Lecture 5

Applications:
Numerical Linear Algebra
Announcements

• Partners?
Today’s lecture

• Finishing up with Vectorization
• Optimizing for the memory hierarchy
  ‣ Matrix Multiplication
• Gaussian Elimination
  ‣ The parallel algorithm
What prevents vectorization

• Data dependencies
• We have a loop carried dependence

for (int i = 1; i < n; i++)

    b[i] = b[i-1] + 2;

note: not vectorized: unsupported use in stmt.

b[1] = b[0] + 2;
Blocking for Cache
Matrix Multiplication

• An important core operation in many numerical algorithms

• Given two *conforming* matrices $A$ and $B$, form the matrix product $A \times B$
  
  $A$ is $m \times n$
  $B$ is $n \times p$

• Operation count: $O(n^3)$ multiply-adds for an $n \times n$ square matrix

• Discussion follows from Demmel

www.cs.berkeley.edu/~demmel/cs267_Spr99/Lectures/Lect02.html
Unblocked Matrix Multiplication

for $i = 0$ to $n-1$
for $j = 0$ to $n-1$
for $k = 0$ to $n-1$

$$C[i,j] += A[i,k] \times B[k,j]$$
Analysis of performance

for $i = 0$ to $n-1$
  // for each iteration $i$, load all of $B$ into cache
  // for each iteration $i$, load $A[i,:]$ into cache
  for $j = 0$ to $n-1$
    // for each iteration $(i,j)$, load and store $C[i,j]$ 
    for $k = 0$ to $n-1$
      $C[i,j] += A[i,k] \times B[k,j]$

Scott B. Baden / CSE 160 / Winter 2011
Analysis of performance

for $i = 0$ to $n-1$

// $n \times n^2 / L$ loads $= n^3/L$, $L =$ cache line size  
B[:, :]

// $n^2 / L$ loads $= n^2/L$  
A[i, :]

for $j = 0$ to $n-1$

// $n^2 / L$ loads + $n^2 / L$ stores $= 2n^2 / L$  
C[i, j]

for $k = 0$ to $n-1$

$C[i, j] += A[i, k] \times B[k, j]$  
Total: $(n^3 + 3n^2) / L$
Let $q = \#\text{ flops} / \text{main memory reference}$

$$q = \frac{2n^3}{n^3 + 3n^2}$$

$\approx 2$ as $n \to \infty$
Blocked Matrix Multiply

- Divide A, B, C into sub blocks
- Each sub block is $b \times b$
  - There are $N \times N$ blocks
  - $b=n/N$ is called the block size
  - How do we establish $b$?

- Assume we have a good quality library to perform matrix multiplication on sub blocks
  - ATLAS, GotoBLAS2
  - Intel Math Kernel Library
Blocked Matrix Multiplication

for i = 0 to N-1
    for j = 0 to N-1
        // load each block C[i,j] into cache, once : $n^2$

        // $b = n/N =$ cache line size
        for k = 0 to N-1
            // load each block $A[i,k]$ and $B[k,j]$ $N$ times
            // $= 2(N \times N^2) \times (n/N)^2 = 2Nn^2$
            $C[i,j] += A[i,k] \times B[k,j]$ // do the matrix multiply

        // write each block $C[i,j]$ once :
        // $n^2$

Total: $(2N+2)n^2$
Flops to memory ratio

Let \( q = \frac{\text{# flops}}{\text{main memory reference}} \approx \frac{n}{N} = b \) as \( n \to \infty \)

\[
q = \frac{2n^3}{(2N + 2)n^2} = \frac{n}{N + 1}
\]
The results

<table>
<thead>
<tr>
<th>N,B</th>
<th>Unblocked Time</th>
<th>Blocked Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>256, 64</td>
<td>0.6</td>
<td>0.002</td>
</tr>
<tr>
<td>512,128</td>
<td>15</td>
<td>0.24</td>
</tr>
</tbody>
</table>

Amortize memory accesses by increasing memory reuse
More on blocked algorithms

• Data in the sub-blocks are contiguous within rows only
• We may incur conflict cache misses
• Idea: since re-use is so high… let’s copy the subblocks into contiguous memory before passing to our matrix multiply routine

“The Cache Performance and Optimizations of Blocked Algorithms,”
M. Lam et al., *ASPLOS IV*, 1991

http://www-suif.stanford.edu/papers/lam91.ps
Gaussian Elimination
Linear systems of equations

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

\[
\begin{align*}
(1) \quad 2x + 3y &= 8 \\
(2) \quad 3x + 2y &= 7
\end{align*}
\]

• 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: \( A\mathbf{x} = \mathbf{b} \)

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

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

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

- We multiplied row 1 of \( A \) by 3/2 and subtracting the scaled version from row 2 of \( A \) and \( \mathbf{b} \)
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:

\[ A = \begin{pmatrix} 2 & 3 \\ 3 & 2 \end{pmatrix} \]

\[ A' = \begin{pmatrix} 2 & 3 \\ 0 & -2.5 \end{pmatrix} \]

• Similarly for \( b \)
Gaussian Elimination

• The process of eliminating the non-zero values under the main diagonal is called \textit{Gaussian Elimination}, named after the mathematician \textit{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 \textit{upper triangular matrix} $U$
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 $O(n^2)$
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

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

\[
[4 \ 3 \ 4 \ 8] - 4*[1 \ 1 \ 1 \ 3] = [0 \ -1 \ 0 \ -4]
\]
3 \times 3 example

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

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

\[
\begin{bmatrix}
9 & 3 & 4 & 7 \\
1 & 1 & 1 & 3 \\
0 & -1 & 0 & -4 \\
0 & -6 & -5 & -20
\end{bmatrix}
\]

\[
\begin{bmatrix}
9 & 3 & 4 & 7 \\
1 & 1 & 1 & 3 \\
0 & -1 & 0 & -4 \\
0 & -6 & -5 & -20
\end{bmatrix}
\]
3 × 3 example

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

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

\[
\begin{bmatrix}
0 & -6 & -5 & -20
\end{bmatrix}
+ -6\begin{bmatrix}
0 & -1 & 0 & -4
\end{bmatrix}
= \begin{bmatrix}
0 & 0 & -5 & 4
\end{bmatrix}
\]
Visualizing the algorithm

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

\[
\text{for } k = 0 \text{ to } n-1 \quad // \text{For each column } k \\
\text{for } i = k+1 \text{ to } n-1 \quad // \text{Eliminate entries below the diagonal:} \\
\quad // \text{subtract a multiple of row } k \\
\quad // \text{from succeeding rows } i \\
A[i,k+1:n] = (A[i,k] / A[k,k]) \times A[k, k+1:n]
\]

```
Column 0
0 . . .
. . . .
. . . .
0 . . .

Column 1
0 . . .
0 . . .
0 . . .
0 . . .

Column 2
0 . . .
0 . . .
0 . . .
0 . . .

Column n-1
0 . . .
0 . . .
0 . . .
0 . . .
```
Eliminating the entries below the diagonal

- For each column \( k \) in 0 to \( n-1 \)
- … subtract various 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 \( A[k, k] \times m_k - A[j, k] = 0 \)
- We only update to the right of and below \( A[k, k] \)

for \( i = k+1 \) to \( n-1 \)

\[
A[i, k+1:n] - m_{ik} \times A[k, k+1:n] = 0
\]
Putting it all together

for k = 0 to n-1  // For each column k
    for i = k+1 to n-1  // for each row i > k
        A[i, k+1:n] -= m[i] * A[k,k+1:n]  // Scale row k by m_{ik}
        // and subtract from row i
    end for
end for
Problems with roundoff

• The rank-1 update step uses division …

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

• 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*
- 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

\[
x = \begin{bmatrix} 1 \\ 1 \end{bmatrix}
\]
Stability problems due to roundoff

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

\[
L = \begin{bmatrix}
10^{-4} & 1 & 1 \\
0 & 1 - 10^4 & 2 - 10^4
\end{bmatrix}
\]

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

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

• Thus

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

• We now back substitute to solve for \( x_2 \) and then \( x_1 \)

\[-10^4 x_2 = -10^4 \implies x_2 = 1 \]

• Substituting the value of \( x_2 \) into the first equation

\[10^{-4} x_1 + 1 \times x_2 = 1 \implies 10^{-4} x_1 = 0 \implies x_1 = 0\]

• **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 $A = PLU$, 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

- We’ll use 1D vertical strip partitioning
- Each thread owns $N/P$ columns
- Consider the case where $p=N=6$
- The ■ represents 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 $A[k, k+1: n]$

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

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

  for $i = k+1$ to $n-1$ // for each row $i > k$

    // Scale row $k$ by $m_{ik}$ and
    // subtract from row $i$

    $A[i, k+1: n] = m[i] * A[k, k+1:n]$
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
• 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
2D Block CYCLIC decomposition

- A cyclic mapping tessellates a template of the processes over the entire domain
- Improved locality as the block size increases
- What do we give up in exchange?
Distributed GE with a 2D Block Cyclic Layout

Choose pivot, compute multipliers, make swap decision

Coordinate swap decision and multipliers so each block column can participate

Carry out the elimination
Fin