Parallel Algorithms II

- Topics: matrix and graph algorithms

Solving Systems of Equations

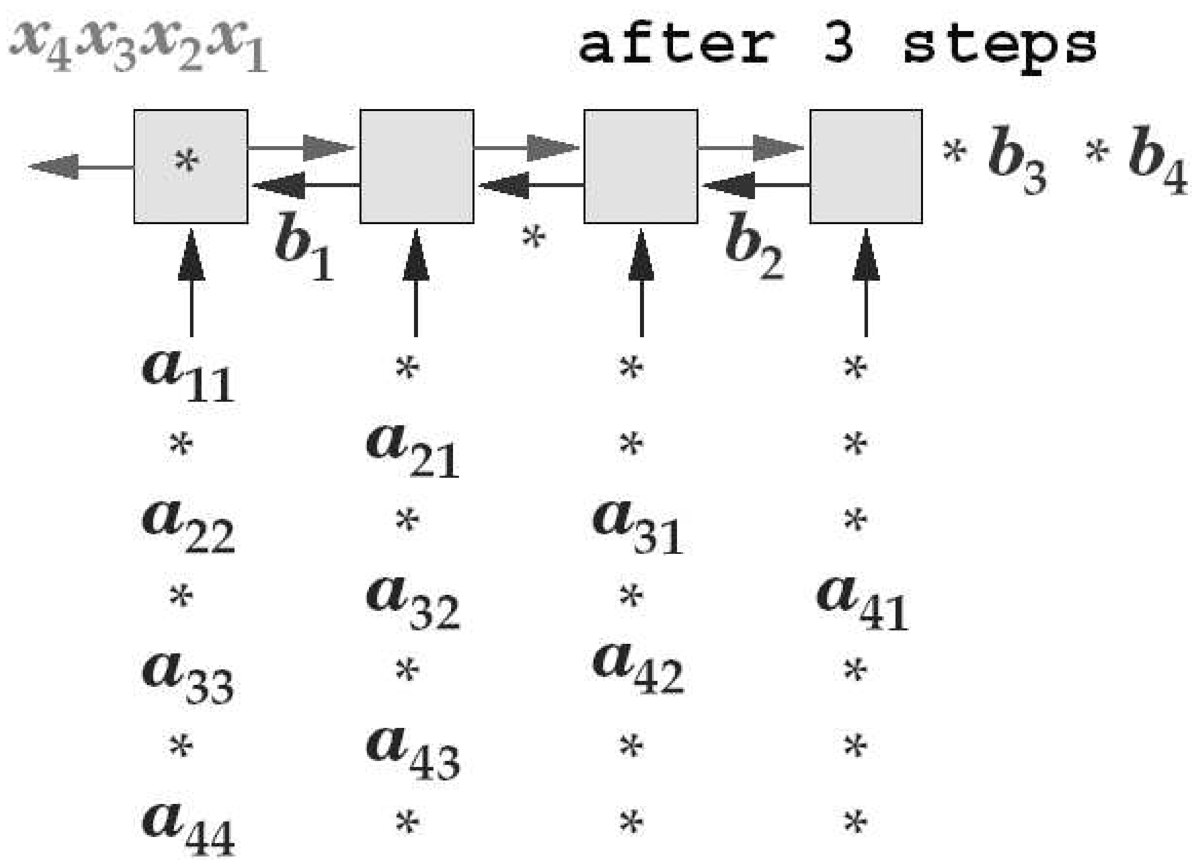
- Given an N x N lower triangular matrix A and an N-vector b, solve for x, where Ax = b (assume solution exists)

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

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

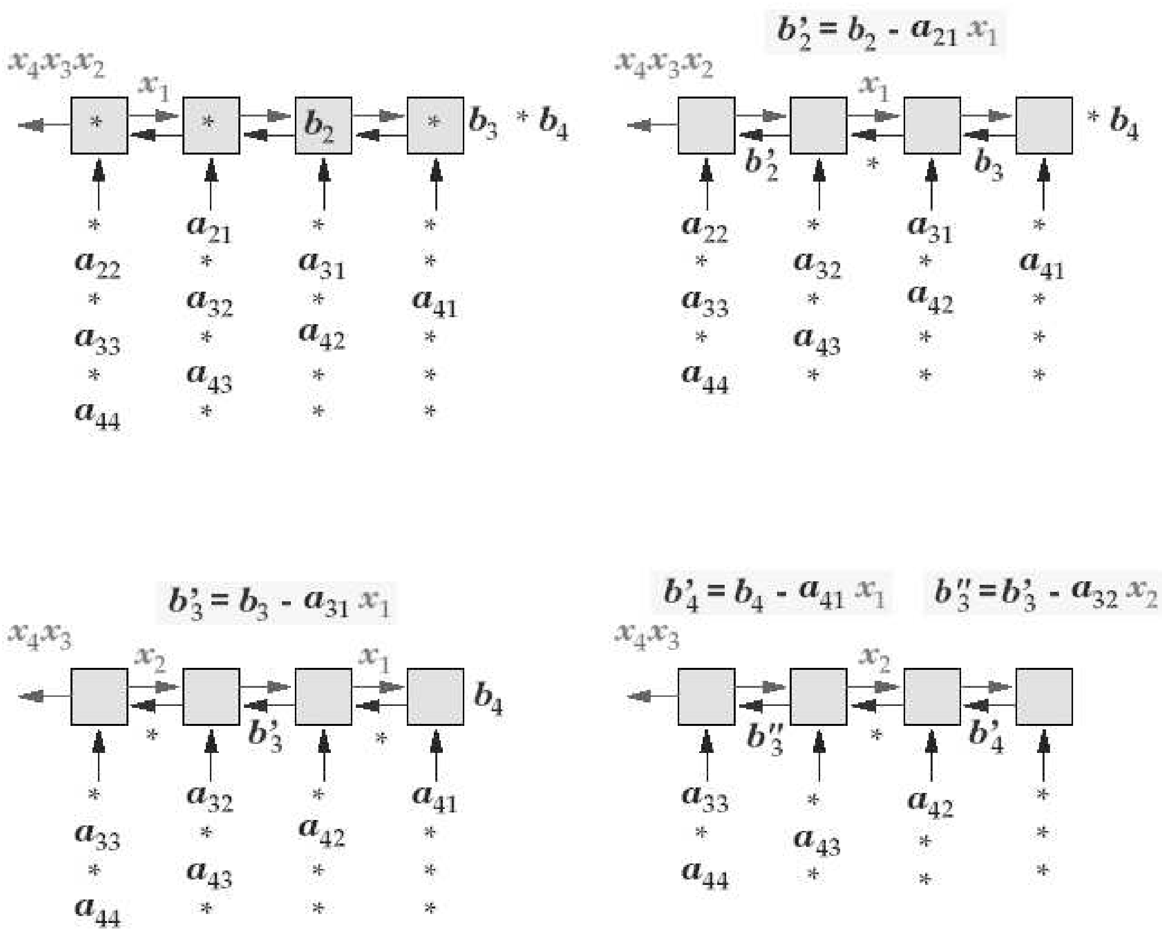
Equation Solver

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



Equation Solver 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*



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

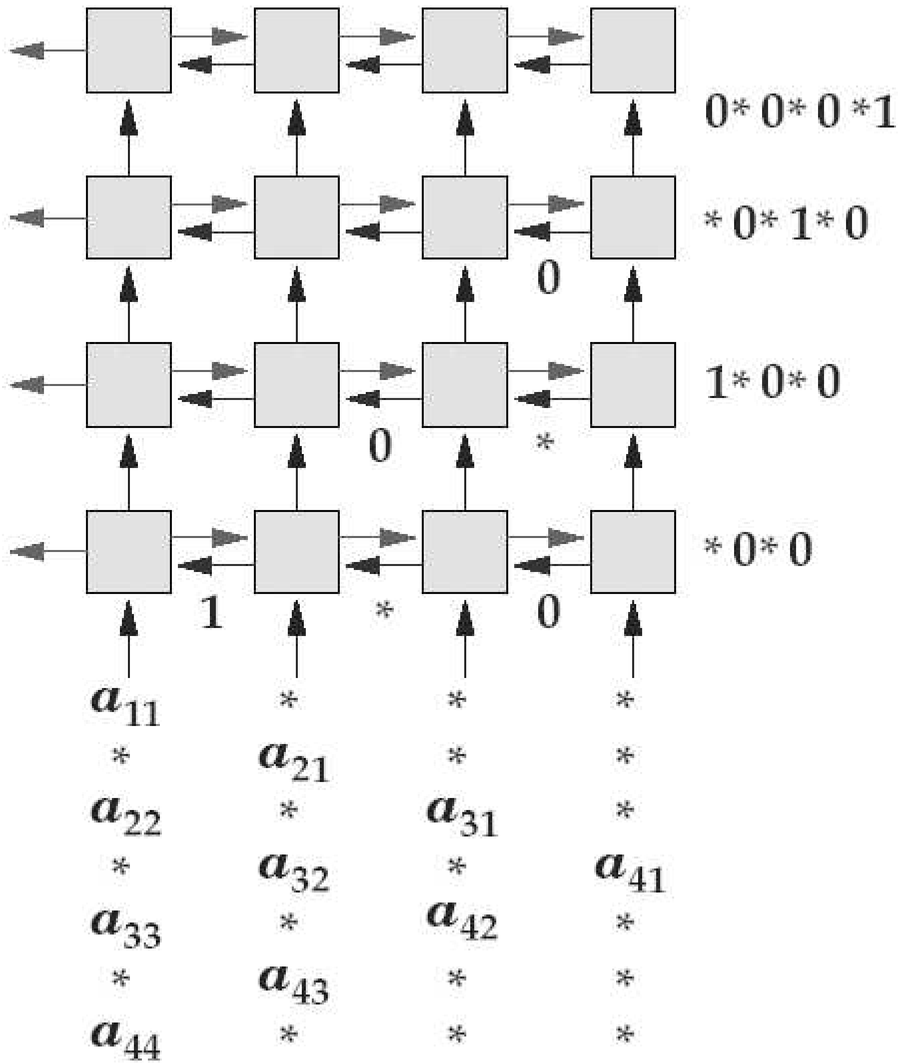
Inverting Triangular Matrices

- 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 jth unit vector (0,…, 0, 1, 0,…, 0) and x_j is the jth 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 $|i - j| > 1$

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

Solve $Ax = b$ for a vector b.

- Can be solved recursively with odd-even reduction

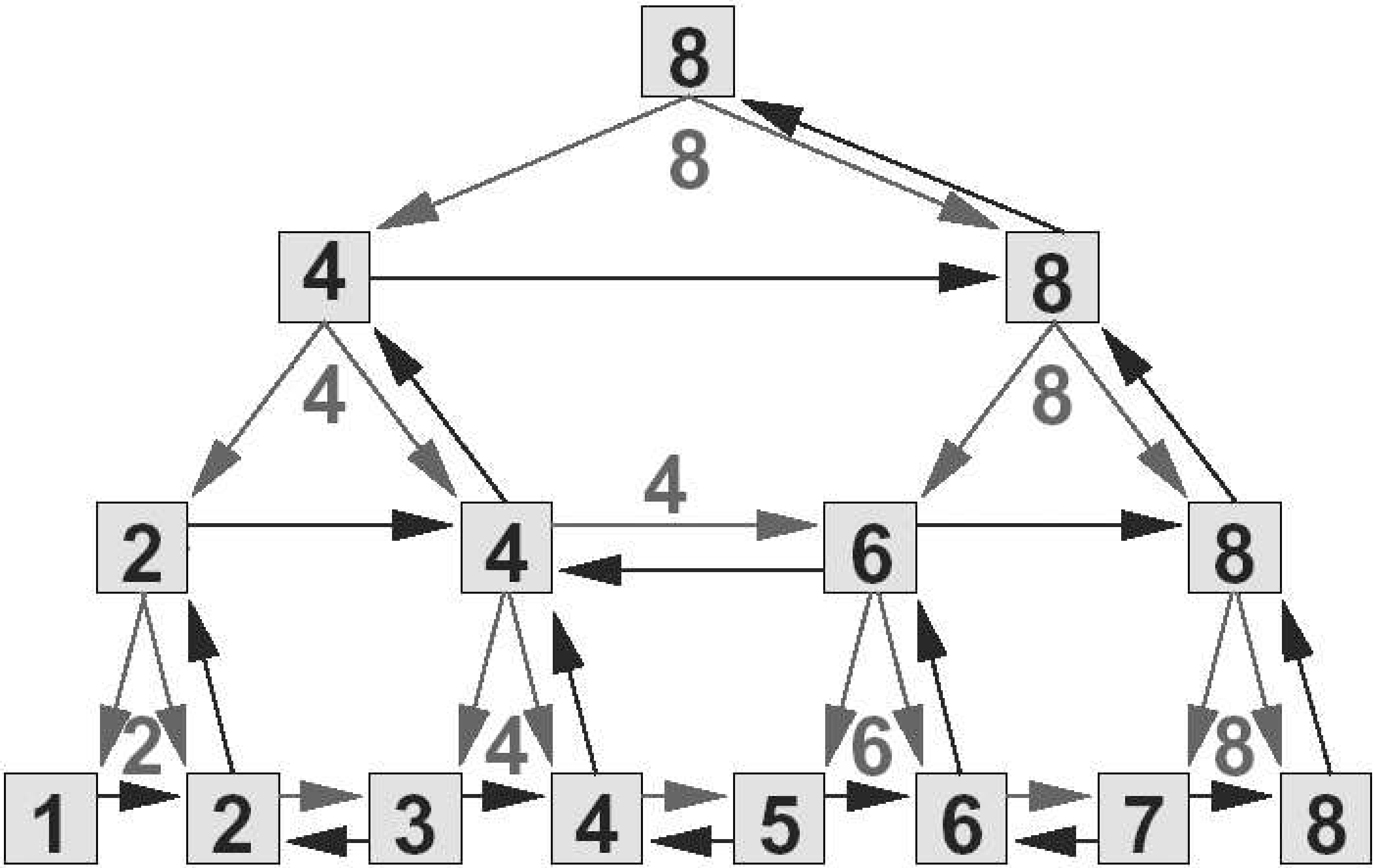
Odd-Even Reduction

- 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

$$\begin{array}{rrrccr} 2 & 4 & -7 & x1 & & 3 \\ 3 & 6 & -10 & x2 & = & 4 \\ -1 & 3 & -4 & x3 & & 6 \end{array}$$

$$\begin{array}{rrrccr} 1 & 2 & -7/2 & x1 & & 3/2 \\ 3 & 6 & -10 & x2 & = & 4 \\ -1 & 3 & -4 & x3 & & 6 \end{array}$$

$$\begin{array}{rrrccr} 1 & 2 & -7/2 & x1 & & 3/2 \\ 0 & 0 & 1/2 & x2 & = & -1/2 \\ -1 & 3 & -4 & x3 & & 6 \end{array}$$

$$\begin{array}{rrrccr} 1 & 2 & -7/2 & x1 & & 3/2 \\ 0 & 0 & 1/2 & x2 & = & -1/2 \\ 0 & 5 & -15/2 & x3 & & 15/2 \end{array}$$

$$\begin{array}{rrrccr} 1 & 2 & -7/2 & x1 & & 3/2 \\ 0 & 5 & -15/2 & x2 & = & 15/2 \\ 0 & 0 & 1/2 & x3 & & -1/2 \end{array}$$

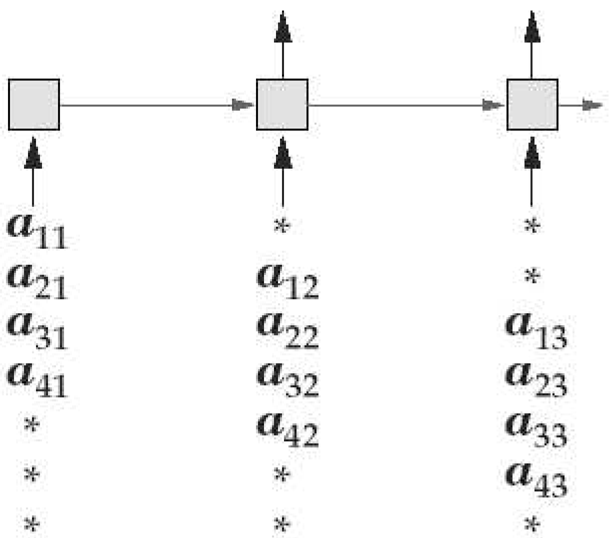
$$\begin{array}{rrrccr} 1 & 2 & -7/2 & x1 & & 3/2 \\ 0 & 1 & -3/2 & x2 & = & 3/2 \\ 0 & 0 & 1/2 & x3 & & -1/2 \end{array}$$

$$\begin{array}{rrrccr} 1 & 0 & -1/2 & x1 & & -3/2 \\ 0 & 1 & -3/2 & x2 & = & 3/2 \\ 0 & 0 & 1/2 & x3 & & -1/2 \end{array}$$

$$\begin{array}{rrrccr} 1 & 0 & -1/2 & x1 & & -3/2 \\ 0 & 1 & -3/2 & x2 & = & 3/2 \\ 0 & 0 & 1 & x3 & & -1 \end{array}$$

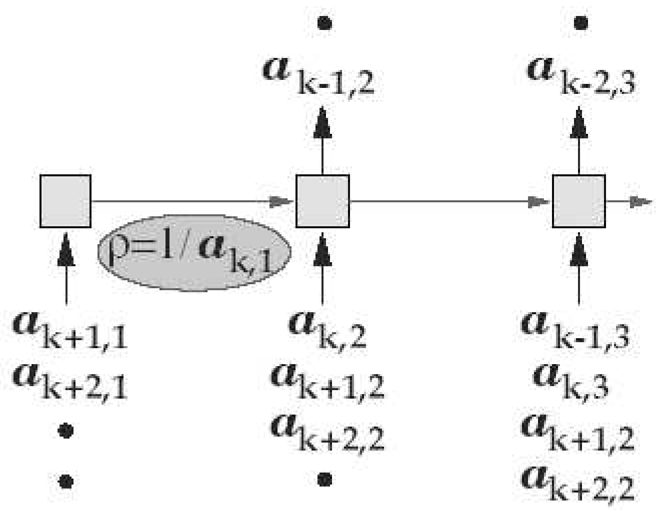
$$\begin{array}{rrrccr} 1 & 0 & 0 & x1 & & -2 \\ 0 & 1 & 0 & x2 & = & 0 \\ 0 & 0 & 1 & x3 & & -1 \end{array}$$

Algorithm Implementation

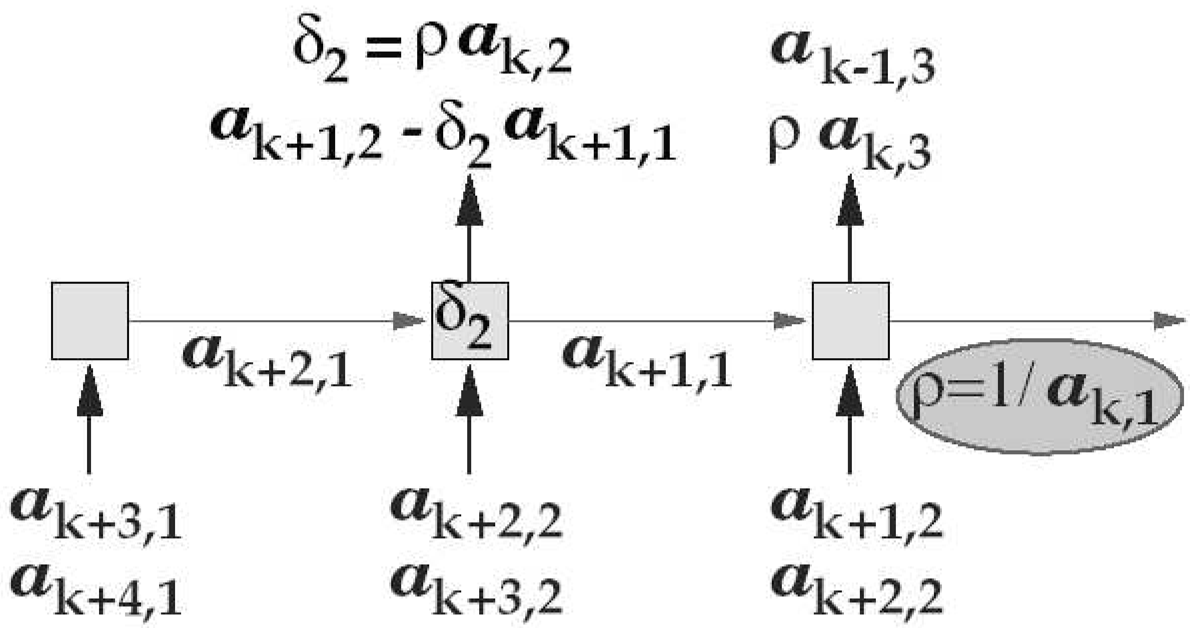


- The matrix is input in staggered form
- The first cell discards inputs until it finds a non-zero element (the pivot row)

- The inverse ρ of the non-zero element is now sent rightward
- ρ arrives at each cell at the same time as the corresponding element of the pivot row



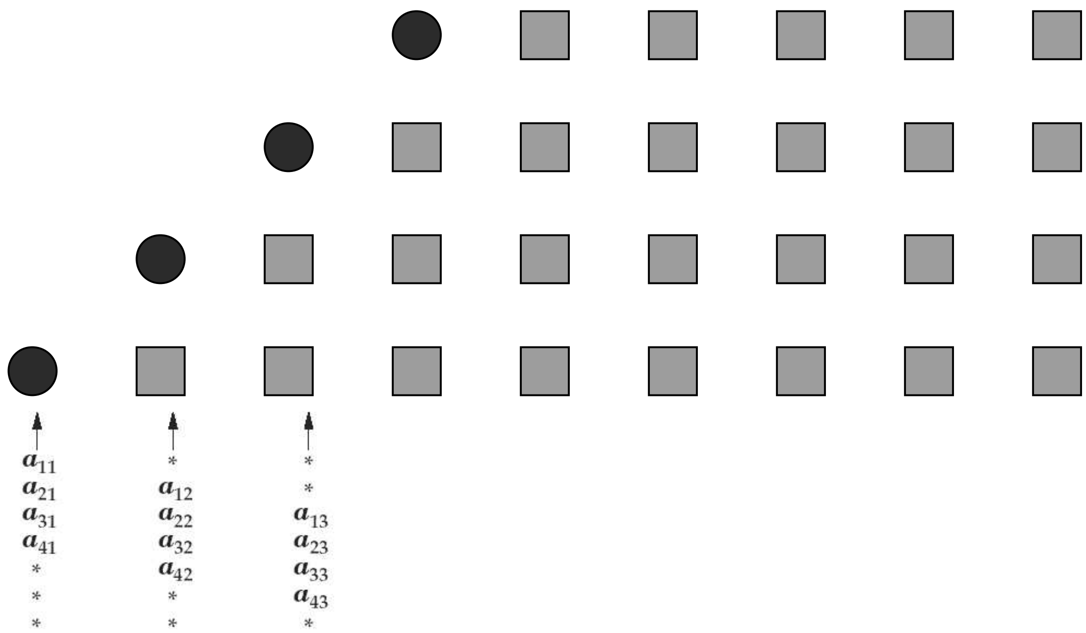
Algorithm Implementation



- Each cell stores $\delta_i = \rho\ a_{k,l}$ – 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



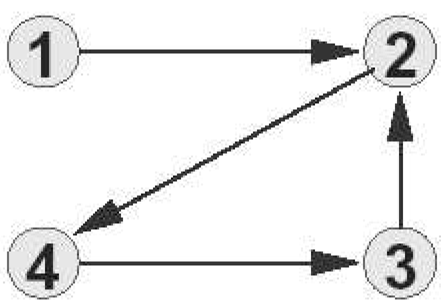
Graph Algorithms

$G = (V, E)$: 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 $G^* = (V, E^*)$,

$$E^* = \{(i,j) \mid j \text{ is reachable from } i \text{ in } G\}.$$



$$A = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \qquad A^* = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

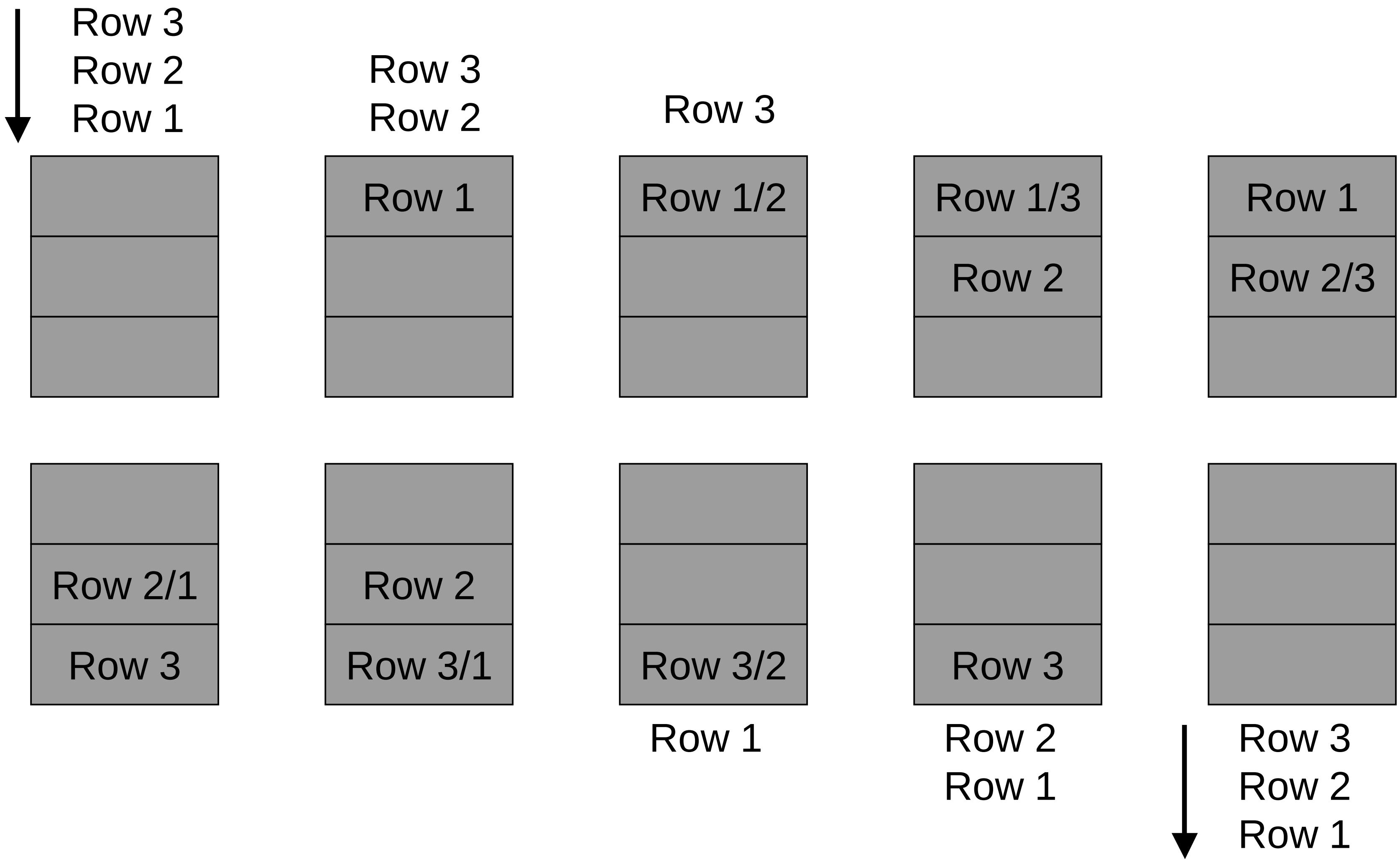
Floyd Warshall Algorithm

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

Then $A^{(N)} = A^*$, $A^{(0)} = A$, and for all $k \geq 1$,

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} \vee \left(a_{ik}^{(k-1)} \wedge a_{kj}^{(k-1)} \right).$$

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 $A^{(k-1)}$

Algorithm Implementation

- When the i^{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