KOBE HPC サマースクール 2026（初級）

11. 並列計算性能の評価方法

（時間計測関数，バリア同期関数，sendrecv関数）

時間計測関数，バリア同期関数

計算時間の計測

- 並列計算の目的は，計算時間の短縮にある．
  - 大規模問題を解くためにメモリ容量を増やすことも目的の一つ

- 同じ結果が得られるが，アルゴリズムや書き方が異なったプログラムのうち，どれが一番良いか？

  ⇨ （正しい結果が得られるならば）計算時間の短いものが良いはず．

- 計算時間を計測して比較する．

計算時間を計測する方法

```
double time0, time2;                    計測のための変数を倍精度実数型で宣言する.
      •
      •
MPI_Barrier( MPI_COMM_WORLD );          MPI_Barrier 関数で，計測開始の足並みを揃える.
time0 = MPI_Wtime();                    MPI_Wtime 関数で開始時刻を time0 に設定

 (計測する部分)

MPI_Barrier( MPI_COMM_WORLD );          全プロセスで終了の足並みを揃える.
time1 = MPI_Wtime();                    MPI_Wtime 関数で終了時刻を time1 に設定

 (time1-time0 を出力する)               time1-time0 が計測した部分の計算時間となる.
```

MPI_Barrier(comm)　：　バリア同期関数

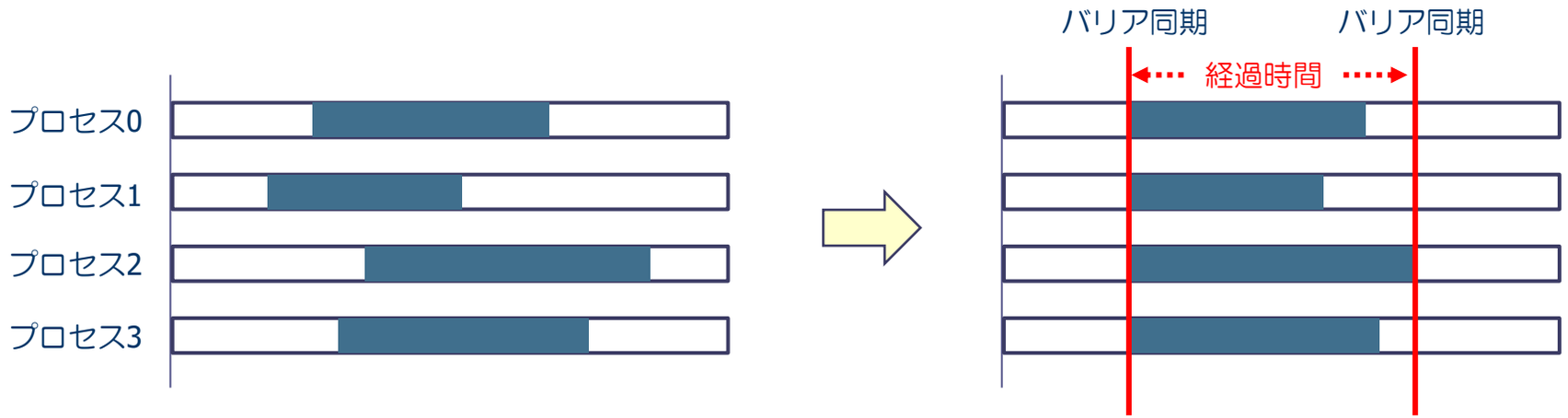
- comm:　　コミュニケータ（例えば，MPI_COMM_WORLD）

var = MPI_Wtime()　：倍精度実数を返す関数（double var;）

時間計測のイメージ

- 各プロセスでの計算時間の測定関数
  - `MPI_Wtime()`
    - ある時点を基準とした経過秒数を倍精度実数型で返す関数

- プログラムのある区間の計算時間の測定
  - プログラムの実行は各プロセスで独立なので，開始時間や終了時間が異なる．
  - ある部分の計算時間の計測では，バリア同期（`MPI_Barrier`）により測定開始と測定終了の足並みを揃えて，計測する．



演習11-1　並列化normalize.c の実行時間計測

- 並列化したベクトルの正規化プログラムの実行時間を計ってみる.
  - 計測範囲は，各プロセスが二乗和を求めるところの直前から，正規化した直後まで，とする.
- n = 50,000，100,000，500,000 に対して，並列数を1，2，4，8，16 と変化させて時間を計測せよ.
  - 1プロセスの実行時間を T(1)，mプロセスの実行時間を T(m) とするとき，速度向上率 T(1)/T(m) を求めよ（次の表参照）.
  - また，速度向上率のグラフを描け（例えば gnuplot を使う）.
    - Gnuplot は，Linux で動作するグラフ描画ソフトウェア.
    - Rokkoサーバ上では，「module load gnuplot」で環境設定後に利用可能.

表：ベクトルサイズと並列数の関係

	n=50,000		n=100,000		n=500,000	
並列数 (m)	計算時間（秒） T(m)	速度向上率 T(1)/T(m)	計算時間（秒） T(m)	速度向上率 T(1)/T(m)	計算時間（秒） T(m)	速度向上率 T(1)/T(m)
1		1.000		1.000		1.000
2						
4						
8						
16						

MPI_Sendrecv関数

演習11-2a 【準備】 関数 MPI_Sendrecv

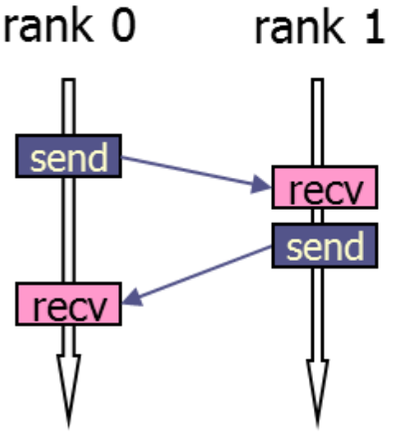
- プログラム srseparate.c は，２つのプロセスで互いにデータを送りあうプログラムである.
  - プログラムをコピー，実行し，動作を確かめる.

```
$ cp /home/guest60/share/srseparate.c  ./
$ icx srseparate.c -lmpi
$ qsub ...   ← ２プロセスの MPI のバッチジョブ（jobm.sh を適宜修正して使用）

$ 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= -99.0
Before exchange... Rank: 1, a0=   2.0,  a1= -99.0
After exchange...  Rank: 0, a0=   1.0,  a1=   2.0
After exchange...  Rank: 1, a0=   2.0,  a1=   1.0
```

プログラム srseparate.c の説明

```
#include <stdio.h>                                  include行
#include <mpi.h>
#define N 100
double a0[N], a1[N];
int main( int argc, char **argv )
{
   int nprocs, myrank, i;
【省略】
   if( myrank == 0 ) {
     for( i = 0; i < N; i++ ){
      a0[i] = 1.0;                          ←——  rank 0では，a0=1.0，a1=-99.0
      a1[i] = -99.0;}
   } else {
     for( i = 0; i < N; i++ ){
      a0[i] = 2.0;                          ←——  rank 1では，a0=2.0，a1=-99.0
      a1[i] = -99.0;}
   }
【省略】
   if( myrank == 0 ) {
      MPI_Send( &a0[0], N, MPI_DOUBLE, 1, 100, MPI_COMM_WORLD );
      MPI_Recv( &a1[0], N, MPI_DOUBLE, 1, 200, MPI_COMM_WORLD, MPI_STATUS_IGNORE );
   } else {
      MPI_Recv( &a1[0], N, MPI_DOUBLE, 0, 100, MPI_COMM_WORLD, MPI_STATUS_IGNORE );
      MPI_Send( &a0[0], N, 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

rank 0 の a0 の値を rank 1 へ送る．
rank 1 からの値を a1 で受け取る．

rank 0 からの値を a1 で受け取る．
rank 1 の a0 の値を 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_INT, MPI_DOUBLE, MPI_CHAR 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．送る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11-2b ：プログラム sendrecv.c

- プログラム srseparate.c のデータの交換部分を MPI_Sendrecv関数で書き換えた「プログラム sendrecv.c」を作成せよ.

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

```
MPI_Sendrecv( . . . );  rank 0 では,  rank 1 に送り, rank 1 から受け取る.
                        rank 1 では,  rank 0 に送り, rank 0 から受け取る.
```

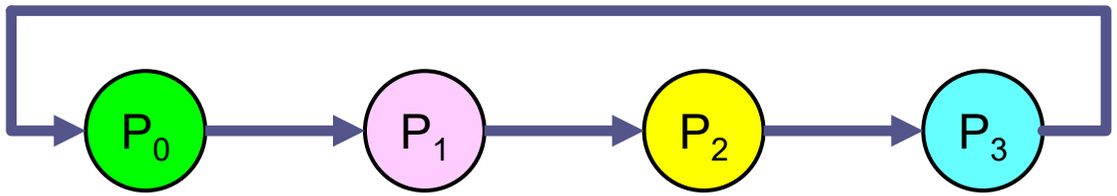
追加演習11-3

- n個の MPIプロセスが，それぞれ rank番号×10 の値を持っている.
- MPIプロセスがリング状につながっているとする.
- この時,
  rank #0 -> #1 -> #2 -> ..... -> #n-1 -> #0
  と値をぐるっとに回すプログラムを作成せよ.

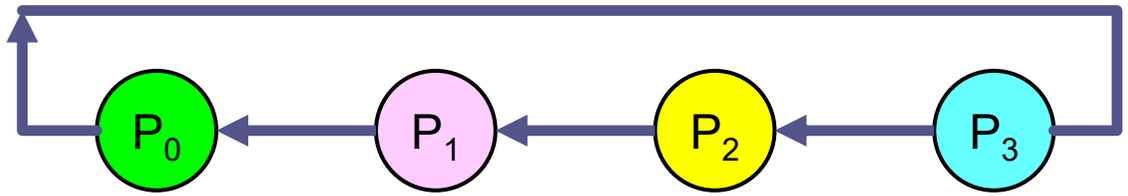
実行前：　　0, 10, 20, 30,　....., 10(n-2), 10(n-1)
実行後：10(n-1),　0, 10, 20,　....., 10(n-3), 10(n-2)

追加演習11-3（続き）

- 前ページのような通信を，リング通信と呼ぶことがある．
  - 初回は，全員が右隣に送信し，左隣から受信する．



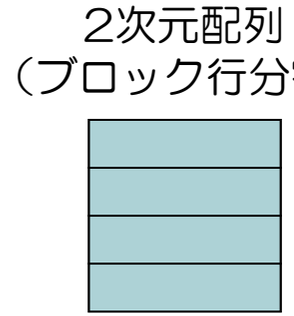
  - 2回目は，全員が左隣に送信し，右隣から受信する．



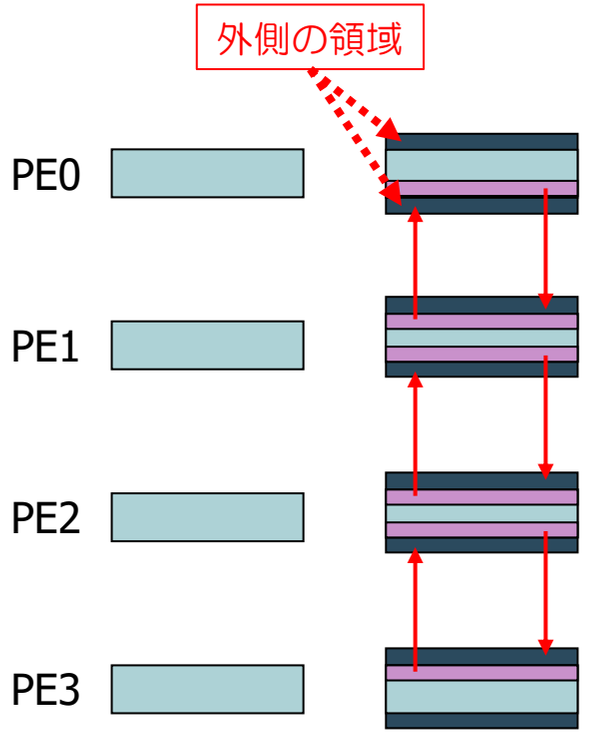
- 集団通信関数の内部実装に使用されることもある基本的な通信方式

【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.

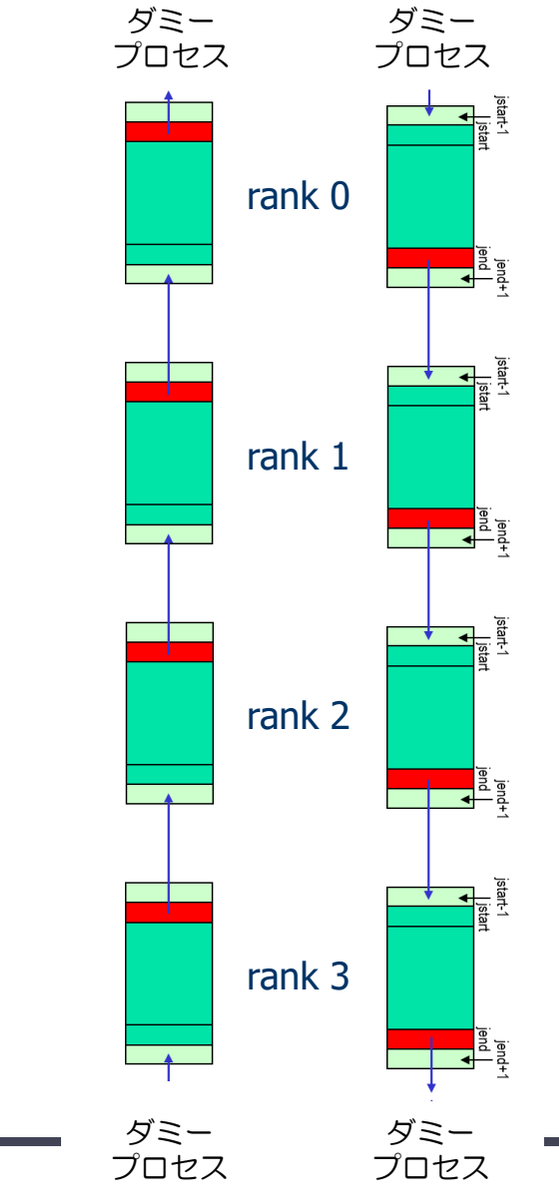


上下のプロセスから1行を受信
（受信用の領域を確保しておく）

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 is ～ ie 行を計算
  - 受信領域を考慮し，is-1，ie+1 行の領域に拡大する．
    - 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関数を使ってやる必要があるので，本スクールではやらない（上級）．

- MPI_Sendrecv による送受信
  - まず，上側に is 行を送り，下側から送られてくるデータを (ie+1) 行で受信
  - 次に，下側に ie 行を送り，上側から送られてくるデータを (is-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_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．

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 ;
     };
  };

/* upward data circulation */                          /* 下側のプロセスから上側へ */
  MPI_Sendrecv(      );                                MPI_Sendrecv による送受信

/* downward data circulation */                        /* 上側のプロセスから下側へ */
  MPI_Sendrecv(      );                                MPI_Sendrecv による送受信

  for( j=0; j<nprocs; j++) {
    if( myrank == j ) {
      printf("Rank=%2d, is:ie=%2d:%2d¥n", myrank, is, ie );
      for( i=is-1; i<=ie+1; i++ ) {
        printf("%6.2f", u[i][N/2] ) ;                  N/2 の列のみプリントして確認している.
      }
      printf("¥n");
      fflush(stdout);
    }
    MPI_Barrier( MPI_COMM_WORLD );
  }

  MPI_Finalize();
  return 0;
}
```

高速化のTips

- メモリ上のデータは連続的にアクセスする方が速い
  - 以下のコードより…

```
do i = 1, n
  do j = 1, n
    c(i,j) = a(i,j) + b(i,j)
  end do
end do
```

```
for(j=0; i<n; i++) {
  for(i=0; j<n; j++) {
    c[i][j] = a[i][j] + b[i][j];
  }
}
```

  - 以下のコードの方が高速に処理できる

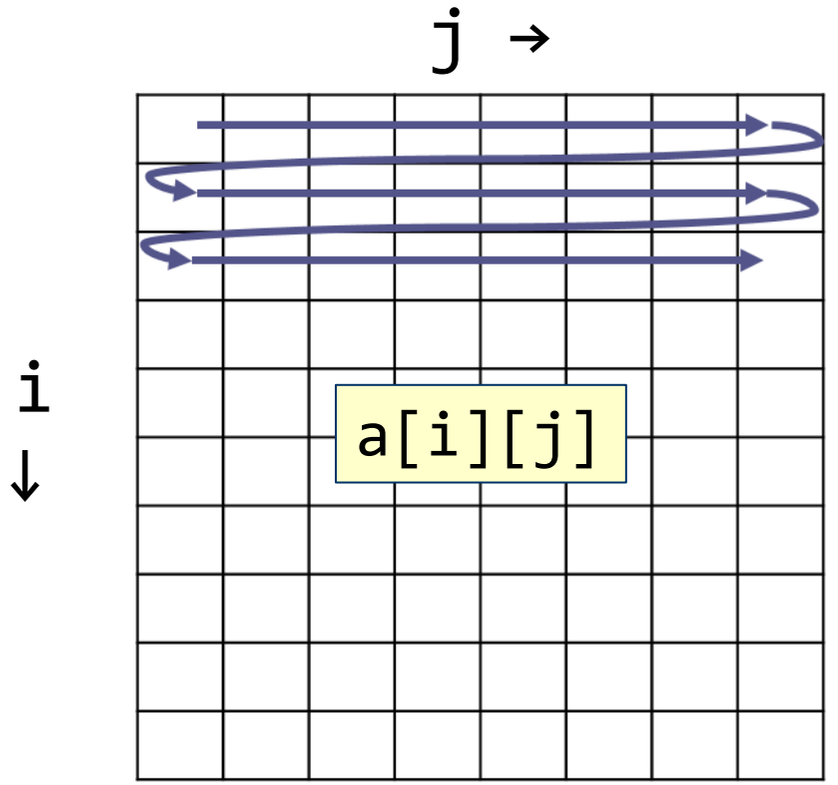
```
do j = 1, n
  do i = 1, n
    c(i,j) = a(i,j) + b(i,j)
  end do
end do
```

```
for(i=0; i<n; i++) {
  for(j=0; j<n; j++) {
    c[i][j] = a[i][j] + b[i][j];
  }
}
```

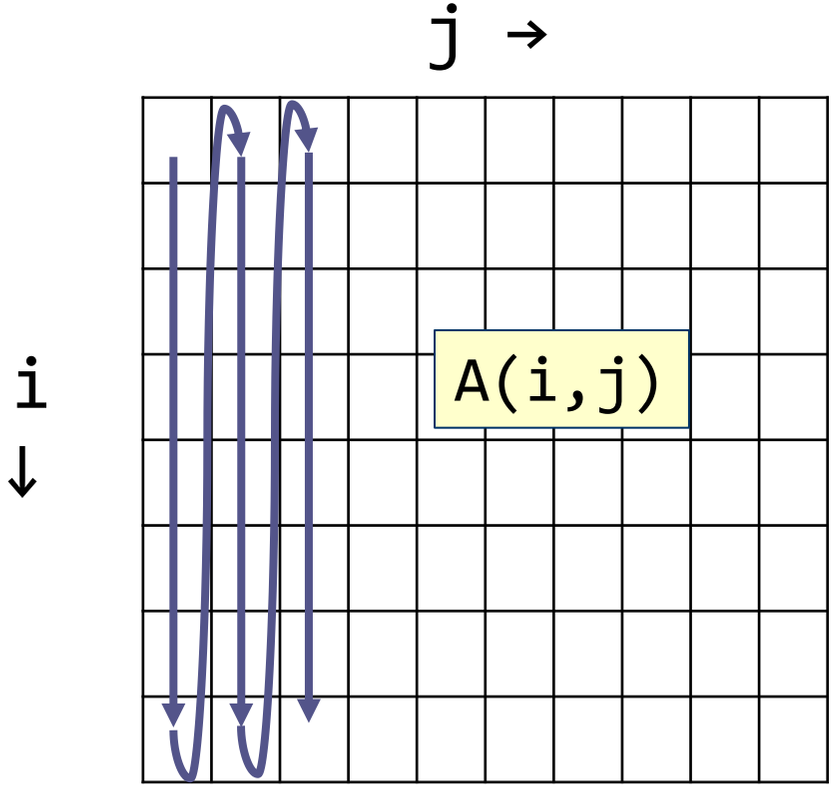
※ C言語の2次元配列のルールは「row-major order」．
※ プログラミング言語により，メモリ配置のルールは異なるので注意（例：Fortran言語では「column-major order」… Cとは逆）

行列のメモリ上の配置

Cの場合
(row-major order)



Fortranの場合
（column-major order）



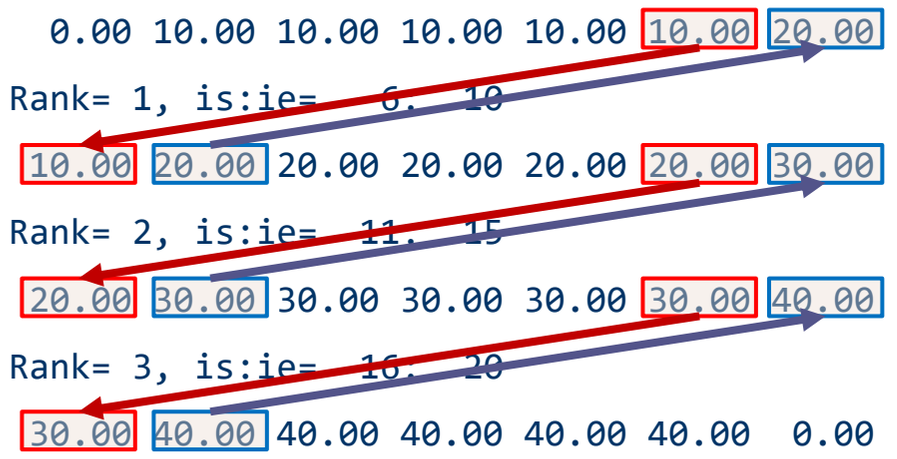
演習11-4：プログラムの完成と実行

- 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 /home/guest60/share/sr_matrix.c  ./
```

- 実行結果

```
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.

参考：Fortran版

計算時間を計測する方法

```
real(DP) :: time0, time2                          計測のための変数を倍精度実数型で宣言する.
      •
      •
call mpi_barrier( MPI_COMM_WORLD, ierr )          mpi_barrier 関数で，計測開始の足並みを揃える.
time0 = mpi_wtime()                               mpi_wtime 関数で開始時刻を time0 に設定

 (計測する部分)

call mpi_barrier( MPI_COMM_WORLD, ierr )          全プロセスで終了の足並みを揃える.
time1 = mpi_wtime()                               mpi_wtime 関数で終了時刻を time1 に設定

 (time1-time0 を出力する)                          time1-time0 が計測した部分の計算時間となる.
```

`mpi_barrier(comm, ierr)` ： バリア同期関数

- comm: コミュニケータ（例えば，MPI_COMM_WORLD）
- ierr: 戻りコード（整数型）

var = mpi_wtime() ：倍精度実数を返す関数

双方向通信：mpi_sendrecv関数

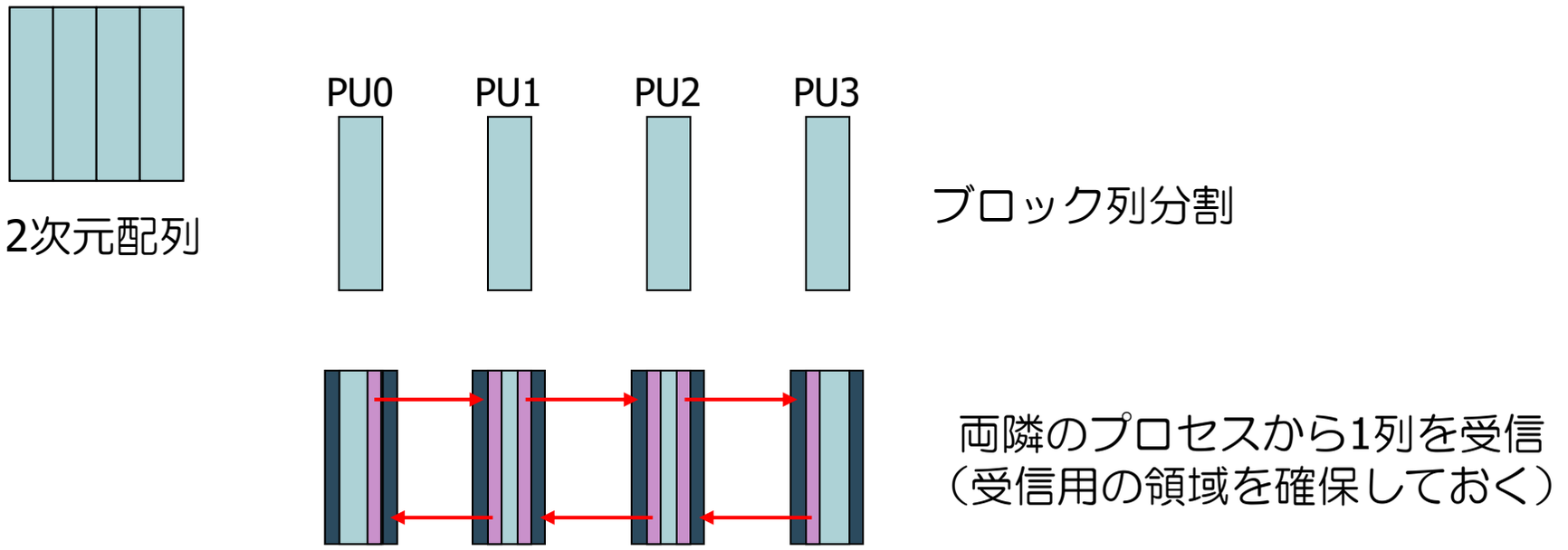
```
mpi_sendrecv( sendbuff, sendcount, sendtype, dest, sendtag,
              recvbuff, recvcount, recvtype, source, recvtag,
              comm, status, ierr )
```

- 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）

【発展】MPI_Sendrecv の行列への適用

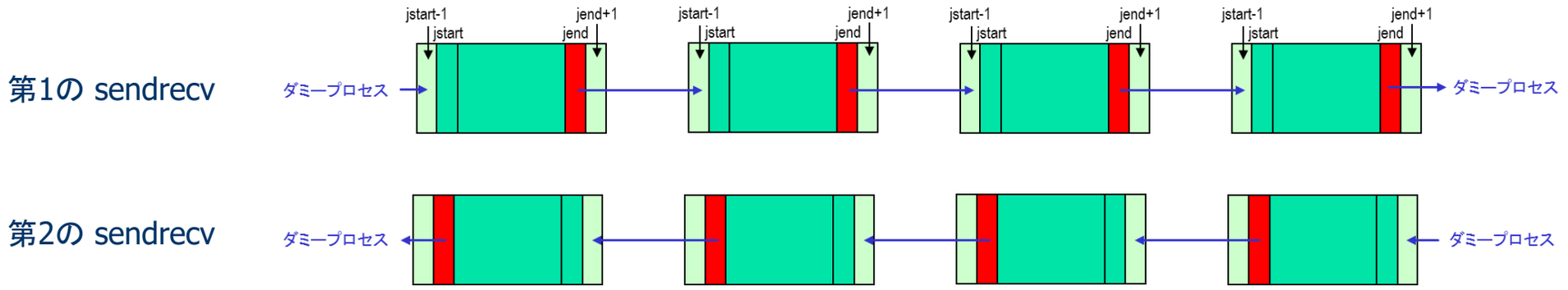
- 2次元配列がブロック列分割されている.

- このとき，自分の両端の１列の要素を，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，その領域にそれぞれ転送するプログラムを作る.



MPI_sendrecvの応用（続き）

- 自プロセスの担当範囲を計算
  - 自プロセスの担当範囲は js列 ～ je列
- mpi_sendrecv による送受信
  - まず，右隣に je 列を送り，左隣から js-1 列に受信
  - 次に，左隣に js 列を送り，右隣から je+1 列に受信
  - 両端のプロセスは，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.f90（続く）

```
program sr_matrix
  use mpi
  implicit none

  integer, parameter :: N=20
  double precision :: u(0:N+1,0:N+1)
  integer :: i, j, js, je

  integer :: nprocs, myrank, left, right, srtag, ierr


  call mpi_init( ierr )
  call mpi_comm_size( MPI_COMM_WORLD, nprocs, ierr )
  call mpi_comm_rank( MPI_COMM_WORLD, myrank, ierr )

  js = (N/nprocs)* myrank + 1
  je = (N/nprocs)*(myrank+1)

  left  = myrank-1
  if( myrank == 0        ) left  = MPI_PROC_NULL
  right = myrank+1
  if( myrank == nprocs-1 ) right = MPI_PROC_NULL

  u(0:N+1,0:N+1) = 0.0
```

各プロセスの担当する列の範囲を計算

左右のプロセスのプロセス番号を計算
（存在しない場合は MPI_PROC_NULL とする）

プログラム sr_matrix.f90 （続き）

```
  u(0:N+1,js:je) = (myrank+1)*10.0

! rightward data circulation
  call mpi_sendrecv( ... )

! leftward data circulation
  call mpi_sendrecv( ... )


  do i=0,nprocs-1
    if( myrank == i ) then
      print '("Rank=",i2,"  js:je=",i4,":",i4)', myrank, js, je
      print '(10f6.2)', (u(N/2,j),j=js-1,je+1)
      flush(6)
    end if
    call MPI_Barrier( MPI_COMM_WORLD, ierr )
  end do

  call mpi_finalize( ierr )

end program sr_matrix
```

mpi_sendrecv による送受信

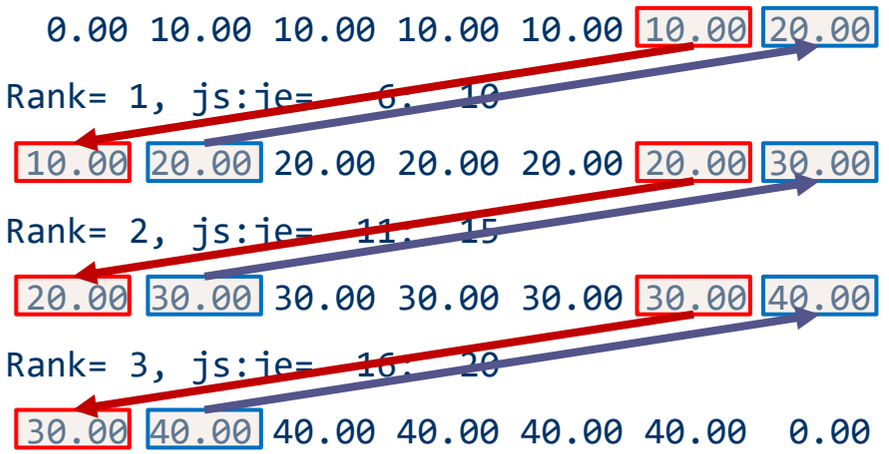
正しく受信できたことを確認

演習11-4F ：プログラムの完成と実行

- sr_matrix.f90 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 /home/guest60/share/sr_matrix.f90  ./
```

- 実行結果



左のプロセスの ie 列が，右のプロセスの (is-1) 列にコピー
右のプロセスの is 列が，下のプロセスの (ie+1) 列にコピー
されていることを確認する.