Annales UMCS Informatica AI 6 (2007) 137-148

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI

http://www.annales.umcs.lublin.pl/

An MPI-based parallel code for high performance 3-D particle-in-cell ion source plasma simulation

Marcin Turek[*1], Marcin Brzuszek[2], Juliusz Sielanko[2]

[1]*Institute of Physics, Maria Curie Sklodowska University, pl. Marii Curie-Skłodowskiej 1, 20-031 Lublin, Poland*
[2]*Institute of Computer Science, Maria Curie Sklodowska University, pl. Marii Curie-Skłodowskiej 1, 20-031 Lublin, Poland*

Abstract

The paper presents parallel version of TRQR code, intended for ion source plasma simulation. The code is written using particle decomposition approach, and enables plasma potential, charge density, ion extraction current, etc. calculations. Parallelisation was done using widely accepted MPI platform. Additionally, the code has been rewritten and optimised in order to achieve better efficiency. The strong- and soft-scaling properties of the parallel code are shown (tests were done using SMP machine), as well as some results of physical meaning. The code in its present form allows calculation involving $5 \cdot 10^8$ pseudo-ions at reasonable wall-clock time (circa 15 hours), which opens the gate for realistic physical conditions plasma calculations.

1. Introduction

Numerical simulation is a powerful tool that helps understand physical phenomena and enables introducing new technologies for industry and science applications. An advent of large scale supercomputers gives opportunity to solve complex problems but requires continuous development of programming techniques, including code optimization and parallelization, which enables effective usage of processing power.

Plasma ion sources are widely used in many fields of physics. One of them is the concept of neutral beam injection (NBI) heating system for thermonuclear reactors like tokamaks and stellarators [1]. NBI heating systems require the intensive ion beams of well defined parameters. Very high neutralization efficiency, the negative ion beams are a promising candidate for this purpose (also in the ITER project). Negative ion beam extraction from plasma sources is quite different compared to a similar problem in the case of positive ion. The

[*]Corresponding author: *e-mail address*: mturek@kft.umcs.lublin.pl

understanding and modelling of the transport properties of negative ions is very important, but also a rather difficult issue, for which the TRQR code is developed [2]. Plasma simulation based on the particle-in-cell (PIC) method [3] requires following trajectories of a huge number (typically up to 10^9) of computational particles (pseudo-particles, macro-particles). In the case of several kinds of calculations (e.g. the influence of magnetic field on ion beam extraction) the simulation must last long enough so the steady state conditions are established (tens of thousands of simulation time steps). Additionally, time step and cell size have to be adjusted to physical parameters describing plasma – they decrease fast with plasma concentration. All these factors make the realistic plasma simulations great challenge for programmers as well as opportunity to test solutions in high performance computing (new techniques in parallel programming [4]; using multi-processor machines – vector, SMP machines or clusters; code optimization [5]).

The particle decomposition approach has been used while writing parallel version of TRQR code. The code was rewritten and optimized. As a parallelization tool widely accepted MPI platform has been used. The paper presents also results of basic efficiency benchmarks as well as exemplary results of large scale simulations.

2. Numerical model

The TRQR code bases on the Particle-In-Cell (PIC) method for computing the trajectories of charged particles in the electromagnetic field. The main feature of the PIC method is using computational particles (macro-particles, pseudo-particles). Each of them represents a large number (10^3-10^9) of real particles (ions or electrons) moving the same way. Macro-particle follows trajectory that is a solution of single-particle equations of motion – at this stage it behaves like a real particle, but is assigned multiple charge when charge density calculations are in progress. Macro-particle is assumed to have finite size and spatial shape – the shape determines the way the macro-particle charge is assigned to spatial grid points, which is the crucial point of the PIC method. There is a variety of PIC method variations depending on how macro-particle shape function is chosen. The simplest one is the nearest grid point scheme (NGP) when the whole macro-particle charge is assigned to only one (nearest to the particle position) grid point. The NGP algorithm is very fast, and easy to implement [6]. In the case of more complex shapes (so called Cloud In-Cell methods) charge is distributed among a larger number of grid points using weighting coefficient depending on particle shape function [3]. The ‘smearing’ of macro-particle charge may involve a large number of near grid points (27 or even more). Such approach yields smoother charge distribution, reduces

oscillation I plasma potential profiles, but costs more CPU time [6,7]. It should be mentioned that the code stores charge densities arrays for each 'species' of particles, as well as total charge density.

Having the total charge density $\rho(x, y, z)$ calculated, the plasma potential is obtained by solving the Poisson equation:

$$\Delta V(x, y, z) = \frac{\rho(x, y, z)}{\varepsilon_0},$$

where ε_0 is the dielectric constant. The Poisson equation is solved using the optimized finite-difference successive over-relaxation (SOR) method [3]. Electrodes' and plasma chamber potentials are set at the primary stage of simulation.

3. Parallelisation of TRQR code

The TRQR code is written in widely accepted in scientific community FORTRAN 77 language. The code consists of following files:

- The 'skeleton' main program contained in the 'TRQR_PLAZMA.f'
- Main subroutine library (file 'trqr_sub.f', connected with the main code via INCLUDE command, as all mentioned below files)
- Header file 'wstep_dat.f' – containing variables and constants declarations and initialisations
- General data file 'gen_dat_PLAZMA.txt' with the most important parameters describing simulation (plasma density, number of pseudo-particles, time of simulation etc)
- Data file for CROSS subroutine (checking if the particle hit the electrode or wall) 'cross_PLAZMA.txt'
- Particle 'creation' area data file 'source_PLASMA_dat.txt'
- File 'pre_calc.f' containing initial calculation subroutines (pseudo-particle charges, charge/mass ratios, coefficients for SOR Poisson equation solver subroutine etc.)
- An optional file 'zapis.f' containing trajectory print-out subroutine

The code has a modular structure. The user may add or remove some features by commenting or adding calls of subroutines from the 'trqr_sub.f' (or any other 'included' file) in appropriate places of the main program.

The partitioning stage of a parallel code design is intended to expose opportunities for parallel execution [8]. There are three main approaches of parallelising the PIC code. One of them is so-called 'domain decomposition'. The simulation area is divided into many sub-domains, assigned to different CPUs. 'Neighbouring' CPUs communicate in order to interchange information about the particles that entered or exited their areas. The communication algorithms may be very complex and hard to be implemented into the existing

code, especially as huge as TRQR (developed over many years). The advantage of such strategy is relatively small memory consumption. On the other hand, the number of subdomains scales with the number of processors, so a) their spatial extent in the direction of parallelization becomes smaller and smaller b) the particle communication is increased.

The second strategy is to employ 'particle decomposition' scheme. Every CPU has full information about simulation area, but only a subset of computational particles is assigned to single CPU. No 'particle communication' is needed. The simulation on every CPU follows almost independently. The one exception is potential calculation – this has to be done by one (say master) CPU. The contributions to charge density from all CPUs have to be summed and used for potential calculations. Then calculated potential is to be distributed among all CPUs. Great advantage of this approach is even load balancing. However, high memory consumption is caused by the fact that a lot of data (charge density distributions) have to be stored separately for each processor. Moreover, a relatively large average number of particles per cell is needed.

The third approach, called 'domain cloning' is the most general one and contains previous techniques as limiting cases. Multiple copies of simulation domain are made, particle communication is necessary only between neighbouring subdomains in one clone. Charge distribution data are summed over all clones. The domain cloning concept offers the opportunity for optimising the scaling property of advanced PIC codes such as the TORB code [4].

The parallelisation of TRQR was made using the particle decomposition technique, mostly due to the fact that this approach is relatively easy to implement in the case of large existing codes. In order to enable code transformation to the parallel version, TRQR was rewritten to a more clear form:

- Many obsolete features have been removed.
- Most of loops have been 'hidden' in subroutines.
- Branching instructions (GO TO, arithmetic IF) have been removed to improve performance and make the code clear.
- Code was given a more clear, structuralized form (many features may be added or removed by comment sign manipulations instead of multitude of switches and IF instructions).

The code was parallelised using the 'particle decomposition' approach as mentioned before. MPI initialisation (MPI_INIT), getting the number of available processors NUMPROC (via MPI_COMM_SIZE tool) and assigning ranks to them (MPI_COMM_RANK) is placed in 'wstep_dat.f'. The file contains also such constants as: the size of particle position and velocity matrices NCZ (approximately 10-15 % bigger than the number of particles in the chamber NNJON); the size of spatial grid N[XX-ZZ]; over-relaxation parameter

etc. The other input file 'gen_dat_PLAZMA.txt' stores information about cell size (DE[X-Z]), starting coordinates of the grid [X-Z]BSTART, limitations of simulation area [X-Z]C[MIN-MAX], simulation time step DET as well as the total number of main loop runs NDET, total plasma concentration GEST_CALK (in particles/m^3 units), ratios of four plasma components XMSTVR[1-4] and their masses AMASC[1-3], and initial energies ELAB[1-4]. The 'source_ PLAZMA_dat.txt' provides information for particle distribution subroutines (size of the plasma chamber and its placement). Moreover, new parameter NSSXYZ, the number of REAL type data to be exchanged by CPUs is introduced. After input data are read from files every process determines particle positions and velocities (see block scheme of sequential code). Particles are randomly 'created' with the uniform distribution inside the plasma chamber. Then the initial charge density is calculated by the GEST_PL1 subroutine. Note, that the NNJON parameter i.e the number of pseudo-ions in the chamber, fact, is the number of pseudo-ions in the chamber **governed by one CPU.** That is why the number of real ions per pseudo-ion (WSP_Q) has to be rescaled by the number of CPUs. The 'master' process (of rank 0) reads information about electrode geometry (ELE_LAG_M subroutine), then sends appropriate data (IVE and V arrays containing information about electrodes' shape and potential) to all CPUs (by MPI_BCAST collective communication tool).

The simulation runs almost independently on all processors. Data describing electromagnetic field are identical for all processes. Every CPU follows trajectories of 'their' particles, checks whether they hit the electrode or chamber wall or not, calculates contribution to charge density and sends it to the 'master' process. The total charge density distribution is calculated on-the-fly, by means of 'global sum' operation i.e. MPI_REDUCE with MPI_SUM option. The only non-parallel piece of the code is potential calculation – mainly because of the fact that it requires different, domain decomposition approach [9]. The 'master' uses total charge density to solve the Poisson equation using the optimised iterative SOR method. Potential data are broadcast to all processes by means of MPI_BCAST command. Then all processes calculate electric field values, move particles etc. (see the simplified block scheme of parallelized TRQR – Figure 1).

It should be noted, that the data output is also managed by the 'master' process. This is simple in the case of potential data output (PROBE, ZAPIS_ GEST) because all information about potential is available by 'master'. On the other hand, charge density contributions from all CPUs have to be added before writing to file (ZAPIS_GEST). This is done by the above mentioned 'global sum' mechanism. The same applies to ANUM matrices containing distributions of particles crossing some selected planes (RESULTS subroutine).

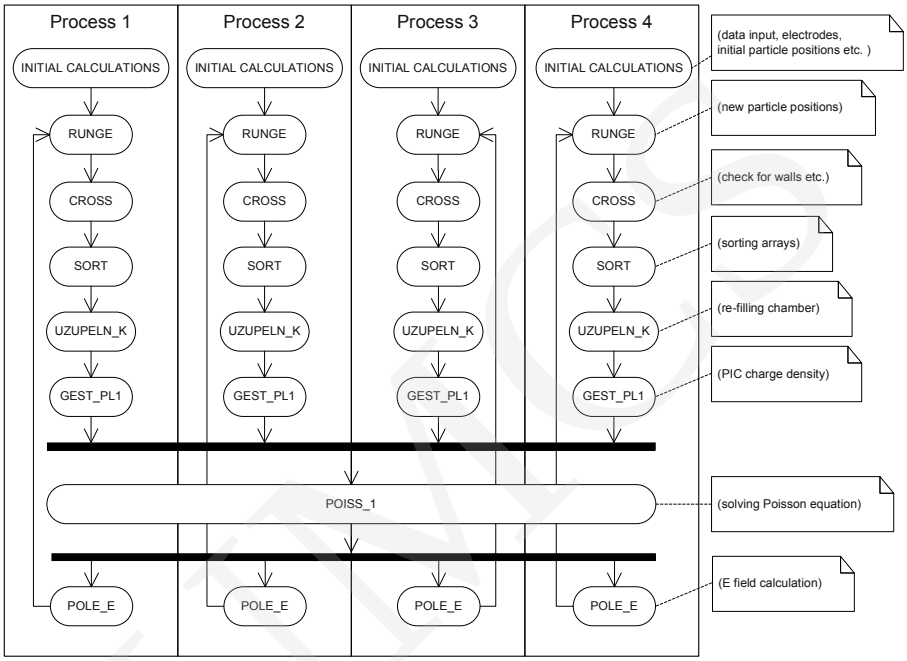

Fig. 1. The block scheme of TRQR code (after parallelisation)

4. Performance benchmarks

After parallelising the TRQR code some basic performance benchmarks were performed. First of them is so-called 'strong scaling' test. During that test the size of the problem (in our case – the total number of particles) remains unchanged, whilst the number of used CPUs increases. The test shows the speed-up as a result of using many processors. It enables estimation how much faster reasonable results could be obtained the using increased number of CPUs. In the second test (so called 'weak scaling') the size of the problem increases proportionally to the number of processors (putting it in other words, the number of particles per CPU remains constant). The normalized speed-up gives us information how much faster a large numerical problem is solved by many processors compared to the time of calculations using single CPU. One should be aware that very often solving a huge numerical task is impossible using a single CPU machine, most often due to the lack of memory, so the single-processor time should be considered as a theoretical concept.

The tests were performed using single 32-way node of Rechenzentrum Garching 'Regatta' SMP cluster (1.3 GHz Power4 IBM p690 system running on AIX 5.3 operating system). The executable was built with IBM XLF 9.1

compiler (with –q64 large memory area access switch, -qipa interprocedural optimization option and –O3 optimization level). The code was running under the POE parallel environment. It should be mentioned that benchmarks were made during normal operation of the cluster, using interactive node available for other users.

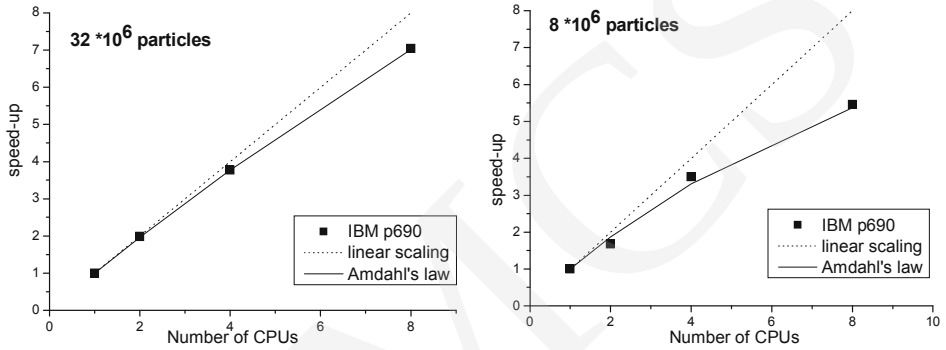

Fig. 2. Hard scaling properties of the parallel TRQR for two different numbers of pseudo-particles

Strong scaling benchmarks were done for two values of the total particle number: $32 \cdot 10^6$ and $8 \cdot 10^6$ – Fig.2. One can see that the influence of communication overheads is larger in the case of a small particle number. The speed-up factor S_n for n CPUs is calculated according to formula:

$$S_n = \frac{T_1}{T_n},$$

where T_1 and T_n are times for single and nCPU computations, respectively) achieves in the case of 8 CPUs reasonable value of 7.04 for $32 \cdot 10^6$ particles, whilst only 5.5 for $8 \cdot 10^6$ particles. Those values correspond to the efficiencies of 0.88 and 0.69, respectively. Efficiency is a performance metric showing how well-utilized CPUs are compared to the effort of communication and synchronization. Defined as

$$E_n = \frac{S_n}{n}$$

has values between 0 and 1. The obtained speed-up values are presented together with the linear speed-up line as well as the values predicted by Amdahl's law [10], which states that even small sequential pieces of a parallel code are a bottleneck and hinder performance. Amdahl's law could be written in the following form:

$$S_n = \frac{1}{F + (1-F)/n},$$

where *F* is the fraction of the code that is sequential. The values of F are taken from Table 1, which presents the execution time fractions for most important pieces of the code.

Note quite satisfactory agreement of theoretical predictions with the obtained speed-up values.

Table 1. Execution time fractions for most important pieces of the code. (MOVE – particles 'push' subroutine, check for wall hits, etc. RE-FILL – refilling the plasma chamber with a proper number of particles in order to keep plasma density invariant, CHARGE - charge density calculations, POISSON – Poisson equation solver, FIELD – electric field calculations)

Block of TRQR code	Block execution time percentage	
	Case of $8 \cdot 10^6$particles	Case of $32 \cdot 10^6$particles
MOVE	80.0	87.6
RE-FILL	1.32	1.48
CHARGE	9.20	8.47
POISSON	8.47	2.03
FIELD	0.80	0.23
OTHER	0.23	0.23

The fraction of the POISSON block gets smaller with the increasing number of computational particles (fraction of this piece of the code is dependent mostly on the size of the spatial grid).Execution times of remaining pieces of the code are dependent on the particle (per CPU) number and decrease as CPU numbers get larger (see Figure 3.).

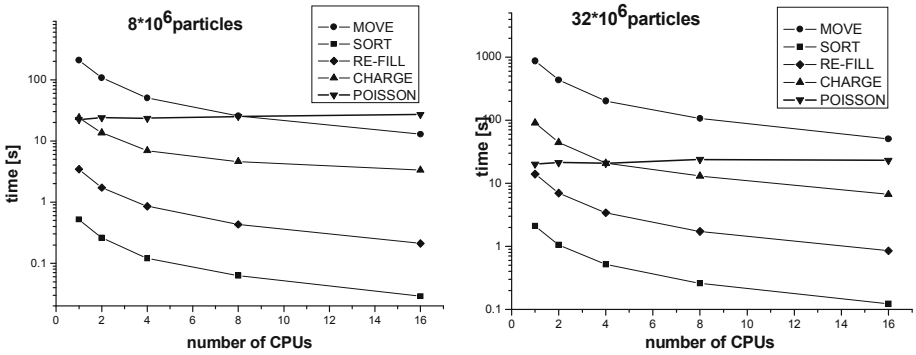

Fig. 3. Execution times for most important code pieces as a function of CPU number

The MOVE block, including particle 'push' subroutines, check for wall hits etc. is the most CPU time consuming one. However, for a relatively small number of particles per CPU, its execution time becomes even smaller than that

of POISSON solver. Hence, under such conditions the increase of CPU number is pointless.

Figure 4 presents the weak scaling properties of the TRQR code. Up to 8 processors the scaling is almost linear. The results are very similar for both presented cases.

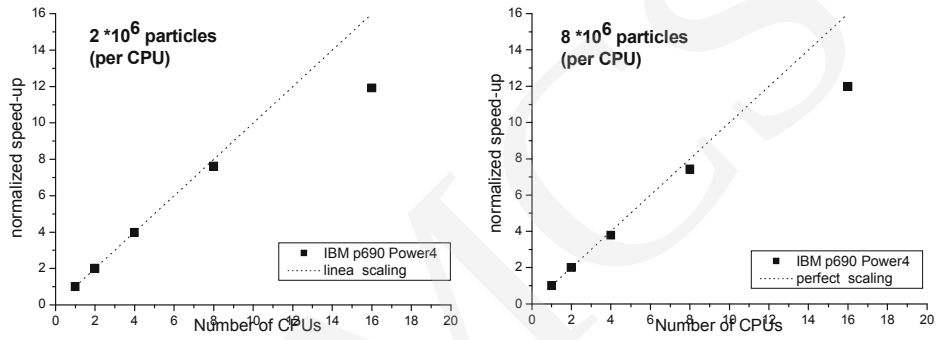

Fig. 4. Weak scaling properties of parallel TRQR

Once again one can observe reduction of efficiency caused by the sequential POISSON block from 0.94 for 8 CPUs to approximately 0.75 when the number of CPUs doubles. Nevertheless, using a large number of CPUs with the parallel TRQR code gives opportunity to run massive simulations involving hundreds of millions of particles.

5. Exemplary results

In this section we present some results obtained using the parallel TRQR code. Behaviour of the plasma in the ion source chamber as well as extraction of negative ion beam in the multi-aperture extraction system were studied. Plasma grid and extraction electrodes geometry correspond to those of RF ion source with the CEA grid extraction system installed at IPP in Garching [1]. The geometry of the simulation area is presented in Figure 5 as a cross-section through a three-dimensional structure. Particles are randomly 'created' with uniform distribution inside the shaded area. The plasma in the chamber consists of H^-, H^+, and electrons with the species ratio 0.1/0.5 /0.4 respectively. Initial velocities are randomly directed, and their values correspond to energy of 1 eV. During calculations $64 \cdot 10^6$ of macro-particles were used. The total plasma density was $1 \cdot 10^{16}$ particles/m^3. The simulation was run on the 8 CPU IBM p575 system with Power5 1.9 GHz processors and 32 GB memory.

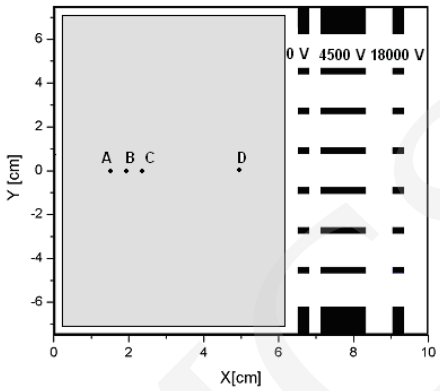

Fig. 5. Schematic view of simulation area and electrode geometry

The NGP charge distribution scheme to the 100×100×100 Cartesian spatial grid was applied. The simulation was stopped after 3150 iterations. The time step was $5 \cdot 10^{-11}$ s. The code recorded plasma component charge densities and plasma potential distribution every 450 iterations.

Figure 6 presents the potential profiles along the central line of the chamber at four stages of simulation. The initial profile is very steep, in fact, it does not differ much from the potential profile in the case of empty chamber. However, after some time screening properties of plasma make it flatter and flatter inside the chamber – there is no electric field in the plasma.

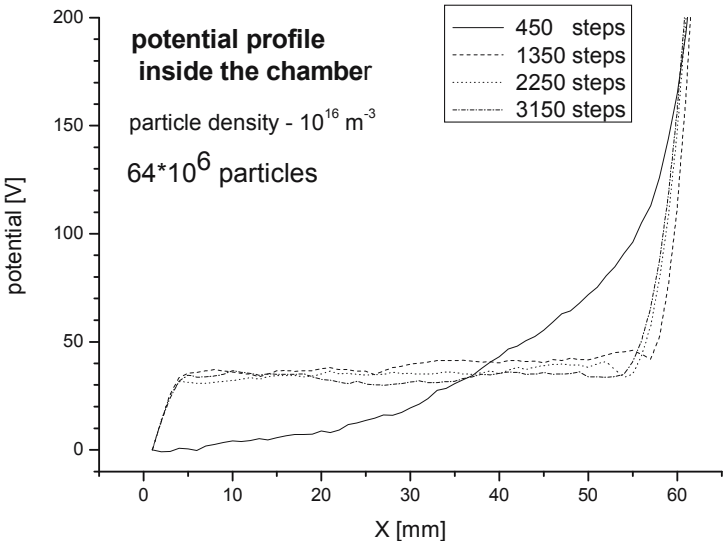

Fig. 6. Evolution of the plasma potential profile

Figures 7 a), b), c) and d) show the cross-sections of potential density and electron, H^- ions and H^+ ions charge densities, respectively, after 3150 time steps. One can see multi-meniscus formation, collective behaviour of negative and positive particles as well as sheath on the walls of the chamber.

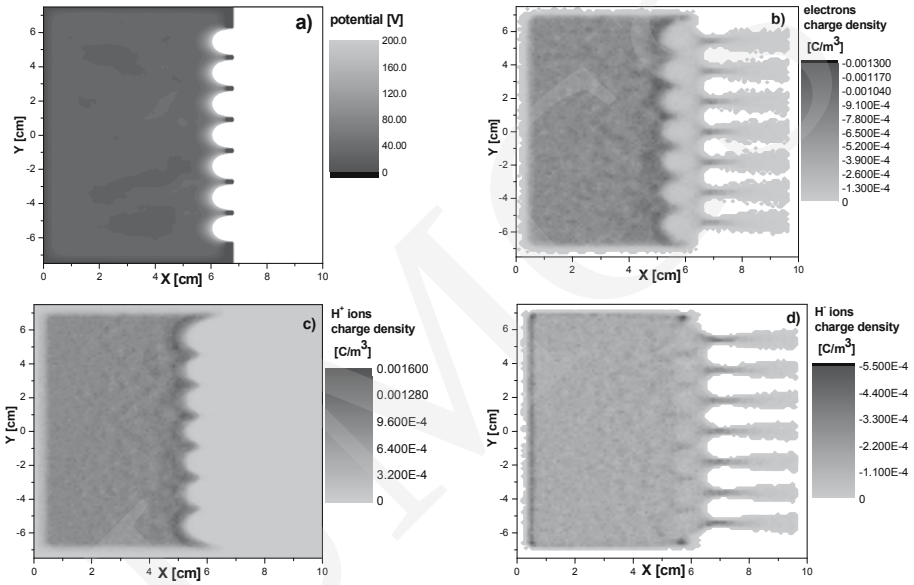

Fig. 7. The plasma potential and charge density distribution at the final stage of simulation (after 3150 time steps)

Summary

In the paper a new, effective tool for ion source plasma modelling is presented. The TRQR code employing the MPI parallelization platform shows good efficiency during huge simulations on the medium-scale SMP clusters and yields results of physical meaning. Better scalabilty could be achieved by a) parallelizing the Poisson solver (using e.g. domain decomposition scheme or quite a new solving algorithm based on the FFT technique) b) mixing MPI message communication with shared-memory based parallel platforms like OpenMP, which is a new trend in large-scale parallel computing for waste SMP clusters (fast OpenMP is employed for intra-node parallelization, while the slowest MPI supports communication between nodes). This will be a goal for future studies.

Acknowledgement

This work is supported by Maria Curie-Skłodowska University in Lublin in the frame of the grant of MCSU Rector from the Polish Ministry of Science and Higher Education funds.

References

[1] Vollmer O., Heinemann B., Kraus W., McNeely P., Riedl R., Speth E., Trainham R., Wilhem R., Fusion Eng. and Des., 56-57 (2001) 485.
[2] Staebler A., Sielanko J., Goetz S., Speth E., Fusion Technology. 26(2) (1994) 145.
Sielanko J., Muszyński M., Electron Technology, 30(4) (1997) 352.
Turek M., Sielanko J., Franzen P., Speth E., AIP Conference Proceedings, 812 (2006) 153.
Turek M., Droździel A., Pyszniak K., Sielanko J., Vacuum, 78 **(**2005) 649.
[3] Birdsall C.K., Langdon A.B., *Plasma Physics Via Computer Simulation*, McGraw-Hill, New York (1985).
Hockney R., Eastwood J., *Computer Simulation Using Particles*. Mir, Moskwa, (1987).
[4] Kim C. C., Parker S. E., J. Comp. Phys., 161 (2000) 589.
Hatzky R., Parallel Computing, 32 (2006) 325.
[5] Decyk V.K., Karmesin S.R., de Boer A., Liewer P.C., Computers in Physics, 10 (1996) 290.
[6] Sielanko J., Turek M., Tanga A., Annales UMCS-Informatica AI, 2 (2004) 251.
[7] Brzuszek M., Turek M., Sielanko J., accepted to Annales UMCS-Informatica.
[8] Foster I., *Designing and Building Parallel Programs*.
available online at http://www- unix.mcs.anl.gov/dbpp/
Wilkinson B., Allen M., *Parallel Programming*, Prentice Hall, Upper Sadle River, (2005)
[9] Turek M., Franzen P., Sielanko J., Annales UMCS Informatica AI, 3 (2005) 235.
[10] Amdahl G., AFIPS Conference Proceedings, 30 (1967) 483.