ハイパフォーマンスコンピューティング：LU分解

LU分解法

- LU分解法は、以下3ステップでAx=bの解の計算を行う。

1. 行列AのLU分解を行う

$$A = LU$$

$$L = \begin{bmatrix} l_{11} & & & \\ l_{21} & l_{22} & & \\ \vdots & & \ddots & \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \qquad U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ & u_{22} & & u_{2n} \\ & & \ddots & \vdots \\ & & & u_{nn} \end{bmatrix}$$

下三角行列　　上三角行列

計算量: $(2/3)*n^3 + O(n^2)$回の演算で実行できる。

LU分解法

$$LUx = b \quad \Rightarrow \quad \begin{cases} Ly = b \\ Ux = y \end{cases}$$

1. 前進代入：Ly = b を解く。(L_{ii} = 1に注意)

$$\begin{bmatrix} l_{11} & & & \\ l_{21} & l_{22} & & \\ \vdots & & \ddots & \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \qquad \begin{cases} y_1 = b_1 \\ y_2 = b_2 - l_{21} y_1 \\ \vdots \\ y_n = b_n - \sum_{j=1}^{n-1} l_{ij} y_j \end{cases}$$

2. 後退代入：Ux = y を解く。

$$\begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ & u_{22} & & u_{2n} \\ & & \ddots & \vdots \\ & & & u_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \qquad \begin{cases} x_n = y_n / u_{nn} \\ x_{n-1} = (y_{n-1} - u_{n-1,n} y_n) / u_{n-1,n-1} \\ \vdots \\ x_1 = (y_1 - \sum_{j=2}^{n} u_{1j} y_j) / u_{11} \end{cases}$$

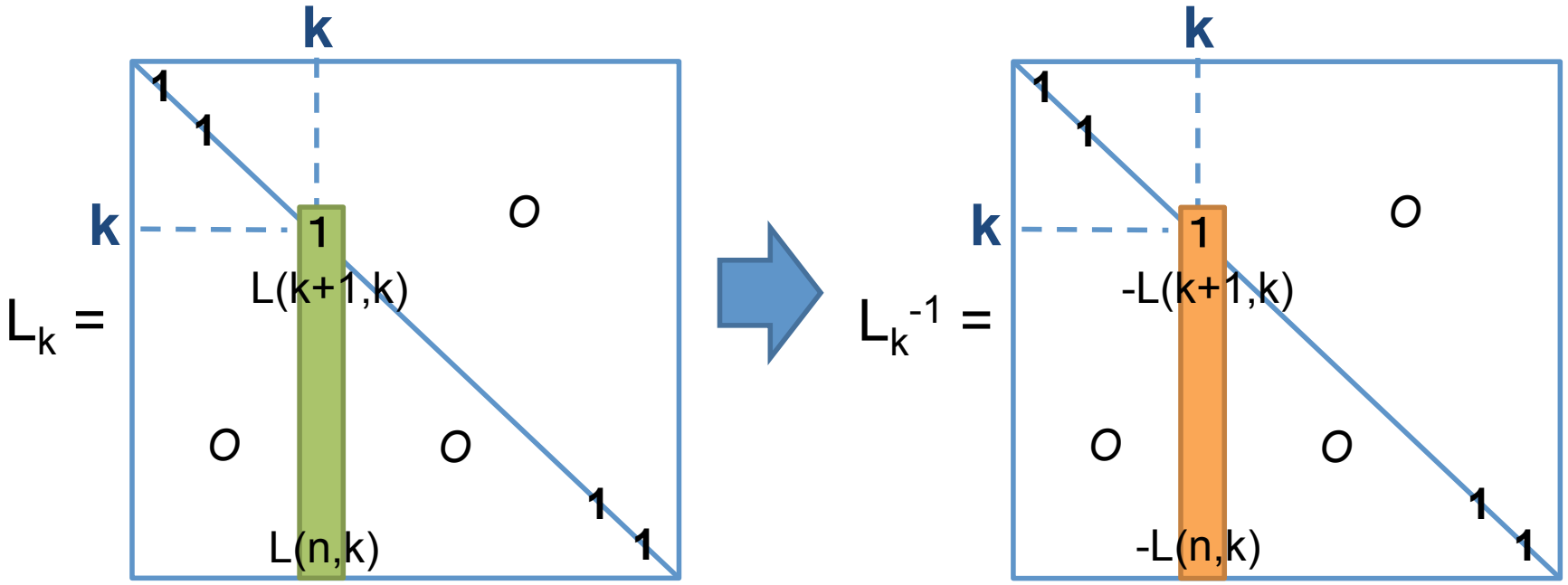
計算量：LU分解が既に実行されていれば、前進・後退代入については
O(n^2)回の演算で実行できる（右辺bを変更してもOK）。

LU分解の仕組み(1)

- ガウスの消去法と同様に、列の消去を行う。
- k段目の消去の際、行列L_kを以下とする（対角成分は1）。

$$L_k(i,k) = A_k(i,k)/A_k(k,k), \quad i = k+1:n$$

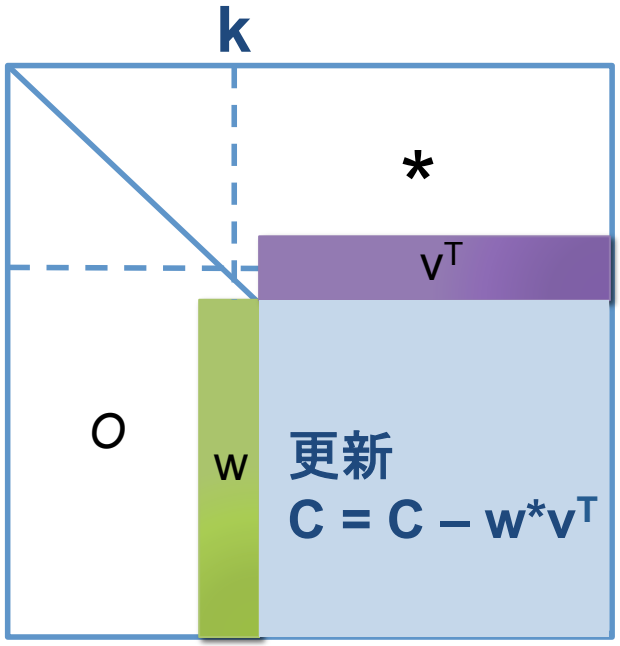
- L_k^{-1}は右図のようになる。



LU分解の仕組み(2)

- 行列L_kからA_{k+1}が得られる。これを、k = 1:n-1と繰り返す。

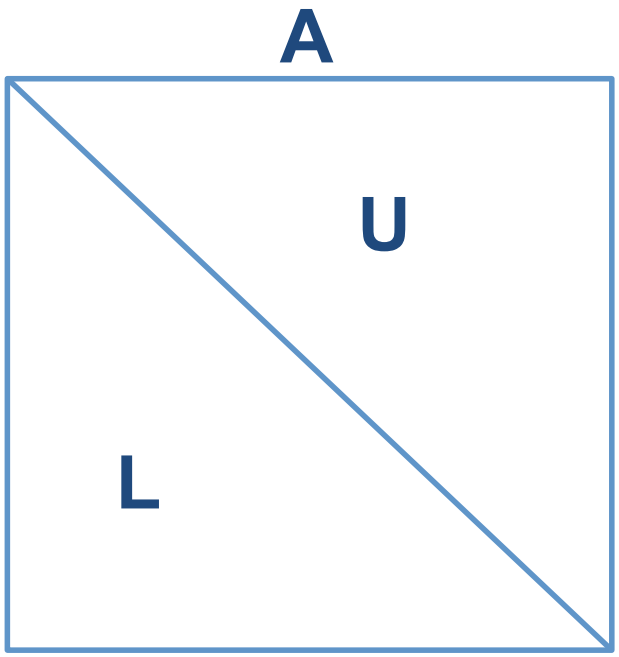
$A_{k+1} = L_k^{-1}A_k =$



- $A_n = L_{n-1}^{-1}...L_2^{-1}L_1^{-1}A = U$（上三角行列）
- $L_1L_2...L_{n-1} = L$（単位下三角行列）
- ただし、$L(i,k) = L_k(i,k)$, i = k+1:n

プログラム作成のヒント

メモリ量の節約のため、
Aを上書きして更新し、LとUを格納する。



※Lの対角要素は1であることを仮定

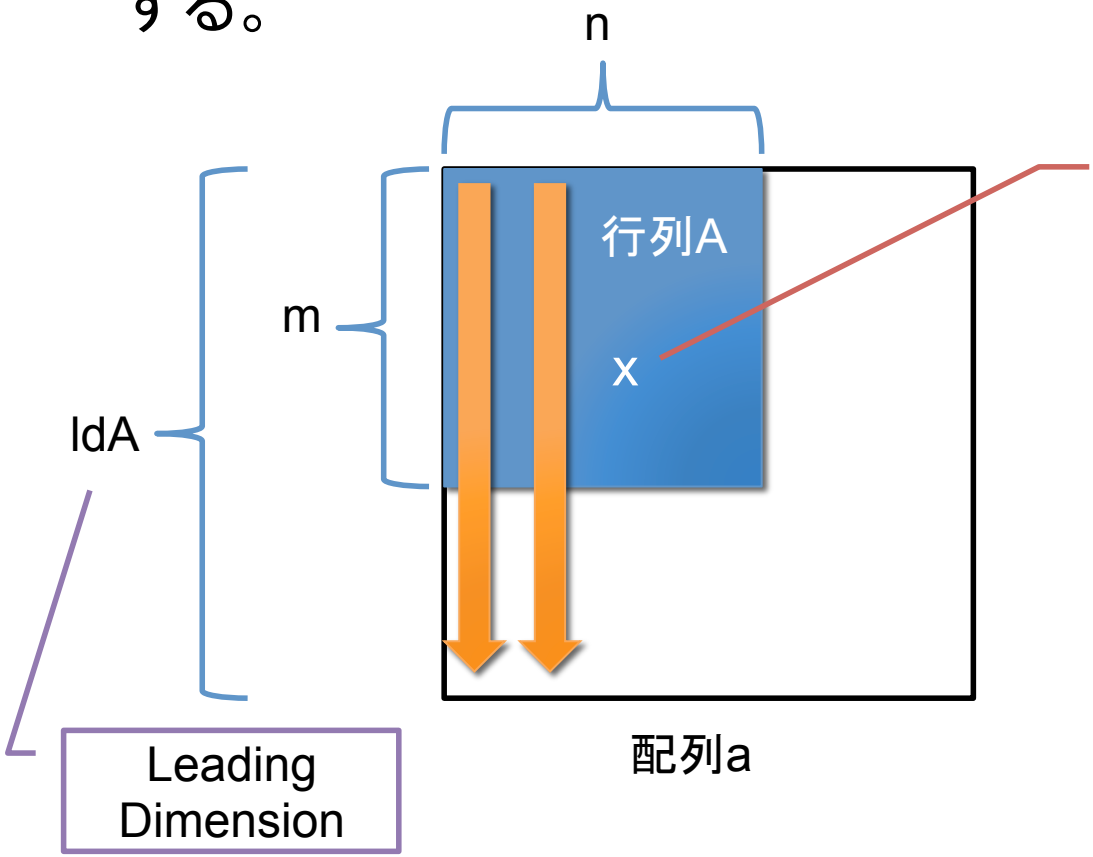
```
for k=1:n-1
    for i=k+1:n  % Lの部分
        A(i,k) = A(i,k)/A(k,k);
    end
    for j=k+1:n  % Uの部分
        for i=k+1:n
          A(i,j) = A(i,j) – A(i,k)*A(k,j);
        end
    end
end

% 以下は、Matlabの出力用
U = triu(A);  % 上三角部分を取り出す
L = eye(n) + tril(A,-1); % 下三角部分
```

実習:LU分解法のプログラム作成

1. MATLABで作成しましょう。
   1. まず、軸交換しないLU分解のプログラムを作成する。
      - [L,U]=mylu1(A) のように利用できるようにし、適当なテスト行列A（たとえば、A=randn(n) ）について、L*UがAに近いことを確認しよう。
   2. 次に、部分軸交換の場合はどうなるか考えて、部分軸交換付きのLU分解のプログラムを作成する。
      - [L,U,P]=mylu2(A) のように利用できるようにし、L*UがP*Aに近いことを確認しよう。
2. 1ができたらC言語で作成しましょう。

数値計算ライブラリとの互換性を持った行列の表現

- 列方向（下方向）に連続アクセスとなるようにする。
- Leading Dimensionを利用して、部分行列も取り扱えるようにする。



A(i,j)の表現方法

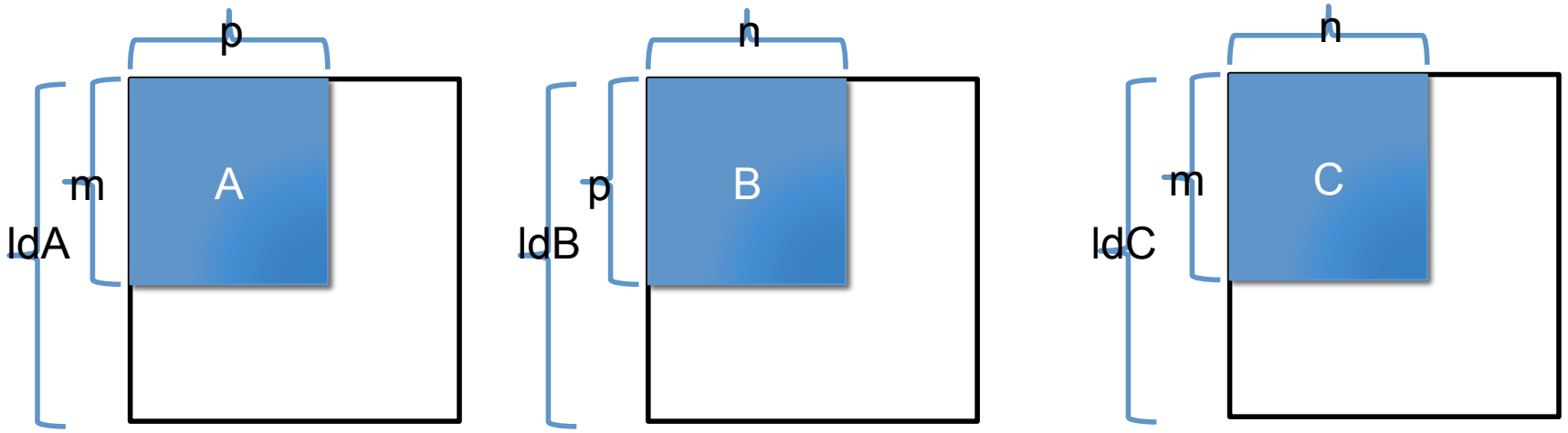
・C言語の場合
a[i+j*ldA]

・Fortranの場合
a(i+(j-1)*ldA)

Fortranの場合は、元々下方向に連続となるので、2次元配列のまま行列を表現しても良い

実習

- 前述のLeading Dimensionを利用した1次元配列による行列の表現を使って、行列積 C = AB を実行する関数とテストプログラムを作成しよう。



- 関数は下記のような形式になる。

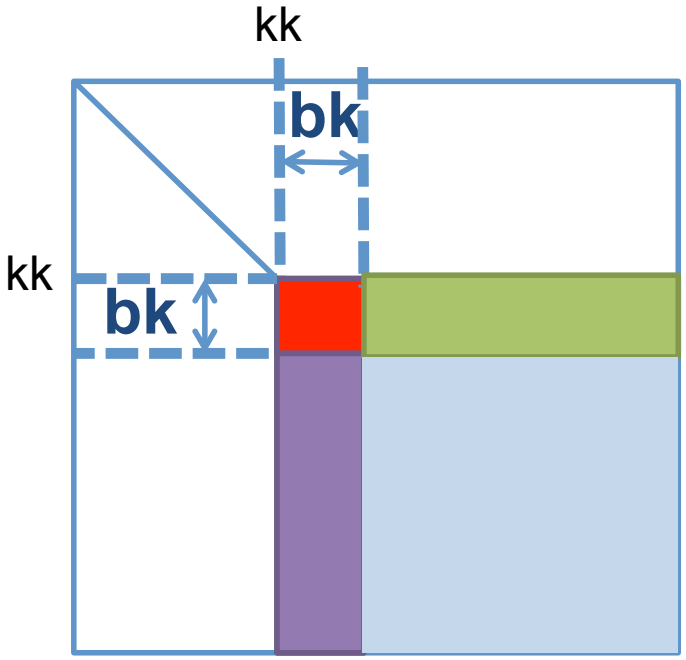
```
void multmm(int m, int p, int n, double *A, int ldA, double *B, int ldB, double *C, int ldC);
```

ブロックLU分解

- 行列Aを小行列に分解し、その小行列単位でLU分解する方法。LU分解と行列積で実現可能。

目標

・ブロック化することで、メモリアクセスの向上。
さらに行列積を利用することで、LU分解の性能を向上させる。



- bk：幅のことをブロック幅と言う。
- ブロック幅を用いて設計されたアルゴリズムをブロック化アルゴリズムという。

ブロックLU分解の仕組み

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11}U_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

($A_{11} = L_{11}U_{11}$と普通にLU分解)

$$= \begin{bmatrix} L_{11} & O \\ L_{21} & I \end{bmatrix} \begin{bmatrix} I & O \\ O & S \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ O & I \end{bmatrix}, \quad S \equiv A_{22} - L_{21}U_{12}$$

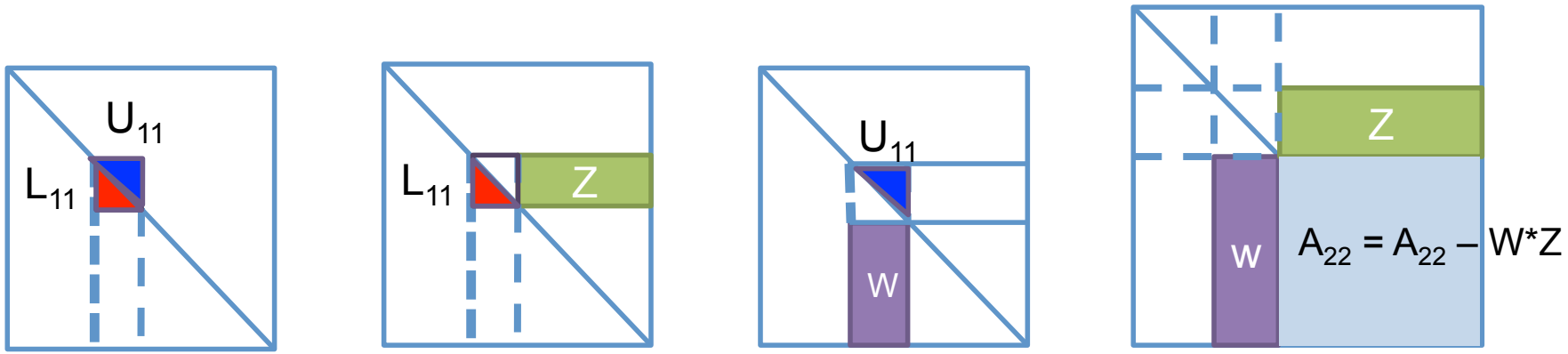
$$= \begin{bmatrix} L_{11} & O \\ L_{21} & I \end{bmatrix} \begin{bmatrix} I & O \\ O & L_{22}U_{22} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ O & I \end{bmatrix}$$

以下、$S = L_{22}U_{22}$のように
再帰的にブロックLU分解する。

$$= \begin{bmatrix} L_{11} & O \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ O & U_{22} \end{bmatrix}$$

ブロックLU分解のアルゴリズム

1. $A_{11} = L_{11}U_{11}$とLU分解する。
2. 行列方程式 $L_{11}Z = A_{12}$ をZ について解く。($Z = U_{12}$)
3. 行列方程式 $WU_{11} = A_{21}$ をW について解く。($W = L_{21}$)
4. $A_{22} = A_{22} – W*Z$ と更新



実習：ブロックLU分解のプログラム作成

1. 以前に作成したLU分解（MATLAB）を基にし、ブロックLU分解のMATLABプログラムを作成しよう。
2. それができたら、部分軸交換付きのブロックLU分解について考えよう。
3. さらに、C言語やFortranに移植しよう。

以下の性能を比較・検証し、考察してみましょう。
(1) シリアル処理のLU分解
(2) ブロックLU分解、
(3) ブロックLU分解（行列積の並列化）

→余裕のある人は、アンローリング等も試してみてください。