interpolation is performed for calculating the magnetic force on each particle each iteration.
The finite difference method is used for both field solvers. Poisson equation is used on the whole mesh and a LU decomposition [11] is performed in order to allow the resolution of the “super” LU matrices. Regarding the Electrostatic field, the LU resolution has to be done each time the self-consistent perturbations are large enough. However, we should note that the LU decomposition is performed only once at the beginning of the simulation. This strategy is very welcome because the LU decomposition is more time consuming than the field solver.

3. Code architecture and implementation

First of all, it should be stated that this simulation is a fully kinetic particle in cell 2D 3V parallel code. This means that the simulation we are implementing uses a simulation box in two dimensions – cylindrical coordinates, and the three components of the velocity vector in order to simulate the particle motion.
It is a parallel code in the sense that it is designed from the beginning to be prepared to run on an system–independent cluster of distributed memory machines or even on a shared memory computer. This is achieved using the Message Passing Interface (MPI) [3]. The number of machines that the program can run on is also arbitrary – it can run either on a laptop PC or on a cluster of an arbitrary number of processors.
The code itself is based on object-oriented programming and is implemented using a MPI library in C. The program is implemented using C++. The MPI routines used are C routines and not C++ for performance and feasibility reasons – the C version of the MPI libraries has proven to be very well implemented comparing with the C++ version [3].
Once compiled and ready to run, the program still allows for modifications on runtime is done using one text file as an input deck of the program. Parameters like the hollow cathode particle flux, the thruster’s inner and outer dimensions, time step interval and others can be easily changed without having to re-compile the program. Several kinds of thrusters with dimensionality differences (and others) can then be simulated without the necessity to write new code or even to compile it again.
The architecture of the program is similar to other PIC codes already done in its skeleton structure, namely it has followed some of the features of OSIRIS [4].
The parallelization scheme used is to divide the simulation box into several spatial domains. Each of these domains is to be processed by a single computer (or processor). Each domain is attributed to a given processor and this processor does all the computations on the grid points as well as on all the particles within this space. The space is, in this case, divided unevenly across processors for practical reasons and because it

is expected a greater number of particles to be processed in some zones of the simulation. It is convenient, of course, that the processing times for each processor and for each main loop are as balanced as possible.
The domain decomposition is done vertically as it can be seen in figure 2.

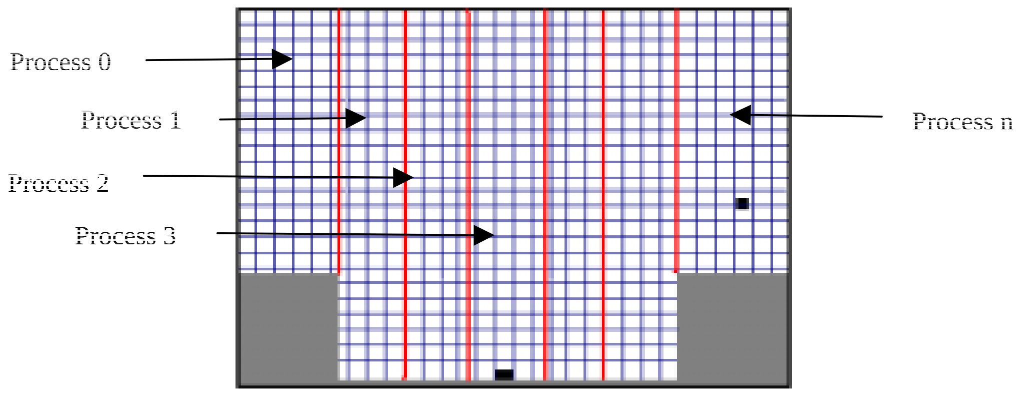

Fig. 2 – Distribution of the simulation area for the parallel processors.

3.1 Process communication

Process communication can be "expensive" in terms of computing time due to the transfer of particles between processes, i.e., in a cluster, the communication done between processes that are running on different machines using physically different memories takes significant time compared with time for advancing particles. Moreover, it is also normal that process communication time over a LAN is determined by the latency time of the network, as the messages passed between processes tend to be small. This has one immediate implication: the number of messages between processes should be reduced to a minimum. Also the communication should be done in order to perturb, at each time, the least number of computers possible.
This second constrain can be met using non-blocking MPI routines like MPI_Isend [3].
There are three types of communications that have to be done: one group of communications to pass particles between neighboring processes, another group of communications to pass the charge densities deposited on the grid, and another one to pass the potential of each required mesh point.
Concerning the first group of communications, it involves two independent send / receive communications – one to send and to ask for the number of particles to be sent and to be received, and the other one to send and to receive the particle information vector itself.
Concerning the two other groups, the code needs to send and to receive the information on the neighboring cells of each process. This is done so that the potential solver and the push sequences can be done locally in each domain.
Apart from these communications, the processes run independently for most of the computing time, thus guaranteeing efficient parallelization.

3.2 Program structure

The program is based on an object-oriented structure and implemented in C++. We have 6 independent classes: fluid, particle, grid, input, output and frontier.
For each of these classes we create an object (in each process), except for the particle class that is used to create different objects for different kinds of particles. Also, each object has the capabilities to do what is expected from it – there are no methods out of the objects. These capabilities are, for example, for the class particle, the capability of charge deposition onto the grid.
The main program structure for each process running in parallel is as follows: first, particles are injected in the simulation zone by the hollow cathode or inserted from previous simulations (from a file). Then, the potential solver executes (using the boundary conditions from the cathode and anode). After that, particle dynamics is updated – this includes moving particles, creating ion-electron pairs and so on. Charge densities on the grid are then calculated from particles position and the iteration comes to the end and the simulation loop is restarted.

This is all done in an arbitrary number of computers except for the communication phases that were described in the previous section.
Also, in several points of the simulation, different kinds of output diagnostics are made to external files. These files are written by process 0 only. This means further communication sequences to process 0 from all other processes, which somehow delays the simulation. However, the other possible solution was to put each process to output its results to independent files on each single computer of the cluster. This has the serious drawback that post-processing becomes a tedious and complicated task.

3.3 Particle exchange process

The particle push method parallelization is now described summarily. The idea is to implement a system where communications between processes are reduced to a minimum. This is done in the following way: for each particle on each process, the particle is first moved using the forces calculated from the self consistent field interpolated from the mesh points in the particle position. Then, boundary conditions are applied to the particle to verify if it remains on the process or if it has bumped on a wall and turned back (still remaining on the process) or if it has passed to a domain that belongs to another process. After that, eventual creation of new ion-electron pairs takes place. All particles are cycled this way.
At the end of this process, and after new particles have been eventually inserted in the hollow cathode zone – an operation that only happens in the correct process, the particles are exchanged between neighboring processes. This is done using standard MPI routines.
As observed, two communications are done in order to allow for an arbitrary number of particles to be sent and received in each process. The particles received on each process are then stored in memory and the dynamics of the particles is finished.
All the computation was done using only two sets of communications in this part of the code.

3.4 Field Solver

The parallelization of a field solver has been the most challenging task. Although the initial LU decomposition has no priori need for parallelization (because it is only done once), the amount of memory for storing all the necessary coefficients is a constraining factor for the maximum number of mesh points. So, not only the LU solver had to be developed in a parallel fashion, but also the initial LU decomposition. For both cases, the algorithms cannot be fully parallelized. Indeed, each simulation process needs the results of the resolution of an adjacent process in order to proceed. However, we have developed an algorithm that theoretically may achieve 99% parallelization efficiency, where the parallelization efficiency is defined by:

$$\varepsilon = \frac{T_{n-parallel}}{n \cdot T_1}$$

where $T_{n\text{-parallel}}$ is the time of a run performed by n processors in parallel, n is the number of parallel processors and T_1 is the time of a run performed by one processor alone.

The parallelization efficiency depends not only of the algorithm used but also of the number of processors, the number and distribution of mesh points and the communication delay.
However, the algorithm developed requires a considerable amount of communication which may decrease the parallelization efficiency below 70%, which is still acceptable.

3.5 Post processing and diagnostics

The results of the program are all sent to different files. These are binary computer files, each including the information of the iteration that it refers, to as well as other important informations. The output format chosen was the HDF – SDS (Hierarchical Data Format – Scientific Data Set). This has the main advantage that allows us to use post-processing graphical routines already done. In our case we mainly use an IDL interface and take the routines already developed at GoLP [8].

The kinds of outputs we are interested in are the electric field lines, the electronic, ionic and total charge density, the electric potential, among others. These can all be done using C HDF libraries.

4. Facilities

All the simulations have been designed to primarly run at the EP2 cluster at IST. The use of a computer cluster instead of a traditional supercomputer for the problem sizes typical of a hall thruster 2 D 3 V simulation is perfectly suitable. This is mainly because of the lower turnover time in clusters (This is, of course, directly related to the size of the problem we are trying to solve).
The EP2 cluster is based on the Dual Power Mac G4 computer from Apple and on the Appleseed paradigm. The choice of Macintosh computers over the usual Intel x86 Beowulf architecture in the initial development of the cluster, was mainly related to the low installation and maintenance requirements that characterize Macintosh systems. As for performance, the Power PC G4 processor, from Apple, Motorola and IBM, is capable of a sustained performance of over one Gigaflop, and presents excellent characteristics for scientific computing. Furthermore, Apple's operating system currently available on the cluster, Mac OS X, has a UNIX core which allows us to use most computational libraries available, and, together with hardware integration done at the manufacturer, results in superior reliability and stability.
Other reason for this choice is the fact that MPI routines are all well implemented to be used on a UNIX system. The same routines exist to use in windows based systems, but their reliability and operating speeds are not assured.

Processor	**OS**	**Comm. Libs.**	**NIC**	**Network**	**RAM**	**Storage**	**Peak Performance**
16 Dual Power Mac G4/450, 2x G4 PPC/450 Mhz	Mac OS X 10.1.2	LAM/MPI 6.5.6 (including MPE and ROM-IO)	1Gbit/s Ethernet Cards	Asant\'e Intracore 8000 10/100 Mbit/s switch	1.125 Gbyte/node, 18 Gbyte total	30 Gbyte/node, 480 Gbyte total	57.6 GFlop/s

Table 1 – EP2 Cluster characteristics

Table 1 presents the general EP2 Cluster characteristics. The network backbone is provided by a Asant\'e Intracore 8000 10/100 Mbit/s switch. This switch was chosen for its performance and scalability, providing a fully meshed network and assuring a 100 Mbit/s network connection between any two ports. Power supply is done through two Pulsar 3000 UPS units from MGE. The cluster also includes two 17" CRT monitors and two 8 port KVM-USB switches from Dr. Bott for installation and maintenance purposes. The cluster room had to be fitted with an appropriate air conditioning system, given the high power dissipation from the 16 computers [9].

5. Conclusion

Fully kinetic two dimensional Hall thruster parallel simulation on the EP2 cluster has been successfully developed. Its architecture is suitable to allow model upgrades for different phenomena. Charge exchange collision is one of the necessary upgrades to include on the simulation because it may be an important phenomenon responsible for plume divergence [1]. The architecture is also suitable to easily change the thruster parameters, dimensions and even to include additional electromagnetic boundaries (magnets, electrets). An evaluation campaign will be performed soon in order to validate the code. This state of the art Hall thruster simulator will allow understanding hidden physics and perform experimental research on thruster optimizations/design such as: plume focusing, plume deflection, two stage Hall thruster development, oscillations and instabilities, new propellant utilization.
On the other hand, the code may be run on different type of clusters, and with unlimited number of processors. All this versatile architecture makes it suitable for future upgrade and utilization.

The main development done here, was the creation of a code suitable to perfomr two dimensional hall thruster simulations, running in parallel in almost any parallel system, compiled in only minor modifications. Furthermore, due to its object-oriented design, it has the advantage to be sufficiently flexible to include new features like coils, different boundary conditions for particles, new dimensions and many other relevant characteristics.

6. Reference Documents

[1] – I. Vieira, Convergence and Deflection of a Hall thruster plume, IEPC – March 2003

[2] – R. W. Hockney and J. W. Eastwood, Computer Simulation Using Particles, Institute of Physics Publishing

[3] – W. Gropp, E. Lusk, R. Thakur, Using MPI-2 – Advanced features of the message passing interface, MIT press

[4] – R. Fonseca, L. Silva, R. Hemker, F. Tsung, V. Decyk, W. Lu, C. Ren, W. Mori, S. Deng, S. Lee, T. Katsouleas and J. Adam, OSIRIS: a three dimensional, fully relativistic particle in cell code for modeling plasma based accelerators, Lectures Notes in Computer Science Vol. 2329, III-342 (Springer-Verlag, Heidelberg 2002)

[5] – James J. Szabo, Manuel Martinez-Sanchez, Fully Kinetic Hall Thruster Modelling, 27th IEPC, October 2001

[6] – David G. Fearn, The influence of Charge Exchange Ions on the Beam Divergence of an Ion Thruster, 27th IEPC, October 2001

[7] - Intermediate Report, Convergence and Deflection of a Hall thruster plume - ESTEC/Contract No 15686/01/NL/CK, GoLP/IST 2003

[8] – R. Fonseca, IST PhD thesis, 2002

[9] – http://:cfp.ist.utl.pt/golp/epp/

[10] – C. K. Birdsall, A. B. Langdon, Plasma Physics via Computer Simulations, IOP 1991

[11] – W. H. Press, B. P. Flannery, S. A. Teukolsky, W. T. Vetterling, Numerical Recipes in C: the Art of Scientific Computing, Cambridge University Press; 2nd edition (January 1993)