

# Algorithmique pour l'algèbre linéaire creuse

Abdou Guermouche

26 septembre 2008

# Outline

## 1. Introduction to Sparse Matrix Computations

- Motivation and main issues
- Sparse matrices
- Gaussian elimination
- Symmetric matrices and graphs
- The elimination graph model

# A selection of references

## ★ Books

- ▶ Duff, Erisman and Reid, Direct methods for Sparse Matrices, Clarenton Press, Oxford 1986.
- ▶ Dongarra, Duff, Sorensen and H. A. van der Vorst, Solving Linear Systems on Vector and Shared Memory Computers, SIAM, 1991.
- ▶ George, Liu, and Ng, Computer Solution of Sparse Positive Definite Systems, book to appear

## ★ Articles

- ▶ Gilbert and Liu, Elimination structures for unsymmetric sparse LU factors, SIMAX, 1993.
- ▶ Liu, The role of elimination trees in sparse factorization, SIMAX, 1990.
- ▶ Heath and E. Ng and B. W. Peyton, Parallel Algorithms for Sparse Linear Systems, SIAM review 1991.

# Outline

## 1. Introduction to Sparse Matrix Computations

- Motivation and main issues
- Sparse matrices
- Gaussian elimination
- Symmetric matrices and graphs
- The elimination graph model

# Motivations

- ★ solution of linear systems of equations → key algorithmic kernel

*Continuous problem*



*Discretization*



*Solution of a linear system  $Ax = b$*

- ★ Main parameters :
  - ▶ Numerical properties of the linear system (symmetry, pos. definite, conditioning, ...)
  - ▶ Size and structure :
    - Large ( $> 100000 \times 100000$  ?), 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*

# Outline

## 1. Introduction to Sparse Matrix Computations

- Motivation and main issues
- **Sparse matrices**
- Gaussian elimination
- Symmetric matrices and graphs
- The elimination graph model



# Sparse matrices

Example :

$$\begin{array}{rclclcl} 3x_1 & + & 2x_2 & & & = & 5 \\ & & 2x_2 & - & 5x_3 & = & 1 \\ 2x_1 & & & + & 3x_3 & = & 0 \end{array}$$

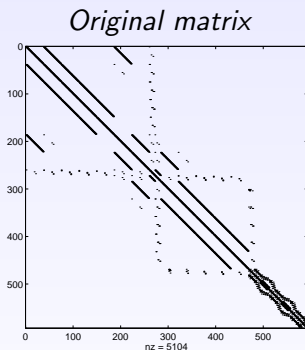
can be represented as

$$\mathbf{Ax} = \mathbf{b},$$

$$\text{where } \mathbf{A} = \begin{pmatrix} 3 & 2 & 0 \\ 0 & 2 & -5 \\ 2 & 0 & 3 \end{pmatrix}, \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, \text{ and } \mathbf{b} = \begin{pmatrix} 5 \\ 1 \\ 0 \end{pmatrix}$$

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 $\mathbf{Ax} = \mathbf{b}$

- ★  $\mathbf{A}$  is unsymmetric :
  - ▶  $\mathbf{A}$  is factorized as :  $\mathbf{A} = \mathbf{LU}$ , where  $\mathbf{L}$  is a lower triangular matrix, and  $\mathbf{U}$  is an upper triangular matrix.
  - ▶ Forward-backward substitution :  $\mathbf{Ly} = \mathbf{b}$  then  $\mathbf{Ux} = \mathbf{y}$
- ★  $\mathbf{A}$  is symmetric :
  - ▶  $\mathbf{A} = \mathbf{LDL}^T$  or  $\mathbf{LL}^T$
- ★  $\mathbf{A}$  is rectangular  $m \times n$  with  $m \geq n$  and  $\min_x \|\mathbf{Ax} - \mathbf{b}\|_2$  :
  - ▶  $\mathbf{A} = \mathbf{QR}$  where  $\mathbf{Q}$  is orthogonal ( $\mathbf{Q}^{-1} = \mathbf{Q}^T$  and  $\mathbf{R}$  is triangular).
  - ▶ Solve :  $\mathbf{y} = \mathbf{Q}^T \mathbf{b}$  then  $\mathbf{Rx} = \mathbf{y}$

# Factorization process

## Solution of $\mathbf{Ax} = \mathbf{b}$

- ★  $\mathbf{A}$  is unsymmetric :
  - ▶  $\mathbf{A}$  is factorized as :  $\mathbf{A} = \mathbf{LU}$ , where  $\mathbf{L}$  is a lower triangular matrix, and  $\mathbf{U}$  is an upper triangular matrix.
  - ▶ Forward-backward substitution :  $\mathbf{Ly} = \mathbf{b}$  then  $\mathbf{Ux} = \mathbf{y}$
- ★  $\mathbf{A}$  is symmetric :
  - ▶  $\mathbf{A} = \mathbf{LDL}^T$  or  $\mathbf{LL}^T$
- ★  $\mathbf{A}$  is rectangular  $m \times n$  with  $m \geq n$  and  $\min_x \|\mathbf{Ax} - \mathbf{b}\|_2$  :
  - ▶  $\mathbf{A} = \mathbf{QR}$  where  $\mathbf{Q}$  is orthogonal ( $\mathbf{Q}^{-1} = \mathbf{Q}^T$  and  $\mathbf{R}$  is triangular).
  - ▶ Solve :  $\mathbf{y} = \mathbf{Q}^T \mathbf{b}$  then  $\mathbf{Rx} = \mathbf{y}$

# Factorization process

## Solution of $\mathbf{Ax} = \mathbf{b}$

- ★  $\mathbf{A}$  is unsymmetric :
  - ▶  $\mathbf{A}$  is factorized as :  $\mathbf{A} = \mathbf{LU}$ , where  $\mathbf{L}$  is a lower triangular matrix, and  $\mathbf{U}$  is an upper triangular matrix.
  - ▶ Forward-backward substitution :  $\mathbf{Ly} = \mathbf{b}$  then  $\mathbf{Ux} = \mathbf{y}$
- ★  $\mathbf{A}$  is symmetric :
  - ▶  $\mathbf{A} = \mathbf{LDL}^T$  or  $\mathbf{LL}^T$
- ★  $\mathbf{A}$  is rectangular  $m \times n$  with  $m \geq n$  and  $\min_x \|\mathbf{Ax} - \mathbf{b}\|_2$  :
  - ▶  $\mathbf{A} = \mathbf{QR}$  where  $\mathbf{Q}$  is orthogonal ( $\mathbf{Q}^{-1} = \mathbf{Q}^T$  and  $\mathbf{R}$  is triangular).
  - ▶ Solve :  $\mathbf{y} = \mathbf{Q}^T \mathbf{b}$  then  $\mathbf{Rx} = \mathbf{y}$

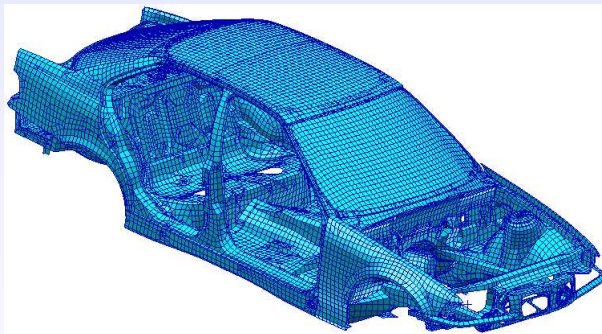
# Difficulties

- ★ Only non-zero values are stored
- ★ Factors  $\mathbf{L}$  and  $\mathbf{U}$  have far more nonzeros than  $\mathbf{A}$
- ★ 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 :

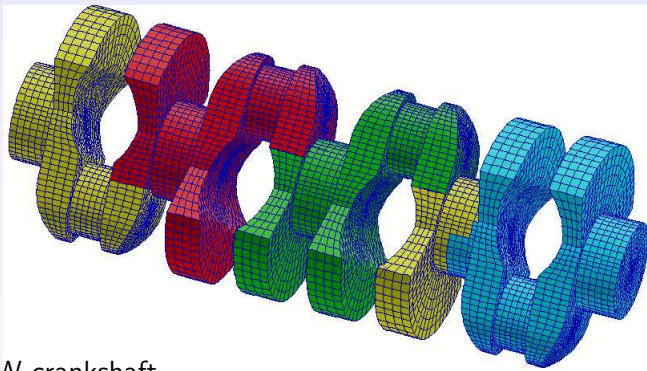


BMW car body,  
227,362 unknowns,  
5,757,996 nonzeros,  
MSC.Software

Size of factors : 51.1 million entries  
Number of operations :  $44.9 \times 10^9$



# Typical test problems :



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 $\mathbf{A}$

## What needs to be represented

- ★ Assembled matrices :  $M \times N$  matrix  $\mathbf{A}$  with NNZ nonzeros.
- ★ Elemental matrices (unassembled) :  $M \times N$  matrix  $\mathbf{A}$  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

	1	2	3
1	a11		
2		a22	a23
3	a31		a33

- ★ Coordinate format

$$\text{IRN} \quad [1 : NNZ] = 1 \quad 3 \quad 2 \quad 2 \quad 3$$

$$\text{JCN} \quad [1 : NNZ] = 1 \quad 1 \quad 2 \quad 3 \quad 3$$

$$\text{VAL} \quad [1 : NNZ] = a_{11} \quad a_{31} \quad a_{22} \quad a_{23} \quad a_{33}$$

- ★ Compressed Sparse Column (CSC) format

$$\text{IRN} \quad [1 : NNZ] = 1 \quad 3 \quad 2 \quad 2 \quad 3$$

$$\text{VAL} \quad [1 : NNZ] = a_{11} \quad a_{31} \quad a_{22} \quad a_{23} \quad a_{33}$$

$$\text{COLPTR} \quad [1 : N + 1] = 1 \quad 3 \quad 4 \quad 6$$

column  $J$  is stored in IRN/A locations  $\text{COLPTR}(J) \dots \text{COLPTR}(J+1)-1$

- ★ Compressed Sparse Row (CSR) format :

Similar to CSC, but row by row

# Classical Data Formats for Assembled Matrices

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

	1	2	3
1	a11		
2		a22	a23
3	a31		a33

- ★ Coordinate format

$$\text{IRN} \quad [1 : NNZ] = 1 \quad 3 \quad 2 \quad 2 \quad 3$$

$$\text{JCN} \quad [1 : NNZ] = 1 \quad 1 \quad 2 \quad 3 \quad 3$$

$$\text{VAL} \quad [1 : NNZ] = a_{11} \quad a_{31} \quad a_{22} \quad a_{23} \quad a_{33}$$

- ★ Compressed Sparse Column (CSC) format

$$\text{IRN} \quad [1 : NNZ] = 1 \quad 3 \quad 2 \quad 2 \quad 3$$

$$\text{VAL} \quad [1 : NNZ] = a_{11} \quad a_{31} \quad a_{22} \quad a_{23} \quad a_{33}$$

$$\text{COLPTR} \quad [1 : N + 1] = 1 \quad 3 \quad 4 \quad 6$$

column  $J$  is stored in IRN/A locations  $\text{COLPTR}(J) \dots \text{COLPTR}(J+1)-1$

- ★ Compressed Sparse Row (CSR) format :

Similar to CSC, but row by row

# Classical Data Formats for Assembled Matrices

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

	1	2	3
1	a11		
2		a22	a23
3	a31		a33

- ★ Coordinate format

$$\text{IRN} \quad [1 : NNZ] = 1 \quad 3 \quad 2 \quad 2 \quad 3$$

$$\text{JCN} \quad [1 : NNZ] = 1 \quad 1 \quad 2 \quad 3 \quad 3$$

$$\text{VAL} \quad [1 : NNZ] = a_{11} \quad a_{31} \quad a_{22} \quad a_{23} \quad a_{33}$$

- ★ Compressed Sparse Column (CSC) format

$$\text{IRN} \quad [1 : NNZ] = 1 \quad 3 \quad 2 \quad 2 \quad 3$$

$$\text{VAL} \quad [1 : NNZ] = a_{11} \quad a_{31} \quad a_{22} \quad a_{23} \quad a_{33}$$

$$\text{COLPTR} \quad [1 : N + 1] = 1 \quad 3 \quad 4 \quad 6$$

column  $J$  is stored in IRN/A locations  $\text{COLPTR}(J) \dots \text{COLPTR}(J+1)-1$

- ★ Compressed Sparse Row (CSR) format :

Similar to CSC, but row by row

# Sparse Matrix-vector products

Assume we want to compute  $Y \leftarrow AX$ .

Various algorithms for matrix-vector product depending on sparse matrix format :

- ★ Coordinate format :

```
Y(1:N) = 0
DO i=1,NNZ
  Y(IRN(i)) = Y(IRN(i)) + VAL(i) * X(JCN(i))
ENDDO
```

- ★ CSC format :



# Sparse Matrix-vector products

Assume we want to compute  $Y \leftarrow AX$ .

Various algorithms for matrix-vector product depending on sparse matrix format :

★ Coordinate format :

```

Y(1:N) = 0
DO i=1,NNZ
  Y(IRN(i)) = Y(IRN(i)) + VAL(i) * X(JCN(i))
ENDDO

```

★ CSC format :

```

Y(1:N) = 0
DO J=1,N
  DO I=COLPTR(J),COLPTR(J+1)-1
    Y(IRN(I)) = Y(IRN(I)) + VAL(I)*X(J)
  ENDDO
ENDDO

```

# Example of elemental matrix format

$$\mathbf{A}_1 = \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} \begin{pmatrix} -1 & 2 & 3 \\ 2 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \quad \mathbf{A}_2 = \begin{matrix} 3 \\ 4 \\ 5 \end{matrix} \begin{pmatrix} 2 & -1 & 3 \\ 1 & 2 & -1 \\ 3 & 2 & 1 \end{pmatrix}$$

★  $N=5$      $NELT=2$      $NVAR=6$      $\mathbf{A} = \sum_{i=1}^{NELT} \mathbf{A}_i$

ELTPTR    [1 :NELT+1]    =    1 4 7

★ ELTVAR    [1 :NVAR]    =    1 2 3 3 4 5

ELTVAL    [1 :NVAL]    =    -1 2 1 2 1 1 3 1 1 2 1 3 -1 2 2 3 -1 1

★ Remarks :

- ▶  $NVAR = ELTPTR(NELT+1)-1$
- ▶  $NVAL = \sum S_i^2$  (unsym) ou  $\sum S_i(S_i + 1)/2$  (sym), avec  
 $S_i = ELTPTR(i + 1) - ELTPTR(i)$
- ▶ storage of elements in ELTVAL : by columns

# File storage : Rutherford-Boeing

- ★ Standard ASCII format for files
- ★ Header + Data (CSC format). key xyz :
  - ▶  $x=[rcp]$  (real, complex, pattern)
  - ▶  $y=[suhzr]$  (sym., uns., herm., skew sym., rectang.)
  - ▶  $z=[ae]$  (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

```

DNV-Ex 1 : Tubular joint-1999-01-17
      1733710      9758      492558      1231394      0
rsa      97578      97578      4925574      0
(10I8)      (10I8)      (3e26.16)
      1      49      96      142      187      231      274      346      417      487
      556      624      691      763      834      904      973      1041      1108      1180
      1251      1321      1390      1458      1525      1573      1620      1666      1711      1755
      1798      1870      1941      2011      2080      2148      2215      2287      2358      2428
      2497      2565      2632      2704      2775      2845      2914      2982      3049      3115
...
      1      2      3      4      5      6      7      8      9      10
      11      12      49      50      51      52      53      54      55      56
      57      58      59      60      67      68      69      70      71      72
      223      224      225      226      227      228      229      230      231      232
      233      234      433      434      435      436      437      438      2      3
      4      5      6      7      8      9      10      11      12      49
      50      51      52      53      54      55      56      57      58      59
...
-0.2624989288237320E+10      0.6622960540857440E+09      0.2362753266740760E+11
0.3372081648690030E+08      -0.4851430162799610E+08      0.1573652896140010E+08
0.1704332388419270E+10      -0.7300763190874110E+09      -0.7113520995891850E+10
0.1813048723097540E+08      0.2955124446119170E+07      -0.2606931100955540E+07
0.1606040913919180E+07      -0.2377860366909130E+08      -0.1105180386670390E+09
0.1610636280324100E+08      0.4230082475435230E+07      -0.1951280618776270E+07
0.4498200951891750E+08      0.2066239484615530E+09      0.3792237438608430E+08
0.9819999042370710E+08      0.3881169368090200E+08      -0.4624480572242580E+08

```

# Outline

## 1. Introduction to Sparse Matrix Computations

- Motivation and main issues
- Sparse matrices
- **Gaussian elimination**
- Symmetric matrices and graphs
- The elimination graph model

## Gaussian elimination

$$\mathbf{A} = \mathbf{A}^{(1)}, \mathbf{b} = \mathbf{b}^{(1)}, \mathbf{A}^{(1)}\mathbf{x} = \mathbf{b}^{(1)} :$$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \quad \begin{array}{l} 2 \leftarrow 2 - 1 \times a_{21}/a_{11} \\ 3 \leftarrow 3 - 1 \times a_{31}/a_{11} \end{array}$$

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

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(2)} \end{pmatrix} \quad \begin{array}{l} b_2^{(2)} = b_2 - a_{21}b_1/a_{11} \dots \\ a_{32}^{(2)} = a_{32} - a_{31}a_{12}/a_{11} \dots \end{array}$$

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

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & 0 & a_{33}^{(3)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(3)} \end{pmatrix} \quad a_{(33)}^{(3)} = a_{(33)}^{(2)} - a_{32}^{(2)}a_{23}^{(2)}/a_{22}^{(2)} \dots$$

$$\text{Typical Gaussian elimination step } k : \boxed{a_{ij}^{(k+1)} = a_{ij}^{(k)} - \frac{a_{ik}^{(k)}a_{kj}^{(k)}}{a_{kk}^{(k)}}$$

# Relation with $\mathbf{A} = \mathbf{LU}$ factorization

- ★ One step of Gaussian elimination can be written :

$$\mathbf{A}^{(k+1)} = \mathbf{L}^{(k)} \mathbf{A}^{(k)}, \text{ with}$$

$$\mathbf{L}^k = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -l_{k+1,k} & \ddots & \\ & & \vdots & \ddots & \\ & & -l_{n,k} & & 1 \end{pmatrix} \text{ and } l_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}.$$

- ★ Then,  $\mathbf{A}^{(n)} = \mathbf{U} = \mathbf{L}^{(n-1)} \dots \mathbf{L}^{(1)} \mathbf{A}$ , which gives  $\boxed{\mathbf{A} = \mathbf{LU}}$ ,

$$\text{with } \mathbf{L} = [\mathbf{L}^{(1)}]^{-1} \dots [\mathbf{L}^{(n-1)}]^{-1} = \begin{pmatrix} 1 & & & 0 \\ & \ddots & & \\ & & \ddots & \\ & & & 1 \\ & & & & l_{i,j} \\ & & & & & \ddots \\ & & & & & & 1 \end{pmatrix}.$$

- ★ In dense codes, entries of  $\mathbf{L}$  and  $\mathbf{U}$  overwrite entries of  $\mathbf{A}$ .
- ★ Furthermore, if  $\mathbf{A}$  is symmetric,  $\boxed{\mathbf{A} = \mathbf{LDL}^T}$  with  $d_{kk} = a_{kk}^{(k)}$  :  
 $A = LU = A^t = U^t L^t$  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) :

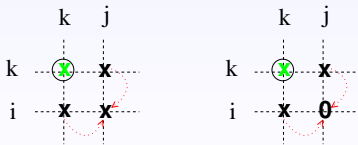
- ★ For  $i > k$  compute  $l_{ik} = a_{ik}/a_{kk}$  ( $= a'_{ik}$ ),
- ★ For  $i > k, j > k$

$$a'_{ij} = a_{ij} - \frac{a_{ik} \times a_{kj}}{a_{kk}}$$

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  $\rightarrow$  its non-zero value must be stored



*fill-in*



- ★ 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

$$\begin{pmatrix} X & X & X & X & X \\ X & X & & & \\ X & & X & & \\ X & & & X & \\ X & & & & X \end{pmatrix}$$

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

# Example

$$\begin{pmatrix} X & & & & X \\ & X & & & X \\ & & X & & X \\ & & & X & X \\ X & X & X & X & X \end{pmatrix}$$

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

# Efficient implementation of sparse solvers

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

```
do i = 1, m
```

```
    A( ind(i) ) = A( ind(i) ) + alpha * w( i )
```

```
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

# Outline

## 1. Introduction to Sparse Matrix Computations

- Motivation and main issues
- Sparse matrices
- Gaussian elimination
- **Symmetric matrices and graphs**
- The elimination graph model

# Symmetric matrices and graphs

- ★ Assumptions :  $\mathbf{A}$  symmetric and pivots are chosen on the diagonal
- ★ Structure of  $\mathbf{A}$  symmetric represented by the graph  $G = (V, E)$ 
  - ▶ Vertices are associated to columns :  $V = \{1, \dots, n\}$
  - ▶ Edges  $E$  are defined by :  $(i, j) \in E \leftrightarrow a_{ij} \neq 0$
  - ▶  $G$  undirected (symmetry of  $\mathbf{A}$ )

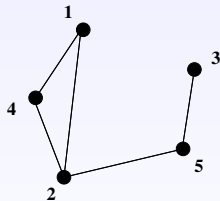
# Symmetric matrices and graphs

★ Remarks :

- ▶ Number of nonzeros in column  $j = |Adj_G(j)|$
- ▶ Symmetric permutation  $\equiv$  renumbering the graph

	1	2	3	4	5
1	×	×		×	
2	×	×		×	×
3			×		×
4	×	×		×	
5		×	×		×

Symmetric matrix



Corresponding graph

# Outline

## 1. Introduction to Sparse Matrix Computations

- Motivation and main issues
- Sparse matrices
- Gaussian elimination
- Symmetric matrices and graphs
- The elimination graph model



# The elimination graph model

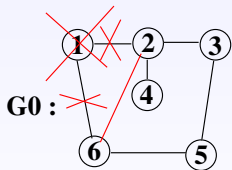
Construction of the elimination graphs

Let  $v_i$  denote the vertex of index  $i$ .  $G_0 = G(\mathbf{A})$ ,  $i = 1$ .

At each step delete  $v_i$  and its incident edges

Add edges so that vertices in  $Adj(v_i)$  are pairwise adjacent in  $G_i = G(\mathbf{H}_i)$ .

$G_i$  are the so-called *elimination graphs*.

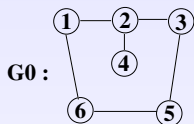


**H0 =**

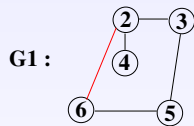
<del>1</del>	<del>×</del>	<del>×</del>	<del>×</del>		
<del>×</del>	2	×	×		×
<del>×</del>	×	3		×	
<del>×</del>	×		4		
<del>×</del>		×		5	×
<del>×</del>	×			×	6

The matrix H0 is a 6x6 matrix with 'x' marks indicating non-zero entries. Red circles and lines highlight the first row and column, and the new edge (2,4) in the matrix.

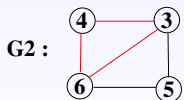
## A sequence of elimination graphs



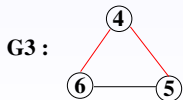
$$\mathbf{H0} = \begin{bmatrix} 1 & \times & & & & \times \\ \times & 2 & \times & \times & & \\ & \times & 3 & & \times & \\ & \times & & 4 & & \\ & & \times & & 5 & \times \\ \times & & & & \times & 6 \end{bmatrix}$$



$$\mathbf{H1} = \begin{bmatrix} 2 & \times & \times & & & + \\ \times & 3 & & \times & & \\ \times & & 4 & & & \\ & \times & & 5 & \times & \\ + & & & \times & 6 & \end{bmatrix}$$



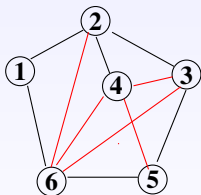
$$\mathbf{H2} = \begin{bmatrix} 3 & + & \times & + \\ + & 4 & & + \\ \times & & 5 & \times \\ + & + & \times & 6 \end{bmatrix}$$



$$\mathbf{H3} = \begin{bmatrix} 4 & + & + \\ + & 5 & \times \\ + & \times & 6 \end{bmatrix}$$

# Introducing the filled graph $G^+(\mathbf{A})$

- ★ Let  $\mathbf{F} = \mathbf{L} + \mathbf{L}^T$  be the filled matrix, and  $G(\mathbf{F})$  the *filled graph* of  $\mathbf{A}$  denoted by  $G^+(\mathbf{A})$ .
- ★ 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  $(v_k, v_j) \in G^+$ .



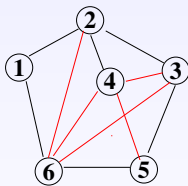
$$G^+(\mathbf{A}) = G(\mathbf{F})$$

$$\begin{bmatrix} \mathbf{1} & \times & & & & \times \\ \times & \mathbf{2} & \times & \times & & + \\ & \times & \mathbf{3} & + & \times & + \\ & \times & + & \mathbf{4} & + & + \\ & & \times & + & \mathbf{5} & \times \\ \times & + & + & + & \times & \mathbf{6} \end{bmatrix}$$

$$\mathbf{F} = \mathbf{L} + \mathbf{L}^T$$

# 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 :



$$G^+(A) = G(F)$$

$$\begin{bmatrix} 1 & \times & & & & \times \\ \times & 2 & \times & \times & & + \\ & \times & 3 & + & \times & + \\ & & \times & + & 4 & + & + \\ & & & \times & + & 5 & \times \\ \times & + & + & + & \times & 6 \end{bmatrix}$$

$$F = L + L^T$$

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

# 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  $\mathbf{A}$  be a symmetric positive-definite matrix,  $\mathbf{A} = \mathbf{L}\mathbf{L}^T$  its Cholesky factorization, and  $G^+(\mathbf{A})$  its filled graph (graph of  $\mathbf{F} = \mathbf{L} + \mathbf{L}^T$ ).

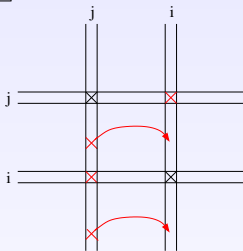
## Definition

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



# Properties of the elimination tree

- ★ Another perspective also leads to the elimination tree

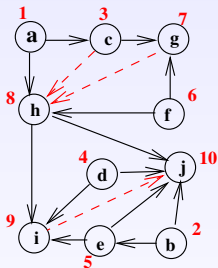


- ★ Dependency between columns of  $\mathbf{L}$  :

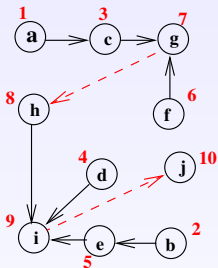
1. Column  $i > j$  depends on column  $j$  iff  $l_{ij} \neq 0$
2. Use a directed graph to express this dependency
3. Simplify redundant dependencies (*transitive reduction* in graph theory)

- ★ *The transitive reduction of the directed filled graph gives the elimination tree structure*

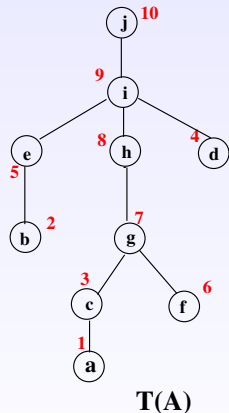
## Directed filled graph and its transitive reduction



Directed filled graph



Transitive reduction

 $T(A)$



# Outline

## 2. Ordering sparse matrices

- Objectives/Outline
- Fill-reducing orderings
- Impact of fill reduction algorithm on the shape of the tree
  - Postorderings and memory usage
- Reorder unsymmetric matrices to special forms
- Combining approaches

# Outline

## 2. Ordering sparse matrices

- Objectives/Outline
- Fill-reducing orderings
- Impact of fill reduction algorithm on the shape of the tree
  - Postorderings and memory usage
- Reorder unsymmetric matrices to special forms
- Combining approaches

# 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

# Outline

## 2. Ordering sparse matrices

- Objectives/Outline
- **Fill-reducing orderings**
- Impact of fill reduction algorithm on the shape of the tree
  - Postorderings and memory usage
- Reorder unsymmetric matrices to special forms
- Combining approaches

# 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).

# Fill-reducing orderings

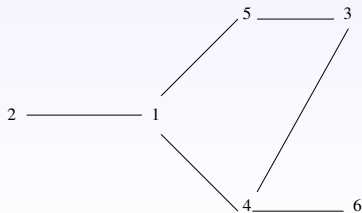
- ★ Local heuristics : At each step of the factorization, selection of the pivot that is likely to minimize fill-in.
  - ▶ Method is characterized by the way pivots are selected.
  - ▶ Markowitz criterion (for a general matrix).
  - ▶ Minimum degree (for symmetric matrices).
- ★ Hybrid approaches : Once the matrix is permuted in order to obtain a block structure, local heuristics are used within the blocks.

## Cuthill-McKee and Reverse Cuthill-McKee

Consider the matrix :

$$\mathbf{A} = \begin{bmatrix} x & x & & x & x & \\ x & x & & & & \\ & & x & x & x & \\ x & & x & x & & x \\ x & & x & & x & \\ & & & x & & x \end{bmatrix}$$

The corresponding graph is



# 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 :

$$A = \begin{bmatrix} x & x & & & & \\ x & x & x & x & & \\ & x & x & & x & x \\ & x & & x & x & \\ & & x & x & x & \\ & & x & & & x \end{bmatrix}$$



# 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 :

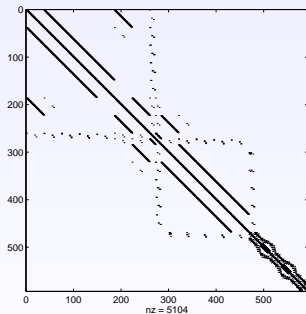
$$A = \begin{bmatrix} x & & & & & x \\ & x & x & x & & \\ & & x & x & & x \\ x & x & & x & x & \\ & & & x & x & x & x \\ & & & & & x & x \end{bmatrix}$$

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

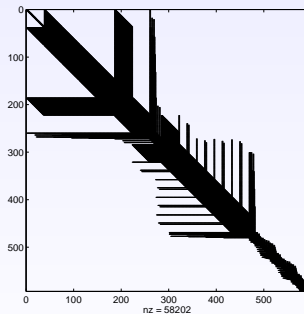
## Illustration : Reverse Cuthill-McKee on matrix dwt\_592.rua

*Harwell-Boeing matrix* : dwt\_592.rua, structural computing on a submarine. NZ(LU factors)=58202

*Original matrix*



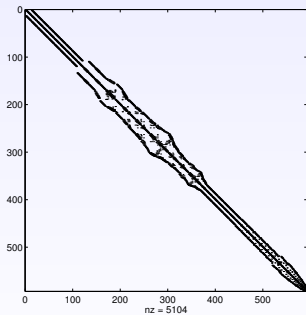
*Factorized matrix*



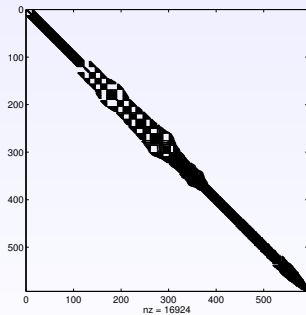
## Illustration : Reverse Cuthill-McKee on matrix dwt\_592.rua

NZ(LU factors)=16924

*Permuted matrix  
(RCM)*



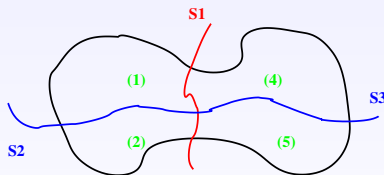
*Factorized permuted matrix*



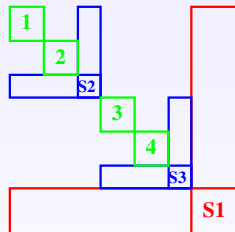
# Nested Dissection

Recursive approach based on graph partitioning.

*Graph partitioning*



*Permuted matrix*



# 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  $Metric$  such that  $Metric(v_i) < Metric(v_j)$  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,

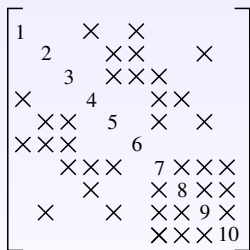
Step3 : update  $Metric(v_j)$  for all non-selected nodes  $v_j$ .

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

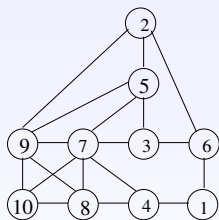
# Minimum degree algorithm

## ★ Step 1 :

Select the vertex that possesses the smallest number of neighbors in  $G^0$ .



(a) Sparse symmetric matrix

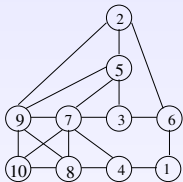


(b) Elimination graph

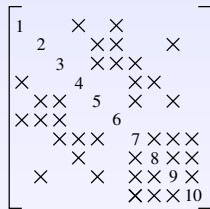
The node/variable selected is 1 of degree 2.

## Illustration

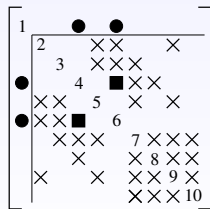
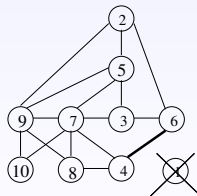
## Step 1 : elimination of pivot 1



(a) Elimination graph



(b) Factors and active submatrix



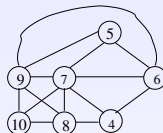
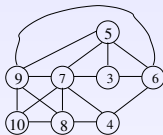
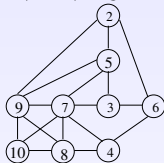
x Initial nonzeros

● Nonzeros in factors

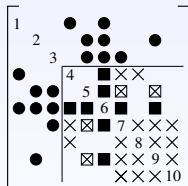
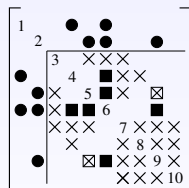
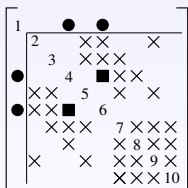
■ Fill-in

## Illustration (cont'd)

Graphs  $G_1, G_2, G_3$  and corresponding reduced matrices.



(a) Elimination graphs



(b) Factors and active submatrices

× Original nonzero

⊠ Original nonzero modified

■ Fill-in

● Nonzeros in factors



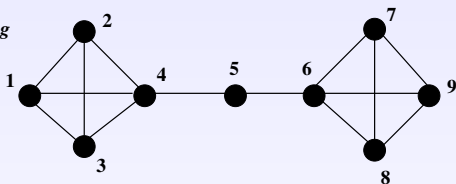
# Minimum Degree does not always minimize fill-in !!!

Consider the following matrix

$$\begin{bmatrix} 1 & \times & \times & \times & & & & & & & \\ \times & 2 & \times & \times & & & & & & & \\ \times & \times & 3 & \times & & & & & & & \\ \times & \times & \times & 4 & \times & & & & & & \\ & \times & & & 5 & \times & & & & & \\ & \times & & & & 6 & \times & \times & \times & & \\ & & & & & \times & 7 & \times & \times & & \\ & & & & & & \times & \times & 8 & \times & \\ & & & & & & & \times & \times & \times & 9 \end{bmatrix}$$

Remark: Using initial ordering

↓  
No fill-in

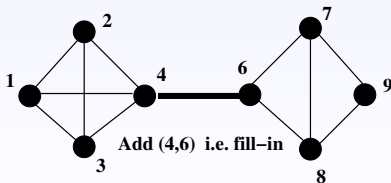


Corresponding elimination graph

Step 1 of Minimum Degree:

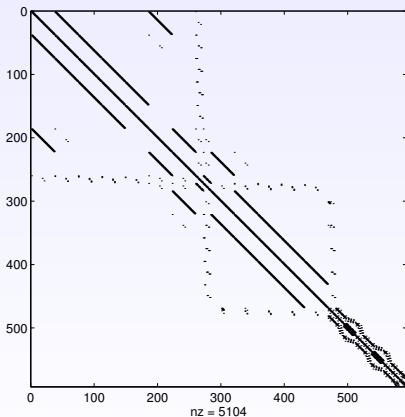
Select pivot 5 (minimum degree = 2)

Updated graph



# Influence on the structure of factors

*Harwell-Boeing matrix* : dwt\_592.rua, structural computing on a submarine.  $NZ(LU \text{ factors})=58202$





# Outline


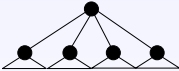
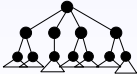
## 2. Ordering sparse matrices

- Objectives/Outline
- Fill-reducing orderings
- Impact of fill reduction algorithm on the shape of the tree
  - Postorderings and memory usage
- Reorder unsymmetric matrices to special forms
- Combining approaches

## Impact of fill reduction on the shape of the tree (1/2)

Reordering technique	Shape of the tree	observations
<b>AMD</b>		<ul style="list-style-type: none"><li>★ Deep well-balanced</li><li>★ Large frontal matrices on top</li></ul>
<b>AMF</b>		<ul style="list-style-type: none"><li>★ Very deep unbalanced</li><li>★ Small frontal matrices</li></ul>

## Impact of fill reduction on the shape of the tree (2/2)

Reordering technique	Shape of the tree	observations
<b>PORD</b>		<ul style="list-style-type: none"> <li>★ deep unbalanced</li> <li>★ Small frontal matrices</li> </ul>
<b>SCOTCH</b>		<ul style="list-style-type: none"> <li>★ Very wide well-balanced</li> <li>★ Large frontal matrices</li> </ul>
<b>METIS</b>		<ul style="list-style-type: none"> <li>★ Wide well-balanced</li> <li>★ Smaller frontal matrices (than SCOTCH)</li> </ul>

# 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

# Impact of fill-reducing heuristics

## Size of factors (millions of entries)

	METIS	SCOTCH	PORD	AMF	AMD
GUPTA2	8.55	12.97	9.77	<b>7.96</b>	8.08
SHIP_003	73.34	79.80	73.57	<b>68.52</b>	91.42
TWOTONE	25.04	25.64	28.38	22.65	<b>22.12</b>
WANG3	<b>7.65</b>	9.74	7.99	8.90	11.48
XENON2	<b>94.93</b>	100.87	107.20	144.32	159.74

## Peak of active memory (millions of entries)

	METIS	SCOTCH	PORD	AMF	AMD
GUPTA2	58.33	289.67	78.13	<b>33.61</b>	52.09
SHIP_003	25.09	23.06	20.86	<b>20.77</b>	32.02
TWOTONE	13.24	13.54	11.80	<b>11.63</b>	17.59
WANG3	3.28	3.84	<b>2.75</b>	3.62	6.14
XENON2	14.89	15.21	<b>13.14</b>	23.82	37.82

# Impact of fill-reducing heuristics

## Number of operations (millions)

	METIS	SCOTCH	PORD	AMF	AMD
GUPTA2	2757.8	4510.7	4993.3	2790.3	<b>2663.9</b>
SHIP_003	<b>83828.2</b>	92614.0	112519.6	96445.2	155725.5
TWOTONE	29120.3	<b>27764.7</b>	37167.4	29847.5	29552.9
WANG3	<b>4313.1</b>	5801.7	5009.9	6318.0	10492.2
XENON2	<b>99273.1</b>	112213.4	126349.7	237451.3	298363.5

## Matrix CONESHL (SAMTECH, 1 million equations)

Matrix	order	factor entries	Total memory required	Floating-point operations
CONESHL	METIS	$687 \times 10^6$	8.9 GBytes	$1.6 \times 10^{12}$
	PORD	$746 \times 10^6$	8.4 GBytes	$2.2 \times 10^{12}$



# Impact of fill-reducing heuristics/MUMPS

## Time for factorization (seconds)

		1p	16p	32p	64p	128p
CONESHL	METIS	970	60	41	27	14
	PORD	1264	104	67	41	26
AUDI	METIS	2640	198	108	70	42
	PORD	1599	186	146	83	54

## Matrices with quasi dense rows :

## Impact on the ordering time (seconds) of GUPTA2 matrix

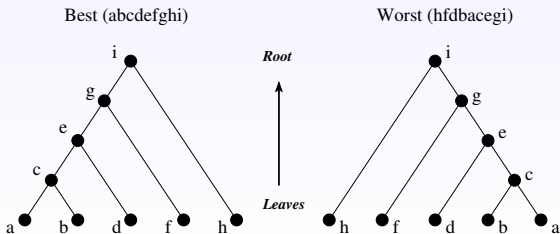
	AMD	METIS	QAMD
Analysis	361	52	23
Total	379	76	59

# 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 a postordering-based tree traversal more interesting than a random topological ordering?

# 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 a postordering-based tree traversal more interesting than a random topological ordering ?
- ★ Furthermore, memory usage also depends on the postordering chosen :



# Outline

## 2. Ordering sparse matrices

- Objectives/Outline
- Fill-reducing orderings
- Impact of fill reduction algorithm on the shape of the tree
  - Postorderings and memory usage
- Reorder unsymmetric matrices to special forms
- Combining approaches

# Reordering unsymmetric matrices

## 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).

# Maximum weighted matching

Structural+numerical problem :

★ **Maximum weighted transversal :**

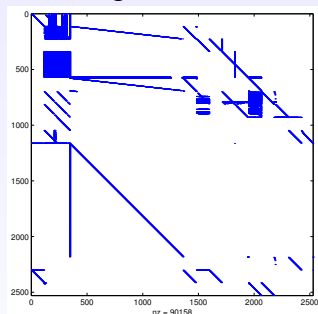
how to find a permutation  $P$  of  $A$  columns such that the product (or another metric : min, sum ...) of the diagonal entries of  $AP$  is maximum ?

★ **Maximum weighted matching ?**

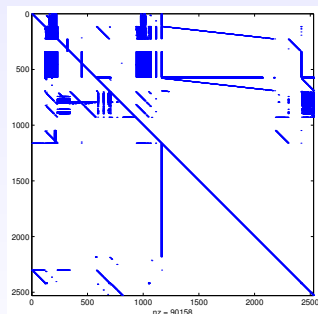
Same techniques as in maximum matching, but branch and bound more complicated and it is more efficient to do breadth first search. MC64 in fortran 77 or 90 (HSL library).

# Illustration of maximum transversal ordering (matrix `ORANI678`)

Original matrix



Permuted matrix



- ★ Better numerical stability
- ★ Easier to factorize for solvers that work on  $A + A^T$



# Impact of Maximum weighted matching algorithms (MUMPS)

## ★ *Influence of maximum weighted matching on the performance*

Matrix		Symmetry	$ LU $ ( $10^6$ )	Flops ( $10^9$ )	Backwd Error
TWOTONE	<b>OFF</b>	28	235	1221	
	<b>ON</b>	43	22	29	
FIDAPM11	<b>OFF</b>	100	16	10	
	<b>ON</b>	46	28	29	

- ★ On very unsymmetric matrices : **reduce flops, factor size and memory used.**
- ★ In general **improve accuracy**, and reduce number of iterative refinements.
- ★ **Limit numerical pivoting / improve reliability** of memory estimates.

# Impact of Maximum weighted matching algorithms (MUMPS)

## ★ *Influence of maximum weighted matching on the performance*

Matrix		Symmetry	$ LU $ ( $10^6$ )	Flops ( $10^9$ )	Backwd Error
TWO-TONE	<b>OFF</b>	28	235	1221	$10^{-6}$
	<b>ON</b>	43	22	29	$10^{-12}$
FIDAPM11	<b>OFF</b>	100	16	10	$10^{-10}$
	<b>ON</b>	46	28	29	$10^{-11}$

- ★ On very unsymmetric matrices : **reduce flops, factor size and memory used.**
- ★ In general **improve accuracy**, and reduce number of iterative refinements.
- ★ **Limit numerical pivoting / improve reliability** of memory estimates.

# Reduction to Block Triangular Form (BTF)

- ★ Suppose that there exist permutations matrices  $\mathbf{P}$  and  $\mathbf{Q}$  such that

$$\mathbf{PAQ} = \begin{pmatrix} \mathbf{B}_{11} & & & & & & \\ \mathbf{B}_{21} & \mathbf{B}_{22} & & & & & \\ \mathbf{B}_{31} & \mathbf{B}_{32} & \mathbf{B}_{33} & & & & \\ \cdot & \cdot & \cdot & \cdot & \cdot & & \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \\ \mathbf{B}_{N1} & \mathbf{B}_{N2} & \mathbf{B}_{N3} & \cdot & \cdot & \cdot & \mathbf{B}_{NN} \end{pmatrix}$$

- ★ If  $N > 1$   $\mathbf{A}$  is said to be *reducible* (irreducible otherwise)
- ★ Each  $\mathbf{B}_{ii}$  is supposed to be irreducible (otherwise finer decomposition is possible)
- ★ Advantage : to solve  $\mathbf{Ax} = \mathbf{b}$  only  $\mathbf{B}_{ii}$  need be factored  
 $\mathbf{B}_{ii}x_i = (\mathbf{Pb})_i - \sum_{j=1}^{i-1} \mathbf{B}_{ij}y_j, i = 1, \dots, N$  with  $\mathbf{y} = \mathbf{Q}^T \mathbf{x}$

# A two-stage approach to compute the reduction

- ★ Stage (1) : compute a (column) permutation matrix  $\mathbf{Q}_1$  such that  $\mathbf{A}\mathbf{Q}_1$  has non zeros on the diagonal (find a maximum transversal of  $\mathbf{A}$ ), then
- ★ Stage(2) : compute a (symmetric) permutation matrix  $\mathbf{P}$  such that  $\mathbf{P}\mathbf{A}\mathbf{Q}_1\mathbf{P}^t$  is BTF.

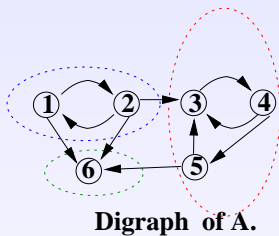
The diagonal blocks of the BTF are uniquely defined. Techniques exist to directly compute the BTF form. They do not present advantage over this two stage approach.

# Main components of the algorithm

- ★ Objective : assume  $\mathbf{A}\mathbf{Q}_1$  has non zeros on the diagonal and compute  $\mathbf{P}$  such that  $\mathbf{P}\mathbf{A}\mathbf{Q}_1\mathbf{P}^t$  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  $\mathbf{B}_{ii}$  of the BTF format.

## Main components of the algorithm

$$\mathbf{A} = \begin{bmatrix}
 \mathbf{1} \times \times & & & & & \\
 \times \mathbf{2} \times \times & & & & & \\
 & \mathbf{6} & & & & \\
 & & \mathbf{3} \times & & & \\
 & & \times \mathbf{4} \times & & & \\
 & \times & \times & \mathbf{5} & & 
 \end{bmatrix}$$



$$\mathbf{PAP}^t = \begin{bmatrix}
 \mathbf{6} & & & & & \\
 & \mathbf{3} \times & & & & \\
 & \times \mathbf{4} \times & & & & \\
 \times & \times & \mathbf{5} & & & \\
 \times & & & \mathbf{1} \times & & \\
 \times & \times & & \times \mathbf{2} & & 
 \end{bmatrix}$$

BTF of  $\mathbf{A}$

# Outline

## 2. Ordering sparse matrices

- Objectives/Outline
- Fill-reducing orderings
- Impact of fill reduction algorithm on the shape of the tree
  - Postorderings and memory usage
- Reorder unsymmetric matrices to special forms
- Combining approaches

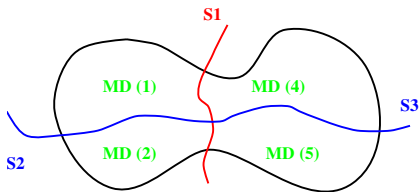
# Example (1) of hybrid approach

- ★ 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
  - ▶ purely local heuristics

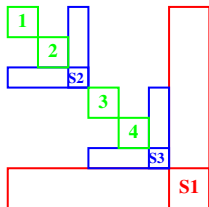


# (1 cont) hybrid approach

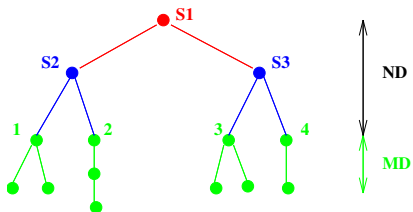
*Graph partitioning*



*Permuted matrix*



*Elimination graph*

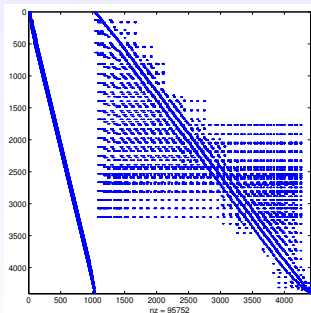


## Example (2) : combine maximum transversal and fill-in reduction

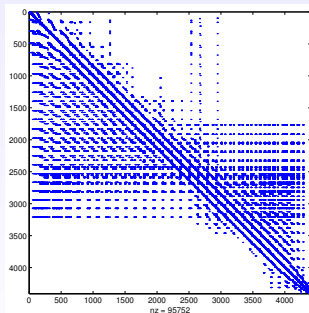
- ★ Consider the **LU** factorization  $\mathbf{A} = \mathbf{LU}$  of an unsymmetric matrix.
- ★ Compute the column permutation  $\mathbf{Q}$  leading to a maximum numerical transversal of  $\mathbf{A}$ .  $\mathbf{AQ}$  has large (in some sense) numerical entries on the diagonal.
- ★ Find best ordering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

# Permuting large entries on the diagonal using MC64 (matrix AV4408)

Original matrix

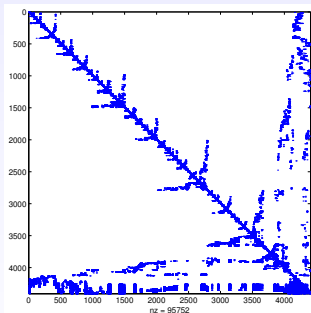


Permuted matrix

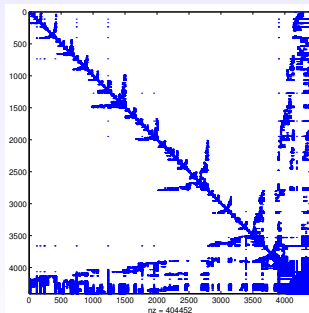


# Symmetric reordering and factorization (matrix AV4408)

AMD ordering



LU factors



### 3. Factorization of sparse matrices

- Introduction
- Elimination tree and Multifrontal approach
- Task mapping and scheduling
  - Influence of scheduling on the makespan
- Distributed memory approaches
- Some parallel solvers
- Concluding remarks

# 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

# Outline

## 3. Factorization of sparse matrices

- Introduction
- Elimination tree and Multifrontal approach
- Task mapping and scheduling
  - Influence of scheduling on the makespan
- Distributed memory approaches
- Some parallel solvers
- Concluding remarks

# 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  $\mathbf{A}$  (symmetric/unsymmetric orderings, scalings)
- ▶ Build the dependency graph (elimination tree, eDAG ...)

## 2. Factorization ( $A = \mathbf{LU}$ , $\mathbf{LDL}^T$ , $\mathbf{LL}^T$ , $\mathbf{QR}$ )

Numerical pivoting

## 3. Solution based on factored matrices

- ▶ triangular solves :  $Ly = b$ , 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 ( $u = 1 \rightarrow$  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

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

$u$	Nonzeros in <b>LU</b> factors	Error
1.0	16767	$3 \times 10^{-9}$
0.25	14249	$6 \times 10^{-10}$
0.1	13660	$4 \times 10^{-9}$
0.01	15045	$4 \times 10^{-5}$
$10^{-4}$	16198	$1 \times 10^2$
$10^{-10}$	16553	$3 \times 10^{23}$

# Iterative refinement for linear systems

Suppose that a solver has computed  $\mathbf{A} = \mathbf{LU}$  (or  $\mathbf{LDL}^T$  or  $\mathbf{LL}^T$ , and a solution  $\tilde{\mathbf{x}}$  to  $\mathbf{Ax} = \mathbf{b}$ .

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

# Outline

## 3. Factorization of sparse matrices

- Introduction
- **Elimination tree and Multifrontal approach**
- Task mapping and scheduling
  - Influence of scheduling on the makespan
- Distributed memory approaches
- Some parallel solvers
- Concluding remarks

# Elimination tree and Multifrontal approach

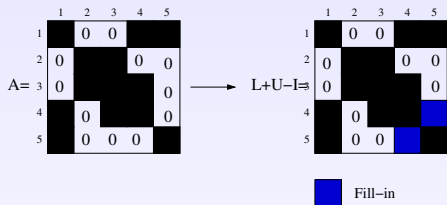
## We recall that :

- ★ 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

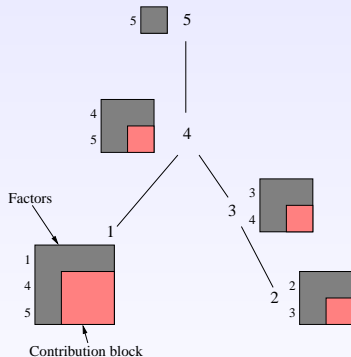
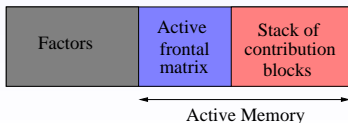
- ★ Permute matrix (to reduce fill-in)  $\mathbf{PAP}^T$ .
  - ★ Build filled matrix  $\mathbf{A}_F = \mathbf{L} + \mathbf{L}^T$  where  $\mathbf{PAP}^T = \mathbf{LL}^T$
  - ★ Transitive reduction of associated filled graph
- Each column corresponds to a node of the graph. Each node  $k$  of the tree corresponds to the factorization of a frontal matrix whose row structure is that of column  $k$  of  $\mathbf{A}_F$ .

# 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

# Supernodal methods

## Definition

A *supernode* (or *supervariable*) is a set of contiguous columns in the factors  $\mathbf{L}$  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

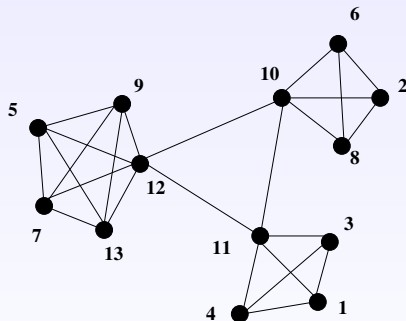
## ★ HOW?

- ▶ Relax the number of nonzeros of the matrix
- ▶ Amalgamation of nodes of the elimination tree

# Amalgamation and Supervariables

*Amalgamation of supervariables does not cause fill-in*

**Initial Graph :**

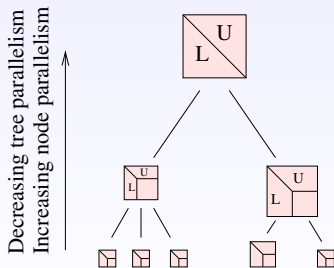


**Reordering :** 1, 3, 4, 2, 6, 8, 10, 11, 5, 7, 9, 12, 13

**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

Computer	#procs	Multifrontal factorization			
		(1)		(2)	
		MFlops	(speed-up)	MFlops	(speed-up)
Alliant FX/80	8	15	(1.9)	34	(4.3)
IBM 3090J/6VF	6	126	(2.1)	227	(3.8)
CRAY-2	4	316	(1.8)	404	(2.3)
CRAY Y-MP	6	529	(2.3)	1119	(4.8)

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.

# Outline

## 3. Factorization of sparse matrices

- Introduction
- Elimination tree and Multifrontal approach
- **Task mapping and scheduling**
  - Influence of scheduling on the makespan
- Distributed memory approaches
- Some parallel solvers
- Concluding remarks

# 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

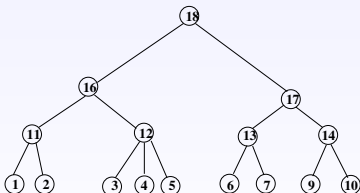
## Objective :

*Assign processes/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).



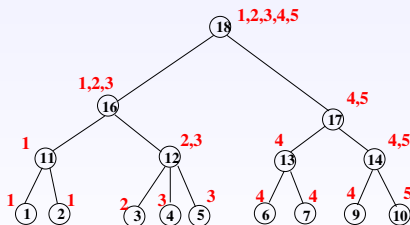
Ordering not so good in terms of working memory.



# Static scheduling : 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.



Mapping of the tasks onto the 5 processors

# Outline

## 3. Factorization of sparse matrices

- Introduction
- Elimination tree and Multifrontal approach
- Task mapping and scheduling
  - Influence of scheduling on the makespan
- **Distributed memory approaches**
- Some parallel solvers
- Concluding remarks

# 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

## Left-looking approach

```

for  $j = 1$  to  $n$  do
  for  $k \in Struct(row \mathbf{L}_{j,1:j-1})$  do
     $cmod(j, k)$ 
   $cdiv(j)$ 

```

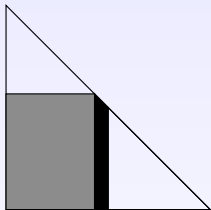
## Right-looking approach

```

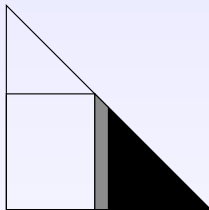
for  $k = 1$  to  $n$  do
   $cdiv(k)$ 
  for  $j \in Struct(col \mathbf{L}_{k+1:n,k})$  do
     $cmod(j, k)$ 

```

# Illustration of Left and right looking



**Left-looking**



**Right-looking**

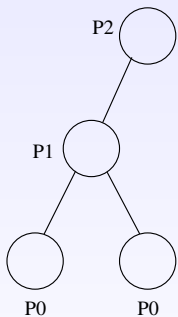


**used for modification**

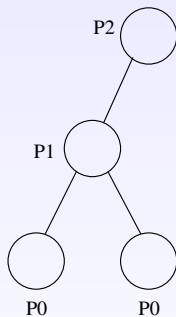


**modified**

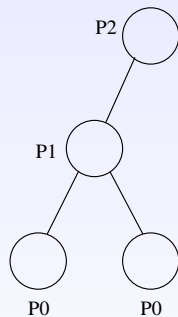
# Multifrontal, Fan-in, Fan-out



(a) Fan-in.



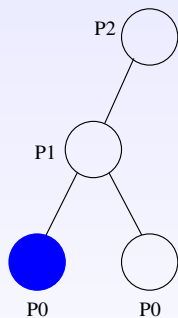
(b) Fan-out.



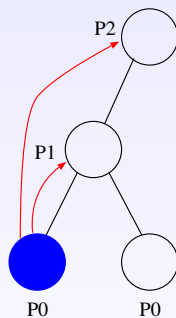
(c) Multifrontal.

**FIG.:** Communication schemes for the three approaches.

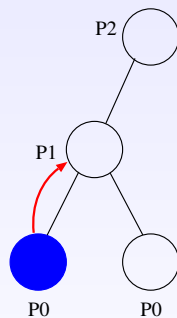
# Multifrontal, Fan-in, Fan-out



(a) Fan-in.



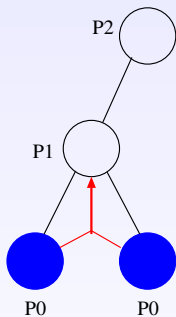
(b) Fan-out.



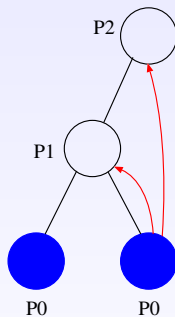
(c) Multifrontal.

**FIG.:** Communication schemes for the three approaches.

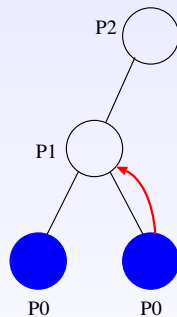
# Multifrontal, Fan-in, Fan-out



(a) Fan-in.



(b) Fan-out.

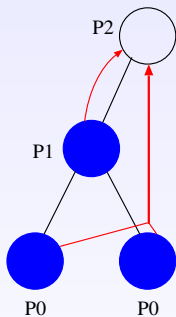


(c) Multifrontal.

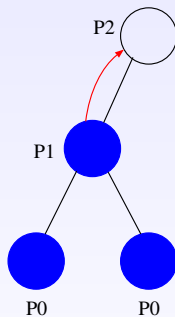
**FIG.:** Communication schemes for the three approaches.



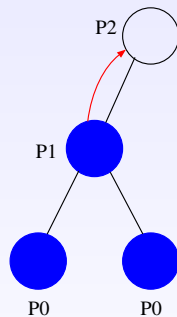
# Multifrontal, Fan-in, Fan-out



(a) Fan-in.



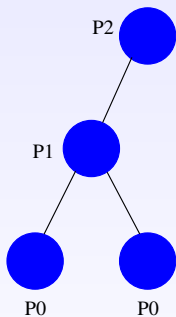
(b) Fan-out.



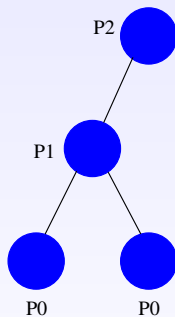
(c) Multifrontal.

FIG.: Communication schemes for the three approaches.

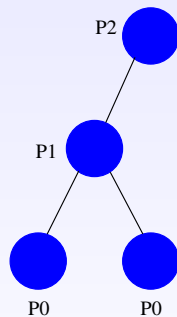
# Multifrontal, Fan-in, Fan-out



(a) Fan-in.



(b) Fan-out.



(c) Multifrontal.

FIG.: Communication schemes for the three approaches.

# Outline

## 3. Factorization of sparse matrices

- Introduction
- Elimination tree and Multifrontal approach
- Task mapping and scheduling
  - Influence of scheduling on the makespan
- Distributed memory approaches
- **Some parallel solvers**
- Concluding remarks

# Shared memory sparse direct codes

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

<sup>†</sup> Only object code for SGI is available

## Distributed-memory sparse direct codes

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

<sup>§</sup> [dept-info.labri.u-bordeaux.fr/~ramet/pastix](http://dept-info.labri.u-bordeaux.fr/~ramet/pastix)

<sup>‡</sup> Only object code for IBM is available. No numerical pivoting performed.

# Outline

## 3. Factorization of sparse matrices

- Introduction
- Elimination tree and Multifrontal approach
- Task mapping and scheduling
  - Influence of scheduling on the makespan
- Distributed memory approaches
- Some parallel solvers
- **Concluding remarks**

# 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