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)

Sparse Column Cholesky

For \( j = 1, \ldots, n \) Do:
\[
l(j : n, j) = \alpha(j : n, j)
\]
For \( k = 1, \ldots, j - 1 \) Do:
\[
// \text{cmod}(k,j):
\]
\[
l_{j:n,j} := l_{j:n,j} - l_{j:k} \times l_{j:n,k}
\]
EndDo
\[
// \text{cdiv}(j) \quad \text{[Scale]}
\]
\[
l_{j,j} = \sqrt{l_{j:j}}
\]
\[
l_{j+1:n,j} := l_{j+1:n,j} / l_{j:j}
\]
EndDo

The four essential stages of a solve

1. Reordering: \( A \rightarrow A := P A P^T \)

- 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 \( A = L L^T \)

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

4. Triangular solves: Solve \( L y = b \) then \( L^T x = y \)
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, consider \( A = LL^T \), and \( G^F = \text{‘filled’ graph} = \text{graph of } L + L^T \). Then
  \[
  \text{Parent}(j) = \min(i) \text{ s.t. } a_{ij} \neq 0 \text{ and } i > j
  \]
- Defines a tree rooted at column \( n \) (Elimination tree).

Example: Original matrix and Graph

\[
\begin{bmatrix}
1 & * & * & * \\
* & 2 & * & * \\
* & * & 3 & * \\
* & * & * & 4 \\
* & * & * & * \\
* & * & * & * \\
* & * & * & * \\
* & * & * & 8
\end{bmatrix}
\]

Filled matrix + graph

\[
\begin{bmatrix}
1 & * & * & * \\
* & 2 & * & * \\
* & * & 3 & * \\
* & * & * & 4 \\
* & * & * & * \\
* & * & * & * \\
* & * & * & * \\
* & * & * & * \\
* & * & * & 8
\end{bmatrix}
\]
**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.

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

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

**Where does the elimination tree come from?**

Answer in the form of an excercise.

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?
Properties

- The elimination tree is a spanning tree of the filled graph \([a \text{ tree containing all vertices}] - \text{obtained by removing edges.}\)

- 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 \(i \sim k\)

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

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 \(i > j, l_{ij} \neq 0\) if \(j\) is an ancestor of some \(k \in Adj_A(i)\) in the elimination tree.

In other words:

\[ l_{ij} \neq 0, i > j \iff \exists k \in Adj_A(i) \text{ s.t. } j \sim k \]

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.
Notation:

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

**ALGORITHM : 1. Symbolic factorization**

1. Set: $\text{nz}(L) = \text{tril}(\text{nz}(A))$.
2. Set: list($j$) = $\emptyset$, $j = 1, \cdots, n$
3. For $j = 1 : n$
4. for $k \in \text{list}(j)$ do
5. $\text{nz}(L_{:,j}) := \text{nz}(L_{:,j}) \cup \text{nz}(L_{:,k})$
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:

\[
\begin{bmatrix}
1 & 2 & 6 & 4 & 7 & 8 & 5 & 3 \\
        & 2 & 6 & 4 & 7 & 8 & 5 & 3 \\
          &   &     & 4 & 7 & 8 & 5 & 3 \\
            &   &   &     & 7 & 8 & 5 & 3 \\
              &   &   &   &     & 8 & 5 & 3 \\
                &   &   &   &   &     & 5 & 3 \\
                    &   &   &   &   &   & 3 & \\
                        &   &   &   &   &   &   & 1
\end{bmatrix}
\]

List={1}
List=empty
$L=\{2,5,8\}$, $p=2$
$\text{List=empty } L=\{2,5,8\}$, $p=2$
$\text{List=empty } L=\{5,8\}$, $p=5$
$\text{List=empty } L=\{6,7\}$, $p=6$
$\text{List=empty } L=\{2,5\}$, $p=2$
$\text{List=empty } L=\{5,8\}$, $p=5$
$\text{List=empty } L=\{6,7\}$, $p=6$