J. Plasma Physics (2015), *vol.* 81, 435810201 © Cambridge University Press 2015
doi:10.1017/S0022377815000069

Numerical methods used in fusion science numerical modeling

M. Yagi†

Plasma Theory and Simulation Group, Rokkasho Fusion Institute, Japan Atomic Energy Agency, Obuchi, Rokkasho, Aomori, 039-3212, Japan

(Received 18 September 2014; revised 21 December 2014; accepted 5 January 2015; first published online 5 February 2015)

The dynamics of burning plasma is very complicated physics, which is dominated by multi-scale and multi-physics phenomena. To understand such phenomena, numerical simulations are indispensable. Fundamentals of numerical methods used in fusion science numerical modeling are briefly discussed in this paper. In addition, the parallelization technique such as open multi processing (OpenMP) and message passing interface (MPI) parallel programing are introduced and the loop-level parallelization is shown as an example.

1. Introduction

The dynamics of burning plasma is very complicated physics, which is dominated by multi-scale and multi-physics phenomena. To understand such phenomena, numerical simulations are indispensable. In future, the burning plasma simulator might be necessary to control the demo fusion reactor, where the installation of measurement equipments are limited. The large-scale computations are required for the burning plasma simulator. In the present stage, benchmark tests between various codes are intensively performed for verification and validation to improve the prediction capability of these codes. The wind tunnel realized by computer simulation is a well-known example which indicates numerical experiments contribute to the engineering design (Miyoshi 1994). In addition to development of simulation code followed by verification and validation, another important issue is to develop the technology for high-performance computing (HPC). Recently, distributed shared-memory machines are standard in super-computer architecture. MPI provides an efficient means of parallel communication among a distributed collection of machines (Gropp et al. 1999), on the other hand, OpenMP provides a mean of implementing shared-memory parallelism in FORTRAN and C/C++ programs (Chapman et al. 2008). OpenMP is used for intra-node communication and MPI is used for inter-node communication. The combination of these two methods gives us the most efficient means for large-scale computations using modern super-computers.

In this paper, fundamentals of numerical methods used in fusion science numerical modeling are briefly discussed as an introduction to computer simulation. Advanced readers may take a look at books on computer simulation in plasma physics (Tajima 1989; Birsall and Langdon 2005; Jardin 2010). In addition, the loop-level parallelization is explained using OpenMP and MPI parallel programing.

† Email address for correspondence: yagi.masatoshi@jeae.go.jp

https://doi.org/10.1017/S0022377815000069 Published online by Cambridge University Press

The paper is organized as follows: In Sec. 2, the boundary value problem (BVP) is discussed, where the one-dimensional Poisson equation is adopted. In Sec. 3, the initial value problem is discussed, where the advection equation is introduced. In Sec. 4, the eigenvalue problem is discussed. In Sec. 5, the loop-level parallelization technique is explained using OpenMP and MPI. In Sec. 6, Summary is given.

2. Boundary value problem

In this section, we discuss numerical scheme used to solve BVP, which frequently appears in many fields of physics including fusion science. A typical example of BVP will be considered, the Poisson equation

$$\frac{d^2}{dx^2}\theta(x) = q(x), \tag{2.1}$$

where the Dirchlet boundary condition is given by $\theta(x=0) = \theta(x=1) = 0$. Here, we introduce explicit space discretization to solve it numerically. Assuming an equidistant grid, we expand

$$\theta_{i\pm1} \equiv \theta(x \pm \Delta x) = \theta(x) \pm \Delta x \theta_x(x) + \frac{\Delta x^2}{2}\theta_{xx}(x) \pm \cdots, \tag{2.2}$$

where the subscript x denotes differentiation and the subscript i refers to the index of data points. Three types of differences are introduced

$$\begin{aligned} &\textit{Forward}: \quad \langle\theta_x\rangle_i = \frac{\theta_{i+1} - \theta_i}{\Delta x} + O(\Delta x), \\ &\textit{Backward}: \langle\theta_x\rangle_i = \frac{\theta_i - \theta_{i-1}}{\Delta x} + O(\Delta x), \\ &\textit{Central}: \quad \langle\theta_x\rangle_i = \frac{\theta_{i+1} - \theta_{i-1}}{2\Delta x} + O(\Delta x^2). \end{aligned} \tag{2.3}$$

The one-sided second-order finite difference which is useful for Neumann boundary condition is derived as follows. Firstly, we assume it is given by

$$\langle\theta_x\rangle_i = \frac{a\theta_i + b\theta_{i-1} + c\theta_{i-2}}{\Delta x} + O(\Delta x^2), \tag{2.4}$$

where a, b, c are numerical coefficients. Expanding θ_{i-2} and θ_{i-1} around i,

$$\begin{aligned} \theta_{i-2} &= \theta_i - 2\Delta x\langle\theta_x\rangle_i + 2\Delta x^2\langle\theta_{xx}\rangle_i + \cdots, \\ \theta_{i-1} &= \theta_i - \Delta x\langle\theta_x\rangle_i + \frac{\Delta x^2}{2}\langle\theta_{xx}\rangle_i + \cdots, \end{aligned} \tag{2.5}$$

and substituting (2.5) into (2.4), we have the relation

$$\langle\theta_x\rangle_x = \frac{(a+b+c)\theta_i}{\Delta x} - (2c+b)\langle\theta_x\rangle_x + \frac{\Delta x}{2}(4c+b)\langle\theta_{xx}\rangle_i + O(\Delta x^2). \tag{2.6}$$

Since $a+b+c=0$, $2c+b=-1$, $4c+b=0$ should be satisfied to be an identity, we obtain the formula

$$\langle\theta_x\rangle_i = \frac{3\theta_i - 4\theta_{i-1} + \theta_{i-2}}{2\Delta x} + O(\Delta x^2). \tag{2.7}$$

Similarly, we derive the formula

$$\langle\theta_x\rangle_i = \frac{-3\theta_i + 4\theta_{i+1} - \theta_{i+2}}{2\Delta x} + O(\Delta x^2). \tag{2.8}$$

https://doi.org/10.1017/S0022377815000069 Published online by Cambridge University Press

The second derivative in the central difference scheme is given by

$$\langle \theta_{xx} \rangle_i = \frac{\theta_{i+1} - 2\theta_i + \theta_{i-1}}{\Delta x^2} + O(\Delta x^2). \tag{2.9}$$

The finite difference equation for (2.1) gives

$$\theta_{i+1} - 2\theta_i + \theta_{i-1} = \Delta x^2 q_i, \quad i = 1, \ldots, N-1 \tag{2.10}$$

with $\theta_0 = \theta_N = 0$. In the matrix form, it is given by

$$\begin{pmatrix} -2 & 1 & & & & & \\ 1 & -2 & 1 & & & & \\ & . & . & . & & & \\ & & . & . & . & & \\ & & & . & . & . & \\ & & & & 1 & -2 & 1 \\ & & & & & 1 & -2 \end{pmatrix} \begin{pmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ . \\ \theta_{N-3} \\ \theta_{N-2} \\ \theta_{N-1} \end{pmatrix} = \begin{pmatrix} \Delta x^2 q_1 - \theta_0 \\ \Delta x^2 q_2 \\ \Delta x^2 q_3 \\ . \\ \Delta x^2 q_{N-3} \\ \Delta x^2 q_{N-2} \\ \Delta x^2 q_{N-1} - \theta_N \end{pmatrix}. \tag{2.11}$$

2.1. *Tomas algorithm*

Now, the solution of BVP reduces to solving the linear system $A\theta = q$, where the matrix A is tridiagonal if the boundary conditions are Dirichlet types. There consists three steps to solve the tridiagonal system called as 'Tomas Algorithm' (Fletcher 1991; Press et al. 1992; Karniadakis and Kirby II 2003).

- LU decomposition of matrix such as $A = LU$, where L is lower triangular matrix and U is upper triangular matrix.
- Forward substitution $Ly = q$.
- Backward substitution $U\theta = y$.

For the first step, we may express $A = LU$ as

$$\begin{pmatrix} a_1 & c_1 & & & \\ b_2 & a_2 & c_2 & & \\ & . & . & . & \\ & b_{N-1} & a_{N-1} & c_{N-1} & \\ & & b_N & a_N & \end{pmatrix} = \begin{pmatrix} 1 & & & & \\ l_2 & 1 & & & \\ & . & . & & \\ & & l_{N-1} & 1 & \\ & & & l_N & 1 \end{pmatrix} \begin{pmatrix} d_1 & u_1 & & & \\ & d_2 & u_2 & & \\ & . & . & & \\ & & & d_{N-1} & u_{N-1} \\ & & & & d_N \end{pmatrix}.$$

Comparing both sides, we obtain a series of algebraic equations:

$$\begin{aligned} d_1 &= a_1, \quad u_1 = c_1, \\ l_i &= b_i / d_{i-1}, \quad d_i = a_i - l_i u_{i-1}, \quad u_i = c_i, \\ l_N &= b_N / d_{N-1}, \quad d_N = a_N - l_N u_{N-1}, \end{aligned} \tag{2.12}$$

where $i = 2, \ldots, N-1$.

For the second step, we may express $Ly = q$ as

$$\begin{pmatrix} 1 & & & \\ l_2 & 1 & & \\ & . & . & \\ & & l_{N-1} & 1 \\ & & & l_N & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ . \\ y_{N-1} \\ y_N \end{pmatrix} = \begin{pmatrix} q_1 \\ q_2 \\ . \\ q_{N-1} \\ q_N \end{pmatrix}.$$

https://doi.org/10.1017/S0022377815000069 Published online by Cambridge University Press

Then, we obtain a series of algebraic equations:

$$y_1 = q_1, \quad y_i = q_i - l_i y_{i-1}, \tag{2.13}$$

where $i = 2, \ldots, N$.

For the third step, we may express $U\theta = y$ as

$$\begin{pmatrix} d_1 & u_1 & & & & \\ & d_2 & u_2 & & & \\ & . & . & & & \\ & & & d_{N-1} & u_{N-1} & \\ & & & & d_N \end{pmatrix} \begin{pmatrix} \theta_1 \\ \theta_2 \\ . \\ \theta_{N-1} \\ \theta_N \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ . \\ y_{N-1} \\ y_N \end{pmatrix}.$$

Similarly, we have a series of algebraic equations:

$$\theta_N = y_N/d_N, \quad \theta_i = \frac{y_i - u_i\theta_{i+1}}{d_i}, \tag{2.14}$$

where $i = N-1, \ldots, 1$. Tomas algorithm will always converge if the diagonal component is dominant compared to off-diagonal components.

3. Initial value problem

In this section, the advection equation is introduced as an example of initial value problem. The one-dimensional advection equation is given by

$$\frac{\partial \Theta}{\partial t} + U\frac{\partial \Theta}{\partial x} = 0, \tag{3.1}$$

where the periodic boundary condition $\Theta(x = 0, t) = \Theta(x = 1, t)$ is assumed and the initial condition is given by $\Theta(x, t = 0) = \sin(2\pi x)$. We will solve this equation numerically.

3.1. *Explicit methods*

As typical examples of the explicit time integration scheme, Euler-forward and central differencing (EF/CD) scheme and Euler-forward and upwind differencing (EF/UD) scheme are shown. The EF/CD scheme is given by

$$\frac{\Theta_j^{n+1} - \Theta_j^n}{\Delta t} + U\frac{\Theta_{j+1}^n - \Theta_{j-1}^n}{2\Delta x} = 0, \tag{3.2}$$

where $j = 1, \ldots, N-1$, $\Theta_0^n = \Theta_N^n$ and $\Theta_j^0 = \sin(2\pi x_j)$. The von Neumann stability analysis is performed setting

$$\Theta_j^n = \sum_{k=-\infty}^{+\infty} a_k^n e^{2\pi k x_j} \tag{3.3}$$

and substituting it into (3.2). We obtain

$$a_k^{n+1} = a_k^n(1 - iC\sin(2\pi k\Delta x)), \tag{3.4}$$

where $C = U\Delta t/\Delta x$ is Courant, Friedrichs and Lewy (CFL) number, which characterize the numerical stability of the scheme, namely

$$\frac{|a_k^{n+1}|}{|a_k^n|} = (1 + C^2\sin^2 2\pi k\Delta x)^{1/2} > 1. \tag{3.5}$$

https://doi.org/10.1017/S0022377815000069 Published online by Cambridge University Press

This inequality implies the EF/CD scheme is absolutely unstable, since the mode amplitude will grow in time. Next, we consider Euler forward and upwind differencing (EF/UD) scheme

$$\frac{\Theta_j^{n+1}-\Theta_j^n}{\Delta t}+U\frac{\Theta_j^n-\Theta_{j-1}^n}{\Delta x}=0, \tag{3.6}$$

where $j=1,\ldots,N-1$. It can be rewritten as

$$\Theta_j^{n+1}=\Theta_j^n-\frac{C}{2}\left(\Theta_{j+1}^n-\Theta_{j-1}^n\right)+\frac{C}{2}\left(\Theta_{j+1}^n-2\Theta_j^n+\Theta_{j-1}^n\right), \tag{3.7}$$

where the last term in right-hand side indicates the numerical diffusion. The von Neumann stability analysis gives us

$$a_k^{n+1}=a_k^n(1-C(1-e^{-2\pi ik\Delta x})), \tag{3.8}$$

and the amplitude equation is

$$\frac{|a_k^{n+1}|}{|a_k^n|}=(1+2(C^2-C)(1-\cos 2\pi k\Delta x))^{1/2}. \tag{3.9}$$

The necessary condition for the numerical stability is given by

$$C\leqslant 1. \tag{3.10}$$

Assuming $a_n^k\propto e^{\lambda t_k}$, the eigenvalue λ is given by

$$\lambda\Delta t=-C(1-\cos 2\pi k\Delta x)-iC\sin 2\pi k\Delta x, \tag{3.11}$$

where we use the relation

$$\frac{a_k^{n+1}-a_k^n}{\Delta t}=\lambda a_k^n. \tag{3.12}$$

Using (3.11), we can examine the stability region of the EF/UD scheme in the complex $\lambda\Delta t$ region.

Heun's method is the improved Euler's method, which is a predictor-corrector method with forward Euler's method as predictor and trapezoidal method as corrector (Süli and Mayers 2003). The accuracy of the Euler method improves only linearly with the step size is decreased, whereas the Heun method improves accuracy quadratically. It is given by

$$\begin{gathered}\frac{\Theta_j^*-\Theta_j^n}{\Delta t}+U\frac{\Theta_j^n-\Theta_{j-1}^n}{\Delta x}=0\\ \frac{\Theta_j^{n+1}-\Theta_j^n}{\Delta t}+\frac{U}{2}\left(\frac{\Theta_j^n-\Theta_{j-1}^n}{\Delta x}+\frac{\Theta_j^*-\Theta_{j-1}^*}{\Delta x}\right)=0.\end{gathered} \tag{3.13}$$

3.2. *Implicit methods*

Next, we show Crank–Nicolson and center differencing (CN/CD) scheme as an example for the implicit time integration scheme.

$$\frac{\Theta_j^{n+1}-\Theta_j^n}{\Delta t}+\frac{U}{2}\left(\frac{\Theta_{j+1}^{n+1}-\Theta_{j-1}^{n+1}}{2\Delta x}+\frac{\Theta_{j+1}^n-\Theta_{j-1}^n}{2\Delta x}\right)=0, \tag{3.14}$$

where $j=1,\ldots,N-1$. The von Neumann stability analysis gives us

$$a_k^{n+1}=\frac{1-i(C/2)\sin 2\pi k\Delta x}{1+i(C/2)\sin 2\pi k\Delta x}a_k^n \tag{3.15}$$

https://doi.org/10.1017/S0022377815000069 Published online by Cambridge University Press

and

$$\frac{|a_k^{n+1}|}{|a_k^n|} = 1. \tag{3.16}$$

It indicates that the CN/CD scheme is neutrally stable. Similarly, the eigenvalue is calculated as

$$\lambda\Delta t = \frac{(C^2/4)\sin^2 2\pi k\Delta x - i(C/2)\sin 2\pi k\Delta x}{1-(C^2/4)\sin^2 2\pi k\Delta x}, \tag{3.17}$$

which are on the left half-plane for any positive value of C. Equation (3.14) is expressed in the matrix form as

$$\begin{pmatrix} 1 & C/4 & & & & & \\ -C/4 & 1 & C/4 & & & & \\ & \cdot & \cdot & \cdot & & & \\ & & -C/4 & 1 & C/4 & & \\ & & & \cdot & \cdot & & \\ & & & & -C/4 & 1 & \end{pmatrix} \begin{pmatrix} \Theta_1^{n+1} \\ \Theta_2^{n+2} \\ \Theta_3^{n+1} \\ \cdot \\ \Theta_{N-3}^{n+1} \\ \Theta_{N-2}^{n+1} \\ \Theta_{N-1}^{n+1} \end{pmatrix} = \begin{pmatrix} C\Theta_0^n/4 + \Theta_1^n - C\Theta_2^n/4 + C\Theta_0^{n+1}/4 \\ C\Theta_1^n/4 + \Theta_2^n - C\Theta_3^n/4 \\ C\Theta_2^n/4 + \Theta_3^n - C\Theta_4^n/4 \\ \cdot \\ C\Theta_{N-4}^n/4 + \Theta_{N-3}^n - C\Theta_{N-2}^n/4 \\ C\Theta_{N-3}^n/4 + \Theta_{N-2}^n - C\Theta_{N-1}^n/4 \\ C\Theta_{N-2}^n/4 + \Theta_{N-1}^n - C\Theta_N^n/4 + C\Theta_N^{n+1}/4 \end{pmatrix}.$$

Tomas Algorithm is available to solve CN/CD scheme. This method is not restricted by CFL condition like EF/UD method, however, it needs the matrix inversion.

Now, we will consider the effect of boundary condition (Karniadakis and Kirby II 2003). Consider the advection equation with the initial condition $\Theta(x, t=0) = \sin 2\pi x$ for $0 < x < 1$ and the one side boundary condition $\Theta(x=0, t) = -\sin 2\pi t$. Using CN/CD scheme, $j = N$ point is given by

$$\frac{\Theta_N^{n+1} - \Theta_N^n}{\Delta t} + \frac{U}{2}\left(\frac{\Theta_{N+1}^{n+1} - \Theta_{N-1}^{n+1}}{2\Delta} + \frac{\Theta_{N+1}^n - \Theta_{N-1}^n}{2\Delta}\right) = 0. \tag{3.18}$$

To evaluate this equation, we need the information of the point $j = N+1$. It could be evaluated by linear extrapolation.

$$\Theta_{N+1} = \Theta_N + \Delta x\frac{\Theta_N - \Theta_{N-1}}{\Delta x} = 2\Theta_N - \Theta_{N-1} \tag{3.19}$$

$$\frac{\Theta_N^{n+1} - \Theta_N^n}{\Delta t} + \frac{U}{2}\left(\frac{\Theta_N^{n+1} - \Theta_{N-1}^{n+1}}{\Delta x} + \frac{\Theta_N^n - \Theta_{N-1}^n}{\Delta x}\right) = 0. \tag{3.20}$$

It is found that the addition of upwind derivative does not influence the stability of CN scheme. Namely, the associated error will be advected out of the domain.

The backward differential formula (BDF) is a family of implicit methods for the numerical integration of ordinary differential equations (Gear 1971). These methods are especially used for the solution of stiff differential equations. Consider an initial value problem

$$\frac{dy}{dt} = f(t, y),\ y(t_0) = y_0. \tag{3.21}$$

The BDF is given by

$$\sum_{j=1}^{k}\frac{1}{j}\nabla^j y_{n+1} = \Delta t f_{n+1}, \tag{3.22}$$

https://doi.org/10.1017/S0022377815000069 Published online by Cambridge University Press

where ∇ is the backward difference operator $\nabla y_{n+1} = y_{n+1} - y_n$, $\nabla^2 y_{n+1} = \nabla y_{n+1} - \nabla y_n = y_{n+1} - 2y_n + y_{n-1}$ and $y_{n+1} = y(t_{n+1})$, $f_{n+1} = f(t_{n+1}, y_{n+1})$. These work for $k \leqslant 6$.

4. Eigenvalue problem

In this section, we consider the eigenvalue problem

$$Ax = \lambda x, \tag{4.1}$$

where A is $n \times n$ matrix and x is the eigenvector. Firstly, we introduce the power method, which gives the maximum eigenvalue.

$$x^{k+1} = cAx^k, \tag{4.2}$$

where c is a normalization constant. Here, we assume

$$|\lambda_1| > |\lambda_2| \geqslant |\lambda_3| \cdots \geqslant |\lambda_n|. \tag{4.3}$$

We assume the initial guess x^0 is written by

$$x^0 = c_1 v_1 + c_2 v_2 + \cdots + c_n v_n. \tag{4.4}$$

Then, we obtain

$$x^k = Ax^{k-1} = \cdots = A^k x^0 = c_1 \lambda_1^k v_1 + \cdots + c_n \lambda_n^k v_n. \tag{4.5}$$

Now we see that

$$\frac{x^k}{c_1 \lambda_1^k} = v_1 + \frac{c_2}{c_1}\left(\frac{\lambda_2}{\lambda_1}\right)^k v_2 + \cdots + \frac{c_n}{c_1}\left(\frac{\lambda_n}{\lambda_1}\right)^k v_n \tag{4.6}$$

which converges to v_1. The convergence rate is given by $|\lambda_2|/|\lambda_1|$.

$$\begin{aligned} &Give : x^0 \\ &do\ k = 1, 2, \ldots \\ &\hat{x}^k = Ax^{k-1} \\ &x^k = \frac{\hat{x}^k}{max(\hat{x}^k)} \\ &end\ do \end{aligned}$$

where $max(y)$ of a vector y returns the entry of y with the maximum modulus. For $k \to \infty$, $max(\hat{x}^k) \to \lambda_1$ and $x^k \to v_1$. In this calculation, the function $max()$ plays a role of the norm, but the Rayleigh quotient can be used to calculate the eigenvalue as an alternative way:

$$R(A, x^k) = \frac{(x^k)^T A x^k}{||x^k||^2}. \tag{4.7}$$

Next, we discuss the inverse shifted power method. To compute the minimum eigenvalue, we can apply inverse power method. In this case, the iteration procedure is given by

$$Ax^{k+1} = cx^k, \tag{4.8}$$

where c is a normalization constant. This method is improved by a proper shift

$$(A - \sigma I)x^{k+1} = cx^k. \tag{4.9}$$

https://doi.org/10.1017/S0022377815000069 Published online by Cambridge University Press

The solution will converge to the smallest shifted eigenvalue $|\lambda_n - \sigma|$ and the corresponding eigenvector. The rate of convergence is given by $|\lambda_n - \sigma|/|\lambda_{n-1} - \sigma|$. The algorithm is written by

$$
\begin{aligned}
&Give : x^0 \\
&\sigma_0 = \frac{(x^0)^T A x^0}{(x^0)^T x^0} \\
&do\ k = 0, 1, \ldots \\
&L^k U^k = A - \sigma_k I \\
&\hat{x}^{k+1} = (U^k)^{-1}(L^k)^{-1} x^k \\
&x^{k+1} = \frac{\hat{x}^{k+1}}{max(\hat{x}^{k+1})} \\
&\sigma_{k+1} = \frac{(x^{k+1})^T A x^{k+1}}{(x^{k+1})^T x^{k+1}} \\
&if\ |\sigma_{k+1} - \sigma_k| < \epsilon,\ return \\
&end\ do
\end{aligned}
$$

5. Loop-level parallelization

The HPC cluster consists of a number of computation nodes, where the interconnect plays a vital role for data communication. On the other hand, each node contains multi-cores which are connected by non-uniform memory access architecture. In HPC, parallelism may be built in at two levels at a low level, with threads, or at a higher level, with MPI. In this section, the loop-level parallelization is discussed.

```
[do k = 1, kmax
 [do j = 1, jmax
  [do i = 1, imax
   [floating point operations]
  end do ]Vectorization
 end do ]OMP
end do ]MPI
```

The inner loop is usually vectorized, manually or through compiler settings. The middle loop is parallelized using OpenMP. The outer loop is parallelized using MPI. The inner loop and middle loop will be performed in each node, the outer loop will be allocated at different nodes. The vectorization is supported by Intel Advanced Vector Extension (AVX) which makes it possible to realize 256 bit single instruction multiple data. It is done by setting compiler option such as

```
$ cc/ifort -xAVX
```

The math kernel library (MKL) is provided by Intel, which is already optimized for AVX. More detail information will be available on Intel web site (https://software.intel.com/en-us/articles/introduction-to-intel-advanced-vector-extensions). Also, GPU may offer alternative option (Ruetsch and Fatica 2013).

https://doi.org/10.1017/S0022377815000069 Published online by Cambridge University Press

5.1. *Using OpenMP*

The following fortran program shows an example of OpenMP parallelization for two-dimensional Laplace equation by Jacobi method. The discrete Laplace equation is given by

$$\Theta_{i,j}^{n+1} = \frac{1}{4}\left(\Theta_{i+1,j}^{n} + \Theta_{i,j+1}^{n} + \Theta_{i-1,j}^{n} + \Theta_{i,j-1}^{n}\right). \tag{5.1}$$

For details on the algorithm, see the reference (Karniadakis and Kirby II 2003).

```
!$omp  parallel
do istep = 1,nstep
!$omp do
   do j=1,n
      do i=1,n
        g(i,j)=0.25d0*(f(i-1,j)+f(i+1,j)  &
                +f(i,j-1)+f(i,j+1)
      end do
   end do
!$omp end do nowait
!$omp single
   er=0.0d0
!$omp end single
!$omp do reduction(+: er)
   do j=1,n
      do i=1,n
      er=er+(g(i-1,j)-4.0d0*g(i,j)+g(i+1,j)  &
          +g(i,j-1)+g(i,j+1))**2
       end do
   end do
if(er < threshold) call terminate
!$omp do
   do  j=1,n
      do  i=1,n
       f(i,j)=g(i,j)
      end do
   end do
end do
!$omp end parallel
```

The compile option openmp is necessary for Intel fortran compiler to distinguish OpenMP directive and the environment variable OMP_NUM_THREADS should be set when it is executed.

$ ifort -openmp ···
$ export OMP_NUM_THREADS=4
$ ./a.out

The parallel construct plays a crucial role in OpenMP. A program without a parallel construct will be executed sequentially. The syntax in C/C++ is different from that in Fortran, *#pragma omp parallel* ··· instead of !$*omp parallel* ···!$*omp end parallel*. The work-sharing constructs are the next most important feature of OpenMP because they are used to distribute computation among the threads in a team. A work-sharing

https://doi.org/10.1017/S0022377815000069 Published online by Cambridge University Press

region must bind to an active parallel region in order to have an effect. Fortran has four work-sharing constructs, namely, !$*omp do*, !$*omp sections*, !$*omp single* and !$*omp workshare*. See the reference for the details (Chapman et al. 2008).

5.2. *Using MPI*

In this subsection, the same fortran program in the previous subsection is parallelized using MPI. The essential functions in MPI are six, namely, MPI_INIT (Initialze MPI), MPI_COMM_SIZE (Find out how many processes there are), MPI_COMM_RANK (Find out which process I am), MPI_SEND (Send a message), MPI_RECV (Receive a message) and MPI_FINALIZE (Terminate MPI). In this program, three additional routines appear. The routine MPI_CART_CREATE creates a Cartesian decomposition of the processes with the number of dimensions. The routine MPI_CART_SHIFT can be used to find the destination and source of a shift for each process. The routine MPI_ALLREDUCE provides the result of the reduction operation in all processes, which is equivalent MPI_REDUCE followed by an MPI_BCAST. See the reference for the details (Gropp et al. 1999).

```
call MPI_INIT(ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD,myid,ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD,nprocs,ierr)
call MPI_CART_CREATE(MPI_COMM_WORLD,1,nprocs,.false., &
  .true.,comm1d,ierr)
call MPI_COMM_RANK(comm1d,myid,ierr)
call MPI_CART_SHIFT(comm1d,0,1,nbrbottom,nbrtop,ierr)
   s=1+myid*floor(n/nprocs)
   if(myid == nprocs-1) then
      e=n
   else
      e=s+floor(n/nprocs)-1
   end if
do j=s,e
   do  i=1,n
   gloc(i,j)=0.25d0*(floc(i-1,j)+floc(i+1,j) &
               +floc(i,j-1)+floc(i,j+1)
   end do
end do
call MPI_SENDRECV( &
gloc(1,e),n,MPI_DOUBLE_PRECISION,nbrtop,0, &
gloc(1,s-1),n,MPI_DOUBLE_PRECISION,nbrbottom,0, &
comm1d,status,ierr)
call MPI_SENDRECV( &
gloc(1,s),n,MPI_DOUBLE_PRECISION,nbrottom, 1, &
gloc(1,e+1),n,MPI_DOUBLE_PRECISION,nbrtop,1, &
comm1d,status,ierr)
erloc=0.0d0
do j=s,e
   do i=1,n
   erloc=erloc+(gloc(i-1,j)-4.0d0*gloc(i,j)+gloc(i+1,j) &
            +gloc(i,j-1)+gloc(i,j+1))**2
   end do
```

https://doi.org/10.1017/S0022377815000069 Published online by Cambridge University Press

```
end do
call MPI_ALLREDUCE( erloc, er, 1, MPI_DOUBLE_PRECISION,  &
                        MPI_SUM, comm1d, ierr )
if(er < threshold) then
call MPI_FINALYZE(ierr)
stop
end if
do  j=s,e
   do  i=1,n
   floc(i,j)=gloc(i,j)
   end do
end do
```

The compilation of the source code and the execution are shown as follows:

```
$ mpiifort ···
$ mpirun -np 4 ./a.out
```

It is seen that MPI program is a little bit complicated compared to OpenMP program. However, the structure of MPI makes it straightforward to port existing codes and to write new ones, once you understand the fundamentals of MPI functions (six elementary functions plus collective communication functions such as MPI_BCAST, MPI_REDUCE, MPI_SCATTER, MPI_GATHER, MPI_ALLTOALL and so on).

5.3. *Using MPI and OpenMP*

The following example shows the hybrid MPI and OpenMP parallel programing.

```
call MPI_INIT(ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD,myid,ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD,nprocs,ierr)
   s=1+myid*floor(n/nprocs)
   if(myid == nprocs-1) then
      e=n
   else
      e=s+floor(n/nprocs)-1
   end if
!$ OMP PARALLEL DO PRIVATE(i)
do j=s,e
   y(j)=0.0
   do i=1,n
   y(j)=y(j)+A(j,i)*x(i)
   end do
end do
!$OMP END PARALLEL  DO
call MPI_GATHER(y,e-s+1,MPI_REAL, &
ygl, e-s+1,MPI_REAL, MPI_COMM_WORLD, ierr)
```

```
$ mpiifort -openmp ···
$ export OMP_NUM_THREADS=4
$ mpiexec -npernode  2 ./a.out
```

In this example, the outer loop j will be parallelized via MPI and OpenMP. The mpiexec command may depend on your environment of the cluster. If you

https://doi.org/10.1017/S0022377815000069 Published online by Cambridge University Press

are interested in the hybrid programing, you may visit some web sites, where you can find tutorial documents (http://www.training.prace-ri.eu/uploads/tx_pracetmo/hybridProgramming.pdf, http://mc.stanford.edu/cgi-bin/images/7/78/Hybrid_MPI_openMP.pdf).

6. Summary

In this paper, fundamentals of numerical methods used in fusion science numerical modeling are briefly discussed. The BVP, the initial value problem and the eigenvalue problem are introduced, which are essential in general physics as well as plasma physics. The basic idea to solve these problems is shown. For realistic applications, commercial library such as Intel MKL library and open libraries such as LAPACK(www.netlib.org/lapack) and PETSc(www.mcs.anl.gov/petsc), etc. are useful.

In addition, the parallelization technique such as OpenMP and MPI parallel programing are introduced and the loop-level parallelization is discussed. The hybrid programing will play a crucial role for high performance exascale computing (http://ascr-discovery.science.doe.gov/exascale/exa_ov1.shtml, http://www.exascale-computing.eu/, http://www.aics.riken.jp/en/exe_project/Exascale-Supercomputer-Project-launched.html).

Acknowledgement

The author (M. Y.) acknowledges travel funding supported by 7th ITER International School.

REFERENCES

Birsall, C. K. and Langdon, A. B. 2005 *Plasma Physics via Computer Simulation*. New York: Taylor & Francis.

Chapman, B., Jost, G. and van der Pas, R. 2008 *Using OpenMP*. Cambridge, Massachusetts: The MIT Press.

Fletcher, C. A. J. 1991 *Computational Techniques for Fluid Dynamics*, Vol. 1, 2nd edn. New York: Springer.

Gear, C. W. 1971 *Numerical Initial Value Problems in Ordinary Differential Equations*. London: Pentice-Hall, Inc.

Gropp, W., Lusk, E. and Skjellum, A. 1999 *Using MPI*, 2nd edn. Cambridge, Massachusetts: The MIT Press.

Jardin, S. 2010 *Computational Methods in Plasma Physics*. New York: CRC Press.

Karniadakis, G. E. and Kirby II, R. M. 2003 *Parallel Scientific Computing in C++ and MPI*. Cambridge, United Kingdom: Cambridge University Press.

Miyoshi, H. et al. 1994 Development and achievement of NAL numerical wind tunnel (NWT) for CFD computations. In: *Supercomputing '94 Proc. 1994 ACM/IEEE Conf. on Supercomputing*, Los Alamitos, CA, USA: IEEE Computer Society Press, pp. 685–692.

Press, W. H., Teukolsky, S. A., Vetterling, W. T. and Flannery, B. P. 1992 *Numerical Recipes in Fortran*, 2nd edn. Cambridge, United Kingdom: Cambridge University Press.

Ruetsch, G. and Fatica, M. 2013 *CUDA Fortran for Scientists and Engineers: Best Practices for Efficient CUDA Fortran Programming*. Amsterdam: ELSEVIER.

Süli, E. and Mayers, D. F. 2003 *An Introduction to Numerical Analysis*. Cambridge, United Kingdom: Cambridge University Press.

Tajima, T. 1989 *Computational Plasma Physics: With Applications to Fusion and Astrophysics*. Redwood City: Addison-Wesley Publishing Company, Inc.

https://doi.org/10.1017/S0022377815000069 Published online by Cambridge University Press