Lecture 11

Matrix Factorization
Programming Heterogeneous Architectures
Announcements

• Office hours cancelled on Thurs
• Project presentations
## Results

<table>
<thead>
<tr>
<th></th>
<th>1060 N</th>
<th>1060 SM</th>
<th>1060 B</th>
<th>570N</th>
<th>570SM</th>
<th>570L1</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP</td>
<td>348(1K)</td>
<td>113 (4K)</td>
<td></td>
<td>596(1K)</td>
<td></td>
<td>265(4K)</td>
</tr>
<tr>
<td>DP</td>
<td>66.7 (1K)</td>
<td></td>
<td></td>
<td>79/80 (1K)</td>
<td>134(1K)</td>
<td>132(4K)</td>
</tr>
</tbody>
</table>
Today’s lecture

• Matrix Factorization
• Programming Heterogeneous architectures
Linear systems of equations

- A linear system of 2 equations in 2 unknowns …
  \[ 2x_1 + 3x_2 = 8 \]
  \[ 3x_1 + 2x_2 = 7 \]
- Solution is \( x=1, y=2 \)
- We may express the system using 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 solve for \( x_1 = (8-3x_2)/2 \) and substitute into the 2nd equation, we reduce the matrix to an equivalent form
  \[
  A = \begin{pmatrix}
    2 & 3 \\
    0 & -2.5
  \end{pmatrix},
  b = \begin{pmatrix}
    8 \\
    -5
  \end{pmatrix}
  \]
- We multiply row 1 of \( A \) by 3/2 and subtracting the scaled version from row 2 of \( A \) and \( b \)
A 3 × 3 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

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

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

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

\[
\begin{bmatrix}
4 & 3 & 4 & 8 \\
\end{bmatrix}
- 4\times\begin{bmatrix}
1 & 1 & 1 & 3 \\
\end{bmatrix} = \begin{bmatrix}
0 & -1 & 0 & -4 \\
\end{bmatrix}
\]
3 × 3 example

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

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

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

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

\[
\begin{bmatrix}
0 & -6 & -5 & -20 \\
0 & -1 & 0 & -4 \\
0 & -6 & -5 & -20
\end{bmatrix}
+ -6 \times
\begin{bmatrix}
0 & -1 & 0 & -4 \\
0 & 0 & -5 & 4
\end{bmatrix}
= \begin{bmatrix}
0 & 0 & -5 & 4
\end{bmatrix}
\]
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)

• Output: $n \times n$ matrix with non-zero values above the main diagonal only

*upper triangular matrix* $U$
What are we computing?

- GE computes the *LU factorization* $A = L \ U$, where $L$ is a *lower triangular matrix*
- Plugging $LU$ into the original equation $Ax = b$
  
  $Ax = (LU) \ x = L \ (Ux) = Ly = b$ where $y = Ux$
Solving the system of linear equations

- Step 1: obtain the upper triangular matrix $U$ …
- Step 2: solve the corresponding upper triangular system $Ux = c$ by *back substitution*
- Focus on step 1, which is much more expensive
  - $O(n^3)$ vs $O(n^2)$
Cost

• To solve $A\mathbf{x} = \mathbf{b}$
  ‣ Plug LU into the original equation $A\mathbf{x} = \mathbf{b}$
    
    $A\mathbf{x} = (LU) \mathbf{x} = L (U\mathbf{x}) = Ly = \mathbf{b}$ where $y = U\mathbf{x}$
  ‣ Factorize $A = LU$ using GE \((2/3 \ n^3 \text{ flops})\)
  ‣ Solve $Ly = \mathbf{b}$ for $y$ using substitution \((n^2 \text{ flops})\)
  ‣ Solve $U\mathbf{x} = y$ for $\mathbf{x}$ using back substitution \((n^2 \text{ flops})\)

• We don’t compute $U$ explicitly unless we are solving for multiple right hand sides $\mathbf{b}$
Visualizing the algorithm

- Eliminate non-zeroes below the diagonal …
  - One column at a time
  - Scanning from left to right

for k = 0 to n-1
  // For each column k
  for i = k+1 to n-1
    // Eliminate entries below the diagonal:
    // subtract a multiple of row k
    // from succeeding rows
  end for
end for

Column k=0  Column k 1  Column k=2  Column k= n-1

Courtesy J. Demmel
Rank-1 updates

- For each column $k$ in 0 to $n-1$
- … subtract multiples of row $k$: $A[k,k+1:n]$
- … from rows $i = k+1$ to $n$
  - Multipliers $m_{ik} = A[i,k]/A[k,k]$
  - … cancel the elements below the diagonal: $A[k+1:n-1,k]$
  - Thus $A[k,k]*m_k - A[j,k] = 0$

- Update only to right and below $A[k,k]$

- Rank 1 updates are costly
Blocked algorithm

Blocked algorithm eliminates a block column at a time
Aggregate the trailing submatrix updates from many columns
Delayed updated uses fast BLAS 3 ops rather than slow BLAS 2 ops (rank 1 update) to update the trailing submatrix

\[ a_3 = a_3 - a_1 * a_2 \]

then apply nb to rest of matrix

\[ a_2 = L^{-1} a_2 \]

apply sequence to nb

Courtesy J. Dongarra
Pivoting

• The rank-1 update step uses division …

\[ A[i, \, k+1:n \, ] \, \text{=} \, ( \, A[i,k]/A[k,k] \, ) \, \times \, A[k,k+1:n] \]

• How 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*
- Also handles tiny pivots
- Assume we carry 3 decimal digits of precision
- Consider the following A matrix and RHS b

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

- We compute \( x = [1 \ 0]^T \)
- The correct solution is \( x = [1 \ 1]^T \)
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 with partial pivoting, we compute $A = PLU$, 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

- Consider 1D vertical strip partitioning
- Each thread owns $N/P$ columns
- Let $P=N=6$
- The ■ represents outstanding work in succeeding $k$ iterations
Parallelization

- Consider 1D vertical strip partitioning
- Each thread owns N/P columns
- Let P=N=6

- ■ = outstanding work in succeeding k iterations
Communication and control

- Each thread is in charge of eliminating N/P columns
- One thread chooses the pivot row and computes the multipliers
- The other threads share this value
- All threads carry out updates
Analyzing the Parallel Control flow

• All threads carry out updates
• At each step \( k \), thread \( k \div p \) is in charge:
  it computes the multipliers \( m[ ] \)
• Elements in \( A[ k , k+1: n ] \) (row \( k \)) have different owners
• Thread \( j \div p \) owns \( A[k,j] \) in \( A[ k , k+1: n ] \)

\[
\text{for } k = 0 \text{ to } n-1 \quad \text{// For each column } k
\]

\[
\begin{align*}
\text{// Compute Multipliers} \\
m[k+1:n-1] & = A[k+1:n-1,k] / A[ k , k] \\
\text{for } i = k+1 \text{ to } n-1 \quad \text{// for each row } i > k
\end{align*}
\]

\[
\begin{align*}
\text{// Scale row } k \text{ by } m_{ik} \text{ and} \\
\text{// subtract from row } i
\end{align*}
\]

\[
A[ i , k+1: n] - = m[ i ] * A[ k , k+1:n ]
\]

end for
end for
Performance

• Finding the pivot row is a serial bottleneck
  ‣ Only one thread owns the intersecting column

• Another bottleneck is load imbalance
  ‣ When eliminating a column, processors to its left sit 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

- 1D is not scalable; 2D block cyclic decompositions required
- More complicated since additional communication steps are required
- Used in Scalapack
  www.netlib.org/scalapack