課題S-2解説

2006年6月6日

中島研吾

並列計算プログラミング（616-2057）・先端計算機演習I（616-4009）

課題S2（1/2）

- 「<$S2>/heat_jacobi.f」,「<$S2>/heat_jacobi.c」を参考にして，一次元熱伝導方程式をJacobi法によって解くプログラムを並列化せよ（**S2-1**）。
- 「<$S2>/heat_gs.f」,「<$S2>/heat_gs.c」を参考にして，一次元熱伝導方程式をGauss-Seidel法によって解くプログラムを並列化せよ（**S2-2**）。

- 「<$S2>/1d-srb1.f, 1d-srb2.f」,「<$S2>/1d-srb1.c, 1d-srb2.c」を参考にして「一般化された通信テーブル」を使用せよ。
  - BUFのかわりに温度をやりとりする

課題S2（2/2）

- NG=100およびNG=103の場合について，1，2，4，8CPUを使用して計算してみよ。
  - 実は，N=100程度では並列化の効果は余り･･･全く得られない
- Gauss-Seidel法の場合，単純に並列化すると，領域分割数を増加させた場合，反復回数が増加する。その理由について考えてみよ。
  - Jacobi法の場合はこのような現象が生じない。その理由についても併せて検討すること。

概　要

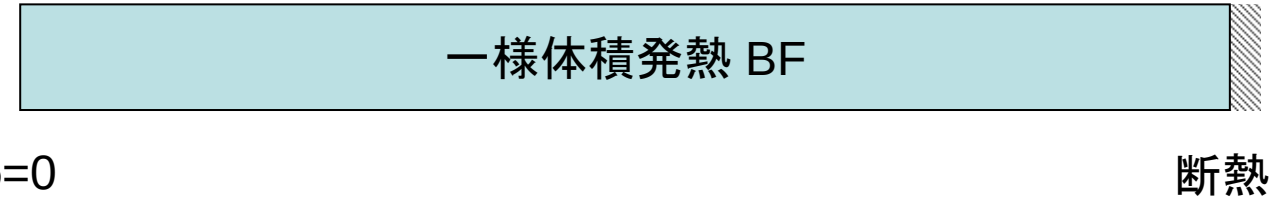
- 復習
  - 一次元熱伝導プログラム
  - 一次元差分法の並列化に必要な局所データ構造
- 通信テーブル（一般化）
- 課題S2解説

一次元熱伝導方程式(1/7)

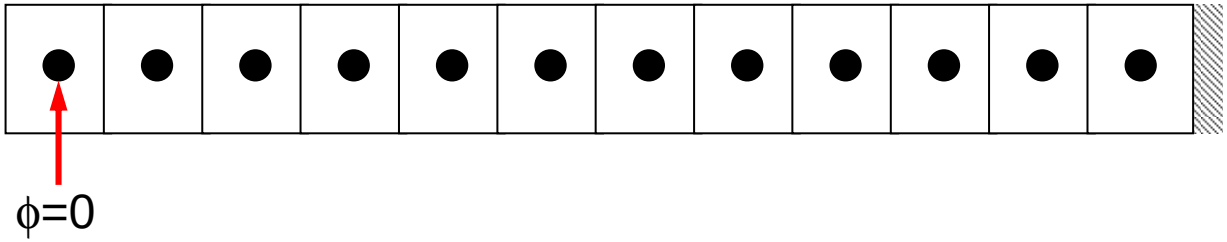
支配方程式:熱伝導率＝1(一様)

$$\frac{d^2\phi}{dx^2} + BF = 0, \quad \phi = 0 @ x = 0, \frac{d\phi}{dx} = 0 @ x = x_{\max}$$

$$\phi = -\frac{1}{2} BF\, x^2 + BF\, x_{\max}\, x$$

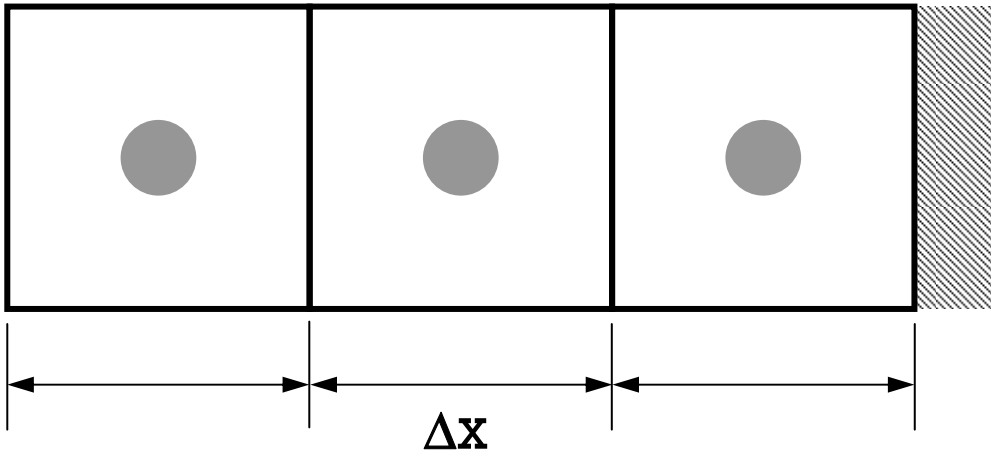


以下のような離散化(要素中心で従属変数を定義)をしているので注意が必要



断熱となっているのはこの面,しかし温度は計算されない

差分格子:要素中心で従属変数を定義

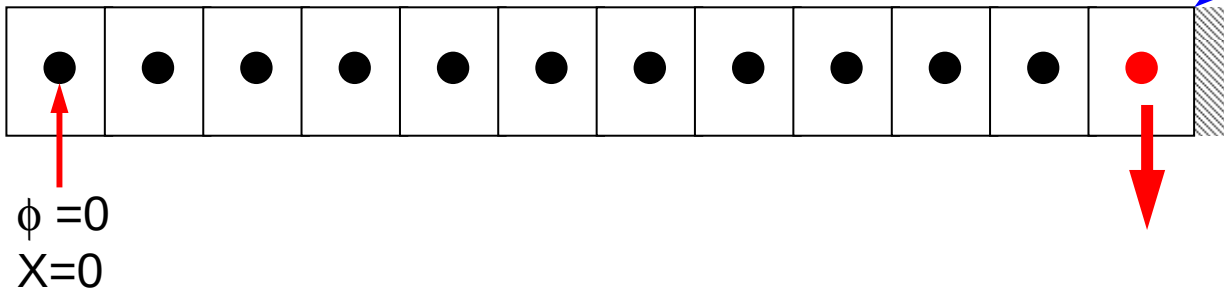


一次元熱伝導方程式(2/7)

解析解

$$\phi = -\frac{1}{2} BF\, x^2 + BF\, x_{\max}\, x$$

断熱となって
いるのはこの面,
しかし温度は計算
されない(X=X_{max})。



Δx=1.d0, メッシュ数=50, とすると, X_{max}=49.5,
●の点のX座標は49.0となる。BF=1.0d0とすると●での温度は:

$$\phi = -\frac{1}{2} 49^2 + 49.5 \times 49 = -1200.5 + 9850.5 = 1225$$

一次元熱伝導方程式(3/7)

要素単位の線形方程式

- 差分法による離散化

$$\left(\frac{d^2\phi}{dx^2}\right)_i \approx \frac{\left(\frac{d\phi}{dx}\right)_{i+1/2} - \left(\frac{d\phi}{dx}\right)_{i-1/2}}{\Delta x} = \frac{\frac{\phi_{i+1}-\phi_i}{\Delta x} - \frac{\phi_i-\phi_{i-1}}{\Delta x}}{\Delta x} = \frac{\phi_{i+1}-2\phi_i+\phi_{i-1}}{\Delta x^2}$$

- 各要素における線形方程式は以下のような形になる

$$\frac{d^2\phi}{dx^2} + BF = 0 \quad \rightarrow$$

$$\frac{\phi_{i+1}-2\phi_i+\phi_{i-1}}{\Delta x^2} + BF(i) = 0 \quad (1 \le i \le N)$$

$$\frac{1}{\Delta x^2}\phi_{i+1} - \frac{2}{\Delta x^2}\phi_i + \frac{1}{\Delta x^2}\phi_{i-1} + BF(i) = 0 \quad (1 \le i \le N)$$

$$A_L(i)\times\phi_{i-1} + A_D(i)\times\phi_i + A_R(i)\times\phi_{i+1} = BF(i) \quad (1 \le i \le N)$$

$$A_L(i) = \frac{1}{\Delta x^2},\ A_D(i) = -\frac{2}{\Delta x^2},\ A_R(i) = \frac{1}{\Delta x^2}$$

N=8の場合：連立一次方程式

自分とその周囲のみに非ゼロ成分：疎行列

	1	2	3	4	5	6	7	8			
1	$A_D(1)$	$A_R(1)$							×	$\phi(1)$	= $BF(1)$
2	$A_L(2)$	$A_D(2)$	$A_R(2)$							$\phi(2)$	$BF(2)$
3		$A_L(3)$	$A_D(3)$	$A_R(3)$						$\phi(3)$	$BF(3)$
4			$A_L(4)$	$A_D(4)$	$A_R(4)$					$\phi(4)$	$BF(4)$
5				$A_L(5)$	$A_D(5)$	$A_R(5)$				$\phi(5)$	$BF(5)$
6					$A_L(6)$	$A_D(6)$	$A_R(6)$			$\phi(6)$	$BF(6)$
7						$A_L(7)$	$A_D(7)$	$A_R(7)$		$\phi(7)$	$BF(7)$
8							$A_L(8)$	$A_D(8)$		$\phi(8)$	$BF(8)$

$$A_D(1)\times\phi(1)+A_R(1)\times\phi(2)=BF(1)$$

$$A_L(2)\times\phi(1)+A_D(2)\times\phi(2)+A_R(2)\times\phi(3)=BF(2)$$

$$A_L(3)\times\phi(2)+A_D(3)\times\phi(3)+A_R(3)\times\phi(4)=BF(3)$$

$$A_L(4)\times\phi(3)+A_D(4)\times\phi(4)+A_R(4)\times\phi(5)=BF(4)$$

$$A_L(5)\times\phi(4)+A_D(5)\times\phi(5)+A_R(5)\times\phi(6)=BF(5)$$

$$A_L(6)\times\phi(5)+A_D(6)\times\phi(6)+A_R(6)\times\phi(7)=BF(6)$$

$$A_L(7)\times\phi(6)+A_D(7)\times\phi(7)+A_R(7)\times\phi(8)=BF(7)$$

$$A_L(8)\times\phi(7)+A_D(8)\times\phi(8)=BF(8)$$

$$A_L(i)\times\phi(i-1)+A_D(i)\times\phi(i)+A_R(i)\times\phi(i+1)=BF(i)$$

$$A_L(i)=\frac{1}{\Delta x^2},\ A_D(i)=-\frac{2}{\Delta x^2},\ A_R(i)=\frac{1}{\Delta x^2}$$

一次元熱伝導方程式(4/7)

連立一次方程式の解法:古典的反復法(1)

各要素における線形方程式

$$A_L(i)\times\phi_{i-1}+A_D(i)\times\phi_i+A_R(i)\times\phi_{i+1}=BF(i)\quad(1\le i\le N)$$

$$A_L(i)=\frac{1}{\Delta x^2},\ A_D(i)=-\frac{2}{\Delta x^2},\ A_R(i)=\frac{1}{\Delta x^2}$$

解

$$A_D(i)\times\phi(i)=BF(i)-A_L(i)\times\phi_{i-1}-A_R(i)\times\phi_{i+1}$$

$$\phi(i)=\frac{BF(i)-A_L(i)\times\phi_{i-1}-A_R(i)\times\phi_{i+1}}{A_D(i)}\quad(1\le i\le N)$$

- 各点においてφ(i)の値の変化が無くなるまで反復を繰り返す。
- 前の反復における解を φ⁰(i)とする
- 逆行列を使って解く方法(ガウスの消去法等)もある

一次元熱伝導方程式(5/7)
連立一次方程式の解法:古典的反復法(2)

- ヤコビ法(Jacobi)
  - ϕ_{i-1}およびϕ_{i+1}の値として前の反復における $\phi^0(i-1), \phi^0(i+1)$ を使用する。
  - 収束は遅い。

$$\phi(i) = \frac{BF(i) - A_L(i) \times \phi^0(i-1) - A_R(i) \times \phi^0(i+1)}{A_D(i)}$$

- ガウス＝ザイデル(Gauss-Seidel)法
  - 計算の終了した値 ϕ_{i-1}については最新の値 $\phi(i-1)$, ϕ_{i+1}の値としては前の反復における値 $\phi^0(i+1)$を使用する。
  - Jacobi法より速い。

$$\phi(i) = \frac{BF(i) - A_L(i) \times \phi(i-1) - A_R(i) \times \phi^0(i+1)}{A_D(i)}$$

一次元熱伝導方程式(6/7)

連立一次方程式の解法:古典的反復法(3)

- SOR(Successive-Over Relaxation)法
  - Gauss-Seidel法によって求められた解を ϕ_{GS} とすると, $\phi(i) = \phi^0(i) + \omega\ (\phi_{GS} - \phi^0(i))$

$$\phi_{GS}(i) = \frac{BF(i) - A_L(i) \times \phi(i-1) - A_R(i) \times \phi^0(i+1)}{A_D(i)}$$

$$\phi_{SOR}(i) = \phi^0(i) + \omega \times \left[\phi_{GS}(i) - \phi^0(i)\right]$$

一次元熱伝導方程式(7/7)
連立一次方程式の解法:古典的反復法(4)

- SOR(Successive-Over Relaxation)法(続き)
  - ω=1の場合はGauss-Seidelと同じ，ω>1の場合を過緩和(over relaxation)，ω<1の場合を不足緩和(under relaxation)と呼ぶ。通常 1<ω<2 の値を使用する。
  - ω>1とすることによって収束が加速される場合がある。
    - 値が大きすぎると発散する場合がある。
  - 一次元線形問題における最適値(メッシュ数=N)，Nが大きくなると2に近づく:境界条件等によって変わる

$$\varpi_{opt} \approx \frac{2}{1+\sin\left(\pi / N\right)}$$

一次元熱伝導方程式：Jacobi法 heat_jacobi.f（1/3）

```
!C
!C   1D Poisson Equation Solver by
!C   Jacobi Method
!C
!C   d/dx(dPHI/dx) + BF = 0
!C   PHI=0@x=0
!C
      program JACOBI_poi
      implicit REAL*8 (A-H,O-Z)

      integer :: N, ITERmax
      real(kind=8) :: dx, RESID, dPHI, dPHImax, BF, EPS
      real(kind=8), dimension(:), allocatable :: PHI, RHS, PHI0
      real(kind=8), dimension(:), allocatable :: rAD, AR, AL

!C
!C-- INIT.
      open  (11, file='input.dat', status='unknown')
       read (11,*) N
       read (11,*) dX, BF
       read (11,*) ITERmax
       read (11,*) EPS
      close (11)
```

一次元熱伝導方程式:Jacobi法 heat_jacobi.f（2/3）

```
      allocate (PHI(N+1), rAD(N), AR(N), AL(N), RHS(N), PHI0(N))

       PHI = 0.d0
       PHI0= 0.d0

       AR = 1.d0/dX
       AL = 1.d0/dX
      rAD = 1.d0/(-2.d0/dX)
      RHS=  -BF * dX

       AL (1)= 0.d0
       AR (1)= 0.d0
      rAD (1)= 1.d0
       RHS(1)= 0.d0

       AR (N)= 0.d0
      rAD (N)= 1.d0/(-1.d0/dX)
```

$$\frac{d^2\phi}{dx^2} + BF = 0, \quad \phi = 0 @ x = 0, \frac{d\phi}{dx} = 0 @ x = x_{\max}$$

一次元熱伝導方程式：Jacobi法 heat_jacobi.f（2/3）

```
      allocate (PHI(N+1), rAD(N), AR(N), AL(N), RHS(N), PHI0(N))

       PHI = 0.d0
       PHI0= 0.d0

       AR = 1.d0/dX
       AL = 1.d0/dX
      rAD = 1.d0/(-2.d0/dX)
      RHS=  -BF * dX

       AL (1)= 0.d0
       AR (1)= 0.d0
      rAD (1)= 1.d0
       RHS(1)= 0.d0

       AR (N)= 0.d0
      rAD (N)= 1.d0/(-1.d0/dX)
```

$$\left(\frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{\Delta x^2} \right) \times V + BF \times V = 0$$

$$\left(\frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{\Delta x^2} \right) \times \Delta x + BF \times \Delta x = 0$$

一次元熱伝導方程式：Jacobi法 heat_jacobi.f（2/3）

```
      allocate (PHI(N+1), rAD(N), AR(N), AL(N), RHS(N), PHI0(N))

       PHI = 0.d0
       PHI0= 0.d0

       AR = 1.d0/dX
       AL = 1.d0/dX
      rAD = 1.d0/(-2.d0/dX)
      RHS=  -BF * dX

       AL (1)= 0.d0
       AR (1)= 0.d0
      rAD (1)= 1.d0
       RHS(1)= 0.d0

       AR (N)= 0.d0
      rAD (N)= 1.d0/(-1.d0/dX)
```

$$rA_D(i) = \frac{1}{A_D(i)}$$

割り算は計算時間がかかるため

$$\left(\frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{\Delta x^2} \right) \times \Delta x + BF \times \Delta x = 0$$

$$\frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{\Delta x} = \left(\frac{1}{\Delta x} \right) \phi_{i-1} - \left(\frac{2}{\Delta x} \right) \phi_i + \left(\frac{1}{\Delta x} \right) \phi_{i+1} = -BF \times \Delta x$$

AL　AD　AR　RHS

境界条件の処理：i=1

```
AL (1)= 0.d0
AR (1)= 0.d0
rAD (1)= 1.d
RHS (1)= 0.d0
```

$$\phi = 0 @ x = 0 \Rightarrow \phi_1 = 0$$

$$\Rightarrow (0)\phi_2 + (1)\phi_1 + (0)\phi_0 = 0$$

AR AD AL RHS

境界条件の処理：i=N

```
AL (N)= 1.d0/dX
AR (N)= 0.d0
rAD(N)= 1.d0/(-1.d0/dX)
```

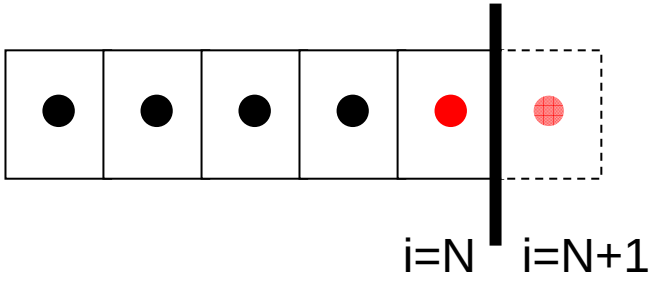
$$\frac{d\phi}{dx} = 0 @ x = x_{\max} \Rightarrow \frac{\phi_{N+1} - \phi_N}{\Delta x} = 0$$

$$\left(\frac{\phi_{N+1} - 2\phi_N + \phi_{N-1}}{\Delta x^2} \right) \times \Delta x + BF \times \Delta x = 0$$

$$\Rightarrow \left(\frac{-\phi_N + \phi_{N-1}}{\Delta x^2} \right) \times \Delta x + BF \times \Delta x = 0$$

$$\Rightarrow (0)\phi_{N+1} + \left(\frac{-1}{\Delta x} \right) \phi_N + \left(\frac{1}{\Delta x} \right) \phi_{N-1} = -BF \times \Delta x$$

AR **AD** **AL** **RHS**



境界面で断熱条件が成立するためには，$\phi_{N+1}=\phi_N$を満たすような仮想的な要素があると都合が良い

一次元熱伝導方程式：Jacobi法 heat_jacobi.f（3/3）

```
!C
!C-- ITERATIONs
      do iter= 1, ITERmax
        dPHImax= -1.d0
        do i= 1, N
          RESID  = RHS(i) - AL(i)*PHI0(i-1) - AR(i)*PHI0(i+1)
          dPHI   = RESID*rAD(i) - PHI0(i)
          dPHImax= dmax1 (dabs(dPHI), dPHImax)
          PHI(i) = PHI0(i) + dPHI
        enddo

        do i= 1, N
          PHI0(i) = PHI(i)
        enddo

        if (dPHImax.lt.EPS) exit
      enddo

!C
!C-- OUTPUT
      Xmax= (dfloat(N-1)+0.5d0)*dX
      do i= 1, N
        XX= dfloat(i-1)*dX
        T = -0.5d0*BF*(XX**2) + BF*Xmax*XX
        write (*,'(i8, 2(1pe16.6))') i, PHI(i), T
      enddo

      end program JACOBI_poi
```

$$\phi(i) = \frac{RHS(i) - A_L(i)\times\phi^0(i-1) - A_R(i)\times\phi^0(i+1)}{A_D(i)}$$

$$= \left[\frac{RHS(i) - A_L(i)\times\phi^0(i-1) - A_R(i)\times\phi^0(i+1)}{A_D(i)} - \phi^0(i) \right] + \phi^0(i)$$

$$T = \phi(analy.) = -\frac{1}{2} BF\, x^2 + BF\, x_{\max}\, x$$

一次元熱伝導方程式：SOR法
heat_sor.f（1/3）

```
!C
!C   1D Poisson Equation Solver by
!C   SOR (Successive Over Relaxation) Method
!C
!C   d/dx(dPHI/dx) + BF = 0
!C   PHI=0@x=0
!C
      program SOR_poi
      implicit REAL*8 (A-H,O-Z)

      integer :: N, ITERmax
      real(kind=8) :: dx, RESID, dPHI, dPHImax, BF, EPS
      real(kind=8), dimension(:), allocatable :: PHI, RHS, PHI0
      real(kind=8), dimension(:), allocatable :: rAD, AR, AL

!C
!C-- INIT.
      open  (11, file='input.dat', status='unknown')
       read (11,*) N
       read (11,*) dX, BF
       read (11,*) ITERmax
       read (11,*) EPS, OMEGA
      close (11)

      if (OMEGA.le.0.d0) then
        PI= 4.d0 * atan(1.d0)
        OMEGA= 2.d0/(1.d0+dsin(PI/dfloat(N)))
      endif
```

一次元熱伝導方程式：SOR法 heat_sor.f（1/3）

```
!C
!C   1D Poisson Equation Solver by
!C   SOR (Successive Over Relaxation) Method
!C
!C   d/dx(dPHI/dx) + BF = 0
!C   PHI=0@x=0
!C
      program SOR_poi
      implicit REAL*8 (A-H,O-Z)

      integer :: N, ITERmax
      real(kind=8) :: dx, OMEGA, RESID, dPHI, dPHImax, BF, EPS
      real(kind=8), dimension(:), allocatable :: PHI, RHS
      real(kind=8), dimension(:), allocatable :: rAD, AR, AL

!C
!C-- INIT.
      open  (11, file='input.dat', status='unknown')
       read (11,*) N
       read (11,*) dX, BF
       read (11,*) ITERmax
       read (11,*) EPS, OMEGA
      close (11)

      if (OMEGA.le.0.d0) then
        PI= 4.d0 * atan(1.d0)
        OMEGA= 2.d0/(1.d0+dsin(PI/dfloat(N)))
      endif
```

ωの最適値
（<0.0が入力された場合）

一次元熱伝導方程式:SOR法 heat_sor.f（2/3）

```
      allocate (PHI(N+1), rAD(N), AR(N), AL(N), RHS(N), PHI0(N))

       PHI = 0.d0
       PHI0= 0.d0

       AR = 1.d0/dX
       AL = 1.d0/dX
      rAD = 1.d0/(-2.d0/dX)
      RHS=  -BF * dX

       AL (1)= 0.d0
       AR (1)= 0.d0
      rAD (1)= 1.d0
       RHS(1)= 0.d0

       AR (N)= 0.d0
      rAD (N)= 1.d0/(-1.d0/dX)
```

$$\frac{d^2\phi}{dx^2} + BF = 0, \quad \phi = 0 @ x = 0, \frac{d\phi}{dx} = 0 @ x = x_{\max}$$

一次元熱伝導方程式:SOR法 heat_sor.f（2/3）

```
      allocate (PHI(N+1), rAD(N), AR(N), AL(N), RHS(N), PHI0(N))

       PHI = 0.d0
       PHI0= 0.d0

       AR = 1.d0/dX
       AL = 1.d0/dX
      rAD = 1.d0/(-2.d0/dX)
      RHS=  -BF * dX

       AL (1)= 0.d0
       AR (1)= 0.d0
      rAD (1)= 1.d0
       RHS(1)= 0.d0

       AR (N)= 0.d0
      rAD (N)= 1.d0/(-1.d0/dX)
```

$$\left(\frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{\Delta x^2} \right) \times V + BF \times V = 0$$

$$\left(\frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{\Delta x^2} \right) \times \Delta x + BF \times \Delta x = 0$$

一次元熱伝導方程式:SOR法 heat_sor.f（2/3）

```
      allocate (PHI(N+1), rAD(N), AR(N), AL(N), RHS(N), PHI0(N))

       PHI = 0.d0
       PHI0= 0.d0

       AR = 1.d0/dX
       AL = 1.d0/dX
      rAD = 1.d0/(-2.d0/dX)
      RHS=  -BF * dX

       AL (1)= 0.d0
       AR (1)= 0.d0
      rAD (1)= 1.d0
       RHS(1)= 0.d0

       AR (N)= 0.d0
      rAD (N)= 1.d0/(-1.d0/dX)
```

$$rA_D(i) = \frac{1}{A_D(i)}$$

割り算は計算時間がかかるため

$$\left(\frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{\Delta x^2} \right) \times \Delta x + BF \times \Delta x = 0$$

$$\frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{\Delta x} = \left(\frac{1}{\Delta x} \right) \phi_{i-1} - \left(\frac{2}{\Delta x} \right) \phi_i + \left(\frac{1}{\Delta x} \right) \phi_{i+1} = -BF \times \Delta x$$

AL　AD　AR　RHS

一次元熱伝導方程式：SOR法 heat_sor.f（3/3）

```
!C
!C-- ITERATIONs
      do iter= 1, ITERmax
        dPHImax= -1.d0
        do i= 1, N
          RESID  = RHS(i) - AL(i)*PHI(i-1) - AR(i)*PHI(i+1)
          dPHI   = OMEGA * (RESID*rAD(i)-PHI(i))
          dPHImax= dmax1 (dabs(dPHI), dPHImax)
          PHI(i) = PHI(i) + dPHI
        enddo

        if (dPHImax.lt.EPS) exit
      enddo

!C
!C-- OUTPUT
      Xmax= (dfloat(N-1)+0.5d0)*dX
      do i= 1, N
        XX= dfloat(i-1)*dX
        T = -0.5d0*BF*(XX**2) + BF*Xmax*XX
        write (*,'(i8, 2(1pe16.6))') i, PHI(i), T
      enddo

      end program SOR_poi
```

$$\phi_{GS}(i) = \frac{BF(i) - A_L(i) \times \phi(i-1) - A_R(i) \times \phi^0(i+1)}{A_D(i)}$$

$$\phi_{SOR}(i) = \phi^0(i) + \omega \times \left[\phi_{GS}(i) - \phi^0(i) \right]$$

JacobiとSOR

JACOBI

```
      do iter= 1, ITERmax
        do i= 1, N
          RESID  = RHS(i) - AL(i)*PHI0(i-1) - AR(i)*PHI0(i+1)
          dPHI   = RESID*rAD(i) - PHI0(i)
          PHI(i) = PHI0(i) + dPHI
        enddo

        do i= 2, N
          PHI0(i) = PHI(i)
        enddo
      enddo
```

反復の間ϕの値は不変(ϕ^0)

SOR

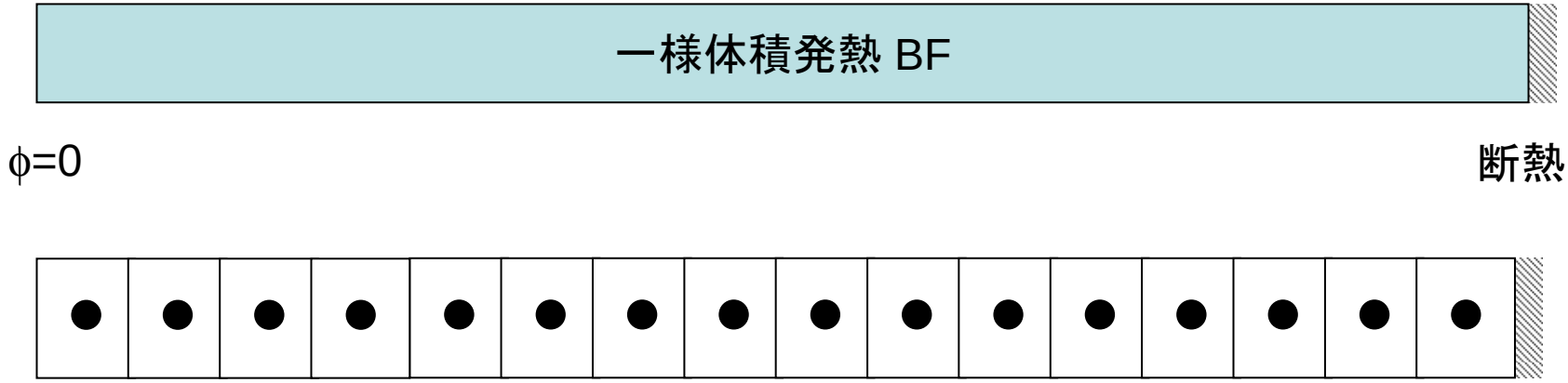
```
      do iter= 1, ITERmax
        do i= 1, N
          RESID  = RHS(i) - AL(i)*PHI(i-1) - AR(i)*PHI(i+1)
          dPHI   = OMEGA * (RESID*rAD(i)-PHI(i))
          PHI(i) = PHI(i) + dPHI
        enddo
      enddo
```

反復の間ϕの値は常に
最新値を使用

一次元熱伝導方程式ソルバーの並列化

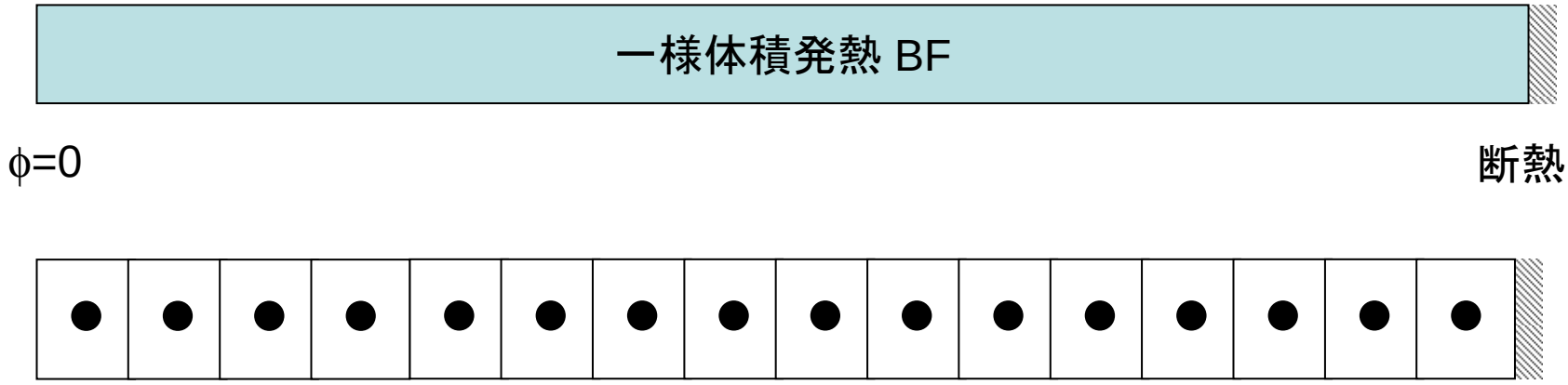
$$\frac{d^2\phi}{dx^2} + BF = 0, \quad \phi = 0 @ x = 0, \frac{d\phi}{dx} = 0 @ x = x_{\max}$$

$$\phi = -\frac{1}{2} BF\, x^2 + BF\, x_{\max}\, x$$



一次元熱伝導方程式ソルバーの並列化

- 全体データと局所データ
- 例えば，NG=16の問題を4PEで実行する場合，各PEにおいてはNL=16/4=4となる

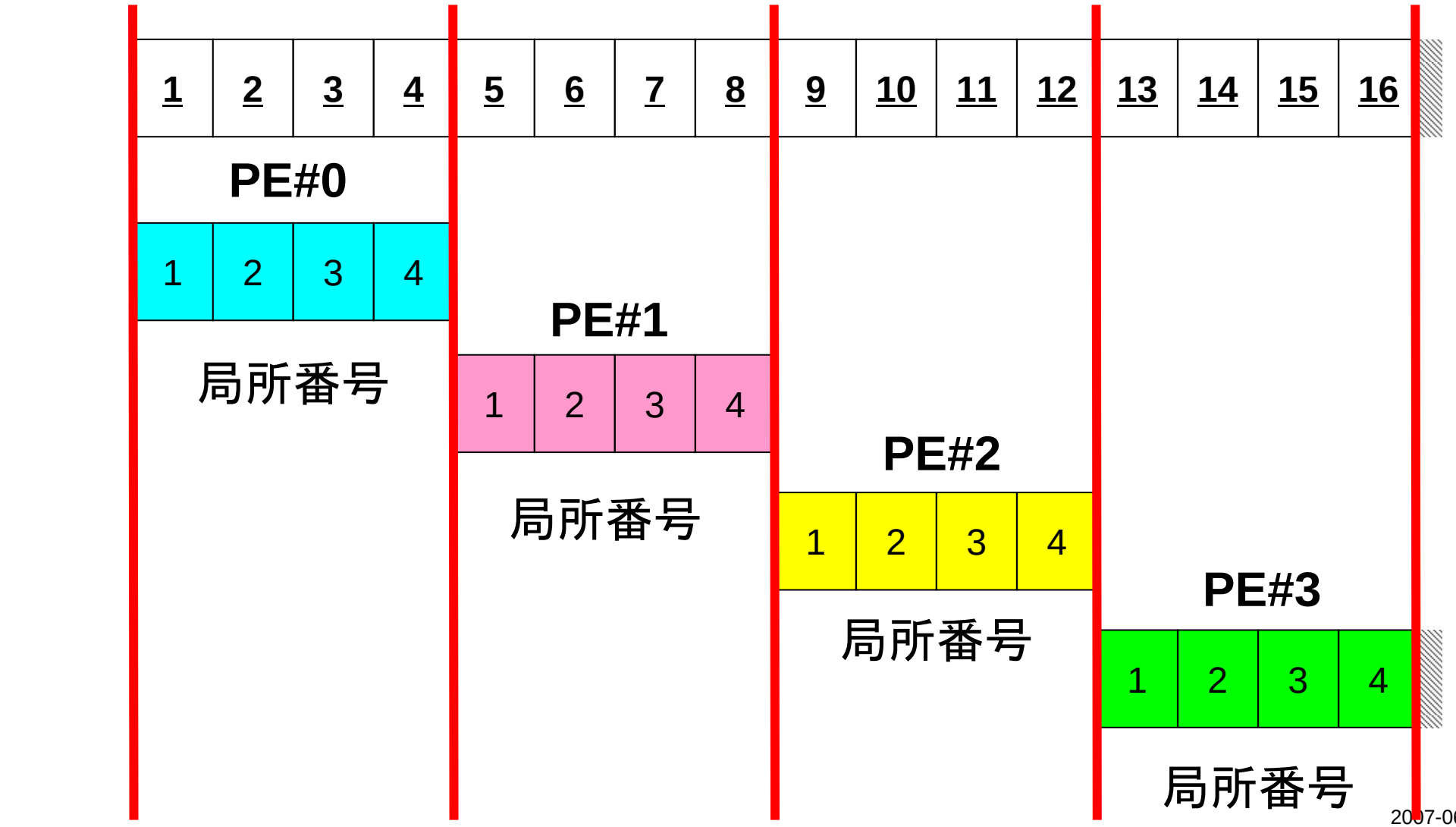


全体データと局所データ
NG=16, PE#=4

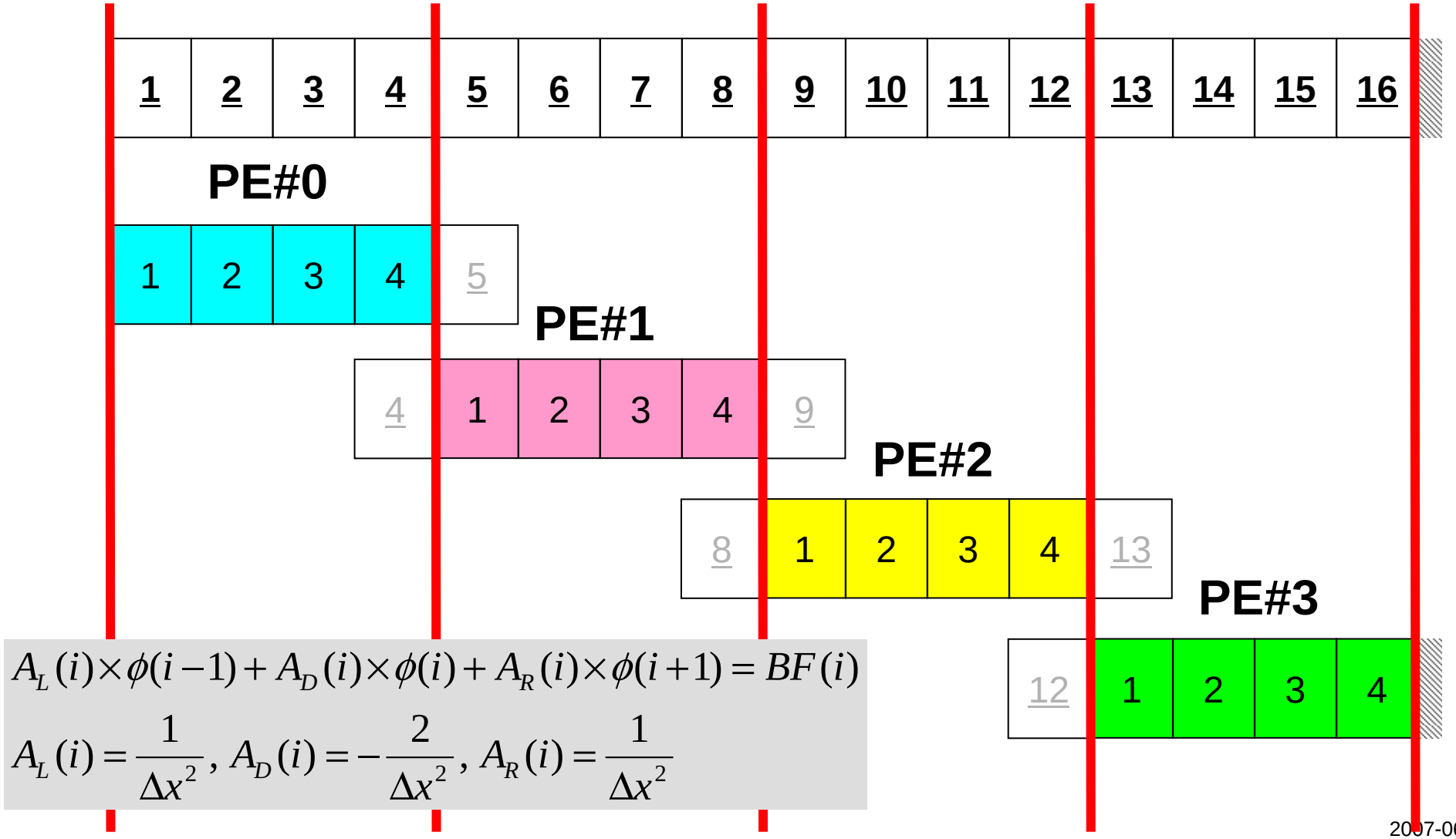
全体番号

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

全体データと局所データ
NG=16, PE#=4



差分法のオペレーションを実施するためには隣接要素の情報が必要



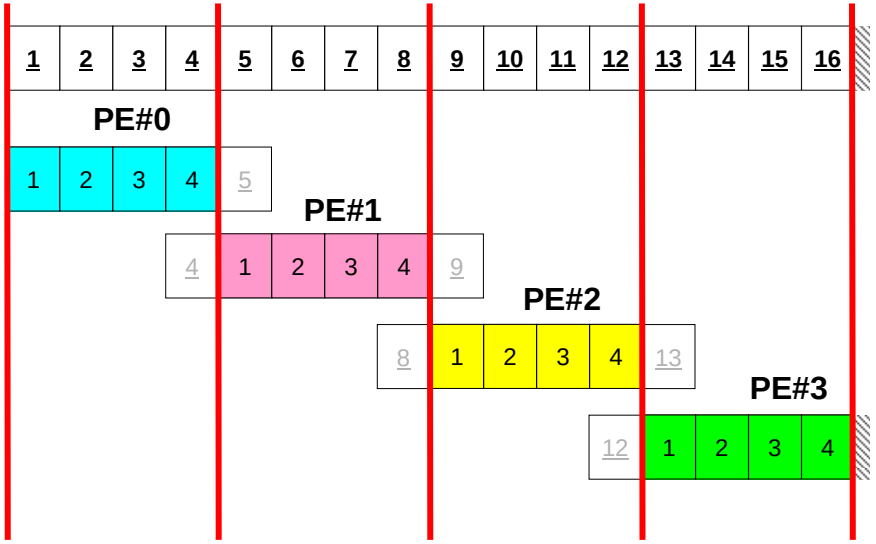
$$A_L(i)\times\phi(i-1)+A_D(i)\times\phi(i)+A_R(i)\times\phi(i+1)=BF(i)$$

$$A_L(i)=\frac{1}{\Delta x^2},\ A_D(i)=-\frac{2}{\Delta x^2},\ A_R(i)=\frac{1}{\Delta x^2}$$

一次元熱伝導方程式ソルバーの並列化

- 全体データと局所データ
- 差分法のオペレーションには隣接要素の値が必要
  - 領域（PE）の境界では隣接領域に属する要素からの情報が必要
  - 例えば，PE#1での計算を完結させるためには，PE#0の「4」番，PE#2の「1」番要素の情報が必要

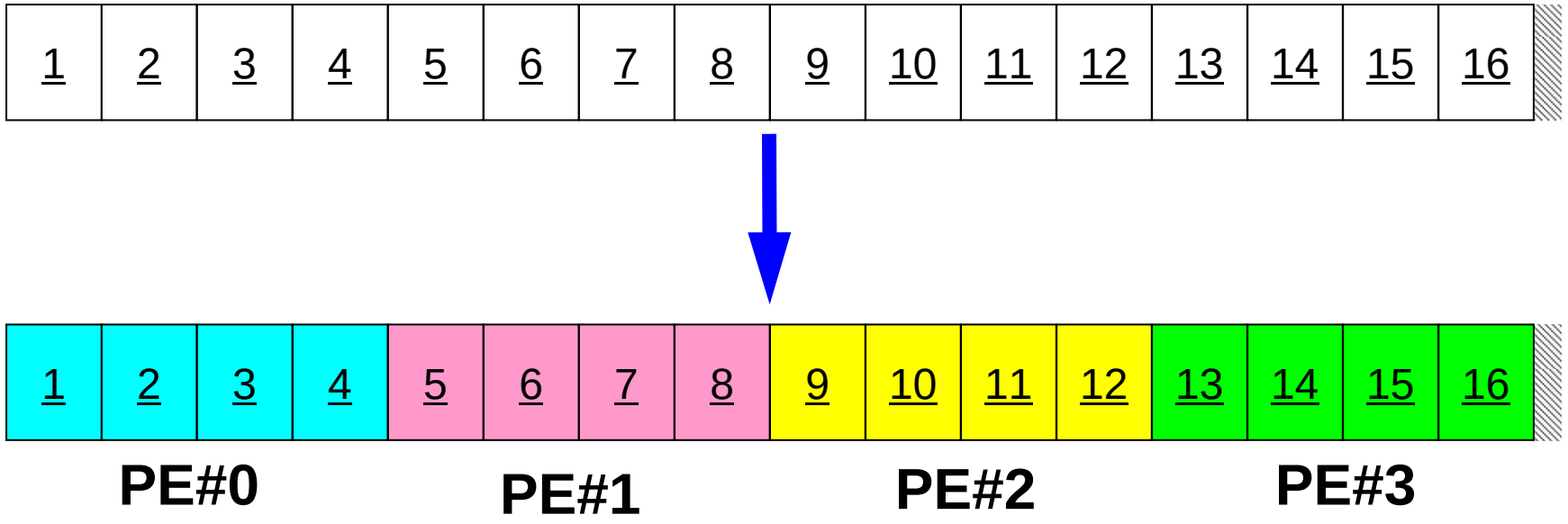
- 隣接領域からの情報を考慮可能なデータ構造が必要
  - それほど単純では無い・・・



一次元差分法モデル局所データ構造(1/5)

NG=16の一次元領域

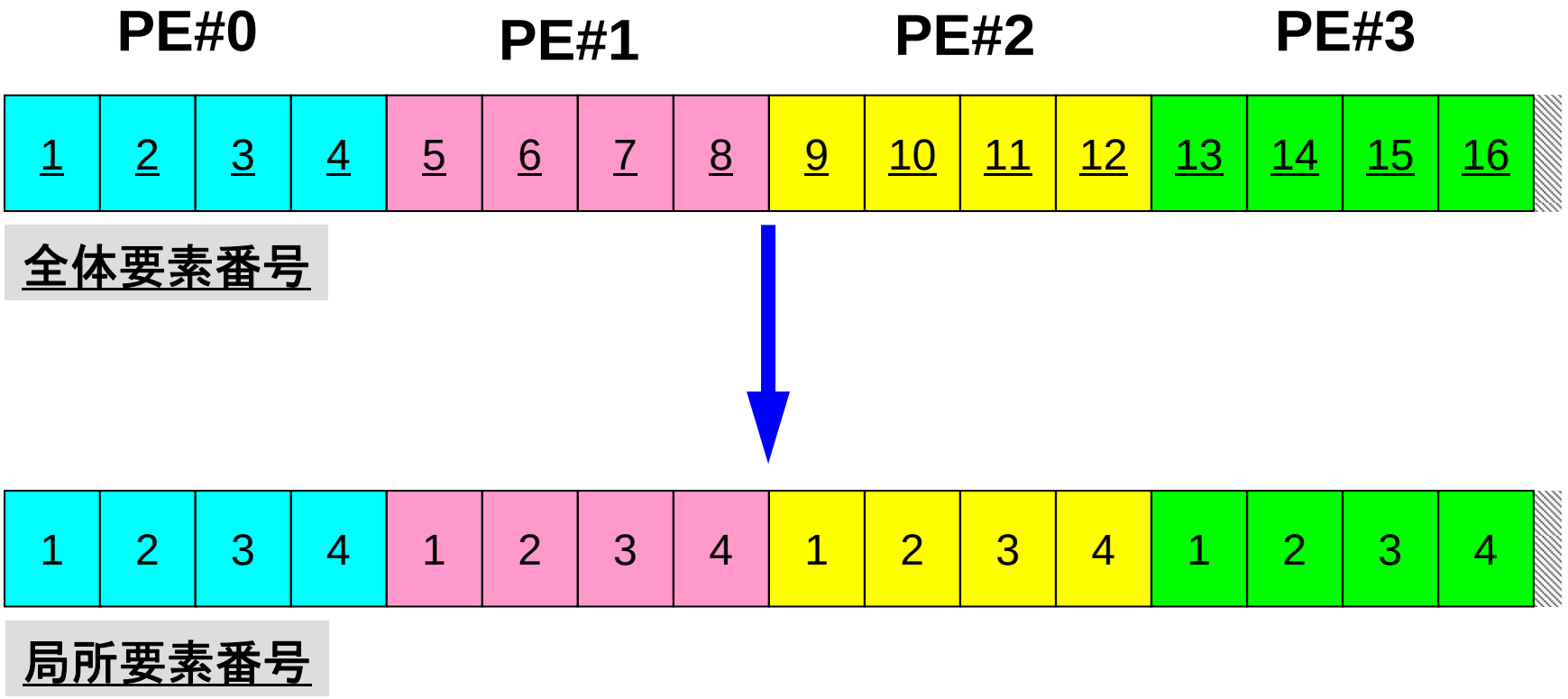
- NG=16の一次元領域を4領域(プロセッサ)に分割して並列に差分法(二次精度)の計算を実施するために必要なデータ構造を考える。



一次元差分法モデル局所データ構造(2/5)

局所要素番号

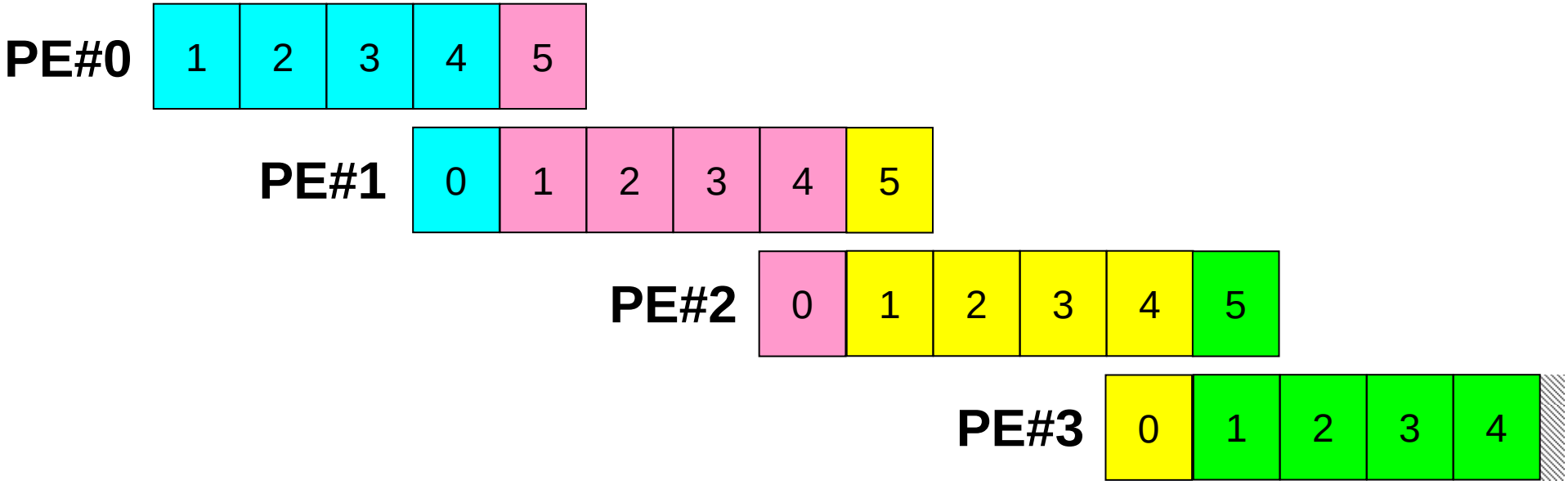
- 各領域(プロセッサ)において，N=4の局所データを扱う



一次元差分法モデル局所データ構造(3/5)

領域外の要素

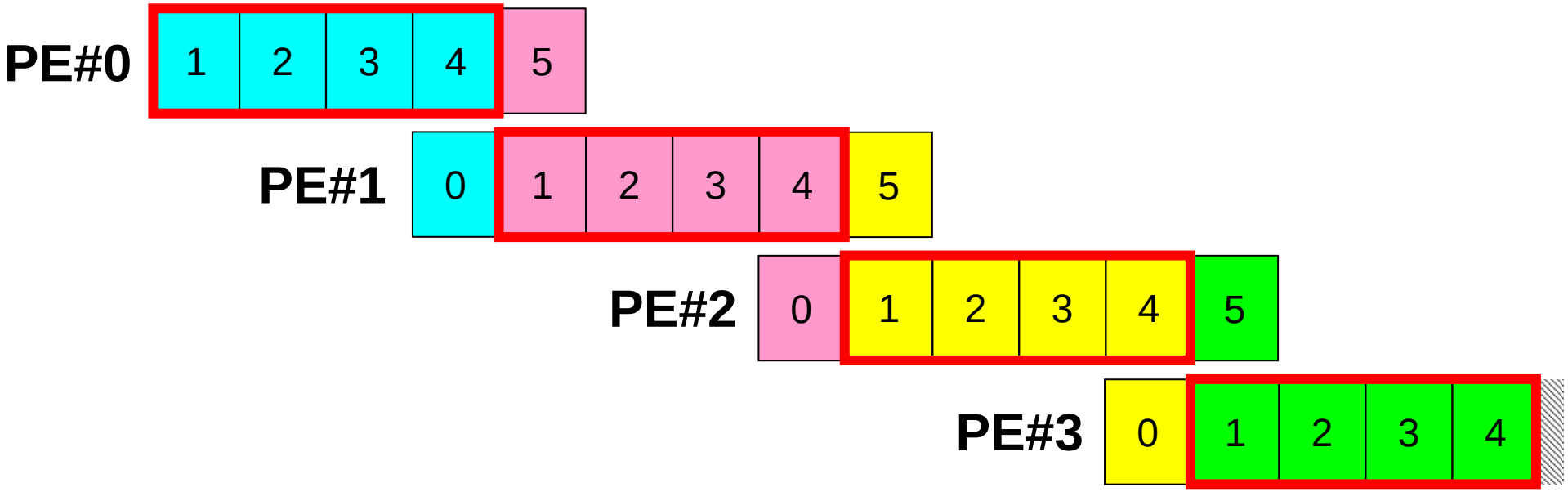
- 差分法(二次精度中央差分)の計算を並列に実施するためには，本来領域外の要素の影響を考慮する必要がある。
  - 領域間オーバーラップ
- 要素番号が0，N+1(=5)の要素を加えた，局所データ構造



一次元差分法モデル局所データ構造(4/5)

必要な情報

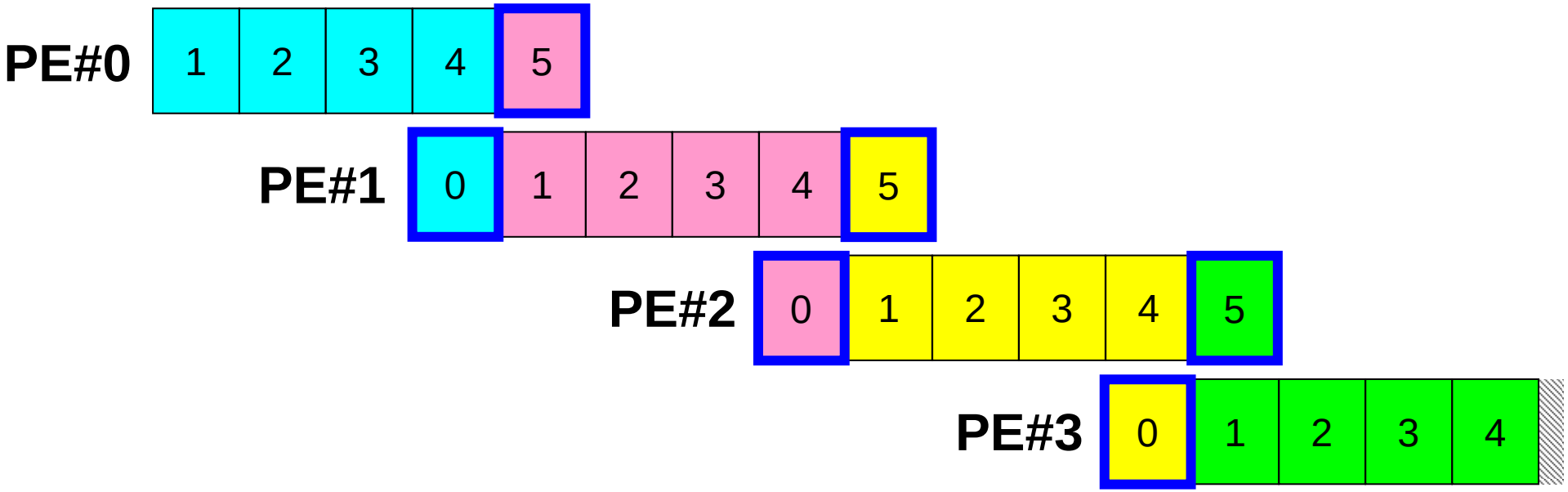
- 領域内の要素，オーバーラップ(本来領域外)要素の区別
  - 領域内要素(i=1~N)　　:内点(Interior/Internal Points)



一次元差分法モデル局所データ構造(4/5)

必要な情報

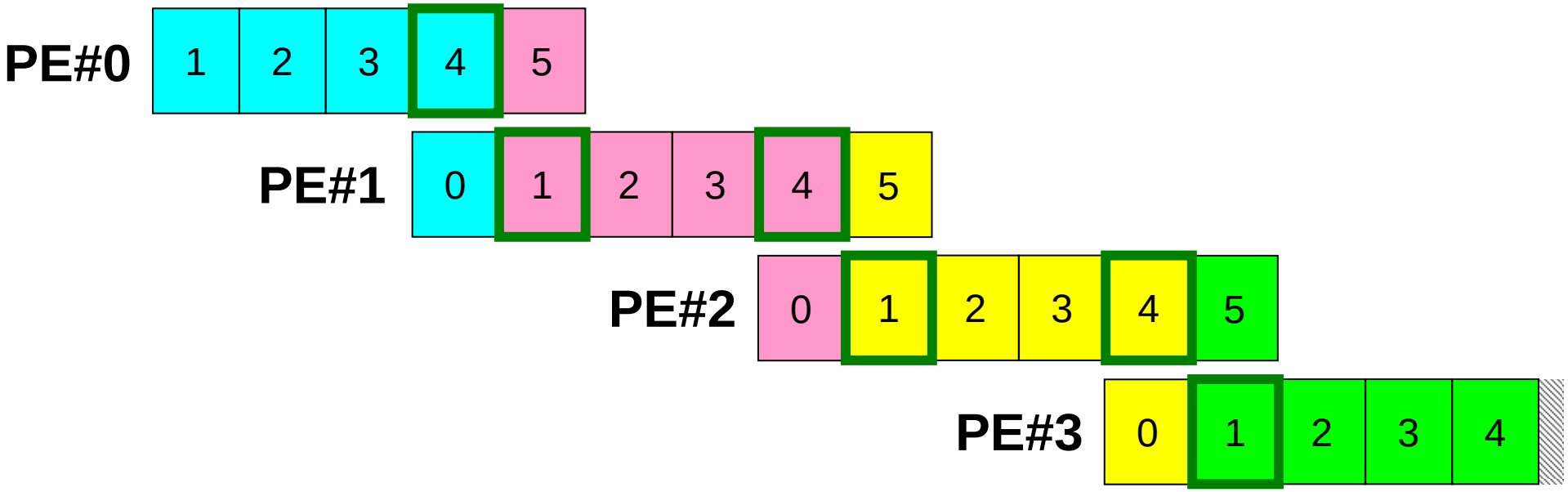
- 領域内要素，オーバーラップ(本来領域外)要素の区別
  - 領域内要素(i=1~N)　　:内点(Interior/Internal Points)
  - オーバーラップ要素(i=0, N+1):外点(Exterior/External Points)
    - i=0，i=N+1の要素が存在しない場合もある



一次元差分法モデル局所データ構造(4/5)

必要な情報

- 領域内要素，オーバーラップ(本来領域外)要素の区別
  - 領域内要素(i=1~N)　　:内点(Interior/Internal Points)
  - オーバーラップ要素(i=0, N+1):外点(Exterior/External Points)
    - i=0，i=N+1の要素が存在しない場合もある
  - 「内点」のうち他領域の「外点」となっている要素:境界点(Boundary Points)



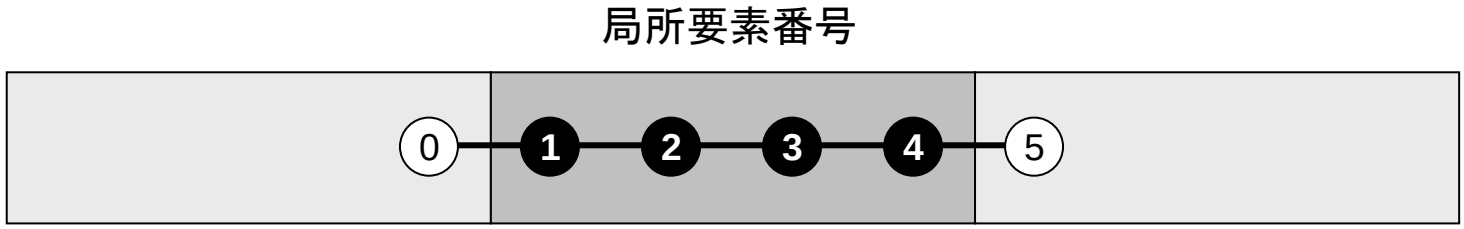
一次元差分法モデル局所データ構造(5/5)

必要な情報(続き)

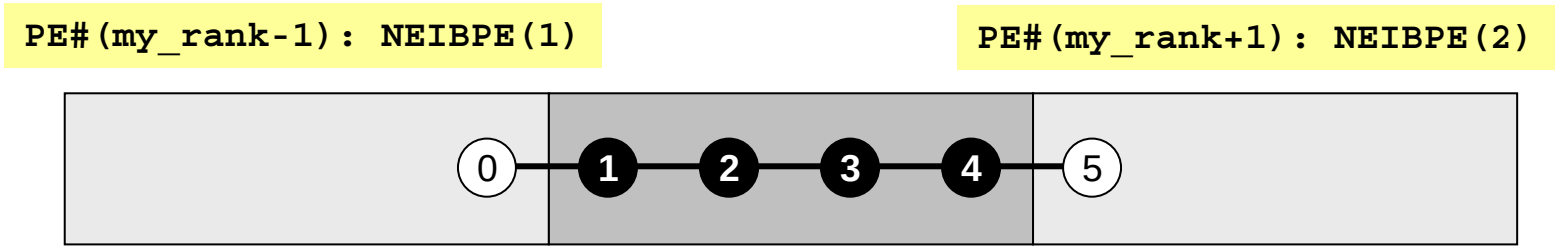
- 「領域間オーバーラップ」を考慮した並列計算を実現するためには？
  - 各領域(プロセッサ)における「境界点」の情報を，各隣接領域に「外点」の情報として配信する必要がある。
  - そのための局所データ構造が必要
- 隣接している領域数，隣接している領域番号
  - 1対1通信の場合重要
  - オーバーラップ要素を共有している領域(プロセッサ)
- 隣接領域との「通信テーブル」
  - 境界点：隣接領域への「送信」(send)
  - 外点　：隣接領域からの「受信」(recv)

- 復習
  - 一次元熱伝導プログラム
  - 一次元差分法の並列化に必要な局所データ構造
- 通信テーブル(一般化)
- 課題S2解説

1D差分法の並列計算に必要な情報(1/3)
内点・外点・(境界点)

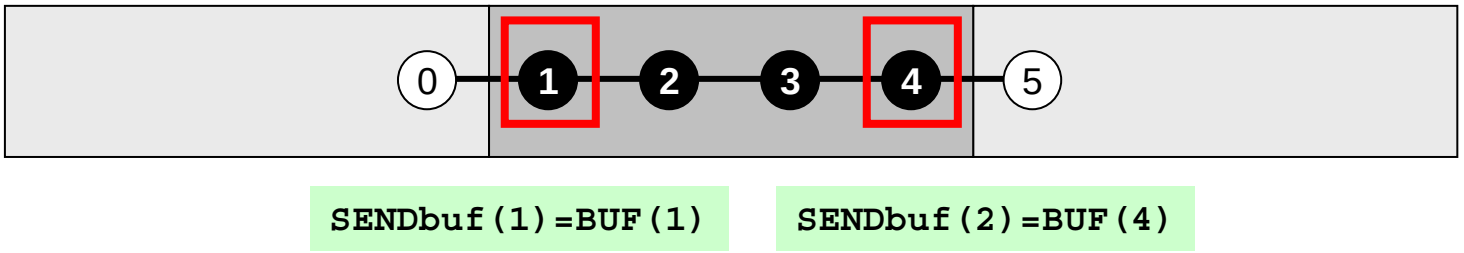


1D差分法の並列計算に必要な情報(2/3)
隣接プロセッサ情報

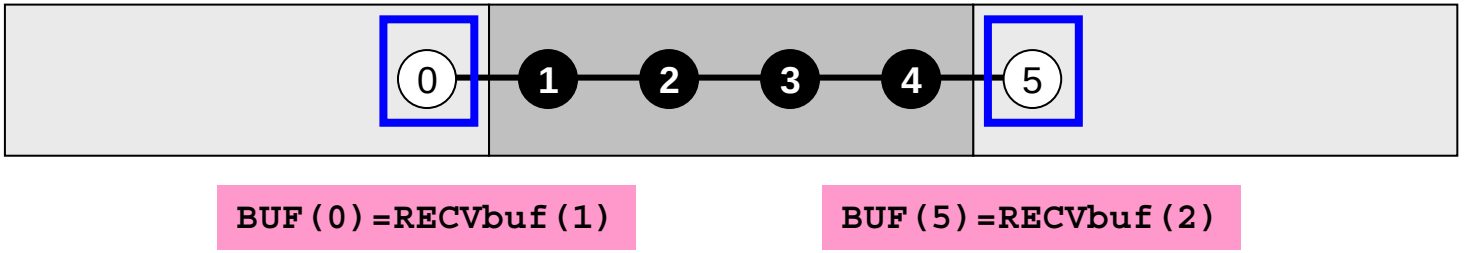


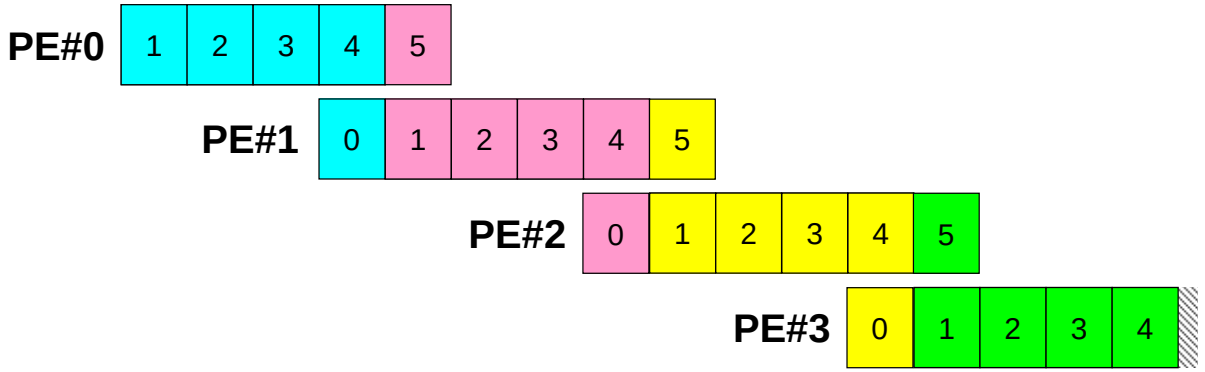
1D差分法の並列計算に必要な情報
通信テーブル

隣接PEへの
送信(境界点)



隣接PEからの
受信(外点)

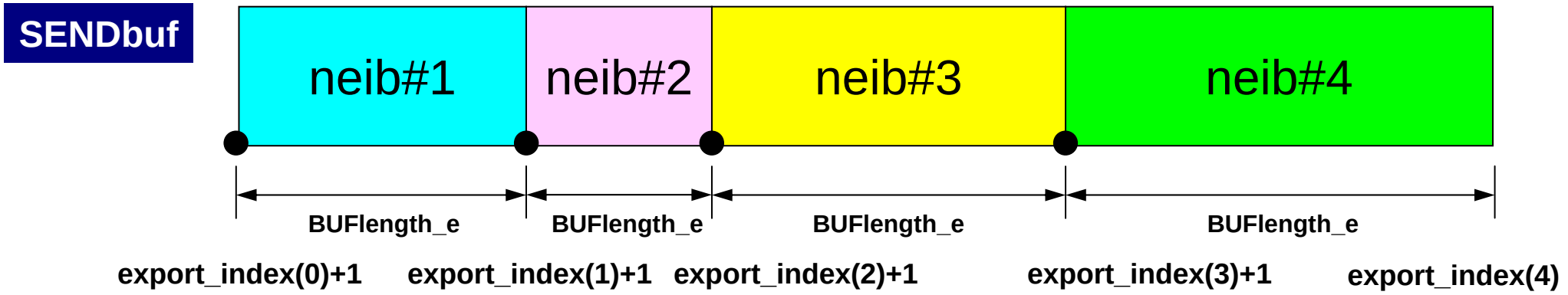




一般化された通信テーブル：送信

- 送信相手
  - NEIBPETOT，NEIB(neib)
- それぞれの送信相手に送るメッセージサイズ
  - export_index(neib), neib= 1, NEIBPETOT
- 「境界点」番号
  - export_item(k), k= 1, export_index(NEIBPETOT)
- それぞれの送信相手に送るメッセージ
  - SENDbuf(k), k= 1, export_index(NEIBPETOT)

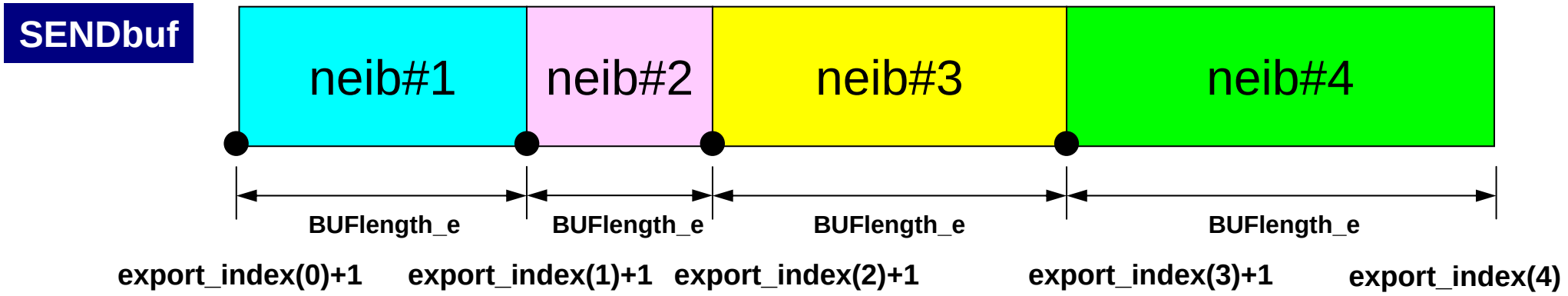
送信（MPI_Isend/Irecv/Waitall）




```
 do neib= 1, NEIBPETOT
   do k= export_index(neib-1)+1, export_index(neib)
     kk= export_item(k)
     SENDbuf(k)= VAL(kk)
   enddo
 enddo

 do neib= 1, NEIBPETOT
   iS_e= export_index(neib-1) + 1
   iE_e= export_index(neib  )
   BUFlength_e= iE_e + 1 - iS_e

   call MPI_ISEND                                                    &
&         (SENDbuf(iS_e), BUFlength_e, MPI_INTEGER, NEIBPE(neib), 0,&
&          MPI_COMM_WORLD, request_send(neib), ierr)
 enddo

 call MPI_WAITALL (NEIBPETOT, request_send, stat_recv, ierr)
```

送信バッファへの代入
温度などの変数を直接送信，受信に使うのではなく，このようなバッファへ一回代入して計算することを勧める。

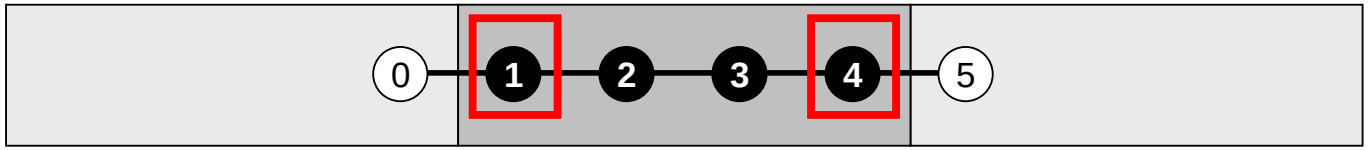
送信（MPI_Sendrecv）




```
 do neib= 1, NEIBPETOT
   do k= export_index(neib-1)+1, export_index(neib)
     kk= export_item(k)
     SENDbuf(k)= VAL(kk)
   enddo
 enddo

 do neib= 1, NEIBPETOT
   iS_e= export_index(neib-1) + 1
   iE_e= export_index(neib  )
   BUFlength_i= iE_e + 1 - iS_e

   call MPI_SENDRECV                                                  &
&         (SENDbuf(iS_e), BUFlength_e, MPI_INTEGER, NEIBPE(neib), 0,&
&          RECVbuf(iS_i), BUFlength_i, MPI_INTEGER, NEIBPE(neib), 0,&
&          MPI_COMM_WORLD, stat_sr, ierr)
  enddo
```

送信バッファへの代入

送信:一次元問題



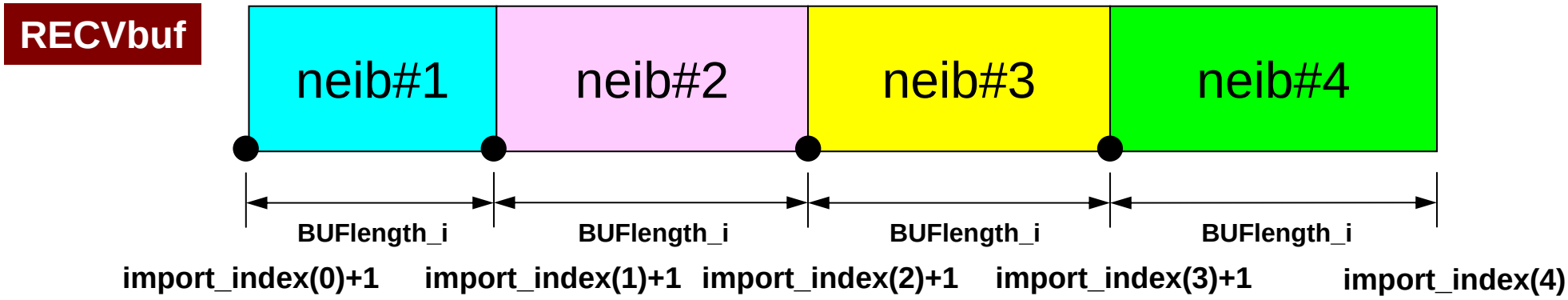
- 送信相手
  - NEIBPETOT, NEIB(neib)
    - NEIBPETOT=2, NEIB(1)= my_rank-1, NEIB(2)= my_rank+1
- それぞれの送信相手に送るメッセージサイズ
  - export_index(neib), neib= 1, NEIBPETOT
    - export_index(0)=0, export_index(1)= 1, export_index(2)= 2
- 「境界点」番号
  - export_item(k), k= 1, export_index(NEIBPETOT)
    - export_item(1)= 1, export_item(2)= N
- それぞれの送信相手に送るメッセージ
  - SENDbuf(k), k= 1, export_index(NEIBPETOT)
    - SENDbuf(1)= BUF(1), SENDbuf(2)= BUF(N)

一般化された通信テーブル：受信

- 受信相手
  - NEIBPETOT，NEIB(neib)
- それぞれの受信相手から受け取るメッセージサイズ
  - import_index(neib)
- 「外点」番号
  - import_item(k), k= 1, import_index(NEIBPETOT)
- それぞれの受信相手から受け取るメッセージ
  - RECVbuf(k), k= 1, import_index(NEIBPETOT)

受信（MPI_Isend/Irecv/Waitall）

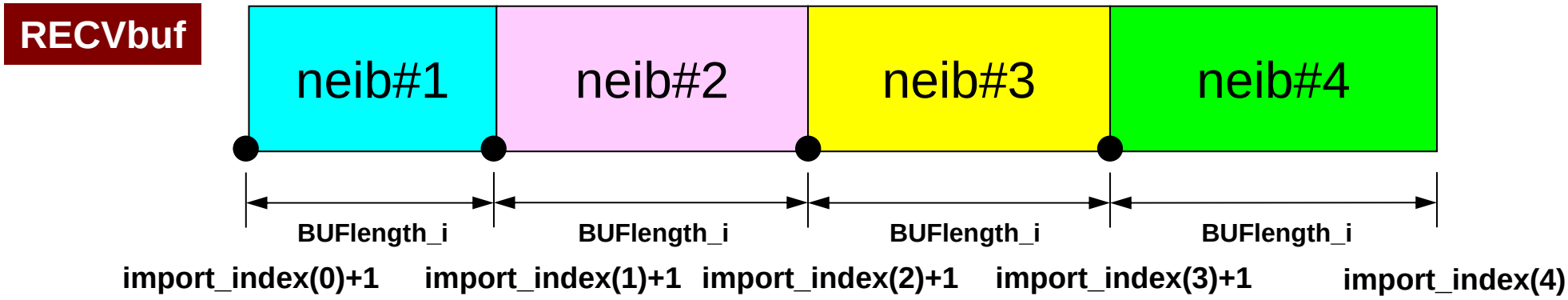
```
 do neib= 1, NEIBPETOT
   iS_i= import_index(neib-1) + 1
   iE_i= import_index(neib  )
   BUFlength_i= iE_i + 1 - iS_i

   call MPI_IRECV                                                  &
&          (RECVbuf(iS_i), BUFlength_i, MPI_INTEGER, NEIBPE(neib), 0,&
&           MPI_COMM_WORLD, request_recv(neib), ierr)
  enddo

  call MPI_WAITALL (NEIBPETOT, request_recv, stat_recv, ierr)

  do neib= 1, NEIBPETOT
     do k= import_index(neib-1)+1, import_index(neib)
       kk= import_item(k)
       VAL(kk)= RECVbuf(k)                          受信バッファから代入
     enddo
  enddo
```



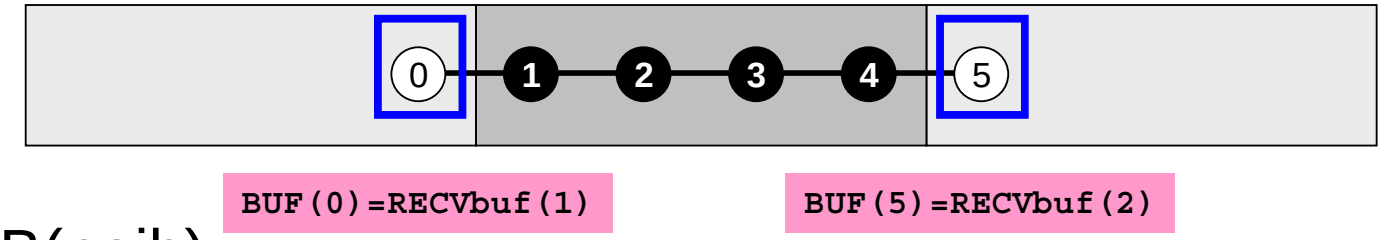
受信（MPI_Sendrecv）

```
 do neib= 1, NEIBPETOT
   iS_i= import_index(neib-1) + 1
   iE_i= import_index(neib  )
   BUFlength_i= iE_i + 1 - iS_i

   call MPI_SENDRECV                                                  &
&         (SENDbuf(iS_e), BUFlength_e, MPI_INTEGER, NEIBPE(neib), 0,&
&          RECVbuf(iS_i), BUFlength_i, MPI_INTEGER, NEIBPE(neib), 0,&
&          MPI_COMM_WORLD, stat_sr, ierr)
  enddo

  do neib= 1, NEIBPETOT
    do k= import_index(neib-1)+1, import_index(neib)
      kk= import_item(k)
      VAL(kk)= RECVbuf(k)
    enddo
  enddo
```

受信バッファからの代入



受信:一次元問題



- 受信相手
  - NEIBPETOT, NEIB(neib)
    - NEIBPETOT=2, NEIB(1)= my_rank-1, NEIB(2)= my_rank+1
- それぞれの受信相手から受け取るメッセージサイズ
  - import_index(neib), neib= 1, NEIBPETOT
    - import_index(0)=0, import_index(1)= 1, import_index(2)= 2
- 「外点」番号
  - import_item(k), k= 1, import_index(NEIBPETOT)
    - import_item(1)= 0, import_item(2)= N+1
- それぞれの受信相手から受け取るメッセージ
  - RECVbuf(k), k= 1, import_index(NEIBPETOT)
    - BUF(0)=RECVbuf(1), BUF(N+1)=RECVbuf(2)

一般化された通信テーブル：1d-srb1.f（1/7）

```
!C
!C***
!C*** program SEND_RECV
!C***
!C
      implicit REAL*8 (A-H,O-Z)
      include  'mpif.h'

      integer(kind=4) :: my_rank, PETOT
      integer(kind=4) :: N, NEIBPETOT, BUFlength
      integer(kind=4), dimension(2) :: NEIBPE

      integer(kind=4), dimension(0:2) :: import_index, export_index
      integer(kind=4), dimension(  2) :: import_item , export_item

      integer(kind=4), dimension(2) :: SENDbuf, RECVbuf

      integer(kind=4), dimension(:), allocatable :: BUF

      integer(kind=4), dimension(:,:), allocatable :: stat_send
      integer(kind=4), dimension(:,:), allocatable :: stat_recv
      integer(kind=4), dimension(:  ), allocatable :: request_send
      integer(kind=4), dimension(:  ), allocatable :: request_recv

!C
!C +-----------+
!C | INIT. MPI |
!C +-----------+
!C===
      call MPI_INIT      (ierr)
      call MPI_COMM_SIZE (MPI_COMM_WORLD, PETOT, ierr )
      call MPI_COMM_RANK (MPI_COMM_WORLD, my_rank, ierr )
!C===
```

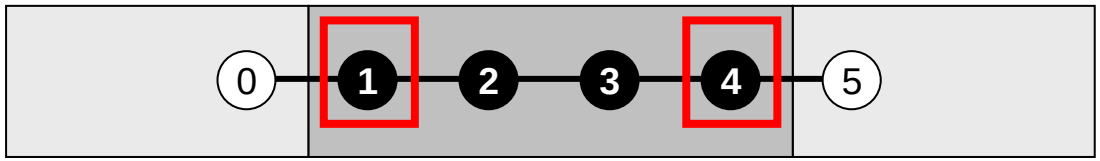
一般化された通信テーブル：1d-srb1.f（2/7）

```
!C
!C +------+
!C | INIT |
!C +------+
!C===
      N= 4
      allocate (BUF(0:N+1))
      BUF= 0

      if (my_rank.eq.0) open (11, file='1d.0', status='unknown')
      if (my_rank.eq.1) open (11, file='1d.1', status='unknown')
      if (my_rank.eq.2) open (11, file='1d.2', status='unknown')
      if (my_rank.eq.3) open (11, file='1d.3', status='unknown')
        do i= 1, N
          read (11,*) BUF(i)
        enddo
      close (11)

      iSTART= 0
      iEND  = N+1
      if (my_rank.eq.0      ) iSTART= 1
      if (my_rank.eq.PETOT-1) iEND  = N
      do i= iSTART, iEND
        write (*,'(a, 3i8)') '%%% before', my_rank, i, BUF(i)
      enddo
```

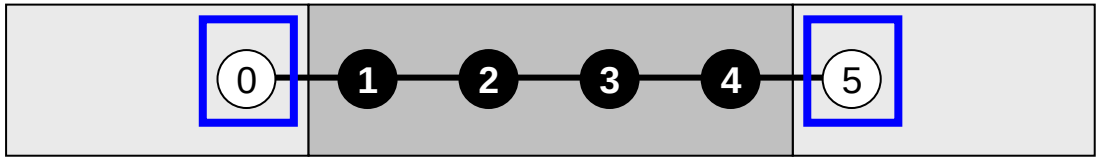
一般化された通信テーブル：1d-srb1.f（3/7）

```
!C
!C-- COMMUNICATION
      NEIBPETOT= 2
      if (my_rank.eq.0      ) NEIBPETOT= 1
      if (my_rank.eq.PETOT-1) NEIBPETOT= 1
      if (PETOT.eq.1)         NEIBPETOT= 0

      NEIBPE(1)= my_rank - 1
      NEIBPE(2)= my_rank + 1

      if (my_rank.eq.0      ) NEIBPE(1)= my_rank + 1
      if (my_rank.eq.PETOT-1) NEIBPE(1)= my_rank - 1

      import_index= 0
      export_index= 0
      import_item = 0
      export_item = 0

      import_index(1)= 1
      import_index(2)= 2
      import_item (1)= 0
      import_item (2)= N+1

      export_index(1)= 1
      export_index(2)= 2
      export_item (1)= 1
      export_item (2)= N

      if (my_rank.eq.0) then
        import_item (1)= N+1
        export_item (1)= N
      endif

      BUFlength= 1

      write (*,'(a,10i5)') '#NEIB (my_rank, NEIBPETOT, NEPBPE)',         &
     &                     my_rank, NEIBPETOT, (NEIBPE(i),i=1,NEIBPETOT)
!C===
```

一般化された通信テーブル



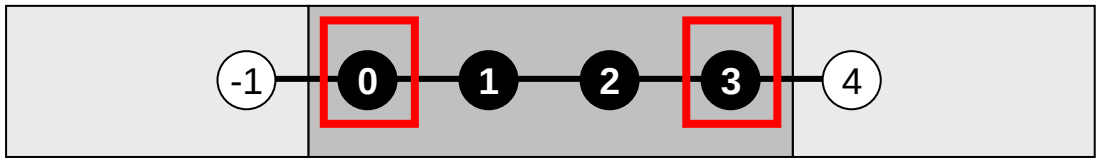
SENDbuf(1)=BUF(1)　　SENDbuf(2)=BUF(4)



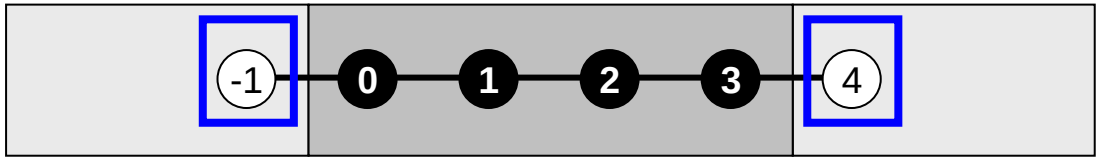
BUF(0)=RECVbuf(1)　　BUF(5)=RECVbuf(2)

```
NEIBPETOT= 2
NEIBPE(1)= my_rank - 1
NEIBPE(2)= my_rank + 1

import_index(1)= 1
import_index(2)= 2
import_item  (1)= 0
import_item  (2)= N+1

export_index(1)= 1
export_index(2)= 2
export_item  (1)= 1
export_item  (2)= N

if (my_rank.eq.0) then
  import_item  (1)= N+1
  export_item  (1)= N
  NEIBPE(1)= my_rank+1
endif
```

一般化された通信テーブル:C言語



SENDbuf(1)=BUF(1)　　SENDbuf(2)=BUF(4)



BUF(0)=RECVbuf(1)　　BUF(5)=RECVbuf(2)

C言語の場合BUF[-1]という配列は本来存在しないが,
コンパイラによっては通ってしまう場合がある。

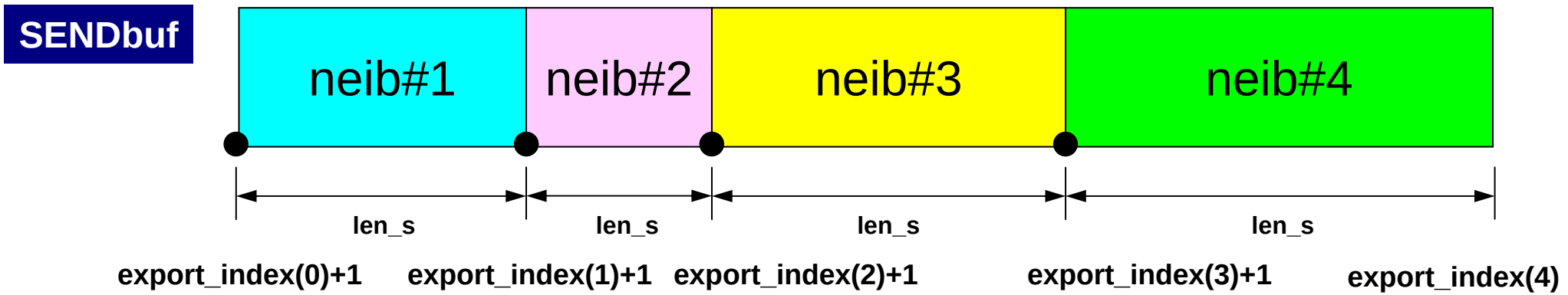
```
NEIBPETOT= 2
NEIBPE(1)= my_rank - 1
NEIBPE(2)= my_rank + 1

import_index(1)= 1
import_index(2)= 2
import_item (1)= -1
import_item (2)= N

export_index(1)= 1
export_index(2)= 2
export_item (1)= 0
export_item (2)= N-1

if (my_rank.eq.0) then
  import_item (1)= N
  export_item (1)= N-1
  NEIBPE(1)= my_rank+1
endif
```

一般化された通信テーブル：1d-srb1.f（4/7）

```
!C
!C +-----------------------------+
!C | INIT. arrays for MPI_WAITALL |
!C +-----------------------------+
!C===
      allocate (stat_send(MPI_STATUS_SIZE,NEIBPETOT))
      allocate (stat_recv(MPI_STATUS_SIZE,NEIBPETOT))
      allocate (request_send(NEIBPETOT))
      allocate (request_recv(NEIBPETOT))
!C===
```

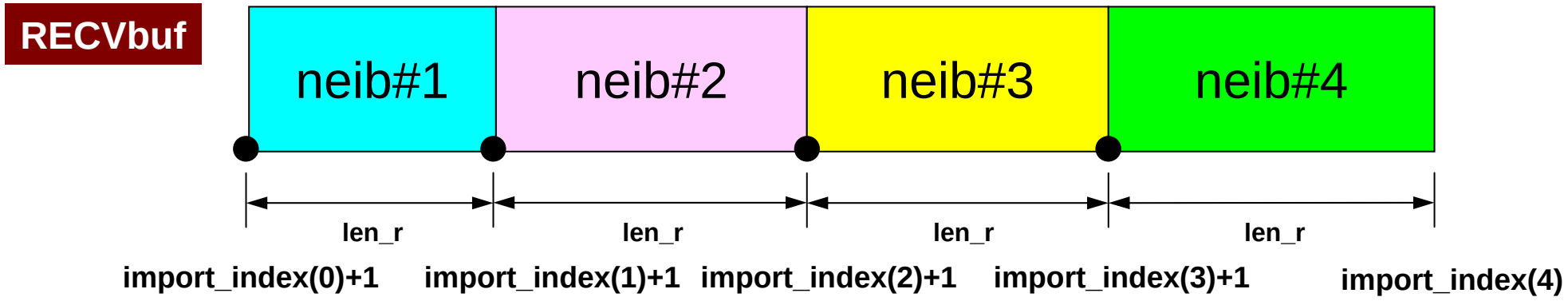
一般化された通信テーブル：1d-srb1.f（5/7）

```
!C
!C-- PREPARE send  buffer
      do neib= 1, NEIBPETOT
        do k= export_index(neib-1)+1, export_index(neib)
          kk= export_item(k)
          SENDbuf(k)= BUF(kk)
        enddo
      enddo

!C
!C-- SEND
      do neib= 1, NEIBPETOT
        is   = export_index(neib-1) + 1
        len_s= export_index(neib) - export_index(neib-1)
        call MPI_ISEND (SENDbuf(is), len_s, MPI_INTEGER,                &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD,                &
     &                  request_send(neib), ierr)
      enddo
```



一般化された通信テーブル：1d-srb1.f（6/7）

```
!C
!C-- RECV
      do neib= 1, NEIBPETOT
        ir   = import_index(neib-1) + 1
        len_r= import_index(neib) - import_index(neib-1)
        call MPI_IRECV (RECVbuf(ir), len_r, MPI_INTEGER,                &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD,                &
     &                  request_recv(neib), ierr)
      enddo

!C
!C-- WAITall for RECV
      call MPI_WAITALL (NEIBPETOT, request_recv, stat_recv, ierr)

!C
!C-- update array
      do neib= 1, NEIBPETOT
        do k= import_index(neib-1)+1, import_index(neib)
          kk= import_item(k)
          BUF(kk)= RECVbuf(k)
        enddo
      enddo
```



一般化された通信テーブル：1d-srb1.f（7/7）

```
!C
!C +-------------------+
!C | WAITall for SEND |
!C +-------------------+
!C===
      call MPI_WAITALL (NEIBPETOT, request_send, stat_send, ierr)
!C===

      call MPI_FINALIZE (ierr)

      end
```

- 復習
  - 一次元熱伝導プログラム
  - 一次元差分法の並列化に必要な局所データ構造
- 通信テーブル(一般化)
- 課題S2解説

課題S2（1/2）

- 「<$S2>/heat_jacobi.f」,「<$S2>/heat_jacobi.c」を参考にして，一次元熱伝導方程式をJacobi法によって解くプログラムを並列化せよ（**S2-1**）。
- 「<$S2>/heat_gs.f」,「<$S2>/heat_gs.c」を参考にして，一次元熱伝導方程式をGauss-Seidel法によって解くプログラムを並列化せよ（**S2-2**）。

- 「<$S2>/1d-srb1.f, 1d-srb2.f」,「<$S2>/1d-srb1.c, 1d-srb2.c」を参考にして「一般化された通信テーブル」を使用せよ。
  - BUFのかわりに温度をやりとりする

課題S2（2/2）

- NG=100およびNG=103の場合について，1，2，4，8CPUを使用して計算してみよ。
  - 実は，N=100程度では並列化の効果は余り･･･全く得られない
- Gauss-Seidel法の場合，単純に並列化すると，領域分割数を増加させた場合，反復回数が増加する。その理由について考えてみよ。
  - Jacobi法の場合はこのような現象が生じない。その理由についても併せて検討すること。

課題S2-1,S2-2の方針

- 基本的に一次元例題プログラムのアプローチを踏襲
  - **<$S2>/1d-srb1.f, 1d-srb2.f**
  - **<$S2>/1d-srb1.c, 1d-srb2.c**

- 1D熱伝導方程式
  - Jacobi, GS法⇒収束するまで反復を繰り返す
  - 各反復の最初に温度の値を更新する必要がある⇒1対1通信
  - 最大残差値⇒グループ通信によって全領域での最大値を求める必要がある

ファイル

```
>$ cd <$07S>
```
各自作成したディレクトリ

FORTRAN

```
>$ cp /home/nakajima/class/2007summer/F/s2r-f.tar .
>$ tar xvf s2r-f.tar
```

C

```
>$ cp /home/nakajima/class/2007summer/C/s2r-c.tar .
>$ tar xvf s2r-c.tar
```

直下に「/S2-ref」というディレクトリができている。
<$07S>/S2-refを<$S2R>と呼ぶ。

ファイル

s2-1a.f/c S2-1(Jacobi), MPI_Isend/Irecv/Waitall
s2-1b.f/c S2-1(Jacobi), MPI_Isend/Irecv/Waitall

s2-2a.f/c S2-2(GS), MPI_Isend/Irecv/Waitall
s2-2b.f/c S2-2(GS), MPI_Sendrecv
s2-2c.f/c S2-2(GS), MPI_Sendrecv（逐次版）

input.dat
cinput.dat
go.sh

1d.0~1d.3 s2-1bで使用

実行例

input.dat

```
  103
  1.d0 1.d0
  50000
  1.d-07 1.50
```

```
$ mpif90 -O s2-1a.f
$ <modify go.sh>
$ qsub go.sh
```

PE数を変えてそれぞれ計算してみよ

```
#!/bin/sh
#PBS -N 1d
#PBS -o test.lst
#PBS -e err
#PBS -l nodes=2:ppn=2
#PBS -l walltime=10:00
cd $PBS_O_WORKDIR
NPROCS=`wc -l < $PBS_NODEFILE`
mpirun -v -machinefile
$PBS_NODEFILE -np $NPROCS a.out
```

課題S2-1：1D熱伝導方程式（Jacobi）

s2-1a.f（1/7）

```
!C
!C   1D Poisson Equation Solver by
!C   Jacobi Method
!C
!C   d/dx(dPHI/dx) + BF = 0
!C   PHI=0@x=0
!C
      program JACOBI_1
      implicit REAL*8 (A-H,O-Z)
      include 'mpif.h'

      integer :: PETOT, my_rank, ierr
      integer :: Ng, N, ITERmax
      real(kind=8) :: dx, OMEGA, RESID, dPHI, dPHImax, BF, EPS
      real(kind=8), dimension(:), allocatable :: PHI, PHI0, RHS
      real(kind=8), dimension(:), allocatable :: rAD, AR, AL

      integer(kind=4) :: NEIBPETOT, BUFlength
      integer(kind=4), dimension(2) :: NEIBPE

      integer(kind=4), dimension(0:2) :: import_index, export_index
      integer(kind=4), dimension(  2) :: import_item , export_item

      real(kind=8), dimension(2) :: SENDbuf, RECVbuf

      integer(kind=4), dimension(:,:), allocatable :: stat_send
      integer(kind=4), dimension(:,:), allocatable :: stat_recv
      integer(kind=4), dimension(:  ), allocatable :: request_send
      integer(kind=4), dimension(:  ), allocatable :: request_recv
```

課題S2-1：1D熱伝導方程式（Jacobi）

s2-1a.f（2/7）

```
!C
!C +-------+
!C | INIT. |
!C +-------+
!C===

!C
!C-- MPI init.
      call MPI_INIT      (ierr)
      call MPI_COMM_SIZE (MPI_COMM_WORLD, PETOT, ierr )
      call MPI_COMM_RANK (MPI_COMM_WORLD, my_rank, ierr )
!C
!C-- CTRL data
      if (my_rank.eq.0) then
        open  (11, file='input.dat', status='unknown')
          read (11,*) Ng
          read (11,*) dX, BF
          read (11,*) ITERmax
          read (11,*) EPS
        close (11)
      endif

      call MPI_BCAST (Ng     , 1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
      call MPI_BCAST (ITERmax, 1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
      call MPI_BCAST (dx,    1, MPI_DOUBLE_PRECISION, 0,                &
     &                                                MPI_COMM_WORLD, ierr)
      call MPI_BCAST (BF,    1, MPI_DOUBLE_PRECISION, 0,                &
     &                                                MPI_COMM_WORLD, ierr)
      call MPI_BCAST (EPS,   1, MPI_DOUBLE_PRECISION, 0,                &
     &                                                MPI_COMM_WORLD, ierr)
!C
!C-- LOCAL MESH size
      N = Ng / PETOT
      nr= Ng - N*PETOT
      if (my_rank+1.le.nr) N= N + 1
```

課題S2-1：1D熱伝導方程式（Jacobi）

s2-1a.f（2/7）

```
!C
!C +-------+
!C | INIT. |
!C +-------+
!C===

!C
!C-- MPI init.
      call MPI_INIT      (ierr)
      call MPI_COMM_SIZE (MPI_COMM_WORLD, PETOT, ierr )
      call MPI_COMM_RANK (MPI_COMM_WORLD, my_rank, ierr )
!C
!C-- CTRL data
      if (my_rank.eq.0) then
        open  (11, file='input.dat', status='unknown')
          read (11,*) Ng
          read (11,*) dX, BF
          read (11,*) ITERmax
          read (11,*) EPS
        close (11)
      endif

      call MPI_BCAST (Ng     , 1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
      call MPI_BCAST (ITERmax, 1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
      call MPI_BCAST (dx,    1, MPI_DOUBLE_PRECISION, 0,                  &
```

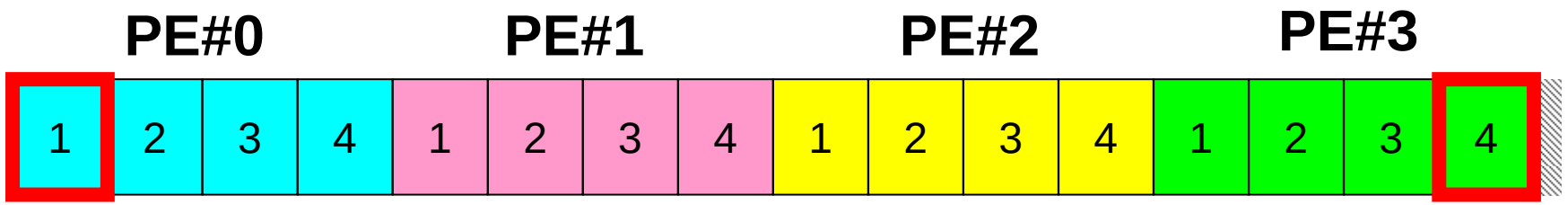
Ng=100, PETOT=4：全てのPEでN=25
Ng=103, PETOT=4：N=26（PE#0），N=26（PE#1），N=26（PE#2），N=25（PE#3）

```
!C
!C-- LOCAL MESH size
      N = Ng / PETOT
      nr= Ng - N*PETOT
      if (my_rank+1.le.nr) N= N + 1
```

課題S2-1 ：1D熱伝導方程式（Jacobi）

s2-1a.f （3/7）

```
!C
!C-- MATRIX
      allocate (PHI(0:N+1), rAD(N), AR(N), AL(N), RHS(N))
      allocate (PHI0(0:N+1))
       PHI = 0.d0
       PHI0= 0.d0
       AR  = 1.d0/dX
       AL  = 1.d0/dX
      rAD  = 1.d0/(-2.d0/dX)
      RHS  =  -BF * dX

      if (my_rank.eq.0) then
         AL (1)= 0.d0
         AR (1)= 0.d0
        rAD (1)= 1.d0
         RHS(1)= 0.d0
      endif

      if (my_rank.eq.PETOT-1) then
         AR (N)= 0.d0
        rAD (N)= 1.d0/(-1.d0/dX)
      endif
```



境界条件の処理：i=1

```
if (my_rank.eq.0) then
   AL (1)= 0.d0
   AR (1)= 0.d0
  rAD (1)= 1.d
  RHS (1)= 0.d0
endif
```

$$\phi = 0 @ x = 0 \Rightarrow \phi_1 = 0$$

$$\Rightarrow (0)\phi_2 + (1)\phi_1 + (0)\phi_0 = 0$$

AR AD AL RHS

境界条件の処理：i=N

```
if (my_rank.eq.PETOT-1) then
   AR (N)= 0.d0
  rAD (N)= 1.d0/(-1.d0/dX)
endif
```

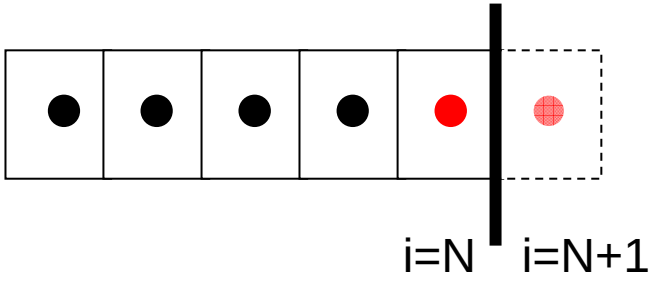
$$\frac{d\phi}{dx} = 0 @ x = x_{\max} \Rightarrow \frac{\phi_{N+1} - \phi_N}{\Delta x} = 0$$

$$\left(\frac{\phi_{N+1} - 2\phi_N + \phi_{N-1}}{\Delta x^2} \right) \times \Delta x + BF \times \Delta x = 0$$

$$\Rightarrow \left(\frac{-\phi_N + \phi_{N-1}}{\Delta x^2} \right) \times \Delta x + BF \times \Delta x = 0$$

$$\Rightarrow (0)\phi_{N+1} + \left(\frac{-1}{\Delta x} \right) \phi_N + \left(\frac{1}{\Delta x} \right) \phi_{N-1} = -BF \times \Delta x$$

AR **AD** **AL** **RHS**



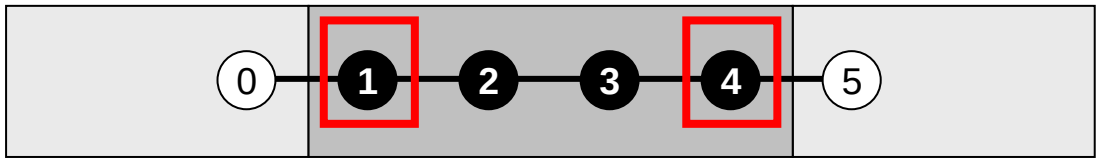
境界面で断熱条件が成立するためには，$\phi_{N+1}=\phi_N$を満たすような仮想的な要素があると都合が良い

課題S2-1：1D熱伝導方程式（Jacobi）

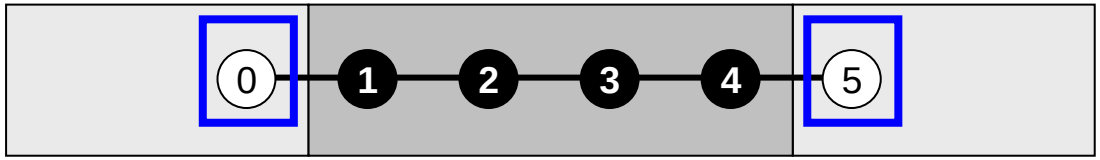
s2-1a.f（4/7）

```
!C
!C-- COMMUNICATION
      NEIBPETOT= 2
      if (my_rank.eq.0      ) NEIBPETOT= 1
      if (my_rank.eq.PETOT-1) NEIBPETOT= 1
      if (PETOT.eq.1)         NEIBPETOT= 0

      NEIBPE(1)= my_rank - 1
      NEIBPE(2)= my_rank + 1

      if (my_rank.eq.0      ) NEIBPE(1)= my_rank + 1
      if (my_rank.eq.PETOT-1) NEIBPE(1)= my_rank - 1

      import_index= 0
      export_index= 0
      import_item = 0
      export_item = 0

      import_index(1)= 1
      import_index(2)= 2
      import_item (1)= 0
      import_item (2)= N+1

      export_index(1)= 1
      export_index(2)= 2
      export_item (1)= 1
      export_item (2)= N

      if (my_rank.eq.0) then
        import_item (1)= N+1
        export_item (1)= N
      endif

      BUFlength= 1
```

一般化された通信テーブル



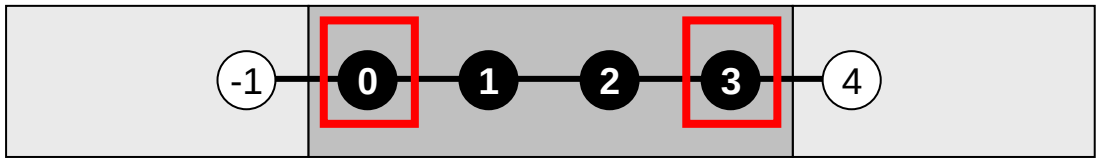
SENDbuf(1)=BUF(1)　　SENDbuf(2)=BUF(4)



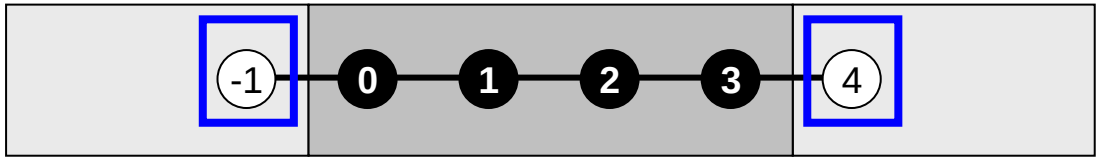
BUF(0)=RECVbuf(1)　　BUF(5)=RECVbuf(2)

```
NEIBPETOT= 2
NEIBPE(1)= my_rank - 1
NEIBPE(2)= my_rank + 1

import_index(1)= 1
import_index(2)= 2
import_item  (1)= 0
import_item  (2)= N+1

export_index(1)= 1
export_index(2)= 2
export_item  (1)= 1
export_item  (2)= N

if (my_rank.eq.0) then
  import_item  (1)= N+1
  export_item  (1)= N
  NEIBPE(1)= my_rank+1
endif
```

一般化された通信テーブル:C言語



SENDbuf(1)=BUF(1)　　SENDbuf(2)=BUF(4)



BUF(0)=RECVbuf(1)　　BUF(5)=RECVbuf(2)

C言語の場合BUF[-1]という配列は本来存在しないが,
コンパイラによっては通ってしまう場合がある。

```
NEIBPETOT= 2
NEIBPE(1)= my_rank - 1
NEIBPE(2)= my_rank + 1

import_index(1)= 1
import_index(2)= 2
import_item (1)= -1
import_item (2)= N

export_index(1)= 1
export_index(2)= 2
export_item (1)= 0
export_item (2)= N-1

if (my_rank.eq.0) then
  import_item (1)= N
  export_item (1)= N-1
  NEIBPE(1)= my_rank+1
endif
```

課題S2-1 ：1D熱伝導方程式（Jacobi）

s2-1a.f （5/7）

```
!C
!C-- INIT. arrays for MPI_WAITALL

      allocate (stat_send(MPI_STATUS_SIZE,NEIBPETOT))
      allocate (stat_recv(MPI_STATUS_SIZE,NEIBPETOT))
      allocate (request_send(NEIBPETOT))
      allocate (request_recv(NEIBPETOT))
!C===

!C
!C +------------+
!C | ITERATIONs |
!C +------------+
!C===
      STARTtime= MPI_WTIME()
      do iter= 1, ITERmax
        dPHImax= -1.d0
!C
!C-- PREPARE send  buffer
        do neib= 1, NEIBPETOT
          do k= export_index(neib-1)+1, export_index(neib)
            kk= export_item(k)
            SENDbuf(k)= PHI0(kk)
          enddo
        enddo
!C
!C-- SEND.
        do neib= 1, NEIBPETOT
          is   = export_index(neib-1) + 1
          len_s= export_index(neib) - export_index(neib-1)
          call MPI_ISEND (SENDbuf(is), len_s,                          &
     &                    MPI_DOUBLE_PRECISION,                        &
     &                    NEIBPE(neib), 0, MPI_COMM_WORLD,             &
     &                    request_send(neib), ierr)
        enddo
```

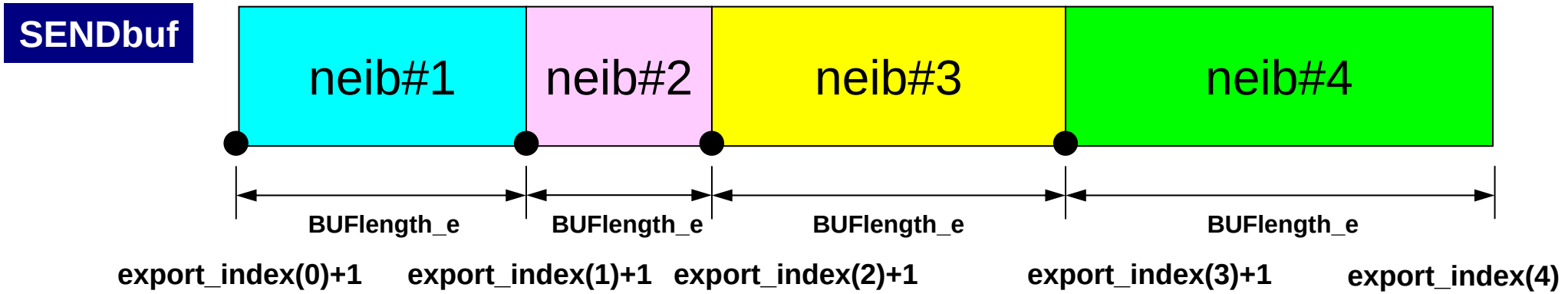
課題S2-1：1D熱伝導方程式（Jacobi）

s2-1a.f（5/7）

```
!C
!C-- INIT. arrays for MPI_WAITALL

      allocate (stat_send(MPI_STATUS_SIZE,NEIBPETOT))
      allocate (stat_recv(MPI_STATUS_SIZE,NEIBPETOT))
      allocate (request_send(NEIBPETOT))
      allocate (request_recv(NEIBPETOT))
!C===

!C
!C +------------+
!C | ITERATIONs |
!C +------------+
!C===

      STARTtime= MPI_WTIME()
      do iter= 1, ITERmax
        dPHImax= -1.d0
!C
!C-- PREPARE send  buffer
        do neib= 1, NEIBPETOT
          do k= export_index(neib-1)+1, export_index(neib)
            kk= export_item(k)
            SENDbuf(k)= PHI0(kk)
          enddo
        enddo
!C
!C-- SEND.
        do neib= 1, NEIBPETOT
          is   = export_index(neib-1) + 1
          len_s= export_index(neib) - export_index(neib-1)
          call MPI_ISEND (SENDbuf(is), len_s,                    &
     &                    MPI_DOUBLE_PRECISION,                  &
     &                    NEIBPE(neib), 0, MPI_COMM_WORLD,       &
     &                    request_send(neib), ierr)
        enddo
```

各反復の最初にまず通信を実施し，最新のPHI0の値を得ておく（送信）

送信（MPI_Isend/Irecv/Waitall）




```
 do neib= 1, NEIBPETOT
   do k= export_index(neib-1)+1, export_index(neib)
     kk= export_item(k)
     SENDbuf(k)= VAL(kk)
   enddo
 enddo

 do neib= 1, NEIBPETOT
   iS_e= export_index(neib-1) + 1
   iE_e= export_index(neib  )
   BUFlength_e= iE_e + 1 - iS_e

   call MPI_ISEND                                                    &
&         (SENDbuf(iS_e), BUFlength_e, MPI_INTEGER, NEIBPE(neib), 0,&
&          MPI_COMM_WORLD, request_send(neib), ierr)
 enddo

 call MPI_WAITALL (NEIBPETOT, request_send, stat_recv, ierr)
```

送信バッファへの代入
温度などの変数を直接送信，受信に使うのではなく，このようなバッファへ一回代入して計算することを勧める。

課題S2-1：1D熱伝導方程式（Jacobi）

s2-1a.f（6/7）

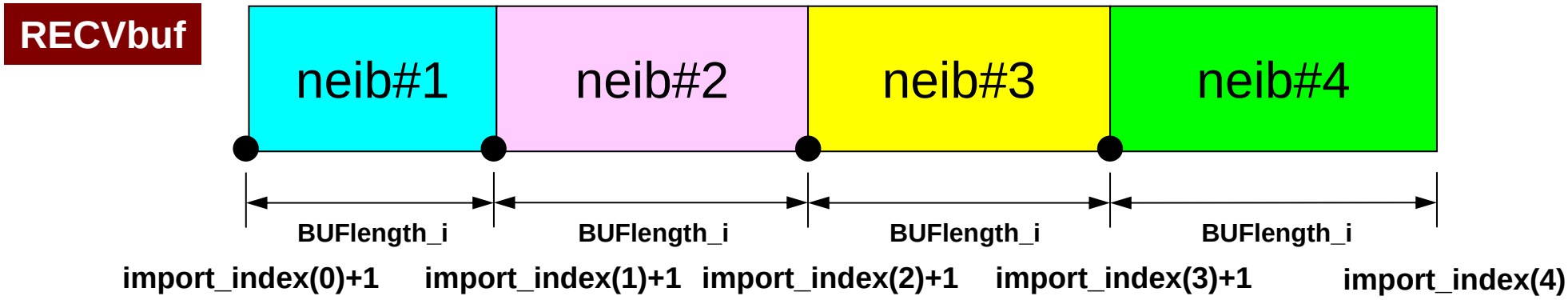
```
!C
!C-- RECV.
        do neib= 1, NEIBPETOT
          ir    = import_index(neib-1) + 1
          len_r= import_index(neib) - import_index(neib-1)
          call MPI_IRECV (RECVbuf(ir), len_r,                          &
     &                    MPI_DOUBLE_PRECISION,                        &
     &                    NEIBPE(neib), 0, MPI_COMM_WORLD,             &
     &                    request_recv(neib), ierr)
        enddo
        call MPI_WAITALL (NEIBPETOT, request_recv, stat_recv, ierr)

!C
!C-- UPDATE
        do neib= 1, NEIBPETOT
          do k= import_index(neib-1)+1, import_index(neib)
            kk= import_item(k)
            PHI0(kk)= RECVbuf(k)
          enddo
        enddo
```

各反復の最初にまず通信を実施し，最新のPHI0の値を得ておく（受信）

受信（MPI_Sendrecv）

```
 do neib= 1, NEIBPETOT
   iS_i= import_index(neib-1) + 1
   iE_i= import_index(neib  )
   BUFlength_i= iE_i + 1 - iS_i

   call MPI_SENDRECV                                                  &
&         (SENDbuf(iS_e), BUFlength_e, MPI_INTEGER, NEIBPE(neib), 0,&
&          RECVbuf(iS_i), BUFlength_i, MPI_INTEGER, NEIBPE(neib), 0,&
&          MPI_COMM_WORLD, stat_sr, ierr)
  enddo

  do neib= 1, NEIBPETOT
    do k= import_index(neib-1)+1, import_index(neib)
      kk= import_item(k)
      VAL(kk)= RECVbuf(k)
    enddo
  enddo
```

受信バッファからの代入



課題S2-1 ：1D熱伝導方程式（Jacobi）

s2-1a.f （7/7）

```
        do i= 1, N
          RESID  = RHS(i) - AL(i)*PHI0(i-1) - AR(i)*PHI0(i+1)
          dPHI   = RESID*rAD(i) - PHI0(i)
          dPHImax= dmax1 (dabs(dPHI), dPHImax)
          PHI(i) = PHI0(i) + dPHI
        enddo

        do i= 1, N
          PHI0(i)= PHI(i)
        enddo

        call MPI_WAITALL (NEIBPETOT, request_send, stat_send, ierr)
        call MPI_allREDUCE (dPHImax, dPHImaxG, 1, MPI_DOUBLE_PRECISION, &
     &                      MPI_MAX, MPI_COMM_WORLD, ierr)


        if (dPHImaxG.lt.EPS) exit
      enddo
      ENDtime= MPI_WTIME()

!C===
      call MPI_FINALIZE (ierr)
      end program JACOBI_1
```

各領域でJacobi法の計算（オリジナルと全く同じ）

課題S2-1：1D熱伝導方程式（Jacobi）

s2-1a.f（7/7）

```
        do i= 1, N
          RESID  = RHS(i) - AL(i)*PHI0(i-1) - AR(i)*PHI0(i+1)
          dPHI   = RESID*rAD(i) - PHI0(i)
          dPHImax= dmax1 (dabs(dPHI), dPHImax)
          PHI(i) = PHI0(i) + dPHI
        enddo

        do i= 1, N
          PHI0(i)= PHI(i)
        enddo

        call MPI_WAITALL (NEIBPETOT, request_send, stat_send, ierr)
        call MPI_allREDUCE (dPHImax, dPHImaxG, 1, MPI_DOUBLE_PRECISION, &
     &                      MPI_MAX, MPI_COMM_WORLD, ierr)

        if (dPHImaxG.lt.EPS) exit
      enddo
      ENDtime= MPI_WTIME()

!C===
      call MPI_FINALIZE (ierr)
      end program JACOBI_1
```

残差の最大値を求める
（MPI_Allreduce使用）

Jacobi法：オリジナル版と並列版

- 本体部分は全く同じ
- dPHImaxG
  - 各PEでdPHImaxを書き出してみよ

```
do i= 1, N
  RESID  = RHS(i) - AL(i)*PHI0(i-1) - AR(i)*PHI0(i+1)
  dPHI   = RESID*rAD(i) - PHI0(i)
  dPHImax= dmax1 (dabs(dPHI), dPHImax)
  PHI(i) = PHI0(i) + dPHI
enddo

do i= 1, N
  PHI0(i)= PHI(i)
enddo

if (dPHImax.lt.EPS) exit
```

```
do i= 1, N
  RESID  = RHS(i) - AL(i)*PHI0(i-1) - AR(i)*PHI0(i+1)
  dPHI   = RESID*rAD(i) - PHI0(i)
  dPHImax= dmax1 (dabs(dPHI), dPHImax)
  PHI(i) = PHI0(i) + dPHI
enddo

do i= 1, N
  PHI0(i)= PHI(i)
enddo

call MPI_WAITALL (NEIBPETOT, request_send, stat_send, ierr)
call MPI_allREDUCE (dPHImax, dPHImaxG, 1, MPI_DOUBLE_PRECISION, &
     MPI_MAX, MPI_COMM_WORLD, ierr)

if (dPHImaxG.lt.EPS) exit
```

課題S2-1：1D熱伝導方程式（Jacobi）

s2-1b.f

- 基本的に「s2-1a.f」と同じ。
- 「N」の値を各PEからファイル入力する。
  - PE数を変更するとそのたびにファイルを準備しなければならないので，本ケースの場合は不便
  - 二次元，三次元ではこのような方法をとる場合が多い

s2-1a.f

```
!C
!C-- LOCAL MESH size
      N = Ng / PETOT
      nr= Ng - N*PETOT
      if (my_rank+1.le.nr) N= N + 1
```

s2-1b.f

```
!C
!C-- LOCAL MESH size
      if (my_rank.eq.0) filename= '1d.0'
      if (my_rank.eq.1) filename= '1d.1'
      if (my_rank.eq.2) filename= '1d.2'
      if (my_rank.eq.3) filename= '1d.3'

      open (21, file= filename, status= 'unknown')
        read (21,*) N
        write (*,*) my_rank, N
      close (21)
```

実行例

input.dat

```
  103
  1.d0 1.d0
  50000
  1.d-07 1.50
```

```
$ mpif90 -O s2-1b.f
$ <modify go.sh>
$ qsub go.sh
```

PE数は4の場合のみ
他のPE数でも実行できるように
するにはどうすればよいか？

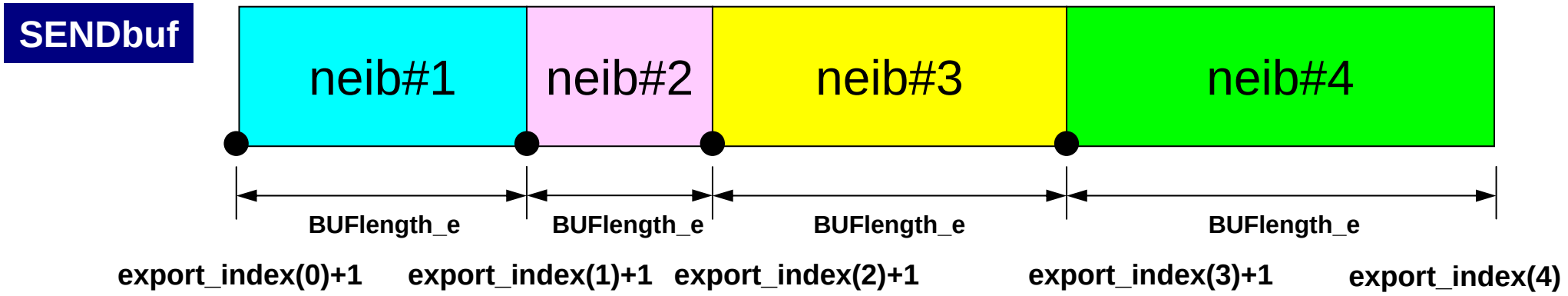
課題S2-1：1D熱伝導方程式（GS）

s2-2b.f（1/2）

```
!C
!C-- INIT. arrays for MPI_WAITALL
      allocate (stat1(MPI_STATUS_SIZE))
!C===

!C
!C +------------+
!C | ITERATIONs |
!C +------------+
!C===
      STARTtime= MPI_WTIME()
      do iter= 1, ITERmax
        dPHImax= -1.d0
!C
!C-- PREPARE send  buffer
        do neib= 1, NEIBPETOT
          do k= export_index(neib-1)+1, export_index(neib)
            kk= export_item(k)
            SENDbuf(k)= PHI(kk)
          enddo
        enddo

!C
!C-- SEND & RECV.
        do neib= 1,  NEIBPETOT
          is   = export_index(neib-1) + 1
          ir   = import_index(neib-1) + 1
          len_s= export_index(neib) - export_index(neib-1)
          len_r= import_index(neib) - import_index(neib-1)
          call MPI_SENDRECV                                          &
     &         (SENDbuf(is), len_s, MPI_DOUBLE_PRECISION,NEIBPE(neib),0,&
     &          RECVbuf(ir), len_r, MPI_DOUBLE_PRECISION,NEIBPE(neib),0,&
     &          MPI_COMM_WORLD, stat1, ierr)
        enddo
```

JacobiではPHI0を送っていたが，GSではPHIを送る。

送信（MPI_Sendrecv）



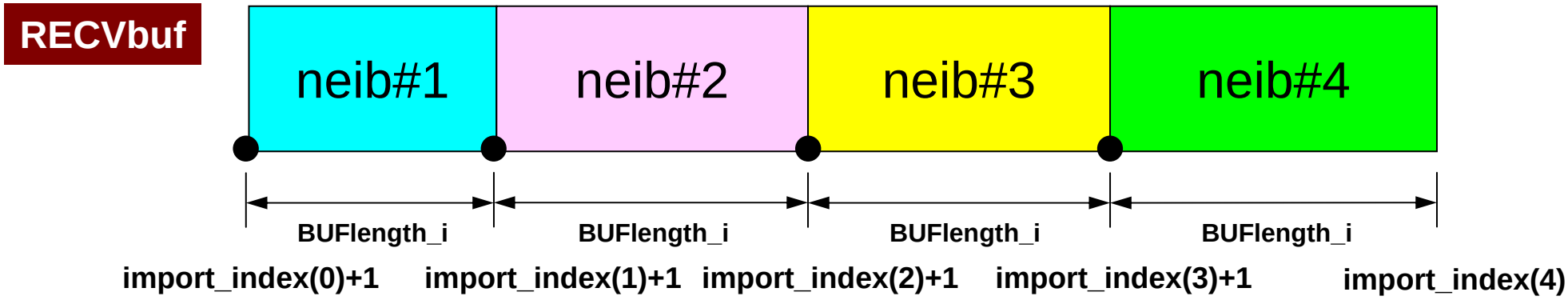

```
 do neib= 1, NEIBPETOT
   do k= export_index(neib-1)+1, export_index(neib)
     kk= export_item(k)
     SENDbuf(k)= VAL(kk)
   enddo
 enddo

 do neib= 1, NEIBPETOT
   iS_e= export_index(neib-1) + 1
   iE_e= export_index(neib  )
   BUFlength_i= iE_e + 1 - iS_e

   call MPI_SENDRECV                                                  &
&         (SENDbuf(iS_e), BUFlength_e, MPI_INTEGER, NEIBPE(neib), 0,&
&          RECVbuf(iS_i), BUFlength_i, MPI_INTEGER, NEIBPE(neib), 0,&
&          MPI_COMM_WORLD, stat_sr, ierr)
  enddo
```

送信バッファへの代入

受信（MPI_Sendrecv）

```
 do neib= 1, NEIBPETOT
   iS_i= import_index(neib-1) + 1
   iE_i= import_index(neib  )
   BUFlength_i= iE_i + 1 - iS_i

   call MPI_SENDRECV                                                  &
&         (SENDbuf(iS_e), BUFlength_e, MPI_INTEGER, NEIBPE(neib), 0,&
&          RECVbuf(iS_i), BUFlength_i, MPI_INTEGER, NEIBPE(neib), 0,&
&          MPI_COMM_WORLD, stat_sr, ierr)
  enddo

  do neib= 1, NEIBPETOT
    do k= import_index(neib-1)+1, import_index(neib)
      kk= import_item(k)
      VAL(kk)= RECVbuf(k)
    enddo
  enddo
```

受信バッファからの代入



課題S2-1：1D熱伝導方程式（GS）

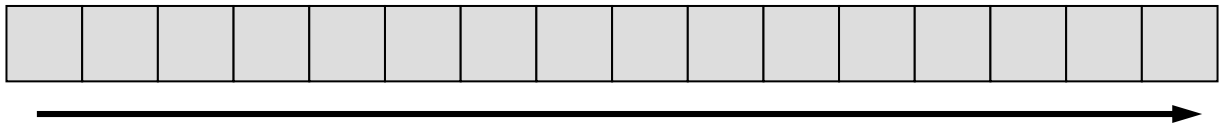
s2-2b.f（2/2）

```
!C
!C- UPDATE

        do neib= 1, NEIBPETOT
          do k= import_index(neib-1)+1, import_index(neib)
            kk= import_item(k)
            PHI(kk)= RECVbuf(k)
          enddo
        enddo

        do i=  1, N
          RESID  = RHS(i) - AL(i)*PHI(i-1) - AR(i)*PHI(i+1)
          dPHI   = RESID*rAD(i)-PHI(i)
          dPHImax= dmax1 (dabs(dPHI), dPHImax)
          PHI(i) = PHI(i) + dPHI
        enddo

        call MPI_allREDUCE (dPHImax, dPHImaxG, 1, MPI_DOUBLE_PRECISION, &
     &                      MPI_MAX, MPI_COMM_WORLD, ierr)


        if (dPHImaxG.lt.EPS) exit
      enddo
      ENDtime= MPI_WTIME()

!C===
      call MPI_FINALIZE (ierr)
      end program GS_2
```

GS,SORを並列化する際の注意

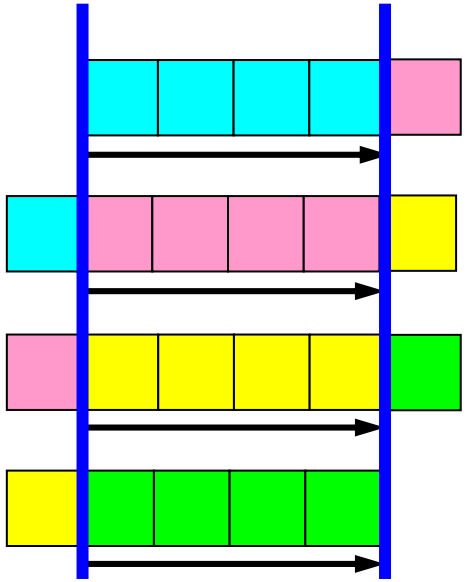
- 反復回数の増加
  - 逐次の場合と比較して，外点において一回前の反復の値を使用するため。
    - 通信してから計算するため，これは仕方がない。

逐次計算



常に最新の値を使用する

並列計算



外点の値は一回前の値

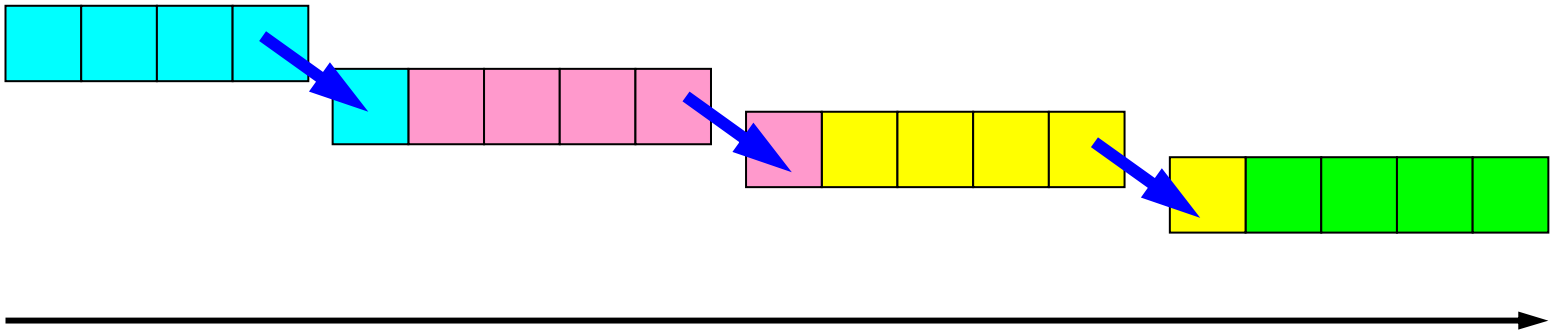
逐次並列版

- 反復回数を変化させない
  - データは分散しているが，計算は逐次的に実施する。
  - 隣のPEが計算を終わるまで待っている。
  - 当然並列化効率は悪い（実は並列版でも遅いが）。

逐次計算

常に最新の値を使用する

逐次並列計算

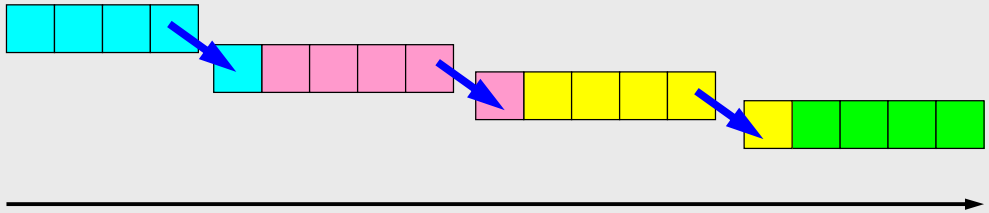


常に最新の値を使用する（隣のPEが終わるまで待つ）

課題S2-1：一次元熱伝導方程式（GS）

逐次並列版：s2-2c.f（1/2）

```
!C
!C +------------+
!C | ITERATIONs |
!C +------------+
!C===
      STARTtime= MPI_WTIME()
      do iter= 1, ITERmax
        dPHImax= -1.d0

      do ip= 1, PETOT
        if (my_rank.eq.ip-1) then
          do i=  1, N
            RESID  = RHS(i) - AL(i)*PHI(i-1) - AR(i)*PHI(i+1)
            dPHI   = RESID*rAD(i)-PHI(i)
            dPHImax= dmax1 (dabs(dPHI), dPHImax)
            PHI(i) = PHI(i) + dPHI
          enddo
        endif
!C
!C- PREPARE SENDbuf
        do neib= 1, NEIBPETOT
          do k= export_index(neib-1)+1, export_index(neib)
            kk= export_item(k)
            SENDbuf(k)= PHI(kk)
          enddo
        enddo
!C
!C-- SEND.& RECV.
        do neib= 1,  NEIBPETOT
          is= export_index(neib-1) + 1; len_s= export_index(neib) - export_index(neib-1)
          ir= import_index(neib-1) + 1; len_r= import_index(neib) - import_index(neib-1)
          call MPI_SENDRECV                                          &
     &         (SENDbuf(is), len_s, MPI_DOUBLE_PRECISION,NEIBPE(neib),0,&
     &          RECVbuf(ir), len_r, MPI_DOUBLE_PRECISION,NEIBPE(neib),0,&
     &          MPI_COMM_WORLD, stat1, ierr)
        enddo
```



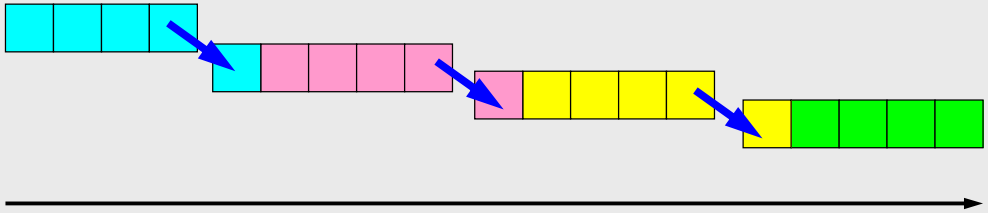
常に最新の値を使用する（隣のPEが終わるまで待つ）

PE数だけループをまわす。
ループのカウンタが（my_rank+1）
のときだけ計算を実施する
（自分の順番が回ってくるまで待つ）。

課題S2-1：一次元熱伝導方程式（GS）

逐次並列版：s2-2c.f（1/2）

```
!C
!C +------------+
!C | ITERATIONs |
!C +------------+
!C===
      STARTtime= MPI_WTIME()
      do iter= 1, ITERmax
        dPHImax= -1.d0

      do ip= 1, PETOT
        if (my_rank.eq.ip-1) then
          do i=  1, N
            RESID  = RHS(i) - AL(i)*PHI(i-1) - AR(i)*PHI(i+1)
            dPHI   = RESID*rAD(i)-PHI(i)
            dPHImax= dmax1 (dabs(dPHI), dPHImax)
            PHI(i) = PHI(i) + dPHI
          enddo
        endif

!C
!C- PREPARE SENDbuf
        do neib= 1, NEIBPETOT
          do k= export_index(neib-1)+1, export_index(neib)
            kk= export_item(k)
            SENDbuf(k)= PHI(kk)
          enddo
        enddo
!C
!C-- SEND.& RECV.
        do neib= 1,  NEIBPETOT
          is= export_index(neib-1) + 1; len_s= export_index(neib) - export_index(neib-1)
          ir= import_index(neib-1) + 1; len_r= import_index(neib) - import_index(neib-1)
          call MPI_SENDRECV                                              &
     &         (SENDbuf(is), len_s, MPI_DOUBLE_PRECISION,NEIBPE(neib),0,&
     &          RECVbuf(ir), len_r, MPI_DOUBLE_PRECISION,NEIBPE(neib),0,&
     &          MPI_COMM_WORLD, stat1, ierr)
        enddo
```



常に最新の値を使用する（隣のPEが終わるまで待つ）

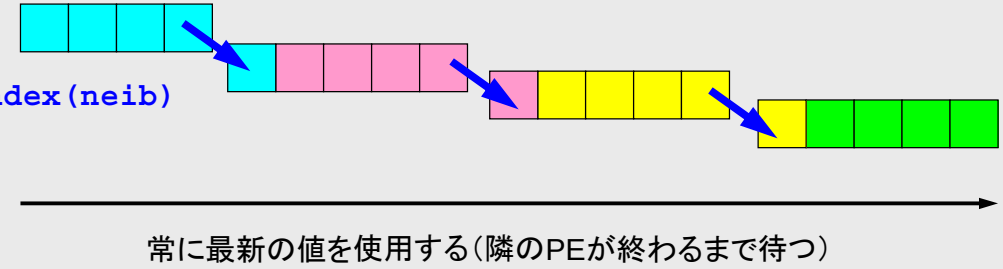
最新の計算結果を隣のPEに送受信する（ここでは全PEで送受信を実施している）。

課題S2-1：一次元熱伝導方程式（GS）

逐次並列版：s2-2c.f（2/2）

```
!C
!C-- UPDATE
      do neib= 1, NEIBPETOT
          do k= import_index(neib-1)+1, import_index(neib)
            kk= import_item(k)
            PHI(kk)= RECVbuf(k)
          enddo
        enddo
      enddo

      enddo
```

これをPE数分だけ繰り返す




```
        call MPI_allREDUCE (dPHImax, dPHImaxG, 1, MPI_DOUBLE_PRECISION, &
     &                      MPI_MAX, MPI_COMM_WORLD, ierr)

        if (my_rank.eq.PETOT-1.and.mod(iter,1000).eq.0) then
          write (*,'(i8, a, 1pe16.6, a, 1pe16.6)')                    &
     &           iter, ' iters, RESID=', dPHImaxG, ' PHI(N)= ', PHI(N)
        endif

        if (dPHImaxG.lt.EPS) exit
      enddo
      ENDtime= MPI_WTIME()
!C===

      call MPI_FINALIZE (ierr)
      end program GS_3
```

実行例

input.dat

```
  103
  1.d0 1.d0
  50000
  1.d-07 1.50
```

```
$ mpif90 -O s2-1a.f
$ <modify go.sh>
$ qsub go.sh
```

```
$ mpif90 -O s2-2a.f
$ <modify go.sh>
$ qsub go.sh
```

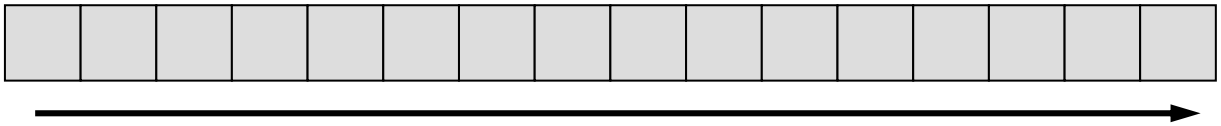
```
$ mpif90 -O s2-2c.f
$ <modify go.sh>
$ qsub go.sh
```

PE数を変えてそれぞれ計算してみよ

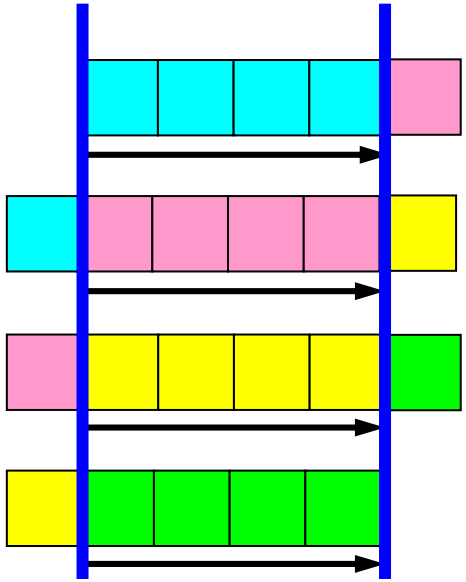
SORを並列化する際の注意

- NG/m（m：PE数）の問題をm個解いているのと同じため，SORでは，ωについてNGのときの値を使用すると発散することがある。

逐次計算



並列計算



その他

- 今回解説したものは，計算結果が各プロセッサに分散したままである。

- MPI_Gatherv/Allgathervなどを使って，「全体ベクトル」を作るようにしてみよ⇒各自自習課題
  - 課題S1を参考にする。