Résolution de systèmes linéaires creux de grande taille pour des applications de simulation

Emmanuel Agullo (LIP - ENS Lyon)
Emmanuel.Agullo@ens-lyon.fr

Some material from P. Amestoy (ENSEEIHT-IRIT) and J.-Y. L'Excellent (INRIA and LIP-ENS Lyon)

Jeudi 9 février 2006

Actualité des Nombres et du Calcul
CRDP Amiens

Plan de l'exposé

Motivations

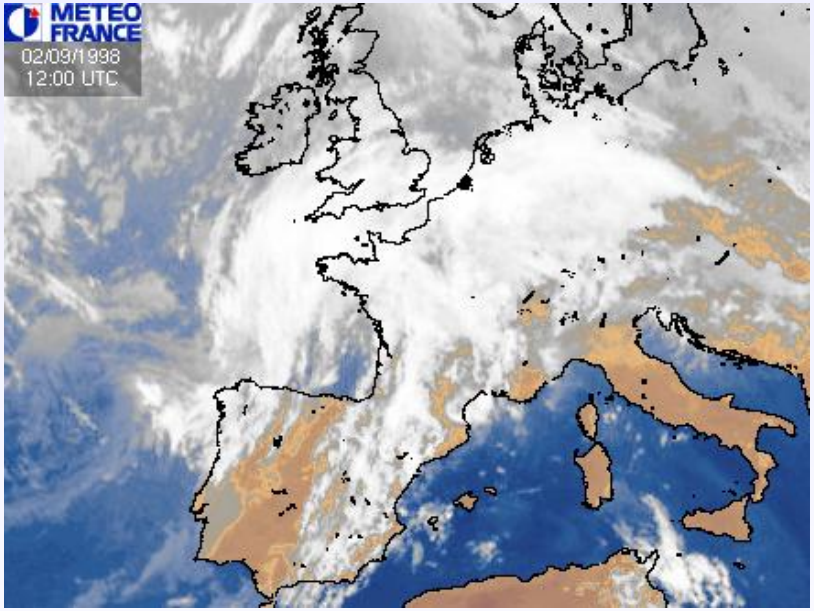
- Besoins croissants des applications en puissance de calcul :
  - simulation,
  - modélisation,
  - optimisation numérique
- Typiquement :
  Problème continu ⇒ Discrétisation (maillage) ⇒ Algorithme numérique de résolution (selon lois physiques)
- Besoins :
  - Modélisations de plus en plus précises
  - Problèmes de plus en plus complexes
  - Applications critiques en temps de réponse
  - Minimisation des coûts du calcul

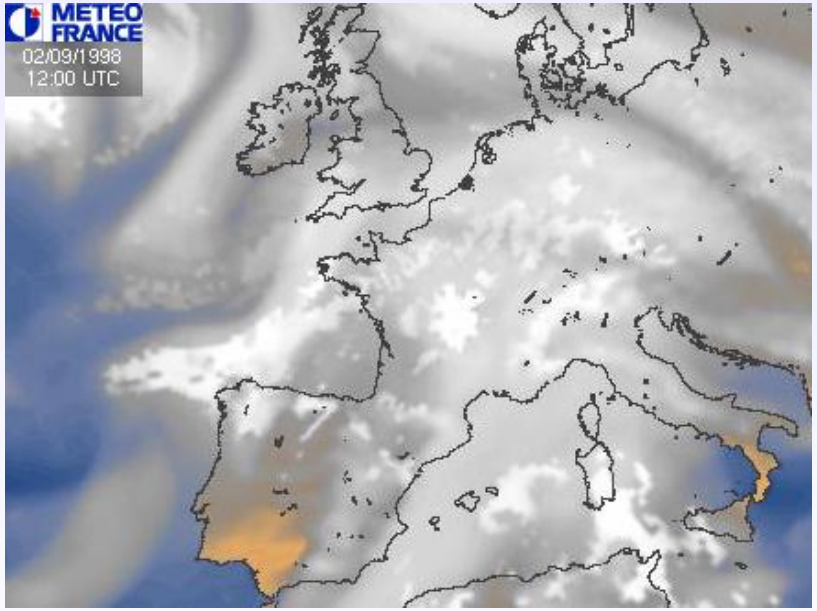
⇒ Calculateurs parallèles / haute performance.

⇒ Algorithmes numériques et outils permettant de tirer le meilleur parti de ces calculateurs.

Quelques exemples dans le domaine du calcul scientifique

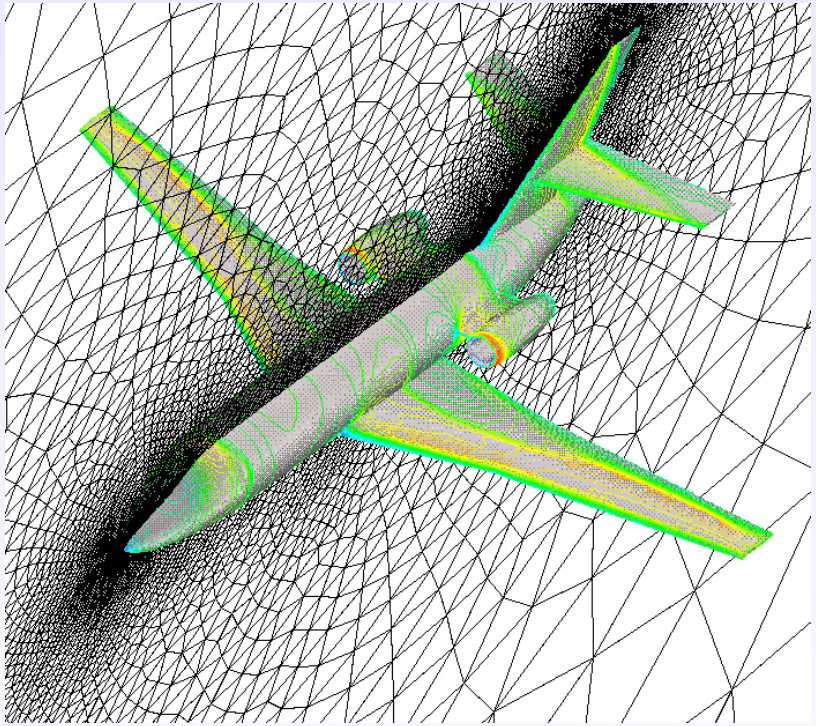
- Contraintes de durée : prévision du climat

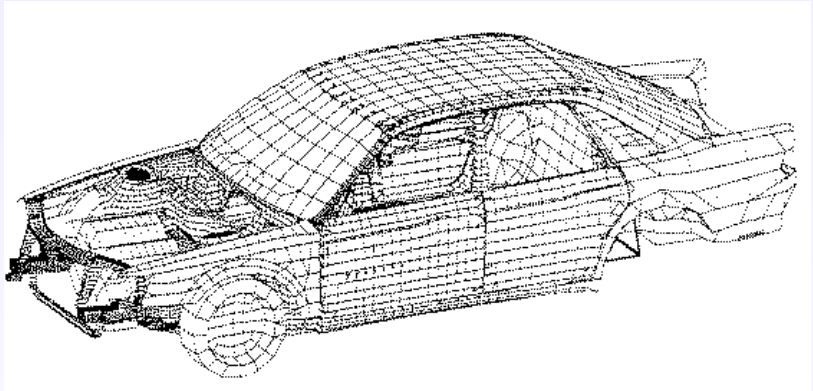


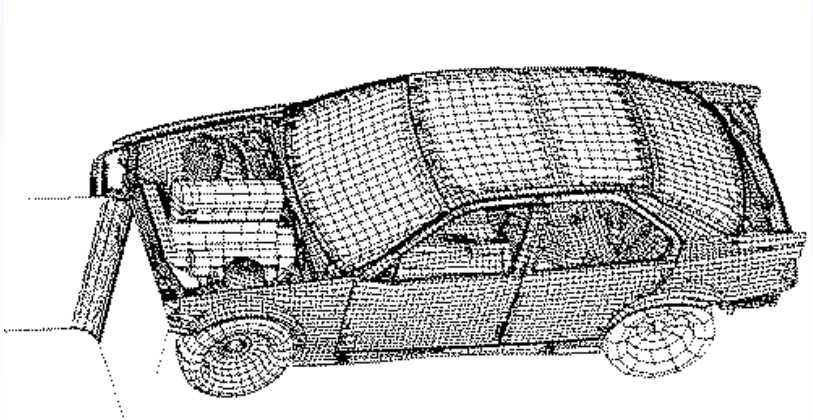


Quelques exemples dans le domaine du calcul scientifique

- Cost constraints : wind tunnels, crash simulation, ...

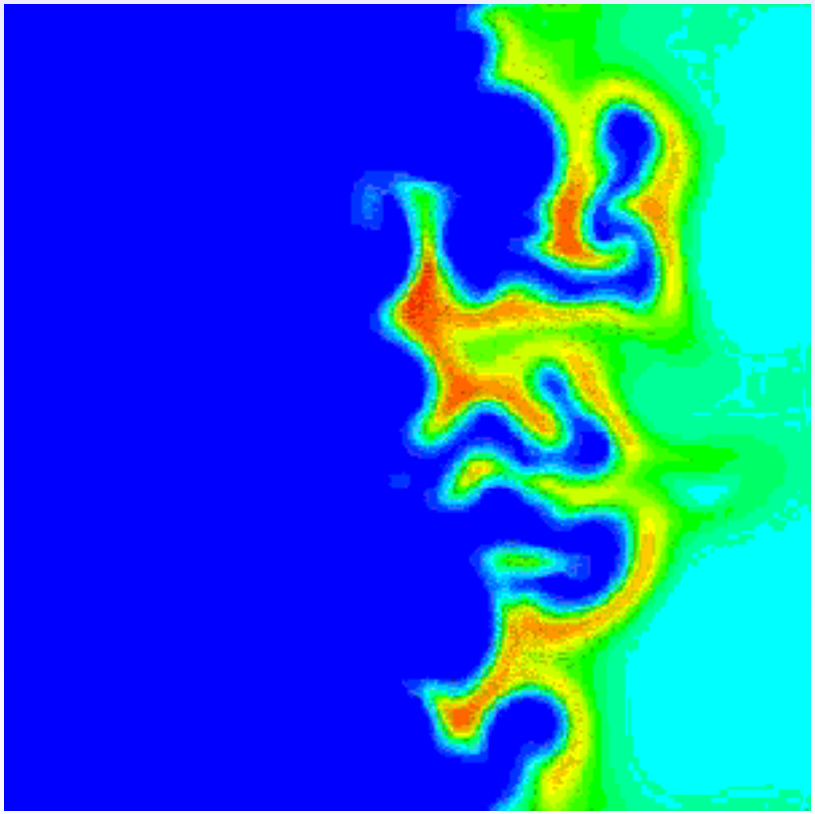






Scale Constraints

- large scale : climate modelling, pollution, astrophysics
- tiny scale : combustion, quantum chemistry



Plan de l'exposé

Plan de l'exposé

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*

Plan de l'exposé

Sparse matrices

Example :

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

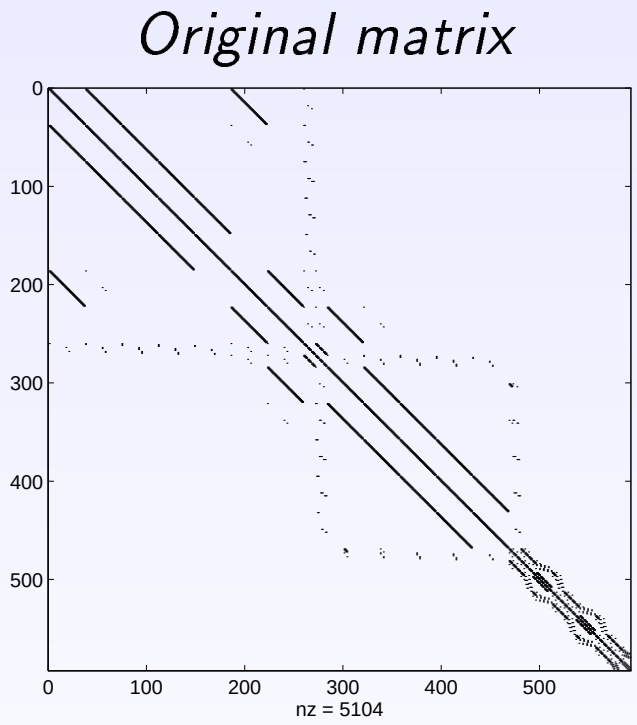
can be represented as

$$\mathbf{A}\mathbf{x} = \mathbf{b},$$

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 ?



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 :
  - $\mathbf{A} = \mathbf{LDL}^{\mathrm{T}}$ or $\mathbf{LL}^{\mathrm{T}}$

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 :
  - $\mathbf{A} = \mathbf{LDL}^{\mathrm{T}}$ or $\mathbf{LL}^{\mathrm{T}}$

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

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

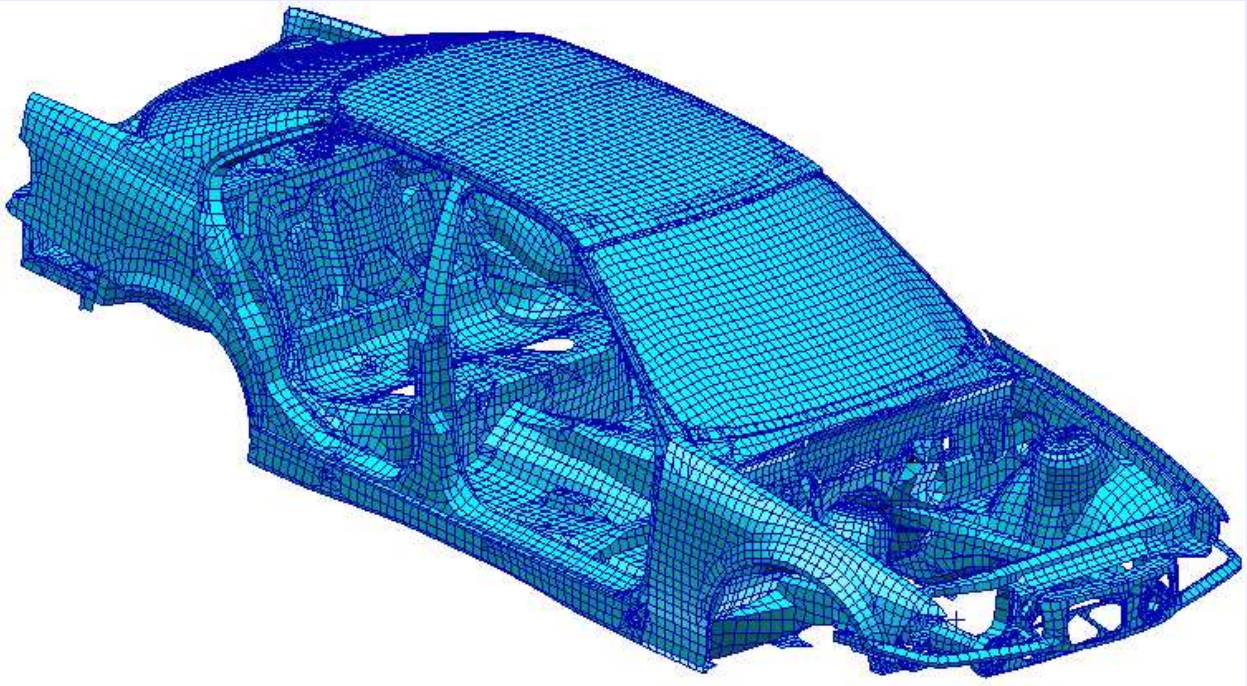
Key numbers :

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

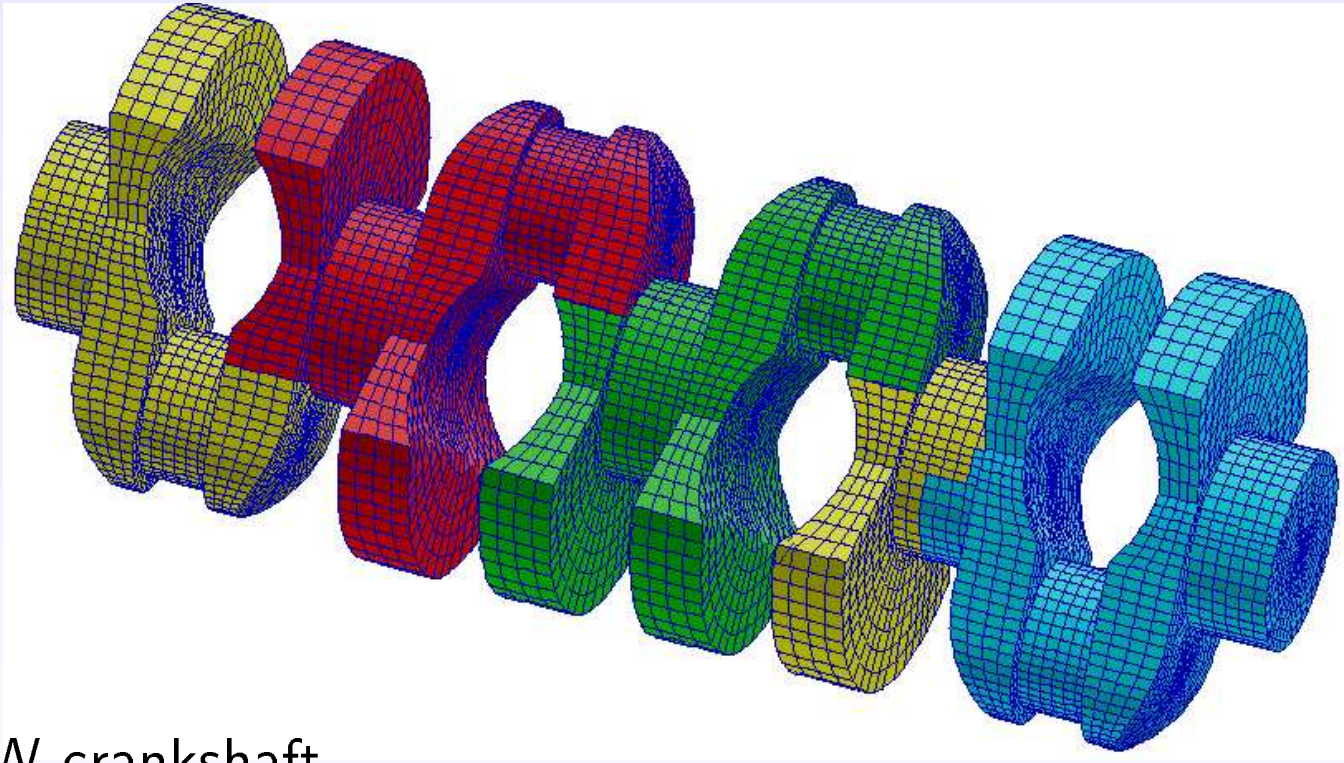
Typical test problems :



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 ?

Assembled matrix : illustration

- Example of a 3x3 matrix with 5 nonzeros

	1	2	3
1	a11		
2		a22	a23
3	a31		a33

- Coordinate format

IRN	$[1:NNZ]$	$=$	1	3	2	2	3
JCN	$[1:NNZ]$	$=$	1	1	2	3	3
A	$[1:NNZ]$	$=$	a_{11}	a_{31}	a_{22}	a_{23}	a_{33}

- Compressed Sparse Column (CSC) format

IRN	$[1:NNZ]$	$=$	1	3	2	2	3
A	$[1:NNZ]$	$=$	a_{11}	a_{31}	a_{22}	a_{23}	a_{33}
IP	$[1:N+1]$	$=$	1	3	4	6	

column J corresponds to IRN/A locations IP(J)...IP(J+1)-1

Assembled matrix : illustration

- Example of a 3x3 matrix with 5 nonzeros

	1	2	3
1	a11		
2		a22	a23
3	a31		a33

- Coordinate format

IRN	$[1:NNZ]$	$=$	1	3	2	2	3
JCN	$[1:NNZ]$	$=$	1	1	2	3	3
A	$[1:NNZ]$	$=$	a_{11}	a_{31}	a_{22}	a_{23}	a_{33}

- Compressed Sparse Column (CSC) format

IRN	$[1:NNZ]$	$=$	1	3	2	2	3
A	$[1:NNZ]$	$=$	a_{11}	a_{31}	a_{22}	a_{23}	a_{33}
IP	$[1:N+1]$	$=$	1	3	4	6	

column J corresponds to IRN/A locations IP(J)...IP(J+1)-1

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).

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
```

Plan de l'exposé

Gaussian elimination

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

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

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

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

<u>Finally</u> $\mathbf{A}^{(3)}\mathbf{x} = \mathbf{b}^{(3)}$

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

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

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
  $$\mathbf{L}^k = \begin{pmatrix} \mathbf{1} & & & & & & \\ & \cdot & & & & & \\ & & \cdot & & & & \\ & & & \mathbf{1} & & & \\ & & & \mathbf{-l_{k+1,k}} & \cdot & & \\ & & & \vdots & & \cdot & \\ & & & \mathbf{-l_{n,k}} & & & \mathbf{1} \end{pmatrix} \text{ and } l_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}.$$
- Then, $\mathbf{A}^{(n)} = \mathbf{U} = \mathbf{L}^{(n-1)} \dots \mathbf{L}^{(1)}\mathbf{A}$, which gives $\boxed{\mathbf{A} = \mathbf{LU}}$,
  with $\mathbf{L} = [\mathbf{L}^{(1)}]^{-1} \dots [\mathbf{L}^{(n-1)}]^{-1} = \begin{pmatrix} \mathbf{1} & & \mathbf{0} & \\ & \cdot & & \\ & & \cdot & \\ & & & \cdot \\ & & \mathbf{l_{i,j}} & & \mathbf{1} \end{pmatrix}$,
- 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 $d_{kk} = a_{kk}^{(k)}$:
  $A = LU = A^t = U^t L^t$ 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 $\mathbf{LU}$ factorization (a_{kk} pivot) :

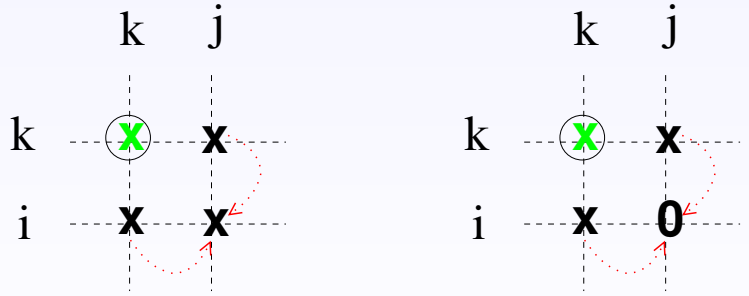
- For $i > k$ compute $l_{ik} = a_{ik}/a_{kk}$ $(= a'_{ik})$,
- For $i > k, j > k$

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

  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

- Idem for Cholesky :
- For $i > k$ compute $l_{ik} = a_{ik}/\sqrt{a_{kk}}$ $(= a'_{ik})$,
- For $i > k, j > k, j \leq i$ (lower triang.)

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

or

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

Example

- Original matrix

```
x     x     x     x     x
x     x
x           x
x                 x
x                       x
```

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

```
X           X
    X       X
        X   X
            X   X
X   X   X   X   X
```

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

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

Procedure	Total storage	Flops	Time (sec.) on CRAY J90
Full Syst.	4084 Kwords	5503×10^6	34.5
Sparse Syst.	71 Kwords	1073×10^6	3.4
Sparse Syst. and reordering	14 Kwords	42×10^3	0.9

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

Effect of switch to dense calculations

Matrix from 5-point discretization of the Laplacian on a 50×50 grid (Dongarra etal 91)

Density for switch to full code	Order of full matrix	Millions of flops	Time (sec.)
No switch	0	7	21.8
1.00	74	7	21.4
0.80	190	8	15.0
0.60	235	11	12.5
0.40	305	21	9.0
0.20	422	50	5.5
0.10	531	100	3.7
0.005	1420	1908	6.1

Sparse structure should be exploited if density $< 10\%$.

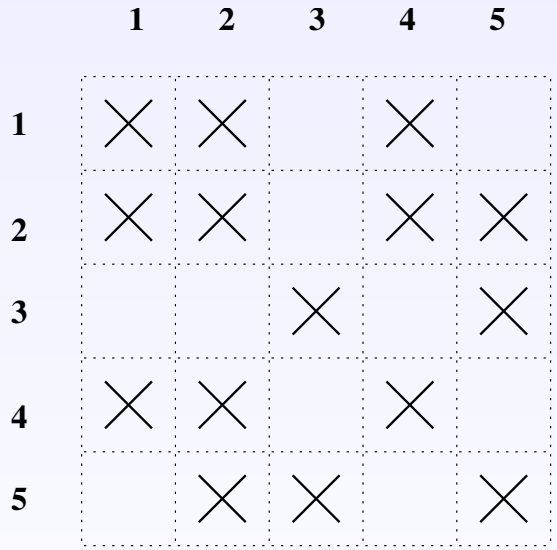
Plan de l'exposé

Symmetric matrices and graphs

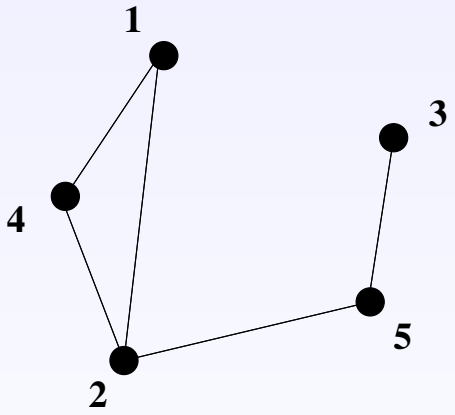
- Assumptions : $\mathbf{A}$ symmetric and pivots are chosen on the diagonal
- Structure of $\mathbf{A}$ symmetric represented by the graph $G^0 = (V^0, E^0)$
  - Vertices are associated to columns : $V^0 = \{1, ..., n\}$
  - Edges E^0 are defined by : $(i, j) \in E^0 \leftrightarrow a_{ij} \neq 0$
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

Symmetric matrices and graphs

- Remarks :
  - Number of nonzeros in column $j = |Adj_{G^0}(j)|$
  - Symmetric permutation $\equiv$ renumbering the graph



Symmetric matrix



Corresponding graph

The elimination graph model

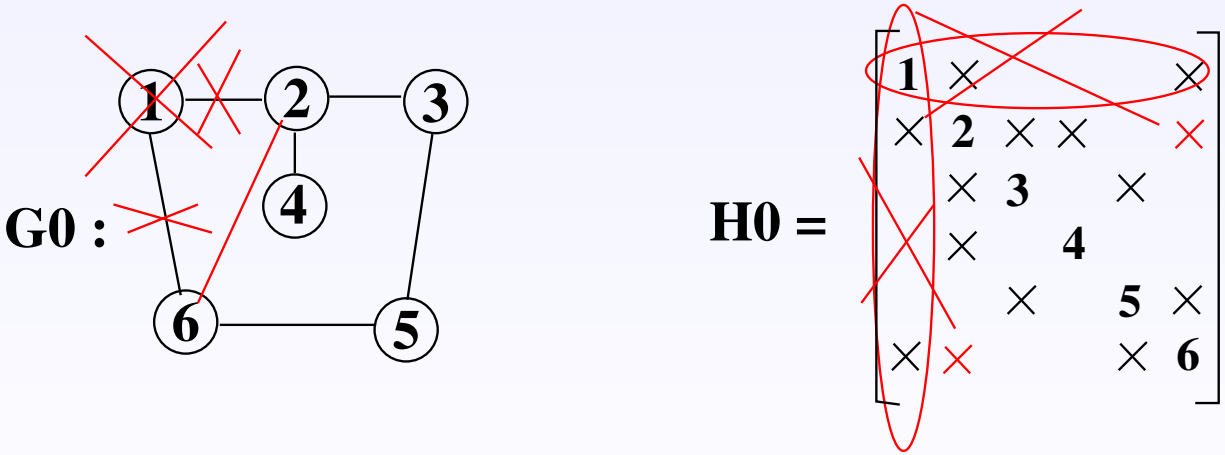
Construction of the elimination graphs

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

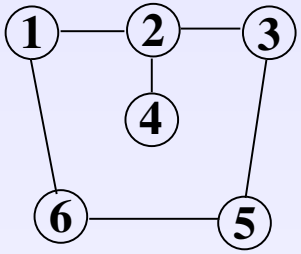
Add edges so that vertices in $Adj(v_i)$ are pairwise adjacent in $G_i = G(\mathbf{H}_i)$.

G_i are the so-called *elimination graphs*.



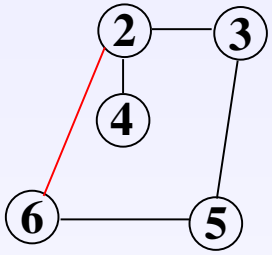
A sequence of elimination graphs

G0 :



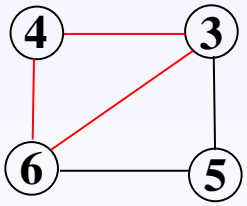
$$H0 = \begin{bmatrix} 1 & \times & & & & \times \\ \times & 2 & \times & \times & & \\ & \times & 3 & & \times & \\ & \times & & 4 & & \\ & & \times & & 5 & \times \\ \times & & & & \times & 6 \end{bmatrix}$$

G1 :



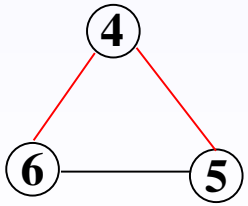
$$H1 = \begin{bmatrix} 2 & \times & \times & & + \\ \times & 3 & & \times & \\ \times & & 4 & & \\ & \times & & 5 & \times \\ + & & & \times & 6 \end{bmatrix}$$

G2 :



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

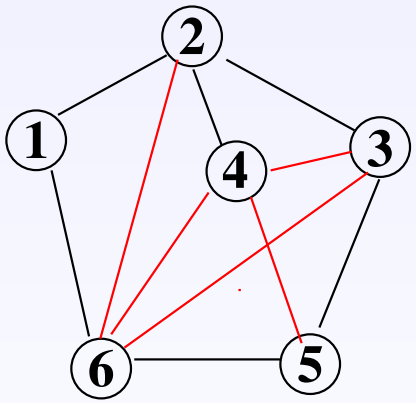
G3 :



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

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

- Let $\mathbf{F} = \mathbf{L} + \mathbf{L}^{\mathrm{T}}$ 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 $(v_k, v_j) \in G^+$.



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

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

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

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 $\mathbf{F} = \mathbf{L} + \mathbf{L}^{\mathrm{T}}$).

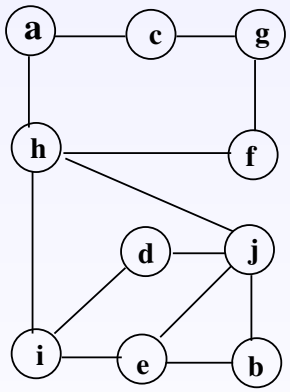
Definition

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

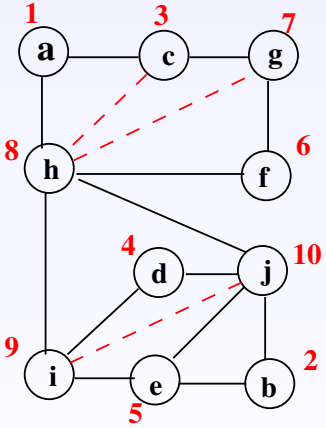
Graph structures

$$A = \begin{bmatrix} a & \times & & & & & \times & & \\ & b & & \times & & & & & \times \\ \times & & c & & & \times & & & \\ & & & d & & & & \times & \times \\ & \times & & & e & & & \times & \times \\ & & & & & f & \times & \times & \\ & & \times & & & \times & g & & \\ \times & & & & & \times & & h & \times & \times \\ & & & \times & \times & & & \times & i & \\ & \times & & \times & \times & & & \times & & j \end{bmatrix} \qquad F = \begin{bmatrix} a & \times & & & & & \times & & \\ & b & & \times & & & & & \times \\ \times & & c & & & \times & + & & \\ & & & d & & & & \times & \times \\ & \times & & & e & & & \times & \times \\ & & & & & f & \times & \times & \\ & & \times & & & \times & g & + & \\ \times & & + & & & \times & + & h & \times & \times \\ & & & \times & \times & & & \times & i & + \\ & \times & & \times & \times & & & \times & + & j \end{bmatrix}$$

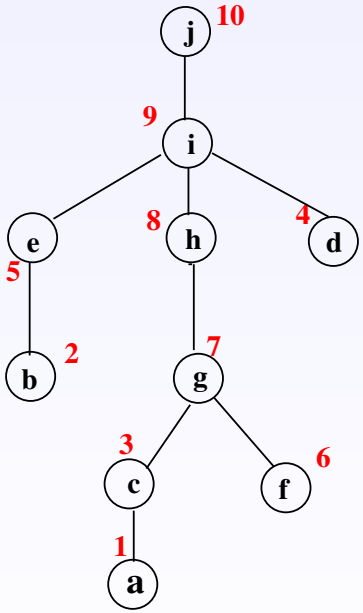
+ fill–in entries
○ entries in T(A)



G(A)



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

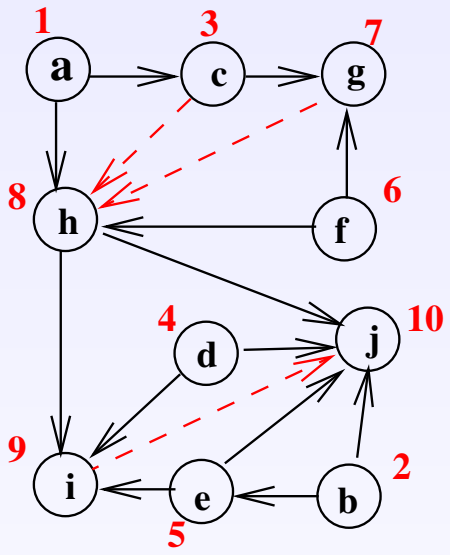


T(A)

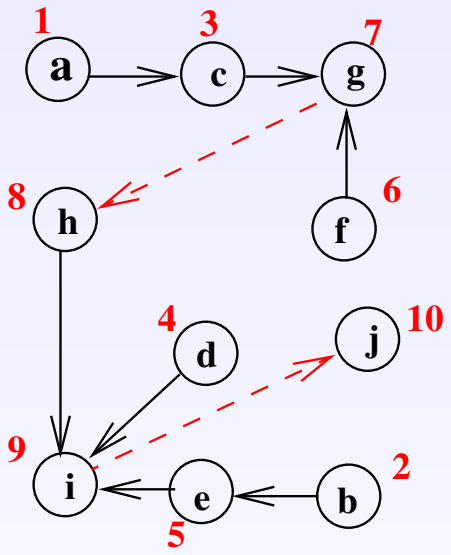
Properties of elimination tree

- 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*

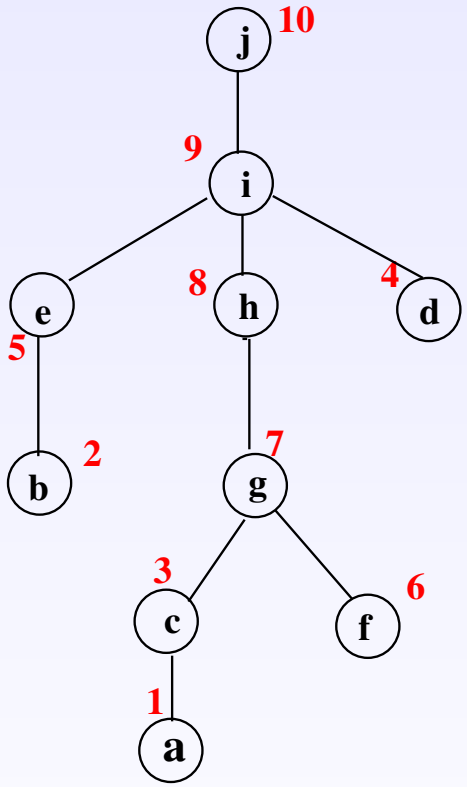
Directed filled graph and its transitive reduction



Directed filled graph



Transitive reduction



T(A)

Plan de l'exposé

Ordering sparse matrices

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

Plan de l'exposé

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

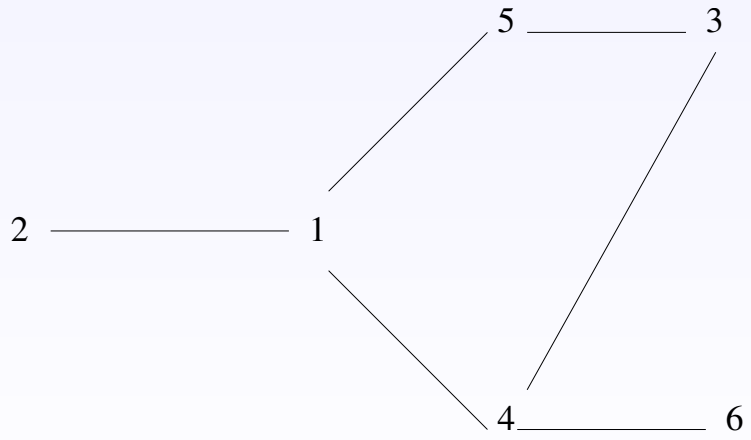
- Local heuristics : At each step of the factorization, selection of the pivot that is likely to minimize fill-in.
  - Method is characterized by the way pivots are selected.
  - Markowitz criterion (for a general matrix).
  - Minimum degree (for symmetric matrices).
- Hybrid approaches : Once the matrix 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 :

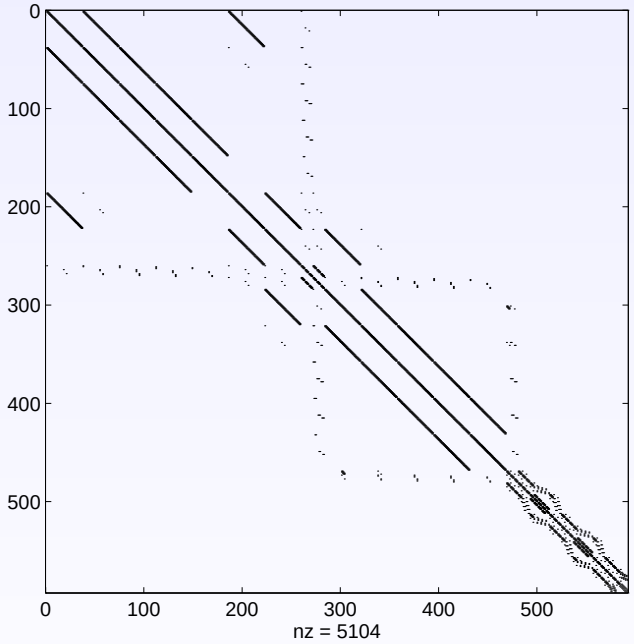
$$A = \begin{bmatrix} x & & & x & & \\ & x & x & x & & \\ & x & x & & x & \\ x & x & & x & x & \\ & & x & x & x & x \\ & & & & x & x \end{bmatrix}$$

- 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

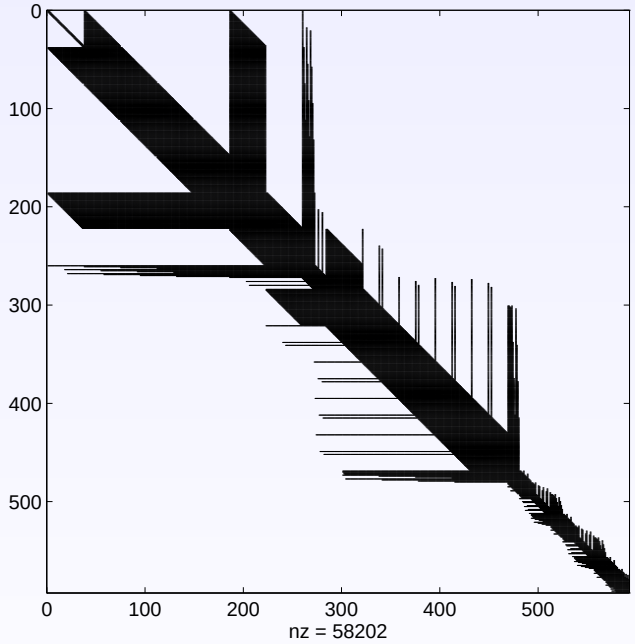
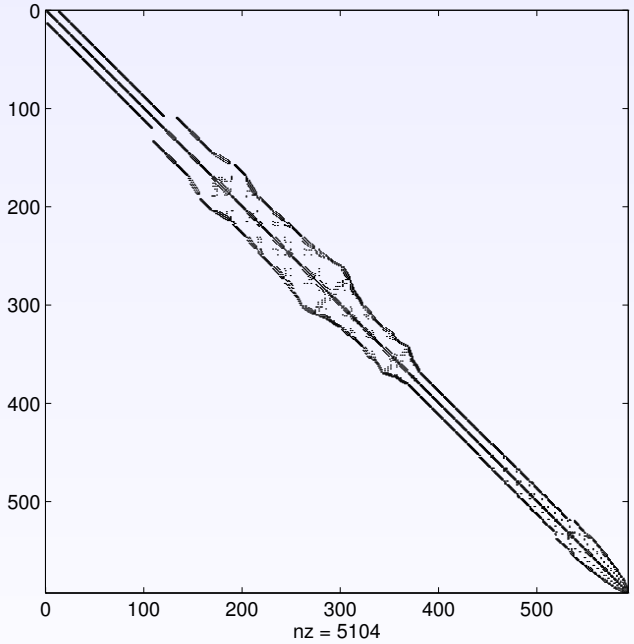


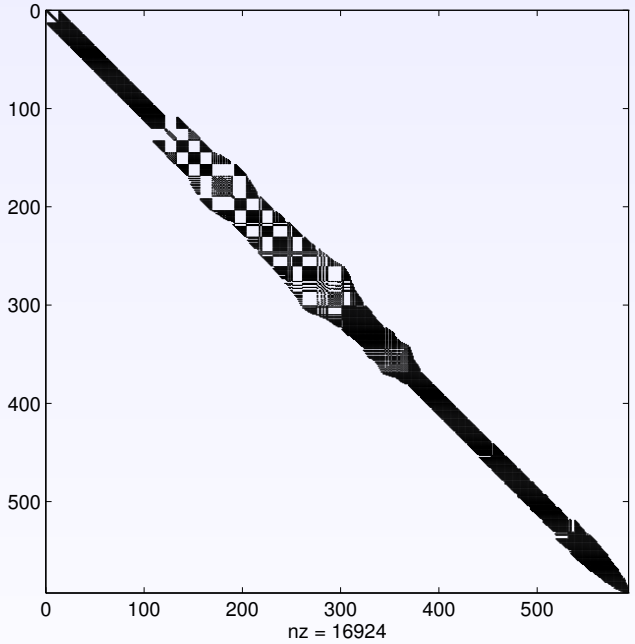
Illustration : Reverse Cuthill-McKee on matrix dwt_592.rua

NZ(LU factors)=16924

Permuted matrix (RCM)



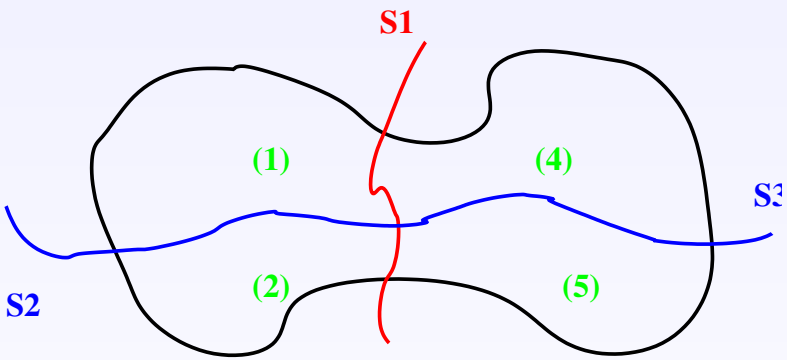
Factorized permuted matrix



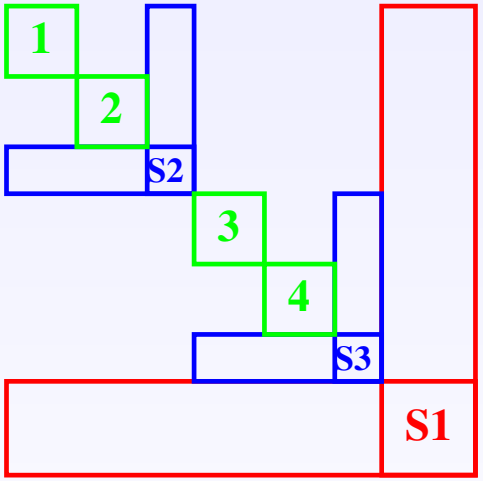
Nested Dissection

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,

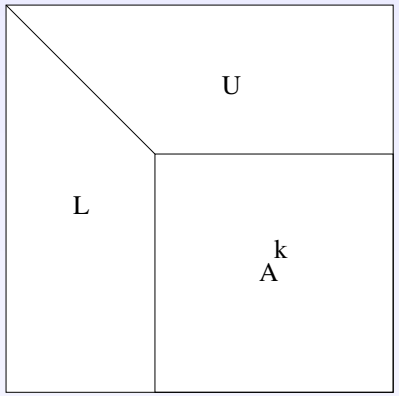
Step2 : update elimination graph,

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.

Reordering unsymmetric matrices : Markowitz criterion

- At step k of Gaussian elimination :



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

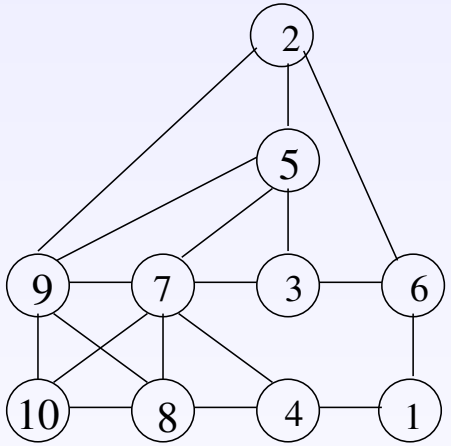
- $\boxed{\text{Minimum degree}}$: Markowitz criterion for symmetric diagonally dominant matrices

Minimum degree algorithm

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

$$\begin{bmatrix}
1 & & & \times & & \times & & & & \\
 & 2 & & & \times & \times & & & \times & \\
 & & 3 & & \times & \times & \times & & & \\
\times & & & 4 & & & \times & \times & & \\
 & \times & \times & & 5 & & \times & & \times & \\
\times & \times & \times & & & 6 & & & & \\
 & & \times & \times & \times & & 7 & \times & \times & \times \\
 & & & \times & & & \times & 8 & \times & \times \\
 & \times & & & \times & & \times & \times & 9 & \times \\
 & & & & & & \times & \times & \times & 10
\end{bmatrix}$$

(a) Sparse symmetric matrix



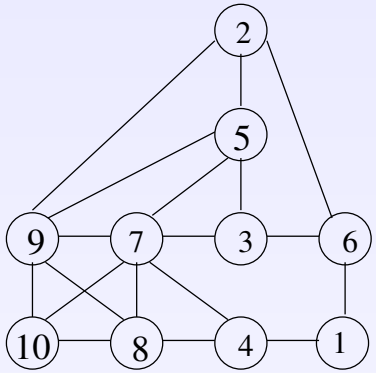
(b) Elimination graph

The node/variable selected is 1 of degree 2.

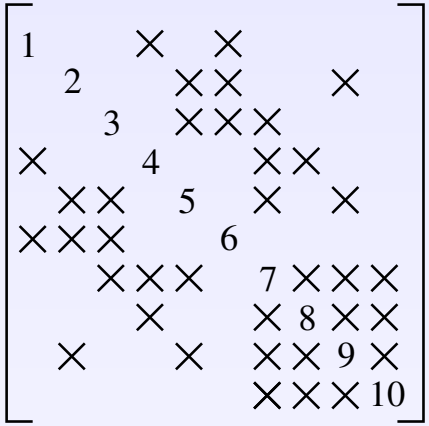
- **Notation for the elimination graph**
  - Let $G^k = (V^k, E^k)$, the graph built at step k.
  - G^k describes the structure of $\mathbf{A}_k$ after elimination of k pivots.
  - G^k is non-oriented ($\mathbf{A}_k$ is symmetric)
  - Fill-in in $\mathbf{A}_k \equiv$ adding edges in the graph.

Illustration

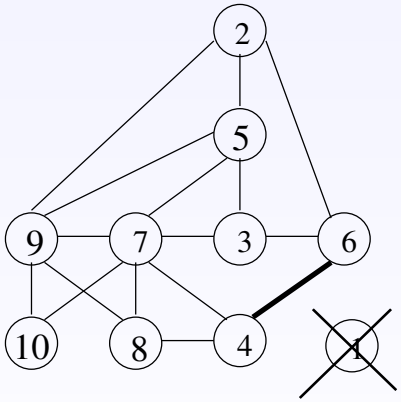
Step 1 : elimination of pivot 1

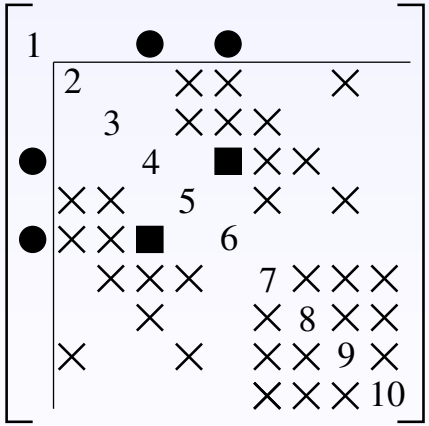


(a) Elimination graph



(b) Factors and active submatrix





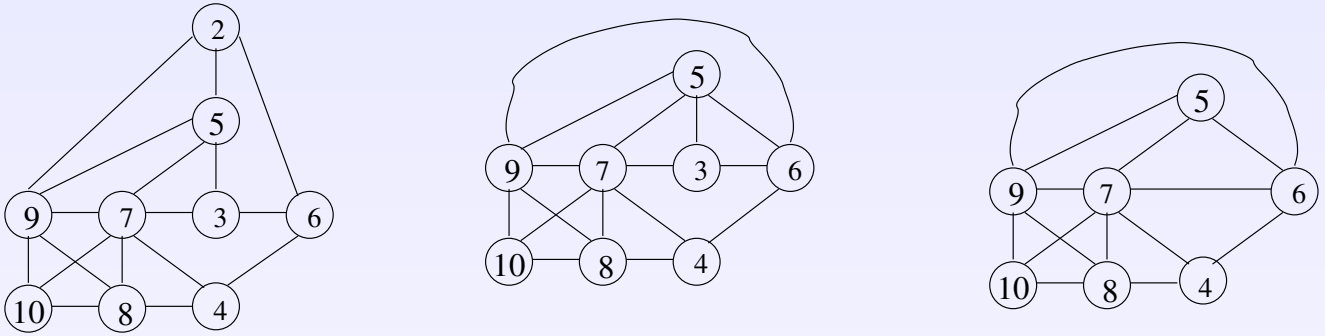
× **Initial nonzeros** ● **Nonzeros in factors** ■ **Fill–in**

Minimum degree algorithm applied to the graph :

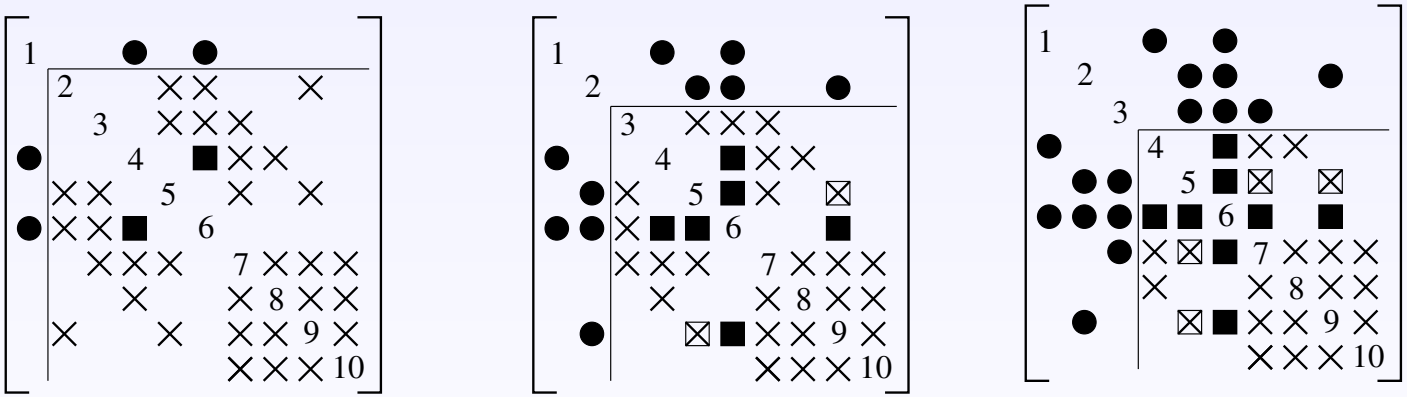
- Step k : Select the node with the smallest number of neighbors
- G^k is built from G^{k-1} by *suppressing the pivot* and *adding edges* corresponding to fill-in.

Illustration (cont'd)

Graphs G_1, G_2, G_3 and corresponding reduced matrices.



(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 ! ! !

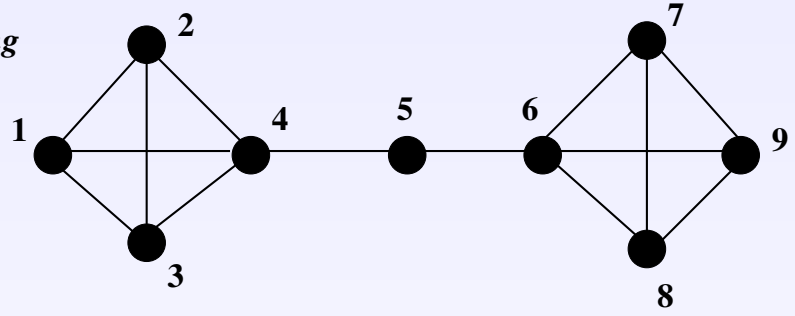
Consider the following matrix

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

Remark: Using initial ordering

$\Downarrow$

No fill–in

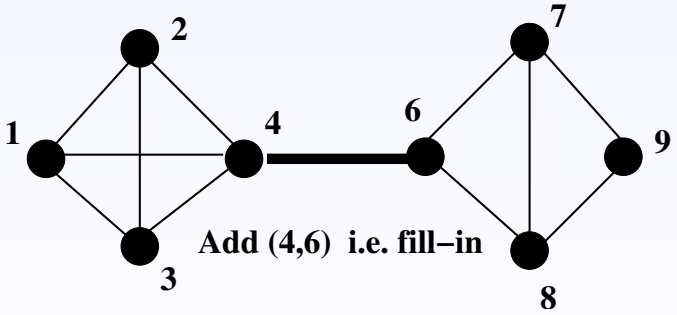


Corresponding elimination graph

Step 1 of Minimum Degree:

Select pivot 5 (minimum degree = 2)

Updated graph



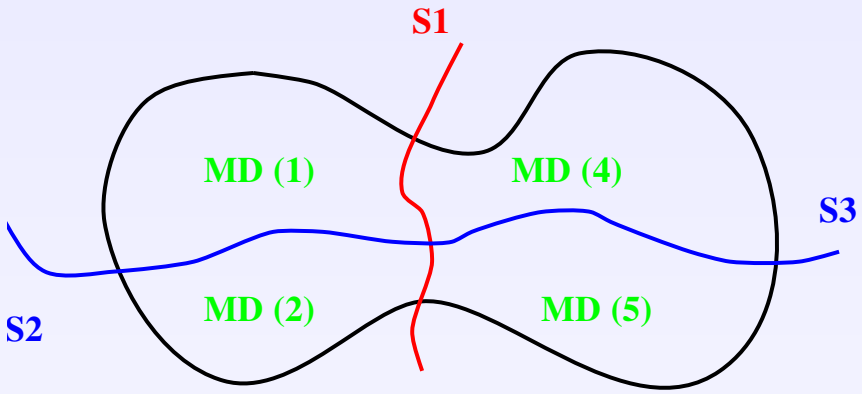
Combining reordering techniques

Example (1) of hybrid approach

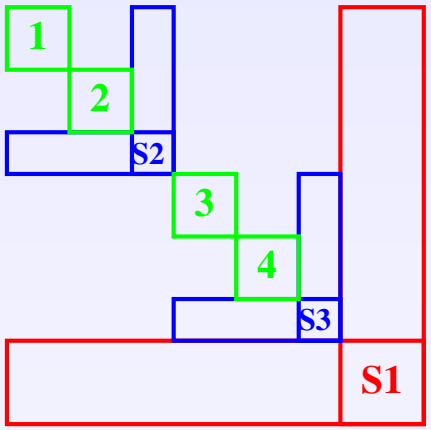
- Top-down followed by bottom-up processing of the graph :
  Top-down : Apply nested dissection (ND) on complete graph
  Bottom-up : Local heuristic on each subgraph
- Generally better for large-scale irregular problems than
  - pure nested dissection
  - local heuristics

(1 cont) hybrid approach

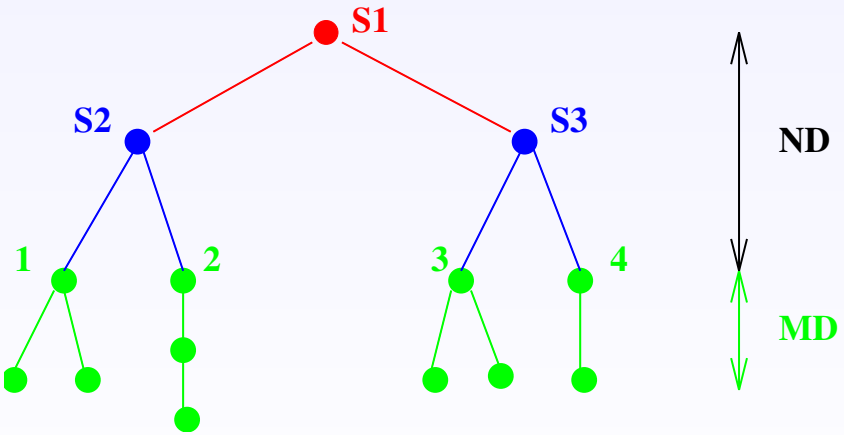
Graph partitioning



Permuted matrix

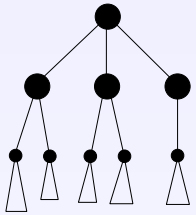
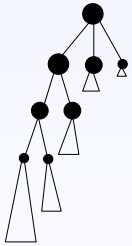


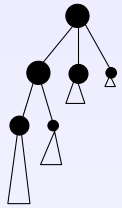
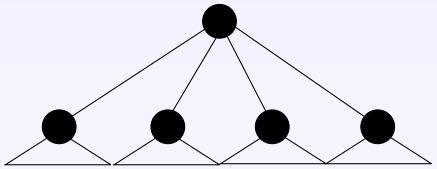
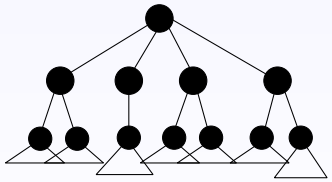
Elimination graph



Plan de l'exposé

Impact of fill reduction on the shape of the tree

Reordering technique	Shape of the tree	observations
AMD		• Deep well-balanced • Large frontal matrices on top
AMF		• Very deep unbalanced • Small frontal matrices

Reordering technique	Shape of the tree	observations
PORD		• deep unbalanced • Small frontal matrices
SCOTCH		• Very wide well-balanced • Large frontal matrices
METIS		• Wide well-balanced • Smaller frontal matrices (than SCOTCH)

Importance of the shape of the tree

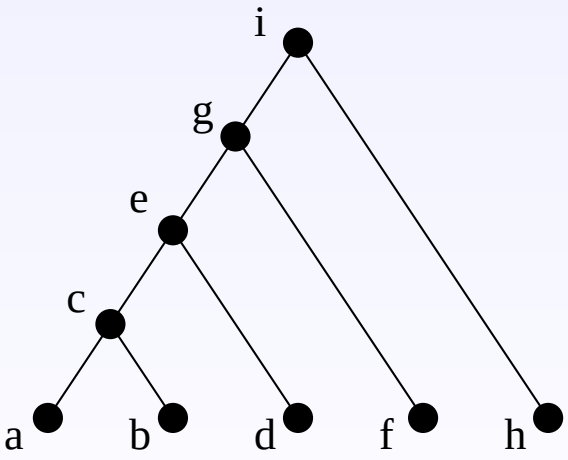
Suppose that each node in the tree corresponds to a task that :
- consumes temporary data from the children,
- produces temporary data, that is passed to the parent node.

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

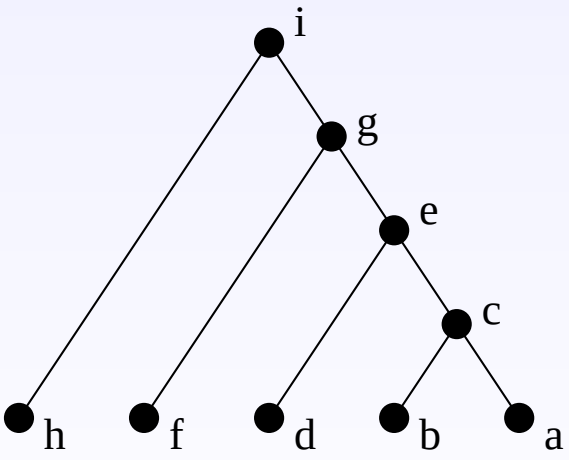
Scheduling tasks in the tree (tree traversal)

- Assumption : parents are processed as soon as all children have completed (postorder of the tree)
- Given a tree, memory usage depends on tree traversal.

Best (abcdefghi)

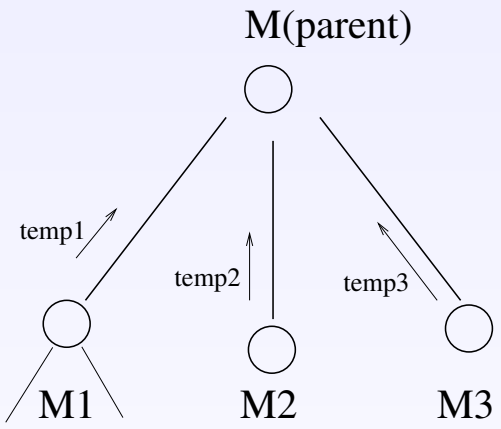


Worst (hfdbacegi)



Memory-minimizing schedules

- M_i : memory peak for complete subtree rooted at i,
- $temp_i$: temporary memory produced by node i,
- m : memory for storing the parent.



$$M_{parent} = \max(\max_{j=1}^{nbchildren}(M_i + \sum_{k=1}^{j-1} temp_k), m + \sum_{j=1}^{nbchildren} temp_j)$$

Memory-minimizing schedules

Theorem

The minimum of $\max_j(x_j + \sum_{i=1}^{j-1} y_j)$ *is obtained when the sequence* (x_i, y_i) *is sorted in decreasing order of* $x_i - y_i$*,*

Corollary

An optimal child sequence is obtained by rearranging the children nodes in decreasing order of $M_i - temp_i$*.*

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

$\Rightarrow$ Apply on complete tree starting from the leaves.

Plan de l'exposé

Projects

- GRAAL/LIP (ENS Lyon / INRIA / UCBL / CNRS)
  - Scheduling for parallel sparse direct solvers
- Strong collaboration with ENSEEIHT-IRIT (Toulouse)
- Software is vital to validate/experiment our research

Projects

- GRAAL/LIP (ENS Lyon / INRIA / UCBL / CNRS)
  - Scheduling for parallel sparse direct solvers
- Strong collaboration with ENSEEIHT-IRIT (Toulouse)
- Software is vital to validate/experiment our research

Software

MUMPS (MUltifrontal Massively Parallel Solver)

MUMPS solves large systems of linear equations of the form Ax=b by factorizing A into A=LU or LDLT

- Symmetric or unsymmetric matrices (partial pivoting)
- Parallel factorization and solution phases (uniprocessor version also available)
- Iterative refinement and backward error analysis
- Various matrix input formats
  - assembled format
  - distributed assembled format
  - sum of elemental matrices
- Partial factorization and Schur complement matrix
- Version for complex arithmetic
- Several orderings interfaced : AMD, AMF, PORD, METIS

Software (2)

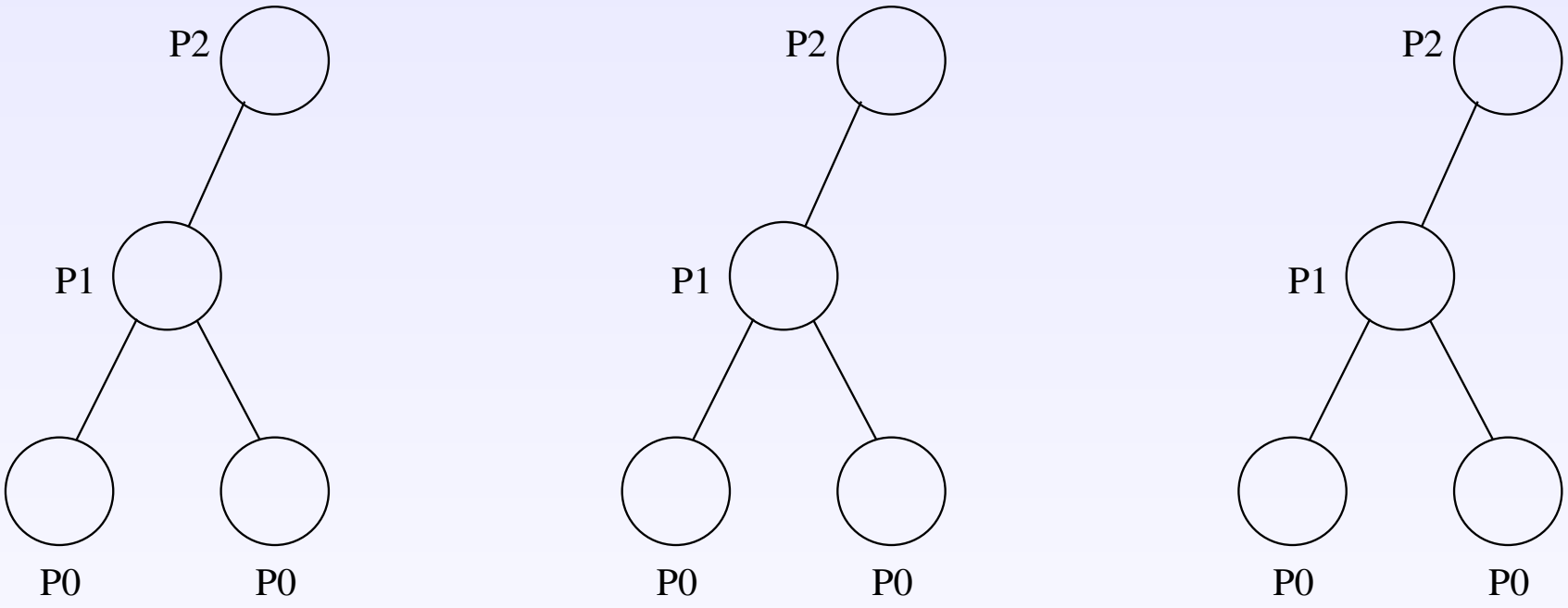
MUMPS (MUltifrontal Massively Parallel Solver)

- Main contributors : P. Amestoy, I. Duff, A. Guermouche, J.Koster, J.-Y. L'Excellent, S. Pralet
- Recent features :
  - sparse, multiple right-hand sides,
  - hybrid scheduling,
  - improved numerical features for symmetric matrices,
  - distributed (2D cyclic) Schur complement,
  - preprocessing duplicate entries,
  - scilab and matlab interfaces, ...
- $\approx$ 800 users (academic + industrial users, eg : Boeing, BRGM, EADS, CEA, Dassault, EADS, EDF, MIT, NASA, SAMTECH, ...),
- Freely available
- More info : http ://graal.ens-lyon.fr/MUMPS or http ://www.enseeiht.fr/apo/MUMPS

MUMPS (Multifrontal sparse solver)

1. **Analysis and Preprocessing**
   - Preprocessing (max. transversal, scaling)
   - Fill-in reduction on $\mathbf{A} + \mathbf{A}^T$
   - Partial static mapping (elimination tree)
2. **Factorization**
   - Multifrontal (elimination tree of $\mathbf{A} + \mathbf{A}^T$)
     $$Struct(\mathbf{L}) = Struct(\mathbf{U})$$
   - Partial threshold pivoting
   - Node parallelism
     - Partitioning (1D Front - 2D Root)
     - Dynamic distributed scheduling
3. **Solution step and iterative refinement**

Multifrontal variant



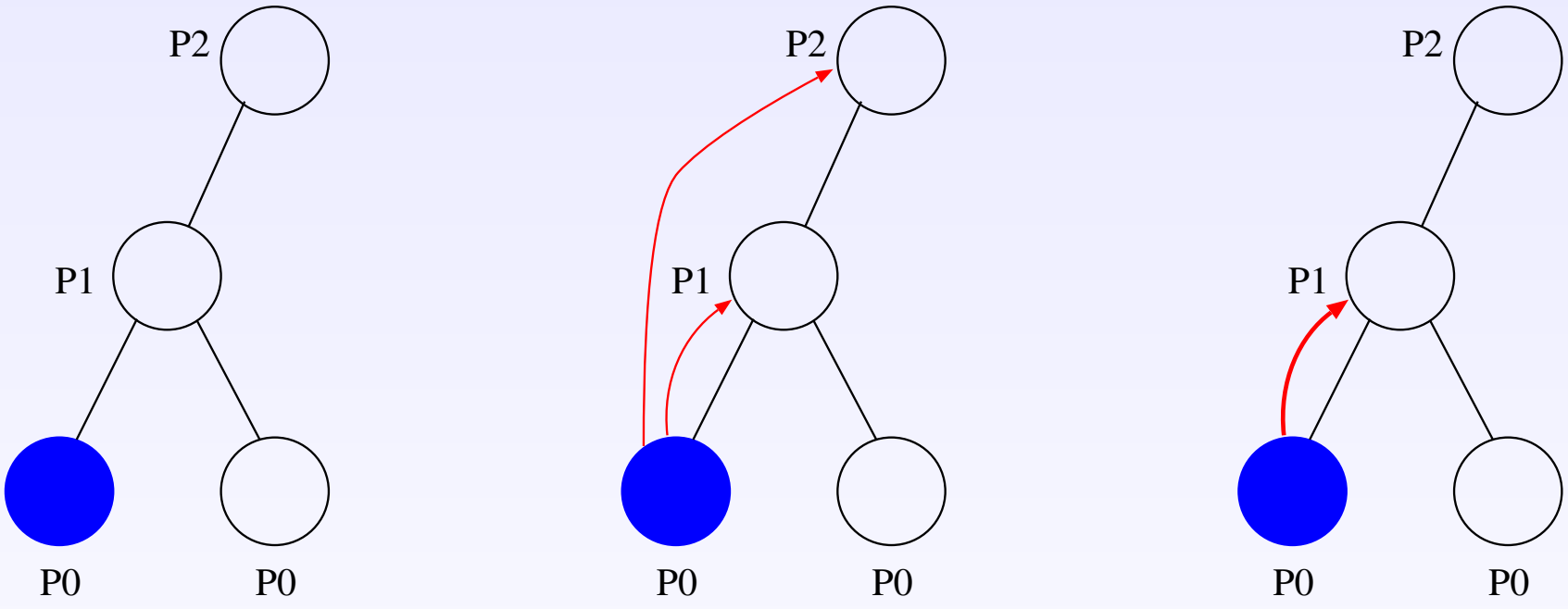
(a) Fan-in.

(b) Fan-out.

(c) Multi-frontal.

FIG.: Communication schemes for three approaches.

Multifrontal variant



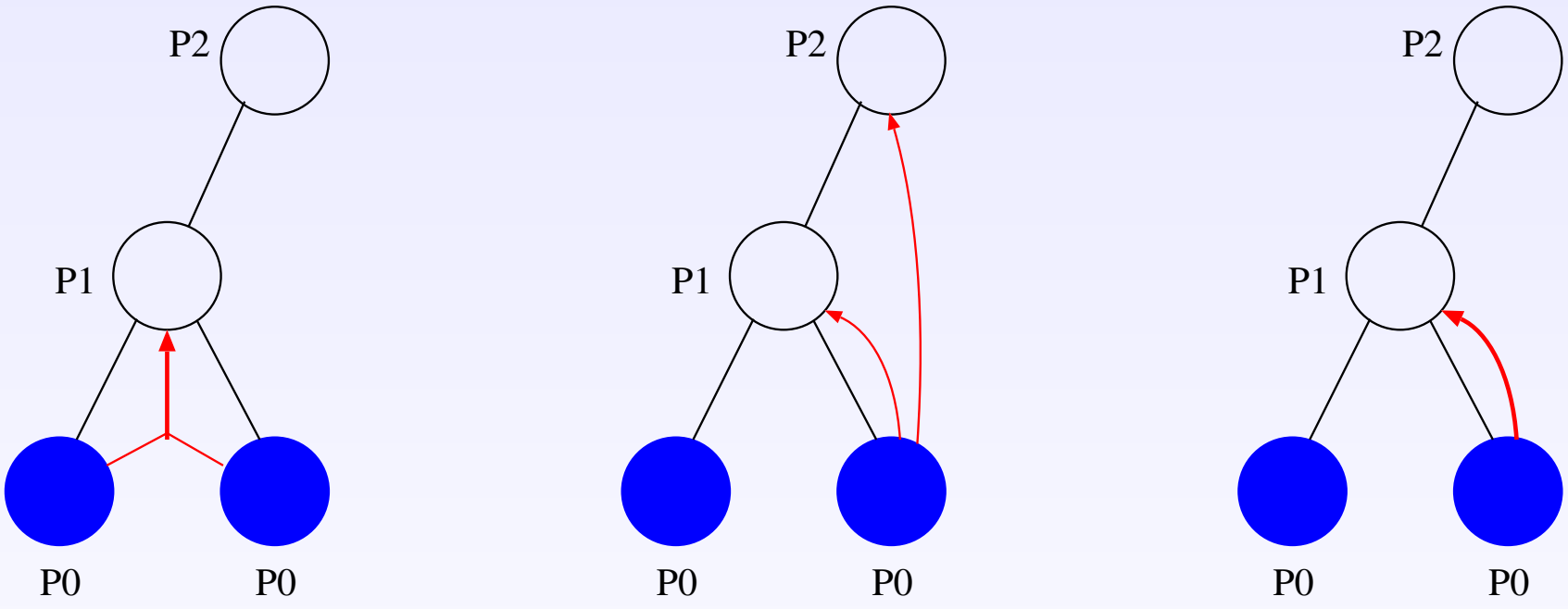
(a) Fan-in.

(b) Fan-out.

(c) Multi-frontal.

FIG.: Communication schemes for three approaches.

Multifrontal variant



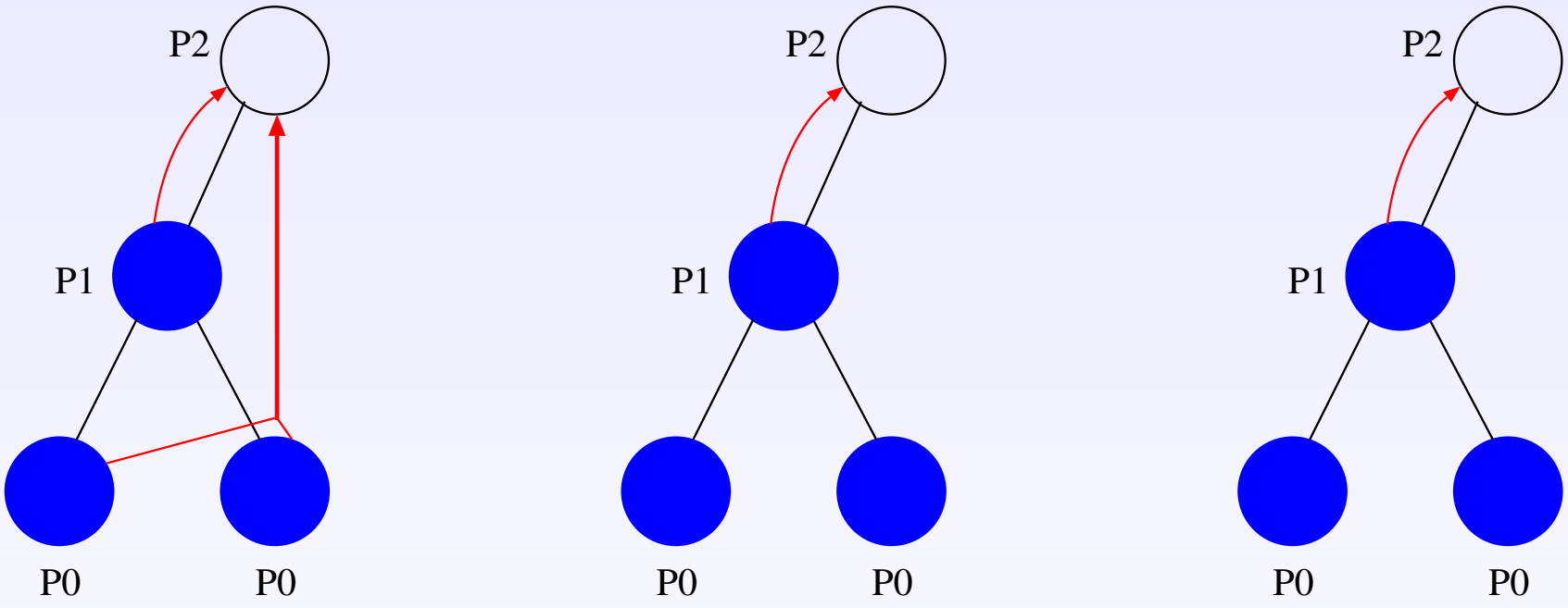
(a) Fan-in.

(b) Fan-out.

(c) Multi-frontal.

FIG.: Communication schemes for three approaches.

Multifrontal variant



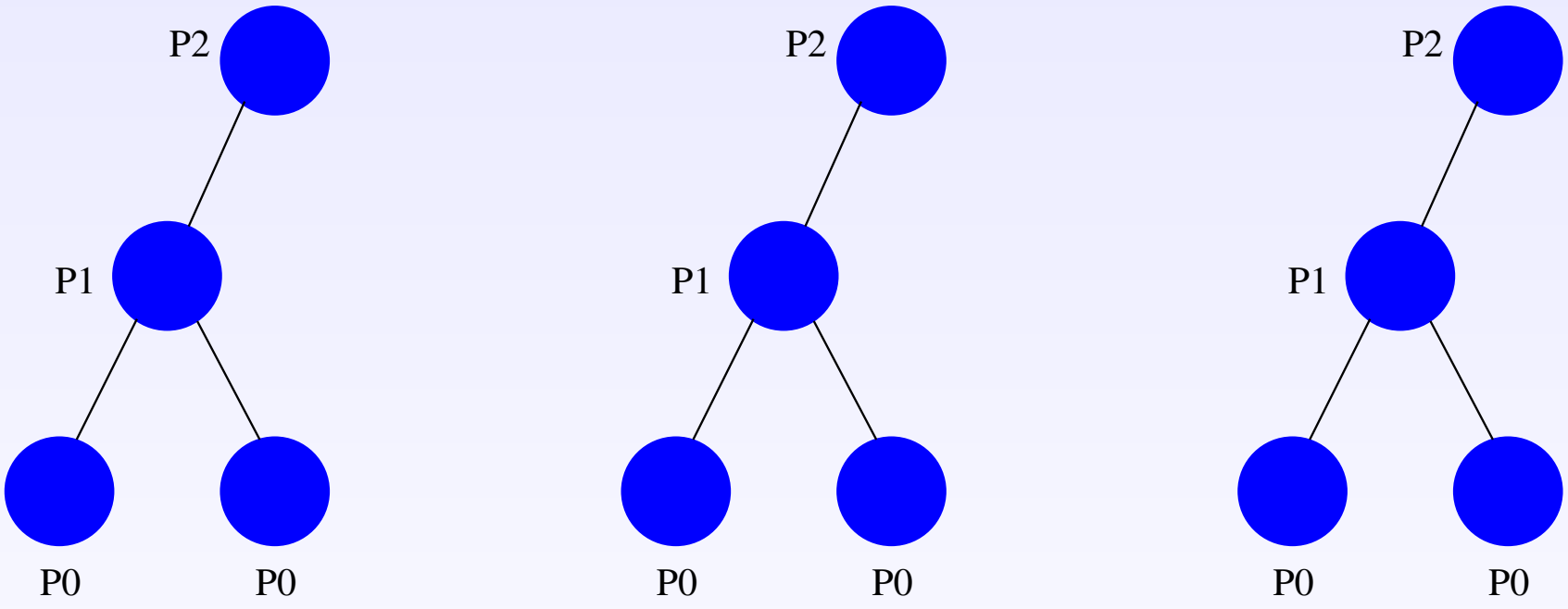
(a) Fan-in.

(b) Fan-out.

(c) Multi-frontal.

FIG.: Communication schemes for three approaches.

Multifrontal variant



(a) Fan-in.

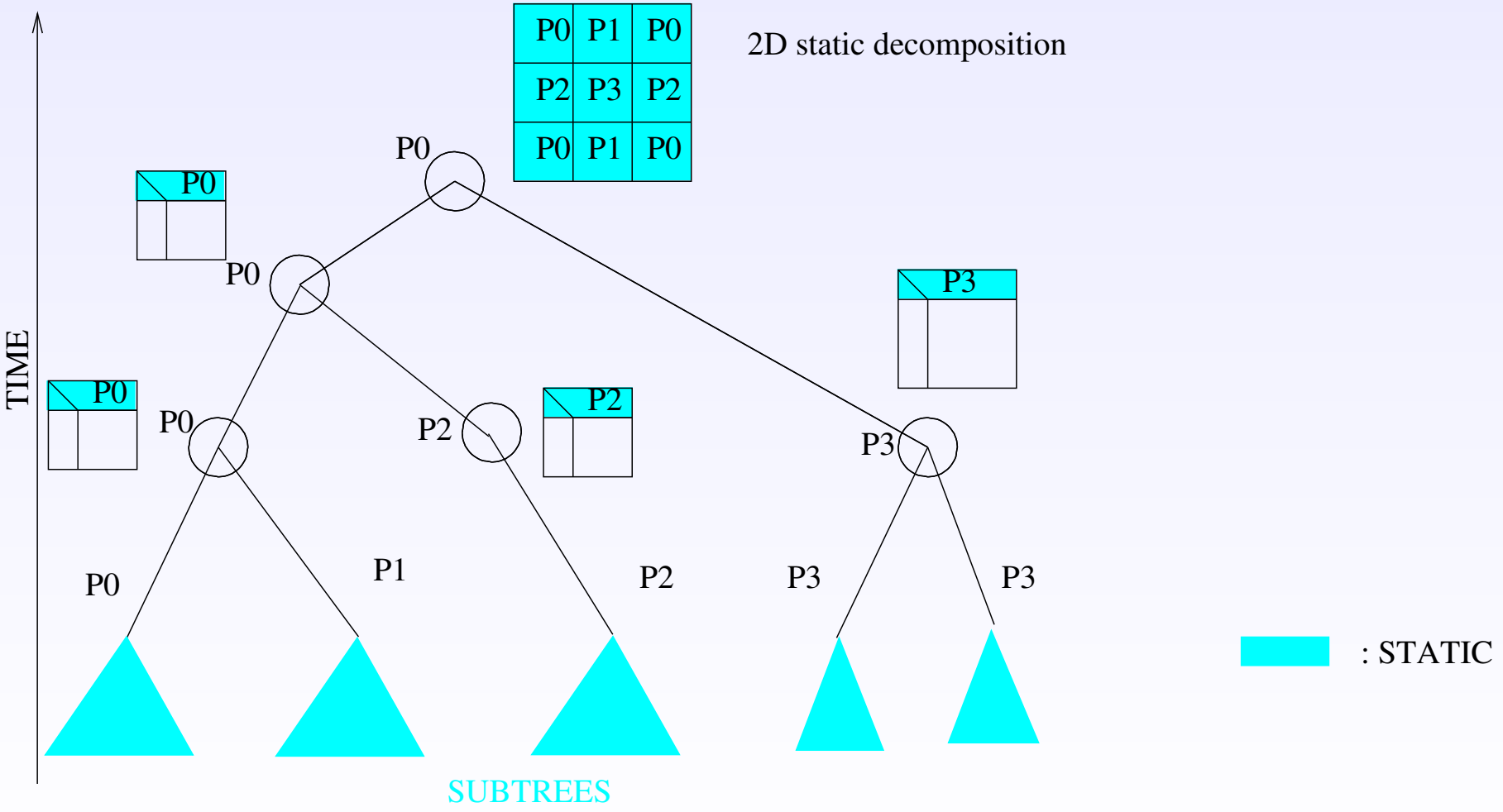
(b) Fan-out.

(c) Multi-frontal.

FIG.: Communication schemes for three approaches.

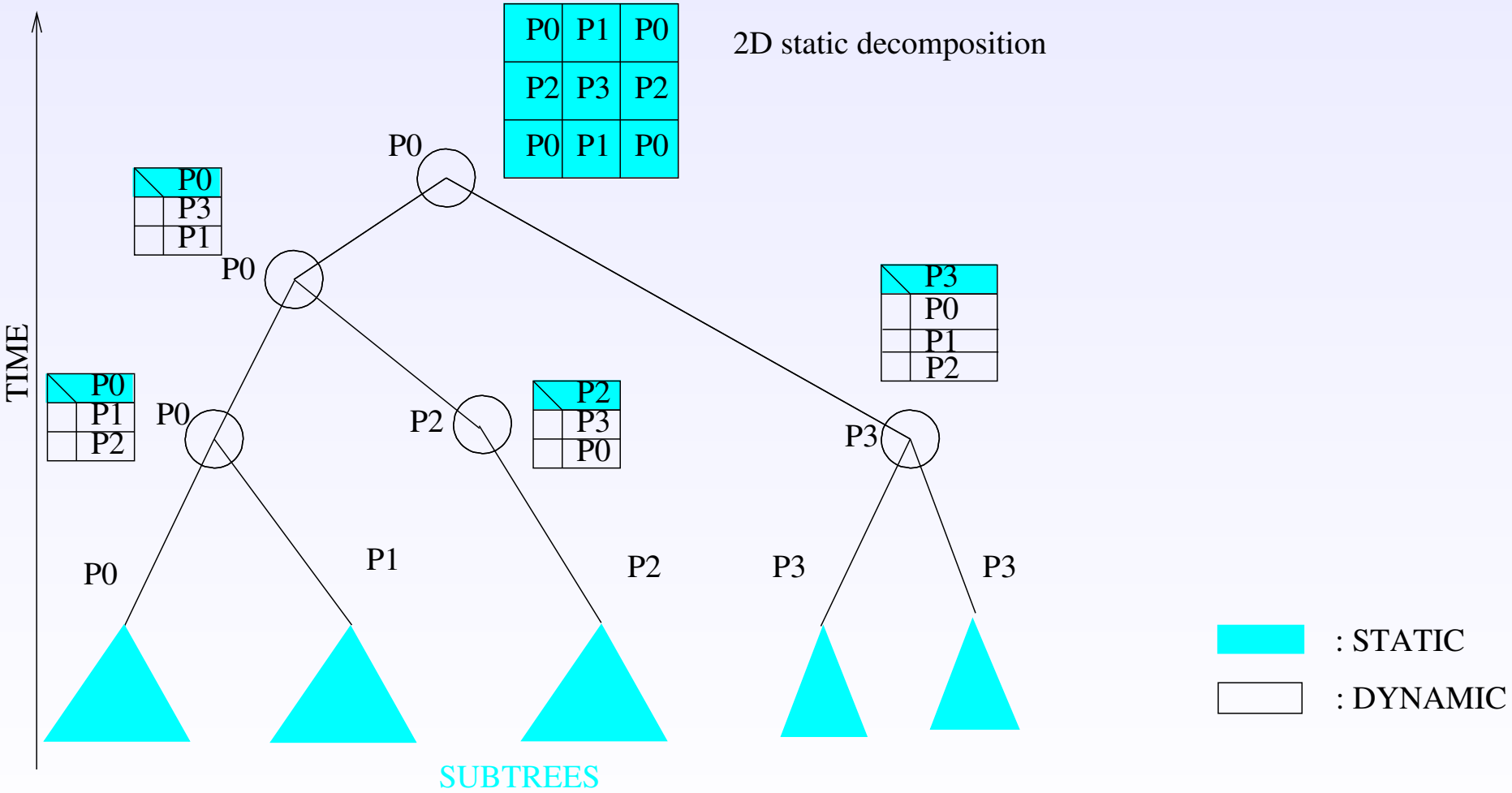
MUMPS : dynamic scheduling

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



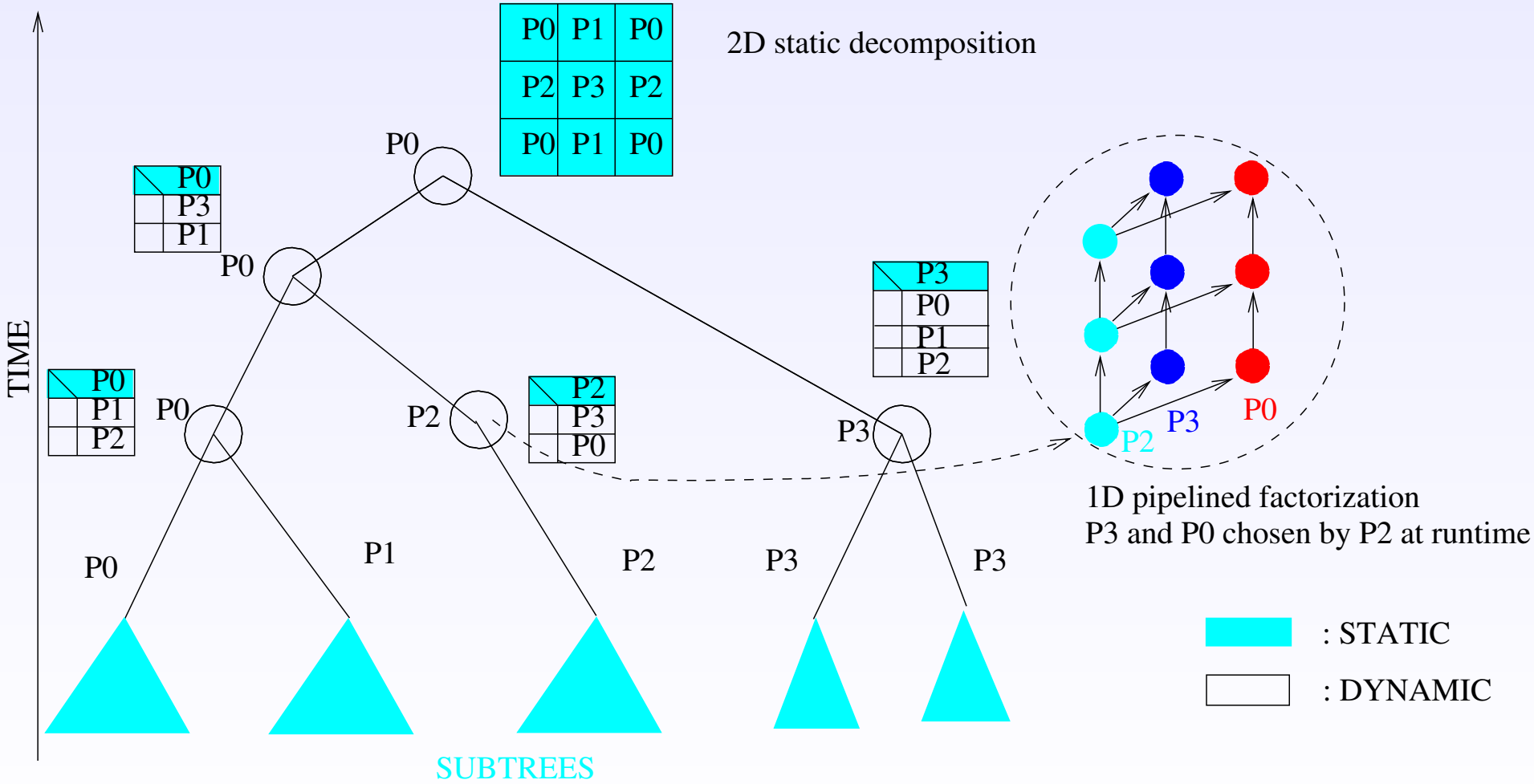
MUMPS : dynamic scheduling

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

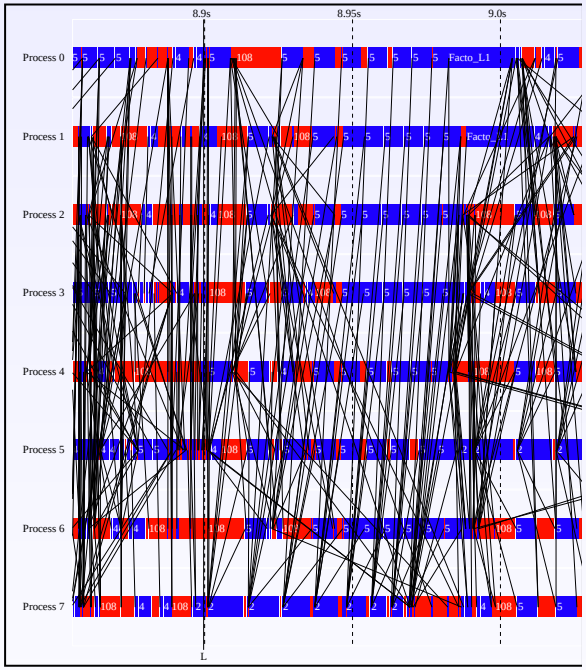


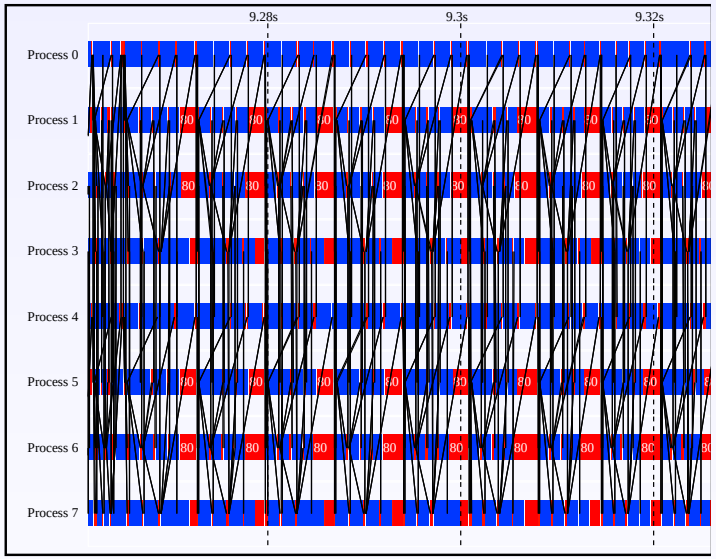
MUMPS : dynamic scheduling

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



Trace of execution(BBMAT, 8 proc. CRAY T3E)





Current/related work

- Parallel Out-of-core Solvers
  - strong demand from users
  - PhD Emmanuel Agullo, ENS Lyon
- Improve performance of solution phase
  - successive solution steps with same matrix
  - out-of-core context
  - PhD Mila Slavova, CERFACS
- Provide functionalities for external solvers
  - Schur complement
  - hybrid direct-iterative solvers : PhD Azzam Haidar, CERFACS
- Software/engineer work
  - Aurélia Fèvre : engineer, employed by INRIA
- Grid TLSE project :
  - a web expertise site for sparse linear algebra
  - project coordinator : ENSEEIHT-IRIT

Current/related work

- Parallel Out-of-core Solvers
  - strong demand from users
  - PhD Emmanuel Agullo, ENS Lyon
- Improve performance of solution phase
  - successive solution steps with same matrix
  - out-of-core context
  - PhD Mila Slavova, CERFACS
- Provide functionalities for external solvers
  - Schur complement
  - hybrid direct-iterative solvers : PhD Azzam Haidar, CERFACS
- Software/engineer work
  - Aurélia Fèvre : engineer, employed by INRIA
- Grid TLSE project :
  - a web expertise site for sparse linear algebra
  - project coordinator : ENSEEIHT-IRIT

Plan de l'exposé

L'out-of-core

Avion Rafale

$Ax = b$: 1 M variables

$\Rightarrow A = LU$ (Méthodes directes)

Limites actuelles

Matrice BRGM :

- 5 M variables
- 140 M non zéros

Contrainte physique

... et vite !

Minimiser le temps écoulé

Tout de suite.

L'out-of-core

Avion Rafale

$Ax = b$: 1 M variables

$\Rightarrow A = LU$ (Méthodes directes)

Limites actuelles

Matrice BRGM :

- 5 M variables
- 140 M non zéros

Contrainte physique

... et vite !

Minimiser le temps écoulé

Tout de suite.

L'out-of-core

Avion Rafale

$Ax = b$: 1 M variables

$\Rightarrow A = LU$ (Méthodes directes)

Limites actuelles

Matrice BRGM :

- 5 M variables
- 140 M non zéros

Contrainte physique

Mémoire physique

Mémoire nécessaire

Crash mémoire

... et vite !

Minimiser le temps écoulé

Tout de suite.

L'out-of-core

Avion Rafale

$Ax = b$: 1 M variables

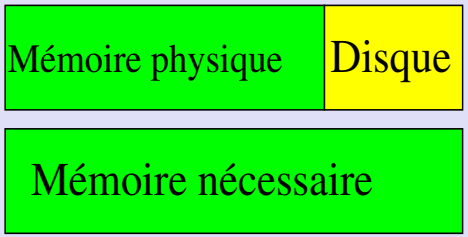
$\Rightarrow A = LU$ (Méthodes directes)

Limites actuelles

Matrice BRGM :

- 5 M variables
- 140 M non zéros

Out-of-core, svp



Recours aux disques

... et vite !

Minimiser le temps écoulé

Tout de suite.

L'out-of-core

Avion Rafale

$Ax = b$: 1 M variables

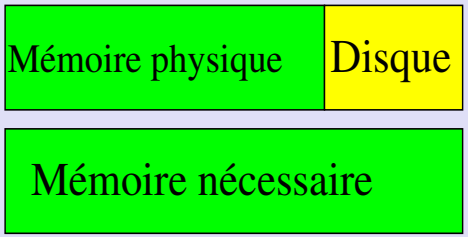
$\Rightarrow A = LU$ (Méthodes directes)

Limites actuelles

Matrice BRGM :

- 5 M variables
- 140 M non zéros

Out-of-core, svp



Recours aux disques

... et vite !

Minimiser le temps écoulé

Tout de suite.

L'out-of-core

Avion Rafale

$Ax = b$: 1 M variables

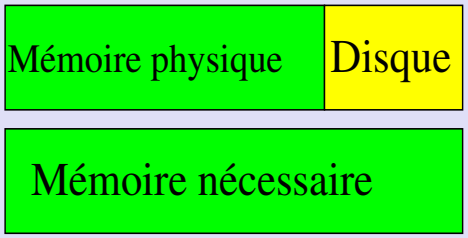
$\Rightarrow A = LU$ (Méthodes directes)

Limites actuelles

Matrice BRGM :

- 5 M variables
- 140 M non zéros

Out-of-core, svp



Recours aux disques

Objectifs

1. Maximiser recouvrement E/S - calculs
2. Minimiser volume d'E/S

Tout de suite.

L'out-of-core

Avion Rafale

$Ax = b$: 1 M variables

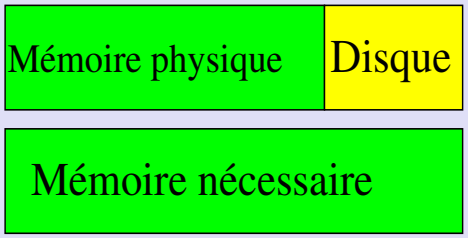
$\Rightarrow A = LU$ (Méthodes directes)

Limites actuelles

Matrice BRGM :

- 5 M variables
- 140 M non zéros

Out-of-core, svp



Recours aux disques

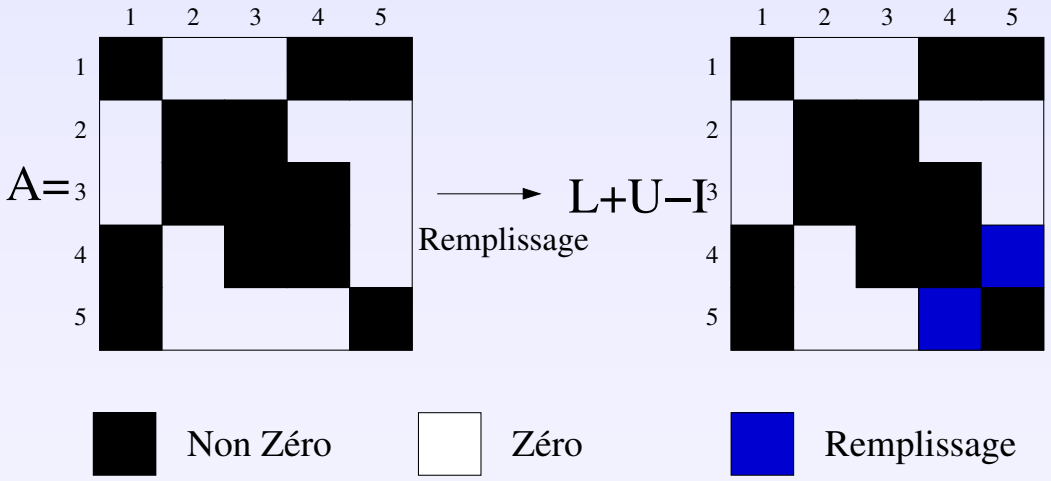
Objectifs

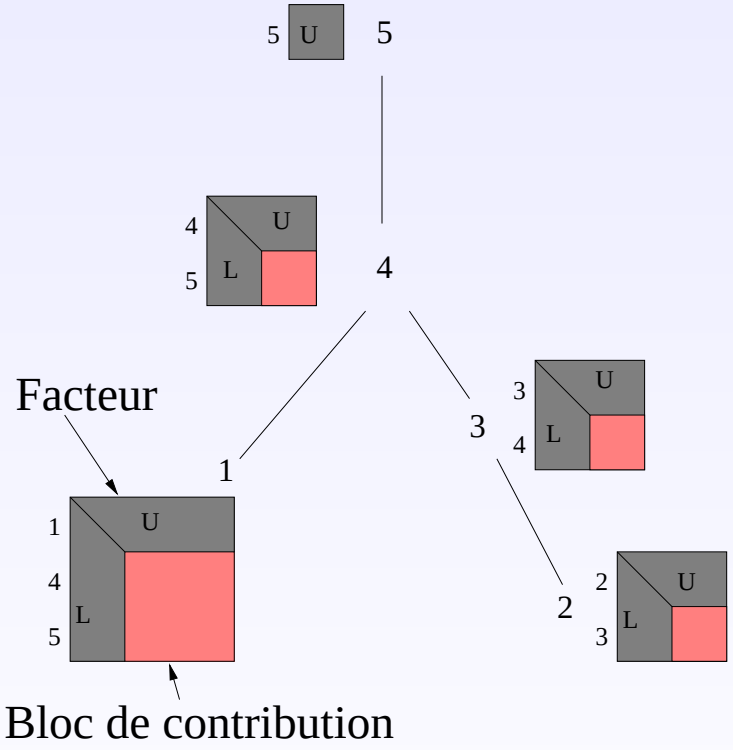
1. Maximiser recouvrement *E/S* - calculs
2. Minimiser volume d'*E/S*

Libertés :

- *Quoi* écrire
- *Quand*
- *Comment*
- ...

Méthode multifrontale (Duff, Reid'83)

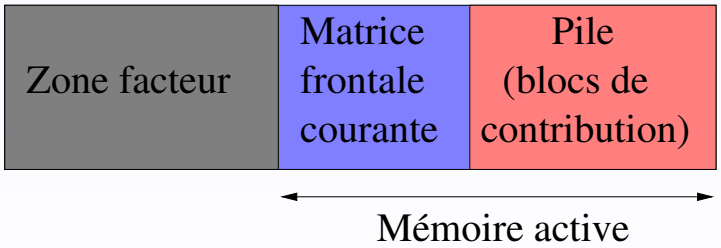




Arbre d'assemblage

Mémoire divisée en deux parties :

- mémoire active ;
- facteurs.



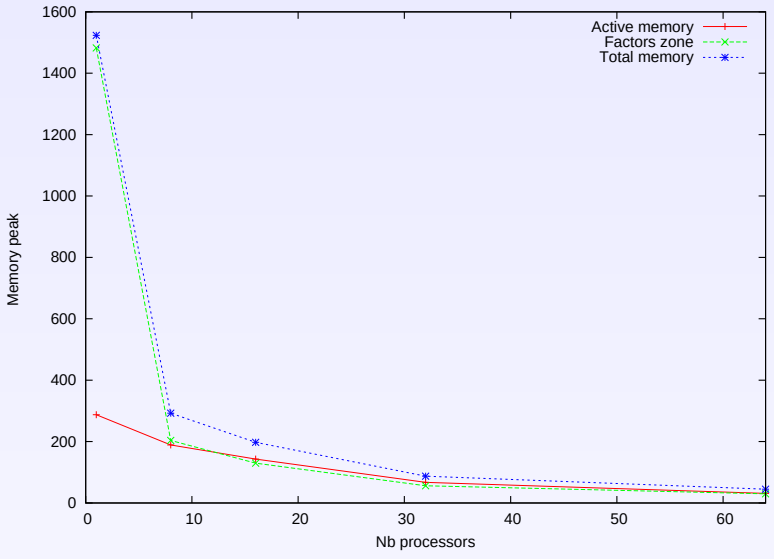
Plan de l'exposé

Preliminary Study

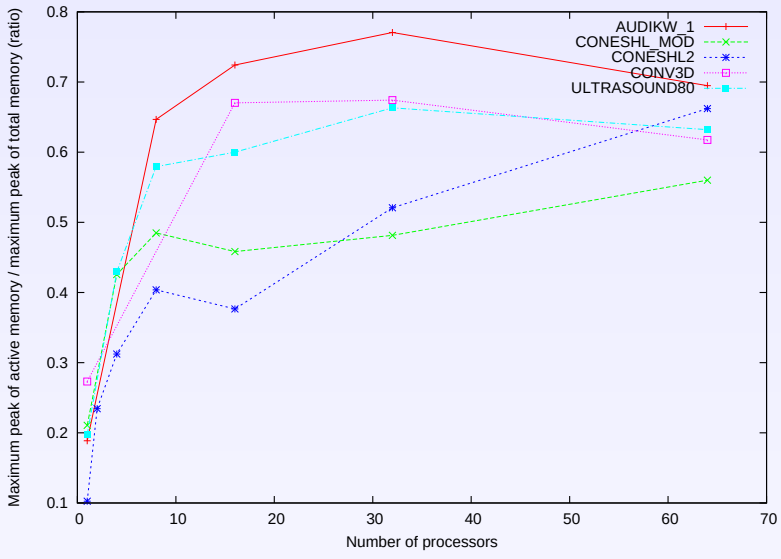
- MUMPS : Multifrontal Parallel Solver for both LU and LDL^T.
- Simulation of an out-of-core behaviour :
  - Free factors as soon as they are computed
  - Only factorization step is possible (factors are lost)
- Selected values : the bigger over all processors for :
  - the size of factors
  - the peak of active memory
  - the peak of total memory

Typical Memory Behaviour

- Typical memory behaviour (`AUDIKW_1` matrix, METIS)



(a) Memory



(b) Active memory / total memory

Consequence

- First step : store factors on disk (well adapted for few processors)
- Second step : stack should also be out-of-core (larger problems or many processors)

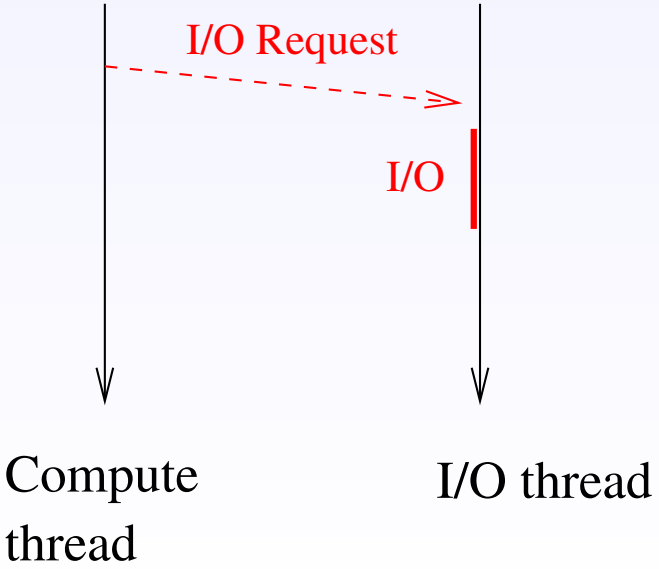
Plan de l'exposé

Out-of-core Storage of the Factors

- Synchronous Version :
  - Use standard write operations
  - Factors are written to disk (possibly with low-level system buffering) as soon as they are computed
  - Solution step :
    1. Read a factor block
    2. Work with the factor

    ⇒ Factors may be read twice (forward elimination and backward substitution)

- Asynchronous Version :
  - Threaded version with buffers
  - Solution step : still synchronous



Plan de l'exposé

Experimental Environment

- MUMPS : Multifrontal Parallel Solver for both LU and LDL^T.
- Test platform : *IBM* machine at *IDRIS* (Orsay, France) composed of 4-way and 32-way Power4+ processors.
  Memory limits per processor :

Number of procs	1	2-16	17-64	65-
Max memory	16 GB	4GB	3.5GB	1.3GB

- Test problems : large matrices (from PARASOL, SAMTECH, CEA/CESTA, M. Sosonkina).

	Order	nnz	$nnz(L\|U) \times 10^6$	Ops$\times 10^9$
Symmetric matrices				
AUDIKW_1	943695	39297771	1368.6	5682
CONESHL_MOD	1262212	43007782	790.8	1640
Unsymmetric matrices				
CONV3D64	836550	12548250	2693.9	23880
ULTRASOUND80	531441	33076161	981.4	3915

(Statistics with METIS)

Results : we can solve

- bigger problems
- same problems with less memory (cf preliminary study)

  example : ULTRASOUND80

	total mem per proc	active mem per proc
1 proc (16GB)	1101 million reals	218 million reals
4 procs	360 million reals	154 million reals

- same problems with less processors

Matrix	Strategy	min procs
ULTRASOUND80	in-core	8
	out-of-core	2
CONV3D64	in-core	32
	out-of-core	16

- CONV3D64 on 1 proc with 16 GB memory :

  OOC version ok, IC version runs out of memory

Plan de l'exposé

Preliminary Performance Analysis

- Compare performance of IC and OOC strategies (when enough memory for both)
  - synchronous I/O
  - asynchronous I/O with a buffer
  - in-core
- Time for factorization :

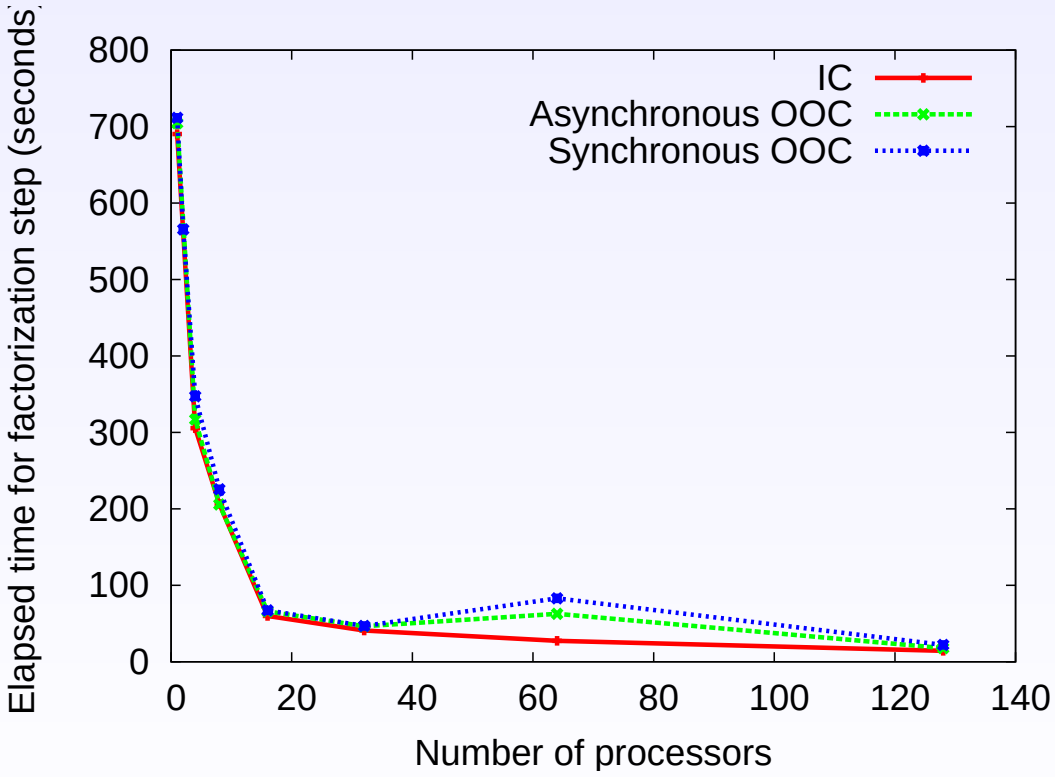
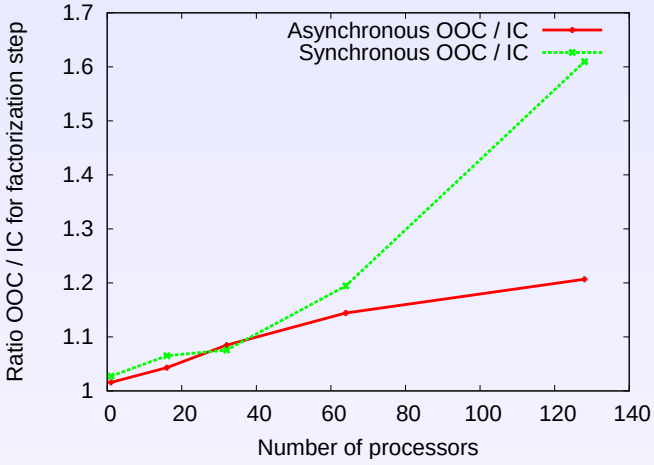
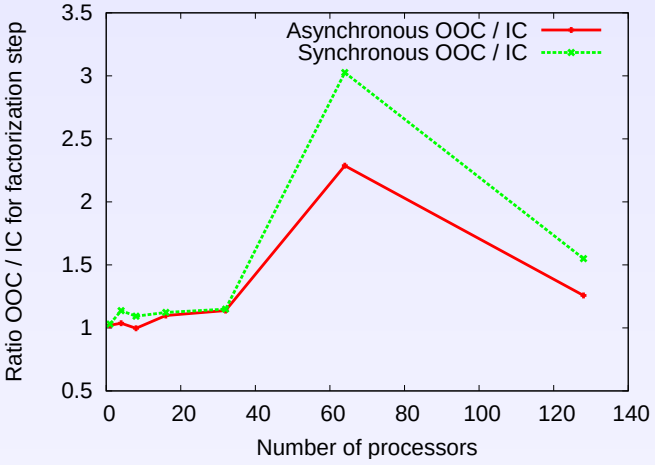


FIG.: Factorization time (matrix CONESHL_MOD)

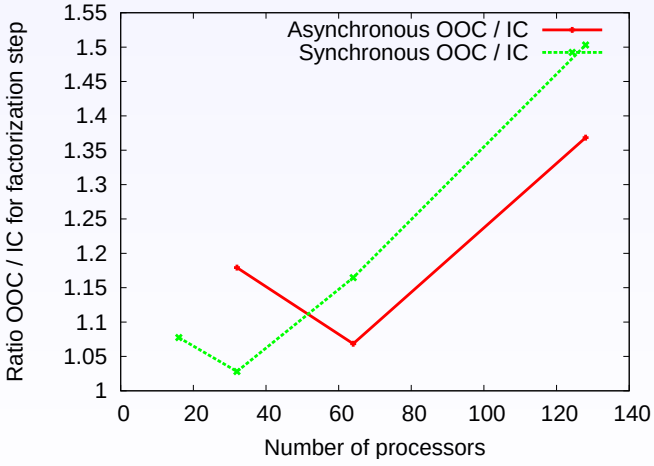
RED : $\frac{\textit{time synchronous version}}{\textit{time in-core}}$ GREEN : $\frac{\textit{time asynchronous version}}{\textit{time in-core}}$



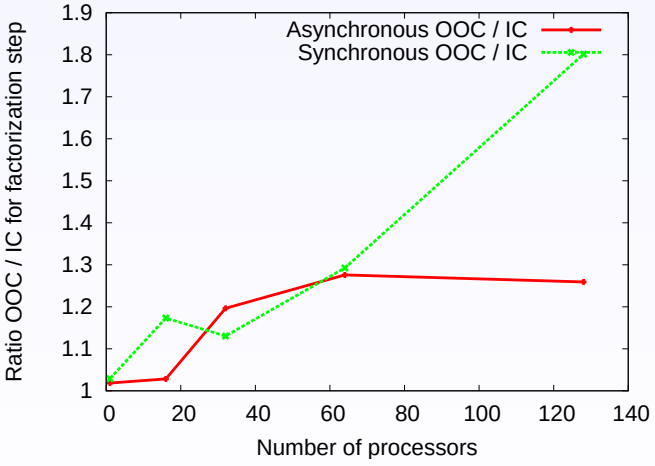
(a) AUDIKW_1



(b) CONESHL_MOD



(c) CONV3D64



(d) ULTRASOUND80

Remarks

Impact of locality

- In several cases, out-of-core version as good as in-core version !
- Explanation : better memory locality (frontal matrix always in the same area of memory)

Impact of platform

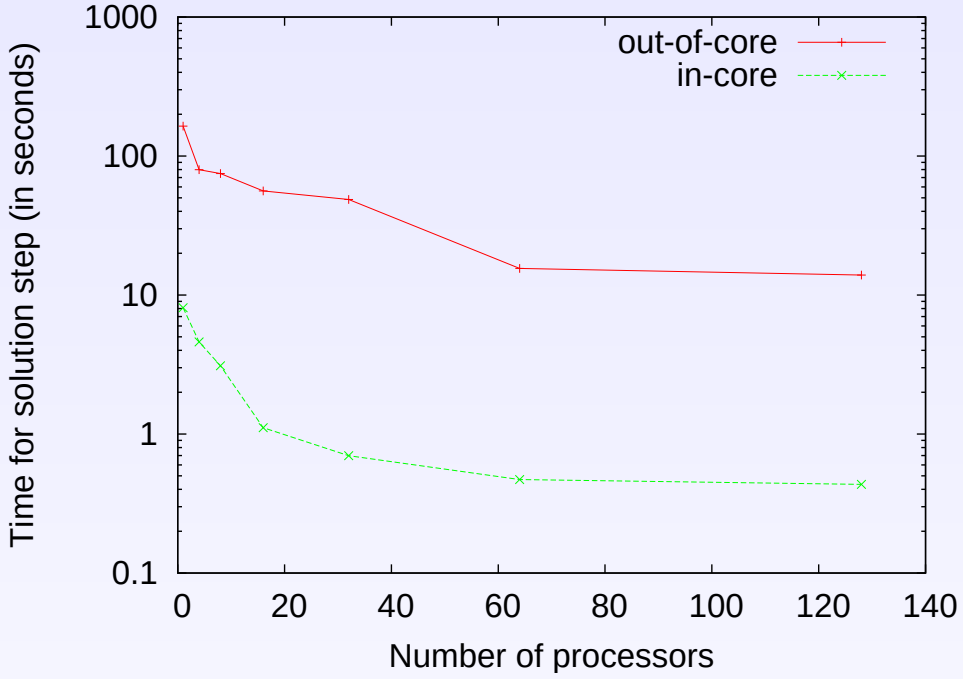
- (GPFS) no guarantee that each processor accesses its own disk...

$\Rightarrow$ Disk contention when increasing the number of procs

First experiments with guaranteed access to local disks

- cluster of Itanium2 processors (Grenoble), 3 GB per node
- In parallel : between 1 and 10 % performance loss maximum

Study of the solution step



Solution time becomes critical

- asynchronous prefetch mechanisms needed
- avoid I/O of small granularity
- more complex memory management (multiple or cyclic workspaces)

Limitations of the Multifrontal Method ?

Out-of-Core : left-looking vs multifrontal

- Rothberg and Schreiber (1999) ; Rotkin and Toledo (2004)
- (switch to) left-looking to avoid large frontal matrices
- possibly more I/O in multifrontal (if active memory is OOC)

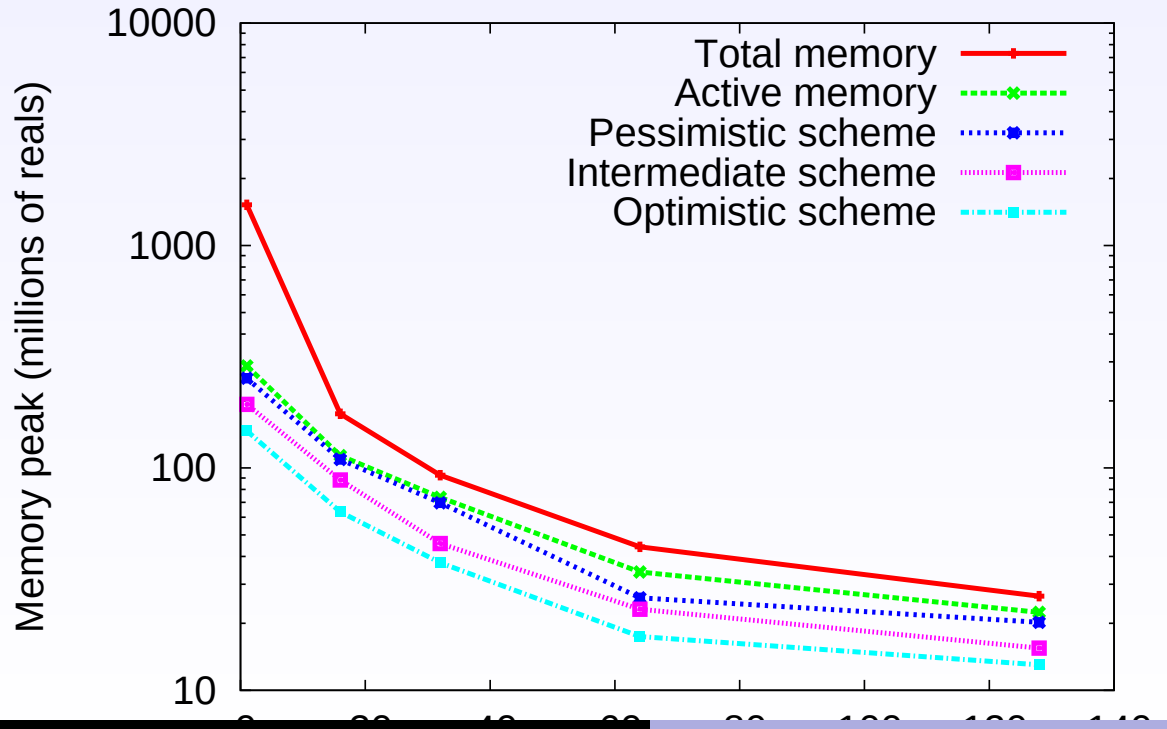
However :

- Frontal matrices can be distributed on several processors
- Multifrontal method : each data is written once, read once
- Guermouche, L'Excellent '05 : pre-allocating the parent can reduce the volume of active memory (and of I/O)

Simulation of an out-of-core stack management

When assembling a node (type 1, master or slave of type 2, supposed to fit in memory), different scenarios :

- pessimistic scheme : all its children have been prefetched
- intermediate scheme : children loaded from disk one by one
- optimistic scheme : only load a small block of each child



Preliminary results

Matrix AUDI_KW (METIS)

Plan de l'exposé

Out-of-core factorization : Future work

- Assess memory limits of parallel multifrontal approach (simulations)
- Out-of-core stack memory
  - Sequential case : window mechanism
  - Parallel case :
    - Stack memory is not exactly accessed in LIFO order
    $\Rightarrow$ find heuristics to prefetch contribution blocks and/or modify scheduling
- Adapt hybrid scheduling strategies to parallel out-of-core factorization
- Cases that almost fit in memory : try to keep most of the factors in-core
- (parallel) out-of-core processing (assembly/factorization) of large frontal matrices