Parallel Algorithms II

• Topics: matrix and graph algorithms
Solving Systems of Equations

• Given an $N \times N$ lower triangular matrix $A$ and an $N$-vector $b$, solve for $x$, where $Ax = b$ (assume solution exists)

\[
a_{11}x_1 = b_1 \\
a_{21}x_1 + a_{22}x_2 = b_2 , \text{ and so on…}
\]

Define $t_1 \equiv b_1$, $t_i \equiv b_i - \sum_{j=1}^{i-1} a_{ij}x_j, 2 \leq i \leq N$. Then $x_i = t_i / a_{ii}$. 
Equation Solver

Define $t_1 = b_1$, $t_i = b_i - \sum_{j=1}^{i-1} a_{ij}x_j$, $2 \leq i \leq N$. Then $x_i = t_i / a_{ii}$.

$\begin{align*}
x_4x_3x_2x_1 & \quad \text{after 3 steps} \\
* & \quad b_1 & \quad * & \quad b_2 & \quad * & \quad b_3 & \quad * & \quad b_4
\end{align*}$

$\begin{align*}
a_{11} & \quad * \\
a_{22} & \quad * & \quad a_{21} & \quad * \\
a_{33} & \quad * & \quad a_{32} & \quad a_{31} & \quad * \\
a_{44} & \quad * & \quad a_{43} & \quad a_{42} & \quad a_{41} & \quad *
\end{align*}$
Equation Solver Example

• When an $x$, $b$, and $a$ meet at a cell, $ax$ is subtracted from $b$
• When $b$ and $a$ meet at cell 1, $b$ is divided by $a$ to become $x$

\[ b_2' = b_2 - a_{21}x_1 \]

\[ b_3' = b_3 - a_{31}x_1 \]

\[ b_4' = b_4 - a_{41}x_1 \]

\[ b_3'' = b_3' - a_{32}x_2 \]
Complexity

- Time steps = $2N - 1$

- Speedup = $O(N)$, efficiency = $O(1)$

- Note that half the processors are idle every time step – can improve efficiency by solving two interleaved equation systems simultaneously
Inverting Triangular Matrices

• Finding $X$, such that $AX = I$, where $A$ is a lower triangular matrix

• For each row $j$, $A x_j = e_j$, where $e_j$ is the $j$th unit vector $(0, \ldots, 0, 1, 0, \ldots, 0)$ and $x_j$ is the $j$th row of matrix $X$

• Simple extension of the earlier algorithm – it can be applied to compute each row individually
Inverting Triangular Matrices
Solving Tridiagonal Matrices

Tridiagonal matrix: for all $i,j$, the $(i,j)$-th entry is 0 if $|i-j| > 1$

$$A = \begin{pmatrix}
  d_1 & u_1 & & & \\
  l_2 & d_2 & u_2 & & \\
    & \ddots & \ddots & \ddots & \\
  0 & l_{N-1} & d_{N-1} & u_{N-1} & \\
  & & 0 & l_N & d_N
\end{pmatrix}$$

Solve $Ax = b$ for a vector $b$.

• Can be solved recursively with odd-even reduction
Odd-Even Reduction

• For each odd $i$, the corresponding equation $E_i$ is represented as:
  $$x_i = \frac{1}{d_i}(b_i - l_i x_{i-1} - u_i x_{i+1}).$$

• This equation is substituted in equations $E_{i-1}$ and $E_{i+1}$

• Therefore, equation $E_{i-1}$ now has the following unknowns: $x_{i-1}, x_{i+1}, x_{i-3}$, (note that $i$ is odd)

• We now have $N/2$ equations involving only even unknowns – repeat this process until there is only 1 equation with 1 unknown – after computing this unknown, back-substitute to get other unknowns
X-Tree Implementation
The Algorithm

• The $i^{th}$ leaf receives the inputs $u_i$, $d_i$, $l_i$, and $b_i$

• Each leaf sends its values to both neighboring processors (purple sideways arrows) and every even leaf computes the $u$, $d$, $l$, and $b$ values for the second level of equations

• These values are sent to the next higher level (upward purple arrows)

• After the root computes the value of $x_N$, it is propagated down and to the sides until all $x_i$ are computed (green arrows)
Gaussian Elimination

• Solving for $x$, where $Ax=b$ and $A$ is a nonsingular matrix

• Note that $A^{-1}Ax = A^{-1}b = x$; keep applying transformations to $A$ such that $A$ becomes $I$; the same transformations applied to $b$ will result in the solution for $x$

• Sequential algorithm steps:
  - Pick a row where the first ($i^{th}$) element is non-zero and normalize the row so that the first ($i^{th}$) element is 1
  - Subtract a multiple of this row from all other rows so that their first ($i^{th}$) element is zero
  - Repeat for all $i$
## Sequential Example

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>-7</td>
<td>x1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>-10</td>
<td>x2</td>
<td>= 4</td>
</tr>
<tr>
<td>-1</td>
<td>3</td>
<td>-4</td>
<td>x3</td>
<td>6</td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>-7/2</td>
<td>x1</td>
<td>3/2</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>-10</td>
<td>x2</td>
<td>= 4</td>
</tr>
<tr>
<td>-1</td>
<td>3</td>
<td>-4</td>
<td>x3</td>
<td>6</td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>-1/2</td>
<td>x1</td>
<td>-3/2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-3/2</td>
<td>x2</td>
<td>= 3/2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>x3</td>
<td>-1</td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>x1</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>x2</td>
<td>= 0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>x3</td>
<td>-1</td>
</tr>
</tbody>
</table>
Algorithm Implementation

- The matrix is input in staggered form
- The first cell discards inputs until it finds a non-zero element (the pivot row)

- The inverse $\rho$ of the non-zero element is now sent rightward
- $\rho$ arrives at each cell at the same time as the corresponding element of the pivot row
Algorithm Implementation

- Each cell stores $\delta_i = \rho a_{k,1}$ – the value for the normalized pivot row
- This value is used when subtracting a multiple of the pivot row from other rows
- What is the multiple? It is $a_{j,1}$
- How does each cell receive $a_{j,1}$? It is passed rightward by the first cell
- Each cell now outputs the new values for each row
- The first cell only outputs zeroes and these outputs are no longer needed
Algorithm Implementation

- The outputs of all but the first cell must now go through the remaining algorithm steps
- A triangular matrix of processors efficiently implements the flow of data
- Number of time steps?
- Can be extended to compute the inverse of a matrix
Graph Algorithms

\[ G = (V, E) : \text{a directed graph, } V = \{1, \ldots, N\} \]

The adjacency matrix \( A = (a_{ij}) \) of \( G \) is

\[
    a_{ij} = \begin{cases} 
        1 & \text{if either } (i, j) \in E \text{ or } i = j, \\
        0 & \text{otherwise.}
    \end{cases}
\]

The transitive closure of \( G \) is \( G^* = (V, E^*) \),

\[ E^* = \{ (i, j) \mid j \text{ is reachable from } i \text{ in } G \}. \]
Floyd Warshall Algorithm

\[ A^{(k)} = \text{def} \ (a^{(k)}_{ij}), \text{ where for each } k, 0 \leq k \leq N, \ a^{(k)}_{ij} = 1 \text{ if } j \text{ is reachable from } i \text{ passing through only nodes } \leq k \text{ and } 0 \text{ otherwise.} \]

Then \[ A^{(N)} = A^*, \ A^{(0)} = A, \text{ and for all } k \geq 1, \]
\[ a^{(k)}_{ij} = a^{(k-1)}_{ij} \lor \left( a^{(k-1)}_{ik} \land a^{(k-1)}_{kj} \right). \]
Implementation on 2d Processor Array
Algorithm Implementation

- Diagonal elements of the processor array can broadcast to the entire row in one time step (if this assumption is not made, inputs will have to be staggered)

- A row sifts down until it finds an empty row – it sifts down again after all other rows have passed over it

- When a row passes over the 1st row, the value of $a_{i1}$ is broadcast to the entire row – $a_{ij}$ is set to 1 if $a_{i1} = a_{1j} = 1$ – in other words, the row is now the $i^{th}$ row of $A^{(1)}$

- By the time the $k^{th}$ row finds its empty slot, it has already become the $k^{th}$ row of $A^{(k-1)}$
• When the i\textsuperscript{th} row starts moving again, it travels over rows \(a_k\) \((k > i)\) and gets updated depending on whether there is a path from \(i\) to \(j\) via vertices \(< k\) (and including \(k\))
Title

• Bullet