Annales UMCS Informatica AI 3 (2005) 235-242

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI

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

Application of MPI standard in computer modelling of the ion source plasma

Marcin Turek*, P. Franzen[a], Juliusz Sielanko

Institute of Physics, Maria Curie-Skłodowska University,
pl. M.Curie-Skłodowskiej 1, 20-031 Lublin, Poland
[a]Max Planck Institute of Plasma Physics, Garching,Germany

Abstract

In the paper the adaptation of the TRQR code to run on the massive parallel computer using MPI standard is described. We present the modification of the procedure used for the calculations of the potential spatial inside the plasma chamber of the ion source based on PIC (Particle In Cell) model and the Poisson equation. Parallelisation of the procedure enables us to reduce the CPU time by the factor 6-7 in the case of 8 processor job queue.

1. Introduction

Neutral beam injection for heating of the fusion plasma in the TOKAMAK require the intensive well defined ion beams. Because of a very high neutralization efficiency, the negative ion beam is a very promising candidate for this purpose. Negative ion beam formation and extraction from the plasma source is quite different in comparison with that for a positive ions. The understanding and modelling of the transport properties of negative (mainly H^- and D^-) ions is a very important, but also rather difficult issue. The TRQR code [1-3] based on the PIC method was developed for the purpose of study of the plasma behavior as well as extraction and formation of the ion beams emitted from the plasma ion source. The trajectories of the charged particles in the electric field created by the electrode system as well as self consistent field are calculated by solving the equation:

$$\frac{d\vec{v}_i}{dt} = \frac{q_i}{m_i}\left[\vec{E} + \vec{v}_i \times \vec{B}\right]$$

where m_i, v_i, q_i, E, B are the particle mass, velocity, charge, electric field and magnetic field for the i-th particle of the plasma species respectively. The spatial

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

distribution of the electric potential V is obtained by solving the Poisson equation:

$$\nabla^2 V(x,y,z) = -\frac{\rho(x,y,z)}{\varepsilon_0}$$

where ε_0 is the dielectric constant and $\rho(x,y,z)$ is the charge density.

After each time step, determined by the solution of the motion of the plasma particles, new positions of the particles generate new electric field distribution. In consequence, the Poisson equation should be solved after each iteration step. Because of the dense spatial grid required for the good accuracy of the calculation of spatial distribution of the potential $V(x,y,z,)$, the CPU time consumption by the code may be great. Some lowering of the CPU time of the calculatin can be obtained by the parallelisation of the code and its run on the cluster or the multiprocessor computer. The purpose of this work was the adaptation of the TRQR code to run on the massive parallel computer using MPI standard and parallelisation of the procedure responsible for solving the Poisson equation.

2. Parallel calculations of the plasma potential

In the original TRQR code the Poisson equation solving is done by the sligthly modified SOR iterative method. This part of the code can be very CPU time-consuming due to rapidly changing charge distribution. In the parallel version of the TRQR we are going to use the domain decomposition approach. In this method the whole domain in which the Poisson equation is solved (typically – 100×100×100 Cartesian grid) is divided into smaller pieces, subdomains. Each of them is assigned to a separate processor. In every subdomain the potential is calculated by the SOR method, which usually converges the faster, the smaller is the subdomain.

It should be mentioned that the border grid points have to be shared by neighbouring subdomains, which is schematicaly shown in Figure 1.

In our code the one-dimensional communicator (named and ordered group of processes) is created by MPI_CART_CREATE command, then the process ranks (idn) are obtained using MPI_RANK. Ranks of neighbour processes, the next and the previous one, are provided by another MPI subroutine: MPI_CART_SHIFT.

Keeping in mind the way array elements are stored in memory we decided to slice the domain perpendicular to z-axis, see Figure 2. This allows us to avoid using MPI derived data types during the iteration phase, as long as we consider one-dimensional communicator.

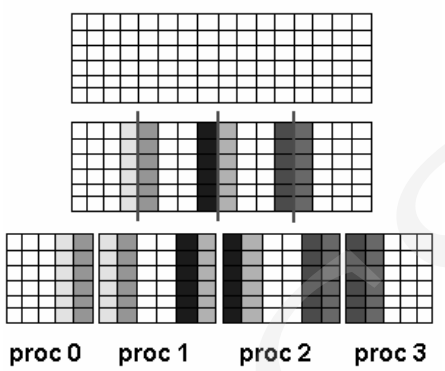

Fig. 1. Schematic view of 2D grid divided into 4 subdomains.
Note, that the border grid points are common

Before the SOR iteration can be started, the initial values of potential (that is the potential values on electrodes) have to be distributed among processes. Neither MPI_SCATTER nor MPI_SCATTERV utility can be used in this case, because they cannot refer to the same place in memory more than once. Consequently, we decided to use the pairs of MPI_SEND and MPI_REDUCE instead of collective comunication.

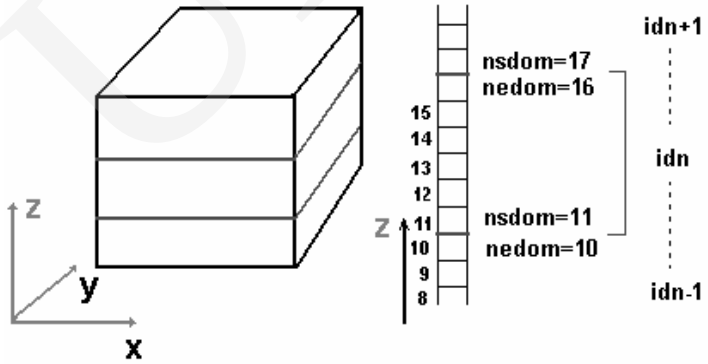

Fig. 2. Schematic view of the domain division in z-direction

The SOR iteration for the potential value in the grid point (i,j,k) is described by the formula

$$\begin{aligned} V(i,j,k) = c_0 \cdot V(i,j,k) + c_1 \cdot \left(V(i-1,j,k) - V(i+1,j,k)\right) \\ + c_2 \cdot \left(V(i,j-1,k) - V(i,j+1,k)\right) \\ + c_3 \cdot \left(V(i,j,k-1) - V(i,j,k+1)\right) \\ + c_4 \cdot \rho(i,j,k) \end{aligned}$$

where c_i are the coefficients depending on the overrelaxation parameter ω and the sizes of the grid cell (Δx, Δy, Δz**)** are precalculated at the code beginning and then distributed by MPI_BCAST:

$$c_0 = 1 - \frac{1}{2}\omega,$$

$$c_1 = \frac{\omega}{1 + \left(\frac{\Delta y}{\Delta x}\right)^2 + \left(\frac{\Delta z}{\Delta x}\right)^2},$$

$$c_2 = \frac{\omega}{1 + \left(\frac{\Delta x}{\Delta y}\right)^2 + \left(\frac{\Delta z}{\Delta y}\right)^2},$$

$$c_3 = \frac{\omega}{1 + \left(\frac{\Delta x}{\Delta z}\right)^2 + \left(\frac{\Delta y}{\Delta z}\right)^2},$$

$$c_4 = \frac{\omega}{\varepsilon\left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2} + \frac{1}{\Delta z^2}\right)}.$$

The iteration runs from the lower limit of subdomain (nsdom) up to the upper limit (nedom) in the z-axis direction, and along the diagonal lines in the xy plane. Once the run of iteration loop is completed in every subdomain, the new potential values in the border grid points have to be exchanged between the neigboring processes. This is done using the pairs of MPI_SENDRECV subroutines and illustrated in Figure 3.

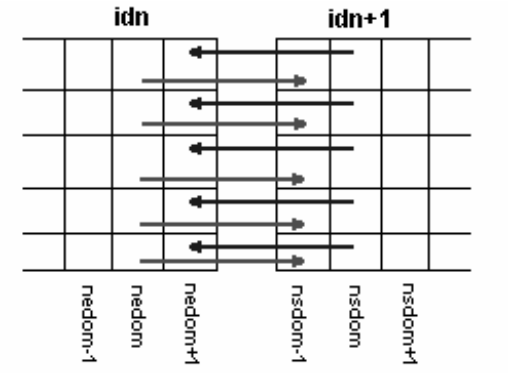

Fig. 3. The exchange of the potential values between the border grid points

The iteration loop stops when the difference between old and new potential values is smaller than the precision parameter. The maximal differences are

exchanged by MPI_ALLREDUCE subroutine with the MPI_MAX flag. After the iteration is completed, the new potential values are gathered to the master process by the pairs of MPI_SEND and MPI_RECV subroutines.

Timing for transmission and iteration phases is provided by the pairs of MPI_BARRIER and MPI_WTIME commands, note that is wall-clock, not CPU time.

Fig. 4 presents the initial data used to test or parallel code. The shape of electrodes and potential values resembles much those in 'production' simulations of ion sources. Test runs were performed on 32-way IBM eServer p690 machine with 1.3 GHz

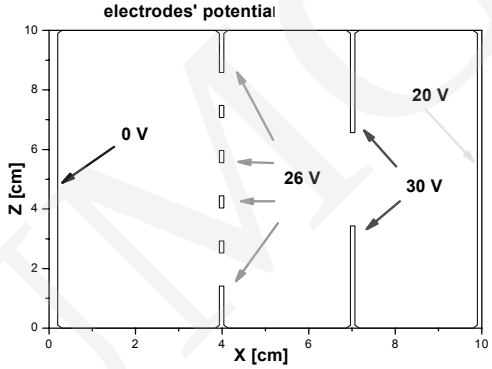

Fig. 4. Block scheme of the electrodes and its potential values used in tests

POWER4 processors and 64 GB of memory. The machine is a node of 'Regatta' cluster in the Institute of Plasma Physics in Garching running under AIX 5L system. The code was compiled using IBM XL Fortran v. 9.1 compiler with –O2 and –qipa optimization switches.

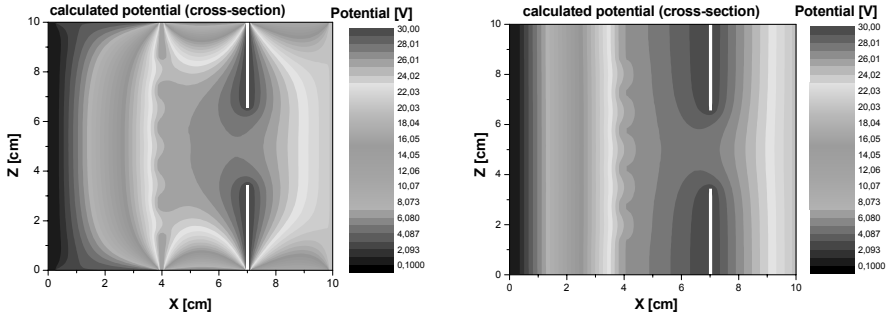

Fig. 5. Potential distribution obtained in the case of boundary conditions I (left) and in the case of boundary condition II (right)

We considered boundary conditions of two types: potential values are well-defined and invariant on the grid boundary (boundary condition of kind I), or the

case of vanishing gradient of potential on the boundary (boundary condition of kind II). Figure 5 presents the calculated potential distributions for both cases.

The test run results presented in Figure 6 show the gain of code parallelisation. We used up to 24 processors. The time of potential calculation (SOR iteration itself and data transmission) initally decreases very rapidly, the time of calculation is almost perfectly inverserly proportional to the number of used processors. However, one can observe that the time of calculations does not change significantly for a larger number of processors (aproximately 12), or even grovs slightly. For many parallel codes such a slow-down is caused by the increasing amount of data to exchange and hardware limitations. Figure 7 hows that this is not the case. The transmission time is negligibly small compared to the time iteration needs to converge.

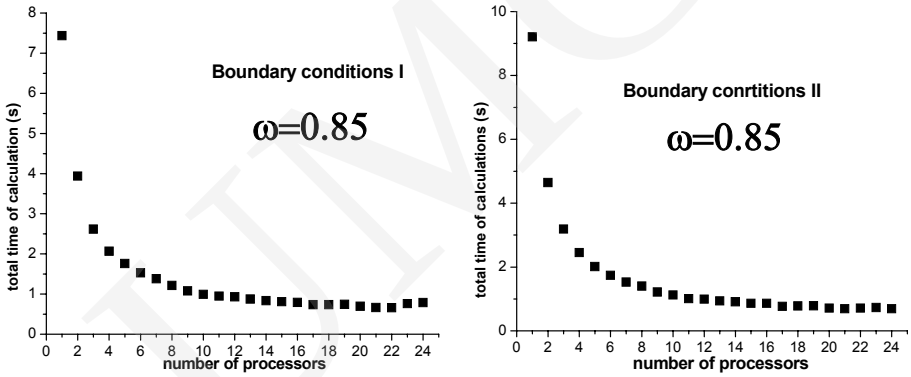

Fig. 6. Total time of solving Poisson equation as a function of a number of used processors. Iteration converges more slowly in case of boundary condition II

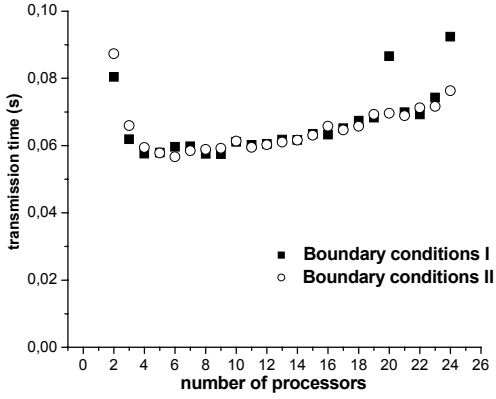

Fig. 7. Data transmission time while solving of Poisson equation

Every process needs to be exchanged approximately 8 MB of data during one loop run, in the case of SMP class machine the transmission time is nearly constant. Hence, transmission delays cannot affect the total time of calculations.

The major factor is the increasing number of iteration loop runs, as one can see in Figure 8 for both cases of boundary conditions.

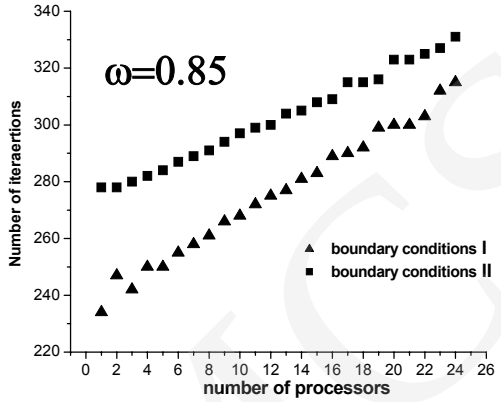

Fig. 8. Number of iteration the SOR method needs to converge as a function of the number of used processors

The number of iteration loops grovs despite the fact that the subdomain size gets smaller. This is not surprising when one keeps in mind how the speed of SOR iteration depends on the overrelaxation parameter ω. There is an optimal value ω_0 and the number of iterations increases very quickly for ω greater than ω_0. The optimal value ω_0 depends mainly on the size of the grid. The larger is the number of used processors, the smaller are subdomains, and $\omega = 0.85$ becomes more and more distant from the optimal ω_0. If the overrelaxation parameter remains unchanged there is no point in using a large number of processors. It may even happen that SOR iteration fails to converge, as shown in Figure 9.

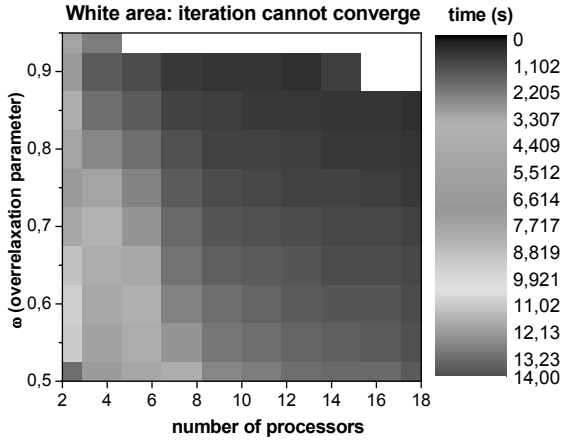

Fig. 9. A map showing the dependence of SOR computation time on the number of processors and the chosen overrelaxation parameter. If it is chosen too big, the SOR metod cannot converge

What we can see in Figure 9 is a kind of valley with one slope rather flat and the other extremely steep. The ω value very close to the optimal one in the case of N processors may be far too large for N+1 or N+2 processors, as shown in Figure 10.

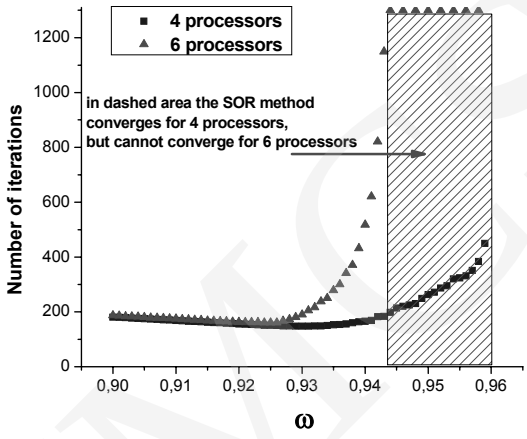

Fig. 10. The value vs. the number of iterations for 4 and 6 processors

Finally, we conclude that ω value should be carefully adjusted, e.g. using the simplest trial-and-error method.

3. Conclusion

In the sequential version of TRQR code solving the Poisson equation takes approximately 15-20 % of CPU time, depending on plasma concentration. Using parallel methods enables us to reduce this time by the factor 6-7 in the case of 8 processor job queue, which is quite satisfactory. The next step of the modification of the TRQR code is the parallelisation of the iteration loop combined with calculations of the plasma particle trajectories.

References

[1] Staebler A., Sielanko J., Goetz S., Speth E., Fusion Technology, 26(2) (1994) 145.
[2] Sielanko J., Muszyński M., Electron Technology, 30(4) (1997) 352.
[3] Sielanko J., Turek M., Tanga A., Annales UMCS Informatica 2 (2004) 251.