

行列-ベクトル積

東京大学情報基盤センター 准教授 埴 敏博

2020年5月26日(火) 10:25-12:10

レポートおよびコンテスト課題
(締切:
2020年8月3日(月)24時 厳守

講義日程(工学部共通科目)

- ~~1. 4月14日 ガイダンス~~
- ~~2. 4月21日~~
 - ~~● 並列数値処理の基本演算(座学)~~
- ~~3. 4月28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12日~~
 - ~~● 高性能プログラミング技法の基礎1
(階層メモリ、ループアンローリング)~~
- ~~5. 5月19日~~
 - ~~● 高性能プログラミング技法の基礎2
(キャッシュブロック化)~~
6. 5月26日
 - 行列-ベクトル積の並列化
7. 6月2日
 - べき乗法の並列化
8. 6月9日
 - 行列-行列積の並列化(1)
9. 6月16日
 - 行列-行列積の並列化(2)
10. 6月23日
 - LU分解法(1)
 - コンテスト課題発表
11. 6月30日
 - LU分解法(2)
12. 7月7日
 - LU分解法(3)、非同期通信
13. 7月14日
 - RB-Hお試し、研究紹介他

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

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-ofp.tar.gz
- ジョブスクリプトファイル**mat-vec.bash** 中のキュー名を
lecture-flat から
lecture5-flat (工学部共通科目)
に変更し、**pjsub** してください。
 - **lecture-flat** : 実習時間外のキュー
 - **lecture5-flat**: 実習時間内のキュー
 - グループを**gt00**から**gt45**に変える

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

```
$ cd /work/gt45/t45xxx
```

```
$ cp /work/gt45/z30105/Mat-vec-ofp.tar.gz ./
```

```
$ tar xvfz Mat-vec-ofp.tar.gz
```

```
$ 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
```

- ジョブスクリプトを修正したら

```
$ 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 = 10880

Mat-Vec time = 0.973228 [sec.]
205.501696 [MFLOPS]

OK!

N = 10880

Mat-Vec time = 0.699561 [sec.]
285.893533 [MFLOPS]

OK!

N = 10880

Mat-Vec time = 0.523037 [sec.]
382.382158 [MFLOPS]

OK!

コアの割り当てが偏っている
(デフォルト)

コアの最適割り当て

```
source /usr/local/bin/mpi_core_setting.sh
```

+MCDRAMの利用(なるべく)

```
export I_MPI_HBW_POLICY  
=hbw_preferred
```


実行結果 (Fortran言語)

- 以下のような結果が出ればOK。

N = 10880

Mat-Vec time[sec.] = 1.33347105979919

MFLOPS = 149.984506994225 コアの割り当てが偏っている
(デフォルト)

N = 10880

Mat-Vec time[sec.] = 0.781502008438110

MFLOPS = 255.917447857573 コアの最適割り当て

OK!

```
source /usr/local/bin/mpi_core_setting.sh
```

N = 10880

Mat-Vec time[sec.] = 0.481024980545044

MFLOPS = 415.778821441728 +MCDRAMの利用(なるべく)

OK!

```
export I_MPI_HBW_POLICY  
=hbw_preferred
```

サンプルプログラムの説明(C言語)

- `#define N 10880`
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

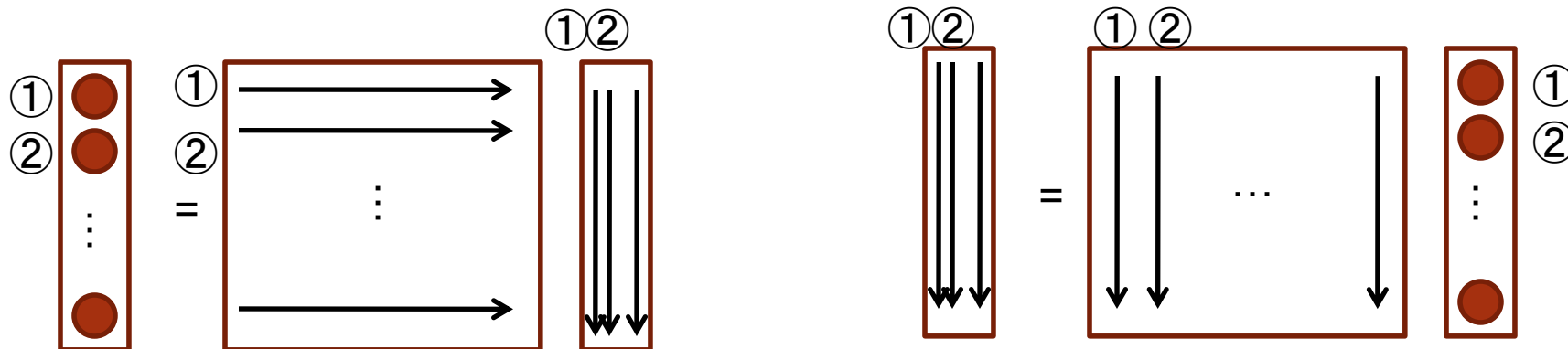
- 行列サイズ変数が、NNとなっています。
`integer,parameter :: NN=10880`

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 1088**
にしてください。
 - 多すぎて大変な場合は、N、およびジョブスクリプト中のMPIプロセス数(proc=数字)を小さくしてください。
 - **#define DEBUG 1**
にして、結果を検証してください。

行列とベクトルの積

- ・ **<行方式>**と**<列方式>**がある。
 - ・ **<データ分散方式>**と**<方式>**の組み合わせがあり、少し面白い



```

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

```

```

do j=1, n
    y(j) = 0.0
enddo
do j=1, n
    do i=1, n
        y(i) = y(i) + a(i, j) * x(j)
    enddo
enddo

```

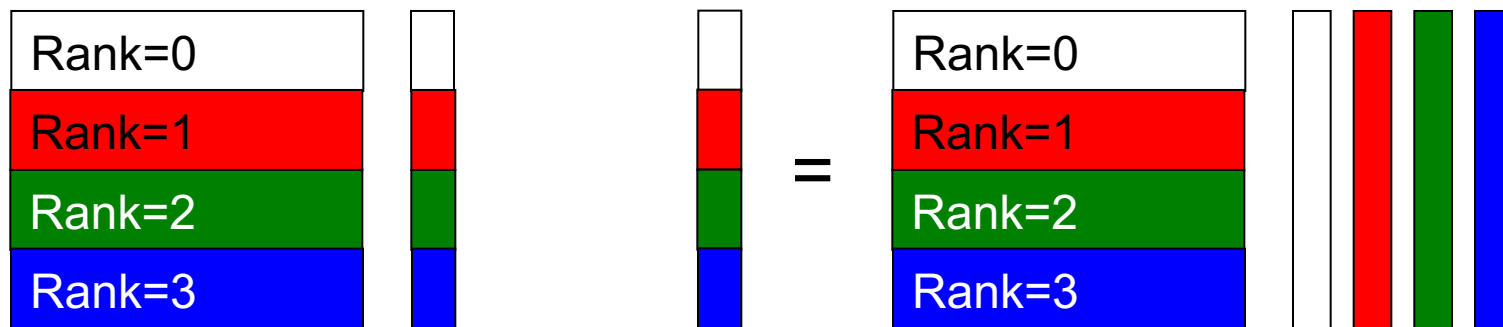
<行方式>： 自然な実装
C言語向き

<列方式>： Fortran言語向き

行列とベクトルの積

＜行方式の場合＞

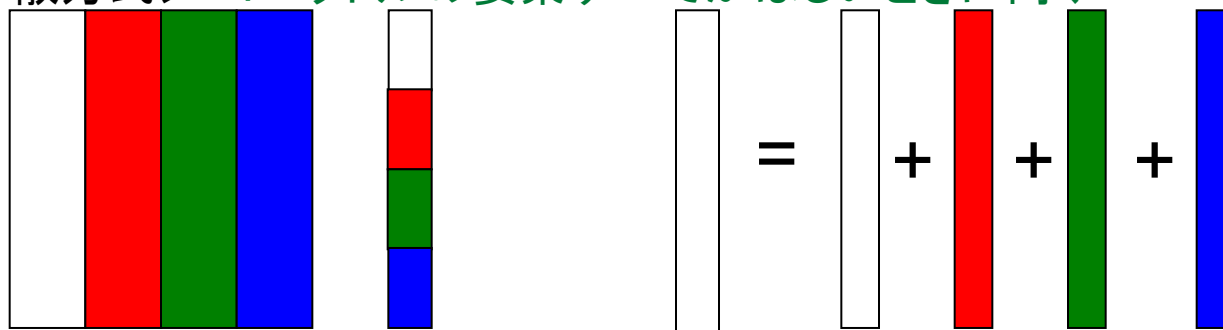
＜行方向分散方式＞ : 行方式に向く分散方式



右辺ベクトルを `MPI_Allgather` 関数
を利用し、全ランクで所有する

各ランク内で行列ベクトル積を行う

＜列方向分散方式＞ : ベクトルの要素すべてがほしいときに向く



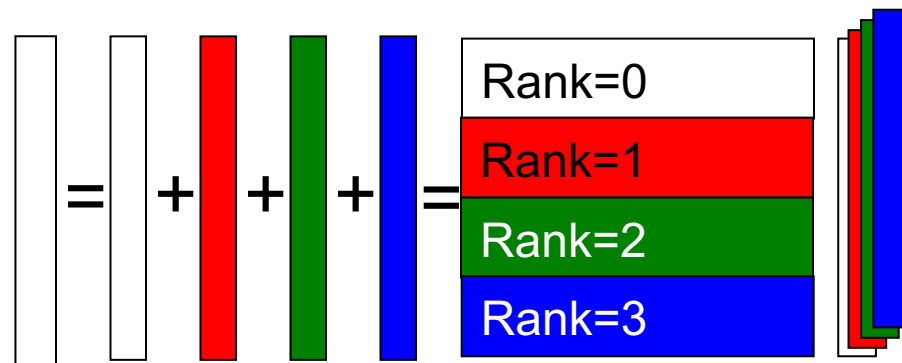
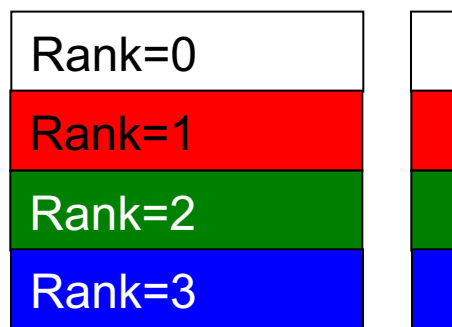
各ランク内で行列-ベクトル積
を行う

`MPI_Reduce` 関数で総和を求める
(※あるランクにベクトルすべてが集まる)

行列とベクトルの積

<列方式の場合>

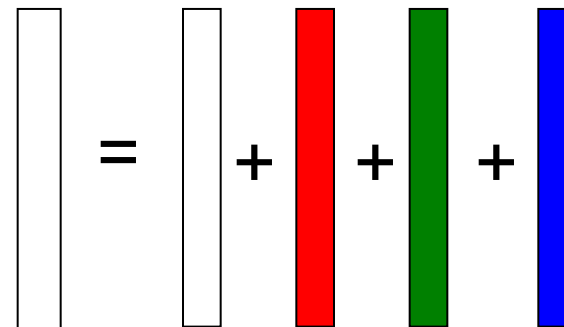
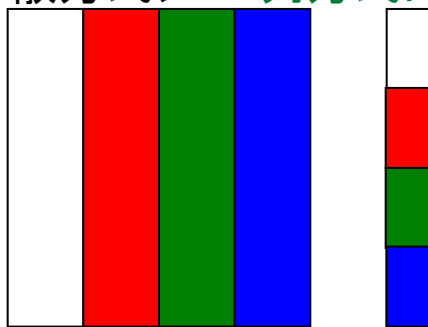
<行方向分散方式> : 無駄が多く使われない



右辺ベクトルを `MPI_Allgather` 関数
を利用して、全ランクで所有する

結果を `MPI_Reduce` 関数により
総和を求める

<列方向分散方式> : 列方式に向く分散方式



各ランク内で行列-ベクトル積
を行う

`MPI_Reduce` 関数で総和を求める
(※あるランクにベクトルすべてが集まる)

演習課題の注意

- データが各ランク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ランク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ランク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ランクで対応する
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、
 - **すべてのランクで、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

- 各ランクは、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

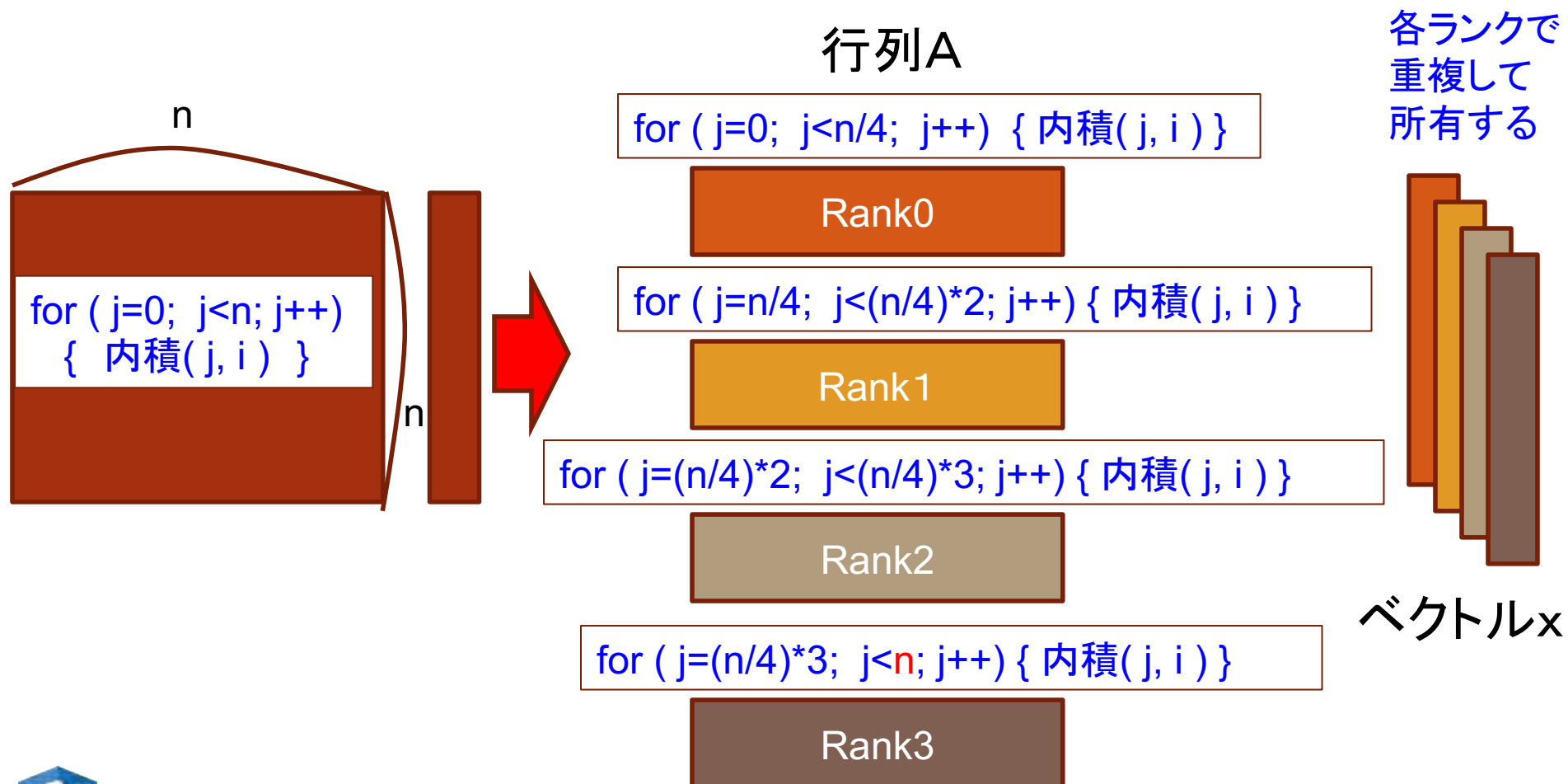
- **myid, numprocs は大域変数です**
 - myid (=自分のID)、および、numprocs(=世の中のランク台数)の変数は大域変数です。

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、並列化できません。

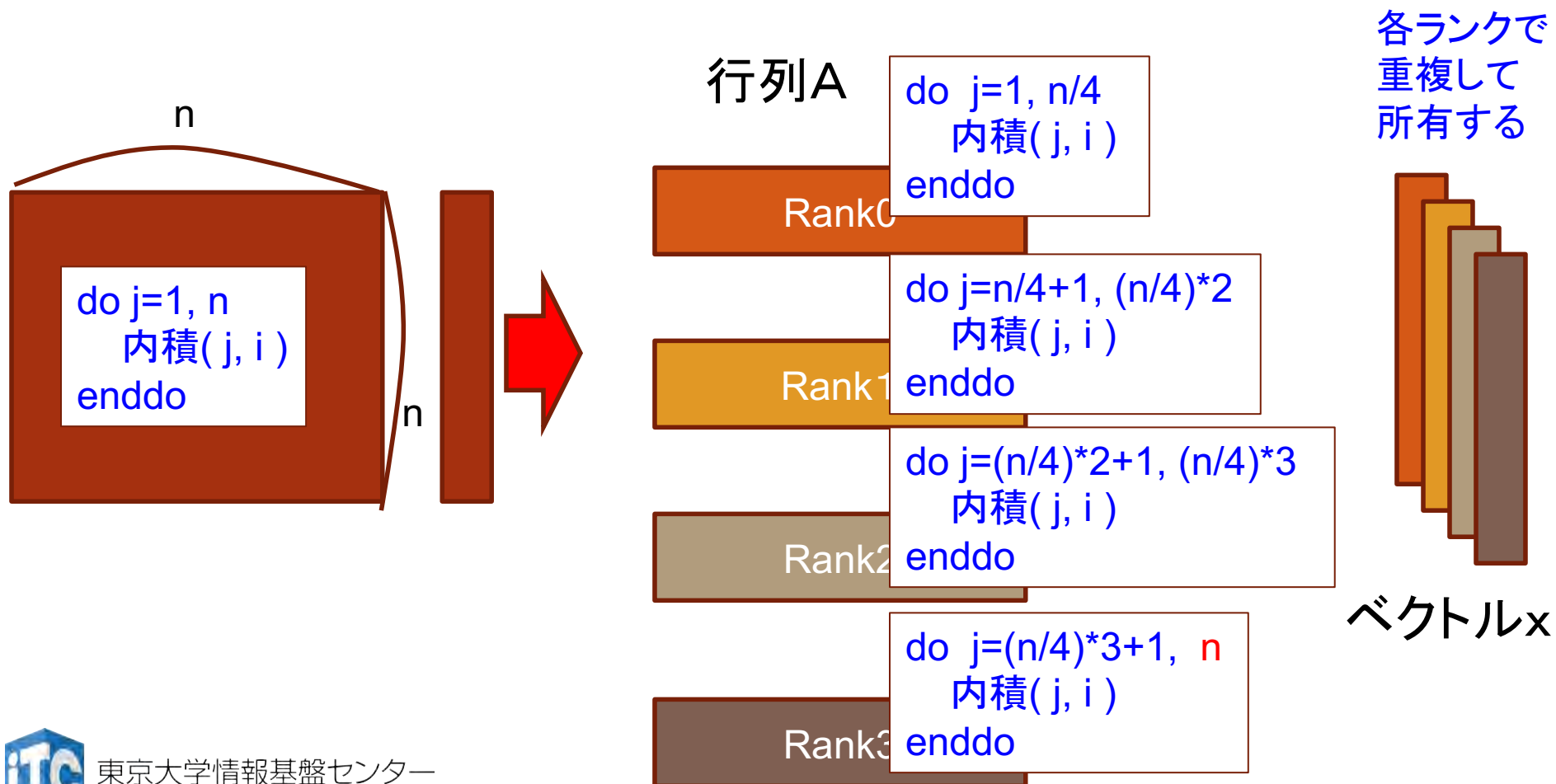
並列化の考え方(C言語)

• SIMDアルゴリズムの考え方(4ランクの場合)



並列化の考え方 (Fortran言語)

• SIMDアルゴリズムの考え方 (4ランクの場合)



初心者が注意すること

- 各ランクでは、**独立した配列が個別に確保**されます。



- `myid`変数は、`MPI_Comm_rank()`関数が呼ばれた段階で、**各ランク固有の値**になっています。



並列化の方針(C言語)

1. 全ランクで行列Aを $N \times N$ 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. 各ランクは、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)各ランク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. 全ランクで行列Aを $N \times N$ 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. 各ランクは、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)各ランク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)

- 全ランクで $N \times N$ 行列を持つ場合

Rank0

```
for (j=0; j<(n/4); j++) { 内積(j, i) }
```

```
for (j=(n/4)*2; j<(n/4)*3; j++) { 内積(j, i) }
```

Rank2

Rank1

```
for (j=(n/4); j<(n/4)*2; j++) { 内積(j, i) }
```

```
for (j=(n/4)*3; j<n; j++) { 内積(j, i) }
```

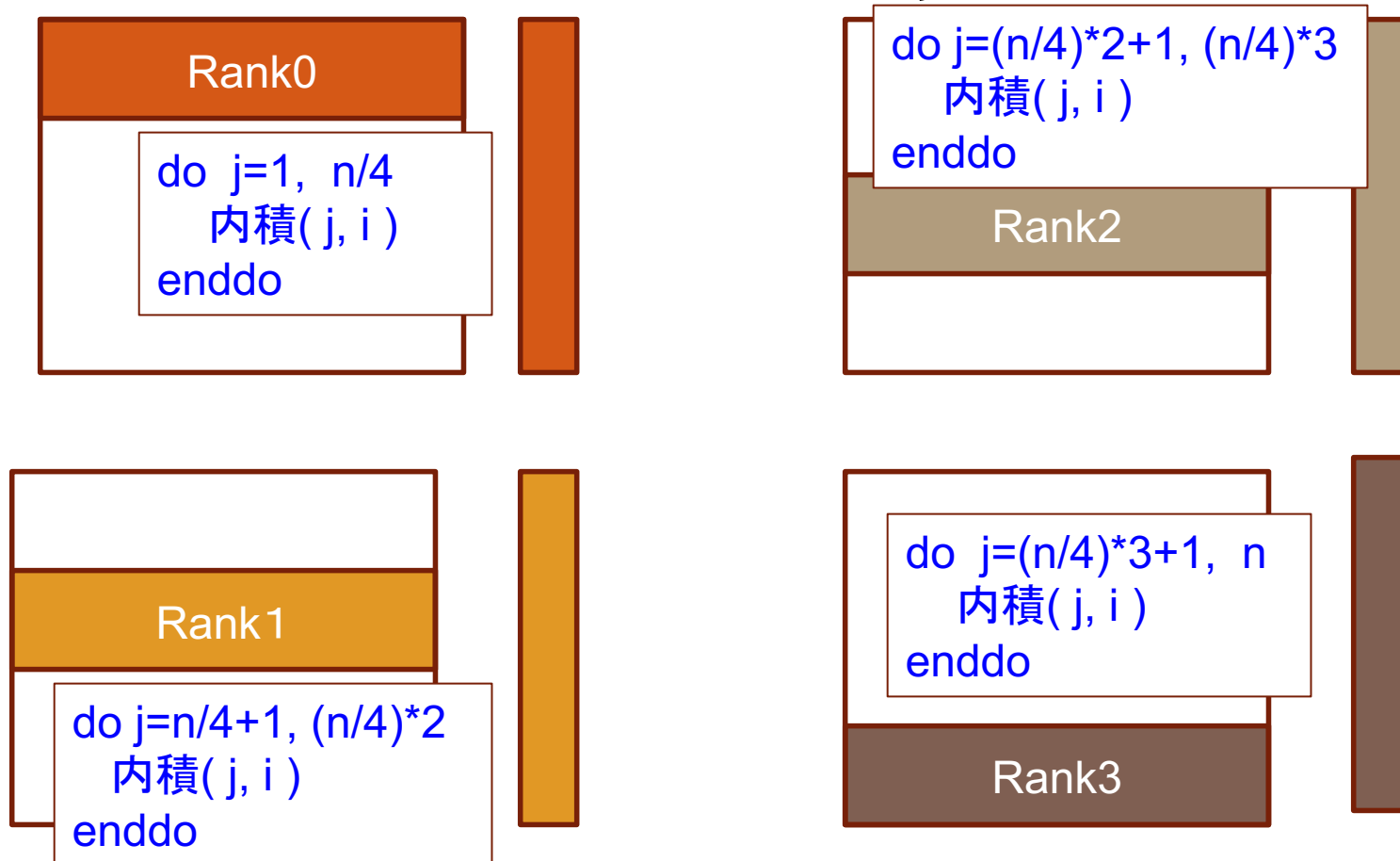
Rank3

※各ランク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)

- 全ランクで $N \times N$ 行列を持つ場合



※各ランク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！



並列化時の注意

- 演習環境は、1088ランクです。
- 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、ランク0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。**

発展実装 (Nがランク数で割切れない時)

- Nがランク数の1088で割り切れない場合
 - 配列確保: $A[N/1088 + (N - (N/1088) * 1088)][N]$
 - ループ終了値: ランク1087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++) { ... }
```

発展実装(担当データしか持たない時)

- 担当データ分しか所有しない場合
 - 各ランクが、ローカルインデックス(0~n/1088、もしくは0 ~ (n/1088+(N-(N/1088)*1088)))のほかに、各ランクが所有するデータのグローバルインデックス(0~n)を知る必要がある。
 - ベクトルxデータを集めた後、ベクトルxデータにアクセスする際
 - A、y: ローカルインデックスでアクセス
 - x: グローバルインデックスでアクセス
 - ブロック分散なら簡単。
 - サイクリック分散だと、ちょっと工夫がいる。
 - モジュロ関数($a\%b$)を利用する。

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、全ランクで $N \times N$ 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、各ランク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1/1088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(16ノード、1088コア)を
駆使して、性能評価せよ。
- 1ノードあたり、68MPI実行、1MPIx68スレッド実行、
2MPIx68スレッド実行、64MPIx17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