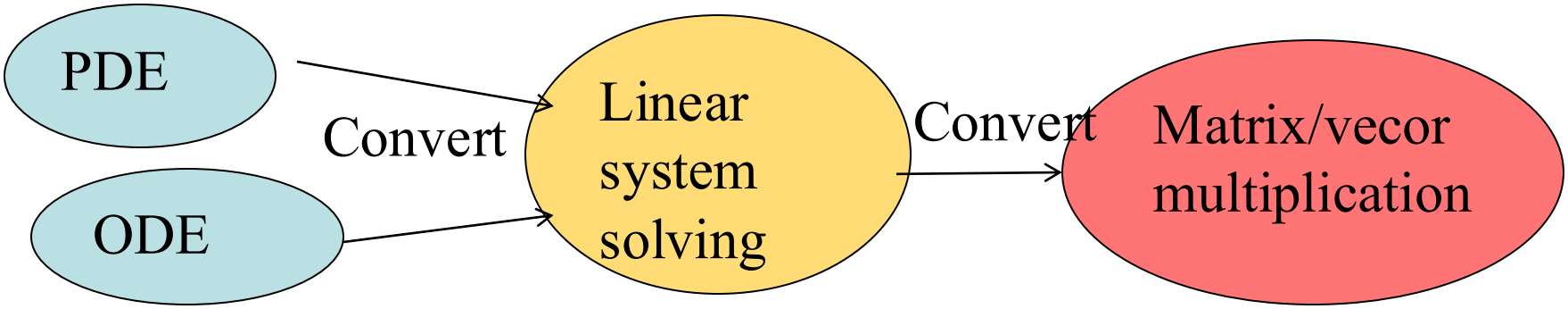
Parallel Scientific Computing Algorithms

CS140 Tao Yang, UCSB

Scientific Computing Algorithms

- **Basic operations:**
  - Vector-vector multiplication
  - Matrix vector multiplication
  - Matrix-matrix multiplication
- **Solving linear systems of equations**
- **Solving non-linear systems**
  - Finite-difference methods for solving ordinary differential equations (ODEs)
  - Finite-difference for partial differential equations PDEs



Solving Linear System of Equations

Direct methods: Gaussian Elimination

- **Step 1: Forward elimination**

$$
\begin{array}{lrcll}
(1) & 4x_1 - 9x_2 + 2x_3 & = & 2 & \\
(2) & 2x_1 - 4x_2 + 4x_3 & = & 3 & \\
(3) & -x_1 + 2x_2 + 2x_3 & = & 1 & \\
(2)\text{-}(1)*\frac{2}{4} & 0.5x_2 + 3x_3 & = & 2 & (4) \\
(3)\text{-}(1)*\text{-}\frac{1}{4} & -\frac{1}{4}x_2 + \frac{5}{2}x_3 & = & \frac{3}{2} & (5) \\
(5)\text{-}(4)*\text{-}\frac{1}{2} & 4x_3 & = & \frac{5}{2} &
\end{array}
$$

$$
\begin{aligned}
4x_1 - 9x_2 + 2x_3 &= 2 \\
\tfrac{1}{2}x_2 + 3x_3 &= 2 \\
4x_3 &= \tfrac{5}{2}
\end{aligned}
$$

Solving Linear System of Equations: GE

Step 2: Backward substitution

$$
\begin{aligned}
4x_1 - 9x_2 + 2x_3 &= 2 \\
\tfrac{1}{2}x_2 + 3x_3 &= 2 \\
4x_3 &= \tfrac{5}{2}
\end{aligned}
$$

$$
\begin{aligned}
x_3 &= \frac{5}{8} \\
x_2 &= \frac{2-3x_3}{\frac{1}{2}} = \frac{1}{4} \\
x_1 &= \frac{2+9x_2-2x_3}{4} = \frac{3}{4}
\end{aligned}
$$

Gaussian Elimination in a Matrix Form

- Use an augmented matrix to express elimination process for solving Ax = b in a form of (A | b)

$$
\begin{aligned}
(1) \quad & 4x_1 - 9x_2 + 2x_3 &&= 2 \\
(2) \quad & 2x_1 - 4x_2 + 4x_3 &&= 3 \\
(3) \quad & -x_1 + 2x_2 + 2x_3 &&= 1
\end{aligned}
$$

$$
(A|b) = \begin{pmatrix} 4 & -9 & 2 & 2 \\ 2 & -4 & 4 & 3 \\ -1 & 2 & 2 & 1 \end{pmatrix}
\xrightarrow[(3)=(3)-(1)*\frac{-1}{4}]{(2)=(2)-(1)*\frac{2}{4}}
\begin{pmatrix} 4 & -9 & 2 & 2 \\ 0 & \frac{1}{2} & 3 & 2 \\ 0 & \frac{-1}{4} & \frac{5}{2} & \frac{3}{2} \end{pmatrix}
$$

$$
\Rightarrow \begin{pmatrix} 4 & -9 & 2 & 2 \\ 0 & 1/2 & 3 & 2 \\ 0 & 0 & 4 & 5/2 \end{pmatrix}
$$

Gaussian Elimination Algorithm

- Forward elimination

Use Row k to modify rows k+1, k+2, …, n

For $k = 1$ **to** $n - 1$

  For $i = k + 1$ **to** n

    $a_{ik} = a_{ik}/a_{kk};$

    For $j = k + 1$ to $n + 1$

      $a_{ij} = a_{ij} - a_{ik} * a_{kj};$

    endfor

  endfor

endfor

$$\begin{pmatrix} 4 & -9 & 2 & 2 \\ 2 & -4 & 4 & 3 \\ -1 & 2 & 2 & 1 \end{pmatrix} \xrightarrow[(3)=(3)-(1)*\frac{-1}{4}]{(2)=(2)-(1)*\frac{2}{4}} \begin{pmatrix} 4 & -9 & 2 & 2 \\ 0 & \frac{1}{2} & 3 & 2 \\ 0 & \frac{-1}{4} & \frac{5}{2} & \frac{3}{2} \end{pmatrix}$$

Gaussian Elimination Algorithm

Step 2: Backward substitution

For $i = n$ **to** 1

  For $j = i + 1$ **to** n

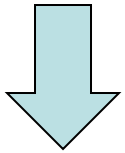
    $x_i = x_i - a_{i,j} * x_j;$

  Endfor

  $x_i = x_i / a_{i,i};$

Endfor

$$
\begin{aligned}
4x_1 - 9x_2 + 2x_3 &= 2 \\
\tfrac{1}{2}x_2 + 3x_3 &= 2 \\
4x_3 &= \tfrac{5}{2}
\end{aligned}
$$



$$
\begin{aligned}
x_3 &= \tfrac{5}{8} \\
x_2 &= \frac{2-3x_3}{\frac{1}{2}} = \tfrac{1}{4} \\
x_1 &= \frac{2+9x_2-2x_3}{4} = \tfrac{3}{4}
\end{aligned}
$$

Complexity of Gaussian Elimination

For $k = 1$ **to** $n-1$
  For $i = k+1$ **to** n
    $a_{ik} = a_{ik}/a_{kk}$;
    For $j = k+1$ to $n+1$
      $a_{ij} = a_{ij} - a_{ik} * a_{kj}$;
    endfor
  endfor
endfor

For $i = n$ **to** 1
  For $j = i+1$ **to** n
    $x_i = x_i - a_{i,j} * x_j$;
  Endfor
  $x_i = x_i/a_{i,i}$;
Endfor

Each division, multiplication, subtraction counts one time unit ω. Ignore loop overhead.

#Operations in forward elimination:

$$\sum_{k=1}^{n-1}\sum_{i=k+1}^{n}\left(1+\sum_{j=k+1}^{n}2+2\right)\omega$$

$$=\sum_{k=1}^{n-1}\sum_{i=k+1}^{n}(2(n-k)+3)\omega \approx 2\omega\sum_{k=1}^{n-1}(n-k)^2 \approx \frac{2n^3}{3}\omega$$

#Operations in backward substitution:

$$\sum_{k=1}^{n}(1+\sum_{i=k+1}^{n}2)\omega \approx 2\omega\sum_{k=1}^{n}(n-k) \approx n^2\omega$$

Total #Operations: $\approx \frac{2n^3}{3}\omega$.

Total space: $\approx n^2$ double-precision numbers.

Partitioning for Parallel Gaussian Elimination

Focus on **forward elimination** which is dominating the cost

Computation partitioning:

For $k = 1$ **to** $n - 1$

  For $i = k + 1$ **to** n

T_k^i : $a_{ik} = a_{ik}/a_{kk}$

  For $j = k + 1$ **to** $n + 1$

    $a_{ij} = a_{ij} - a_{ik} * a_{kj}$

  EndFor

Another option:

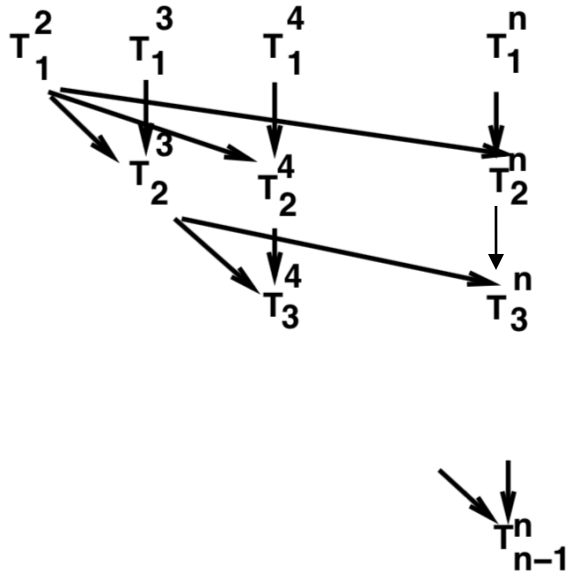
```
For k = 1 to n − 1
    For i = k + 1 to n
        a_ik = a_ik/a_kk;
        For j = k + 1 to n + 1
            a_ij = a_ij − a_ik * a_kj;
        endfor
    endfor
endfor
```

Row-Oriented Parallel Gaussian Elimination

Focus on **forward elimination** which is dominating the cost

Computation partitioning:

Task graph:



For $k = 1$ **to** $n - 1$

For $i = k + 1$ **to** n

T_k^i :

$a_{ik} = a_{ik}/a_{kk}$

For $j = k + 1$ **to** $n + 1$

$a_{ij} = a_{ij} - a_{ik} * a_{kj}$

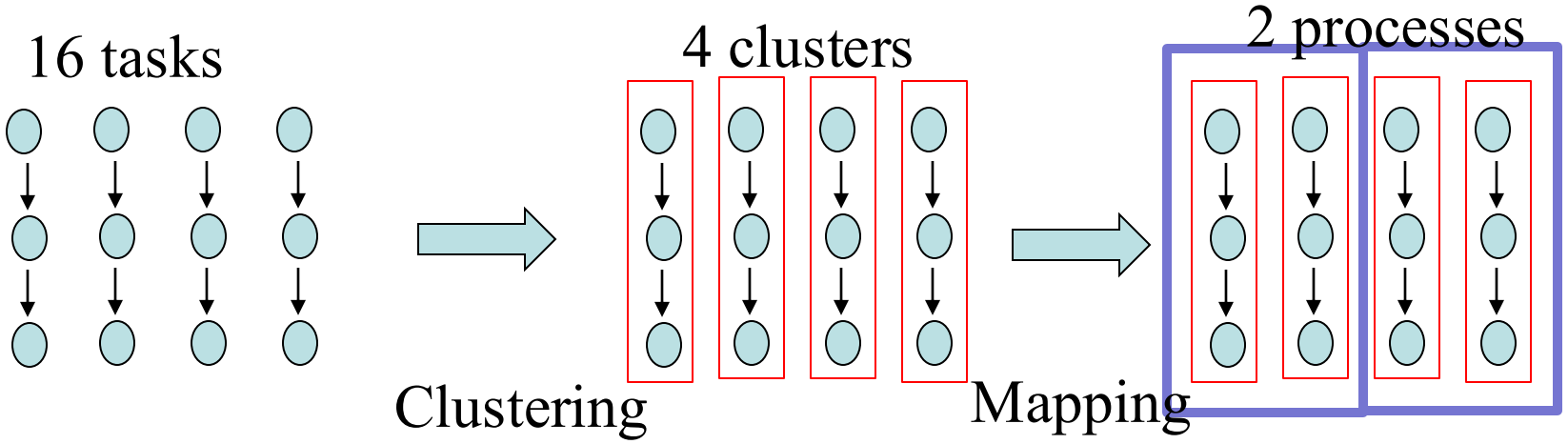
EndFor

T_k^i : Read rows A_k, A_i

Write row A_i

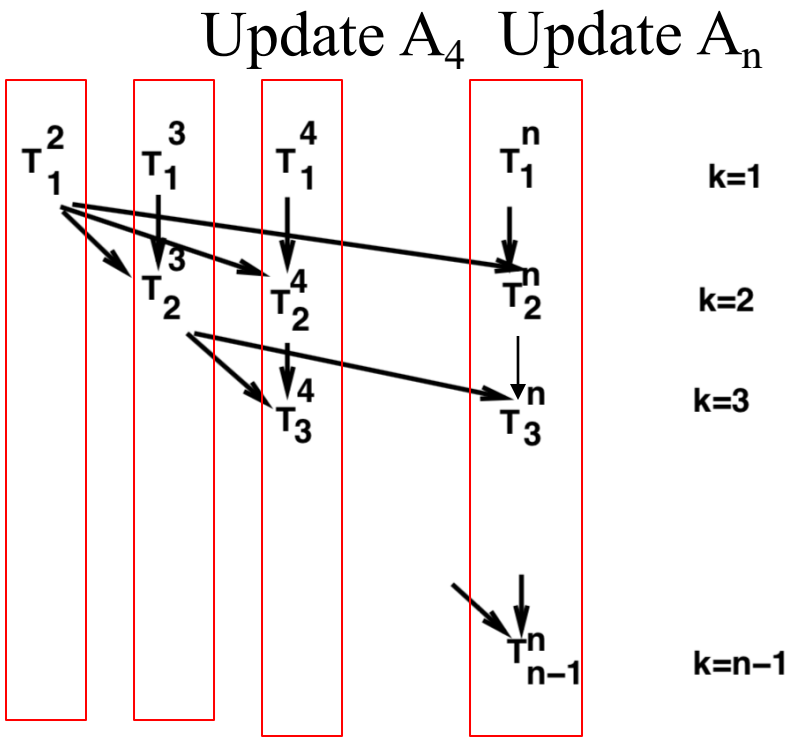
Strategies for Mapping and Scheduling

- **Option 1:** Directly map tasks to p processes (threads)
- **Option 2:**
  - Step 1. Assume there are enough parallelism.
    - Cluster tasks to reduce unnecessary communication/synchronization
    - If needed, assign data ownership (e.g. owner-compute rule)
  - Step 2. Map clusters to p processes (threads)



Map to Parallel Processes, Decide data ownership, and Schedule Tasks

Step 1: map to n-1 clusters while preserving parallelism



- Identify write patterns
- Cluster vertically to reduce communication
- Assign data ownership

  Cluster i owns i-th row

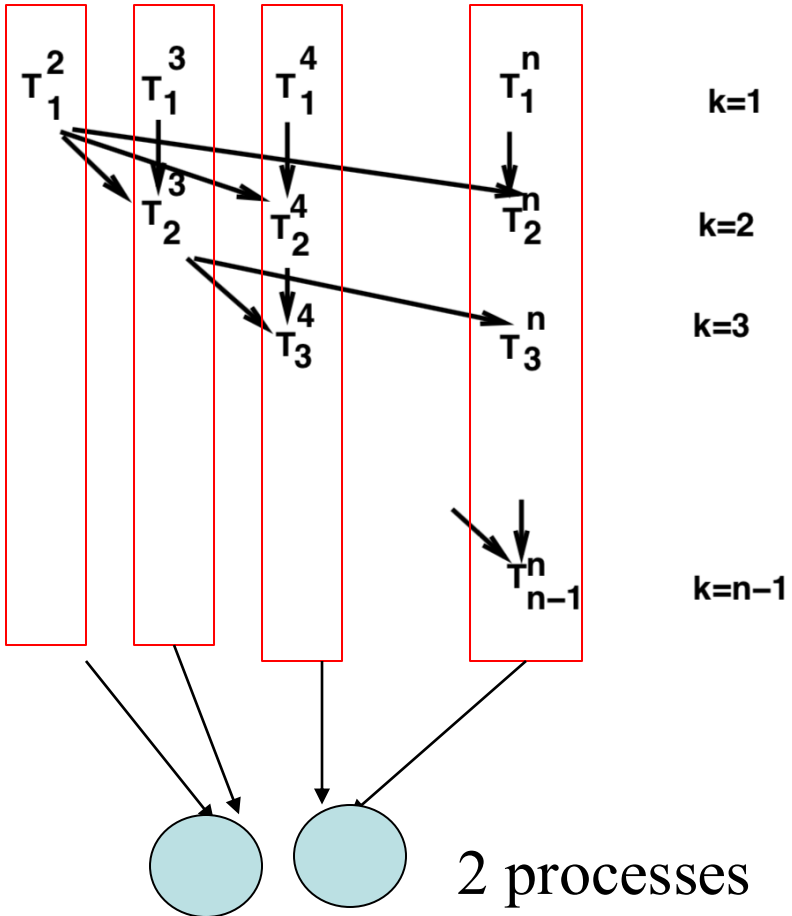
T_k^i : Read rows A_k, A_i

Write row A_i

Map to p Parallel Processes, Decide data ownership, and Schedule Tasks

Step 2: Assign n-1 clusters (virtual processes) to p processes

Mapping options? Cyclic or block mapping



Cyclic for load balancing

Cost of Cluster i increases as i increases.

Parallel Algorithm for Gaussian Elimination

For $k = 1$ **to** $n - 1$

For $i = k + 1$ **to** n

T_k^i : $a_{ik} = a_{ik}/a_{kk}$

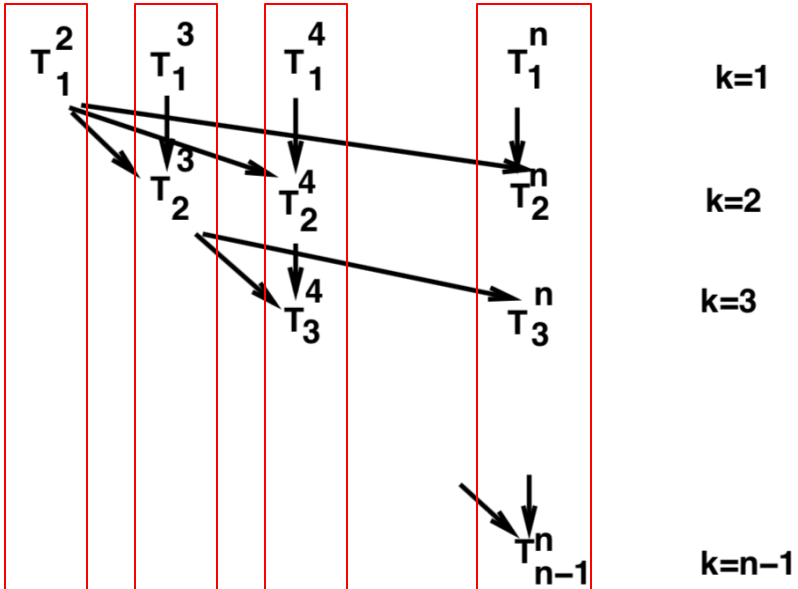
For $j = k + 1$ **to** $n + 1$

$a_{ij} = a_{ij} - a_{ik} * a_{kj}$

EndFor

T_k^i : Read rows A_k, A_i

Write row A_i



Parallelism:

Tasks $T_k^{k+1}\ T_k^{k+2}\ \ldots\ T_k^n$ are independent.

Parallel Algorithm(Basic idea)

For $k = 1$ **to** $n - 1$

Do $T_k^{k+1}\ T_k^{k+2}\ \ldots\ T_k^n$ in parallel on p processors.

Parallel Algorithm:

Proc 0 broadcasts Row 1

For $k = 1$ **to** $n - 1$

Do $T_k^{k+1} \ldots T_k^n$ in parallel

$(T_k^i \rightarrow$ `proc_map(i)`$)$.

Broadcast row $k + 1$.

endfor

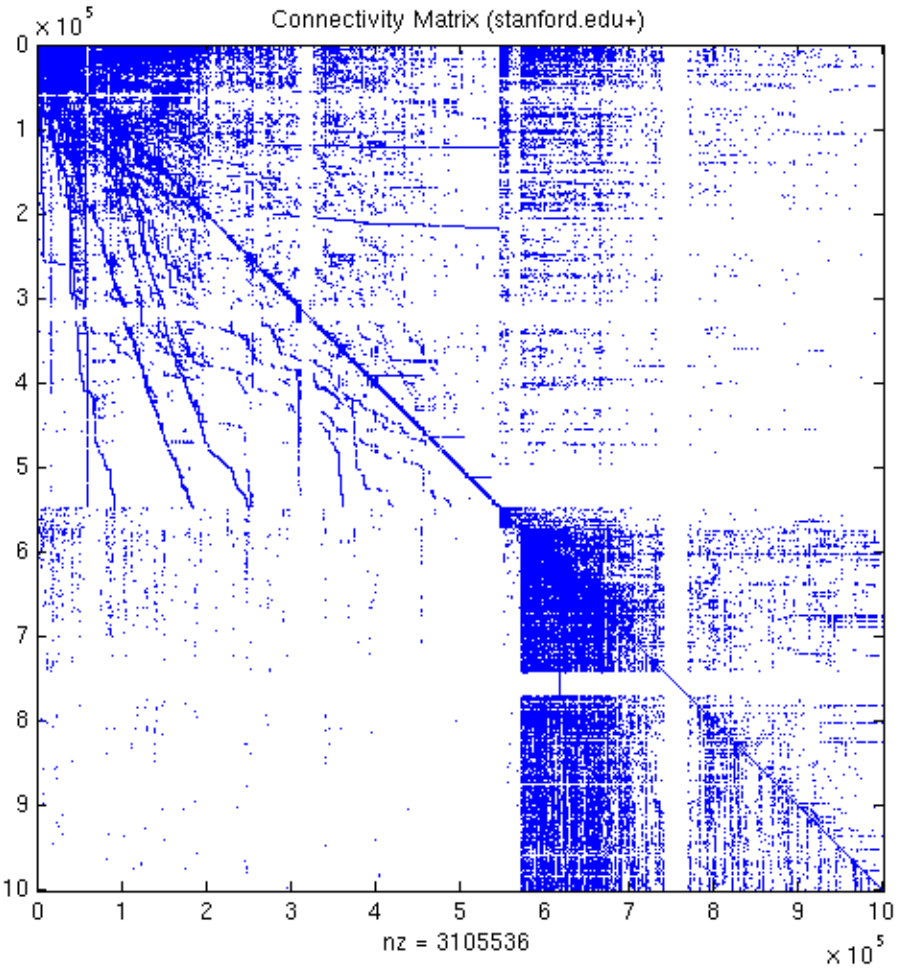
Solving linear system Ax=b in practice

- Large dimension size n
- A is sparse matrix

Assume matrix dimension size n= 1 billion

How long does it take with $O(n^3)$ algorithm?

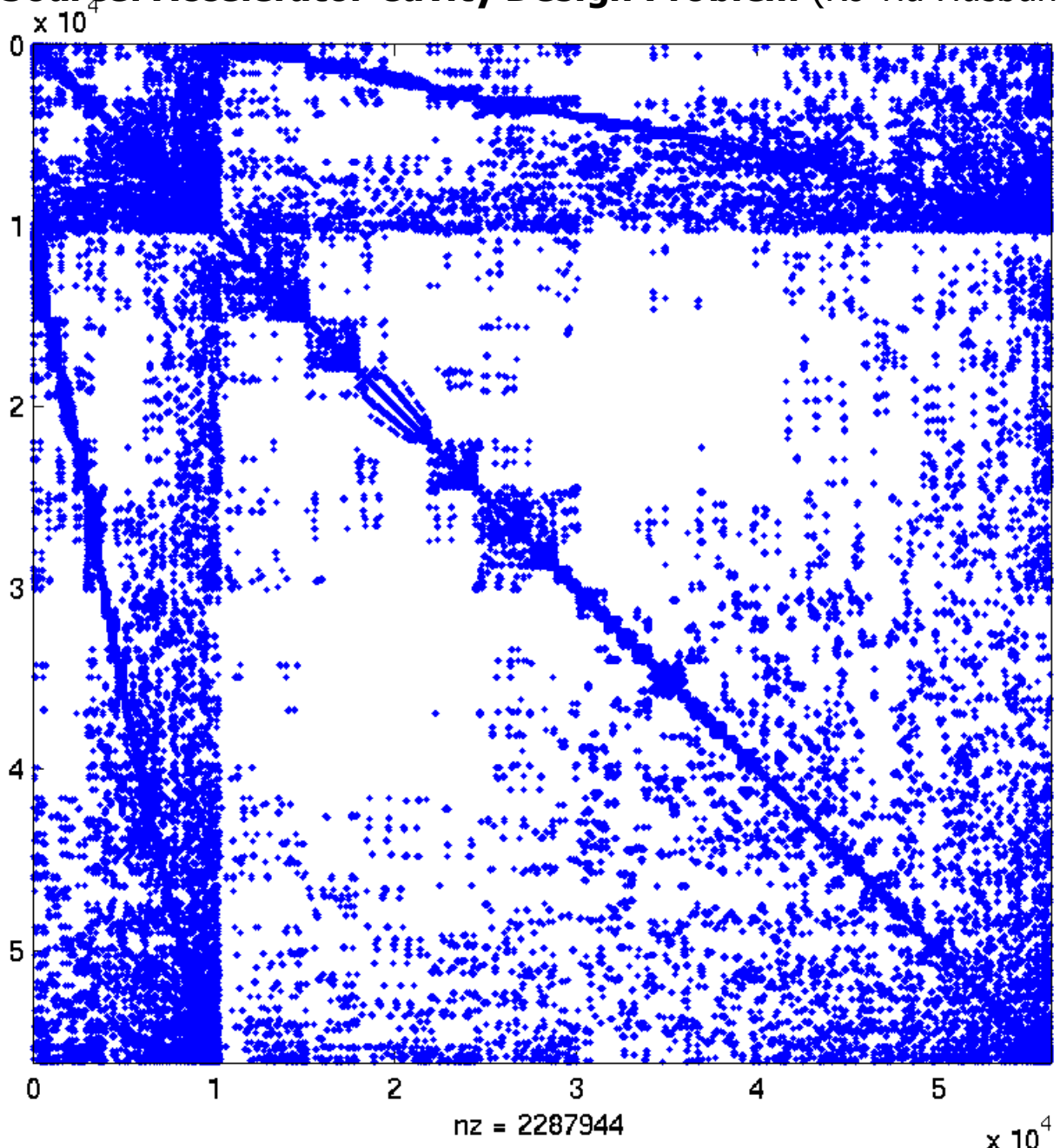
Too expensive



Example of sparse matrix for social network applications

Sparse matrix pattern in Physics/Material

Source: Accelerator Cavity Design Problem (Ko via Husbands)



Iterative Methods for Linear System Solving

- More effective for sparse matrices
- Start from an initial guess of solutions
- Derive an update of solutions using equations

Utilize new solutions as soon as they are available.

$$
\begin{aligned}
(1) \quad & 6x_1 - 2x_2 + x_3 = 11 \\
(2) \quad & -2x_1 + 7x_2 + 2x_3 = 5 \\
(3) \quad & x_1 + 2x_2 - 5x_3 = -1
\end{aligned}
$$

$$
\begin{aligned}
x_1 &= \frac{11}{6} - \frac{1}{6}(-2x_2 + x_3) \\
x_2 &= \frac{5}{7} - \frac{1}{7}(-2x_1 + 2x_3) \\
x_3 &= \frac{1}{5} - \frac{1}{-5}(x_1 + 2x_2)
\end{aligned}
$$

$$
\begin{aligned}
x_1^{(k+1)} &= \frac{1}{6}(11 - (-2x_2^{(k)} + x_3^{(k)})) \\
x_2^{(k+1)} &= \frac{1}{7}(5 - (-2x_1^{(k)} + 2x_3^{(k)})) \\
x_3^{(k+1)} &= \frac{1}{-5}(-1 - (x_1^{(k)} + 2x_2^{(k)}))
\end{aligned}
$$

Iterative Methods for Linear System Solving

Initial Approximation: $x_1 = 0, x_2 = 0, x_3 = 0$

Iter	0	1	2	3	4	$\cdots$	8
x_1	0	1.833	2.038	2.085	2.004	$\cdots$	2.000
x_2	0	0.714	1.181	1.053	1.001	$\cdots$	1.000
x_3	0	0.2	0.852	1.080	1.038	$\cdots$	1.000

- Derive an update of solutions using equations

$$x_1^{(k+1)} = \tfrac{1}{6}(11 - (-2x_2^{(k)} + x_3^{(k)}))$$
$$x_2^{(k+1)} = \tfrac{1}{7}(5 - (-2x_1^{(k)} + 2x_3^{(k)}))$$
$$x_3^{(k+1)} = \tfrac{1}{-5}(-1 - (x_1^{(k)} + 2x_2^{(k)}))$$

Stop when $\| \vec{x}^{(k+1)} - \vec{x}^{(k)} \| < 10^{-4}$

Need to define **norm** $\| \vec{x}^{(k+1)} - \vec{x}^{(k)} \|$.

Jacobi Method for Linear System Solving in a Matrix Notation

- Represent iteartive computation in a matrix notation

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}^{k+1} = \begin{bmatrix} 0 & \frac{2}{6} & -\frac{1}{6} \\ \frac{2}{7} & 0 & -\frac{2}{7} \\ \frac{1}{5} & \frac{2}{5} & 0 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}^{k} + \begin{pmatrix} \frac{11}{6} \\ \frac{5}{7} \\ \frac{1}{5} \end{pmatrix}$$

General iterative method:

Assign an initial value to $\vec{x}^{(0)}$

k=0

Do

$$\vec{x}^{(k+1)} = H * \vec{x}^{(k)} + d$$

until $\| \vec{x}^{(k+1)} - \vec{x}^{(k)} \| < \varepsilon$

Matrix nototation:

$x^{k+1} = d + H\, x^{k}$

Definition: Norm of a Vector

Given $x = (x_1, x_2, \cdots x_n)$:

$$\| x \|_1 = \sum_{i=1}^{n} | x_i |$$

$$\| x \|_2 = \sqrt{\sum | x_i |^2}$$

$$\| x \|_\infty = max | x_i |$$

$$x = (-1, 1, 2)$$

Example

$$\| x \|_1 = 4$$

$$\| x \|_2 = \sqrt{1 + 1 + 2^2} = \sqrt{6}$$

$$\| x \|_\infty = 2$$

21

Iterative Methods for Linear System Solving: Jacobi method vs. Gauss Seidel method

- Gauss Seidel uses updated solutions ASAP

$\Longrightarrow$ Jacobi method.

$$
\begin{aligned}
x_1^{k+1} &= \tfrac{1}{6}(11 - (-2x_2^k + x_3^k)) \\
x_2^{k+1} &= \tfrac{1}{7}(5 - (-2x_1^k + 2x_3^k)) \\
x_3^{k+1} &= \tfrac{1}{-5}(-1 - (x_1^k + 2x_2^k))
\end{aligned}
$$

$\Longrightarrow$ Gauss-Seidel method.

$$
\begin{aligned}
x_1^{k+1} &= \tfrac{1}{6}(11 - (-2x_2^k + x_3^k)) \\
x_2^{k+1} &= \tfrac{1}{7}(5 - (-2x_1^{k+1} + 2x_3^k)) \\
x_3^{k+1} &= \tfrac{1}{-5}(-1 - (x_1^{k+1} + 2x_2^{k+1}))
\end{aligned}
$$

Matrix nototation:

$x^{k+1} = d + H\, x^k$

22

Example with Gauss-Sidel

$\Longrightarrow$ Gauss-Seidel method.

$$
\begin{aligned}
x_1^{k+1} &= \tfrac{1}{6}(11 - (-2x_2^k + x_3^k)) \\
x_2^{k+1} &= \tfrac{1}{7}(5 - (-2x_1^{k+1} + 2x_3^k)) \\
x_3^{k+1} &= \tfrac{1}{-5}(-1 - (x_1^{k+1} + 2x_2^{k+1}))
\end{aligned}
$$

$$\varepsilon = 10^{-4}$$

	0	1	2	3	4	5
x_1	0	1.833	2.069	1.998	1.999	2.000
x_2	0	1.238	1.002	0.995	1.000	1.000
x_3	0	1.062	1.015	0.998	1.000	1.000

It converges faster than Jacobi's method.

23

Convergence of Iterative Methods

Notation:

$x^* \leftrightarrow$ exact solution

$x^k \leftrightarrow$ solution vector at step k

Definition: Sequence $x^0, x^1, x^2, \cdots, x^n$ converges to the solution x^* with respect to norm $\| \, . \, \|$ if $\| \, x^k - x^* \, \| < \varepsilon$ when k is very large.

i.e. $k \to \infty, \| \, x^k - x^* \, \| \to 0$

Example of Ax=b that Jacobi and GS converge

Give an example of matrix A so that solving Ax = b iteratively can converge?

Definition: Matrix A is **strictly diagonally dominant** if

$$| a_{ii} | > \sum_{j=1, j \neq i}^{n} | a_{ij} | \qquad \text{i=1,2,...,n}$$

Theorem: If A is strictly diagonally dominant. Then both Gauss-Seidel and Jacobi methods converge.

Example of Ax=b that can be solved by iterative methods

$$\begin{pmatrix} 6 & -2 & 1 \\ -2 & 7 & 2 \\ 1 & 2 & -5 \end{pmatrix} x = \begin{pmatrix} 11 \\ 5 \\ -1 \end{pmatrix}$$

$$|6| > 2 + 1$$

$$7 > 2 + 2$$

$$5 > 1 + 2$$

Matrix A is strictly diagonally dominant:

Both Jacobi and G.S. methods will converge.

Sparse matrix in linear system solving

- **Given a matrix A for linear system Ax=b**

  If it contains a lot of zeros, the code design should take advantage of this:

  - Not store too many known zeros.
  - Code should explicitly skip those operations applied to zero elements.

  Example: $y_0 = y_{n+1} = 0.$

$$y_0 - 2y_1 + y_2 = h^2$$

$$y_1 - 2y_2 + y_3 = h^2$$

$$\vdots$$

$$y_{n-1} - 2y_n + y_{n+1} = h^2$$

Sparse matrix in linear system solving

- **Iterative solution for solving linear**

This set of equations can be rewritten as:

$$\begin{pmatrix} -2 & 1 & & & & \\ 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & & \ddots & 1 & \\ & & & 1 & -2 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{pmatrix} = \begin{pmatrix} h^2 \\ h^2 \\ \vdots \\ h^2 \\ h^2 \end{pmatrix}$$

Jacobi method with sparse matrix multiplication notation y = d + H*y

Code format:

Repeat

  For i= 1 to n

    $y_i^{new} = 0.5(y_{i-1}^{old} + y_{i+1}^{old} - h^2)$

  Endfor

Until $\| \vec{y}^{new} - \vec{y}^{old} \| < \varepsilon$

How to represent Gauss-Seidel?

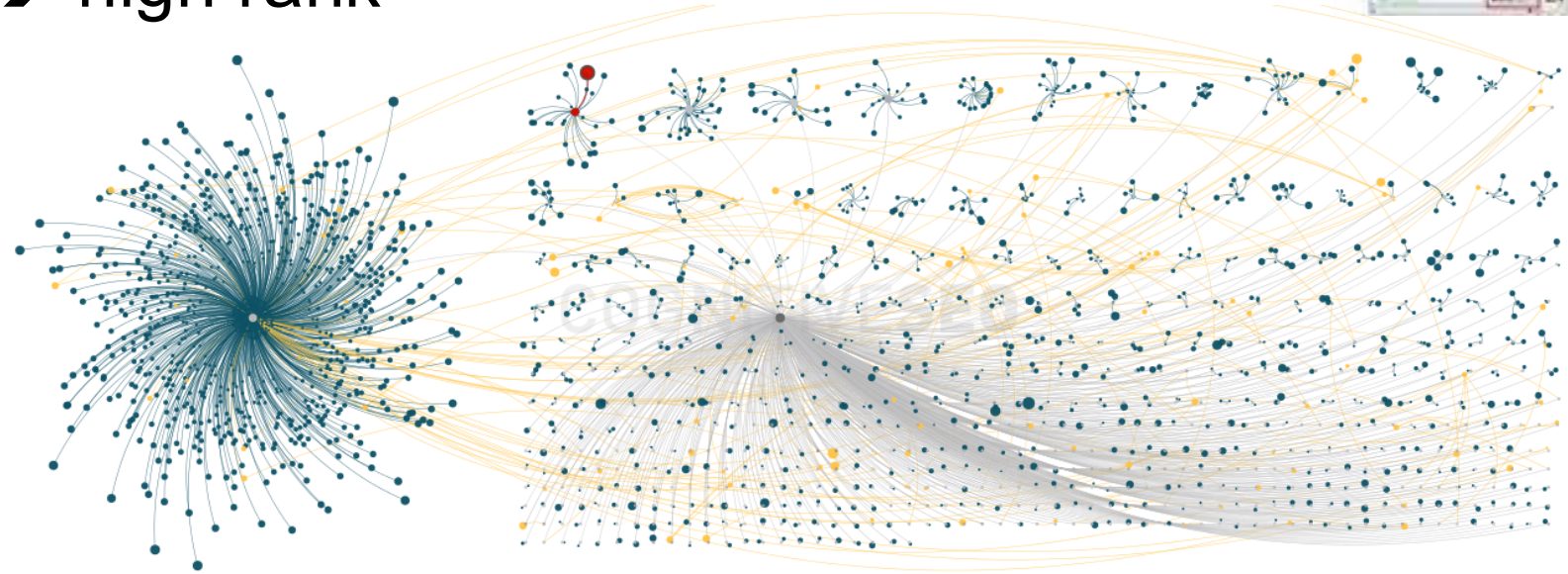
Use of Iterative Solver for Web Page Ranking with Google PageRank

- Set up a linear equation for each web page
- There are billions of pages → billions of equations

Ranking Pages based on their Popularity

Give pages ranks (scores) based on links to them

- Links from many pages ➔ high rank
- Link from a high-rank page ➔ high rank

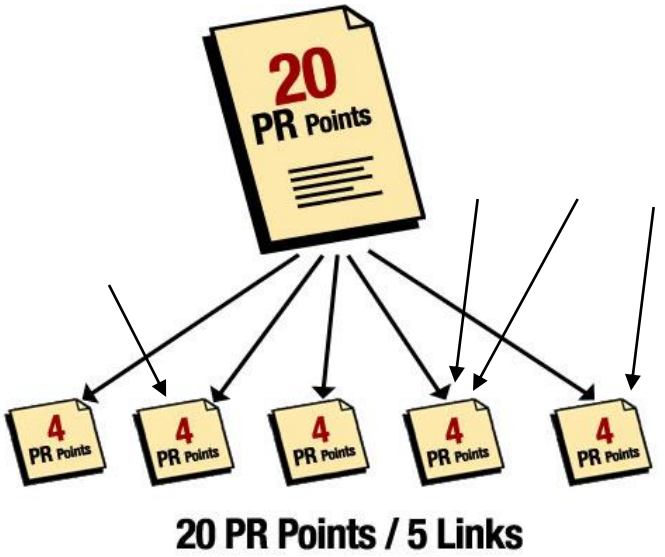


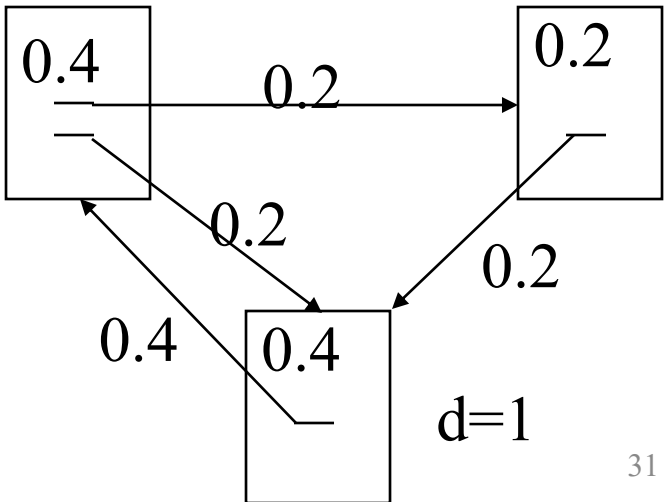
PageRank Algorithm for Modeling Page Reputation in Web Search Ranking

- Model page reputation for every page x.
- PR(x) is the page rank of each page.

$$PR(x) = (1-d) + d\sum_{i=1}^{n} \frac{PR(t_i)}{C(t_i)}$$

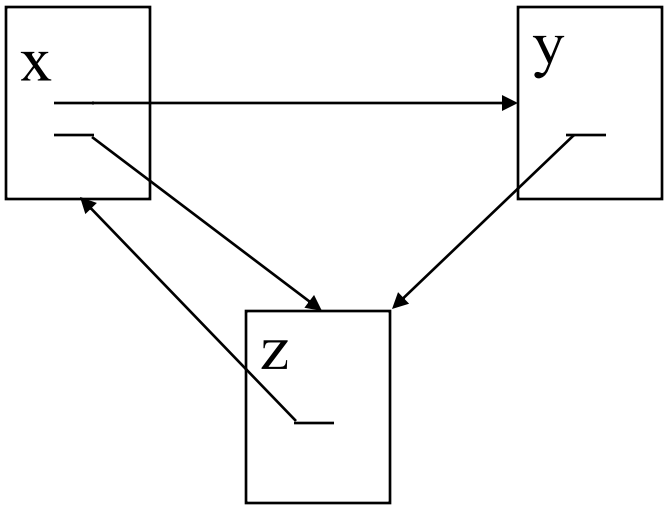
- C(t) is out-degree of parent node t.
- d is a damping factor. 0 ≤ d≤1





Set equations for a graph with 3 web pages

$$PR(x) = (1-d) + d\sum_{i=1}^{n}\frac{PR(t_i)}{C(t_i)}$$



d=0.85

PR(x)=0.15 +0.85 *PR(z)

PR(y)=0.15 +0.85 *PR(x)/2

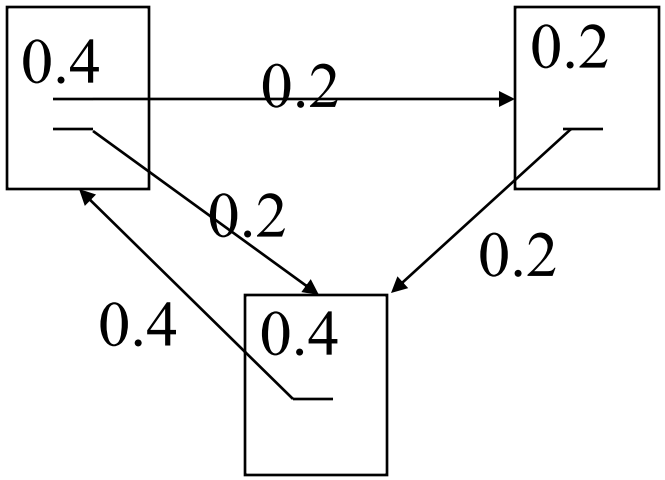
PR(z)=0.15 +0.85 *(PR(x)/2 + PR(y))

d=1

PR(x)=PR(z)

PR(y)=PR(x)/2

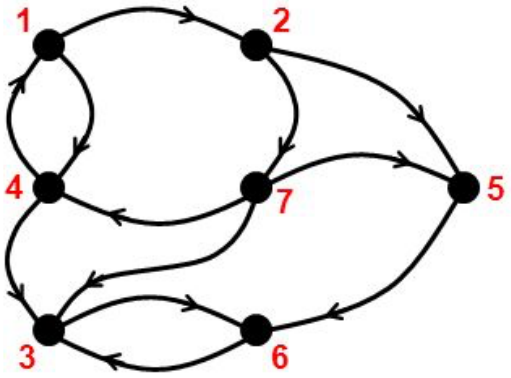
PR(z)=PR(x)/2 + PR(y)



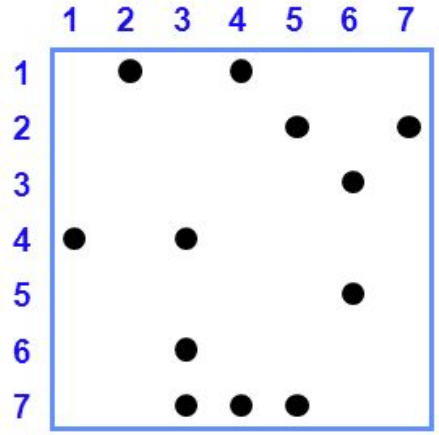
Matrix representation on link relationship for PageRank computation

$$PR(x) = (1-d) + d\sum_{i=1}^{n} \frac{PR(t_i)}{C(t_i)}$$

Link analysis of the web



Graph representation

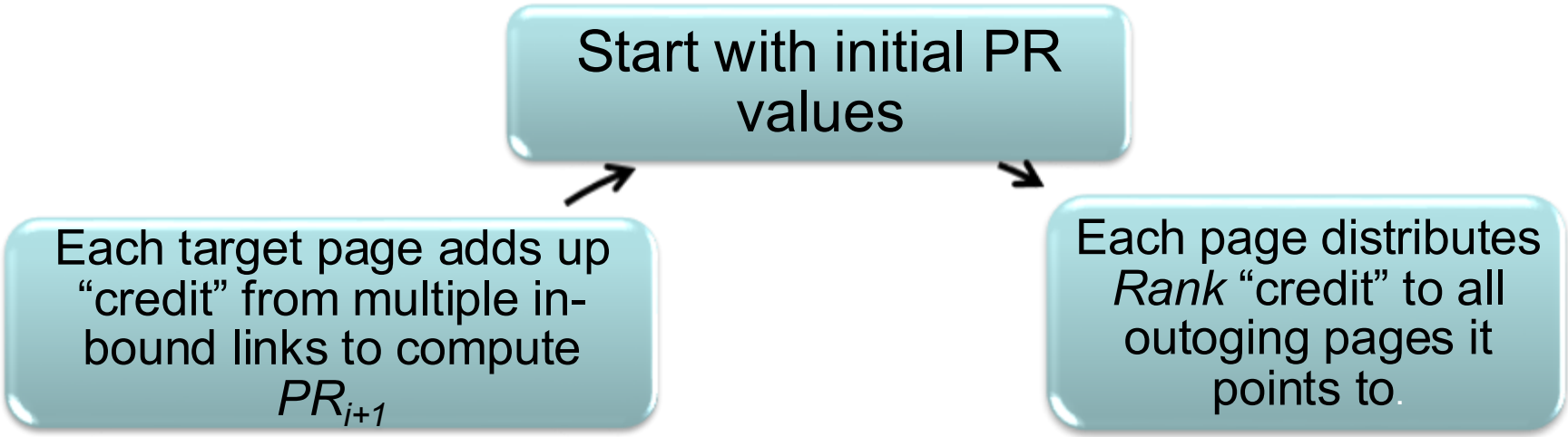


matrix representation

- Web page = vertex
- Link = directed edge
- Link matrix: A_{ij} = 1 if page i links to page j

Computing PageRank Iteratively with Jacobi Method

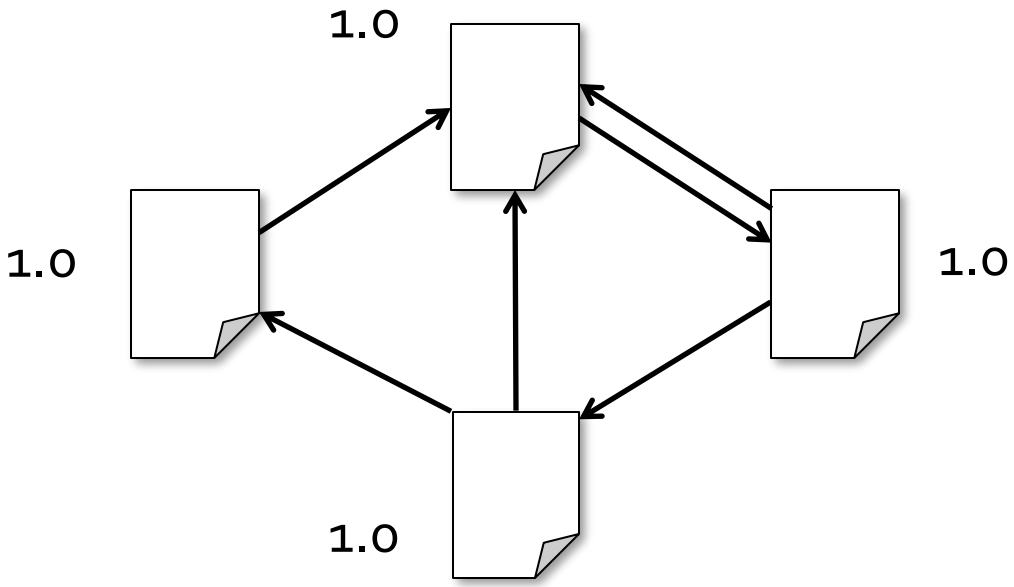
There are n equations for n web pages



- Effects at each iteration is local. i+1th iteration depends only on ith iteration
- At iteration i, PageRank for individual pages can be computed independently

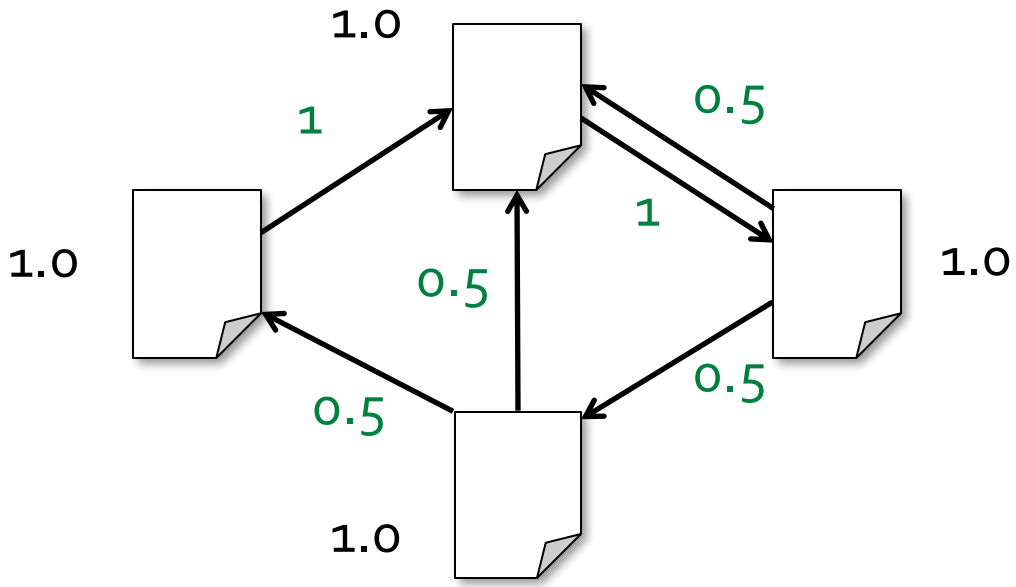
Demo for the Iterative Algorithm: Round 1

1. Start each page with initial page rank value 1
2. On each round, have page p contribute $rank_p$ / $|outdegree_p|$ to its outgoing neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs



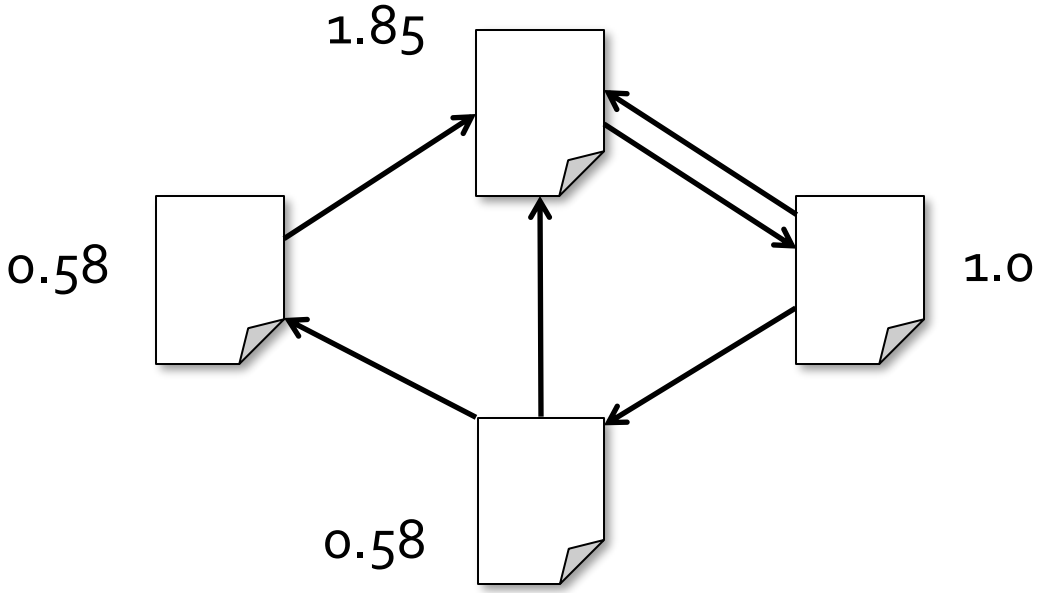
Demo for the Iterative Algorithm: Round 2

1. Start each page at a rank of 1
2. On each round, have page *p* contribute $rank_p$ / $|outdegree_p|$ to its outgoing neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs



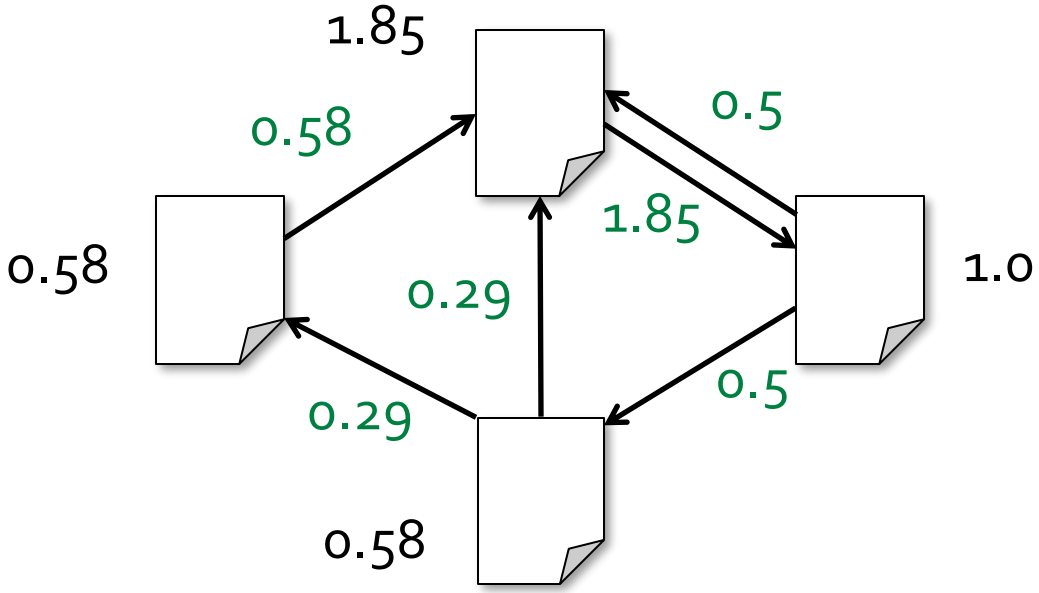
Demo for the Iterative Algorithm: Round 3

1. Start each page at a rank of 1
2. On each round, have page p contribute $rank_p$ / $|outdegree_p|$ to its outgoing neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs



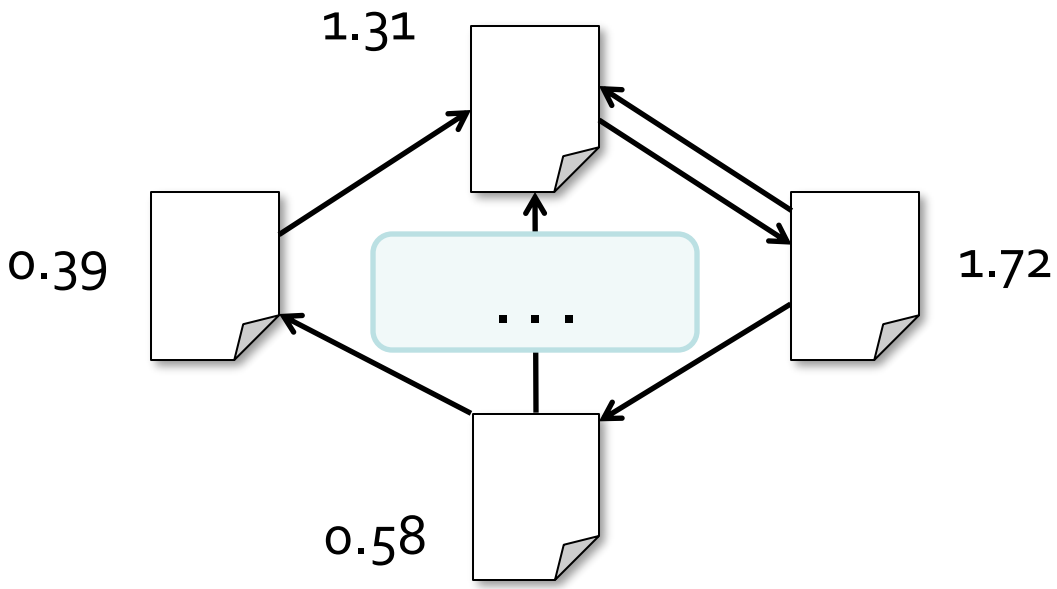
Demo for the Iterative Algorithm: Round 4

1. Start each page at a rank of 1
2. On each round, have page p contribute $rank_p$ / $|outdegree_p|$ to its neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs



Demo for the Iterative Algorithm: Round 5

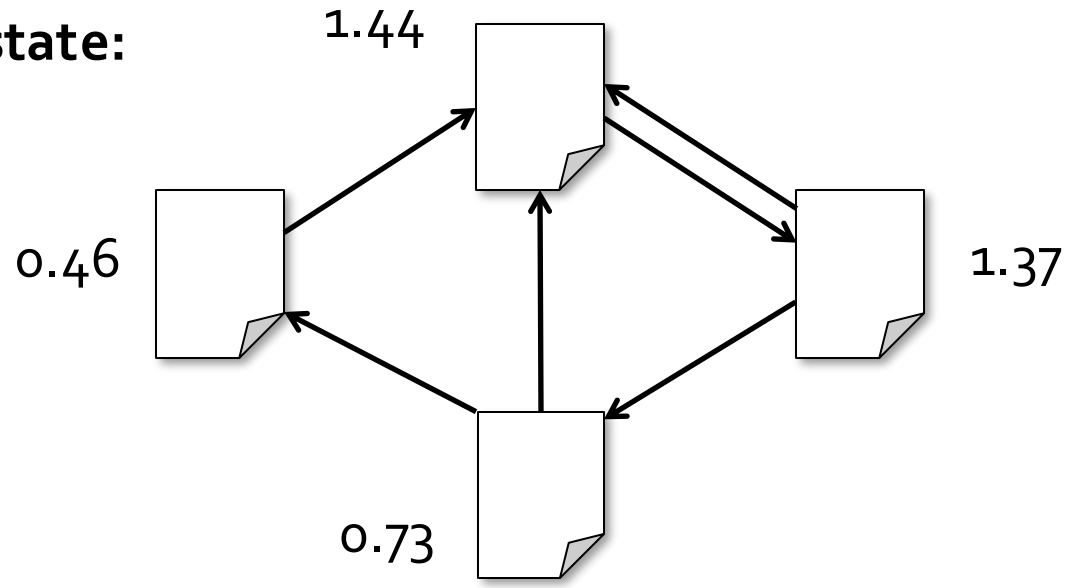
1. Start each page at a rank of 1
2. On each rount, have page p contribute $rank_p$ / $|outdegree_p|$ to its neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs



Demo for the Iterative Algorithm: Round 6

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $rank_p$ / $|outdegree_p|$ to its outgoing neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs

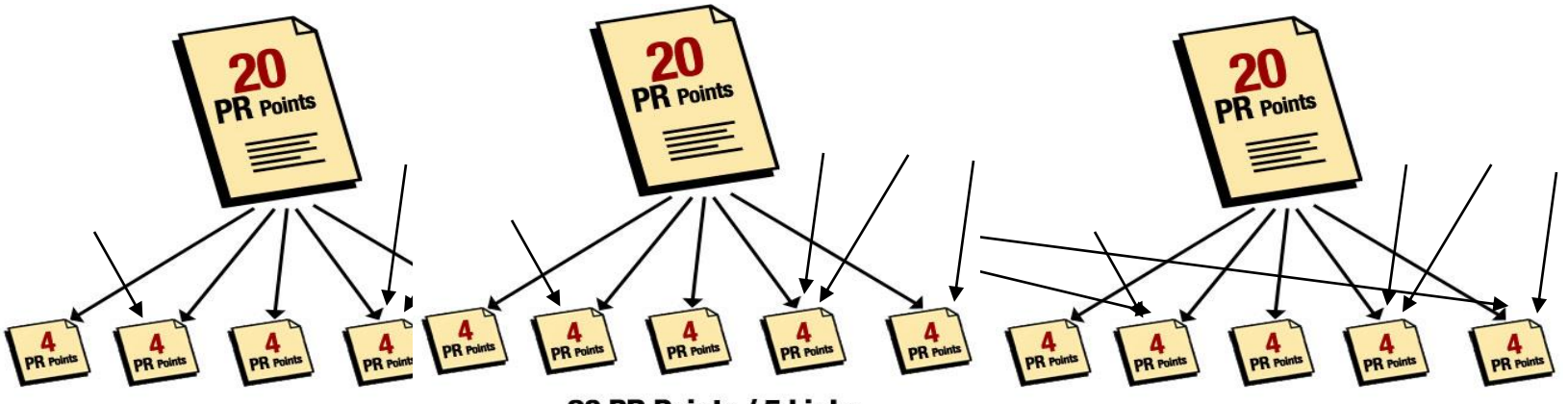
Final state:



Parallel Iterative Algorithm for a Large-Scale Web Graph

Let each process (or thread) be responsible for a subset of graph vertices (web pages). Repeat the following map-reduce phases:

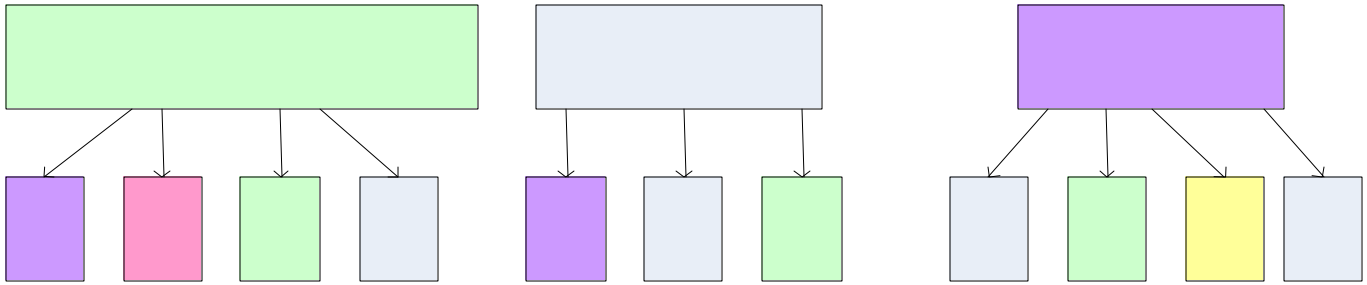
1. **Map:** Every process sends credits of web pages to their outgoing neighbors (children)
2. **Reduce:** Every process receives credits from the parents of its assigned web pages, and updates their page rank value



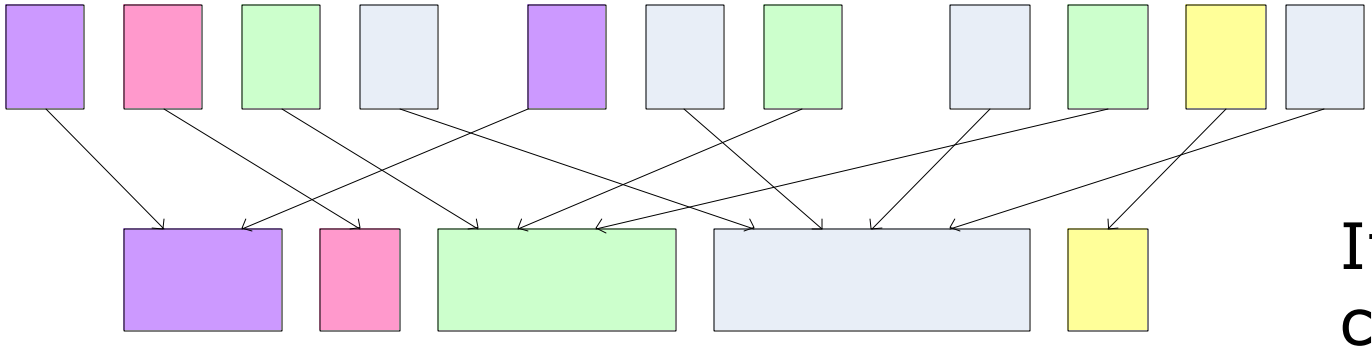
Parallel Algorithm for PageRank

Let each color represent an assigned thread (or process)

Map Phase: distribute PageRank "credit" to outgoing neighbors



Reduce Phase: gather up PageRank "credit" from multiple sources to compute new PageRank value



Iterate until convergence

Source of Image: Lin 2008

Summary:Parallel Scientific Computing Algorithms

- **Basic operations**
- **Solving linear systems of equations**
  - Gaussian Elimination direct method for dense matrices
  - Jacobi/Gauss-Seidel iterative method for sparse matrices
- **Use of iterative solver for Google PageRank**
  - Equations involve billions of unknown variables
  - Parallel Jacobi method in a sparse matrix format