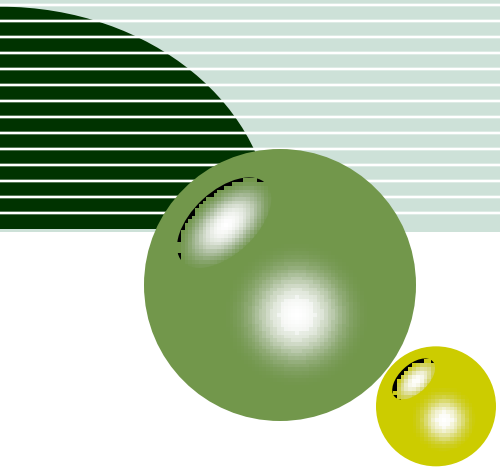
分散メモリ型並列計算機上での LU分解の並列化

280867018　青山俊介
280867050　加藤雅之
280867190　西田成志



Outline

- LU分解
- LU分解のブロック化
  - BLAS3の適用
- 分散メモリ型計算機上での並列化
  - ブロックサイクリック列分割による並列化
  - 双方向のブロックサイクリック分割による並列化
- 結果
- 結論

LU分解

- 連立1次方程式

$$A\boldsymbol{x} = \boldsymbol{b}$$

- 行列AのLU分解

$$A = LU$$

$$LU = \begin{bmatrix} 1 & & \\ 1 & & 0 \\ & \ddots & \\ * & 1 & \\ & & 1 \end{bmatrix} \begin{bmatrix} * & & \\ * & & * \\ & \ddots & \\ 0 & & * \\ & & * \end{bmatrix}$$

$$LU\boldsymbol{x} = \boldsymbol{b}$$

$$\begin{cases} L\boldsymbol{y} = \boldsymbol{b} \\ U\boldsymbol{x} = \boldsymbol{y} \end{cases}$$

⇒ 連立1次方程式を簡単に解くことができる

BLASにおけるデータ再利用性と並列粒度

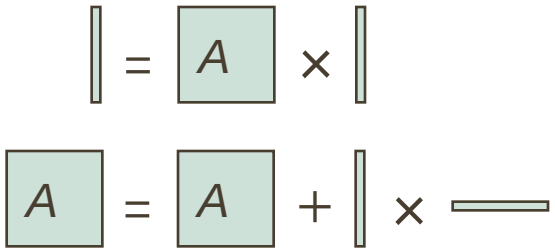
（講義資料より引用）

BLAS1

- 演算量： $O(N)$，　データ量： $O(N)$
- データ再利用性： なし
- 並列粒度： $O(N/p)$ （N: ベクトルの次元，p: プロセッサ台数）

BLAS2

- 演算量： $O(N)$，　データ量： $O(N)$
- ベクトルデータのみ再利用性あり
- 並列粒度： $O(N^2/p)$

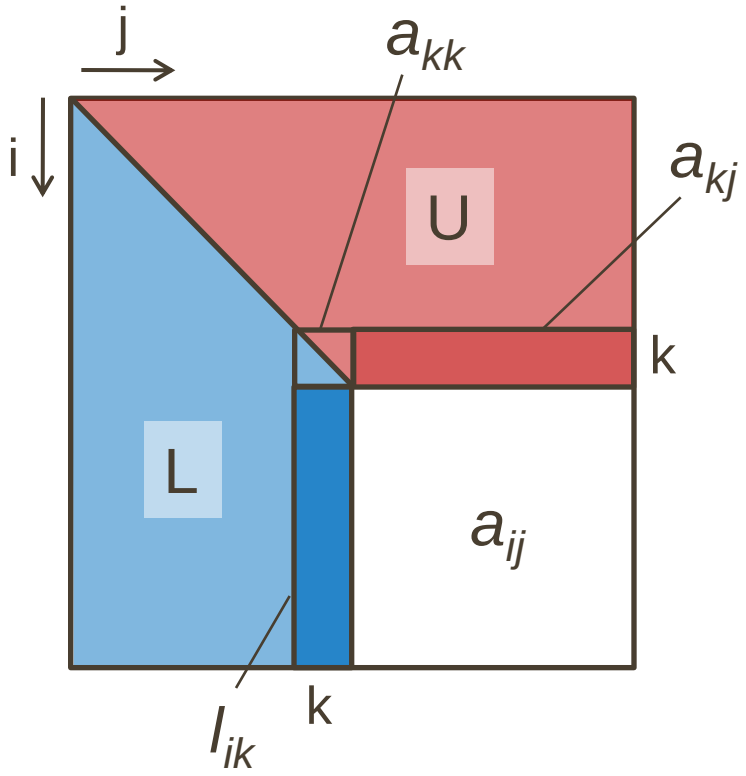


BLAS3

- 演算量： $O(N^3)$，　データ量： $O(N^2)$
- **$O(N)$回のデータ再利用**が可能
- 並列粒度： $O(N^3/p)$

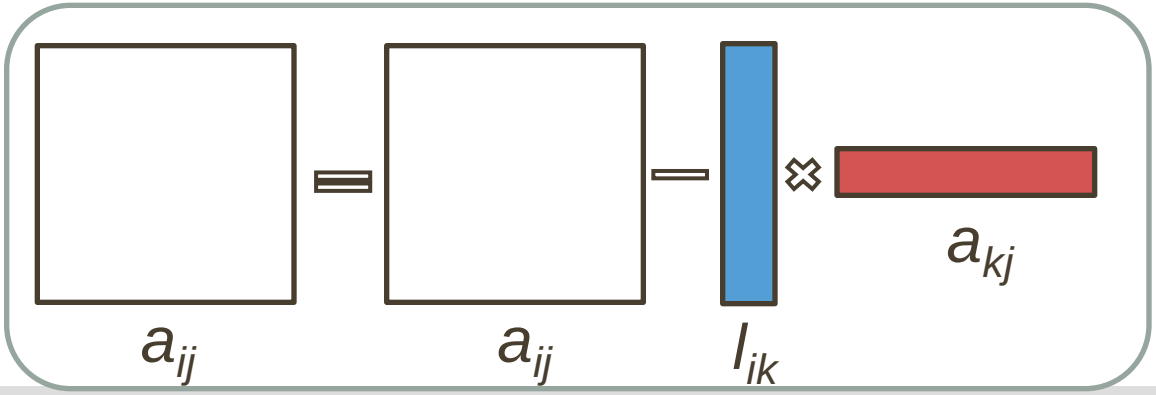
演算をできる限り **BLAS2, 3**で行うことが
高性能化のポイント

ブロック化されていないLU分解



□アルゴリズム

```
do k = 1, n
    do i = k+1, n
        l_ik = a_ik / a_kk
    end do
    do j = k+1, n
        do i = k+1, n
            a_ij = a_ij - l_ik a_kj
        end do
    end do
end do
```

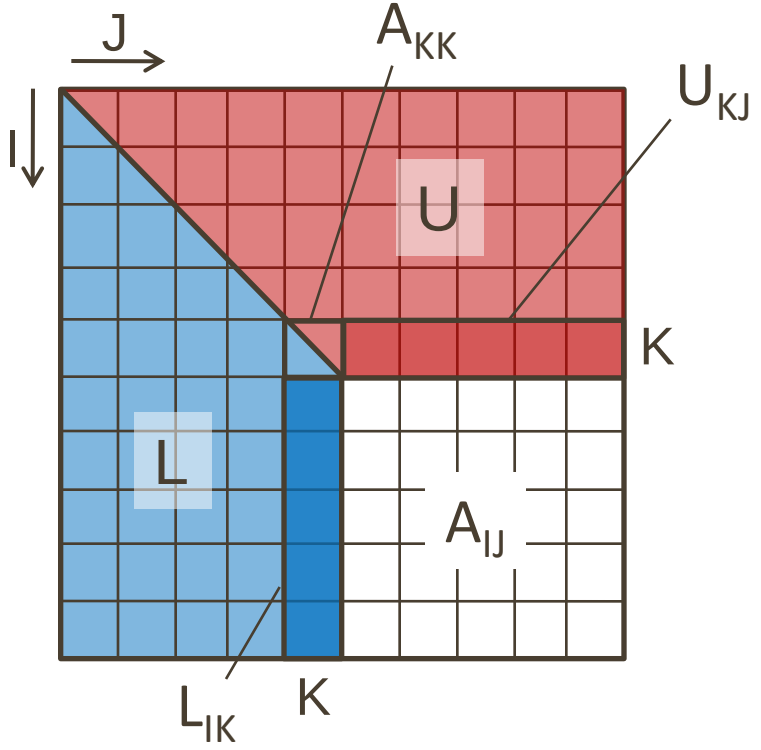


行列とベクトルの演算

⇩

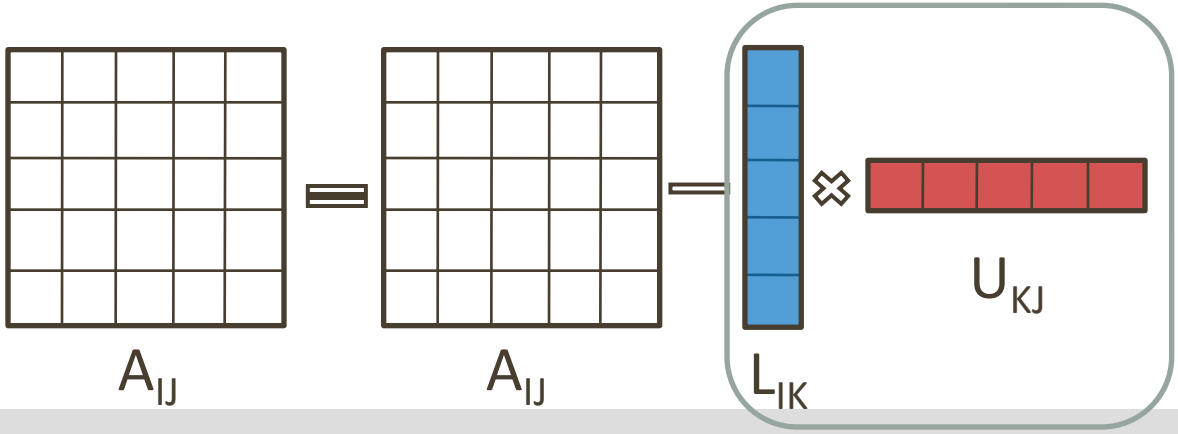
BLAS2を適用

LU分解のブロック化



- 行列をN×N個のブロックに分割
- 第Kブロック段での処理

1. $A_{KK} = L_{KK}U_{KK}$ を計算(LU 分解)
2. $L_{IK} = A_{IK}U^{-1}{}_{KK}$(I = K + 1, ・・・,N)を計算
3. $U_{KJ} = L^{-1}{}_{KK}A_{KJ}$ (J = K + 1, ・・・,N)を計算
4. $A_{IJ} := A_{IJ} - L_{IK}U_{KJ}$ を計算



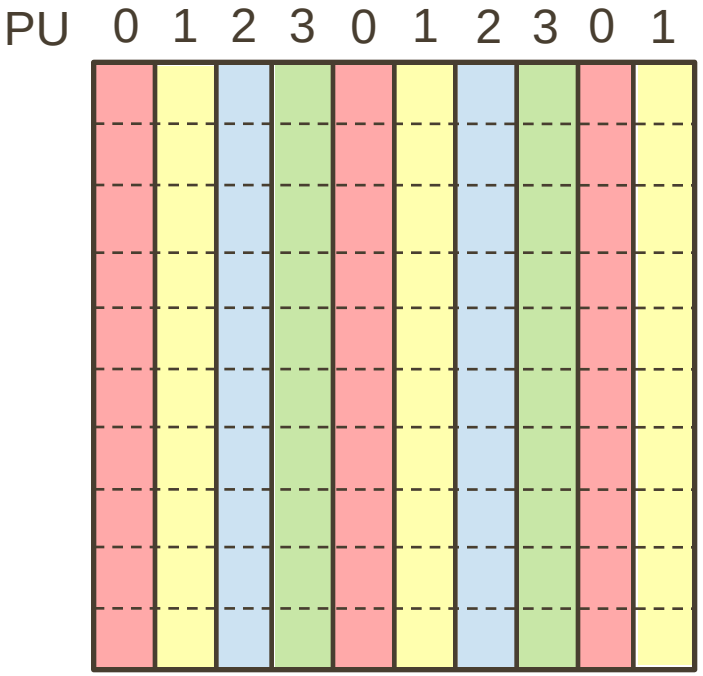
行列と行列の演算

⇩

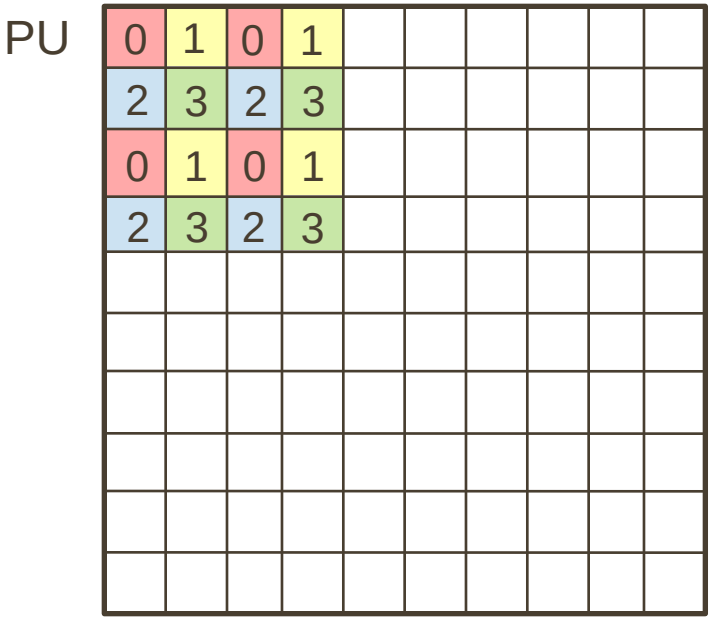
BLAS3を適用

分散メモリ型計算機上での並列化

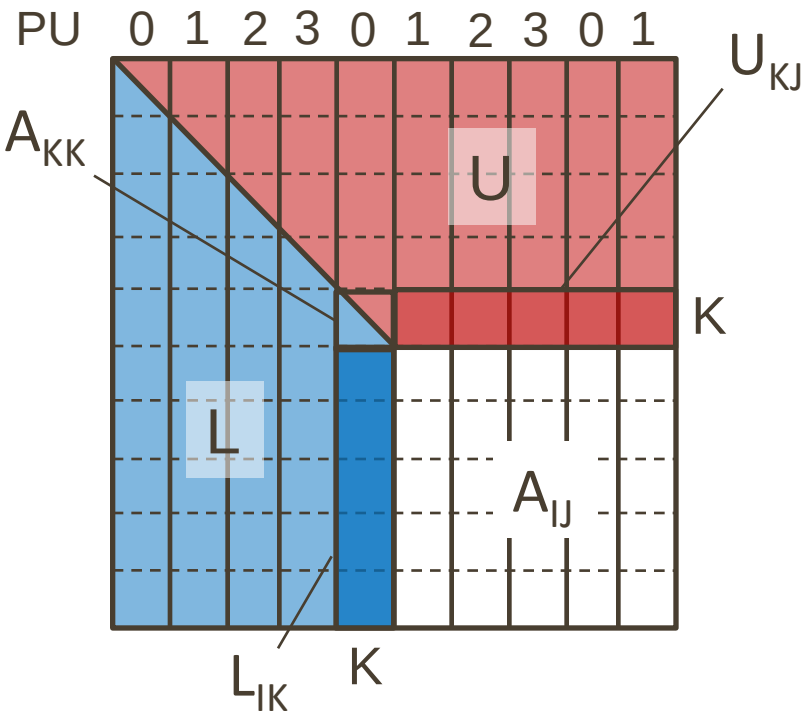
■ブロックサイクリック列分割



■双方向のブロックサイクリック分割



ブロックサイクリック列分割による並列化



□第Kブロック段での処理

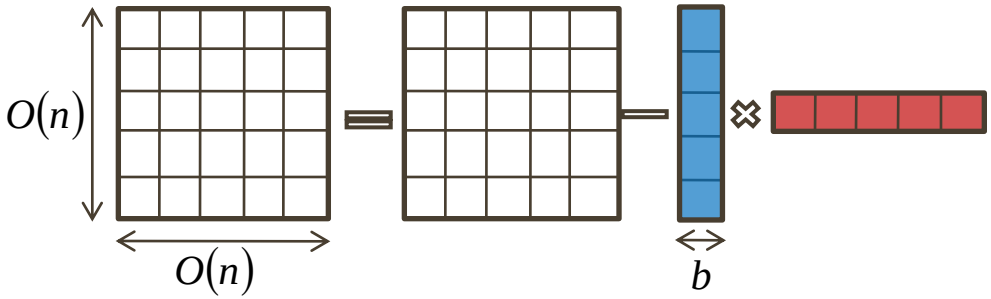
1. 第k 列を持つプロセッサは
   $A_{KK} = L_{KK}U_{KK}$ を計算(LU 分解)
2. 第k 列を持つプロセッサは
   $L_{IK} = A_{IK}U^{-1}{}_{KK}(I = K + 1, \cdots, N)$ を計算
3. 第K 列を持つプロセッサは$L_{IK}(I = K, \cdots, N)$を全プロセッサに送信(ブロードキャスト)
4. 各プロセッサはL_{IK} を受け取る.
5. 各プロセッサは自分の担当する列について
   $U_{KJ} = L^{-1}{}_{KK}A_{KJ}$ を計算.
6. 各プロセッサは自分の担当する列について
   $A_{IJ} := A_{IJ} - L_{IK}U_{KJ}$ を計算.

並列化の効果

□演算量

– 1ブロック段・1PUあたり

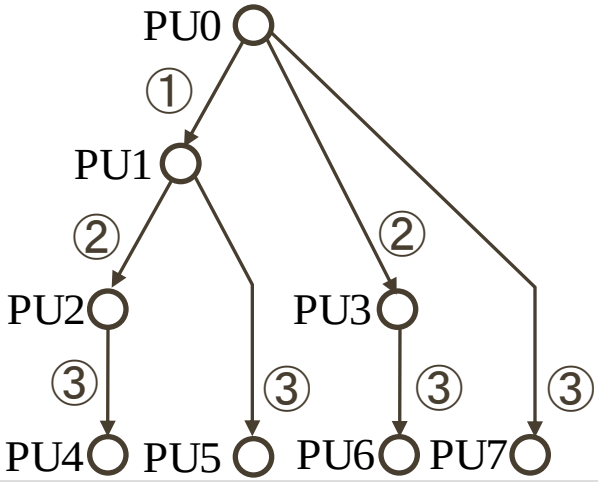
$$\frac{O\left(n^2 \cdot b\right)}{P} = O\left(\frac{n^2 \cdot b}{P}\right)$$

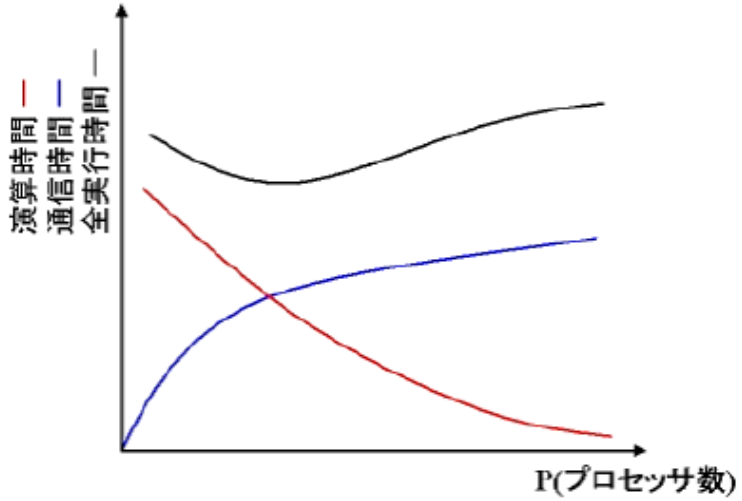


□通信量

– データ量　$O(n) \times b = O(nb)$

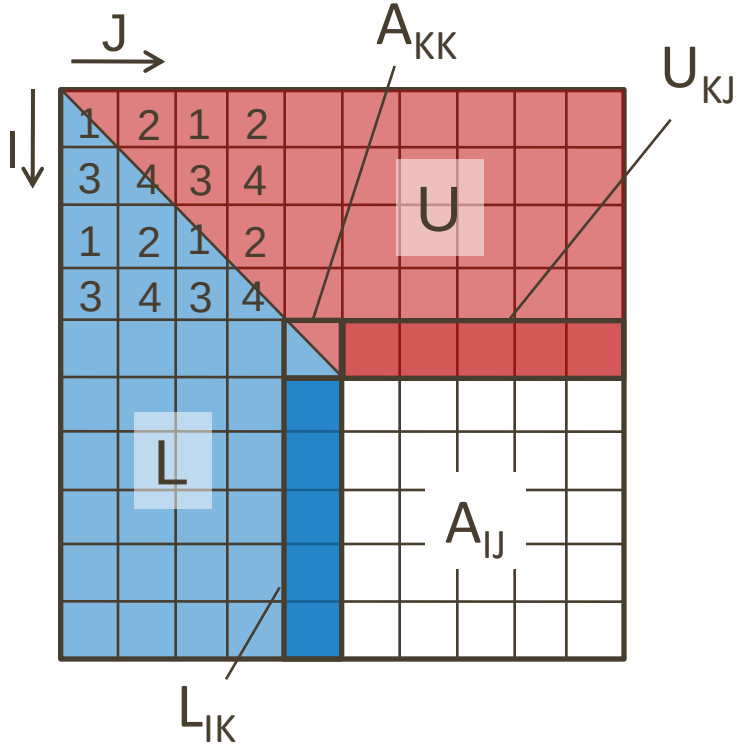
– P個の場合　$O(nb) \times \log_2 P = O(nb \cdot \log_2 P)$





（講義資料より引用）

双方向のブロックサイクリック分割による並列化



□第Kブロック段での処理

1. A_{KK} を持つPU(プロセッサ) は$A_{KK} = L_{KK}U_{KK}$とLU 分解
2. A_{KK} を持つPUはU_{KK} を縦方向にブロードキャスト
3. 第K 列担当のPU は自分の担当するブロックについて$L_{IK} = A_{IK}U^{-1}_{KK}$ を計算
4. A_{KK} を持つPUはL_{KK} を横方向にブロードキャスト
5. 第K 行担当のPU は自分の担当するブロックについて$U_{KJ} = L^{-1}_{KK}A_{KJ}$ を計算
6. 第K 列担当のPU は自分の持つL_{IK} を横方向にブロードキャスト
7. 第K 行担当のPU は自分の持つU_{KJ} を縦方向にブロードキャスト
8. 各PU は自分の担当するブロックについて$A_{IJ} := A_{IJ} - L_{IK}U_{KJ}$ を計算

並列化の効果

□演算量

– 1ブロック段・1PUあたり

$$\frac{O\left(n^2 \cdot b\right)}{P} = O\left(\frac{n^2 \cdot b}{P}\right)$$

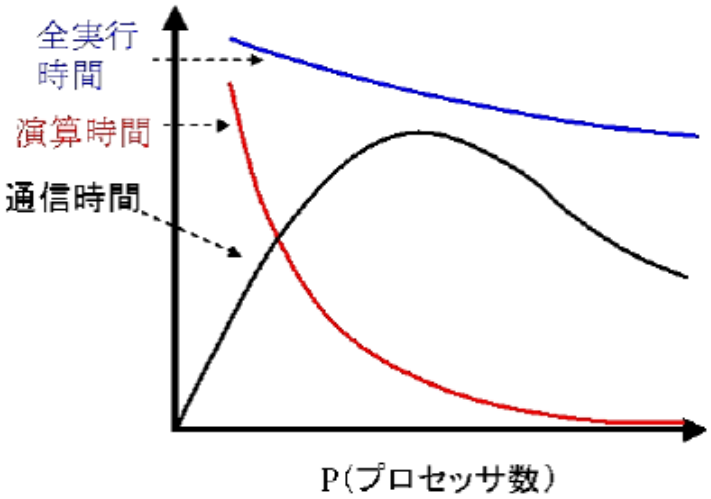
□通信量

– データ量（1PUあたり） $O\left(\frac{nb}{\sqrt{P}}\right)$

– 送り先の数 $\sqrt{P}$

– 送り回数 $\log_2 \sqrt{P}$

– 通信時間 $O\left(\frac{nb}{\sqrt{P}}\right) \times \log_2 \sqrt{P} = O\left(nb\frac{\log_2 P}{\sqrt{P}}\right) \to 0 \ (P \to \infty)$



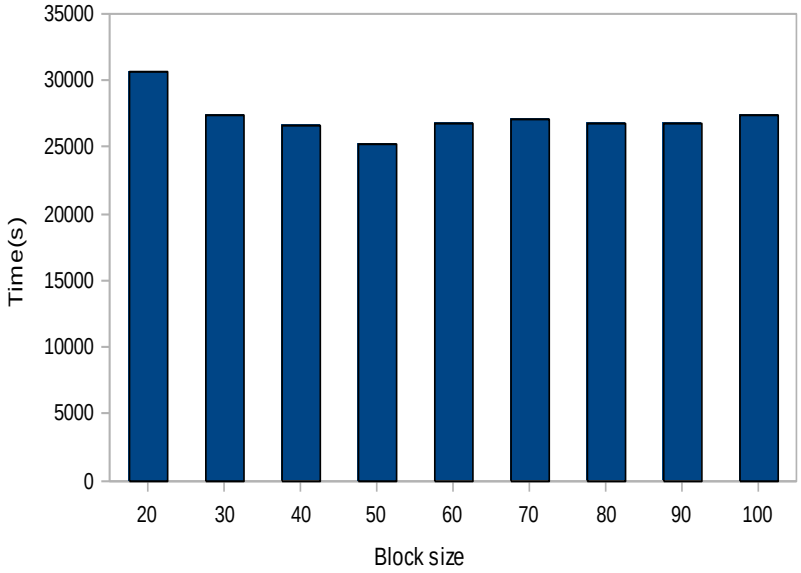
（講義資料より引用）

結果

□行列をブロック化してBLAS3を適用(行列サイズn=5000)

ブロック化適用前	498.3(s)
ブロック化適用後 (ブロックサイズ b=50)	25.2(s)

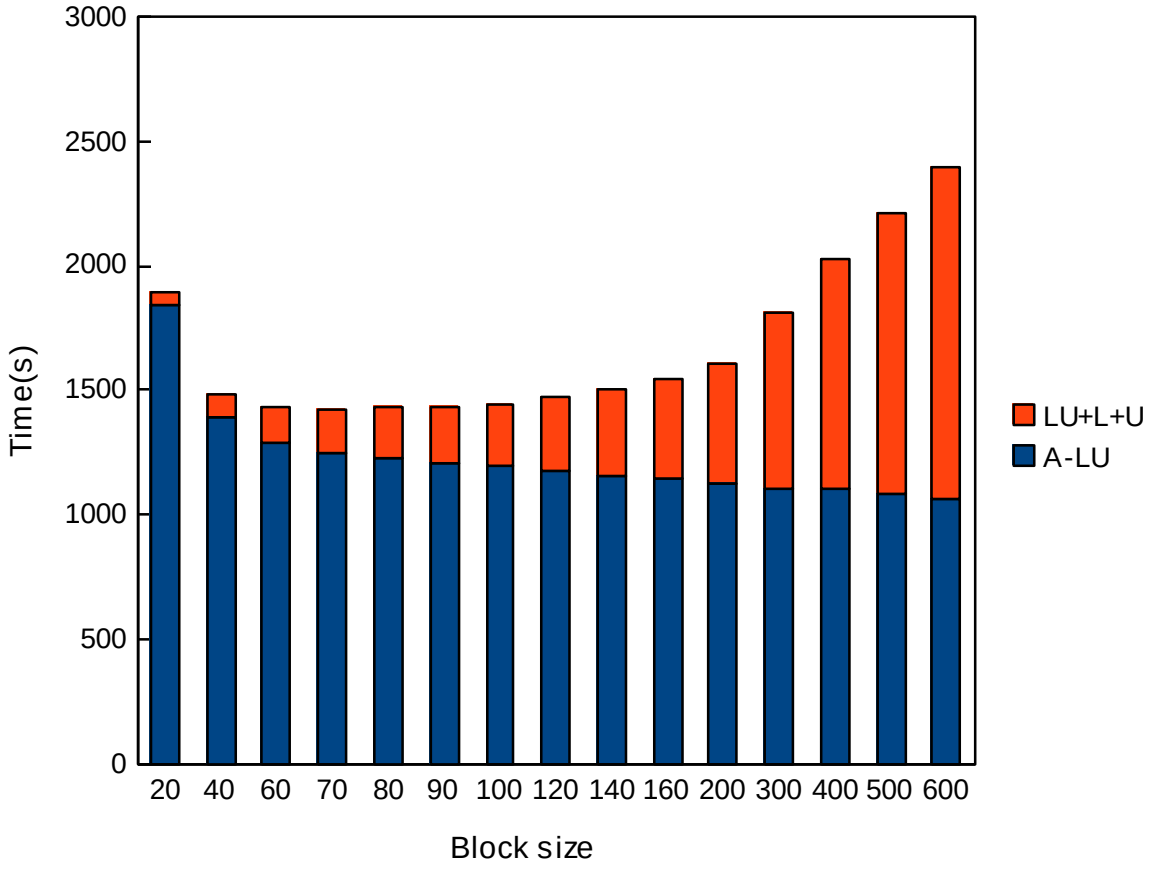
約20倍の高速化



計算機環境　Fujitsu PRIMEPOWER HPC 2500

結果

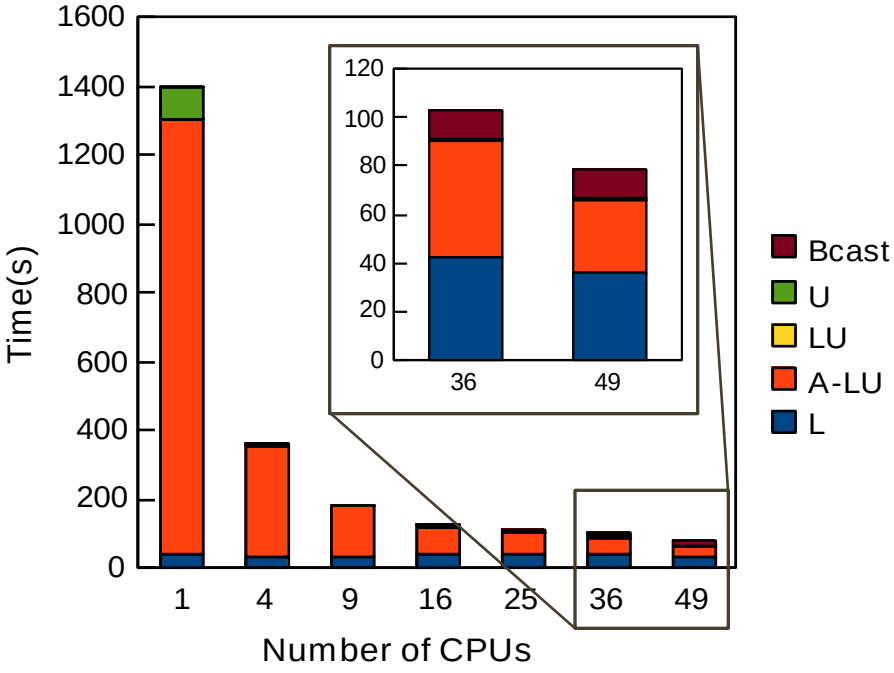
- 行列サイズn=20000の場合



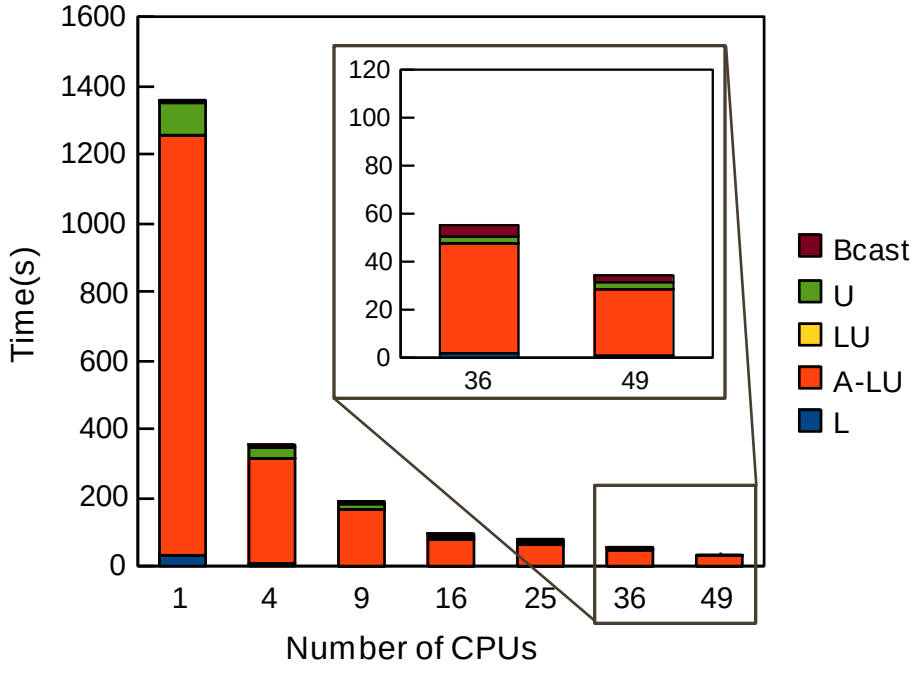
結果

□MPIによる並列化

行列サイズ　　n=20000
ブロックサイズ　b=60



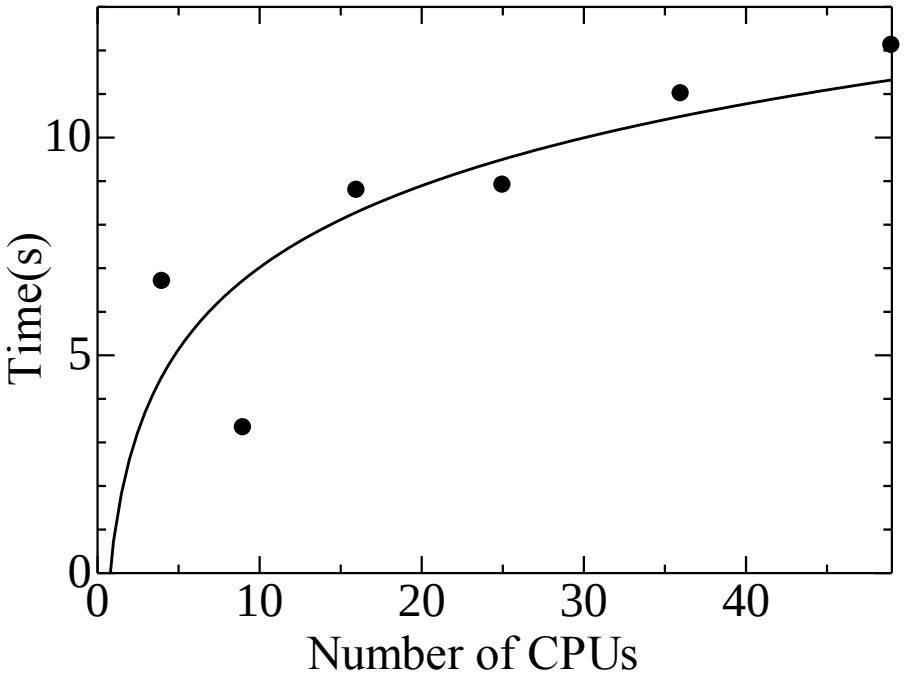
ブロックサイクリック列分割



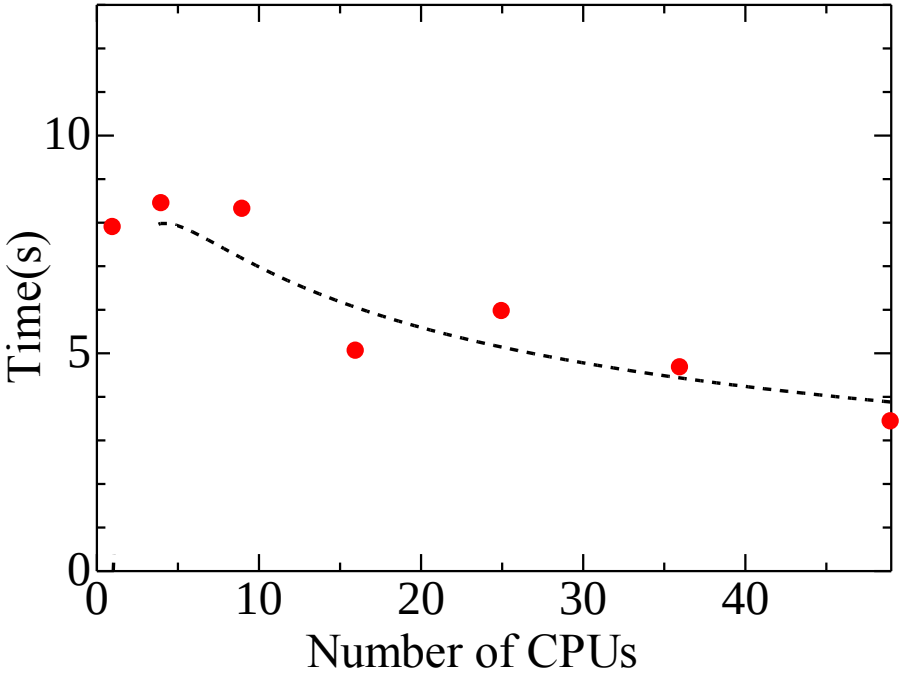
双方向のブロックサイクリック分割

結果

□通信時間



ブロックサイクリック列分割

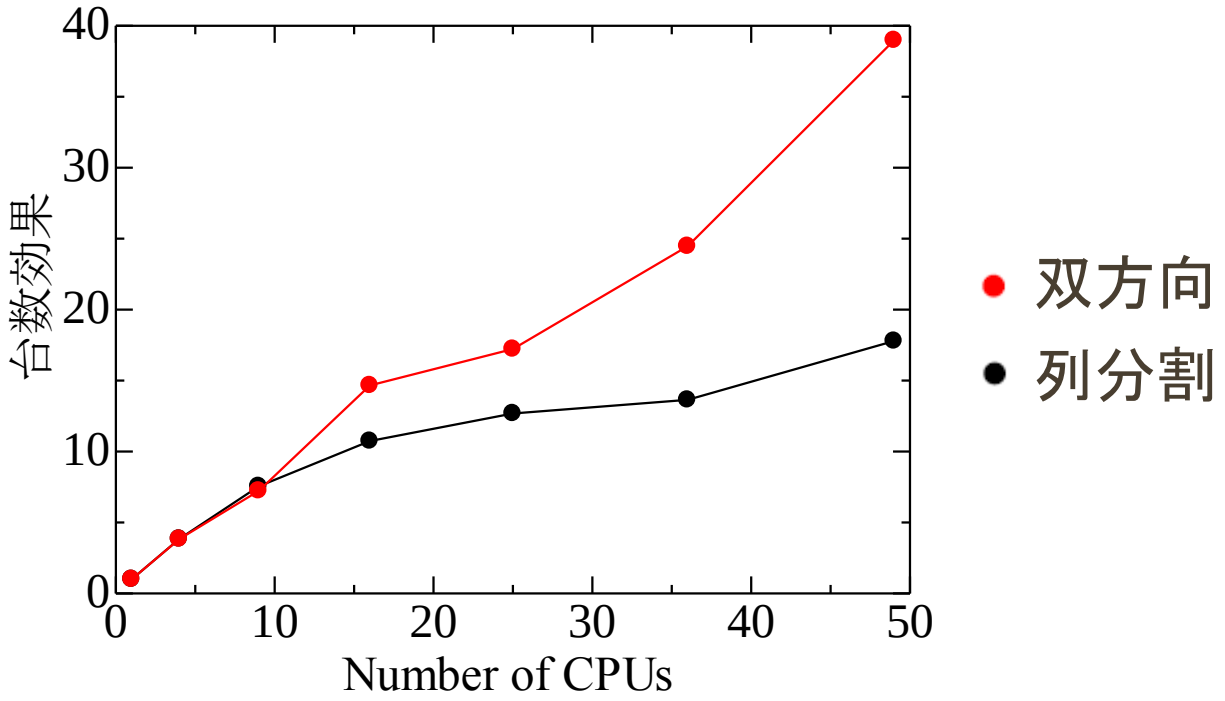


双方向のブロックサイクリック分割

結果

□台数効果

$$\text{台数効果} = \frac{\text{1CPUでの経過時間}}{\text{経過時間}}$$



結論

- LU分解をブロック化し、性能評価を行った
  - BLAS3を適用することにより計算の高速化が実現できた
- MPIを用いて列方向に並列化し、性能評価を行った
  - PU数を多くしたときに並列効果があまり得られない
  - 通信量が増加するため、さらにPU数を大きくした場合、計算時間が遅くなることが予想される
- MPIを用いて双方向に並列化し、性能評価を行った
  - 列方向の場合のボトルネックが解消されたため、PU数を大きくしたときにも十分な並列効果が得られた