「情報システム学特別講義３」
べき乗法の並列化

東京大学情報基盤センター准教授　片桐孝洋

２０１４年６月１７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

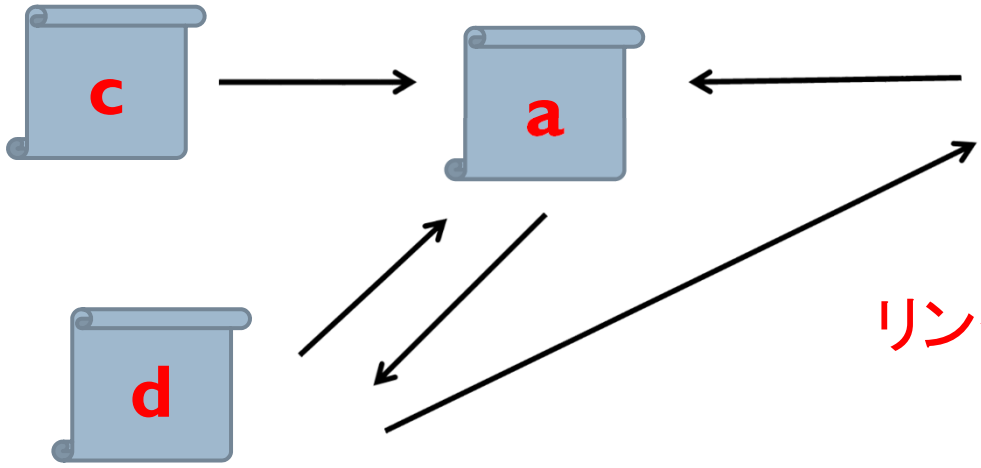
1. べき乗法とは
2. べき乗法のサンプルプログラムの実行
3. サンプルプログラムの説明
4. 並列化実習
5. レポート課題

べき乗法とは

簡単な数値アルゴリズム

べき乗法の応用例

- Googleにおけるページランキングのアルゴリズム
  [S.Brain and L.Page, 1998]
- ページ参照(リンク先)の関連を表す行列をAとする



リンク先

リンク元

$$A = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

接続行列A
リンクがあるとき：要素1
リンクがないとき；要素0

推移確率行列Mの作成

- 接続行列Aを転置した行列を作る
- 各列の1の要素数を数え、それで各要素を割る
- 上記の行列を、推移確率行列Mと定義する

$$M = \begin{pmatrix} 0 & 1 & 1 & 1/2 \\ 0 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

固有値問題への帰着

- このとき、以下の標準固有値問題を考える

$$Mx = \lambda x \quad \cdots\cdots \text{式}(1)$$

- 上記の固有値問題の固有値、固有ベクトルは、
  M が非対称・実数行列のため、一般に、
  - 固有値は複素固有値
  - 固有ベクトルは複素固有ベクトル

  になる
- ただし、係数行列Mの特性から
  - 最大固有値は実数で、かつ、 $\lambda = 1$

確率推移行列Mの最大固有値に付随する最大固有ベクトルの意味

- 最大固有値 $\lambda = 1$ に付随する固有ベクトル x を正規化（ベクトルx の内積値で各要素を割ったもの）したベクトルの要素は、各ページのランク値になる
- 上記が、Googleにおけるページランク
  - 高いほど先に表示されるページ
  - ランダム・サーファーモデルにおける各ページの到達確率
    - 確率の高いページはリンクされている度合いが高い（権威のあるページ）

$$x = \begin{pmatrix} \alpha \\ \beta \\ \chi \\ \delta \end{pmatrix}$$

←ページaのページランク値
←ページbのページランク値
←ページcのページランク値
←ページdのページランク値

収束の加速

- 式(1)を、後述のべき乗法でそのまま解く場合、
  収束性が悪い(収束するまでの反復数が多くなる)
- そこで、収束の加速として、以下の
  - ダンピング・ファクターd

を導入し、行列Aの要素を広範囲に広げる行列G を作り、
収束の加速をしている

- G はGoogle行列と呼ばれる

$$G = dA + (1 - d)\frac{1}{n}\tilde{1} \quad \cdots\cdots式(2)$$

ここで、行列Aはn x n行列。

$\tilde{1}$ はn x n行列で、要素がすべて1の行列。

Google行列Gの意味

- 式（2）において
  - $d = 1$ なら、もとの行列 A　そのもの
  - $d = 0$ なら、各要素が 1/nの行列
    ←全てのページが、全てのページにリンクしている

以上から

- d が1に近いと、元の行列のリンク状況に近くなる
- d が0に近いと、実際にはリンクされていないが、全てのページに、均等にリンクされている状況に近くなる

Google行列Gに対する“べき乗法”の収束性

- dが0に近づくにつれて、収束が良くなる。

  ←理由は、行列Gの数値特性（固有値分布）が、
  べき乗法に向くように改善されるため。

- d=0.85が、経験的に良いとされている[1]
  - 理由？
- また、反復回数は多くしなくても、実用上の精度は問題ないといわれている。
  - 50回～100回の反復でよい[1]
- 注意：べき乗法に限らず、Jacobi法、Arnordi法などでも解ける

[1] David Austin (Grand Valley State University) :How Google Finds Your Needle in the Web's Haystack
http://www.ams.org/samplings/feature-column/fcarc-pagerank

Google行列に対するべき乗法と、本演習でのべき乗法の対象の違い

- Google行列に対するべき乗法
  - 行列は非対称、実数行列
  - 疎行列（0要素が多い行列）
    - 疎行列の特性用いて、演算量を$O(n)$に低下させる実装
  - 固有値、固有ベクトルは複素数
  - 最大固有値は1で実数固有値、実数固有ベクトルなので、最大固有値に付属する固有ベクトル計算は実数計算で可能
- ここでのべき乗法
  - 行列は対称、実数行列
  - 密行列（0要素が無い）
    - 計算量は$O(n^2)$
  - 固有値、固有ベクトルは実数

べき乗法とは

- べき乗法は、標準固有値問題の＜最大固有値＞と、それに付随する＜固有ベクトル＞を計算できます。
  - 標準固有値問題: $Ax = \lambda x$
  - 固有値: λ　固有ベクトル: x
- いま、行列Aを$n \times n$の正方行列とします。
- 行列Aの固有値を、絶対値の大きい方から整列し、かつ重複していないものを $\lambda_1, \lambda_2, \cdots, \lambda_n$ とします。
- （正規直交な）ベクトルを $x_1, x_2, \cdots, x_n$ とします。
- このとき、任意のベクトルは、以下の線形結合で表わされます。

$$u = c_1 x_1 + c_2 x_2 + ,..., + c_n x_n$$

べき乗法とは

- Aを左辺に作用させると

$$Au = A(c_1 x_1 + c_2 x_2 +,\ldots,+ c_n x_n)$$

- 標準固有値問題の等式を考慮すると

$$Au = c_1 \lambda_1 x_1 + c_2 \lambda_2 x_2 +,\ldots,+ c_n \lambda_n x_n$$

$$= \lambda_1 \left[c_1 x_1 + c_2 \frac{\lambda_2}{\lambda_1} x_2 +,\ldots,+ c_n \frac{\lambda_n}{\lambda_1} x_n \right]$$

べき乗法とは

- Auの積を、n回行うと

$$A^k u = \lambda_1^{\,k}\left[c_1 x_1 + c_2\left[\frac{\lambda_2}{\lambda_1}\right]^k x_2 + ,\ldots, + c_n\left[\frac{\lambda_n}{\lambda_1}\right]^k x_n\right]$$

- すなわち、kが増えていくと、段々　x_1 以外のベクトルの係数が小さくなっていく。
  →最大固有値、および、それに付随する固有ベクトルに収束する

べき乗法とは

- 内積を(x, y)と記載する。このとき、以下の計算を考える。

$$\frac{(A^{k+1}u,\ A^{k+1}u)}{(A^{k+1}u,\ A^{k}u)} = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j \lambda_i{}^{k+1} \lambda_j{}^{k+1} (x_i, x_j)}{\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j \lambda_i{}^{k+1} \lambda_j{}^{k} (x_i, x_j)}$$

$$= \frac{\lambda_1{}^{2k+2}\left[c_1{}^2 |x_1|^2 + \sum_{i=2}^{n} c_i{}^2 \left[\frac{\lambda_i}{\lambda_1}\right]^{2k+2} |x_i|^2 \right]}{\lambda_1{}^{2k+1}\left[c_1{}^2 |x_1|^2 + \sum_{i=2}^{n} c_i{}^2 \left[\frac{\lambda_i}{\lambda_1}\right]^{2k+1} |x_i|^2 \right]} \approx \lambda_1 \quad (k \to \infty)$$

べき乗法のアルゴリズム

- 以下の手順を、収束するまで行う

1. 適当なベクトルxを作り、正規化;
2. λ_0 = 0.0;　i=1;
3. 行列積 y = A x ;
4. 近似固有値 λ_i ＝ (y, y) / (y, x) を計算;
5. |λ_i - λ_{i-1}| が十分小さいとき:
   - 収束したとみなし終了;
6. そうでないなら:
   - x を正規化して x = y;
   - i = i +1; 3.へ戻る;

サンプルプログラムの実行（べき乗法）

はじめての数値アルゴリズムの並列化

行列-ベクトル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

  PowM-fx.tar

- ジョブスクリプトファイルpown.bash
  - lecture : 実習時間外のキュー

べき乗法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/PowM-fx.tar ./
$ tar xvf PowM-fx.tar
$ cd PowM
```

- 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 powm.bash
```

- 実行が終了したら、以下を実行する

```
$ cat powm.bash.oXXXXXX
```

べき乗法のサンプルプログラムの実行（C言語）

- 以下のような結果が見えれば成功

```
N  = 4000
Power Method time  = 0.472348 [sec.]
Eigenvalue  = 2.000342e+03
Iteration Number: 7
Residual 2-Norm ||A x - lambda x||_2  = 7.656578e-09
```

べき乗法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

```
N = 4000
Power Method time[sec.] = 0.3213765330146998
Eigenvalue = 2000.306721217447
Iteration Number: 6
Residual 2-Norm ||A x - lambda x||_2 =
  4.681124813641846E-07
```

サンプルプログラムの説明

- #define N 4000
  の、数字を変更すると、行列サイズが変更できます
- PowM関数の仕様
  - 戻り値は、最大固有値（Double型）
  - Double型の配列xに、最大固有値に付随する固有ベクトルが格納される。
  - 引数n_iterに収束するまでの反復回数が入る。
    - “－1”が戻る場合、反復回数の上限MAX_ITERまでに収束しなかったことを意味する。

Fortran言語のサンプルプログラムの注意

- 行列サイズNN、およびMAX_ITERの宣言は、以下のファイルにあります。

  pown.inc

- 行列サイズ変数が、NNとなっています。

```
integer  NN
parameter (NN=4000)
```

サンプルプログラムの概略（PowM関数内）

```
/* Normizeation of x */
  d_tmp1 = 0.0;
  for(i=0; i<n; i++) {
    d_tmp1 += x[i] * x[i];
  }
  d_tmp1 = 1.0 / sqrt(d_tmp1);
  for(i=0; i<n; i++) {
    x[i] = x[i] * d_tmp1;
  }

  /* Main iteration loop ---------------------- */
  for(i_loop=1; i_loop<MAX_ITER; i_loop++)  {

    /* Matrix Vector Product */
    MyMatVec(y, A, x, n);

    /* innner products */
      d_tmp1 = 0.0;
      d_tmp2 = 0.0;
      for (i=0; i<n; i++) {
      d_tmp1 += y[i] * y[i];
      d_tmp2 += y[i] * x[i];
    }

    /* current approximately eigenvalue */
    dlambda = d_tmp1 / d_tmp2;

      /* Convergence test*/
      if (fabs(d_before-dlambda) < EPS ) {
        *n_iter = i_loop;
        return dlambda;
      }

    /* keep current value */
    d_before = dlambda;

/* Normalization and set new x */
    d_tmp1 = 1.0 / sqrt(d_tmp1);
    for(i=0; i<n; i++)
      x[i] = y[i] * d_tmp1;

  }  /* end of i_loop ------------------------- */
```

ベクトルx
正規化部分

行列-ベクトル積
部分

行列xとyの
内積部分

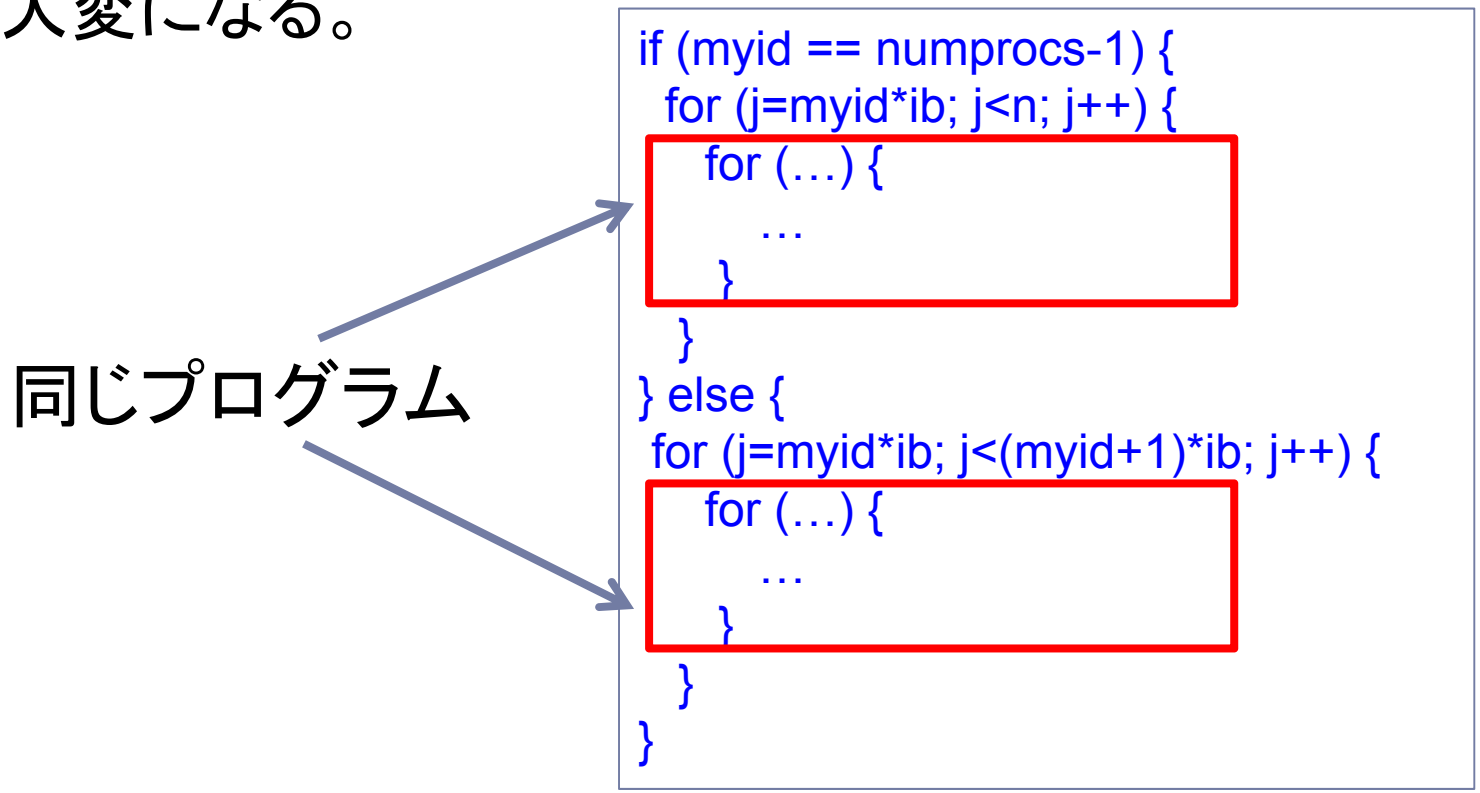
正規化と
新しいx
設定部分

演習課題

- PowM関数（手続き）を並列化してください。
  - デバック時は、#define N 192 としてください。
  - 前回演習の並列行列-ベクトル積ルーチン を利用してください。
- サンプルプログラムでは、残差ベクトルAx-λxの2-ノルムを計算しています。デバックに活用してください。
  - つまり、この値が十分小さくないとバグっています。
  - 固有ベクトルxの分散方式により、残差ベクトル計算部分の並列化が必要になります。注意してください。
- 並列化すると、反復回数（＝実行時間）や残差の2ノルム値が変化することがあります。

並列化の注意

- 以下のようなプログラムを書くと、美しくない＆コードマネージが大変になる。



- 並列化の対象のループは1つにし、ループ制御変数を工夫し、並列化するように心がける。

並列化のヒント

- 前回示した方針のように、すべてのPEで重複して、行列AをN×N、ベクトルx、yをNのサイズで確保すると、実装が簡単です。
- 以下の分散方式を仮定します。
  (先週の行列-ベクトル積の演習と同じ)
  - 行列A:
    1次元行方向ブロック分割方式
  - ベクトルx:
    全PEで、N次元ベクトルを重複所有
  - ベクトルy:
    ブロック分割方式

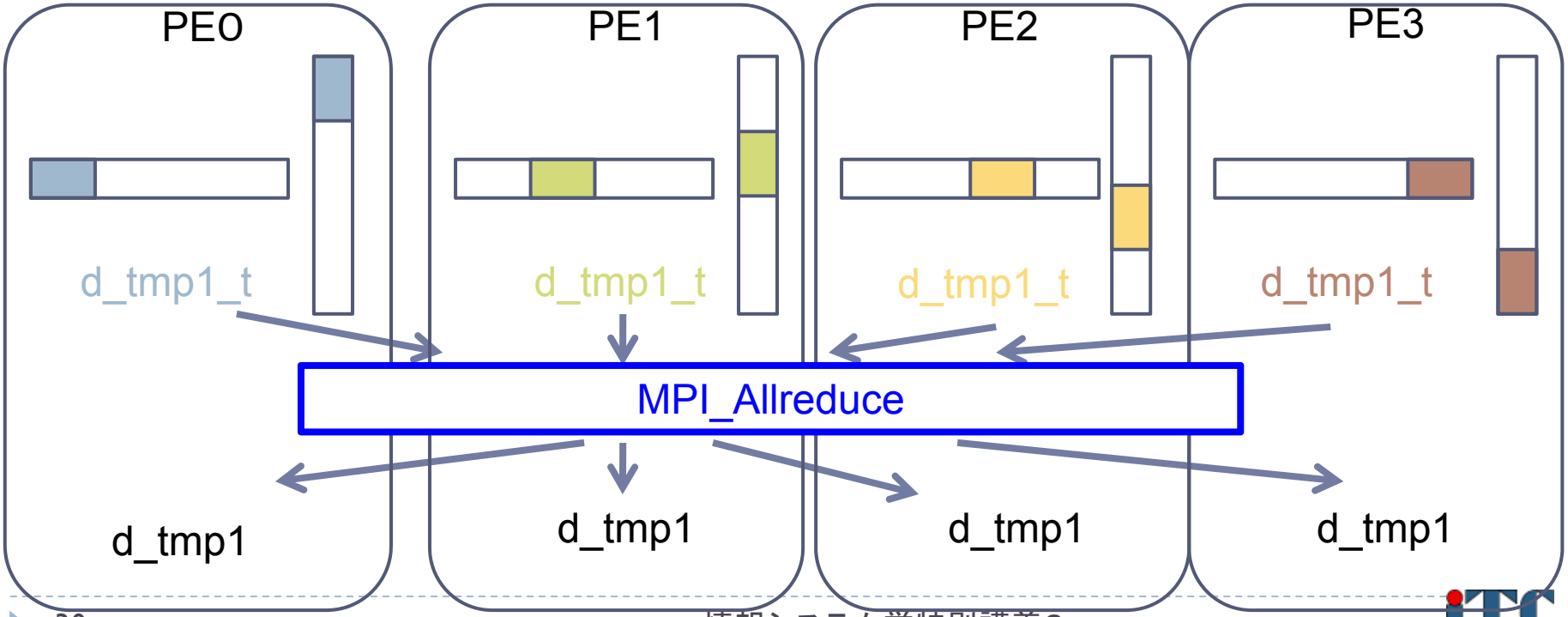
並列化のヒント（1.「行列-ベクトル積」のみ）

- 以下の2通りの＜並列化方針＞があります
  - 方法1：「行列-ベクトル積」 のみ並列化
  - 方法2： すべてを並列化
- 最も簡単な方法は方法1。以下はその手順：

1. 開発した「並列行列-ベクトル積」コードを使う
2. y = Ax の y が分散されて戻るため、以降の計算が逐次の結果と合わない。逐次結果と一致させるため、
   - MPI関数を PowM関数中の MyMatVec() が 呼ばれる直後に入れて、分散されたyの要素すべてを収集する。
   - 最も簡単な実装は、MPI_Allreduce() を用いる実装
3. MPI_Allreduce()を利用するため、配列の初期化（ゼロクリア）を実装する。（後述の方式）

並列化のヒント（２.すべてを並列化時）

- PowM関数中の処理を、以下の方針で並列化
  1. ベクトルxの正規化部分
     - ① ブロック分割の内積計算を計算後、MPI_Allreduce関数（下図）を呼ぶ
     - ② MPI_Allreduce関数を使って、部分的に計算された計算結果を、全PEが全ベクトル要素を所有するようにする（後述）



並列化のヒント（２.すべてを並列化時）

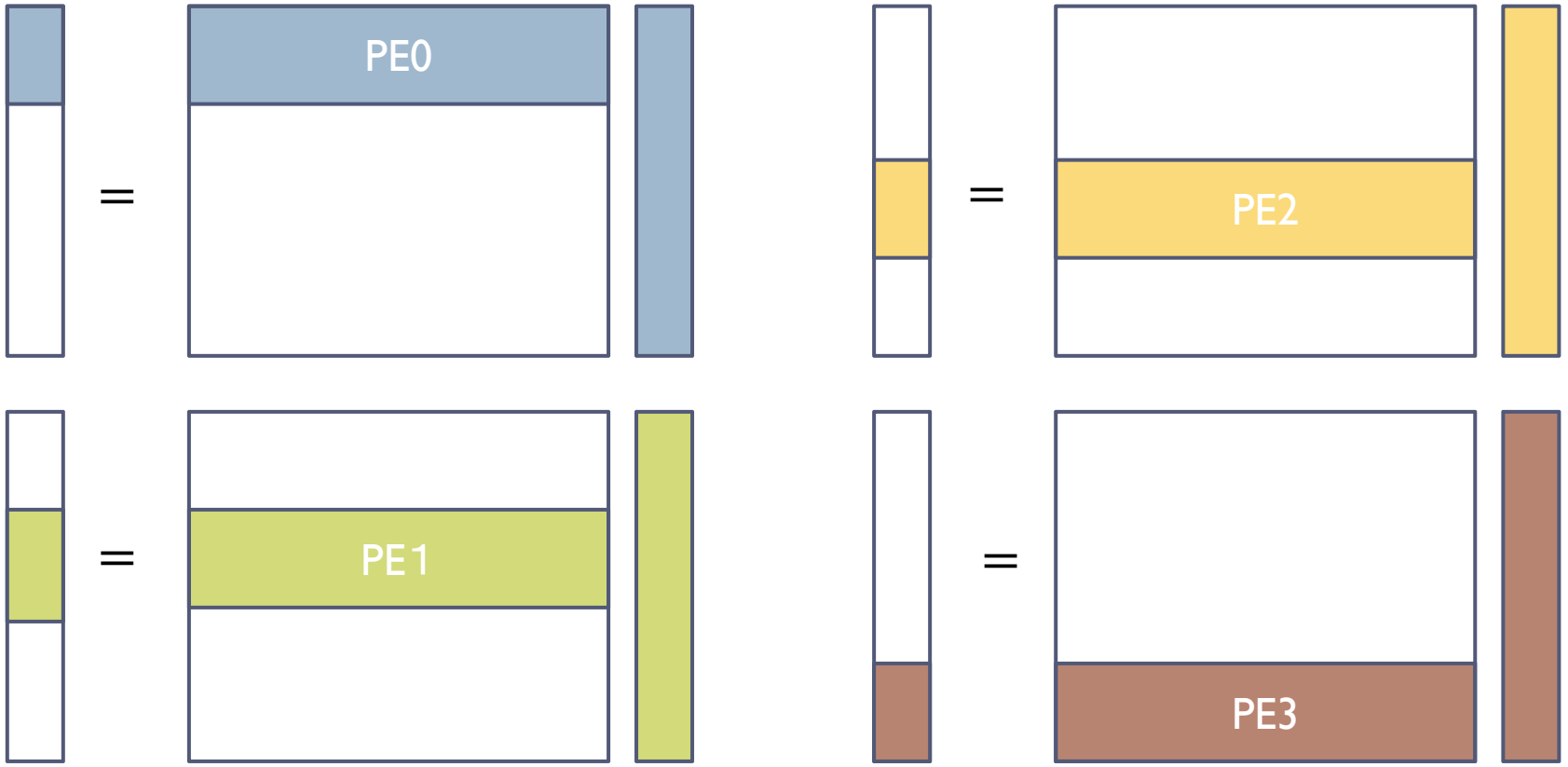
- 以下のようなプログラムになる

```
/* Normizeation of x */
…
  d_tmp1_t = 0.0;
  for(i=myid*ib; i<i_end; i++) {
    d_tmp1_t += x[i] * x[i];
  }
  MPI_Allreduce(&d_tmp1_t,  &d_tmp1, 1, MPI_DOUBLE,
     MPI_SUM, MPI_COMM_WORLD);

  d_tmp1 = 1.0 / sqrt(d_tmp1);
  for(i=myid*ib; i<i_end; i++) {
    x_t[i] = x[i] * d_tmp1;
  }
  （x_t[ ]は、ゼロに初期化をしているか要確認）
 MPI_Allreduce(x_t, x, n, MPI_DOUBLE, MPI_SUM,
     MPI_COMM_WORLD);
 ….
```

並列化のヒント（方法１および方法２）

2. 行列-ベクトル積部分（MyMatVec関数）
   - 前回演習の並列ルーチンを使う



並列化のヒント（方法１および方法２）

3. ベクトルx とyの内積部分
   - ブロック分散されているとして計算する
   - 正しい内積結果を得るため、MPI_Allreduce関数を使うことを忘れずに

並列化のヒント（２.すべてを並列化時）

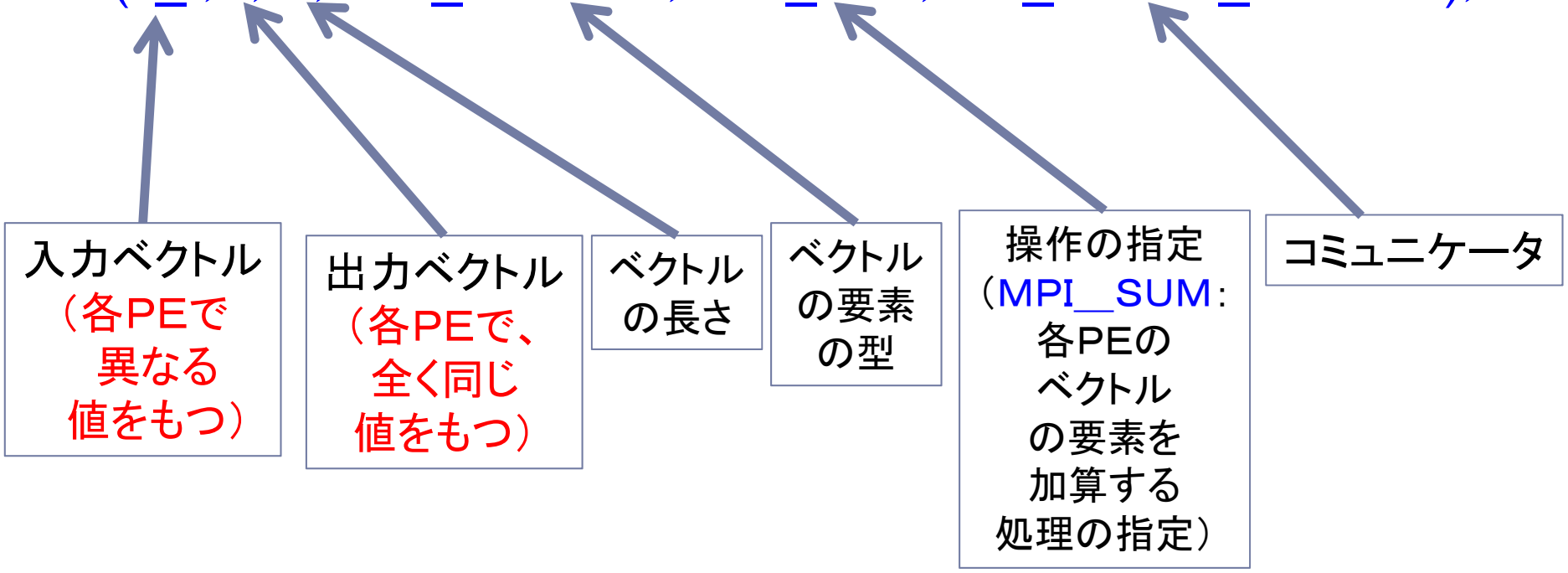
4. 正規化と新しいxの設定部分
   - x：全PEで同じN次元ベクトルを所有； y:ブロック分散
   - 正規化はブロック分散部分のみを行い、xに結果を代入
   - xは、各PEで計算結果が分散されている。
     （xは計算結果がブロック分散）
   - xは、全PEで全要素を重複して所有していないと、次の並列行列-ベクトル積が実行できない。
   - 各PEに分散されているベクトルxのデータを集めるため、MPI_Allreduce 関数を使って集める（後述）。
     - MPI_Allreduce関数を使うため、xの計算結果部分以外に0を代入したバッファx_tを用意。

```
MPI_Allreduce( x_t,  x,  n,  MPI_DOUBLE, MPI_SUM,
                              MPI_COMM_WORLD );
```

MPI_Allreduce関数の復習（C言語）

- **MPI_Allreduce**

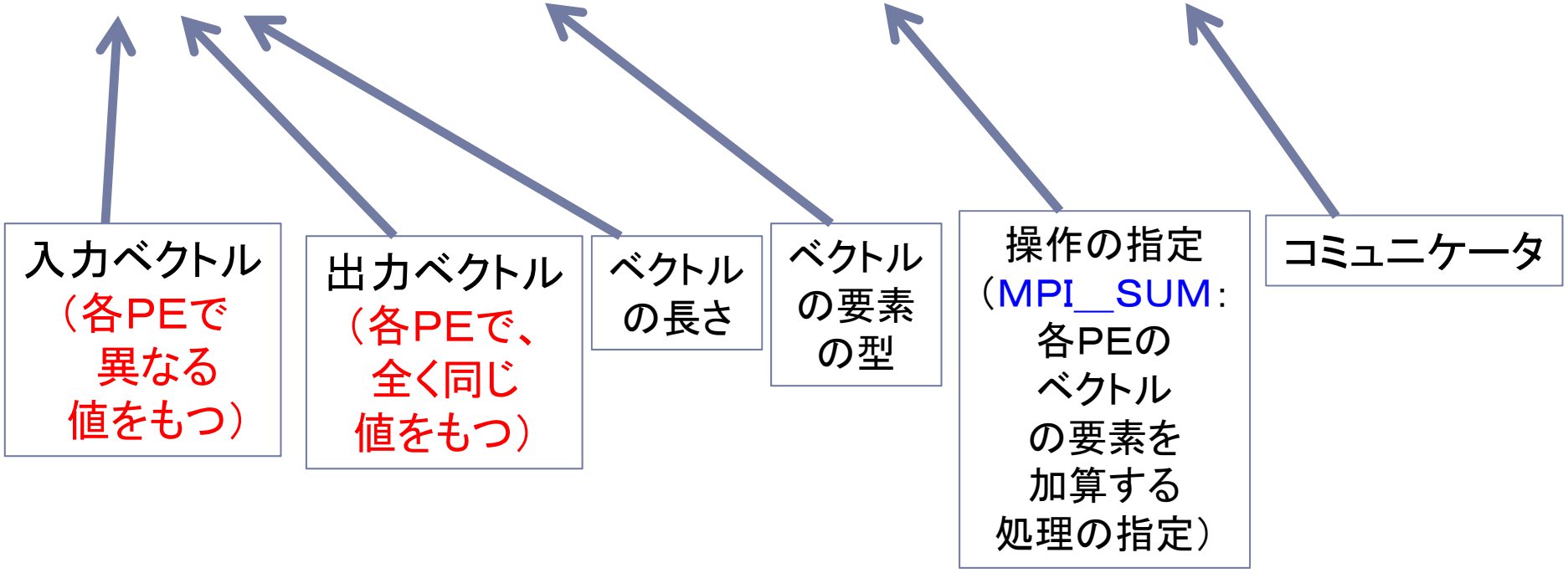
```
(x_t, x,  n,  MPI_DOUBLE,  MPI_SUM, MPI_COMM_WORLD);
```



MPI_Allreduce関数の復習（Fortran言語）

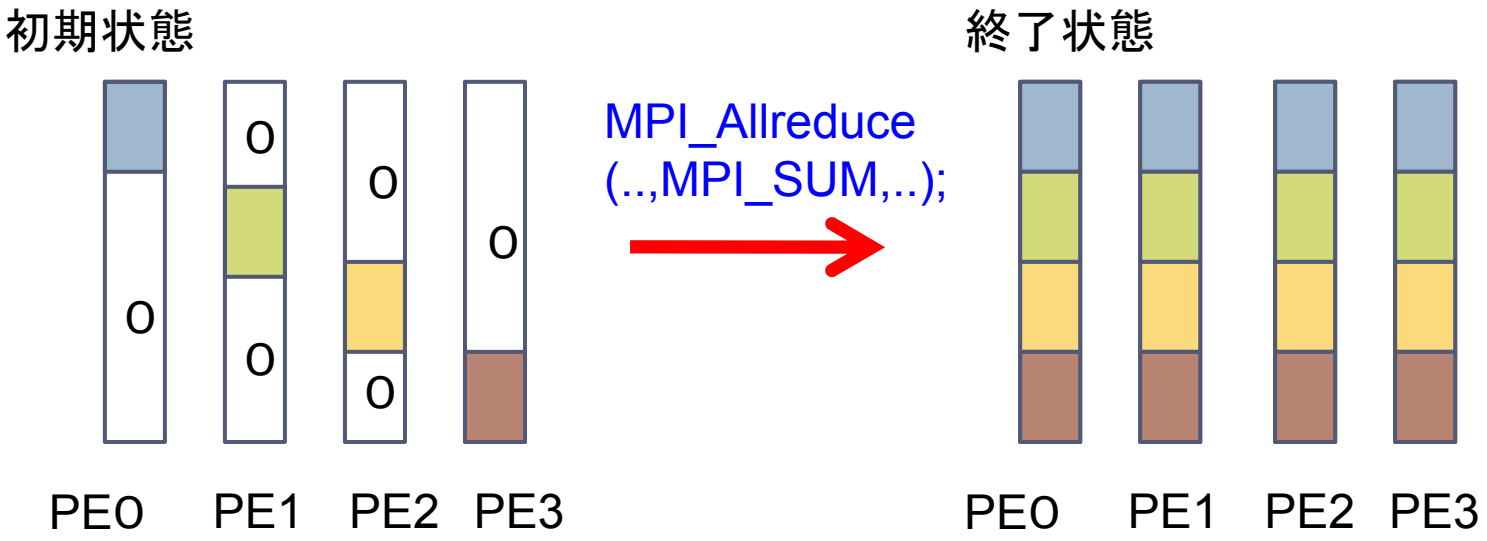
- **MPI_ALLREDUCE**

(x_t, x, n, MPI_DOUBLE_PRECISON, MPI_SUM, MPI_COMM_WORLD, ierr)



MPIの技(MPI_Allreduceでベクトル収集)

- MPI_Allreduce関数で、各PEに分散されたデータを収集し、全PEに演算結果を＜重複して＞所有させる方法
  - iop に、MPI_SUM を指定する
  - 自分が所有するデータ以外の箇所は、0に初期化されている。
  - そのうえで、以下のような処理を考える



※MPI_Gather関数を使っても実装できます。

実習の手順

- はじめに、簡単な
  方法１:「行列-ベクトル積」のみ並列化
  を先に行ってください。

- それが終わったら、
  方法2:すべてを並列化
  を行ってください。

模範解答

●方法1：「行列-ベクトル積」 のみ並列化の解答例

```
double y_t[N];
…
for(i=0; i<n; i++) y_t[i] = 0.0d0;
for(i_loop=1; i_loop<=MAX_ITER; i_loop++)  {

    /* Matrix Vector Product */
    MyMatVec(y_t, A, x, n);

    MPI_Allreduce (y_t, y,  n,  MPI_DOUBLE,
          MPI_SUM, MPI_COMM_WORLD);
    …
     あとはそのまま
```

←分散データ用のベクトルの確保

←計算しない部分に0を代入

←並列化した行列-ベクトル積の結果をy_tに収納

←分散したデータをyに収集

レポート課題

1. [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、yのデータは、全PEでN×Nのサイズを確保してよい。なお、いろいろな問題サイズ（Nの大きさ）について性能評価し、その結果の考察を行え。
2. [L20] サンプルプログラムを並列化し、性能評価と考察を行え。このとき、行列Aおよびベクトルx、y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。また、1と同様な性能評価と考察を行え。特に、1. と2. で実行時間の違いはあるか評価・考察せよ。

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（続き）

4. [L5] コンパイラによる最適化により、実行時間がどのように変化するか調査せよ。コンパイラの最適化方式により、反復回数が変化することがある。そこで、１反復あたりの実行時間を計算した上で、性能評価を行い考察せよ。
5. [L20] サンプルプログラムの並列化プログラムについて、通信処理をノンブロッキングにするなどの改良して高速化を行え。いろいろな問題サイズについて性能評価を行い、高速化する前のプログラムに対して考察をせよ。
6. [L10～L20] 並列化したプログラムに対し、ピュアMPI実行、および、ハイブリッドMPI実行を行い、演習環境を駆使して、性能評価を行え。（L10）
また、どういう条件でピュアMPI実行が高速になるか、条件を導出し、性能評価結果と検証を行え。（L20）

来週へつづく

行列-行列積の並列化