行列-行列積（２）

東京大学情報基盤センター　准教授　片桐孝洋

2015年6月16日（火）10:25－12:10

講義日程（工学部共通科目）

レポートおよびコンテスト課題
（締切：
2015年8月3日（月）24時　厳守

- ~~4月14日：ガイダンス~~

1. ~~4月21日~~
   - ~~並列数値処理の基本演算（座学）~~
2. ~~4月28日：座学のみ~~
   - ~~ソフトウエア自動チューニング~~
   - ~~非同期通信~~
3. ~~5月12日：スパコン利用開始~~
   - ~~ログイン作業、テストプログラム実行~~
4. ~~5月19日~~
   - ~~高性能演算技法1（ループアンローリング）~~
5. ~~6月2日（8：30－10：15）~~
   - ~~高性能演算技法2（キャッシュブロック化）~~
5. ~~6月2日（10：25－12：10）~~
   - ~~行列-ベクトル積の並列化~~
6. ~~6月9日（8：30－10：15）~~
   ~~★大演習室2~~
   - ~~べき乗法の並列化~~
7. ~~6月9日（10：25－12：10）~~
   - ~~行列-行列積の並列化（1）~~
8. 6月16日
   - 行列－行列積の並列化（2）
9. 6月23日
   - LU分解法（1）
   - コンテスト課題発表
10. 6月30日
    - LU分解法（2）
11. 7月7日
    - LU分解法（3）

講義の流れ

1. 行列-行列積（２）のサンプルプログラムの実行
2. サンプルプログラムの説明
3. 演習課題（２）：ちょっと難しい完全分散版
4. 並列化のヒント

行列 - 行列積の演習の流れ

- 演習課題（１）
  - 簡単なもの（30分程度で並列化）
  - 通信関数が一切不要
- **演習課題（２）**
  - ちょっと難しい（１時間以上で並列化）
  - １対１通信関数が必要

サンプルプログラムの実行
（行列-行列積（２））

行列-行列積のサンプルプログラムの注意点

- C言語版/Fortran言語版のファイル名

  Mat-Mat-d-fx.tar

- ジョブスクリプトファイルmat-mat-d.bash　中のキュー名を　lecture から

  lecture7　(工学部共通科目)、

  に変更し、

- pjsub　してください。
  - lecture : 実習時間外のキュー
  - lecture7: 実習時間内のキュー

行列-行列積(2)のサンプルプログラムの実行

- 以下のコマンドを実行する

  $ cp /home/z30082/Mat-Mat-d-fx.tar ./
  $ tar xvf Mat-Mat-d-fx.tar
  $ cd Mat-Mat-d

- 以下のどちらかを実行

  $ cd C :C言語を使う人
  $ cd F :Fortran言語を使う人

- 以下共通

  $ make
  $ pjsub mat-mat-d.bash

- 実行が終了したら、以下を実行する

  $ cat mat-mat-d.bash.oXXXXXX

行列-行列積のサンプルプログラムの実行（C言語版）

- 以下のような結果が見えれば成功

```
N  = 384
Mat-Mat time  = 0.000135 [sec.]
 841973.194818 [MFLOPS]
 Error! in ( 0 , 2 )-th argument in PE 0
 Error! in ( 0 , 2 )-th argument in PE 61
 Error! in ( 0 , 2 )-th argument in PE 51
 Error! in ( 0 , 2 )-th argument in PE 59
 Error! in ( 0 , 2 )-th argument in PE 50
 Error! in ( 0 , 2 )-th argument in PE 58
....
```

並列化が完成していないのでエラーが出ます。ですが、これは正しい動作です

行列-行列積のサンプルプログラムの実行（Fortran言語）

- 以下のような結果が見えれば成功

```
NN  =  384
Mat-Mat time  =  1.29550899146124 7E-03
MFLOPS =  87414.45135502046
 Error! in ( 1 , 3 )-th argument in PE  0
 Error! in ( 1 , 3 )-th argument in PE  61
 Error! in ( 1 , 3 )-th argument in PE  51
 Error! in ( 1 , 3 )-th argument in PE  58
 Error! in ( 1 , 3 )-th argument in PE  55
 Error! in ( 1 , 3 )-th argument in PE  63
 Error! in ( 1 , 3 )-th argument in PE  60
```

並列化が
完成して
いないので
エラーが出ます。
ですが、
これは正しい
動作です。

サンプルプログラムの説明

- #define N 384
  - 数字を変更すると、行列サイズが変更できます
- #define DEBUG 1
  - 「0」を「1」にすると、行列-行列積の演算結果が検証できます。
- MyMatMat関数の仕様
  - Double型の行列A((N/NPROCS)×N行列)と B((N×(N/NPROCS)行列))の行列積をおこない、Double型の(N/NPROCS)×N行列Cに、その結果が入ります。

Fortran言語のサンプルプログラムの注意

- 行列サイズNの宣言は、以下のファイルにあります。

  mat-mat-d.inc

- 行列サイズ変数が、NNとなっています。

  integer NN

  parameter (NN=384)

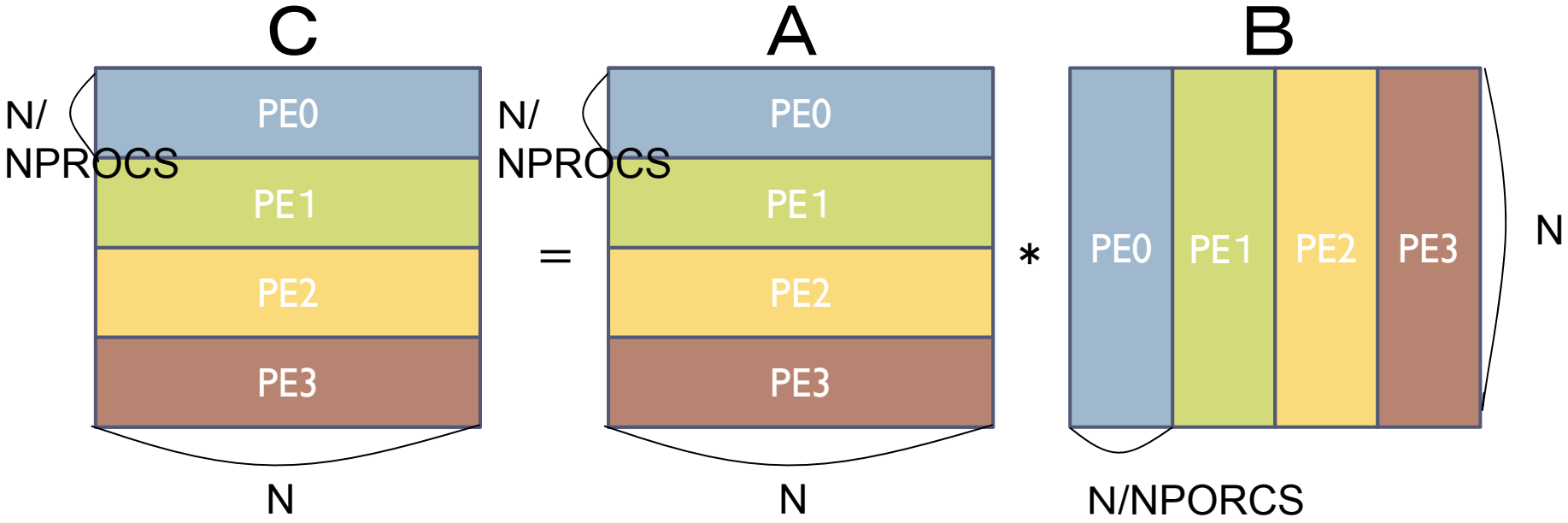
演習課題（１）

- MyMatMat関数（手続き）を並列化してください。
  - デバック時は
    #define　N　384
    としてください。

- 行列A、B、Cの初期配置（データ分散）を、十分に考慮してください。

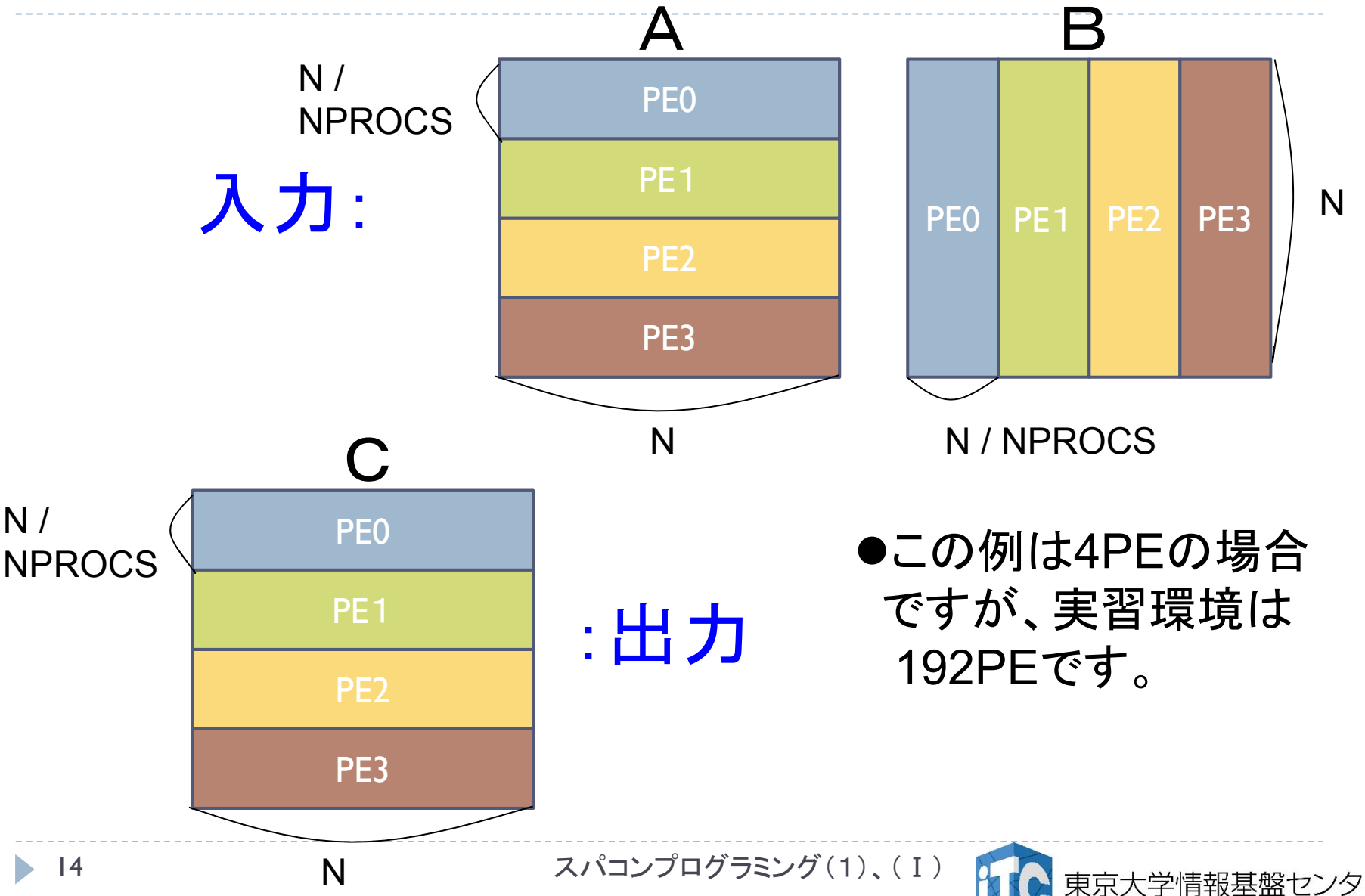
行列A、B、Cの初期配置

- 行列A、B、Cの配置は以下のようになっています。
  （ただし以下は4PEの場合で、実習環境は192PEです。）



- １対１通信関数が必要です。
- 行列A、B、Cの配列のほかに、受信用バッファの配列が必要です。

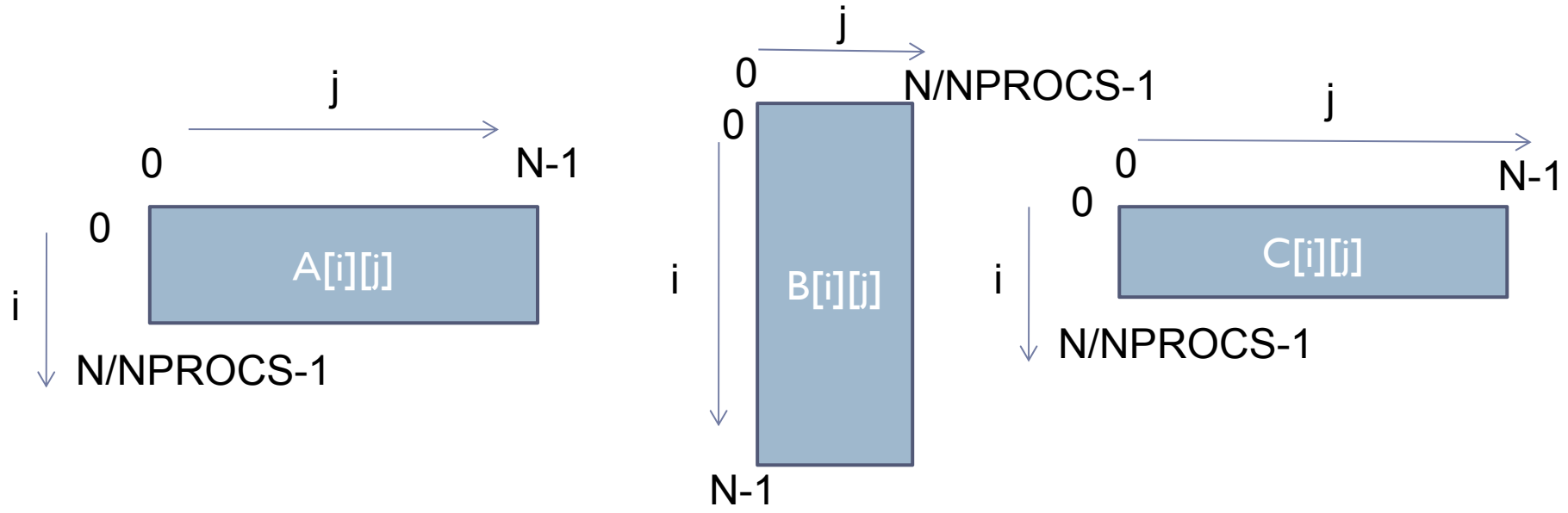
入力と出力仕様



●この例は4PEの場合ですが、実習環境は192PEです。

並列化の注意（C言語）

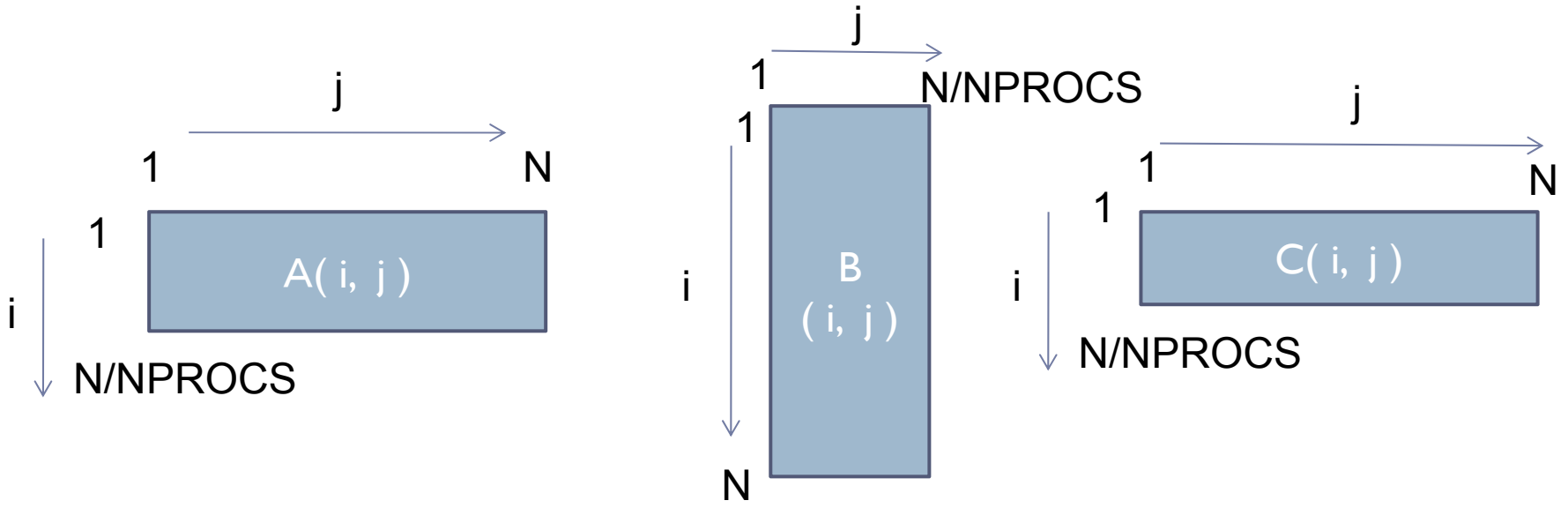
- 各配列は、完全に分散されています。
- 各PEでは、以下のようなインデックスの配列となっています。



- 各PEで行う、ローカルな行列-行列積演算時の
  インデックス指定に注意してください。

並列化の注意（Fortran言語）

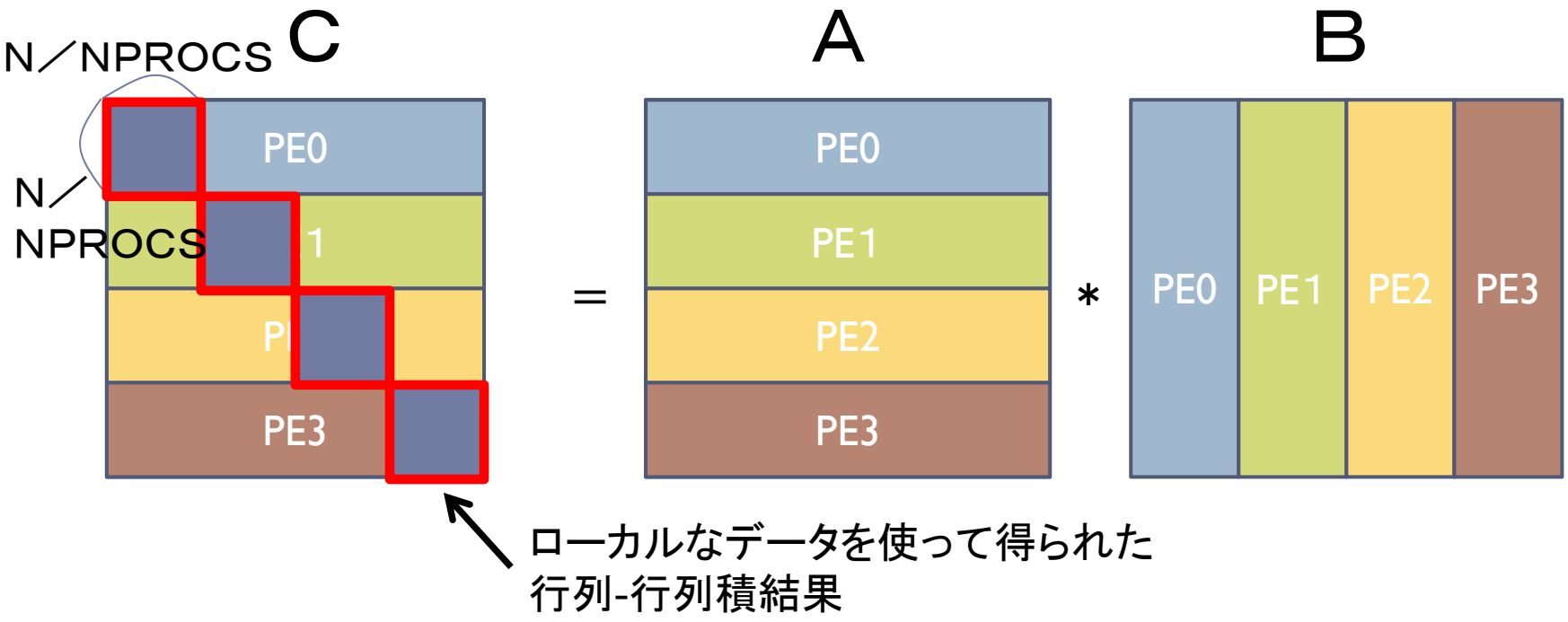
- 各配列は、完全に分散されています。
- 各PEでは、以下のようなインデックスの配列となっています。



- 各PEで行う、ローカルな行列-行列積演算時の
  インデックス指定に注意してください。

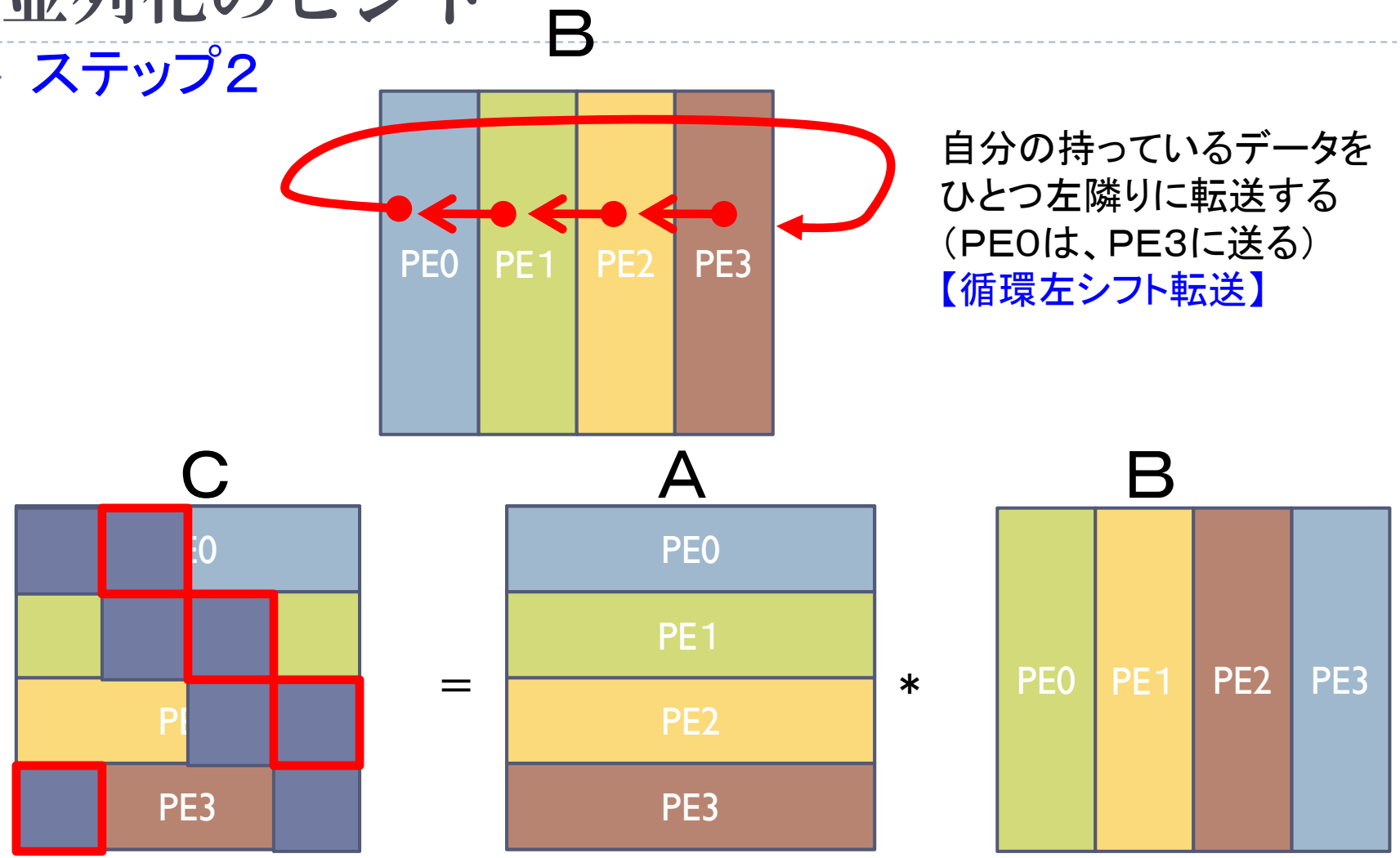
並列化のヒント

- 行列積を計算するには、各PEで完全な行列Bのデータがないので、行列Bのデータについて通信が必要です。
- たとえば、以下のように計算する方法があります。
- ステップ1



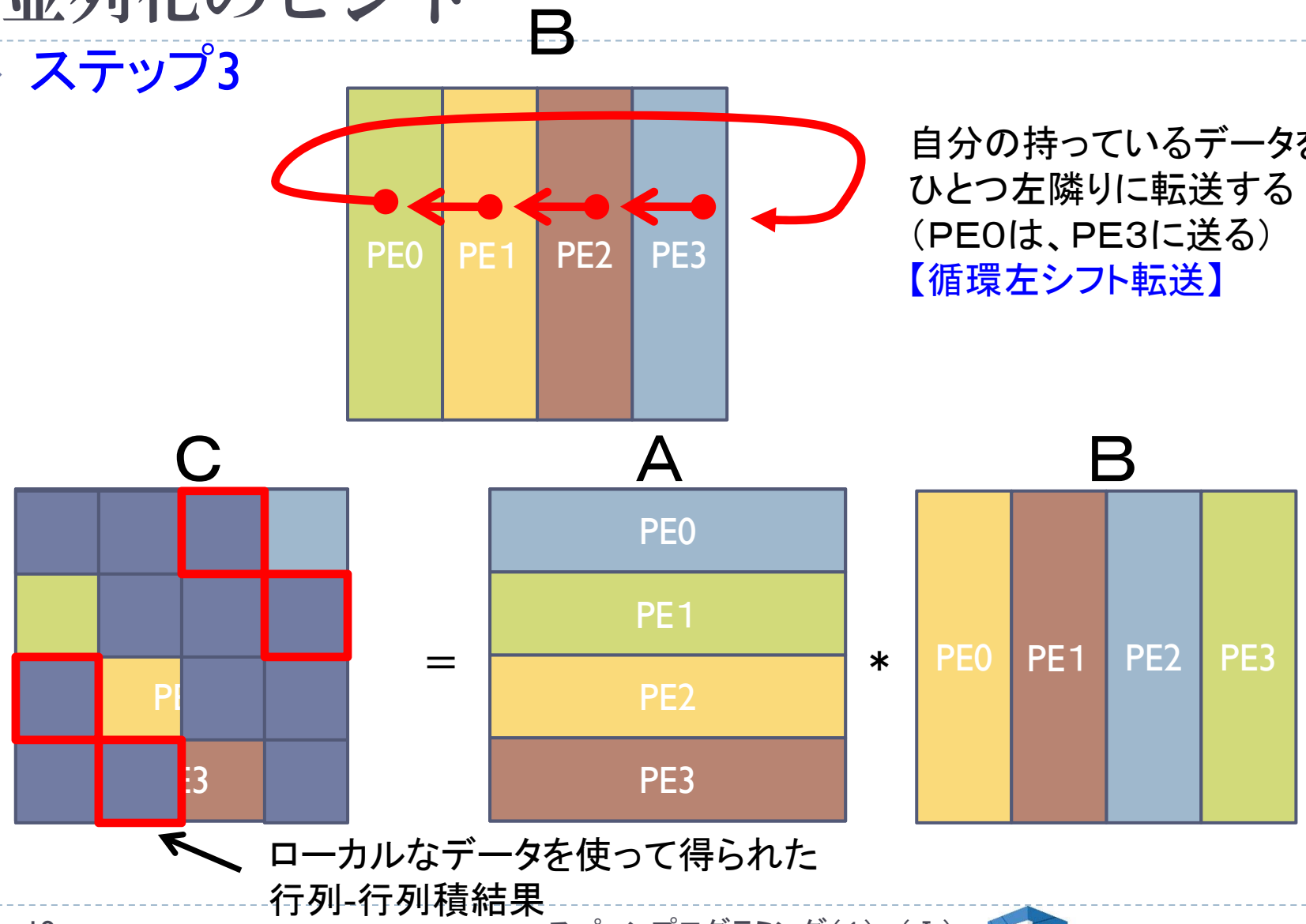
並列化のヒント

- ステップ2



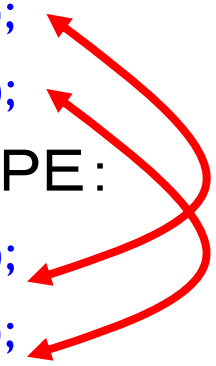
並列化のヒント

- ステップ3



並列化の注意

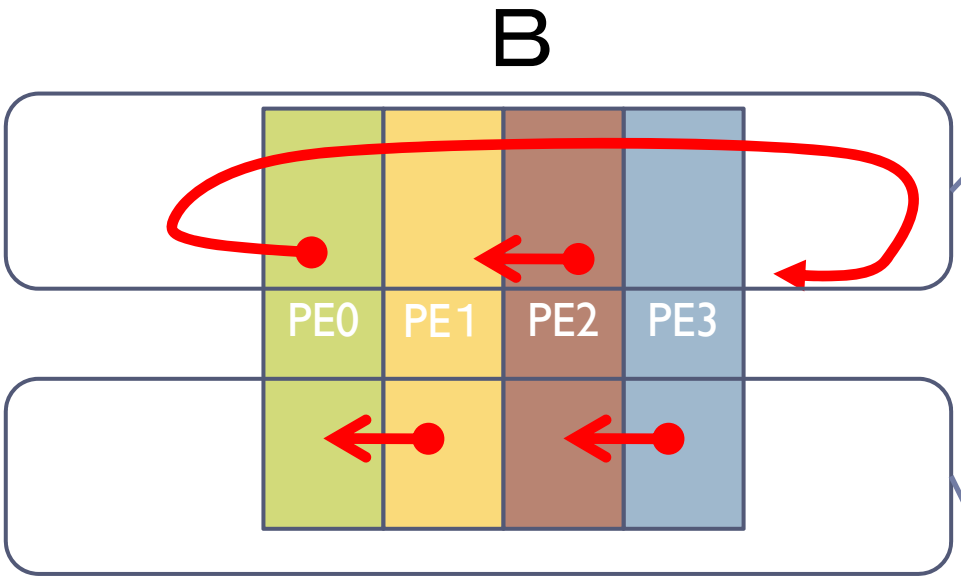
- 循環左シフト転送を実装する際、全員がMPI_Sendを先に発行すると、その場所で処理が止まる。
  （正確には、動いたり、動かなかったり、する）
  - MPI_Sendの処理中で、場合により、バッファ領域がなくなる。
  - バッファ領域が空くまで待つ（スピンウェイトする）。
  - しかし、バッファ領域不足から、永遠に空かない。
- これを回避するため、以下の実装を行う。
  - PE番号が2で割り切れるPE:
    - MPI_Send();
    - MPI_Recv();
  - それ以外のPE:
    - MPI_Recv();
    - MPI_Send();



それぞれに対応

並列化の注意

- つまり、以下の2ステップで、循環左シフト通信をする



ステップ1：
2で割り切れるPEが
データを送る

ステップ2：
2で割り切れないPEが
データを送る

基礎的なMPI関数—MPI_Send

```
ierr = MPI_Send(sendbuf, icount, idatatype, idest,
       itag, icomm);
```

- sendbuf : 送信領域の先頭番地を指定する
- icount : 整数型。送信領域のデータ要素数を指定する
- idatatype : 整数型。送信領域のデータの型を指定する
- idest : 整数型。送信したいPEのicomm内でのランクを指定する。
- itag : 整数型。受信したいメッセージに付けられたタグの値を指定する。
- icomm : 整数型。プロセッサー集団を認識する番号であるコミュニケータを指定する。
- ierr (戻り値) : 整数型。エラーコードが入る。

基礎的なMPI関数—MPI_Recv（１／２）

- ierr = MPI_Recv(recvbuf, icount, idatatype, isource, itag, icomm, istatus);

  - recvbuf : 受信領域の先頭番地を指定する。
  - icount : 整数型。受信領域のデータ要素数を指定する。
  - idatatype : 整数型。受信領域のデータの型を指定する。
    - MPI_CHAR (文字型) 、MPI_INT (整数型)、MPI_FLOAT (実数型)、 MPI_DOUBLE(倍精度実数型)
  - isource : 整数型。受信したいメッセージを送信するPEのランクを指定する。
    - 任意のPEから受信したいときは、MPI_ANY_SOURCE を指定する。

基礎的なMPI関数—MPI_Recv（２／２）

- itag： 整数型。受信したいメッセージに付いているタグの値を指定する。
  - 任意のタグ値のメッセージを受信したいときは、MPI_ANY_TAG を指定する。
- icomm： 整数型。PE集団を認識する番号であるコミュニケータを指定する。
  - 通常ではMPI_COMM_WORLD を指定すればよい。
- istatus： MPI_Status型（整数型の配列）。受信状況に関する情報が入る。
  - 要素数がMPI_STATUS_SIZEの整数配列が宣言される。
  - 受信したメッセージの送信元のランクが istatus[MPI_SOURCE]、タグが istatus[MPI_TAG] に代入される。
- ierr(戻り値)： 整数型。エラーコードが入る。

実装上の注意

- タグ(itag)について
  - MPI_Send(), MPI_Recv()で現れるタグ（itag）は、任意のint型の数字を指定してよいです。
  - ただし、同じ値（0など）を指定すると、どの通信に対応するかわからなくなり、誤った通信が行われるかもしれません。
  - 循環左シフト通信では、MPI_Send()と MPI_Recv()の対が、2つでてきます。これらを別のタグにした方が、より安全です。
  - たとえば、一方は最外ループの値iloopとして、もう一方をiloop+NPROCSとすれば、全ループ中でタグがぶつかることがなく、安全です。

さらなる並列化のヒント

以降、本当にわからない人のための資料です。
ほぼ回答が載っています。

並列化のヒント

1. 循環左シフトは、PE総数-1回 必要
2. 行列Bのデータを受け取るため、行列B[][]に関するバッファ行列B_T[][]が必要
3. 受け取ったB_T[][] を、ローカルな行列-行列積で使うため、B[][]へコピーする。
4. ローカルな行列-行列積をする場合の、対角ブロックの初期値：ブロック幅*myid。
   ループ毎にブロック幅だけ増やしていくが、
   Nを超えたら0に戻さなくてはいけない。

並列化のヒント（ほぼ回答、Ｃ言語）

- 以下のようなコードになる。

```
ib = n/numprocs;
for (iloop=0;  iloop<NPROCS;  iloop++ )  {
    ローカルな行列-行列積  C  =  A * B;
    if (iloop != (numprocs-1) ) {
      if (myid % 2 == 0 ) {
          MPI_Send(B, ib*n, MPI_DOUBLE, isendPE,
                iloop, MPI_COMM_WORLD);
          MPI_Recv(B_T, ib*n, MPI_DOUBLE, irecvPE,
                iloop+numprocs, MPI_COMM_WORLD, &istatus);
      } else {
          MPI_Recv(B_T, ib*n, MPI_DOUBLE, irecvPE,
                iloop, MPI_COMM_WORLD, &istatus);
          MPI_Send(B, ib*n, MPI_DOUBLE, isendPE,
                iloop+numprocs, MPI_COMM_WORLD);
      }
      B[ ][ ]  へ  B_T[ ][ ]  をコピーする;
    }
  }
```

並列化のヒント（ほぼ回答、Ｃ言語）

- ローカルな行列-行列積は、以下のようなコードになる。

```
jstart=ib*( (myid+iloop)%NPROCS );
for (i=0;  i<ib;  i++) {
   for(j=0;  j<ib;  j++) {
      for(k=0;  k<n;  k++) {
         C[ i ][ jstart + j ] += A[ i ][ k ] * B[ k ][ j ];
      }
   }
}
```

並列化のヒント（ほぼ回答，Fortran言語）

- 以下のようなコードになる。

```
ib = n/numprocs
do iloop=0,  NPROCS-1
    ローカルな行列-行列積  C  =  A * B
    if (iloop .ne. (numprocs-1) ) then
      if (mod(myid, 2)  .eq. 0 ) then
         call MPI_SEND(B, ib*n, MPI_DOUBLE_PRECISION,  isendPE,
&                iloop, MPI_COMM_WORLD,  ierr)
         call MPI_RECV(B_T,  ib*n,  MPI_DOUBLE_PRECISION,  irecvPE,
&                iloop+numprocs, MPI_COMM_WORLD,  istatus,  ierr)
      else
         call MPI_RECV(B_T,  ib*n,  MPI_DOUBLE_PRECISION,  irecvPE,
&                iloop,  MPI_COMM_WORLD,  istatus,  ierr)
         call MPI_SEND(B,  ib*n,  MPI_DOUBLE_PRECISION,  isendPE,
&                iloop+numprocs,  MPI_COMM_WORLD,  ierr)
      endif
      B  へ  B_T  をコピーする
    endif
  enddo
```

並列化のヒント（ほぼ回答，Fortran言語）

- ローカルな行列-行列積は、以下のようなコードになる。

```
imod = mod( (myid+iloop), NPROCS )
jstart = ib* imod
do i=1, ib
  do j=1, ib
    do k=1, n
      C( i , jstart + j ) = C( i , jstart + j ) +  A( i , k ) * B( k , j )
    enddo
  enddo
enddo
```

来週へつづく

LU分解法（1）