Linear Equations

\[ a_{00} x_0 + a_{01} x_1 + \ldots + a_{0n-1} x_{n-1} = b_0 \]

\[ a_{10} x_0 + a_{11} x_1 + \ldots + a_{1n-1} x_{n-1} = b_1 \]

\[ a_{20} x_0 + a_{21} x_1 + \ldots + a_{2n-1} x_{n-1} = b_2 \]

\[ \ldots \ldots \]

\[ a_{n-1 \ 0} x_0 + a_{n-1 \ 1} x_1 + \ldots a_{n-1 \ n-1} x_{n-1} = b_{n-1} \]

In matrix notation: \( Ax=b \)

matrix A, column vectors x and b
Solving Linear Equations:
Gaussian Elimination

- **Reduce** $Ax=b$ into $Ux=y$
  - $U$ is an upper triangular
  - Diagonal elements $U_{ii} = 1$

$$x_0 + u_{01} x_1 + ... + u_{0,n-1} x_{n-1} = y_0$$
$$x_1 + ... + u_{1,n-1} x_{n-1} = y_1$$
$$........$$
$$x_{n-1} = y_{n-1}$$

- **Back substitute**
Example

\[
\begin{align*}
242 & \quad x_0 = 16 & 121 & \quad x_0 = 8 & 121 & \quad x_0 = 8 \\
374 & \quad x_1 = 29 & 374 & \quad x_1 = 29 & 011 & \quad x_1 = 5 \\
254 & \quad x_2 = 24 & 254 & \quad x_2 = 24 & 012 & \quad x_2 = 8 \\
\end{align*}
\]

\[
\begin{align*}
121 & \quad x_0 = 8 & x_0 + 2x_1 + x_2 = 8 & x_0 = 1 \\
011 & \quad x_1 = 5 & x_1 + x_2 = 5 & x_1 = 2 \\
001 & \quad x_2 = 3 & & x_2 = 3 \\
\end{align*}
\]
Upper Triangularization - Sequential

- Two phases repeated n times
- Consider the k-th iteration (0 ≤ k < n)
- **Phase 1: Normalize k-th row of A**
  
  \[
  \text{for } j = k+1 \text{ to } n-1 \\
  A_{kj} /= A_{kk} \\
  y_k = b_k / A_{kk} \\
  A_{kk} = 1
  \]

- **Phase 2: Eliminate by * - row k**
  
  Using k-th row, make k-th column of A zero for row# > k
  
  \[
  \text{for } i = k+1 \text{ to } n-1 \\
  \text{for } j = k+1 \text{ to } n-1 \\
  A_{ij} = A_{ik} \times A_{kj} \\
  b_i - = a_{ik} \times y_k \\
  A_{ik} = 0
  \]

\[O(n^2)\] divides, \[O(n^3)\] subtracts and multiplies
Upper Triangularization – (Pipelined) Parallel

- \( p = n \), row-striped partition
  
  for all \( P_i \) \((i = 0 .. n-1)\) do in parallel
  
  for \( k = 0 \) to \( n-1 \)
  
  if \((i == k)\)
    
    perform \(k\)-th phase 1: normalize
    
    send normalized row \( k \) down
  
  if \((i > k)\)
    
    receive row \( k \), send it down
    
    perform \(k\)-th phase 2: eliminate with row \( k \)
Example

2 4 2 16
3 7 4 29
2 5 4 24

3 7 4 29
- 3(1 2 1 8) = 0 1 1 5

2 5 4 24
- 2(1 2 1 8) = 0 1 2 8

1 2 1 8

0 1 1 5

0 1 2 8
- 1(0 1 1 5) = 0 0 1 3

x_0 = 1
x_1 = 2
x_2 = 3
Pivoting in Gaussian elimination

- **What if** $A_{kk} \sim 0$?
  - We get a big error, because we divide by $A_{kk}$

- **Find a lower row** $e$ **with largest** $A_{ek}$ **and exchange rows**

- **What if all** $A_{ek} \sim 0$ ?
  - Find a column with largest $A_{ke}$ and exchange columns

- **This complicates the parallel algorithm**
  - Need to keep track of the permutations
Back-substitute – Pipelined row striped

\[
x_0 + u_{0,1} x_1 + \ldots + u_{0,n-1} x_{n-1} = y_0 \\
x_1 + \ldots + u_{1,n-1} x_{n-1} = y_1 \\
\ldots \ldots \\
x_{n-1} = y_{n-1}
\]

for \( k = n-1 \) down to 0

\[
P_k: x_k = y_k ; \text{send } x_k \text{ up} \\
P_{i \ (i<k)}: \text{send } x_k \text{ up, } y_i = y_i - x_k \cdot u_{ik}
\]
LU Decomposition

- In Gaussian elimination we transform $Ax=b$ into $Ux=y$
  i.e., we transform both $A$ & $b$.
- So if we get a new system $Ax=c$, we need to do the whole transformation again.
- LU decomposition treats $A$ independently:
  \[ A = LU \]
  - $L$ unit lower triangular matrix
  - $U$ upper triangular matrix
Forward / backward substitution

- $Ax=b \rightarrow LUx = b$
- Define $Ux=y$, first solve $Ly=b$ then $Ux=y$
- Solving $Ly=b$ forward substitution:
  - fwd substitute $y_1$:
    
    \[
    \begin{align*}
    1 & 0 & 0 & y_1 = b_1 \\
    2 & 1 & 0 & y_2 = b_2 \\
    3 & 4 & 1 & y_3 = b_3 
    \end{align*}
    \]
    
    - $y_2 = b_2 - 2b_1$
    - $y_3 = b_3 - 3b_1$
    
    now fwd substitute $y_2$ and then $y_3$
- Solving $Ux=y$ backward substitution
  - same mechanism, studied for Gaussian elimination
- Both forward and backward easily pipelined
LU decomposition

- **Recursive approach**
  - \( n=1: \) done! \( L=I_1, U = A \)
  - \( n>1: A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & w^T \\ v & A' \end{bmatrix} \)
  - \( v \) and \( w: (n-1)\)-sized vectors
  - \( A = \begin{bmatrix} a_{11} & w^T \\ v & A' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ v/a_{11} & I_{n-1} \end{bmatrix} \begin{bmatrix} a_{11} & w^T \\ 0 & A' - vw^T/a_{11} \end{bmatrix} \)

You can check this by matrix multiplication

Do it by hand for \( n=2 \) and \( n=3 \)

**We have created the first step of the LU decomposition.**
\[ n=2 \]

\[
\begin{bmatrix}
  a_{11} & a_{12} \\
  a_{21} & a_{22}
\end{bmatrix}
= \begin{bmatrix}
  1 & 0 \\
  a_{21}/a_{11} & 1
\end{bmatrix}
\begin{bmatrix}
  a_{11} & a_{12} \\
  0 & p
\end{bmatrix}
\]

\[ p = ? \]

\[ a_{22} = \frac{(a_{21} \times a_{12})}{a_{11}} + p \]
\[ n=2 \]

\[
\begin{bmatrix}
  a_{11} & a_{12} \\
  a_{21} & a_{22}
\end{bmatrix}
= 
\begin{bmatrix}
  1 & 0 \\
  a_{21}/a_{11} & 1
\end{bmatrix}
\begin{bmatrix}
  a_{11} & a_{12} \\
  0 & a_{22}
\end{bmatrix}
\]

\[ p = ? \]
\[ a_{22} = \frac{(a_{21} \times a_{12})}{a_{11}} + p \]

\[ p = a_{22} - \frac{(a_{21} \times a_{12})}{a_{11}} \]
LU decomposition: step 1

- **Recursive approach**
  - n=1: done! $L=I_1$, $U = A$
  - n>1: $A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & w^T \\ v & A' \end{bmatrix}$

  \[ A = \begin{bmatrix} a_{11} & w^T \\ v & A' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \frac{v}{a_{11}} & I_{n-1} \end{bmatrix} \begin{bmatrix} a_{11} & w^T \\ 0 & A' - vw^T / a_{11} \end{bmatrix} \]

You can check this by matrix multiplication.

**We have created the first step of the LU decomposition.**
Inductive Leap

- Solve the rest recursively, ie $A' - vw^T / a_{11} = L' U'$

Element wise this means: $A_{ij} = A_{ik} * A_{kj}$

Very similar to Gauss ($A_{kj} = A_{kk} A_{ij} - A_{ik} * A_{kj}$)

then $A = \begin{bmatrix} 1 & 0 \\ v/a_{11} & I_{n-1} \end{bmatrix} \begin{bmatrix} a_{11} & w^T \\ 0 & L' U' \end{bmatrix}

= \begin{bmatrix} 1 & 0 \\ v/a_{11} & L' \end{bmatrix} \begin{bmatrix} a_{11} & w^T \\ 0 & U' \end{bmatrix}$

- In place: $L$ and $U$ stored and computed in $A$
  - 0-s and 1-s of $L$ and $U$ are implicit (not stored)
The code

for k = 1 to n {
    for i = k+1 to n
        Aik /= Akk
    for i = k+1 to n
        for j = k+1 to n
            Aij -= Aik*Akj
}

1 4 5
2 10 16
3 20 42

k=1:
    divide down by 1
    then *

1 4 5
2 2 6
3 8 27

k=2:
    divide down by 2
    then *

1 4 5
2 2 6
3 4 3

CHECK!!!

how?
Pivoting

**Why?**

A_{kk} can be close to 0 causing big error in 1/A_{kk}, so we should find the absolute largest A_{ik} and swap rows i and k.

**How?**

- We can register this in a permutation matrix or better an n sized array π[i]: position of row i.

**No pivoting:**

- LU code is a straightforward translation of our recurrence.
LUD: data dependence

\[
\begin{align*}
&\text{for } k = 1 \text{ to } n \{ \\
&\quad \text{for } i = k+1 \text{ to } n \\
&\quad \quad A_{ik} /= A_{kk} \\
&\quad \text{for } i = k+1 \text{ to } n \\
&\quad \quad \text{for } j = k+1 \text{ to } n \\
&\quad \quad \quad A_{ij} -= A_{ik} \times A_{kj}
\}
\end{align*}
\]
Pipelining

- Iteration $k+1$ can start after iteration $k$ has finished row $k+1$
  - neither row $k$ nor column $k$ are read or written anymore

- This naturally leads to a streaming algorithm
  - every iteration becomes a process
  - results from iteration/process $k$ flow to process $k+1$
  - process $k$ uses row $k$ to update sub matrix $A[k+1:n, k+1:n]$

- Actually, only a fixed number $P$ of processes run in parallel
  - after that, data is re-circulated
Pipelined LU

- Pipeline of P processes
  - parallel sections in OpenMP lingo
  - algorithm runs in \( \text{ceil}(N/P) \) sweeps

- In one sweep P outer \( k \) iterations run in parallel

- After each sweep P rows and columns are done
  - Problem size \( n \) is reduced to \((n-P)^2\)
  - Rest of the (updated) array is re-circulated

- Each process owns a row
  - Process \( p \) owns row \( p + (i-1)P \) in sweep \( i \), \( i=1 \) to \( \text{ceil}(N/P) \)
  - Array elements are updated while streaming through the pipe of processes

- Total work \( O(n^3) \), speedup \( \sim P \)

- Forward / backward substitution are simple loops
First, flip the matrix vertically.
Feed matrix to Processor 1
Feed matrix to Processor 2
Memories:
- M1 -> M2 on even sweeps
- M2 -> M1 on odd sweeps
- M3 : siphoning off finished results

Processes are grouped in OpenMP style parallel sections with intermediate data in streams

- **Process 0** reads either from M1 or M2, writes to stream S0
- **Process i** reads from stream Si-1, writes to Si
- **Process P-1**
  - reads from stream Sp-1
  - writes finished (black) data to M3
  - writes rest to M1 or M2