NATIONAL INSTITUTE FOR FUSION SCIENCE

Development of Three-Dimensional Neoclassical Transport Simulation Code with High Performace Fortran on a Vector-Parallel Computer

S. Satake, M. Okamoto, N. Nakajima and H. Takamaru

(Received - Oct. 24, 2005)

NIFS-828 Nov. 2005

RESEARCH REPORT
NIFS Series

TOKI, JAPAN

Development of Three-Dimensional Neoclassical Transport Simulation Code with High Performance Fortran on a Vector-Parallel Computer

Shinsuke Satake[1], Masao Okamoto[1], Noriyoshi Nakajima[1] and Hisanori Takamaru[2]

[1] National Institute for Fusion Science, Toki, Japan

[2] Department of Computer Science, Chubu University, Kasugai, Japan

Abstract

A neoclassical transport simulation code (FORTEC-3D) applicable to three-dimensional configurations has been developed using High Performance Fortran (HPF). Adoption of computing techniques for parallelization and a hybrid simulation model to the δf Monte-Carlo method transport simulation, including non-local transport effects in three-dimensional configurations, makes it possible to simulate the dynamism of global, non-local transport phenomena with a self-consistent radial electric field within a reasonable computation time. In this paper, development of the transport code using HPF is reported. Optimization techniques in order to achieve both high vectorization and parallelization efficiency, adoption of a parallel random number generator, and also benchmark results, are shown.

keywords : High Perfomance Fortran (HPF), parallel computing

1 Introduction

In research activities on magnetic confined fusion plasma, one of the basic and important issues is to evaluate the confinement performance of the plasma. The loss mechanisms of plasma can be classified roughly into two categories. One is caused by orbit motion of charged particles and their diffusion by Coulomb collision, and the other is caused by several types of instabilities occurring in plasmas such as MHD instabilities and micro-turbulences. In this paper, we focus on the development of transport simulation code for the former transport process, which is called "neoclassical transport"[1, 2] in fusion research activities.

In Fig. 1, two typical configurations of plasma confinement devices are shown. The nested surfaces shown in the figures are called "magnetic flux surfaces", which consist of twisting magnetic field lines around those surfaces. Fig. 1 (a) is a tokamak configuration, which has a symmetry in the toroidal direction, and (b) is a helical configuration of Large Helical Device (LHD)[3] in National Institute for Fusion Science (NIFS), Japan, which is a typical configuration of so-called heliotron devices[4]. The coordinate system (ρ, θ, ζ) (magnetic coordinates) we use here is also shown in Fig. 2. Here, $\rho = \sqrt{\psi/\psi_{edge}}$ is a normalized radius, θ is the poloidal angle, and ζ is the toroidal angle. ψ is the toroidal magnetic flux inside a flux surface ρ =const, and ψ_{edge} is the value of ψ at the outermost surface, respectively. The change of magnetic field strength $|B|$ along a field line is illustrated in Fig. 3. In the tokamak case, the modulation of $|B|$ is caused by its toroidicity. In a heliotron configuration, modulation of $|B|$ caused by helically winding coils is superimposed on the toroidal modulation.

The guiding center velocity of a charged particle can be decomposed as $\mathbf{v} = v_{\parallel}\mathbf{b} + \mathbf{v}_E + \mathbf{v}_d$, where $v_{\parallel}$ is the parallel velocity, $\mathbf{b} = \mathbf{B}/B$, $\mathbf{B}$ is the magnetic field, $\mathbf{v}_E = \mathbf{E} \times \mathbf{B}/B^2$,

(a) (b)

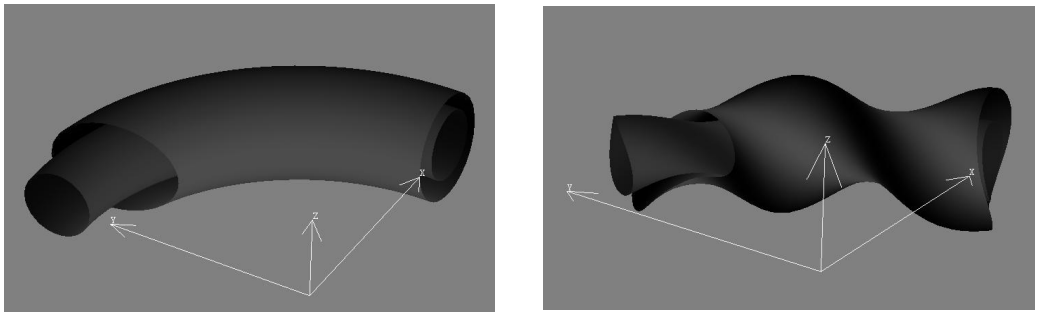

Figure 1: Cut images of the magnetic flux surfaces (a) tokamak (two-dimensional) configuration, (b) Large Helical Device (LHD) configuration.

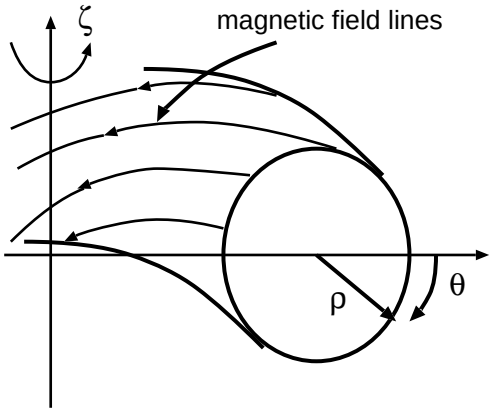

Figure 2: Illustration of a magnetic coordinates (ρ, θ, ζ). Flux surface ρ =const. is formed by twisting magnetic field lines.

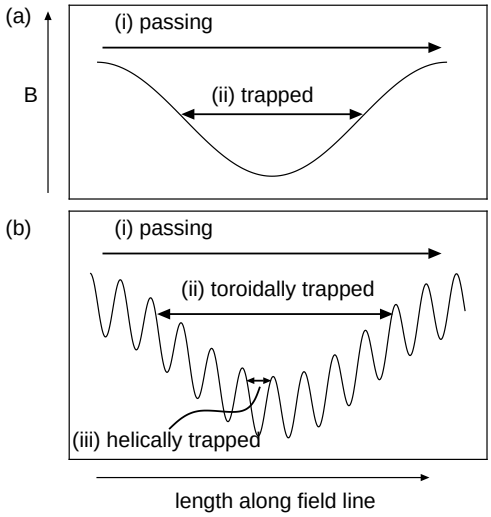

Figure 3: Illustrations of guiding center motions in magnetic field : (a) in a tokamak configuration and (b) in a helical heliotron-type configuration.

$\mathbf{E} = -\nabla\Phi$ is the static electric field in plasma, and $\mathbf{v}_d$ is the drift velocity, which is caused by the gradient and curvature of magnetic field. Since Φ is considered to be constant on a flux surface, i. e., $\Phi = \Phi(\rho)$, the $\mathbf{E} \times \mathbf{B}$ direction is along the flux surface. Therefore, only the drift velocity $\mathbf{v}_d$ has a component in the radial direction. Usually the dominant component of particle motion is the parallel motion $v_\parallel \mathbf{b}$. On the other hand, the magnetic moment $\mu = m{v_\perp}^2/2B$ is a constant of the motion, where ${v_\perp}^2 = v^2 - {v_\parallel}^2$, $mv^2/2 = \mathcal{E} - e\Phi$, and $\mathcal{E}$ is the total energy of a charged particle. Since $\mu B \leq mv^2/2$, particles with large μ are trapped in a weak-B region as illustrated in Fig. 3. Compared with passing particles, trapped particles have a large excursion of orbit in radial direction, called "orbit width" Δ_ρ, caused by the drift motion. Neoclassical transport theory treats the enhancement of transport by these trapped particles in torus plasmas.

The radial electric field evolves in time so that the radial ion and electron particle fluxes Γ_i, Γ_e satisfy the ambipolar condition $\Gamma_i(\rho, E_\rho) = \Gamma_e(\rho, E_\rho)$, where $E_\rho = -d\Phi/d\rho$. Compared with tokamak cases, particle orbit in helical configurations becomes complicated as shown in Fig. 3(b), and it is difficult to be treated exactly by an analytic way. Since neoclassical ion flux in helical plasma strongly depends on $\mathbf{v}_E$, the self-consistent radial electric field by treating properly the particle orbits in a three-dimensional magnetic field is a key to evaluate transport level correctly in helical plasma. Another point which is difficult to treat in analytic way is the non-local effect on transport brought by

the finiteness of orbit width (finite-orbit-width (FOW) effect)[5]-[7]. Conventional neoclassical theory has been established under the assumption of the local transport model (small-orbit-width limit) in which Δ_ρ is treated as a higher-order small value. In order to evaluate precisely the neoclassical transport level in a realistic plasma, the FOW effect is needed to be considered.

To simulate the dynamic transport process and formation of an ambipolar electric field, including non-local effects in helical configurations, we have been developing the δf Monte Carlo code FORTEC-3D[8]. The δf method[9, 10] directly solves the drift-kinetic equation, which describes the time evolution of plasma distribution function, by using a Monte Carlo technique. The outline of the code is explained in Sec. 2. 1. Since electron motion is much faster than that of ions while the FOW effect on electron transport is negligible, it is inappropriate to treat both ions and electrons by the δf method, in practice. In our simulation model, only the ion transport is solved by the δf method, while the electron transport is obtained from a reduced kinetic equation solver GSRAKE[11, 12], which is more compact than FORTEC-3D but does not include the non-local effect. The adoption of this hybrid simulation model enables us to simulate neoclassical transport including the FOW effect of ions and self-consistent evolution of E_ρ within a reasonable computation time.

To carry out a simulation in the full volume of a helical configuration like LHD by the δf method, a large amount of memory and high calculation performance are needed. We have developed FORTEC-3D code by using High Performance Fortran (HPF)[13] in order to achieve both high parallelization and vectorization efficiency as well as to develop the code easily, on the SX-7 supercomputer system (NEC Corporation, Japan) at the Theory and Computer Simulation Center, NIFS. The system has five nodes connected by high-speed network switches, and each node has 32 PE (processor elements) and 256GB memory. The total peak performance is 1412 GFLOPS/160 PE. Parallelization of codes with HPF can be achieved by specifying the data mapping on the distributed memory and instructing the parallel calculation by embedding HPF directives (in the form of `!HPF$ distribute, !HPF$ independent`, etc.), which is easier than programming a parallelized code using Fortran with Message Passing Interface (MPI). We have also adopted a parallelized pseudorandom number generator with a preferable vectorization efficiency because a non-parallelized random number generator will be a bottleneck of Monte Carlo simulation. Since the Monte Carlo method itself is suitable for parallel computing, as it basically treats independent events, we have achieved a high computation performance

on the SX-7 system, as fast as 30% of the peak performance at the full five-node, 160 PE. The technique to optimize the code and some benchmark results, both in two-dimensional and three-dimensional simulations, are shown in Sec. 2. 2 and 2. 3. Section 3 contains a discussion and a summary.

2 Development of FORTEC-3D code with HPF

2.1 Outline of FORTEC-3D code

In FORTEC-3D, the drift-kinetic equation for the deviation of plasma distribution function from local Maxwellian, $\delta f = f - f_M$, is solved by using the two-weight scheme[9, 10]. The flowchart of FORTEC-3D code is shown in Fig. 4. In the two-weight scheme, two weights w and p are introduced which satisfy the relations $wg = \delta f$ and $pg = f_M$, where g is the distribution function of simulation markers. Each marker moves like a charged particle in plasma, and time evolution of marker weights are solved. The effect of collisions is implemented numerically by random kicks of marker velocity. In FORTEC-3D, the time evolution of radial electric field E_ρ is solved according to $\partial E_\rho/\partial t = -e\left[\Gamma_i - \Gamma_e\right]/\epsilon_\perp$, where subscripts i and e describe particle species, $\epsilon_\perp$ is the dielectric constant in the torus plasma. As mentioned in Sec. 1, the hybrid simulation model for evaluating neoclassical particle fluxes is adopted, in which only Γ_i is solved by the δf method while the table of $\Gamma_e(\rho, E_\rho)$, which is prepared by using GSRAKE, is referred to in FORTEC-3D. Since the collisionality of fusion plasma is very low, collision operator is solved once after `nss` times orbit calculation using 4th-order Runge-Kutta method. The field particle operator is introduced to retain the conservation property of the Fokker-Planck collision term. The marker weights w and p are averaged in the phase space $(\rho, \theta, \zeta, v_\parallel, v_\perp)$ to reduce the statistical noise in the simulation. Markers escaped from the simulation region are recycled, and the assignment procedure of new weights for the recycled markers is integrated into the weight-averaging procedure[14]. The procedures with star marks in Fig. 4 contain a part to take some ensemble averages and reflect them on the time evolution of the simulation, which make FORTEC-3D different from a simple Monte Carlo code that treats completely independent phenomena. In parallel computing, as we will explain in the next subsection, communication among parallel processes is needed in these procedures.

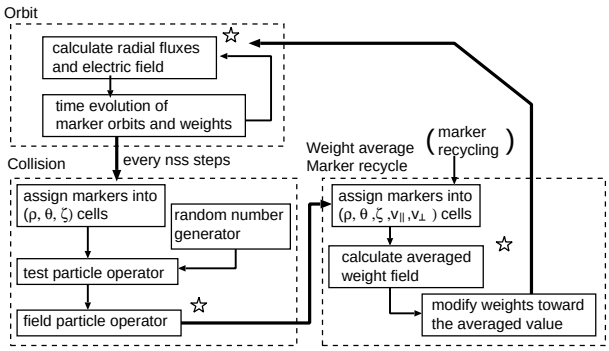

Figure 4: Flowchart of FORTEC-3D code. Procedures with star mark involve reduction calculation and communication between HPF processes.

2.2 Optimization

2.2.1 Parallelized random number generator.

On implementation of test particle collisions in a Monte Carlo way, a long sequence of random numbers $\{X_i\}$ is needed. Though there are many types of pseudorandom number generators used in simulations, we need to use the one which has both a good statistical properties as a random numbers and ability to generate random numbers as fast as possible. Taking account of the above points, we have adopted Mersenne Twister[15] (MT) in FORTEC-3D. We have also adopted the Dynamic Creation scheme[16] to create independent sets of MT, which enables generating independent random number sequences in parallel.

At first, we have tuned the subroutine of MT to achieve high vectorization efficiency. The original source code of MT returns one random number for each calling. We made a subroutine `grnd(rnd,n)` returns `n` sequence of MT random number in the array `rnd(1:n)`. It is known that MT has a long period of pseudorandom number sequence, which is equal to Mersenne prime number $2^p - 1$, where $p = 521,\ 4423,\ 9941,\ 19937$, and so on. It is found that, as the index p becomes larger, the vector length becomes longer. Therefore, we decided to use $p = 19937$ version of MT, which is longest one available for generating 32 bits pseudorandom numbers. The vector length of `grnd` becomes 216 (max=256 on SX-7), and the vector operation ratio is 99.5%.

For parallelization on our SX-7, 160 PE system, we have created 160 data sets which specify the form of recurrences in MT. MT characteristic polynomials for each data set are independent each other, and therefore the sequences of random number are also independent each other. It took about 4 days to create 160 independent MT data sets on

an Athlon XP desktop PC. Subroutine `grnd` is parallelized by using HPF. If one needs to generate total `n` pseudorandom numbers in an HPF code, `grnd(rnd,ni)` is called, where `ni` = `n`/`ncpu`. Then each HPF process refers to different data sets and creates `ncpu` independent sequences of random numbers into `rnd(1:ni,1:ncpu)` at once in parallel, where `rnd` is distributed on `ncpu` HPF processes. The running time of parallel MT to generate total 1.924×10^7 random numbers is shown in Fig. 5, where each PE generates 1.924×10^7/`ncpu` numbers. It can be seen that the overhead time cost for parallelizing MT by HPF is small even if `ncpu` becomes larger. Because of the good parallelization efficiency of the `grnd` routine, the ratio of time consumption of `grnd` on the total simulation time of FORTEC-3D is suppressed as small as 0.2%. The statistical independence of parallelized MT random numbers was not evaluated well. On purpose to check it, we devised a test scheme, named "checker-board test", which can check the independence of two sequences of random numbers $\{X_i\}$ and $\{Y_i\}$ which are uniformly distributed, as illustrated in Fig. 6. In the test, $N = n \times n$ cells are considered, and the flag at the cell (X_i, Y_i) is turned on at i-th step and the number of cells $m(i)$ which are remain off is counted. If $\{X_i\}$ and $\{Y_i\}$ are independent, the mean and variance of $m(i)$ at i-th step become

$$E[m(i)] = N\left\{1 - \left(\frac{N-1}{N}\right)^i\right\} \simeq N\exp(-t), \tag{1}$$

$$\sigma^2[m(i)] = N\left\{N + (N-1)\left(\frac{N-2}{N}\right)^i - (2N-1)\left(\frac{N-1}{N}\right)^i\right\}, \tag{2}$$

$$d(t) = \{m(t) - E[m(t)]\}/\sigma[m(t)], \tag{3}$$

where t is the normalized time step $t = i/N$. By plotting $d(t)$ as shown in Fig. 7, one can easily check if two sequences $\{X_i\}$ and $\{Y_i\}$ have an inappropriate correlation or not. If $\{X_i\}$ and $\{Y_i\}$ are independent, the amplitude of $d(t)$ will not grow as the time step goes on. We have checked every ${}_{160}C_2$ combinations of independent MT created from Dynamic Creation scheme in this way by using $N = (2^{14})^2$ checker-board and found that there is no combination of random number sequences which has an inappropriate correlation. Moreover, we have also checked some other statistical values such as the average ratio of time steps on which the deviation of $m(i)$ from the mean becomes larger than $\pm\sigma$, $\pm 2\sigma$, $\pm 3\sigma$, and the average steps $t_{\rm end}$ to fill all the N cells. The results are shown in Table. 1. One can see that the parallelized MT random numbers follow the expected statistical properties.

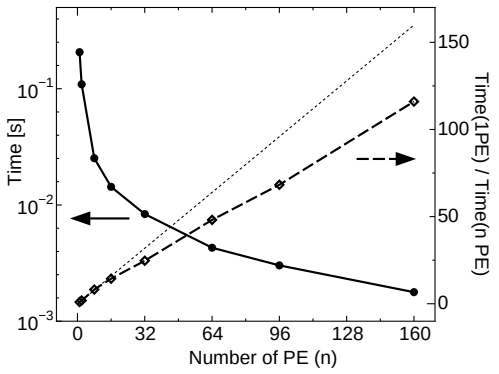

Figure 5: Time consumption of pseudo-random number generators to generate total 1.924×10^7 random numbers by the parallelized Mersenne twisters of which periods is $2^{19937} - 1$. Solid line is the calculation time and dashed line shows the speedup ratio; Time(1 PE) / Time(n PE), respectively. Dotted line is the ideal maximum of the speedup.

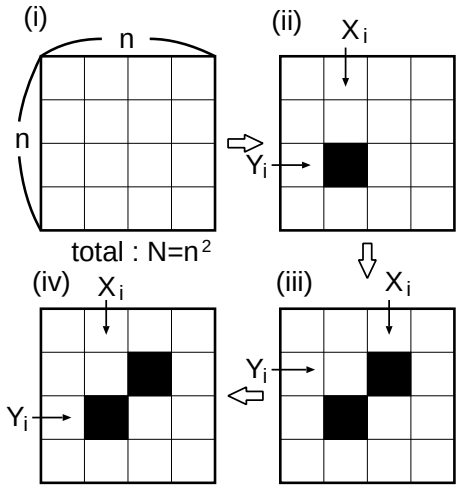

Figure 6: A diagram of the checker-board test. (i) Initially, all the flags of the cells are off. (ii) Generate two independent random numbers $\{X_i\}$, $\{Y_i\}$ and turn on the flag at (X_i, Y_i). (iii) Continue the procedure and count the number of the flags that are still off. (iv) The flag does not change if (X_i, Y_i) hits the cell of which flag has already been turned on.

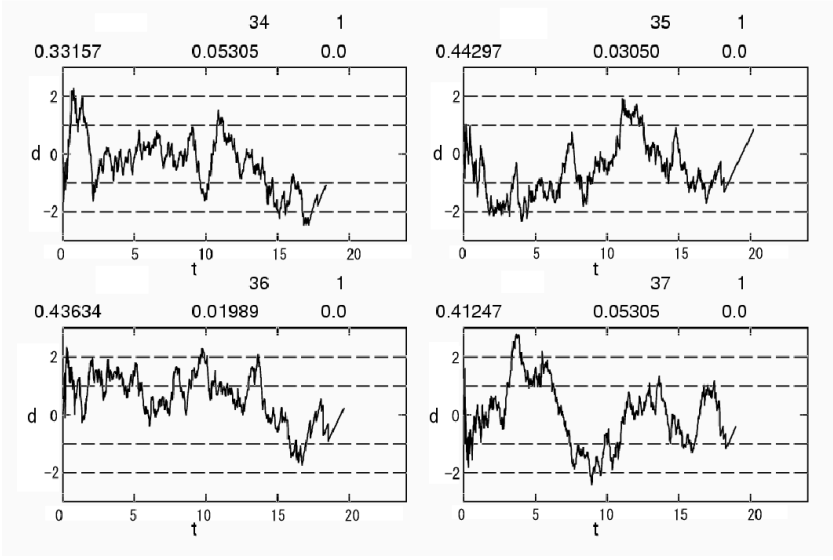

Figure 7: Example of the results of checker-board test (between parallel MT #1 and #34 to 37). The horizontal axis is the normalized time step $t = i/N$ and the vertical axis is the normalized deviation of the number of remaining cells from the mean $d(t) = (m(t) - E[m(t)])/\sigma(t)$. The triplet on each graph are the proportions of time steps on which $m(t)$ deviates $\pm\sigma$, $\pm 2\sigma$, $\pm 3\sigma$ from the mean, respectively.

Table 1: statistical check for parallel MT

$\lvert m - E[m] \rvert \geq$	σ	2σ	3σ	time steps $t_{\rm end}$
test result of ${}_{160}C_2$ parallelized Mersenne Twisters				
	0.3191	0.04588	0.00288	19.992
expected value if two MT are independent				
	0.3173	0.04550	0.00270	19.408

2.2.2 Parallelization of FORTEC-3D.

On SX-7 5 node system, there are two cases to parallelize a code, as shown in Fig. 8. In Fig. 8(a), an HPF process is executed on a PE with a distributed memory. On the other hand, in the model (b), an HPF process is executed on multiple PEs with a shared memory, and each HPF process is further parallelized among those PEs which shares the memory. The shared memory parallelization in an HPF process is automatically done (or using embedded directives like `!CDIR PARALLEL DO`) by SX-7 Fortran compiler. Since non-distributed arrays in HPF program case (b) is shared by several PEs, memory consumption and the number of communication events in the model (b) is expected to be smaller than in the model (a). If we adopt the model (b), however, it is needed to optimize the ratio of HPF processes to shared memory processes in a node (for example 16 : 2, 8 : 4, etc.) to achieve the best running performance of a parallelized code, and coding such a hybrid parallel program is more difficult than in the one-by-one model (a) to tune the parallelization. Moreover, there is a possibility that the bank-conflict will increase in the shared memory parallelization. Therefore, we have adopted the model (a) for FORTEC-3D with HPF. The consumption of memory in the one-by-one model would be a problem if one is running a code which has large non-distributed arrays. We will check the memory usage in FORTEC-3D in the next subsection.

In FORTEC-3D, all the parallel procedures and data distribution are assigned according to the index of markers. In Fig. 9, an example source of HPF code is shown, in which the total marker number is `ntot`, and the number of HPF processes is `ncpu`. We explicitly added a extra dimension for parallelization to help the compiler to recognize the structure of the source code. In the sample code, data distribution is defined by `!HPF$` directives, where `!HPF$ TEMPLATE` and `!HPF$ DISTRIBUTE` make a template pattern to distribute arrays where `dst` is only a dummy array for the template. Then `!HPF$ ALIGN` instructs

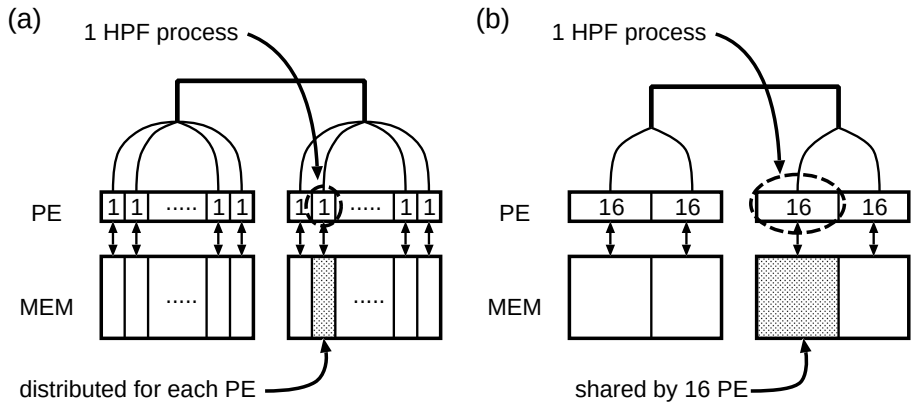

Figure 8: A diagram of distributing processors and memory in running HPF parallelized code. (a) an HPF process is executed on a PE with a distributed memory.(b) HPF processes with shared memory parallelization on each HPF process. Note that the diagram is truncated to 2 nodes though actual system has 5 nodes.

that the second dimension of arrays `w(:,:)` and `p(:,:)` are distributed in the same way as the template `dst`. Packing both the declarations of arrays and `!HPF$` directives for distribution together into a module is an easy way to write an HPF code; otherwise, one has to write `!HPF$` directives in the beginning of every subroutines that uses distributed arrays in it.

Almost all of the communication occurring in FORTEC-3D is related to reduction calculations to take some moments of marker weights, as shown in Fig. 9, where `w(ni,ncpu)` and `p(ni,ncpu)` are summed up from all of the HPF processes to `wpsum(1:2)` at the do loop with the directive `!HPF$ INDEPENDENT, REDUCTION`. Procedures that have reduction calculation are marked in the flow-chart in Fig. 4. An optimization of communication is taken in this example by summing up `w` and `p` not into a separate variable, as in the original source, but into the same array `wpsum`. It serves to pack the data to be communicated, and to reduce the overhead time on the communication.

Optimization for vectorization has also been taken. One of the most effective tunings concerns orbit calculation, because about 80% of the total simulation time was consumed in this procedure. In the Runge-Kutta routine to solve the marker motion, magnetic field data on each marker's position $(\rho_i, \theta_i, \zeta_i)$ given by the form $B = \sum_{m,n} B_{m,n}(\rho)\cos(m\theta - n\zeta)$ need to be referred to. Here, Fourier spectrum data $B_{m,n}(\rho)$ are given as a discrete set of tables on the ρ-grid, and each marker refers to the data on the grid that is closest to the marker position. However, marker radial positions $\{\rho_i\}$ are not aligned in the ρ-direction about the marker index number i. Therefore referring of the $B_{m,n}$-table becomes

a random access to memory, which causes the memory bank conflict and makes the vector operation slower. Fortunately, as explained in the Introduction, the marker motion is mainly directed to the field line, and the radial drift $\mathbf{v}_d \cdot \nabla\rho$ is slow. Therefore, It is not needed to refer to a different entry of the $B_{m,n}$-table on every steps in the Runge-Kutta routine until the closest grid position for each marker moves to another grid. To reduce the bank conflict, each marker holds the field data $B_{m,n}(\rho)$ on the closest radial grid, and renews it only when the closest grid has changed. This optimization makes orbit calculation time almost twice faster compared with the original version.

Original source

```
module VARIABLES
   real, allocatable(:) :: w,p
   integer ntot
end module

program main
use VARIABLES
        .
        .
read(5)ntot
allocate (w(ntot),p(ntot))
        .
        .
call reduce_wp(wsum,psum)
        .
        .

subroutine reduce_wp(wsum,psum)
use VARIABLES
wsum=0.0
psum=0.0
do i=1,ntot
    wsum=wsum+w(i)
    psum=psum+p(i)
end do
return
        .
        .
        .
```

HPF parallel source

```
module VARIABLES
   real, allocatable(:,:) :: w,p
   integer ntot,ni
   parameter :: ncpu=32
!HPF$ TEMPLATE dst(ncpu)
!HPF$ DISTRIBUTE dst(block)
!HPF$ ALIGN (*,i) with dst(i) : w,p
end module

program main
use VARIABLES
real wpsum(2)
        .
read(5)ntot
ni=ntot/ncpu
allocate (w(ni,ncpu),p(ni,ncpu))
        .
        .
call reduce_wp(wpsum)
        .
        .
subroutine reduce_wp(wpsum)
use VARIABLES
real wpsum(2)
wpsum(:)=0.0
!HPF$ INDEPENDENT, REDUCTION(+:wpsum)
do nd=1,ncpu
   do i=1,ni
      wpsum(1)=wpsum(1)+w(i,nd)
      wpsum(2)=wpsum(2)+p(i,nd)
   end do
end do
return
        .
        .
```

Figure 9: Example of HPF source code and its original code to parallelize some reduction calculations. Here, `ntot` is the total marker number and `ncpu` is the total number of HPF processes, respectively. This is a calculation of the sum of marker weights `w` and `p`, which is a typical procedure in FORTEC-3D.

2.3 Benchmark results

In this section, benchmark results of simulations, both in 2D (tokamak) and 3D (LHD) configurations, are shown. The total marker number is `ntot` $= 1.344 \times 10^7$ for 2D cases and 3.072×10^7 for 3D cases. The radial electric field is calculated on 60 radial mesh points. $(20, 20, 1)$ meshes in (ρ, θ, ζ) for 2D cases ($(20, 20, 10)$ for 3D cases) and $(20, 10)$ meshes in $(v_\parallel, v_\perp)$ are used. To benchmark FORTEC-3D by changing number of nodes, we have chosen a somewhat small `ntot` here compared with that used in a practical run. In table 2, number of nodes and PEs, marker number per 1 PE (`ni`), total simulation time, total FLOP count, and communication time are shown. These values were measured by using FTRACE and PROGINF run-time options of the Fortran compiler on SX-7. The total performance of floating point operation and total simulation time on each run are shown in Fig. 10, where the GFLOPS value is total FLOP count/total simulation time. One can see that the GFLOPS value is almost linearly growing with the number of PEs, which indicates the good efficiency of parallelization of FORTEC-3D with HPF. The vector length and vector operation ratio of the benchmark runs observed by FTRACE are about 254 and 98.0%, and they hardly change as the number of PEs changes. Therefore, it can be said that HPF has good affinity for a vector computer. The maximum performance of FORTEC-3D reaches 369 GFLOPS on run #9. In fact, GFLOPS value becomes higher if more markers are used, because vector length becomes longer while the amount of data communicated does not change much by changing `ntot`. The fastest run we had ever done was the one in which `ntot` $= 6.4 \times 10^7$ was used in a 3D case, and it reached 417GFLOPS, which is about 30% of the peak performance.

Next, the ratio of communication time to the total simulation time is shown in Fig. 11. It becomes larger as the number of PEs becomes larger. Although we have optimized the reduction communications that appear in FORTEC-3D by packing the communication data, the increase of the time consumption for communication in simulation runs with many HPF processes is inevitable. It is to be noted that the ratio is very large in run #1, in which only one node is used. It seems that the bank conflict or the imbalance of calculation time among HPF processes at the orbit calculation increases in run #1, which results in the increase of waiting time of the reduction communication at the subroutine that solves the time evolution of radial electric field. We have not found the reason why the bank conflict or the imbalance of calculation time increased so much in run #1, and the problem is still under investigation.

Finally, the memory usage of FORTEC-3D is shown in Fig. 12. One can see that it

Table 2: Description of benchmark runs

two-dimensional case, `ntot` $= 1.344 \times 10^7$					
	run #1	run #2	run #3	run #4	run #5
# of node	1	2	2	3	5
# of PE (ncpu)	32	32 (16×2)	64	96	160
`ni`($\times 10^4$)	42	42	21	14	8.4
tot. time (s)	3194	2499	1511	1126	762
tot. FLOP count ($\times 10^{14}$)	2.259	2.259	2.260	2.261	2.262
comm. time (s)	502	71	157	206	195
three-dimensional case, `ntot` $= 3.072 \times 10^7$					
		run #6	run #7	run #8	run #9
# of node		2	2	3	5
# of PE (ncpu)		32 (16×2)	64	96	160
`ni`($\times 10^{14}$)		96	48	32	19.2
tot. time (s)		8563	5208	3500	2309
tot. FLOP count ($\times 10^{14}$)		8.247	8.255	8.264	8.280
comm. time (s)		101	412	412	365

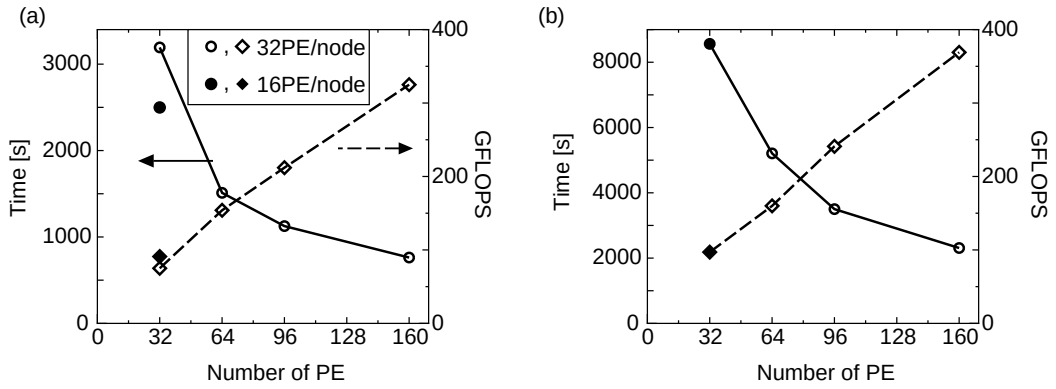

Figure 10: Total simulation time (solid line) and GFLOPS value (dashed line) of the benchmarks for the two-dimension case (a) and three-dimension case (b). Filled marks are the results of run #2 and #6 in the table 2 in which only 16 of 32 PE in a node were used, while in the other runs full 32 PE in each node were used.

changes proportional to the number of PEs. Since the total number of markers is fixed in changing the number of PEs, the total amount of memory used in the simulation is expected to be described in the following form

$$\text{total memory (GB)} \ = \alpha \times \texttt{ntot} + \beta \times \texttt{ncpu}, \tag{4}$$

where the coefficients α and β are related to the memory usage by distributed data (*ex.* marker positions and weights) and non-distributed common data (*ex.* the tables of magnetic field $B_{m,n}$ and electron flux $\Gamma_e(\rho, E_\rho)$; in HPF, all the variables without distributing assignments are copied into the part of the memory region to which each PE refers to), respectively. From the benchmark results, we found that $\alpha = 1.4 \times 10^{-5}$, and $\beta = 2.26$ ($= 2.36$) for the 2D (3D) case. In FORTEC-3D, memory consumption by the distributed data; that is, $\alpha \times \texttt{ntot}$, is large. Therefore, using the one-by-one model explained in Fig. 8 to run the HPF code does not bring about a problem of memory shortage.

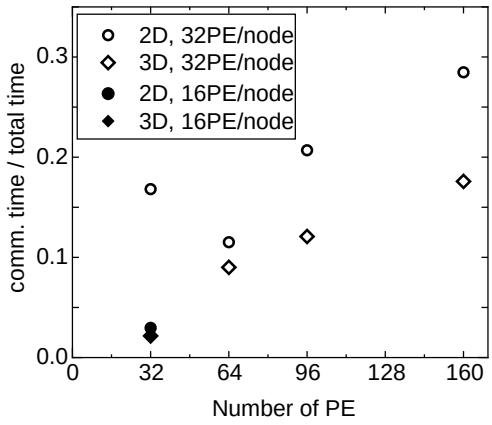

Figure 11: Ratio of communication time over total simulation time.

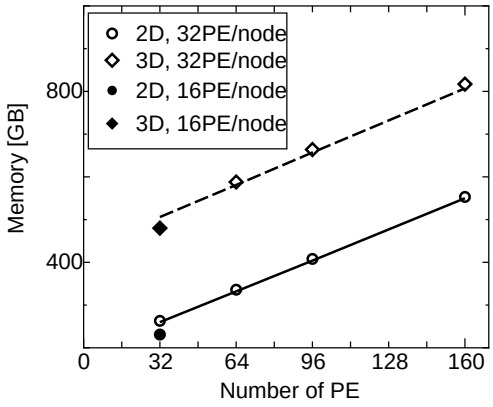

Figure 12: Memory used in the benchmark calculations. Solid and dashed lines are obtained from the fitting formula eq. (4).

3 Summary

In this paper, we have reported the development of a neoclassical transport code FORTEC-3D on a vector-parallel supercomputer. It is shown that, because of the good parallelization efficiency of the code written in HPF language and the high affinity of HPF language for vector supercomputer, we have succeeded to obtain high performance in running a large scale simulation on the multi-node system of SX-7 supercomputer. We have also adopted and optimized a scheme to generate parallel random number generator to avoid one of the bottlenecks for parallelized Monte Carlo simulation. Owing to the high performance of FORTEC-3D, we can investigate several issues in neoclassical transport phenomena in torus plasmas such as non-local effect on transport and time evolution of radial electric field in a complicated three-dimensional configuration[8, 14, 17], which cannot be treated properly by the other conventional methods.

We have benchmarked the memory usage in the HPF code which assigns one HPF process to one PE in Sec. 3. Although we have not suffered from shortage of memory so far, some measures should be taken to reduce the memory consumption if we develop the transport code further. Moreover, the largeness of the number of HPF processes would degrade the parallelization efficiency because of the increase of the communication time. In future, we will transform our code by adopting a hybrid parallel model using both HPF and shared memory parallelization as shown in Fig. 8 (b), and the performance of these codes will be compared with the present version of FORTEC-3D.

Acknowledgements

One of the authors (S. S.) would like to thank Prof. C. D. Beidler in Max-Planck Institute for offering us GSRAKE code, and Mr. Yasuharu Hayashi in NEC Corporation for offering helpful information about HPF compiler and SX-7 system. This work is performed under the auspices of the NIFS Collaborative Research Program, No. NIFS05KDAD004 and No. NIFS05KNXN040.

References

[1] P. Helander and D. J. Sigmar, *Collisional Transport in Magnetized Plasma* (Cambridge Univ. Press, Cambridge, UK, 2002).

[2] R. Balescu, *Transport Process in Plasmas* vol. 1 and 2 (Elsevier Science Publishers, Amsterdam, Netherlands, 1988).

[3] A. Iiyoshi *et al.*, Fusion Technol. **17**, 169 (1990).

[4] M. Wakatani, *Stellarator and Heliotron devices* (Oxford Univ. Press, New York, USA, 1998), p. 291.

[5] K. C. Shaing, W. A. Houberg and P. I. Strand, Phys. Plasmas **9**, 1654 (2002).

[6] P. Helander, Phys. Plasmas **7**, 2878 (2000).

[7] S. Satake, M. Okamoto and H. Sugama, Phys. Plasmas **9**, 3946 (2002).

[8] S. Satake *et al.*, *Conference Proc. 20th IAEA Fusion Energy Conf.* Villamoura, Portugal, Nov. 1-6, 2004, TH/P2-18 (International Atomic Energy Agency, Vienna, Austria, 2005) (CD-ROM).

[9] W. X. Wang *et al.*, Plasma Phys. Control. Fusion **41**, 1091 (1999).

[10] S. Brunner *et al.*, Phys. Plasmas **6**, 4504 (1999).

[11] C. D. Beidler *et al.*, Plasma Phys. Control. Fusion **37**, 463 (1995).

[12] C. D. Beidler and H. Maaßberg, Plasma Phys. Control. Fusion **43**, 1131 (2001).

[13] High Performance Fortran Forum, *High Performance Fortran Language Specification Version 2.0* (Springer-Verlag Tokyo, 1997)

[14] S. Satake *et al.*, *"Non-local Simulation of the Formation of Neoclassical Ambipolar Electric Field in Non-axisymmetric Configurations"*, submitted to J. Plasma Fusion Res.

[15] M. Matsumoto and T. Nishimura, ACM Transactions on Modeling and Computer Simulation **8**, 3 (1998).

[16] M. Matsumoto and T. Nishimura, *"Dynamic Creation of Pseudorandom Number Generators"*, (Monte Carlo and Quasi-Monte Carlo Methods 1998, Springer, 2000, pp 56–69.)

[17] S. Satake *et al.*, *"Non-local neoclassical transport simulation of geodesic acoustic mode"*, to be published on Nuclear Fusion.