

## SPARSE DIRECT METHODS

- Building blocks for sparse direct solvers
- SPD case. Sparse Column Cholesky/
- Elimination Trees - Symbolic factorization

## Direct Sparse Matrix Methods

**Problem addressed:** Linear systems

$$Ax = b$$

- We will consider mostly Cholesky –
- We will consider some implementation details and tricks used to develop efficient solvers

**Basic principles:**

- Separate computation of structure from rest [symbolic factorization]
- Do as much work as possible statically
- Take advantage of clique formation (supernodes, mass-elimination).

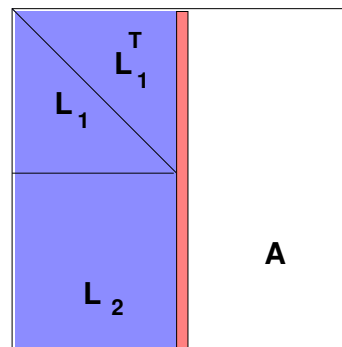
8-2

Davis: Chap. 4 – Direct

## Sparse Column Cholesky

```

For  $j = 1, \dots, n$  Do:
   $l(j : n, j) = a(j : n, j)$ 
  For  $k = 1, \dots, j - 1$  Do:
    // cmod(k,j):
     $l_{j:n,j} := l_{j:n,j} - l_{j,k} * l_{j:n,k}$ 
  EndDo
  // cdiv (j) [Scale]
   $l_{j,j} = \sqrt{l_{j,j}}$ 
   $l_{j+1:n,j} := l_{j+1:n,j} / l_{j,j}$ 
EndDo
    
```



## The four essential stages of a solve

**1. Reordering:**  $A \rightarrow A := PAP^T$

- Preprocessing: uses graph [Min. deg, AMD, Nested Dissection]

**2. Symbolic Factorization:** Build static data structure.

- Exploits 'elimination tree', uses graph only.
- Also: 'supernodes'

**3. Numerical Factorization:** Actual factorization  $A = LL^T$

- Pattern of  $L$  is known. Uses static data structure. Exploits supernodes (blas3)

**4. Triangular solves:** Solve  $Ly = b$  then  $L^T x = y$

8-3

Davis: Chap. 4 – Direct

8-4

Davis: Chap. 4 – Direct

## ELIMINATION TREES

### The notion of elimination tree

- Elimination trees are useful in many different ways [theory, symbolic factorization, etc..]
- For a matrix whose graph is a tree, parent of column  $j < n$  is defined by

$$\text{Parent}(j) = i, \text{ where } a_{ij} \neq 0 \text{ and } i > j$$

- For a general matrix matrix, consider  $A = LL^T$ , and  $G^F =$  'filled' graph = graph of  $L + L^T$ . Then

$$\text{Parent}(j) = \min(i) \text{ s.t. } a_{ij} \neq 0 \text{ and } i > j$$

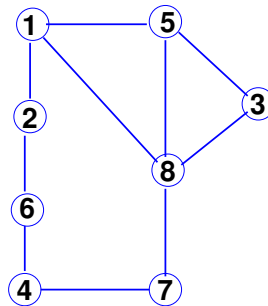
- Defines a tree rooted at column  $n$  (Elimintion tree).

8-6

Davis: Chap. 4 – Direct

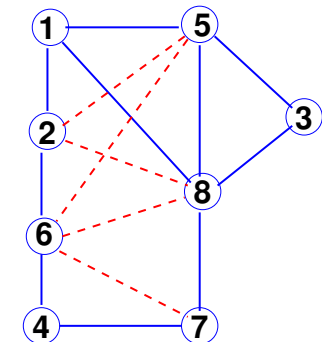
### Example: Original matrix and Graph

$$\begin{bmatrix} 1 & * & & * & & * & & & \\ * & 2 & & & & * & & & \\ & & 3 & & * & & * & & \\ & & & 4 & & * & * & & \\ * & & * & & 5 & & & * & \\ & * & & * & & 6 & & & \\ & & & * & & & 7 & * & \\ * & * & * & * & * & * & * & 8 & \end{bmatrix}$$



### Filled matrix+graph

$$\begin{bmatrix} 1 & * & & * & & * & & & \\ * & 2 & & \blacksquare & * & & \blacksquare & & \\ & & 3 & & * & & * & & \\ & & & 4 & & * & * & & \\ * & \blacksquare & * & & 5 & \blacksquare & & * & \\ & * & & * & & \blacksquare & 6 & \blacksquare & \blacksquare \\ & & & * & & & \blacksquare & 7 & * \\ * & \blacksquare & * & * & * & \blacksquare & * & * & 8 \end{bmatrix}$$



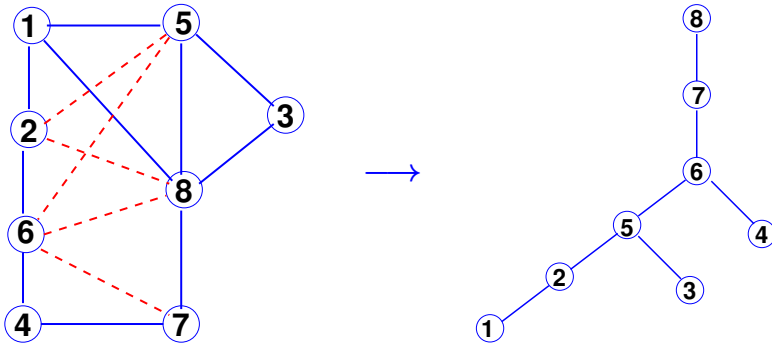
8-7

Davis: Chap. 4 – Direct

8-8

Davis: Chap. 4 – Direct

## Corresponding Elimination Tree



- $\text{Parent}(i) = \text{'first nonzero entry in } L(i+1:n,i)\text{'}$
- $\text{Parent}(i) = \min \{j > i \mid j \in \text{Adj}_{G^F}(i)\}$

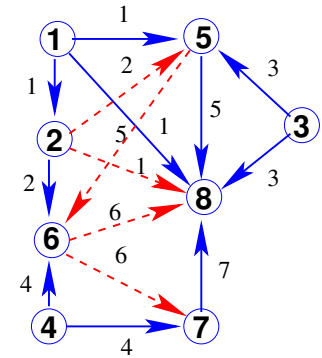
8-9

Davis: Chap. 4 – Direct

## Where does the elimination tree come from?

- Answer in the form of an exercise.

Consider the elimination steps for the previous example. A directed edge means a row (column) modification. It shows the task dependencies. There are unnecessary dependencies. For example:  $1 \rightarrow 5$  can be removed because it is subsumed by the path  $1 \rightarrow 2 \rightarrow 5$ .



**To do:** Remove all the redundant dependencies.. What is the result?

8-10

Davis: Chap. 4 – Direct

## Facts about elimination trees

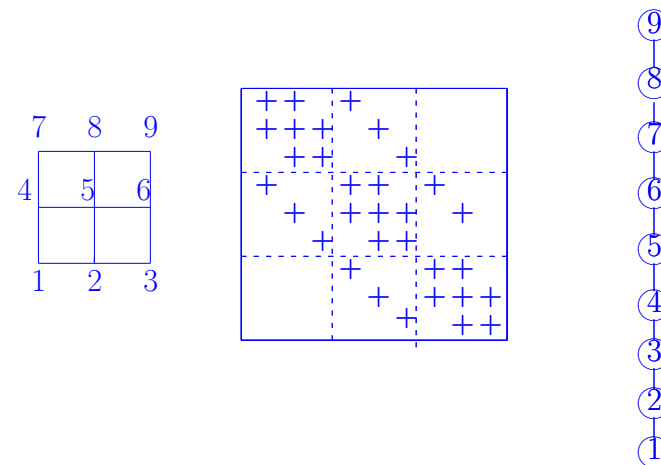
- Elimination Tree defines dependencies between columns.
- The root of a subtree cannot be used as pivot before any of its descendents is processed.
- Elimination tree depends on ordering;
- Can be used to define 'parallel' tasks.
- For parallelism: flat and wide trees  $\rightarrow$  good; thin and tall (e.g. of tridiagonal systems)  $\rightarrow$  Bad.
- For parallel executions, Nested Dissection gives better trees than Minimum Degree ordering.

8-11

Davis: Chap. 4 – Direct

## Elim. tree depends on ordering (Not just the graph)

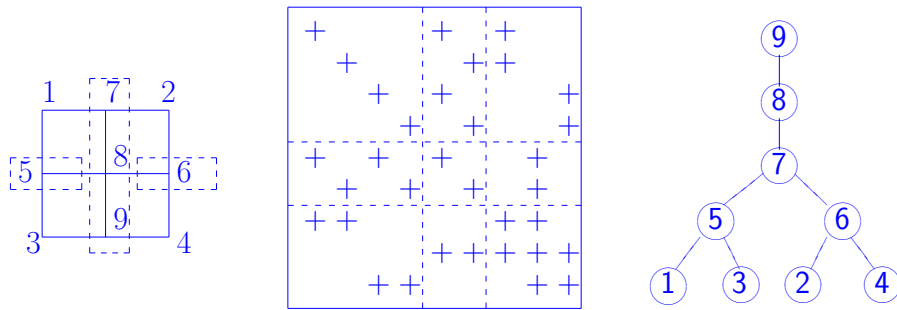
**Example:**  $3 \times 3$  grid for 5-point stencil [natural ordering]



8-12

Davis: Chap. 4 – Direct

- Same example with nested dissection ordering




8-13

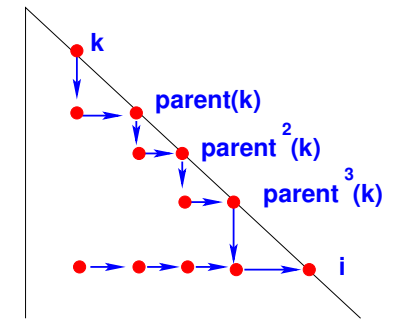
Davis: Chap. 4 – Direct

## Properties

- The elimination tree is a spanning tree of the filled graph [a tree containing all vertices] - obtained by removing edges.

- If  $l_{ik} \neq 0$  then  $i$  is an ancestor of  $k$  in the tree

 In the previous example: follow the creation of the fill-in (6,8).



In particular: if  $a_{ik} \neq 0, k < i$  then  $i \rightsquigarrow k$

- Consequence: no fill-in between branches of the same subtree

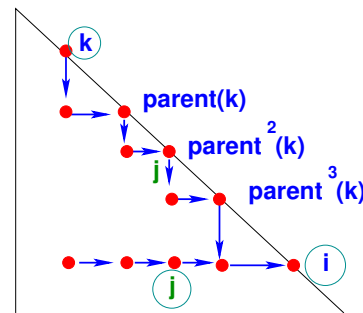
8-14

Davis: Chap. 4 – Direct

## Elimination trees and the pattern of $L$

- It is easy to determine the sparsity pattern of  $L$  because the pattern of a given column is “inherited” by the ancestors in the tree.

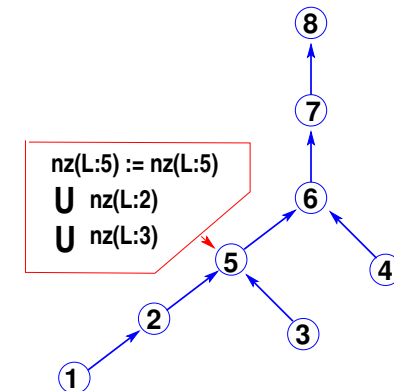
**Theorem:** For  $i > j, l_{ij} \neq 0$  iff  $j$  is an ancestor of some  $k \in Adj_A(i)$  in the elimination tree.



In other words:

$$l_{ij} \neq 0, i > j \text{ iff } \left| \begin{array}{l} \exists k \in Adj_A(i) \text{ s.t.} \\ j \rightsquigarrow k \end{array} \right.$$

In theory: To construct the pattern of  $L$ , go up the tree and accumulate the patterns of the columns. Initially  $L$  has the same pattern as  $TRIL(A)$ .



- However: Let us assume tree is not available ahead of time
- Solution: Parents can be obtained dynamically as the pattern is being built.
- This is the basis of symbolic factorization.

8-16

Davis: Chap. 4 – Direct

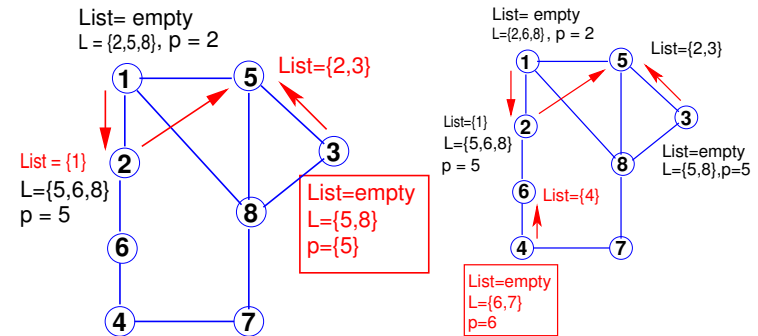
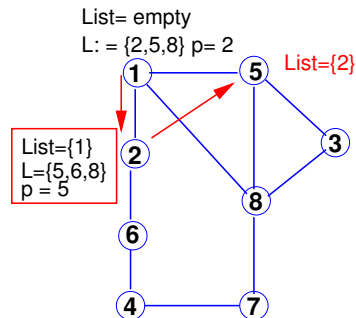
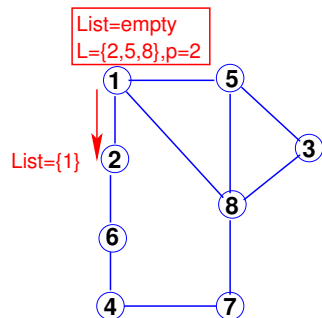
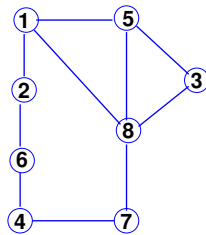
Notation :

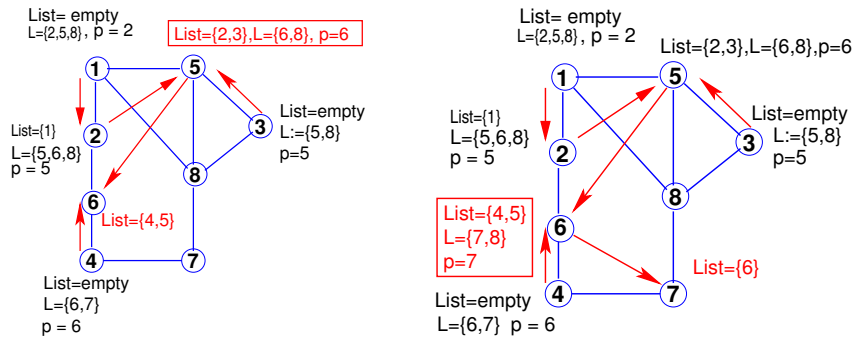
- $nz(X)$  is the pattern of  $X$  (matrix or column, or row). A set of pairs  $(i, j)$
- $tril(X)$  = Lower triangular part of pattern [matlab notation]  $\{(i, j) \in X \mid i > j\}$
- Idea: dynamically create the list of nodes needed to update  $L_{:,j}$ .

### ALGORITHM : 1. Symbolic factorization

1. Set:  $nz(L) = tril(nz(A))$ ,
2. Set:  $list(j) = \emptyset, j = 1, \dots, n$
3. For  $j = 1 : n$
4.   for  $k \in list(j)$  do
5.      $nz(L_{:,j}) := nz(L_{:,j}) \cup nz(L_{:,k})$
6.   end
7.    $p = \min\{i > j \mid L_{i,j} \neq 0\}$
8.    $list(p) := list(p) \cup \{j\}$
9. End

**Example:** Consider the earlier example:





## Multifrontal methods

- Start with the frontal method.
- Recall: Finite element matrix:

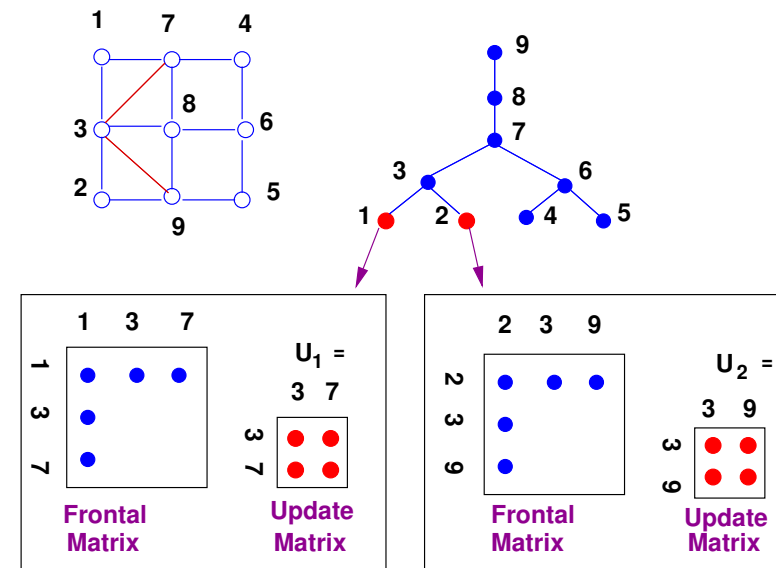
$$A = \sum A^{[e]}$$

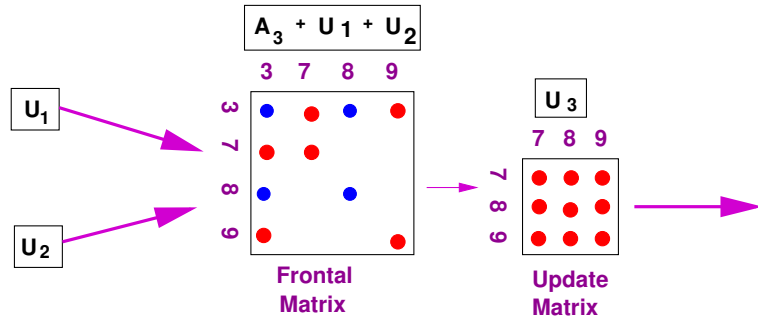
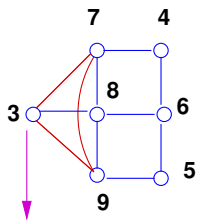
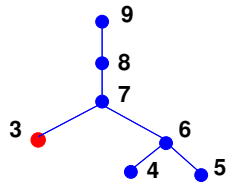
$A^{[e]}$  = element matrix associated with element  $e$ .

- An old idea: Execute Gaussian elimination as the elements are being assembled
- Dependency: variables  $\leftrightarrow$  elements, creates an **assembly tree**.
- Method is called *the frontal method*
- Very popular among finite element users: **saves storage**

## Multifrontal methods: extension to general matrices

- Elimination tree replaces assembly tree
- Proceed in post-order traversal of elimination tree in order not to violate task dependencies.
- When a node is eliminated an **update matrix** is created.
- This matrix is passed to the parent which adds it to its **frontal matrix**.
- Requires a stack of pending update matrices
- Update matrices popped out as they are needed
- Often implemented with nested dissection-type ordering
- More complex than a left-looking algorithm





**Eliminating nodes 1 and 2:**

What happens on matrix

1		*			*		
	2	*					*
*	*	3			■	*	■
			4	*	*		
				5	*		*
			*	*	6		*
*		■	*			7	*
		*		*	*	8	*
	*	■	*			*	9

$\leftarrow U_1(3, :) \leftarrow U_2(3, :)$

$\leftarrow U_1(7, :)$

$\leftarrow U_2(9, :)$

**Supernodes**

➤ Contiguous columns tend to inherit the pattern of the columns from they are updated → Many columns with same sparsity pattern. Supernode = a set of contiguous columns in the Cholesky factor  $L$  that have the same sparsity pattern.

➤ The set  $\{j, j + 1, \dots, j + s\}$  is a supernode if

$$NZ(L_{*,k}) = NZ(L_{*,k+1}) \cup \{k + 1\} \quad j \leq k < j + s$$

where  $NZ(L_{*,k})$  is nonzero set of column  $k$  of  $L$ .

➤ Other terms used: Mass elimination, indistinguishible nodes, active variables in front, subscript compression,...

➤ Gain in performance due to savings in Gather-Scatter operations.