A new framework for integrated simulation model using MPMD approach

A.Takayama, K.Shimizu[1)], Y.Tomita, and T.Takizuka[1)]

National Institute for Fusion Science, Toki,Gifu, Japan
[1)] *Japan Atomic Energy Agency, Naka, Ibaraki, Japan*

(Received: 18 November 2009 / Accepted: 5 April 2010)

We introduce a new effective framework to combine computer codes into integrated simulation model. The framework employs MPMD (Multiple Program Multiple Data) approach and each computer code is written with MPI (Message Passing Interface) library. Adopting MPMD approach makes each computer code independent, which leads to the capability for maintenance and improvement of the integrated simulation model. The validity and usability of this approach is shown through a simple example model, which simulates the integrated divertor code, SONIC .

Keywords: MPMD (Multiple Program Multiple Data), MPI (Message Passing Interface), integrated simulation

1. Introduction

Integrated simulations, such as multiscale simulations, multiple-physics simulations, and so on, have come up and been performed, recently [1, 2, 3, 4]. In such simulations, at least several computer codes based on different physics or different spatio-temporal scales have to be combined with each other. Establishing a simple and useful way to combine computer codes is significant for promoting integrated simulations. Though it is possible to combine computer codes simply by restructuring and bundling together into a single program, such a procedure diminishes the capability for maintenance and improvement of the integrated simulation code. The simple and straightforward approach of unifying program is usable for proving validity of integrated simulation. It, however, is not suitable for continuing model development, especially not for group or team development.

Here we introduce a new effective framework to combine computer codes into integrated simulation model. The framework employs MPMD (Multiple Program Multiple Data) approach and each computer code is written with MPI (Message Passing Interface) library [5], [6]. Adopting MPMD approach makes each computer code independent, or at least not strongly coupled, which leads to the capability for maintenance and improvement of the integrated simulation model. Using MPI enables to execute each code on a computer system suited to it, which might enhance the performance of the integrated simulation execution.

2. MPMD approach

Each computer code in an integrated simulation can be regarded as a function or mapping f_i : $(\boldsymbol{x}_i^{(j)}, \boldsymbol{p}_i) \mapsto \boldsymbol{y}_i^{(j)}$, where $\boldsymbol{x}_i^{(j)}$, $\boldsymbol{p}_i$, $\boldsymbol{y}_i^{(j)}$, and $i \in [1, N]$ are set of input variables, input parameters, output variables for j-th term of sequence, and identifier of functions, respectively. Sets of variables ($\boldsymbol{x}_i^{(j)}$) can be determined by sets of the previous outputs ($\boldsymbol{y}_i^{(j-1)}$), ($i \in [1, N]$) for well-posed problems. In the single program approach all functions f_i are implemented in a single program and all variables $\boldsymbol{x}_j$ are stored into the single program. On the other hand, our MPMD approach separates functions $f_i(\boldsymbol{x}_i, \boldsymbol{p}_i)$, ($i \in [1, N]$) into independent program and uses a master program or master programs in order to exchange and transform variables appropriately.

Grouping

In our MPMD approach simulation codes are separated into several programs. MPI library is employed for communication between programs. Adequate grouping of programs is prefereble for using MPI effectively. Figure 1 is a schematic of grouping method. In this case there are 14 processes, 4 program groups plus master group. The master group governs integrated simulation and communicates with each other program groups. The process with MPI rank 0 in master group is called as 'grand master' process. First of all, we group programs according to an identifier which consists of a character string. Communication between program groups is performed between MPI rank 0 process in each program groups. For the purpose of this inter-group communication, we construct the other MPI group, which we call 'data exchange group'.

Construction of General MPI Datatype

Integrated simulations have many data to be exchanged. In order to exchange such data effectively with MPI library we should make data blocks according to timing of exchanging the data and construct a General MPI Datatype for each data block. Using

author's e-mail: takayama.arimichi@nifs.ac.jp

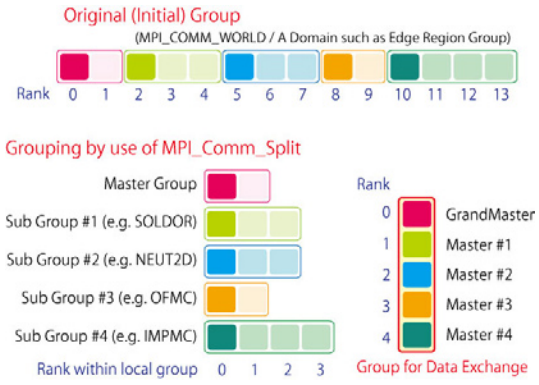

Fig. 1 Schematic view of grouping processes/programs. There are 14 processes, which are grouped into 4 sub (program) groups and a master group. Inter-group communication is performed within 'data exchange group', which consists of process with MPI rank 0 within each local group.

a routine for easy construction of datatype offered in our MPMD library helps this step. The usage is the followings:

```
call initMakeType
call addVariable( time )
call addVariable( pwi  )
call addVariable( pwe  )
call termMakeType( newtype )
```

In this case a general MPI datatype `newtype` containing 3 variables of `time`, `pwi`, and `pwe` is constructed.

Condition message

In order to organize multiple programs, each program should notify the its own condition or status to the grand master process at an appropriate timing. For this purpose, we define some condition messages of named integer constant. They are used as the tag of MPI communication.

`READY` means that the process is ready for calculation and requests the condition of the next time interval.

`DATA_INQUIRE` means that the process requests input data for execution.

`FAST_MODE` means that the process has detected an abrupt change of physical quantity or condition during the calculation.

`CALC_DONE` means that the process has completed the calculation for the requested time interval and sends results.

Other condition messages can be implemented into this framework if necessary.

General MPI Datatype for time related information

In order to exchange the condition of the next time interval a General MPI Datatype of 3 real and a integer variables is defined. 3 real variables are for the current time, the next time interval, and the time step for calculation, and the integer variable is used for a flag. Some bits of the integer variable expressed in binary form have the meaning:

`TERMINATE` indicates termination of the execution.

`WAIT` means a request for some wait.

`FASTMODE` indicates that all process are in the fine time resolution calculation mode described below.

`CALCFAST` indicates whether the time step control is carried out.

Control of execution

We introduce two types of execution control. One is a control of time step. If the control of time step is on, when an abrupt change of physical quantity or condition is detected, we rewind time steps and re-execute with finer time step during several time steps.

The other is a sequence control of program execution. Each program group depends on other program group(s). If all processes are executed in parallel, the dependency could have been violated. When the sequence control of program execution is switched on, the grand master process controls the order of program execution. This control, however, could make processes wait for a while and degrades the efficiency of computation.

Structure of Grand Master process

The aim of grand master is governing integrated simulation and exchanging data. At first initialization routines for using MPMD approach is called. In this step, grouping of programs and construction of exchange-data structures are carried out. Then the grand master process receives initial condition data from the local master process(es) of the other program group(s). The structure of the main calculation loop is as follows:

```
mainLoop: do
 call MPI_Probe(MPI_ANY_SOURCE&TAG..)
 select case( istat(MPI_TAG) )
 case( READY )
  call MPI_Recv(..READY..)
   ! termination condition is satisfied?
   call MPI_Send(..READY..)
   ! all processes are being terminated?
   ! if not, cycle mainLoop
   exit mainLoop
 case( DATA_INQUIRE )
  call MPI_Recv(kDataset...)
  select case( kDataset )
  case( Prog#1 )
   call MPI_Send(time,1,prog#1_Itype..)
  ...
  end select
```

```
 case( FAST_MODE )
  call MPI_Recv(..FAST_MODE..)
 case( CALC_DONE )
  call MPI_Recv(kDataset..)
  select case( kDataset )
  case( Prog#1 )
   call MPI_Recv(time,1,prog#1_Otype..)
   ! store received data
  ...
  end select
  ! all processes have completed?
  ! if not, cycle mainLoop
  call MPI_Barrier(dewld,ierr)
  ! notify whether abrupt change occured
 end select
enddo mainLoop
```

Structure of Local Master process

Each local master process communicates with the grand master process and exchanges data. In local master process initialization routines for using MPMD approach should be called, too. Then it sends initial condition data to the grand master process. The structure of the main calculation loop is as follows:

```
do
 call MPI_Send(..READY..)
 call MPI_Recv(time,1,timType..)
 if(is_msg(t_msg,WAIT)) cycle ! wait
 call MPI_Send(kProg,..,DATA_INQUIRE..)
 call MPI_Recv(time,1,Prog_Itype..)
 call MPI_Bcast ! broadcast to my group
 if(is_msg(t_msg,TERMINATE)) exit
 do i=1,iter
  call pls_exec( kcnd )
  ! abrupt change is detected? -> exit
 enddo
 call MPI_Send(kProg,..,CALC_DONE..)
 call MPI_Send(time,1,Prog_Otype..)
 call MPI_Barrier(dewld,ierr)
 call MPI_Probe(MPI_ANY_SOURCE&TAG)
 if( istat(MPI_TAG)==FAST_MODE )then
  call MPI_Recv(..FAST_MODE..)
  ! rewind data
 endif
enddo
```

3. Example problem

In order to verify the validity of the MPMD approach introduced in section 2, we make a simple example problem, which simulates the SONIC code [1] combined with the OFMC code. The SONIC code is a simulation code package designed for analyzing edge region of 2-dimensional magnetic confinement systems and consists of the SOLDOR code which solves 2-dimensional fluid equations for ion and electron, the NEUT2D code which solves kinetic equations for deuterium neutrals (D and D_2) with Monte Carlo (MC) method, and the IMPMC code which solves kinetic equations for impurity neutrals and ions with MC method. Absorbed power of NBI heating is determined by OFMC code. The transports of plasma, neutral and impurity are solved iteratively.

The simple model equations simulating the SONIC+OFMC code are introduced as follows. They consist of five parts, namely soldor describing the time evolution of total particle number and stored energy, neut2d describing the particle source by global particle confinement, impmc for the total radiation power, and ofmc for the absorbed power of NBI heating.

Model of soldor

The number of plasma particles N and the energy W can be modeled using the following equations:

$$\frac{\mathrm{d}N}{\mathrm{d}t} = -\frac{N}{\tau_{\mathrm{N}}} + S_{\mathrm{N}}, \tag{1}$$

$$\frac{\mathrm{d}W}{\mathrm{d}t} = -\frac{W}{\tau_W} + P_{\mathrm{OH}} + P_{\mathrm{NBI}} - P_{\mathrm{rad}}. \tag{2}$$

Plasma confinement property described by τ_N and τ_W are assumed to be

$$\tau_N = 0.7\tau_W, \tag{3}$$

$$\tau_W = \begin{cases} 0.85[\mathrm{s}] & (\mathrm{OH\ phase}) \\ 0.30[\mathrm{s}] & (\mathrm{L\text{-}mode\ phase}) \\ 0.92[\mathrm{s}] & (\mathrm{H\text{-}mode\ phase}) \end{cases} \tag{4}$$

L→H and H→L mode transition conditions are taken to be $T_{\mathrm{e}} = 0.8[\mathrm{keV}]$ and $T_{\mathrm{e}} = 0.6[\mathrm{keV}]$ at $x = 0.95$, respectively. Here the temperature T_{e} is assumed to have a given profile and the value is determined by the energy of plasma W and the number of plasma particles N.

Model of neut2d

Particle source term in equation (1) is given by the summation of particle flux due to recycling Γ_{recy}, gas puffing Γ_{puff} and neutral beam injection (NBI) Γ_{NBI}. These three fluxes are given by

$$\Gamma_{\mathrm{recy}} = 0.6\frac{N}{\tau_N}, \tag{5}$$

$$\Gamma_{\mathrm{puff}} = \begin{cases} 0.0 & t < 0.25 \\ 0.75 \times 10^{21} & 0.25 \le t < 3.8 \\ \mathrm{density\ control} & 3.8 \le t, \end{cases} \tag{6}$$

$$\Gamma_{\mathrm{NBI}} = \frac{P_{\mathrm{NBI}}}{80\mathrm{keV}}. \tag{7}$$

For $t \geq 3.8$, $\Gamma_{\rm puff}$ is adjusted to fix the plasma density of $n_{\rm e0} = 6.0 \times 10^{19}$.

Model of ofmc

Input power due to NBI $P^{\rm inj}_{\rm NBI}$ and absorbed power into plasma $P_{\rm NBI}$ are modeled by

$$P^{\rm inj}_{\rm NBI} = \begin{cases} 0.0[\rm MW] & t < 0.80 \\ 16.0[\rm MW] & 0.80 \leq t < 4.5 \\ 10.0[\rm MW] & 4.5 \leq t, \end{cases} \tag{8}$$

$$P_{\rm NBI} = \eta P^{\rm inj}_{\rm NBI}, \tag{9}$$

where absorption rate η, which is a function of $n_{\rm e0}$ determined by N, is given by

$$\eta = \begin{cases} 0.4 + (n_{\rm e0}/10^{19} - 2.0) \cdot 0.2 & n_{\rm e0} < 4 \times 10^{19} \\ 1.0 & n_{\rm e0} \geq 4 \times 10^{19}. \end{cases} \tag{10}$$

Model of impmc

Radiation loss term $P_{\rm rad}$ in equation (2) is modeled by the summation of radiation loss due to carbon and argon impurity;

$$P_{\rm rad\_C} = 0.2(P_{\rm OH} + P_{\rm NBI}), \tag{11}$$

$$P_{\rm rad\_Ar} = P_{\rm Ar}(t)\left(1 + 0.5\frac{T_{\rm e0}}{10}\right). \tag{12}$$

Here a level of the radiation loss power due to argon is given by

$$P_{\rm Ar}(t) = \begin{cases} 0 & t < 2.3 \\ 1[\rm MW] & t \geq 2.3. \end{cases} \tag{13}$$

4. Solution of example problem with MPMD approach

We solve the example problem given by equations (1)-(13) by use of the MPMD approach described in section 2. The solution is shown in figure 2. In this solution, seven MPI processes are used. Each one process is for master, soldor, neut2d and ofmc, and three processes are for impmc.

The three impmc processes are for local-master, radiation loss due to carbon and radiation loss due to argon. The local-master of impmc communicates with grand-master process, distributes preconditions to and gathers solutions from two processes used for carbon and argon.

The soldor process receives N, W, τ_N and τ_W of soldor previous results, S_N determined by the particle flux from neut2d, $P_{\rm NBI}$ of ofmc, and $P_{\rm rad}$ of impmc, solves eqs.(1-4), and returns N, W, τ_N and τ_W. $P_{\rm OH}$ is given as an input parameter.

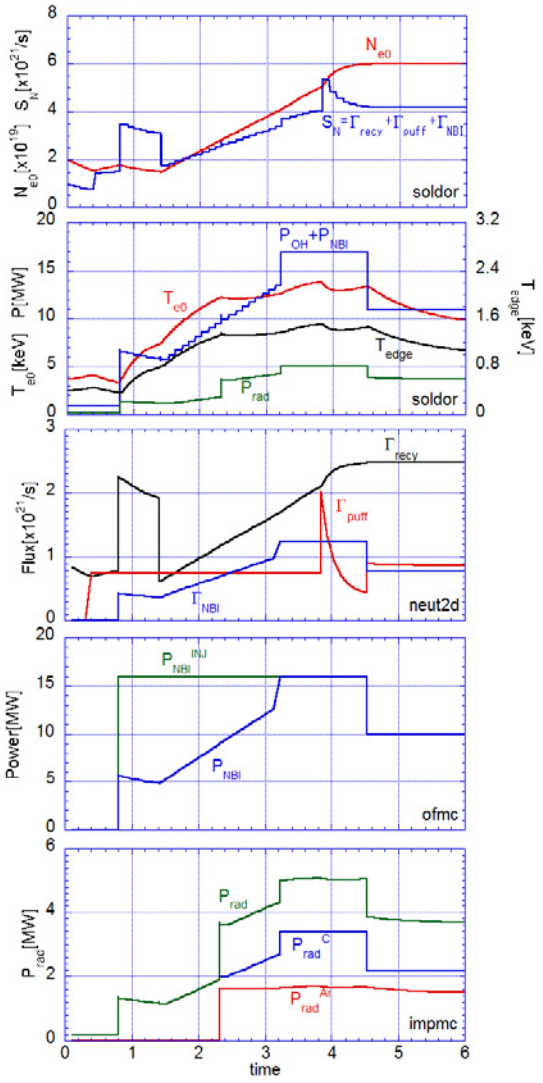

Fig. 2 Solution of example problem by use of the MPMD approach with the control of time step and the sequence control of program execution. These figure shows plasma density, temperature, fluxes, input and absorbed power, and radiation loss obtained by soldor, neut2d, ofmc, impmc process, respectively.

The neut2d process receives N, τ_N of soldor and $P_{\rm NBI}$ of ofmc, solves eqs.(5-7), and returns $\Gamma_{\rm recy}$, $\Gamma_{\rm puff}$, $\Gamma_{\rm NBI}$.

The ofmc process receives $n_{\rm e0}$ of soldor, solves eqs.(8-10), and returns $P_{\rm NBI}$. Input power due to NBI described by eq.(9) is given as a parameter.

The impmc group receives $P_{\rm OH}$, $P_{\rm NBI}$, and $T_{\rm e0}$ of soldor, solves eqs.(11-13), and returns $P_{\rm rad}$.

The time interval of gathering all processes ΔT is 0.1[s] for usual case. The time step for each calculation Δt is 0.02[s] for soldor, and 0.1[s] for the others, which equals to the gathering time interval. When abrupt change of physical quantity or condition, fine

time resolution iteration with $\Delta T = \Delta t = 0.01$[s] is carried out five times. After the fine time resolution calculation, ΔT and Δt are turned back.

At $t = 0.800$ NBI begins as equation (9). ofmc detects the change and all processes move to the finer time step mode. The jump of $P_{\rm NBI}^{\rm inj}$ causes jumps of $P_{\rm NBI}$ and $\Gamma_{\rm NBI}$, which leads to the increase of the temperature $T_{\rm e0}$. After the fine time resolution calculation of five times iteration, all processes return to the normal time step mode.

At $t \sim 1.40$ $T_{\rm edge}$ exceeds 0.8keV and the system undergoes a transition to H-mode, which is detected by soldor and all processes move to the finer time step mode. Using the finer time step, $T_{\rm edge}$ is found to be lower than 0.8keV, which means that the transition to H-mode in the normal time step calculation is mis-predicted one. During finer time step calculation, $T_{\rm edge}$ exceeds 0.8keV at $t = 1.417$ and the system undergoes a transition to H-mode. This causes the drop of $\Gamma_{\rm recy}$, and then density $N_{\rm e0}$ begins to increase.

At $t = 2.320$ argon injection begins, which is detected by impmc and all processes move to the finer time step mode. $P_{\rm rad\_Ar}$ jumps, which degrades tempereture increase. After the fine time resolution calculation of five times iteration, all processes return to the normal time step mode.

At $t = 3.225$, $N_{\rm e0} > 4.0 \times 10^{19}$ and this causes the jump of η. soldor detects this rapid change. At $t = 3.830$ neut2d detects the beginning of the density control and jump of $\Gamma_{\rm puff}$. At $t = 4.535$ ofmc detects the change of $P_{\rm NBI}^{\rm inj}$. At the three timings, all processes move to the finer time step mode once again. After the fine time resolution calculation of five times iteration, all processes return to the normal time step mode.

5. Concluding remarks

As shown in the previous section, our MPMD approach successfully carried out the calculation. Independent programs, each of which solves different equations, cooperates through the intermediation of the grand master process.

The two types of execution control, that is, the control of time step and the sequence control of program execution, also works well. Figures 3 and 4 are the case without the control of time step or the sequence control of program execution, respectively. In those cases simulation conditions except for the execution control are same as the case in section 4. These figures show that lack of execution control alters or degrades the solution. Without the sequence control of program execution, the previous time interval data is used in some cases, or the synchronization of data could be lost. This is because the degradation of the solution occurs.

This indicates that appropriate execution control is required for the MPMD approach introduced in this article for the accuracy and efficiency of calculation.

Acknowledgment

The authors acknowledge Prof. A. Fukuyama at Kyoto University and Prof. M. Yagi at Kyushu University for intensive discussions and comments.

[1] K. Shimizu, T.Takizuka, K. Ohya, K. Inai, T. Nakano, A. Takayama, H. Kawashima and K. Hoshino, Nucl. Fusion **49** (2009) 065028.

[2] A. Fukuyama, S. Murakami, M. Honda, Y. Izumi, M.Yagi, N. Nakajima, Y. Nakamura and T. Ozeki, *Proc. of 20th IAEA Fusion Energy Conf.* (Villamoura, Portugal, 2004) IAEA-CSP-25/CD/TH/P2-3.

[3] M. Sato, S. Toda, Y. Nakamura, K.-Y. Watanabe, A. Fukuyama, S. Murakami, M. Yokoyama, H.Funaba, H. Yamada and N. Nakajima, Plasma Fusion Res. **3** (2008) S1063.

[4] B. Guillerminet, M. Airaj, P. Huynh, G. Huysmans, F. Iannone, F. Imbeaux, J. B. Lister, G. Manduchi, P. Strand and the contributors to the European Task Force on Integrated Modelling Activity, Fusion Eng. Des. **83** (2008) 442.

[5] Y. Aoyama, 'Introductory Course in Parallel Programming (MPI version)' (in Japanese), available from http://accc.riken.jp/hpc/training/text.html

[6] P.S. Pacheco, 'Parallel Programming with MPI', (Morgan Kaufmann Publishers, 1997); Japanese translation (Baifukan, 2001)

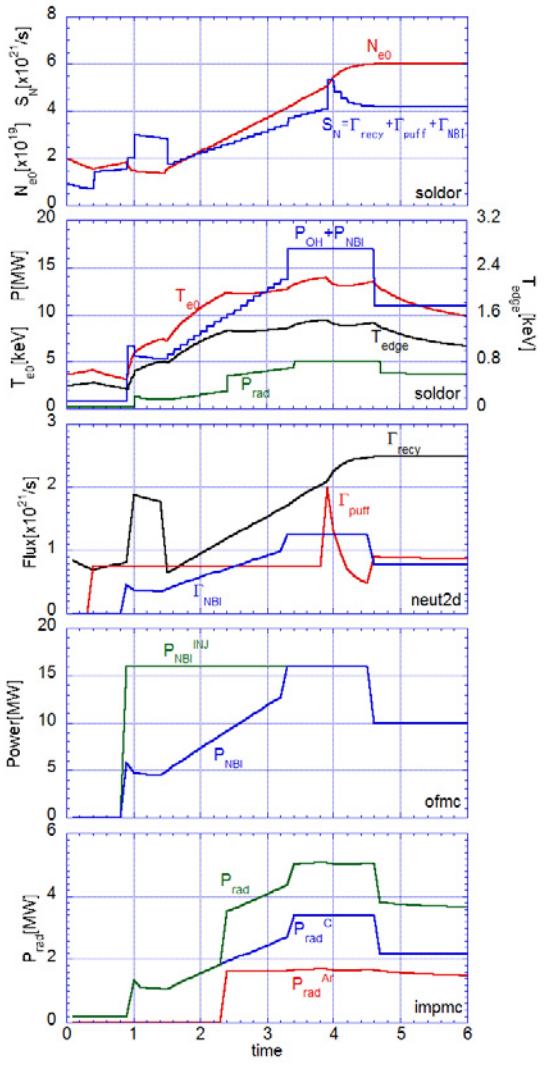

Fig. 3 Solution of example problem by use of the MPMD approach with the sequence control of program execution. The difference between this figure and figure 2 is in that the control of time step is switched off in this case.

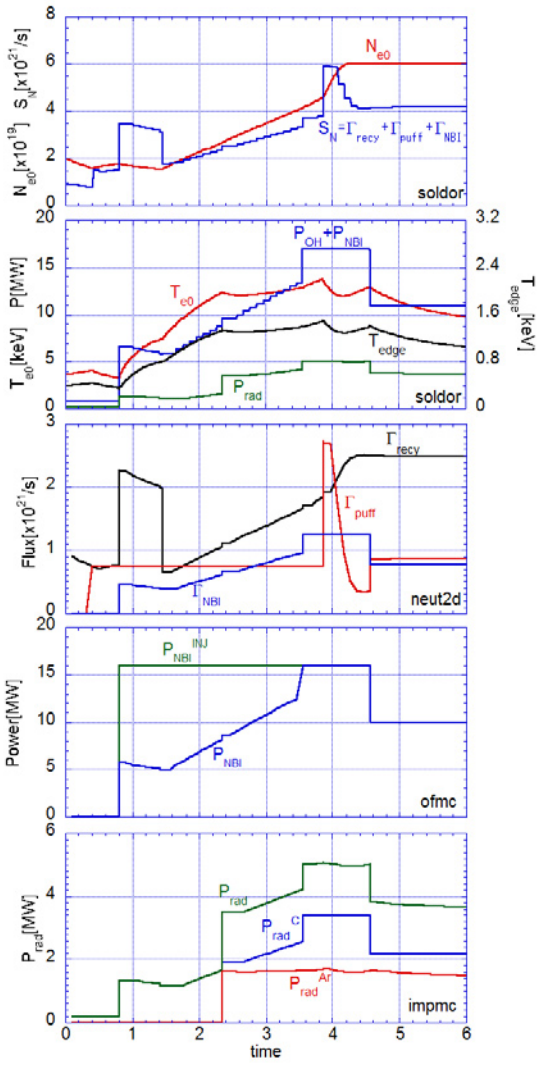

Fig. 4 Solution of example problem by use of the MPMD approach with the control of time step. The difference between this figure and figure 2 is in that the sequence control of program execution is switched off in this case.