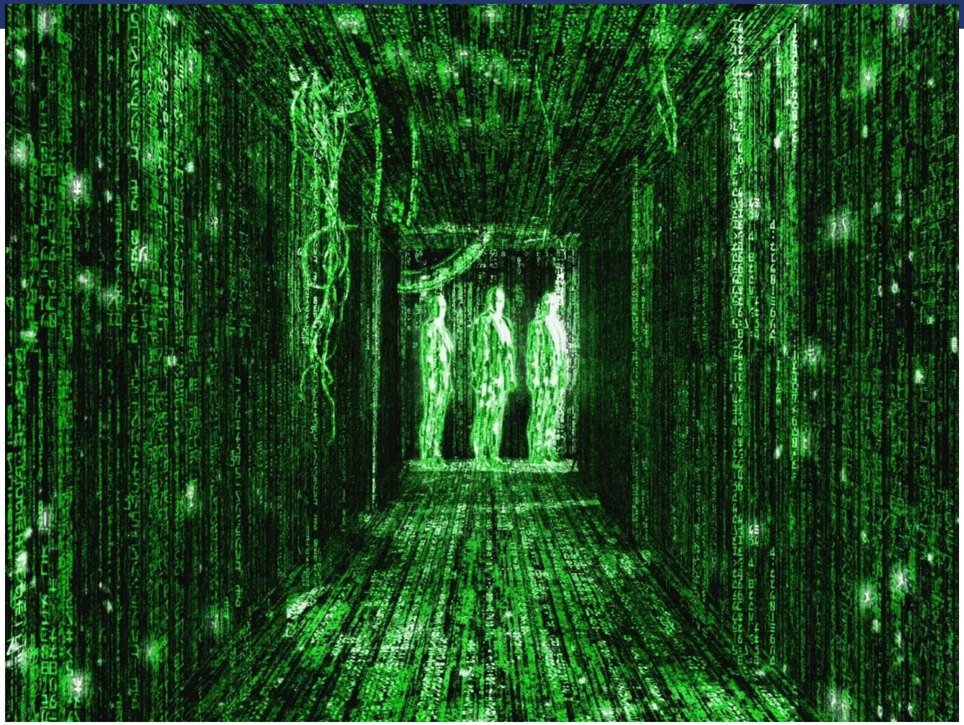
Parallel programming

Matrix Algorithms in OpenMP and MPI



FAKULTA ELEKTROTECHNICKÁ ČVUT V PRAZE

Today's topic

- Coding seminar
- Goals
  - Practice the theory from the lectures
  - Practice OpenMP and MPI
- 4 Tasks
  - Matrix multiplication (OpenMP)
  - LU factorization (OpenMP)
  - Gaussian elimination (MPI)
  - Gaussian elimination with cyclic row distribution (MPI)

Matrix multiplication

- Consider 2 matrices **A** and **B**, we want matrix **C** such that
  - $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$
- Matrix multiplication
  - Computational operations: $\mathbf{2n^3}$
  - Memory operations: $\mathbf{3n^2}$
- Naive algorithm might not be efficient
  - Too many memory operations
  - Cache size is limited
- If we are able to reuse data we can do something better
  - Use **blocks**!

Matrix Multiplication

`MatrixMultiplication.cpp`

- Open the provided template and complete the empty functions according to the guidelines

Block matrix multiplication

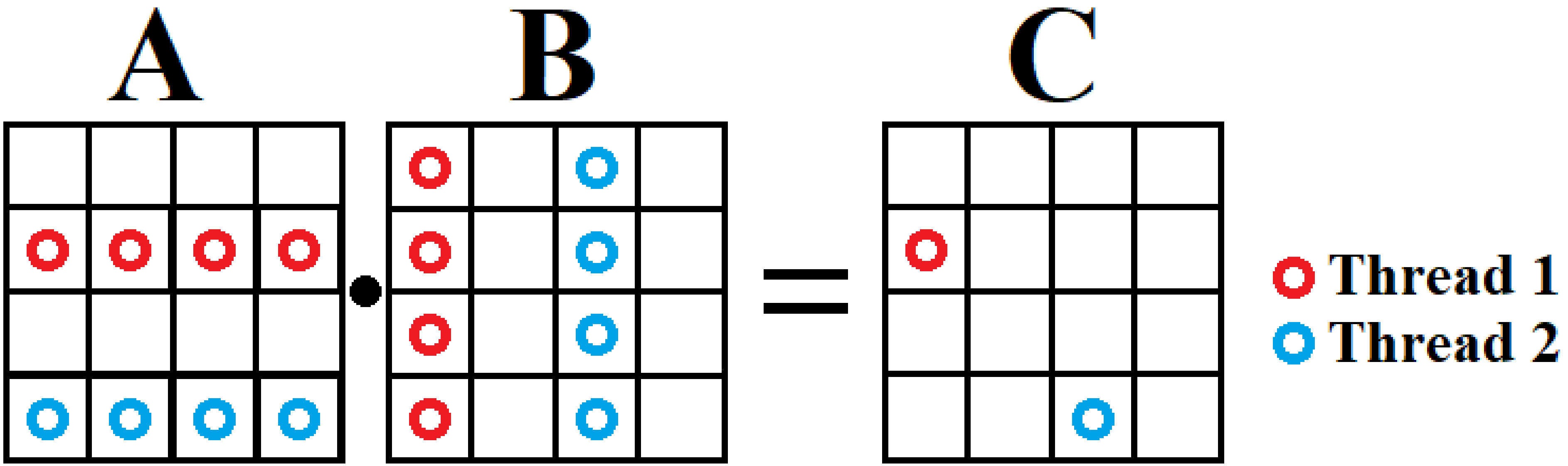
- We can divide **A** into blocks of rows and **B** into blocks of columns
  - If rows and columns are too large, they will not fit in the cache!
- Divide **A** and **B** into blocks of size b × b

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{pmatrix}$$

- Then $C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21} + A_{13} \cdot B_{31}$
  - Each $A_{ij} \cdot B_{ji}$ operation has $\mathbf{2b^2}$ memory operations and $\mathbf{2b^3}$ computational operations
- Chose b so that an entire block can fit into the cache!

Parallel block matrix multiplication

- Using block matrix multiplication
- Use tasks to parallelize the algorithm
  - Beware of race conditions
  - Beware of correct data sharing among threads



Block calculations

Element Calculations:

1. c_{11} **(top-left element):**

$$c_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21} + A_{13} \cdot B_{31}$$

2. c_{12} **(top-center element):**

$$c_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22} + A_{13} \cdot B_{32}$$

3. c_{13} **(top-right element):**

$$c_{13} = A_{11} \cdot B_{13} + A_{12} \cdot B_{23} + A_{13} \cdot B_{33}$$

4. c_{21} **(middle-left element):**

$$c_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21} + A_{23} \cdot B_{31}$$

5. c_{22} **(middle-center element):**

$$c_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22} + A_{23} \cdot B_{32}$$

6. c_{23} **(middle-right element):**

$$c_{23} = A_{21} \cdot B_{13} + A_{22} \cdot B_{23} + A_{23} \cdot B_{33}$$

7. c_{31} **(bottom-left element):**

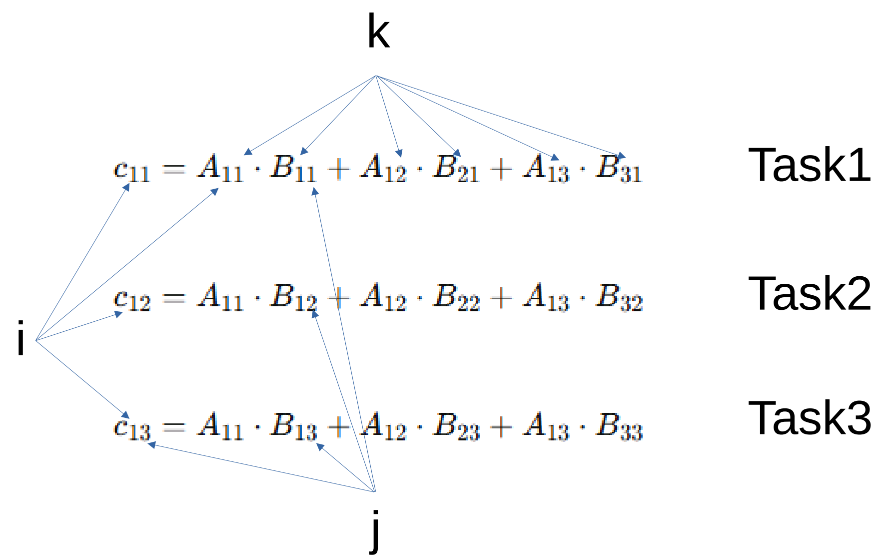
$$c_{31} = A_{31} \cdot B_{11} + A_{32} \cdot B_{21} + A_{33} \cdot B_{31}$$

8. c_{32} **(bottom-center element):**

$$c_{32} = A_{31} \cdot B_{12} + A_{32} \cdot B_{22} + A_{33} \cdot B_{32}$$

9. c_{33} **(bottom-right element):**

$$c_{33} = A_{31} \cdot B_{13} + A_{32} \cdot B_{23} + A_{33} \cdot B_{33}$$




```
for i
    for j
        for k
```

Sharing A block data

Element Calculations:

1. c_{11} **(top-left element):**

$$c_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21} + A_{13} \cdot B_{31}$$

2. c_{12} **(top-center element):**

$$c_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22} + A_{13} \cdot B_{32}$$

3. c_{13} **(top-right element):**

$$c_{13} = A_{11} \cdot B_{13} + A_{12} \cdot B_{23} + A_{13} \cdot B_{33}$$

4. c_{21} **(middle-left element):**

$$c_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21} + A_{23} \cdot B_{31}$$

5. c_{22} **(middle-center element):**

$$c_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22} + A_{23} \cdot B_{32}$$

6. c_{23} **(middle-right element):**

$$c_{23} = A_{21} \cdot B_{13} + A_{22} \cdot B_{23} + A_{23} \cdot B_{33}$$

7. c_{31} **(bottom-left element):**

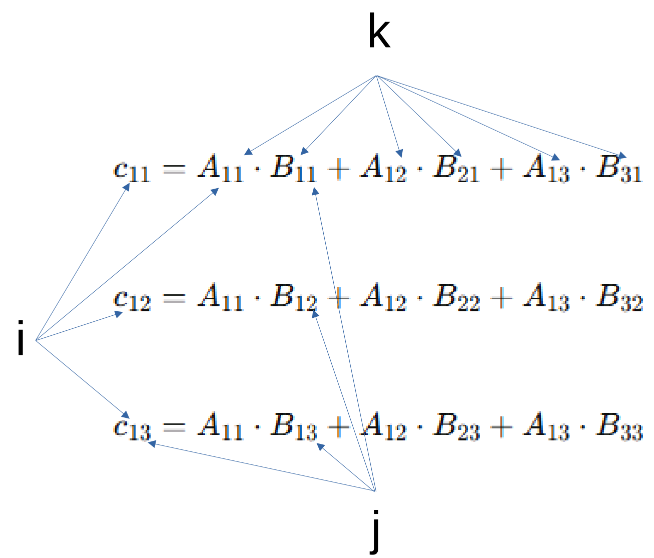
$$c_{31} = A_{31} \cdot B_{11} + A_{32} \cdot B_{21} + A_{33} \cdot B_{31}$$

8. c_{32} **(bottom-center element):**

$$c_{32} = A_{31} \cdot B_{12} + A_{32} \cdot B_{22} + A_{33} \cdot B_{32}$$

9. c_{33} **(bottom-right element):**

$$c_{33} = A_{31} \cdot B_{13} + A_{32} \cdot B_{23} + A_{33} \cdot B_{33}$$



Task1

```
for i
    for k
        for j
```

Gaussian Elimination

- Useful for solving systems of linear equations
  - Row reduction
  - Can also be used to compute the rank of a matrix, the determinant of a square matrix, and the inverse of an invertible matrix
- Sequence of row operations
  - Multiplying a row by a nonzero number
  - Adding a multiple of one row to another row
- Each row operation needs a **pivot** row that defines the multiplying coefficients

Gaussian elimination - reminder

$$\left(\begin{array}{ccc|c} -1 & 0 & 3 & 3 \\ 2 & 1 & 3 & 4 \\ 1 & 1 & 2 & 2 \end{array}\right)$$

$$R_1 \leftarrow -1R_1, \qquad R_2 \leftarrow R_2 + 2R_1, \qquad R_3 \leftarrow R_3 + 1R_1$$

$$\left(\begin{array}{ccc|c} 1 & 0 & -3 & -3 \\ 0 & 1 & 9 & 10 \\ 0 & 1 & 5 & 5 \end{array}\right)$$

$$R_3 \leftarrow R_3 - 1R_2$$

$$\left(\begin{array}{ccc|c} 1 & 0 & -3 & -3 \\ 0 & 1 & 9 & 10 \\ 0 & 0 & -4 & -5 \end{array}\right)$$

$$R_3 \leftarrow -\frac{1}{4}R_3$$

$$\left(\begin{array}{ccc|c} 1 & 0 & -3 & -3 \\ 0 & 1 & 9 & 10 \\ 0 & 0 & 1 & 5/4 \end{array}\right)$$

Gaussian Elimination Pseudocode

```
for k = 1 to (n-1)
    for i = (k+1) to n
        factor = A(i, k) / A(k, k)
        for j = k to n
            A(i, j) = A(i, j) - factor * A(k, j)
        end for
    b(i) = b(i) - factor * b(k)
    end for
end for
```

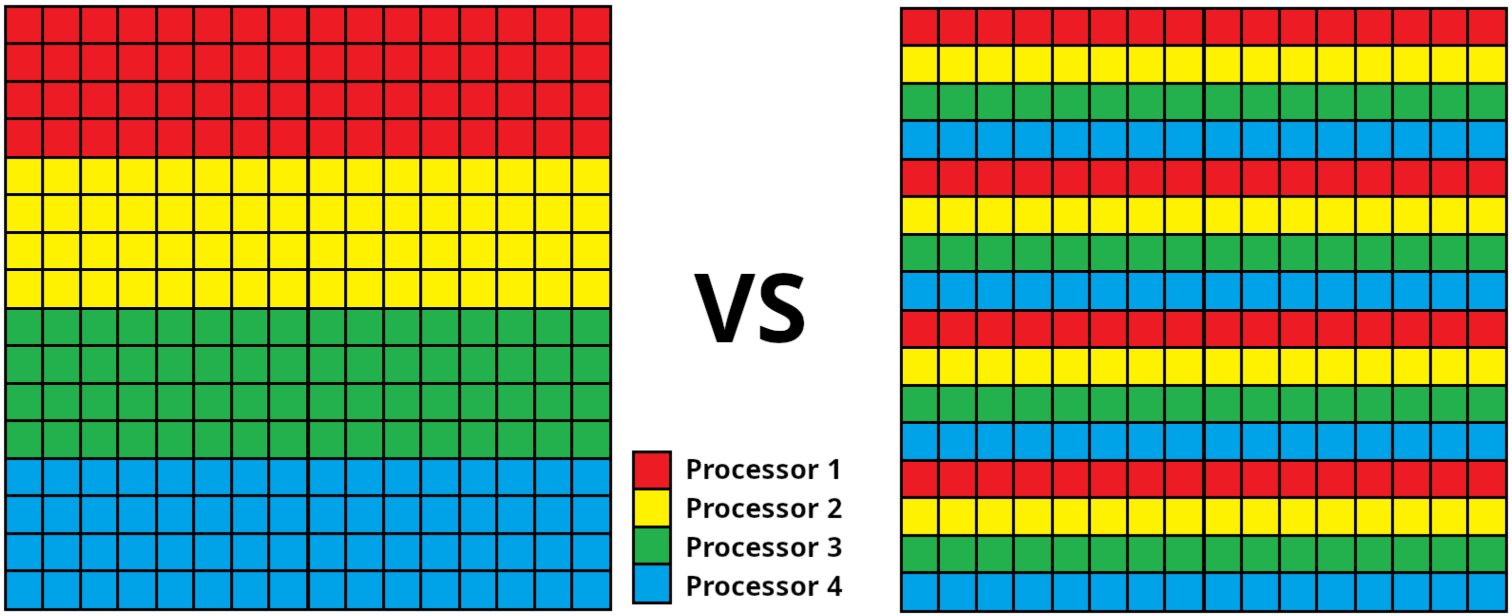
```
for i = n to (step-1)
    x(i) = b(i)
    for j = i+1 to n
        x(i) = x(i) - A(i, j) * x(j)
    end for
    x(i) = x(i) / A(i, i)
end for
```

Distributed Gaussian Elimination

1. Scatter the matrix rows
2. For each iteration select a pivot row
3. Processor with the pivot row in the k-th iteration performs an operation on the pivot row to obtain a 1 at the k-th position
4. Processor with the pivot row broadcasts the pivot row
5. Perform row reduction for rows under the pivot row to obtain a 0 at the k-th position
6. Steps 2-5 are repeated until the last row is reached
7. Gather the updated rows at processor 0

Row distribution for distributed GE

- Using a naive distribution may not be the efficient method
  - After a processor updates all of its rows, it will not do any work
- Using a cyclic row distribution
  - More efficient (processors will be working almost until the end)



Gaussian Elimination

`GaussEliminationBlock.cpp`

- Open the provided template and implement parallel Gaussian elimination using MPI with **block row distribution**, follow the provided guidelines.

`GaussEliminationCyclic.cpp`

- Open the provided template and implement parallel Gaussian elimination using MPI with **cyclic row distribution**, follow the provided guidelines.

LU Factorization

- LU factorization of matrix A
  - **A = L U**
  - **L** is a lower triangular matrix
  - **U** is an upper triangular matrix
- Useful for solving linear equations
  - **A x = b** <=> **L (U x) = b**
  - **L y = b** => obtain vector **y** using backward triangular substitution
  - **U x = y** => obtain vector **x** using backward triangular substitution

LU Factorization as Gaussian elimination

- LU factorization is the matrix form of Gaussian elimination
  - The operations performed durring Gaussian elimination can be written as a matrix lower triangular matrix **E**
  - $\mathbf{E\ A = U} \iff \mathbf{A = E^{-1} U}$
  - **The inverse of a lower triangular matrix is also a lower triangular matrix!**
  - $\mathbf{A = E^{-1} U = LU} \iff \mathbf{E^{-1} = L} \iff \mathbf{L^{-1} = E}$
  - We can obtain **E** using Gaussian elimination

Uniqueness of LU decomposition

- LU decomposition is not unique!
- Pre-determing the values on the main diagonal of either **L** or **U** makes it unique.
- The computation is simpler if we choose the main diagonal of L to be unitary.

LU Factorization - Example

$$\left(\begin{array}{ccc|ccc} -1 & 0 & 3 & 1 & 0 & 0 \\ 2 & 1 & 3 & 0 & 1 & 0 \\ 1 & 1 & 2 & 0 & 0 & 1 \end{array}\right)$$

$$R_2 \leftarrow R_2 + 2R_1, \qquad R_3 \leftarrow R_3 + 1R_1$$

$$\left(\begin{array}{ccc|ccc} -1 & 0 & 3 & 1 & 0 & 0 \\ 0 & 1 & 9 & 2 & 1 & 0 \\ 0 & 1 & 5 & 1 & 0 & 1 \end{array}\right)$$

$$R_3 \leftarrow R_3 - 1R_2$$

$$\left(\begin{array}{ccc|ccc} -1 & 0 & 3 & 1 & 0 & 0 \\ 0 & 1 & 9 & 2 & 1 & 0 \\ 0 & 0 & -4 & 1 & -1 & 1 \end{array}\right)$$

$$L^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -1 & 1 & 1 \end{pmatrix}, \qquad U = \begin{pmatrix} -1 & 0 & 3 \\ 0 & 1 & 9 \\ 0 & 0 & -4 \end{pmatrix}$$

$$\left(\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 2 & 1 & 0 & 0 & 1 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 \end{array}\right)$$

$$R_2 \leftarrow R_2 - 2R_1, \qquad R_3 \leftarrow R_3 - 1R_1$$

$$\left(\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & -2 & 1 & 0 \\ 0 & -1 & 1 & -1 & 0 & 1 \end{array}\right)$$

$$R_3 \leftarrow R_3 - 1R_2$$

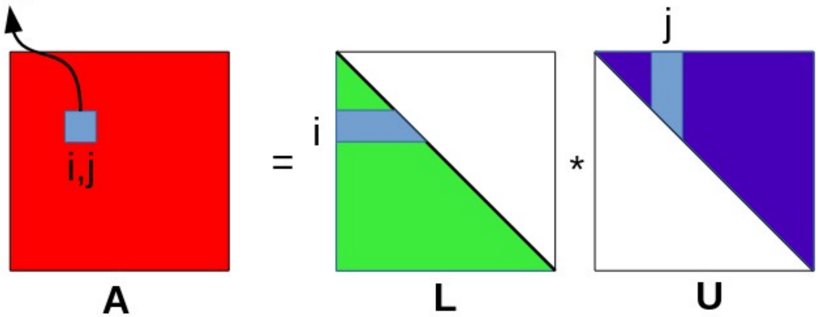
$$\left(\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & -2 & 1 & 0 \\ 0 & 0 & 1 & -1 & 1 & 1 \end{array}\right)$$

$$L = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -1 & 1 & 1 \end{pmatrix}, \qquad U = \begin{pmatrix} -1 & 0 & 3 \\ 0 & 1 & 9 \\ 0 & 0 & -4 \end{pmatrix}$$

Computation LU Factorization

- We can compute the LU factorization directly

$$a_{ij} = \sum_{r=0}^{n-1} l_{ir} u_{rj} \qquad \forall i, j \in \{0, \ldots, n-1\}$$



$$l_{ir} = \begin{cases} 0 \text{ if } r > i \\ 1 \text{ if } r = i \\ l_{ir} \text{ otherwise} \end{cases}$$

$$u_{rj} = \begin{cases} 0 \text{ if } r > j \\ u_{rj} \text{ otherwise} \end{cases}$$

$$a_{ij} = \sum_{r=0}^{n-1} l_{ir} u_{rj} = \sum_{r=0}^{i} l_{ir} u_{rj} = \sum_{r=0}^{i-1} l_{ir} u_{rj} + l_{ii} u_{ij} = \sum_{r=0}^{i-1} l_{ir} u_{rj} + 1 u_{ij}$$

$$\Rightarrow \quad u_{ij} = a_{ij} - \sum_{r=0}^{i-1} l_{ir} u_{rj}$$

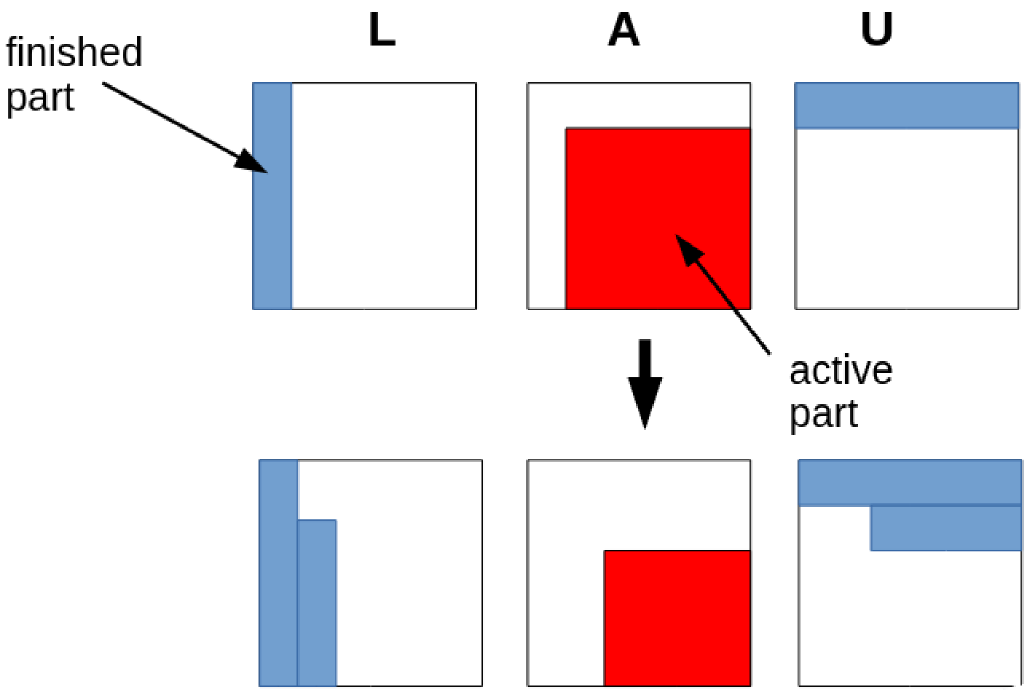
$$a_{ij} = \sum_{r=0}^{n-1} l_{ir} u_{rj} = \sum_{r=0}^{j} l_{ir} u_{rj} = \sum_{r=0}^{j-1} l_{ir} u_{rj} + l_{ij} u_{jj}$$

$$\Rightarrow \quad l_{ij} = \frac{1}{u_{jj}} \left(a_{ij} - \sum_{r=0}^{j-1} l_{ir} u_{rj} \right)$$

Parallel LU Factorization

For each row **i** of **A**

- Compute the **i**th row of **U** $$u_{ij} = a_{ij} - \sum_{r=0}^{i-1} l_{ir} u_{rj}$$
- Compute the **i**th column of **L** $$l_{ij} = \frac{1}{u_{jj}} \left(a_{ij} - \sum_{r=0}^{j-1} l_{ir} u_{rj} \right)$$



LU Decomposition

`LUDecomposition.cpp`

- Open the provided template and implement parallel LU factorization of matrix A using OpenMP