「情報システム学特別講義３」
演習：行列-ベクトル積の並列化

東京大学情報基盤センター准教授　片桐孝洋

2014年6月10日（火）14:40－16:10

東京大学情報基盤センター
Information Technology Center,The University of Tokyo

講義日程
（情報システム学特別講義３）

レポートおよびコンテスト課題
（締切：
2014年8月11日（月）24時　厳守

1. ~~4月8日：ガイダンス~~
2. ~~4月15日~~
   - ~~プログラム高速化の基礎（その1）~~
3. ~~4月22日~~
   - ~~プログラム高速化の基礎（その2）~~
4. ~~5月13日~~
   - ~~MPIの基礎~~
5. ~~5月20日~~
   - ~~OpenMPの基礎~~
6. ~~5月27日~~
   - ~~Hybrid並列化技法（MPIとOpenMPの応用編）~~
7. ~~6月3日~~
   - ~~プログラム高速化の応用~~
8. 6月10日
   - 行列ーベクトル積の並列化
9. 6月17日
   - べき乗法の並列化
10. 6月24日
    - 行列−行列積の並列化
11. 7月8日
    - LU分解の並列化
12. 7月15日
    - 非同期通信
    - 疎行列反復解法の並列化
13. 7月22日
    - ソフトウェア自動チューニング
14. 8月5日（補講日）
    - エクサフロップスコンピューティングに向けて

講義の流れ

1. 行列-ベクトル積のサンプルプログラムの実行
2. 並列化の注意点
3. 並列化実習
4. レポート課題

並列化の方針

行列 - ベクトル積とは

- 以下の演算

$$y = Ax$$

  - ここで、
    - Aは、実数の密行列。
      - 配列はA[N][N]。
    - x、y　は、実数のベクトル。
      - 配列は、x[N]、y[N]。

行列 - ベクトル積のプログラム（C言語）

```
for( j=0; j<n; j++)  {
        y[ j ] = 0.0;
        for(i=0; i<n; i++)  {
          y[ j ] += A[ j ][ i ] * x[ i ];
         }
}
```

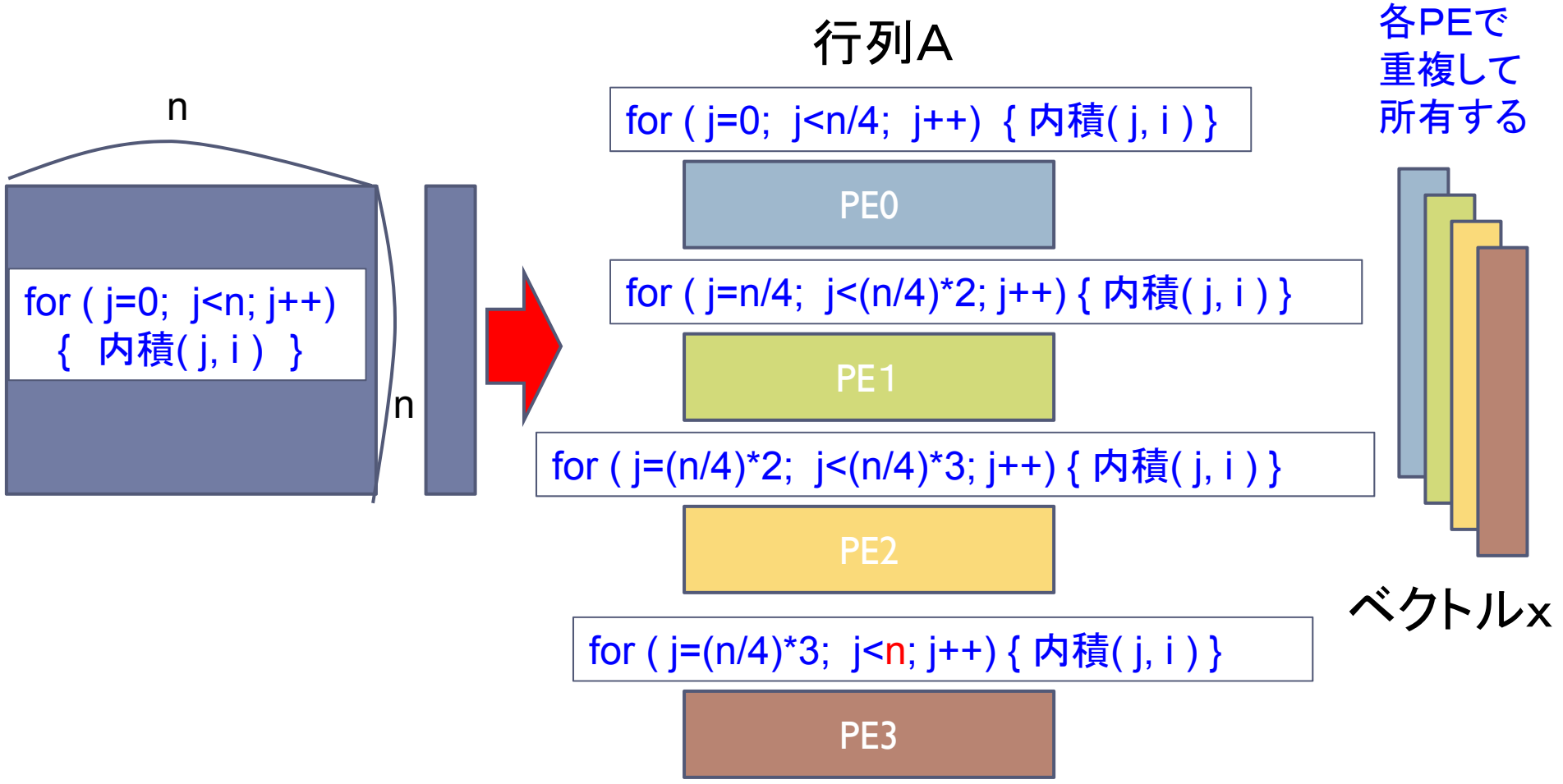
```
for( j=0; j<n; j++)  {
      内積( j, i )
}
```

内積(j, i)

```
y[ j ] = 0.0;
for(i=0; i<n; i++)  {
   y[ j ] += A[ j ][ i ] * x[ i ]; }
```

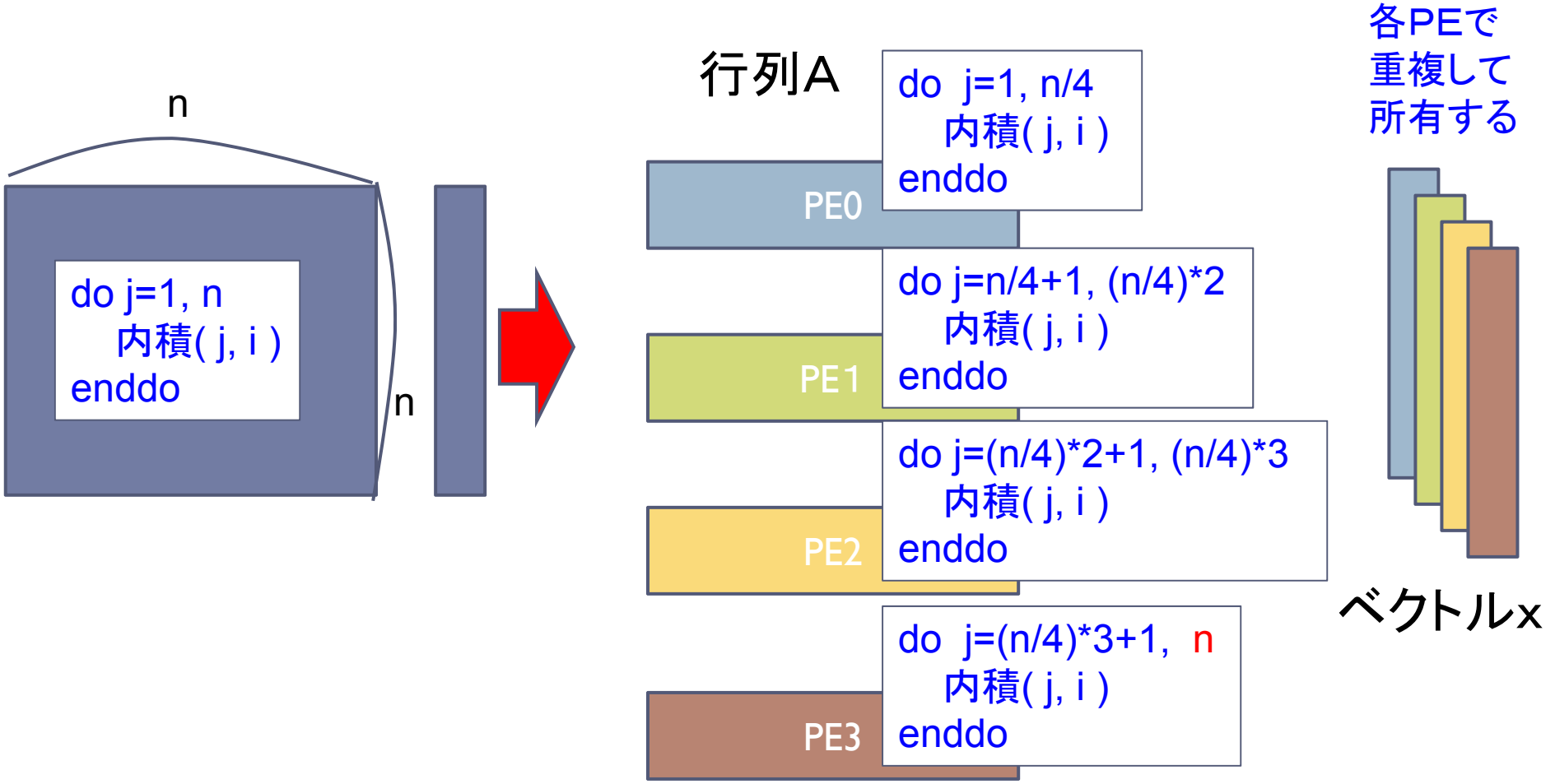
並列化の考え方（C言語）

- SIMDアルゴリズムの考え方（4PEの場合）



並列化の考え方（Fortran言語）

- SIMDアルゴリズムの考え方（4PEの場合）

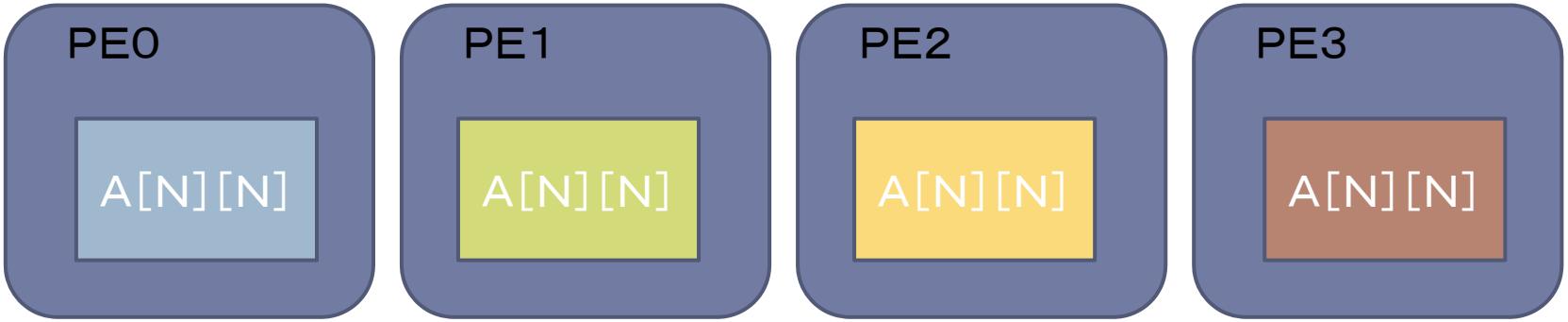


ここでの注意

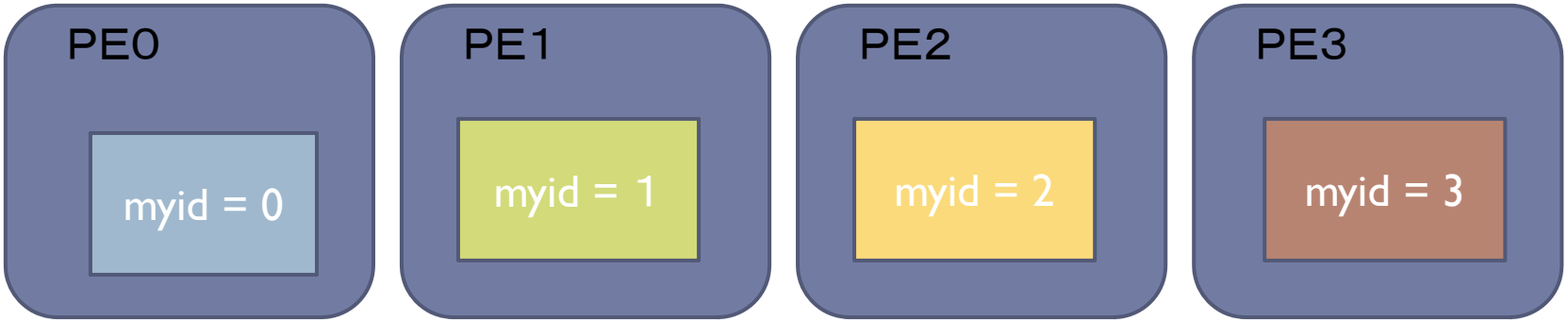
- 4MPIプロセス実行だからといって、SIMDの考え方のループを4つ書いてはいけない！
  - １万並列なら１万個書くのか？
  - 書けるわけがない
- MPIの並列化における本質課題
  - SIMDの考え方のループが、
    １つのプログラム上で
    どうやって汎用的に書けるのか
    考えること

初心者が注意すること

- 各PEでは、独立した配列が個別に確保されます。



- myid変数は、MPI_Init()関数が呼ばれた段階で、
  各PE固有の値になっています。



並列プログラム開発の指針

1. 正しく動作する逐次プログラムを作成する
2. 1．のプログラムで、適切なテスト問題を作成する
3. 2．のテスト問題の実行について、適切な処理の単位ごとに、正常動作する計算結果を確認する
4. 1．の逐次プログラムを並列化し、並列プログラミングを行う
5. 2．のテスト問題を実行して動作検証する
6. このとき3．の演算結果と比較し、正常動作をすることを確認する。もし異常であれば、4．に戻りデバックを行う。

データ分散方式に関する注意

- 負荷分散を考慮し、多彩なデータ分散方式を採用可能
- しかし、数学的に単純なデータ分散方式が良い
  - ◎：ブロック分散、サイクリック分散（ブロック幅＝１）
  - △～○：ブロック・サイクリック分散（ブロック幅＝任意）
  - 理由：
    - 複雑な（一般的な）データ分散は、各MPIプロセスが所有するデータ分散情報（インデックスリスト）を必要とするため、メモリ量が余分に必要なる
      - 例：１万並列では、少なくとも１万次元の整数配列が必要
      - 数学的に単純なデータ分散の場合は、インデックスリストは不要

並列化の方針（C言語）

1. 全PEで行列AをN×Nの大きさ、ベクトルx、yをNの大きさ、確保してよいとする。
2. 各PEは、担当の範囲のみ計算するように、ループの開始値と終了値を変更する。
   - ブロック分散方式では、以下になる。
     （n が numprocs で割り切れる場合）

```
ib = n / numprocs;
for ( j=myid*ib;  j<(myid+1)*ib;  j++)    { … }
```

3. （2の並列化が完全に終了したら）各PEで担当のデータ部分しか行列を確保しないように変更する。
   - 上記のループは、以下のようになる。

```
for ( j=0;  j<ib;  j++)  { … }
```

並列化の方針（Fortran言語）

1. 全PEで行列AをN×Nの大きさ、ベクトルx、yをNの大きさ、確保してよいとする。
2. 各PEは、担当の範囲のみ計算するように、ループの開始値と終了値を変更する。
   - ブロック分散方式では、以下になる。
     （n が numprocs で割り切れる場合）

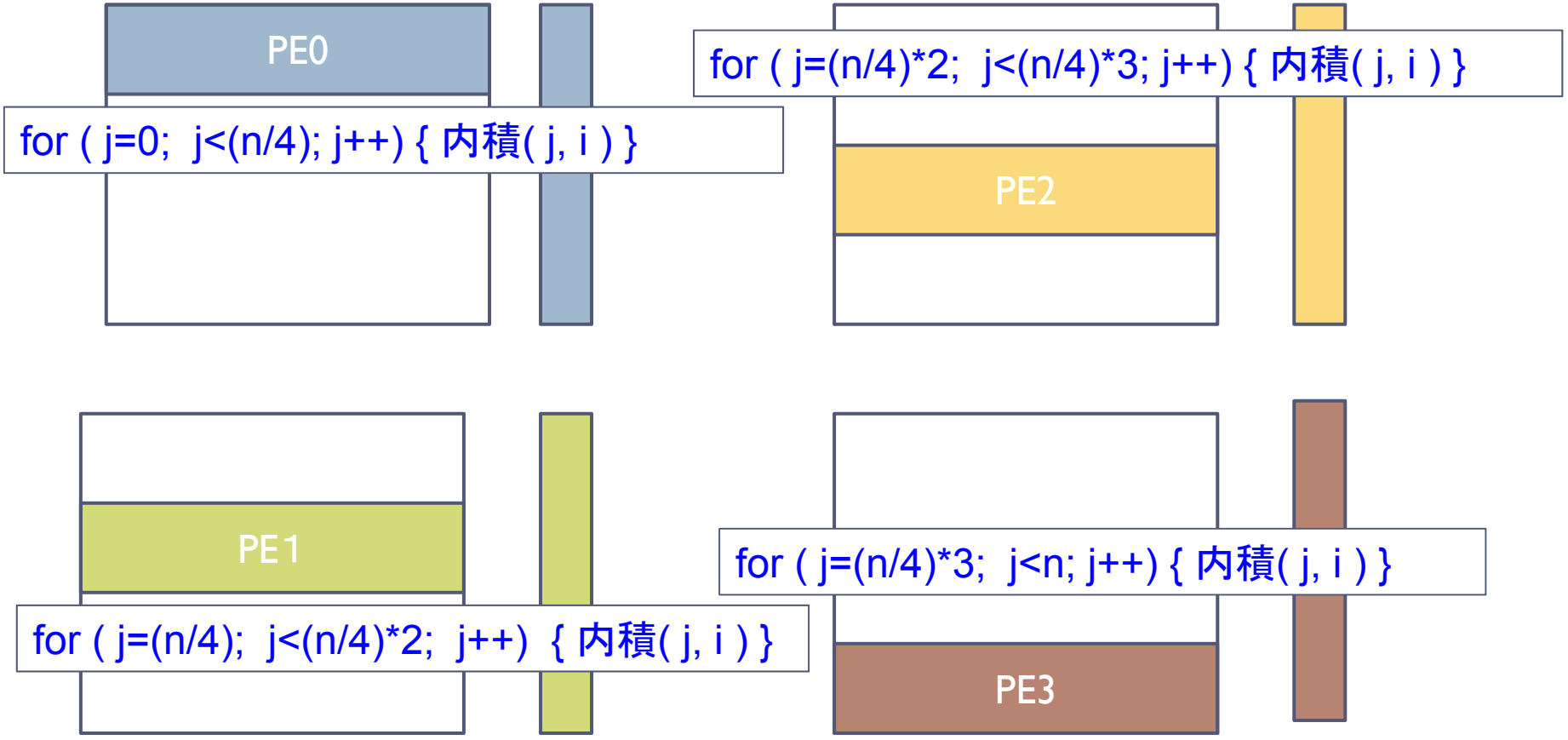
```
ib = n / numprocs
do j=myid*ib+1, (myid+1)*ib   ....   enddo
```

3. （2の並列化が完全に終了したら）各PEで担当のデータ部分しか行列を確保しないように変更する。
   - 上記のループは、以下のようになる。

```
do  j=1, ib  ....  enddo
```

並列化の方針（行列-ベクトル積）（C言語）

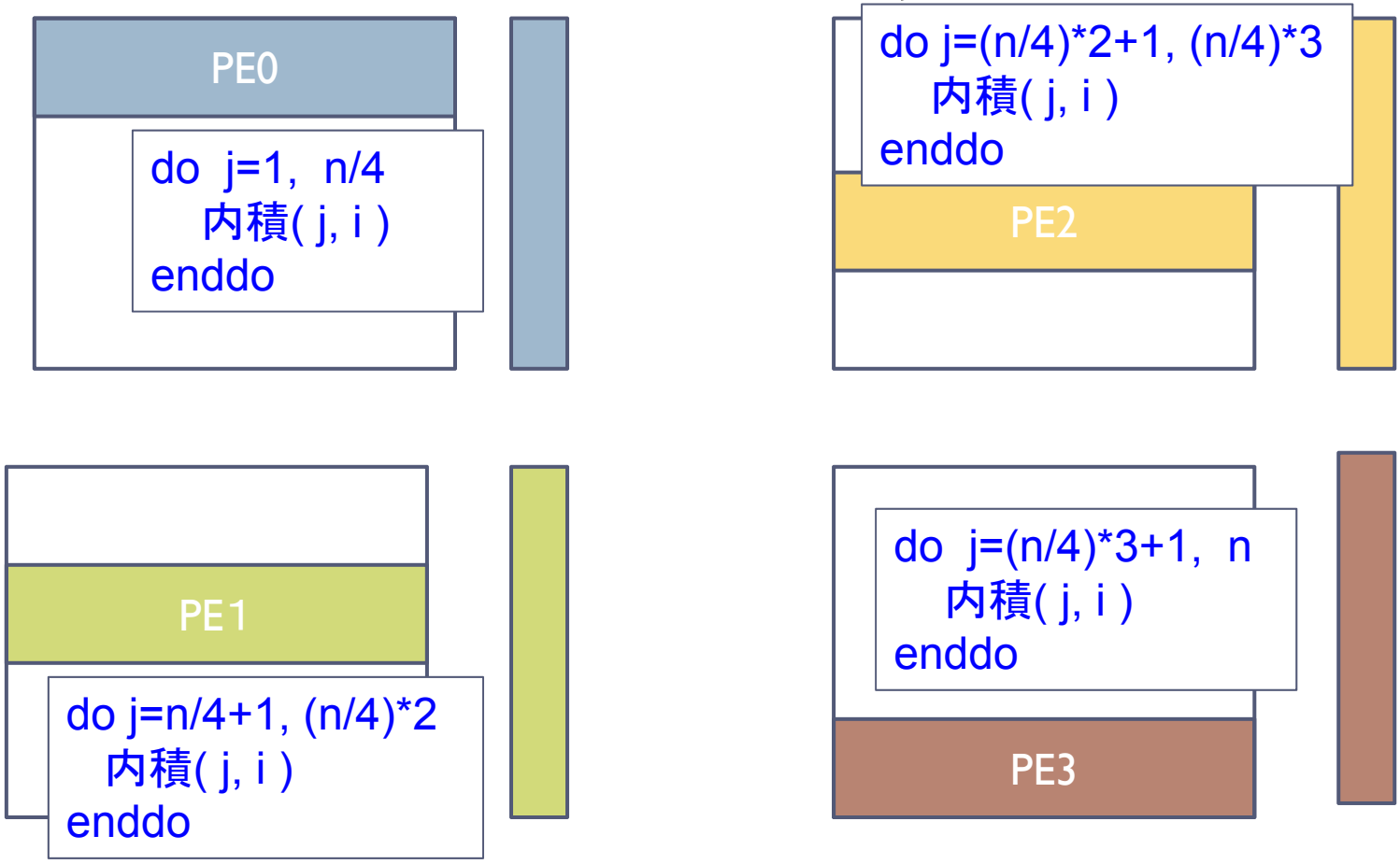
- 全PEでN×N行列を持つ場合



※各PEで使われない領域が出るが、担当範囲指定がしやすいので実装がしやすい。

並列化の方針（行列-ベクトル積）（Fortran 言語）

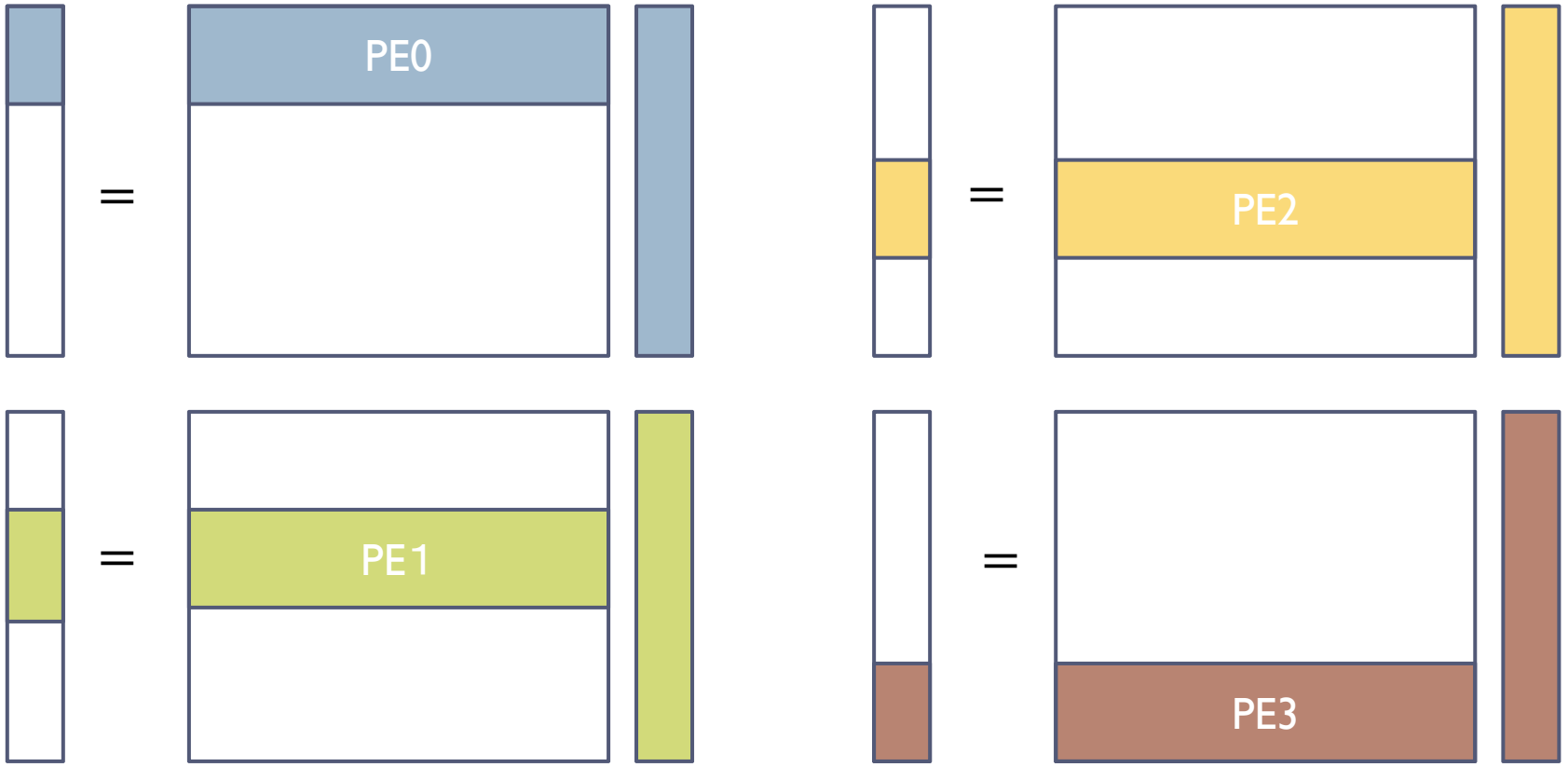
- 全PEでN×N行列を持つ場合



※各PEで使われない領域が出るが、担当範囲指定がしやすいので実装がしやすい。

並列化の方針（行列-ベクトル積）

- この方針では、y＝Axのベクトルyは、以下のように一部分しか計算されないことに注意！



並列化の方針のまとめ

- 行列全体（A[N][N]）を各プロセスで確保することで、SIMDの考え方を、逐次プログラムに容易に適用できる
  - ループの開始値、終了値のみ変更すれば、並列化が完成する
  - この考え方は、MPI、OpenMPに依存せず、適用できる。
  - 欠点
    - 最大実行可能な問題サイズが、利用ノード数によらず、1ノードあたりのメモリ量で制限される（メモリに関するスケーラビリティが無い）
- ステップ2のデバックの困難性を低減できる
  - 完全な並列化（ステップ2）の際、ステップ1での正しい計算結果を参照できる
  - 数値計算上の処理単位ごとに、ステップごとに完全並列化ができる（モジュールごとに、デバックできる）

サンプルプログラムの実行（行列-ベクトル積）

はじめての基本演算

EMACSコマンドの再確認

- C- : Control キーを押しながら
- M- : Esc キーを押しながら
- C-x C-s : データセーブ
- C-x C-c : 終了
- C-g : わからなくなったとき
- C-k : 1行消去してバッファにコピー
  （連続して入力すると複数行消去可）
- C-y : 上記のバッファをカーソル位置にコピー
- C-s : 文字列を検索し、その場所に移動。以降 C-s で次の候補に移動する。移動したい関数名を入れて利用する。
- M-x goto-line : 行きたい行に飛ぶ。入力後、行の番号を聞いてくる。

行列-ベクトル積のサンプルプログラムの注意点

- C言語／Fortran言語版のファイル名

  Mat-vec-fx.tar

- ジョブスクリプトファイルmat-vec.bash
  - lecture : 実習のキュー

行列－ベクトル積のサンプルプログラムの実行（C言語）

- 以下のコマンドを実行する

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

- 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-vec.bash

- 実行が終了したら、以下を実行する

  $ cat mat-vec.bash.oXXXXXX

実行結果（C言語）

- 以下のような結果が出ればOK。

```
N  = 10000
Mat-Vec time  = 0.171097 [sec.]
 1168.927027 [MFLOPS]
OK!
```

実行結果（Fortran言語）

- 以下のような結果が出ればOK。

N = 10000
Mat-Vec time[sec.] = 0.16659261297900 23
MFLOPS = 1200.533420532020
OK!

サンプルプログラムの説明（C言語）

- #define N 10000
  数字を変更すると、行列サイズが変更できます

- #define DEBUG 1
  「1」としてコンパイルすると、演算結果が正しいことがチェックできます。
- 再コンパイルは、以下のように入力します。
  % make clean
  % make

Fortran言語のサンプルプログラムの注意

- 行列サイズNNの宣言は、以下のファイルにあります。

  mat-vec.inc

- 行列サイズ変数が、NNとなっています。

```
integer  NN
parameter (NN=10000)
```

演習課題

- MyMatVec関数（手続き）の＜中身＞を並列化してください。
  - デバック時には、
    - #define N 192
      にしてください。そうしないと、実行時間が大変かかってしまいます。
    - #define DEBUG 1
      にして、結果を検証してください。

演習課題の注意

- データが各PEに完全に分散された状態から初めてください。
  （データ分散の処理は不要です）
- 以下はデータの中身を気にする人に:
  - 結果を検証する場合、行列とベクトルの初期データはすべて1です。
  - 結果を検証しない場合には、行列とベクトルの初期データに、疑似乱数を使っています。
    - 疑似乱数は、乱数の種を固定しない限り各PEで同じ値になることは保証されません。
    - このサンプルプログラムでは、srand()関数で乱数の種を固定していますので全PEで同じ乱数系列が発生されます。
    - 逐次と同じデータの中身を並列版で保障する場合、自分の担当部分まで乱数を発生させて、不要な場所は発生した乱数を捨てる必要があります。

演習課題の注意

- 本実習では、MPI通信関数は不要です。
- このサンプルプログラムでは、
  演算結果検証部分が並列化されていない

ため、MatVec関数のみを並列化しても、
検証部でエラーとなります。

- 検証部分も、計算されたデータに各PEで対応するように、並列化してください。
- 検証部分においても、行列-ベクトル積と同様のループとなります。

MPI並列化の大前提（再確認）

- SPMD
  - 対象のメインプログラム（mat-vec.c）は、
    - すべてのPEで、かつ、
    - 同時に起動された状態

    から処理が始まる。
- 分散メモリ型並列計算機
  - 各PEは、完全に独立したメモリを持っている。（共有メモリではない）

本実習プログラムのTIPS

- myid, numprocs は大域変数です
  - myid (=自分のID)、および、numprocs(=世の中のPE台数)の変数は大域変数です。

    MyMatVec関数内で、引数設定や宣言なしに、参照できます。
- myid, numprocs の変数を使う必要があります
  - MyMatVec関数を並列化するには、myid 、および、numprocs変数を利用しないと、並列化ができません。

並列化時の注意

- 演習環境は、I92PEです。
- 動作確認には、サンプルプログラムにあるデバック機能を利用しましょう。
  - 並列化は、＜できた＞と思ってもバグっていることが多い！
  - このサンプルでは、PE0がベクトルyの要素すべてを所有することが前提となっています。

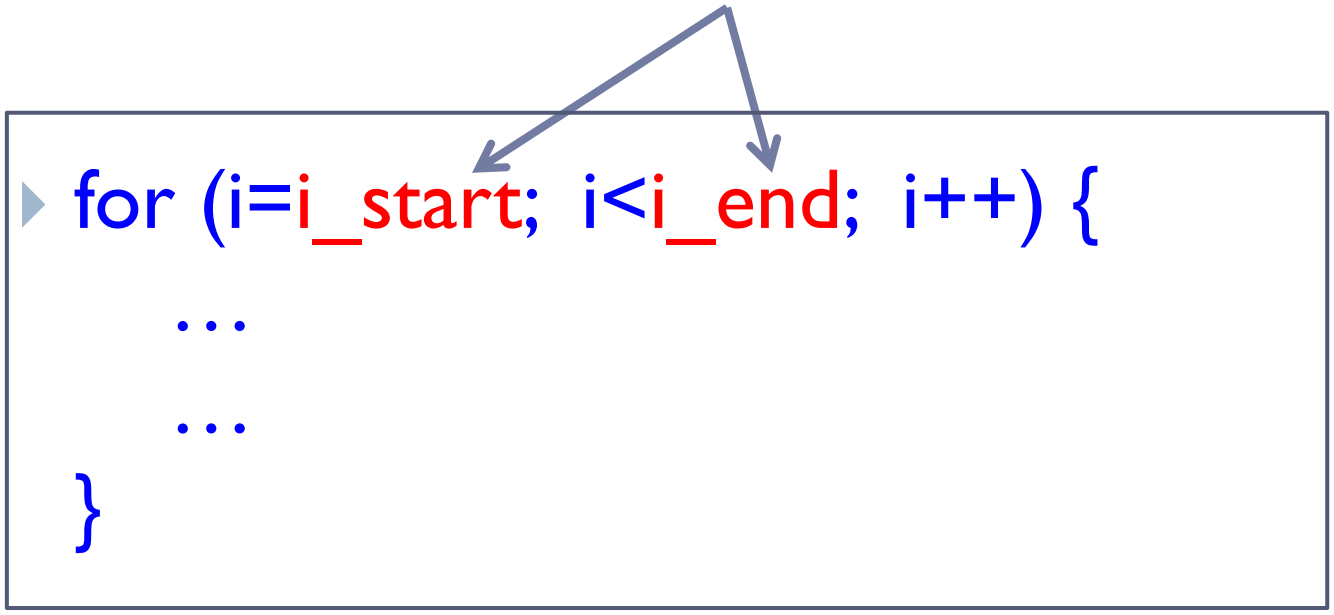
出力結果を考慮して検証部分も並列化してください。

  - Nを小さくして、printfで結果（ベクトルy）を目視することも、デバックになります。しかし、Nを目視できないほど大きくする場合にバグることがあります。目視のみデバックは、経験上お勧めしません。
  - 数学ライブラリ開発では、できるだけ数学（線形代数）の知識を利用した方法で、理論的な解と結果を検証することをお勧めします。

実装上の注意

- ループ変数をグローバル変数にすると、性能が出ません。
  必ずローカル変数か、定数（　2　など）にしてください。

ローカル変数にすること



```
for (i=i_start;  i<i_end;  i++) {
    …
    …
}
```

nがMPIプロセス数で割切れない時

- nがプロセス数のnumprocsで割り切れない場合
  - 配列確保： A（N/numprocs + mod(N, numprocs), N）
  - ループ終了値：numprocs-1のみ終了値がnとなるように実装

```
ib = n / numprocs;
if ( myid == (numprocs - 1) ) {
      i_end = n;
} else {
      i_end = (myid+1)*ib;
}
for ( i=myid*ib;  i<i_end;  i++) { … }
```

余りが多い場合

- mod(N, numprocs)が大きいと、負荷バランスが悪化
  - 例：N=10、numprocs=6
    - int(10/6)=1なので、
      プロセス0～4は1個のデータ、プロセス5は4個のデータを持つ
  - 各プロセスごとの開始値、終了値のリストを持てば改善可能
    - プロセス0： i_start(0)=1, i_end(0)=2, 2個
    - プロセス1： i_start(1)=3, i_end(1)=4, 2個
    - プロセス2： i_start(2)=5, i_end(2)=6, 2個
    - プロセス3： i_start(3)=7, i_end(3)=8, 2個
    - プロセス4： i_start(4)=9, i_end(4)=9, 1個
    - プロセス5： i_start(5)=10, i_end(5)=10, 1個
  - 欠点：プロセス数が多いと、上記リストのメモリ量が増える

模範解答

行列－ベクトル積のピュアMPI並列化の例（C言語）

```
ierr = MPI_Init(&argc, &argv);
ierr = MPI_Comm_rank(MPI_COMM_WORLD, &myid);
ierr = MPI_Comm_size(MPI_COMM_WORLD,
&numprocs);
…
ib = n/numprocs;
jstart = myid * ib;
jend = (myid+1) * ib;
if ( myid == numprocs-1)  jend=n;

for( j=jstart;  j<jend;  j++) {
    y[ j ] = 0.0;
    for(i=0; i<n; i++) {
      y[ j ] += A[ j ][ i ] * x[ i ];
     }
 }
```

ブロック分散を仮定した担当ループ範囲の定義

MPIプロセスの担当ごとに縮小したループの構成

行列－ベクトル積のピュアMPI並列化の例（Fortran言語）

```
call MPI_INIT(ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD, myid, ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD, numprocs,
ierr)
…
ib = n/numprocs
jstart = 1 + myid * ib
jend = (myid+1) * ib
if ( myid .eq. numprocs-1)  jend = n

do j = jstart,  jend
   y( j ) = 0.0d0
   do i=1, n
        y( j ) = y( j ) + A( j, i ) * x( i )
   enddo
enddo
```

ブロック分散を仮定した担当ループ範囲の定義

MPIプロセスの担当ごとに縮小したループの構成

レポート課題

1. [L10] 行列-ベクトル積において、列方式、および行方式の性能を比較し、考察せよ。なお、並列化する必要はない。
2. [L10] サンプルプログラムを並列化せよ。このとき、行列Aおよびベクトルx、yのデータは、全PEでN×Nのサイズを確保してよい。
3. [L15] サンプルプログラムを並列化せよ。このとき、行列Aおよびベクトルxは、初期状態では、各PEに割り当てられた分の領域しか確保してはいけない。
   （すなわち、逐次のメモリ量の 1/ 192とすること。ただし、並列化のための作業領域分は除く。）

問題のレベルに関する記述：
- L00: きわめて簡単な問題。
- L10: ちょっと考えればわかる問題。
- L20: 標準的な問題。
- L30: 数時間程度必要とする問題。
- L40: 数週間程度必要とする問題。複雑な実装を必要とする。
- L50: 数か月程度必要とする問題。未解決問題を含む。

※L40以上は、論文を出版するに値する問題。

レポート課題

4. [L20] サンプルプログラムを並列化したうえで、ピュアMPI実行、および、ハイブリッドMPI実行で性能が異なるか、実験環境（12ノード、192コア）を駆使して、性能評価せよ。
   - 1ノードあたり、12MPI実行、1MPI+16スレッド実行、2MPI+8スレッド実行、4MPI+4スレッド実行など、組み合わせが多くある。

来週へつづく

べき乗法