# *A short course on: Preconditioned Krylov subspace methods*

**Yousef Saad**
**University of Minnesota**
**Dept. of Computer Science and Engineering**

**Universite du Littoral, Jan 19-30, 2005**

## *Outline*

**Part 1**

- Introd., discretization of PDEs
- Sparse matrices and sparsity
- Basic iterative methods (Relaxation..)

**Part 2**

- Projection methods
- Krylov subspace methods

**Part 3**

- Preconditioned iterations
- Preconditioning techniques
- Parallel implementations

**Part 4**

- Eigenvalue problems
- Applications –



**INTRODUCTION TO SPARSE MATRICES**

## *Typical Problem:*

**Physical Model**
↓
**Nonlinear PDEs**
↓
**Discretization**
↓
**Linearization (Newton)**
↓
**Sequence of Sparse Linear Systems** $Ax = b$

## What are sparse matrices?

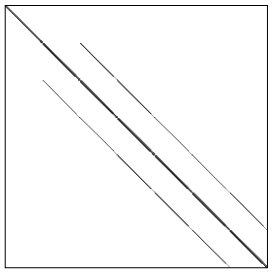
**Usual definition:** "..matrices that allow special techniques to take advantage of the large number of zero elements and the structure." (J. Wilkinson)

**A few applications of sparse matrices:** Structural Engineering, Reservoir simulation, Electrical Networks, optimization problems, ...

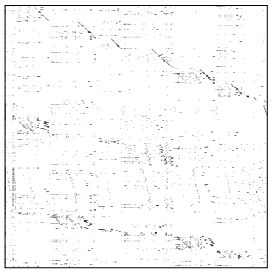
**Goals:** Much less storage and work than dense computations.

**Observation:** $A^{-1}$ is usually dense, but $L$ and $U$ in the LU factorization may be reasonably sparse (if a good technique is used).





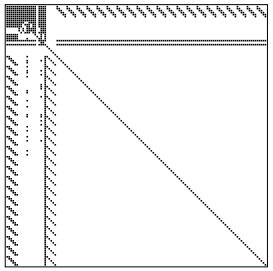
PORES3: Unsymmetric MATRIX FROM PORES



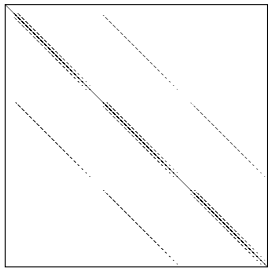
BP_1000: UNSYMMETRIC BASIS FROM LP PROBLEM BP



## Nonzero patterns of a few sparse matrices



ARC130: Unsymmetric matrix from laser problem. a.r.curtis, oct 1974



SHERMAN5: fully implicit black oil simulator 16 by 23 by 3 grid, 3 unk

- **Two types of matrices:** structured (e.g. Sherman5) and unstructured (e.g. BP_1000)
- **Main goal of Sparse Matrix Techniques:** To perform standard matrix computations economically i.e., without storing the zeros of the matrix.
- **Example:** To add two square dense matrices of size $n$ requires $O(n^2)$ operations. To add two sparse matrices $A$ and $B$ requires $O(nnz(A) + nnz(B))$ where $nnz(X) =$ number of nonzero elements of a matrix $X$.
- For typical Finite Element /Finite difference matrices, number of nonzero elements is $O(n)$.

## Sparse Matrices in Typical Applications

- The matrices PORES3 and SHERMAN5 are from Oil Reservoir Simulation. Typically: 3 unknowns per mesh point
- Initially Oil reservoir simulators often used rectangular grids. Oil companies = first commercial buyers of (vector) supercomputers
- New simulators getting more sophisticated ➠ larger and more difficult problems to solve.
- Number of unknowns per node higher (e.g. combustion models).
- Finer, more accurate, 3-D models.
- A naive but representative problem: $100 \times 100 \times 100$ grid + $\approx 10$ unknowns per node ➠ $N \approx 10^7$, and $nnz \approx 7 \times 10^8$.



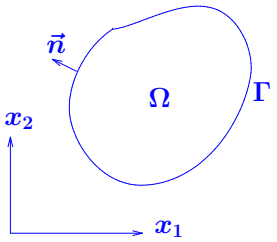
DISCRETIZATION OF PDES - INTRODUCTION

## Discretization of PDEs

➠ Common Partial Differential Equation (PDE) :

$$\frac{\partial^2 u}{\partial x_1^2} + \frac{\partial^2 u}{\partial x_2^2} = f, \text{ for } x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \text{ in } \Omega$$

where $\Omega$ = bounded, open domain in $\mathbb{R}^2$.




➠ Requires boundary conditions:

| | |
|---|---|
| Dirichlet: | $u(x) = \phi(x)$ |
| Neumann: | $\frac{\partial u}{\partial \vec{n}}(x) = 0$ |
| Cauchy: | $\frac{\partial u}{\partial \vec{n}}(x) + \alpha(x)u(x) = \gamma(x)$ |



## Discretization of PDEs - Basic approximations

Formulas derives from Taylor series expansion:

$$u(x+h) = u(x) + h\frac{du}{dx} + \frac{h^2}{2}\frac{d^2u}{dx^2} + \frac{h^3}{6}\frac{d^3u}{dx^3} + \frac{h^4}{24}\frac{d^4u}{dx^4}(\xi_+),$$

➠ Simplest scheme: forward difference

$$\frac{du}{dx} = \frac{u(x+h) - u(x)}{h} - \frac{h}{2}\frac{d^2u(x)}{dx^2} + O(h^2) \approx \frac{u(x+h) - u(x)}{h}$$

➠ Centered differences for second derivative:

$$\frac{d^2u(x)}{dx^2} = \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} - \frac{h^2}{12}\frac{d^4u(\xi)}{dx^4},$$

where $\xi_- \leq \xi \leq \xi_+$.

## Difference Schemes for the Laplacean

Using centered differences for both the $\frac{\partial^2}{\partial x_1^2}$ and $\frac{\partial^2}{\partial x_2^2}$ terms - with mesh sizes $h_1$ and $h_2$ :

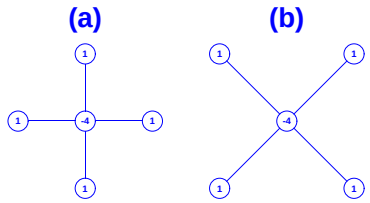
$$\Delta u(x) \approx \frac{u(x_1+h_1,x_2)-2u(x_1,x_2)+u(x-h_1,x_2)}{h_1^2} + \frac{u(x_1,x_2+h_2)-2u(x_1,x_2)+u(x_1,x_2-h_2)}{h_2^2}.$$

If $h_1 = h_2 = h$ then

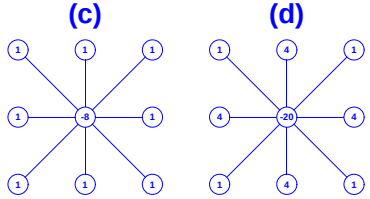
$$\Delta u(x) \approx \frac{1}{h^2}\left[u(x_1+h,x_2)+u(x_1-h,x_2)+u(x_1,x_2+h) + u(x_1,x_2-h)-4u(x_1,x_2)\right], \quad (1)$$



Another approximation: exploit the four points $u(x_1 \pm h, x_2 \pm h)$ [part (b) of Figure]




Five-point stencils (a) the standard stencil, (b) skewed stencil.




Two nine-point centered difference stencils



Error for the 5-point approximation (1) is

$$\frac{h^2}{12}\left(\frac{\partial^4 u}{\partial^4 x_1}+\frac{\partial^4 u}{\partial^4 x_2}\right)+O(h^3).$$

The two other schemes (c) and (d) obtained by combining the standard and skewed stencils are second order accurate. But (d) is sixth order for harmonic functions, i.e., functions whose Laplacean is zero.



## Finite Differences for 1-D Problems

Consider the one-dimensional equation,

$$-u''(x) = f(x) \text{ for } x \in (0,1) \quad (2)$$

$$u(0) = u(1) = 0. \quad (3)$$

discretize [0,1] uniformly:

$$x_i = i \times h,\ i = 0,\ldots,n+1 \quad \text{where} \quad h = 1/(n+1)$$

Dirichlet boundary conditions $u(x_0) = u(x_{n+1}) = 0$

With centered differences the equation at the point $\xi_i$,

$$-u_{i-1} + 2u_i - u_{i+1} = h^2\, f_i,$$

the unknowns $u_j$ approximates $u(\xi_j)$ and $f_i = f(\xi_i)$.

$$Ax = f$$

where

$$A = \frac{1}{h^2}\begin{pmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & -1 & 2 & -1 & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{pmatrix}.$$



# Finite Differences for 2-D Problems

Consider this simple problem,

$$-\left(\frac{\partial^2 u}{\partial x_1^2} + \frac{\partial^2 u}{\partial x_2^2}\right) = f \quad \text{in } \Omega \tag{4}$$

$$u = 0 \quad \text{on } \Gamma \tag{5}$$

where $\Omega$ is now the rectangle $(0, l_1) \times (0, l_2)$ and $\Gamma$ its boundary.

Discretize uniformly :

$$x_{1,i} = i \times h_1, i = 0, \ldots, n_1 + 1 \quad x_{2,j} = j \times h_2, j = 0, \ldots, n_2 + 1$$

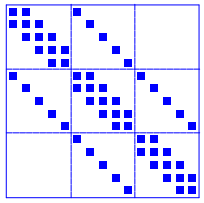
where

$$h_1 = \frac{l_1}{n_1 + 1} \quad h_2 = \frac{l_2}{n_2 + 1}.$$



The resulting matrix has the following block structure:

$$A = \frac{1}{h^2}\begin{pmatrix} B & -I & \\ -I & B & -I \\ & -I & B \end{pmatrix} \quad \text{with} \quad B = \begin{pmatrix} 4 & -1 & & \\ -1 & 4 & -1 & \\ & -1 & 4 & -1 \\ & & -1 & 4 \end{pmatrix}.$$



Matrix for $7 \times 5$ finite difference mesh



DATA STRUCTURES

## Data Structures. General Observations

- The use of a proper data structures is critical to achieving good performance.
- Many data structures; sometimes unnecessary variants.
- These seem more useful for iterative methods than for direct methods.
- Basic linear algebra kernels (e.g., matrix-vector products) depend on data structures.



## Some Common Data Structures

- **DNS** Dense
- **BND** Linpack Banded
- **COO** Coordinate
- **CSR** Compressed Sparse Row
- **CSC** Compressed Sparse Column
- **MSR** Modified CSR
- **ELL** Ellpack-Itpack
- **DIA** Diagonal
- **BSR** Block Sparse Row
- **SSK** Symmetric Skyline
- **NSK** Nonsymmetric Skyline
- **JAD** Jagged Diagonal



## The coordinate format (COO)

$$A = \begin{pmatrix} 1. & 0. & 0. & 2. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 6. & 0. & 7. & 8. & 9. \\ 0. & 0. & 10. & 11. & 0. \\ 0. & 0. & 0. & 0. & 12. \end{pmatrix}$$

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AA = | 12. | 9. | 7. | 5. | 1. | 2. | 11. | 3. | 6. | 4. | 8. | 10. |
| JR = | 5 | 3 | 3 | 2 | 1 | 1 | 4 | 2 | 3 | 2 | 3 | 4 |
| JC = | 5 | 5 | 3 | 4 | 1 | 4 | 4 | 1 | 1 | 2 | 4 | 3 |



## Compressed Sparse Row (CSR) format

$$A = \begin{pmatrix} 1. & 0. & 0. & 2. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 6. & 0. & 7. & 8. & 9. \\ 0. & 0. & 10. & 11. & 0. \\ 0. & 0. & 0. & 0. & 12. \end{pmatrix}$$

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AA = | 1. | 2. | 3. | 4. | 5. | 6. | 7. | 8. | 9. | 10. | 11. | 12. |
| JA = | 1 | 4 | 1 | 2 | 4 | 1 | 3 | 4 | 5 | 3 | 4 | 5 |

| | | | | | | |
|---|---|---|---|---|---|---|
| IA = | 1 | 3 | 6 | 10 | 12 | 13 |

Variants/ related formats: Compressed Sparse Column format, Modi£ed Sparse Row format (MSR).

## The Diagonal (DIA) format

$$A = \begin{pmatrix} 1. & 0. & 2. & 0. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 0. & 6. & 7. & 0. & 8. \\ 0. & 0. & 9. & 10. & 0. \\ 0. & 0. & 0. & 11. & 12. \end{pmatrix}$$

DA =

| | | |
|---|---|---|
| * | 1. | 2. |
| 3. | 4. | 5. |
| 6. | 7. | 8. |
| 9. | 10. | * |
| 11 | 12. | * |

IOFF = -1 0 2



## The Ellpack-Itpack format

$$A = \begin{pmatrix} 1. & 0. & 2. & 0. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 0. & 6. & 7. & 0. & 8. \\ 0. & 0. & 9. & 10. & 0. \\ 0. & 0. & 0. & 11. & 12. \end{pmatrix}$$

AC =

| | | |
|---|---|---|
| 1. | 2. | 0. |
| 3. | 4. | 5. |
| 6. | 7. | 8. |
| 9. | 10. | 0. |
| 11 | 12. | 0. |

JC =

| | | |
|---|---|---|
| 1 | 3 | 1 |
| 1 | 2 | 4 |
| 2 | 3 | 5 |
| 3 | 4 | 4 |
| 4 | 5 | 5 |



## BLOCK MATRICES

$$A = \begin{pmatrix} 1. & 2. & 0. & 0. & 3. & 4. \\ 5. & 6. & 0. & 0. & 7. & 8. \\ 0. & 0. & 9. & 10. & 11. & 12. \\ 0. & 0. & 13. & 14. & 15. & 16. \\ 17. & 18. & 0. & 0. & 20. & 21. \\ 22. & 23. & 0. & 0. & 24. & 25. \end{pmatrix}$$

AA =

| | | |
|---|---|---|
| 1. 3. | 9. 11. | 17. 20. |
| 5. 7. | 15. 13. | 22. 24. |
| 2. 4. | 10. 12. | 18. 21. |
| 6. 8. | 14. 16. | 23. 25. |

JA = 1 5 3 5 1 5

IA = 1 3 5 7



- Each column in AA holds a 2 x 2 block. JA(k) = col. index of (1,1) entries of k-th block. AA may be declared as AA(2,2,6)
- Block formats are important in many applications..
- Also valuable: block structure with variable block size.

- Can also store the blocks row-wise in AA.

AA =

| | | | |
|---|---|---|---|
| 1. | 5. | 2. | 6. |
| 3. | 7. | 4. | 8. |
| 9. | 15. | 10. | 14. |
| 11. | 13. | 12. | 16. |
| 17. | 22. | 18. | 23. |
| 20. | 24. | 21. | 25. |

JA = 1 5 3 5 1 5

IA = 1 3 5 7

- Each row of AA holds a 2 x 2 block (Drawback).
- $JA, IA$ same as before. AA can be declared as AA(6,2,2)
- If elements of blocks are accessed at same time: scheme 1 better.
- If elements of similar positions in different blocks are accessed at same time then scheme 2 better.



## Relation Performance-Data Structures

- Performance can vary significantly for the same operation.

| Storage Scheme/ Kernel | CRAY-2 | Alliant |
|---|---|---|
| Compressed Sparse Row | 8.5 | 7.96 |
| Compressed Sparse Column | 14.5 | 0.6 |
| Ellpack-Itpack | 31 | 6.72 |
| Diagonal | 66.5 | 9.53 |
| Diagonal Unrolled | 99 | 13.92 |

MFLPS for matrix-vector ope's on CRAY-2 & Alliant FX-8 [From sparse matrix benchmark, (Wijshoff & Saad, 1989)]

- Best algorithm for one machine may be worst for another.



## BASIC LINEAR ALGEBRA KERNELS

Philosophy: select a numnber of common linear algebar 'kernels' (e.g. dot-products) and develop highly tuned libraries for them on each machine.

(A) BLAS1: vector operations.

(B) BLAS2: Matrix - Vector type operations

(C) BLAS3: Matrix - Matrix type operations (Blocking)



## Graph Representations of Sparse Matrices

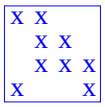
- Graph theory is a fundamental tool in sparse matrix techniques.

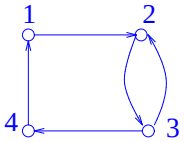
Graph $G = (V, E)$ of an $n \times n$ matrix $A$ defined by

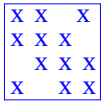
Vertices $V = \{1, 2, ...., N\}$.

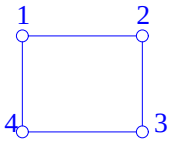
Edges $E = \{(i, j) | a_{ij} \neq 0\}$.

- Graph is undirected if matrix has symmetric structure: $a_{ij} \neq 0$ iff $a_{ji} \neq 0$.










**Example:** Adjacency graph of:

$$A = \begin{pmatrix} \star & \star & & & \star & \\ \star & \star & \star & & & \star \\ & \star & \star & & & \\ & & & \star & \star & \\ \star & & & \star & \star & \star \\ & \star & & & \star & \star \end{pmatrix}.$$

**Example:** For any matrix $A$, what is the graph of $A^2$? [interpret in terms of paths in the graph of $A$]



## DIRECT VERSUS ITERATIVE METHODS

**Current consensus:**

- For two-dimensional problems direct solvers are often preferred.
- For three-dimensional problems, iterative methods are advantageous.
- For problems with a large degree of freedom per grid point, say $\geq 10$, situation similar to 3-D problems.

**Difficulty:**

- No robust 'black-box' iterative solvers.



## Reorderings and graphs

- Let $\pi = \{i_1, \cdots, i_n\}$ a permutation
- $A_{\pi,*} = \{a_{\pi(i),j}\}_{i,j=1,\ldots,n}$ = matrix $A$ with its $i$-th row replaced by row number $\pi(i)$.
- $A_{*,\pi}$ = matrix $A$ with its $j$-th column replaced by column $\pi(j)$.
- Define $P_\pi = I_{\pi,*}$ = "Permutation matrix" – Then:

(1) Each row (column) of $P_\pi$ consists of zeros and exactly one "1"
(2) $A_{\pi,*} = P_\pi A$
(3) $P_\pi P_\pi^T = I$
(4) $A_{*,\pi} = A P_\pi^T$

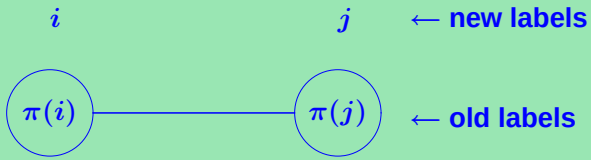
Consider now:

$$A' = A_{\pi,\pi} = P_\pi A P_\pi^T$$

- Element in position $(i, j)$ in matrix $A'$ is exactly element in position $(\pi(i), \pi(j))$ in $A$. ($a'_{ij} = a_{\pi(i),\pi(j)}$)

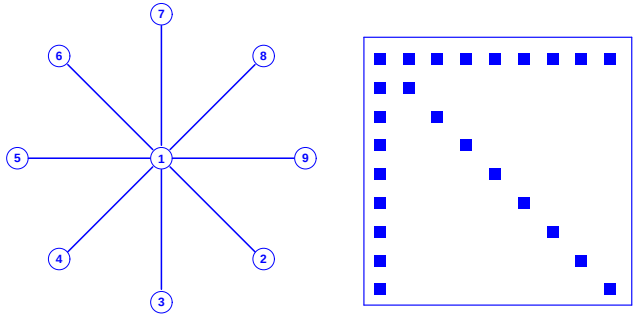
$$(i,j) \in E_{A'} \iff (\pi(i), \pi(j)) \in E_A$$

- General picture :



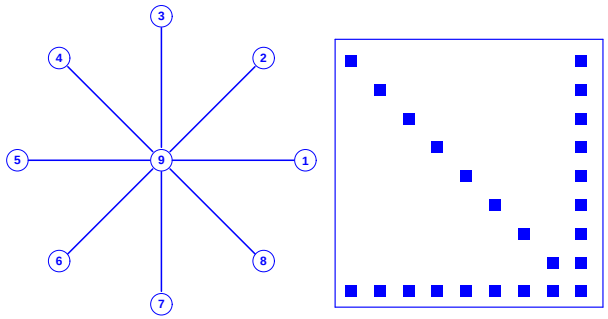



**Example** A 9 × 9 'arrow' matrix and its adjacency graph.






Graph and matrix after permuting the nodes in reverse order.

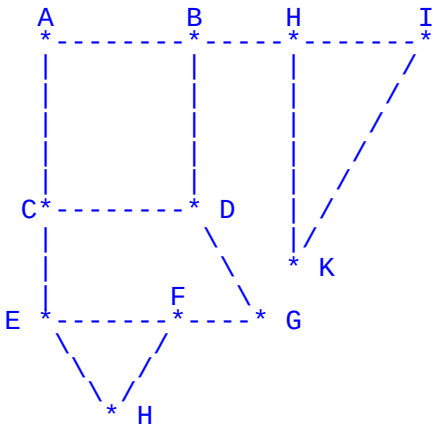





## *Cuthill-McKee, reverse Cuthill-McKee orderings*

- A class of reordering techniques proceeds by levels in the graph.
- Related to *Breadth First Search* (BFS) traversal in graph theory.
- Idea of BFS is to visit the nodes by 'levels'. Level 0 = level of starting node.
- Start with a node, visit its neighbors, then the (unmarked) neighbors of its neighbors, etc...

**Example:**




```
BFS from node A:
Level 0: A
Level 1: B, C;
Level 2: E, D, H;
Level 3: I, K, E, F, G, H.
```



## Implementation using levels

**Algorithm $BFS(G, v)$ – by level sets –**

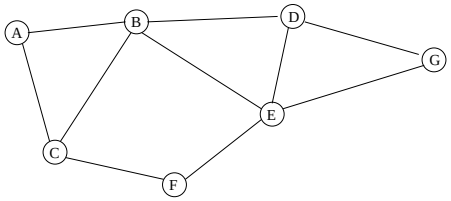
- **Initialize $S = \{v\}$, $seen = 1$; Mark $v$;**
- **While $seen < n$ Do**
  - **$S_{new} = \emptyset$;**
  - **For each node $v$ in $S$ do**
    - **For each unmarked $w$ in adj($v$) do**
      - **Add $w$ to $S_{new}$;**
      - **Mark $w$;**
      - **$seen + +$;**
  - **$S := S_{new}$**



## *Cuthill McKee ordering*

➨ **Algorithm proceeds by levels – and is identical with BFS, except that within each level nodes are odered by increasing degree.**

**Example**




| Level | Nodes | Deg. | Order |
|---|---|---|---|
| 0 | A | 2 | A |
| 1 | B, C | 4, 3 | C, B |
| 2 | D, E, F | 3, 4, 2 | F, D, E |
| 3 | G | 2 | G |



## *Reverse Cuthill McKee ordering*

➨ **Observation: The Cuthill - Mc Kee ordering has a tendency to create small arrow matrices ordered upward (the wrong way).**

➨ **Idea: Take the reverse ordering: Reverse Cuthill M Kee ordering.**

## Orderings used in direct solution methods

- Two broad types of orderings used:
  - Minimal degree ordering + many variations
  - Nested dissection ordering + many variations
- Minimal degree ordering is easiest to describe:

**At each step of GE, select next node to eliminate, as the node $v$ of smallest degree. After eliminating node $v$, update degrees and repeat.**

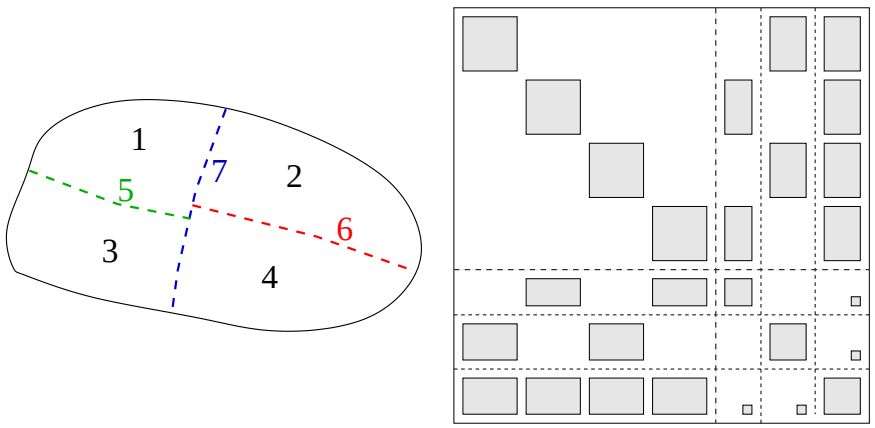


## Nested Dissection

- Easily described by using recursivity and by exploiting 'separators'.
- Main step: 'separate' the graph in three parts, two of which have no coupling between each other. The third set has couplings with vertices from both of the first sets and is referred to as a sepator.
- Key idea: dissect the graph; take the subgraphs and dissect them recursively.



## Nested dissection ordering and corresponding reordered matrix






For regular simple $q \times q$ meshes, it can be shown that fill-in is of order $q^2 \log q$ and computational cost of factorization is $O(q^3)$.

**A useful result** [Rose-Tarjan fill-path theorem]

- A **Fill-path** is a path between two vertices $i$ and $j$ in the graph of $A$ such that all vertices in the path, except the end points $i$ and $j$, are numbered less than $i$ and $j$.

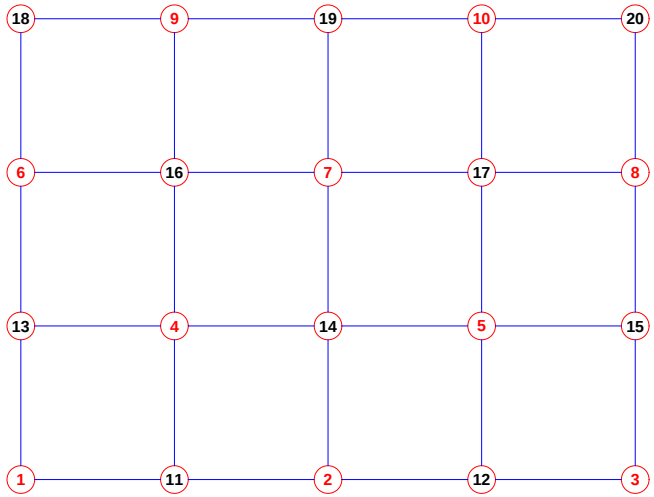
**THEOREM** There is a fill-in in entry $(i, j)$ at the completion of the Gaussian elimination process **if and only if**, there exists a fill-path between $i$ and $j$.

- Separating a graph $\equiv$ finding 3 sets of vertices: $V_1, V_2, S$ such that $V = V_1 \cup V_2 \cup S$ and $V_1$ and $V_2$ have no couplings. Labeling nodes of $S$ last prevents fill-ins between nodes of $V_1$ and $V_2$.

## Multicoloring

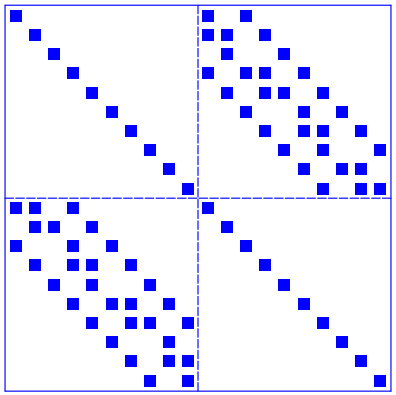
- General technique that can be exploited in many different ways to introduce parallelism – generally of order $N$.
- Constitutes one of the most successful techniques (when applicable) on (vector) supercomputers.

**Simple example:** Red-Black ordering.






## Corresponding matrix



- Observe: L-U solves (or SOR sweeps) will require only diagonal scalings + matrix-vector products with matrices of size $N/2$.



## Solution of Red-black systems

Red-Black ordering leads to equations of the form:

$$\begin{pmatrix} D_1 & E \\ F & D_2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

$D_1$ and $D_2$ are diagonal.

**Question:** How to solve such a system?

**Method 1:** use ILU(0) on this block system. $O(N)$ Parallelism.

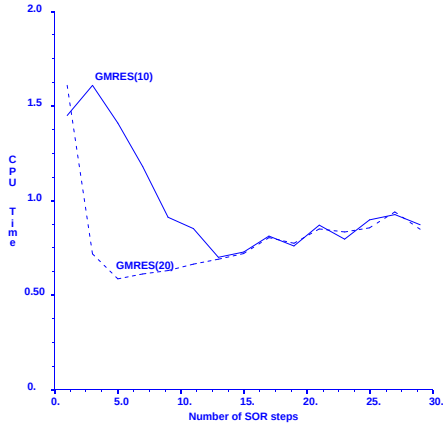
- Often, the number of iterations is higher than with the natural ordering. But still competitive for easy problems.
- Could use a more accurate preconditioner [e.g., ILUT]. However: $O(N)$ parallelism lost.

**Method 2:** SOR/SSOR(k) preconditioner.

▶ No such difficulty with SOR/SSOR...

▶ A 'more accurate preconditioner' means more SOR/ SSOR steps. [SOR(k) → k steps of SOR]. Can perform quite well.

Iteration times versus $k$ for SOR($k$) preconditioned GMRES






**Method 3:** Eliminate the red unknowns from the system

▶ Reduced system: $(D_2 - FD_1^{-1}E)x_2 = b_2 - FD_1^{-1}b_1.$

- Again a sparse linear system with half as many unknowns.
- Can often be efficiently solved with only diagonal preconditioning.

**Question:** Can we recursively color the reduced system and get a "second-level" reduced system?

**Answer:** Yes but need to generalize red-black ordering.



## *How to generalize Red-Black ordering?*

Answer: **Multicoloring** & **independent sets**

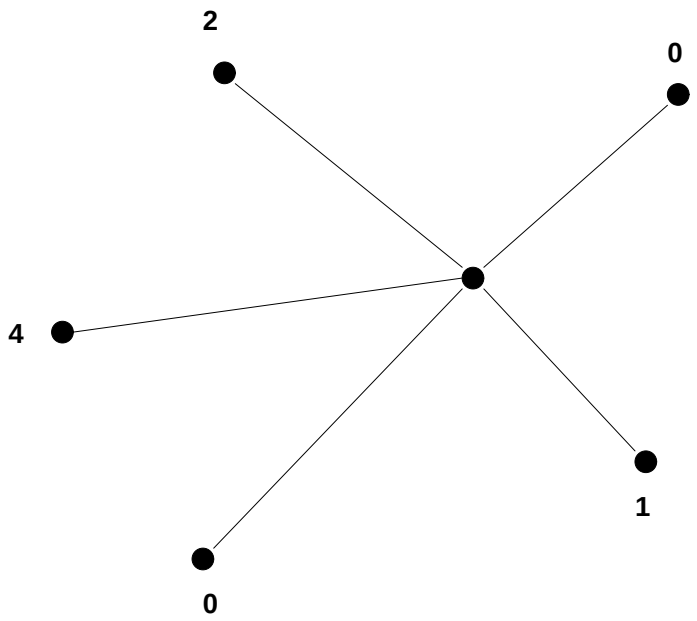
**A greedy multicoloring technique:**

- Initially assign color number zero (uncolored) to every node.
- Choose an order in which to traverse the nodes.
- Scan all nodes in the chosen order and at every node $i$ do

$$Color(i) = \min\{k \neq 0 | k \neq Color(j), \forall\, j \in \text{Adj (i)}\}$$

Adj(i) = set of nearest neighbors of $i$ = $\{k \mid a_{ik} \neq 0\}$.





## Independent Sets

An independent set (IS) is a set of nodes that are not coupled by an equation. The set is maximal if all other nodes in the graph are coupled to a node of IS. If the unknowns of the IS are labeled first, then the matrix will have the form:

$$\begin{pmatrix} B & F \\ E & C \end{pmatrix}$$

in which $B$ is a diagonal matrix, and $E$, $F$, and $C$ are sparse.

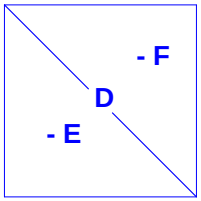
Greedy algorithm: Scan all nodes in a certain order and at every node $i$ do: if $i$ is not colored color it Red and color all its neighbors Black. Independent set: set of red nodes. Complexity: $O(|E|+|V|)$.



## BASIC RELAXATION METHODS

## BASIC RELAXATION SCHEMES

Relaxation schemes: based on the decomposition $A = D - E - F$




$D$ = diag(A), $-E$ = strict lower part of $A$ and $-F$ its strict upper part.

Gauss-Seidel iteration for solving $Ax = b$:

$$(D - E)x^{(k+1)} = Fx^{(k)} + b$$

→ idea: correct the $j$-th component of the current approximate solution, $j = 1, 2, ..n$, to zero the $j - th$ component of residual.



Can also define a backward Gauss-Seidel Iteration:

$$(D - F)x^{(k+1)} = Ex^{(k)} + b$$

and a Symmetric Gauss-Seidel Iteration: forward sweep followed by backward sweep.

Over-relaxation is based on the decomposition:

$$\omega A = (D - \omega E) - (\omega F + (1 - \omega)D)$$

→ successive overrelaxation, (SOR):

$$(D - \omega E)x^{(k+1)} = [\omega F + (1 - \omega)D]x^{(k)} + \omega b$$

## Iteration matrices

Jacobi, Gauss-Seidel, SOR, & SSOR iterations are of the form

$$x^{(k+1)} = Mx^{(k)} + f$$

- $M_{Jac} = D^{-1}(E+F) = I - D^{-1}A$
- $M_{GS}(A) = (D-E)^{-1}F == I - (D-E)^{-1}A$
- $M_{SOR}(A) = (D-\omega E)^{-1}(\omega F + (1-\omega)D) = I - (\omega^{-1}D - E)^{-1}A$
- $M_{SSOR}(A) = I - (2\omega^{-1}-1)(\omega^{-1}D-F)^{-1}D(\omega^{-1}D-E)^{-1}A$
  $= I - \omega(2\omega-1)(D-\omega F)^{-1}D(D-\omega E)^{-1}A$



## General convergence result

Consider the iteration: $x^{(k+1)} = Gx^{(k)} + f$

(1) Assume that $\rho(A) < 1$. Then $I - G$ is non-singular and $G$ has a fixed point. Iteration converges to a fixed point for any $f$ and $x^{(0)}$.

(2) If iteration converges for any $f$ and $x^{(0)}$ then $\rho(G) < 1$.

Example: Richardson's iteration $x^{(k+1)} = x^{(k)} + \alpha(b - A^{(k)})$

◇Assume $\Lambda(A) \subset \mathbb{R}$. When does the iteration converge?

➨ Jacobi and Gauss-Seidel converge for diagonal dominant $A$

➨ SOR converges for $0 < \omega < 2$ for SPD matrices



## An observation. Introduction to Preconditioning

➨ The iteration $x^{(k+1)} = Mx^{(k)} + f$ is attempting to solve $(I - M)x = f$. Since $M$ is of the form $M = I - P^{-1}A$ this system can be rewritten as

$$P^{-1}Ax = P^{-1}b$$

where for SSOR, we have

$$P_{SSOR} = (D-\omega E)D^{-1}(D-\omega F)$$

referred to as the SSOR 'preconditioning' matrix.

In other words:

Relaxation Scheme $\Longleftrightarrow$ Preconditioned Fixed Point Iteration