Algorithmique pour l'algèbre linéaire creuse

Pascal Hénon

12 janvier 2009

Contributions

Many thanks to Patrick Amestoy, Abdou Guermouche and Jean-Yves l'Excellent for their large contribution to these slides.

Pascal Hénon

Plan du cours

- Introduction aux méthodes de résolutions des systèmes linéaires creux
- Méthodes directes
- Quelques méthodes itératives

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.
- Saad, Yousef, Iterative methods for sparse linear systems (2nd edition), SIAM press, 2003

★ 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

Motivations

- ⋆ solution of linear systems of equations $\rightarrow$ key algorithmic kernel

Continuous problem

$\downarrow$

Discretization

$\downarrow$

Solution of a linear system $Ax = b$

- ⋆ Main parameters :
  - Numerical properties of the linear system (symmetry, pos. definite, conditioning, . . .)
  - Size and structure :
    - Large ($> 1000000 \times 1000000$?), square/rectangular
    - Dense or sparse (structured / unstructured)
    - Target computer (sequential/parallel)

$\rightarrow$ *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

Sparse matrices

Example :

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

can be represented as

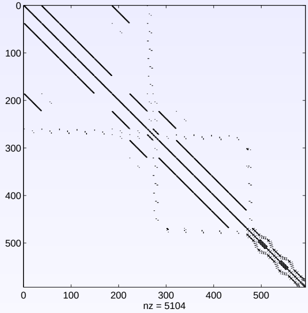
$$\mathbf{Ax} = \mathbf{b},$$

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

Sparse matrix : only nonzeros are stored.

Sparse matrix ?

Original matrix



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

Factorization process (direct method)

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 (direct method)

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 (direct method)

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 :
  - $\mathbf{A} = \mathbf{LDL}^{\mathrm{T}}$ or $\mathbf{LL}^{\mathrm{T}}$
- $\mathbf{A}$ 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 ($\mathbf{Q}^{-1} = \mathbf{Q}^{\mathrm{T}}$ and $\mathbf{R}$ is triangular).
  - Solve : $\mathbf{y} = \mathbf{Q}^{\mathrm{T}}\mathbf{b}$ 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

  $\longrightarrow$ *out-of-core solvers*

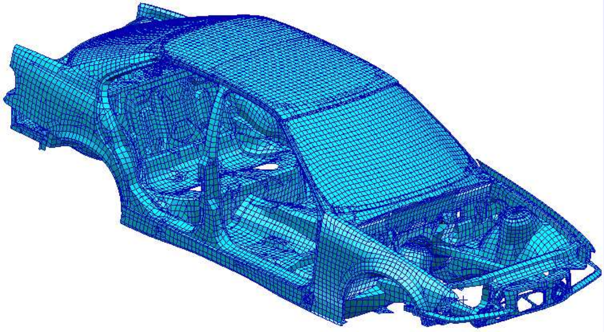
Key numbers :

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

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

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

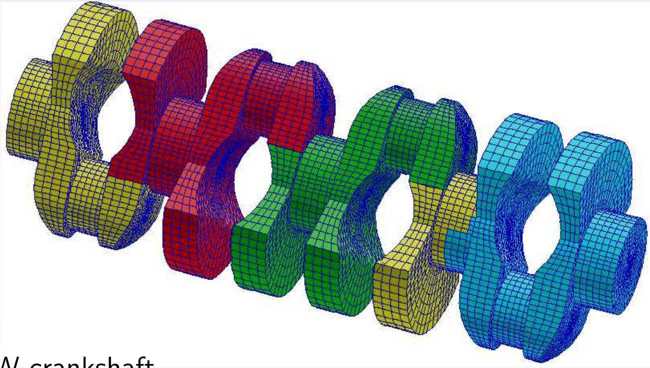
Typical test problems :



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 de 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 : MxN matrix $\mathbf{A}$ with NNZ nonzeros.
- Elemental matrices (unassembled) : MxN matrix $\mathbf{A}$ with NELT elements.
- Arithmetic : Real (4 or 8 bytes) or complex (8 or 16 bytes)
- Symmetric (or Hermitian)
  $\rightarrow$ 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

$$
\begin{array}{c|ccc|}
 & 1 & 2 & 3 \\
\hline
1 & a11 & & \\
2 & & a22 & a23 \\
3 & a31 & & a33 \\
\hline
\end{array}
$$

- Coordinate format

$$
\begin{array}{llcccccc}
\text{IC} & [1:NNZ] & = & 1 & 3 & 2 & 2 & 3 \\
\text{JC} & [1:NNZ] & = & 1 & 1 & 2 & 3 & 3 \\
\text{VAL} & [1:NNZ] & = & a_{11} & a_{31} & a_{22} & a_{23} & a_{33}
\end{array}
$$

- Compressed Sparse Column (CSC) format

$$
\begin{array}{llcccccc}
\text{IA} & [1:NNZ] & = & 1 & 3 & 2 & 2 & 3 \\
\text{VAL} & [1:NNZ] & = & a_{11} & a_{31} & a_{22} & a_{23} & a_{33} \\
\text{JA} & [1:N+1] & = & 1 & 3 & 4 & 6 &
\end{array}
$$

column J is stored in IA and VAL at indices JA(J)...JA(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

$$
\begin{array}{c|c|c|c|}
 & 1 & 2 & 3 \\
\hline
1 & a11 & & \\
\hline
2 & & a22 & a23 \\
\hline
3 & a31 & & a33 \\
\hline
\end{array}
$$

- Coordinate format

IC	$[1:NNZ]$	$=$	1	3	2	2	3
JC	$[1:NNZ]$	$=$	1	1	2	3	3
VAL	$[1:NNZ]$	$=$	a_{11}	a_{31}	a_{22}	a_{23}	a_{33}

- Compressed Sparse Column (CSC) format

IA	$[1:NNZ]$	$=$	1	3	2	2	3
VAL	$[1:NNZ]$	$=$	a_{11}	a_{31}	a_{22}	a_{23}	a_{33}
JA	$[1:N+1]$	$=$	1	3	4	6	

column J is stored in IA and VAL at indices JA(J)...JA(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

$$
\begin{array}{c|c|c|c|}
 & 1 & 2 & 3 \\
\hline
1 & a11 & & \\
\hline
2 & & a22 & a23 \\
\hline
3 & a31 & & a33 \\
\hline
\end{array}
$$

- Coordinate format

$$
\begin{array}{llcccccc}
\text{IC} & [1:NNZ] & = & 1 & 3 & 2 & 2 & 3 \\
\text{JC} & [1:NNZ] & = & 1 & 1 & 2 & 3 & 3 \\
\text{VAL} & [1:NNZ] & = & a_{11} & a_{31} & a_{22} & a_{23} & a_{33}
\end{array}
$$

- Compressed Sparse Column (CSC) format

$$
\begin{array}{llcccccc}
\text{IA} & [1:NNZ] & = & 1 & 3 & 2 & 2 & 3 \\
\text{VAL} & [1:NNZ] & = & a_{11} & a_{31} & a_{22} & a_{23} & a_{33} \\
\text{JA} & [1:N+1] & = & 1 & 3 & 4 & 6 &
\end{array}
$$

column J is stored in IA and VAL at indices JA(J)...JA(J+1)-1

- Compressed Sparse Row (CSR) format :
Similar to CSC, but row by row

Sparse Matrix-vector products

Assume we want to comute $Y \leftarrow AX$.
Various algorithms for matrix-vector product depending on sparse matrix format :

- Coordinate format :

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

- CSC format :

Sparse Matrix-vector products

Assume we want to comute $Y \leftarrow AX$.
Various algorithms for matrix-vector product depending on sparse matrix format :

- Coordinate format :

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

- CSC format :

```
Y(1:N) = 0
DO J=1,N
   DO I=JA(J),JA(J+1)-1
      Y(IA(I)) = Y(IA(I)) + VAL(I)*X(J)
   ENDDO
ENDDO
```

Exercices

- ⋆ Ecrire le produit matrice vecteur dans le cas d'une matrice en CSR.
- ⋆ Ecrire une résolution triangulaire inférieure avec le format CSC puis le format CSR.

Example of elemental matrix format

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

- 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).

Format description can be found at :
`http://math.nist.gov/MatrixMarket/formats.html`

File storage : Rutherford-Boeing

```
DNV-Ex 1 : Tubular joint-1999-01-17                                    M_T1
      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

Gaussian elimination

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

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

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

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

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

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

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 :
  $\mathbf{A}^{(k+1)} = \mathbf{L}^{(k)}\mathbf{A}^{(k)}$, with
  $$\mathbf{L}^k = \begin{pmatrix} \mathbf{1} & & & & & \\ & \ddots & & & & \\ & & \mathbf{1} & & & \\ & & -\mathbf{l_{k+1,k}} & \ddots & & \\ & & \vdots & & \ddots & \\ & & -\mathbf{l_{n,k}} & & & \mathbf{1} \end{pmatrix} \text{ and } l_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}.$$
- Then, $\mathbf{A}^{(n)} = \mathbf{U} = \mathbf{L}^{(n-1)} \dots \mathbf{L}^{(1)}\mathbf{A}$, which gives $\boxed{\mathbf{A} = \mathbf{LU}}$,
  with $\mathbf{L} = [\mathbf{L}^{(1)}]^{-1} \dots [\mathbf{L}^{(n-1)}]^{-1} = \begin{pmatrix} \mathbf{1} & & \mathbf{0} & \\ & \ddots & & \\ & & \ddots & \\ & \mathbf{l_{i,j}} & & \mathbf{1} \end{pmatrix}$,
- In dense codes, entries of $\mathbf{L}$ and $\mathbf{U}$ overwrite entries of $\mathbf{A}$.
- Furthermore, if $\mathbf{A}$ is symmetric, $\boxed{\mathbf{A} = \mathbf{LDL}^{\mathrm{T}}}$ 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 $\mathbf{LU}$ factorization (a_{kk} pivot) :

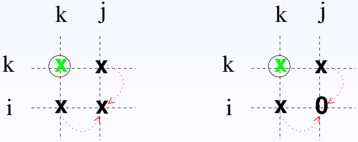
- ⋆ For $i > k$ compute $l_{ik} = a_{ik}/a_{kk}$ ($= a'_{ik}$),
- ⋆ For $i > k, j > k$

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

or

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

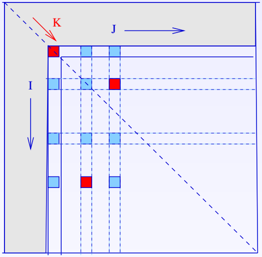
- ⋆ If $a_{ik} \neq 0$ et $a_{kj} \neq 0$ then $a'_{ij} \neq 0$
- ⋆ If a_{ij} was zero $\rightarrow$ its non-zero value must be stored



fill-in

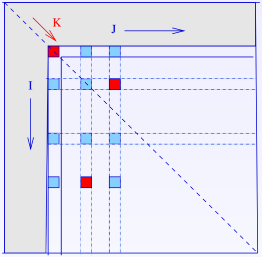
Factorisation LU (version KIJ)

pour $k = 1$ *to* $n - 1$ **faire**
    pour $i = k + 1$ *to* n **faire**
        $a_{ik} := a_{ik}/a_{kk}$
        pour $j = k + 1$ *to* n **faire**
            $a_{ij} := a_{ij} - a_{ik} * a_{kj}$
        fin
    fin
fin



Factorisation LDLt (version KIJ)

pour $k = 1$ *to* $n - 1$ **faire**
    pour $j = k + 1$ *to* n **faire**
        $t := a_{jk}/a_{kk}$
        pour $i = j$ *to* n **faire**
            $a_{ij} := a_{ij} - a_{ik} * t$
        fin
        $a_{jk} := t;$
    fin
fin



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

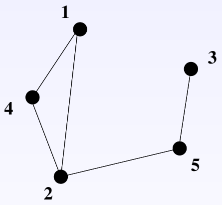
- Assumptions : $\mathbf{A}$ symmetric and pivots are chosen on the diagonal
- Structure of $\mathbf{A}$ symmetric represented by the graph $G = (V, E)$
  - Vertices are associated to columns : $V = \{1, ..., n\}$
  - Edges E are defined by : $(i, j) \in E \leftrightarrow a_{ij} \neq 0$
  - G undirected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

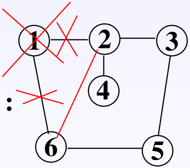
Let v_i denote the vertex of index i. $G_0 = G(\mathbf{A})$, $i = 1$.

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*.

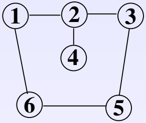
G0 :



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

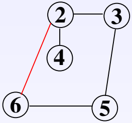
A sequence of elimination graphs

G0 :



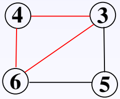
$$H0 = \begin{bmatrix} 1 & \times & & & & \times \\ \times & 2 & \times & \times & & \\ & \times & 3 & & \times & \\ & \times & & 4 & & \\ & & \times & & 5 & \times \\ \times & & & & \times & 6 \end{bmatrix}$$

G1 :



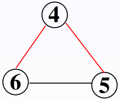
$$H1 = \begin{bmatrix} 2 & \times & \times & & + \\ \times & 3 & & \times & \\ \times & & 4 & & \\ & \times & & 5 & \times \\ + & & & \times & 6 \end{bmatrix}$$

G2 :



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

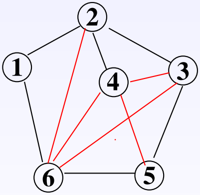
G3 :



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

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 $(v_k, v_j) \in G^+$.



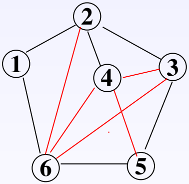
G$^+$(A) = G(F)

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

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 :



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

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

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

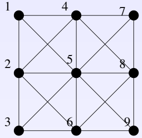
Fill-in theorem

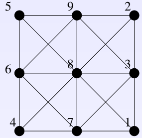
Theorem

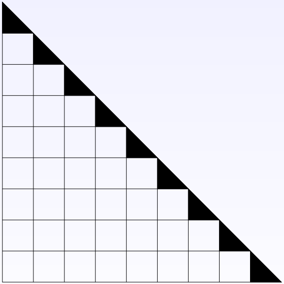
Any $A_{ij} = 0$ will become a non-null entry L_{ij} or $U_{ij} \neq 0$ in $A = L.U$ if and only if it exists a path in $G_A(V, E)$ from vertex i to vertex j that only goes through vertices with a lower number than i and j.

Exercice

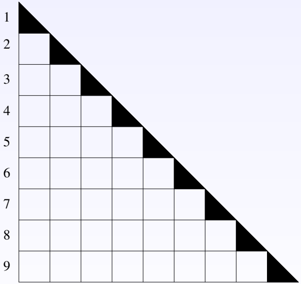
Trouver toutes les termes de remplissages.







Matrice 3x3 : 1ere numerotation



Matrice 3x3 : 2eme numerotation

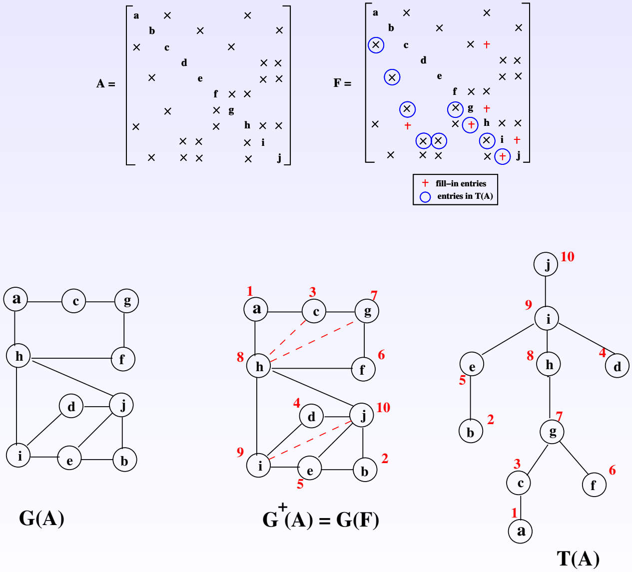
A first definition of the elimination tree

- 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, $\mathbf{A} = \mathbf{L}\mathbf{L}^{\mathrm{T}}$ its Cholesky factorization, and $G^{+}(\mathbf{A})$ its filled graph (graph of $\mathbf{F} = \mathbf{L} + \mathbf{L}^{\mathrm{T}}$).

Definition

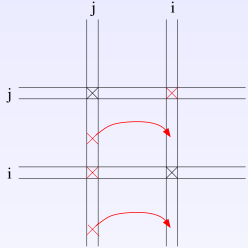
The **elimination tree** of $\mathbf{A}$ is a spanning tree of $G^{+}(\mathbf{A})$ satisfying the relation $PARENT[j] = min\{i > j | l_{ij} \neq 0\}$.

Graph structures



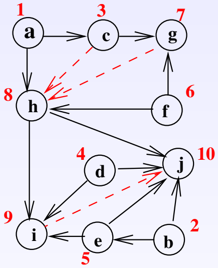
Properties of the elimination tree

- $\boxed{\text{Another perspective}}$ also leads to the elimination tree

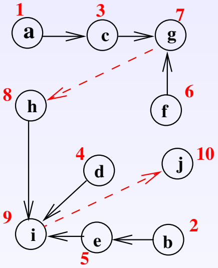


- Dependency between columns of $\mathbf{L}$:
  1. Column $i > j$ depends on column j iff $l_{ij} \neq 0$
  2. Use a directed graph to express this dependency
  3. Simplify redundant dependencies (*transitive reduction* in graph theory)
- *The transitive reduction of the directed filled graph gives the elimination tree structure*

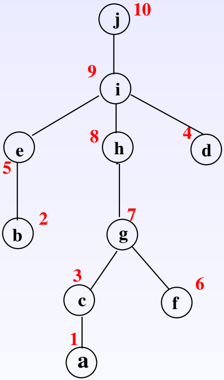
Directed filled graph and its transitive reduction



Directed filled graph



Transitive reduction



T(A)

Exercice

Trouver l'arbre d'élimination des 2 matrices de l'exercice précédent.

Major steps for solving sparse linear systems

There are 4 steps :

1. Reordering :find a (symmetric) permutation P such that it minimizes fill-in in the factorization of $P.A.P^t$. Furthermore, in a parallel context, it should create as much as possible independent computation tasks.
2. Symbolic factorization : this step aims at computing the non-zeros structure of the factors before the actual numeric factorization. It avoids to manage a costly dynamic structure and allows to do some load-balancing.
3. Numerical factorization : compute the factorization by using the preallocated structure from the symbolic factorization.
4. Triangular solve : obtain the solution of $A.x = L.(U.x) = b$. Forward solve $L.y = b$ then a backward solve $U.x = y$. In some cases, it is required to use iterative refinements to increase the accuracy of the solution.

Outline

2. Ordering sparse matrices

Fill-reducing orderings

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

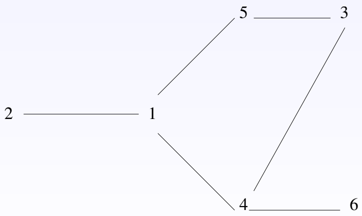
- Selection of next best pivot (e. g. : minimum degree for symmetric matrices).
- Cuthill-McKee (block tridiagonal matrix)
- Nested dissections ("block bordered" matrix).

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

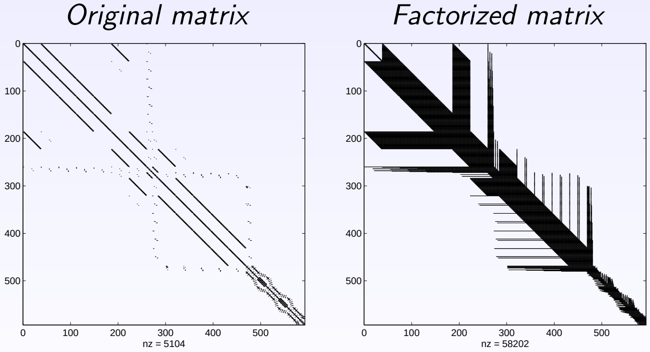
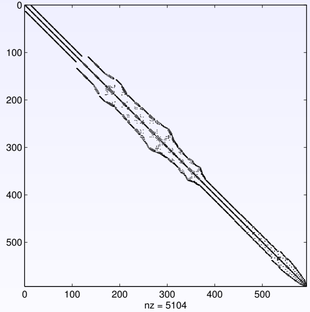


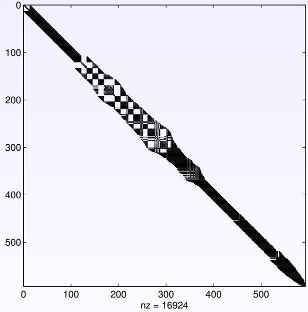
Illustration : Reverse Cuthill-McKee on matrix dwt_592.rua

NZ(LU factors)=16924

Permuted matrix (RCM)



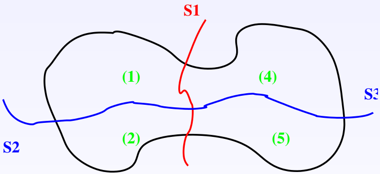
Factorized permuted matrix



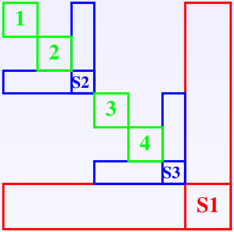
Nested Dissection

Recursive approach based on graph partitioning.

Graph partitioning



Permuted matrix



Nested Dissection : Algorithm

G(V,E) is the adjacency graph of A (V = vertices, E = edges).
In the recursive algorithm k is a global variable initialized to $n = \text{card}(G)$.
It represented the next number to be given.

NestedDissection(G) :
if *G non dissecable* **then**
    Number the vertices of V from k to $k := k - |V| + 1$
end
else
    Find a partition $V = A \bigcup B \bigcup S$ with S a separator of G
    Number the vertices of S from k to $k := k - |S| + 1$
    NestedDissection(G(A)) :
    NestedDissection(G(B)) :
end

Ordering : efficient strategy

The modern software (e.g. METIS http://glaros.dtc.umn.edu/gkhome/views/metis/ or SCOTCH http://www.labri.fr/perso/pelegrin/scotch/) are based on on the nested dissection algorithm but :

- ⋆ they use hybrid ordering ND + local heuristics (e.g. Minimum degree) ;
- ⋆ they use multilevel approaches : graph coarsening, reordering on the reduced graph, graph uncoarsening.

Outline

3. Symbolic factorization
   - Symbolic factorization : column algorithm
   - Symbolic factorization : column-block algorithm
   - Improve the block structure : amalgamation technic

Outline

Symbolic factorization

The goal of this algorithm is to build the non-zero pattern of L (and U). We will consider the symmetric case (graph of $A + A^t$ if A has an unsymmetric NZ pattern). In this case the symbolic factorization is really cheaper than the factorization algorithm.

Fundamental property

The symbolic factorization relies on the elimination tree of A.

Symbolic factorization : Algorithm (1/3)

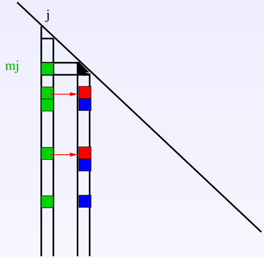
For a sparse matrix A we will denote by :

Definition

Row$(A_{i*}) = \{k < i / A_{ik} \neq 0\}$, for $i = 1..n$
Col$(A_{*j}) = \{k > j / A_{kj} \neq 0\}$, for $j = 1..n$
We will denote by SRow and SCol the sorted set.

Symbolic factorization : Algorithm (2/3)

for $j = 1$ *to* $n - 1$ **do** Build SCol(A_{*j})
for $j = 1$ *to* $n - 1$ **do**
    $m_j :=$ first elt of SCol(A_{*j})
    SCol$(A_{*m_j}) :=$ Merge(SCol(A_{*m_j}), SCol$(A_{*j}) - m_j$)
end



Symbolic factorization : Algorithm (3/3)

At the end of algorithm we have :
SCol(A_{*j}) for $j = 1..n$
The algorithm uses two loops :

Complexity

The complexity of the symbolic factorization is in $O(\|E^*\|)$ the number of edges in the elimination graph.

Outline

3. Symbolic factorization
   - Symbolic factorization : column algorithm
   - Symbolic factorization : column-block algorithm
   - Improve the block structure : amalgamation technic

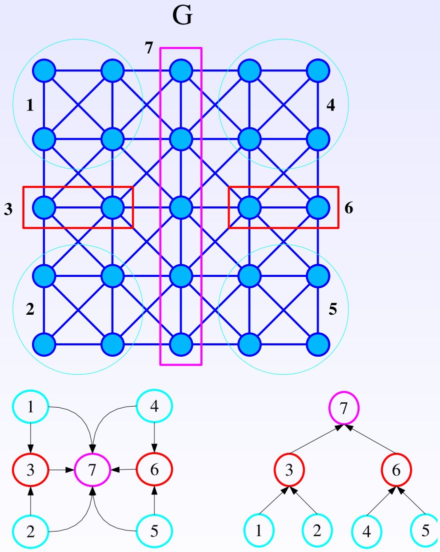
Block Symbolic factorization

The problem in the symbolic factorization is the memory needs.
It is of the same order than the factorization.
In fact, we can use the partition deduced from the ordering to compute a block structure of the matrix.

Definition

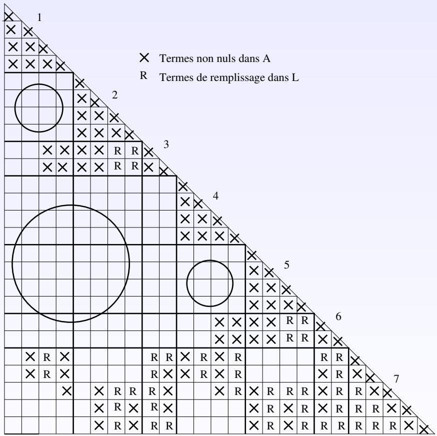
A *supernode* (or supervariable) is a set of contiguous columns in the factors $\mathbf{L}$ that share essentially the same sparsity structure.

Quotient graph and block elimination tree

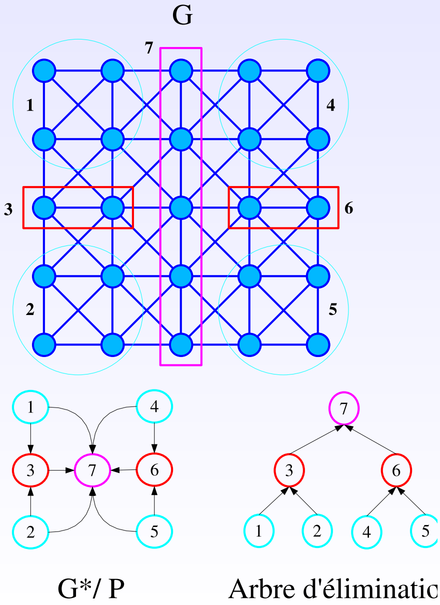


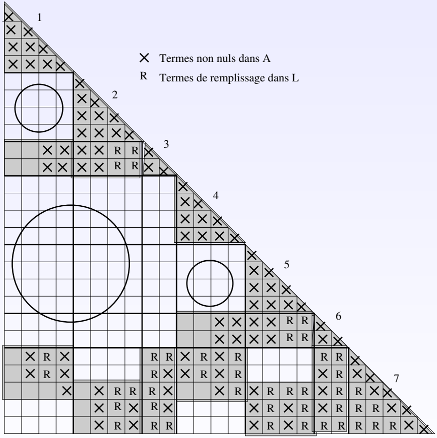
G*/ P

Arbre d'éliminatic



Quotient graph and block elimination tree



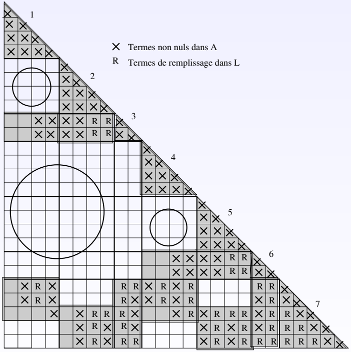


Quotient graph and block elimination tree

The block symbolic factorization relies on

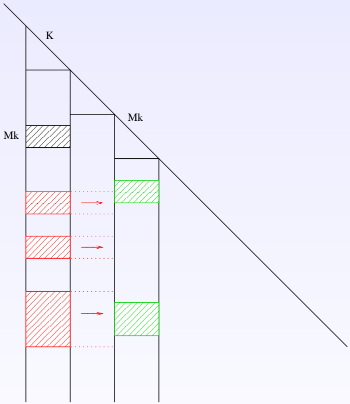
Property of the elimination graph

$$Q(G,P)^* = Q(G^*,P)$$



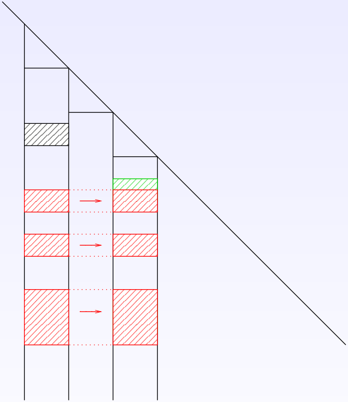
Block Symbolic factorization : Algo

for $k = 1$ *to* $N - 1$ **do**
  Build I_k = the list of block intervals
end
for $k = 1$ *to* $N - 1$ **do**
  $m_k := n(k, 1)$ *(first extra-diagonal block in* k*)*
  $I_{m_k} := \text{Merge}(I_{m_k}, (I_k - [m_k]))$
end



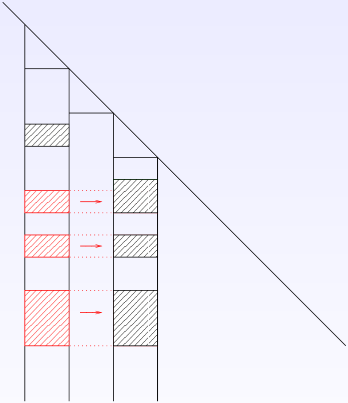
Block Symbolic factorization : Algo

for $k = 1$ *to* $N - 1$ **do**
    Build I_k = the list of block intervals
end
for $k = 1$ *to* $N - 1$ **do**
    $m_k := n(k, 1)$ *(first extra-diagonal block in k)*
    $I_{m_k} :=$ Merge$(I_{m_k}, (I_k - [m_k]))$
end



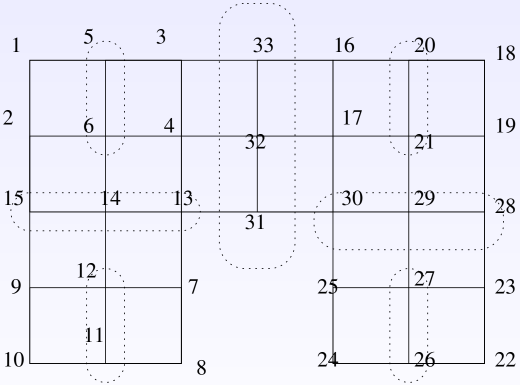
Block Symbolic factorization : Algo

for $k = 1$ *to* $N - 1$ **do**
  Build I_k = the list of block intervals
end
for $k = 1$ *to* $N - 1$ **do**
  $m_k := n(k, 1)$ *(first extra-diagonal block in k)*
  $I_{m_k} := \text{Merge}(I_{m_k}, (I_k - [m_k]))$
end



Exercice

Faire la factorisation par bloc de la matrice correspondant au graphe :



Outline

3. Symbolic factorization

- Symbolic factorization : column algorithm
- Symbolic factorization : column-block algorithm
- Improve the block structure : amalgamation technic

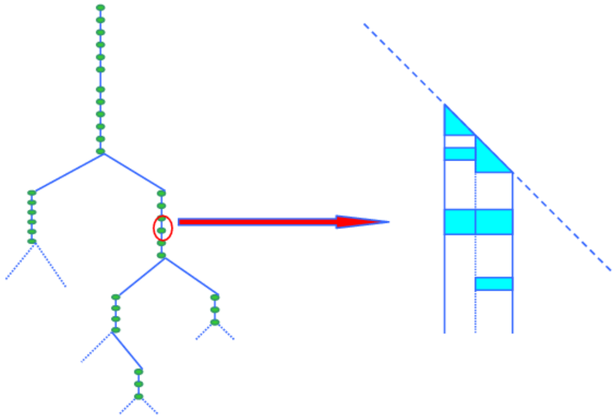
Algorithme d'amalgamation

- Première étape : chercher les supernœuds que l'on peut exhiber de la structure symbolique.
- Cela ne suffit généralement pas (blocs trop petits pour une bonne efficacité BLAS3).
- Idée : admettre du remplissage supplémentaire pour former des supernœuds (et donc des blocs plus gros).
- Ex : réduire le nombre de supernœuds au mieux en autorisant 10 % de remplissage supplémentaire (éventuellement des vrais zéros)

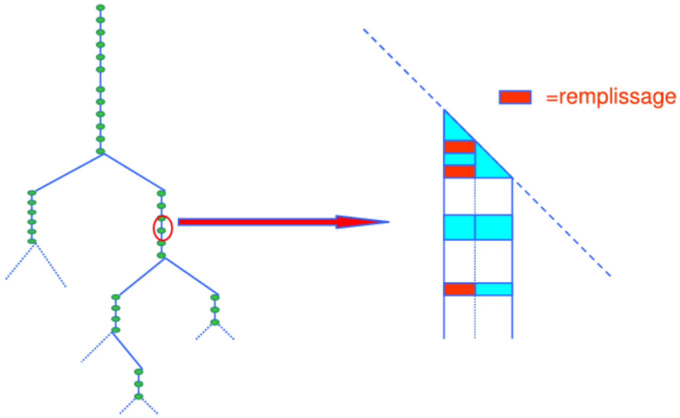
Algorithme d'amalgamation

Initialisation : Rechercher la partition P_0 des supernœuds naturels
tant que *le remplissage* $< tol$ **faire**
    Chercher le couple de supernœuds (fils/père) $(S_1, \text{father}(S_1))$ qui peuvent être regroupés en ajoutant le moins de remplissage
    $S_1' =$ fusion(S_1, father(S_1))
    Mettre à jour le coût de fusion $(\text{son}(S_1), S_1)$ et $(S_1, \text{father}(S_1))$
fin

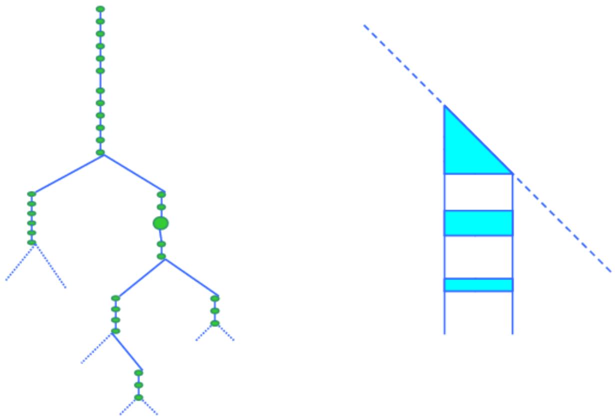
Algorithme d'amalgamation



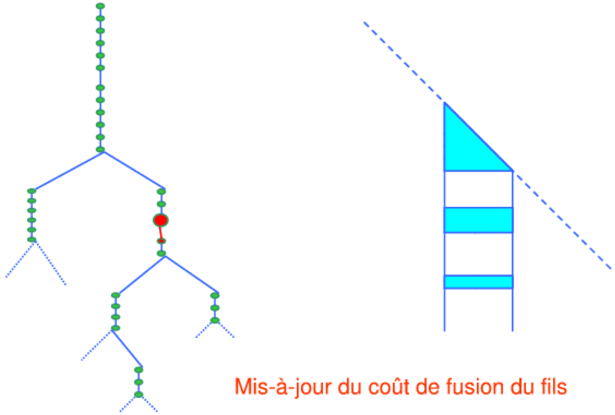
Algorithme d'amalgamation



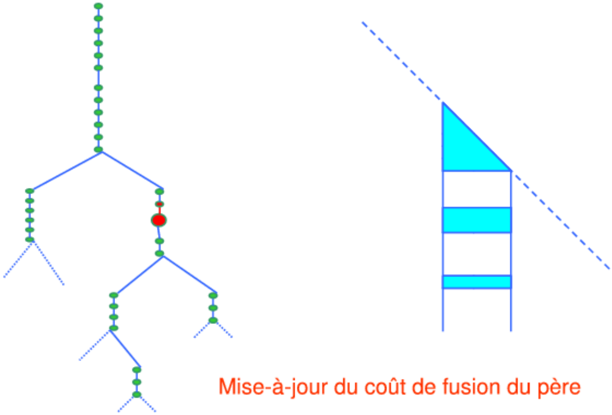
Algorithme d'amalgamation



Algorithme d'amalgamation

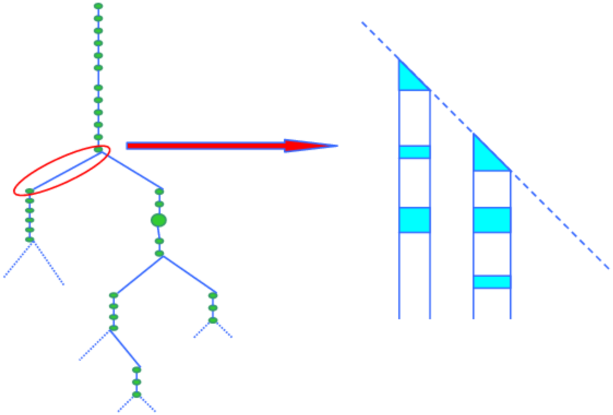


Algorithme d'amalgamation

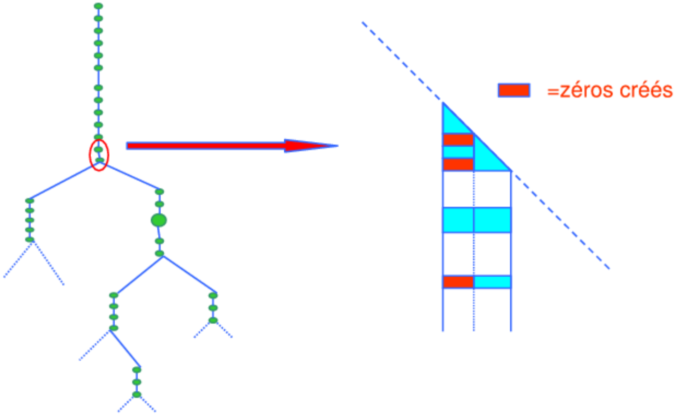


Mise-à-jour du coût de fusion du père

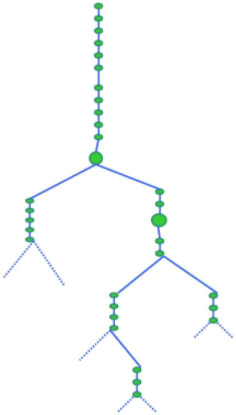
Algorithme d'amalgamation

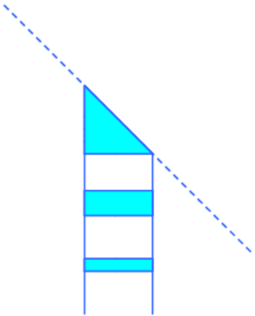


Algorithme d'amalgamation

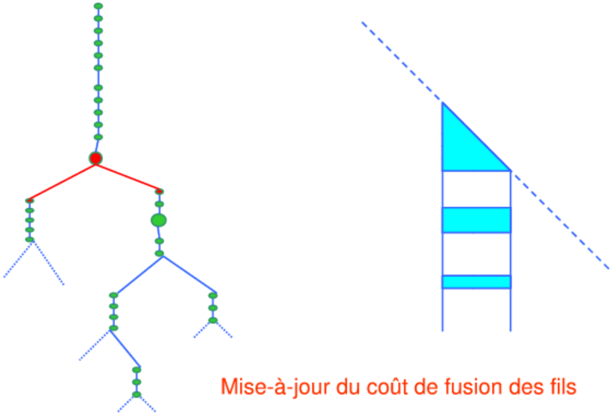


Algorithme d'amalgamation



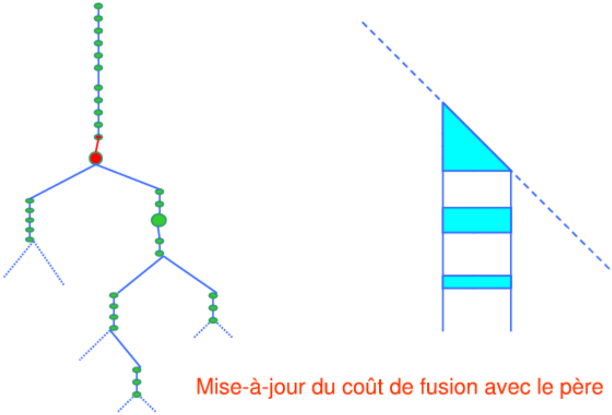


Algorithme d'amalgamation



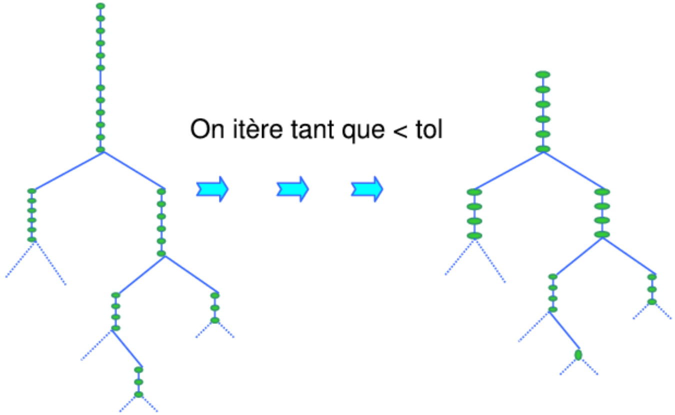
Mise-à-jour du coût de fusion des fils

Algorithme d'amalgamation



Mise-à-jour du coût de fusion avec le père

Algorithme d'amalgamation



Algorithme d'amalgamation

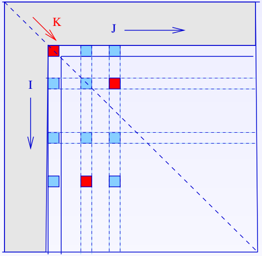
- Algorithme très peu coûteux en regard des autres étapes :
  - A chaque fusion de supernœuds on doit mettre à jour le "gain" de remplissage (père/fils) ainsi que celui de l'ascendant et des fils du nouveau supernœud ;
  - complexité bornée par $0(D.N_0 + N_0.Log(N_0))$ avec
    N_0 le nombre de supernœud exacts
    D le nombre maximal de blocs extradiagonaux dans un supernœud.

Outline

4. Block factorizations
   - Parallelization of Direct methods

Factorisation LDLt (Cholesky) : scalar version

pour $k = 1$ *to* $n - 1$ **faire**
    pour $j = k + 1$ *to* n **faire**
        $t := a_{jk}/a_{kk}$
        pour $i = j$ *to* n **faire**
            $a_{ij} := a_{ij} - a_{ik} * t$
        fin
        $a_{jk} := t;$
    fin
fin



Cholesky block factorization

pour $k := 1 \ldots N$ **faire**

  Factoriser $A_{k,k}$ en $L_k.D_k.L_k^t$

  pour $p := 1 \ldots s_k$ **faire**

  $A_{(p),k} := A_{(p),k}.(L_k^t)^{-1}$

  pour $p := 1 \ldots s_k$ **faire**

    $F := A_{(p),k}.D_k^{-1}$

    pour $q := p \ldots s_k$ **faire**

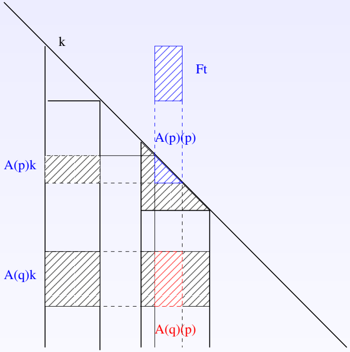
      $A_{(q),(p)} := A_{(q),(p)} - A_{(q),k}.F^t$

    fin

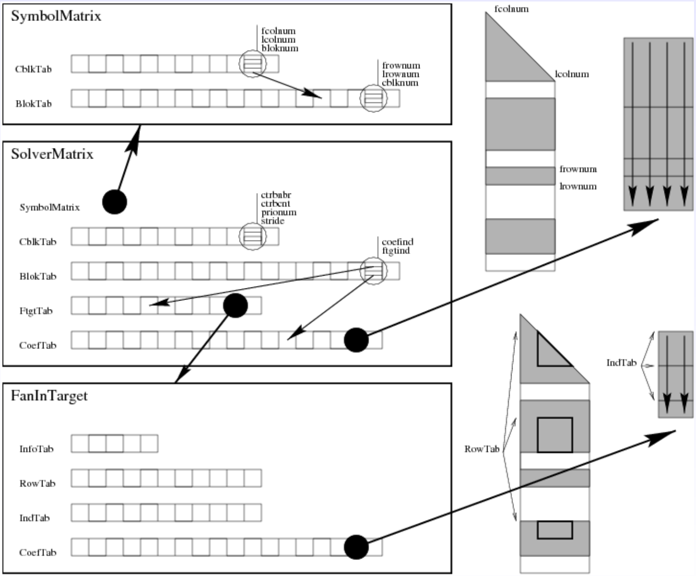
    $A_{(p),k} := F$

  fin

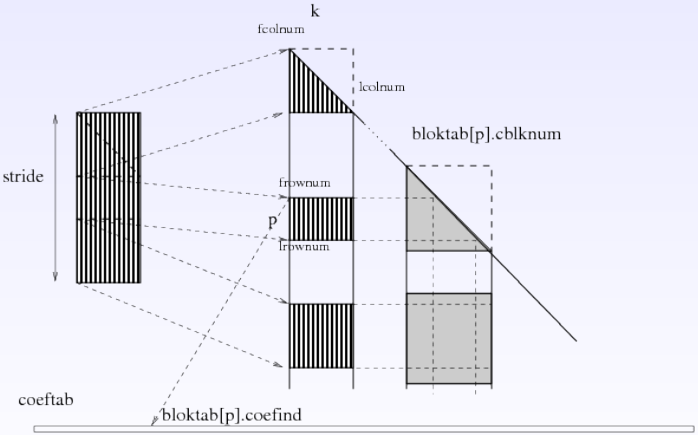
fin



Cholesky block factorization : implementation



Cholesky block factorization : implementation

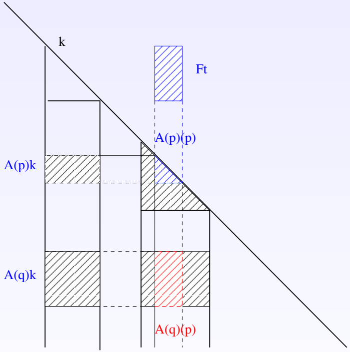


Cholesky block factorization

pour $k := 1 \dots N$ **faire**

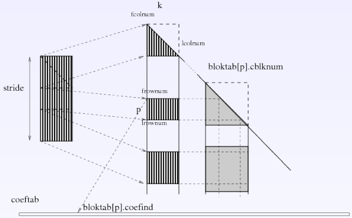
- Factoriser $A_{k,k}$ en $L_k.D_k.L_k^t$
- **pour** $p := 1 \dots s_k$ **faire** $A_{(p),k} := A_{(p),k}.(L_k^t)^{-1}$
- **pour** $p := 1 \dots s_k$ **faire**
  - $F := A_{(p),k}.D_k^{-1}$
  - **pour** $q := p \dots s_k$ **faire**
    - $A_{(q),(p)} := A_{(q),(p)} - A_{(q),k}.F^t$
  - **fin**
  - $A_{(p),k} := F$
- **fin**

fin



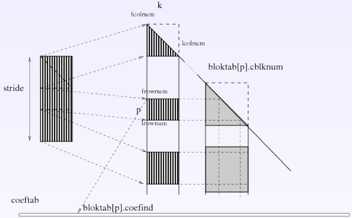
Cholesky block factorization

pour $k := 1 \ldots N$ **faire**
  Factoriser $A_{k,k}$ en $L_k.D_k.L_k^t$
  pour $p := 1 \ldots s_k$ **faire**
  $A_{(p),k} := A_{(p),k}.(L_k^t)^{-1}$
  pour $p := 1 \ldots s_k$ **faire**
    $F := A_{(p),k}.D_k^{-1}$
    pour $q := p \ldots s_k$ **faire**
      $A_{(q),(p)} := A_{(q),(p)} - A_{(q),k}.F^t$
    fin
    $A_{(p),k} := F$
  fin
fin



Cholesky block factorization

pour $k := 1 \dots N$ **faire**
    Factoriser $A_{k,k}$ en $L_k.D_k.L_k^t$
    $A_{p=1..s_k,k} := A_{p=1..s_k,k}.(L_k^t)^{-1}$
    pour $p := 1 \dots s_k$ **faire**
        $F := A_{(p),k}.D_k^{-1}$
        $W := A_{(q=p..s_k),k}.F^t$
        pour $q := p \dots s_k$ **faire**
            $A_{(q),(p)} := A_{(q),(p)} - W_{(q),k}.F^t$
        fin
        $A_{(p),k} := F$
    fin
fin



Outline

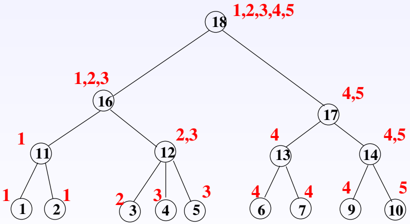
3 levels of parallelism

1. from the sparsity : use the elimination tree (subtrees can be computed indepedently)
2. inside large supernode : dense block factorization (like in scalapack). We need to split the column-block.
3. last level is ensured by using BLAS routines ; scalar pipelines at the microprocessor level.

Static scheduling : Proportional mapping

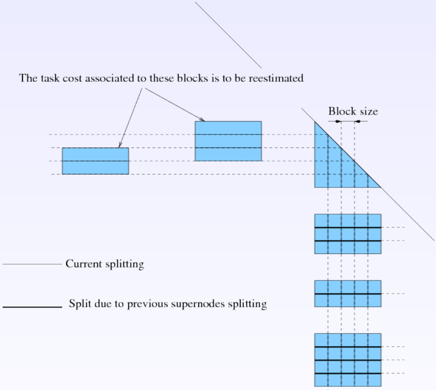
Main objective : reduce the volume of communication between processors.

- 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

Elimination tree repartitioning



Parallel factorization : notations

After the proportional mapping step, each column-block is assigned to a processor. Let denote by map the distribution function.
map(k) the processor responsible for column block k.

Parallel factorization : notations

In order to manage the message receive for the column-block k we need to know :

- ⋆ Row(k) = set of column-block from which the column-block k receive some contributions.
- ⋆ Col(k) = set of column-block from which the column-block k receive some contributions.

And the processors on which the column-block in Row(k) and Col(k) are distributed.
We denote by $n(k,p)$ the number of the column block which diagonal block is in the same row than $A_{(p),k}$.

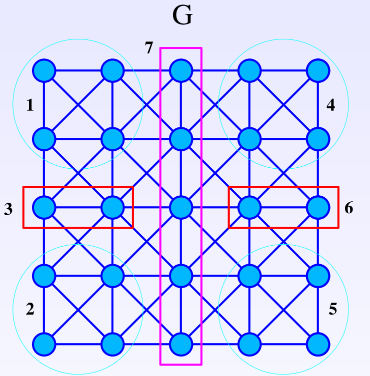
Parallel factorization : notations

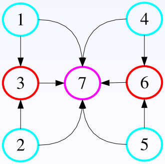
In order to manage the message receive for the column-block k we need to know :

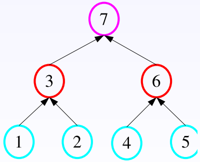
- Row$(A_{i*}) = \{k < i / A_{ik} \neq 0\}$, for $i = 1..n$
- Col$(A_{*j}) = \{k > j / A_{kj} \neq 0\}$, for $j = 1..n$

And the processors on which the column-block in Row(k) and Col(k) are distributed.
We denote by $n(k,p)$ the number of the column block which diagonal block is in the same row than $A_{(p),k}$.

Reminder : quotient graph and block elimination tree

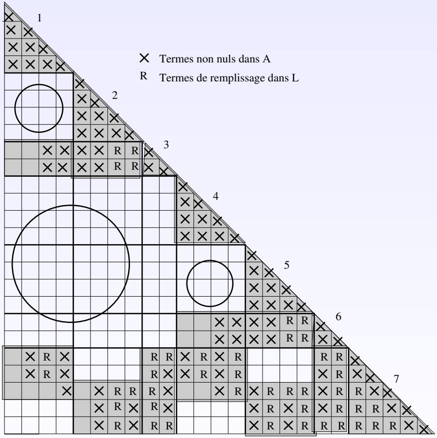






G*/ P

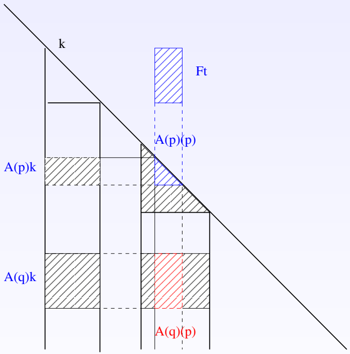
Arbre d'éliminatic



Parallel Cholesky block factorization (1/2)

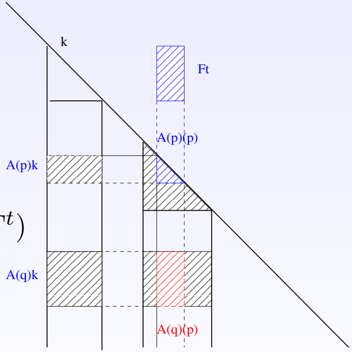
Algorithm for column-block k (executed by processor map(k)) :

$R := Row(k)$
while $R \neq \emptyset$ **do**
    Receive-from-anyone$(c, \text{ind}, \text{bloc})$
    # c is the processor that sent the message
    # ind is the indices and bloc is the values of the contribution block
    if *ind* $\neq$ *EOT* **then**
        $A_{\text{ind}} := A_{\text{ind}} - \text{bloc}$
    else
        $R := R - \{c\}$
    end
end
...



Parallel Cholesky block factorization (2/2)

...
Factorize $A_{k,k}$ in $L_k.D_k.L_k^t$
for $p := 1 \dots s_k$ **do**
    $A_{(p),k} := A_{(p),k}.(L_k^t)^{-1}$
for $p := 1 \dots s_k$ **do**
    $F := A_{(p),k}.D_k^{-1}$
    for $q := p \dots s_k$ **do**
        Send(map($n(k,p)$), $((q),(p))$, $A_{(q),k}.F^t$)
    end
    $A_{(p),k} := F$
    Send(map($n(k,p)$), EOT, $\emptyset$)
end



Parallel Cholesky block factorization (2/2)

...
Factorize $A_{k,k}$ in $L_k.D_k.L_k^t$
for $p := 1 \dots s_k$ **do**
    $A_{(p),k} := A_{(p),k}.(L_k^t)^{-1}$
for $p := 1 \dots s_k$ **do**
    $F := A_{(p),k}.D_k^{-1}$
    for $q := p \dots s_k$ **do**
        Send(map($n(k,p)$), $((q),(p))$, $A_{(q),k}.F^t$)
    end
    $A_{(p),k} := F$
    Send(map($n(k,p)$), EOT, $\emptyset$)
end

This messages can be compacted in a single message (same dest.).