Lecture 18

Solving systems of equations:
LU Decomposition

Building a linear time barrier with locks

Lock arrival=UNLOCKED, departure=LOCKED;
int count=0;
void Barrier( )
    arrival.lock( );  // atomically count the
    count++;        // waiting threads
    if (count < n$proc) arrival.unlock( );
    else departure.unlock( ); // last processor
        // enables all to go
departure.lock( );
count--;        // atomically decrement
if (count > 0) departure.unlock( );
else arrival.unlock( ); // last processor resets
    state
Linear systems of equations

- A common application scientific computation is to solve a system of linear equations
- Consider the linear system of 2 equations in 2 unknowns \( x \) and \( y \)
  
  \[
  \begin{align*}
  (1) & \quad 2x + 3y = 8 \\
  (2) & \quad 3x + 2y = 7
  \end{align*}
  \]
- Rewriting equation (1)
  \[
  x = \frac{(8-3y)}{2}
  \]
- Substituting \( x \) into the LHS of equation (2)
  \[
  3\left(\frac{8-3y}{2}\right) + 2y = \frac{(24-9y)}{2} + 2y \\
  \Rightarrow 24 - 9y + 4y = 14 \Rightarrow 10 = 5y \Rightarrow y = 2
  \]
- Back substituting the value of \( y \) into equation (1)
  \[
  x = 1
  \]

Matrix vector equations

- Our linear system of 2 equations in 2 unknowns …
  \[
  \begin{align*}
  2x_1 + 3x_2 &= 8 \\
  3x_1 + 2x_2 &= 7
  \end{align*}
  \]
- may be conveniently expressed in matrix notation: \( Ax = b \)

  \[
  A = \begin{pmatrix} 2 & 3 \\ 3 & 2 \end{pmatrix}, x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, b = \begin{pmatrix} 8 \\ 7 \end{pmatrix}
  \]

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

  \[
  A = \begin{pmatrix} 2 & 3 \\ 0 & -2.5 \end{pmatrix}, b = \begin{pmatrix} 8 \\ -5 \end{pmatrix}
  \]

- We did this by multiplying row 1 of \( A \) by \( 3/2 \) and subtracting the scaled version from row 2 of \( A \) and \( b \)
Rank 1 updates

- We call this a rank-1 update
- Multiplying row 1 by 3/2: \[ \begin{pmatrix} 3 & 9/2 \end{pmatrix} \]
- Subtracting from row 2:
- Similarly for \( b \)
- An applet for looking at 2x2 systems: www.freeboundaries.org/java/la_applets/GaussElim/

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 \( n \times n \) matrix corresponding to a linear system of \( n \) equations in \( n \) unknowns (must have non-trivial sol’n)
- Eliminate the non-zero values under the main diagonal to produce an upper triangular matrix \( U \)
Solving the system of linear equations

- Once we have the upper triangular matrix U, the remaining step is trivial.
- Solve the corresponding upper triangular system $Ux = c$ by back substitution.

What are we computing?

- GE computes the *LU factorization* $A = LU$, where $L$ is a lower triangular matrix.
- Plugging $LU$ into the original equation $Ax = b$
  
  $Ax = (LU)x = L(Ux) = Ly = b$ where $y = Ux$. 

\[
\begin{bmatrix}
A \\
L \\
0
\end{bmatrix} =
\begin{bmatrix}
0 \\
U \\
0
\end{bmatrix}
\]
Solving the equations

- Plugging LU into the original equation $Ax = b$
  
  $$Ax = (LU) x = L (Ux) = Ly = b \text{ where } y = Ux$$

- To solve $Ax = b$
  
  - Factorize $A = LU$ using GE \((2/3 n^3 \text{ flops})\)
  
  - Solve $Ly = b$ for $y$ using substitution \((n^2 \text{ flops})\)
  
  - Solve $Ux = y$ for $x$ using back substitution \((n^2 \text{ flops})\)

- We don’t compute $U$ unless we are solving for multiple right hand sides $b$

The algorithm

- We eliminate non-zeroes below the diagonal …
  
  - One column at a time
  
  - Scanning from left to right
Eliminating the entries below the diagonal

- Say we are reducing column \( k \)
- We subtract various multiples of row \( k \), \( A[k,k+1:n] \), from rows \( j = k+1 \) to \( n \)
- These multipliers are chosen to zero out the below diagonal elements
- We only update to the right of and below \( A[k,k] \)

- The multiplier for row \( j, j = k+1:n \) is \( A[j,k]/A[k,k] \)
- Observe that \( A[k,k] \cdot A[j,k]/A[k,k] - A[j,k] = 0 \)

A closer look

- \( A[k+1:n-1,j] \)
- \( A[k,k+1:n-1] \)
- \( A[k+1:n-1,k+1:n-1] \)
An example

- Consider the following system of equations

\[
\begin{align*}
x_0 + x_1 + x_2 &= 3 \\
4x_0 + 3x_1 + 4x_2 &= 8 \\
9x_0 + 3x_1 + 4x_2 &= 7
\end{align*}
\]

- We usually write the system as an augmented matrix

\[
\begin{bmatrix}
1 & 1 & 1 & 3 \\
4 & 3 & 4 & 8 \\
9 & 3 & 4 & 7
\end{bmatrix}
\]

An example

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

\[
\begin{bmatrix}
1 & 1 & 1 & 3 \\
4 & 3 & 4 & 8 \\
9 & 3 & 4 & 7
\end{bmatrix}
\]

\[\begin{bmatrix}
4 & 3 & 4 & 0 \\
-1 & -1 & -4 & -4
\end{bmatrix}
\]

\[\begin{bmatrix}
4 & 3 & 4 & 0 \\
-1 & -1 & -4 & -4
\end{bmatrix}
\]
An example

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

\[
\begin{bmatrix}
1 & 1 & 1 & 3 \\
0 & -1 & 0 & -4 \\
9 & 3 & 4 & 7 \\
\end{bmatrix}
\]

\[
[4 \ 3 \ 4 \ 8] - 4[1 \ 1 \ 1 \ 3] = [0 \ -1 \ 0 \ -4]
\]

An example

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

\[
\begin{bmatrix}
1 & 1 & 1 & 3 \\
0 & -1 & 0 & -4 \\
0 & -6 & -5 & -20 \\
\end{bmatrix}
\]

\[
[9 \ 3 \ 4 \ 7] - 9[1 \ 1 \ 1 \ 3] = [0 \ -6 \ -5 \ -20]
\]
An example

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

\[
\begin{bmatrix}
1 & 1 & 1 \\
0 & -1 & 0 \\
0 & 0 & -5 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 1 & 1 & 3 \\
0 & -1 & 0 & -4 \\
0 & 0 & -5 & 4 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
0 & -6 & -5 & -20 \\
\end{bmatrix}
+ \begin{bmatrix}
-6 & 0 & 0 & -4 \\
\end{bmatrix} = \begin{bmatrix}
0 & 0 & -5 & 4 \\
\end{bmatrix}
\]

The computational inner loops

- In column k
  - Scale row k according to the multiplier
  - Subtract from row j
- The factors can be determined for the entire column by computing

\[
A[k+1:n-1,k] / A[k,k]
\]
Putting it all together

- In column \( k \)
  - Multiply row \( k \) by a factor
  - Subtract from row \( j \)
- Factors \( p[k+1:n-1] \)
  \[
  \begin{bmatrix}
    A[k+1:n-1,k] / A[k,k]
  \end{bmatrix}
  \]

for \( k = 0 \) to \( n-1 \)  
  // For each column \( k \)
  \[
  m[k+1:n-1] = \frac{A[k+1:n-1,k]}{A[k,k]}
  \]
  for \( j = k+1 \) to \( n-1 \)  
    // Update by adding a multiple of row \( k \) to all succeeding rows \( j \)
    \[
    A[j,:) -= m[j] \cdot A[k,:]
    \]
end for
double

Avoiding severe roundoff errors

- Because the rank-1 update step uses division …
  \[
  A[j,:] -= \left( \frac{A[j,i]}{A[i,i]} \right) \cdot A[i,:]
  \]
- we need to be careful about a vanishing denominator or one that is 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

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

$$L = \begin{bmatrix} 10^{-4} & 1 \\ 0 & 1 - 10^4 \end{bmatrix}$$

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

$$L = \begin{bmatrix} 10^{-4} & 1 \\ 0 & -10^4 \end{bmatrix}$$
Stability problems due to roundoff

• Thus

\[
L | b = \begin{bmatrix}
10^{-4} & 1 & 1 \\
0 & -10^4 & -10^4
\end{bmatrix}
\]

• We now back substitute to solve for \( x_2 \) and then \( x_1 \)
  \[-10^4 x_2 = -10^4 \Rightarrow x_2 = 1\]

• Substituting the value of \( x_2 \) into the first equation
  \[10^{-4} x_1 + 1* x_2 = 1 \Rightarrow 10^{-4} x_1 = 0 \Rightarrow x_1 = 0\]

• But the correct solution is \( x_1 = x_2 = 1 \)

Partial Pivoting

• The general rule is to pick the largest value in the column we are eliminating, and choose the intersecting row as the pivot row

• This is called partial pivoting, because only rows are swapped

• It can be shown that when with partial pivoting, we compute \( A = PLU \), where \( P \) is a permutation matrix expressing the rows swaps

• We can also swap columns: \textit{full pivoting}

• But full pivoting is more expensive to implement
Parallelization

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

```
for k = 0 to n-1     // For each column k
    m[k+1:n-1] = A[k+1:n-1, k] / A[ k , k]
    for j = k+1 to n-1  // Update by adding a multiple of row k to all succeeding rows j
    end for
end for
```

Analyzing the code

- Let’s look at the data parallel code to understand where the communication is occurring
- Assume blocked decomposition on the 2nd axis
- Parallelism occurs in array statements
Determining communication requirements

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

Determining communication requirements

- But the elements $A[k,:]$ can have different owners.
- Processor $j \div p$ owns $A[k,j]$.
- What operation is needed to carry out the multiplication $m[j] * A[k,:]$?

for $k = 0$ to $n-1$
  $m[k+1:n-1] = A[k+1:n-1,k] / A[k,k]$
  for $j = k+1$ to $n-1$
    $A[j,:] - = m[j] * A[k,:]$
  end for
end for
Communication and control

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

Communication and control

- All processors carry out updates
Performance

- Finding the pivot row is a serial bottleneck
- Only one processor owns the intersecting column
- Another bottleneck is load imbalance

Load imbalance

- This is a classic problem with a convenient soln
- When we are eliminating column k, processors to the left of k’s owner will sit idle
Load imbalance

- Vertical decomposition
- \( n \gg p \)
- Each processor is active for only part of the computation
Load imbalance

- Vertical decomposition
- \( n \gg p \)
- 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 are needed; 1D is not scalable
• The algorithm is a bit more complicated since more communication steps are required
• The algorithm is blocked as with matrix multiply
• See the Demmel reader if you are interested in the details
• Scalapack is a well known library that performs GE and many other useful operations involving matrices
• See [http://www.netlib.org/scalapack/](http://www.netlib.org/scalapack/)