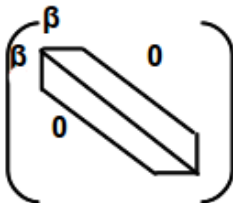


2.5.4. $c = Ab$ for Banded Matrix



- Bandwidth β (symmetric)
- $2\beta+1$ diagonals: main diag. + β subdiag. + β superdiag.
- $\beta = 1$: tridiagonal



Notation: Banded Matrices A and \tilde{A}

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1,\beta+1} & 0 & \cdots & 0 \\ \vdots & a_{22} & \ddots & \ddots & \cdots & \vdots \\ a_{\beta+1,1} & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & a_{n-\beta,n} \\ \vdots & \vdots & \ddots & \ddots & a_{n-1,n-1} & \vdots \\ 0 & \cdots & 0 & a_{n,n-\beta} & \cdots & a_{nn} \end{pmatrix} \rightarrow$$



Notation: Banded Matrices A and \tilde{A}

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1,\beta+1} & 0 & \cdots & 0 \\ \vdots & a_{22} & \ddots & \ddots & \cdots & \vdots \\ a_{\beta+1,1} & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & a_{n-\beta,n} \\ \vdots & \vdots & \ddots & \ddots & a_{n-1,n-1} & \vdots \\ 0 & \cdots & 0 & a_{n,n-\beta} & \cdots & a_{nn} \end{pmatrix} \rightarrow$$

$$\tilde{A} = \begin{pmatrix} \tilde{a}_{10} & \cdots & \tilde{a}_{1,\beta} & 0 & \cdots & 0 \\ \vdots & \tilde{a}_{20} & \ddots & \ddots & \cdots & \vdots \\ \tilde{a}_{\beta+1,-\beta} & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \tilde{a}_{n-\beta,\beta} \\ \vdots & \vdots & \ddots & \ddots & \tilde{a}_{n-1,0} & \vdots \\ 0 & \cdots & 0 & \tilde{a}_{n,-\beta} & \cdots & \tilde{a}_{n,0} \end{pmatrix}$$



$c = Ab$ for Banded Matrix

Storing entries diagonalwise: $n(2\beta + 1)$ matrix instead of n^2 .

$$\tilde{a}_{i,s} = a_{i,i+s} \quad \text{for row } i = 1, \dots, n$$
$$1 \leq i + s \leq n \quad \text{and} \quad -\beta \leq s \leq \beta \quad \text{and} \quad 1 \leq i \leq n$$



$c = Ab$ for Banded Matrix

Storing entries diagonalwise: $n(2\beta + 1)$ matrix instead of n^2 .

$$\tilde{a}_{i,s} = a_{i,i+s} \quad \text{for row } i = 1, \dots, n$$

$$1 \leq i + s \leq n \quad \text{and} \quad -\beta \leq s \leq \beta \quad \text{and} \quad 1 \leq i \leq n$$

$$1 - i \leq s \leq n - i \quad \text{and} \quad -\beta \leq s \leq \beta$$

↓ in row i

$$s \in [l_i, r_i] = [\max\{-\beta, 1 - i\}, \min\{\beta, n - i\}]$$



$c = Ab$ for Banded Matrix

Storing entries diagonalwise: $n(2\beta + 1)$ matrix instead of n^2 .

$$\tilde{a}_{i,s} = a_{i,i+s} \quad \text{for row } i = 1, \dots, n$$

$$1 \leq i + s \leq n \quad \text{and} \quad -\beta \leq s \leq \beta \quad \text{and} \quad 1 \leq i \leq n$$

$$1 - i \leq s \leq n - i \quad \text{and} \quad -\beta \leq s \leq \beta$$

↓ *in row i*

$$s \in [l_i, r_i] = [\max\{-\beta, 1 - i\}, \min\{\beta, n - i\}]$$

$$1 - s \leq i \leq n - s \quad \text{and} \quad 1 \leq i \leq n$$

↓ *in diag. s*

$$i \in [\tilde{l}_s, \tilde{r}_s] = [\max\{1, 1 - s\}, \min\{n, n - s\}]$$



Computation of the mtx-vec product based on storage scheme on vector CPUs

$$\text{For } i = 1, \dots, n: c_i = A_{i \bullet} \cdot b = \sum_j a_{ij} b_j = \sum_{s=l_i}^{r_i} a_{i,i+s} b_{i+s} = \sum_{s=l_i}^{r_i} \tilde{a}_{i,s} b_{i+s}$$

- General TRIAD, no SAXPY:

```
for s = -β : β
```

```
    for i = max{1 - s, 1} : min{n - s, n}
```

```
        c_i = c_i + \tilde{a}_{i,s} b_{i+s}
```

```
    end
```

```
end
```



Computation of the mtx-vec product based on storage scheme on vector CPUs

$$\text{For } i = 1, \dots, n: c_i = A_{i\bullet} \cdot b = \sum_j a_{ij} b_j = \sum_{s=l_i}^{r_i} a_{i,i+s} b_{i+s} = \sum_{s=l_i}^{r_i} \tilde{a}_{i,s} b_{i+s}$$

- General TRIAD, no SAXPY:

```
for s = -β : β
```

```
  for i = max{1 - s, 1} : min{n - s, n}
```

```
    c_i = c_i + \tilde{a}_{i,s} b_{i+s}
```

```
  end
```

```
end
```

- or, partial DOT-product:

```
for i = 1 : n
```

```
  for s = max{-β, 1 - i} : max{β, n - i}
```

```
    c_i = c_i + \tilde{a}_{i,s} b_{i+s}
```

```
  end
```

```
end
```

- Sparsity \Rightarrow less operations, but also loss of efficiency.



Band Ab in Parallel

- Partitioning:

$$\langle 1, n \rangle = \bigcup_{r=1}^R I_r, \text{ disjoint}$$

for $i \in I_r$

$$c_i = \sum_{s=l_i}^{r_i} \tilde{a}_{is} b_{i+s}$$

end

- Processor P_r gets rows to index set $I_r := [m_r, M_r]$ in order to compute its part of the final vector c .
- What part of vector b does processor P_r need in order to compute its part of c ?



Band Ab in Parallel

- Necessary for l_r : $b_j = b_{i+s}$:

$$j = i + s \geq m_r + \max\{-\beta, 1 - m_r\} = \max\{m_r - \beta, 1\}$$

$$j = i + s \leq M_r + r_{M_r} = M_r + \min\{\beta, n - M_r\} = \min\{M_r + \beta, n\}$$

- Processor P_r with index set l_r needs from b the indices

$$j \in [\max\{1, m_r - \beta\}, \min\{n, M_r + \beta\}]$$

$$\left(\begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \\ \text{A} \end{array} \right) \left(\begin{array}{c} | \\ | \\ | \\ \text{b} \end{array} \right)$$



2.6. Analysis of Matrix-Matrix Product

$$A = (a_{ij})_{\substack{i=1,\dots,n \\ j=1,\dots,m}} \in \mathbb{R}^{n \times m}, \quad B = (b_{ij})_{\substack{i=1,\dots,m \\ j=1,\dots,q}} \in \mathbb{R}^{m \times q},$$

$$C = AB = (c_{ij})_{\substack{i=1,\dots,n \\ j=1,\dots,q}} \in \mathbb{R}^{n \times q}$$

for $i = 1 : n$

 for $j = 1 : q$

$$c_{ij} = \sum_{k=1}^m a_{ik} b_{kj}$$

 end

end

$$\begin{pmatrix} * & * & * \\ a_{i1} & \cdots & a_{im} \\ * & * & * \end{pmatrix} \cdot \begin{pmatrix} * & \boxed{b_{1j}} & * \\ * & \vdots & * \\ * & \boxed{b_{mj}} & * \end{pmatrix} = \begin{pmatrix} * & * & * \\ * & \boxed{c_{ij}} & * \\ * & * & * \end{pmatrix}$$



2.6.1. Vectorization

- Algorithm 1: (ijk)-Form:

```
for  $i = 1 : n$ 
```

```
  for  $j = 1 : q$ 
```

```
    for  $k = 1 : m$ 
```

```
       $c_{ij} = c_{ij} + a_{ik} b_{kj}$  } DOT-product of length  $m$ 
```

```
    end
```

```
  end
```

```
end
```

$c_{ij} = A_{i\bullet} \bullet B_{\bullet j}$ for all i, j

- All entries c_{ij} are fully computed, one after another.
- Access to A and C is rowwise, to B columnwise (depends on inner most loops!)



Other View on the Matrix-Matrix Product

Matrix A considered as combination of **columns** or **rows**

$$\begin{aligned}
 A &= A_1 e_1^T + \dots + A_m e_m^T = (A_1 \ 0 \ \dots) + (0 \ A_2 \ 0 \ \dots) + \dots + (\dots \ 0 \ A_m) \\
 &= e_1 a_1 + \dots + e_n a_n = \begin{pmatrix} a_1 \\ 0 \\ \vdots \end{pmatrix} + \dots + \begin{pmatrix} \vdots \\ 0 \\ a_n \end{pmatrix}
 \end{aligned}$$

$$AB = \sum_{j=1}^n A_j e_j^T \sum_{k=1}^m e_k b_k = \sum_{k,j} A_j (e_j^T e_k) b_k = \sum_{k=1}^m \underbrace{A_k b_k}_{\text{full } n \times q \text{ matrices}}$$

as a sum of full matrices $A_k b_k$ by outer product of the k th column of A and the k th row of B .



Algorithm 2: (jki)-Form

```

for j=1,...,q
  for k=1,...,m
    for i=1,...,n
       $c_{ij} = c_{ij} + a_{ik} b_{kj}$ 
    end
  end
end
end

```

$\left. \begin{array}{l} \text{SAXPY} \\ \text{GAXPY} \end{array} \right\}$

- Vector update: $c_{\bullet j} = c_{\bullet j} + a_{\bullet k} b_{kj}$
- Sequence of SAXPYs for the same vector: $c_{\bullet j} = \sum_k b_{kj} a_{\bullet k}$
- C computed columnwise; access to A columnwise. Access to B columnwise, but delayed.



Algorithm 3: (kji)-Form

```

for k=1,...,m
  for j=1,...,q
    for i=1,...,n
       $c_{ij} = c_{ij} + a_{ik}b_{kj}$ 
    end
  end
end
end

```

} SAXPY

- Vector update: $c_{\bullet j} = c_{\bullet j} + a_{\bullet k}b_{kj}$
- Sequence of SAXPYs for different vectors $c_{\bullet j}$ (no GAXPY)
- Access to A columnwise. Access to B rowwise + delayed.
 C computed with intermediate values $c_{ij}^{(k)}$ which are computed columnwise.



Overview of Different Forms

	ijk Alg. 1	ikj	kij	jik	jki Alg. 2	kji Alg. 3
Access to A by	row	—	—	row	column	column
Access to B by	column	row	row	column	—	—
Comput. of C	row	row	row	column	column	column
Computat ion of c_{ij}	direct	delayed	delayed	direct	delayed	delayed
Vector ope- ration	DOT	GAXPY	SAXPY	DOT	GAXPY	SAXPY
Vector length	m	q	q	m	n	n

Better: GAXPY (longer vector length).

Access to matrices according to storage scheme (rowwise or columnwise)



2.6.2. Matrix-Matrix Product in Parallel

$$\langle 1, n \rangle = \bigcup_{r=1}^R I_r, \quad \langle 1, m \rangle = \bigcup_{s=1}^S K_s, \quad \langle 1, q \rangle = \bigcup_{t=1}^T J_t$$

Distribute the blocks relative to index sets I_r , K_s , and J_t to processor array P_{rst} :

$$I_r \left(\begin{array}{c|c|c} & & \\ \hline & & \\ \hline & A_{rs} & \\ \hline & & \end{array} \right)_{K_s} \cdot \left(\begin{array}{c|c|c} & & \\ \hline & B_{st} & \\ \hline & & \end{array} \right)_{J_t} = \left(\begin{array}{c|c|c} & & \\ \hline & C_{rt}^{(s)} & \\ \hline & & \end{array} \right)_{J_t} I_r$$

1. Processor P_{rst} computes small matrix-matrix product. All processors in parallel: $C_{rt}^{(s)} = A_{rs} B_{st}$
2. Compute sum by fan-in in s :

$$C_{rt} = \sum_{s=1}^S C_{rt}^{(s)}$$



Mtx-Mtx in Parallel: Special Case $S = 1$

$$I_r \left(\begin{array}{c} \hline A_r \hline \end{array} \right) \cdot \left(\begin{array}{c} J_t \\ \hline B_t \hline \end{array} \right) = \left(\begin{array}{c} J_t \\ \hline \boxed{c_{rt}} \hline \end{array} \right) I_r$$

- Each processor P_{rt} can compute its part of c , c_{rt} , independently without communication.
- Each processor needs
 - full block of rows of A , relative to index set I_r , and
 - full block of columns of B , relative to index set J_t ,
 to compute c_{rt} relative to rows I_k and columns J_t .



Mtx-Mtx in Parallel: Special Case $S = 1$

$$I_r \begin{pmatrix} \text{---} \\ \text{---} A_r \text{---} \\ \text{---} \end{pmatrix} \cdot \begin{pmatrix} J_t \\ | \\ B_t \\ | \\ \text{---} \end{pmatrix} = \begin{pmatrix} J_t \\ | \\ \boxed{c_{rt}} \\ | \\ \text{---} \end{pmatrix} I_r$$

- With $n \cdot q$ processors each processor has to compute one DOT-product with $\mathcal{O}(m)$ parallel time steps.

$$c_{rt} = \sum_{k=1}^m a_{rk} b_{kt}$$

- Fan-in by $m \cdot nq$ additional processors for all DOT-products reduces number of parallel time steps to $\mathcal{O}(\log(m))$.



1D-Parallelization of $A \cdot B$

- 1D: p processors linear, each processor gets full A and column slice of B , computing the related column slice of $C = AB$

A, B_1

A, B_2

.....

A, B_{np}

- Communication: $N^2 p$ for A and $(N \cdot \frac{N}{p}) \cdot p = N^2$ for B

- Granularity: $\frac{N^3}{N^2(1+p)} = \frac{N}{1+p}$

- Blocking only in i , the columns of B !

for $i = 1 : n$

 for $j = 1 : n$

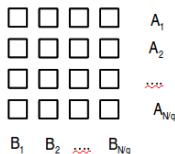
 for $k = 1 : n$

$$C_{j,i} = C_{j,i} + A_{j,k} B_{k,i}$$



2D-Parallelization of $A \cdot B$

- 2D: p processors square, $q := \sqrt{p}$, each proc. gets row slice of A and column slice of B computing full subblock of $C = AB$



- Communication: $N^2(p/\sqrt{p})$ for A and $N^2\sqrt{p}$ for B

- Granularity: $\frac{N^3}{2N^2\sqrt{p}} = \frac{N}{2\sqrt{p}}$

- Blocking in i and j , the columns of B and the rows of A !

for $i = 1 : n$

 for $j = 1 : n$

 for $k = 1 : n$

$$C_{j,i} = C_{j,i} + A_{j,k}B_{k,i}$$



3D-Parallelization $A \cdot B$

- 3D: p processors cubic, each processor gets subblock of A and subblock of B , computing part of subblock of $C = AB$.

Additional fan-in to collect parts to full subblock of C . ($q = p^{\frac{1}{3}}$).

- Communication:

$$N^2 p^{\frac{1}{3}} \text{ for } A \text{ and for } B \left(= p \cdot \frac{N^2}{p^{\frac{2}{3}}} = p \cdot \text{blocksize} \right), \text{ fan-in: } N^2 p^{\frac{1}{3}}$$

- Granularity: $\frac{N^3}{3N^2 p^{\frac{1}{3}}} = \frac{N}{3p^{\frac{1}{3}}}$

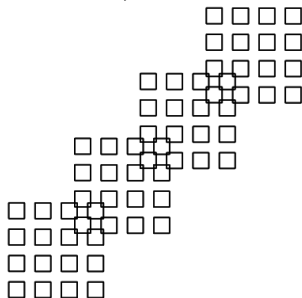
- Blocking in i, j , and k !

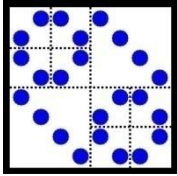
for $i = 1 : n$

 for $j = 1 : n$

 for $k = 1 : n$

$$C_{j,i} = C_{j,i} + A_{j,k} B_{k,i}$$





Vectorization of GE

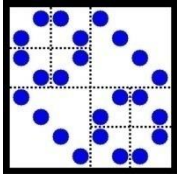
(kij)-form (standard form):

```
For k = 1 : n-1
  For i = k+1 : n
     $l_{i,k} = a_{i,k} / a_{k,k}$ 
  end
  For i = k+1 : n
    For j = k+1 : n
       $a_{i,j} = a_{i,j} - l_{i,k} a_{k,j}$ 
    end
  end
end
end
```

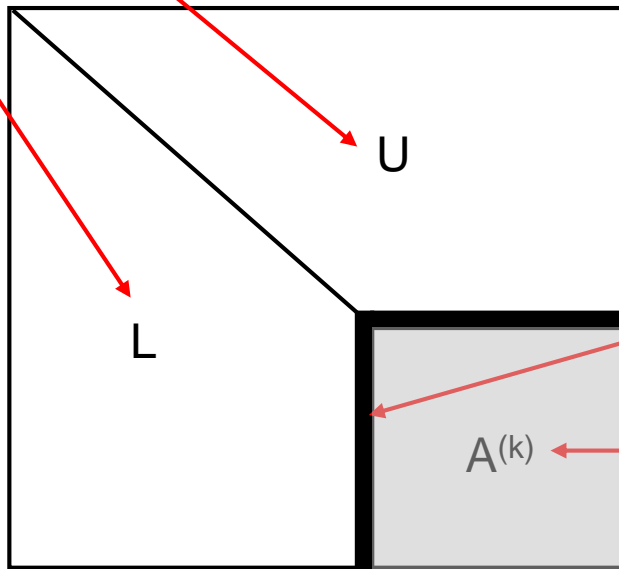
} Vector operation
 $\alpha \cdot \vec{x}$

} SAXPY in rows a_i and a_k } No GAXPY

U computed rowwise, L columnwise.



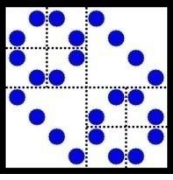
already computed, remains unchanged,
not used anymore



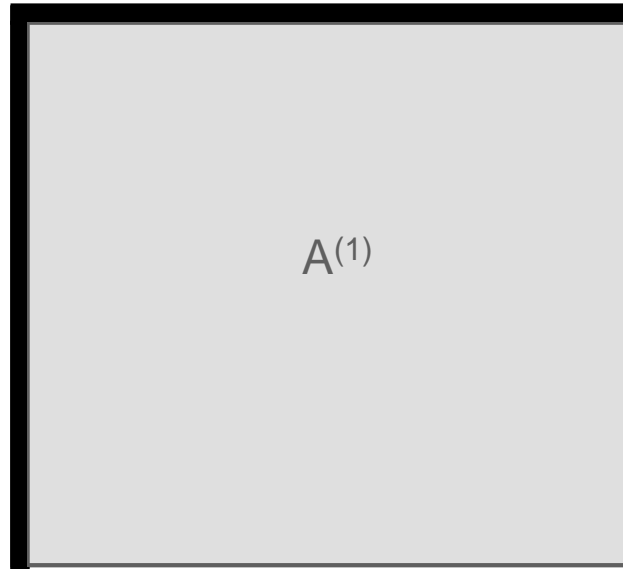
newly computed

updated in every step

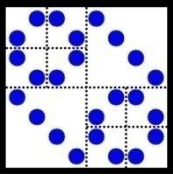
Standard (kij) form is also called “rightlooking GE”.



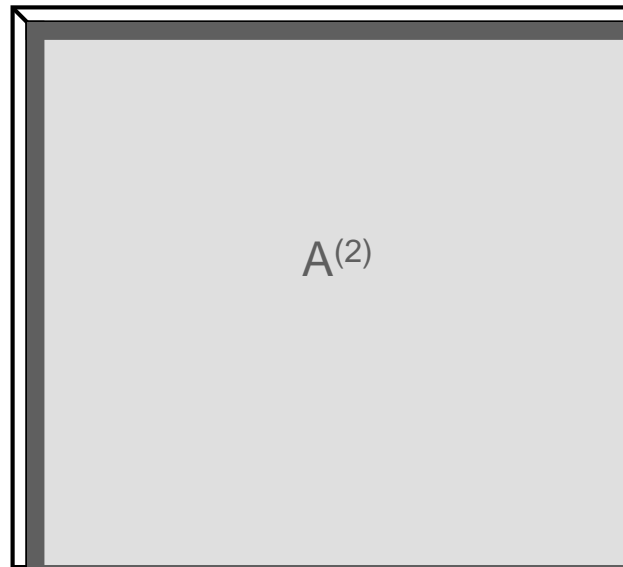
First Elimination step:



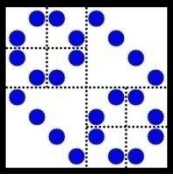
Compute first column of L
Update $A^{(1)}$



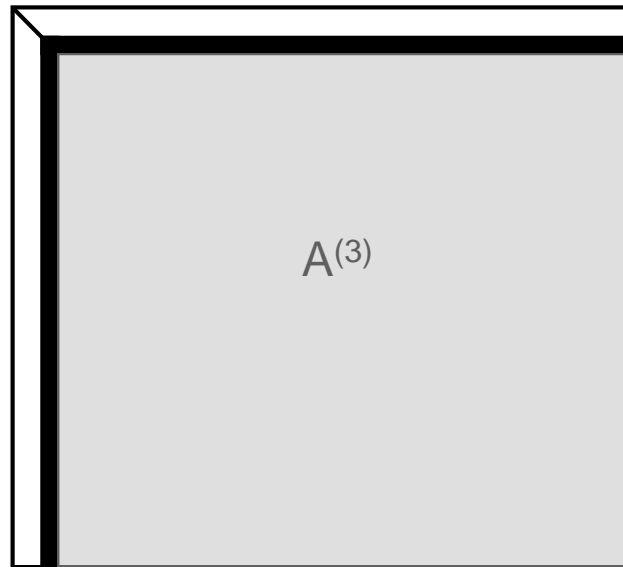
Second step:



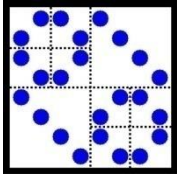
Compute second column of L
Update $A^{(2)}$



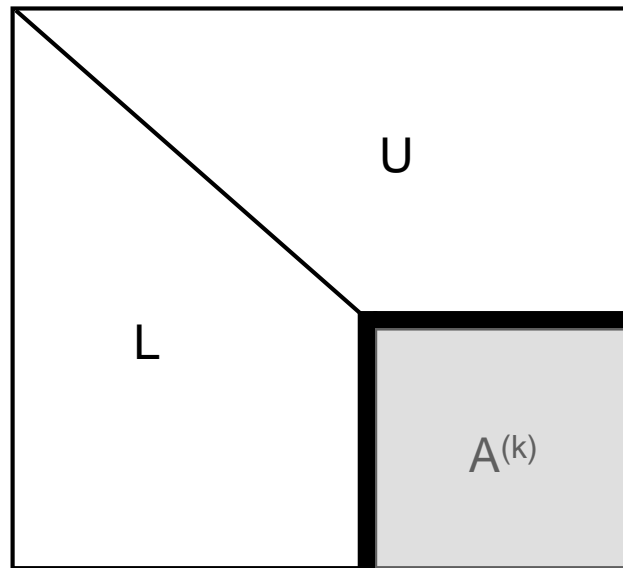
Second step:



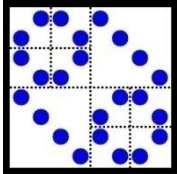
Compute third column of L
Update $A^{(3)}$



k-1st step:



Compute k-th column of L
Update $A^{(k)}$



Rules for different i,j,k forms:

In the following we again interchange the kij loops.

Necessary conditions:

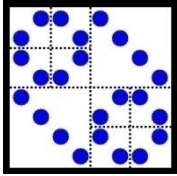
$$1 \leq k < i \leq n$$
$$1 \leq k < j \leq n$$

Furthermore:

Innermost index i,j, or k determines whether the computation is done row, column, or block-wise.

Outermost index shows how the final parts are derived.

Weights l_{jk} have to be computed before they are used to eliminate related entries.



(ikj)-form:

$$1 \leq k < i \leq n$$

$$1 \leq k < j \leq n$$

For $i = 2 : n$

For $k = 1 : i-1$

$$l_{i,k} = a_{i,k} / a_{k,k};$$

For $j = k+1 : n$

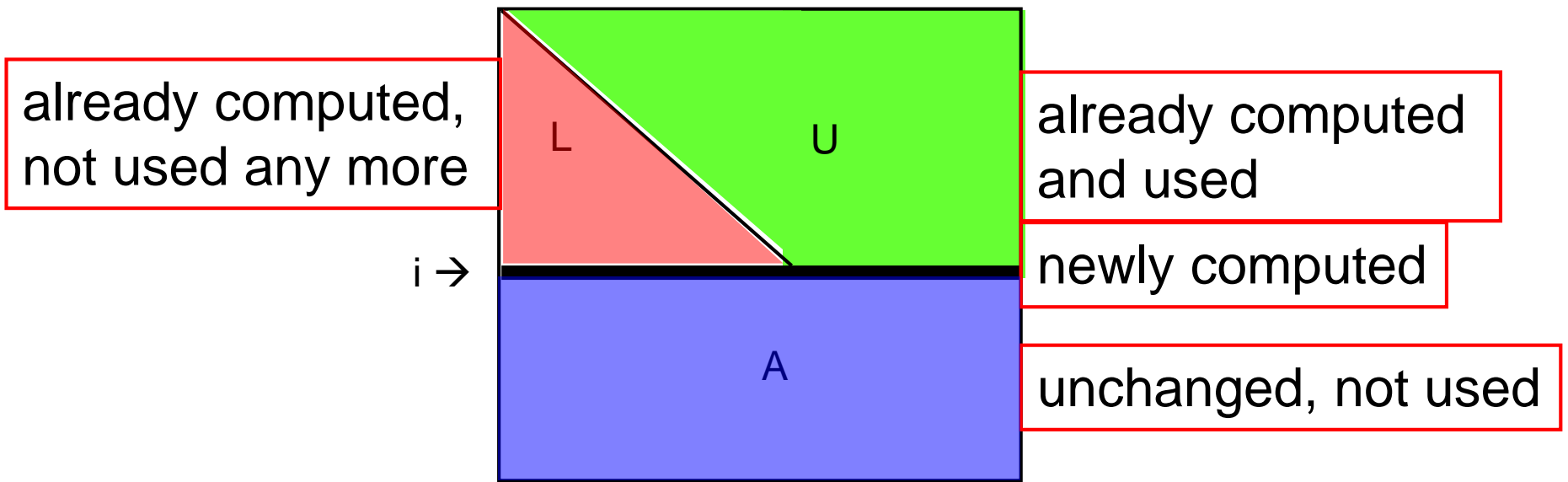
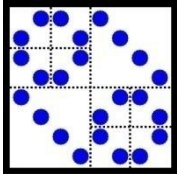
$$a_{i,j} = a_{i,j} - l_{i,k} a_{k,j};$$

end

end

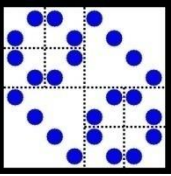
end

} GAXPY in $a_{.i}$

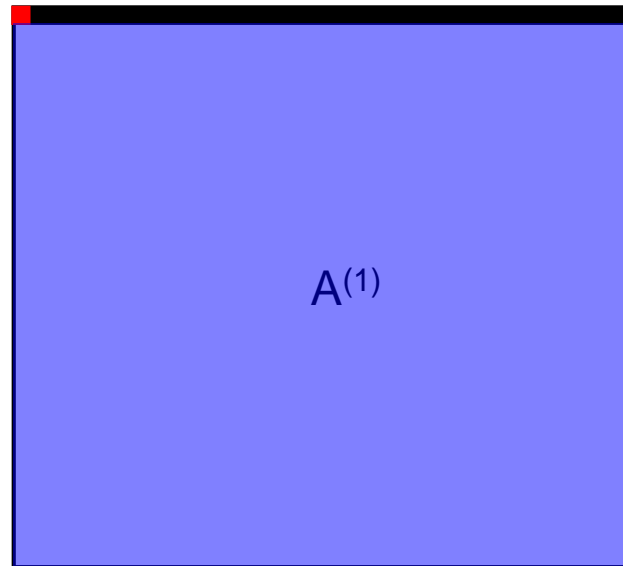


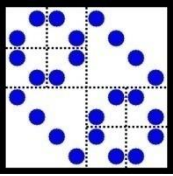
L and U computed rowwise.

Compute $l_{i,1}$, then SAXPY for 1st and i -th row;
then $l_{i,2}$ and so on ...

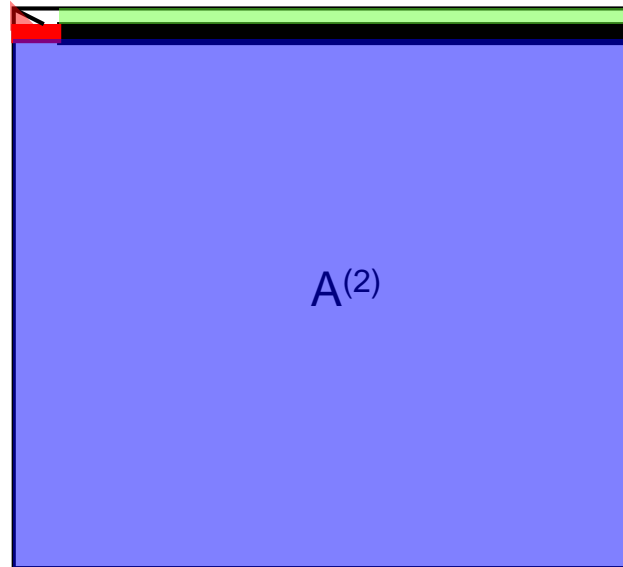


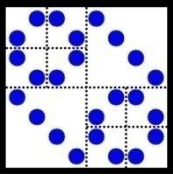
First step



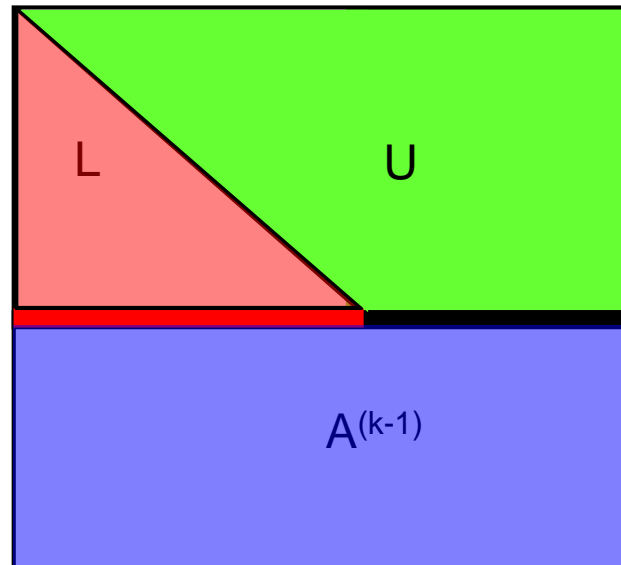


Second step





k-1-st step

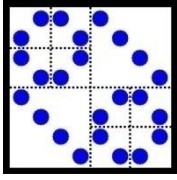




(ijk)-form:

$$1 \leq k < i \leq n$$

$$1 \leq k < j \leq n$$



For $i = 2 : n$

For $j = 2 : i$

$$l_{i,j-1} = a_{i,j-1} / a_{j-1,j-1};$$

For $k = 1 : j-1$

$$a_{i,j} = a_{i,j} - l_{i,k} a_{k,j};$$

end

end

For $j = i+1 : n$

For $k = 1 : i-1$

$$a_{i,j} = a_{i,j} - l_{i,k} a_{k,j};$$

end

end

end

Compute $l_{i,1}$ and update $a_{i,2}$; then compute $l_{i,2}$ and update $a_{i,2}$ and $a_{i,3}, \dots$

Accumulating $a_{i,j}$

new row

Dot product

left part

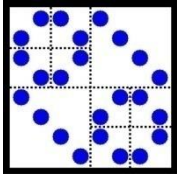
Dot product

right part



$$1 \leq k < i \leq n$$

$$1 \leq k < j \leq n$$

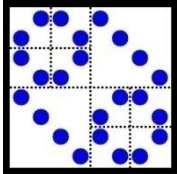


(jki)-form:

```
For j = 2 : n
  For k = j : n
     $l_{k,j-1} = a_{k,j-1} / a_{j-1,j-1};$ 
  end
  For k = 1 : j-1
    For i = k+1 : n
       $a_{i,j} = a_{i,j} - l_{i,k} a_{k,j};$ 
    end
  end
end
end
```

$\alpha \cdot \vec{x}$ new column of L

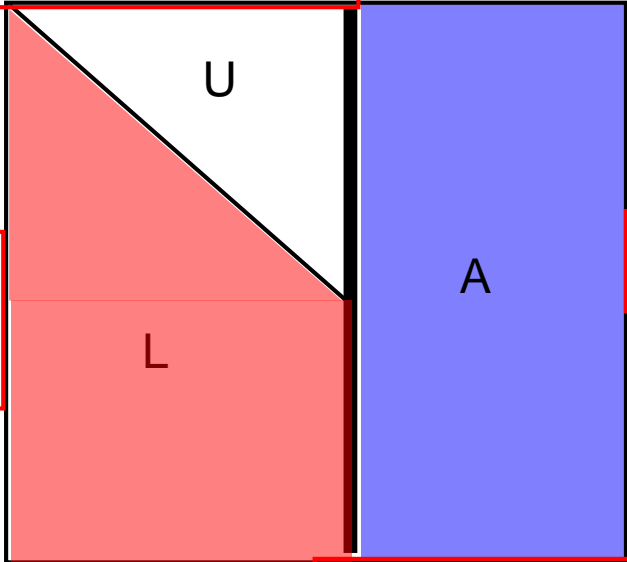
GAXPY in $a_{.j}$



Left looking GE

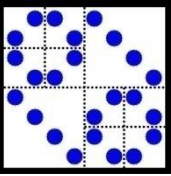
computed, not used

already computed and used

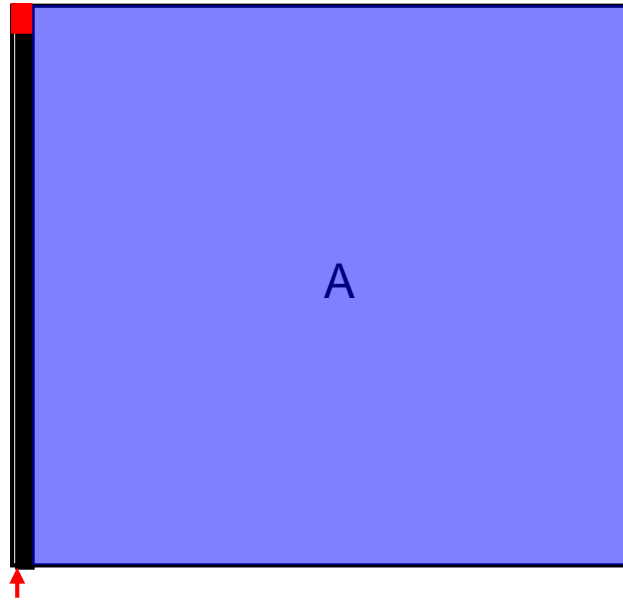


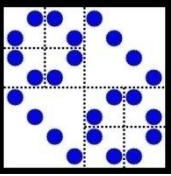
unchanged, not used

$j-1$, newly computed

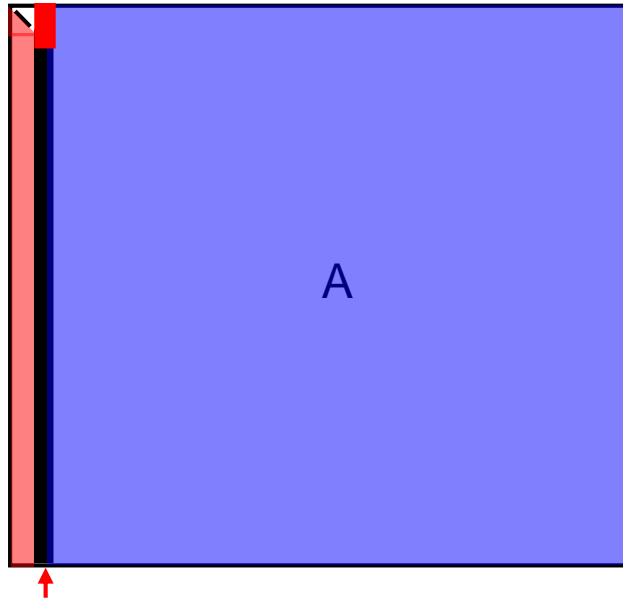


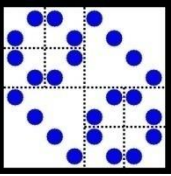
First step



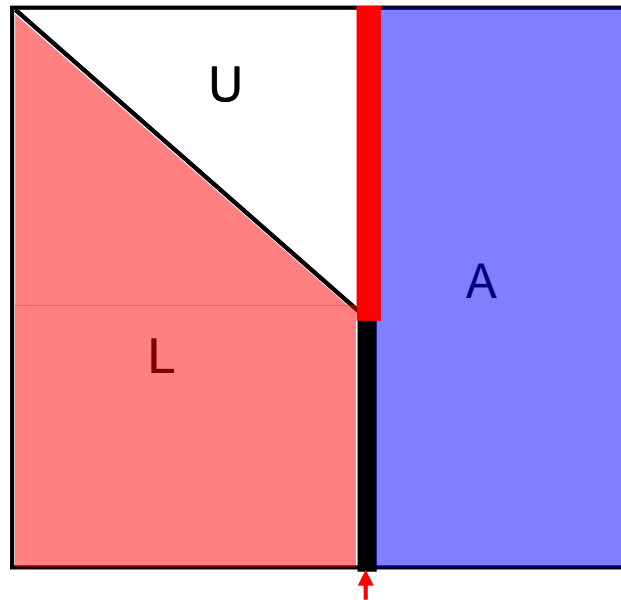


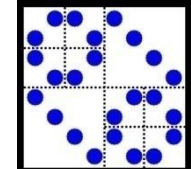
Second step





k-1-st step





Overview

	kij	kji	ikj	ijk	jki	jik
Access to A and U	row	column	row	column	column	column
Access to L	-----	column	-----	row	column	row
Computation of U	row	row	row	row	column	column
Computation of L	column	column	row	row	column	column
Vector operation	SAXPY	SAXPY	GAXPY	DOT	GAXPY	DOT
Vector length	$2/3 n$	$2/3 n$	$2/3 n$	$n/3$	$2/3 n$	$n/3$

Vector length = average of occurring vector lengths

Optimal form depends on storage of matrices and vector length.

Parallel Numerics, WT 2017/2018

3 Linear Systems of Equations with Dense Matrices



Contents

- 1 Introduction
 - 1.1 Computer Science Aspects
 - 1.2 Numerical Problems
 - 1.3 Graphs
 - 1.4 Loop Manipulations
- 2 Elementary Linear Algebra Problems
 - 2.1 BLAS: Basic Linear Algebra Subroutines
 - 2.2 Matrix-Vector Operations
 - 2.3 Matrix-Matrix-Product
- 3 Linear Systems of Equations with Dense Matrices**
 - 3.1 Gaussian Elimination
 - 3.2 Parallelization
 - 3.3 QR-Decomposition with Householder matrices
- 4 Sparse Matrices
 - 4.1 General Properties, Storage
 - 4.2 Sparse Matrices and Graphs
 - 4.3 Reordering
 - 4.4 Gaussian Elimination for Sparse Matrices
- 5 Iterative Methods for Sparse Matrices
 - 5.1 Stationary Methods
 - 5.2 Nonstationary Methods
 - 5.3 Preconditioning
- 6 Domain Decomposition
 - 6.1 Overlapping Domain Decomposition
 - 6.2 Non-overlapping Domain Decomposition
 - 6.3 Schur Complements



3.1. Linear Systems of Equations with Dense Matrices

3.1.1. Gaussian Elimination: Basic Properties

- Linear system of equations:

$$\begin{aligned} a_{11}x_1 + \dots + a_{1n}x_n &= b_1 \\ &\vdots \\ a_{n1}x_1 + \dots + a_{nn}x_n &= b_n \end{aligned}$$

- Solve $Ax = b$

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$$

- Generate simpler linear equations (matrices). Transform A in triangular form: $A = A^{(1)} \rightarrow A^{(2)} \rightarrow \dots \rightarrow A^{(n)} = U$.



Transformation to Upper Triangular Form

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$

row transformations: $(2) \rightarrow (2) - \frac{a_{21}}{a_{11}} \cdot (1), \dots, (n) \rightarrow (n) - \frac{a_{n1}}{a_{11}} \cdot (1)$
leads to

$$A^{(2)} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} & \cdots & a_{3n}^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n2}^{(2)} & a_{n3}^{(2)} & \cdots & a_{nn}^{(2)} \end{pmatrix}$$

next transformations: $(3) \rightarrow (3) - \frac{a_{32}^{(2)}}{a_{22}^{(2)}} \cdot (2), \dots, (n) \rightarrow (n) - \frac{a_{n2}^{(2)}}{a_{22}^{(2)}} \cdot (2)$



Transformation to Triangular Form (cont.)

$$A^{(3)} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & \cdots & a_{3n}^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & a_{n3}^{(3)} & \cdots & a_{nn}^{(3)} \end{pmatrix}$$

next transformations: $(4) \rightarrow (4) - \frac{a_{43}^{(3)}}{a_{33}^{(3)}} \cdot (3)$, \dots , $(n) \rightarrow (n) - \frac{a_{n3}^{(3)}}{a_{33}^{(3)}} \cdot (3)$

$$A^{(n)} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & \cdots & a_{3n}^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn}^{(n)} \end{pmatrix} = U$$



Pseudocode Gaussian Elimination (GE)

Simplification: assume that no pivoting is necessary.

$$a_{kk}^{(k)} \neq 0 \quad \text{or} \quad |a_{kk}^{(k)}| \geq \rho > 0 \quad \text{for } k = 1, 2, \dots, n$$

```

for  $k = 1 : n - 1$ 
  for  $i = k + 1 : n$ 
     $l_{i,k} = \frac{a_{i,k}}{a_{k,k}}$ 
  end
  for  $i = k + 1 : n$ 
    for  $j = k + 1 : n$ 
       $a_{i,j} = a_{i,j} - l_{i,k} \cdot a_{k,j}$ 
    end
  end
end
end

```

In practice:

- Include pivoting and include right hand side b .
- There is still to solve a triangular system in U !



Intermediate Systems

$$A^{(k)}, k = 1, 2, \dots, n \quad \text{with} \quad A = A^{(1)} \quad \text{and} \quad U = A^{(n)}$$

$$\begin{pmatrix} a_{11}^{(1)} & \cdots & a_{1,k-1}^{(1)} & a_{1,k}^{(1)} & \cdots & a_{1,n}^{(1)} \\ 0 & \ddots & \vdots & \vdots & \ddots & \vdots \\ \vdots & \ddots & a_{k-1,k-1}^{(k-1)} & a_{k-1,k}^{(k-1)} & \cdots & a_{k-1,n}^{(k-1)} \\ 0 & \cdots & 0 & a_{k,k}^{(k)} & \cdots & a_{k,n}^{(k)} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & a_{n,k}^{(k)} & \cdots & a_{n,n}^{(k)} \end{pmatrix}$$

Main part of $A^{(k)}$ that will be used and changed in the following computations.



Define Auxiliary Matrices

$$L = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ l_{2,1} & 1 & \ddots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ l_{n,1} & \cdots & l_{n,n-1} & 1 \end{pmatrix} \quad \text{and} \quad U = A^{(n)}$$

$$L_k := \begin{pmatrix} 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & l_{k+1,k} & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & l_{n,k} & 0 & \cdots & 0 \end{pmatrix}, \quad L = I + \sum_k L_k$$



Elimination Step in Terms of Auxiliary Matrices

$$A^{(k+1)} = (I - L_k) \cdot A^{(k)} = A^{(k)} - L_k \cdot A^{(k)}$$

$$U = A^{(n)} = (I - L_{n-1}) \cdot A^{(n-1)} = \dots = (I - L_{n-1}) \cdots (I - L_1) A^{(1)} = \tilde{L} \cdot A$$

$$\tilde{L} := (I - L_{n-1}) \cdots (I - L_1)$$

$$A = \tilde{L}^{-1} \cdot U \quad \text{with } U \text{ upper triangular and } \tilde{L} \text{ lower triangular}$$



Elimination Step in Terms of Auxiliary Matrices

$$A^{(k+1)} = (I - L_k) \cdot A^{(k)} = A^{(k)} - L_k \cdot A^{(k)}$$

$$U = A^{(n)} = (I - L_{n-1}) \cdot A^{(n-1)} = \dots = (I - L_{n-1}) \cdots (I - L_1) A^{(1)} = \tilde{L} \cdot A$$

$$\tilde{L} := (I - L_{n-1}) \cdots (I - L_1)$$

$$A = \tilde{L}^{-1} \cdot U \quad \text{with } U \text{ upper triangular and } \tilde{L} \text{ lower triangular}$$

- **Theorem 2:** $\tilde{L}^{-1} = L$ and therefore $A = LU$.
- Advantage: Every further problem $Ax = b_j$ can be reduced to $(LU)x = b_j$ for arbitrary j .
- Solve two triangular problems $(LU)x = Ly = b$ and $Ux = y$.



Theorem 2: $\tilde{L}^{-1} = L \rightarrow A = LU$

$$\text{for } i \leq j: \quad L_i \cdot L_j = \begin{pmatrix} 0 & \boxed{i} & & & \\ & \ddots & & & \\ & & 0 & & \\ & & * & 0 & \\ & & \vdots & & \ddots \\ & & * & & & 0 \end{pmatrix} \begin{pmatrix} 0 & \boxed{j} & & & \\ & \ddots & & & \\ & & 0 & & \\ & & * & 0 & \\ & & \vdots & & \ddots \\ & & * & & & 0 \end{pmatrix} = 0$$

$$(I + L_j)(I - L_j) = I + L_j - L_j - L_j^2 = I \Rightarrow (I - L_j)^{-1} = I + L_j$$

$$\underbrace{(I + L_i)(I + L_j) = I + L_i + L_j + L_i L_j = I + L_i + L_j}$$

Theorem 2: $\tilde{L}^{-1} = L \rightarrow A = LU$

$$\text{for } i \leq j: \quad L_i \cdot L_j = \begin{pmatrix} 0 & & & & & & \\ & \ddots & & & & & \\ & & \boxed{i} & & & & \\ & & 0 & & & & \\ & & * & 0 & & & \\ & & \vdots & & \ddots & & \\ & & * & & & & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & & & & & & \\ & \ddots & & & & & \\ & & \boxed{j} & & & & \\ & & 0 & & & & \\ & & * & 0 & & & \\ & & \vdots & & \ddots & & \\ & & * & & & & 0 \end{pmatrix} = 0$$

$$(I + L_j)(I - L_j) = I + L_j - L_j - L_j^2 = I \Rightarrow (I - L_j)^{-1} = I + L_j$$

$$\underbrace{(I + L_i)(I + L_j) = I + L_i + L_j + L_i L_j = I + L_i + L_j}$$

$$\begin{aligned} \tilde{L}^{-1} &= [(I - L_{n-1}) \cdots (I - L_1)]^{-1} = (I - L_1)^{-1} \cdots (I - L_{n-1})^{-1} = \\ &(I + L_1)(I + L_2) \cdots (I + L_{n-1}) = I + L_1 + L_2 + \cdots + L_{n-1} = L \end{aligned}$$



3.2. GE in Parallel: Blockwise

Main idea: Blocking of GE to avoid data transfer between processors.

Basic Concepts:

Replace GE or large LU -decomposition of full matrix by small intermediate steps (by sequence of small block operations):

- Solving collection of small triangular systems $LU_k = B_k$ (parallelism in columns of U)
- $A \rightarrow A - LU$ updating matrices (also easy to parallelize)
- small $B = LU$ -decompositions (parallelism in rows of B)



How to Choose Blocks in L , resp. U Satisfying $LU = A$

$$\begin{pmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} =$$

$$= \begin{pmatrix} L_{11}U_{11} & L_{11}U_{12} & L_{11}U_{13} \\ L_{21}U_{11} & L_{21}U_{12} + L_{22}U_{22} & L_{21}U_{13} + L_{22}U_{23} \\ L_{31}U_{11} & L_{31}U_{12} + L_{32}U_{22} & * \end{pmatrix}$$



How to Choose Blocks in L , resp. U Satisfying $LU = A$

$$\begin{pmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} =$$

$$= \begin{pmatrix} L_{11}U_{11} & L_{11}U_{12} & L_{11}U_{13} \\ L_{21}U_{11} & L_{21}U_{12} + L_{22}U_{22} & L_{21}U_{13} + L_{22}U_{23} \\ L_{31}U_{11} & L_{31}U_{12} + L_{32}U_{22} & *$$

Different ways of computing L and U depending on

- start (assume first entry/row/column of L/U as given)
- how to compute new entry/row/column of L/U
- update of block structure of L/U by grouping in
 - known blocks
 - blocks newly to compute
 - blocks to be computed later



Crout Form

$$\begin{pmatrix} L_{11} & & \\ L_{21} & L_{22} & \\ L_{31} & L_{32} & * \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} & U_{13} \\ & U_{22} & U_{23} \\ & & * \end{pmatrix} = A$$

Already computed To compute in this step

$$\begin{pmatrix} L_{22}U_{22} & L_{22}U_{23} \\ L_{32}U_{22} & * \end{pmatrix} = \begin{pmatrix} A_{22} - L_{21}U_{12} & A_{23} - L_{21}U_{13} \\ A_{32} - L_{31}U_{12} & * \end{pmatrix} = \begin{pmatrix} \hat{A}_{22} & \hat{A}_{23} \\ \hat{A}_{32} & * \end{pmatrix}$$

Leads to two subproblems in and in



Crout Form (cont.)

1. Solve

$$\begin{pmatrix} L_{22} \\ L_{32} \end{pmatrix} \cdot U_{22} = \begin{pmatrix} \hat{A}_{22} \\ \hat{A}_{32} \end{pmatrix}$$

by small LU -decomposition of the modified part of $A \rightarrow L_{22}, L_{32}$, and U_{22} .

2. Solve

$$L_{22} \cdot U_{23} = \hat{A}_{23}$$

by solving small triangular systems of equations in $L_{22} \rightarrow U_{23}$.



Crout Form (cont.)

1. Solve

$$\begin{pmatrix} L_{22} \\ L_{32} \end{pmatrix} \cdot U_{22} = \begin{pmatrix} \hat{A}_{22} \\ \hat{A}_{32} \end{pmatrix}$$

by small LU -decomposition of the modified part of $A \rightarrow L_{22}, L_{32}$, and U_{22} .

2. Solve

$$L_{22} \cdot U_{23} = \hat{A}_{23}$$

by solving small triangular systems of equations in $L_{22} \rightarrow U_{23}$.

Initial steps:

$$L_{11} U_{11} = A_{11}, \quad \begin{pmatrix} L_{21} \\ L_{31} \end{pmatrix} U_{11} = \begin{pmatrix} A_{21} \\ A_{31} \end{pmatrix}, \quad L_{11}(U_{12} \ U_{13}) = (A_{12} \ A_{13})$$



New Partitioning

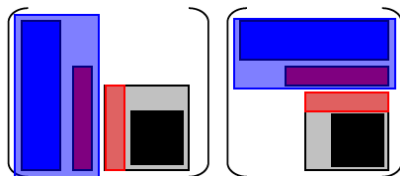
$$A = \left(\begin{array}{cc|c} \boxed{\begin{matrix} L_{11} & & \\ L_{21} & L_{11} & L_{22} \end{matrix}} & & \\ \hline \boxed{\begin{matrix} L_{31,1} & L_{21} & L_{32,1} \end{matrix}} & \boxed{\begin{matrix} L_{33,2} \\ L_{33,1} \end{matrix}} & \\ \hline \boxed{\begin{matrix} L_{31,2} & L_{32,2} \\ L_{31} & L_{32} \end{matrix}} & & L_{33,new} \end{array} \right) \cdot \left(\begin{array}{cc|cc} \boxed{\begin{matrix} U_{11} & U_{12} \\ U_{11} & U_{22} \end{matrix}} & & \boxed{\begin{matrix} U_{13,1} \\ U_{12} \\ U_{23,1} \end{matrix}} & \boxed{\begin{matrix} U_{13,2} \\ U_{13} \\ U_{23,2} \end{matrix}} \\ \hline & & \boxed{\begin{matrix} U_{33,1} \\ U_{32} \\ U_{33,2} \end{matrix}} & \boxed{\begin{matrix} U_{33,2} \\ U_{33} \end{matrix}} \\ \hline & & & U_{33,new} \end{array} \right)$$

- Combine already computed parts from second column of L and second row of U into first column of L and first row of U .
- Split the until now ignored parts L_{33} and U_{33} into new columns/rows.
- Repeat this overall procedure until L and U are fully computed.



Block Structure

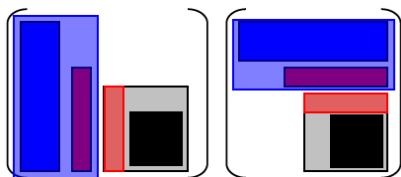
Intermediate block structure:



Solve for red blocks.

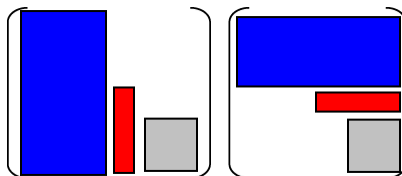
Block Structure

Intermediate block structure:



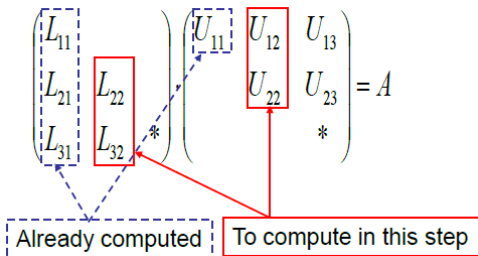
Solve for red blocks.

Reconfigure the block structure:

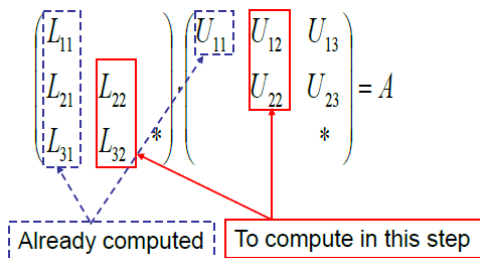


Repeat until done.

Left Looking GE



Left Looking GE



- Solve $L_{11} U_{12} = A_{12}$ by a couple of parallel triangular solves and

$$\begin{pmatrix} L_{22} \\ L_{32} \end{pmatrix} U_{22} = \begin{pmatrix} A_{22} \\ A_{32} \end{pmatrix} - \begin{pmatrix} L_{21} \\ L_{31} \end{pmatrix} U_{12} =: \begin{pmatrix} \hat{A}_{22} \\ \hat{A}_{32} \end{pmatrix}$$

update part of A and perform small LU -decomposition.



Left Looking GE

$$\begin{pmatrix} L_{11} & & \\ L_{21} & L_{22} & \\ L_{31} & L_{32} & * \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} & U_{13} \\ & U_{22} & U_{23} \\ & & * \end{pmatrix} = A$$

- Solve $L_{11} U_{12} = A_{12}$ by a couple of parallel triangular solves and

$$\begin{pmatrix} L_{22} \\ L_{32} \end{pmatrix} U_{22} = \begin{pmatrix} A_{22} \\ A_{32} \end{pmatrix} - \begin{pmatrix} L_{21} \\ L_{31} \end{pmatrix} U_{12} =: \begin{pmatrix} \hat{A}_{22} \\ \hat{A}_{32} \end{pmatrix}$$

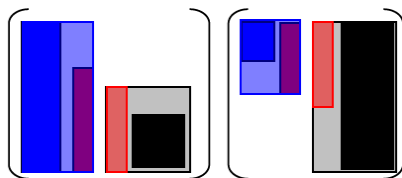
update part of A and perform small LU -decomposition.

- Reorder blocks and repeat until ready. Start: $L_{11} U_{11} = A_{11}$, $L_{21} U_{11} = A_{21}$, and $L_{31} U_{11} = A_{31}$.



Block Structure

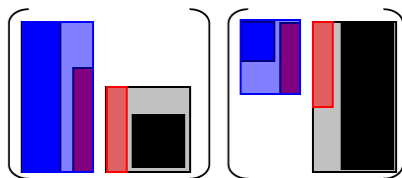
Intermediate block structure:



Solve for red blocks.

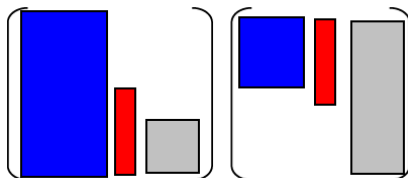
Block Structure

Intermediate block structure:



Solve for red blocks.

Reconfigure the block structure:



Repeat until done.