行列-行列積（１）

東京大学情報基盤センター　准教授　片桐孝洋

２０１５年６月９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）

行列 - 行列積の演習の流れ

- 演習課題（１）
  - 本日行う
  - 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）
  - 来週行う
  - 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

行列-行列積とは

実装により性能向上が見込める基本演算

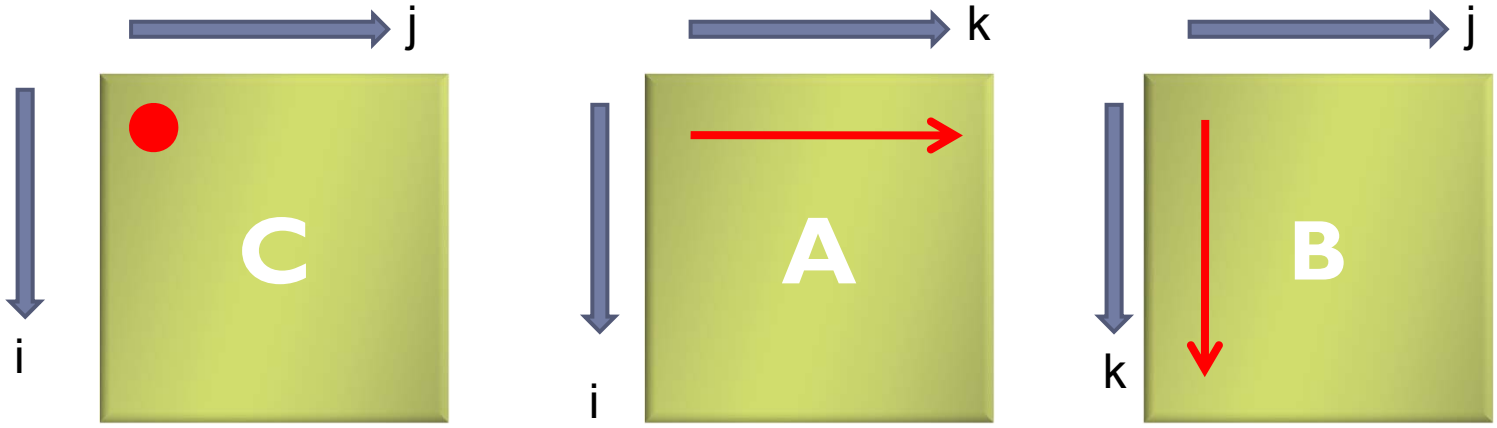
１.５　行列の積

- 行列積　Ｃ＝Ａ・Ｂ は、コンパイラや計算機のベンチマークに使われることが多い
  - 理由１：実装方式の違いで性能に大きな差がでる
  - 理由２：手ごろな問題である（プログラムし易い）
  - 理由３：科学技術計算の特徴がよく出ている
    1. 非常に長い＜連続アクセス＞がある
    2. キャッシュに乗り切らない＜大規模なデータ＞に対する演算である
    3. メモリバンド幅を食う演算（メモリ・インテンシブ）な処理である

行列積コード例（C言語）

- コード例

```
for (i=0; i<n;  i++)
  for (j=0; j<n; j++)
    for (k=0; k<n; k++)
      C[i][j] += A[i][k] *B[k][j];
```



1.5　行列の積

- 行列積　$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \ (i, j = 1, 2, \ldots, n)$

  の実装法は、次の二通りが知られている：

1. ループ交換法
   - 連続アクセスの方向を変える目的で、行列-行列積を実現する3重ループの順番を交換する
2. ブロック化（タイリング）法
   - キャッシュにあるデータを再利用する目的で、あるまとまった行列の部分データを、何度もアクセスするように実装する

1.5　行列の積（C言語）

- ループ交換法
  - 行列積のコードは、以下のような3重ループになる

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

  - 最内部の演算は、外側の3ループを交換しても、計算結果が変わらない
    → 6通りの実現の方法がある

1.5　行列の積（Fortran言語）

- ループ交換法
  - 行列積のコードは、以下のような3重ループになる

```
do i=1, n
  do j=1, n
    do k=1, n
      c( i , j ) = c( i, j) + a( i ,k ) * b( k , j )
    enddo
  enddo
enddo
```

  - 最内部の演算は、外側の3ループを交換しても、計算結果が変わらない
    → 6通りの実現の方法がある

東京大学情報基盤センター
INFORMATION TECHNOLOGY CENTER, THE UNIVERSITY OF TOKYO

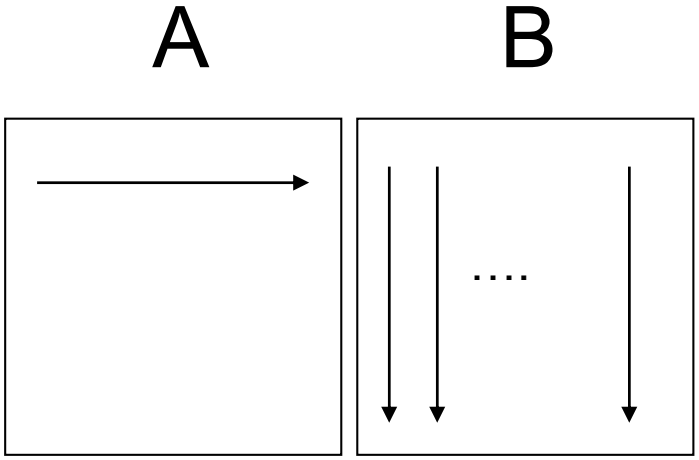
1.5　行列の積

- 行列データへのアクセスパターンから、
  以下の3種類に分類できる
  1. 内積形式 (inner-product form)
     最内ループのアクセスパタンが
     ＜ベクトルの内積＞と同等
  2. 外積形式 (outer-product form)
     最内ループのアクセスパタンが
     ＜ベクトルの外積＞と同等
  3. 中間積形式 (middle-product form)
     内積と外積の中間

1.5　行列の積（C言語）

- 内積形式 (inner-product form)
  - ijk, jikループによる実現

```
for (i=0; i<n; i++) {
  for (j=0; j<n; j++) {
    dc = 0.0;
    for (k=0; k<n; k++){
      dc = dc + A[ i ][ k ] * B[ k ][ j ];
    }
    C[ i ][ j ]= dc;
  }
}
```



●行方向と列方向のアクセスあり
→行方向・列方向格納言語の
両方で性能低下要因
解決法：
A, Bどちらか一方を転置しておく

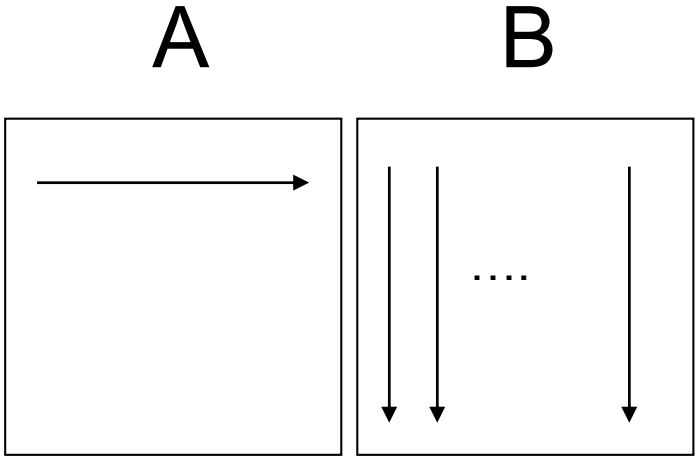
※以降、最外のループからの変数の順番で実装法を呼ぶ。たとえば上記のコードは＜ijkループ＞。

1.5　行列の積（Fortran言語）

- 内積形式 (inner-product form)
  - ijk, jikループによる実現

```
do i=1, n
  do j=1, n
    dc = 0.0d0
    do k=1, n
      dc = dc + A( i , k ) * B( k , j )
    enddo
    C( i , j ) = dc
  enddo
enddo
```

※以降、最外のループからの変数の順番で実装法を呼ぶ。たとえば上記のコードは＜ijkループ＞。

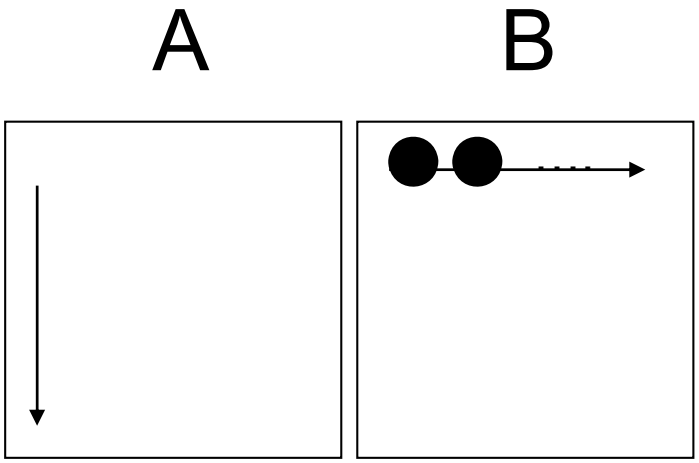


●行方向と列方向のアクセスあり
→行方向・列方向格納言語の
両方で性能低下要因
解決法：
A, Bどちらか一方を転置しておく

1.5　行列の積（C言語）

- 外積形式 (outer-product form)
  - kij, kjiループによる実現

```
for (i=0; i<n; i++) {
  for (j=0; j<n; j++) {
    C[ i ][ j ] = 0.0;
  }
}
for (k=0; k<n; k++) {
  for (j=0; j<n; j++) {
    db = B[ k ][ j ];
    for (i=0; i<n; i++) {
      C[ i ][ j ]= C[ i ][ j ]+ A[ i ][ k ]* db;
    }
  }
}
```

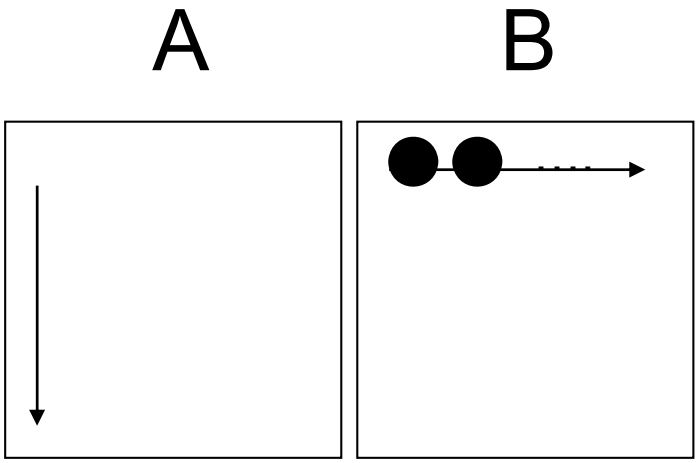


●kjiループでは
列方向アクセスがメイン
→列方向格納言語向き
（Fortran言語）

1.5　行列の積（Fortran言語）

- 外積形式 (outer-product form)
  - kij, kjiループによる実現

```
do i=1, n
  do j=1, n
    C( i , j ) = 0.0d0
  enddo
enddo
do k=1, n
  do j=1, n
    db = B( k , j )
    do i=1, n
      C( i , j ) = C( i , j )+ A( i , k ) * db
    enddo
  enddo
enddo
```



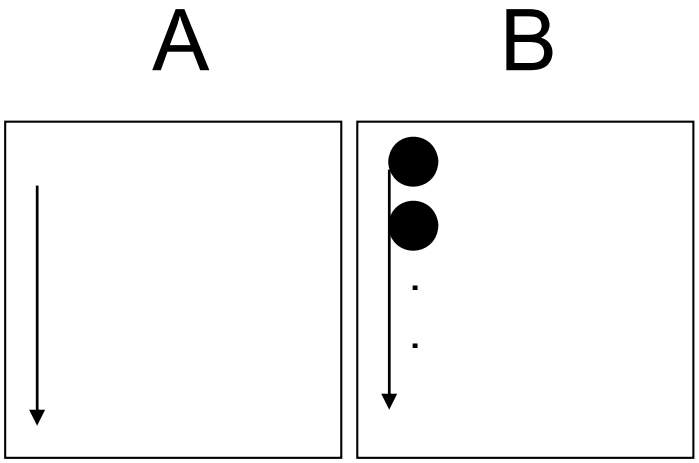
●kjiループでは
列方向アクセスがメイン
→列方向格納言語向き
（Fortran言語）

東京大学情報基盤センター
INFORMATION TECHNOLOGY CENTER, THE UNIVERSITY OF TOKYO

1.5　行列の積（C言語）

- 中間積形式 (middle-product form)
  - ikj, jkiループによる実現

```
for (j=0; j<n; j++) {
  for (i=0; i<n; i++) {
    C[ i ][ j ] = 0.0;
  }
  for (k=0; k<n; k++) {
    db = B[ k ][ j ];
    for (i=0; i<n; i++) {
      C[ i ][ j ] = C[ i ][ j ] + A[ i ][ k ] * db;
    }
  }
}
```



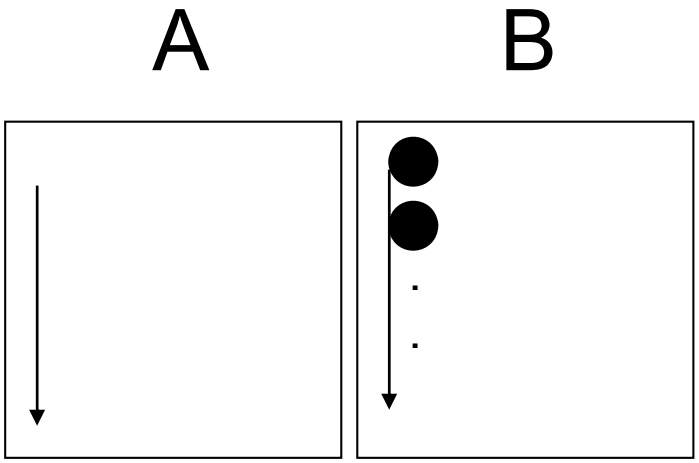
●jkiループでは
全て列方向アクセス
→列方向格納言語に
最も向いている
（Fortran言語）

東京大学情報基盤センター
INFORMATION TECHNOLOGY CENTER, THE UNIVERSITY OF TOKYO

1.5　行列の積（Fortran言語）

- 中間積形式 (middle-product form)
  - ikj, jkiループによる実現

```
do j=1, n
  do i=1, n
    C( i , j ) = 0.0d0
  enddo
  do k=1, n
    db = B( k , j )
    do i=1, n
      C( i , j ) = C( i , j ) + A( i , k ) * db
    enddo
  enddo
enddo
```



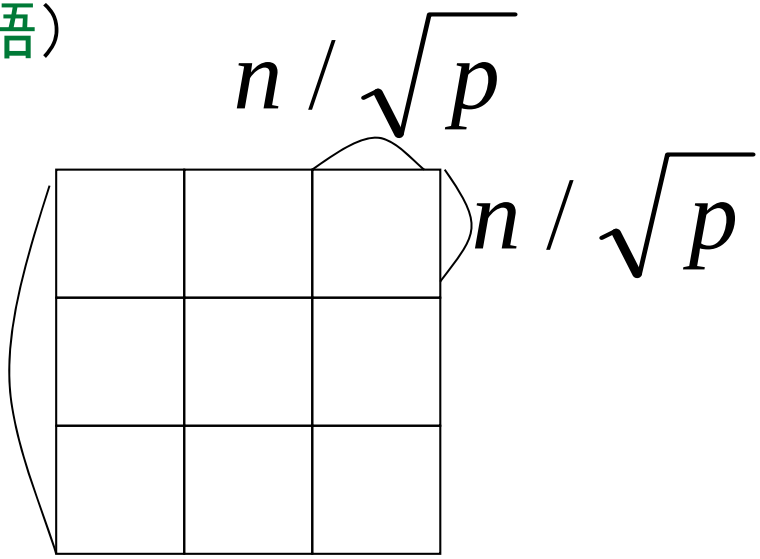
●jkiループでは
全て列方向アクセス
→列方向格納言語に
最も向いている
（Fortran言語）

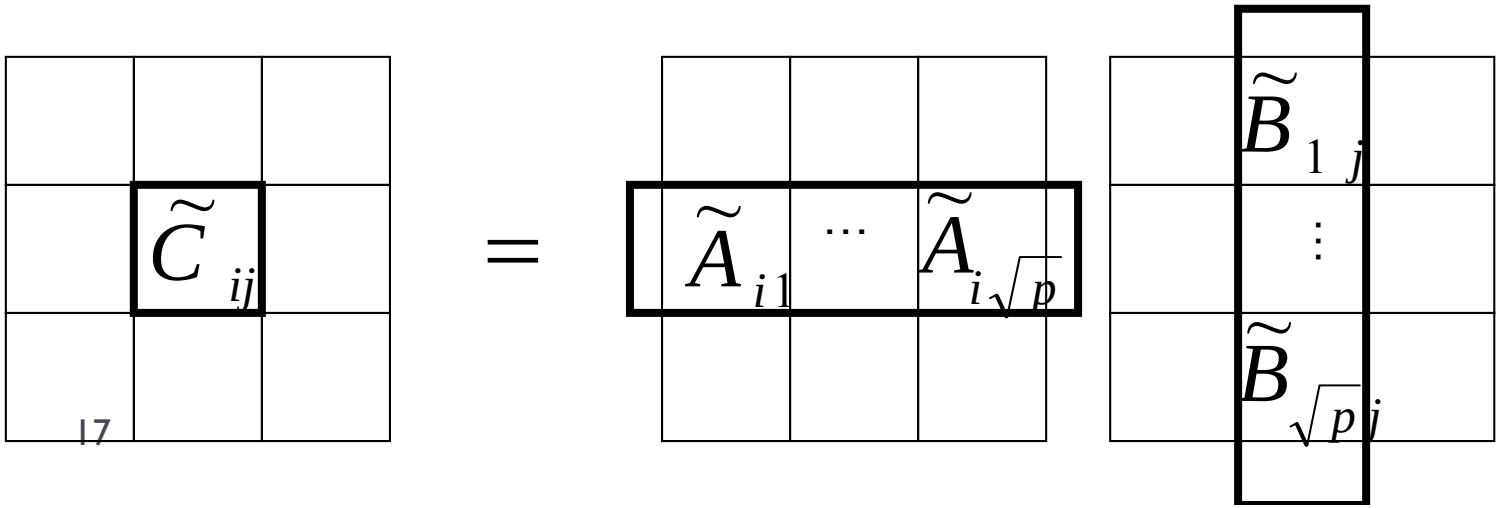
東京大学情報基盤センター
INFORMATION TECHNOLOGY CENTER, THE UNIVERSITY OF TOKYO

1.5　行列の積

- 小行列ごとの計算に分けて（配列を用意し）計算
  （ブロック化、タイリング：　コンパイラ用語）
- 以下の計算

$$\tilde{C}_{ij} = \sum_{k=1}^{\sqrt{n}} \tilde{A}_{ik} \tilde{B}_{kj}$$



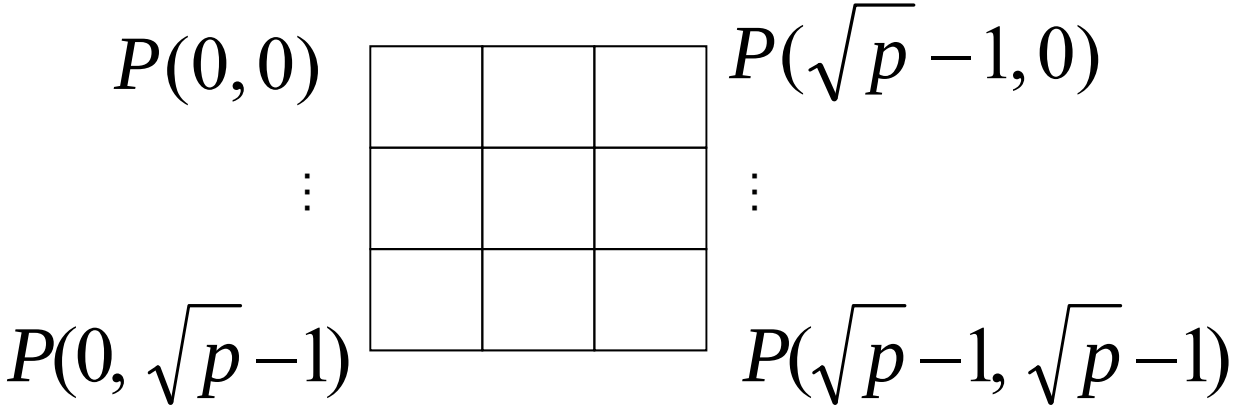


1.5　行列の積

- 各小行列をキャッシュに収まるサイズにする。
  1. ブロック単位で高速な演算が行える
  2. 並列アルゴリズムの変種が構築できる
- 並列行列積アルゴリズムは、データ通信の形態から、以下の2種に分類可能：
  1. セミ・シストリック方式
     - 行列A、Bの小行列の一部をデータ移動
       （Cannonのアルゴリズム）
  2. フル・シストリック方式
     - 行列A、Bの小行列のすべてをデータ移動
       （Foxのアルゴリズム）

1.5.1 Cannonのアルゴリズム

- データ分散方式の仮定
  - プロセッサ・グリッドは　二次元正方



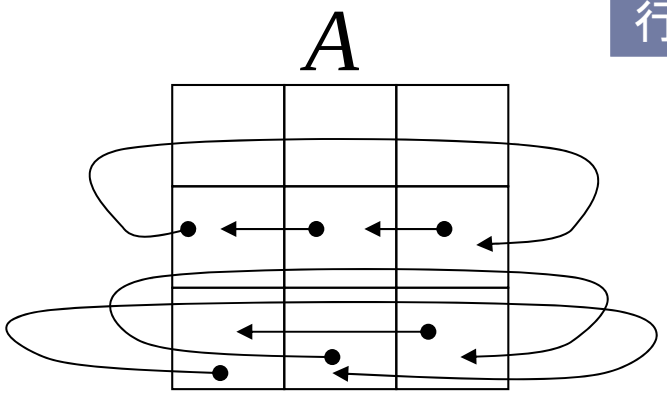
  - PE数が、2のべき乗数しかとれない
  - 各PEは、行列A、B、Cの対応する各小行列を、1つづつ所有
  - 行列A、Bの小行列と同じ大きさの作業領域を所有

言葉の定義－放送と通信

- 通信
  - ＜通信＞とは、1つのメッセージを1つのPEに送ることである
  - MPI_Send関数、MPI_Recv関数で記述できる処理のこと
  - 1対1通信ともいう
- 放送
  - ＜放送＞とは、同一メッセージを複数のPEに（同時に）通信することである
  - MPI_Bcast関数で記述できる処理のこと
  - 1対多通信ともいう
  - 通信の特殊な場合と考えられる

1.5.1　Cannonのアルゴリズム

- アルゴリズムの概略
  - 第一ステップ



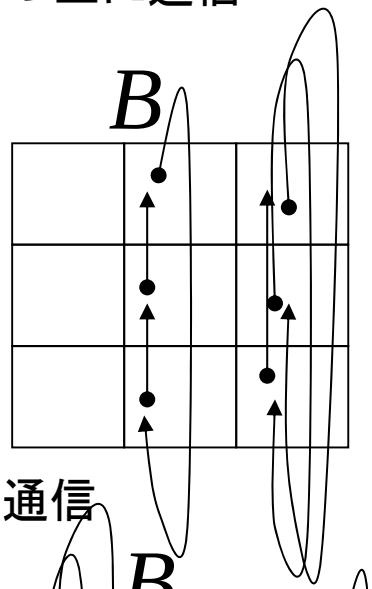
ローカルな
行列-行列積の後

:1つ右に通信

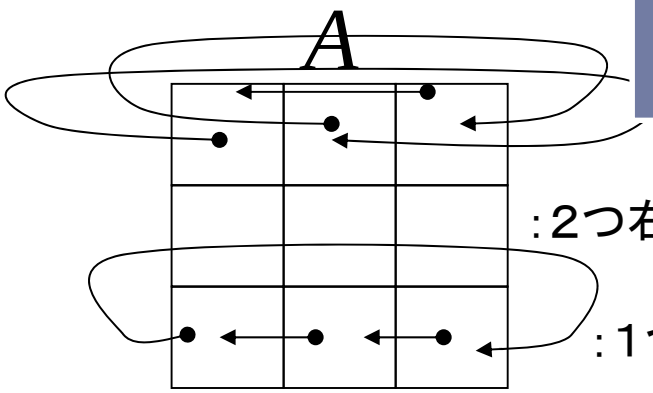
:2つ右に通信

1つ上に通信

2つ上に通信



  - 第二ステップ

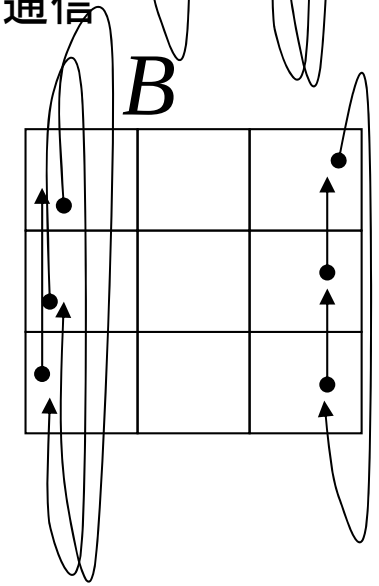


ローカルな
行列-行列積の後

:2つ右に通信

:1つ右に通信

2つ上に通信



1つ上に通信

【通信パターンが
1つ右に循環シフト】

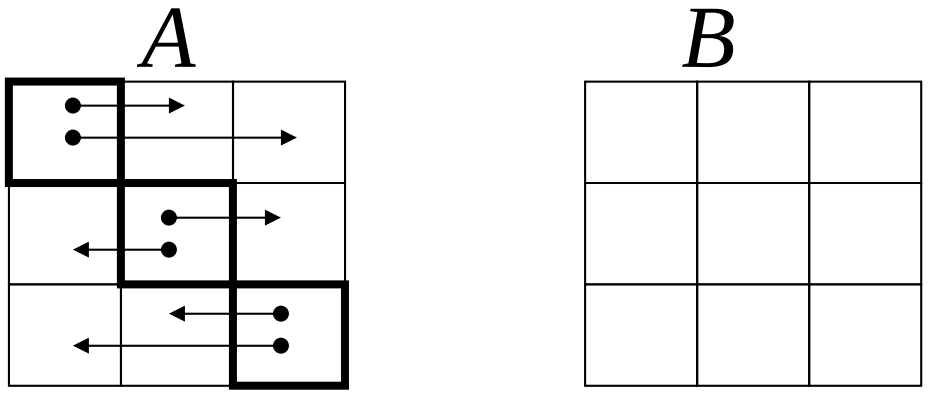
【通信パターンが1つ下に循環シフト】

1.5.1　Cannonのアルゴリズム

- まとめ
  - ＜循環シフト通信＞のみで実現可能
  - １対１通信（隣接通信）のみで実現可能
  - 物理的に隣接PEにしかつながっていないネットワーク（例えば二次元メッシュ）向き
  - 放送処理がハードウエアでできるネットワークをもつ計算機では、遅くなることも

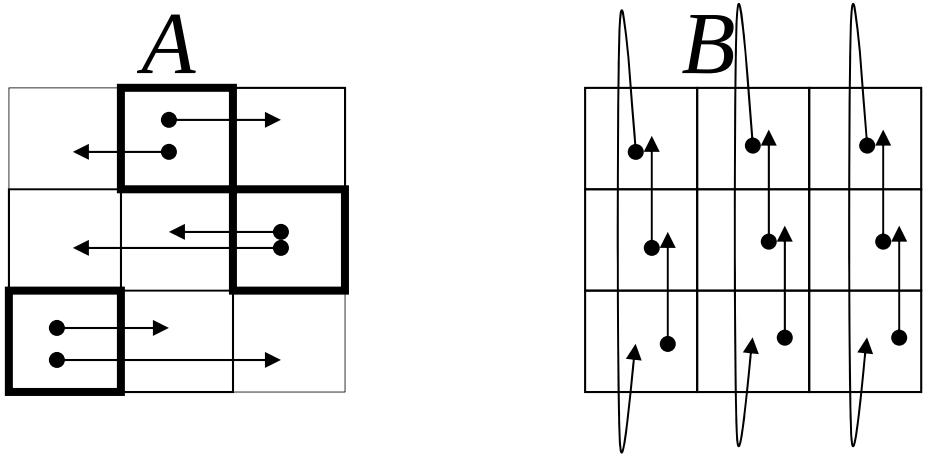
1.5.2　Foxのアルゴリズム

- アルゴリズムの概要
  - 第一ステップ



  - 第二ステップ

【放送PEが
1つ右に
循環シフト】



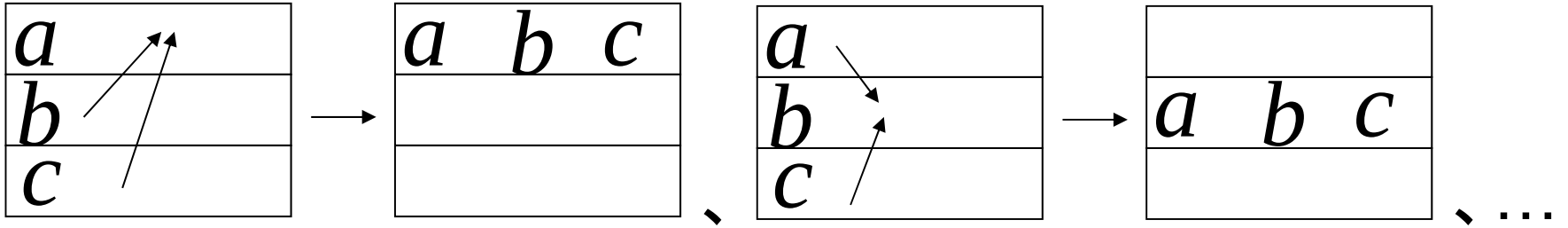
1つ上に通信

1.5.2　Foxのアルゴリズム

- まとめ
  - ＜同時放送（マルチキャスト）＞が必要
  - 物理的に隣接PEにしかつながっていないネットワーク（例えば二次元メッシュ）で性能が悪い（通信衝突が多発）
  - 同時放送がハードウエアでできるネットワークをもつ計算機では、Cannonのアルゴリズムに比べ高速となる

1.5.3　転置を行った後での行列積

- 仮定
  1. データ分散方式：
     行列A、B、C：行方向ブロック分散方式（Block，＊）
  2. メモリに十分な余裕があること：
     分散された行列Bを各PEに全部収集できること
- どうやって、行列Bを収集するか？
  - 2.4節の行列転置の操作をプロセッサ台数回実行



１.5.3　転置を行った後での行列積

- 特徴
  - 一度、行列Bの転置行列が得られれば、*一切通信は不要*
  - 行列Bの転置行列が得られているので、たとえば行方向連続アクセスのみで行列積が実現できる（行列転置の処理が不要）

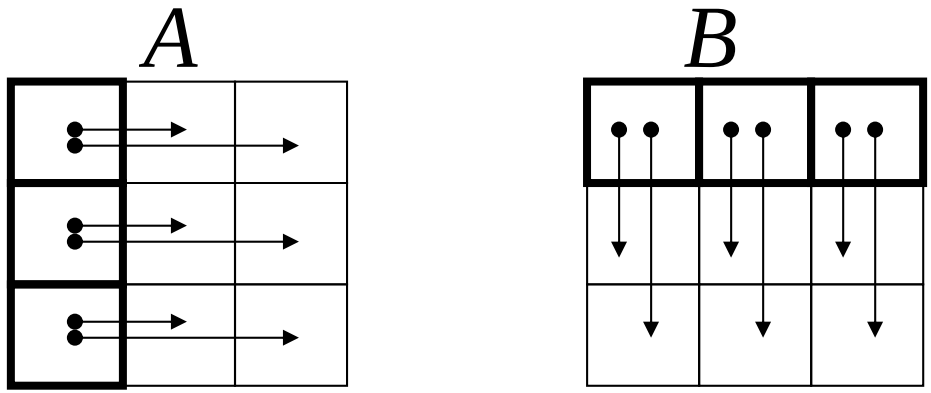
1.5.4　SUMMA、PUMMA

- 近年提案された並列アルゴリズム

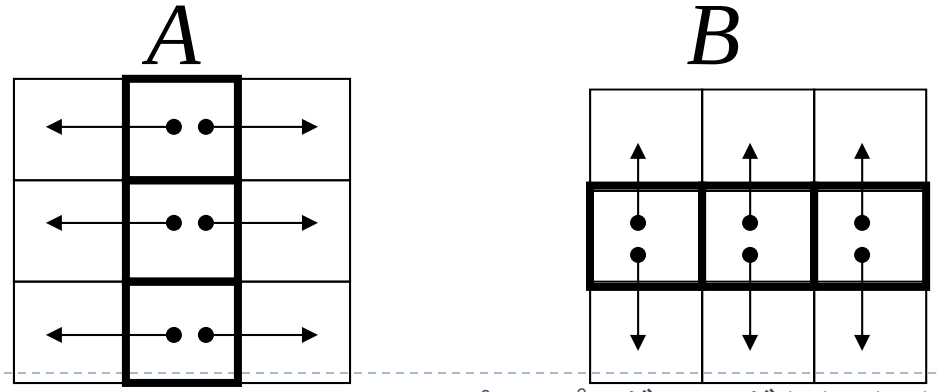
1. SUMMA（Scalable Universal Matrix Multiplication Algorithm）
   - R.Van de Geijinほか、1997年
   - 同時放送（マルチキャスト）のみで実現
2. PUMMA（Parallel Universal Matrix Multiplication Algorithms）
   - Choiほか、1994年
   - 二次元ブロックサイクリック分散方式むきのFoxのアルゴリズム

1.5.4 SUMMA

- アルゴリズムの概略
  - 第一ステップ

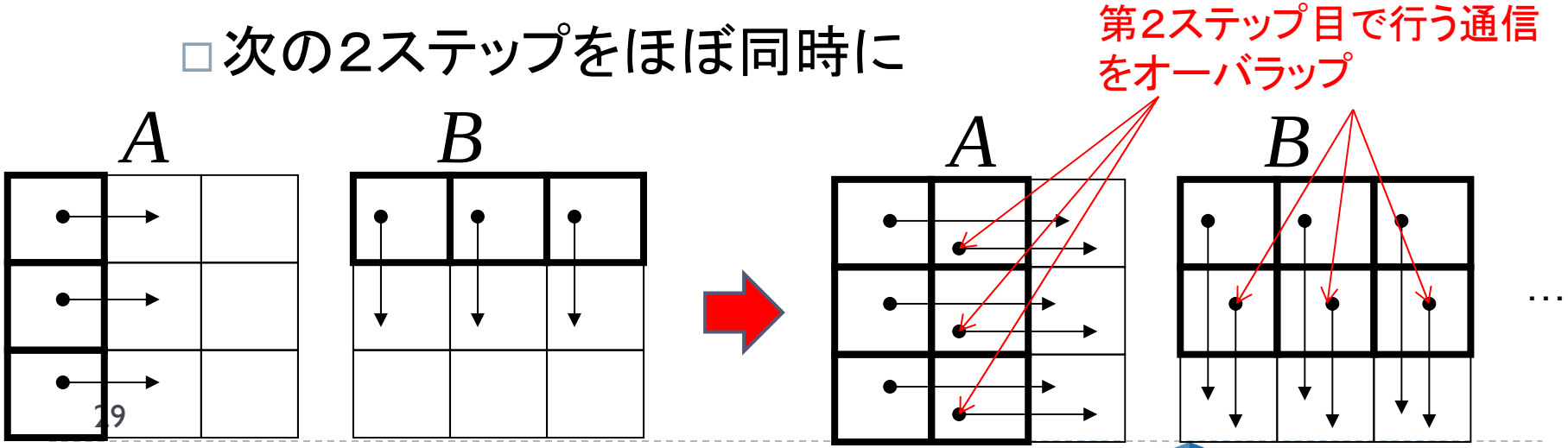


  - 第二ステップ



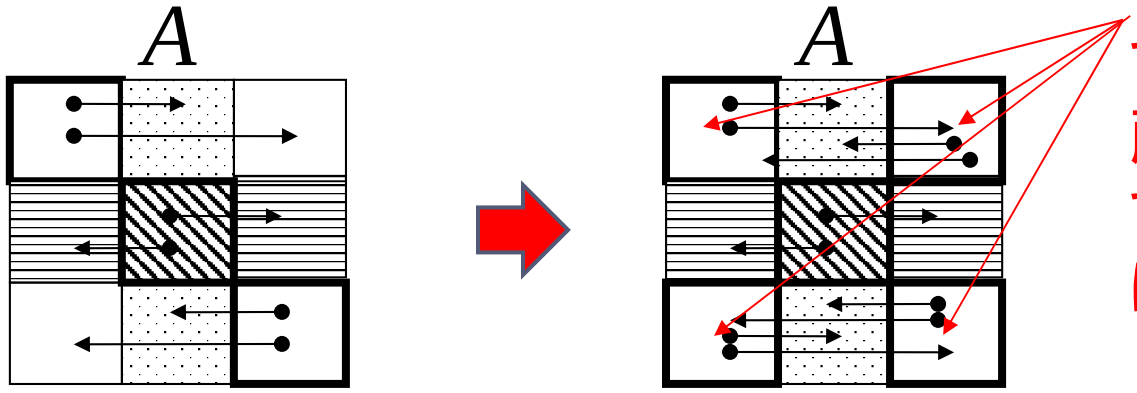
1.5.4 SUMMA

- 特徴
  - 同時放送をブロッキング関数（例．MPI_Bcast）で実装すると、同期回数が多くなり性能低下の要因になる
  - SUMMAにおけるマルチキャストは、非同期通信の1対1通信（例．MPI_Isend）で実装することで、通信と計算のオーバラップ（ ***通信隠蔽*** ）可能
    - 次の2ステップをほぼ同時に



1.5.4　PUMMA

- 概略
  - 二次元ブロックサイクリック分散方式用のFoxアルゴリズム
  - ScaLAPACKが二次元ブロックサイクリック分散を採用していることから開発された
  - 例：



＜同じPE＞が所有しているデータだから、所有データをまとめて＜同一宛先PE＞に一度に送る

1.5.5　Strassenのアルゴリズム

- 素朴な行列積：n^3 の乗算と $(n-1)^3$ の加算
- Strassenのアルゴリズムでは $n^{\log 7}$ の演算
- アイデア：　＜分割統治法＞
  - 行列を小行列に分割して、計算を分割
- 実際の性能
  - 再帰処理や行列のコピーが必要
  - 素朴な実装法より遅くなることがある
  - 再帰の打ち切り、再帰処理展開などの工夫をすれば、（nが大きい範囲で）効率の良い実装が可能

1.5.5　Strassenのアルゴリズム

- 並列化の注意
  - アルゴリズムを単純に分散メモリ型並列計算機に実装すると通信が多発
    - 性能がでない
- PE内の行列積をStrassenで行い、PE間をSUMMAなどで実装すると効率的な実装が可能
- ところが通信量は、アルゴリズムの性質から、通常の行列 - 行列積アルゴリズムに対して減少する。この性質を利用して、近年、Strassenを用いた通信回避アルゴリズムが研究されている。

サンプルプログラムの実行（行列-行列積）

行列-行列積のサンプルプログラムの注意点

- C言語版/Fortran言語版の共通ファイル名

  Mat-Mat-fx.tar

- ジョブスクリプトファイルmat-mat.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-fx.tar ./

  $ tar xvf Mat-Mat-fx.tar

  $ cd Mat-Mat

- 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.bash

- 実行が終了したら、以下を実行する

  $ cat mat-mat.bash.oXXXXXX

東京大学情報基盤センター
INFORMATION TECHNOLOGY CENTER, THE UNIVERSITY OF TOKYO

行列-行列積のサンプルプログラムの実行（C言語）

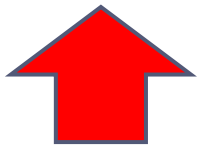
- 以下のような結果が見えれば成功

N = 1000

Mat-Mat time = 0.209609 [sec.]

9541.570931 [MFLOPS]

OK!



1コアのみで、9.5GFLOPSの性能

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 = 1000

Mat-Mat time[sec.] = 0.204734672995982 7

MFLOPS = 9768.741003580422

OK!

1コアのみで、9.7GFLOPSの性能

サンプルプログラムの説明

- #define N 1000
  の、数字を変更すると、行列サイズが変更できます
- #define DEBUG 0
  の「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N×N行列AとBの行列積をおこない、Double型N×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.inc

- 行列サイズ変数が、NNとなっています。

```
integer  NN
parameter (NN=1000)
```

演習課題（１）

- MyMatMat関数を並列化してください。
  - #define　N　　192
  - #define　DEBUG　1

  として、デバッグをしてください。
- 行列A、B、Cは、各PEで重複して、かつ全部（N×N）所有してよいです。

演習課題（１）

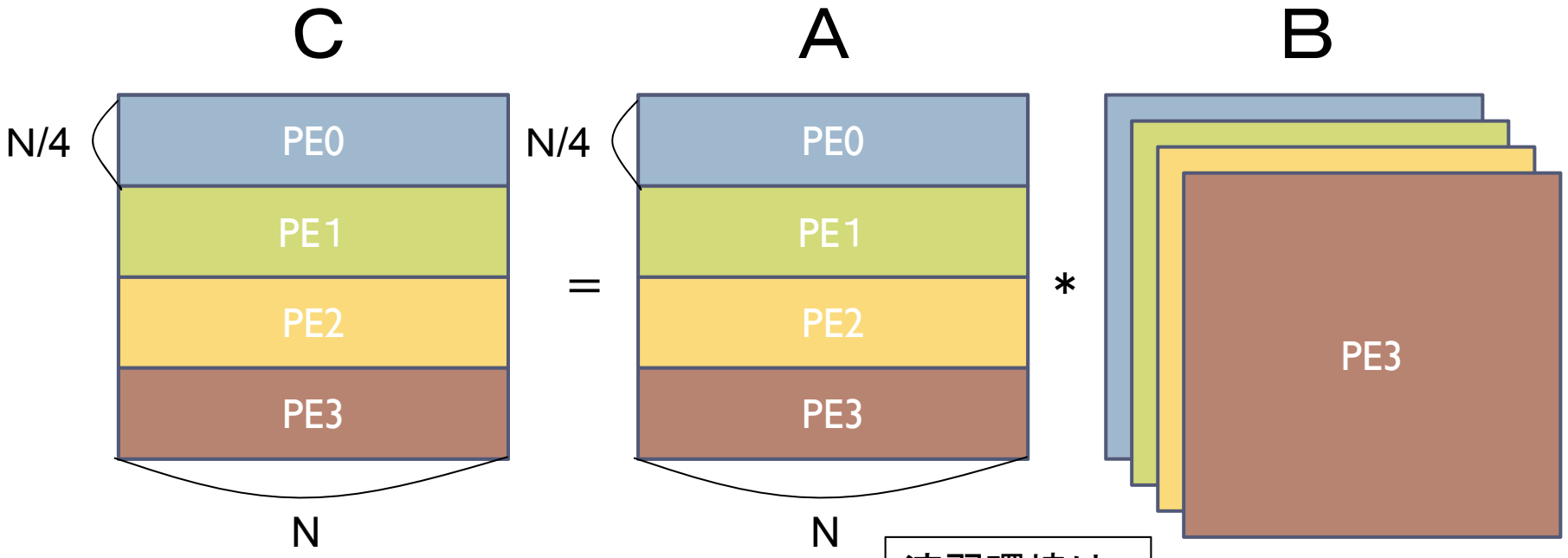
- サンプルプログラムでは、行列A、Bの要素を全部1として、行列-行列積の結果をもつ行列Cの全要素がNであるか調べ、結果検証しています。デバックに活用してください。
  - 行列Cの分散方式により、

演算結果チェックルーチンの並列化が必要

になります。注意してください。

並列化のヒント

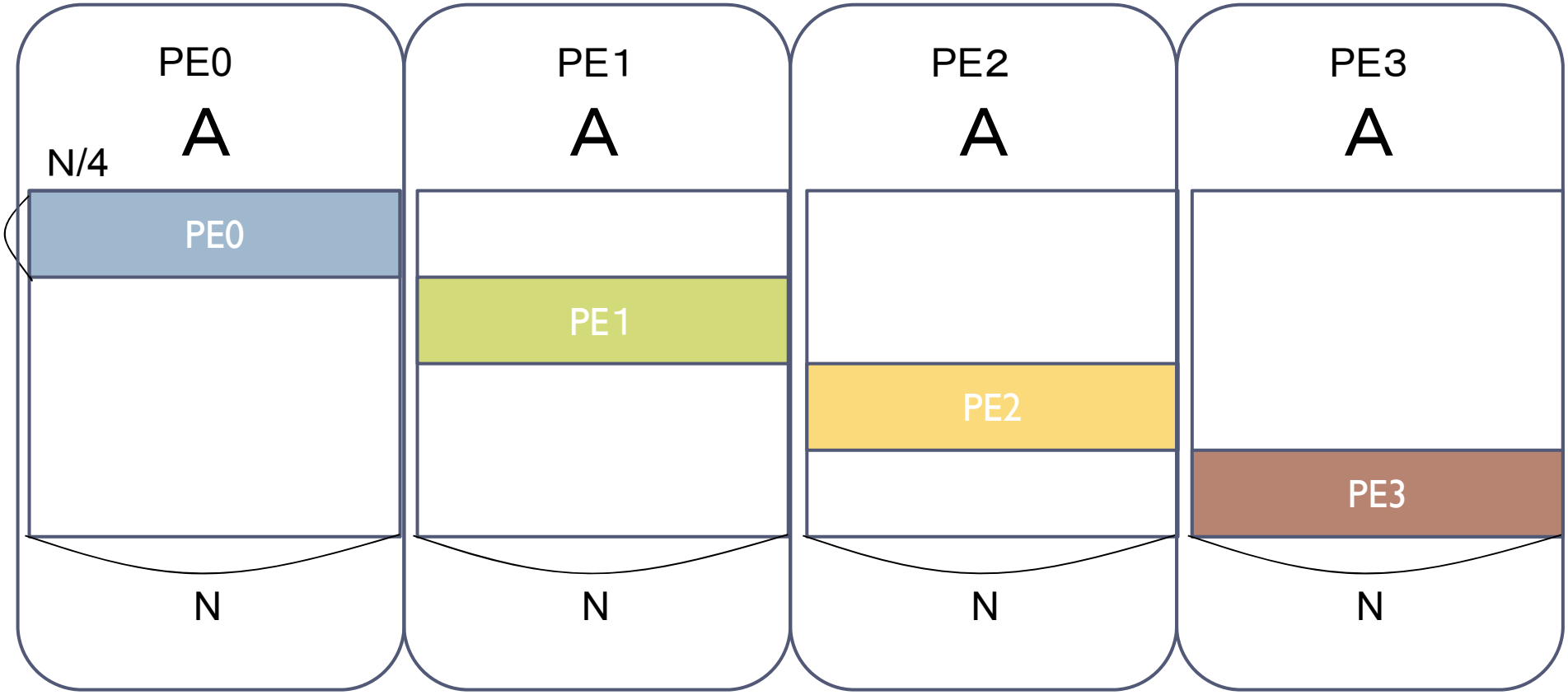
- 以下のようなデータ分割にすると、とても簡単です。



- 通信関数は一切不要です。
- 行列-ベクトル積の演習と同じ方法で並列化できます。

各PEでの配列の確保状況

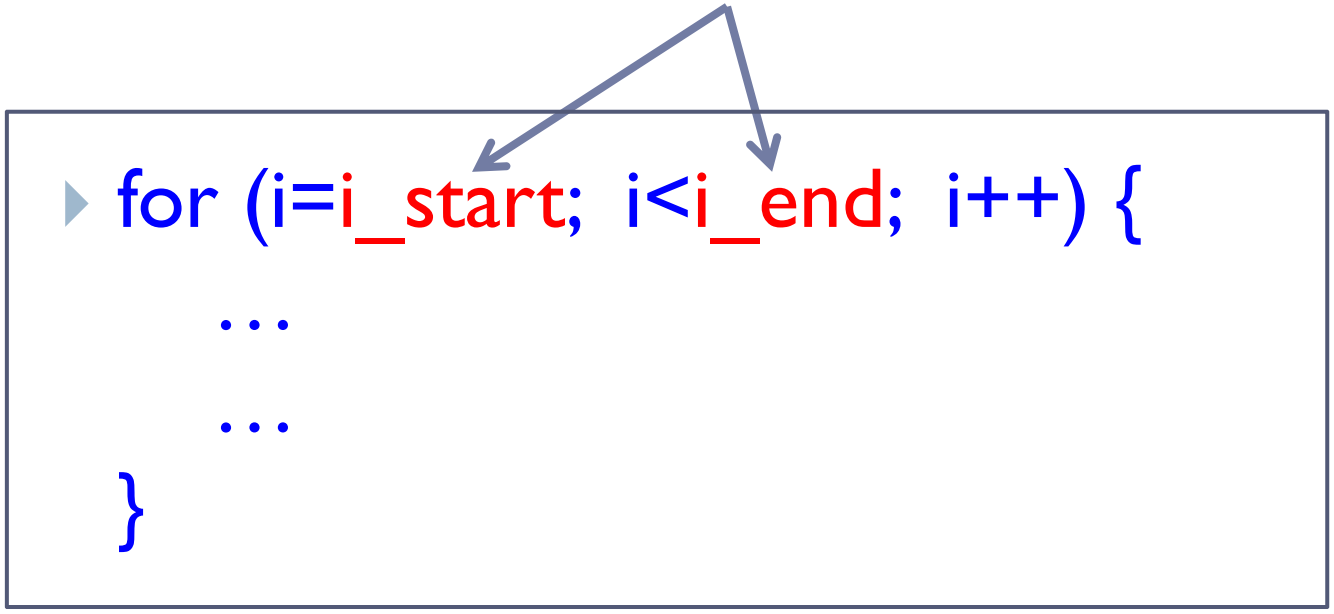
- 実際は、以下のように配列が確保されていて、
  部分的に使うだけになります



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



レポート課題

1. [L10]　行列-行列積を並列化せよ。
   ここで、行列A、B、Cについての初期状態は
   各PEで重複したデータをもってよい。
2. [L15]　行列-行列積を並列化せよ。ここで、行列A、B、Cの初期状態は各PEで重複したデータをもってはならない。（来週の演習課題（2））

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

3. [L25]　Cannonのアルゴリズムを実装せよ。
4. [L25]　Foxのアルゴリズムを実装せよ。
5. [L35]　SUMMAのアルゴリズムを実装せよ。
   ここで放送処理はマルチキャストを用いよ。
6. [L35]　PUMMAのアルゴリズムを実装せよ。
   ここで放送処理はマルチキャストを用いよ。
7. [L40]　１対１通信関数を用いて通信のオーバラップを行ったSUMMAのアルゴリズムを実装せよ。また、マルチキャスト版SUMMAと、性能を比較せよ。

レポート課題

8. [L20]　並列化したコードについて、
ピュアMPI実行とハイブリッドMPI実行を行い、
演習環境を駆使して性能評価を行え。また、
ピュアMPI実行が高速となる条件を算出し、
妥当性を実験結果から検証せよ。

来週へつづく

行列－行列積（２）