KOBE HPC サマースクール 2018（初級）

11. 熱伝導問題の並列計算

（関数 MPI_Sendrecv）

演習11-1 【準備】 関数 MPI_Sendrecv

- プログラム sr.c は，２つのプロセスで互いにデータを送りあうプログラムである.
  - プログラムをコピー，実行し，動作を確かめる.

```
$ cp  /tmp/Summer/M-4/sr.c  ./
$ icc sr.c -lmpi
$ qsub ...   ← ２プロセスのMPIのバッチジョブ

$ cat xxxxx.onnnnnn   ← バッチジョブの実行結果を確認
Before exchange... Rank: 0, a0= 1.0, a1= 0.0
Before exchange... Rank: 1, a0= 0.0, a1= 2.0
After exchange...  Rank: 1, a0= 1.0, a1= 2.0
After exchange...  Rank: 0, a0= 1.0, a1= 2.0
```

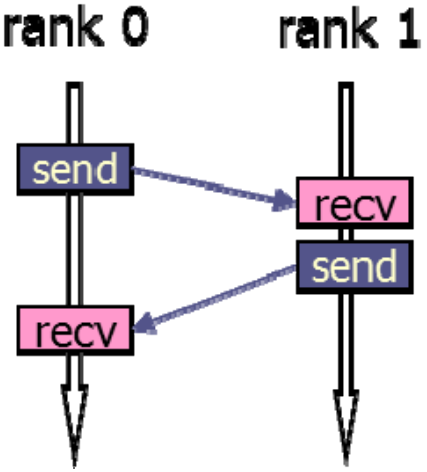
プログラム sr.c の説明

```
#include <mpi.h>                          include行
int main( int argc, char **argv )
{
   double a0, a1;
   int nprocs, myrank;
   MPI_Status status;
【省略】
   if( myrank == 0 ) {
      a0 = 1.0 ;      ←――――――――――――rank 0では, a0=1.0, a1=0.0
      a1 = 0.0 ;
   } else {
      a0 = 0.0 ;      ←――――――――――――rank 1では, a0=0.0, a1=2.0
      a1 = 2.0 ;
   };

【省略】

   if( myrank == 0 ) {
      MPI_Send( &a0, 1, MPI_DOUBLE, 1, 100, MPI_COMM_WORLD );
      MPI_Recv( &a1, 1, MPI_DOUBLE, 1, 200, MPI_COMM_WORLD, &status );
   } else {
      MPI_Recv( &a0, 1, MPI_DOUBLE, 0, 100, MPI_COMM_WORLD, &status );
      MPI_Send( &a1, 1, MPI_DOUBLE, 0, 200, MPI_COMM_WORLD );
   };

【省略】
}
```



※ 交換部分：各プロセスでのsend, recvの実行順序に注意

a0の値を rank 0からrank 1へ送る.
rank 1 から a1の値を受け取る.

rank 0 から a0の値を受け取る.
a1の値を rank 1からrank 0へ送る.

※キーワード：ブロッキング関数

双方向通信：MPI_Sendrecv関数

```
MPI_Sendrecv( void *sendbuf, int sendcount, MPI_Datatype sendtype, int dest,
              int sendtag,
              void *recvbuf, int recvcount, MPI_Datatype recvtype, int source,
              int recvtag,
              MPI_Comm comm, MPI_Status *status )
```

- sendbuf:　送信するデータのための変数名（先頭アドレス）
- sendcount:　送信するデータの数
- sendtype:　送信するデータの型
  - MPI_INTEGER, MPI_REAL8, MPI_CHARACTER など
- dest:　送信する相手のプロセス番号（destination）
- sendtag:　メッセージ識別番号．送られて来たデータを区別するための番号
- recvbuf:　受信するデータのための変数名（先頭アドレス）
- recvcount:　受信するデータの数（整数型）
- recvtype:　受信するデータの型
- source:　送信してくる相手のプロセス番号
- recvtag:　メッセージ識別番号．送られて来たデータを区別するための番号
- comm:　コミュニケータ（例えば，MPI_COMM_WORLD）
- status:　受信の状態を格納する変数

演習11-2 ：プログラムsr.c の書き換え

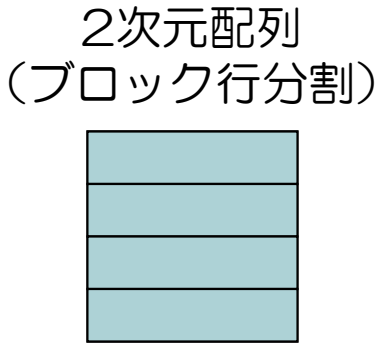
- プログラム sr.c のデータの交換部分を MPI_Sendrecv関数で書き換えよ.

```
if( myrank == 0 ) {
   MPI_Send( &a0, 1, MPI_DOUBLE, 1, 100, MPI_COMM_WORLD );
   MPI_Recv( &a1, 1, MPI_DOUBLE, 1, 200, MPI_COMM_WORLD, &status );
} else {
   MPI_Recv( &a0, 1, MPI_DOUBLE, 0, 100, MPI_COMM_WORLD, &status );
   MPI_Send( &a1, 1, MPI_DOUBLE, 0, 200, MPI_COMM_WORLD );
};
```

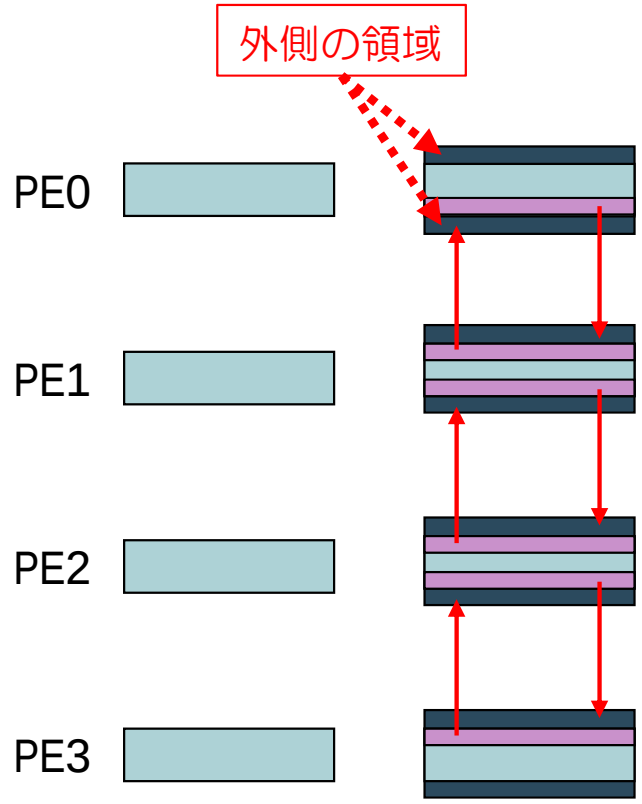
```
MPI_Sendrecv( . . . );   rank 0 から rank 1
MPI_Sendrecv( . . . );   rank 1 から rank 0
```

【発展】MPI_Sendrecvの行列への適用

- 2次元配列（u[N+2][N+2] ）をブロック行分割する.



- このとき，自分の上下の１行の要素を，隣接するプロセスの持つ領域の外側に受信用の領域を確保し，その領域に各プロセスが転送するプログラムを作る.

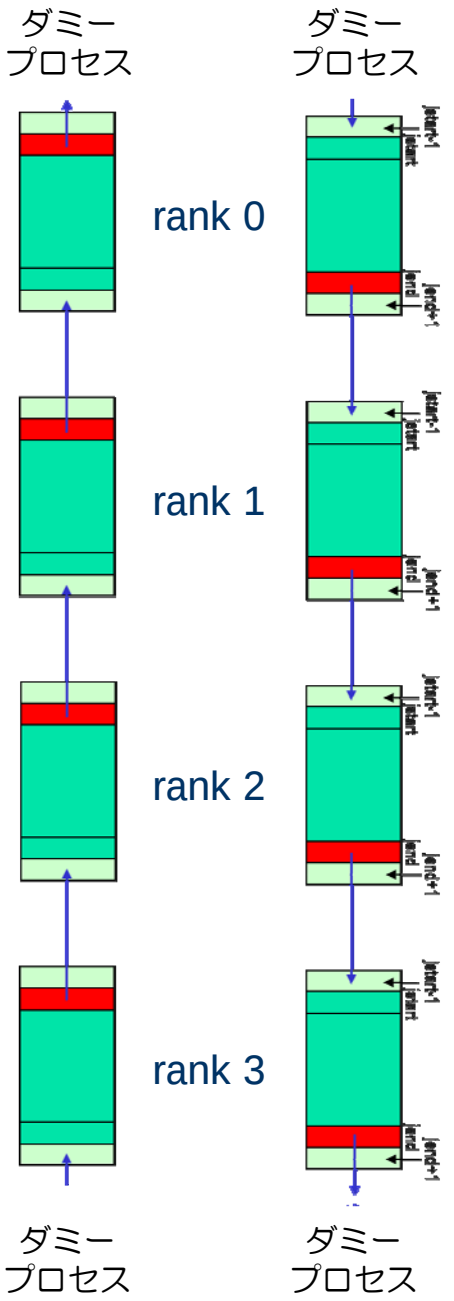


上下のプロセスから1行を受信
（受信用の領域を確保しておく）

MPI_Sendrecvの発展（続き）

- 自分の担当範囲を計算
  - 変数myrankを用いて，担当範囲 js ～ je 行を計算
  - 受信領域を考慮し，js-1，je+1 行の領域を扱う．
    - メモリ節約の観点では，自分の領域分のメモリがあれば良いが，c言語のポインタ，malloc関数を使ってやるひつようがあるので，本スクールではやらない．

- MPI_Sendrecv による送受信
  - まず，上側に js 行を送り，下側から送られてくるデータを (je+1) 行で受信
  - 次に，下側に je 行を送り，上側から送られてくるデータを (js-1) 行に受信
    - 最上下端のプロセスは，ダミープロセス（MPI_PROC_NULL） と送受信するようにする．
      - MPI_Sendrecv の source, dest に使うことが出来る．ダミープロセスを指定すると，実際にはどこからにもデータを送らないし，どこからもデータを受け取らない．



プログラム sr_matrix.c（続く）

```
#include <stdio.h>
#include <mpi.h>

#define N 20

int main( int argc, char **argv )
{
  double u[N+2][N+2];
  int  i, j, is, ie;

  int  nprocs, myrank, upper, lower, srtag;
  MPI_Status status;

  MPI_Init( &argc, &argv );
  MPI_Comm_size( MPI_COMM_WORLD, &nprocs );
  MPI_Comm_rank( MPI_COMM_WORLD, &myrank );


  is = (N/nprocs)* myrank + 1;
  ie = (N/nprocs)*(myrank+1);

  upper = myrank-1;
  if( myrank == 0        ) upper = MPI_PROC_NULL;

  lower = myrank+1 ;
  if( myrank == nprocs-1 ) lower = MPI_PROC_NULL;
```

N = 20 とする

(N+2)次の行列として宣言する．後で分かる．

Recvで必要な配列変数の宣言

各プロセスの担当する行の範囲を計算

上側プロセスのプロセス番号を設定
（存在しない場合は MPI_PROC_NULL とする）

下側プロセスのプロセス番号を設定

プログラム sr_matrix.c（続き）

```
  for( i=is; i<=ie; i++ ) {
     for( j=0; j<N+2; j++ ) {
        u[i][j] = (myrank+1)*10.0 ;
     };
  };

/* from the lower process to the upper one */
  MPI_Sendrecv(      );

/* from the upper process to the lower one */
  MPI_Sendrecv(      );


  printf("Rank = %d %d %d¥n", myrank, is, ie ) ;
  for( i=is-1; i<=ie+1; i++ ) {
     printf("%6.2f", u[i][N/2] ) ;
  };
  printf("¥n") ;


  MPI_Finalize();
  return 0;

}
```

/* 下側のプロセスから上側へ */
MPI_Sendrecv による送受信

/* 上側のプロセスから下側へ */
MPI_Sendrecv による送受信

N/2 の列のみプリントして確認している.

演習11-3 ：プログラムの完成と実行

- sr_matrix.c は未完成である.
- MPI_Sendrecvの部分を完成させ，コンパイルして，2，4 プロセスで実行し，データの送受信が正しくできていることを確かめよ.

```
$  cp  /tmp/Summer/M-4/sr_matrix.c  ./
```

- 実行結果（出力順は，以下とは異なる）.

```
Rank= 0, is:ie=   1:   5
  0.00 10.00 10.00 10.00 10.00 10.00 20.00
Rank= 1, is:ie=   6:  10
 10.00 20.00 20.00 20.00 20.00 20.00 30.00
Rank= 2, is:ie=  11:  15
 20.00 30.00 30.00 30.00 30.00 30.00 40.00
Rank= 3, is:ie=  16:  20
 30.00 40.00 40.00 40.00 40.00 40.00  0.00
```

上のプロセスのie行が，下のプロセスの(is-1）行にコピー

下のプロセスのis行が，上のプロセスの(ie+1)行にコピー

されていることを確認する.

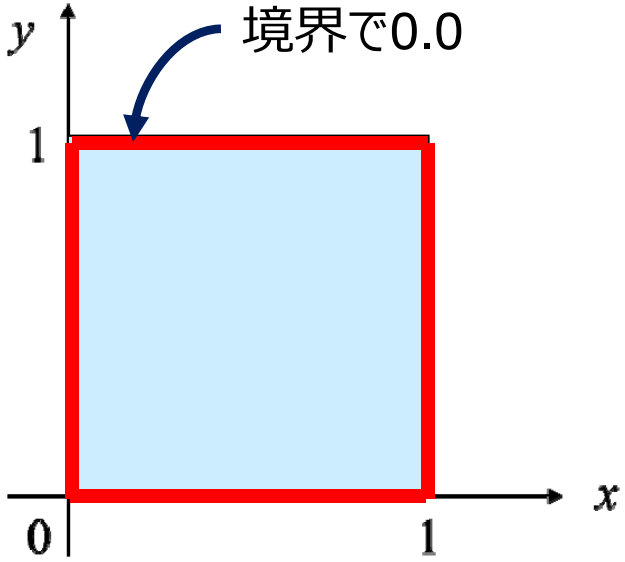
KOBE HPC サマースクール 2018（初級）

2次元定常熱伝導問題の MPIによる並列計算

2次元定常熱伝導問題

- 2次元正方形領域 [0,1]×[0,1] に一定の熱を加え続けた時の領域の温度がどうなるか？

$$-\Delta u = -\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = 40.0$$



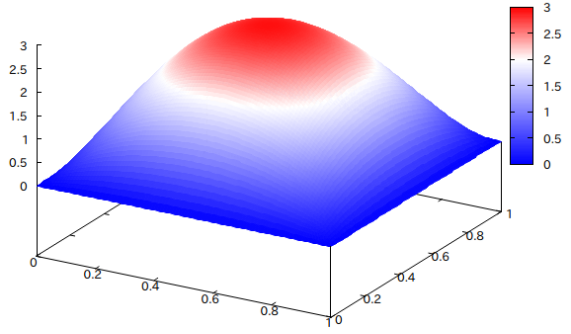
- 境界条件

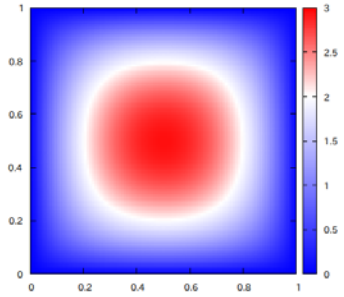
$$u(0, y) = 0.0 \quad (0 \le y \le 1)$$

$$u(1, y) = 0.0 \quad (0 \le y \le 1)$$

$$u(x, 0) = 0.0 \quad (0 \le x \le 1)$$

$$u(x, 1) = 0.0 \quad (0 \le x \le 1)$$

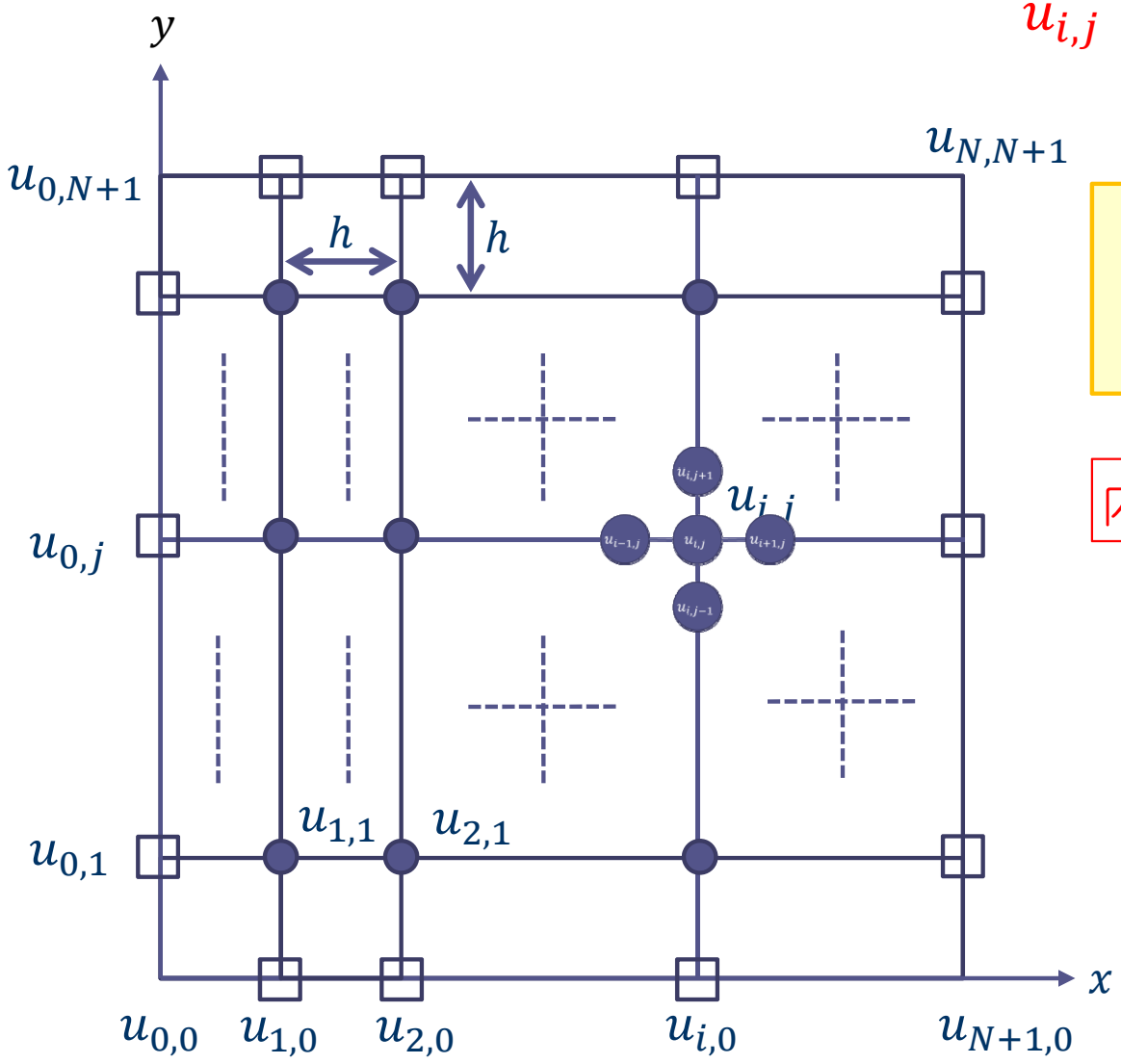




離散化格子と変数

x方向，y方向どちらも $(N+1)$ 等分する． $h = 1/(N+1)$

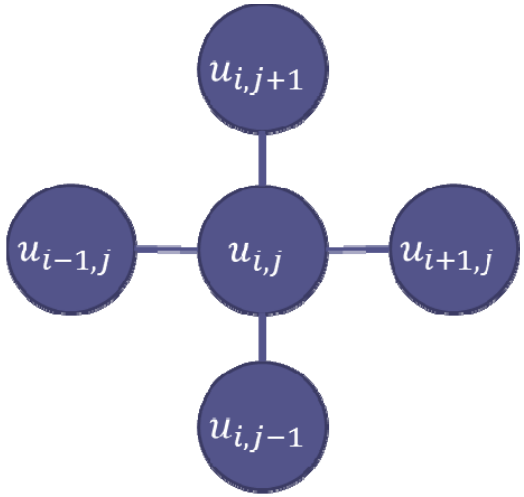
$u_{i,j} \triangleq u(ih, jh) \quad (i, j = 0, 1, 2, \ldots, N+1)$



□ 境界条件として既知の値

● 内部の格子点（未知数）

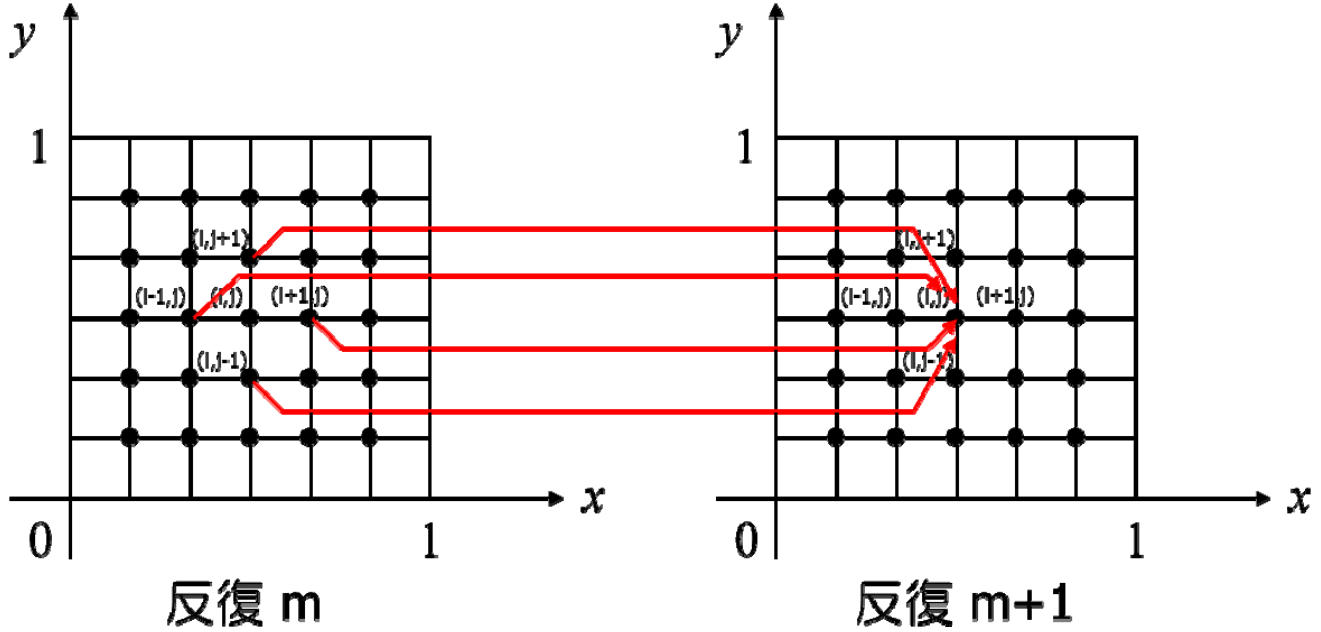
内部の格子点に関する関係式を作る．



熱伝導問題へのヤコビ法の適用

- 反復ベクトル生成のアルゴリズム

```
for( i=1; i<=N; i++ ) {
  for( j=1; j<=N; j++ ) {
    u_{i,j}^{(m+1)} = (u_{i-1,j}^{(m)}+u_{i+1,j}^{(m)}+u_{i,j-1}^{(m)}+u_{i,j+1}^{(m)})*0.25 + 10.0*dx*dx;
  };
};
```



演習11-4 ：プログラム heat2d.c の実行

- 2次元定常熱伝導問題の逐次プログラム heat2d.c をコンパイルして実行せよ.

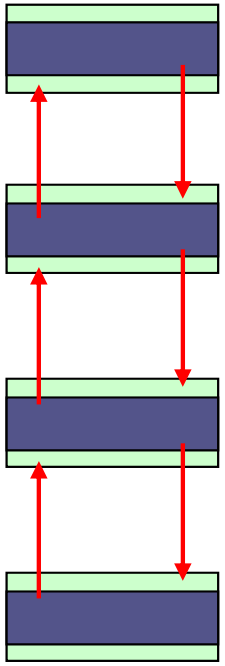
```
$ cp  /tmp/Summer/M-4/heat2d.c ./
$ icc -O3 heat2d.c
$ ./a.out
```

  - プログラムでは $\mathrm{N} = 80$ の時に，反復ベクトルの差のノルム $\left\| u_{i,j}^{\mathrm{m}+1} - u_{i,j}^{\mathrm{m}} \right\|_2$ がある閾値$\epsilon = 1.0 \times 10^{-4}$ より小さくなったら反復を停止している.

heat2d.c のMPIによる並列化

- 並列化のヒント
  - 2次元配列 u，u0 をブロック行分割
  - u の計算をする前に，u[is:ie][]をu0にコピーし，
    - 上のプロセスのu0の je行を下のプロセスの(js-1)行へ，
    - 下のプロセスのu0の js 行を上のプロセスの(je+1)行へ，

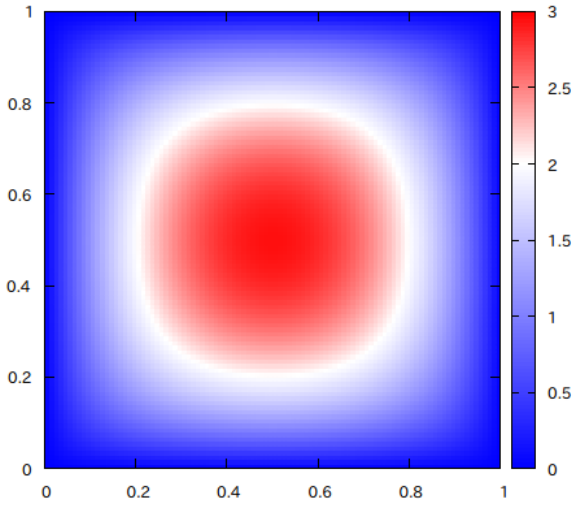
    送る.
    - 並列化sr_matrix.c と同様に，mpi_sendrecv を用いて送受信
  - u の js ～ je 行の計算を行う.
  - 反復ベクトルの差のノルムの部分和を計算し，MPI_Allgather で総和を取り，収束の判定を行うようにする.
    - ローカルで残差の部分和を計算し，各プロセスでノルムを計算する.



上下のプロセスから1行を受信
（受信用の領域を確保しておく）

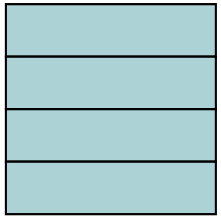
演習11-5 heat2d.cの並列化

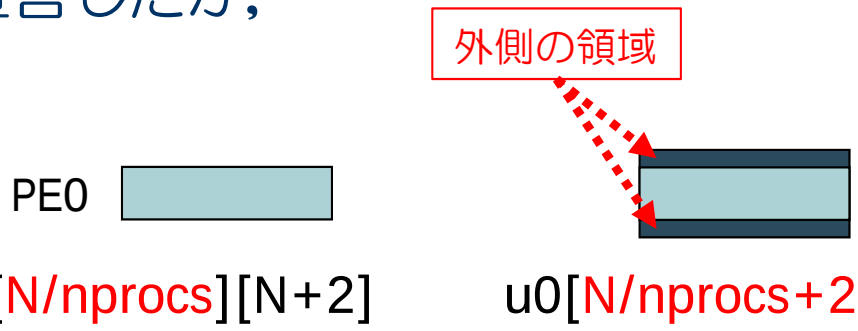
- heat2d.cをMPIを用いて並列化せよ.
- N=128とし，プロセス数1, 2, 4, 8 として，反復開始から終了までの計算時間を計測し，速度向上率を求めよ．また，それをグラフにせよ.
- 計算結果を gnuplotで図示せよ.
  - 計算結果の出力は，プロセス0にデータを集め，ファイルに出力する.



上級者になるために．．．

- 大規模問題を解く場合には，1プロセスのメモリ使用容量を少なくする必要が出てくる．
- 今回のプログラムでは，どのプロセスも u[N+2,N+2], u0[N+2,N+2] として，計算全領域の変数を宣言したが，





各プロセスは，これだけの大きさをもてば良いはず．

- 実行前から並列数（nprocs）が固定されていれば，上記のように，プログラムの先頭で小さい領域を宣言することも出来るが，汎用のプログラムとしては，nprocsを実行開始時に決めたい．
- 配列を動的に確保する必要がある（malloc関数）．
  - サンプルプログラム　/tmp/Summer/M-4/alloc_mpi_heat2d.c

補足：2次元定常熱伝導問題の離散化

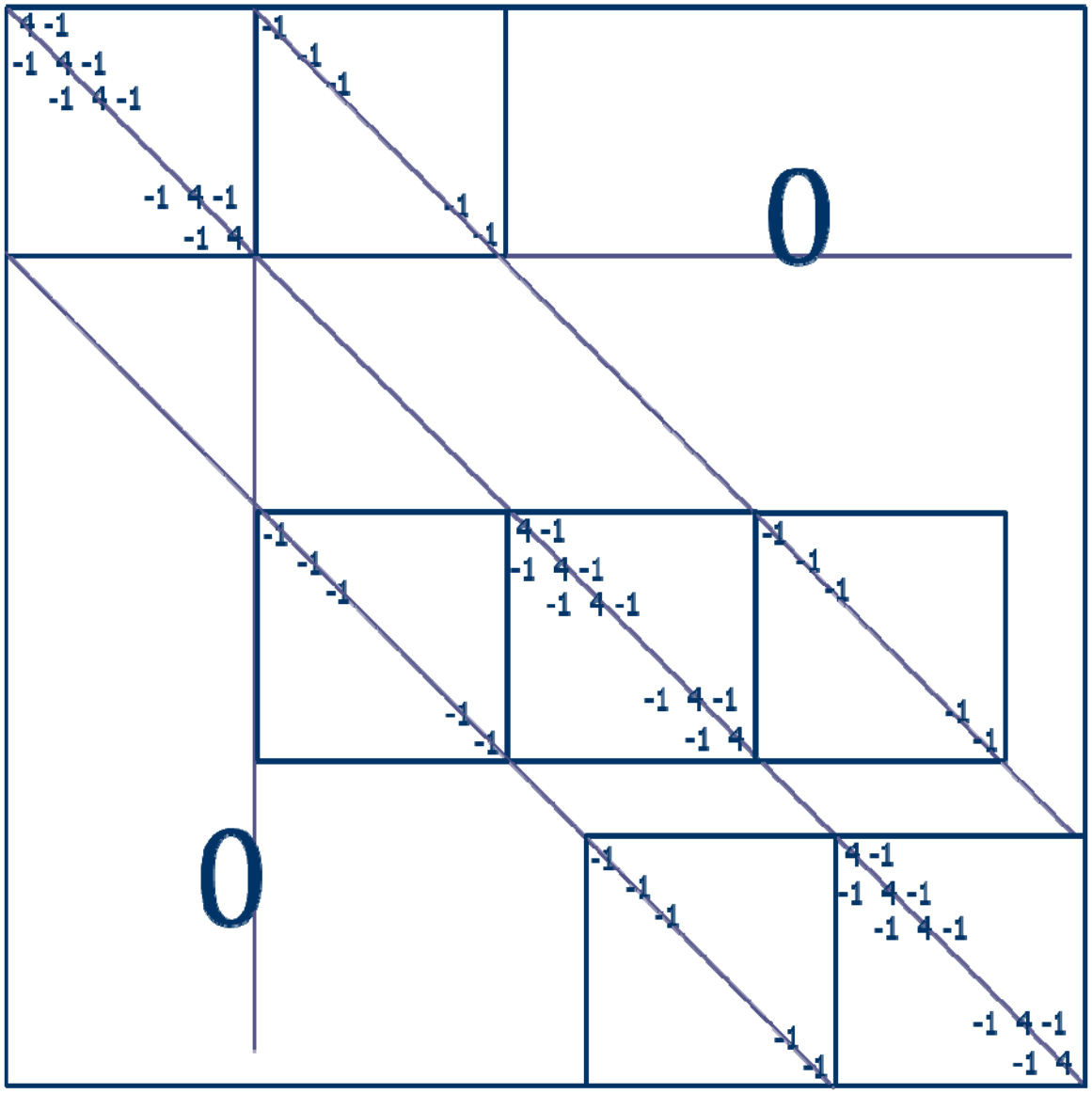
$u_{i,j}$を辞書式順序で並べたベクトルを$\boldsymbol{u}$とすると，点(i,j)に関する離散式は

$$u_{i,j-1} + u_{i-1,j} - 4u_{i,j} + u_{i+1,j} + u_{i,j+1} = -10\,h^2 \quad (i,j = 1,2,\ldots,N)$$

$$\boldsymbol{u} = \begin{bmatrix} u_{1,1} \\ u_{2,1} \\ u_{3,1} \\ \vdots \\ u_{N,1} \\ u_{1,2} \\ u_{2,2} \\ u_{3,2} \\ \vdots \\ u_{N,2} \\ \vdots \\ \vdots \\ u_{1,j} \\ \vdots \\ u_{i,j} \\ \vdots \\ u_{N,j} \\ \vdots \\ \vdots \\ u_{1,N} \\ u_{2,N} \\ u_{3,N} \\ \vdots \\ u_{N,N} \end{bmatrix}$$

$$[0 \quad \cdots \quad \underbrace{1 \quad \cdots \quad 1}_{\text{N離れている}} \quad -4 \quad \underbrace{1 \quad \cdots \quad 1}_{\text{N離れている}} \quad 0 \quad \cdots \quad 0] \begin{bmatrix} u_{1,1} \\ u_{2,1} \\ u_{3,1} \\ \vdots \\ u_{N,1} \\ u_{1,2} \\ u_{2,2} \\ u_{3,2} \\ \vdots \\ u_{N,2} \\ \vdots \\ \vdots \\ u_{1,j} \\ \vdots \\ u_{i,j} \\ \vdots \\ u_{N,j} \\ \vdots \\ \vdots \\ u_{1,N} \\ u_{2,N} \\ u_{3,N} \\ \vdots \\ u_{N,N} \end{bmatrix} = \begin{bmatrix} b_{1,1} \\ b_{2,1} \\ b_{3,1} \\ \vdots \\ b_{N,1} \\ b_{1,2} \\ b_{2,2} \\ b_{3,2} \\ \vdots \\ b_{N,2} \\ \vdots \\ \vdots \\ b_{1,j} \\ \vdots \\ b_{i,j} \\ \vdots \\ b_{N,j} \\ \vdots \\ \vdots \\ b_{1,N} \\ b_{2,N} \\ b_{3,N} \\ \vdots \\ b_{N,N} \end{bmatrix}$$

行列を用いた表現



$$\begin{bmatrix} u_{1,1} \\ u_{2,1} \\ u_{3,1} \\ \vdots \\ u_{N,1} \\ u_{1,2} \\ u_{2,2} \\ u_{3,2} \\ \vdots \\ u_{N,2} \\ \vdots \\ u_{1,j} \\ \vdots \\ u_{i,j} \\ \vdots \\ u_{N,j} \\ \vdots \\ u_{1,N} \\ u_{2,N} \\ u_{3,N} \\ \vdots \\ u_{N,N} \end{bmatrix} = \begin{bmatrix} b_{1,1} \\ b_{2,1} \\ b_{3,1} \\ \vdots \\ b_{N,1} \\ b_{1,2} \\ b_{2,2} \\ b_{3,2} \\ \vdots \\ b_{N,2} \\ \vdots \\ b_{1,j} \\ \vdots \\ b_{i,j} \\ \vdots \\ b_{N,j} \\ \vdots \\ b_{1,N} \\ b_{2,N} \\ b_{3,N} \\ \vdots \\ b_{N,N} \end{bmatrix}$$

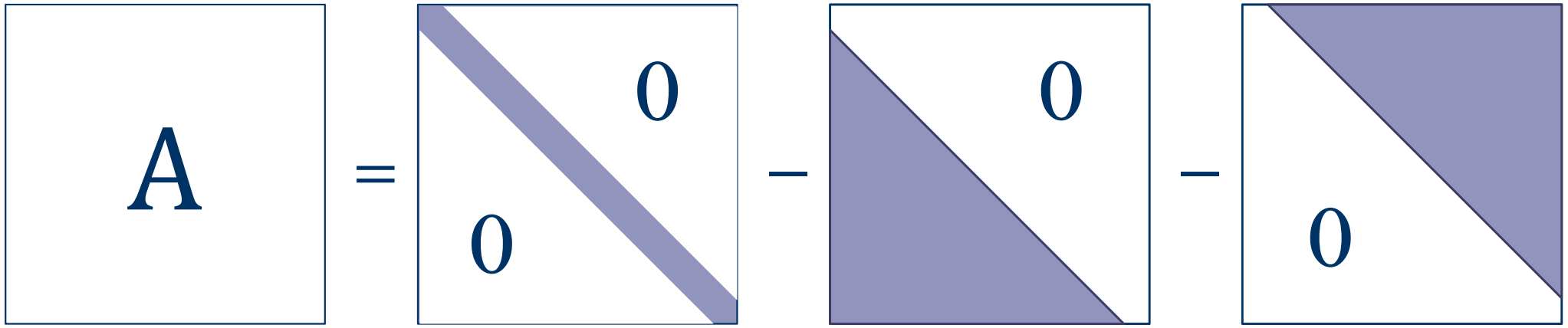
連立一次方程式の反復解法

- 反復解法
  - $x^{(0)}$を真の解x^*の近似値とし，反復ベクトルの系列 $x^{(0)} \rightarrow x^{(1)} \rightarrow x^{(2)} \rightarrow \cdots \rightarrow x^{(m)}$により，真の解に近づける方法
  - 反復ベクトルの系列の作り方によりいろいろな解法がある．

  - 定常反復法
    - ヤコビ法（Jacobi法）
    - ガウス・ザイデル法（Gauss-Seidel法）
    - 逐次過大緩和法（Successive Over-Relaxation：SOR法）
    - 交互方向法（Alternating-Direction Implicit Iterative Method: ADI法）など

  - 非定常反復法
    - 共役勾配法（Conjugate Gradient Method，CG法）
    - 双共役勾配法（Bi-CG法）
    - 一般化最小残差法（GMRES法）など

行列分離



行列分離： $\mathrm{A} = \mathrm{D} - \mathrm{E} - \mathrm{F}$

$\mathrm{D} = \mathrm{diag}(a_{11}, a_{22}, \cdots, a_{nn})$：対角行列

E：狭義下三角行列

F：狭義上三角行列

$$\mathrm{A}\boldsymbol{x} = \boldsymbol{b} \quad \Rightarrow \quad (\mathrm{D} - \mathrm{E} - \mathrm{F})\boldsymbol{x} = \boldsymbol{b}$$

ヤコビ法（その1）

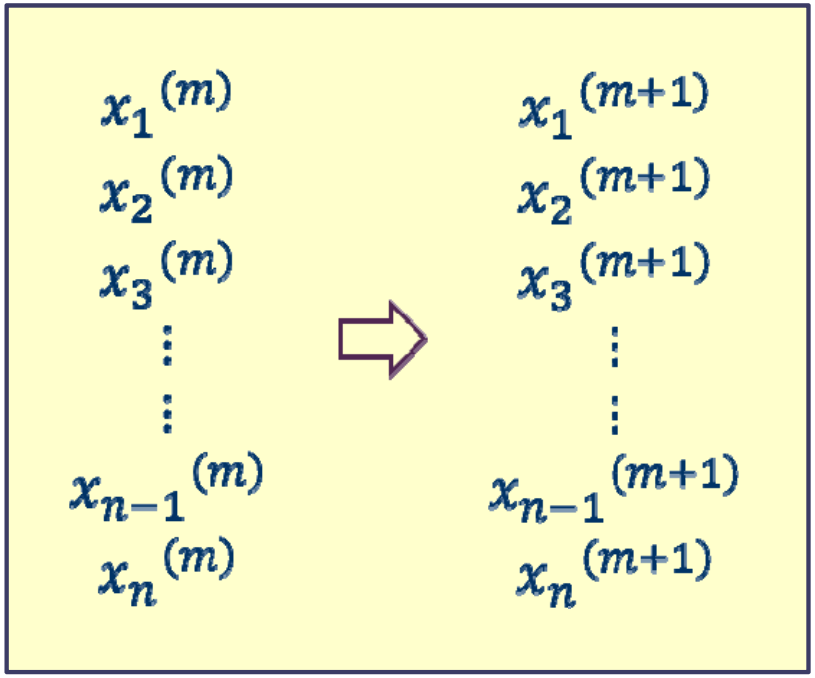
$$(\mathrm{D}-\mathrm{E}-\mathrm{F})\boldsymbol{x}=\boldsymbol{b} \quad \Rightarrow \quad \mathrm{D}\boldsymbol{x}=(\mathrm{E}+\mathrm{F})\boldsymbol{x}+\boldsymbol{b}$$

$$\boldsymbol{x}^{(m+1)} = \underbrace{\mathrm{D}^{-1}(\mathrm{E}+\mathrm{F})}_{\text{ヤコビ行列}}\boldsymbol{x}^{(m)} + \mathrm{D}^{-1}\boldsymbol{b} \quad (m \geq 0)$$

で，反復ベクトル系列を生成する.

$$x_i^{(m+1)} = \left(-\sum_{j=1, j\neq i}^{n} a_{ij} x_j^{(m)} + b_i \right) / a_{ii}$$

$$(1 \leq i \leq n, m \geq 0)$$



参考：Fortran版

双方向通信：mpi_sendrecv関数

```
mpi_sendrecv( sendbuff, sendcount, sendtype, dest, sendtag,
              recvbuff, recvcount, recvtype, source, recvtag,
              comm, status, ierr )
```

- sendbuff:　送信するデータのための変数名（先頭アドレス）
- sendcount:　送信するデータの数（整数型）
- sendtype:　送信するデータの型（MPI_REAL，MPI_INTEGERなど）
- dest:　送信する相手プロセスのランク番号
- sendtag :　メッセージ識別番号．送くるデータを区別するための番号
- recvbuff:　受信するデータのための変数名（先頭アドレス）
- recvcount:　受信するデータの数（整数型）
- recvtype:　受信するデータの型（MPI_REAL，MPI_INTEGERなど）
- source:　送信してくる相手プロセスのランク番号
- recvtag:　メッセージ識別番号．送られて来たデータを区別するための番号
- comm:　コミュニケータ（例えば，MPI_COMM_WORLD）
- status:　受信の状態を格納するサイズMPI_STATUS_SIZEの配列（整数型）
- ierr:　戻りコード（整数型）