Development of Parallel Software Code for Calculating the Problem of Radiation Magnetic Gas Dynamics and the Study of Plasma Dynamics in the Channel of Plasma Accelerator

Vladimir Bakhtin [1,2,3] [0000-0003-0343-3859], Dmitry Zakharov [1] [0000-0002-6319-5090], Andrey Kozlov [1,2] [0000-0002-6242-0911] and Venyamin Konovalov[1][0000-0002-5106-7622]

[1] Keldysh Institute of Applied Mathematics, Miusskaya sq., 4, 125047, Moscow, Russia
[2] Lomonosov Moscow State University, GSP-1, Leninskie Gory, 11999, Moscow, Russia
[3] Bauman Moscow State Technical University, ul. Baumanskaya 2-ya, 5/1, 105005, Moscow, Russia
dvm@keldysh.ru

Abstract. DVM-system is designed for the development of parallel programs of scientific and technical calculations in C-DVMH and Fortran-DVMH languages. These languages use a single parallel programming model (DVMH model) and are extensions of the standard C and Fortran languages with parallelism specifications, written in the form of directives to the compiler. The DVMH model makes it possible to create efficient parallel programs for heterogeneous computing clusters, in the nodes of which accelerators (graphic processors or Intel Xeon Phi coprocessors) can be used as computing devices along with universal multi-core processors. The article describes the experience of successful use of DVM-system for the development of parallel software code for calculating the problem of radiation magnetic hydrodynamics and the study of plasma dynamics in the channel of plasma accelerator.

Keywords: Automation of Development of Parallel Programs, DVM-system, GPU, Fortran, Plasma Accelerator, Radiation Magnetic Hydrodynamics.

1 Introduction

The current level of experimental and numerical studies allows to simultaneously determine the local values of macroscopic plasma parameters and radiation characteristics. This opens up new opportunities for complex research, validation of models and approximation of the results of calculations with the possibilities of experimental studies. This determines the relevance of the development of numerical models of radiation transport, considered as injectors for thermonuclear installations. The study of ionizing gas and plasma flows, respectively, for the first and second stages of quasi-stationary plasma accelerators (hereinafter QSPA) is carried out on the basis of the developed evolutionary models of radiation magnetic hydrodynamics, the effective solution of which required the development of parallel software codes for calculations

on high-performance multiprocessor computer systems. Studies of plasma flows in the second stage of the QSPA on the basis of the model of radiation magnetic hydrodynamics allowed to reveal the conditions providing generation of deuterium-tritium or D-T plasma flows with thermonuclear parameters at discharge currents up to 1 MA in the second stage of the QSPA.

2 Mathematical formulation of the problem

The problem formulation includes a system of MHD equations taking into account the finite conductivity of the medium, thermal conductivity and radiation transport. Under the condition of quasineutrality $n_i = n_e = n$ and equality $\mathbf{V}_i = \mathbf{V}_e = \mathbf{V}$ for fully ionized plasma we have:

$$\frac{\partial \rho}{\partial t} + div(\rho \mathbf{V}) = 0, \quad \rho \frac{d\mathbf{V}}{dt} + \nabla P = \frac{1}{c}\mathbf{j}\times\mathbf{H}, \quad \frac{d}{dt} = \frac{\partial}{\partial t} + (\mathbf{V},\nabla), \tag{1}$$

$$\frac{\partial}{\partial t}(\rho\varepsilon) + div(\rho\varepsilon\mathbf{V}) + P\,div\mathbf{V} = \frac{\mathbf{j}^2}{\sigma} - div\mathbf{q} - div\mathbf{W},$$

$$\varepsilon = 2c_v T, \quad \mathbf{q} = -\kappa\nabla T,$$

$$\frac{\partial \mathbf{H}}{\partial t} = rot(\mathbf{V}\times\mathbf{H}) - c\,rot\frac{\mathbf{j}}{\sigma}, \qquad \mathbf{j} = \frac{c}{4\pi}rot\mathbf{H},$$

here $\rho = m_i n$ is the density, $P = P_i + P_e = 2k_B nT$ is the total pressure, $\mathbf{q}$ is the heat flux, κ is the thermal conductivity, $\sigma = e^2 n_e / m_e \nu_e$ is the electrical conductivity, $\mathbf{W}$ is the density of the radiation energy flux defined by (3) and (4).

The radial plasma current flowing between electrodes and the azimuthal magnetic field provide the acceleration of plasma behind the ionization front due to the Ampere force $\frac{1}{c}\mathbf{j}\times\mathbf{H}$, here $\mathbf{j}$ is the current density in plasma.

As units of measurement we will choose a length of channel L, the characteristic concentration or gas density at the inlet of accelerator channel n_o ($\rho_o = m\,n_o$) and temperature T_o. The characteristic value of the azimuthal magnetic field H_o the inlet to channel is defined by discharge current in device J_p so that $H_o = 2J_p / cR_o$, where R_o is the characteristic radius of channel. These values allow to form the units of pressure $P_o = {H_o}^2 / 4\pi$, velocity $V_o = H_o / \sqrt{4\pi\rho_o}$, time $t_o = L/V_o$ and plasma current $j_o = cH_o / 4\pi L$. The set of the MHD equations in the dimensionless variables contains such dimensionless parameters as the ratio of the characteristic gas pressure to magnetic one $\beta = 8\pi P_o / {H_o}^2$ and $\mathrm{Re}_m = \nu^{-1} = \sigma_o T^{3/2}$, as well as dimensionless values of thermal conductivity and flux $\mathbf{W}$.

Formulation of the problem includes the boundary conditions at the electrodes and at the inlet and outlet of the accelerator channel. We assume that at the inlet of chan-

nel $z = 0$ the plasma is supplied with the known values of the density $\rho(r) = f_1(r)$ and the temperature $T(r) = f_2(r)$. Not solving an additional electric circuit equation it is possible to assume that the current is kept constant and comes into the system only through electrodes. Then at $z = 0$ we have $j_z = 0$ or $r\,H_\varphi = r_o = const$ where $r_o = R_o / L$. The boundary conditions at electrodes $r = r_a(z)$ and $r = r_\kappa(z)$ forming the wall of the channel are based on the assumptions that the electrode surfaces are equipotential ($E_\tau = 0$) and impermeable for plasma ($V_n = 0$). When $r = 0$ we have: $V_r = 0$, $V_\varphi = 0$, $H_\varphi = 0$.

The algorithm for the numerical solution of equations (1) assumes the mapping of computational domain onto a unit square in (y, z) plane using the relation:

$$r = (1 - y)\, r_\kappa(z) + y\, r_a(z). \tag{2}$$

The finite-difference scheme with flux correction is used to calculate the hyperbolic part of the MHD equations. Magnetic viscosity and thermal conductivity calculation is based on the flux variant of the back-substitution method. Quasistationary flows are calculated by the establishment method. The radiation transfer problem was solved within the framework of the developed 3D model.

Rate of the radiation propagation is significantly higher than the specific rates of the plasmodynamic processes. In this case the radiation field instantly adapts to distribution of the flow parameters, and it is possible to reduce the problem to solve the stationary equation of the radiation transport:

$$\mathbf{\Omega} \cdot \nabla I_\nu(\mathbf{r}, \mathbf{\Omega}) = \eta_\nu(\mathbf{r}) - \kappa_\nu(\mathbf{r}) \cdot I_\nu(\mathbf{r}, \mathbf{\Omega}), \tag{3}$$

here $I_\nu(\mathbf{r}, \mathbf{\Omega})$ is radiation intensity with the particular frequency ν, emitting in direction of the spatial angle $\mathbf{\Omega}$, corresponds to the point with the coordinate $\mathbf{r}$. The main radiation characteristics are the density of the radiation energy U and density of the radiation energy flux $\mathbf{W}$ are determined by the radiation intensity:

$$U(\mathbf{r}) = \frac{1}{c} \int_0^\infty \int_0^{4\pi} I_\nu(\mathbf{r}, \mathbf{\Omega})\, d\,\mathbf{\Omega}\, d\nu, \quad \mathbf{W}(\mathbf{r}) = \int_0^\infty \int_0^{4\pi} I_\nu(\mathbf{r}, \mathbf{\Omega})\, \mathbf{\Omega}\, d\,\mathbf{\Omega}\, d\nu. \tag{4}$$

The absorption coefficient $\kappa_\nu(\mathbf{r})$ and emissivity $\eta_\nu(\mathbf{r})$ are known functions that depend on the condition of medium, its density and temperature and are sums of three parts corresponding to a) absorption and emission in lines; b) photo-ionization and photo-recombination; and c) scattering.

In accordance with equations (4) the problem of the radiation transport in flow of the ionizing gas and plasma should be solved in the three-dimensional formulation.

It is easy to obtain a grid for the 3D problem of the radiation transport as rotation the initial grid in plane of the variables (z, r) by 360 degrees around the axis of channel with a certain step. The radiation intensity has to be determined in different directions for the further calculation of the integral values in relations (4) in any node or cell of a three-dimensional grid. For this purpose an additional angular grid is built in the azimuth and polar angles. The splitting of the complete spatial angle $\mathbf{\Omega} = 4\pi$ into elements of an angular grid is made by the method providing the uniform distribution

of rays in all directions. For each node in calculation we use as a rule 440 rays for the complete spatial angle.

The ray tracing is carried out in accordance with the method of the long characteristics in order to determine the points of the crossing of a ray with the faces of cells in the three-dimensional grid and the position of the crossing of a ray with one of the boundaries of the three-dimensional computational domain. The invisible shadow regions are excluded from calculation of the radiation energy flux for the given node of grid in the tracing process of the computational domain by rays emerging from any node of the grid. Coefficients $\kappa_\nu(\mathbf{r})$ and $\eta_\nu(\mathbf{r})$ are calculated by the average value of density and temperature in the center of the cell. Intensity along rays or characteristics passing through any number of homogeneous regions with known coefficients κ_ν и η_ν is determined as a result of addition of solutions for homogeneous areas.

The more detailed formulation of the problem is presented in [1–3].

3 Development of a parallel version of the program

A parallel program code for the study of high-speed plasma flows in the channels of quasi-stationary plasma accelerators is implemented using the DVM-system [4, 5] in the Fortran-DVMH language.

Let's consider the process of parallelization of this software complex. The main difficulty of developing a parallel program for a cluster is the need to make global decisions on the distribution of data and computations taking into account the properties of the entire program, and then perform the complex work of modifying the program and debugging it. The large amount of program code, multi-modularity, multi-functionality makes it difficult to make decisions on the consistent distribution of data and computations.

Incremental parallelization can be used to facilitate the development of a parallel version of the program. The idea of this method is that not the whole program is parallelized, but its parts (areas of parallelization) - additional instances of the required data are created in them, the distribution of this data and the corresponding computations are made. The regions are selected based on the times obtained by profiling the sequential program.

To interact with those parts of the program that have not been parallelized, copy operations are used from original to additional (distributed) data and back. In this way, we can isolate the areas of code we are interested in and parallelize each area separately from the rest of the program. This allows you to test different approaches to data distribution and calculations distribution within the scope, which can significantly change the structure of data storage, but any changes within the scope will not require any modifications of the code outside the scope. At the same time, it is much easier to find the most optimal distribution of data and computations in a small local area than for the whole program.

After parallelizing individual areas, parallel program optimization is performed to minimize the amount of data copied. To do this, several regions can be combined into

one if an effective common distribution can be found for a common area. It is also possible to add less time-consuming fragments to the area if this reduces the number of data copying operations. This allows you to minimize the amount of program code for analysis and parallelization-it is enough to consider only those fragments that work with data already used within the parallelized area.

Having optimal solutions for distributing data and calculations in local areas makes it a little easier to find a common distribution for the combined area. However, in the case where the optimal global distribution cannot be found for the combined domain, incremental parallelization makes the process of finding local optimal distributions much easier. This is very useful if the program consists of several parts, the data in which differ significantly in structure.

To solve this problem, several different mathematical models are used at once, so it was decided to use incremental parallelization, which was successfully applied. The most time-consuming parts of the program were determined using the performance analyzer, which is part of the DVM system. From the point of view of command execution time, the most essential part of the software package was the run_radiat function, which corresponds to the calculation of radiation transfer in the 3D formulation of the problem. In addition, considerable time was required to perform an iteration loop responsible for the calculations of axisymmetric flows based on the MHD model.

We describe the process of parallelizing the main iteration loop and the run_radiat function.

The iteration loop was a regular loop with a fixed number of iterations. The inner loops have been described using labels (Fig. 1):

```
            do 43 L = 1,NZ
               do 43 M = 1,NR
                  HFI(M,L) = HRFI(M,L) / RAD(M,L)
43              continue
```

Fig. 1. Fragment of the source program

For convenience, all internal loops were rewritten without the use of labels, and all arrays used in them were replaced (Fig. 2). The rejection of the use of labels was a purely technical transformation with the aim of increasing the readability of the program and facilitating parallelization.

```
         do L = 1,NZ
           do M = 1,NR
             new_HFI(M,L)=new_HRFI(M,L)/new_RAD(M,L)
           end do
         end do
```

Fig. 2. Fragment of program after conversion

Arrays were replaced as a preparation for further distribution of these arrays in isolation from the rest of the program. Since the iterative loop is only some part of the whole complex, it was necessary to make sure that the distribution of arrays would

not require any changes to the rest of the program. In total, about 45 arrays were replaced in this way. All these arrays, both original and copies, were replaced by dynamic ones, and their selection occurred just before the start of the iterative loop. At that moment, the original arrays were destroyed on the contrary, so as not to consume extra memory (Fig. 3):

```
allocate(new_HRFI)
new_HRFI = HRFI
deallocate(HRFI)
<итерационный цикл>
allocate(HRFI)
HRFI = new_HRFI
deallocate(new_HRFI)
```

Fig. 3. Work with copies of arrays

The above transitions from one array to another were placed before and after the iteration loop if the data available in these arrays were used respectively inside or after the iteration loop. Most of the loops had a fairly simple array access patterns - either using the elements appropriate for loop iteration, or the nearest neighbors of those elements (Fig. 4):

```
do L = 2,NZM1
  do M = 2,NRM1
        FGB = (GAM - 1.) / BB * new_RDY(L) / new_TEM(M,L)
        DRHDY = (new_HRFI(M + 1,L) - new_HRFI(M - 1,L)) / DY2
        DRHDZ = (new_HRFI(M,L + 1) - new_HRFI(M,L - 1)) / DZ2
        ! more code
     end do
   end do
```

Fig. 4. Typical loop in the program

For arrays such as new_HRFI from the example above, the corresponding shadow edges were specified during distribution (SHADOW specification). The vast majority of loops were not closely nested, which did not allow them to be distributed simultaneously in two dimensions. It was also impossible to carry out a distribution for any one distribution, since there were loops with direct dependencies in both one and the other dimensions (Fig. 5):

```
do L = 2,NZM1
  ! more code
  do N = 2,NRM1
    M = NR - N + 1
    WL(M)=((1.-C(M)/A(M)*BET(M))*(B(M)*WL(M+1)-
           F(M))+C(M)*GAMM(M))/ A(M)
    PROR(M) = (A(M) * GAMM(M) + BET(M) * (F(M) -
           B(M) * WL(M + 1)) / A(M)
  end do
  !more code
end do
```

Fig. 5. Dependency loop

Similar loops were in another dimension. A significant number of loops also had a significant amount of access to potentially remote data for one of the dimensions (Fig. 6):

```
do L = 2,NZM1
   ! more code
   PPL = VR1 * new_RAD(1,L) * new_RDY(L) * new_PLT(1,L)
   !more code
end do
```

Fig. 6. Potentially remote data

To solve these problems, it was decided to use two data distribution templates (template specification) - one for each dimension. In one case, it turned out that each process received a part of "rows" of a two-dimensional array and worked with them. In the other-each process received a part of "columns" of the two-dimensional array and processed them. For each loop, a dimension without dependencies and without accesses to remote data was selected, and parallelization was performed on it. Previously created copies of arrays were also distributed to the dimensions on which the loop in which they were used was parallelized. If the array was used in loops, among which there were parallel loops in both one and the other dimension, 2 copies were created for it at once. Between loops parallelized by different dimensions, if necessary, switching between templates was made (Fig. 7):

```
!DVM$ PARALLEL(L) ON new_PLT(*,L), PRIVATE(M)
      do L = 2,NZ
        do M = 1,NR
          ! more code
        end do
      end do
      new2_PLT = new_PLT
!DVM$ PARALLEL (M) ON new2_PLT(M,*), PRIVATE(L)
      do M = 2,NR
        do L = 1,NZ
```

```
              ! more code
           end do
         end do
```

Fig. 7. Switching between templates

There were only 2 serious switches in the program - the transfer of 8 arrays from the distribution on the second dimension to the distribution on the first, and the transfer of the same 8 arrays back. The program required another similar translation back and forth, but only for one array. About 30 arrays were distributed over the created templates, of which about 20 had copies for both distribution templates. Also, more than 50 different loops were parallelized.

In the run_radiat function, 5 loops were parallelized. These loops used a variety of data structures and arrays of unique formats (Fig. 8):

```
do inode = 1,mesh3d%NNodes
        den3d%fval(inode) = 0.e0_r8p
        tem3d%fval(inode) = 0.e0_r8p
        do i = 1,valsInterp%ielem(inode)%nSrc
           n1  = valsInterp%ielem(inode)%iSrc(i)
           eps = valsInterp%ielem(inode)%wSrc(i)
           den3d%fval(inode) = den3d%fval(inode) + den2d%fval(n1)*eps
           tem3d%fval(inode) = tem3d%fval(inode) + tem2d%fval(n1)*eps
        end do
      end do
```

Fig. 8. The use of complex data structures

Since the main arrays in these loops had a unique structure that could not be correlated well with each other, 4 different distribution templates were created for all 5 loops (since 2 out of 5 loops were parallelized using the same template). All distributed arrays were replaced with ordinary arrays of standard types, the remaining arrays and data structures were left in their original form. The final version of the loop from the example above after transformations and parallelization looked like this (Fig. 9):

```
!DVM$ PARALLEL(inode) ON new_den3d(inode),PRIVATE(n1,eps,i)
do inode = 1,m3d
       new_den3d(inode) = 0.e0_r8pч
       new_tem3d(inode) = 0.e0_r8p
       do i = 1,valsInterp%ielem(inode)%nSrc
          n1  = valsInterp%ielem(inode)%iSrc(i)
          eps = valsInterp%ielem(inode)%wSrc(i)
          new_den3d(inode) = new_den3d(inode) + new2_den2d(n1)*eps
          new_tem3d(inode) = new_tem3d(inode) + new2_tem2d(n1)*eps
    end do
 end do
```

Fig. 9. Loop after transformations

The main difficulty in parallelizing this part of the program was the indirect addressing highlighted in the examples above. During any iteration of the loop, the program can access an arbitrary number of different elements of the addressable array, perhaps even all. Moreover, the situation when each process will request almost all the elements of an indirectly addressed array (that is, the values of n1 for each set of turns allocated to the process pass through almost all possible elements of the den2d% fval and tem2d% fval arrays) is normal for this task. Therefore, it was decided to write data to distributed arrays, and if in subsequent loops this array is addressed indirectly, it was redistributed by all processes. Given that every process uses almost every element of the array, the extra overhead of making a full copy is negligible. After the loop indicated above, for example, redistribution took place in the form (Fig. 10):

```
new2_den3d = new_den3d
new2_tem3d = new_tem3d
```

Fig. 10. Copy data after loop execution

The prefix new_ pointed to distributed arrays and new2_ to non distributed arrays. A separate problem was the presence of reduction operation *sum* with indirect addressing (Fig. 11):

```
Do idir = 1,NDirs
    do iEn = 1,NEnGroups
        do isegm = 1,NPoints
            inode = pTrace%rayTree(idir)%segm(isegm)%inode
            !more code
            cUden%fval(inode) = cUden%fval(inode) + cU_ptX
        end do
    end do
end do
```

Fig. 11. Loop with a reduction

In this fragment of the program, when any iteration of the loop is performed, an arbitrary element of the array **cUden%fval(inode)** can be written to, and this data needs to be summed. To solve this problem, a special distributed array was created, allowing each iteration of the loop to carry out the summation in its own "column". After this, the summation over the last dimension of the array was performed (Fig. 12):

```
do idir = 1,NDirs
   !DVM$ PARALLEL(iEn) ON new_cuden3d(*,iEn)
   do iEn = 1,NEnGroups
      do isegm = 1,NPoints
         inode = pTrace%rayTree(idir)%segm(isegm)%inode
         !more code
      new_cuden3d(inode,iEn) = new_cuden3d(inode,iEn) + cU_ptX
      end do
   end do
```

```
end do
!DVM$ PARALLEL(iEn) ON uni_cuden3d(*,iEn), PRIVATE(inode),
!DVM$* REDUCTION(SUM(new2_cuden3d))
do iEn = 1,NEnGroups
   do inode = 1,m3d
      new2_cuden3d(inode)=new2_cuden3d(inode)+ new_cuden3d(inode,iEn)
   end do
end do
```

Fig. 12. Reduction implementation

As a result, each process received a correct copy of the reduction array.

Based on the results of parallelization, a version of the program was obtained that can be run on multi-core clusters.

4 Efficiency of parallel program

Calculations using a parallel version of the program were performed on two multiprocessor high-performance computing systems K-100 (KIAM RAS) and MVS1P5 (JSCC RAS). The execution times of the program in seconds on different number of cores are shown in the Figures 13–14.

The calculation of one time step is shown, determined by the Courant condition in solving the evolutionary problem based on the MHD model. Curves 1 in the figures correspond to calculations using the long characteristics method, which requires the most computational resources in the case of solving a fully three-dimensional problem and the calculation of integral characteristics in all nodes of the three-dimensional coordinate grid. Curves 2 and 3 correspond to the calculations of the integral characteristics of radiation in one plane of variables taking into account the axial symmetry of the flow. Along with the method of long characteristics, the method of short characteristics was implemented, which is answered by Curves 2. The method of short characteristics leads to numerical diffusion in the calculation of the radiation field, and it requires more time to calculate compared to the method of long characteristics, which is represented by the curves 3 in the figures, provided that the integral characteristics of radiation in one plane of variables are calculated taking into account the axial symmetry of the flow.

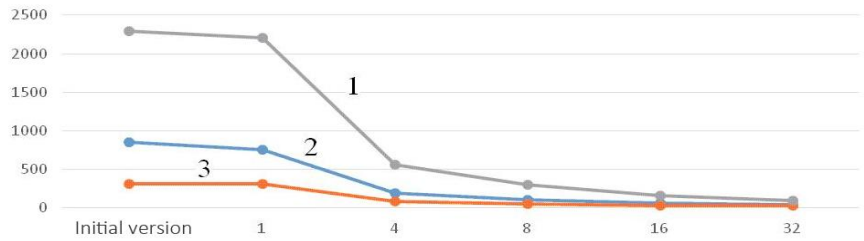

Fig. 13. Program execution time in seconds on a different number of K-100 cores

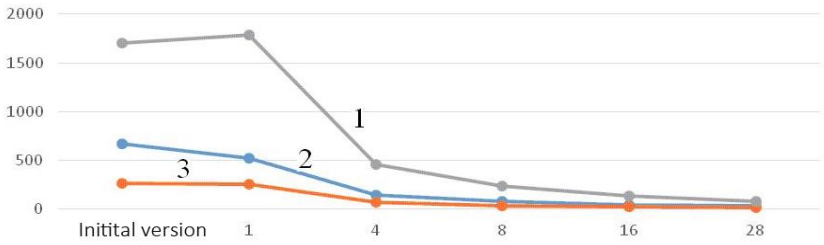

Fig. 14. Program execution time in seconds on a different number of MVS1P5 cores

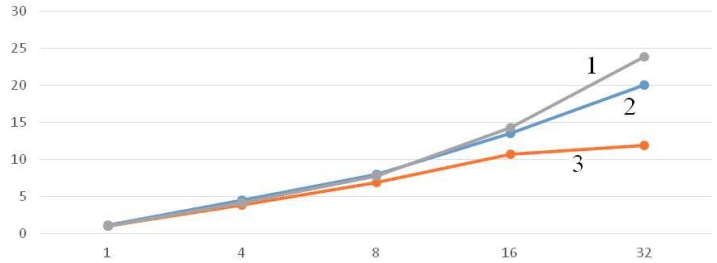

Fig. 15. Acceleration of the program execution process relative to the initial version on a different number of K-100 cores

Figures 15 and 16 show how the acceleration of the parallel version of the program changes relative to the original serial version on a different number of cores of K-100 and MVS1P5 computer systems. Curves 1, 2 and 3 in these figures meet the same calculation conditions specified above.

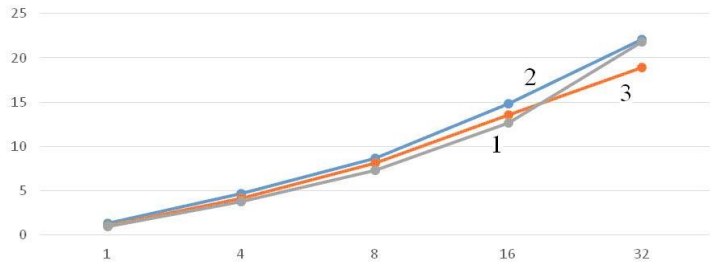

Fig. 16. Acceleration of the program execution process relative to the initial version on a different number of MVS1P5 cores

Weaker acceleration on K-100 for curve 3 in Fig. 15 for 32 processors is explained by the fact that the program execution time is already very short, less than 30 seconds, and overhead costs begin to occupy a significant percentage of the program time.

The acceleration of the parallel version of the program on a single processor can be explained by more optimal memory accesses. Replacing some pointer-based structures with a set of regular arrays has reduced the total number of memory accesses. It

also allowed in some cases to remove indirect addressing that increased the efficiency of the cache memory. The resulting acceleration is significantly dependent on the grid used, the mode of operation of the program and characteristics of memory for a particular machine. It can be enough to cover all the overhead arising from parallelization and even speed up the program on a single processor a little.

Conclusions

This article describes the experience of successful use of DVM-system for the development of parallel software code for calculating the problem of radiation magnetic hydrodynamics and the study of plasma dynamics in the channel of plasma accelerator.

The use of DVM-system allowed to accelerate the process of developing parallel program. At the same time, the following results were achieved to accelerate the program execution process relative to the original version: up to 24 times on 32 processors of the K-100 computer complex (KIAM RAS) and up to 22 times on 28 processors of the MVS1P5 computer system (JSCC RAS).

Numerous calculations performed using the parallel version of the program have shown that the value of the discharge current required to achieve ion energy increases in proportion to the size of the installation. It was found that a decrease in the characteristic plasma concentration at the entrance to the accelerator channel can significantly reduce the values of discharge currents. The values of the discharge current in the plant were determined, providing the ion energy at the output at the level that is necessary for the subsequent d-T plasma synthesis reaction in magnetic traps to hold the plasma. The found discharge currents are quite acceptable for existing QSPA installations.

References

1. Kozlov, A.N., Konovalov, V.S.: Numerical study of the ionization process and radiation transport in the channel of plasma accelerator. Communications in Nonlinear Science and Numerical Simulation, 51, 169–179 (2017).
2. Kozlov, A.N. The study of plasma flows in accelerators with thermonuclear parameters. Plasma Physics and Controlled Fusion, 51(11), Ar. 115004, 1–7 (2017), https://doi.org/10.1088/1361-6587/aa86be.
3. Kozlov, A.N., Konovalov, V.S.: Radiation transport in the ionizing gas flow in the quasi-steady plasma accelerator. Journal of Physics: Conference Series, vol. 946 (2018), https://doi.org/10.1088/1742-6596/946/1/012165.
4. C-DVMH language, C-DVMH compiler, compilation, execution and debugging of DVMH programs, http://dvm-system.org/static_data/docs/CDVMH-reference-en.pdf, last accessed 2019/11/21.
5. Fortran DVMH langauge, Fortran DVMH compiler, compilation, execution and debugging of DVMH programs, http://dvm-system.org/static_data/docs/FDVMH-user-guide-en.pdf, last accessed 2019/11/21.