High Performance Matrix Computations

J.-Y. L’Excellent (INRIA/LIP-ENS Lyon)
Jean-Yves.L.Excellent@ens-lyon.fr
Office 352

2007-2008
prepared in collaboration with P. Amestoy, L.Giraud, M. Daydé (ENSEEIHT-IRIT)

Lectures related to sparse direct methods – novembre 2007

Contents

1 Introduction to Sparse Matrix Computations ................................................. 1
  1.1 Motivation and main issues ................................................................. 2
  1.2 Sparse matrices .................................................................................. 3
  1.3 Gaussian elimination ......................................................................... 6
  1.4 Symmetric matrices and graphs ......................................................... 9

2 Ordering sparse matrices ............................................................................ 13
  2.1 Objectives/Outline ............................................................................ 13
  2.2 Fill-reducing orderings .................................................................... 13
  2.3 Impact of fill reduction algorithm on the shape of the tree ............. 19
  2.4 Postorderings and memory usage ..................................................... 21
  2.5 Equivalent orderings and elimination trees ..................................... 24
  2.6 Reorder unsymmetric matrices to special forms ........................... 25
  2.7 Combining approaches .................................................................... 29

3 Factorization of sparse matrices ................................................................. 30
  3.1 Introduction ....................................................................................... 30
  3.2 Elimination tree and Multifrontal approach .................................... 32
  3.3 Comparison between 3 approaches for LU factorization ............... 37
  3.4 Task mapping and scheduling ............................................................ 39
  3.5 Distributed memory approaches ....................................................... 41
  3.6 Some parallel solvers ....................................................................... 44
  3.7 Case study: comparison of MUMPS and SuperLU ......................... 45
  3.8 Concluding remarks ....................................................................... 50

4 Related topics ............................................................................................. 50

1 Introduction to Sparse Matrix Computations

A selection of references

- Books
  - George, Liu, and Ng, Computer Solution of Sparse Positive Definite Systems, book to appear
1.1 Motivation and main issues

Motivations

- solution of linear systems of equations ➔ key algorithmic kernel

\[ \text{Continuous problem} \downarrow \text{Discretization} \downarrow \text{Solution of a linear system} \ Ax = b \]

- Main parameters:
  - Numerical properties of the linear system (symmetry, pos. definite, conditioning, ...)
  - Size and structure:
    * Large (> 100000 × 100000 ?), square/rectangular
    * Dense or sparse (structured / unstructured)
    * Target computer (sequential/parallel)

→ Algorithmic choices are critical

Motivations for designing efficient algorithms

- Time-critical applications
- Solve larger problems
- Decrease elapsed time (parallelism ?)
- Minimize cost of computations (time, memory)

Difficulties

- Access to data :
  - Computer : complex memory hierarchy (registers, multilevel cache, main memory (shared or distributed), disk)
  - Sparse matrix : large irregular dynamic data structures.

→ Exploit the locality of references to data on the computer (design algorithms providing such locality)

- Efficiency (time and memory)
  - Number of operations and memory depend very much on the algorithm used and on the numerical and structural properties of the problem.
  - The algorithm depends on the target computer (vector, scalar, shared, distributed, clusters of Symmetric Multi-Processors (SMP), GRID).

→ Algorithmic choices are critical
1.2 Sparse matrices

Sparse matrices

Example:

\[
\begin{align*}
3 \, x_1 & + 2 \, x_2 & = 5 \\
2 \, x_2 & - 5 \, x_3 & = 1 \\
2 \, x_1 & + 3 \, x_3 & = 0
\end{align*}
\]

can be represented as

\[Ax = b,\]

where

\[
A = \begin{pmatrix}
3 & 2 & 0 \\
0 & 2 & -5 \\
2 & 0 & 3
\end{pmatrix}, \quad
x = \begin{pmatrix}
x_1 \\
x_2 \\
x_3
\end{pmatrix}, \quad
\text{and } b = \begin{pmatrix}
5 \\
1 \\
0
\end{pmatrix}
\]

Sparse matrix: only nonzeros are stored.

Sparse matrix ?

Matrix dwt_592.rua (N=592, NZ=5104); Structural analysis of a submarine

Factorization process

Solution of \(Ax = b\)

- \(A\) is unsymmetric:
  - \(A\) is factorized as: \(A = LU\), where \(L\) is a lower triangular matrix, and \(U\) is an upper triangular matrix.
  - Forward-backward substitution: \(Ly = b\) then \(Ux = y\)

- \(A\) is symmetric:
  - \(A = LDL^T\) or \(LL^T\)

- \(A\) is rectangular \(m \times n\) with \(m \geq n\) and \(\min_x \|Ax - b\|_2\):
  - \(A = QR\) where \(Q\) is orthogonal \((Q^{-1} = Q^T\) and \(R\) is triangular).
  - Solve: \(y = Q^Tb\) then \(Rx = y\)
Difficulties

- Only non-zero values are stored
- Factors $L$ and $U$ have far more nonzeros than $A$
- Data structures are complex
- Computations are only a small portion of the code (the rest is data manipulation)
- Memory size is a limiting factor → out-of-core solvers

Key numbers:

1- **Average size**: 100 MB matrix; Factors = 2 GB; Flops = 10 Gflops;

2- **A bit more “challenging”**: Lab. Géosciences Azur, Valbonne
   - Complex matrix arising in 2D $16 \times 10^6$ , $150 \times 10^6$ nonzeros
   - Storage : 5 GB (12 GB with the factors ?)
   - Flops : tens of TeraFlops

3- **Typical performance** (MUMPS):
   - PC LINUX (P4, 2GHz) : 1.0 GFlops/s
   - Cray T3E (512 procs) : Speed-up $\approx 170$, Perf. 71 GFlops/s

Typical test problems:

- **BMW car body**, 227,362 unknowns, 5,757,996 nonzeros, MSC.Software
  - Size of factors: 51.1 million entries
  - Number of operations: $127.9 \times 10^9$

- **BMW crankshaft**, 148,770 unknowns, 5,396,386 nonzeros, MSC.Software
  - Size of factors: 97.2 million entries
  - Number of operations: $127.9 \times 10^9$
Sources of parallelism
Several levels of parallelism can be exploited:

- At problem level: problem can be decomposed into sub-problems (e.g. domain decomposition)
- At matrix level arising from its sparse structure
- At submatrix level within dense linear algebra computations (parallel BLAS, ...)

Data structure for sparse matrices

- Storage scheme depends on the pattern of the matrix and on the type of access required
  - band or variable-band matrices
  - “block bordered” or block tridiagonal matrices
  - general matrix
  - row, column or diagonal access

Data formats for a general sparse matrix A

What needs to be represented

- Assembled matrices: MxN matrix A with NNZ nonzeros.
- Elemental matrices (unassembled): MxN matrix A with NELT elements.
- Arithmetic: Real (4 or 8 bytes) or complex (8 or 16 bytes)
- Symmetric (or Hermitian) → store only part of the data.
- Distributed format?
- Duplicate entries and/or out-of-range values?

Classical Data Formats for Assembled Matrices

- Example of a 3x3 matrix with NNZ=5 nonzeros

\[
\begin{array}{cccc}
1 & 2 & 3 \\
\hline
1 & a_{11} & \\
2 & a_{22} & a_{23} \\
3 & a_{31} & a_{33} \\
\end{array}
\]

- Coordinate format

IRN \[1 : \text{NNZ} = 1 3 2 2 3\]
JCN \[1 : \text{NNZ} = 1 1 2 3 3\]
VAL \[1 : \text{NNZ} = a_{11} a_{31} a_{22} a_{23} a_{33}\]

- Compressed Sparse Column (CSC) format

IRN \[1 : \text{NNZ} = 1 3 2 2 3\]
VAL \[1 : \text{NNZ} = a_{11} a_{31} a_{22} a_{23} a_{33}\]
COLPTR \[1 : N + 1 = 1 3 4 6\]

- Compressed Sparse Column format:
  Similar to CSC, but row by row

- Diagonal format: NDIAG = 4 IDIAG = -2 0 1

\[
\begin{bmatrix}
na & na & a_{31} \\
a_{11} & a_{22} & a_{33} \\
na & a_{23} & na \\
\end{bmatrix}
\]

\(\text{(na: not accessed)}\)
Example of elemental matrix format

\[ \mathbf{A} = \sum_{i=1}^{N_{ELT}} \mathbf{A}_i \]

\[ \mathbf{A}_1 = \begin{pmatrix} 1 & -1 & 2 & 3 \\ 2 & 1 & 1 & 1 \\ 3 & 1 & 1 & 1 \end{pmatrix}, \quad \mathbf{A}_2 = \begin{pmatrix} 3 & 2 & -1 & 3 \\ 2 & 1 & 1 & 1 \\ 3 & 2 & 1 & 1 \end{pmatrix} \]

- \( N=5 \) \( NELT=2 \) \( NVAR=6 \) \( \mathbf{A} = \sum_{i=1}^{N_{ELT}} \mathbf{A}_i \)

\[ \begin{align*}
\text{ELTPTR} \quad [1:N_{ELT}+1] &= 1 \quad 4 \quad 7 \\
\text{ELTVAL} \quad [1:N_{VAL}] &= -1 \quad 2 \quad 1 \quad 2 \quad 1 \quad 2 \quad 1 \quad 3 \quad -1 \quad 2 \quad 3 \quad -1 \\
\text{Remarks:}
\end{align*} \]

- \( NVAR = \text{ELTPTR}(N_{ELT}+1) \)
- \( NVAL = \sum S_i^2 \) (unsym) ou \( \sum S_i(S_i + 1)/2 \) (sym), avec \( S_i = \text{ELTPTR}(i + 1) - \text{ELTPTR}(i) \)
- storage of elements in ELTVAL: by columns

File storage: Rutherford-Boeing

- Standard ASCII format for files
- Header + Data (CSC format), key xyz:
  - \( x=[\text{rcp}] \) (real, complex, pattern)
  - \( y=[\text{suhr}] \) (sym., uns., herm., skew sym., rectang.)
  - \( z=[\text{ae}] \) (assembled, elemental)
- ex: M_T1.RSA, SHIP003.RSE

- Supplementary files: right-hand-sides, solution, permutations...

- Canonical format introduced to guarantee a unique representation (order of entries in each column, no duplicates).

File storage: Rutherford-Boeing

1.3 Gaussian elimination

Gaussian elimination

\[ \mathbf{A} = \mathbf{A}^{(1)}, \quad \mathbf{b} = \mathbf{b}^{(1)}, \quad \mathbf{A}^{(1)} \mathbf{x} = \mathbf{b}^{(1)}: \]

\[ \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots \\ a_{21} & a_{22} & a_{23} & \cdots \\ a_{31} & a_{32} & a_{33} & \cdots \\
\end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \end{pmatrix} \]

\[ 2 - 2 = 2 \quad a_{11} = a_{11}/a_{11} \]

\[ \mathbf{A}^{(2)} \mathbf{x} = \mathbf{b}^{(2)} \]

Finally \( \mathbf{A}^{(3)} \mathbf{x} = \mathbf{b}^{(3)} \)

\[ \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots \\ a_{21} & a_{22} & a_{23} & \cdots \\ a_{31} & a_{32} & a_{33} & \cdots \\
\end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \end{pmatrix} \]

Typical Gaussian elimination step \( k \):

\[ a_{ij}(k+1) = a_{ij}(k) - \frac{a_{ik}(k)}{a_{kk}(k)} \]
Relation with \( A = LU \) factorization

- One step of Gaussian elimination can be written: \( A^{(k+1)} = L^{(k)}A^{(k)} \), with \( L^k = \begin{pmatrix} 1 & 0 & \ldots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \ldots & 1 & 0 \\ -a_{k+1,k} & \ldots & -a_{n,k} & 1 \end{pmatrix} \)

and \( l_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} \).

- Then, \( A^{(n)} = U = L^{(n-1)} \ldots L^{(1)}A \), which gives \( A = LU \), with \( L = [L^{(1)}]^{-1} \ldots [L^{(n-1)}]^{-1} = \begin{pmatrix} 1 & 0 & \ldots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \ldots & 1 & 0 \\ -l_{1,j} & \ldots & -l_{n,j} & 1 \end{pmatrix} \).

- In dense codes, entries of \( L \) and \( U \) overwrite entries of \( A \).

- Furthermore, if \( A \) is symmetric, \( A = LDL^T \) with \( d_{kk} = a_{kk}^{(k)} \):

\[ A = LU = A^t = U^tL^t \] implies \((U)(L^t)^{-1} = L^{-1}U^t = D \) diagonal and \( U = DL^t \), thus \( A = L(DL^t) = LDL^t \).

**Gaussian elimination and sparsity**

**Step k of** \( LU \) **factorization (**\( a_{kk} \) **pivot):**

- For \( i > k \) compute \( l_{ik} = a_{ik}/a_{kk} \) (= \( a_{ik}^{(k)} \)).

- For \( i > k, j > k \)

\[ a_{ij}' = a_{ij} - \frac{a_{ik} \times a_{kj}}{a_{kk}} \]

or

\[ a_{ij}' = a_{ij} - l_{ik} \times a_{kj} \]

- If \( a_{ik} \neq 0 \) et \( a_{kj} \neq 0 \) then \( a_{ij}' \neq 0 \)

- If \( a_{ij} \) was zero → its non-zero value must be stored

\[
\begin{array}{ccc}
  k & j \\
  i & \\
  \cdots & \cdots & \cdots \\
  \text{fill-in}
\end{array}
\]

- Idem for Cholesky :

- For \( i > k \) compute \( l_{ik} = a_{ik}/\sqrt{a_{kk}} \) (= \( a_{ik}' \)).

- For \( i > k, j > k, j \leq i \) (lower triang.)

\[ a_{ij}' = a_{ij} - \frac{a_{ik} \times a_{jk}}{\sqrt{a_{kk}}} \]

or

\[ a_{ij}' = a_{ij} - l_{ik} \times a_{jk} \]
Example

- Original matrix

\[
x \ x \ x \ x \ x \\
x \ x \\
x \ x \\
x \ x \\
x \ x \\
\]

- Matrix is full after the first step of elimination
- After reordering the matrix (1st row and column ↔ last row and column)

\[
x \ x \\
x \ x \\
x \ x \\
x \ x \\
x \ x \ x \ x \ x \\
\]

- No fill-in
- Ordering the variables has a strong impact on
  - the fill-in
  - the number of operations

Table 1: Benefits of Sparsity on matrix of order $2021 \times 2021$ with 7353 nonzeros. (Dongarra etal 91).

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Total storage</th>
<th>Flops</th>
<th>Time (sec.) on CRAY J90</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Syst.</td>
<td>4084 Kwords</td>
<td>$5.503 \times 10^8$</td>
<td>34.5</td>
</tr>
<tr>
<td>Sparse Syst.</td>
<td>71 Kwords</td>
<td>$1.073 \times 10^6$</td>
<td>3.4</td>
</tr>
<tr>
<td>Sparse Syst. and reordering</td>
<td>14 Kwords</td>
<td>$4.2 \times 10^3$</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Efficient implementation of sparse solvers

- Indirect addressing is often used in sparse calculations: e.g. sparse SAXPY

\[
\text{do } i = 1, m \\
A( \text{ind}(i) ) = A( \text{ind}(i) ) + \text{alpha} \ast w( i ) \\
\text{enddo}
\]

- Even if manufacturers provide hardware for improving indirect addressing
  - It penalizes the performance

Switching to dense calculations as soon as the matrix is not sparse enough

Effect of switch to dense calculations

Matrix from 5-point discretization of the Laplacian on a $50 \times 50$ grid (Dongarra etal 91)
Sparse structure should be exploited if density < 10%.
1.4 Symmetric matrices and graphs

Symmetric matrices and graphs

- Assumptions: \( A \) symmetric and pivots are chosen on the diagonal
- Structure of \( A \) symmetric represented by the graph \( G = (V, E) \)
  - Vertices are associated to columns: \( V = \{1, ..., n\} \)
  - Edges \( E \) are defined by: \( (i, j) \in E^0 \iff a_{ij} \neq 0 \)
  - \( G \) undirected (symmetry of \( A \))
- Remarks:
  - Number of nonzeros in column \( j = |\text{Adj}(j)| \)
  - Symmetric permutation \( \equiv \) renumbering the graph

The elimination graph model

Construction of the elimination graphs
Let \( v_i \) denote the vertex of index \( i \). \( G_0 = G(A), i = 1 \).
At each step delete \( v_i \) and its incident edges
Add edges so that vertices in \( \text{Adj}(v_i) \) are pairwise adjacent in \( G_i = G(H_i) \).
\( G_i \) are the so-called elimination graphs.
A sequence of elimination graphs

\[ G_0 : \]
\[ \begin{align*}
&\begin{array}{c}
\text{1} \\
\text{2} \\
\text{3} \\
\text{4} \\
\text{6} \\
\text{5}
\end{array} \\
&\begin{array}{c}
\text{1} \\
\text{2} \\
\text{3} \\
\text{4} \\
\text{6} \\
\text{5}
\end{array}
\end{align*} \]

\[ G_1 : \]
\[ \begin{align*}
&\begin{array}{c}
\text{2} \\
\text{4} \\
\text{3} \\
\text{6} \\
\text{5}
\end{array} \\
&\begin{array}{c}
\text{2} \\
\text{4} \\
\text{3} \\
\text{6} \\
\text{5}
\end{array}
\end{align*} \]

\[ G_2 : \]
\[ \begin{align*}
&\begin{array}{c}
\text{4} \\
\text{3} \\
\text{6} \\
\text{5}
\end{array} \\
&\begin{array}{c}
\text{4} \\
\text{3} \\
\text{6} \\
\text{5}
\end{array}
\end{align*} \]

\[ G_3 : \]
\[ \begin{align*}
&\begin{array}{c}
\text{4} \\
\text{6} \\
\text{5}
\end{array} \\
&\begin{array}{c}
\text{4} \\
\text{6} \\
\text{5}
\end{array}
\end{align*} \]

Introducing the filled graph \( G^+ (A) \)

- Let \( F = L + L^T \) be the filled matrix, and \( G(F) \) the filled graph of \( A \) denoted by \( G^+ (A) \).
- Lemma (Parter 1961) : \((v_i, v_j) \in G^+ \) if and only if \((v_i, v_j) \in G \) or \( \exists k < \min(i, j) \) such that \((v_i, v_k) \in G^+ \) and \((v_k, v_j) \in G^+ \).

\[ G^+ (A) = G(F) \]
\[ F = L + L^T \]

Modeling elimination by reachable sets

- The fill edge \((v_4, v_6)\) is due to the path \((v_4, v_2, v_6)\) in \( G_1 \). However \((v_2, v_6)\) originates from the path \((v_2, v_1, v_6)\) in \( G_0 \).
- Thus the path \((v_4, v_2, v_1, v_6)\) in the original graph is in fact responsible of the fill in edge \((v_4, v_6)\).
- Illustration :

\[ G^+ (A) = G(F) \]
\[ F = L + L^T \]

- This has motivated George and Liu to introduce the notion of reachable sets.
Reachable sets

- Let $G = (V, E)$ be a graph and $S$ be a subset of $V$.
- Definition: The vertex $v$ is said to be reachable from a vertex $w$ through $S$ if there exists a path $(w, v_1, \ldots, v_k, v)$ such that all $v_i \in S$ (where $k$ could be zero).
- Reach$(w, S)$ is the set of reachable vertices from $w$ through $S$.
- Illustration: let $S = \{s_1, s_2, s_3, s_4\}$ then Reach$(w, S) = \{a, b, c\}$

Fill-in and reachable sets

The filled graph $G^+(A)$ can be characterized with reachable sets. Let $S_i = \{v_1 \ldots v_i\}$ denote the set of vertices eliminated at step $i$.

**Theorem 1.** For $i < j$, $(v_i, v_j) \in G^+(A)$ if and only if $v_j \in \text{Reach}(v_i, S_{i-1})$.

In terms of the matrix the set $\text{Reach}(v_i, S_{i-1})$ is the set of row subscripts that correspond to non-zeros entries below the diagonal in column $i$ of $L$.

**Illustration:** Reach$(v_1, S_0) = \{5, 8\}$; Reach$(2, S_1) = \{4, 5\}$; Reach$(3, S_2) = \{8\}$; Reach$(4, S_3) = \{5, 7\}$; Reach$(5, S_4) = \{6, 7, 8\}$

Structure of $L$ factors

**Theorem 2.** Let $w$ be a vertex in $G_i$. The set of vertices adjacent to $w$ in $G_i$ (column structure of $w$) is given by Reach$(w, S_i)$ where the reach operator is applied to the original graph $G_0(A)$.

The structure of $L$ can thus be characterized in terms of the structure of $A$.

**Illustration:** $S_2 = \{1, 2\}$, Reach$(3, S_2) = \{4, 5, 6\}$; Reach$(4, S_2) = \{3, 6\}$; Reach$(5, S_2) = \{3, 6\}$; and Reach$(6, S_2) = \{3, 4, 5\}$ due to the paths $(3, 2, 1, 6)$, $(4, 2, 1, 6)$, $(5, 6)$.

- Previous theorem shows that the whole elimination process can be described implicitly by the sequence of reach operator. It can be considered as an implicit model for elimination, as opposed to the explicit model using elimination graphs.
• This theorem also gives indication on the impact of reordering on the fill-in:

Suppose that $S$ separates $G$ into two disconnected components $C_1$ and $C_2$. If the vertices in $S$ are labelled after $C_1$ and $C_2$ then from previous theorem there cannot be fill-edge between vertices of the disconnected components.

A first definition of the elimination tree

• A spanning tree of a connected graph $G$ is a subgraph $T$ of $G$ such that if there is a path in $G$ between $i$ and $j$ then there exists a path between $i$ and $j$ in $T$.

• Let $A$ be a symmetric positive-definite matrix, $A = LL^T$ its Cholesky factorization, and $G^+(A)$ its filled graph (graph of $F = L + L^T$).

**Definition 3.** The elimination tree of $A$ is a spanning tree of $G^+(A)$ satisfying the relation $PARENT[j] = \min\{i > j | l_{ij} \neq 0\}$.

**Graph structures**

Properties of the elimination tree

• Another perspective also leads to the elimination tree

• Dependency between columns of $L$:
  1. Column $i > j$ depends on column $j$ iff $l_{ij} \neq 0$
  2. Use a directed graph to express this dependency
  3. Simplify redundant dependencies (transitive reduction in graph theory)

• The transitive reduction of the directed filled graph gives the elimination tree structure
2 Ordering sparse matrices

2.1 Objectives/Outline

Ordering sparse matrices: objectives/outline

- Reduce fill-in and number of operations during factorization (local and global heuristics):
  - Increase parallelism (wide tree)
  - Decrease memory usage (deep tree)
  - Equivalent orderings: (Traverse tree to minimize working memory)
- Reorder unsymmetric matrices to special forms:
  - block upper triangular matrix:
  - with (large) non-zero entries on the diagonal (maximum transversal).
- Combining approaches

2.2 Fill-reducing orderings

Fill-reducing orderings

Three main classes of methods for minimizing fill-in during factorization

- Global approach: The matrix is permuted into a matrix with a given pattern
  - Fill-in is restricted to occur within that structure
  - Cuthill-McKee (block tridiagonal matrix)
  - Nested dissections (“block bordered” matrix).
- Local heuristics: At each step of the factorization, selection of the pivot that is likely to minimize fill-in.
  - Method is characterized by the way pivots are selected.
  - Markowitz criterion (for a general matrix).
  - Minimum degree (for symmetric matrices).
- Hybrid approaches: Once the matrix is permuted in order to obtain a block structure, local heuristics are used within the blocks.

Consider the matrix:
The corresponding graph is

\[
A = \begin{bmatrix}
  x & x & x & x \\
  x & x & x & x \\
  x & x & x & x \\
  x & x & x & x \\
  x & x & x & x \\
\end{bmatrix}
\]

Cuthill-McKee algorithm

- Goal: reduce the profile/bandwidth of the matrix
  (the fill is restricted to the band structure)

- Level sets (such as Breadth First Search) are built from the vertex of minimum degree (priority to the vertex of smallest number)

  We get: \( S_1 = \{2\}, S_2 = \{1\}, S_3 = \{4, 5\}, S_4 = \{3, 6\} \) and thus the ordering 2, 1, 4, 5, 3, 6.

The reordered matrix is:

\[
A = \begin{bmatrix}
  x & x & x & x \\
  x & x & x & x \\
  x & x & x & x \\
  x & x & x & x \\
  x & x & x & x \\
\end{bmatrix}
\]

Reverse Cuthill-McKee

- The ordering is the reverse of that obtained using Cuthill-McKee i.e. on the example \{6, 3, 5, 4, 1, 2\}

- The reordered matrix is:

\[
A = \begin{bmatrix}
  x & x & x \\
  x & x & x \\
  x & x & x \\
  x & x & x \\
  x & x & x \\
\end{bmatrix}
\]

- More efficient than Cuthill-McKee at reducing the envelop of the matrix.

Nested Dissection

Recursive approach based on graph partitioning.
Local heuristics to reduce fill-in during factorization

Let $G(A)$ be the graph associated to a matrix $A$ that we want to order using local heuristics. Let $\text{Metric}$ such that $\text{Metric}(v_i) < \text{Metric}(v_j)$ implies $v_i$ is a better than $v_j$.

Generic algorithm
Loop until all nodes are selected
   Step1: select current node $p$ (so called pivot) with minimum metric value,
   Step2: update elimination graph,
   Step3: update $\text{Metric}(v_j)$ for all non-selected nodes $v_j$.
$\text{Step3 should only be applied to nodes for which the Metric value might have changed.}$

Reordering unsymmetric matrices: Markowitz criterion

- At step $k$ of Gaussian elimination:

  $A_k \rightarrow L \rightarrow U$

  - $r_i^k =$ number of non-zeros in row $i$ of $A^k$
  - $c_j^k =$ number of non-zeros in column $j$ of $A^k$
  - $a_{kk}$ must be large enough and should minimize $(r_i^k - 1) \times (c_j^k - 1)$ $\forall i, j > k$

- **Minimum degree**: Markowitz criterion for symmetric diagonally dominant matrices

Minimum degree algorithm

- **Step 1**: Select the vertex that possesses the smallest number of neighbors in $G^0$.

\[ \begin{pmatrix}
  1 & X & X & X \\
  X & 2 & X & X \\
  X & X & 3 & X \\
  X & X & X & 4 \\
  X & X & X & X \\
  X & X & X & 5 \\
  X & X & X & X \\
  X & X & X & X \\
  X & X & X & 6 \\
\end{pmatrix} \]

(a) Sparse symmetric matrix

(b) Elimination graph

The node/variable selected is 1 of degree 2.

- **Notation for the elimination graph**
  - Let $G^k = (V^k, E^k)$ be the graph built at step $k$.
  - $G^k$ describes the structure of $A_k$ after eliminating $k$ pivots.
  - $G^k$ is non-oriented ($A_k$ is symmetric)
  - Fill-in in $A_k \equiv$ adding edges in the graph.
Illustration
Step 1: elimination of pivot 1

Minimum degree algorithm applied to the graph:
• Step \( k \): Select the node with the smallest number of neighbors
• \( G^k \) is built from \( G^{k-1} \) by suppressing the pivot and adding edges corresponding to fill-in.

Illustration (cont’d)
Graphs \( G_1, G_2, G_3 \) and corresponding reduced matrices.

Minimum Degree does not always minimize fill-in !!!
Consider the following matrix:

\[
\begin{bmatrix}
1 & x & x \\
x & 1 & x \\
x & x & 1 \\
x & x & x \\
x & x & x \\
x & x & x \\
\end{bmatrix}
\]

Remark: Using initial ordering
No fill-in
Corresponding elimination graph

Step 1 of Minimum Degree:
Select pivot 5 (minimum degree = 2)
Updated graph
Add (4,6) i.e. fill-in

Reduce time complexity
1. Accelerate selection of pivots and update of the graph:
   - 1.1 Supervariables: if several variables have the same adjacency structure in \(G^k\), they can be eliminated simultaneously.
   - 1.2 Two non-adjacent nodes of same degree can be eliminated simultaneously (multiple eliminations).
   - 1.3 Degree update of neighbours of the pivot can be effected in an approximate way (Approximate Minimum Degree).

Reduce memory complexity
2. Decrease size of working space: Using the elimination graph, working space is of order \(O(\#\text{nonzeros in factors})\).
   - Fill-in: Let pivot be the pivot at step \(k\)
     \[\text{If } i \in \text{Adj}_{G^{k-1}}(\text{pivot}) \text{ then } \text{Adj}_{G^{k-1}}(\text{pivot}) \subseteq \text{Adj}_{G^k}(i)\]
     Structure of pivot column included in filled structure of column \(i\).
   - We can then use an implicit representation of fill-in by defining the notion of element (variable already eliminated) and quotient graph. A variable of the quotient graph is adjacent to variables and elements.
   - One can show that \(\forall k \in [1 \ldots N]\), the size of the quotient graph is \(O(G^0)\)

Influence on the structure of factors
Harwell-Boeing matrix: dwt_592.rua, structural computing on a submarine. NZ(LU factors)=58202
Detection of *supervariables* allows to build more regularly structured factors (easier factorization).

**Comparison of 3 implementations of Minimum Degree**

- Let **V0** be the initial algorithm (based on the elimination graph)
- **MMD** the version including 1.1/ + 1.2/ + 2/ (Multiple Minimum Degree, Liu 1985, 1989), used in MATLAB
- **AMD** the version including 1.1/ + 1.3/ + 2/ (Approximate Minimum Degree, Amestoy, Davis, Duff 1995).

**Execution times (secs) on a SUN Sparc 10**

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Order</th>
<th>Nonzeros</th>
<th>Minimum Degree</th>
<th>Minimum Degree</th>
<th>Minimum Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>V0</td>
<td>MMD</td>
<td>AMD</td>
</tr>
<tr>
<td>dwt_2680</td>
<td>2680</td>
<td>13853</td>
<td>35</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>Min. memory size</td>
<td></td>
<td></td>
<td>250KB</td>
<td>110KB</td>
<td>110KB</td>
</tr>
<tr>
<td>Wang4</td>
<td>26068</td>
<td>75552</td>
<td>-</td>
<td>11</td>
<td>5</td>
</tr>
<tr>
<td>Orani678</td>
<td>2529</td>
<td>85426</td>
<td>-</td>
<td>125</td>
<td>5</td>
</tr>
</tbody>
</table>

- Fill-in is similar
- Memory space for MMD and AMD : ≈ 2 × NZ integers
- V0 was not able to perform reordering for the 2 last matrices (lack of memory after 2 hours of computations)

**Minimum fill-in heuristics**

Recalling the generic algorithm. Let $G(A)$ be the graph associated to a matrix $A$ that we want to order using local heuristics. Let $Metric$ be such that $Metric(v_i) < Metric(v_j) \equiv v_i$ is a better than $v_j$.

**Generic algorithm**

Loop until all nodes are selected

Step1: Select current node $p$ (so called pivot) with minimum metric value,
Step2: update elimination (or quotient) graph,
Step3: update $Metric(v_j)$ for all non-selected nodes $v_j$.

*Step3 should only be applied to nodes for which the Metric value might have changed.*
Minimum fill based algorithm

- Metric \( v_i \) is the amount of fill-in that \( v_i \) would introduce if it were selected as a pivot.
- Illustration: \( r \) has a degree \( d = 4 \) and a fill-in metric of \( d \times (d - 1)/2 = 6 \) whereas \( s \) has degree \( d = 5 \) but a fill-in metric of \( d \times (d - 1)/2 - 9 = 1 \).

Minimum fill-in properties

- The situation typically occurs when \( \{i_1, i_2, i_3\} \) and \( \{i_2, i_3, i_4, i_5\} \) were adjacent to two already selected nodes (here \( e_2 \) and \( e_1 \)).

- The elimination of a node \( v_k \) affects the degree of nodes adjacent to \( v_k \). The fill-in metric of \( \text{Adj} (\text{Adj}(v_k)) \) is also affected.
- Illustration: selecting \( r \) affects the fill-in metric of \( i_1 \) (because of fill edge \( (j_3, j_4) \)).

How to compute the fill-in metrics

*Computing the exact minimum fill-in metric is too costly*

- Only nodes adjacent to current pivot are updated.
- Only approximated metrics (using clique structures) are computed.
- Let \( d_k \) be the degree of node \( k \); \( d_k \times (d_k - 1)/2 \) is an upper bound of the fill \( (s \rightarrow d_s = 5 \rightarrow d_s \times (d_s - 1)/2 = 10) \).
- Several possibilities:
  1. Deduce the clique area of the "last" selected pivot adjacent to \( k \) \( (s \rightarrow \text{clique of } e_2) \).
  2. Deduce the largest clique area of all adjacent selected pivots \( (s \rightarrow \text{clique of } e_1) \).
  3. If for \( d_k \) we use instead AMD then cliques of all adjacent selected pivots can be deduced.

2.3 Impact of fill reduction algorithm on the shape of the tree

Impact of fill reduction on the shape of the tree
<table>
<thead>
<tr>
<th>Reordering technique</th>
<th>Shape of the tree</th>
<th>observations</th>
</tr>
</thead>
</table>
| AMD                  | ![deep tree]       | • Deep well-balanced  
                   |                   | • Large frontal matrices on top |
| AMF                  | ![unbalanced tree] | • Very deep unbalanced  
                   |                   | • Small frontal matrices |
| PORD                 | ![deep tree]       | • deep unbalanced  
                   |                   | • Small frontal matrices |
| SCOTCH               | ![wide tree]       | • Very wide well-balanced  
                   |                   | • Large frontal matrices |
| METIS                | ![wide tree]       | • Wide well-balanced  
                   |                   | • Smaller frontal matrices (than SCOTCH) |
Importance of the shape of the tree

Suppose that each node in the tree corresponds to a task that:
- consumes temporary data from the children,
- produces temporary data, that is passed to the parent node.

- Wide tree
  - Good parallelism
  - Many temporary blocks to store
  - Large memory usage

- Deep tree
  - Less parallelism
  - Smaller memory usage

2.4 Postorderings and memory usage

Trees, topological orderings and postorderings

- A rooted tree is a tree for which one node has been selected to be the root.
- A topological ordering of a rooted tree is an ordering that numbers children nodes before their parent.
- Postorderings are topological orderings which number nodes in any subtree consecutively.

Trees, topological orderings and postorderings

Postorderings and memory usage

- Assumptions:
  - Tree processed from the leaves to the root
  - Parents processed as soon as all children have completed (postorder of the tree)
  - Each node produces and sends temporary data consumed by its father.

- Exercise: In which sense is postordering-based tree traversal be more interesting than a random topological ordering?

- Furthermore, memory usage also depends on the postordering chosen:
Example 1: Processing a wide tree

Example 2: Processing a deep tree
Modelization of the problem

- $M_i$: memory peak for complete subtree rooted at $i$,
- $\text{temp}_i$: temporary memory produced by node $i$,
- $m_{\text{parent}}$: memory for storing the parent.

\[
M_{\text{parent}} = \max\left( \max_{j=1}^{\text{nbchildren}} \left( M_i + \sum_{k=1}^{j-1} \text{temp}_k \right), m_{\text{parent}} + \sum_{j=1}^{\text{nbchildren}} \text{temp}_j \right)
\]

Objective: order the children to minimize $M_{\text{parent}}$

Memory-minimizing schedules

Theorem 4. [Liu, 86] The minimum of $\max_j \left( x_j + \sum_{i=1}^{j-1} y_i \right)$ is obtained when the sequence $(x_i, y_i)$ is sorted in decreasing order of $x_i - y_i$.

Corollary 5. An optimal child sequence is obtained by rearranging the children nodes in decreasing order of $M_i - \text{temp}_i$.

Interpretation: At each level of the tree, child with relatively large peak of memory in its subtree ($M_i$ large with respect to $\text{temp}_i$) should be processed first.

⇒ Apply on complete tree starting from the leaves (or from the root with a recursive approach)
Optimal tree reordering

Objective: Minimize peak of stack memory

Tree_Reorder \((T)\):

\[
\begin{align*}
\text{Begin} \\
\text{for all } i \text{ in the set of root nodes do} \\
\quad \text{Process\_Node}(i); \\
\text{end for} \\
\text{End}
\end{align*}
\]

\(\text{Process\_Node}(i):\)

\[
\begin{align*}
\text{if } i \text{ is a leaf then} \\
\quad M_i = m_i \\
\text{else} \\
\quad \text{for } j = 1 \text{ to } \text{nbchildren do} \\
\quad \quad \text{Process\_Node}(j^{th} \text{ child}); \\
\quad \text{end for} \\
\quad \text{Reorder the children of } i \text{ in decreasing order of } (M_j - temp_j); \\
\quad \text{Compute } M_{\text{parent}} \text{ at node } i \text{ using Formula } 1; \\
\text{end if}
\end{align*}
\]

2.5 Equivalent orderings and elimination trees

Equivalent orderings of symmetric matrices

Let \(F\) be the filled matrix of a symmetric matrix \(A\) (that is, \(F = L + L^T\), where \(A = LL^T\) \(G^+(A)\) is the associated filled graph.

Definition 6 (Equivalent orderings). \(P\) and \(Q\) are said to be equivalent orderings iff \(G^+(PAP^T) = G^+(QAQ^T)\)

By extension, a permutation \(P\) is said to be an equivalent ordering of a matrix \(A\) iff \(G^+(PAP^T) = G^+(A)\)

It can be shown that an equivalent reordering also preserves the amount of arithmetic operations for sparse Cholesky factorization.

Relation with elimination trees

- Let \(A\) be a reordered matrix, and \(G^+(A)\) be its filled graph
- In the elimination tree, any tree traversal (that processes children before parents) corresponds to an equivalent ordering \(P\) of \(A\) and the elimination tree of \(PAP^T\) is identical to that of \(A\).

Tree rotations

Definition 7. An ordering that does not introduce any fill is referred to as a perfect ordering

Natural ordering is a perfect ordering of the filled matrix \(F\).
Theorem 8. For any node \( x \) of \( G^+(A) = G(F) \), there exists a perfect ordering on \( G(F) \) such that \( x \) is numbered last.

- Essence of tree rotations:
  - Nodes in the clique of \( x \) in \( F \) are numbered last
  - Relative ordering of other nodes is preserved.

Example of equivalent orderings

On the right hand side tree rotations applied on \( w \): (clique of \( w \) is \( \{w, x\} \) and for other nodes relative ordering w.r.t. tree on the left is preserved).

Remark: Tree rotations can help reducing the temporary memory usage!

2.6 Reorder unsymmetric matrices to special forms

Unsymmetric matrices and graphs

An unsymmetric matrix can be seen as a bipartite graph:

- \( G = (V_r, V_c, E \subset V_r \times V_c) \)
- \((r, c) \in E \) iff there is an entry at row \( r \) column \( c \).

or a directed graph (digraph):

- \( G = (V, E) \)
- There is an oriented edge from the source \( r \) to the target \( c \) \((r, c) \in E) \) iff there is an entry at row \( r \) column \( c \).

Maximum matching/transversal orderings

Let \( A \) be a sparse matrix and \( \mathcal{G} = (V_r, V_c, E \subset V_r \times V_c) \) its associated bipartite graph.
\( \mathcal{M} \subset E \) is a matching iff for all \((r_1, c_1), (r_2, c_2) \) in \( \mathcal{M} \) distinct, \( r_1 \neq r_2 \) and \( c_1 \neq c_2 \).

Structural problem:

- Maximum transversal: how to permute the columns of \( A \) to have the maximum number of non zero entries on the diagonal?
- Maximum matching: how to find a matching of maximum size?

dmperm in MATLAB, MC21 in fortran 77 (HSL library).
Components of the Maximum Matching algorithm

- Suppose that matching of size $k$ has been found (i.e. there exists permutations to place entries on the first $k$ diagonal positions).
- augmenting paths: try to find a matching of size $k + 1$ using a matching of size $k$,
- branch and bound techniques on depth first search.

Structurally singular matrices
If the matching is not maximal (cannot be extended) then the matrix is said to be structurally singular or symbolically singular. In the example Columns 1 and 6 are multiple of each other.
Maximum weighted matching
Structural+numerical problem:

- Maximum weighted transversal: how to find a permutation $P$ of $A$ columns such that the product (or another metric: min, sum . . . ) of the diagonal entries of $AP$ is maximum?

- Maximum weighted matching?

  Same techniques as in maximum matching, but branch and bound more complicated and it is more efficient to do breadth first search. MC64 in fortran 77 or 90 (HSL library).

Reduction to Block Triangular Form (BTF)

- Suppose that there exists permutations matrices $P$ and $Q$ such that

  $\begin{pmatrix}
  B_{11} & B_{21} & B_{31} & \cdots & B_{N1} \\
  B_{12} & B_{22} & B_{32} & \cdots & B_{N2} \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  B_{1N} & B_{2N} & B_{3N} & \cdots & B_{NN}
  \end{pmatrix}$

- If $N > 1$ $A$ is said to be reducible (irreducible otherwise)

- Each $B_{ii}$ is supposed to be irreducible (otherwise finer decomposition is possible)

- Advantage: to solve $Ax = b$ only $B_{ii}$ need be factored

  $B_{ii}x_i = (Pb)_i - \sum_{j=1}^{i-1} B_{ij}y_j$, $i = 1, \ldots, N$ with $y = Q^Tx$

A two-stage approach to compute the reduction

- Stage (1): compute a (column) permutation matrix $Q_1$ such that $AQ_1$ has non zeros on the diagonal (find a maximum transversal of $A$), then

- Stage(2): compute a (symmetric) permutation matrix $P$ such that $PAQ_1P^t$ is BTF.

The diagonal blocks of the BTF are uniquely defined. Techniques exits to directly compute the BTF form. They do not present advantage over this two stage approach.

Main components of the algorithm

- Objective : assume $AQ_1$ has non zeros on the diagonal and compute $P$ such that $PAQ_1P^t$ is BTF.

- Use of a digraph (directed graph) associated to the matrix.

- Symmetric permutations on digraph $\equiv$ relabelling nodes of the graph.

- If there is no closed path through all nodes in the digraph then the digraph can be subdivided into two parts.

- Strong components of a graph are the set of nodes belonging to a closed path.

- The strong components of the graph are the diagonal blocks $B_{ii}$ of the BTF format.
Preamble: Finding the triangular form of a matrix

- **Observations**: a triangular matrix has a BTF form with blocks of size 1 (BTF will be considered as generalization of the triangular form where each diagonal entry is now a strong component). Note that if the matrix has triangular form the digraph has no cycle (i.e. *acyclic*).

- **Sargent and Westerberg algorithm**:
  1. Select a starting node and trace a path until finding a node from which no path leaves.
  2. Number the last node first, eliminate it (and all edges pointing to it).
  3. Continue from previous node on the path (or choose a new starting node).

Generalizing Sargent and Westerberg algorithm to compute a BTF

A *composite node* is a set nodes belonging to a closed path.
Start from any node and follow a path until
(1) a closed path is found:
   - collapse all nodes in a closed path into a composite node
   - the path continues from the composite node.
(2) reaching a node or composite node from which no path leaves:
   - the node or composite node is numbered next.
Hall property

- Consider the bipartite representation of an unsymmetric matrix
- A bipartite graph with $m$ rows and $n$ columns has the **Hall property** if every set of $k$ column vertices is adjacent to at least $k$ row vertices for all $1 \leq k \leq n$.
- Hall theorem: A bipartite graph has a column matching iff it has the Hall property

Strong Hall property

- A bipartite graph with $m$ rows and $n$ columns has the **strong Hall property** if every set of $k$ column vertices is adjacent to at least $k + 1$ row vertices for all $1 \leq k < n$.
- Any matrix that is not strong Hall can be permuted in block lower triangular form.

2.7 Combining approaches

Example (1) of hybrid approach

- Top-down followed by bottom-up processing of the graph: Top-down: Apply nested dissection (ND) on complete graph Bottom-up: Local heuristic on each subgraph
- Generally better for large-scale irregular problems than
  - pure nested dissection
  - local heuristics
Example (2) : combine maximum transversal and fill-in reduction

- Consider the LU factorization $A = LU$ of an unsymmetric matrix.
- Compute the column permutation $Q$ leading to a maximum numerical transversal of $A$. $AQ$ has large (in some sense) numerical entries on the diagonal.
- Find best ordering of $AQ$ preserving the diagonal entries. Equivalent to finding symmetric permutation $P$ such that the factorization of $PAQP^T$ has reduced fill-in.

3 Factorization of sparse matrices

Outline

1. Introduction
2. Elimination tree and multifrontal method
3. Comparison between multifrontal, frontal and general approaches for LU factorization
4. Task mapping and scheduling
5. Distributed memory approaches: fan-in, fan-out, multifrontal
6. Some parallel solvers; case study on MUMPS and SuperLU.
7. Concluding remarks

3.1 Introduction

Recalling the Gaussian elimination

Step $k$ of LU factorization ($a_{kk}$ pivot):

- For $i > k$ compute $l_{ik} = a_{ik}/a_{kk}$ ($= a'_{ik}$).
- For $i > k, j > k$ such that $a_{ik}$ and $a_{kj}$ are nonzeros
  \[
  a'_{ij} = a_{ij} - \frac{a_{ik} \times a_{kj}}{a_{kk}}
  \]
• If \( a_{ik} \neq 0 \) et \( a_{kj} \neq 0 \) then \( a'_{ij} \neq 0 \)
• If \( a_{ij} \) was zero \( \rightarrow \) its non-zero value must be stored
• Orderings (minimum degree, Cuthill-McKee, ND) limit fill-in, the number of operations and modify the tasks graph

Three-phase scheme to solve \( Ax = b \)

1. Analysis step
   • Preprocessing of \( A \) (symmetric/unsymmetric orderings, scalings)
   • Build the dependency graph (elimination tree, eDAG ...)

2. Factorization \((A = LU, LDL^T, LL^T, QR)\)
   Numerical pivoting

3. Solution based on factored matrices
   • triangular solves: \( Ly = b, \text{ then } Ux = y \)
   • improvement of solution (iterative refinement), error analysis

Control of numerical stability: numerical pivoting

• In dense linear algebra partial pivoting commonly used (at each step the largest entry in the column is selected).

• In sparse linear algebra, flexibility to preserve sparsity is offered :
  - Partial threshold pivoting : Eligible pivots are not too small with respect to the maximum in the column. Set of eligible pivots = \( \{ r \mid |a_{rk}^{(k)}| \geq u \times \max_i |a_{ik}^{(k)}| \} \), where \( 0 < u \leq 1 \).
  - Then among eligible pivots select one preserving better sparsity.
  - \( u \) is called the threshold parameter \((u = 1 \rightarrow \text{partial pivoting})\).
  - It restricts the maximum possible growth of: \( a_{ij} = a_{ij} - \frac{a_{ik} \times a_{kj}}{a_{kk}} \)
  - \( u \approx 0.1 \) is often chosen in practice.

• Symmetric indefinite case: requires 2 by 2 pivots, e.g. \( \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \)

Threshold pivoting and numerical accuracy

Table 2: Effect of variation in threshold parameter \( u \) on a 541 \( \times \) 541 matrix with 4285 nonzeros (Dongarra et al 91).

<table>
<thead>
<tr>
<th>( u )</th>
<th>Nonzeros in LU factors</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>16767</td>
<td>( 3 \times 10^{-9} )</td>
</tr>
<tr>
<td>0.25</td>
<td>14249</td>
<td>( 6 \times 10^{-10} )</td>
</tr>
<tr>
<td>0.1</td>
<td>13660</td>
<td>( 4 \times 10^{-9} )</td>
</tr>
<tr>
<td>0.01</td>
<td>15045</td>
<td>( 1 \times 10^{-5} )</td>
</tr>
<tr>
<td>( 10^{-4} )</td>
<td>16198</td>
<td>( 1 \times 10^{2} )</td>
</tr>
<tr>
<td>( 10^{-10} )</td>
<td>16553</td>
<td>( 3 \times 10^{33} )</td>
</tr>
</tbody>
</table>
Iterative refinement for linear systems
Suppose that a solver has computed $A = LU$ (or $LDL^T$ or $LL^T$), and a solution $\tilde{x}$ to $Ax = b$.

1. Compute $r = b - A\tilde{x}$.
2. Solve $LU\delta x = r$.
3. Update $\tilde{x} = \tilde{x} + \delta x$.
4. Repeat if necessary/useful.

### 3.2 Elimination tree and Multifrontal approach

#### Elimination tree and Multifrontal approach

We recall that:
- The elimination tree expresses dependencies between the various steps of the factorization.
- It also exhibits parallelism arising from the sparse structure of the matrix.

#### Building the elimination tree

- Permute matrix (to reduce fill-in) $PAP^T$.
- Build filled matrix $A_F = L + L^T$ where $PAP^T = LL^T$.
- Transitive reduction of associated filled graph

$→$ Each column corresponds to a node of the graph. Each node $k$ of the tree corresponds to the factorization of a frontal matrix whose row structure is that of column $k$ of $A_F$.

#### Illustration of multifrontal factorization

We assume pivot are chosen down the diagonal in order.

$$
\begin{bmatrix}
  \times & \times & \times \\
  \times & \times & \times \\
  \times & \times & \times \\
  \times & \times & \times \\
\end{bmatrix}
$$

Filled matrix

Elimination graph

Treatment at each node:
- Assembly of the frontal matrix using the contributions from the sons.
- Gaussian elimination on the frontal matrix

- Elimination of variable 1 ($a_{11}$ pivot)
  - Assembly of the frontal matrix
    $$
    \begin{bmatrix}
      1 & 2 & 4 \\
      1 & 2 & 4 \\
      2 & z & z \\
      4 & z & z \\
    \end{bmatrix}
    $$
  - Contributions: $a_{ij} = \frac{(a_{ij} \times a_{11})}{a_{11}}$ for $i > 1, j > 1$ on $a_{33}, a_{44}, a_{34}$ and $a_{43}$:
\[ a_{33}^{(1)} = - \frac{(a_{31} \times a_{13})}{a_{11}} \quad a_{34}^{(1)} = - \frac{(a_{31} \times a_{14})}{a_{11}} \]
\[ a_{43}^{(1)} = - \frac{(a_{41} \times a_{13})}{a_{11}} \quad a_{44}^{(1)} = - \frac{(a_{41} \times a_{14})}{a_{11}} \]

Terms \(- \frac{a_{ij} \times a_{jk}}{a_{kl}}\) of the contribution matrix are stored for later updates.

- Elimination of variable 2 \((a_{22} \text{ pivot})\)
  - Assembly of frontal matrix: update of elements of pivot row and column using contributions from previous updates (none here)
    
    \[
    \begin{align*}
    a_{33}^{(2)} &= - \frac{(a_{32} \times a_{23})}{a_{22}} \\
    a_{34}^{(2)} &= - \frac{(a_{32} \times a_{24})}{a_{22}} \\
    a_{43}^{(2)} &= - \frac{(a_{42} \times a_{23})}{a_{22}} \\
    a_{44}^{(2)} &= - \frac{(a_{42} \times a_{24})}{a_{22}}
    \end{align*}
    \]
  - Contributions on \(a_{33}, a_{34}, a_{43},\) and \(a_{44}\).

- Elimination of variable 3.
  - Assembly of frontal matrix
    Update using the previous contributions:
    
    \[
    \begin{align*}
    a_{33}' &= a_{33} + a_{33}^{(1)} + a_{33}^{(2)} \\
    a_{34}' &= a_{34} + a_{34}^{(1)} + a_{34}^{(2)} \quad (a_{34} = 0) \\
    a_{43}' &= a_{43} + a_{43}^{(1)} + a_{43}^{(2)} \quad (a_{43} = 0) \\
    a_{44}' &= a_{44}^{(1)} + a_{44}^{(2)}
    \end{align*}
    \]

    stored as a so called contribution matrix.

Note that \(a_{44}\) is partially summed since contributions are transferred only between son and father.

- Contribution on \(a_{44}\) : \(a_{44}^{(3)} = a_{44}' - \frac{(a_{43}' \times a_{34}')}{a_{43}}\)
- Elimination of variable 4
  - Frontal involves only \(a_{44} : a_{44} = a_{44} + a_{44}^{(3)}\)

The multifrontal method (Duff, Reid’83)

Memory is divided into two parts (that can overlap in time):

- the factors
- the active memory

Elimination tree represents tasks dependencies
Multifrontal method

AT EACH NODE

\[
\begin{array}{c|c}
F_{11} & F_{12} \\
F_{12}^T & F_{22}
\end{array}
\]

\[F_{22} \leftarrow F_{22} - F_{12}^T F^{-1}_{11} F_{12}\]

Pivot can ONLY be chosen from \(F_{11}\) block since \(F_{22}\) is NOT fully summed

Supernodal methods

Definition 9. A supernode (or supervariable) is a set of contiguous columns in the factors \(L\) that share essentially the same sparsity structure.

- All algorithms (ordering, symbolic factor., factor., solve) generalize to blocked versions.
- Use of efficient matrix-matrix kernels (improve cache usage).
- Same concept as supervariables for elimination tree/minimum degree ordering.
- Supernodes and pivoting: pivoting inside a supernode does not increase fill-in.
Amalgamation

- **Goal**
  - Exploit a more regular structure in the original matrix
  - Decrease the amount of indirect addressing
  - Increase the size of frontal matrices

- **How?**
  - Relax the number of nonzeros of the matrix
  - Amalgamation of nodes of the elimination tree

- **Consequences?**
  - Increase in the total amount of flops
  - But decrease of indirect addressing
  - And increase in performance

**Remark**: If $i_1, i_2, \ldots i_f$ is a son of node $j_1, j_2, \ldots j_p$ and if $\{j_1, j_2, \ldots j_p\} \subset \{i_1, i_2, \ldots i_f\}$ then the amalgamation of $i_1$ and $j_1$ is without fill-in

Amalgamation of supernodes (same lower diagonal structure) is without fill-in

**Illustration of amalgamation**

<table>
<thead>
<tr>
<th>Original Matrix</th>
<th>Elimination tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6 (Root)</td>
</tr>
<tr>
<td>2</td>
<td>5, 4</td>
</tr>
<tr>
<td>3</td>
<td>3, 5, 6</td>
</tr>
<tr>
<td>4</td>
<td>2, 4, 5</td>
</tr>
<tr>
<td>5</td>
<td>4, 5, 6</td>
</tr>
<tr>
<td>6</td>
<td>1, 4</td>
</tr>
</tbody>
</table>

Structure of node $i =$ frontal matrix noted $i_1, i_2, \ldots i_f$

**Illustration of amalgamation**

Amalgamation and Supernovariables

*Amalgamation of supernovariables does not cause fill-in*

**Initial Graph:**

Reordering: 1, 3, 4, 2, 6, 8, 10, 11, 5, 7, 9, 12, 13  **Supervariables**: {1, 3, 4} ; {2, 6, 8} ; {10, 11} ; {5, 7, 9, 12, 13}
Parallelization: two levels of parallelism

- Arising from sparsity: between nodes of the elimination tree
  
  first level of parallelism

- Within each node: parallel dense \( LU \) factorization (BLAS)

  second level of parallelism

Exploiting the second level of parallelism is crucial

<table>
<thead>
<tr>
<th>Computer</th>
<th>#procs</th>
<th>MFlops</th>
<th>speed-up</th>
<th>MFlops</th>
<th>speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alliant FX/80</td>
<td>8</td>
<td>15</td>
<td>(1.9)</td>
<td>34</td>
<td>(4.3)</td>
</tr>
<tr>
<td>IBM 3090J/6VF</td>
<td>6</td>
<td>126</td>
<td>(2.1)</td>
<td>227</td>
<td>(3.8)</td>
</tr>
<tr>
<td>CRAY-2</td>
<td>4</td>
<td>316</td>
<td>(1.8)</td>
<td>404</td>
<td>(2.3)</td>
</tr>
<tr>
<td>CRAY Y-MP</td>
<td>6</td>
<td>529</td>
<td>(2.3)</td>
<td>1119</td>
<td>(4.8)</td>
</tr>
</tbody>
</table>

Performance summary of the multifrontal factorization on matrix BCSSTK15. In column (1), we exploit only parallelism from the tree. In column (2), we combine the two levels of parallelism.

Other features

- Dynamic management of parallelism:
  - Pool of tasks for exploiting the two levels of parallelism

- Assembly operations also parallel (but indirect addressing)
• Dynamic management of data
  – Storage of LU factors, frontal and contribution matrices
  – Amount of memory available may conflict with exploiting maximum parallelism

3.3 Comparison between 3 approaches for LU factorization

We compare three general approaches for sparse LU factorization

• General technique
• Frontal method
• Multifrontal approach

*distributed memory multifrontal and supernodal approaches will be compared in another section*

Description of the 3 approaches

• General technique
  – Numerical and sparsity pivoting performed at the same time
  – Dynamic sparse data structures
  – Good preservation of sparsity: local decision influenced by numerical choices.

• Frontal method
  – Extension of band or variable-band schemes
  – No indirect addressing is required in the innermost loop (data are stored in dense matrices)
  – Simple data structure, fast methods, easier to implement.
  – Very popular, easy out-of-core implementation.
  – Sequential by nature

Description of the 3 approaches

• Multifrontal approach
  – Can be seen as an extension of frontal method
  – Analysis phase to compute an ordering
  – Ordering can then be perturbed by numerical pivoting
  – Full matrices are used in the innermost loops. *Compared to frontal schemes:* -complex to implement (assembly of dense matrices, managament of numerical pivoting) -Preserve in a better way the sparse structure.
Frontal method

\[
A = \begin{pmatrix}
& & & & a_{11} & & \\
& & & & & & \\
& & & & & & \\
& & & & & & \\
& & & & & & \\
& & & & & & \\
& & & & & & \\
& & & & & & \\
& & & & & & \\
\end{pmatrix}
\]

Properties: The band is treated as full $\Rightarrow$ Efficient reordering to minimize bandwidth is crucial.

Frontal vs Multifrontal methods

Characteristics of multifrontal method:
- More complex data structures.
- Usually more efficient for preserving sparsity than frontal techniques
- Parallelism arising from sparsity.

Illustration: comparison between 3 software for LU

- **General approach** (MA38: Davis and Duff):
  - Control of fill-in: Markowitz criterion
  - Numerical Stability: partial pivoting
  - Numerical and sparsity pivoting are performed in one step
- **Multifrontal method** (MA41, Amestoy and Duff):
- Reordering before numerical factorization
- Minimum degree type of reordering
- Partial pivoting for numerical stability

- **Frontal method** (MA42, Duff and Scott):
  - Reordering before numerical factorization
  - Reordering for decreasing bandwidth
  - Partial pivoting for numerical stability

- All these software (MA38, MA41, MA42) are available in HSL-Library.

### Test problems from Harwell-Boeing and Tim Davis (Univ. Florida) collections.

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Order</th>
<th>Nb of nonzeros</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orani678.rua</td>
<td>2526</td>
<td>90158</td>
<td>Economic modelling</td>
</tr>
<tr>
<td>Onetone1.rua</td>
<td>36057</td>
<td>341088</td>
<td>Harmonic balance method</td>
</tr>
<tr>
<td>Garon2.rua</td>
<td>13535</td>
<td>390607</td>
<td>2D Navier-Stokes</td>
</tr>
<tr>
<td>Wang3.rua</td>
<td>26064</td>
<td>177168</td>
<td>3D Simulation of semiconductor</td>
</tr>
<tr>
<td>mhda416.rua</td>
<td>416</td>
<td>8562</td>
<td>Spectral problem in Hydrodynamic</td>
</tr>
<tr>
<td>rim.rua</td>
<td>22560</td>
<td>1014951</td>
<td>CFD nonlinear problem</td>
</tr>
</tbody>
</table>

### Execution times on a SUN

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Method</th>
<th>FLops</th>
<th>Size of factors ($\times 10^6$ words)</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Onetone1.rua</td>
<td>General</td>
<td>2</td>
<td>5</td>
<td>59</td>
</tr>
<tr>
<td></td>
<td>Frontal</td>
<td>19</td>
<td>115</td>
<td>6392</td>
</tr>
<tr>
<td></td>
<td>Multifrontal</td>
<td>8</td>
<td>10</td>
<td>193</td>
</tr>
<tr>
<td>Garon2.rua</td>
<td>General</td>
<td>40</td>
<td>8</td>
<td>95</td>
</tr>
<tr>
<td></td>
<td>Frontal</td>
<td>20</td>
<td>9</td>
<td>86</td>
</tr>
<tr>
<td></td>
<td>Multifrontal</td>
<td>4</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>mhda416.rua</td>
<td>General</td>
<td>24</td>
<td>0.16</td>
<td>0.80</td>
</tr>
<tr>
<td></td>
<td>Frontal</td>
<td>3</td>
<td>0.02</td>
<td>0.07</td>
</tr>
<tr>
<td></td>
<td>Multifrontal</td>
<td>16</td>
<td>0.04</td>
<td>0.11</td>
</tr>
</tbody>
</table>

### 3.4 Task mapping and scheduling

- Affect tasks to processors to achieve a goal: makespan minimization, memory minimization, ...
- many approaches:
  - **static**: Build the schedule before the execution and follow it at run-time
  - **Advantage**: very efficient since it has a global view of the system
• **Drawback:** Requires a very-good modelization of the platform
  – *dynamic*: Take scheduling decisions dynamically at run-time

• **Advantage:** Reactive to the evolution of the platform and easy to use on several platforms

• **Drawback:** Decisions taken with local criteria (a decision which seems to be good at time \( t \) can have very bad consequences at time \( t + 1 \))

**Influence of scheduling on the makespan**

**Objective:** Assign process/tasks to processors so that the completion time, also called the makespan is minimized. (We may also say that we minimize the maximum total processing time on any processor.)

**Task scheduling on shared memory computers**

The data can be shared between processors without any communication.

- Dynamic scheduling of the tasks (pool of “ready” tasks).
- Each processor selects a task (order can influence the performance).
- Example of “good” topological ordering (w.r.t time).

![Diagram of task scheduling](image)

Ordering not so good in terms of working memory.

**Static scheduling: Subtree to subcube (or proportional) mapping**

**Main objective:** reduce the volume of communication between processors.

- Recursively partition the processors “equally” between children of a given node.
- Initially all processors are assigned to root node.
- Good at localizing communication but not so easy if no overlapping between processor partitions at each step.

![Diagram of static scheduling](image)

**Mapping of the tasks onto the 5 processors**

**Objective:** Find a layer \( L_0 \) such that subtrees of \( L_0 \) can be mapped onto the processor with a good balance.

**Construction and mapping of the initial level \( L_0 \)**

**Begin**

- Let \( L_0 \leftarrow \) Roots of the assembly tree

**repeat**
Find the node \( q \) in \( L_0 \) whose subtree has largest computational cost
Set \( L_0 \leftarrow (L_0 \setminus \{ q \}) \cup \{ \text{children of } q \} \) (See Figure ??)
Greedy mapping of the nodes of \( L_0 \) onto the processors
Estimate the load unbalance
until load unbalance < threshold
End

**Decomposition of the tree into levels**
- Determination of Level \( L_0 \) based on subtree cost.

- Mapping of top of the tree can be dynamic.
- Could be useful for both shared and distributed memory algo.

### 3.5 Distributed memory approaches

#### Computational strategies for parallel direct solvers
- The parallel algorithm is characterized by:
  - Computational graph dependency
  - Communication graph
- Three classical approaches
  1. “Fan-in”
  2. “Fan-out”
  3. “Multifrontal”

**Preamble: left and right looking approaches for Cholesky factorization**
- \( cmod(j,k) \) : Modification of column \( j \) by column \( k \), \( k < j \),
- \( cdiv(j) \) division of column \( j \) by a scalar

```plaintext
[Left-looking approach]
for \( j = 1 \) to \( n \) do
  for \( k \in \text{Struct}(\text{row } L_{j+}) \) do
    \( cmod(j,k) \)
  \( cdiv(j) \)

[Right-looking approach]
for \( k = 1 \) to \( n \) do
  \( cdiv(k) \)
  for \( j \in \text{Struct}(\text{col } L_{+,k}) \) do
    \( cmod(j,k) \)
```
Illustration of Left and right looking

Left−looking  Right−looking

Assumptions and Notations

- **Assumptions**:
  - We assume that each column of \( L \) / each node of the tree is assigned to a single processor.
  - Each processor is in charge of computing \( cdiv(j) \) for columns \( j \) that it owns.

- **Notations**:
  - \( mycols(p) \) = is the set of columns owned by processor \( p \).
  - \( map(j) \) gives the processor owning column \( j \) (or task \( j \)).
  - \( procs(L_{*k}) = \{map(j) \mid j \in Struct(L_{*k})\} \) (only processors in \( procs(L_{*k}) \) require updates from column \( k \) – they correspond to ancestors of \( k \) in the tree).

Fan-in variant (similar to left looking)

Demand driven algorithm: data required are aggregated update columns computed by sending processor

```
Fan-in \( (p) \)
for \( j = 1 \) to \( p \) do
  \( u = 0 \)
  for all \( k \in Struct(row L_{j,*}) \cap mycols(p) \) do
    \( u = u + cmod(j,k) \)
  end for
  if \( map(j) \neq p \) then
    Send \( u \) to processor \( map(j) \)
  end if
  if \( map(j) = p \) then
    Incorporate \( u \) in column \( j \)
    Receive all necessary updated aggregates on column \( j \) and incorporate them in column \( j \)
    \( cdiv(j) \)
  end if
end for
```
if $map(1) = map(2) = map(3) = p$ and $map(4) \neq p$ (only) one message sent by $p$ to update column 4 → exploits data locality of proportional mapping.

**Fan-out variant (similar to right-looking)**

*Data driven algorithm.* Fan-out($p$):  
  
  for all leaf node $j \in mycols(p)$ do  
    $cdiv(j)$  
    send column $L_{s_j}$ to procs(col $L_{s_j}$)  
    $mycols(p) = mycols(p) - \{j\}$  
  end for  
  
  while $mycols(p) \neq \emptyset$ do  
    Receive any column (say $L_{s_k}$) of $L$  
    for $j \in Struct(col$ $L_{s_k}) \cap mycols(p)$ do  
      $cmod(j, k)$  
      if column $j$ completely updated then  
        $cdiv(j)$  
        send column $L_{s_j}$  
        $mycols(p) = mycols(p) - \{j\}$  
      end if  
    end for  
  end while
if map(2) = map(3) = p and map(4) ≠ p then 2 messages (for column 2 and 3) are sent by p to update column 4.

Fan-out variant

Properties of fan-out:

- Historically the first implemented.
- Incurs greater interprocessor communications than fan-in (or multifrontal) approach both in terms of
  - total number of messages
  - total volume
- Does not exploit data locality of proportional mapping.
- Improved algorithm (local aggregation):
  - send aggregated update columns instead of individual factor columns for columns mapped on a single processor.
  - Improve exploitation of data locality of proportional mapping.
  - But memory increase to store aggregates can be critical (as in fan-in).

Multifrontal variant

Elimination tree

Multifrontal Method

Algorithm:
For k=1 to n do
  Partial factorisation
  Send Contribution Block to Father
Endfor

3.6 Some parallel solvers

Shared memory sparse direct codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Technique</th>
<th>Scope</th>
<th>Availability (<a href="http://www">www</a>.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MA41</td>
<td>Multifrontal</td>
<td>UNS</td>
<td>cse.clrc.ac.uk/Activity/HSL</td>
</tr>
<tr>
<td>MA49</td>
<td>Multifrontal QR</td>
<td>RECT</td>
<td>cse.clrc.ac.uk/Activity/HSL</td>
</tr>
<tr>
<td>PanelLLT</td>
<td>Left-looking</td>
<td>SPD</td>
<td>Ng</td>
</tr>
<tr>
<td>PARDISO</td>
<td>Left-looking</td>
<td>UNS</td>
<td>Schenk</td>
</tr>
<tr>
<td>PSL†</td>
<td>Left-looking</td>
<td>SPD/UNS</td>
<td>SGI product</td>
</tr>
<tr>
<td>SPOOLES</td>
<td>Fun-in</td>
<td>SYM/UNS</td>
<td>netlib.org/linalg/spoole</td>
</tr>
<tr>
<td>SuperLU</td>
<td>Left-looking</td>
<td>UNS</td>
<td>nersc.gov/~xiaoye/SuperLU</td>
</tr>
<tr>
<td>WSMP‡</td>
<td>Multifrontal</td>
<td>SYM/UNS</td>
<td>IBM product</td>
</tr>
</tbody>
</table>

† Only object code for SGI is available
Distributed-memory sparse direct codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Technique</th>
<th>Scope</th>
<th>Availability (<a href="http://www">www</a>.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAPSS</td>
<td>Multifrontal LU</td>
<td>SPD</td>
<td>netlib.org/scalapack</td>
</tr>
<tr>
<td>MUMPS</td>
<td>Multifrontal</td>
<td>SYM/UNS</td>
<td>graal.ens-lyon.fr/MUMPS</td>
</tr>
<tr>
<td>PaStIX</td>
<td>Fan-in</td>
<td>SPD</td>
<td>see caption§</td>
</tr>
<tr>
<td>PSPASES</td>
<td>Multifrontal</td>
<td>SPD</td>
<td>cs.umn.edu/~mjoshi/pspases</td>
</tr>
<tr>
<td>SPOOLES</td>
<td>Fan-in</td>
<td>SYM/UNS</td>
<td>netlib.org/linalg/spooles</td>
</tr>
<tr>
<td>SuperLU</td>
<td>Fan-out</td>
<td>UNS</td>
<td>nersc.gov/~xiaoye/SuperLU</td>
</tr>
<tr>
<td>S+</td>
<td>Fan-out†</td>
<td>UNS</td>
<td>cs.ucsb.edu/research/S+</td>
</tr>
<tr>
<td>WSMP</td>
<td>Multifrontal</td>
<td>SYM</td>
<td>IBM product</td>
</tr>
</tbody>
</table>

§ dept-info.labri.u-bordeaux.fr/~ramet/pastix  † Only object code for IBM is available. No numerical pivoting performed.

Case study: Comparison of MUMPS and SuperLU

3.7 Case study: comparison of MUMPS and SuperLU

MUMPS (Multifrontal sparse solver) Amestoy, Duff, Guermouche, Koster, L’Excellent, Pralet

1. **Analysis and Preprocessing**
   - Preprocessing (max. transversal, scaling)
   - Fill-in reduction on $A + A^T$
   - Partial static mapping (elimination tree)

2. **Factorization**
   - Multifrontal (elimination tree of $A + A^T$)
   - Partial threshold pivoting
   - Node parallelism
     - Partitioning (1D Front - 2D Root)
     - Dynamic distributed scheduling

3. **Solution step and iterative refinement**

   *Features*: Real/complex Symmetric/Unsymmetric matrices; Distributed input; Assembled/Elemental format; Schur complement; multiple sparse right-hand-sides;

SuperLU (Gaussian elimination with static pivoting) X.S. Li and J.W. Demmel

1. **Analysis and Preprocessing**
   - Preprocessing (Max. transversal, scaling)
   - Fill-in reduction on $A + A^T$
   - Static mapping on a 2D grid processes

2. **Factorization**
   - Fan-out (elimination DAGs)
   - Static pivoting
     - if $|a_{ii}| < \sqrt{\varepsilon} \|A\|$ set $a_{ii}$ to $\sqrt{\varepsilon} \|A\|$ 
   - 2D irregular block cyclic partitioning (based on supernode structure)
   - Pipelining / BLAS3 based factorization

3. **Solution step and iterative refinement**

   *Features*: Real and complex matrices; Multiple right-hand-sides.
MUMPS: dynamic scheduling

Graph of tasks = tree Each task = partial factorization of a dense matrix Some parallel tasks mapped at runtime (80%)

1D pipelined factorization
P3 and P0 chosen by P2 at runtime

2D static decomposition

Node level parallelism in multifrontal solver MUMPS: pipelined factorization

SuperLU: 2D block cyclic layout and data structures

Test problems
StrSym : structural symmetry; R.-B.: Rutherford-Boeing set.

Impact of preprocessing and numerical issues

- **Objective**: Maximize diagonal entries of permuted matrix
- **MC64** (Harwell Sub. Lib.) code from Duff and Koster (1999)
  - Unsymmetric permutation (maximum weighted matching) and scaling
### Real Unsymmetric Assembled

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Order</th>
<th>NZ</th>
<th>StrSym</th>
<th>Origin</th>
</tr>
</thead>
<tbody>
<tr>
<td>bbmat</td>
<td>38744</td>
<td>1771722</td>
<td>54</td>
<td>R.-B. (CFD)</td>
</tr>
<tr>
<td>ECL32</td>
<td>51093</td>
<td>380415</td>
<td>93</td>
<td>EECS Dept. UC Berkeley</td>
</tr>
<tr>
<td>INVESTR1</td>
<td>30412</td>
<td>1793881</td>
<td>97</td>
<td>Parasol (Polyflow)</td>
</tr>
<tr>
<td>FIDAPM11</td>
<td>22294</td>
<td>623554</td>
<td>99</td>
<td>SPARSKIT2 (CFD)</td>
</tr>
<tr>
<td>GARON2</td>
<td>13535</td>
<td>390607</td>
<td>100</td>
<td>Davis (CFD)</td>
</tr>
<tr>
<td>LHR71C</td>
<td>70304</td>
<td>1528092</td>
<td>0</td>
<td>Davis (Chem Eng)</td>
</tr>
<tr>
<td>LN3P3937</td>
<td>3937</td>
<td>25407</td>
<td>87</td>
<td>R.-B. (CFD)</td>
</tr>
<tr>
<td>MIXTANK</td>
<td>29957</td>
<td>1995041</td>
<td>100</td>
<td>Parasol (Polyflow)</td>
</tr>
<tr>
<td>RMA1010</td>
<td>46835</td>
<td>2374001</td>
<td>98</td>
<td>Davis (CFD)</td>
</tr>
<tr>
<td>MIXTANK</td>
<td>29957</td>
<td>1995041</td>
<td>100</td>
<td>Parasol (Polyflow)</td>
</tr>
<tr>
<td>TWOTONE</td>
<td>120750</td>
<td>1224224</td>
<td>14</td>
<td>R.-B. (circuit sim)</td>
</tr>
</tbody>
</table>

### Real Symmetric Assembled (RSA)

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Order</th>
<th>NZ</th>
<th>StrSym</th>
<th>Origin</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMWCRA</td>
<td>148170</td>
<td>5396386</td>
<td>100</td>
<td>Parasol (MSC.Software)</td>
</tr>
<tr>
<td>CRANKSG2</td>
<td>63838</td>
<td>7106348</td>
<td>100</td>
<td>Parasol (MSC Software)</td>
</tr>
<tr>
<td>INLINE</td>
<td>503712</td>
<td>18660027</td>
<td>100</td>
<td>Parasol (MSC Software)</td>
</tr>
</tbody>
</table>

- Preprocessed matrix $B = D_1AQD_2$ is such that $|b_{ii}| = 1$ and $|b_{ij}| \leq 1$

- **Expectations**:
  - **MUMPS**: reduce NB of off-diagonal pivots and postponed var. (reduce numerical fill-in)
  - **SuperLU**: reduce NB of modified diagonal entries
  - Improve accuracy.

#### MC64 and Flops ($10^9$) for factorization (and ordering)

<table>
<thead>
<tr>
<th>Matrix</th>
<th>MC64</th>
<th>StrSym</th>
<th>MUMPS</th>
<th>SuperLU</th>
</tr>
</thead>
<tbody>
<tr>
<td>LHR71C</td>
<td>No</td>
<td>0</td>
<td>1431.0(*)</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>21</td>
<td>1.4</td>
<td>0.5</td>
</tr>
<tr>
<td>TWOTONE</td>
<td>No</td>
<td>28</td>
<td>1221.1</td>
<td>159.0</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>43</td>
<td>29.3</td>
<td>8.0</td>
</tr>
<tr>
<td>FIDAPM11</td>
<td>No</td>
<td>100</td>
<td>9.7</td>
<td>8.9</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>29</td>
<td>28.5</td>
<td>22.0</td>
</tr>
</tbody>
</table>

(*) Estimated during analysis, – Not enough memory to run the factorization.

**Backward error analysis**: $Berr = \max_i \frac{|r_i|}{\|A\| \cdot \|x\| + \|b\|}$

**Factorization cost study**

- **SuperLU** preserves/exploits better the sparsity/asymmetry than **MUMPS**. This results in
  ++ smaller size of factors (less memory)

---

47
++ fewer operations
++ more independency/parallelism
−− Extra cost of taking into account asymmetry
−− Smaller block-size for BLAS-3 kernels

Cost of preserving sparsity  (time on T3E 4 Procs)

Nested Dissection versus Minimum Degree orderings  (time on T3E 4 Procs)

Communication issues

Time Ratios of the numerical phases  $\frac{\text{Time(SuperLU)}}{\text{Time(MUMPS)}}$
Performance analysis on 3-D grid problems

Rectangular grids - Nested Dissection ordering

Summary

- **Sparsity and Total memory**
  - SuperLU preserves better sparsity
  - SuperLU (≈ 20%) less memory on 64 Procs (Asymmetry - Fan-out/Multifrontal)

- **Communication**
  - Global volume is comparable
  - MUMPS: much smaller (/10) nb of messages

- **Factorization / Solve time**
  - MUMPS is faster on NPROCS ≤ 64
  - SuperLU is more scalable

- **Accuracy**
  - MUMPS provides a better initial solution
  - SuperLU: one step of iter. refin. often enough

3.8 Concluding remarks

- **Key parameters in selecting a method**
  1. Functionalities of the solver
  2. Characteristics of the matrix
Numerical properties and pivoting.
Symmetric or general
Pattern and density

3. Preprocessing of the matrix
   - Scaling
   - Reordering for minimizing fill-in

4. Target computer (architecture)
   - Substantial gains can be achieved with an adequate solver: in terms of numerical precision, computing and storage
   - Good knowledge of matrix and solvers
   - **Many challenging problems**
     - Active research area

4 Related topics

Related topics / on-going research

- Grid TLSE project (coordinated by ENSEEIHT-IRIT)
- Unsymmetric multifrontal factorization
  - trees become *elimination DAGs*
  - unsymmetric fill-reducing orderings
- Recent improvements to MUMPS solver (Amestoy, Guermouche, L’Excellent, Pralet)
  - Scheduling for SMP platforms
  - Scheduling under memory constraints
  - Improved memory usage
  - Sparse and multiple right-hand side
- **Iterative methods** \((x_{n+1} = f(x_n))\)
  - Another class of methods to solve \(Ax = b\):
  - Hybrid direct-iterative methods
- **Other research directions**: Parallel out-of-core solvers, parallel analysis phase, scheduling for large-scale heterogeneous platforms, …

References


[34] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. TR 95-035, University of Minnesota, June 1995.


