Gaussian Elimination

Linear systems of equations

- A common task in scientific computation is to solve a system of linear equations
- Often result from of discretizing a differential equation
- Example: linear system of 2 equations in 2 unknowns

  (1) $2x + 3y = 8$
  (2) $3x + 2y = 7$

- Rewriting equation (1)

  $x = (8-3y)/2$

- Substituting x into the LHS of equation (2)
  $3(8-3y)/2 + 2y = (24-9y)/2 + 2y$
  $\Rightarrow (24-9y) + 4y = 14 \Rightarrow 10 = 5y \Rightarrow y = 2$
- Back substituting the value of y into equation (1)

  $x = 1$

Matrix vector equations

- Our linear system of 2 equations in 2 unknowns …

  $2x_1 + 3x_2 = 8$

  $3x_1 + 2x_2 = 7$

- may be conveniently expressed in matrix notation: A**x** = b

$$A = \begin{bmatrix} 2 & 3 \\ 3 & 2 \end{bmatrix}, x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, b = \begin{bmatrix} 8 \\ 7 \end{bmatrix}$$

- When we solved for $x_1 = (8-3\ x_2)/2$ and substituted the value of x_1 into the 2nd equation, we reduced the matrix to an equivalent form

$$A = \begin{bmatrix} 2 & 3 \\ 0 & -2.5 \end{bmatrix}, x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, b = \begin{bmatrix} 8 \\ -5 \end{bmatrix}$$

- We multiplied row 1 of A by 3/2 and subtracting the scaled version from row 2 of **A** and **b**
- We call this a ***rank-1 update***

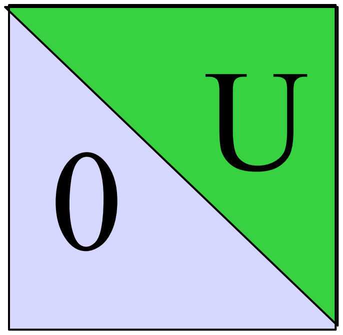
Rank 1 updates

- Multiplying row 1 by 3/2: [3   9/2]
- Subtracting from row 2:

- Similarly for **b**

Gaussian Elimination

- The process of eliminating the non-zero values under the main diagonal is called ***Gaussian Elimination***, named after the mathematician
Johann Carl Friedrich Gauss (1777-1855)
- Input: an n × n matrix corresponding to a linear system of n equations in n unknowns (must have non-trivial sol'n)
- Output: an n × n matrix with zeroes under the main diagonal: an
upper triangular matrix **U**



Cost

- To solve $A\mathbf{x} = \mathbf{b}$
  - Factorize $A = LU$ using GE *(2/3 n^3 flops)*
  - Solve $L\mathbf{y} = \mathbf{b}$ for $\mathbf{y}$ using substitution (n^2 flops)
  - Solve $U\mathbf{x} = \mathbf{y}$ for $\mathbf{x}$ using back substitution (n^2 flops)
- We don't compute U explicitly unless we are solving for multiple right hand sides **b**
- Focus on factorization, which is much more expensive

A 3 × 3 example

- Consider the following system of equations

$$
\begin{aligned}
x_0 &+ x_1 + x_2 = 3 \\
4x_0 &+ 3x_1 + 4x_2 = 8 \\
9x_0 &+ 3x_1 + 4x_2 = 7
\end{aligned}
$$

- We usually write the system as an ***augmented matrix***

$$
\left[\begin{array}{ccc|c}
1 & 1 & 1 & 3 \\
4 & 3 & 4 & 8 \\
9 & 3 & 4 & 7
\end{array}\right]
$$

3×3 example

- Multiply row 0 by 4, and subtract from row 1

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & 3 \\ 4 & 3 & 4 & 8 \\ 9 & 3 & 4 & 7 \end{array}\right]$$

[4 3 4 8] – 4*[1 1 1 3] = [0 -1 0 -4]

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & 3 \\ 0 & -1 & 0 & -4 \\ 9 & 3 & 4 & 7 \end{array}\right]$$

3×3 example

- Multiply row 0 by 9, and subtract from row 2

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & 3 \\ 0 & -1 & 0 & -4 \\ 9 & 3 & 4 & 7 \end{array}\right]$$

[9 3 4 7] – 9*[1 1 1 3] = [0 -6 -5 -20]

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & 3 \\ 0 & -1 & 0 & -4 \\ 0 & -6 & -5 & -20 \end{array}\right]$$

3×3 example

- Eliminate second column
- Multiply row 1 by 6, and add to row 2

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & 3 \\ 0 & -1 & 0 & -4 \\ 0 & -6 & -5 & -20 \end{array}\right]$$

[0 -6 -5 -20] + -6*[0 -1 0 -4]
= [0 0 -5 4]

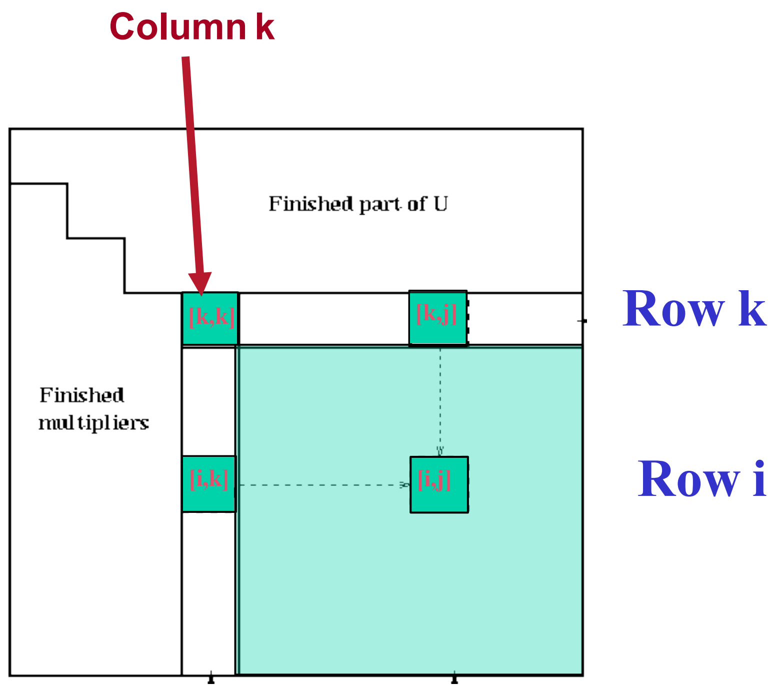
$$\left[\begin{array}{ccc} 1 & 1 & 1 \\ 0 & -1 & 0 \\ 0 & 0 & -5 \end{array}\right] \begin{bmatrix} 3 \\ -4 \\ 4 \end{bmatrix}$$

Gaussian Elimination (GE)

- Add multiples of each row to later rows to make A upper triangular

… for each column k
… zero it out below the diagonal by adding multiples of row k to later rows

```
for k = 0 to n-1
    … for each row i below row k
    for i = k+1 to n-1
        … add a multiple of row k to row I
        for j = k+1 to n-1
            A[i,j] -= A[ i , k] / A[k,k] * A[k,j]
```



Roundoff issues

- The rank-1 update step uses division …

```
A[i, k+1:n ] -= ( A[i,k]/A[k,k]) * A[k,k+1:n]
```

- We need to be able to handle vanishing denominators or ones that are very small
- Gaussian elimination will fail with this matrix

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

- But we can avoid the problem if we swap rows

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Pivoting to avoid stability problems

- We call this process of swapping rows ***partial pivoting***
- Assume we carry 3 decimal digits of precision
- Consider the following A matrix and RHS b

$$A = \begin{bmatrix} 10^{-4} & 1 \\ 1 & 1 \end{bmatrix} \quad b = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

- The correct solution is

$$x = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Roundoff Error

- Eliminate zero in row 2 by subtracting $10^4 \times$ row 0

$$L|b = \left[\begin{array}{cc|c} 10^{-4} & 1 & 1 \\ 0 & 1 - 10^4 & 2 - 10^4 \end{array}\right]$$

- But $1 - 10^4$ rounds to -10^4

$$L|b = \left[\begin{array}{cc|c} 10^{-4} & 1 & 1 \\ 0 & -10^4 & -10^4 \end{array}\right]$$

- Back substituting to solve for x_2 and then x_1

  $-10^4 x_2 = -10^4 \Rightarrow$ $\mathbf{x_2 = 1}$

- Substituting the value of x_2 into the first equation

  $10^{-4} x_1 + 1 * x_2 = 1 \Rightarrow 10^{-4} x_1 = 0 \Rightarrow$ $\mathbf{x_1 = 0}$

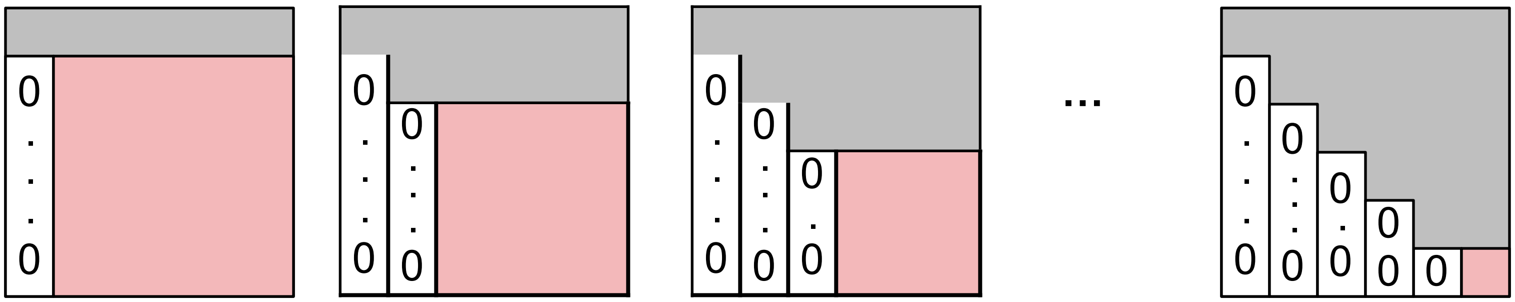
- **But the correct solution is $x_1 = x_2 = 1$**

Partial Pivoting

- Rule: pick the largest value in the column
- This is called partial pivoting, because only rows are swapped
- It can be shown that when with partial pivoting, we compute **A = P L U**, where P is a permutation matrix expressing the rows swaps
- We can also swap columns: ***full pivoting***
- But full pivoting is more expensive to implement

Parallelization

- We’ll use 1D vertical strip partitioning
- Each process owns N/p columns
- Consider the case where p=N=6
- The represents outstanding work in succeeding k iterations



After i=1 **After i=2** **After i=3** **After i=n-1**

Parallelism and data dependencies

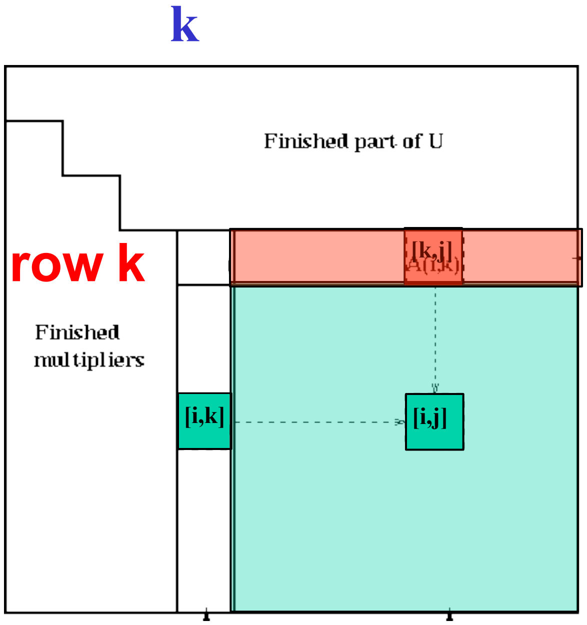
- Analyze the code to determine communication requirements
- Assume blocked decomposition on the 2nd axis
- Each process in charge of eliminating N/P columns
- Parallelism occurs in *array statements*
- One process chooses pivot row, computes multipliers

```
for k = 0 to n-1              // For each column k

   m[k+1:n-1]  =  A[k+1:n-1,k] / A[ k , k]  // Compute Multipliers

   for i = k+1 to n-1         //  for each row i > k
     A[ i , k+1: n-1] - = m[ i ] * A[ k , k+1:n-1 ] // Scale row k by m_ik
                                                     // & subtract from row i

   end for
end for
```

Determining communication requirements: computing the multipliers

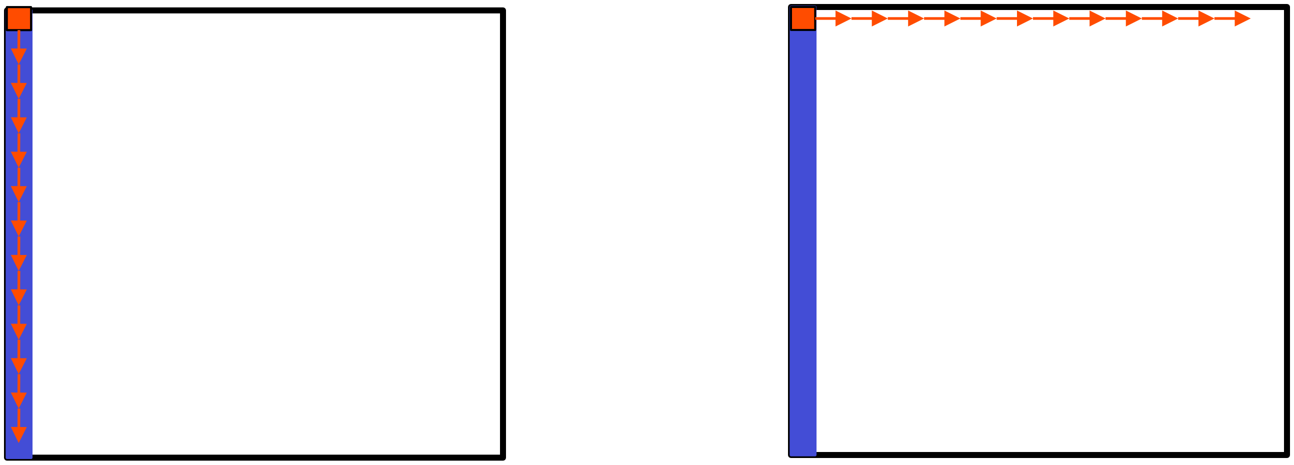
- At each step **k** of the elimination, processor **k div p** is in charge: it computes the multipliers
- No communication is needed: all the required data are ***owned*** by processor **k div p**

```
for k = 0 to n-1
   m[k+1:n-1]  =   A[k+1:n-1,k] / A[ k , k]

   for i = k+1 to n-1
     A[ i , k+1:n-1 ] - = m[ j ] *A[ k ,k+1:n-1]
  end for

end for
```



Communication and control

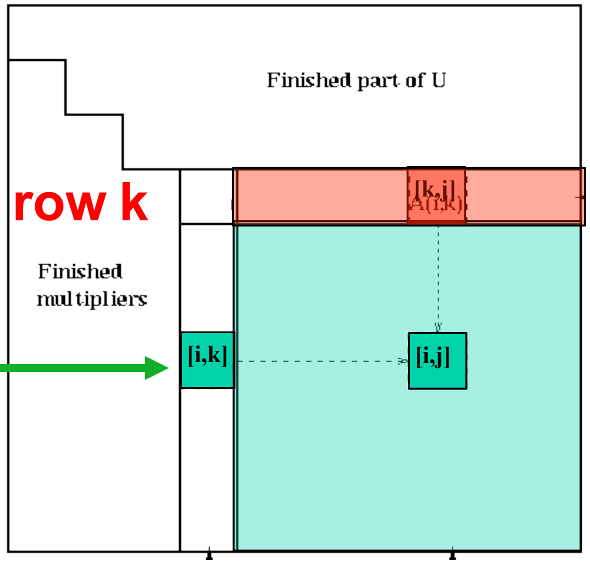
- Each process in charge of eliminating N/P columns
- It chooses the pivot row and computes the multipliers
- The multipliers are then broadcasted



Determining communication requirements: trailing matrix update

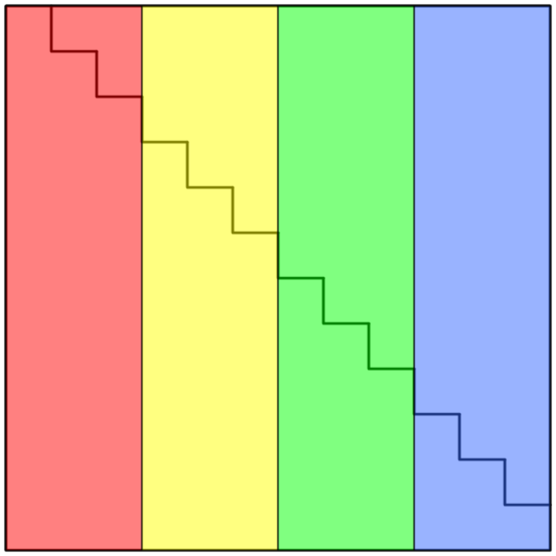
- Elements in A[k , k+1: n] (row k) have different owners
- Process **j div p** owns **A[k,j]** in A[k , k+1: n]
- What operation is needed to carry out the **k** multiplication **m[j] * A[k , :]** ?

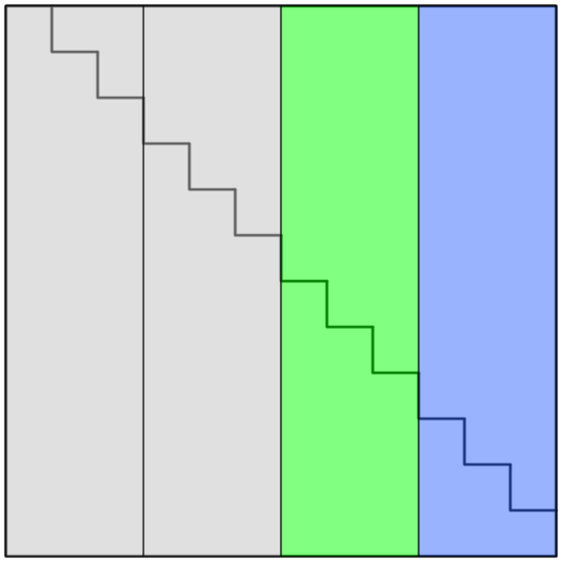
```
for k = 0 to n-1
   m[k+1:n-1]  =   A[k+1:n-1,k] / A[ k , k]
   // Scale row k by m_ik and
   //   subtract from row l
   for j = k+1 to n-1   //  for each row i > k
     A[ j ,k+1:n-1] - = m[ j ] *A[ k ,k+1:n-1]
   end for
end for
```



Performance

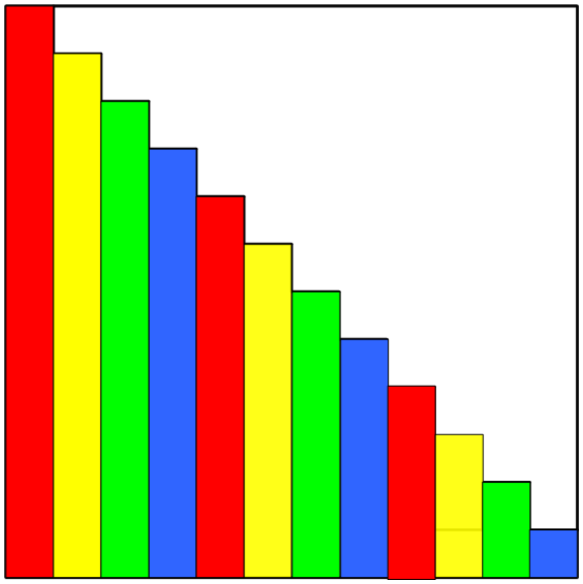
- Finding the pivot row is a serial bottleneck
  - Only one process owns the intersecting column
- Another bottleneck is load imbalance
  - When eliminating a column, processors to the left of are idle
  - Each processor is active for only part of the computation

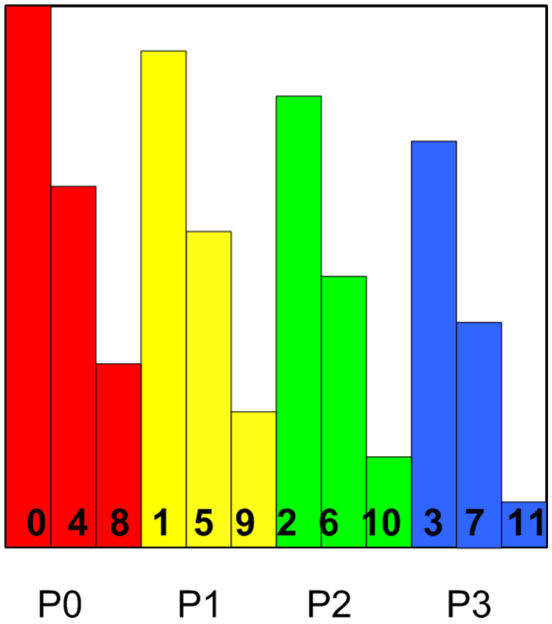


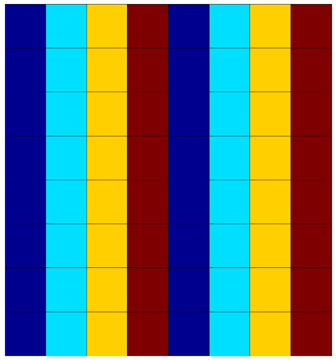


Cyclic decomposition improves load balance

- A *cyclic* decomposition evens out the workload
- A blocked cyclic decomposition improves locality and reduces communication overhead

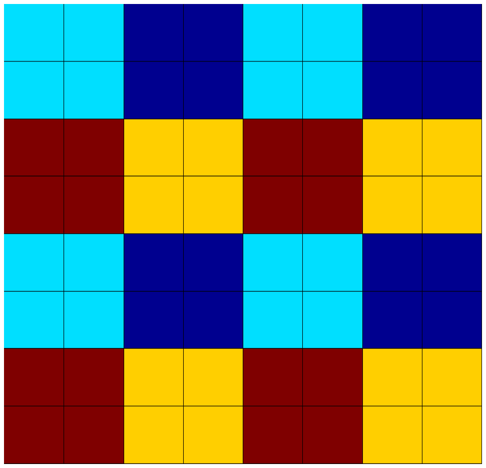






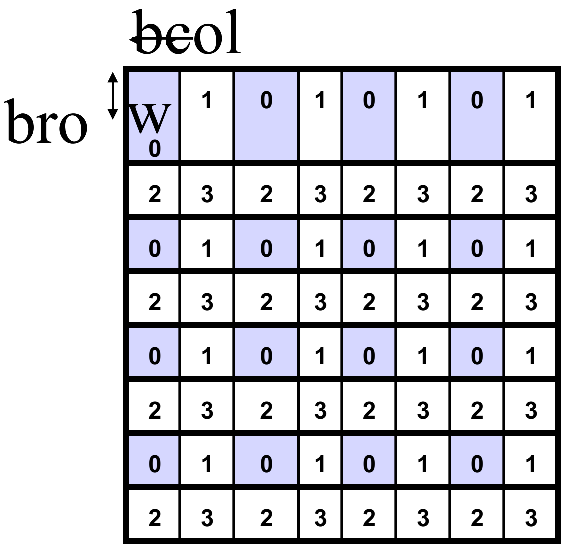
In practice

- 2D block cyclic decompositions required
- Why is 1D block cyclic not scalable?
- More complicated since additional communication steps are required
- The algorithm is blocked as with matrix multiply



- ScaLAPACK is a well known library that performs GE and many other useful operations involving matrices
- See **http://www.netlib.org/scalapack**

Row and Column Block Cyclic Layout



- Processors and matrix blocks are distributed in a 2d array
  - prow × pcol array of processors
  - brow × bcol matrix blocks

- pcol-way parallelism in a column
- calls to BLAS2 and BLAS3 on matrices of size brow × bcol

- Reduces serial bottleneck

- prow ≠ pcol and brow ≠ bcol possible, possibly desirable