

Large Scale Sparse Linear Algebra



P. Amestoy (INP-N7, IRIT)
A. Buttari (CNRS, IRIT)
T. Mary (CNRS, LIP6)

A. Guermouche (Univ. Bordeaux, LaBRI),
J.-Y. L'Excellent (INRIA, LIP, ENS-Lyon)
B. Uçar (CNRS, LIP, ENS-Lyon)
F.-H. Rouet (LSTC, Livermore, USA)
C. Weisbecker (LSTC, Livermore, USA)

November 3, 2020

Birds-eye perspective and basics

Outline

Birds-eye perspective and background

Acknowledgements and References

Direct methods based on Gaussian elimination

Sparse Linear algebra and HPC: a common story

Outline of the lectures

Acknowledgements

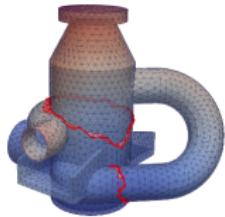
Acknowledgements

- LIP ENS Lyon and CALMIP (Occitanie, Toulouse) for providing access to the target computers used for most recent experiments
1. Distributed memory experiments are done on the eos supercomputer at the CALMIP "Occitanie Mesocentre" in Toulouse (grant 2014-P0989):
 - Two Intel(r) 10-cores Ivy Bridge @ 2,8 GHz
 - Peak per core is 22.4 GF/s
 - 64 GB memory per node
 - Infiniband FDR interconnect
 2. Shared memory experiments are done on grunch at the LIP laboratory of Lyon:
 - Two Intel(r) 14-cores Haswell @ 2,3 GHz
 - Peak per core is 36.8 GF/s
 - Total memory is 768 GB

A selection of references

- Books/HDR thesis
 - Duff, Erisman and Reid, Direct methods for Sparse Matrices, Clarendon Press, Oxford 2017, Second Editon [4]
 - George, Liu, and Ng, *Computer Solution of Sparse Positive Definite Systems*, book to appear (2004)
 - J.-Y. L'Excellent, *Multifrontal Methods: Parallelism, Memory Usage and Numerical Aspects*, HDR thesis, ENS-Lyon, 2012, [14].
<http://tel.archives-ouvertes.fr/tel-00737751>.
- Articles
 - Gilbert and Liu, Elimination structures for unsymmetric sparse LU factors, SIMAX, 1993., [12].
 - Heath and E. Ng and B. W. Peyton, Parallel Algorithms for Sparse Linear Systems, SIAM review, 1991, [13].
 - Liu, The role of elimination trees in sparse factorization, SIMAX, 1990 [16].
 - Davis, Rajamanickam, and Sid-Lakhdar, A survey of direct methods for sparse linear systems (184 pages), Acta Numerica, 2016, [2].

Large scale sparse linear solvers



pump (credits:
Code_Aster)

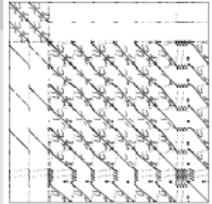
Discretisation of physical problem

→ Solve $\mathbf{AX} = \mathbf{B}$ or $\mathbf{Ax} = \mathbf{b}$, \mathbf{A} large and sparse

- Direct solution: factor $\mathbf{A} = \mathbf{LU}$ (or \mathbf{LDL}^T if symmetric)
then $\mathbf{X} = \mathbf{U}^{-1}(\mathbf{L}^{-1} \mathbf{B})$
- Iterative solvers: $\mathbf{x} = \lim_k \mathbf{x}_k$

Typical numbers (illustration on the pump problem)

- \mathbf{A} : $n = 5.4 \times 10^6$ with $nnz = 2 \times 10^8$ nonzeros
- Target computer:
 - Peak performance: 518 Gflops/s (14 cores × 37 Gflops/s/core)
 - Max memory bandwidth: $bw = 28$ GB/s



Three classes of approaches to solve $A x = b$

Direct methods:

- Gaussian elimination, QR, LU decomposition:

$$A = \begin{pmatrix} 5 & -3 & 0 \\ 0 & 2 & -5 \\ 1 & 0 & 8 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.2 & 0.3 & 1 \end{pmatrix} \times \begin{pmatrix} 5 & -3 & 0 \\ 0 & 2 & -5 \\ 0 & 0 & 9.5 \end{pmatrix} = LU$$

$L(Ux) = b$ solved in two steps:

$$(i) \quad y = L^{-1} \begin{pmatrix} 5 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 5 \\ 1 \\ -1.3 \end{pmatrix} \text{ then (ii)} \quad x = U^{-1}y = \begin{pmatrix} 1.0947368 \\ 0.1578947 \\ -0.1368421 \end{pmatrix}$$

Iterative methods:

Build a sequence of iterates x_0, \dots, x_k until $\|Ax_k - b\|$ small enough

Example (Jacobi): $x_0, x_{k+1} = x_k + D^{-1} \times (b - Ax_k)$, $D = \text{diag}(A)$

$$x_0 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Hybrid methods: mix direct and iterative methods

Three classes of approaches to solve $A x = b$

Direct methods:

- Gaussian elimination, QR, LU decomposition:

$$A = \begin{pmatrix} 5 & -3 & 0 \\ 0 & 2 & -5 \\ 1 & 0 & 8 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.2 & 0.3 & 1 \end{pmatrix} \times \begin{pmatrix} 5 & -3 & 0 \\ 0 & 2 & -5 \\ 0 & 0 & 9.5 \end{pmatrix} = LU$$

$L(Ux) = b$ solved in two steps:

$$(i) \quad y = L^{-1} \begin{pmatrix} 5 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 5 \\ 1 \\ -1.3 \end{pmatrix} \text{ then (ii)} \quad x = U^{-1}y = \begin{pmatrix} 1.0947368 \\ 0.1578947 \\ -0.1368421 \end{pmatrix}$$

Iterative methods:

Build a sequence of iterates x_0, \dots, x_k until $\|Ax_k - b\|$ small enough

Example (Jacobi): $x_0, x_{k+1} = x_k + D^{-1} \times (b - Ax_k)$, $D = \text{diag}(A)$

$$x_1 = \begin{pmatrix} 1 \\ 0.5 \\ 0 \end{pmatrix}$$

Hybrid methods: mix direct and iterative methods

Three classes of approaches to solve $A x = b$

Direct methods:

- Gaussian elimination, QR, LU decomposition:

$$A = \begin{pmatrix} 5 & -3 & 0 \\ 0 & 2 & -5 \\ 1 & 0 & 8 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.2 & 0.3 & 1 \end{pmatrix} \times \begin{pmatrix} 5 & -3 & 0 \\ 0 & 2 & -5 \\ 0 & 0 & 9.5 \end{pmatrix} = LU$$

$L(Ux) = b$ solved in two steps:

$$(i) \quad y = L^{-1} \begin{pmatrix} 5 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 5 \\ 1 \\ -1.3 \end{pmatrix} \text{ then (ii)} \quad x = U^{-1}y = \begin{pmatrix} 1.0947368 \\ 0.1578947 \\ -0.1368421 \end{pmatrix}$$

Iterative methods:

Build a sequence of iterates x_0, \dots, x_k until $\|Ax_k - b\|$ small enough

Example (Jacobi): $x_0, x_{k+1} = x_k + D^{-1} \times (b - Ax_k)$, $D = \text{diag}(A)$

$$x_2 = \begin{pmatrix} 1.3 \\ -0.5 \\ -0.125 \end{pmatrix}$$

Hybrid methods: mix direct and iterative methods

Three classes of approaches to solve $A x = b$

Direct methods:

- Gaussian elimination, QR, LU decomposition:

$$A = \begin{pmatrix} 5 & -3 & 0 \\ 0 & 2 & -5 \\ 1 & 0 & 8 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.2 & 0.3 & 1 \end{pmatrix} \times \begin{pmatrix} 5 & -3 & 0 \\ 0 & 2 & -5 \\ 0 & 0 & 9.5 \end{pmatrix} = LU$$

$L(Ux) = b$ solved in two steps:

$$(i) \quad y = L^{-1} \begin{pmatrix} 5 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 5 \\ 1 \\ -1.3 \end{pmatrix} \text{ then (ii)} \quad x = U^{-1}y = \begin{pmatrix} 1.0947368 \\ 0.1578947 \\ -0.1368421 \end{pmatrix}$$

Iterative methods:

Build a sequence of iterates x_0, \dots, x_k until $\|Ax_k - b\|$ small enough

Example (Jacobi): $x_0, x_{k+1} = x_k + D^{-1} \times (b - Ax_k)$, $D = \text{diag}(A)$

$$x_3 = \begin{pmatrix} 1.3 \\ 0.187 \\ 0.162 \end{pmatrix}$$

Hybrid methods: mix direct and iterative methods

Three classes of approaches to solve $A x = b$

Direct methods:

- Gaussian elimination, QR, LU decomposition:

$$A = \begin{pmatrix} 5 & -3 & 0 \\ 0 & 2 & -5 \\ 1 & 0 & 8 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.2 & 0.3 & 1 \end{pmatrix} \times \begin{pmatrix} 5 & -3 & 0 \\ 0 & 2 & -5 \\ 0 & 0 & 9.5 \end{pmatrix} = LU$$

$L(Ux) = b$ solved in two steps:

$$(i) \quad y = L^{-1} \begin{pmatrix} 5 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 5 \\ 1 \\ -1.3 \end{pmatrix} \text{ then (ii)} \quad x = U^{-1}y = \begin{pmatrix} 1.0947368 \\ 0.1578947 \\ -0.1368421 \end{pmatrix}$$

Iterative methods:

Build a sequence of iterates x_0, \dots, x_k until $\|Ax_k - b\|$ small enough

Example (Jacobi): $x_0, x_{k+1} = x_k + D^{-1} \times (b - Ax_k)$, $D = \text{diag}(A)$

$$x_4 = \begin{pmatrix} 1.1125 \\ 0.09375 \\ -0.1625 \end{pmatrix}$$

Hybrid methods: mix direct and iterative methods

Three classes of approaches to solve $A x = b$

Direct methods:

- Gaussian elimination, QR, LU decomposition:

$$\mathbf{A} = \begin{pmatrix} 5 & -3 & 0 \\ 0 & 2 & -5 \\ 1 & 0 & 8 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.2 & 0.3 & 1 \end{pmatrix} \times \begin{pmatrix} 5 & -3 & 0 \\ 0 & 2 & -5 \\ 0 & 0 & 9.5 \end{pmatrix} = \mathbf{LU}$$

$\mathbf{L}(\mathbf{Ux}) = \mathbf{b}$ solved in two steps:

$$(i) \mathbf{y} = \mathbf{L}^{-1} \begin{pmatrix} 5 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 5 \\ 1 \\ -1.3 \end{pmatrix} \text{ then (ii) } \mathbf{x} = \mathbf{U}^{-1}\mathbf{y} = \begin{pmatrix} 1.0947368 \\ 0.1578947 \\ -0.1368421 \end{pmatrix}$$

Iterative methods:

Build a sequence of iterates x_0, \dots, x_k until $\|Ax_k - b\|$ small enough

Example (Jacobi): $x_0, x_{k+1} = x_k + D^{-1} \times (b - Ax_k)$, $D = \text{diag}(A)$

$$x_5 = \begin{pmatrix} 1.05625 \\ 0.09375 \\ -0.1390625 \end{pmatrix}$$

Hybrid methods: mix direct and iterative methods

Three classes of approaches to solve $A x = b$

Direct methods:

- Gaussian elimination, QR, LU decomposition:

$$A = \begin{pmatrix} 5 & -3 & 0 \\ 0 & 2 & -5 \\ 1 & 0 & 8 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.2 & 0.3 & 1 \end{pmatrix} \times \begin{pmatrix} 5 & -3 & 0 \\ 0 & 2 & -5 \\ 0 & 0 & 9.5 \end{pmatrix} = LU$$

$L(Ux) = b$ solved in two steps:

$$(i) \quad y = L^{-1} \begin{pmatrix} 5 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 5 \\ 1 \\ -1.3 \end{pmatrix} \text{ then (ii)} \quad x = U^{-1}y = \begin{pmatrix} 1.0947368 \\ 0.1578947 \\ -0.1368421 \end{pmatrix}$$

Iterative methods:

Build a sequence of iterates x_0, \dots, x_k until $\|Ax_k - b\|$ small enough

Example (Jacobi): $x_0, x_{k+1} = x_k + D^{-1} \times (b - Ax_k)$, $D = \text{diag}(A)$

$$x_6 = \begin{pmatrix} 1.05625 \\ 0.1523438 \\ -0.1320313 \end{pmatrix}$$

Hybrid methods: mix direct and iterative methods

Three classes of approaches to solve $A x = b$

Direct methods:

- Gaussian elimination, QR, LU decomposition:

$$A = \begin{pmatrix} 5 & -3 & 0 \\ 0 & 2 & -5 \\ 1 & 0 & 8 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.2 & 0.3 & 1 \end{pmatrix} \times \begin{pmatrix} 5 & -3 & 0 \\ 0 & 2 & -5 \\ 0 & 0 & 9.5 \end{pmatrix} = LU$$

$L(Ux) = b$ solved in two steps:

$$(i) \quad y = L^{-1} \begin{pmatrix} 5 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 5 \\ 1 \\ -1.3 \end{pmatrix} \text{ then (ii)} \quad x = U^{-1}y = \begin{pmatrix} 1.0947368 \\ 0.1578947 \\ -0.1368421 \end{pmatrix}$$

Iterative methods:

Build a sequence of iterates x_0, \dots, x_k until $\|Ax_k - b\|$ small enough

Example (Jacobi): $x_0, x_{k+1} = x_k + D^{-1} \times (b - Ax_k)$, $D = \text{diag}(A)$

$$x_7 = \begin{pmatrix} 1.0914062 \\ 0.1699219 \\ -0.1320312 \end{pmatrix}$$

Hybrid methods: mix direct and iterative methods

Three classes of approaches to solve $A x = b$

Direct methods:

- Gaussian elimination, QR, LU decomposition:

$$\mathbf{A} = \begin{pmatrix} 5 & -3 & 0 \\ 0 & 2 & -5 \\ 1 & 0 & 8 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.2 & 0.3 & 1 \end{pmatrix} \times \begin{pmatrix} 5 & -3 & 0 \\ 0 & 2 & -5 \\ 0 & 0 & 9.5 \end{pmatrix} = \mathbf{LU}$$

$\mathbf{L}(\mathbf{Ux}) = \mathbf{b}$ solved in two steps:

$$(i) \mathbf{y} = \mathbf{L}^{-1} \begin{pmatrix} 5 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 5 \\ 1 \\ -1.3 \end{pmatrix} \text{ then (ii) } \mathbf{x} = \mathbf{U}^{-1}\mathbf{y} = \begin{pmatrix} 1.0947368 \\ 0.1578947 \\ -0.1368421 \end{pmatrix}$$

Iterative methods:

Build a sequence of iterates x_0, \dots, x_k until $\|Ax_k - b\|$ small enough

Example (Jacobi): $x_0, x_{k+1} = x_k + D^{-1} \times (b - Ax_k)$, $D = \text{diag}(A)$

$$x_8 = \begin{pmatrix} 1.0914062 \\ 0.1699219 \\ -0.1320312 \end{pmatrix}$$

Hybrid methods: mix direct and iterative methods

Three classes of approaches to solve $A x = b$

Direct methods:

- Gaussian elimination, QR, LU decomposition:

$$\mathbf{A} = \begin{pmatrix} 5 & -3 & 0 \\ 0 & 2 & -5 \\ 1 & 0 & 8 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.2 & 0.3 & 1 \end{pmatrix} \times \begin{pmatrix} 5 & -3 & 0 \\ 0 & 2 & -5 \\ 0 & 0 & 9.5 \end{pmatrix} = \mathbf{LU}$$

$\mathbf{L}(\mathbf{Ux}) = \mathbf{b}$ solved in two steps:

$$(i) \mathbf{y} = \mathbf{L}^{-1} \begin{pmatrix} 5 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 5 \\ 1 \\ -1.3 \end{pmatrix} \text{ then (ii) } \mathbf{x} = \mathbf{U}^{-1}\mathbf{y} = \begin{pmatrix} 1.0947368 \\ 0.1578947 \\ -0.1368421 \end{pmatrix}$$

Iterative methods:

Build a sequence of iterates x_0, \dots, x_k until $\|Ax_k - b\|$ small enough

Example (Jacobi): $x_0, x_{k+1} = x_k + D^{-1} \times (b - Ax_k)$, $D = \text{diag}(A)$

$$x_9 = \begin{pmatrix} 1.1019531 \\ 0.1589355 \\ -0.1377441 \end{pmatrix}$$

Hybrid methods: mix direct and iterative methods

Three classes of approaches to solve $A x = b$

Direct methods:

- Gaussian elimination, QR, LU decomposition:

$$A = \begin{pmatrix} 5 & -3 & 0 \\ 0 & 2 & -5 \\ 1 & 0 & 8 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.2 & 0.3 & 1 \end{pmatrix} \times \begin{pmatrix} 5 & -3 & 0 \\ 0 & 2 & -5 \\ 0 & 0 & 9.5 \end{pmatrix} = LU$$

$L(Ux) = b$ solved in two steps:

$$(i) \quad y = L^{-1} \begin{pmatrix} 5 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 5 \\ 1 \\ -1.3 \end{pmatrix} \text{ then (ii)} \quad x = U^{-1}y = \begin{pmatrix} 1.0947368 \\ 0.1578947 \\ -0.1368421 \end{pmatrix}$$

Iterative methods:

Build a sequence of iterates x_0, \dots, x_k until $\|Ax_k - b\|$ small enough

Example (Jacobi): $x_0, x_{k+1} = x_k + D^{-1} \times (b - Ax_k)$, $D = \text{diag}(A)$

$$x_{10} = \begin{pmatrix} 1.0953613 \\ 0.1556396 \\ -0.1377441 \end{pmatrix}$$

Hybrid methods: mix direct and iterative methods

Three classes of approaches to solve $A x = b$

Direct methods:

- Gaussian elimination, QR, LU decomposition:

$$A = \begin{pmatrix} 5 & -3 & 0 \\ 0 & 2 & -5 \\ 1 & 0 & 8 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.2 & 0.3 & 1 \end{pmatrix} \times \begin{pmatrix} 5 & -3 & 0 \\ 0 & 2 & -5 \\ 0 & 0 & 9.5 \end{pmatrix} = LU$$

$L(Ux) = b$ solved in two steps:

$$(i) \quad y = L^{-1} \begin{pmatrix} 5 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 5 \\ 1 \\ -1.3 \end{pmatrix} \text{ then (ii)} \quad x = U^{-1}y = \begin{pmatrix} 1.0947368 \\ 0.1578947 \\ -0.1368421 \end{pmatrix}$$

Iterative methods:

Build a sequence of iterates x_0, \dots, x_k until $\|Ax_k - b\|$ small enough

Example (Jacobi): $x_0, x_{k+1} = x_k + D^{-1} \times (b - Ax_k)$, $D = \text{diag}(A)$

$$x_{11} = \begin{pmatrix} 1.0933838 \\ 0.1556396 \\ -0.1369202 \end{pmatrix}$$

Hybrid methods: mix direct and iterative methods

Three classes of approaches to solve $A x = b$

Direct methods:

- Gaussian elimination, QR, LU decomposition:

$$A = \begin{pmatrix} 5 & -3 & 0 \\ 0 & 2 & -5 \\ 1 & 0 & 8 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.2 & 0.3 & 1 \end{pmatrix} \times \begin{pmatrix} 5 & -3 & 0 \\ 0 & 2 & -5 \\ 0 & 0 & 9.5 \end{pmatrix} = LU$$

$L(Ux) = b$ solved in two steps:

$$(i) \quad y = L^{-1} \begin{pmatrix} 5 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 5 \\ 1 \\ -1.3 \end{pmatrix} \text{ then (ii)} \quad x = U^{-1}y = \begin{pmatrix} 1.0947368 \\ 0.1578947 \\ -0.1368421 \end{pmatrix}$$

Iterative methods:

Build a sequence of iterates x_0, \dots, x_k until $\|Ax_k - b\|$ small enough

Example (Jacobi): $x_0, x_{k+1} = x_k + D^{-1} \times (b - Ax_k)$, $D = \text{diag}(A)$

$$x_{12} = \begin{pmatrix} 1.0933838 \\ 0.1576996 \\ -0.1366730 \end{pmatrix}$$

Hybrid methods: mix direct and iterative methods

Three classes of approaches to solve $A x = b$

Direct methods:

- Gaussian elimination, QR, LU decomposition:

$$A = \begin{pmatrix} 5 & -3 & 0 \\ 0 & 2 & -5 \\ 1 & 0 & 8 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.2 & 0.3 & 1 \end{pmatrix} \times \begin{pmatrix} 5 & -3 & 0 \\ 0 & 2 & -5 \\ 0 & 0 & 9.5 \end{pmatrix} = LU$$

$L(Ux) = b$ solved in two steps:

$$(i) \quad y = L^{-1} \begin{pmatrix} 5 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 5 \\ 1 \\ -1.3 \end{pmatrix} \text{ then (ii)} \quad x = U^{-1}y = \begin{pmatrix} 1.0947368 \\ 0.1578947 \\ -0.1368421 \end{pmatrix}$$

Iterative methods:

Build a sequence of iterates x_0, \dots, x_k until $\|Ax_k - b\|$ small enough

Example (Jacobi): $x_0, x_{k+1} = x_k + D^{-1} \times (b - Ax_k)$, $D = \text{diag}(A)$

$$x_{13} = \begin{pmatrix} 1.0946198 \\ 0.1583176 \\ -0.1366730 \end{pmatrix}$$

Hybrid methods: mix direct and iterative methods

Three classes of approaches to solve $A x = b$

Direct methods:

- Gaussian elimination, QR, LU decomposition:

$$\mathbf{A} = \begin{pmatrix} 5 & -3 & 0 \\ 0 & 2 & -5 \\ 1 & 0 & 8 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.2 & 0.3 & 1 \end{pmatrix} \times \begin{pmatrix} 5 & -3 & 0 \\ 0 & 2 & -5 \\ 0 & 0 & 9.5 \end{pmatrix} = \mathbf{LU}$$

$\mathbf{L}(\mathbf{Ux}) = \mathbf{b}$ solved in two steps:

$$(i) \mathbf{y} = \mathbf{L}^{-1} \begin{pmatrix} 5 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 5 \\ 1 \\ -1.3 \end{pmatrix} \text{ then (ii) } \mathbf{x} = \mathbf{U}^{-1}\mathbf{y} = \begin{pmatrix} 1.0947368 \\ 0.1578947 \\ -0.1368421 \end{pmatrix}$$

Iterative methods:

Build a sequence of iterates x_0, \dots, x_k until $\|Ax_k - b\|$ small enough

Example (Jacobi): $x_0, x_{k+1} = x_k + D^{-1} \times (b - Ax_k)$, $D = \text{diag}(A)$

$$x_{14} = \begin{pmatrix} 1.0946198 \\ 0.1583176 \\ -0.1366730 \end{pmatrix} \quad \|Ax_{14} - b\| \approx 8.5 \times 10^{-4} \quad (\approx 10^{-15} \text{ with } LU)$$

Hybrid methods: mix direct and iterative methods

Gaussian elimination : some background



Johann Carl Friedrich Gauss (1777-1855)
German mathematician (arithmetic, differential geometry, algebra), astronomer and physicist
(painting by Gottlieb Biermann (1887))

Gaussian elimination

Modern Gauss-Jordan elimination, also called Gauss pivoting method has been introduced in Europe by Carl Friedrich Gauss and Wilhelm Jordan.

Method known in China since (at least) the first century.

Gaussian elimination

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

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

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

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

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

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

Typical Gaussian elimination step k :

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - \frac{a_{ik}^{(k)}a_{kj}^{(k)}}{a_{kk}^{(k)}}$$

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

- One step of Gaussian elimination can be written:

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

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

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

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

- In dense codes, entries of \mathbf{L} and \mathbf{U} overwrite entries of \mathbf{A} .
- Furthermore, if \mathbf{A} is symmetric, $\boxed{\mathbf{A} = \mathbf{LDL}^T}$ with $d_{kk} = a_{kk}^{(k)}$:
 $A = LU = A^t = U^t L^t$ implies $(U)(L^t)^{-1} = L^{-1}U^t = D$ diagonal and
 $U = DL^t$, thus $A = L(DL^t) = LDL^t$

Dense LU factorization

- Step by step columns of \mathbf{A} are set to zero and \mathbf{A} is updated

$$\mathbf{L}^{(n-1)} \dots \mathbf{L}^{(1)} \mathbf{A} = \mathbf{U} \text{ leading to}$$

$$\mathbf{A} = \mathbf{L}\mathbf{U} \text{ where } \mathbf{L} = [\mathbf{L}^{(1)}]^{-1} \dots [\mathbf{L}^{(n-1)}]^{-1}$$

- zero entries in column of \mathbf{A} can be replaced by entries in \mathbf{L}
- row entries of \mathbf{U} can be stored in corresponding locations of \mathbf{A}

Algorithm 1 Dense LU factorization

```
1: for  $k = 1$  to  $n$  do
2:    $L(k, k) = 1$  ;  $L(k + 1 : n, k) = \frac{A(k+1:n, k)}{A(k, k)}$ 
3:    $U(k, k : n) = A(k, k : n)$ 
4:   for  $j = k + 1$  to  $n$  do
5:     for  $i = k + 1$  to  $n$  do
6:        $A(i, j) = A(i, j) - L(i, k) \times U(k, j)$ 
7:     end for
8:   end for
9: end for
```

- Solution \mathbf{x} of $\mathbf{Ax} = \mathbf{b}$ obtained with forward/backward triangular substitutions:
solve $\mathbf{Ly} = \mathbf{b}$, then $\mathbf{Ux} = \mathbf{y}$
- When $|A(k, k)|$ is relatively too small, numerical pivoting required

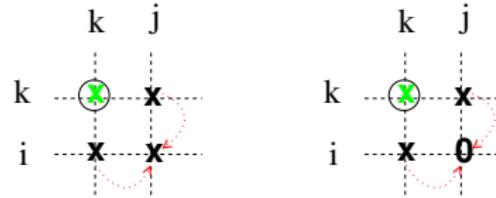
Gaussian elimination and sparsity

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

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

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

- If $a_{ik} \neq 0$ and $a_{kj} \neq 0$ then $a'_{ij} \neq 0$
- If a_{ij} was zero \rightarrow non-zero a'_{ij} must be stored: *fill-in*



Interest of
permuting
a matrix:

$$\begin{pmatrix} X & X & X & X & X \\ X & X & 0 & 0 & 0 \\ X & 0 & X & 0 & 0 \\ X & 0 & 0 & X & 0 \\ X & 0 & 0 & 0 & X \end{pmatrix}$$

$1 \leftrightarrow 5$

$$\begin{pmatrix} X & 0 & 0 & 0 & X \\ 0 & X & 0 & 0 & X \\ 0 & 0 & X & 0 & X \\ 0 & 0 & 0 & X & X \\ X & X & X & X & X \end{pmatrix}$$

Sparse LU factorization : a (too) simple algorithm

Only non-zeros are stored and operated on

Algorithm 2 Simple sparse LU factorization

```
1: Permute matrix A to reduce fill-in and flops (NP complete problem)
2: for  $k = 1$  to  $n$  do
3:    $L(k : k) = 1$  ; For nonzeros in column  $k$ :  $L(k + 1 : n, k) = \frac{A(k+1:n, k)}{A(k, k)}$ 
4:    $U(k, k : n) = A(k, k : n)$ 
5:   for  $j = k + 1$  to  $n$  limited to nonzeros in row  $U(k, :)$  do
6:     for  $i = k + 1$  to  $n$  limited to nonzeros in column  $L(:, k)$  do
7:        $A(i, j) = A(i, j) - L(i, k) \times U(k, j)$ 
8:     end for
9:   end for
10: end for
```

Difficulties

Dynamic data structure for A to accommodate fill-in

Data access efficiency; Can we predict position of fill-in ?

$|A(k, k)|$ too small \longrightarrow numerical permutation needed!

Partial Pivoting

- Partial pivoting: choose at each step the largest element of the column as the pivot
 - avoids large elements in factors matrix (**growth factor**)
- Then (P : permutation), $PA = LU$, $Ly = Pb$, $Ux = y$
- LU with partial pivoting is generally *backward stable*

$$\frac{\|Ax - b\|}{\|A\| \times \|x\| + \|b\|} \approx macheps \quad (1)$$

$$\frac{\|x - x^*\|}{\|x^*\|} \approx macheps \times \kappa(A) \quad (2)$$

- (1) small backward error (and small residual) independently of the conditioning
- (2) accuracy depends on conditioning
if $macheps \approx 10^{-q}$ et $\kappa(A) \approx 10^p$ then x has approximatively $(q - p)$ correct digits

Algorithm 3 LU Factorization with partial pivoting

```
1: for  $k = 1$  to  $n$  do
2:   Find  $j$ ,  $|A(j, k)| = \max_{i=k:n} \{|A(i, k)|\}$ 
3:   if  $|A(j, k)| = 0$  then
4:     Exit. !  $A$  is (almost?) singular
5:   end if
6:   if  $j \neq k$  then
7:     Swap rows  $k$  and  $j$  in  $A(k : n, k : n)$  and  $L(k : n, 1 : k - 1)$ ,
       and in  $b$ 
8:   end if
9:    $L(k : k) = 1$  ;  $L(k + 1 : n, k) = \frac{A(k+1:n,k)}{A(k,k)}$ 
10:  for  $j = k + 1$  to  $n$  do
11:    for  $i = k + 1$  to  $n$  do
12:       $A(i, j) = A(i, j) - L(i, k) \times U(k, j)$ 
13:    end for
14:  end for
15: end for
```

How to control factor growth while preserving sparsity ?

Partial threshold pivoting

- Eligible pivots/threshold u : are not too small (u times) with respect to the maximum in the column.

$$\text{eligible pivots} = \{r \mid |a_{rk}^{(k)}| \geq u \times \max_i |a_{ik}^{(k)}|\}, \quad 0 < u \leq 1.$$

- Among eligible pivots select one preserving well sparsity.

- u is the threshold parameter ($u = 1 \rightarrow$ partial pivoting).
- Restricts the maximum possible growth of: $a_{ij} = a_{ij} - \frac{a_{ik} \times a_{kj}}{a_{kk}}$ to $1 + \frac{1}{u}$ which is sufficient to preserve numerical stability.
- $u \approx 0.1$ is often chosen in practice.

Matrices and graphs

Square unsymmetric matrix: rows/columns and nonzeros correspond to respectively vertices and edges of a graph.

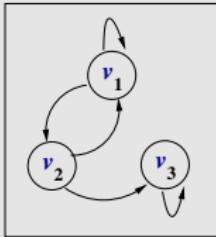
Vertex set V is such that for each $a_{ij} \neq 0$, (v_i, v_j) is an edge.

Square
unsymmetric
pattern matrices

$$A = \begin{matrix} & 1 & 2 & 3 \\ 1 & \times & & \\ 2 & & \times & \\ 3 & \times & & \times \end{matrix}$$

Graph models

- Directed graph



- Bipartite graph representation also possible

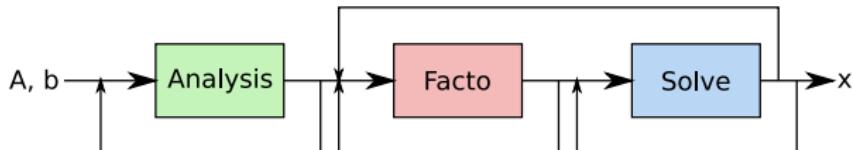
Direct method: matrix factorizations

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

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

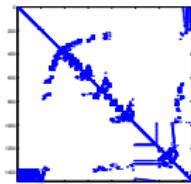
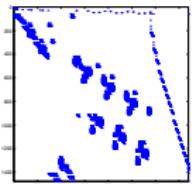
Three-phase scheme to solve $Ax = b$

1. Analysis step
 - preprocess the matrix
 - prepare factorization
2. Factorization
 - compute factors
 - dynamic data-structures
 - most computationally intensive phase
3. Solution based on factored matrices
 - triangular solves
 - improvement of solution (iterative refinement), error analysis



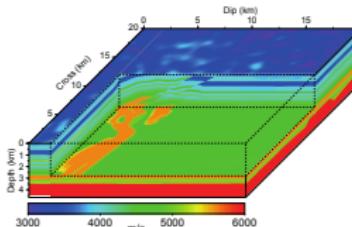
Sparse direct methods: black box for $Ax = b$?

Original ($A = \text{LHR01}$) Preprocessed matrix ($A'(\text{LHR01})$)



- Modified problem: $A'x' = b'$ with $A' = P_nPD_rAD_cQP^tQ_n$
 - Numerical equilibration, **scaling rows/columns** (D_rAD_c)
 - Set large entries on (or near for symmetric) diagonal (AQ)
 - **Row/col permutations** based on non zero structure of $A_1 = PD_rAD_cQ$
 - **Numerical partial pivoting** to preserve accuracy (P_nAQ_n)
- **Symmetric matrices** ($A = LDL^t$):
algorithms must also **preserve symmetry** (flops/memory divided by 2)
- Preprocessing for parallelism: to influence task mapping on the processors and **limit communications/improve data locality**

Matrices from geophysics: Full Waveform Inversion



(3D EAGE/SEG overthrust model)

(credits: SEISCOPE project)



Frequency domain FWI
Helmholtz equations
Complex Unsym. sparse matrix **A**
Multiple (very) sparse **B**
Required accuracy $< 10^{-4}$

fqcy	flops LU	Factor Storage	Peak memory
2 Hz	9.0E+11	3 GB	4 GB
4 Hz	1.6E+13	22 GB	25 GB
8 Hz	5.8E+14	247 GB	283 GB
10 Hz	2.7E+15	728 GB	984 GB

Higher frequency leads to refined model

Iterative methods and preconditioning

Main principles:

- Generate a sequence of approximates x_k converging to the solution at an “estimated” accuracy
- Essentially involves **matrix-vector products**
- Often require **preconditioning techniques**:
 $Ax = b \rightarrow M^{-1}Ax = M^{-1}b$
- Two main classes: stationary (or fixed-point) methods and Krylov methods
- **Hybrid approaches** (e.g. domain decomposition/multigrid) may involve direct solvers on subdomains/coarse grid.

Main issues to performance:

- Quality of preconditioner and [speed of] convergence
- Efficiency relies on: preconditioner - matrix-vector product - direct solver

Interest of preconditioning conjugate gradient

- Let x^* be the exact solution of $Ax = b$; CG generates iterates x_k such that

$$\|x_k - x^*\|_A \leq 2 \cdot \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^k \|x_0 - x^*\|_A$$

where $\kappa(A) = \frac{\lambda_{\max}}{\lambda_{\min}}$, λ_{\max} (resp. λ_{\min}) largest (resp. smallest) eigenvalue,
 $\|\cdot\|$ A -norm

$\kappa(A)$ close to 1 \Rightarrow fast convergence

- If one solves $PAP^T y = Pb$, then $x = P^T y$

Objective: $P^T P \approx A^{-1}$ so that eigenvalues of PAP^T (identical to those of $P^T PA$) are close to 1 (or clustered)

Example of preconditioner:

- $P = L^{-1}$, where $A \approx LL^T$ (approximate factorization of A)

Direct method vs. iterative method

Direct

- Factorization of A
 - may be costly (memory / factors)
 - factors can be reused for multiple/successive right-hand sides
- Very general/robust
 - numerical accuracy (with numerical pivoting)
 - sparse matrices with irregular patterns

Iterative

- Rely on efficient sparse matrix-vector (spmv) product
 - memory effective
 - smaller blocks
 - controlled accuracy is natural
 - solutions with multiple/successive right-hand sides can be problematic/costly
- Efficiency depends on
 - convergence – preconditioning
 - numerical properties – structure of A

Sparse direct solvers main evolutions (1/3)

*Algorithmic evolutions have been motivated by applications needs
but have also been driven by computer architecture evolutions*

From the 70s to the 80s

- General purpose solvers (reordering to preserve sparsity and accuracy)
- Specific band solvers superseded by **Frontal methods** (70-)
 - Simple and efficient use of vector computers
 - Bandwidth reduction preprocessing of A
 - BUT **high complexity on general matrices** and non suitable to //

Reordering (r)evolution (Duff [1980], "A sparse future")

- **Graph based matrix reordering** (numerical values not taken into account)
 - Matrix: $n=1009$, $nnz = 3937$ (large in the 80s!)
 - Reordering time (IBM 370/168): 1970'algorithm: 30 seconds → 1980'algorithm: 0.6 seconds

From the 80s to the 90s

- Supernodal and multifrontal methods
 - reduce flops and memory complexity w.r.t frontal methods
 - suitable for parallel computers,
- Basic Linear Algebra kernels (BLAS) and LAPACK (Linear Algebra Package)
- Memory is critical → Out-Of-Core solvers
(main memory extended to algorithmically use disk/slower memories)
- Reordering (Nested Dissection and Minium Degree)
- Shared memory multithreaded (OpenMP or POSIX) based solvers

Conference in Perugia (1990), comment from a senior researcher:
“... designing efficient direct methods is a solved problem”

Sparse matrix computation on vector computers

1988 Gordon Bell
Award to Boeing-Cray
team



1.68 Gflops/s: "the highest speed ever obtained factoring a non trivial matrix"

A general purpose sparse direct solver

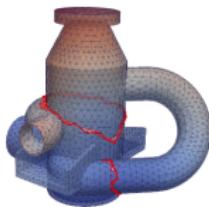
- SMP computer: Cray Y-MP 8 processors (peak performance: 2.5 Gflops/s)
- Matrix: large 3D regular grid (17^3 , 27pt operator, 5 var./node)
 - Matrix $n=24\ 562 \rightarrow 1.68 \text{ Gflops/s}$ with autotasking
(Speed-up of 6, efficiency of 0.672)
- Largest real matrix BCSSTK33:
 - $N = 8\ 738$
 - $663 \text{ Mflops/s} \rightarrow$ efficiency of 0.26

Sparse direct solvers main evolutions (3/3)

Sparse direct solvers after the 90s

- **Distributed memory computers** (1995: Cray T3E, 512 procs),
 - Pure distributed memory (MPI) solvers
 - LU factors are dynamically distributed and **Out-Of-Core**
 - **Partial threshold pivoting and static pivoting**
 - Distributed-memory dense linear algebra (ScaLapack)
- 2000– Increase in the number of processors/cores
 - **Distributed data: \mathbf{A} , \mathbf{X} and \mathbf{B}**
 - **Memory and communication aware** algorithms
 - **Asynchronous** algorithms
- 2010– **Distributed manycore nodes** → mix MPI and multithreading
- Reduce complexity and improve numerical behaviour
 - Exploit **sparsity of \mathbf{B}**
 - **Numerically aware ordering** (often graph based)
 - **Low rank approximation based methods**
- **Heterogeneous computers** with shared-memory nodes + GPU

HPC and sparse linear solvers: time analysis



pump (credits:

Code_Aster)

Discretisation of physical problem

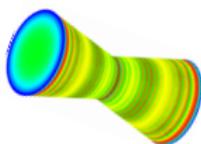
→ Solve $\mathbf{AX} = \mathbf{B}$ or $\mathbf{Ax} = \mathbf{b}$, \mathbf{A} large and sparse

- Direct solution: factor $\mathbf{A} = \mathbf{LU}$ (or \mathbf{LDL}^T if symmetric)
then $\mathbf{X} = \mathbf{U}^{-1}(\mathbf{L}^{-1} \mathbf{B})$
- Iterative solvers: $\mathbf{x} = \lim_k \mathbf{x}_k$

Typical numbers (illustration on the pump problem)

- \mathbf{A} : $n = 5.4 \times 10^6$ with $nnz = 2 \times 10^8$ nonzeros
Peak performance of computer: 518 Gflops/s (14 cores \times 37 Gflops/s/core)
Memory bandwidth: $bw = 28$ GB/s
- Factor $\mathbf{A} = \mathbf{LDL}^T \rightarrow \mathbf{L}$: 5×10^9 nonzeros (40 GB)! (fill -in)
- Direct solution: factor \mathbf{A} ($\mathbf{A} = \mathbf{LDL}^T$) $\rightarrow 25$ Tflops (Tera = 10^{12});
Time(direct) ~ 175 s
- Iterative solution: \mathbf{Ay} : 5×10^8 flops
Min time $\mathbf{Ay} \sim 0.1$ s (7 Gflops/s = $\frac{(2 \text{ flops per access to } \mathbf{A}) \times (bw \text{ in GB/s})}{(8 \text{ Bytes per element of } \mathbf{A})}$)
Time(iterative) ~ 0.1 s $\times NbIter \times NbCol(\mathbf{B}) + t_{precond}$

HPC and sparse linear solvers: cost analysis



Diabolo problem: (\mathbf{A} : $n = 7.9 \times 10^6$, $nnz = 3.2 \times 10^8$)

Credits:

Code_Aster

- Direct solution ($\mathbf{A}=\mathbf{LDL}^T$):
 - $nnz(L) = 4 \times 10^{10}$ (341 GBytes); 1.6 Pflops (Peta = 10^{15})
 - Time(direct, 24 cores)=1 960 s
- Iterative Solution: Compute \mathbf{Ay} : 6.4×10^8 flops → Time ~ 0.1 s (6.4×10^8 / 7 Gflops/s)
 - Time(iterative) ~ 0.1 s × NbIter × NbCol(\mathbf{B}) + $t_{precond}$

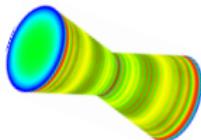
Computer related data

- Peak perf.: 1 100 Gflops/s
(24 cores × 46 Gflops/core)
- Memory bandwidth:
 $bw = 28$ GB/s
- Max Gflop/s for \mathbf{Ay} : $7 = \frac{2 \text{ flops} \times bw}{8 \text{ Bytes}}$
- Typical cost and energy:
0.05€/h/core, 1 kW for 24-core usage

Cost of Direct solution

- Gflops/s rate: 820 Gflops/s (1.6 Pflops / 1960 s)
- Efficiency direct: 74% (820 / 1 100)
- To factorize and solve 1000 systems:
 - ~ 550 hours (NbHours ~ 1000 × 1960 s/3600 s)
 - ~ 660 € (NbH × 24 cores × 0.05€/h)
 - 550 kW.h used during simulation
- Efficiency iterative ~ 0.6%
(7 Gflops/s / 1 100 Gflops/s)

HPC and sparse linear solvers: cost analysis



Diabolo problem: (\mathbf{A} : $n = 7.9 \times 10^6$, $nnz = 3.2 \times 10^8$)

Credits:

Code_Aster

- Direct solution ($\mathbf{A}=\mathbf{LDL}^T$):
 - $nnz(L) = 4 \times 10^{10}$ (341 GBytes); 1.6 Pflops (Peta = 10^{15})
 - Time(direct, 24 cores)=1 960 s
- Iterative Solution: Compute \mathbf{Ay} : 6.4×10^8 flops → Time ~ 0.1 s (6.4×10^8 / 7 Gflops/s)
 - Time(iterative) ~ 0.1 s × NbIter × NbCol(\mathbf{B}) + $t_{precond}$

Today's main technical challenges ($\mathbf{X} = \mathbf{A}^{-1} \mathbf{B}$)

- Reduce flops/memory complexity
- Memory and time scalability
- Efficient use of parallel computers (multicores, GPUs)
- Preserve and/or control accuracy
- \mathbf{B} matrix/vector, dense/sparse
- Application-dependent solvers for iterative ... and direct solution

Outline of the lectures

- PART I: From dense to sparse factorization
 - Reducing complexity (from dense to sparse) and importance of preprocessing
- PART II: Toward an efficient sparse solver
 - Sparsity, fill-in and elimination tree
 - Sparse factorization methods
 - Fill-reducing orderings and complexity of factorization
- PART III: How to address parallelism
 - Algorithms to exploit shared/distributed memory parallelism
 - Computing a complexity bound on parallelism
- PART IV: How to control memory footprint
 - Optimizing memory consumption (sequential and parallel) and computing memory bounds
- PART V: "Fast" Solvers to reduce complexity
 - How to reduce complexity and impact on complexity bounds
 - How to make sparse solver really faster

From dense to efficient sparse factorization methods

Outline

Complexity and performance of dense linear solvers

From dense matrices to band and general systems

Matrix Preprocessing

Fill-reducing heuristics

Fill-in characterization

Preprocessing unsymmetric matrices

Preprocessing symmetric matrices

Complexity of dense LU factorization

- Reminding Algorithm 1:

```
1: for  $k = 1$  to  $n$  do
2:    $L(k : k) = 1$  ;  $L(k + 1 : n, k) = \frac{A(k+1:n,k)}{A(k,k)}$ 
3:    $U(k, k : n) = A(k, k : n)$ 
4:   for  $j = k + 1$  to  $n$  do
5:     for  $i = k + 1$  to  $n$  do
6:        $A(i, j) = A(i, j) - L(i, k) \times U(k, j)$ 
7:     end for
8:   end for
9: end for
```

- $\text{flops}(LU) = \sum_{k=1}^{n-1} (n - k + 2(n - k)^2) \approx 2 \sum_{k=1}^{n-1} (n - k)^2$
 $LU \Rightarrow \frac{2n^3}{3}$ floating-point operations

Complexity of LDL^T factorization

for $k = 1, n - 1$ **do**

if $|\mathbf{A}(k, k)|$ too small exit (small pivots not allowed)

$\mathbf{v}_k = \mathbf{A}(k+1:n, k)^T$ (corresponds to u_k in **LU** Algorithm)

$\mathbf{A}(k+1:n, k) = \mathbf{A}(k+1:n, k) / \mathbf{A}(k, k)$

for $j = k + 1, n$ **do**

$\mathbf{A}(j:n, j) = \mathbf{A}(j:n, j) - \mathbf{A}(j:n, k)^* \mathbf{v}_k(j)$

end for

end for

- flops(LDL^T) = $2 \sum_{k=1}^{n-1} \left(\sum_{i=1}^{n-k} i \right) = 2 \sum_{k=1}^{n-1} \left(\frac{(n-k)(n-k+1)}{2} \right)$

$$\text{flops}(\mathbf{LDL}^T) \approx \sum_{k=1}^{n-1} (n-k)^2 \quad (\text{thus } \frac{1}{2} \text{flops}(\mathbf{LU}))$$

$LDL^T \Rightarrow \frac{n^3}{3}$ floating-point operations

Dense factorization performance influenced by

- Compiler optimizations, use of vector instructions
- Loop optimizations: unrolling, blocking, order of loops (i,j,k in LU)

```
%      do .....
      do .....
      do .....
          a(i,j) = a(i,j) - a(i,k) * a(k,j) / a(k,k)
      end do
      end do
      end do
```

- Locality of access (stride, cache and TLB misses)
- **Arithmetic intensity:** $\frac{\# \text{floating-point operations}}{\text{memory accesses}}$
- Remark for complex arithmetic (1 complex operation=4 flops):
 - $\frac{8n^3}{3}$ flops ($\times 4$) for twice the amount of memory accesses
 - Better **arithmetic intensity** and potential performance

Memory traffic in dense LU factorization

Algorithm 1 can be written as (L and U overwriting entries in A):

```
1: for  $k = 1$  to  $n - 1$  do
2:   Load  $A(k : n, k)$  into cache
3:    $A(k + 1 : n, k) = \frac{A(k+1:n,k)}{A(k,k)}$ 
4:   for  $j = k + 1$  to  $n$  do
5:     Load column  $A(k : n, j)$  into cache
6:      $A(k + 1 : n, j) = A(k + 1 : n, j) - A(k + 1 : n, k) \times A(k, j)$ 
7:   end for
8: end for
```

Assuming $A(k:n,k:n)$ does not fit in cache, volume of memory load operations is:

$$\sum_{k=1}^{n-1} n - k + 1 \text{ for } A(k : n, k) + \sum_{k=1}^{n-1} \left(\sum_{j=k+1}^n (n - k + 1) \right) \text{ for } A(k : n, j)$$

$\approx (\frac{n^3}{3})$ floating-point scalars loaded from memory to cache

Arithmetic intensity ($\frac{\# \text{floating-point operations}}{\text{memory accesses}}$) not so good!

Memory traffic in dense LU factorization

Consider a basic 1D vertical **blocking** $A = (A_1, A_2, \dots, A_N)$,

$$n = N \times b$$

Algorithm 1 becomes:

```
for k = 1 to n step b do
    // Notation: K stands for k:k+b-1, K+1 for k+b:k+2b-1, ...
    Load A(K : N, K) into cache
    Factorize vertical panel A(K : N, K)
    for j = k + b to n step b do
        Load block column A(K : N, J) into cache (J=j:j+b-1)
        A(K + 1 : N, J) = A(K + 1 : N, J) - A(K + 1 : N, K) × A(K, J)
    end for
end for
```

- Memory traffic reduced from $\frac{n^3}{3}$ to $\frac{n^3}{3b} \approx \frac{N^3}{3}$ blocks of size $b \times b = \frac{(n/b)^3}{3} \times b^2$
- Choice of b : as large as possible such that $2 \times N \times b$ scalars fit in cache

Many other optimizations to be done, especially in parallel environments: rely on existing libraries (e.g. BLAS) rather than writing your own code...

Performance illustration

- Intel Sandy bridge processor
- Intel compiler with `-O3 -qarch=native`
- Hand-written version uses KJI variant, without blocking
- Optimized library is Intel MKL, it uses blocking and other optimizations

Comparison of timings in seconds:

Matrix order	Hand-written (Algo 3)	Optimized library 1 thread	16 threads
100x100	0.0035	0.0058	
1000x1000	3.36	0.056	
2000x2000	27.1	1.34	
5000x5000	428.2	6.35	0.66
10000x10000			4.81

Dense linear algebra libraries: BLAS

BLAS : Basic Linear Algebra Subprograms, 3 levels:

- BLAS1 : vector-vector operations - complexity $\mathcal{O}(n)$
- BLAS2 : matrix-vector operations - complexity $\mathcal{O}(n^2)$
- BLAS3 : matrix-matrix operation - complexity $\mathcal{O}(n^3)$

	typical operation	# flop	memory access	ratio
BLAS1 1979	$y = \alpha x + y$	$2n$	$3n + 1$	$\frac{2}{3}$
BLAS2 1988	$y = \alpha Ax + \beta y$	$2n^2$	$n^2 + 3n$	2
BLAS3 1990	$C = \alpha AB + \beta C$	$2n^3$	$4n^2$	$\frac{n}{2}$

Increase of arithmetic intensity from BLAS1 to BLAS3

BLAS Benefits

The BLAS offer several benefits

1. Robustness:

low level details (treatment of exception like overflow are handled by the library).

2. Portability/Efficiency:

thanks to the standardization of the API. Machine-dependent optimization are left to the vendors/specialists (ATLAS: .

3. Readability:

modular description of the mathematical algorithms (Matlab-like).

Several BLAS implementations like ATLAS, GOTO Blas, vendors' BLAS (e.g. MKL for Intel, essl for IBM) provide multithreaded parallelism that can be efficiently exploited on SMP or multicore architectures.

LAPACK: Linear Algebra PACKAGE

- Scientific library developed in Fortran, intensively using BLAS 3 routines
- First public release: 1991. Available on netlib
- Latest release: 3.6.1, June 2016
- Supersedes LINPACK ($Ax = b$ but BLAS 1) and ESIPACK ($Ax = \lambda x$)
- Good numerical robustness (rely on “clean” IEEE arithmetic)
- Parallel implementations on shared memory/multicores inherited from parallel BLAS
- Evolution for distributed memory computers: ScaLAPACK
- Efforts towards massive multicore parallelism and GPU: PLASMA, MAGMA.

Outline

Complexity and performance of dense linear solvers

From dense matrices to band and general systems

Matrix Preprocessing

Fill-reducing heuristics

Fill-in characterization

Preprocessing unsymmetric matrices

Preprocessing symmetric matrices

Factorization complexity of band systems

Let A be a band matrix:

- p (or p_l/p_u): number of lower/upper diagonals of A
- $2p + 1$ (or $p_l + p_u + 1$): bandwidth of A

Complexity

In Algorithm 1, band structure is preserved by LU factorization:

- Step k only updates a $p \times p$ block
- $\text{flops}(LU) \approx 2np^2$ for $n \gg p$
 $(n-p) \times (p \text{ divisions} + p^2 \text{ multiplications} + p^2 \text{ additions}) + \frac{2}{3}p^3$

Warning: partial pivoting increases bandwidth and complexity!

Variable band structure (skyline storage)

Symmetric matrix: store lower-triangular part row-by-row from first non-zero in row until diagonal

Unsymmetric matrix: also store upper-triangular part column-by-column from first non-zero in column down to diagonal element

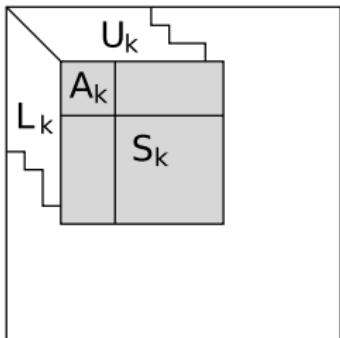
Main interests:

- Band structure is still preserved during Gaussian elimination (without pivoting)
- Less storage than band format
- Can be exploited in frontal solvers after bandwidth/envelope reduction algorithms (e.g. reverse Cuthill-McKee leading to block tri-diagonal structure)

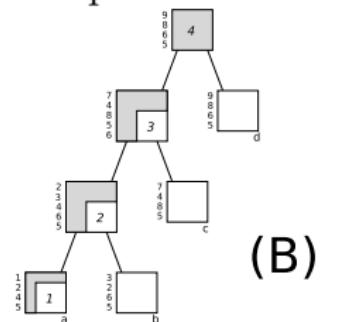
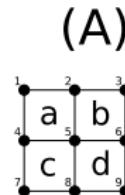
Frontal method

The **frontal method** as initially developed for finite-element problems: $A = \sum_{\ell} A^{[\ell]}$

The set active variables is
a **front**



Assembly of elements and
elimination process



However, complexity of frontal methods (Iron, 1970) remains close to band solvers.

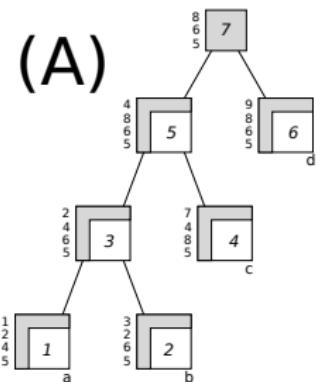
From frontal to multifrontal methods

The **multifrontal method** [9] as a generalization of the frontal method

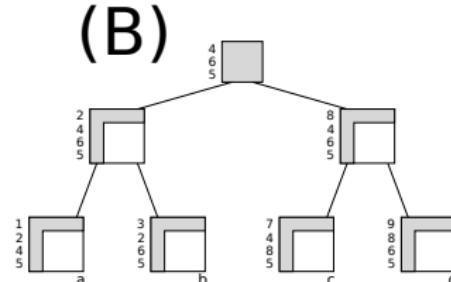
$$(((A^{[a]} + A^{[b]}) + A^{[c]}) + A^{[d]})$$

$$((A^{[a]} + A^{[b]}) + (A^{[c]} + A^{[d]}))$$

Assembly tree for bracketing (A)

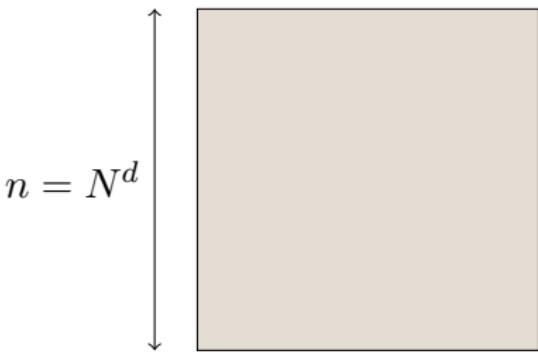
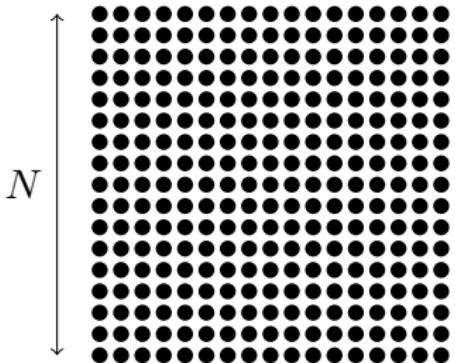


Assembly tree for bracketing (B)



Multifrontal vs Frontal methods: improved complexity (flops and memory) and potential for parallelism

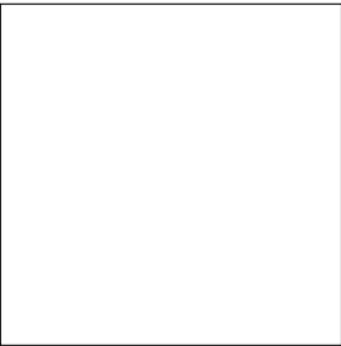
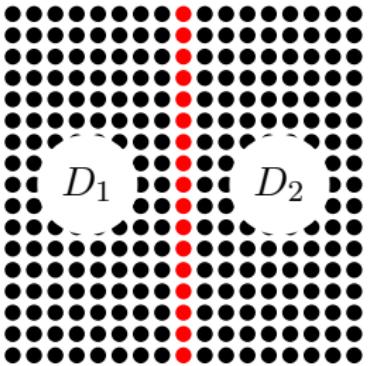
Multifrontal method: separator trees and complexity



2D problem cost \propto

Flops: $O(N^6)$, mem: $O(N^4)$

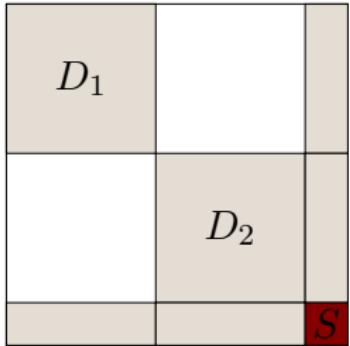
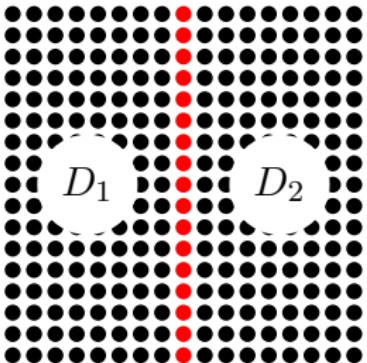
Multifrontal method: separator trees and complexity



2D problem cost \propto

Flops: $O(N^6)$, mem: $O(N^4)$

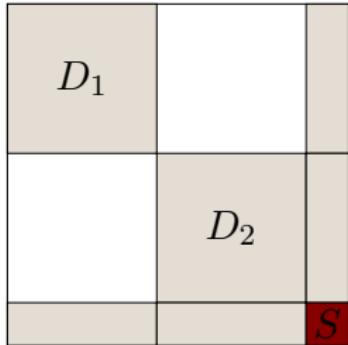
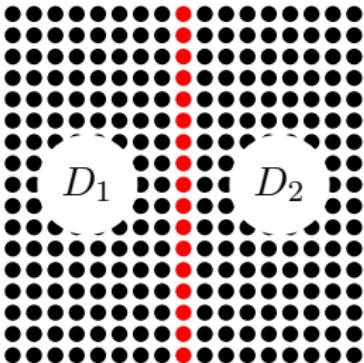
Multifrontal method: separator trees and complexity



2D problem cost \propto

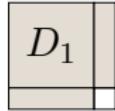
Flops: $O(N^6)$, mem: $O(N^4)$

Multifrontal method: separator trees and complexity

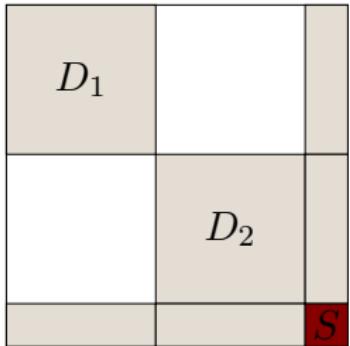
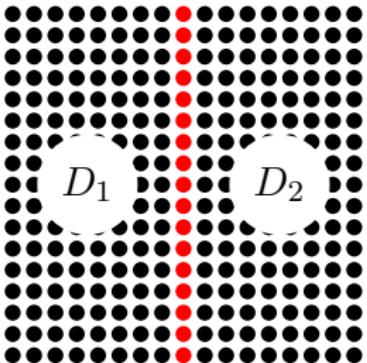


2D problem cost \propto

Flops: $O(N^6)$, mem: $O(N^4)$

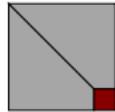


Multifrontal method: separator trees and complexity

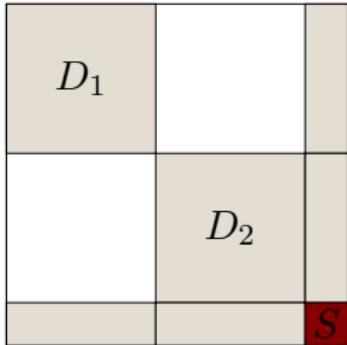
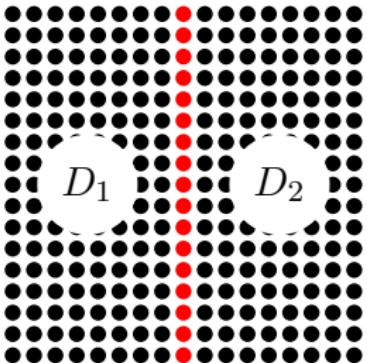


2D problem cost \propto

Flops: $O(N^6)$, mem: $O(N^4)$

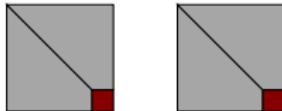


Multifrontal method: separator trees and complexity

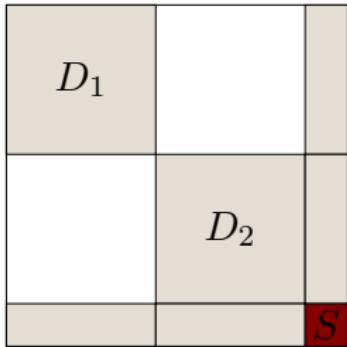
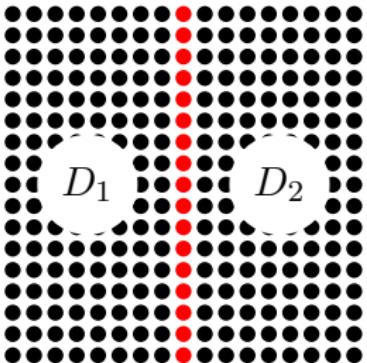


2D problem cost \propto

Flops: $O(N^6)$, mem: $O(N^4)$

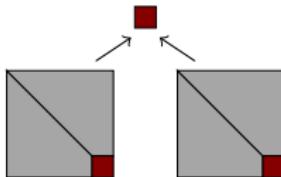


Multifrontal method: separator trees and complexity

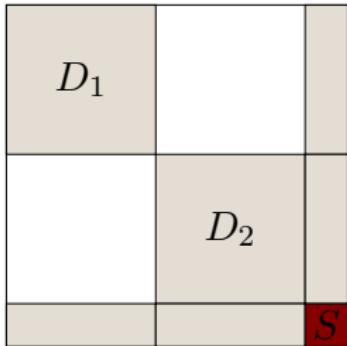
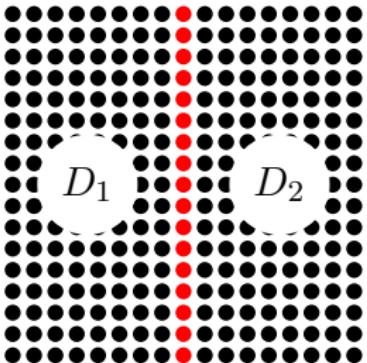


2D problem cost \propto

Flops: $O(N^6)$, mem: $O(N^4)$

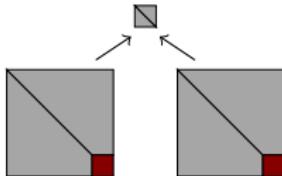


Multifrontal method: separator trees and complexity

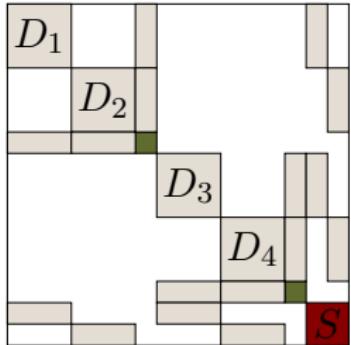
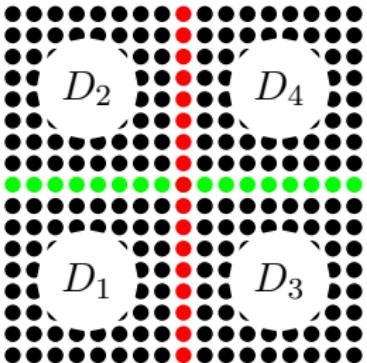


2D problem cost \propto

Flops: $O(N^6)$, mem: $O(N^4)$
→ Flops: $O(N^6/4)$, mem: $O(N^4/2)$



Multifrontal method: separator trees and complexity



2D problem cost \propto

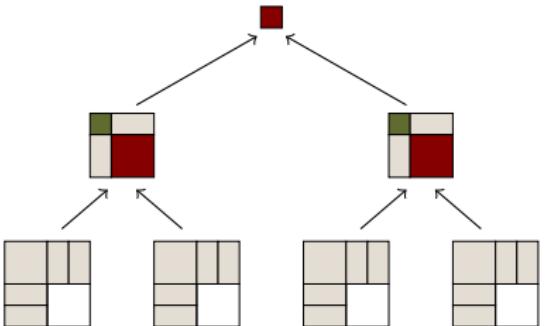
Flops: $O(N^6)$, mem: $O(N^4)$

\rightarrow Flops: $O(N^6/4)$, mem: $O(N^4/2)$

\rightarrow Flops: $O(N^3)$,
mem: $O(N^2 \log(N))$

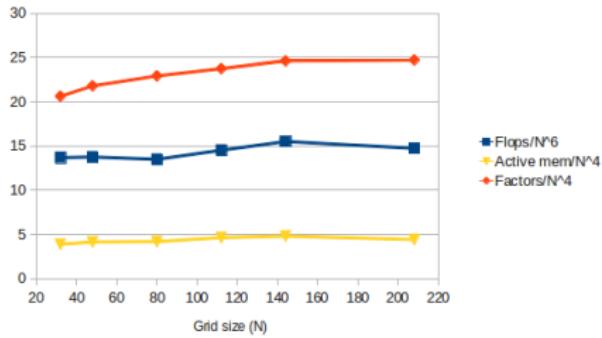
3D problem cost \propto

\rightarrow Flops: $O(N^6)$, mem: $O(N^4)$



Complexity of sparse direct methods

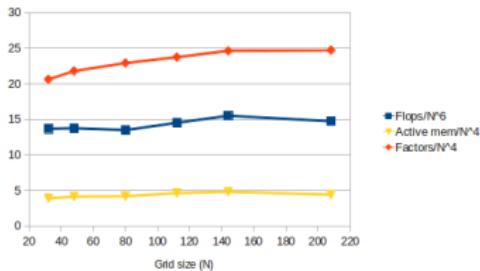
Regular problems (nested dissections)	2D $N \times N$ grid	3D $N \times N \times N$ grid
Nonzeros in original matrix	$\Theta(N^2)$	$\Theta(N^3)$
Nonzeros in factors	$\Theta(N^2 \log(N))$	$\Theta(N^4)$
Floating-point ops	$\Theta(N^3)$	$\Theta(N^6)$



3D example in earth science:
acoustic wave propagation,
27-point finite difference
Objective Seiscope project:
 LU on complete earth

Extrapolation $N^3 = 1000^3$
55 exaflops, 200 TBytes for
factors, 40 TBytes of working
memory!

Cost analysis



3D example in earth science:
acoustic wave propagation, 27-point Finite Difference

Objective Seiscope project:
 LU on complete earth
Extrapolation $N^3 = 1000^3$ grid:
55 exaflops, 200 TBytes for factors,
40 TBytes of working memory!

Computer related data

- Peak perf.: 2 Pflops/s (consumption: 1 MW)
- 0.1€/kW.h

Cost of ONE factorization
assuming 10% efficiency

- $\sim 76 \text{ h } 10\% \times ((55 \text{ exaflops} / 2 \text{ Pflops/s}) / 3600)$
- 76 MW.h** (7600€ for power consumption)

Critical: FLOP/Watt, Efficiency, Flops complexity

Outline

Complexity and performance of dense linear solvers

From dense matrices to band and general systems

Matrix Preprocessing

Fill-reducing heuristics

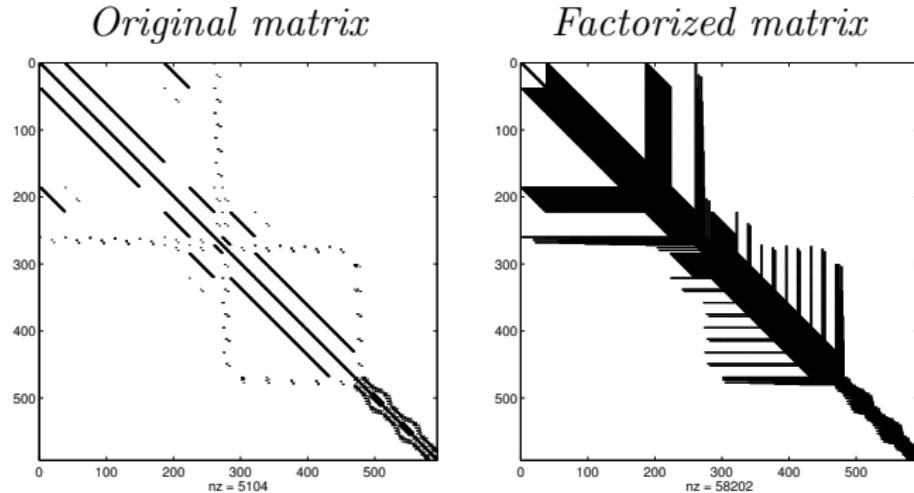
Fill-in characterization

Preprocessing unsymmetric matrices

Preprocessing symmetric matrices

Illustration of fill-in

Harwell-Boeing matrix: dwt_592.rua, structural computing on a submarine. NZ(LU factors)=58202



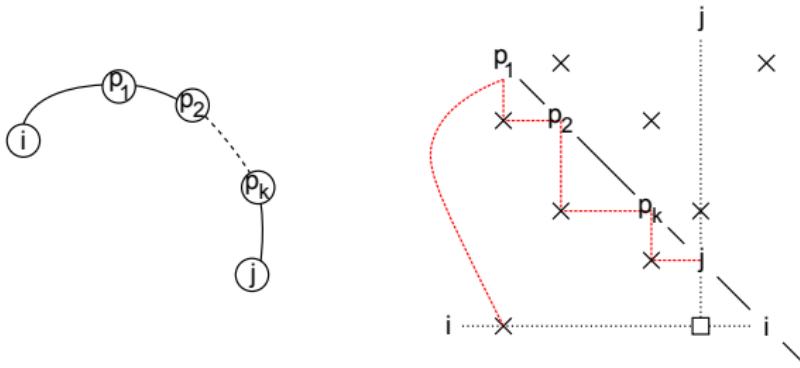
Can we relate entries in \mathbf{L} to entries in \mathbf{A} ?

Fill-in characterization

Let A be a symmetric matrix ($G(A)$ its associated graph), L the matrix of factors $A = LL^t$;

Fill path theorem, Rose, Tarjan, Lueker [20]

$l_{ij} \neq 0$ iff there is a path in $G(A)$ between i and j such that all nodes in the path have indices smaller than both i and j .

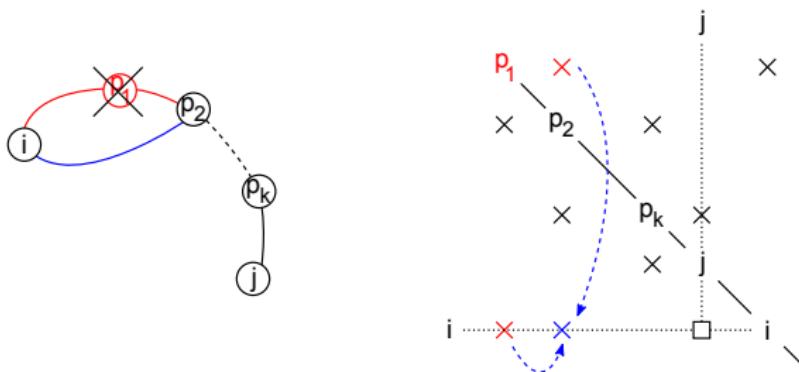


Fill-in characterization (proof intuition)

Let A be a symmetric matrix ($G(A)$ its associated graph), L the matrix of factors $A = LL^t$;

Fill path theorem, Rose, Tarjan, Lueker [20]

$l_{ij} \neq 0$ iff there is a path in $G(A)$ between i and j such that all nodes in the path have indices smaller than both i and j .

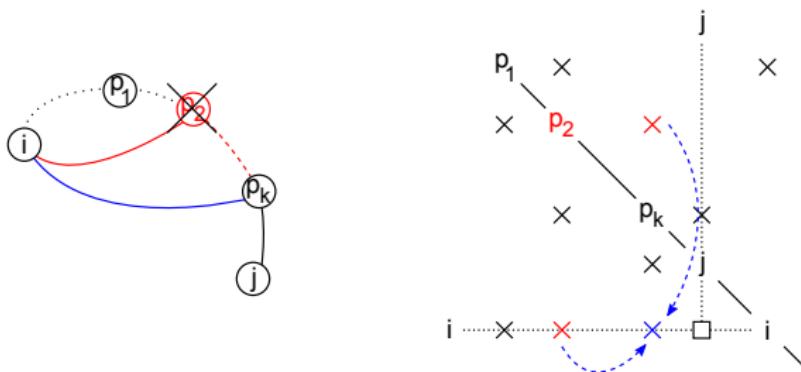


Fill-in characterization (proof intuition)

Let A be a symmetric matrix ($G(A)$ its associated graph), L the matrix of factors $A = LL^t$;

Fill path theorem, Rose, Tarjan, Lueker [20]

$l_{ij} \neq 0$ iff there is a path in $G(A)$ between i and j such that all nodes in the path have indices smaller than both i and j .

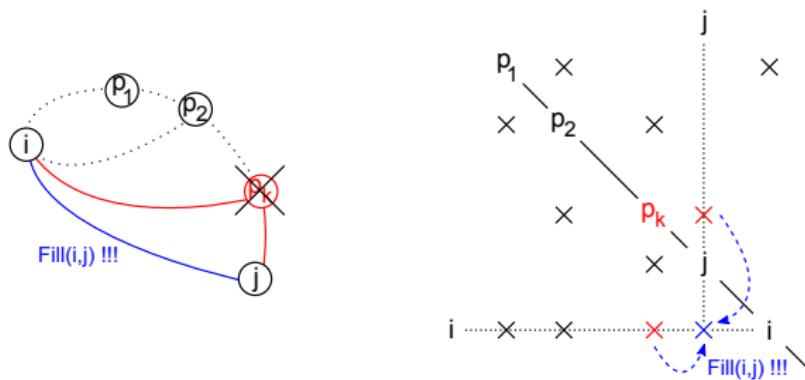


Fill-in characterization (proof intuition)

Let A be a symmetric matrix ($G(A)$ its associated graph), L the matrix of factors $A = LL^t$;

Fill path theorem, Rose, Tarjan, Lueker [20]

$(l_{ij}) \neq 0$ iff there is a path in $G(A)$ between i and j such that all nodes in the path have indices smaller than both i and j .

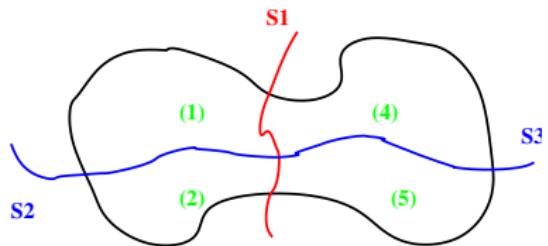


Fill-reducing heuristics

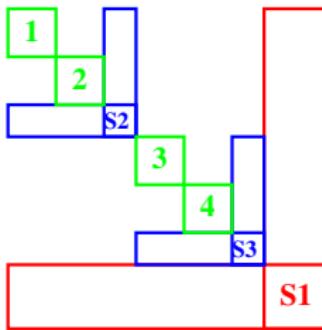
Three main classes of methods for minimizing fill-in during factorization

- **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)
(Remark: interpretation using the fill-path theorem)

Graph partitioning



Permuted matrix



Fill-reducing heuristics

- **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).
 - DMLS/CMLS (Markowitz based orderings with Local Symmetrization)
 - Minimum degree or Minimum fill-in (for symmetric matrices).
- **Hybrid approaches:** Once the matrix is permuted to block structure, local heuristics are used within the blocks.
Metis and SCOTCH software are nested dissection based ordering using minimum degree like orderings on local graphs

Preprocessing unsymmetric matrices - scaling

- **Objective:** Matrix equilibration to help threshold pivoting.
- Row and column scaling : $B = D_r A D_c$ where D_r, D_c are diagonal matrices to respectively scale rows and columns of A
 - reduce the amount of numerical problems

$$\text{Let } A = \begin{bmatrix} 1 & 2 \\ 10^{16} & 10^{16} \end{bmatrix} \rightarrow \text{Let } B = D_r A = \begin{bmatrix} 1 & 2 \\ 1 & 1 \end{bmatrix}$$

- better detect real problems.

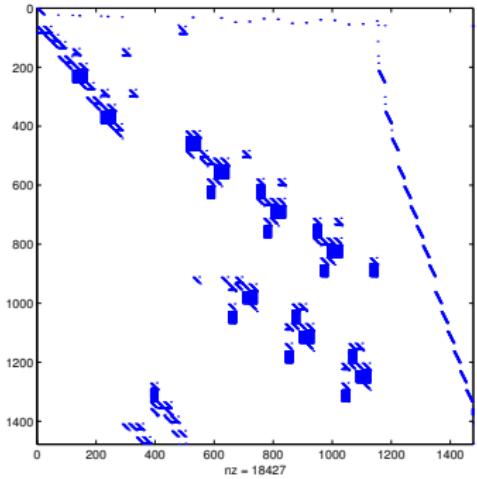
$$\text{Let } A = \begin{bmatrix} 1 & 10^{16} \\ 1 & 1 \end{bmatrix} \rightarrow \text{Let } B = D_r A = \begin{bmatrix} 10^{-16} & 1 \\ 1 & 1 \end{bmatrix}$$

- Influence quality of **fill-in estimations and accuracy**.
- Should be activated when the number of uneliminated variables is large.

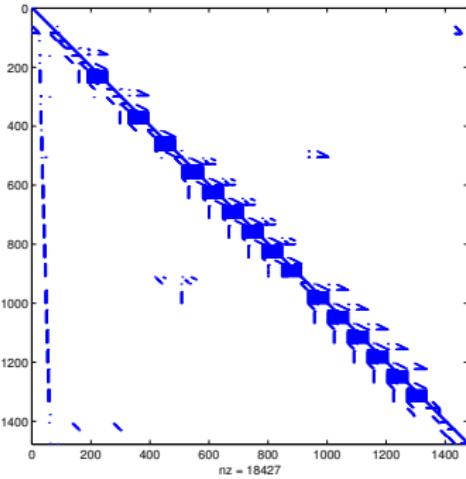
Preprocessing - Maximum weighted matching (I)

- **Objective:** Set large entries on the diagonal
 - Unsymmetric permutation and scaling
 - Preprocessed matrix $\mathbf{B} = \mathbf{D}_1 \mathbf{A} \mathbf{Q} \mathbf{D}_2$ is such that $|b_{ii}| = 1$ and $|b_{ij}| \leq 1$

Original ($A = \text{LHR01}$)



Permuted ($A' = AQ$)



Combine maximum transversal and fill-in reduction

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

Preprocessing - Maximum weighted matching

- *Influence of maximum weighted matching* [5, 6] on the performance

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

- On very unsymmetric matrices: reduce flops, factor size and memory used.

Preprocessing - Maximum weighted matching

- *Influence of maximum weighted matching* [5, 6] on the performance

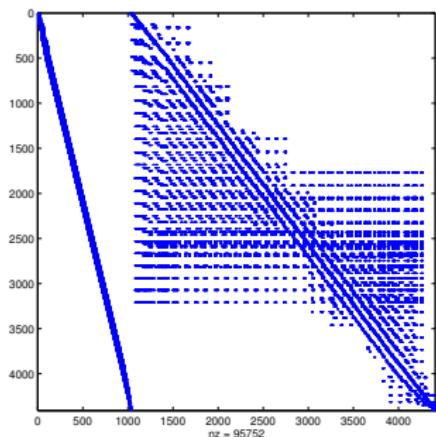
Matrix	Symmetry	$ LU $ (10^6)	Flops (10^9)	Backwd Error
TWOTONE	OFF	28	235	10^{-6}
	ON	43	22	10^{-12}
FIDAPM11	OFF	100	16	10^{-10}
	ON	46	28	10^{-11}

- On very unsymmetric matrices: reduce flops, factor size and memory used.
- In general improve accuracy, and reduce number of iterative refinements.
- Improve reliability of memory estimates.

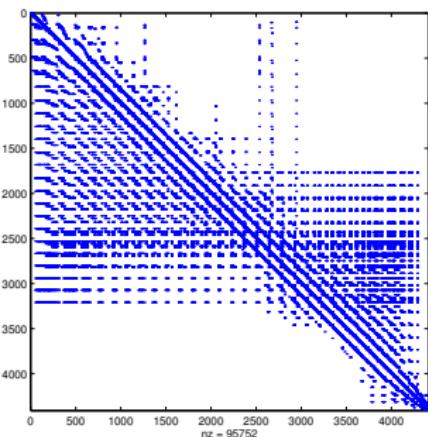
Permuting large entries on the diagonal using MC64

Illustration on matrix AV4408, irregular finite-elements

Original matrix : A



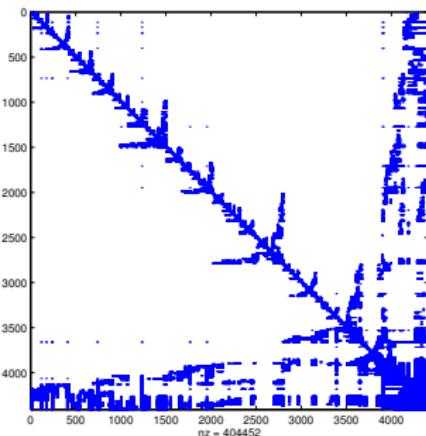
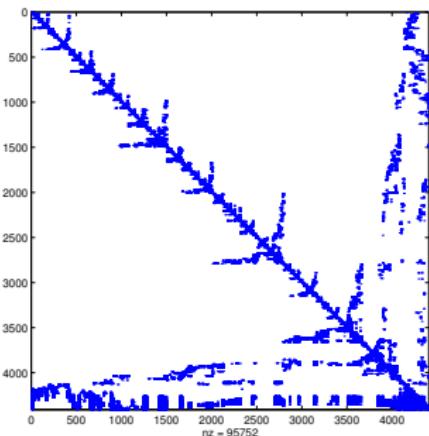
Permuted matrix : AQ



... symmetric reordering and factorization

Illustration on matrix AV4408, irregular finite-elements

AMD ordering: $A_{pre} = PAQP^T$ LU factors of A_{pre}



Preprocessing symmetric matrices [7, 8]

- Symmetric scaling: Adapt MC64 [6] unsymmetric scaling:
let $D = \sqrt{D_r D_c}$, then $B = DAD$ is a symmetrically scaled matrix which satisfies

$$\forall i, |b_{i\sigma(i)}| = \|b_{.\sigma(i)}\|_\infty = \|b_i^T\|_\infty = 1$$

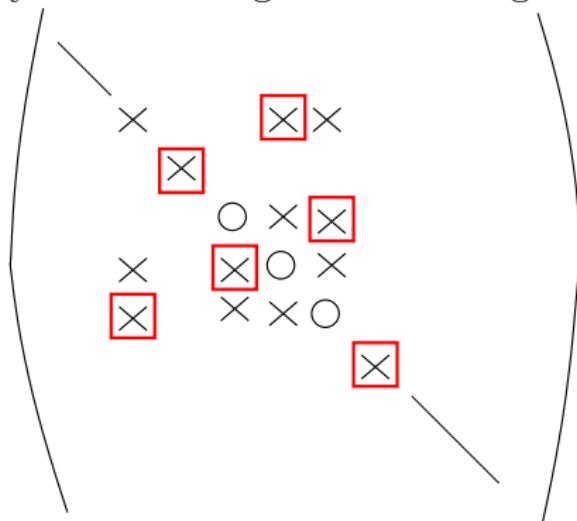
where σ is the permutation from the unsym. transv. algo.

- Influence of scaling on augmented matrices $K = \begin{pmatrix} H & A \\ A^T & 0 \end{pmatrix}$

Scaling :	Total time (seconds)		Nb of entries in factors (millions) (estimated)		Nb of entries in factors (millions) (effective)	
	OFF	ON	OFF	ON	OFF	ON
CONT-300	45	5	12.2	12.2	32.0	12.4
CVXQP3	1816	28	3.9	3.9	62.4	9.3
STOKES128	3	2	3.0	3.0	5.5	3.3

Preprocessing - Compressed ordering

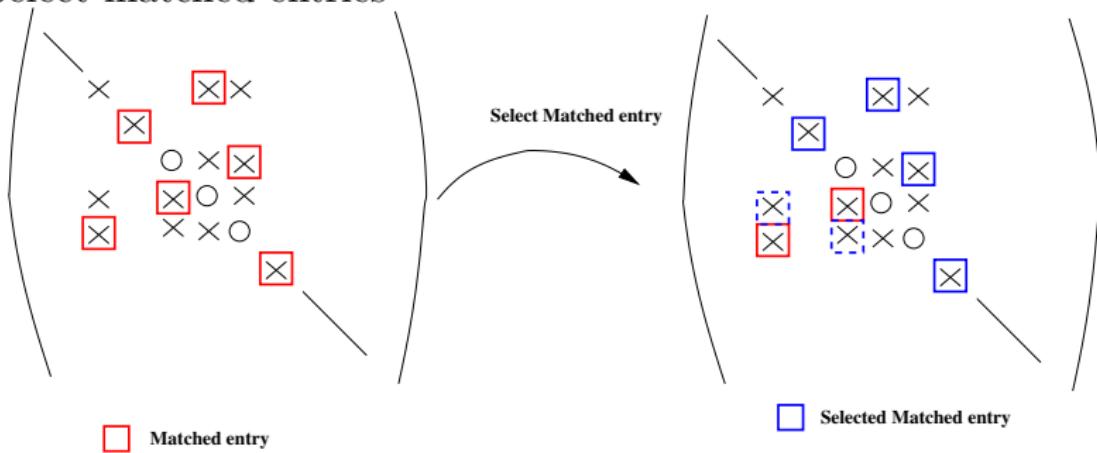
- Perform an unsymmetric weighted matching



Matched entry

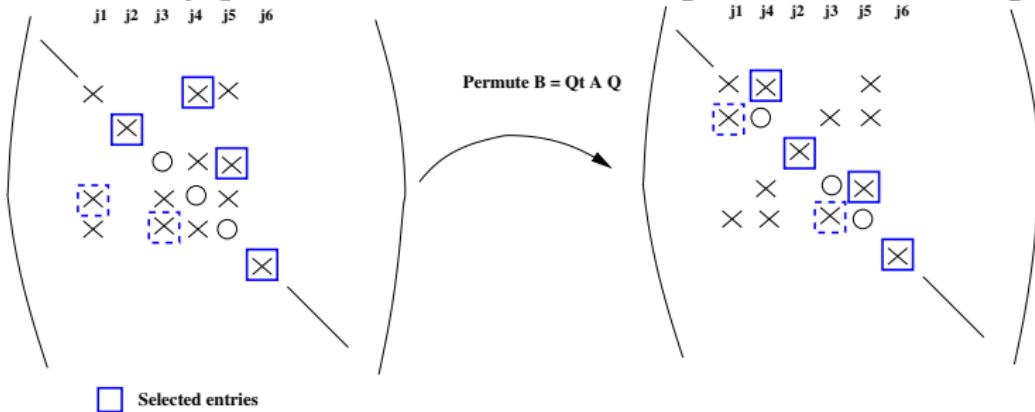
Preprocessing - Compressed ordering

- Perform an unsymmetric weighted matching
- Select matched entries



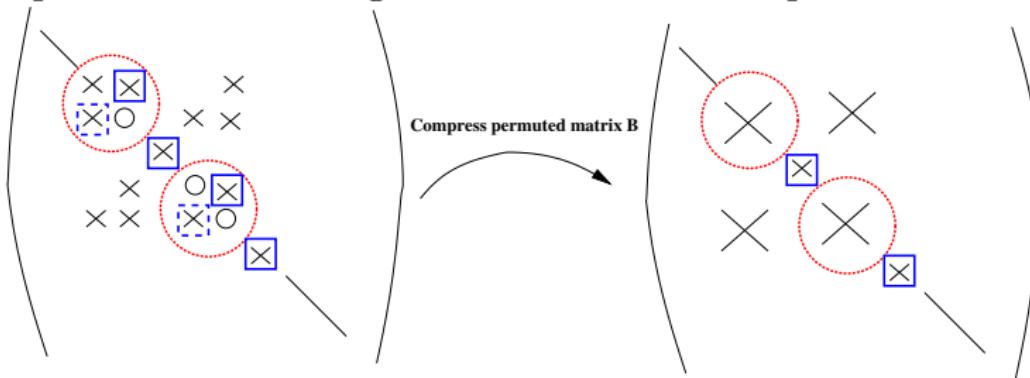
Preprocessing - Compressed ordering

- Perform an unsymmetric weighted matching
- Select matched entries
- Symmetrically permute matrix to set large entries near diagonal



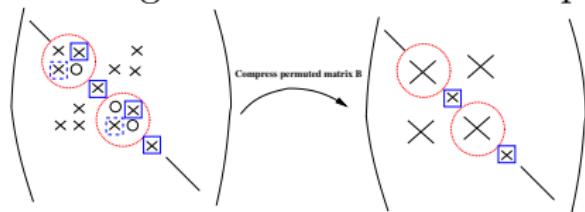
Preprocessing - Compressed ordering

- Perform an unsymmetric weighted matching
- Select matched entries
- Symmetrically permute matrix to set large entries near diagonal
- Compression: 2×2 diagonal blocks become supervariables.



Preprocessing - Compressed ordering

- Perform an unsymmetric weighted matching
- Select matched entries
- Symmetrically permute matrix to set large entries near diagonal
- Compression: 2×2 diagonal blocks become supervariables.



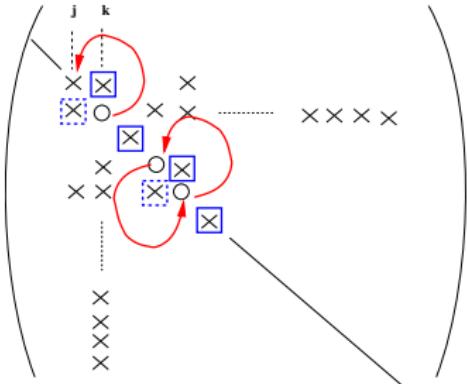
Influence of using a compressed graph (with scaling)

Compression :	Total time (seconds)		Nb of entries in factors in Millions (estimated)		Nb of entries in factors in Millions (effective)	
	OFF	ON	OFF	ON	OFF	ON
CONT-300	5	4	12.3	11.2	12.4	12.4
CVXQP3	28	11	3.9	7.1	9.3	8.5
STOKES128	1	2	3.0	5.7	3.4	5.7

Preprocessing - Constrained ordering

- Part of matrix sparsity is lost during graph compression
- **Constrained ordering** : *only pivot dependency within 2×2 blocks need be respected.*

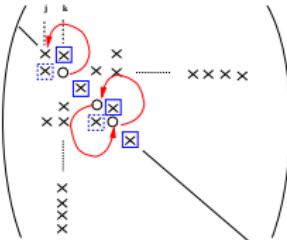
Ex: $k \rightarrow j$ indicates that if k is selected before j then j must be eliminated together with k .



if j is selected first then no more constraint on k .

Preprocessing - Constrained ordering

- Constrained ordering : *only pivot dependency within 2×2 blocks need be respected.*



Influence of using a constrained ordering (with scaling)

Constrained :	Total time (seconds)		Nb of entries in factors in Millions (estimated)		Nb of entries in factors in Millions (effective)	
	OFF	ON	OFF	ON	OFF	ON
CVXQP3	11	8	7.2	6.3	8.6	7.2
STOKES128	2	2	5.7	5.2	5.7	5.3

From the matrix to the assembly tree

Outline

Sparsity, fill-in and dependencies

Matrix factorizations

Fill-in characterization

Dependencies and the elimination tree

Sparse factorization methods

Left-looking sparse Cholesky

Right-looking sparse Cholesky

The Multifrontal method

Sparse triangular solve

Fill-reducing orderings

Local fill-reducing ordering methods

Global fill-reducing ordering methods

Tree amalgamation

Complexity of the factorization

Sparse Matrix Factorizations

- $A \in \mathbb{R}^{m \times m}$, symmetric positive definite $\rightarrow LL^T = A$

$$Ax = b$$

- $A \in \mathbb{R}^{m \times m}$, symmetric $\rightarrow LDL^T = A$

$$Ax = b$$

- $A \in \mathbb{R}^{m \times m}$, unsymmetric $\rightarrow LU = A$

$$Ax = b$$

- $A \in \mathbb{R}^{m \times n}$, $m \neq n \rightarrow QR = A$

$$\min_x \|Ax - b\| \quad \text{if } m > n$$

$$\min \|x\| \text{ such that } Ax = b \quad \text{if } n > m$$

The dense Cholesky factorization

- Let A be a symmetric positive definite matrix of order n
- The LL^T factorization can be described by the equation:

$$\begin{aligned} A = A_0 &= \begin{bmatrix} a_{11} & a_{:1}^T \\ a_{:1} & A_1 \end{bmatrix} \\ &= \begin{bmatrix} \sqrt{a_{11}} & 0 \\ \frac{a_{:1}}{\sqrt{a_{11}}} & I_{n-1} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & A_1 \end{bmatrix} \begin{bmatrix} \sqrt{a_{11}} & \frac{a_{:1}^T}{\sqrt{a_{11}}} \\ 0 & I_{n-1} \end{bmatrix} \\ &= \begin{bmatrix} l_{11} & 0 \\ l_{:1} & I_{n-1} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & A_1 \end{bmatrix} \begin{bmatrix} l_{11} & l_{:1}^T \\ 0 & I_{n-1} \end{bmatrix} \\ &= L_1 A_1 L_1^T, \text{ where, } A_1 = \overline{A_1} - \frac{a_{:1} a_{:1}^T}{a_{11}} = \overline{A_1} - l_{:1} l_{:1}^T \end{aligned}$$

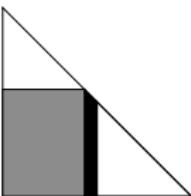
This operation is called trailing submatrix update

- The basic step is applied on $A_1 A_2 \cdots$ to obtain :

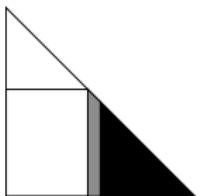
$$A = (L_1 L_2 \cdots L_{n-1}) I_n (L_{n-1}^T \cdots L_2^T L_1^T) = LL^T$$

The dense Cholesky factorization

$$\begin{bmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$



Left-looking



Right-looking

■ used for modification
■ modified

Right-Looking Cholesky

```
for k = 1, ..., n do
     $l_{kk} = \sqrt{a_{kk}^{(k-1)}}$ 
    for i = k + 1, ..., n do
         $l_{ik} = a_{ik}^{(k-1)} / l_{kk}$ 
        for j = k + 1, ..., i do
             $a_{ij}^{(k)} = a_{ij}^{(k-1)} - l_{ik}l_{jk}$ 
        end for
    end for
end for
```

Left-Looking Cholesky

```
for k = 1, ..., n do
    for i = k, ..., n do
        for j = 1, ..., k - 1 do
             $a_{ik}^{(k)} = a_{ik}^{(k-1)} - l_{ij}l_{kj}$ 
        end for
    end for
     $l_{kk} = \sqrt{a_{kk}^{(k)}}$ 
    for i = k + 1, ..., n do
         $l_{ik} = a_{ik}^{(k)} / l_{kk}$ 
    end for
end for
```

Sparsity

Sparse matrices arise in many applications such as structural mechanics, fluid dynamics, geophysics and only have **few nonzeroses per row** ($O(10)$, for example). We want to take advantage of this in order to

- Reduce the complexity of the factorization: fewer floating point operations are needed to complete the factorization because many coefficients are equal to zero and are not modified by the factorization;
- Reduce the storage: the zero coefficients need not be stored in memory;
- Increase parallelism: because of all the zeroes, different steps of the factorization may concern disjoint subsets of the coefficients and can, thus, be done concurrently.

Unfortunately the factorization of a sparse matrix is problematic due to the presence of **fill-in**.

Factorization of sparse matrices: fill-in

The basic Cholesky step (assuming $a_{ij}^0 = a_{ij}$ and $l_{ij} = a_{ij}^{(j)}$):

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - \frac{a_{ik}^{(k-1)} a_{kj}^{(k-1)}}{a_{kk}^{(k-1)}}$$

Even if $a_{ij}^{(k-1)}$ is null, $a_{ij}^{(k)}$ can be a nonzero

$$A = \begin{pmatrix} a & \bullet & & \bullet & \bullet \\ \bullet & c & \bullet & \bullet & \bullet \\ & d & & & \bullet \\ \bullet & e & f & \bullet & \bullet \\ \bullet & g & h & \bullet & \bullet \\ \bullet & \bullet & i & & j \\ \bullet & \bullet & \bullet & & \end{pmatrix} \quad F = \begin{pmatrix} a & \bullet & & \bullet & \bullet \\ \bullet & c & \bullet & \bullet & \bullet \\ & d & & \bullet & \circ \\ \bullet & e & f & \bullet & \bullet \\ \bullet & g & h & \bullet & \bullet \\ \bullet & \circ & \circ & \bullet & \circ \\ \bullet & \bullet & \bullet & \bullet & \circ \\ \bullet & \bullet & \bullet & \bullet & j \end{pmatrix}$$

- the factorization is more expensive than $\mathcal{O}(nz)$
- higher amount of memory required than $\mathcal{O}(nz)$
- more complicated algorithms to achieve the factorization

Modeling fill-in: adjacency graphs

Symmetric pattern matrix: undirected graph

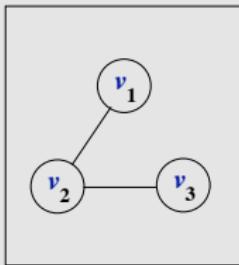
A graph $G = (V, E)$ consists of a finite set V , called the vertex set and a finite, binary relation E on V , called the edge set.

In an **Undirected graph** the edges are unordered pair of vertices, i.e., $\{u, v\} \in E$ for some $u, v \in V$.

The rows/columns and nonzeros of a given sparse matrix correspond (with natural labelling) to the vertices and edges, respectively, of a graph.

Square symmetric pattern matrices

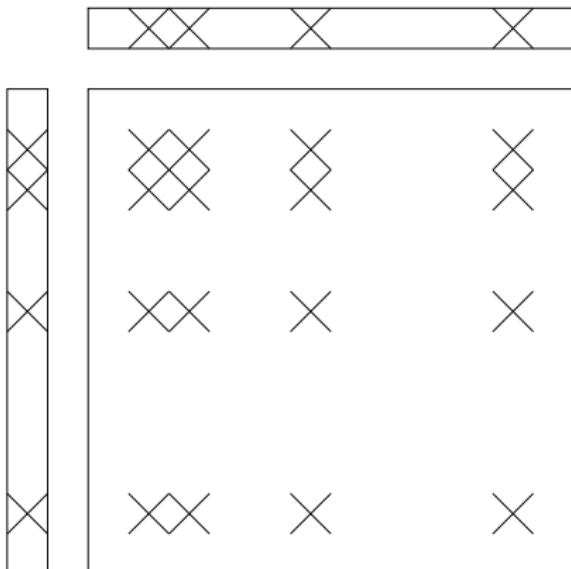
$$A = \begin{pmatrix} & 1 & 2 & 3 \\ 1 & & & \\ 2 & & \times & \\ 3 & \times & \times & \times \\ & & \times & \times \end{pmatrix}$$



Modeling fill-in

Remember the trailing submatrix update $A_1 = \overline{A_1} - \frac{a_{:1}a_{:1}^T}{a_{11}}$.

What is $a_{:1}a_{:1}^T$ in terms of structure?



$a_{:1}$ is the first column of A , thus it contains the **neighbors** of 1 in the adjacency graph of A .

$a_{:1}a_{:1}^T$ results in a dense sub-block in A_1 , i.e., the elimination of a node results in the creation of a **clique** that connects all the neighbors of the eliminated node.

If any of the nonzeros in dense submatrix are not in A , then we have fill-ins.

The elimination process in the graphs

$G_U(V, E) \leftarrow$ undirected graph of A

for $k = 1 : n - 1$ **do**

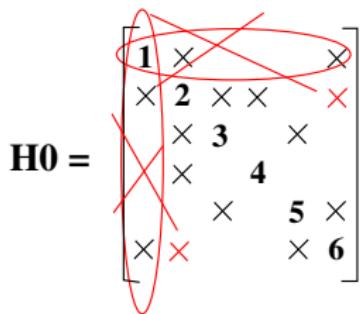
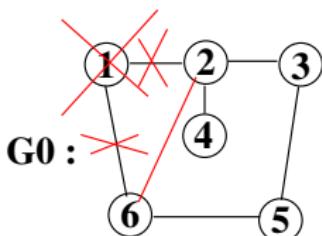
$V \leftarrow V - \{k\}$ {remove vertex k }

$E \leftarrow E - \{(k, \ell) : \ell \in \text{adj}(k)\} \cup \{(x, y) : x \in \text{adj}(k) \text{ and } y \in \text{adj}(k)\}$

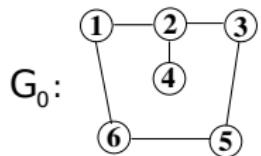
$G_k \leftarrow (V, E)$ {for definition}

end for

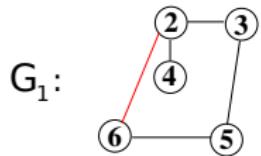
G_k are the so-called **elimination graphs** (Parter [18]).



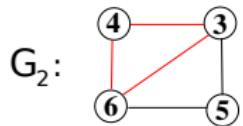
A sequence of elimination graphs



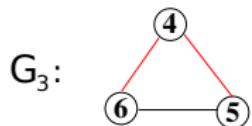
$$A_0 = \begin{bmatrix} 1 & \bullet & & & & \bullet \\ \bullet & 2 & \bullet & \bullet & & \bullet \\ & \bullet & 3 & \bullet & & \bullet \\ & & \bullet & 4 & \bullet & \bullet \\ & & & \bullet & 5 & \bullet \\ & & & & \bullet & 6 \end{bmatrix}$$



$$A_1 = \begin{bmatrix} 2 & \bullet & \bullet & & & \bullet \\ \bullet & 3 & \bullet & \bullet & & \bullet \\ & \bullet & 4 & \bullet & \bullet & \bullet \\ & & \bullet & 5 & \bullet & \bullet \\ & & & \bullet & 6 & \bullet \end{bmatrix}$$



$$A_2 = \begin{bmatrix} 3 & \bullet & \bullet & \bullet \\ \bullet & 4 & \bullet & \bullet \\ & \bullet & 5 & \bullet \\ & & \bullet & 6 \end{bmatrix}$$



$$A_3 = \begin{bmatrix} 4 & \bullet & \bullet \\ \bullet & 5 & \bullet \\ & \bullet & 6 \end{bmatrix}$$

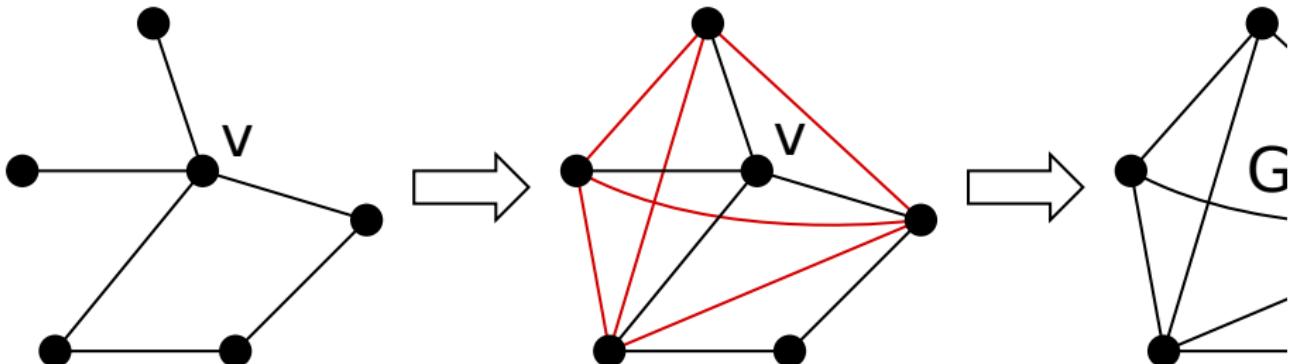
$$Ax = b$$

$$PA\tilde{P}^T P_x = Pb$$

$$\left[\begin{array}{cccccc} l_{11} & & & & & \\ l_{21} & l_{22} & & & & \\ l_{31} & l_{32} & l_{33} & & & \\ l_{41} & l_{42} & l_{43} & l_{44} & & \\ l_{51} & l_{52} & l_{53} & l_{54} & l_{55} & \\ l_{61} & l_{62} & l_{63} & l_{64} & l_{65} & l_{66} \end{array} \right]$$

Elimination process: Formal definitions

- **Deficiency of a vertex:** $D(v)$ is the set of edges defined by
$$D(v) = \{(x, y) : x \in \text{adj}(v) \text{ and } y \in \text{adj}(v) \text{ and } y \notin \text{adj}(x) \text{ and } x \neq y\}$$
- **v -elimination graph:** Apply the elimination process to the vertex v of G to obtain $G_v = V - \{v\}, E(V - \{v\}) \cup D(v)$.



Elimination process: Formal definitions

For a graph $G = (V, E)$, the **elimination process**

$P(G) = [G = G_0, G_1, G_2, \dots, G_{n-1}]$ is the sequence of elimination graphs defined by $G_0 = G$, $G_i = (G_{i-1})_i$

Let $G_i = (V_i, E_i)$ for $i = 0, 1, \dots, n - 1$. The **fill-in** $F(G)$ is defined by

$$F(G) = \bigcup_{i=1}^{n-1} \tau_i$$

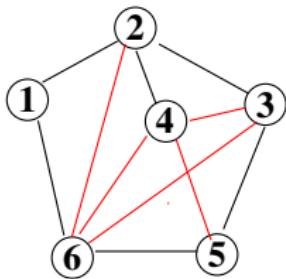
where $\tau_i = D(i)$ in G_{i-1} , and the **elimination graph** is defined by

$$G^+ = (V, E \cup F(G))$$

For a matrix A , τ_i corresponds to the new nonzeros elements, the fill-ins, created during i the step of elimination.

Elimination process

Continuing from the previous sample matrix, we have the filled-graph $G^+(A)$



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

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

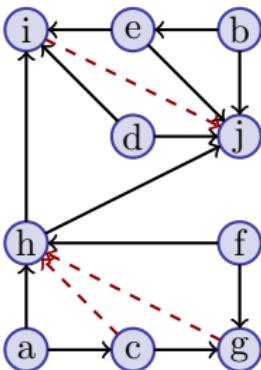
$$F = L + L^T$$

Elimination process

Exercise:

Apply the elimination graphs method to the matrix on the left.

$$A = \begin{pmatrix} a & b & \bullet & & \bullet & & \\ \bullet & c & \bullet & & & & \\ & d & & \bullet & & & \\ \bullet & e & & & \bullet & & \\ & f & & \bullet & \bullet & & \\ \bullet & g & & \bullet & & \bullet & \\ & h & & \bullet & \bullet & \bullet & \\ \bullet & i & & \bullet & & & \\ & j & & & & & \end{pmatrix}$$



$$F = \begin{pmatrix} a & b & \bullet & & \bullet & & \\ \bullet & c & \bullet & & & & \\ & d & & \bullet & \circ & & \\ \bullet & e & & & \bullet & \bullet & \\ & f & & \bullet & \bullet & & \\ \bullet & g & \circ & & \bullet & \bullet & \\ & h & \circ & \bullet & \bullet & \bullet & \\ \bullet & i & \circ & & \bullet & \circ & \\ & j & \circ & & & & \end{pmatrix}$$

You must find the filled graph in the middle which corresponds to the factors structure on the right

Consequences of elimination process

Proposition 5.1

If $i > j$ and $l_{ij} \neq 0$ then the elimination of variable j modifies column i . Therefore i cannot be eliminated before j and we say that i depends on j : $j \rightarrow i$

Proposition 5.2

Let $i > j > k$; if $k \rightarrow j$ (i.e., $l_{jk} \neq 0$) and $k \rightarrow i$ (i.e., $l_{ik} \neq 0$), necessarily $j \rightarrow i$ (i.e., $l_{ij} \neq 0$)

Proposition 5.3

If $j \rightarrow i$ (i.e., $l_{ij} \neq 0$, $i > j$) then

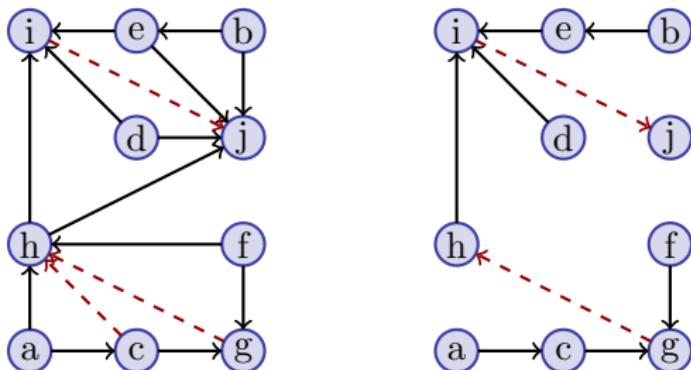
$$\text{struct}(L_{i:n,j}) \subseteq \text{struct}(L_{i:n,i})$$

simply because upon elimination of j , all of its neighbors become neighbors of i because i , itself, is a neighbor of j

Dependencies

How to represent dependencies in a compact way? Remove all redundant dependencies in $G(F)$. This is achieved through an operation called **transitive reduction**.

Assume $i \rightarrow j$ and $j \rightarrow k$; then $i \rightarrow k$, if it exists, is redundant because it is implicitly carried by the chain $i \rightarrow j \rightarrow k$.

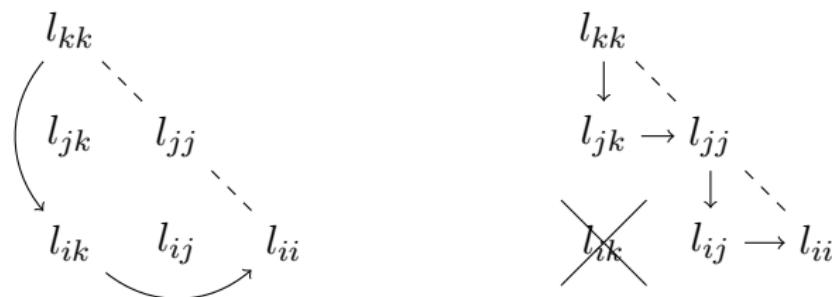


Dependencies

How can we easily identify redundant dependencies?

Remember Proposition 5.2. Let $i > j > k$; if $k \rightarrow j$ and $k \rightarrow i$ then $j \rightarrow i$.

Assume that $l_{ik} \neq 0$ and $l_{jk} \neq 0$ with $i > j > k$; because we know that l_{ij} must necessarily be nonzero, we can suppress l_{ik} because the dependency $k \rightarrow i$ is indirectly represented by the chain of dependencies $k \rightarrow j \rightarrow i$.



As a consequence of this observation, all the subdiagonal coefficients of L except the first express redundant dependencies.

The elimination tree

The graph obtained by removing from the filled graph the edges associated with the subdiagonal coefficients in each column except the first is **an ordered tree with root n** (where n is the matrix order).

Definition 5.1 (The Elimination Tree (Schreiber [21]))

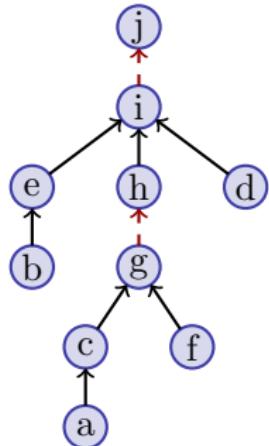
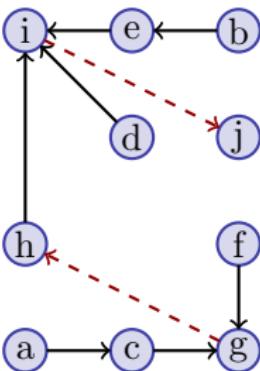
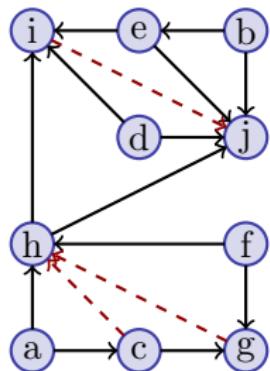
Let A be a sparse, symmetric positive definite matrix of order n with Cholesky factor L . The elimination tree is defined to be the structure with n nodes $\{1, 2, \dots, n\}$ such that the node p is the parent of node j if and only if

$$p = \min\{i > j \mid l_{ij} \neq 0\}.$$

The elimination tree

$$F = \begin{pmatrix} a & \bullet & & & \\ \bullet & b & c & d & e \\ & \bullet & \bullet & \bullet & \bullet \\ & & \bullet & f & g \\ & & & \bullet & h \\ & & & & \bullet \\ & & & & & i \\ & & & & & & j \end{pmatrix}$$

$$F_t = \begin{pmatrix} a & \bullet & & & \\ \bullet & b & c & d & e \\ & \bullet & \bullet & \bullet & \bullet \\ & & \bullet & f & g \\ & & & \bullet & h \\ & & & & \bullet \\ & & & & & i \\ & & & & & & j \end{pmatrix}$$



The elimination tree

Definition 5.2 (Topological order)

A topological order of a directed graph is an ordering of its vertices such that for each directed edge (i, j) , $j > i$.

A topological traversal of the elimination tree is such that a node is always visited after its child nodes (i.e., bottom-up).

The elimination tree

- expresses the order in which variables can be eliminated: the elimination of a variable only affects (directly or indirectly) its **ancestors** and only depends on its **descendants**
Therefore, any **topological order** of the elimination tree leads to a **correct result** and to the **same fill-in**
- expresses concurrence: because variables in separate subtrees do not affect each other, they can be eliminated in **parallel** (more on this later).

Outline

Sparsity, fill-in and dependencies

- Matrix factorizations

- Fill-in characterization

- Dependencies and the elimination tree

Sparse factorization methods

- Left-looking sparse Cholesky

- Right-looking sparse Cholesky

- The Multifrontal method

Sparse triangular solve

Fill-reducing orderings

- Local fill-reducing ordering methods

- Global fill-reducing ordering methods

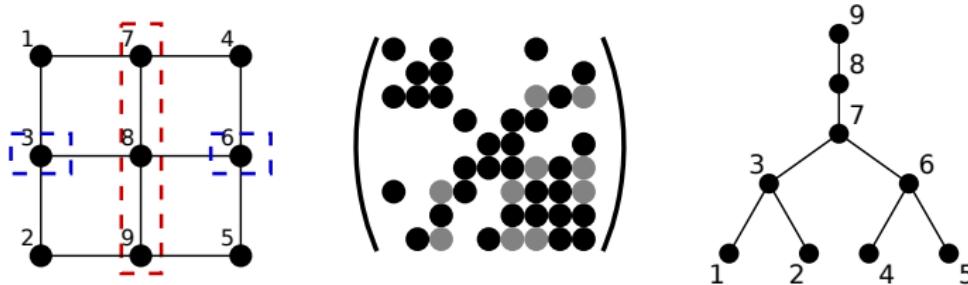
Tree amalgamation

Complexity of the factorization

Cholesky on a sparse matrix

The Cholesky factorization of a sparse matrix can be achieved with a **left-looking**, **right-looking** or **multifrontal** method.

Reference case: regular 3×3 grid ordered by nested dissection.
Nodes in the separators are ordered last (see the section on
orderings)

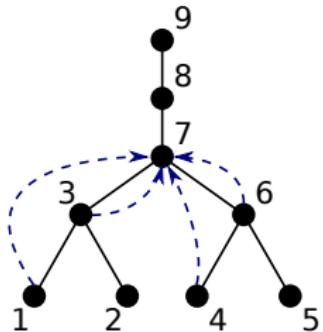
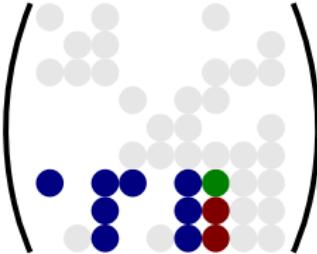


Notation:

- **cdiv(j):** $\sqrt{A(j,j)}$ and then $A(j+1:n,j)/\sqrt{A(j,j)}$
- **cmod(j,k):** $A(j:n,j) - A(j,k) * A(j:n,k)$
- **struct(L(1:k,j)):** the structure of $L(1:k,j)$ submatrix

Sparse left-looking Cholesky

```
left-looking
for j=1 to n do
    for k in struct(L(j,i:j-1)) do
        cmod(j,k)
    end for
    cdiv(j)
end for
```

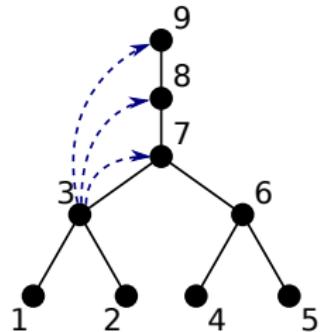
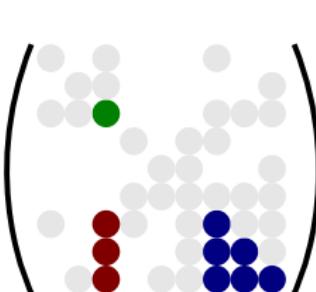


In the left-looking method, before variable j is eliminated, column j is updated with all the columns that have a nonzero on line j . In the example above, $\text{struct}(L(7,1:6)) = \{1, 3, 4, 6\}$.

- this corresponds to receiving updates from nodes **lower** in the subtree rooted at j
- the filled graph is necessary to determine the structure of each row

Sparse right-looking Cholesky

```
right-looking
for k=1 to n do
    cdiv(k)
    for j in struct(L(k+1:n,k)) do
        cmod(j,k)
    end for
end for
```



In the right-looking method, after variable k is eliminated, column k is used to update all the columns corresponding to nonzeros in column k . In the example above, $\text{struct}(L(4:9,3))=\{7,8,9\}$.

- this corresponds to sending updates to nodes **higher** in the elimination tree
- the filled graph is necessary to determine the structure of each column

The Multifrontal method Duff et al. [9]

Take as an example a simple 3×3 sparse matrix where no fill-in is generated:

$$\begin{bmatrix} a_{11} & & a_{13} \\ & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Its factorization can be achieved in three simple steps. The right-looking approach results in:

$$\text{Step 1 } \left\{ \begin{array}{l} l_{11} = \sqrt{a_{11}} \\ l_{31} = a_{31}/l_{11} \\ a'_{33} = a_{33} - l_{31}l_{31} \end{array} \right. \quad \text{Step 2 } \left\{ \begin{array}{l} l_{22} = \sqrt{a_{22}} \\ l_{32} = a_{32}/l_{22} \\ a''_{33} = a'_{33} - l_{32}l_{32} \end{array} \right. \quad \text{Step 3 } \left\{ \begin{array}{l} l_{33} = \sqrt{a''_{33}} \end{array} \right.$$

These computations are inefficient:

- heavy use of **indirect addressing**
- no vectorization nor **cache reuse**

The Multifrontal Method

Thanks to associative property of the addition operation, the three steps before can be rewritten as:

Step 1

$$\begin{bmatrix} a_{11} & a_{13} \\ & a_{22} \quad a_{23} \\ a_{31} & a_{32} \quad a_{33} \end{bmatrix} \rightarrow \begin{bmatrix} a_{11} & a_{13} \\ a_{31} & 0 \end{bmatrix} \rightarrow \begin{bmatrix} l_{11} \\ l_{31} \end{bmatrix}, \quad b = l_{31}l_{31}$$

Step 2

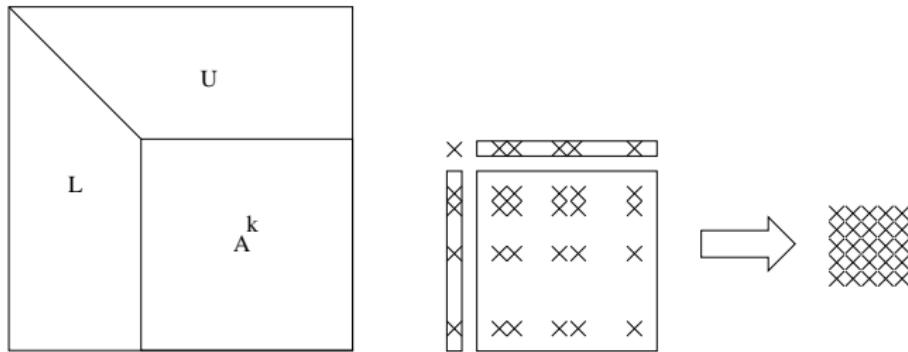
$$\begin{bmatrix} a_{11} & a_{13} \\ & a_{22} \quad a_{23} \\ a_{31} & a_{32} \quad a_{33} \end{bmatrix} \rightarrow \begin{bmatrix} a_{22} & a_{23} \\ a_{32} & 0 \end{bmatrix} \rightarrow \begin{bmatrix} l_{22} \\ l_{32} \end{bmatrix}, \quad c = l_{32}l_{32}$$

Step 3

$$l_{33} = \sqrt{a_{33} + b + c}$$

The Multifrontal Method

REMEMBER: each time a pivot is eliminated, a **clique** is formed in the graph. A clique is a set of nodes fully connected, i.e., a graph associated to a **dense** submatrix



The nonzero values concerned by an elimination step can be stored in a dense matrix and, thus, operations can be carried on by means of BLAS operation

The Multifrontal Method: the extend-add operation

Assume a matrix A with index set \mathcal{I}_A and a matrix B with index set \mathcal{I}_B , the **extend-add** operation \leftrightarrow produces a matrix C with index set $\mathcal{I}_C = \mathcal{I}_A \cup \mathcal{I}_B$ obtained by extending A and B with zeros to make them conform to \mathcal{I}_C and summing them:

$$A = \begin{matrix} & 1 & 3 & 6 \\ \begin{matrix} 1 \\ 3 \\ 6 \end{matrix} & \begin{bmatrix} a_{11} & a_{13} & a_{16} \\ a_{31} & a_{33} & a_{36} \\ a_{61} & a_{63} & a_{66} \end{bmatrix} \end{matrix} \quad B = \begin{matrix} & 2 & 3 & 5 \\ \begin{matrix} 2 \\ 3 \\ 5 \end{matrix} & \begin{bmatrix} b_{22} & b_{23} & b_{25} \\ b_{32} & b_{33} & b_{35} \\ b_{52} & b_{53} & b_{55} \end{bmatrix} \end{matrix}$$

$$C = A \leftrightarrow B = \begin{matrix} & 1 & 2 & 3 & 5 & 6 \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 5 \\ 6 \end{matrix} & \begin{bmatrix} a_{11} & 0 & a_{13} & 0 & a_{16} \\ 0 & 0 & 0 & 0 & 0 \\ a_{31} & 0 & a_{33} & 0 & a_{36} \\ 0 & 0 & 0 & 0 & 0 \\ a_{61} & 0 & a_{63} & 0 & a_{66} \end{bmatrix} \end{matrix} + \begin{matrix} & 1 & 2 & 3 & 5 & 6 \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 5 \\ 6 \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & b_{22} & b_{23} & b_{25} & 0 \\ 0 & b_{32} & b_{33} & b_{35} & 0 \\ 0 & b_{52} & b_{53} & b_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{matrix}$$

The Multifrontal Method

The multifrontal method consists in a topological order traversal of tree and at each node i two operations are performed:

1 Frontal Matrix Assembly:

$$F_i = \begin{bmatrix} f_{11} & f_{12} & \cdots & f_{1m} \\ f_{21} & f_{22} & & f_{2m} \\ \vdots & & \ddots & \vdots \\ f_{m1} & f_{m2} & \cdots & f_{mm} \end{bmatrix} = A(i : n, i) \Updownarrow CB_1 \Updownarrow \cdots \Updownarrow CB_j$$

where F_i is the **frontal matrix** at node i of index set $\mathcal{I}_{F_i} = \{\text{struct}(L(i : n, i))\}$, j is the number of children of i in the elimination tree and CB_1, \dots, CB_j are the **Contribution Blocks (Schur complements)** produced when processing the children (see next slide).

The Multifrontal Method

2 Frontal matrix factorization

$$F_i = \begin{bmatrix} f_{11} & f_{12} & \cdots & f_{1m} \\ f_{21} & f_{22} & & f_{2m} \\ \vdots & \ddots & & \vdots \\ f_{m1} & f_{m2} & \cdots & f_{mm} \end{bmatrix} \rightarrow \begin{bmatrix} l_{11} & & & \\ l_{21} & cb_{22} & \cdots & cb_{2m} \\ \vdots & & \ddots & \vdots \\ l_{m1} & cb_{m2} & \cdots & cb_{mm} \end{bmatrix}$$

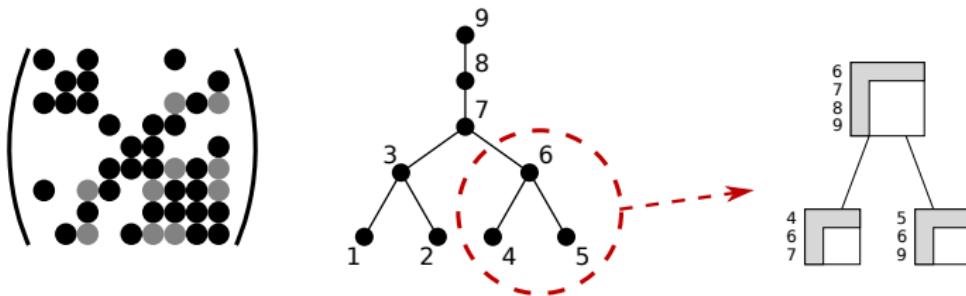
where

$$l_{11} = \sqrt{f_{11}}, \quad \begin{bmatrix} l_{21} \\ \vdots \\ l_{m1} \end{bmatrix} = \begin{bmatrix} f_{21} \\ \vdots \\ f_{m1} \end{bmatrix} / l_{11}$$

$$CB_i = \begin{bmatrix} cb_{22} & & cb_{2m} \\ \ddots & \vdots & \vdots \\ cb_{m2} & \cdots & cb_{mm} \end{bmatrix} = \begin{bmatrix} f_{22} & & f_{2m} \\ \ddots & \vdots & \vdots \\ f_{m2} & \cdots & f_{mm} \end{bmatrix} - \begin{bmatrix} l_{21} \\ \vdots \\ l_{m1} \end{bmatrix} \cdot \begin{bmatrix} l_{21} \\ \vdots \\ l_{m1} \end{bmatrix}^T$$

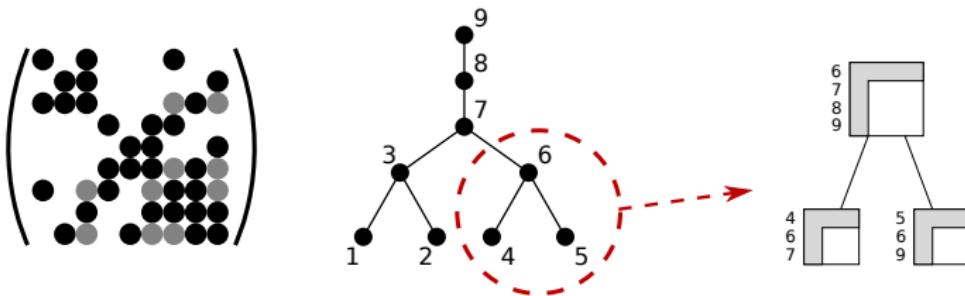
The Multifrontal Method

The **elimination tree** can be regarded as a graph of dependencies which defines where/how to assemble the elimination blocks and which variable to eliminate at each step.



The Multifrontal Method

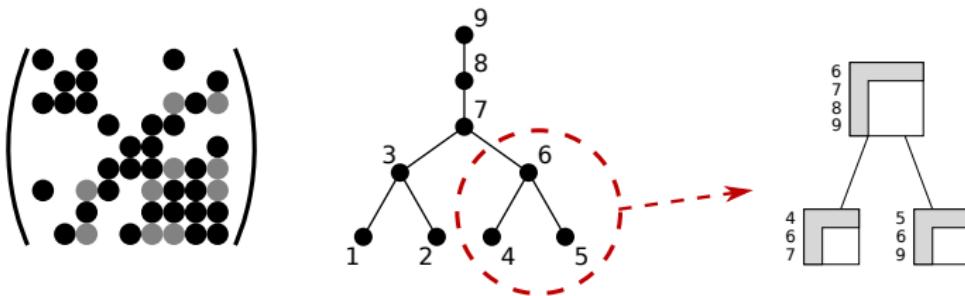
The **elimination tree** can be regarded as a graph of dependencies which defines where/how to assemble the elimination blocks and which variable to eliminate at each step.



$$\begin{bmatrix} a_{44} & a_{46} & a_{47} \\ a_{64} & 0 & 0 \\ a_{74} & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} l_{44} & & \\ l_{64} & b_{66} & b_{67} \\ l_{74} & b_{76} & b_{77} \end{bmatrix}$$

The Multifrontal Method

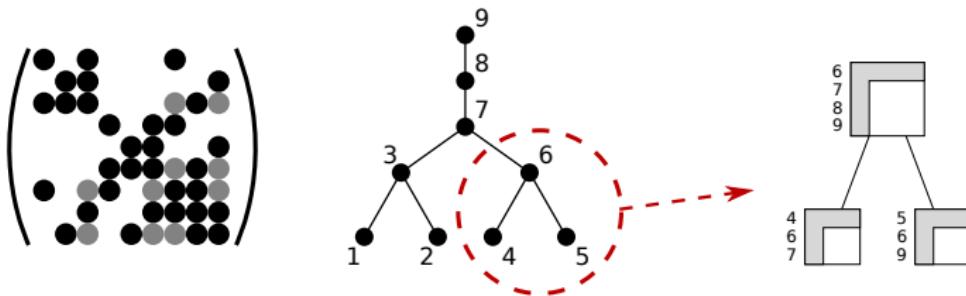
The **elimination tree** can be regarded as a graph of dependencies which defines where/how to assemble the elimination blocks and which variable to eliminate at each step.



$$\begin{bmatrix} a_{55} & a_{56} & a_{59} \\ a_{65} & 0 & 0 \\ a_{95} & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} l_{55} & & \\ l_{65} & c_{66} & c_{69} \\ l_{95} & c_{96} & c_{99} \end{bmatrix}$$

The Multifrontal Method

The **elimination tree** can be regarded as a graph of dependencies which defines where/how to assemble the elimination blocks and which variable to eliminate at each step.

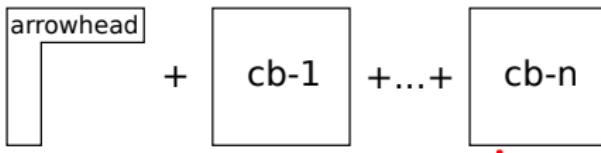


$$\begin{bmatrix} a_{66} & 0 & a_{68} & 0 \\ 0 & 0 & 0 & 0 \\ a_{86} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} b_{66} & b_{67} & 0 & 0 \\ b_{76} & b_{77} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} c_{66} & 0 & 0 & c_{69} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ c_{96} & 0 & 0 & c_{99} \end{bmatrix}$$
$$\rightarrow \begin{bmatrix} l_{66} \\ l_{76} & d_{77} & d_{78} & d_{79} \\ l_{86} & d_{87} & d_{88} & d_{89} \\ l_{86} & d_{97} & d_{98} & d_{99} \end{bmatrix}$$

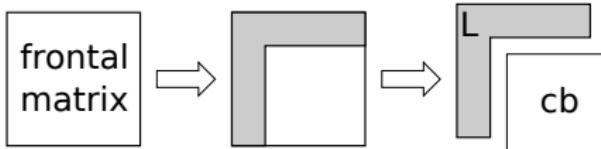
The Multifrontal Method

A dense matrix, called **frontal matrix**, is associated with each node of the elimination tree. The **Multifrontal** method consists in a bottom-up traversal of the tree where at each node two operations are done:

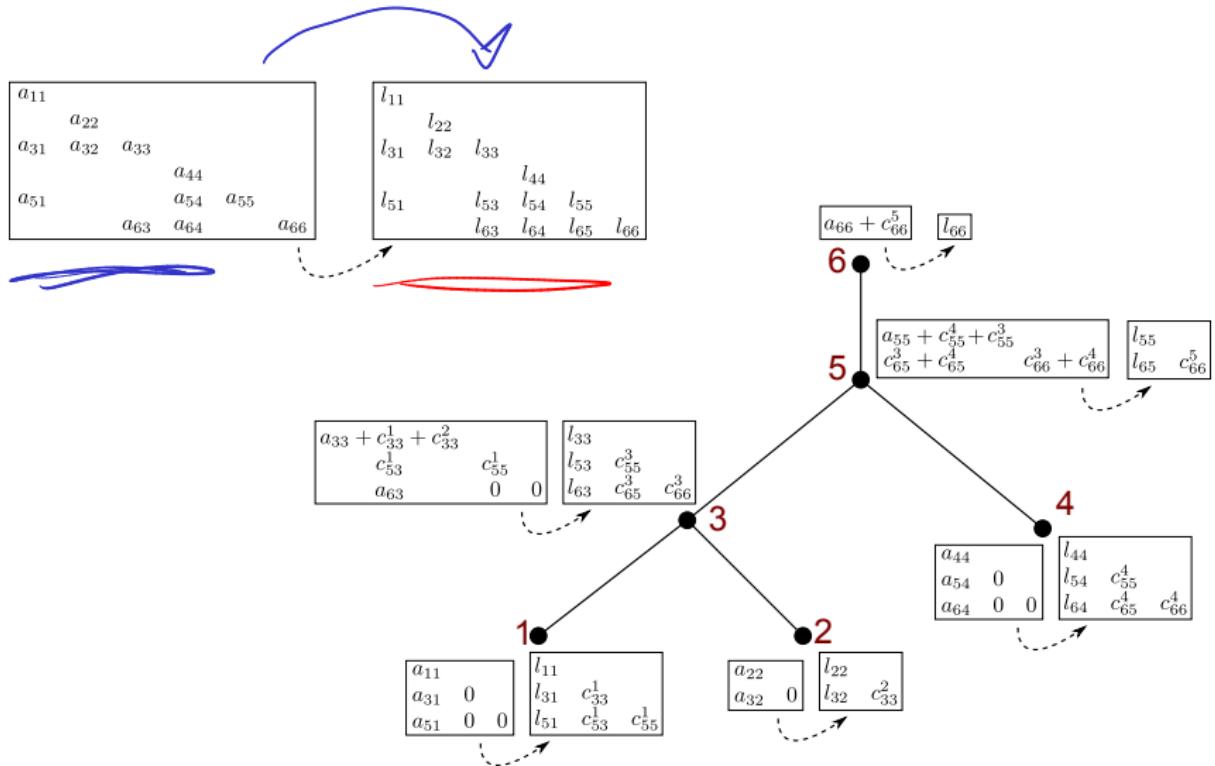
- **Assembly** Nonzeros from the original matrix are assembled together with the contribution blocks from children nodes into the **frontal matrix**



- **Elimination** A partial factorization of the frontal matrix is done. The variable associated to the node of the frontal tree (called **fully assembled**) can be eliminated. This step produces part of the final factors and a Schur complement (contribution block) that will be assembled into the father



The Multifrontal Method: example



$$\begin{array}{c}
 3 \\
 3 \left[\begin{array}{ccc} a_{33} + c'_{33} & c'_{33} & c'_{33} \\ c'_{33} & c_{33} & c'_{33} \\ a_{63} & 0 & 0 \end{array} \right] \rightarrow \begin{array}{l} \ell_{33} = \sqrt{a_{33} + c'_{33} + c'_{33}} \\ \ell_{33} = c'_{33} / \ell_{33} \\ \ell_{63} = a_{63} / \ell_{33} \\ c'_{33} = c_{33} - \ell_{33} \cdot \ell_{33} \end{array} \\
 5 \\
 6
 \end{array}$$

$$\begin{array}{c}
 2 \left[\begin{array}{cc} a_{11} & 3 \\ a_{31} & G \end{array} \right] \rightarrow \left[\begin{array}{cc} \ell_{11} & c'_{33} \\ \ell_{31} & c_{33}^2 \end{array} \right] \left[\begin{array}{cccc} \ell_{33} & c'_{33} & c'_{33} & c'_{33} \\ \ell_{63} & c'_{63} & c'_{63} & c'_{63} \end{array} \right]
 \end{array}$$

$$\begin{array}{c}
 1 \quad 3 \quad 5 \\
 1 \left[\begin{array}{cc} a_{11} & \\ a_{31} & 0 \\ a_{61} & 0 \end{array} \right] \rightarrow \left[\begin{array}{cc} \ell_{11} & \\ \ell_{31} & c'_{33} \\ \ell_{61} & c'_{63} \end{array} \right] \left[\begin{array}{c} c'_{33} \\ c'_{63} \\ c_{33} \end{array} \right]
 \end{array}$$

Outline

Sparsity, fill-in and dependencies

Matrix factorizations $t = L^{-1} b$

Fill-in characterization

Dependencies and the elimination tree

$$\begin{pmatrix} \ell_{11} & & \\ \ell_{12} & \ell_{22} & \\ \ell_{13} & \ell_{23} & \ell_{33} \end{pmatrix} \cdot \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

Sparse factorization methods

Left-looking sparse Cholesky

Right-looking sparse Cholesky

The Multifrontal method

Sparse triangular solve

Fill-reducing orderings

Local fill-reducing ordering methods

Global fill-reducing ordering methods

Tree amalgamation

Complexity of the factorization

$$\ell_{11} z_1 = b_1 \rightarrow z_1 = b_1 / \ell_{11}$$

$$\ell_{11} z_1 + \ell_{12} z_2 = b_2$$

$$z_2 = \frac{b_2 - \ell_{11} z_1}{\ell_{22}}$$

$$z_3 = \frac{b_3 - \ell_{31} z_1 - \ell_{32} z_2}{\ell_{33}}$$

$$z_{ik} = \frac{b_{ik} - \sum_{j=1}^{k-1} \ell_{kj} z_j}{\ell_{kk}}$$

Solve

Once the matrix is factorized, the problem can be solved against one or more right-hand sides:

$$AX = LL^T X = B, \quad A, L \in \mathbb{R}^{n \times n}, \quad X \in \mathbb{R}^{n \times k}, \quad B \in \mathbb{R}^{n \times k}$$

The solution of this problem can be achieved in two steps:

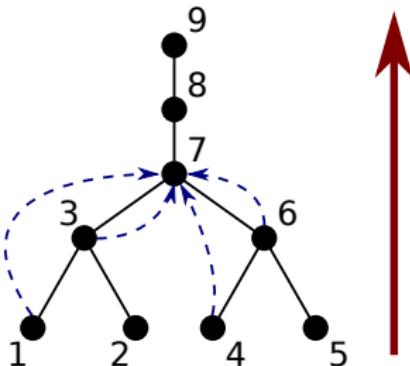
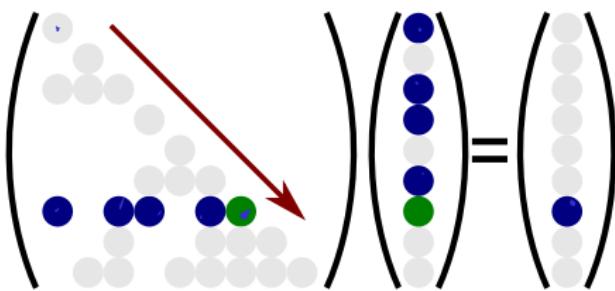
forward elimination $LZ = B$

backward substitution $L^T X = Z$

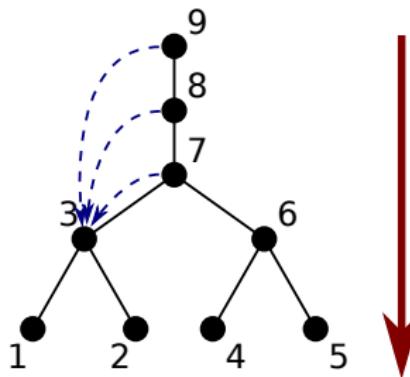
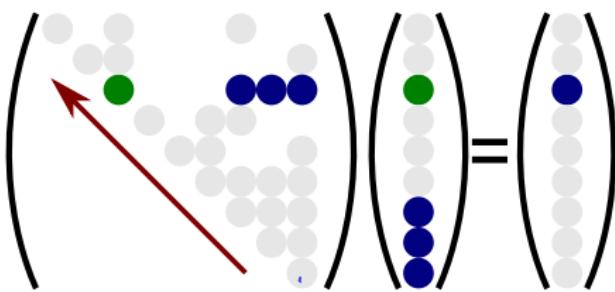
Solve: left-looking

ELIMINATION

Forward Substitution

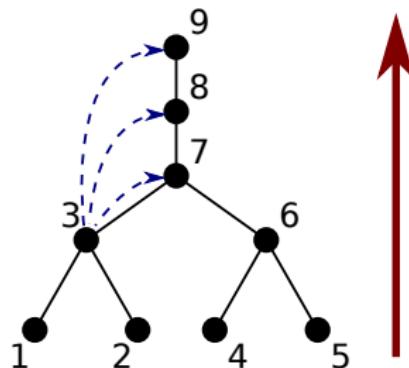
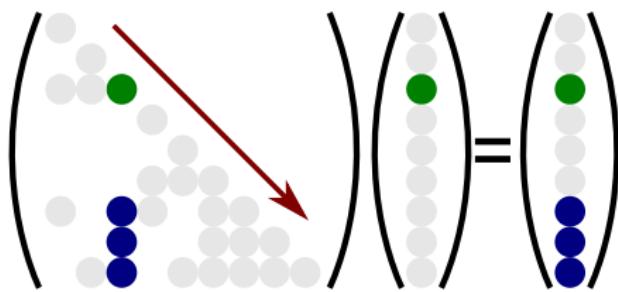


Backward Substitution

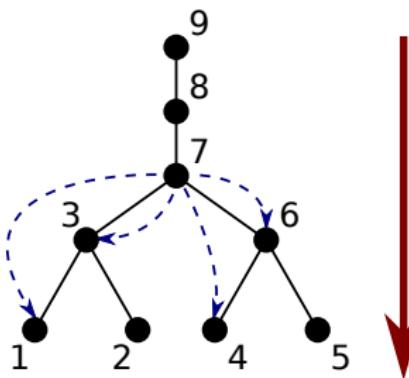
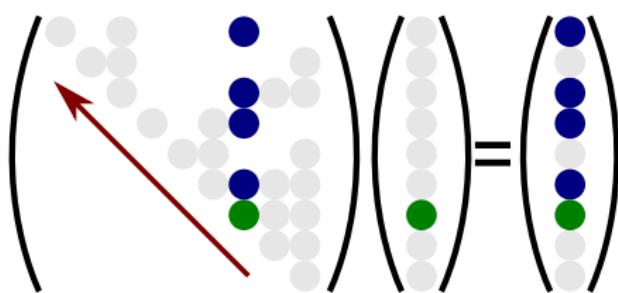


Solve: right-looking

Forward Substitution



Backward Substitution



Outline

Sparsity, fill-in and dependencies

- Matrix factorizations

- Fill-in characterization

- Dependencies and the elimination tree

Sparse factorization methods

- Left-looking sparse Cholesky

- Right-looking sparse Cholesky

- The Multifrontal method

Sparse triangular solve

Fill-reducing orderings

- Local fill-reducing ordering methods

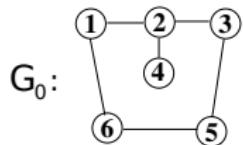
- Global fill-reducing ordering methods

Tree amalgamation

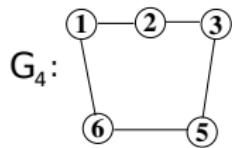
Complexity of the factorization

A sequence of elimination graphs

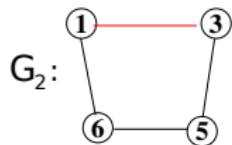
The nodes don't have to be necessarily eliminated in the natural order:



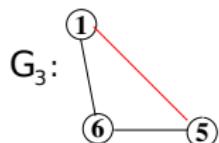
$$A_0 = \begin{bmatrix} 4 & \bullet & & & \\ \bullet & 2 & \bullet & \bullet & \\ \bullet & \bullet & 3 & & \\ \bullet & & 1 & & \\ \bullet & & & 5 & \bullet \\ \bullet & & & \bullet & 6 \end{bmatrix}$$



$$A_4 = \begin{bmatrix} 2 & \bullet & \bullet & & \\ \bullet & 3 & & \bullet & \\ \bullet & \bullet & 1 & & \\ \bullet & & & 5 & \bullet \\ \bullet & & & \bullet & 6 \end{bmatrix}$$



$$A_2 = \begin{bmatrix} 3 & \bullet & \bullet & & \\ \bullet & 1 & & \bullet & \\ \bullet & \bullet & 5 & \bullet & \\ \bullet & & \bullet & 6 \end{bmatrix}$$



$$A_3 = \begin{bmatrix} 1 & \bullet & \bullet & \\ \bullet & 5 & \bullet & \\ \bullet & \bullet & 6 \end{bmatrix}$$

$$\begin{bmatrix} l_{44} & & & & & \\ l_{24} & l_{22} & & & & \\ l_{32} & l_{33} & & & & \\ l_{12} & l_{53} & l_{11} & & & \\ & l_{13} & l_{51} & l_{55} & & \\ & l_{61} & l_{65} & l_{66} & & \end{bmatrix}$$

Fill-reducing ordering methods

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

Local approaches : 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)

Global approaches : The matrix is permuted so as to confine the fill-in within certain parts of the permuted matrix

- Cuthill-McKee, Reverse Cuthill-McKee
- **Nested dissection**

Hybrid approaches : First permute the matrix globally to confine the fill-in, then reorder small parts using local heuristics.

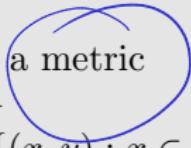
Local heuristics to reduce fill-in during factorization

Let $G(V, E)$ 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 better than v_j

Local reordering method

- 1: $G_U(V, E) \leftarrow$ undirected graph of A
- 2: **for** $i = 1 : n - 1$ **do**
- 3: let k be a vertex that minimizes a metric
- 4: $V \leftarrow V - \{k\}$ {remove vertex k }
- 5: $E \leftarrow E - \{(k, \ell) : \ell \in \text{adj}(k)\} \cup \{(x, y) : x \in \text{adj}(k) \text{ and } y \in \text{adj}(k)\}$
- 6: **update** *Metric*(v_j) for all non-selected nodes v_j
- 7: **end for**

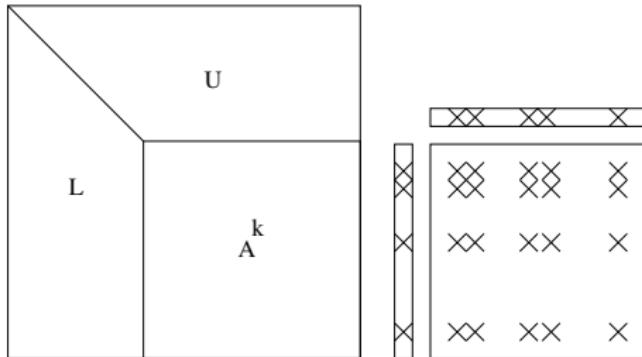


Step 6 should only be applied to nodes for which the Metric value might have changed.

Reordering unsymmetric matrices: Markowitz criterion

At step k of Gaussian elimination:

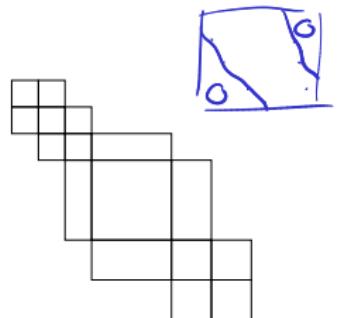
- $r_i^{(k)}$ = number of non-zeros in row i of $A^{(k)}$
- $c_j^{(k)}$ = number of non-zeros in column j of $A^{(k)}$



- **Markowitz criterion:** Candidate pivot a_{ij} should minimize $(r_i^{(k)} - 1) \times (c_j^{(k)} - 1)$ $\forall i, j \geq k$
- **Minimum degree**, i.e., Markowitz criterion for symmetric matrices: Candidate pivot a_{ij} should minimize $(c_j^{(k)} - 1)$ $\forall j \geq k$

CM and RCM: Definitions

- **Bandwidth:** A structurally symmetric matrix A is said to have *bandwidth* $2m + 1$, if m is the smallest integer such that $a_{ij} = 0$, whenever $|i - j| > m$. If no interchanges are performed during elimination, fill-in occurs only within the band.
- **Profile:** Define bandwidth for each row i : $m(i)$ is the smallest integer such that $a_{ij} = 0$, whenever $i - j > m(i)$ for $j < i$. Then profile of a symmetric matrix is $\sum_i m(i)$. If no interchanges are performed during elimination, no fill-in occurs ahead of the first entry in each row.



- **Block tridiagonal form:** Nonzeros are on the diagonal blocks or in a block just above the diagonal or just below the diagonal.

CM: Algorithm

Level sets are built from the vertex of minimum degree. At any level, priority is given to a vertex with smaller number of neighbors.

Cuthill-McKee

pick a vertex v and order it as the first vertex

$$S \leftarrow \{v\}$$

while $S \neq V$ **do**

$S' \leftarrow$ all vertices in $V \setminus S$ which are adjacent to S

order vertices in S' in increasing order of degrees

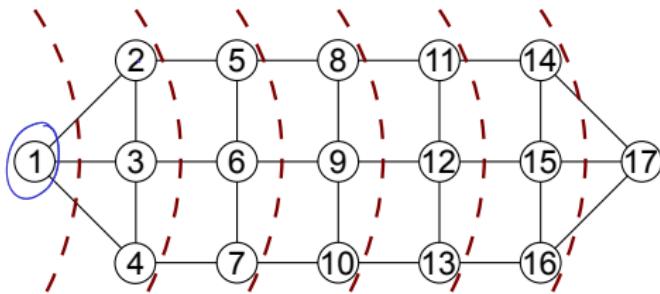
$$S \leftarrow S \cup S'$$

end while

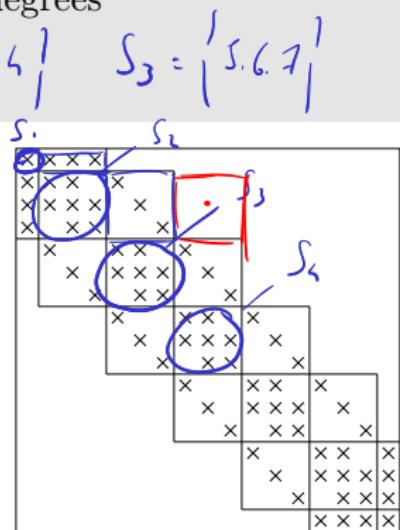
$$S_1 = \{1\}$$

$$S_2 = \{2, 3, 4\}$$

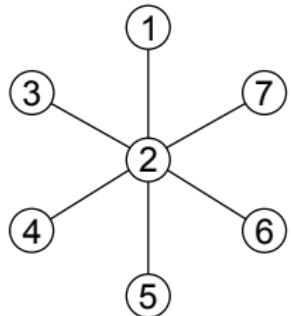
$$S_3 = \{5, 6, 7\}$$



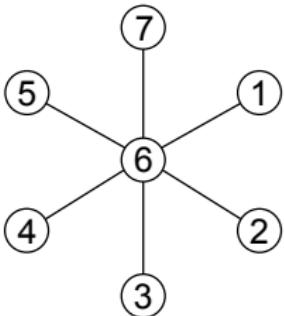
(example from Duff et al. [3])



CM vs RCM



CM Ordering



RCM ordering

$$A_c = \begin{bmatrix} X & X & & & & & \\ X & X & X & X & X & X & X \\ & X & X & & & & \\ X & & X & & & & \\ X & & & X & & & \\ X & & & & X & & \\ X & & & & & X & X \\ X & & & & & & X \end{bmatrix} \quad A_r = \begin{bmatrix} X & & & & & X & \\ & X & & & & X & \\ & & X & & & X & \\ & & & X & & X & \\ & & & & X & X & \\ X & X & X & X & X & X & X \\ X & & & & & X & X \end{bmatrix}$$

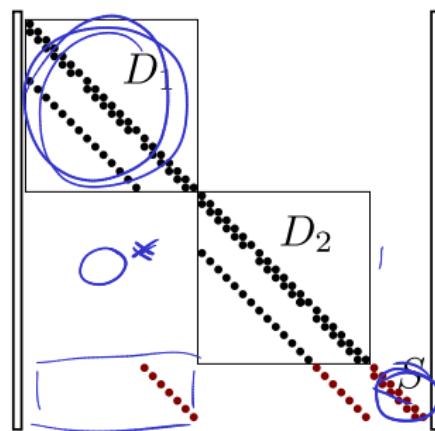
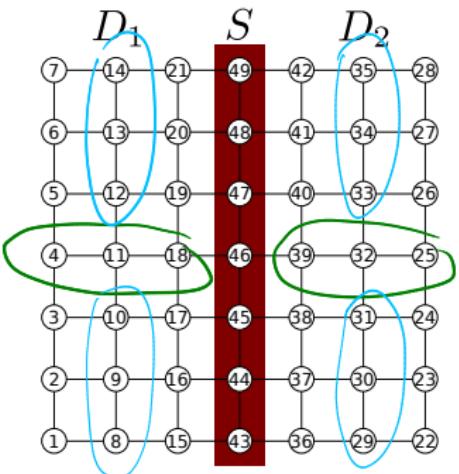
RCM: Simply reverse the order found by the CM algorithm.

It does not change the bandwidth but improves the storage requirements.

(Liu and Sherman [17])

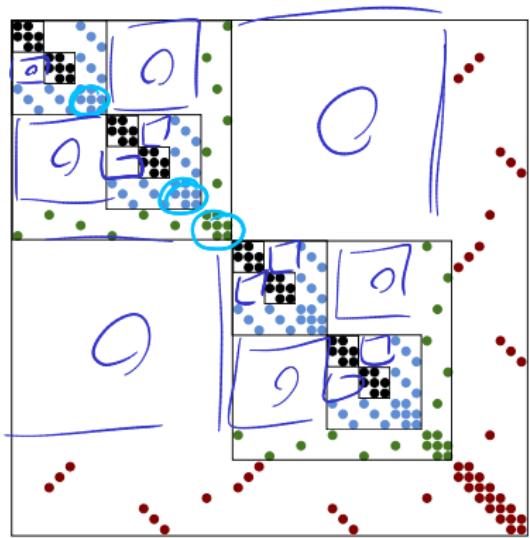
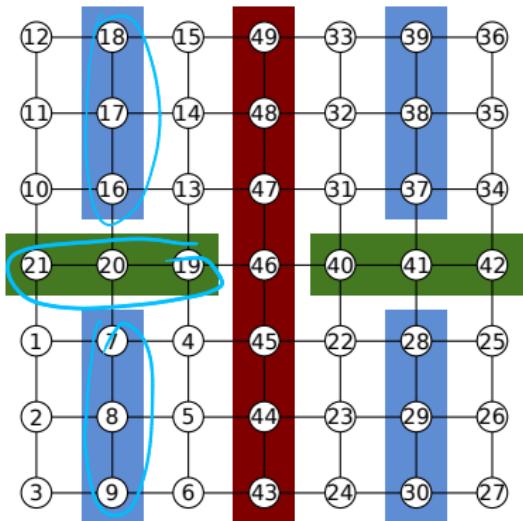
Nested dissection [George, 73]

Remember, in the elimination graphs, when a node is eliminated, new edges are added that connect all its neighbors to each other. Assume a separator S is identified that partitions the domain into two subdomains D_1 and D_2 . All the neighbors of nodes in D_1 (D_2) are either in D_1 (D_2) or in S . Thus, if D_1 is eliminated before D_2 and S , there cannot be any l_{ij} where $v_i \in D_1$ and $v_j \in D_2$



ND of a regular square mesh

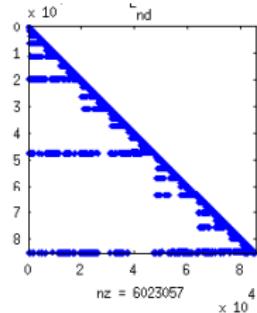
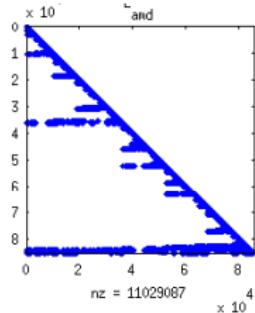
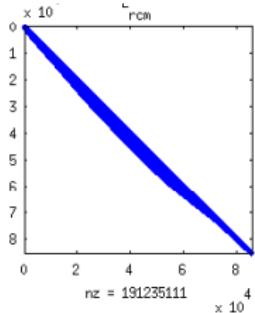
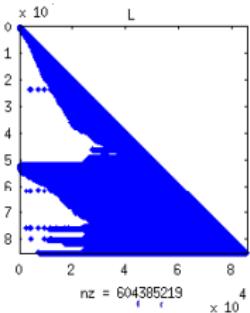
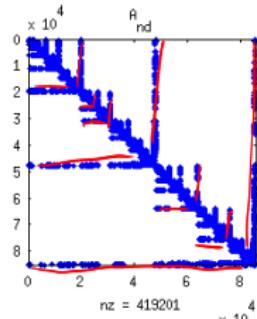
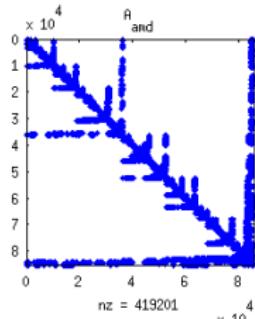
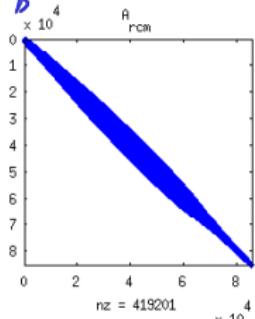
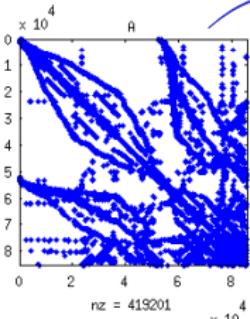
The nested dissection method aims at partitioning the domain so that the fill-in is only generated internally on each subdomain and on the interface by recursively computing bisectors.



The nested dissection method also produces an elimination tree (more details later).

Comparison of methods

matrix: Pothen/onera_dual



Comparison of methods

Matrix name	N	NZ	RCM	AMD	ND	
Boeing/bcsstk36	23052	1143140	14115	612	546	
DNVS/ship_003	121728	3777036	338141	177935	73192	
Wang/wang3	26064	177168	6769	4487	2343	
Ronis/xenon2	157464	3866688	604631	106831	48309	
Gupta/gupta2	62064	4248286	4084096	1561	1597	
Nasa/nasasrb	54870	2677324	6409	4639	3170	
UTEP/Dubcova2	65025	1030225	8085	563	310	
GHS_psdef/cvxbqp1	50000	349968	40231	2692	401	
GHS_indef/olesnik0	88263	744216	15122	1636	511	
Pothen/onera_dual	85567	419201	586066	9320	2180	
Boeing/bcsstk36	23052	1143140	16668641	2777434	2795674	MFlops
DNVS/ship_003	121728	3777036	185064888	85314461	57539811	
Wang/wang3	26064	177168	12781627	5333967	3956314	
Ronis/xenon2	157464	3866688	300447955	73256448	46439697	
Gupta/gupta2	62064	4248286	482498593	5780526	5759500	
Nasa/nasasrb	54870	2677324	17753691	12025490	10174167	
UTEP/Dubcova2	65025	1030225	22140164	3339239	2798209	
GHS_psdef/cvxbqp1	50000	349968	40049236	3969805	1937759	
GHS_indef/olesnik0	88263	744216	34853170	5533895	3422343	
Pothen/onera_dual	85567	419201	191235111	11029087	6023057	NNZ L

Outline

Sparsity, fill-in and dependencies

- Matrix factorizations

- Fill-in characterization

- Dependencies and the elimination tree

Sparse factorization methods

- Left-looking sparse Cholesky

- Right-looking sparse Cholesky

- The Multifrontal method

Sparse triangular solve

Fill-reducing orderings

- Local fill-reducing ordering methods

- Global fill-reducing ordering methods

Tree amalgamation

Complexity of the factorization

BLAS operations

High efficiency can be achieved if the computations of a sparse matrix can be rearranged as operations on dense matrices/blocks. This allows the use of efficient BLAS routines:

- **Level-1 BLAS**: vector-vector operations like inner product or vector sum. $\mathcal{O}(n)$ operations are performed on $\mathcal{O}(n)$ data. Vectorizable but limited by bus speed
- **Level-2 BLAS**: matrix-vector operations like matrix-vector product. $\mathcal{O}(n^2)$ operations are performed on $\mathcal{O}(n^2)$ data. Vectorizable but limited by bus speed
- **Level-3 BLAS**: matrix-matrix operations like matrix-matrix product or rank-k update. $\mathcal{O}(n^3)$ operations are performed on $\mathcal{O}(n^2)$ data. Vectorizable and very efficient thanks to good exploitation of memory hierarchy

A_{11}	
A_{21}	A_{22}

do $i = 1, n/3$

$$\rightarrow L_{kk} = \text{chol}(A_{kk})$$

$$\text{BLAS-3 } L_{ik} = A_{ik} L_k^T$$

$$\text{BLAS-3 } A_{ii} = A_{ii} - L_{ik} \cdot L_{ik}^T$$

end do

$$b \\ A = \begin{pmatrix} A_{11} & A_{21}^T \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & \\ L_{21} & I \end{pmatrix} * \begin{pmatrix} L_{11}^T & L_{21}^T \\ & \tilde{A}_{22} \end{pmatrix}$$

$$A_{11} = L_{11} L_{11}^T \rightarrow L_{11} = \text{chol}(A_{11})$$

$$A_{11} = L_{11} L_{11}^T + \tilde{A}_{11}$$

$$A_{21} = L_{21} L_{11}^T \rightarrow L_{21} = A_{21} L_{11}^{-T}$$

$$\tilde{A}_{22} = A_{22} - L_{21} L_{11}^{-T}$$

Tree Amalgamation

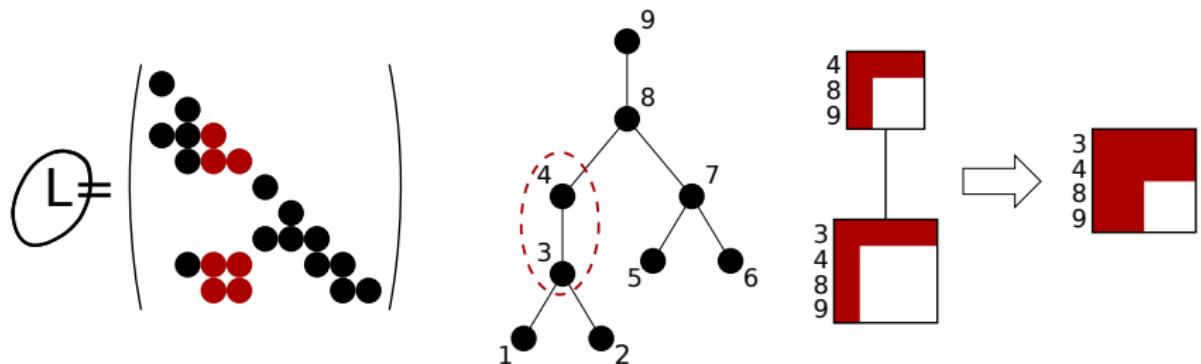
The whole factorization is recast into a sequence of partial dense factorizations of the type:

$$l_{11} = \sqrt{f_{11}}, \quad \begin{bmatrix} l_{21} \\ \vdots \\ l_{n1} \end{bmatrix} = \begin{bmatrix} f_{21} \\ \vdots \\ f_{n1} \end{bmatrix} / l_{11}$$

$$\begin{bmatrix} cb_{22} & cb_{2n} \\ \ddots & \vdots & \vdots \\ cb_{n2} & \cdots & cb_{nn} \end{bmatrix} = \begin{bmatrix} f_{22} & f_{2n} \\ \ddots & \vdots \\ f_{n2} & \cdots & f_{nn} \end{bmatrix} - \begin{bmatrix} l_{21} \\ \vdots \\ l_{n1} \end{bmatrix} \cdot \begin{bmatrix} l_{21} \\ \vdots \\ l_{n1} \end{bmatrix}^T$$

This is still only Level-2 BLAS operations. How to get the efficiency of Level-3 BLAS?

Tree Amalgamation



Amalgamation without fill-in consists in merging all the frontal matrices related to pivots whose columns in the factor L have the same structure. The subset of nodes containing this pivots is called a **supernode**. All the pivots in a supernode can thus be eliminated at once within the same frontal matrix

SANS ANALGAMATION FRONT 3

etat front 1		etat front 2	
1	ℓ_{11}	1	ℓ_{22}
3	$\ell_{31} \quad C_{33}$	1	$\ell_{32} \quad C_{33}^2$
		3	$\ell_{43} \quad C_{43}^2$
		4	$\ell_{44} \quad C_{43}^2 \quad C_{44}$
		2	$\ell_{83} \quad C_{83}^2 \quad C_{84}^2 \quad C_{88}$

Assemblage front 3

3	$a_{33} + C_{33}^2 + C_{33}^2$	-	-	-	-	-
4	$a_{43} + C_{43}^2$	C_{44}	-	-	-	-
8	$a_{83} + C_{83}^2$	C_{84}	C_{88}	-	-	-
9	a_{93}	0	0	0	-	-

Factor Front 3

$$\ell_{33} = (a_{33} + C_{33}^2 + C_{33}^2)$$

$$C_{44}^3 = C_{44}^2 - \ell_{43} \cdot \ell_{43}$$

$$\ell_{43} = (a_{43} + C_{43}^2) / \ell_{33}$$

$$C_{84}^3 = C_{84}^2 - \ell_{83} \cdot \ell_{83}$$

$$\ell_{83} = (a_{83} + C_{83}^2) / \ell_{33}$$

$$C_{88}^3 = C_{88}^2 - \ell_{83} \cdot \ell_{83}$$

$$\ell_{13} = a_{13} / \ell_{33}$$

$$C_{93}^3 = - \ell_{93} \cdot \ell_{83}$$

$$C_{99}^3 = - \ell_{93} \ell_{93}$$

3	ℓ_{33}	-	-	-
4	ℓ_{43}	C_{44}^3	-	-
8	ℓ_{83}	C_{84}^3	C_{88}^3	-
9	ℓ_{93}	C_{94}^3	C_{98}^3	C_{99}^3

SUNS ANALGATION FRONTS

Assembly front ζ

$$\begin{matrix} & a_{44} + C_{44}^3 \\ 4 & \boxed{a_{84} - C_{84}^3} & C_{88}^3 \\ 3 & a_{94} + C_{94}^3 & C_{98}^3 & C_{99}^3 \end{matrix}$$

Factor front ζ

$$l_{44} = \sqrt{a_{44} + C_{44}^3} = \sqrt{a_{44} + C_{44}^3 - l_{43} l_{43}}$$

$$l_{84} = (a_{84} + C_{84}^3) / l_{44} = (a_{84} + C_{84}^3 - l_{83} l_{43}) / l_{44}$$

$$l_{94} = (a_{94} + C_{94}^3) / l_{44} = (a_{94} - l_{93} l_{43}) / l_{44}$$

$$C_{88}^3 = C_{88}^3 - l_{84} \cdot l_{84} = \boxed{C_{88}^3 - l_{83} l_{83} - l_{84} l_{84}}$$

$$C_{98}^3 = C_{98}^3 - l_{94} \cdot l_{84} = \boxed{-l_{93} l_{83} - l_{94} l_{84}}$$

$$C_{99}^3 = C_{99}^3 - l_{94} \cdot l_{94} = \boxed{-l_{93} l_{93} - l_{94} l_{94}}$$

AVEC ANALGATION FRONT 3+5

Assemblage front amalgamé

3	$a_{33} + c_{33}^1 + c_{33}^2$						
4	$a_{43} + c_{43}$		$a_{44} + c_{44}^2$				
8	$a_{83} + c_{83}^2$		$a_{84} + c_{84}^2$	c_{88}			
9	a_{93}	a_{94}		0		0	

Facto front amalgamé

$$l_{33} = \sqrt{a_{33} + c_{33}^1 + c_{33}^2}$$

$$\tilde{c}_{43} = (a_{44} + c_{44}^2) - l_{43} \cdot l_{43}$$

$$l_{43} = (a_{43} + c_{43}^2) / l_{33}$$

$$\tilde{c}_{24} = (a_{24} + c_{24}^2) - l_{23} \cdot l_{23}$$

$$l_{23} = (a_{23} + c_{23}^2) / l_{33}$$

$$\tilde{c}_{83} = \tilde{c}_{23} - l_{23} l_{23}$$

$$l_{93} = a_{93} / l_{33}$$

$$\tilde{c}_{98} = - l_{93} l_{83}$$

$$\tilde{c}_{99} = - l_{93} l_{93}$$

$$l_{44} = \sqrt{\tilde{c}_{44}} = \sqrt{a_{44} + c_{44}^2 - l_{43} \cdot l_{43}}$$

$$l_{84} = \tilde{c}_{84} / l_{44} = (a_{84} + c_{84}^2 - l_{83} l_{43}) / l_{44}$$

$$l_{94} = \tilde{c}_{94} / l_{44} = (a_{94} - l_{93} l_{43}) / l_{44}$$

$$\tilde{c}_{28} = \tilde{c}_{22} - l_{24} l_{24} =$$

$$= \tilde{c}_{22}^2 - l_{23} l_{23} - l_{24} l_{24}$$

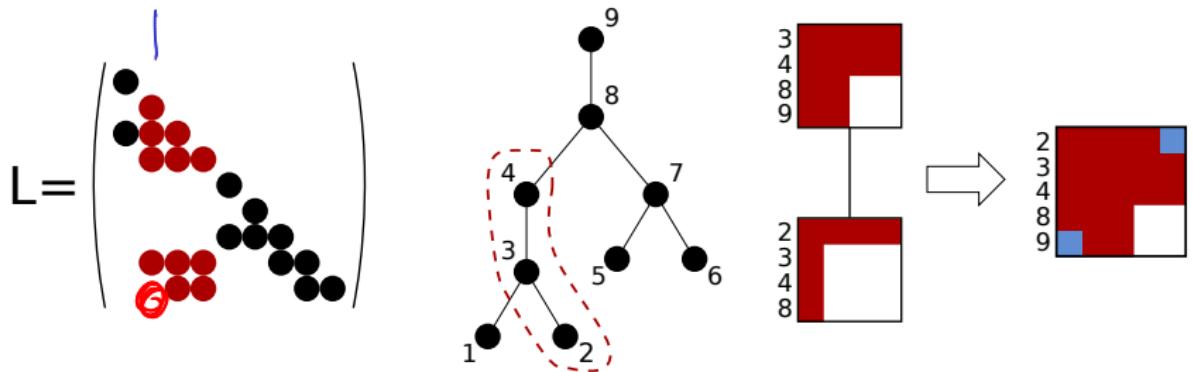
$$l_{93}^2 = \tilde{c}_{92} - l_{93} l_{83} - l_{93} l_{84}$$

$$l_{94}^2 = \tilde{c}_{99} - l_{93} l_{93} - l_{94} l_{94}$$

PINOT 1

PINOT 2

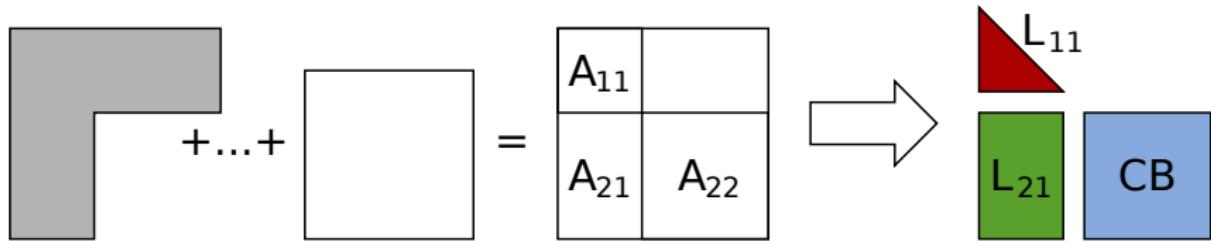
Tree Amalgamation



Amalgamation with fill-in is based on the same principle except that it groups together pivots whose column structure in L is not exactly the same. If the generated fill-in does not exceed a certain threshold, the extra cost is overcome by efficiency

Tree Amalgamation

After amalgamation:



$$L_{11}L_{11}^T = A_{11} \quad (\text{Cholesky factorization})$$

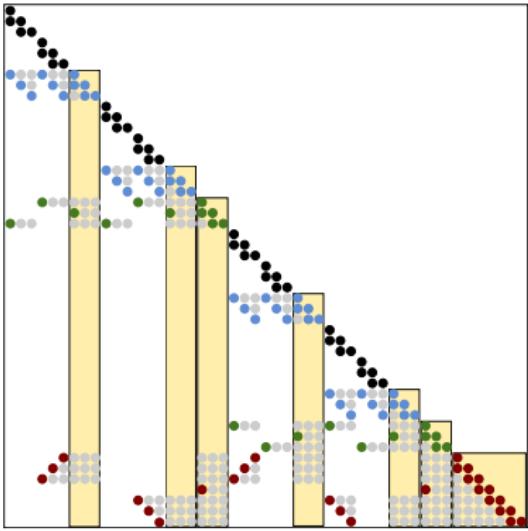
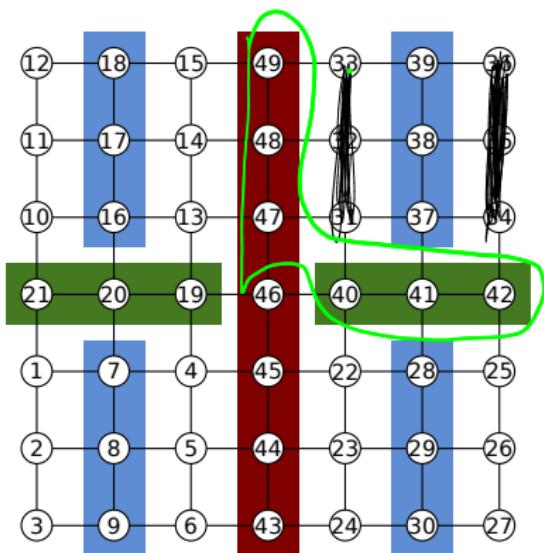
$$L_{21} = A_{21}L_{11}^{-T}$$

$$CB = A_{22} - L_{21}L_{21}^T$$

All the operation related to the frontal matrix can be done through Level-3 BLAS routines

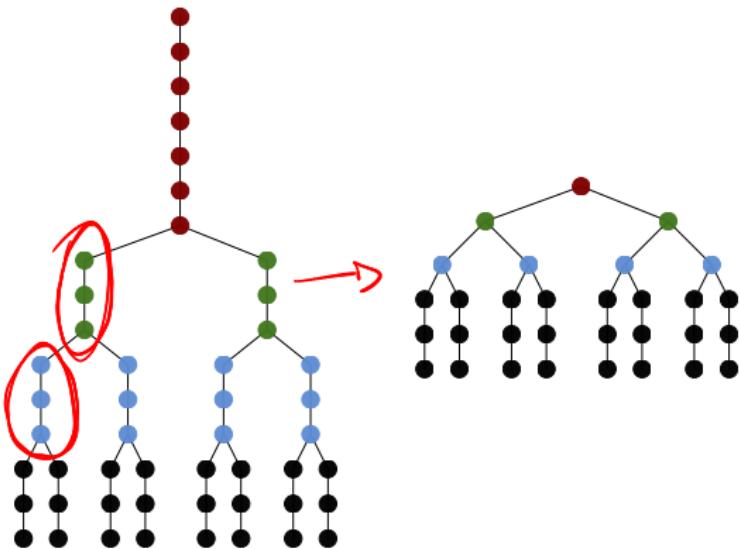
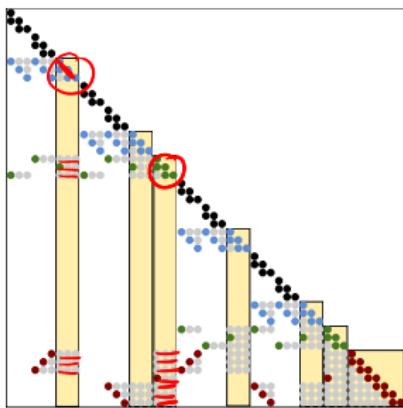
Nested dissection and elimination/assembly trees

Remember the case of a 7x7 regular grid sorted with the nested dissection method:



Nested dissection and elimination/assembly trees

All the L columns of variables within the same separator have the same structure and can, therefore, be amalgamated into a supernode.



For this reason the nested dissection method provides an **assembly tree** implicitly defined by the tree of separators.

Outline

Sparsity, fill-in and dependencies

- Matrix factorizations

- Fill-in characterization

- Dependencies and the elimination tree

Sparse factorization methods

- Left-looking sparse Cholesky

- Right-looking sparse Cholesky

- The Multifrontal method

Sparse triangular solve

Fill-reducing orderings

- Local fill-reducing ordering methods

- Global fill-reducing ordering methods

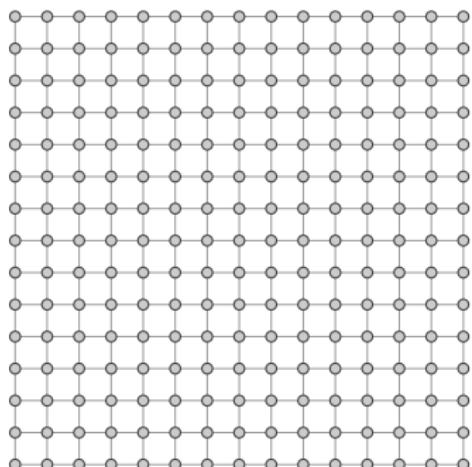
Tree amalgamation

Complexity of the factorization

Complexity of the factorization with ND

Definition 10.1 (2D ND assumptions (George [11]))

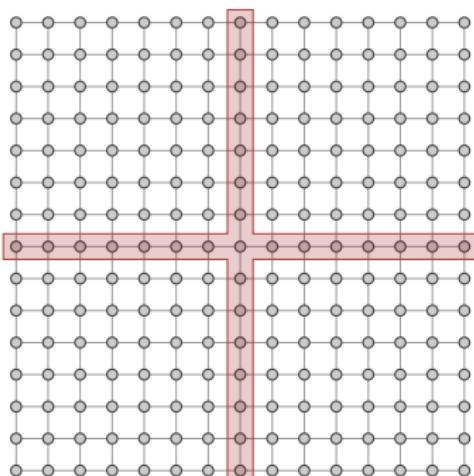
- 2D, square grid of size $N \times N$ and **cross-shaped separators**.
- The size of the separators/fronts is divided by 2 at every level starting at $2N$
- The number of nodes is multiplied by 4 at every level



Complexity of the factorization with ND

Definition 10.1 (2D ND assumptions (George [11]))

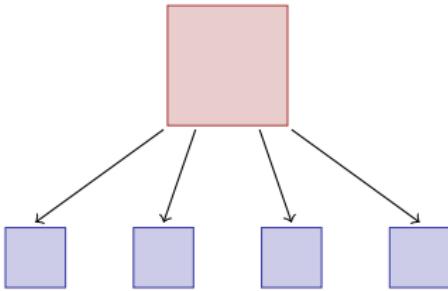
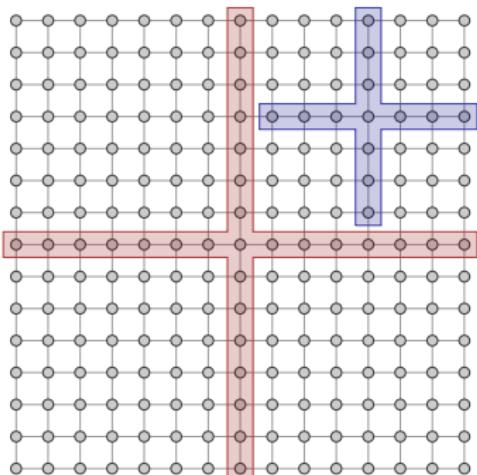
- 2D, square grid of size $N \times N$ and **cross-shaped separators**.
- The size of the separators/fronts is divided by 2 at every level starting at $2N$
- The number of nodes is multiplied by 4 at every level



Complexity of the factorization with ND

Definition 10.1 (2D ND assumptions (George [11]))

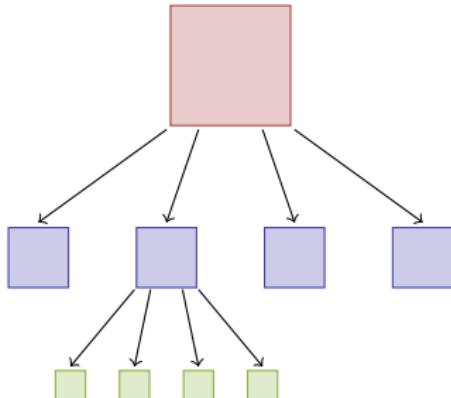
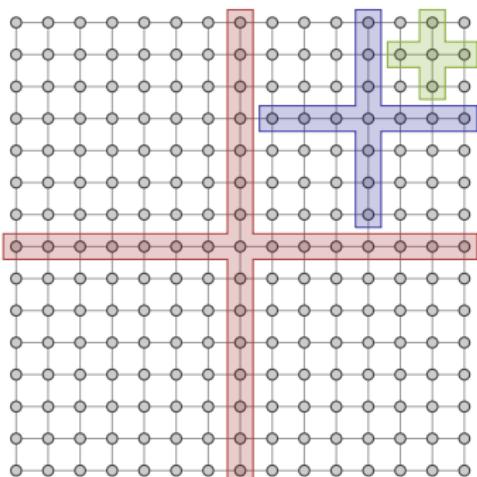
- 2D, square grid of size $N \times N$ and **cross-shaped separators**.
- The size of the separators/fronts is divided by 2 at every level starting at $2N$
- The number of nodes is multiplied by 4 at every level



Complexity of the factorization with ND

Definition 10.1 (2D ND assumptions (George [11]))

- 2D, square grid of size $N \times N$ and **cross-shaped separators**.
- The size of the separators/fronts is divided by 2 at every level starting at $2N$
- The number of nodes is multiplied by 4 at every level



Complexity of the factorization with ND

Flops

The factorization cost for a front of order m is $\mathcal{C}(m) = O(m^3)$

$$\mathcal{C}_{mf} = \sum_{l=0}^{\log_2 N} 4^l \mathcal{C}\left(\frac{2N}{2^l}\right) = O\left(\sum_{l=0}^{\log_2 N} 4^l \left(\frac{N}{2^l}\right)^3\right) = O(N^3)$$

Factors size

The size of factors at a front of order m is $\mathcal{F}(m) = O(m^2)$

$$\mathcal{F}_{mf} = \sum_{l=0}^{\log_2 N} 4^l \mathcal{F}\left(\frac{2N}{2^l}\right) = O\left(\sum_{l=0}^{\log_2 N} 4^l \left(\frac{N}{2^l}\right)^2\right) = O(N^2 \log_2 N)$$

Complexity of the factorization with ND

The same analysis can be done on a 3D cubic domain of order N .

Regular problems (nested dissection)	2D $N \times N$ grid	3D $N \times N \times N$ grid
Nonzeros in original matrix	$O(N^2)$	$O(N^3)$
Nonzeros in factors	$O(N^2 \log N)$	$O(N^4)$
Floating-point ops	$O(N^3)$	$O(N^6)$

The cost of the multifrontal factorization is dominated by the cost of the topmost front factorization.

In 3D this is also the case for the factors size.

Extrapolation $n = N^3 = 1000^3$ grid:

55 exaflops, 200 TBytes for factors, 40 TBytes of working memory!

Parallelism

Outline

General introduction

Shared memory parallelism

Tree parallelism

Node parallelism

Scheduling and memory consumption

Scheduling overhead

Distributed memory parallelism

Tree parallelism

Node parallelism

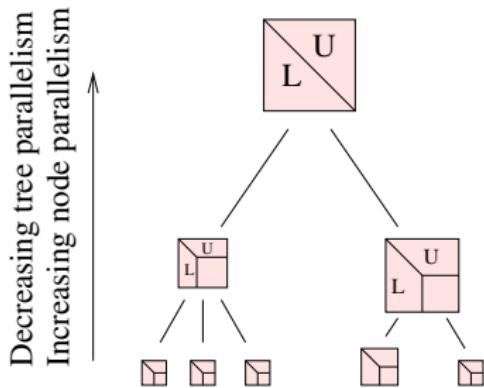
Proportional mapping

Bound on parallelism

Parallelization: two sources of parallelism

tree parallelism arising from sparsity, it is formalized by the fact that nodes in separate subtrees of the elimination tree can be eliminated at the same time

node parallelism within each node:
parallel dense factorization



Using both sources of parallelism is crucial because they are **complementary**:

- Tree parallelism decreases going up because the tree gets more and more narrow
- Node parallelism grows going up because nodes become bigger and bigger

Task mapping and scheduling

Task mapping and scheduling: objective

Organize work to achieve a goal like total execution time minimization, memory minimization or a mixture of the two:

- **Mapping:** where to execute a task?
- **Scheduling:** when and where to execute a task?

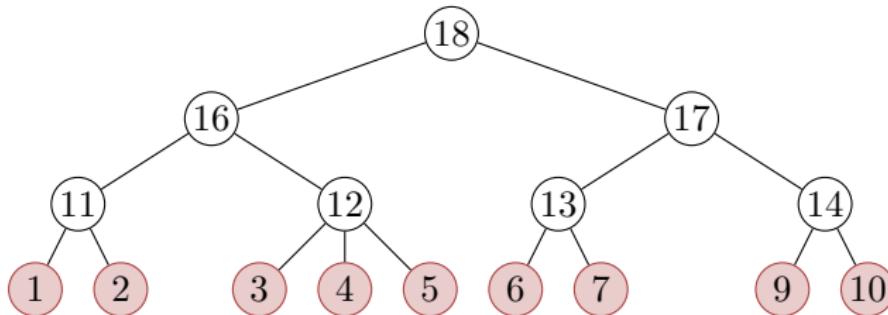
Several approaches are possible:

- **static:** Build the schedule before the execution and follow it at run-time
 - Advantage: very efficient since it has a global view of the system
 - Drawback: Requires a very-good performance model for the platform
- **dynamic:** Take scheduling decisions dynamically at run-time
 - Advantage: Reactive to the evolution of the platform and easy to use on several platforms
 - Drawback: Decisions taken with local criteria (a decision which seems to be good at time t can have very bad consequences at time $t + 1$)
- **hybrid:** Try to combine the advantages of static and dynamic

Task mapping and scheduling

The mapping/scheduling is commonly guided by a number of criteria:

- A mapping/scheduling which is good for **concurrency** is commonly not good in terms of **memory consumption**



- Especially in distributed-memory environments, **data transfers** have to be limited as much as possible
- **Load balance** (both in terms of memory and operations) has to be as high as possible in order to minimize the execution time or memory consumption

Outline

General introduction

Shared memory parallelism

Tree parallelism

Node parallelism

Scheduling and memory consumption

Scheduling overhead

Distributed memory parallelism

Tree parallelism

Node parallelism

Proportional mapping

Bound on parallelism

Shared memory parallelism

In a shared memory environment, data can be shared between processes without any communications (unless we consider cache or NUMA locality). Therefore **mapping is not necessary** because any process can process any node of the tree with the same efficiency.

A **dynamic scheduling** is commonly employed in order to achieve a good load balance because it reduces the processes idle time.

Care should be taken to prevent the memory consumption from becoming excessive.

Shared memory: tree parallelism in multifrontal

Tasks are pushed into a pool along with their mutual dependencies and scheduled dynamically.

```
do n = 1, num_nodes ! in topological order

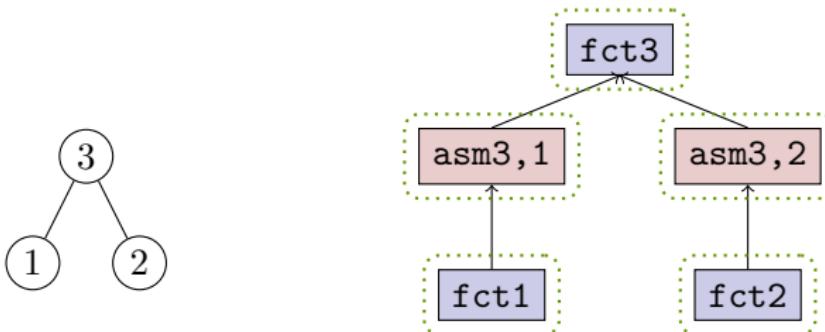
    forall c in children(n)
        !$omp task depend(in:front_c) depend(inout:front_n)
        call assemble(front_c, front_n)
    end do

    !$omp task depend(inout:front_n)
    call factorize(front_n)

end do
```

An assembly task can be executed only when the corresponding child front is factorized. A front can be factorized only when the contribution blocks of all its children have been assembled into it.

Shared memory: tree parallelism in multifrontal



- At iteration n of the loop, `front_n` is `inout` (read and written) to its assembly and factorization operations. This implies that the factorization of a front depends on the related assemblies and thus cannot be executed before them.
- At iteration n , an assembly operation takes in mode `in` (only read) the front `front_c` (a child of `front_n`) which was taken in mode `inout` by the factorization task at iteration c . Therefore, the assembly of `front_c` into `front_n` depends on the factorization of `front_c` and cannot be executed before it.

Node parallelism

The sequential algorithm:

for $k = 1, \dots, n/b$ **do**

$$A_{kk} \rightarrow L_{kk}L_{kk}^T$$

for $i = k + 1, \dots, n/b$ **do**

$$L_{ik} \leftarrow A_{ik}L_{kk}^{-T}$$

end for

$$A_{11} \quad * \quad * \quad *$$

for $j = k + 1, \dots, n/b$ **do**

$$A_{21} \quad A_{22} \quad * \quad *$$

for $i = k + 1, \dots, n/b$ **do**

$$A_{ij} \leftarrow A_{ij} - L_{ik}L_{jk}^T$$

$$A_{31} \quad A_{32} \quad A_{33} \quad *$$

end for

end for

$$A_{41} \quad A_{42} \quad A_{43} \quad A_{44}$$

end for

Node parallelism

The sequential algorithm:

for $k = 1, \dots, n/b$ **do**

$$A_{kk} \rightarrow L_{kk}L_{kk}^T$$

potrf

for $i = k + 1, \dots, n/b$ **do**

$$L_{ik} \leftarrow A_{ik}L_{kk}^{-T}$$

$$A_{11} \quad * \quad * \quad *$$

end for

for $j = k + 1, \dots, n/b$ **do**

$$A_{21} \quad A_{22} \quad * \quad *$$

for $i = k + 1, \dots, n/b$ **do**

$$A_{ij} \leftarrow A_{ij} - L_{ik}L_{jk}^T$$

$$A_{31} \quad A_{32} \quad A_{33} \quad *$$

end for

end for

$$A_{41} \quad A_{42} \quad A_{43} \quad A_{44}$$

end for

Node parallelism

The sequential algorithm:

for $k = 1, \dots, n/b$ **do**

$$A_{kk} \rightarrow L_{kk}L_{kk}^T$$

trsm

for $i = k + 1, \dots, n/b$ **do**

$$L_{ik} \leftarrow A_{ik}L_{kk}^{-T}$$

end for

for $j = k + 1, \dots, n/b$ **do**

for $i = k + 1, \dots, n/b$ **do**

$$A_{ij} \leftarrow A_{ij} - L_{ik}L_{jk}^T$$

$$\begin{pmatrix} A_{11} & * & * & * \\ A_{21} & A_{22} & * & * \\ A_{31} & A_{32} & A_{33} & * \\ A_{41} & A_{42} & A_{43} & A_{44} \end{pmatrix}$$

end for

end for

end for

Node parallelism

The sequential algorithm:

for $k = 1, \dots, n/b$ **do**

$A_{kk} \rightarrow L_{kk}L_{kk}^T$

trsm

for $i = k + 1, \dots, n/b$ **do**

$L_{ik} \leftarrow A_{ik}L_{kk}^{-T}$

end for

for $j = k + 1, \dots, n/b$ **do**

for $i = k + 1, \dots, n/b$ **do**

$A_{ij} \leftarrow A_{ij} - L_{ik}L_{jk}^T$

$$\begin{matrix} A_{11} & * & * & * \\ A_{21} & A_{22} & * & * \\ A_{31} & A_{32} & A_{33} & * \\ A_{41} & A_{42} & A_{43} & A_{44} \end{matrix}$$

end for

end for

end for

Node parallelism

The sequential algorithm:

```
for k = 1, ..., n/b do
     $A_{kk} \rightarrow L_{kk}L_{kk}^T$ 
    for i = k + 1, ..., n/b do
         $L_{ik} \leftarrow A_{ik}L_{kk}^{-T}$ 
    end for
    for j = k + 1, ..., n/b do
        for i = k + 1, ..., n/b do
             $A_{ij} \leftarrow A_{ij} - L_{ik}L_{jk}^T$ 
        end for
    end for
end for
```

trsm

$$\begin{matrix} A_{11} & * & * & * \\ A_{21} & A_{22} & * & * \\ A_{31} & A_{32} & A_{33} & * \\ A_{41} & A_{42} & A_{43} & A_{44} \end{matrix}$$

Node parallelism

The sequential algorithm:

for $k = 1, \dots, n/b$ **do**

$$A_{kk} \rightarrow L_{kk}L_{kk}^T$$

gemm

for $i = k + 1, \dots, n/b$ **do**

$$L_{ik} \leftarrow A_{ik}L_{kk}^{-T}$$

$$A_{11} * * *$$

end for

for $j = k + 1, \dots, n/b$ **do**

$$A_{21} \quad A_{22} * *$$

for $i = k + 1, \dots, n/b$ **do**

$$A_{31} \quad A_{32} \quad A_{33} \quad *$$

$$A_{ij} \leftarrow A_{ij} - L_{ik}L_{jk}^T$$

end for

$$A_{41} \quad A_{42} \quad A_{43} \quad A_{44}$$

end for

end for

Node parallelism

The sequential algorithm:

for $k = 1, \dots, n/b$ **do**

$A_{kk} \rightarrow L_{kk}L_{kk}^T$

gemm

for $i = k + 1, \dots, n/b$ **do**

$L_{ik} \leftarrow A_{ik}L_{kk}^{-T}$

$A_{11} * * *$

end for

for $j = k + 1, \dots, n/b$ **do**

for $i = k + 1, \dots, n/b$ **do**

$A_{ij} \leftarrow A_{ij} - L_{ik}L_{jk}^T$

$A_{21} A_{22} * *$

end for

$A_{31} A_{32} A_{33} *$

end for

$A_{41} A_{42} A_{43} A_{44}$

end for

Node parallelism

The sequential algorithm:

for $k = 1, \dots, n/b$ **do**

$A_{kk} \rightarrow L_{kk}L_{kk}^T$

gemm

for $i = k + 1, \dots, n/b$ **do**

$L_{ik} \leftarrow A_{ik}L_{kk}^{-T}$

$A_{11} * * *$

end for

for $j = k + 1, \dots, n/b$ **do**

for $i = k + 1, \dots, n/b$ **do**

$A_{ij} \leftarrow A_{ij} - L_{ik}L_{jk}^T$

$A_{21} A_{22} * *$

end for

$A_{31} A_{32} A_{33} *$

end for

$A_{41} A_{42} A_{43} A_{44}$

end for

Node parallelism

The sequential algorithm:

for $k = 1, \dots, n/b$ **do**

$A_{kk} \rightarrow L_{kk}L_{kk}^T$

gemm

for $i = k + 1, \dots, n/b$ **do**

$L_{ik} \leftarrow A_{ik}L_{kk}^{-T}$

$A_{11} * * *$

end for

for $j = k + 1, \dots, n/b$ **do**

$A_{21} A_{22} * *$

for $i = k + 1, \dots, n/b$ **do**

$A_{31} A_{32} A_{33} *$

$A_{ij} \leftarrow A_{ij} - L_{ik}L_{jk}^T$

$A_{41} A_{42} A_{43} A_{44}$

end for

end for

end for

Node parallelism

The sequential algorithm:

for $k = 1, \dots, n/b$ **do**

$$A_{kk} \rightarrow L_{kk}L_{kk}^T$$

gemm

for $i = k + 1, \dots, n/b$ **do**

$$L_{ik} \leftarrow A_{ik}L_{kk}^{-T}$$

$$A_{11} * * *$$

end for

for $j = k + 1, \dots, n/b$ **do**

$$A_{21} A_{22} * *$$

for $i = k + 1, \dots, n/b$ **do**

$$A_{ij} \leftarrow A_{ij} - L_{ik}L_{jk}^T$$

$$A_{31} A_{32} A_{33} *$$

end for

end for

end for

$$A_{41} A_{42} A_{43} A_{44}$$

Node parallelism

The sequential algorithm:

for $k = 1, \dots, n/b$ **do**

$A_{kk} \rightarrow L_{kk}L_{kk}^T$

gemm

for $i = k + 1, \dots, n/b$ **do**

$L_{ik} \leftarrow A_{ik}L_{kk}^{-T}$

$A_{11} * * *$

end for

for $j = k + 1, \dots, n/b$ **do**

$A_{21} A_{22} * *$

for $i = k + 1, \dots, n/b$ **do**

$A_{31} A_{32} A_{33} *$

$A_{ij} \leftarrow A_{ij} - L_{ik}L_{jk}^T$

$A_{41} A_{42} A_{43} A_{44}$

end for

end for

end for

Node parallelism

The sequential algorithm:

for $k = 1, \dots, n/b$ **do**

$$A_{kk} \rightarrow L_{kk}L_{kk}^T$$

for $i = k + 1, \dots, n/b$ **do**

$$L_{ik} \leftarrow A_{ik}L_{kk}^{-T}$$

end for

$$A_{11} \quad * \quad * \quad *$$

for $j = k + 1, \dots, n/b$ **do**

$$A_{21} \quad A_{22} \quad * \quad *$$

for $i = k + 1, \dots, n/b$ **do**

$$A_{31} \quad A_{32} \quad A_{33} \quad *$$

$$A_{ij} \leftarrow A_{ij} - L_{ik}L_{jk}^T$$

end for

$$A_{41} \quad A_{42} \quad A_{43} \quad A_{44}$$

end for

end for

Shared memory: node parallelism in multifrontal

Tasks are pushed into a pool along with their mutual dependencies and scheduled dynamically.

```
do k=1, n/b
    !$omp task depend(inout:A[k,k])
    call _potrf(A[k,k])

    do i=k+1, n/b
        !$omp task depend(in:A[k,k]) depend(inout:A[i,k])
        call _trsm( A[i,k], A[k,k])
    end do

    do i=k+1, n/b
        do j=k+1, i
            !$omp task depend(in:A[i,k],A[j,k])
            !$omp&      depend(inout:A[i,j])
            call _gemm(A[i,j], A[i,k], A[j,k])
        end do
    end do
end do
```

Shared memory: tree parallelism and memory

How to reduce the memory consumption? Give higher priority to tasks along a memory-minimizing traversal (more details later).

```
do n = 1, num_nodes ! in memory-minimizing order

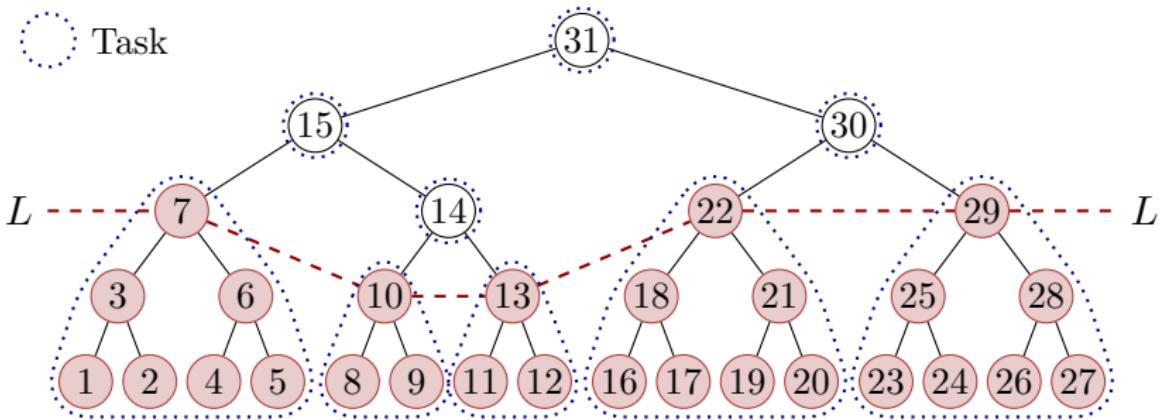
    forall c in children(n)
        !$omp task depend(in:front_c) depend(inout:front_n)
        !$omp&     priority(num_nodes-n)
        call assemble(front_c, front_n)
    end do

    !$omp task depend(inout:front_n)
    !$omp&     priority(num_nodes-n)
    call factorize(front_n)

end do
```

Reducing the scheduling overhead

In a typical elimination tree the number of nodes is much higher than the number of available processes. It can be beneficial to form macro tasks that process multiple nodes or, better, entire subtrees.



We define a layer called L such that each subtree rooted at L is entirely processed sequentially by a single task.

This reduces the **scheduling overhead** and improves the **efficiency of operations**.

Reducing the scheduling overhead

Algorithm 4 The Geist and Ng [10] algorithm.

Let $L \leftarrow$ Roots of the assembly tree

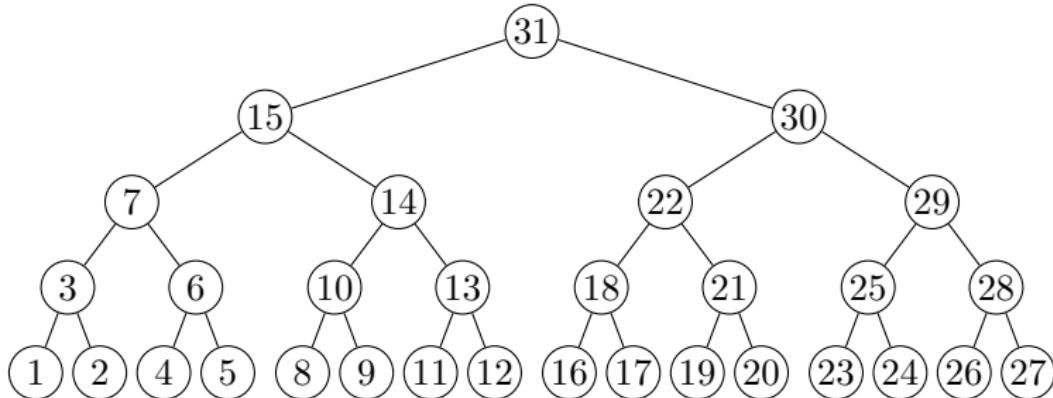
repeat

 Find the node q in L whose subtree is heaviest

 Set $L \leftarrow (L \setminus \{q\}) \cup \{\text{children of } q\}$

 Update weight of tasks

until acceptance criterion satisfied



Reducing the scheduling overhead

Algorithm 4 The Geist and Ng [10] algorithm.

Let $L \leftarrow$ Roots of the assembly tree

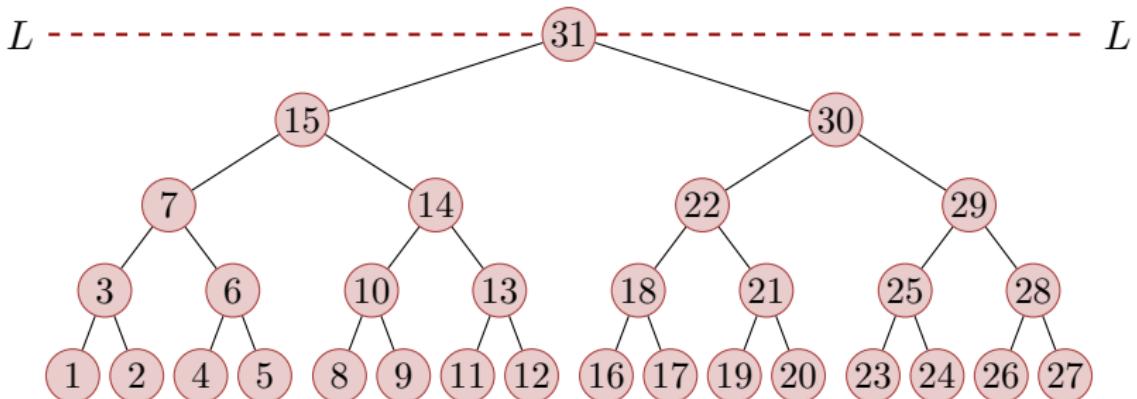
repeat

 Find the node q in L whose subtree is heaviest

 Set $L \leftarrow (L \setminus \{q\}) \cup \{\text{children of } q\}$

 Update weight of tasks

until acceptance criterion satisfied



Reducing the scheduling overhead

Algorithm 4 The Geist and Ng [10] algorithm.

Let $L \leftarrow$ Roots of the assembly tree

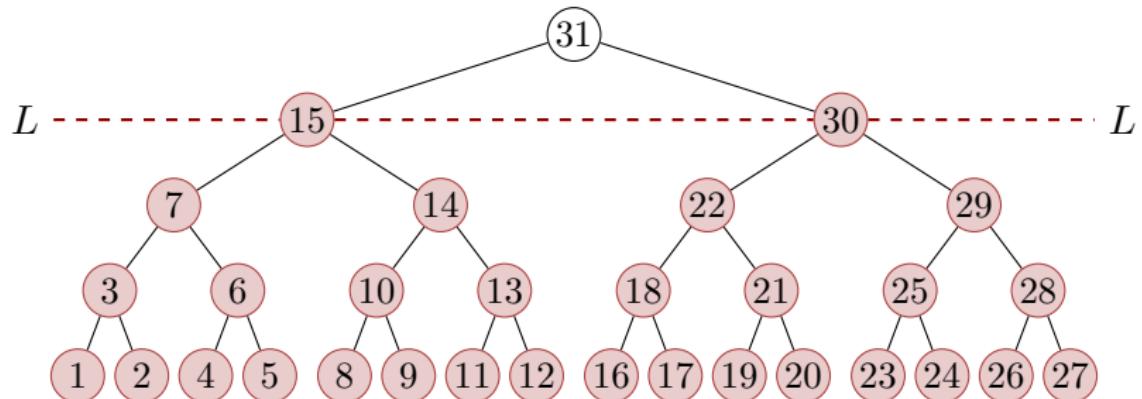
repeat

 Find the node q in L whose subtree is heaviest

 Set $L \leftarrow (L \setminus \{q\}) \cup \{\text{children of } q\}$

 Update weight of tasks

until acceptance criterion satisfied



Reducing the scheduling overhead

Algorithm 4 The Geist and Ng [10] algorithm.

Let $L \leftarrow$ Roots of the assembly tree

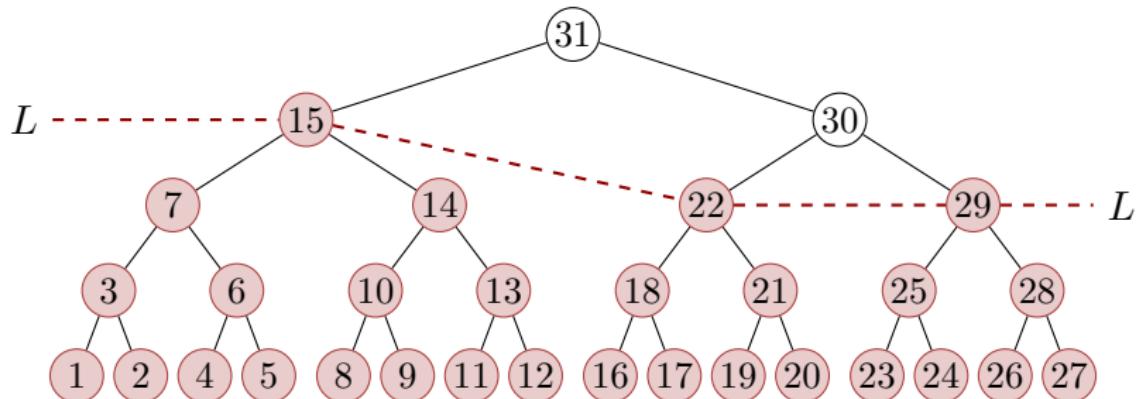
repeat

 Find the node q in L whose subtree is heaviest

 Set $L \leftarrow (L \setminus \{q\}) \cup \{\text{children of } q\}$

 Update weight of tasks

until acceptance criterion satisfied



Reducing the scheduling overhead

Algorithm 4 The Geist and Ng [10] algorithm.

Let $L \leftarrow$ Roots of the assembly tree

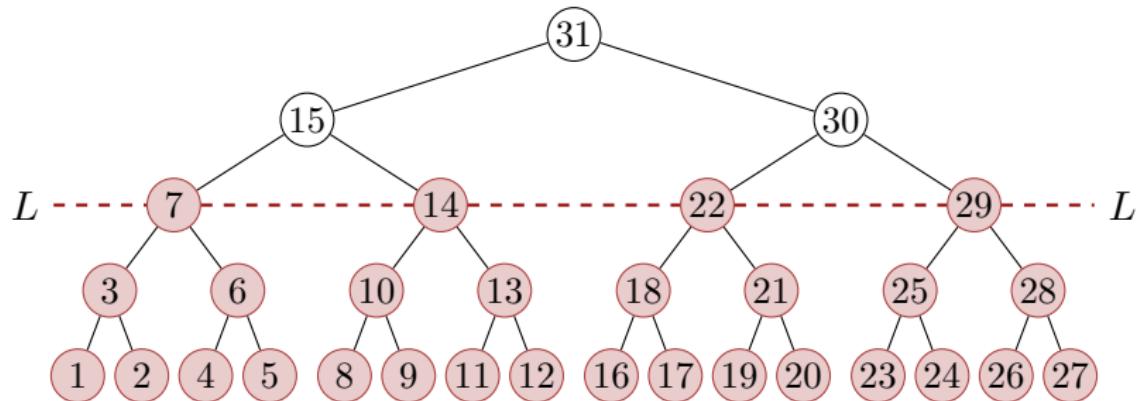
repeat

 Find the node q in L whose subtree is heaviest

 Set $L \leftarrow (L \setminus \{q\}) \cup \{\text{children of } q\}$

 Update weight of tasks

until acceptance criterion satisfied



Reducing the scheduling overhead

Algorithm 4 The Geist and Ng [10] algorithm.

Let $L \leftarrow$ Roots of the assembly tree

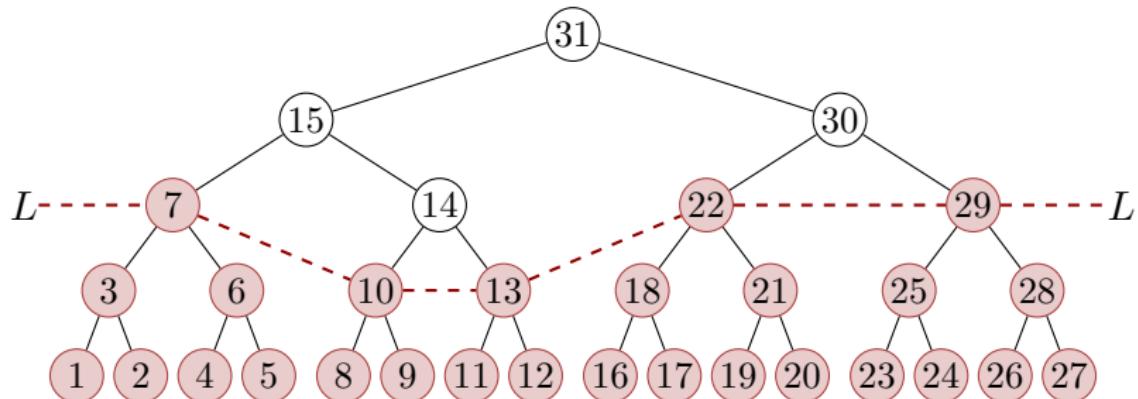
repeat

 Find the node q in L whose subtree is heaviest

 Set $L \leftarrow (L \setminus \{q\}) \cup \{\text{children of } q\}$

 Update weight of tasks

until acceptance criterion satisfied



Reducing the scheduling overhead

Taking the L layer into account in the shared-memory tree parallelism

```
do n = 1, num_nodes ! in topological order
  if(n below L) then
    continue ! skip this node
  else if (n on L) then
    !$omp task depend(inout:front_n)
    call process_subtree(n)
  else if (n above L) then
    forall c in children(n)
      !$omp task depend(in:front_c) depend(inout:front_n)
      call assemble(front_c, front_n)
    end do
    !$omp task depend(inout:front_n)
    call factorize(front_n)
  end if
end do
```

Outline

General introduction

Shared memory parallelism

Tree parallelism

Node parallelism

Scheduling and memory consumption

Scheduling overhead

Distributed memory parallelism

Tree parallelism

Node parallelism

Proportional mapping

Bound on parallelism

Tree parallelism

- Assumptions :
 - **Mapping**: we assume that each column of L /each node of the tree is assigned to a single process.
 - Each process is in charge of computing $cdiv(j)$ for columns j that it owns.
- Notation :
 - **mycols(p)** is the set of columns owned by process p .
 - **map(j)** gives the process owning column j (or task j).
 - **procs($L(:,k)$)** = { $\text{map}(j) — j \in \text{struct}(L(:,k))$ }
(only processes in $\text{procs}(L(:,k))$ require updates from column k – they correspond to ancestors of k in the tree).
 - **father(j)** is the father of node j in the elimination tree

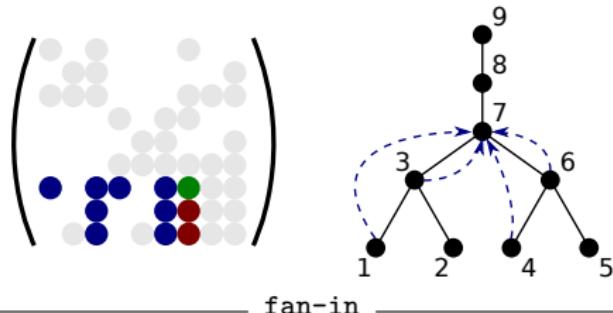
Tree parallelism variants

- Computational graph dependency
- Communication graph

There are three classical approaches to distributed memory parallelism:

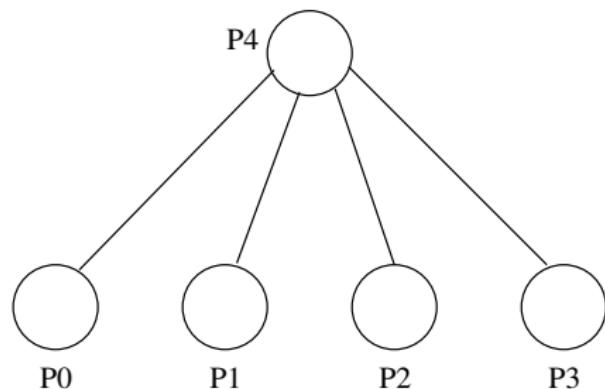
1. **Fan-in** : The fan-in algorithm is very similar to the left-looking approach and is **demand-driven**: data required are aggregated update columns computed by sending process
2. **Fan-out** : The fan-out algorithms is very similar to the right-looking approach and is **data driven**: data is sent as soon as it is produced.
3. **Multifrontal** : The communication pattern follows a bottom-up traversal of the tree. Messages are contribution blocks and are sent to the processor mapped on the father node

Fan-in variant (similar to left looking)

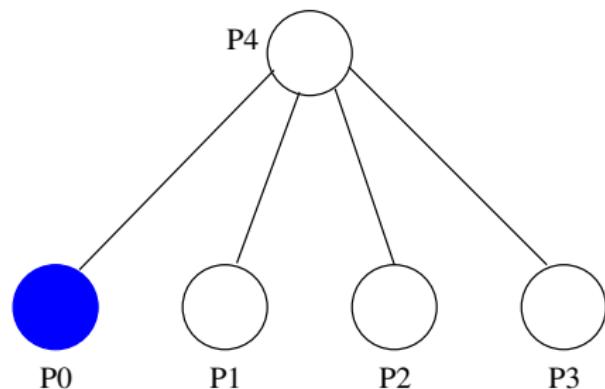


```
for j=1 to n
    u=0
    for all k in (struct(L(j,1:j-1)) ∩ mycols(p) )
        cmod(u,k)
    end for
    if map(j) != p
        send u to processor map(j)
    else
        incorporate u in column j
        receive all the updates on column j and incorporate them
        cddiv(j)
    end if
end for
```

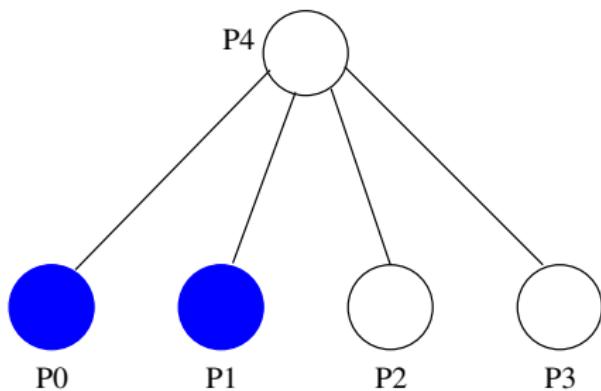
Fan-in variant



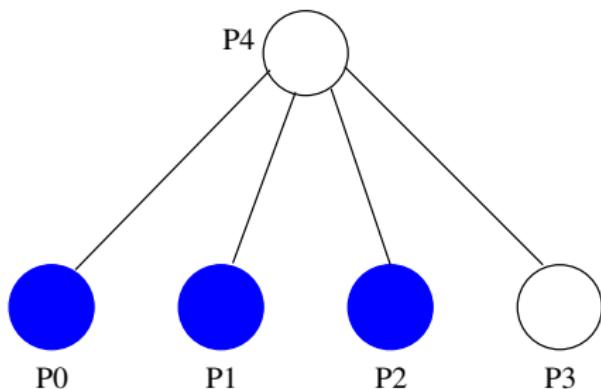
Fan-in variant



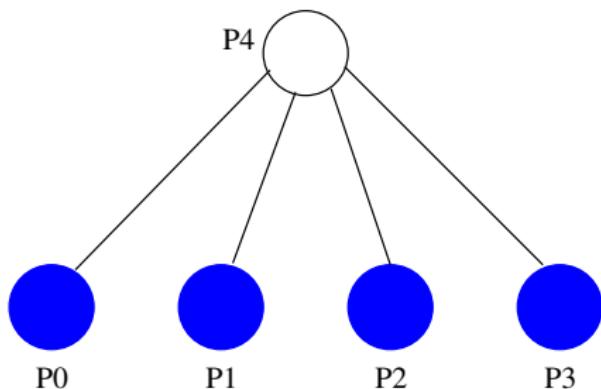
Fan-in variant



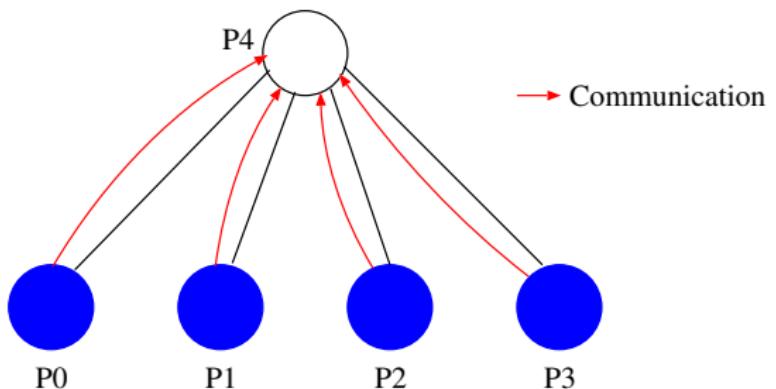
Fan-in variant



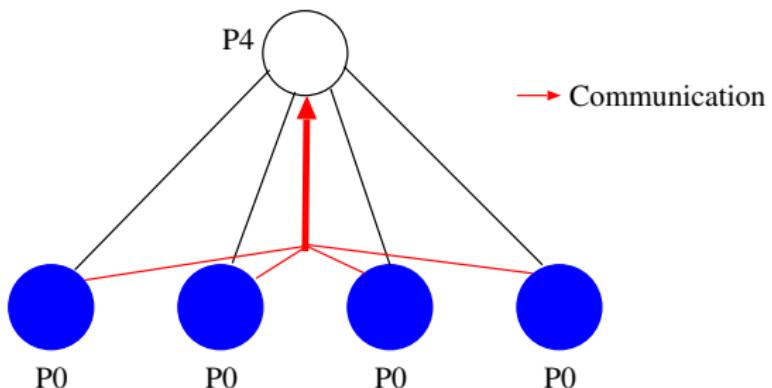
Fan-in variant



Fan-in variant

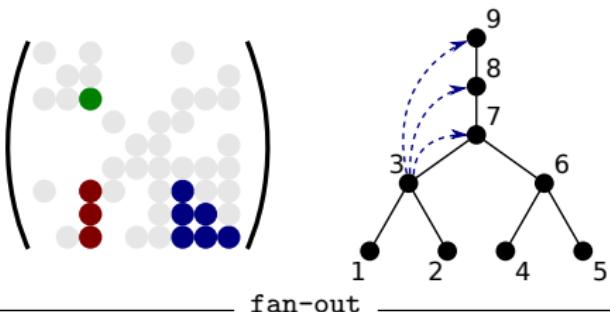


Fan-in variant



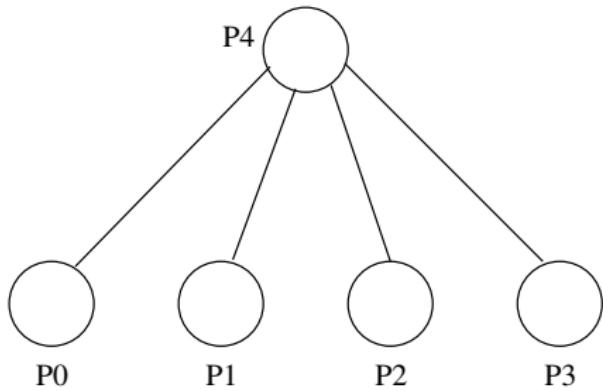
if $\forall i \in \text{children } map(i) = P0$ and $map(father) \neq P0$ (only) one message sent by $P0 \rightarrow$ exploits data locality in the tree mapping.

Fan-out variant (similar to right-looking)

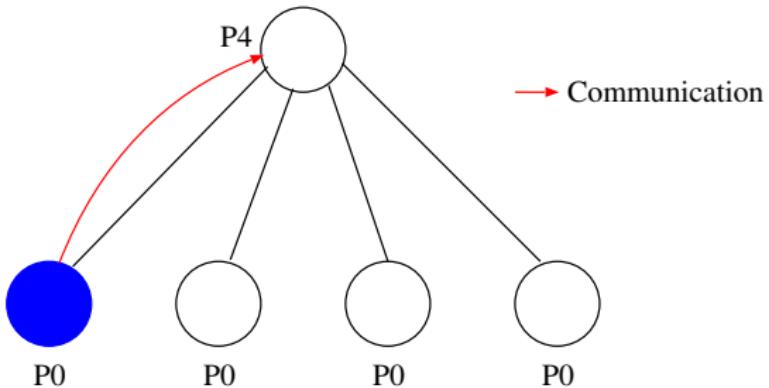


```
for all leaf nodes j in mycols(p)
    cdiv(j)
    send column L(:,j) to procs(L(:,j))
    mycols(p) = mycols(p) - {j}
end for
while mycols(p) != ∅
    receive any column (say L(:,k))
    for j in struct(L(:,k)) ∩ mycols(p)
        cmod(j,k)
        if column j is completely updated
            cdiv(j)
            send column L(:,j) to procs(L(:,j))
            mycols(p) = mycols(p) - {j}
        end if
    end for
end while
```

Fan-out variant

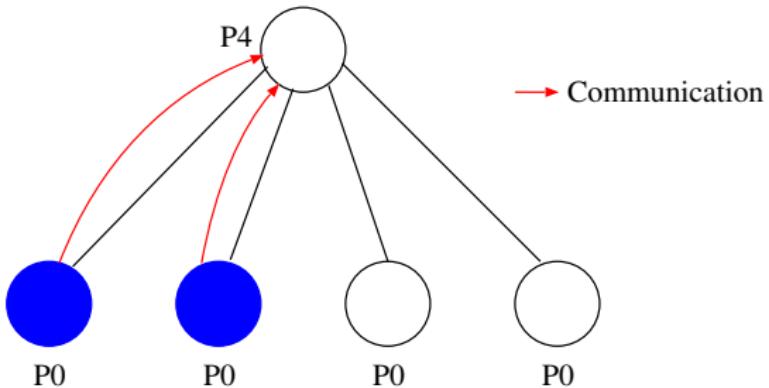


Fan-out variant



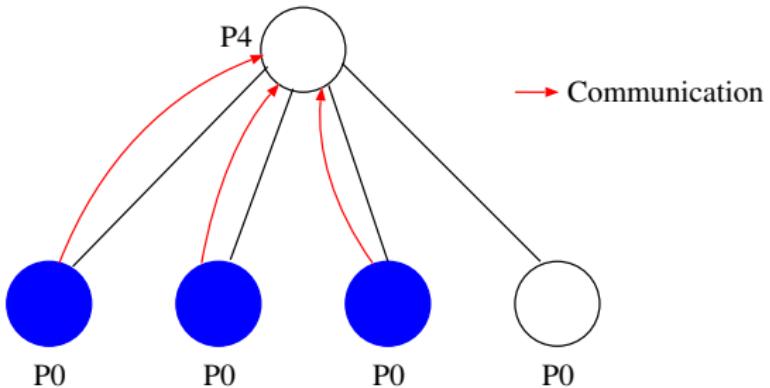
if $\forall i \in \text{children } map(i) = P0$ and $map(father) \neq P0$ then n messages (where n is the number of children) are sent by $P0$ to update the processor in charge of the father

Fan-out variant



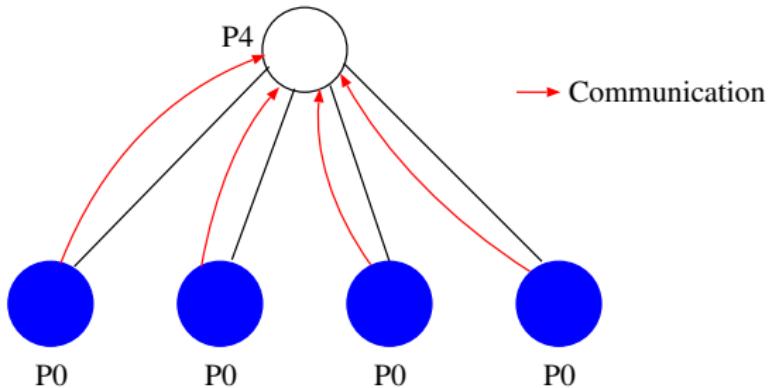
if $\forall i \in \text{children } map(i) = P0$ and $map(father) \neq P0$ then n messages (where n is the number of children) are sent by $P0$ to update the processor in charge of the father

Fan-out variant



if $\forall i \in \text{children } map(i) = P0$ and $map(father) \neq P0$ then n messages (where n is the number of children) are sent by $P0$ to update the processor in charge of the father

Fan-out variant



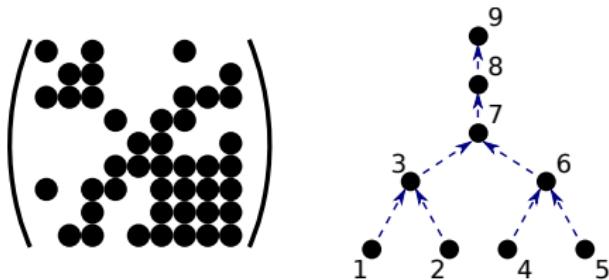
if $\forall i \in \text{children } map(i) = P0$ and $map(father) \neq P0$ then n messages (where n is the number of children) are sent by $P0$ to update the processor in charge of the father

Fan-out variant

Properties of fan-out:

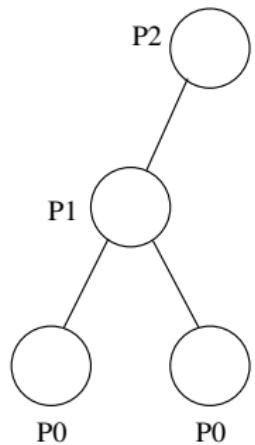
- Historically the first implemented.
- Incurs greater interprocessor communications than fan-in (or multifrontal) approach both in terms of
 - total number of messages
 - total volume
- Does not exploit data locality of proportional mapping.
- Improved algorithm (local aggregation):
 - send aggregated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

Multifrontal

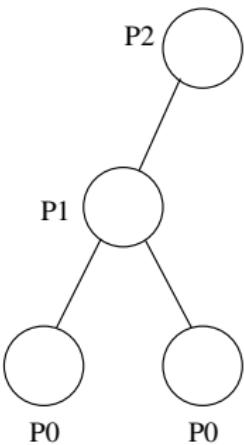


```
for all leaf nodes j in mycols(p)
    assemble front j
    partially factorize front j
    send the schur complement to procs(father(j))
    mycols(p) = mycols(p) - {j}
end for
while mycols(p) != ∅
    receive any contribution block (say for node j)
    assemble contribution block into front j
    if front j is completely assembled
        partially factorize front j
        send the schur complement to procs(father(j))
        mycols(p) = mycols(p) - {j}
    end if
end while
```

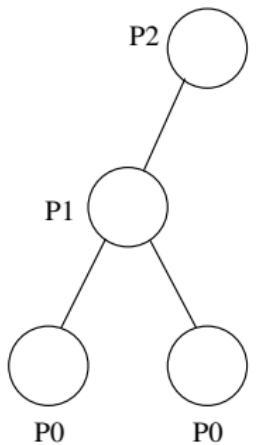
Multifrontal variant



Fan-in

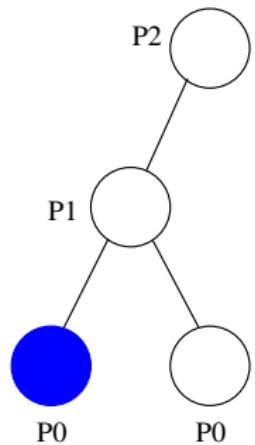


Fan-out

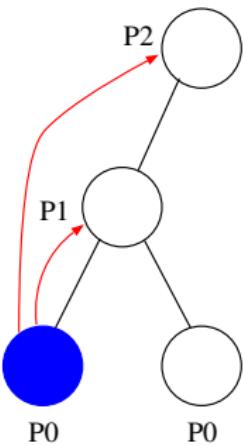


Multifrontal

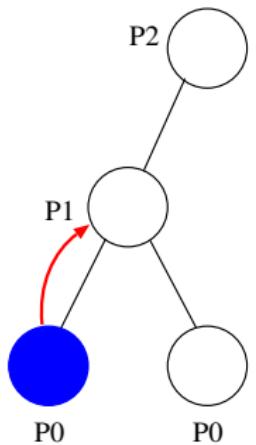
Multifrontal variant



Fan-in

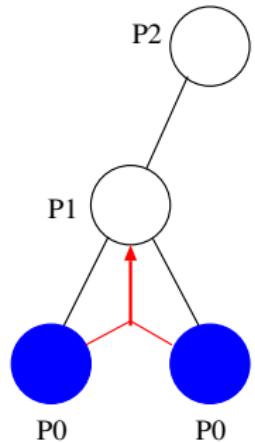


Fan-out

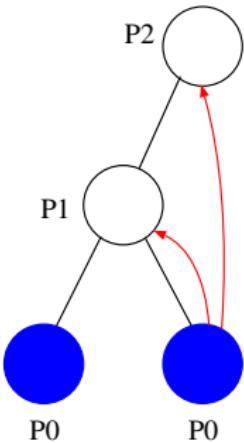


Multifrontal

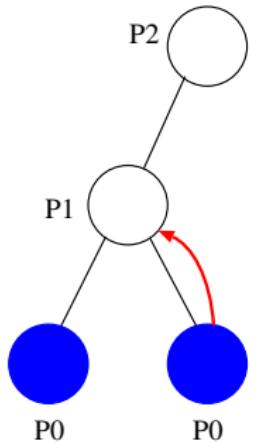
Multifrontal variant



Fan-in

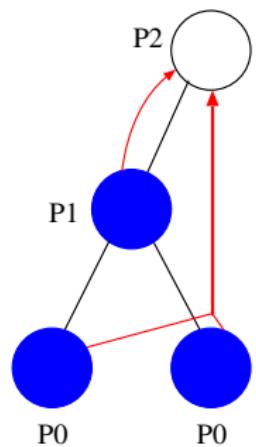


Fan-out

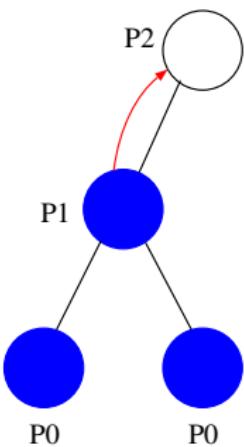


Multifrontal

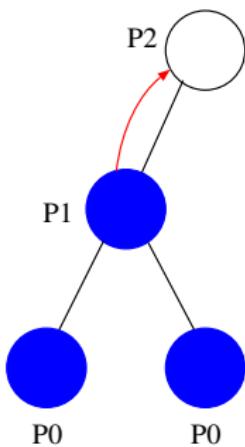
Multifrontal variant



Fan-in

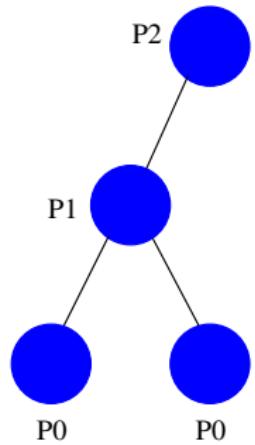


Fan-out

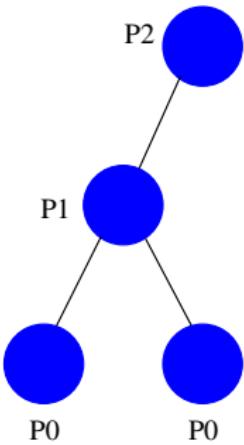


Multifrontal

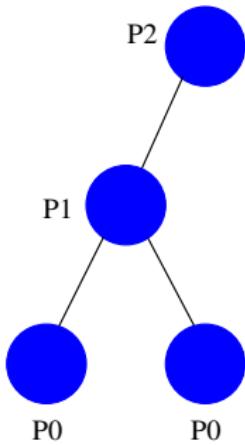
Multifrontal variant



Fan-in



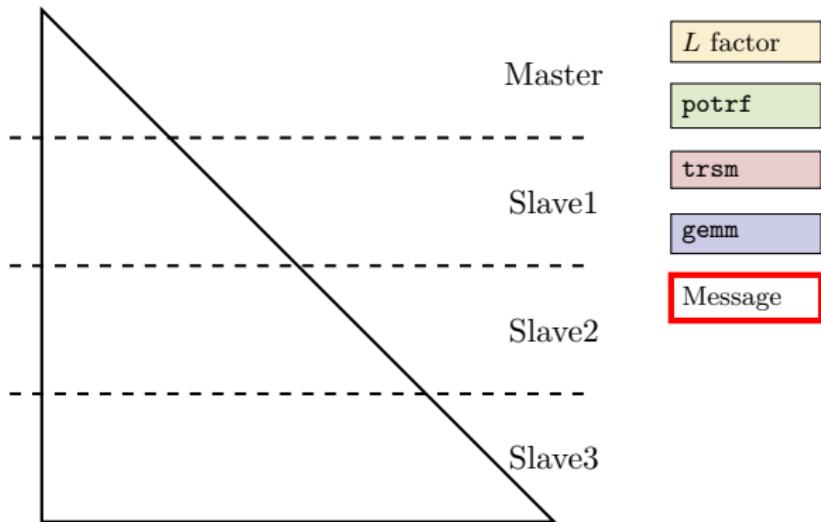
Fan-out



Multifrontal

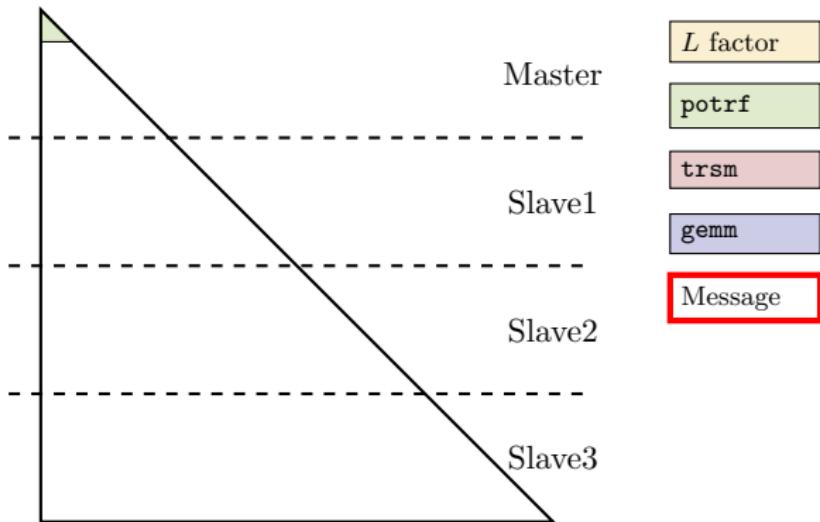
Node parallelism

We assume a block partitioning into block-rows: a **master** process takes all the fully summed rows while **slave** processes share the rows associated with non fully summed variables.



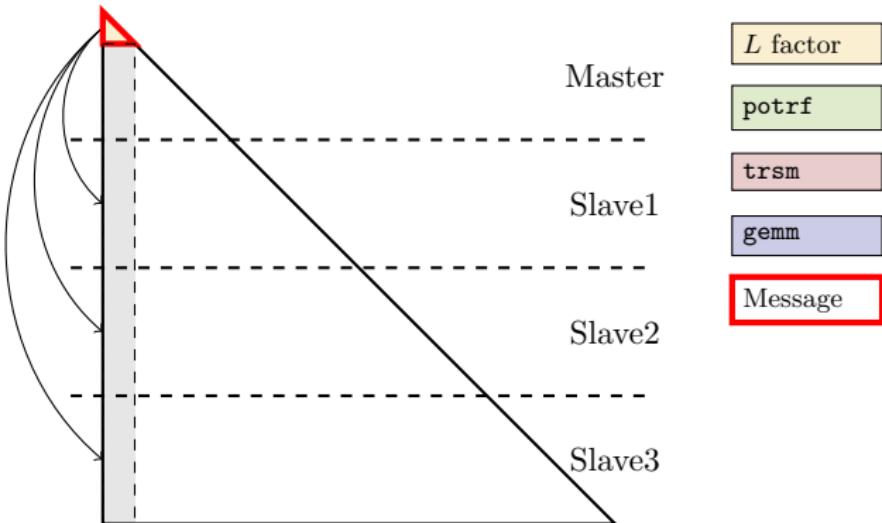
Node parallelism

We assume a block partitioning into block-rows: a **master** process takes all the fully summed rows while **slave** processes share the rows associated with non fully summed variables.



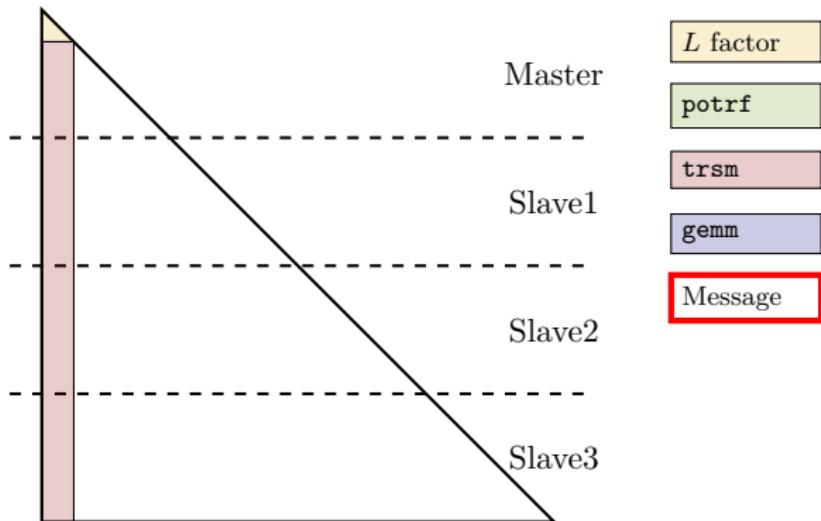
Node parallelism

We assume a block partitioning into block-rows: a **master** process takes all the fully summed rows while **slave** processes share the rows associated with non fully summed variables.



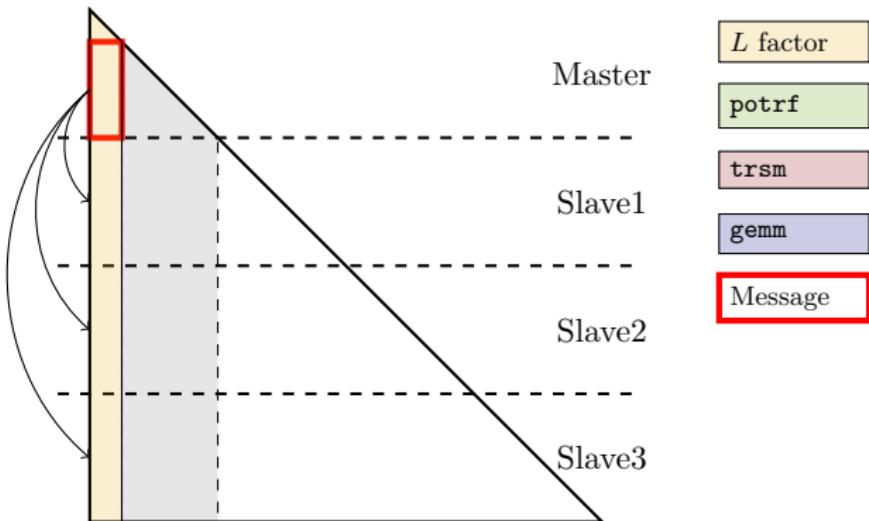
Node parallelism

We assume a block partitioning into block-rows: a **master** process takes all the fully summed rows while **slave** processes share the rows associated with non fully summed variables.



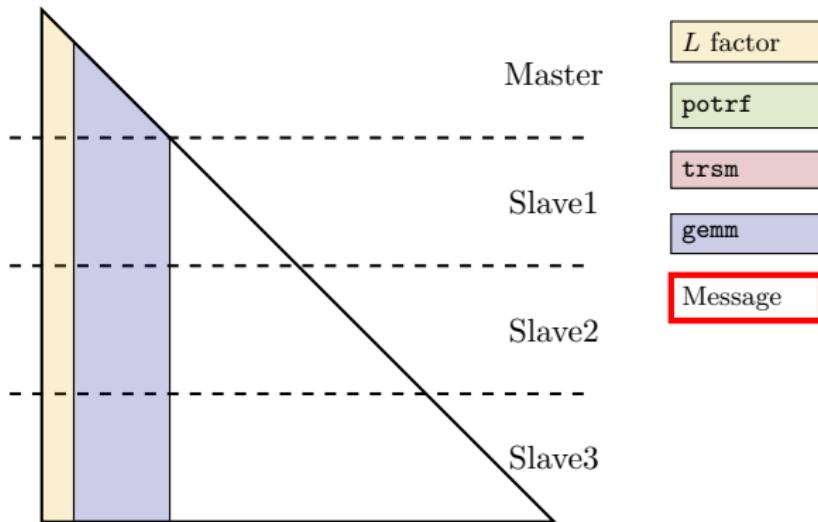
Node parallelism

We assume a block partitioning into block-rows: a **master** process takes all the fully summed rows while **slave** processes share the rows associated with non fully summed variables.



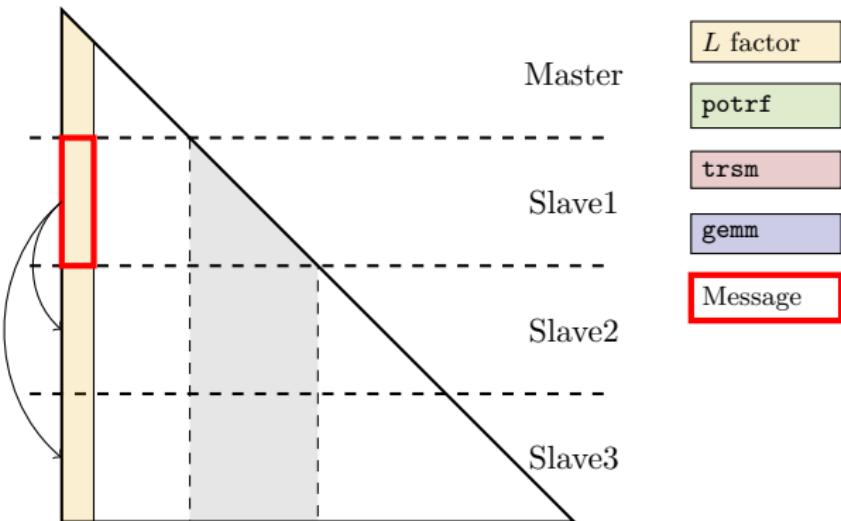
Node parallelism

We assume a block partitioning into block-rows: a **master** process takes all the fully summed rows while **slave** processes share the rows associated with non fully summed variables.



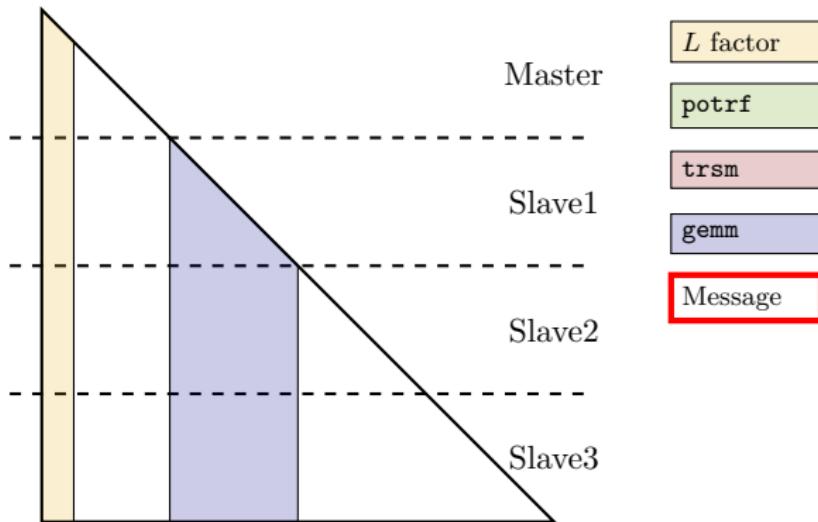
Node parallelism

We assume a block partitioning into block-rows: a **master** process takes all the fully summed rows while **slave** processes share the rows associated with non fully summed variables.



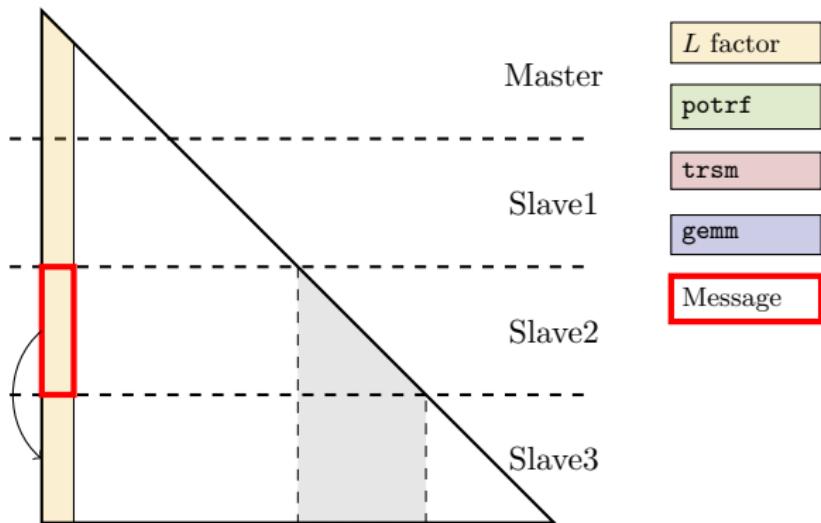
Node parallelism

We assume a block partitioning into block-rows: a **master** process takes all the fully summed rows while **slave** processes share the rows associated with non fully summed variables.



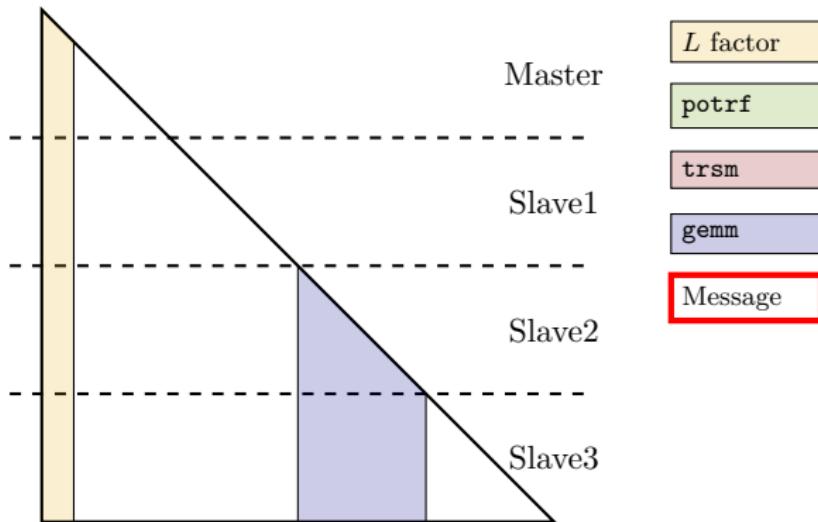
Node parallelism

We assume a block partitioning into block-rows: a **master** process takes all the fully summed rows while **slave** processes share the rows associated with non fully summed variables.



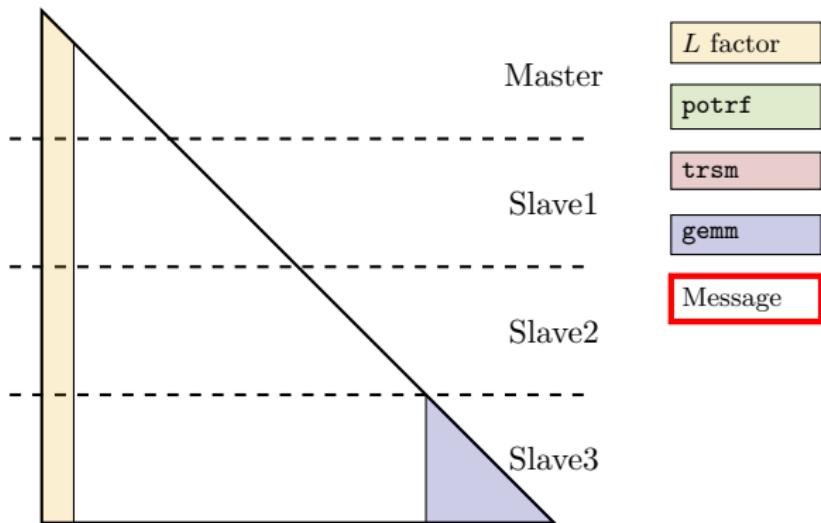
Node parallelism

We assume a block partitioning into block-rows: a **master** process takes all the fully summed rows while **slave** processes share the rows associated with non fully summed variables.



Node parallelism

We assume a block partitioning into block-rows: a **master** process takes all the fully summed rows while **slave** processes share the rows associated with non fully summed variables.

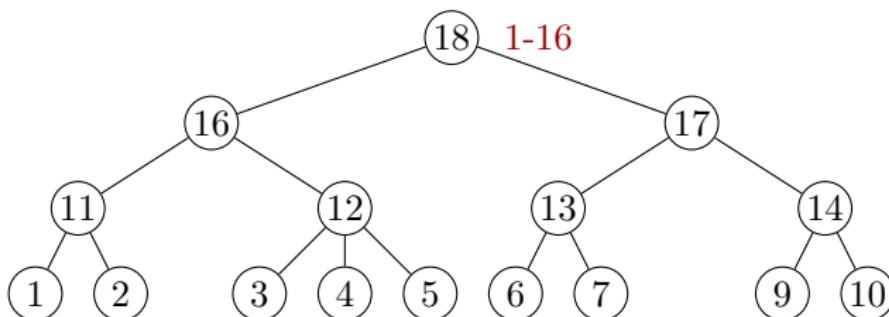


Proportional mapping

The **Proportional Mapping** method was proposed by Pothen et al. [19] and aims at computing an efficient processes-to-nodes mapping for sparse, direct solvers for distributed memory parallelism.

Proportional mapping

- initially assigns all processes to root node.
- performs a top-down traversal of the tree where the processes assigned to a node are subdivided among its children in a way that is proportional to their relative weight

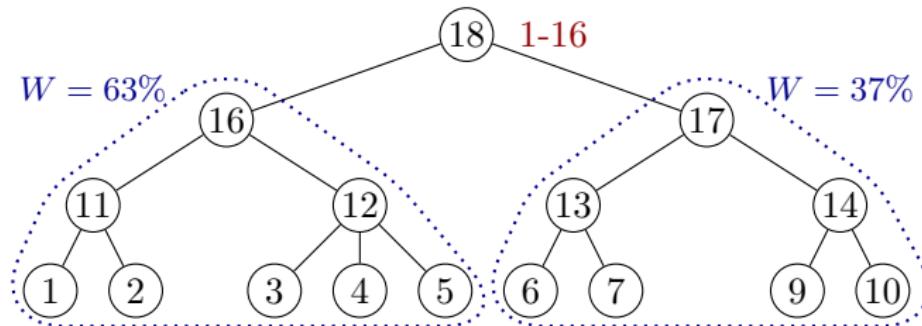


Proportional mapping

The **Proportional Mapping** method was proposed by Pothen et al. [19] and aims at computing an efficient processes-to-nodes mapping for sparse, direct solvers for distributed memory parallelism.

Proportional mapping

- initially assigns all processes to root node.
- performs a top-down traversal of the tree where the processes assigned to a node are subdivided among its children in a way that is proportional to their relative weight

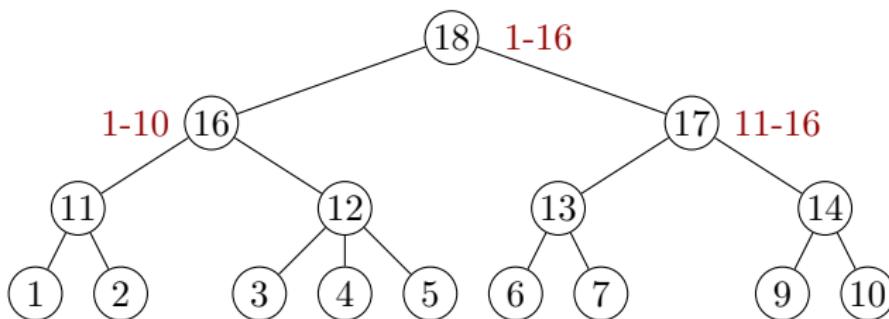


Proportional mapping

The **Proportional Mapping** method was proposed by Pothen et al. [19] and aims at computing an efficient processes-to-nodes mapping for sparse, direct solvers for distributed memory parallelism.

Proportional mapping

- initially assigns all processes to root node.
- performs a top-down traversal of the tree where the processes assigned to a node are subdivided among its children in a way that is proportional to their relative weight

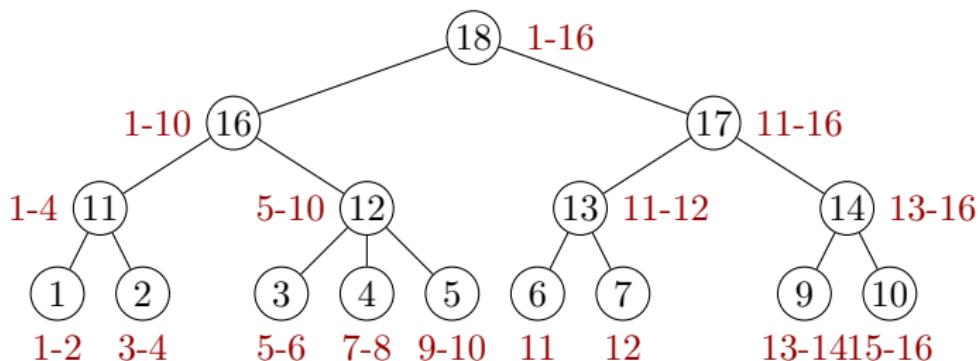


Proportional mapping

The **Proportional Mapping** method was proposed by Pothen et al. [19] and aims at computing an efficient processes-to-nodes mapping for sparse, direct solvers for distributed memory parallelism.

Proportional mapping

- initially assigns all processes to root node.
- performs a top-down traversal of the tree where the processes assigned to a node are subdivided among its children in a way that is proportional to their relative weight



Proportional mapping

Properties of proportional mapping:

- Compatible with node parallelism: more processes are assigned to topmost nodes that are larger
- Reduces communications in tree parallelism:
 - no branch-to-branch communications because the assigned processes are in disjoint subsets
 - processes assigned to a node are also assigned to its parent which reduces communications in assembly operations
- Aims at achieving a good balance between the branches of the tree because of the proportional distribution
- The weight of subtrees can be computed using different metrics (e.g., flops or memory or a combination of the two) which allows for balancing different properties

Outline

General introduction

Shared memory parallelism

Tree parallelism

Node parallelism

Scheduling and memory consumption

Scheduling overhead

Distributed memory parallelism

Tree parallelism

Node parallelism

Proportional mapping

Bound on parallelism

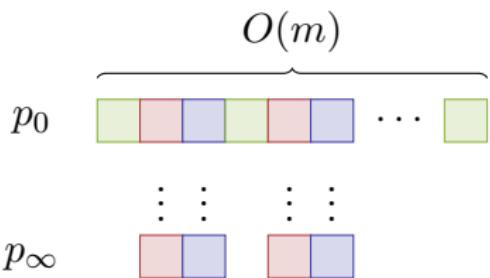
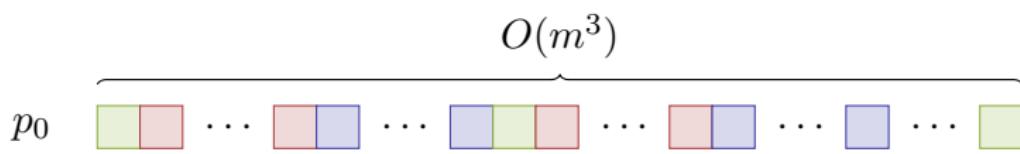
Bound on parallelism

What is the best possible speedup we can achieve in the multifrontal method? Let's make these assumptions:

1. We have an infinite number of processes
2. Proportional mapping is used
3. Communications have no cost
4. We are in the same case as in Definition 10.1. Under these assumptions the **sequential** multifrontal factorization time is $O(N^3)$.

Bound on parallelism

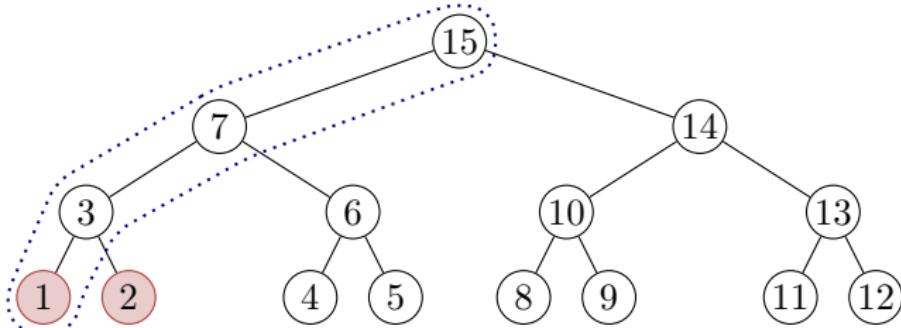
- The number of flops for the factorization of a dense matrix of order m is $O(m^3)$. We will assume that the time for its sequential factorization is also $O(m^3)$
- We will assume that the lower bound on the time for the parallel factorization of a frontal matrix of order m is $O(m)$ because it cannot be lower than the time to traverse the critical path



$A_{kk} \rightarrow L_{kk}L_{kk}^T$
 $A_{ik} \leftarrow A_{ik}L_{ii}^{-T}$
 $A_{ij} \leftarrow A_{ij} - L_{ik}L_{jk}^T$

Bound on parallelism

- When tree parallelism is used all the branches are traversed concurrently; the minimum required time is the time needed to traverse the longest (heaviest) branch. Because, in our case, all the branches are equal, we will measure the time needed for any branch.



Bound on parallelism

Execution time bounds:

- Case 1: only tree parallelism

$$\mathcal{C}_{mf}^p = O\left(\sum_{l=0}^{\log_2 N} \left(\frac{N}{2^l}\right)^3\right) = O(N^3)$$

- Case 2: only node parallelism

$$\mathcal{C}_{mf}^p = O\left(\sum_{l=0}^{\log_2 N} 4^l \frac{N}{2^l}\right) = O(N^2)$$

- Case 3: tree and node parallelism

$$\mathcal{C}_{mf}^p = O\left(\sum_{l=0}^{\log_2 N} \frac{N}{2^l}\right) = O(N)$$

Bound on parallelism

Regular problems (nested dissection)	2D	3D
	$N \times N$ grid	$N \times N \times N$ grid
Sequential	$O(N^3)$	$O(N^6)$
Tree parallelism	$O(N^3)$	$O(N^6)$
Node parallelism	$O(N^2)$	$O(N^3)$
Tree and node parallelism	$O(N)$	$O(N^2)$

Remarks:

- Tree parallelism brings no asymptotic improvement if no node parallelism !!!
- It is crucial to use both sources of parallelism to achieve good parallel efficiency
- Note that in practice in practice we are quite far from the assumptions we have made (no infinite number of processors, no zero-cost communications etc.)

Memory

Outline

Memory consumption in the sequential multifrontal method

- Equivalent orderings and postorder

- Memory-minimizing postorders

Memory consumption in the parallel multifrontal method

Equivalent orderings

Definition 15.1 (Equivalent orderings)

Two orderings P and Q are **equivalent** if the structures of the filled graphs of PAP^T and QAQ^T are isomorphic.

For what's been said before, **all topological orderings on $T(A)$ are equivalent:**

- Let P be the permutation matrix corresponding to a topological ordering of $T(A)$. Then, $G^+(PAP^T)$ and $G^+(A)$ are isomorphic.
- Let P be the permutation matrix corresponding to a topological ordering of $T(A)$. The elimination tree $T(PAP^T)$ and $T(A)$ are isomorphic.

Equivalent orderings: postorder

Definition 15.2 (Postorder)

A postorder is a topological order where all the nodes in each subtree are numbered consecutively.

Among all the topological orders, the postorder has a very favorable property: a node is visited as soon as all of its children have been visited. This has a twofold advantage:

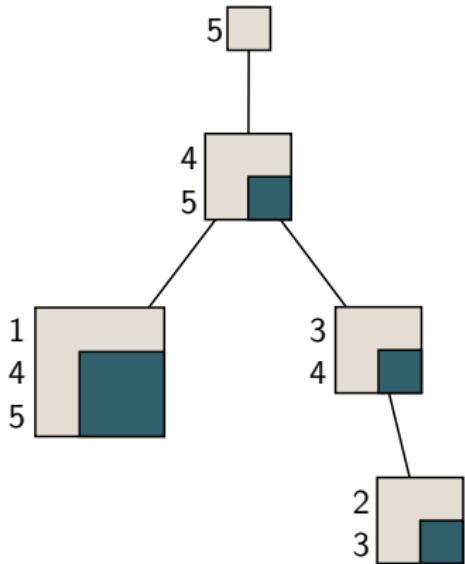
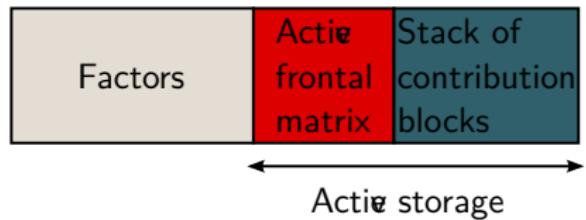
- In a sequential execution, a **stack** data structure can be used to store the contribution blocks. A contribution block is pushed on top of the stack when it is produced (i.e., upon factorization of the corresponding front) and popped from the top of the stack when it is assembled into the parent front
- It allows for a better data locality because the contribution locks that are used for an assembly operations are on top of the stack and thus have been produced recently.

The multifrontal method: memory handling

$$A = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \xrightarrow{\quad L+U-I = \quad} \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \end{matrix} \end{matrix}$$

Storage is divided into two parts:

- Factors
- Active memory



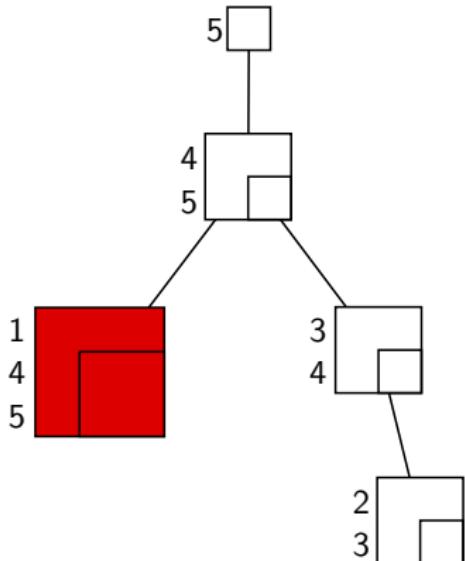
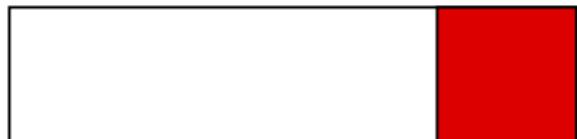
Elimination tree

The multifrontal method: memory handling

$$A = \begin{array}{|c|c|c|c|c|} \hline 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \text{light gray} & & & \\ \hline 2 & & \text{light gray} & & \\ \hline 3 & & & \text{light gray} & \\ \hline 4 & & & & \text{light gray} \\ \hline 5 & & & & \\ \hline \end{array} \rightarrow L+U-I = \begin{array}{|c|c|c|c|c|} \hline 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \text{red} & & & \\ \hline 2 & & \text{white} & & \\ \hline 3 & & & \text{white} & \\ \hline 4 & \text{red} & & & \text{red} \\ \hline 5 & & & & \\ \hline \end{array}$$

Storage is divided into two parts:

- Factors
- Active memory



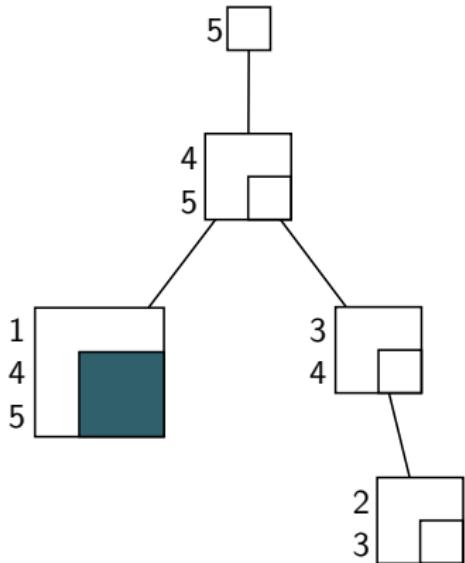
Elimination tree

The multifrontal method: memory handling

$$A = \begin{array}{|c|c|c|c|c|} \hline 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \text{---} & & & \\ \hline 2 & \text{---} & \text{---} & & \\ \hline 3 & \text{---} & \text{---} & \text{---} & \\ \hline 4 & \text{---} & & \text{---} & \\ \hline 5 & \text{---} & & & \text{---} \\ \hline \end{array} \rightarrow L+U-I = \begin{array}{|c|c|c|c|c|} \hline 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \text{---} & & & \\ \hline 2 & \text{---} & \text{---} & & \\ \hline 3 & \text{---} & \text{---} & \text{---} & \\ \hline 4 & \text{---} & & \text{---} & \text{---} \\ \hline 5 & \text{---} & & & \text{---} \\ \hline \end{array}$$

Storage is divided into two parts:

- Factors
- Active memory



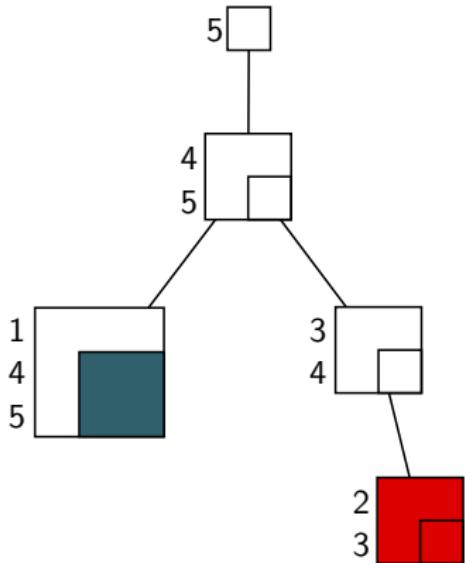
Elimination tree

The multifrontal method: memory handling

$$A = \begin{array}{|c|c|c|c|c|} \hline 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \text{---} & & & \\ \hline 2 & \text{---} & \text{---} & & \\ \hline 3 & \text{---} & \text{---} & \text{---} & \\ \hline 4 & \text{---} & & \text{---} & \\ \hline 5 & \text{---} & & & \text{---} \\ \hline \end{array} \rightarrow L+U-I = \begin{array}{|c|c|c|c|c|} \hline 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \text{---} & & & \\ \hline 2 & \text{---} & \text{---} & \text{---} & \\ \hline 3 & \text{---} & \text{---} & \text{---} & \\ \hline 4 & \text{---} & \text{---} & \text{---} & \\ \hline 5 & \text{---} & \text{---} & \text{---} & \\ \hline \end{array}$$

Storage is divided into two parts:

- Factors
- Active memory



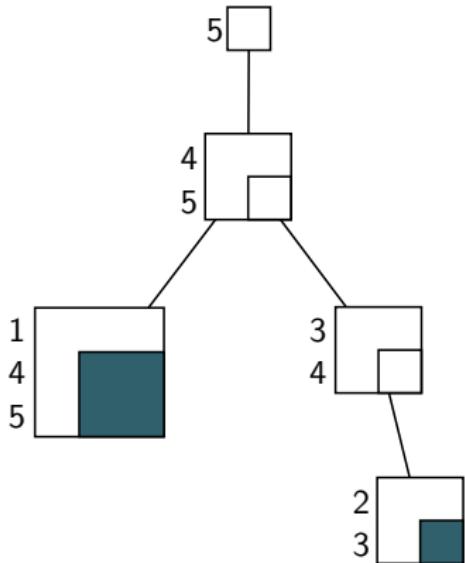
Elimination tree

The multifrontal method: memory handling

$$A = \begin{array}{|c|c|c|c|c|} \hline 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \text{---} & & & \\ \hline 2 & \text{---} & \text{---} & & \\ \hline 3 & \text{---} & \text{---} & \text{---} & \\ \hline 4 & \text{---} & & \text{---} & \\ \hline 5 & \text{---} & & & \text{---} \\ \hline \end{array} \rightarrow L+U-I = \begin{array}{|c|c|c|c|c|} \hline 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \text{---} & & & \\ \hline 2 & \text{---} & \text{---} & & \\ \hline 3 & \text{---} & \text{---} & \text{---} & \\ \hline 4 & \text{---} & & \text{---} & \\ \hline 5 & \text{---} & & & \text{---} \\ \hline \end{array}$$

Storage is divided into two parts:

- Factors
- Active memory



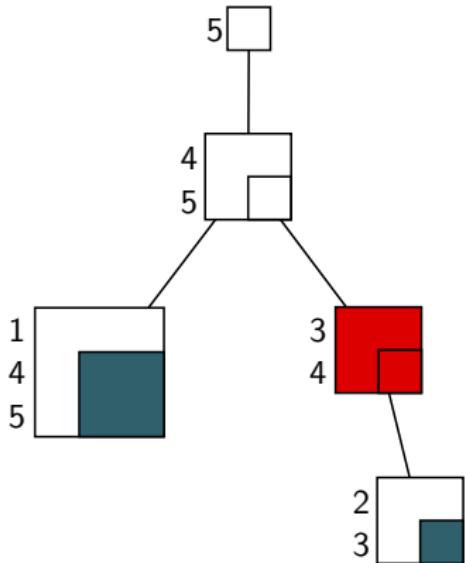
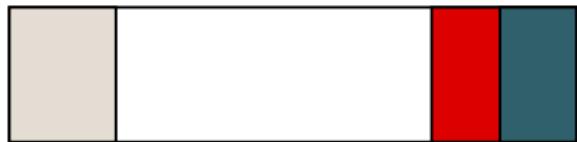
Elimination tree

The multifrontal method: memory handling

$$A = \begin{array}{|c|c|c|c|c|} \hline 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \text{---} & & & \\ \hline 2 & \text{---} & \text{---} & & \\ \hline 3 & \text{---} & \text{---} & \text{---} & \\ \hline 4 & \text{---} & \text{---} & \text{---} & \\ \hline 5 & \text{---} & \text{---} & \text{---} & \\ \hline \end{array} \rightarrow L+U-I = \begin{array}{|c|c|c|c|c|} \hline 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \text{---} & & & \\ \hline 2 & \text{---} & \text{---} & & \\ \hline 3 & \text{---} & \text{---} & \text{---} & \\ \hline 4 & \text{---} & \text{---} & \text{---} & \\ \hline 5 & \text{---} & \text{---} & \text{---} & \\ \hline \end{array}$$

Storage is divided into two parts:

- Factors
- Active memory



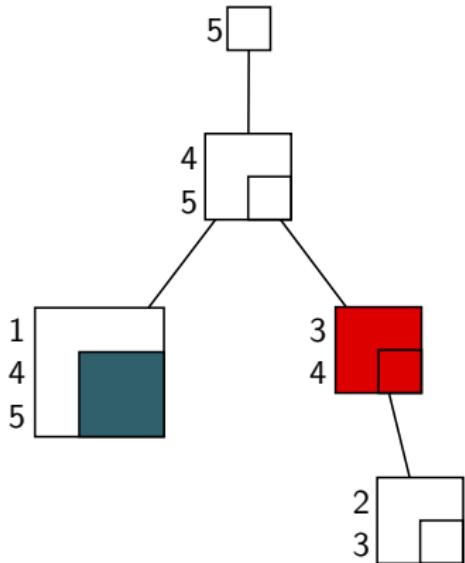
Elimination tree

The multifrontal method: memory handling

$$A = \begin{array}{|c|c|c|c|c|} \hline 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \text{---} & & & \\ \hline 2 & \text{---} & \text{---} & & \\ \hline 3 & \text{---} & \text{---} & \text{---} & \\ \hline 4 & \text{---} & \text{---} & \text{---} & \\ \hline 5 & \text{---} & \text{---} & \text{---} & \\ \hline \end{array} \rightarrow L+U-I = \begin{array}{|c|c|c|c|c|} \hline 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \text{---} & & & \\ \hline 2 & \text{---} & \text{---} & & \\ \hline 3 & \text{---} & \text{---} & \text{---} & \\ \hline 4 & \text{---} & \text{---} & \text{---} & \\ \hline 5 & \text{---} & \text{---} & \text{---} & \\ \hline \end{array}$$

Storage is divided into two parts:

- Factors
- Active memory



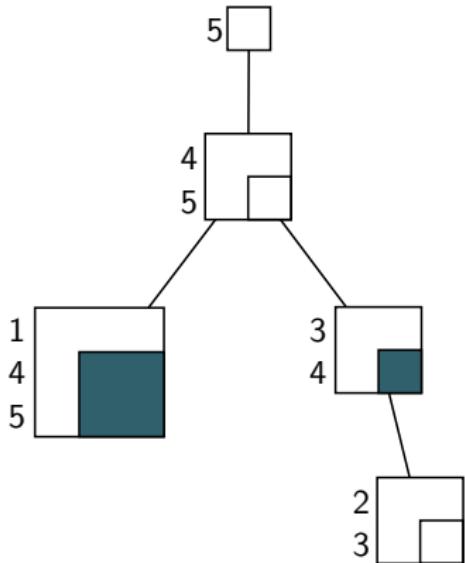
Elimination tree

The multifrontal method: memory handling

$$A = \begin{array}{|c|c|c|c|c|} \hline 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \text{---} & & & \\ \hline 2 & & \text{---} & & \\ \hline 3 & & & \text{---} & \\ \hline 4 & & & & \text{---} \\ \hline 5 & & & & & \text{---} \\ \hline \end{array} \rightarrow L+U-I = \begin{array}{|c|c|c|c|c|} \hline 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \text{---} & & & \\ \hline 2 & & \text{---} & & \\ \hline 3 & & & \text{---} & \\ \hline 4 & & & & \text{---} \\ \hline 5 & & & & & \text{---} \\ \hline \end{array}$$

Storage is divided into two parts:

- Factors
- Active memory



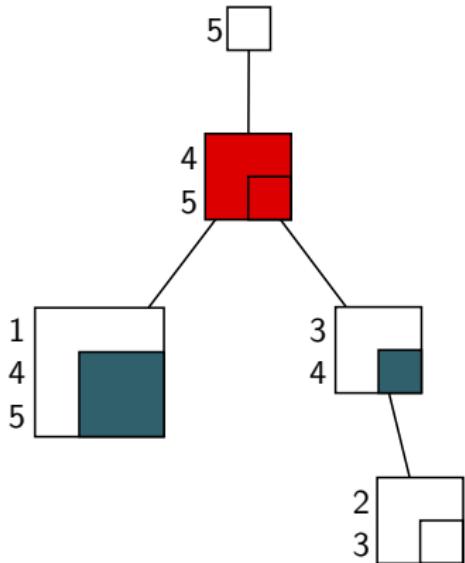
Elimination tree

The multifrontal method: memory handling

$$A = \begin{array}{|c|c|c|c|c|} \hline 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \text{---} & & & \\ \hline 2 & & \text{---} & & \\ \hline 3 & & & \text{---} & \\ \hline 4 & & & & \text{---} \\ \hline 5 & & & & & \text{---} \\ \hline \end{array} \rightarrow L+U-I = \begin{array}{|c|c|c|c|c|} \hline 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \text{---} & & & \\ \hline 2 & & \text{---} & & \\ \hline 3 & & & \text{---} & \\ \hline 4 & & & & \text{---} \\ \hline 5 & & & & & \text{---} \\ \hline \end{array}$$

Storage is divided into two parts:

- Factors
- Active memory



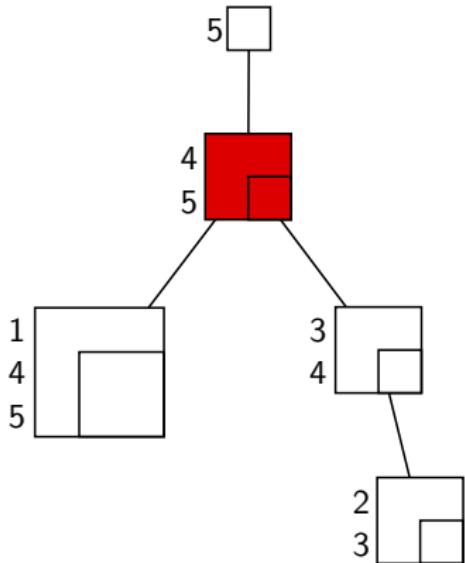
Elimination tree

The multifrontal method: memory handling

$$A = \begin{array}{|c|c|c|c|c|} \hline 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \text{---} & & & \\ \hline 2 & & \text{---} & & \\ \hline 3 & & & \text{---} & \\ \hline 4 & & & & \text{---} \\ \hline 5 & & & & \\ \hline \end{array} \rightarrow L+U-I = \begin{array}{|c|c|c|c|c|} \hline 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \text{---} & & & \\ \hline 2 & & \text{---} & & \\ \hline 3 & & & \text{---} & \\ \hline 4 & & & & \text{---} \\ \hline 5 & & & & \\ \hline \end{array}$$

Storage is divided into two parts:

- Factors
- Active memory



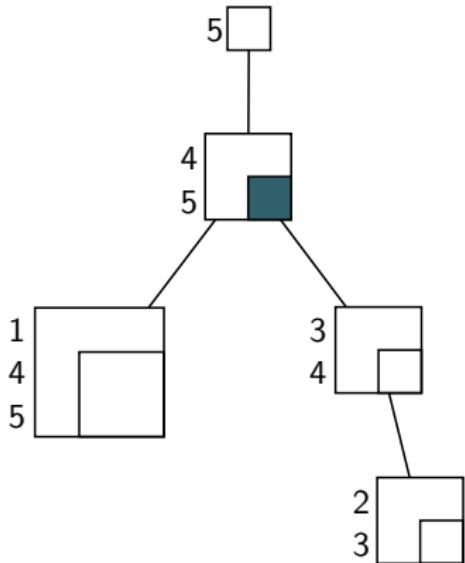
Elimination tree

The multifrontal method: memory handling

$$A = \begin{array}{|c|c|c|c|c|} \hline 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \text{---} & & & \\ \hline 2 & \text{---} & \text{---} & & \\ \hline 3 & \text{---} & \text{---} & \text{---} & \\ \hline 4 & \text{---} & & \text{---} & \\ \hline 5 & \text{---} & & & \text{---} \\ \hline \end{array} \rightarrow L+U-I = \begin{array}{|c|c|c|c|c|} \hline 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \text{---} & & & \\ \hline 2 & \text{---} & \text{---} & & \\ \hline 3 & \text{---} & \text{---} & \text{---} & \\ \hline 4 & \text{---} & & \text{---} & \\ \hline 5 & \text{---} & & & \text{---} \\ \hline \end{array}$$

Storage is divided into two parts:

- Factors
- Active memory



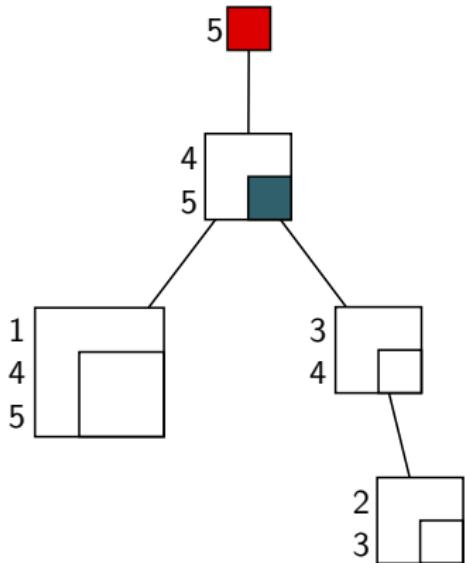
Elimination tree

The multifrontal method: memory handling

$$A = \begin{array}{|c|c|c|c|c|} \hline & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \text{---} & & & & \\ \hline 2 & & \text{---} & & & \\ \hline 3 & & & \text{---} & & \\ \hline 4 & & & & \text{---} & \\ \hline 5 & & & & & \text{---} \\ \hline \end{array} \rightarrow L+U-I = \begin{array}{|c|c|c|c|c|} \hline & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \text{---} & & & & \\ \hline 2 & & \text{---} & & & \\ \hline 3 & & & \text{---} & & \\ \hline 4 & & & & \text{---} & \\ \hline 5 & & & & & \text{---} \\ \hline \end{array}$$

Storage is divided into two parts:

- Factors
- Active memory



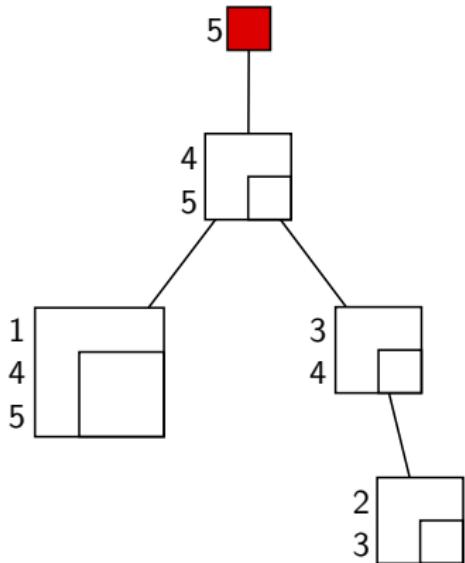
Elimination tree

The multifrontal method: memory handling

$$A = \begin{array}{|c|c|c|c|c|}\hline 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \text{---} & & & \\ \hline 2 & & \text{---} & & \\ \hline 3 & & & \text{---} & \\ \hline 4 & & & & \text{---} \\ \hline 5 & & & & & \text{---} \\ \hline\end{array} \rightarrow L+U-I = \begin{array}{|c|c|c|c|c|}\hline 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \text{---} & & & \\ \hline 2 & & \text{---} & & \\ \hline 3 & & & \text{---} & \\ \hline 4 & & & & \text{---} \\ \hline 5 & & & & & \text{---} \\ \hline\end{array}$$

Storage is divided into two parts:

- Factors
- Active memory



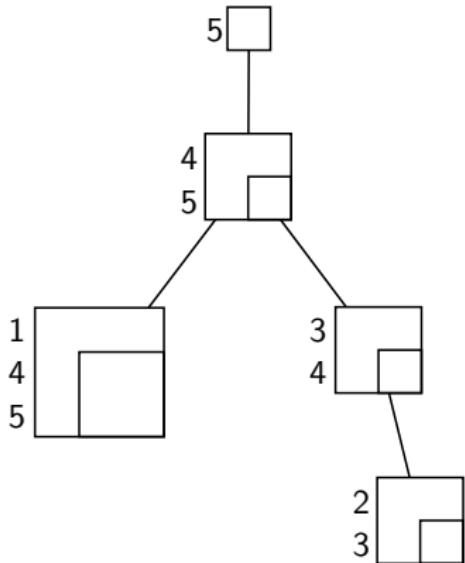
Elimination tree

The multifrontal method: memory handling

$$A = \begin{array}{|c|c|c|c|c|} \hline 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \text{shaded} & & & \\ \hline 2 & & \text{shaded} & & \\ \hline 3 & & & \text{shaded} & \\ \hline 4 & & & & \text{shaded} \\ \hline 5 & & & & \\ \hline \end{array} \rightarrow L+U-I = \begin{array}{|c|c|c|c|c|} \hline 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \text{shaded} & & & \\ \hline 2 & & \text{shaded} & & \\ \hline 3 & & & \text{shaded} & \\ \hline 4 & & & & \text{shaded} \\ \hline 5 & & & & \\ \hline \end{array}$$

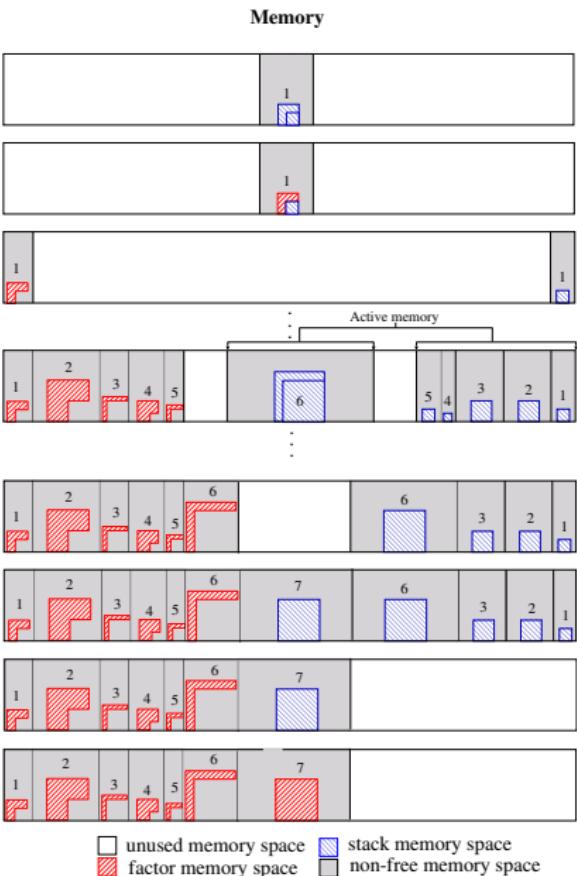
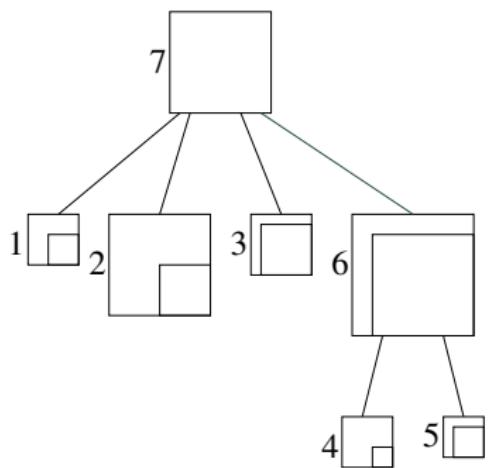
Storage is divided into two parts:

- Factors
- Active memory

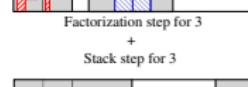
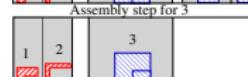
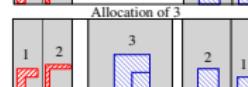
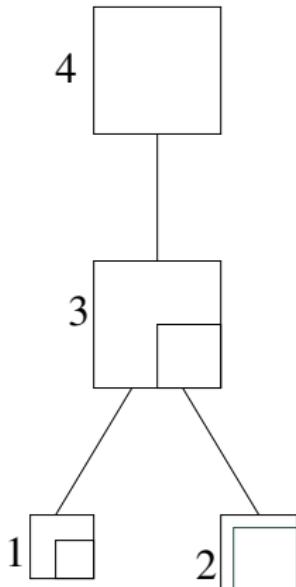


Elimination tree

Example 1: Processing a wide tree



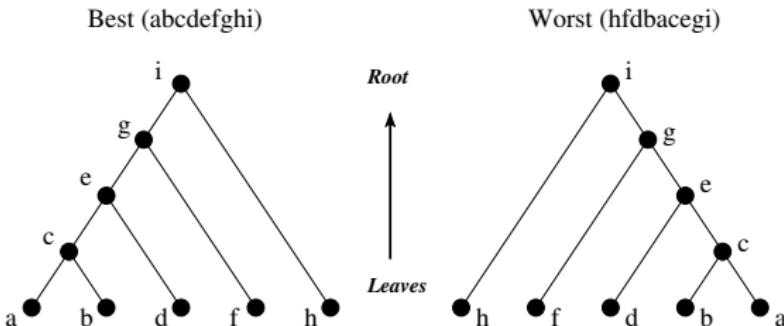
Example 2: Processing a deep tree



□ unused memory space ■ stack memory space
■ factor memory space ▨ non-free memory space

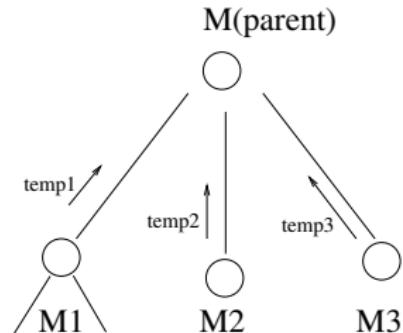
Postorder traversals: memory

Postorder provides a good data locality and better memory consumption than a general topological order since father nodes are assembled as soon as its children have been processed.
But there are still many postorders of the same tree. Which one to choose? the one that **minimizes memory consumption**



Problem model

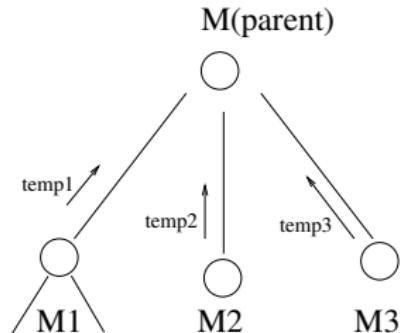
- M_i : memory peak for complete subtree rooted at i ,
- CB_i : (temporary) contribution block produced at node i ,
- m_i : memory for storing frontal matrix i ,
- nc_i : number of children of node i .



$$M_i = \max \left(\underbrace{\max_{j=1}^{nc_i} \left(M_j + \sum_{k=1}^{j-1} CB_k \right)}_{\text{children}}, \underbrace{m_i + \sum_{j=1}^{nc_i} CB_j}_{\text{assembly}} \right) \quad (1)$$

Problem model

- M_i : memory peak for complete subtree rooted at i ,
- CB_i : (temporary) contribution block produced at node i ,
- m_i : memory for storing frontal matrix i ,
- nc_i : number of children of node i .



$$M_i = \max \left(\underbrace{\max_{j=1}^{nc_i} \left(M_j + \sum_{k=1}^{j-1} CB_k \right)}_{\text{children}}, \underbrace{m_i + \sum_{j=1}^{nc_i} CB_j}_{\text{assembly}} \right) \quad (1)$$

Objective: order the children to minimize M_{parent}

Memory-minimizing schedules

Theorem 15.1 (Liu [15])

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

Corollary 15.1

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

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

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

Optimal tree reordering

Objective: Minimize peak of stack memory

Tree_Reorder (T):

```
for all  $i$  in the set of root nodes do
    Process_Node( $i$ );
end for
```

Process_Node(i):

```
if  $i$  is a leaf then
```

```
     $M_i = m_i$ 
```

```
else
```

```
    for  $j = 1$  to  $nc_i$  do
```

```
        Process_Node( $j^{th}$  child);
```

```
    end for
```

```
    Reorder the children of  $i$  in decreasing order of  $(M_j - CB_j)$ ;
```

```
    Compute  $M_i$  using Formula 1;
```

```
end if
```

Outline

Memory consumption in the sequential multifrontal method

- Equivalent orderings and postorder

- Memory-minimizing postorders

Memory consumption in the parallel multifrontal method

Memory consumption in parallel

In parallel, different memory regions scale in different ways:

- **Factor:** the factors produced upon factorization of the frontal matrices can be evenly distributed among the processors and, therefore, the associated memory scales perfectly, i.e., it does not increase globally and each process stores an equal share.
- **Active memory:** In parallel multiple branches have to be traversed at the same time (tree parallelism). This means that a higher number of CBs will have to be stored in memory which means that the global active memory increases.

In order to assess the memory scalability we will use a metric called **memory efficiency**:

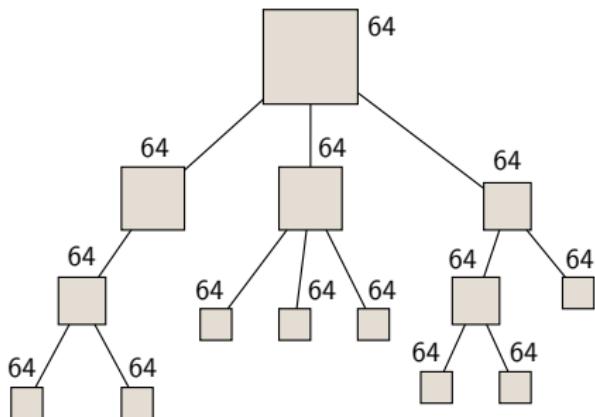
$$e(p) = \frac{S_{seq}}{p \times S_{max}(p)}$$

where we denote $S_{seq} = M_n$ the peak of memory consumption in a sequential execution and $S_{max}(p)$ the maximum peak memory consumption over all the p processes.

Ideally, we would like $e(p) \simeq 1$, i.e. S_{seq}/p on each processor.

Example 1: all-to-all mapping

All-to-all mapping: postorder traversal of the tree, where all the processors work at every node:

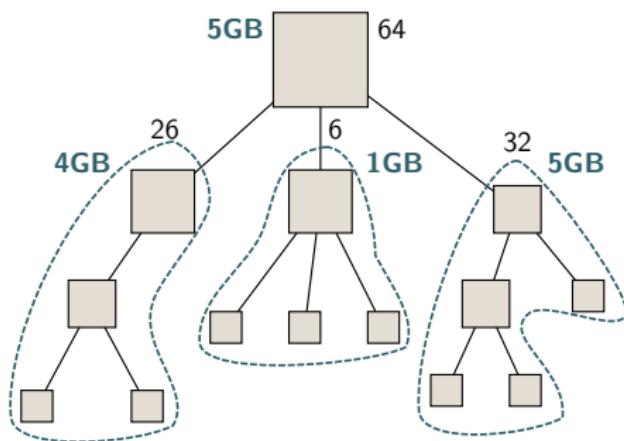


Optimal memory scalability ($S_{max}(p) = S_{seq}/p$) but no tree parallelism and prohibitive amounts of communications.

Example 2: proportional mapping

Proportional mapping: assuming that the sequential peak is 5 GB,

$$S_{max}(p) \geq \max \left\{ \frac{4 \text{ GB}}{26}, \frac{1 \text{ GB}}{6}, \frac{5 \text{ GB}}{32} \right\} = 0.16 \text{ GB} \Rightarrow e(p) \leq \frac{5}{64 \times 0.16} \leq 0.5$$



Memory efficiency bound

Let's compute a bound on the memory efficiency under the following assumptions:

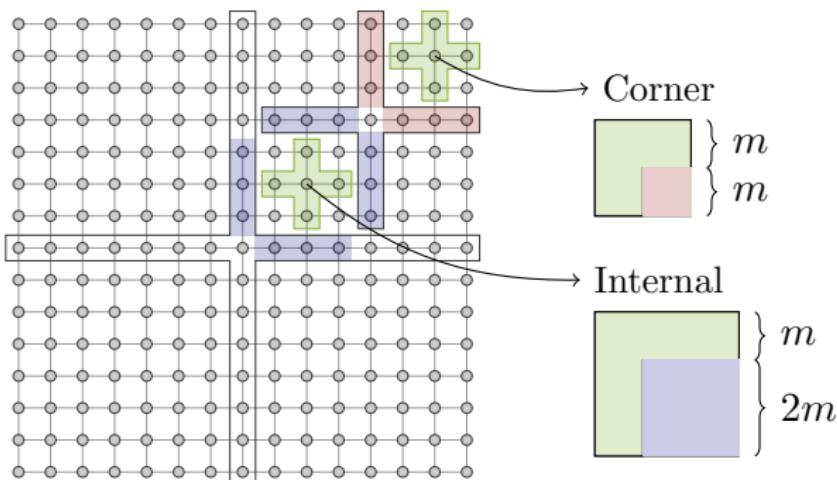
- Same assumptions as in Definition 10.1
- Proportional mapping is used
- Uniform tree (i.e., all siblings are equal)
- The tree has $\log_4(p)$ levels where p is the number of processes.

Memory efficiency bound

- Fronts and contribution blocks order at level l in the tree:

$$m_l = (3 - \alpha^l)^2 \frac{N^2}{2^{2l}} \quad CB_l = (2 - \alpha^l)^2 \frac{N^2}{2^{2l}}$$

with $\alpha < 1$. We use the notation m_l and CB_l to refer to the front and contribution block sizes for any node at level l



Memory efficiency bound

Theorem 16.1 (Peak of a subtree)

The peak memory consumption of a subtree with root at level l is achieved when assembling the root and is equal to:

$$M_l = \left((3 - \alpha^l)^2 + (2 - \alpha^{l+1})^2 \right) \frac{N^2}{2^{2l}} \quad (2)$$

Note that this implies that

$$S_{seq} = M_{l=0} = (4 + (2 - \alpha)^2)N^2$$

Memory efficiency bound

Remember: $M_i = \max \left(\max_{j=1}^{nc_i} \left(M_j + \sum_{k=1}^{j-1} CB_k \right), m_i + \sum_{j=1}^{nc_i} CB_j \right)$

Proof.

By induction. The first part of the theorem holds trivially for the leaf nodes. If equation 2 holds for a node at level $l+1$, then it also holds for a node at level l :

$$\begin{aligned} M_l &= \max \left\{ \begin{array}{l} m_l + 4CB_{l+1} \\ 3CB_{l+1} + M_{l+1} \end{array} \right. \\ &= \max \left\{ \begin{array}{l} (3 - \alpha^l)^2 \frac{N^2}{2^{2l}} + 4(2 - \alpha^{l+1})^2 \frac{N^2}{2^{2l+2}} \\ 3(2 - \alpha^{l+1})^2 \frac{N^2}{2^{2l+2}} + ((3 - \alpha^{l+1})^2 + (2 - \alpha^{l+2})^2) \frac{N^2}{2^{2l+2}} \end{array} \right. \\ &= ((3 - \alpha^l)^2 + (2 - \alpha^{l+1})^2) \frac{N^2}{2^{2l}} \end{aligned}$$

□

Memory efficiency bound

Under the assumption of proportional mapping, the number of processes per node is divided by $4 = 2^2$ at every level. Therefore

$$S_l(p) = \frac{M_l}{p/2^{2l}} = \frac{\left((3 - \alpha^l)^2 + (2 - \alpha^{l+1})^2\right) \frac{N^2}{2^{2l}}}{p/2^{2l}}$$

Because $S_l(p)$ is a monotonically increasing function of l and the maximum number of levels is $\log_4(p)$,

$$S_{max}(p) = \frac{\left((3 - \alpha^{\log_4(p)})^2 + (2 - \alpha^{\log_4(p)+1})^2\right) N^2}{p}$$

$$\lim_{p \rightarrow \infty} \frac{S_{seq}}{p \times S_{max}(p)} = \frac{(4 + (2 - \alpha)^2)}{13}$$

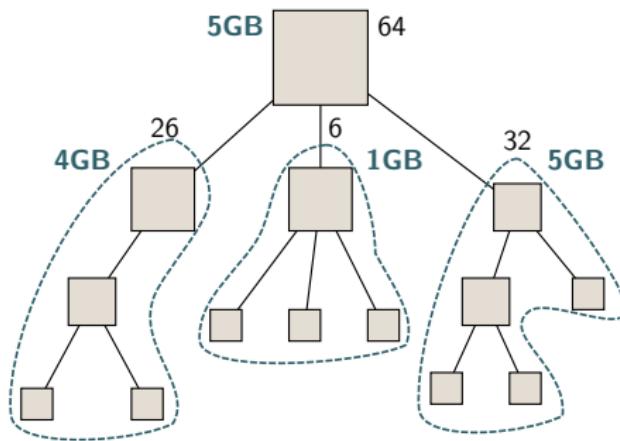
For example, for $\alpha = 0.5$ the efficiency is bounded by 0.48.

With more accurate assumptions (and many more computations) it is possible to show that actually $\lim_{p \rightarrow \infty} \frac{S_{seq}}{p \times S_{max}(p)} = 0$.

“memory-aware” mappings

“Memory-aware” mapping (Agullo et al. [1]): aims at enforcing a given memory constraint (M_0 , maximum memory per processor):

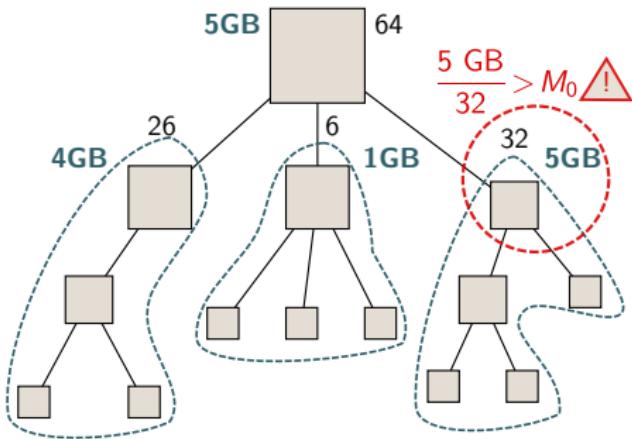
1. Try to apply proportional mapping.



“memory-aware” mappings

“Memory-aware” mapping (Agullo et al. [1]): aims at enforcing a given memory constraint (M_0 , maximum memory per processor):

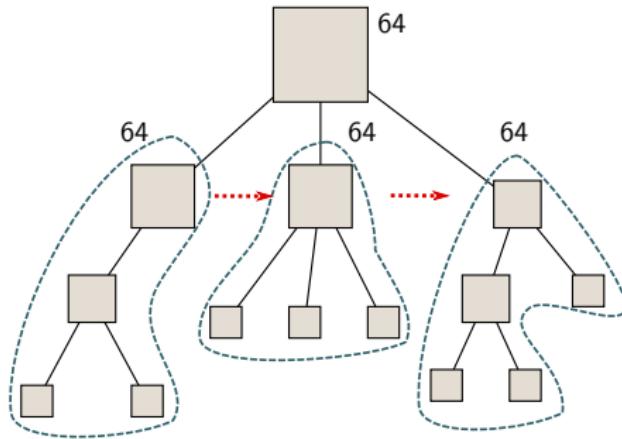
1. Try to apply proportional mapping.
2. Enough memory for each subtree?



“memory-aware” mappings

“Memory-aware” mapping (Agullo et al. [1]): aims at enforcing a given memory constraint (M_0 , maximum memory per processor):

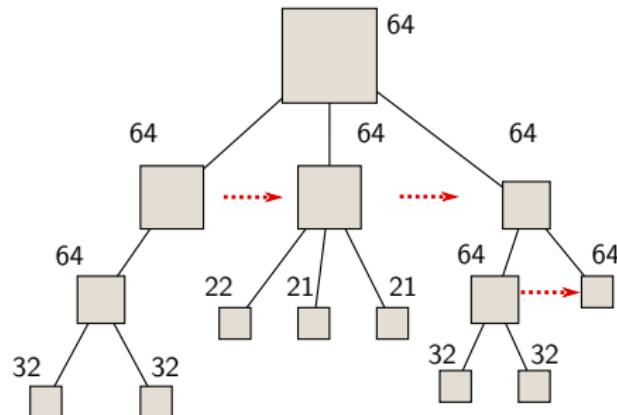
1. Try to apply proportional mapping.
2. Enough memory for each subtree? If not, serialize them, and update M_0 : processors stack equal shares of the CBs from previous nodes.



“memory-aware” mappings

“Memory-aware” mapping (Agullo et al. [1]): aims at enforcing a given memory constraint (M_0 , maximum memory per processor):

1. Try to apply proportional mapping.
2. Enough memory for each subtree?



- Ensures the given memory constraint and provides reliable estimates.
- Tends to assign many processors on nodes at the top of the tree
 \implies performance issues on parallel nodes.

“memory-aware” mappings

```
subroutine ma_mapping(r, p, M, S)
! r           : root of the subtree
! p(r)        : number of processes mapped on r
! M(r)        : memory constraint on r
! S(:)        : peak memory consumption for all nodes

call prop_mapping(r, p, M)

forall (i children of r)
    ! check if the constraint is respected
    if( S(i)/p(i) > M(i) ) then
        ! reject PM and revert to tree serialization
        call tree_serialization(r, p, M)
        exit          ! from forall block
    end if
end forall

forall (i children of r)
    ! apply MA mapping recursively to all siblings
    call ma_mapping(i, p, M, S)
end forall
end subroutine ma_mapping
```

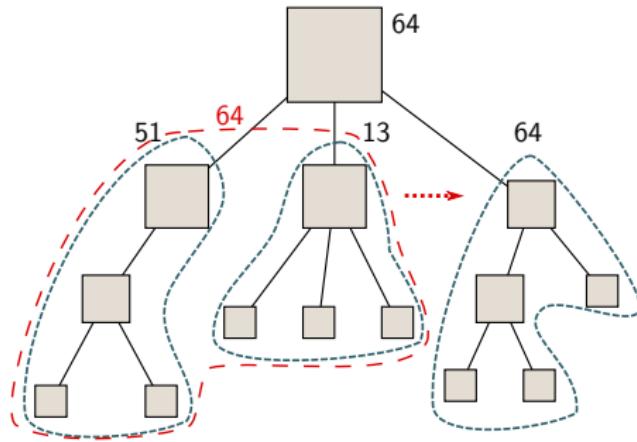
“memory-aware” mappings

```
subroutine prop_mapping(r, p, M)
  forall (i children of r)
    p(i)      = share of the p(r) processes from PM
    M(i)      = M(r)
  end forall
end subroutine prop_mapping

subroutine tree_serialization(r, p, M)
  stack_siblings = 0
  forall (i children of r) ! in appropriate order
    p(i) = p(r)
    ! update the memory constraint for the subtree
    M(i) = M(r) - stack_siblings
    stack_siblings = stack_siblings + cb(i)/p(r)
  end forall
end subroutine tree_serialization
```

“memory-aware” mappings

A finer “memory-aware” mapping? Serializing all the children at once is very constraining: more tree parallelism can be found.



Find groups on which proportional mapping works, and serialize these groups.

Heuristic: follow a given order (e.g. the serial postorder) and **form groups as large as possible**.

Experiments

- Matrix: finite-difference model of acoustic wave propagation, 27-point stencil, $192 \times 192 \times 192$ grid; **Seiscope consortium**.
 $N = 7$ M, $nnz = 189$ M, factors=152 GB (METIS).
Sequential peak of active memory: 46 GB.
- Machine: 64 nodes with two quad-core Xeon X5560 per node.
We use 256 MPI processes.
- Perfect memory scalability: **46 GB/ $256=180$ MB.**

Experiments

Prop Map	Memory-aware mapping			
	$M_0 = 225 \text{ MB}$		$M_0 = 380 \text{ MB}$	
	w/o groups	w/ groups		
Max stack peak (MB)	1932	227	330	381
Avg stack peak (MB)	626	122	177	228
Time (s)	1323	2690	2079	1779

- proportional mapping delivers the fastest factorization because it targets parallelism but, at the same time, it leads to a poor memory scaling with $e(256) = 0.09$
- the memory aware mapping with and without groups respects the imposed memory constraint with a speed that depends on how tight the constraint is
- grouping clearly provides a benefit in term of speed because it delivers more concurrency while still respecting the constraint

Low-rank approximations

Main principle

Principle: build approximated factorization $A_\varepsilon = L_\varepsilon U_\varepsilon$ at given accuracy ε

Part I: asymptotic complexity reduction

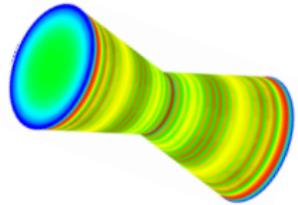
Theoretical proof and experimental validation that (3D case):

- Operations: $\mathcal{O}(N^6) \rightarrow \mathcal{O}(N^5) \rightarrow \mathcal{O}(N^4)$
- Memory: $\mathcal{O}(N^4) \rightarrow \mathcal{O}(N^3 \log N)$

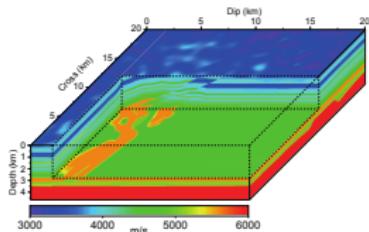
Part II: efficient and scalable algorithms

How to design algorithms to efficiently translate the theoretical complexity reduction into **actual performance and memory gains** for large-scale systems and applications?

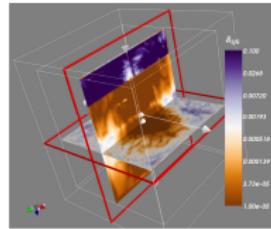
Impact on industrial applications



Structural mechanics
Matrix of order 8M
Required accuracy: 10^{-9}



Seismic imaging
Matrix of order 17M
Required accuracy: 10^{-3}



Electromagnetism
Matrix of order 30M
Required accuracy: 10^{-7}

Results on **900 cores**:

application	factorization time (s)			memory/proc (GB)		
	before	after	ratio	before	after	gain
structural	289.3	104.9	2.5	7.9	5.9	25%
seismic	617.0	123.4	4.9	13.3	10.4	22%
electromag.	1307.4	233.8	5.3	20.6	14.4	30%

Introduction

Rank and rank- k approximation

In the following, B is a dense matrix of size $m \times n$.

Definition 1 (Rank)

The rank k of B is defined as the smallest integer such that there exist matrices X and Y of size $m \times k$ and $n \times k$ such that

$$B = XY^T.$$

Definition 2

We call a rank- k approximation of B at accuracy ε any matrix \tilde{B} of rank k such that

$$\|B - \tilde{B}\| \leq \varepsilon.$$

Numerical rank

Theorem 3 (Eckart-Young)

Let $U\Sigma V^T$ be the SVD decomposition of B and let us note $\sigma_i = \Sigma_{i,i}$ its singular values. Then $\tilde{B} = U_{1:m,1:k}\Sigma_{1:k,1:k}V_{1:n,1:k}^T$ is the optimal rank- k approximation of B and

$$\|B - \tilde{B}\|_2 = \sigma_{k+1}.$$

Definition 4 (Numerical rank)

The numerical rank k_ε of B at accuracy ε is defined as the smallest integer such that there exists a matrix \tilde{B} of rank k_ε such that

$$\|B - \tilde{B}\| \leq \varepsilon.$$

Remark: in $\|\cdot\|_2$,

$$k_\varepsilon = \min_{1 \leq k \leq \min(m,n)} \sigma_{k+1} \leq \varepsilon.$$

Low-rank matrices

If $k_\varepsilon = \min(m, n)$ then B is said to be **full-rank**, otherwise it is **rank-deficient**. A class of rank-deficient matrices of particular interest are **low-rank** matrices.

Definition 5 (Low-rank matrix)

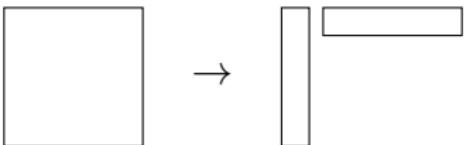
B is said to be low-rank (for a given accuracy ε) if its numerical rank k_ε is small enough such that its rank- k_ε approximation $\tilde{B} = XY^T$ requires less storage than the full-rank matrix B , i.e., if

$$k_\varepsilon(m + n) \leq mn.$$

In that case, \tilde{B} is said to be a **low-rank approximation** of B and ε is called the **low-rank threshold**.

In the following, for the sake of simplicity, we refer to the numerical rank of a matrix at accuracy ε simply as its “rank”.

Compression kernels



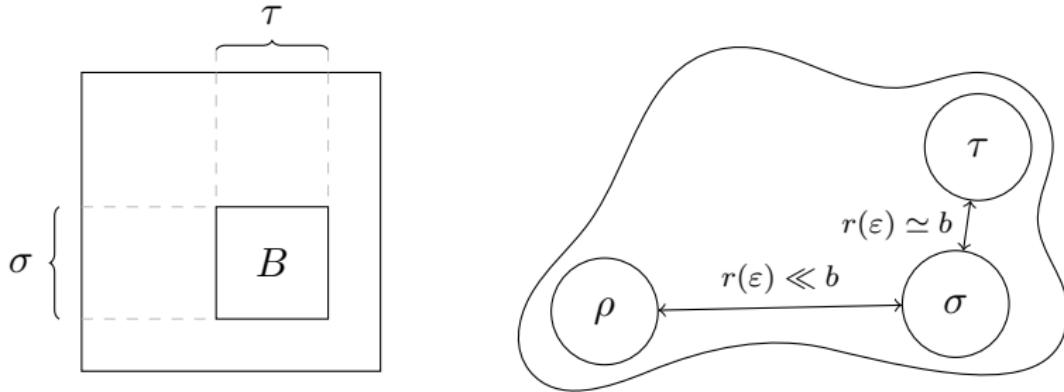
The act of computing \tilde{B} from B is called the **compression** of B .
What are the different methods to compress B ?

- SVD: optimal but expensive: $O(mn \min(m, n))$ operations
- Truncated QR factorization: slightly less accurate but much cheaper: $O(mnk_\varepsilon)$ operations \Rightarrow widely used
- Multiple other methods: randomized algorithms, adaptive cross-approximation, interpolative decomposition, CUR, etc.

In the following, we assume truncated QR is used as compression kernel.

Low-rank subblocks

Frontal matrices are not low-rank but in some applications they exhibit **low-rank blocks**



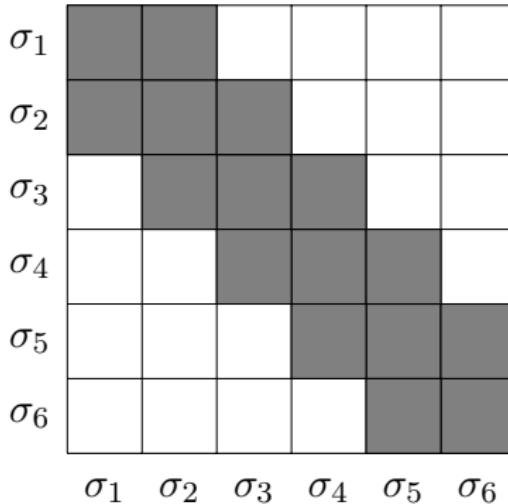
A block B represents the interaction between two subdomains σ and τ . If they have a **small diameter** and are **far away** their interaction is weak \Rightarrow rank is low.

The **block-admissibility condition** formalizes this intuition:

$$\sigma \times \tau \text{ is admissible} \Leftrightarrow \max(\text{diam}(\sigma), \text{diam}(\tau)) \leq \eta \text{dist}(\sigma, \tau)$$

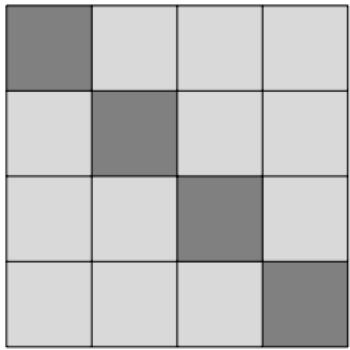
Block Low-Rank matrices

A BLR matrix is defined by a partition $\mathcal{P} = \mathfrak{S} \times \mathfrak{S}$, with $\mathfrak{S} = \{\sigma_1, \dots, \sigma_p\}$.



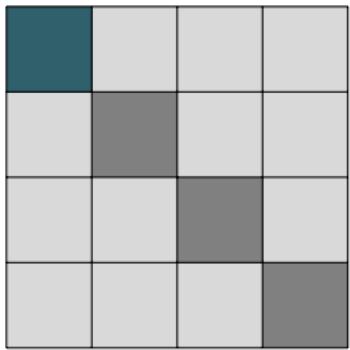
- Gray blocks are non-admissible and therefore kept full-rank
- White blocks are admissible and therefore compressed to low-rank

Standard BLR factorization: FSCU



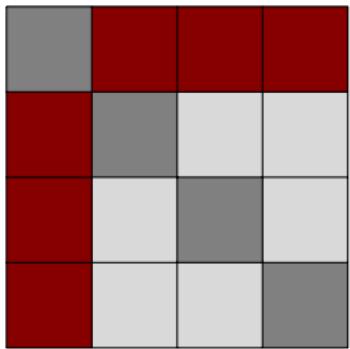
- FSCU

Standard BLR factorization: FSCU



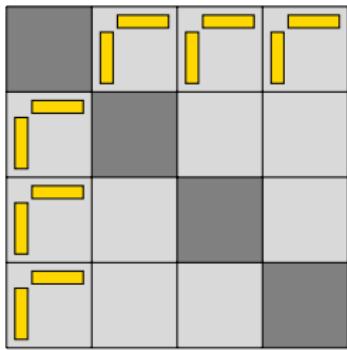
- FSCU (Factor,

Standard BLR factorization: FSCU



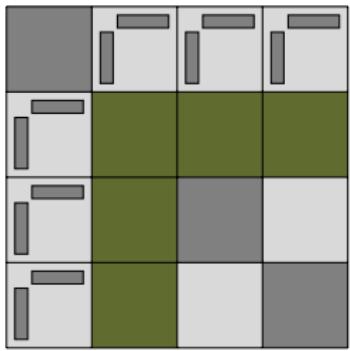
- FSCU (Factor, Solve,

Standard BLR factorization: FSCU



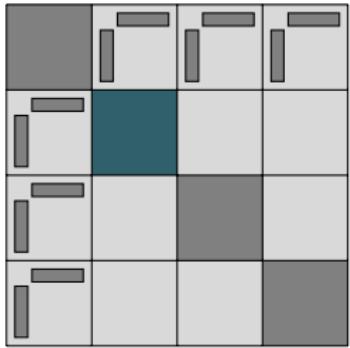
- FSCU (Factor, Solve, Compress,

Standard BLR factorization: FSCU



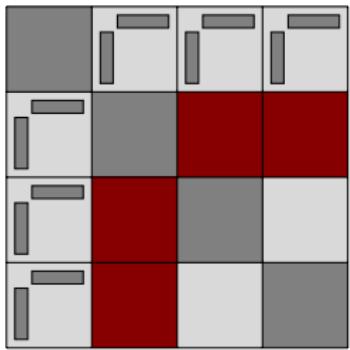
- FSCU (Factor, Solve, Compress, Update)

Standard BLR factorization: FSCU



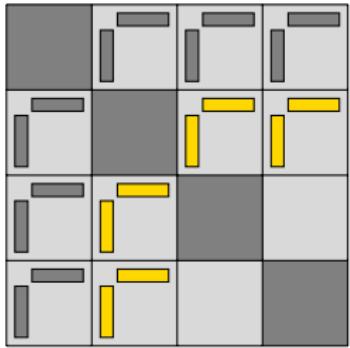
- FSCU (Factor, Solve, Compress, Update)

Standard BLR factorization: FSCU



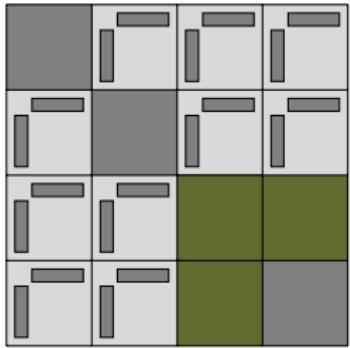
- FSCU (Factor, Solve, Compress, Update)

Standard BLR factorization: FSCU



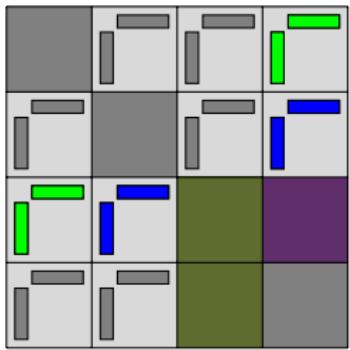
- FSCU (Factor, Solve, Compress, Update)

Standard BLR factorization: FSCU



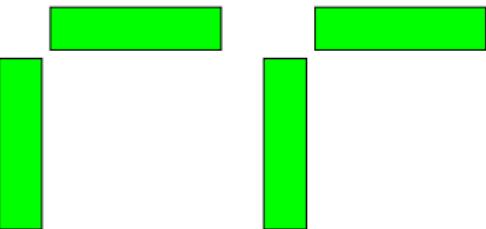
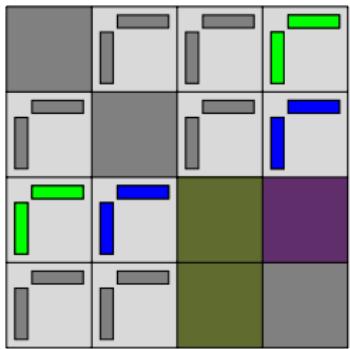
- FSCU (Factor, Solve, Compress, Update)

Standard BLR factorization: FSCU



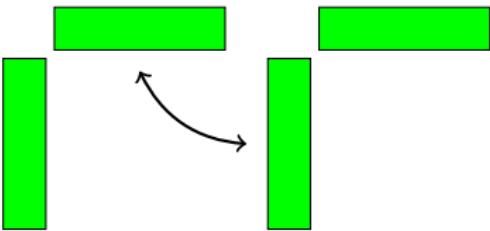
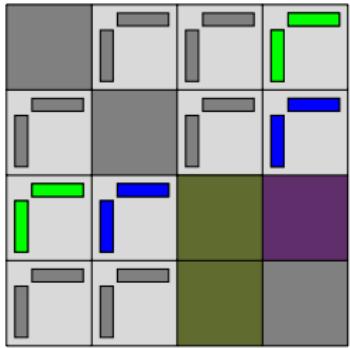
- FSCU (Factor, Solve, Compress, Update)

Standard BLR factorization: FSCU



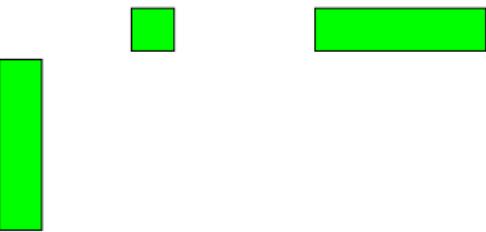
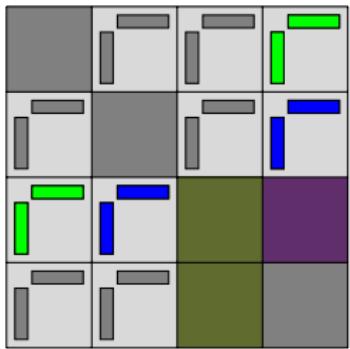
- FSCU (Factor, Solve, Compress, Update)

Standard BLR factorization: FSCU



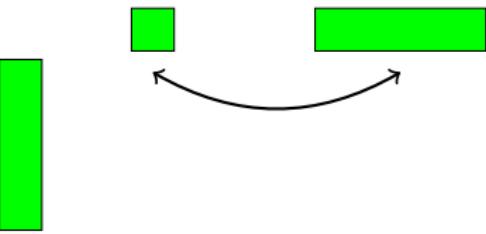
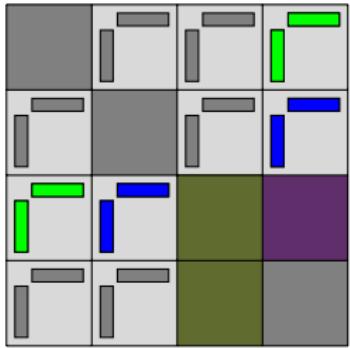
- FSCU (Factor, Solve, Compress, Update)

Standard BLR factorization: FSCU



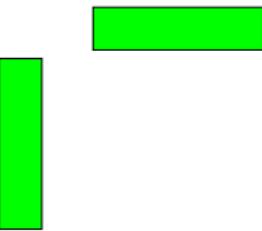
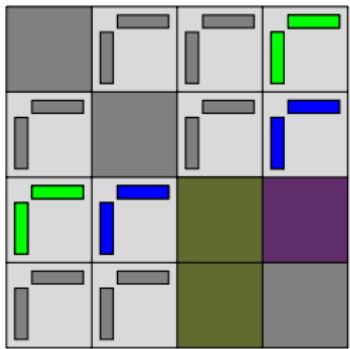
- FSCU (Factor, Solve, Compress, Update)

Standard BLR factorization: FSCU



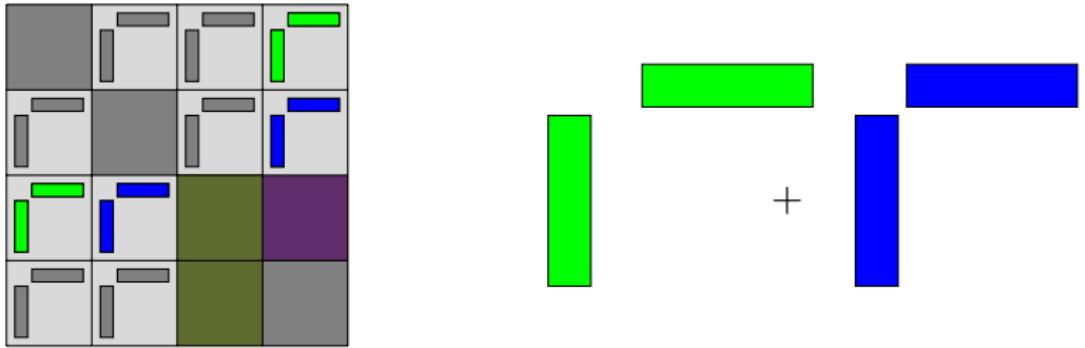
- FSCU (Factor, Solve, Compress, Update)

Standard BLR factorization: FSCU



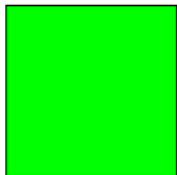
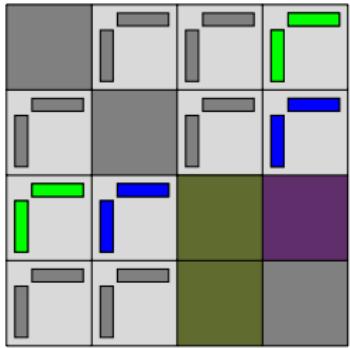
- FSCU (Factor, Solve, Compress, Update)

Standard BLR factorization: FSCU

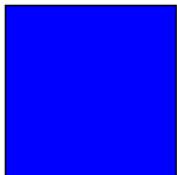


- FSCU (Factor, Solve, Compress, Update)

Standard BLR factorization: FSCU

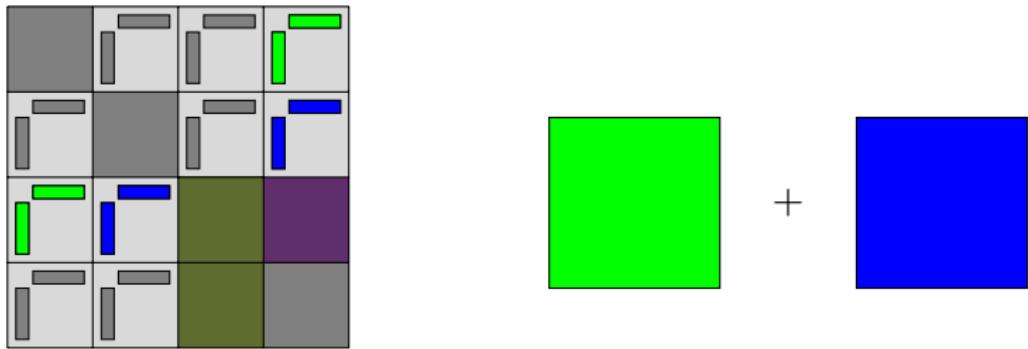


+



- FSCU (Factor, Solve, Compress, Update)

Standard BLR factorization: FSCU



- FSCU (Factor, Solve, Compress, Update)
- To evaluate the complexity of the BLR factorization, we must compute the cost of these four main steps

Part I: complexity of the BLR factorization

Cost analysis of the involved steps

Let us consider two blocks A and B of size $b \times b$ and of rank bounded by r .

step	type	operation	cost
Factor	FR	$A \leftarrow LU$	
Solve	FR-FR	$B \leftarrow BU^{-1}$	
Compress	LR	$A \leftarrow \tilde{A}$	
Update	FR-FR	$C \leftarrow AB$	
Update	LR-FR	$C \leftarrow \tilde{A}B$	
Update	FR-LR	$C \leftarrow A\tilde{B}$	
Update	LR-LR	$C \leftarrow \tilde{A}\tilde{B}$	

Cost analysis of the involved steps

Let us consider two blocks A and B of size $b \times b$ and of rank bounded by r .

step	type	operation	cost
Factor	FR	$A \leftarrow LU$	$\mathcal{O}(b^3)$
Solve	FR-FR	$B \leftarrow BU^{-1}$	
Compress	LR	$A \leftarrow \tilde{A}$	
Update	FR-FR	$C \leftarrow AB$	
Update	LR-FR	$C \leftarrow \tilde{A}B$	
Update	FR-LR	$C \leftarrow A\tilde{B}$	
Update	LR-LR	$C \leftarrow \tilde{A}\tilde{B}$	

Cost analysis of the involved steps

Let us consider two blocks A and B of size $b \times b$ and of rank bounded by r .

step	type	operation	cost
Factor	FR	$A \leftarrow LU$	$\mathcal{O}(b^3)$
Solve	FR-FR	$B \leftarrow BU^{-1}$	$\mathcal{O}(b^3)$
Compress	LR	$A \leftarrow \tilde{A}$	
Update	FR-FR	$C \leftarrow AB$	
Update	LR-FR	$C \leftarrow \tilde{A}B$	
Update	FR-LR	$C \leftarrow A\tilde{B}$	
Update	LR-LR	$C \leftarrow \tilde{A}\tilde{B}$	

Cost analysis of the involved steps

Let us consider two blocks A and B of size $b \times b$ and of rank bounded by r .

step	type	operation	cost
Factor	FR	$A \leftarrow LU$	$\mathcal{O}(b^3)$
Solve	FR-FR	$B \leftarrow BU^{-1}$	$\mathcal{O}(b^3)$
Compress	LR	$A \leftarrow \tilde{A}$	$\mathcal{O}(b^2r)$
Update	FR-FR	$C \leftarrow AB$	
Update	LR-FR	$C \leftarrow \tilde{A}B$	
Update	FR-LR	$C \leftarrow A\tilde{B}$	
Update	LR-LR	$C \leftarrow \tilde{A}\tilde{B}$	

Cost analysis of the involved steps

Let us consider two blocks A and B of size $b \times b$ and of rank bounded by r .

step	type	operation	cost
Factor	FR	$A \leftarrow LU$	$\mathcal{O}(b^3)$
Solve	FR-FR	$B \leftarrow BU^{-1}$	$\mathcal{O}(b^3)$
Compress	LR	$A \leftarrow \tilde{A}$	$\mathcal{O}(b^2r)$
Update	FR-FR	$C \leftarrow AB$	$\mathcal{O}(b^3)$
Update	LR-FR	$C \leftarrow \tilde{A}B$	
Update	FR-LR	$C \leftarrow A\tilde{B}$	
Update	LR-LR	$C \leftarrow \tilde{A}\tilde{B}$	

Cost analysis of the involved steps

Let us consider two blocks A and B of size $b \times b$ and of rank bounded by r .

step	type	operation	cost
Factor	FR	$A \leftarrow LU$	$\mathcal{O}(b^3)$
Solve	FR-FR	$B \leftarrow BU^{-1}$	$\mathcal{O}(b^3)$
Compress	LR	$A \leftarrow \tilde{A}$	$\mathcal{O}(b^2r)$
Update	FR-FR	$C \leftarrow AB$	$\mathcal{O}(b^3)$
Update	LR-FR	$C \leftarrow \tilde{A}B$	$\mathcal{O}(b^2r)$
Update	FR-LR	$C \leftarrow A\tilde{B}$	
Update	LR-LR	$C \leftarrow \tilde{A}\tilde{B}$	

Cost analysis of the involved steps

Let us consider two blocks A and B of size $b \times b$ and of rank bounded by r .

step	type	operation	cost
Factor	FR	$A \leftarrow LU$	$\mathcal{O}(b^3)$
Solve	FR-FR	$B \leftarrow BU^{-1}$	$\mathcal{O}(b^3)$
Compress	LR	$A \leftarrow \tilde{A}$	$\mathcal{O}(b^2r)$
Update	FR-FR	$C \leftarrow AB$	$\mathcal{O}(b^3)$
Update	LR-FR	$C \leftarrow \tilde{A}B$	$\mathcal{O}(b^2r)$
Update	FR-LR	$C \leftarrow A\tilde{B}$	$\mathcal{O}(b^2r)$
Update	LR-LR	$C \leftarrow \tilde{A}\tilde{B}$	

Cost analysis of the involved steps

Let us consider two blocks A and B of size $b \times b$ and of rank bounded by r .

step	type	operation	cost
Factor	FR	$A \leftarrow LU$	$\mathcal{O}(b^3)$
Solve	FR-FR	$B \leftarrow BU^{-1}$	$\mathcal{O}(b^3)$
Compress	LR	$A \leftarrow \tilde{A}$	$\mathcal{O}(b^2r)$
Update	FR-FR	$C \leftarrow AB$	$\mathcal{O}(b^3)$
Update	LR-FR	$C \leftarrow \tilde{A}B$	$\mathcal{O}(b^2r)$
Update	FR-LR	$C \leftarrow A\tilde{B}$	$\mathcal{O}(b^2r)$
Update	LR-LR	$C \leftarrow \tilde{A}\tilde{B}$	$\mathcal{O}(b^2r)$

Cost analysis of the involved steps

Let us consider two blocks A and B of size $b \times b$ and of rank bounded by r .

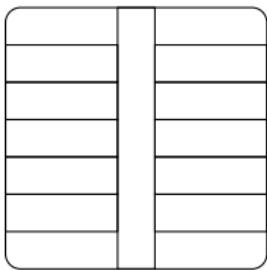
step	type	operation	cost
Factor	FR	$A \leftarrow LU$	$\mathcal{O}(b^3)$
Solve	FR-FR	$B \leftarrow BU^{-1}$	$\mathcal{O}(b^3)$
Compress	LR	$A \leftarrow \tilde{A}$	$\mathcal{O}(b^2r)$
Update	FR-FR	$C \leftarrow AB$	$\mathcal{O}(b^3)$
Update	LR-FR	$C \leftarrow \tilde{A}B$	$\mathcal{O}(b^2r)$
Update	FR-LR	$C \leftarrow A\tilde{B}$	$\mathcal{O}(b^2r)$
Update	LR-LR	$C \leftarrow \tilde{A}\tilde{B}$	$\mathcal{O}(b^2r)$

This is not enough to compute the complexity \Rightarrow we need to bound the number of FR blocks!

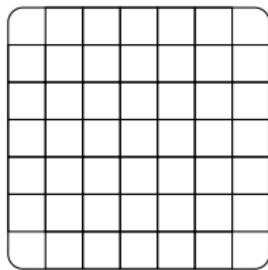
Bounding the number of FR blocks

BLR-admissibility condition of a partition \mathcal{P}

$$\mathcal{P} \text{ is admissible} \Leftrightarrow \begin{cases} \#\{\sigma, \sigma \times \tau \in \mathcal{P} \text{ is not admissible}\} \leq q \\ \#\{\tau, \sigma \times \tau \in \mathcal{P} \text{ is not admissible}\} \leq q \end{cases}$$



Non-Admissible

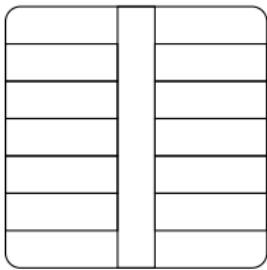


Admissible

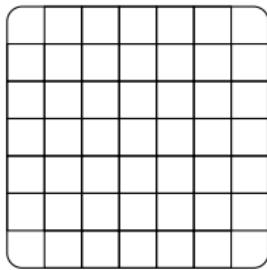
Bounding the number of FR blocks

BLR-admissibility condition of a partition \mathcal{P}

$$\mathcal{P} \text{ is admissible} \Leftrightarrow \begin{cases} \#\{\sigma, \sigma \times \tau \in \mathcal{P} \text{ is not admissible}\} \leq q \\ \#\{\tau, \sigma \times \tau \in \mathcal{P} \text{ is not admissible}\} \leq q \end{cases}$$



Non-Admissible



Admissible

Key result

For **any matrix**, we can build an admissible \mathcal{P} for $q = \mathcal{O}(1)$, s.t. the maximal rank of the admissible blocks of A is r

Memory complexity of the dense BLR factorization

Let us consider a dense (frontal) matrix of order m divided into $p \times p$ blocks of order b , with $p = m/b$.

The memory complexity to store the matrix can be computed as
 $\mathcal{M}_{total}(b, p, r) = \mathcal{M}_{FR}(b, p) + \mathcal{M}_{LR}(b, p, r)$

Memory complexity of the dense BLR factorization

Let us consider a dense (frontal) matrix of order m divided into $p \times p$ blocks of order b , with $p = m/b$.

The memory complexity to store the matrix can be computed as

$$\mathcal{M}_{total}(b, p, r) = \mathcal{M}_{FR}(b, p) + \mathcal{M}_{LR}(b, p, r)$$

- Ex. 1: compute

$$\mathcal{M}_{FR}(b, p) =$$

$$\mathcal{M}_{LR}(b, p, r) =$$

Memory complexity of the dense BLR factorization

Let us consider a dense (frontal) matrix of order m divided into $p \times p$ blocks of order b , with $p = m/b$.

The memory complexity to store the matrix can be computed as

$$\mathcal{M}_{total}(b, p, r) = \mathcal{M}_{FR}(b, p) + \mathcal{M}_{LR}(b, p, r)$$

- Ex. 1: compute

$$\mathcal{M}_{FR}(b, p) = \mathcal{O}(pb^2)$$

$$\mathcal{M}_{LR}(b, p, r) =$$

Memory complexity of the dense BLR factorization

Let us consider a dense (frontal) matrix of order m divided into $p \times p$ blocks of order b , with $p = m/b$.

The memory complexity to store the matrix can be computed as

$$\mathcal{M}_{total}(b, p, r) = \mathcal{M}_{FR}(b, p) + \mathcal{M}_{LR}(b, p, r)$$

- Ex. 1: compute

$$\mathcal{M}_{FR}(b, p) = \mathcal{O}(pb^2)$$

$$\mathcal{M}_{LR}(b, p, r) = \mathcal{O}(p^2br)$$

Memory complexity of the dense BLR factorization

Let us consider a dense (frontal) matrix of order m divided into $p \times p$ blocks of order b , with $p = m/b$.

The memory complexity to store the matrix can be computed as

$$\mathcal{M}_{total}(b, p, r) = \mathcal{M}_{FR}(b, p) + \mathcal{M}_{LR}(b, p, r)$$

- Ex. 1: compute

$$\mathcal{M}_{FR}(b, p) = \mathcal{O}(pb^2)$$

$$\mathcal{M}_{LR}(b, p, r) = \mathcal{O}(p^2br)$$

- Ex. 2: assuming $b = \mathcal{O}(m^x)$ and $r = \mathcal{O}(m^\alpha)$, compute

$$\mathcal{M}_{total}(m, x, \alpha) =$$

Memory complexity of the dense BLR factorization

Let us consider a dense (frontal) matrix of order m divided into $p \times p$ blocks of order b , with $p = m/b$.

The memory complexity to store the matrix can be computed as
 $\mathcal{M}_{total}(b, p, r) = \mathcal{M}_{FR}(b, p) + \mathcal{M}_{LR}(b, p, r)$

- Ex. 1: compute

$$\mathcal{M}_{FR}(b, p) = \mathcal{O}(pb^2)$$

$$\mathcal{M}_{LR}(b, p, r) = \mathcal{O}(p^2 br)$$

- Ex. 2: assuming $b = \mathcal{O}(m^x)$ and $r = \mathcal{O}(m^\alpha)$, compute

$$\mathcal{M}_{total}(m, x, \alpha) = \mathcal{O}(m^{1+x} + m^{2-x+\alpha})$$

Memory complexity of the dense BLR factorization

Let us consider a dense (frontal) matrix of order m divided into $p \times p$ blocks of order b , with $p = m/b$.

The memory complexity to store the matrix can be computed as
 $\mathcal{M}_{total}(b, p, r) = \mathcal{M}_{FR}(b, p) + \mathcal{M}_{LR}(b, p, r)$

- Ex. 1: compute

$$\mathcal{M}_{FR}(b, p) = \mathcal{O}(pb^2)$$

$$\mathcal{M}_{LR}(b, p, r) = \mathcal{O}(p^2 br)$$

- Ex. 2: assuming $b = \mathcal{O}(m^x)$ and $r = \mathcal{O}(m^\alpha)$, compute

$$\mathcal{M}_{total}(m, x, \alpha) = \mathcal{O}(m^{1+x} + m^{2-x+\alpha})$$

- Ex. 3: compute the optimal block size $b^* = \mathcal{O}(m^{x^*})$ and the resulting optimal complexity: $x^* = \dots$, $b^* = \dots$, and

$$\mathcal{M}_{opt}(m, r) = \mathcal{M}_{total}(m, x^*, \alpha) =$$

Memory complexity of the dense BLR factorization

Let us consider a dense (frontal) matrix of order m divided into $p \times p$ blocks of order b , with $p = m/b$.

The memory complexity to store the matrix can be computed as
 $\mathcal{M}_{total}(b, p, r) = \mathcal{M}_{FR}(b, p) + \mathcal{M}_{LR}(b, p, r)$

- Ex. 1: compute

$$\mathcal{M}_{FR}(b, p) = \mathcal{O}(pb^2)$$

$$\mathcal{M}_{LR}(b, p, r) = \mathcal{O}(p^2 br)$$

- Ex. 2: assuming $b = \mathcal{O}(m^x)$ and $r = \mathcal{O}(m^\alpha)$, compute

$$\mathcal{M}_{total}(m, x, \alpha) = \mathcal{O}(m^{1+x} + m^{2-x+\alpha})$$

- Ex. 3: compute the optimal block size $b^* = \mathcal{O}(m^{x^*})$ and the resulting optimal complexity: $x^* = (1 + \alpha)/2$, $b^* = m^{(1+\alpha)/2}$, and

$$\mathcal{M}_{opt}(m, r) = \mathcal{M}_{total}(m, x^*, \alpha) =$$

Memory complexity of the dense BLR factorization

Let us consider a dense (frontal) matrix of order m divided into $p \times p$ blocks of order b , with $p = m/b$.

The memory complexity to store the matrix can be computed as
 $\mathcal{M}_{total}(b, p, r) = \mathcal{M}_{FR}(b, p) + \mathcal{M}_{LR}(b, p, r)$

- Ex. 1: compute

$$\mathcal{M}_{FR}(b, p) = \mathcal{O}(pb^2)$$
$$\mathcal{M}_{LR}(b, p, r) = \mathcal{O}(p^2br)$$

- Ex. 2: assuming $b = \mathcal{O}(m^x)$ and $r = \mathcal{O}(m^\alpha)$, compute

$$\mathcal{M}_{total}(m, x, \alpha) = \mathcal{O}(m^{1+x} + m^{2-x+\alpha})$$

- Ex. 3: compute the optimal block size $b^* = \mathcal{O}(m^{x^*})$ and the resulting optimal complexity: $x^* = (1 + \alpha)/2$, $b^* = m^{(1+\alpha)/2}$, and

$$\mathcal{M}_{opt}(m, r) = \mathcal{M}_{total}(m, x^*, \alpha) = \mathcal{O}\left(m^{3/2}r^{1/2}\right)$$

Flop complexity of the dense BLR factorization

Let us consider a dense (frontal) matrix of order m divided into $p \times p$ blocks of order b , with $p = m/b$.

step	type	cost	number	$\mathcal{C}_{step}(b, p, r)$
Factor	FR	$\mathcal{O}(b^3)$		
Solve	FR-FR	$\mathcal{O}(b^3)$		
Compress	LR	$\mathcal{O}(b^2r)$		
Update	FR-FR	$\mathcal{O}(b^3)$		
Update	LR-FR	$\mathcal{O}(b^2r)$		
Update	LR-LR	$\mathcal{O}(b^2r)$		

- Ex. 1: compute $\mathcal{C}_{step}(b, p, r) = \text{cost} \times \text{number}$

Flop complexity of the dense BLR factorization

Let us consider a dense (frontal) matrix of order m divided into $p \times p$ blocks of order b , with $p = m/b$.

step	type	cost	number	$\mathcal{C}_{step}(b, p, r)$
Factor	FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p)$	$\mathcal{O}(pb^3)$
Solve	FR-FR	$\mathcal{O}(b^3)$		
Compress	LR	$\mathcal{O}(b^2r)$		
Update	FR-FR	$\mathcal{O}(b^3)$		
Update	LR-FR	$\mathcal{O}(b^2r)$		
Update	LR-LR	$\mathcal{O}(b^2r)$		

- Ex. 1: compute $\mathcal{C}_{step}(b, p, r) = \text{cost} \times \text{number}$

Flop complexity of the dense BLR factorization

Let us consider a dense (frontal) matrix of order m divided into $p \times p$ blocks of order b , with $p = m/b$.

step	type	cost	number	$\mathcal{C}_{step}(b, p, r)$
Factor	FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p)$	$\mathcal{O}(pb^3)$
Solve	FR-FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^3)$
Compress	LR	$\mathcal{O}(b^2r)$		
Update	FR-FR	$\mathcal{O}(b^3)$		
Update	LR-FR	$\mathcal{O}(b^2r)$		
Update	LR-LR	$\mathcal{O}(b^2r)$		

- Ex. 1: compute $\mathcal{C}_{step}(b, p, r) = \text{cost} \times \text{number}$

Flop complexity of the dense BLR factorization

Let us consider a dense (frontal) matrix of order m divided into $p \times p$ blocks of order b , with $p = m/b$.

step	type	cost	number	$\mathcal{C}_{step}(b, p, r)$
Factor	FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p)$	$\mathcal{O}(pb^3)$
Solve	FR-FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^3)$
Compress	LR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^2r)$
Update	FR-FR	$\mathcal{O}(b^3)$		
Update	LR-FR	$\mathcal{O}(b^2r)$		
Update	LR-LR	$\mathcal{O}(b^2r)$		

- Ex. 1: compute $\mathcal{C}_{step}(b, p, r) = \text{cost} \times \text{number}$

Flop complexity of the dense BLR factorization

Let us consider a dense (frontal) matrix of order m divided into $p \times p$ blocks of order b , with $p = m/b$.

step	type	cost	number	$\mathcal{C}_{step}(b, p, r)$
Factor	FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p)$	$\mathcal{O}(pb^3)$
Solve	FR-FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^3)$
Compress	LR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^2r)$
Update	FR-FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p)$	$\mathcal{O}(pb^3)$
Update	LR-FR	$\mathcal{O}(b^2r)$		
Update	LR-LR	$\mathcal{O}(b^2r)$		

- Ex. 1: compute $\mathcal{C}_{step}(b, p, r) = \text{cost} \times \text{number}$

Flop complexity of the dense BLR factorization

Let us consider a dense (frontal) matrix of order m divided into $p \times p$ blocks of order b , with $p = m/b$.

step	type	cost	number	$\mathcal{C}_{step}(b, p, r)$
Factor	FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p)$	$\mathcal{O}(pb^3)$
Solve	FR-FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^3)$
Compress	LR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^2r)$
Update	FR-FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p)$	$\mathcal{O}(pb^3)$
Update	LR-FR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^2r)$
Update	LR-LR	$\mathcal{O}(b^2r)$		

- Ex. 1: compute $\mathcal{C}_{step}(b, p, r) = \text{cost} \times \text{number}$

Flop complexity of the dense BLR factorization

Let us consider a dense (frontal) matrix of order m divided into $p \times p$ blocks of order b , with $p = m/b$.

step	type	cost	number	$\mathcal{C}_{step}(b, p, r)$
Factor	FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p)$	$\mathcal{O}(pb^3)$
Solve	FR-FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^3)$
Compress	LR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^2r)$
Update	FR-FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p)$	$\mathcal{O}(pb^3)$
Update	LR-FR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^2r)$
Update	LR-LR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^3)$	$\mathcal{O}(p^3b^2r)$

- Ex. 1: compute $\mathcal{C}_{step}(b, p, r) = \text{cost} \times \text{number}$

Flop complexity of the dense BLR factorization

Let us consider a dense (frontal) matrix of order m divided into $p \times p$ blocks of order b , with $p = m/b$.

step	type	cost	number	$\mathcal{C}_{step}(b, p, r)$	$\mathcal{C}_{step}(m, x, \alpha)$
Factor	FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p)$	$\mathcal{O}(pb^3)$	
Solve	FR-FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^3)$	
Compress	LR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^2r)$	
Update	FR-FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p)$	$\mathcal{O}(pb^3)$	
Update	LR-FR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^2r)$	
Update	LR-LR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^3)$	$\mathcal{O}(p^3b^2r)$	

- Ex. 1: compute $\mathcal{C}_{step}(b, p, r) = \text{cost} \times \text{number}$
- Ex. 2: compute $\mathcal{C}_{step}(m, x, \alpha)$ with $b = \mathcal{O}(m^x)$ and $r = \mathcal{O}(m^\alpha)$.

Flop complexity of the dense BLR factorization

Let us consider a dense (frontal) matrix of order m divided into $p \times p$ blocks of order b , with $p = m/b$.

step	type	cost	number	$\mathcal{C}_{step}(b, p, r)$	$\mathcal{C}_{step}(m, x, \alpha)$
Factor	FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p)$	$\mathcal{O}(pb^3)$	$\mathcal{O}(m^{1+2x})$
Solve	FR-FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^3)$	$\mathcal{O}(m^{2+x})$
Compress	LR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^2r)$	$\mathcal{O}(m^{2+\alpha})$
Update	FR-FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p)$	$\mathcal{O}(pb^3)$	$\mathcal{O}(m^{1+2x})$
Update	LR-FR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^2r)$	$\mathcal{O}(m^{2+\alpha})$
Update	LR-LR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^3)$	$\mathcal{O}(p^3b^2r)$	$\mathcal{O}(m^{3-x+\alpha})$

- Ex. 1: compute $\mathcal{C}_{step}(b, p, r) = \text{cost} \times \text{number}$
- Ex. 2: compute $\mathcal{C}_{step}(m, x, \alpha)$ with $b = \mathcal{O}(m^x)$ and $r = \mathcal{O}(m^\alpha)$.

Flop complexity of the dense BLR factorization

Let us consider a dense (frontal) matrix of order m divided into $p \times p$ blocks of order b , with $p = m/b$.

step	type	cost	number	$\mathcal{C}_{step}(b, p, r)$	$\mathcal{C}_{step}(m, x, \alpha)$
Factor	FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p)$	$\mathcal{O}(pb^3)$	$\mathcal{O}(m^{1+2x})$
Solve	FR-FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^3)$	$\mathcal{O}(m^{2+x})$
Compress	LR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^2r)$	$\mathcal{O}(m^{2+\alpha})$
Update	FR-FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p)$	$\mathcal{O}(pb^3)$	$\mathcal{O}(m^{1+2x})$
Update	LR-FR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^2r)$	$\mathcal{O}(m^{2+\alpha})$
Update	LR-LR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^3)$	$\mathcal{O}(p^3b^2r)$	$\mathcal{O}(m^{3-x+\alpha})$

- Ex. 1: compute $\mathcal{C}_{step}(b, p, r) = \text{cost} \times \text{number}$
- Ex. 2: compute $\mathcal{C}_{step}(m, x, \alpha)$ with $b = \mathcal{O}(m^x)$ and $r = \mathcal{O}(m^\alpha)$.
- Ex. 3: compute the total complexity (sum of all steps)

$$\mathcal{C}_{total}(m, x, \alpha) =$$

Flop complexity of the dense BLR factorization

Let us consider a dense (frontal) matrix of order m divided into $p \times p$ blocks of order b , with $p = m/b$.

step	type	cost	number	$\mathcal{C}_{step}(b, p, r)$	$\mathcal{C}_{step}(m, x, \alpha)$
Factor	FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p)$	$\mathcal{O}(pb^3)$	$\mathcal{O}(m^{1+2x})$
Solve	FR-FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^3)$	$\mathcal{O}(m^{2+x})$
Compress	LR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^2r)$	$\mathcal{O}(m^{2+\alpha})$
Update	FR-FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p)$	$\mathcal{O}(pb^3)$	$\mathcal{O}(m^{1+2x})$
Update	LR-FR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^2r)$	$\mathcal{O}(m^{2+\alpha})$
Update	LR-LR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^3)$	$\mathcal{O}(p^3b^2r)$	$\mathcal{O}(m^{3-x+\alpha})$

- Ex. 1: compute $\mathcal{C}_{step}(b, p, r) = \text{cost} \times \text{number}$
- Ex. 2: compute $\mathcal{C}_{step}(m, x, \alpha)$ with $b = \mathcal{O}(m^x)$ and $r = \mathcal{O}(m^\alpha)$.
- Ex. 3: compute the total complexity (sum of all steps)

$$\mathcal{C}_{total}(m, x, \alpha) = \mathcal{O}(m^{3-x+\alpha} + m^{2+x})$$

Flop complexity of the dense BLR factorization

Let us consider a dense (frontal) matrix of order m divided into $p \times p$ blocks of order b , with $p = m/b$.

step	type	cost	number	$\mathcal{C}_{step}(b, p, r)$	$\mathcal{C}_{step}(m, x, \alpha)$
Factor	FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p)$	$\mathcal{O}(pb^3)$	$\mathcal{O}(m^{1+2x})$
Solve	FR-FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^3)$	$\mathcal{O}(m^{2+x})$
Compress	LR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^2r)$	$\mathcal{O}(m^{2+\alpha})$
Update	FR-FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p)$	$\mathcal{O}(pb^3)$	$\mathcal{O}(m^{1+2x})$
Update	LR-FR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^2r)$	$\mathcal{O}(m^{2+\alpha})$
Update	LR-LR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^3)$	$\mathcal{O}(p^3b^2r)$	$\mathcal{O}(m^{3-x+\alpha})$

- Ex. 1: compute $\mathcal{C}_{step}(b, p, r) = \text{cost} \times \text{number}$
- Ex. 2: compute $\mathcal{C}_{step}(m, x, \alpha)$ with $b = \mathcal{O}(m^x)$ and $r = \mathcal{O}(m^\alpha)$.
- Ex. 3: compute the total complexity (sum of all steps)

$$\mathcal{C}_{total}(m, x, \alpha) = \mathcal{O}(m^{3-x+\alpha} + m^{2+x})$$

- Ex. 4: compute the optimal block size $b^* = \mathcal{O}(m^{x^*})$ and the resulting optimal complexity: $x^* = \quad, b^* = \quad,$ and $\mathcal{C}_{opt}(m, r) = \mathcal{C}_{total}(m, x^*, \alpha) = \quad$

Flop complexity of the dense BLR factorization

Let us consider a dense (frontal) matrix of order m divided into $p \times p$ blocks of order b , with $p = m/b$.

step	type	cost	number	$\mathcal{C}_{step}(b, p, r)$	$\mathcal{C}_{step}(m, x, \alpha)$
Factor	FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p)$	$\mathcal{O}(pb^3)$	$\mathcal{O}(m^{1+2x})$
Solve	FR-FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^3)$	$\mathcal{O}(m^{2+x})$
Compress	LR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^2r)$	$\mathcal{O}(m^{2+\alpha})$
Update	FR-FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p)$	$\mathcal{O}(pb^3)$	$\mathcal{O}(m^{1+2x})$
Update	LR-FR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^2r)$	$\mathcal{O}(m^{2+\alpha})$
Update	LR-LR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^3)$	$\mathcal{O}(p^3b^2r)$	$\mathcal{O}(m^{3-x+\alpha})$

- Ex. 1: compute $\mathcal{C}_{step}(b, p, r) = \text{cost} \times \text{number}$
- Ex. 2: compute $\mathcal{C}_{step}(m, x, \alpha)$ with $b = \mathcal{O}(m^x)$ and $r = \mathcal{O}(m^\alpha)$.
- Ex. 3: compute the total complexity (sum of all steps)

$$\mathcal{C}_{total}(m, x, \alpha) = \mathcal{O}(m^{3-x+\alpha} + m^{2+x})$$

- Ex. 4: compute the optimal block size $b^* = \mathcal{O}(m^{x^*})$ and the resulting optimal complexity: $x^* = (1 + \alpha)/2$, $b^* = m^{(1+\alpha)/2}$, and $\mathcal{C}_{opt}(m, r) = \mathcal{C}_{total}(m, x^*, \alpha) =$

Flop complexity of the dense BLR factorization

Let us consider a dense (frontal) matrix of order m divided into $p \times p$ blocks of order b , with $p = m/b$.

step	type	cost	number	$\mathcal{C}_{step}(b, p, r)$	$\mathcal{C}_{step}(m, x, \alpha)$
Factor	FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p)$	$\mathcal{O}(pb^3)$	$\mathcal{O}(m^{1+2x})$
Solve	FR-FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^3)$	$\mathcal{O}(m^{2+x})$
Compress	LR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^2r)$	$\mathcal{O}(m^{2+\alpha})$
Update	FR-FR	$\mathcal{O}(b^3)$	$\mathcal{O}(p)$	$\mathcal{O}(pb^3)$	$\mathcal{O}(m^{1+2x})$
Update	LR-FR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2b^2r)$	$\mathcal{O}(m^{2+\alpha})$
Update	LR-LR	$\mathcal{O}(b^2r)$	$\mathcal{O}(p^3)$	$\mathcal{O}(p^3b^2r)$	$\mathcal{O}(m^{3-x+\alpha})$

- Ex. 1: compute $\mathcal{C}_{step}(b, p, r) = \text{cost} \times \text{number}$
- Ex. 2: compute $\mathcal{C}_{step}(m, x, \alpha)$ with $b = \mathcal{O}(m^x)$ and $r = \mathcal{O}(m^\alpha)$.
- Ex. 3: compute the total complexity (sum of all steps)

$$\mathcal{C}_{total}(m, x, \alpha) = \mathcal{O}(m^{3-x+\alpha} + m^{2+x})$$

- Ex. 4: compute the optimal block size $b^* = \mathcal{O}(m^{x^*})$ and the resulting optimal complexity: $x^* = (1 + \alpha)/2$, $b^* = m^{(1+\alpha)/2}$, and $\mathcal{C}_{opt}(m, r) = \mathcal{C}_{total}(m, x^*, \alpha) = \mathcal{O}(m^{5/2}r^{1/2})$

Complexity of the sparse multifrontal BLR factorization

Sparse multifrontal complexity with ND

For a dense complexity $\mathcal{C}_{opt}(m, r)$, the sparse complexity is computed as

$$\mathcal{C}_{mf} = \sum_{\ell=0}^{\log_{2^{d-1}} N} 2^{d\ell} \mathcal{C}_{opt}\left(\left(\frac{N}{2^\ell}\right)^{d-1}\right),$$

where d is the dimension (2 or 3).

	operations (OPC)	factor size (NNZ)
$N \times N$ grid		
FR	$\mathcal{O}(N^3)$	$\mathcal{O}(N^2 \log N)$
BLR	$\mathcal{O}(N^{5/2}r^{1/2})$	$\mathcal{O}(N^2)$
$N \times N \times N$ grid		
FR	$\mathcal{O}(N^6)$	$\mathcal{O}(N^4)$
BLR		

Complexity of the sparse multifrontal BLR factorization

Sparse multifrontal complexity with ND

For a dense complexity $\mathcal{C}_{opt}(m, r)$, the sparse complexity is computed as

$$\mathcal{C}_{mf} = \sum_{\ell=0}^{\log_{2^{d-1}} N} 2^{d\ell} \mathcal{C}_{opt}\left(\left(\frac{N}{2^\ell}\right)^{d-1}\right),$$

where d is the dimension (2 or 3).

	operations (OPC)	factor size (NNZ)
$N \times N$ grid		
FR	$\mathcal{O}(N^3)$	$\mathcal{O}(N^2 \log N)$
BLR	$\mathcal{O}(N^{5/2}r^{1/2})$	$\mathcal{O}(N^2)$
$N \times N \times N$ grid		
FR	$\mathcal{O}(N^6)$	$\mathcal{O}(N^4)$
BLR	$\mathcal{O}(N^5r^{1/2})$	

Complexity of the sparse multifrontal BLR factorization

Sparse multifrontal complexity with ND

For a dense complexity $\mathcal{C}_{opt}(m, r)$, the sparse complexity is computed as

$$\mathcal{C}_{mf} = \sum_{\ell=0}^{\log_{2^{d-1}} N} 2^{d\ell} \mathcal{C}_{opt}\left(\left(\frac{N}{2^\ell}\right)^{d-1}\right),$$

where d is the dimension (2 or 3).

	operations (OPC)	factor size (NNZ)
$N \times N$ grid		
FR	$\mathcal{O}(N^3)$	$\mathcal{O}(N^2 \log N)$
BLR	$\mathcal{O}(N^{5/2} r^{1/2})$	$\mathcal{O}(N^2)$
$N \times N \times N$ grid		
FR	$\mathcal{O}(N^6)$	$\mathcal{O}(N^4)$
BLR	$\mathcal{O}(N^5 r^{1/2})$	$\mathcal{O}(N^3 \max(r^{1/2}, \log N))$

Experimental Setting: Matrices

1. Poisson: N^3 grid with a 7-point stencil with $u = 1$ on the boundary $\partial\Omega$

$$\Delta u = f$$

Rank bound is theoretically proven to be $r = \mathcal{O}(1)$.

2. Helmholtz: N^3 grid with a 27-point stencil, ω is the angular frequency, $v(x)$ is the seismic velocity field, and $u(x, \omega)$ is the time-harmonic wavefield solution to the forcing term $s(x, \omega)$.

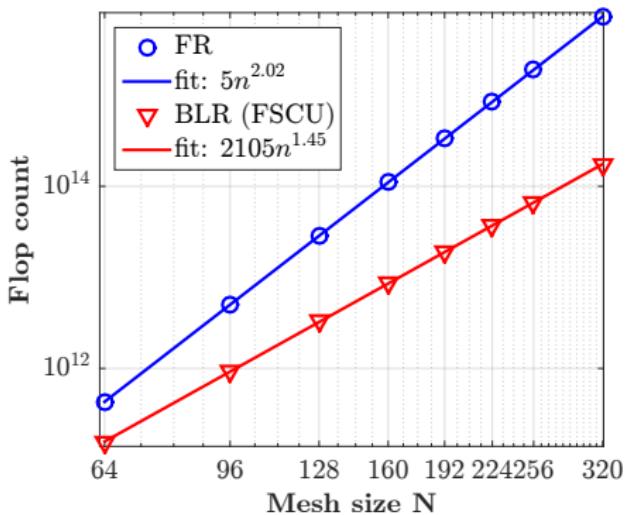
$$\left(-\Delta - \frac{\omega^2}{v(x)^2} \right) u(x, \omega) = s(x, \omega)$$

ω is fixed and equal to 4Hz.

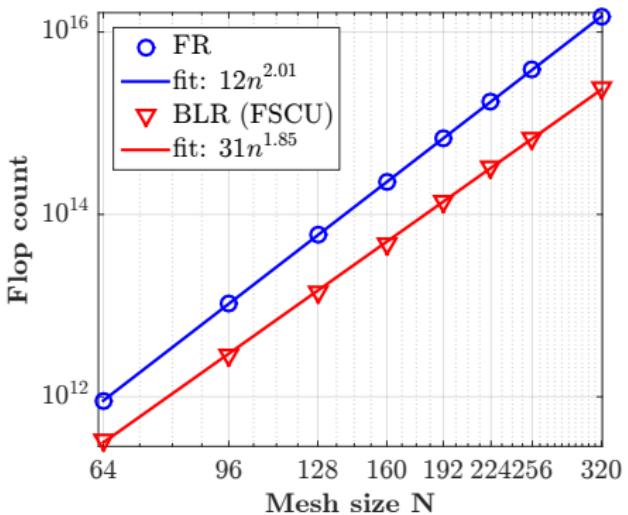
Heuristically, rank bound can be expected to behave as $r = \mathcal{O}(N)$.

Experimental MF flop complexity

Nested Dissection ordering (geometric)

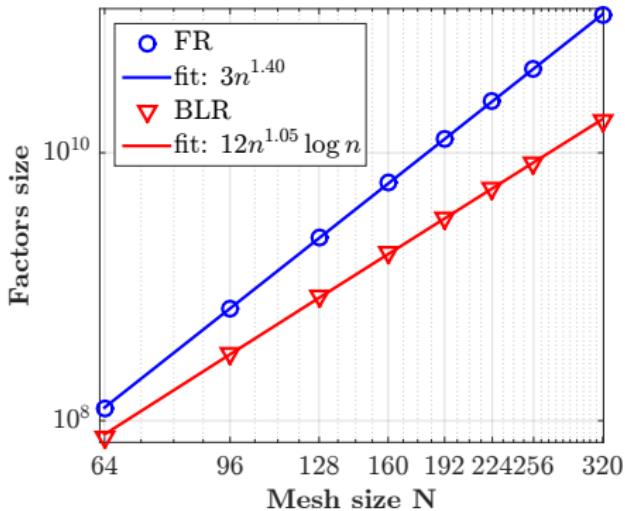


Nested Dissection ordering (geometric)

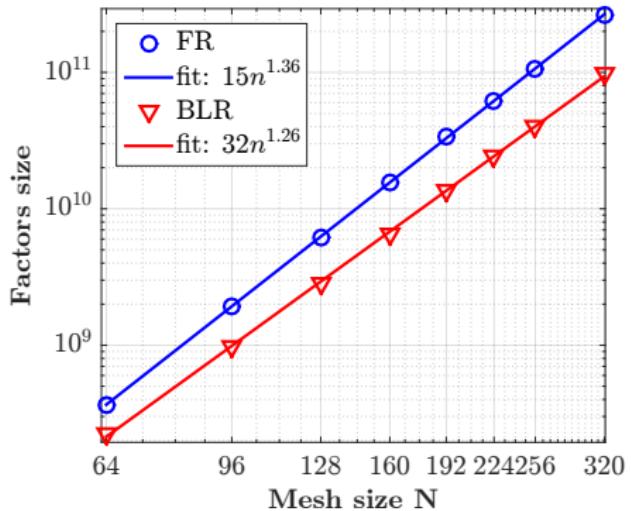


Experimental MF factor size complexity

NNZ
(Poisson)

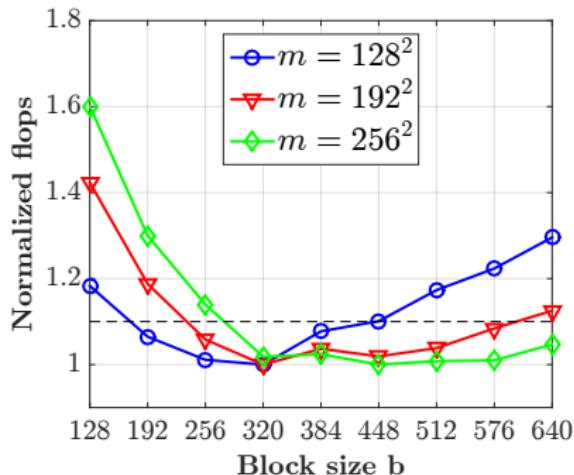


NNZ
(Helmholtz)



Influence of the block size b on the complexity

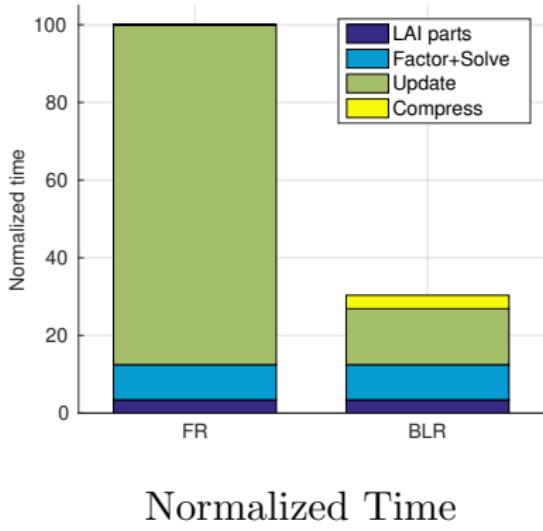
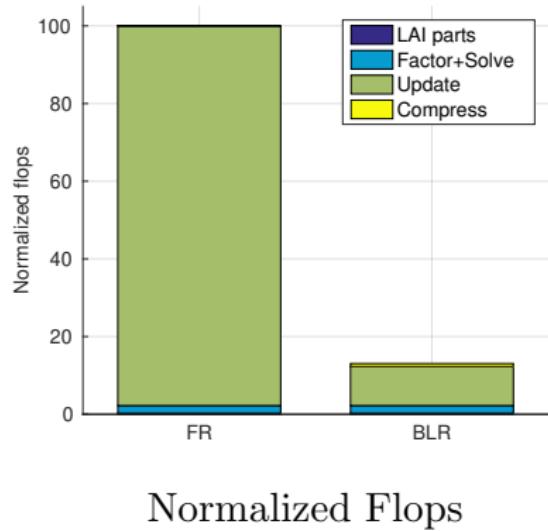
Analysis on the root node (of size $m = N^2$):



- large range of acceptable block sizes around the optimal b^*
⇒ **flexibility** to tune block size for performance
- that range increases with the size of the matrix
⇒ necessity to have **variable block sizes**

Part II: performance of the BLR factorization

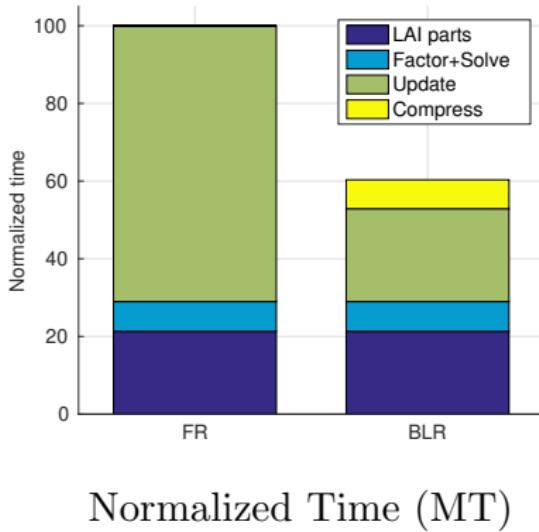
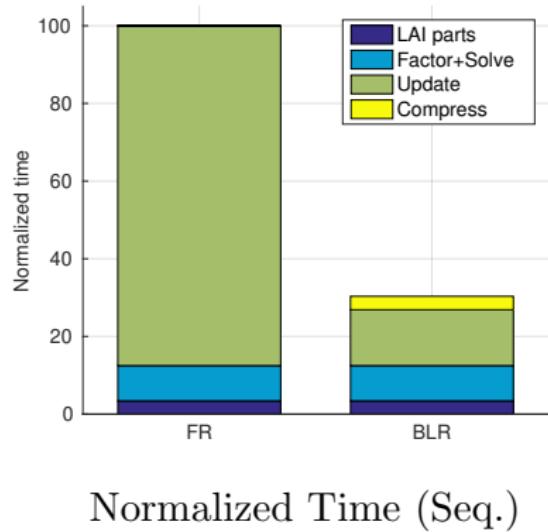
Sequential result



7.7 gain in flops only translated to a 3.3 gain in time: why?

- lower granularity of the Update
- higher relative weight of the FR parts
- inefficient Compress

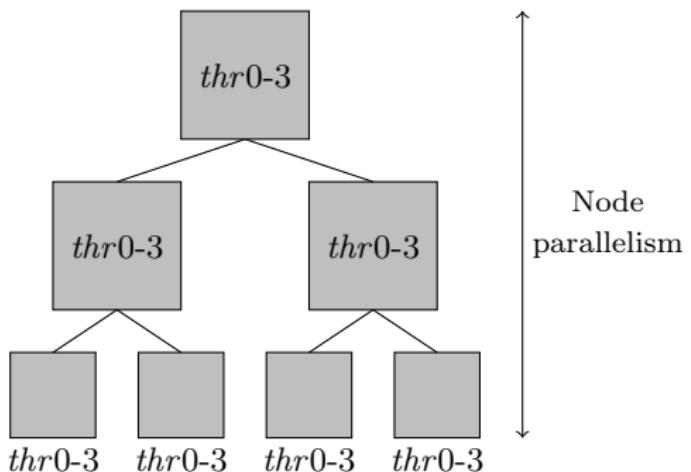
Multithreaded result on 24 threads



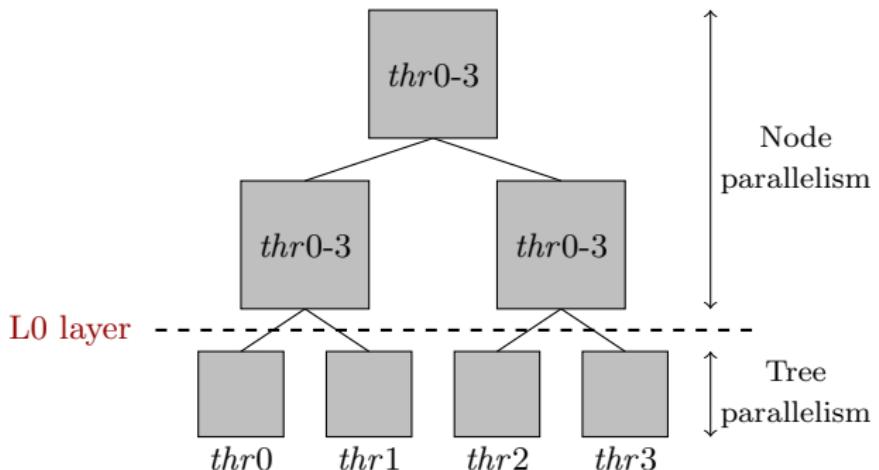
3.3 gain in sequential becomes 1.7 in multithreaded: why?

- LAI parts have become critical
- Update and Compress are memory-bound

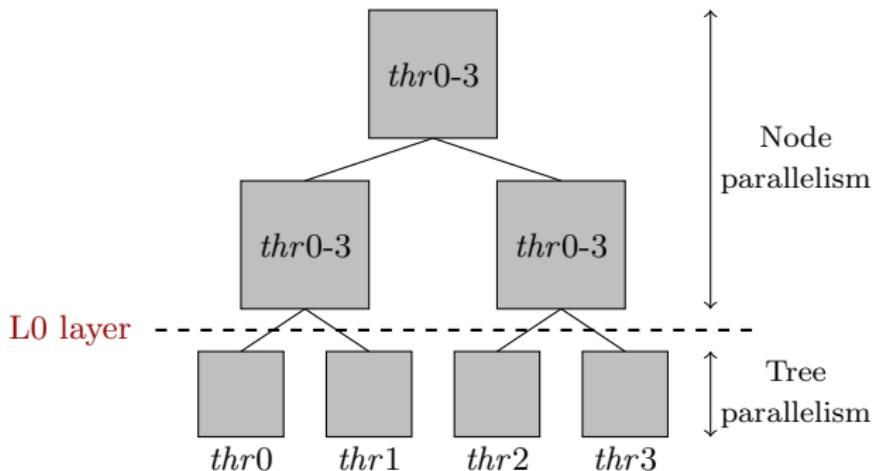
Exploiting tree-based multithreading in MF solvers



Exploiting tree-based multithreading in MF solvers

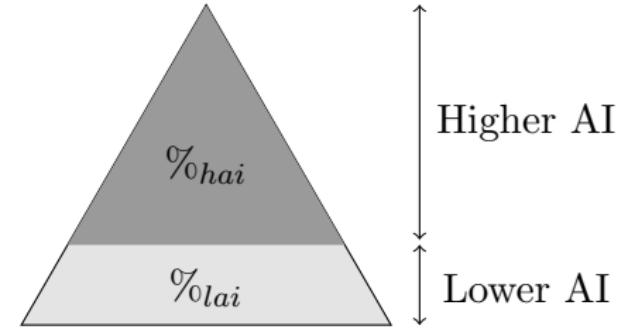


Exploiting tree-based multithreading in MF solvers



⇒ how big an impact can tree-based multithreading make?

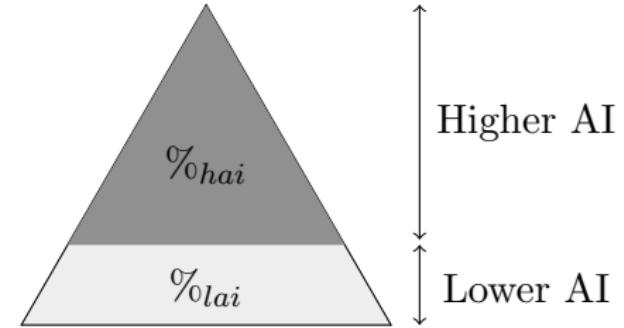
Impact of tree-based multithreading on BLR (24 threads)



	node only	node + tree		
	time	$\%_{lai}$	time	$\%_{lai}$
FR	509	21%		
BLR				

- In FR, top of the tree is dominant

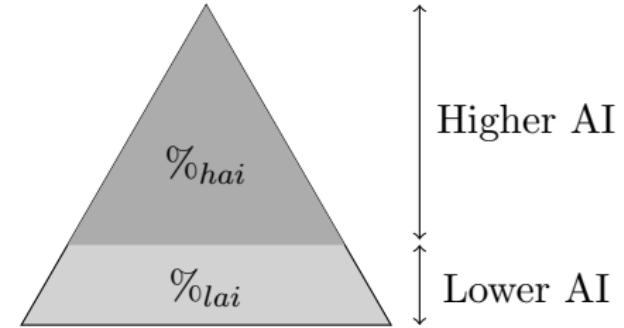
Impact of tree-based multithreading on BLR (24 threads)



	node only	node + tree		
	time	$\%_{lai}$	time	$\%_{lai}$
FR	509	21%	424	13%
BLR				

- In FR, top of the tree is dominant \Rightarrow tree MT brings little gain

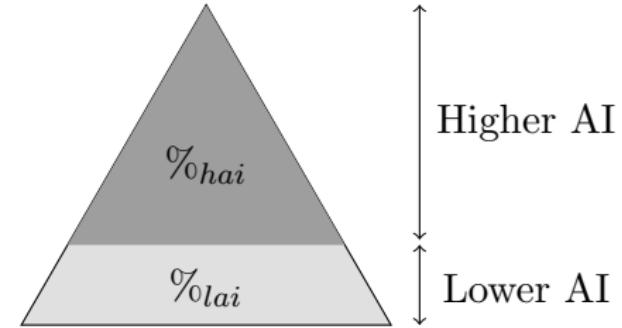
Impact of tree-based multithreading on BLR (24 threads)



	node only		node + tree	
	time	$\%_{lai}$	time	$\%_{lai}$
FR	509	21%	424	13%
BLR	307	35%		

- In FR, top of the tree is dominant \Rightarrow tree MT brings little gain
- In BLR, bottom of the tree compresses less, becomes important

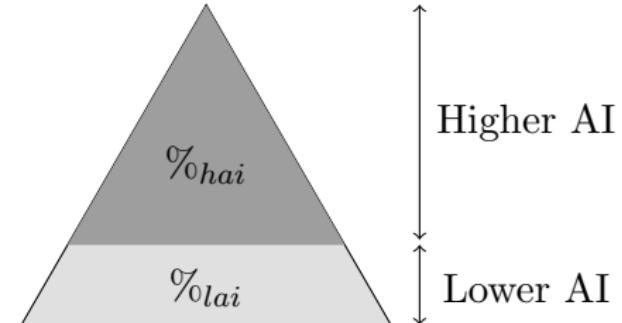
Impact of tree-based multithreading on BLR (24 threads)



	node only		node + tree	
	time	% <i>lai</i>	time	% <i>lai</i>
FR	509	21%	424	13%
BLR	307	35%	221	24%

- In FR, top of the tree is dominant \Rightarrow tree MT brings little gain
- In BLR, bottom of the tree compresses less, becomes important
 \Rightarrow **1.7** gain becomes **1.9** thanks to tree-based multithreading

Impact of tree-based multithreading on BLR (24 threads)



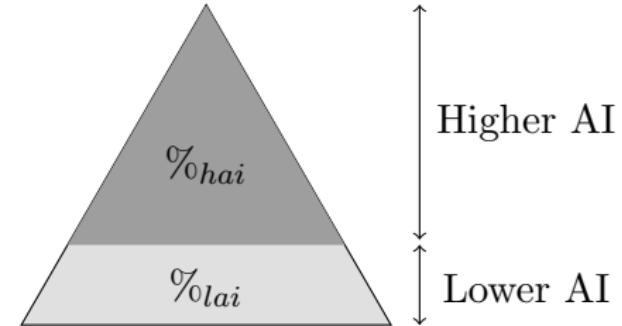
	node only	node + tree		
	time	$\%_{lai}$	time	$\%_{lai}$
FR	509	21%	424	13%
BLR	307	35%	221	24%

- In FR, top of the tree is dominant \Rightarrow tree MT brings little gain
- In BLR, bottom of the tree compresses less, becomes important
 \Rightarrow **1.7** gain becomes **1.9** thanks to tree-based multithreading

Theoretical speedup

	tree only	node only	node + tree
$N \times N \times N$ grid	$\mathcal{O}(1)$	$\mathcal{O}(N^3)$	$\mathcal{O}(N^4)$
	FR		

Impact of tree-based multithreading on BLR (24 threads)



	node only		node + tree	
	time	$\%_{lai}$	time	$\%_{lai}$
FR	509	21%	424	13%
BLR	307	35%	221	24%

- In FR, top of the tree is dominant \Rightarrow tree MT brings little gain
- In BLR, bottom of the tree compresses less, becomes important
 \Rightarrow **1.7** gain becomes **1.9** thanks to tree-based multithreading

Theoretical speedup

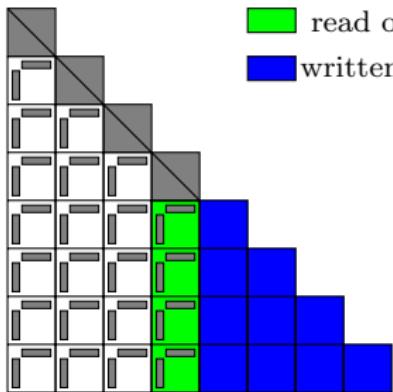
	tree only	node only	node + tree
$N \times N \times N$ grid	$\mathcal{O}(1)$	$\mathcal{O}(N^3)$	$\mathcal{O}(N^4)$
	$\mathcal{O}(1)$	$\mathcal{O}(N^2 r^{1/2})$	$\mathcal{O}(N^3 r^{1/2})$

Right-looking Vs. Left-looking analysis (24 threads)

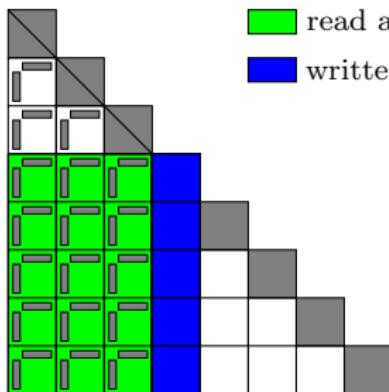
	FR time		BLR time	
	RL	LL	RL	LL
Update	338	336	110	67
Total	424	421	221	175

Right-looking Vs. Left-looking analysis (24 threads)

	FR time		BLR time	
	RL	LL	RL	LL
Update	338	336	110	67
Total	424	421	221	175



RL factorization



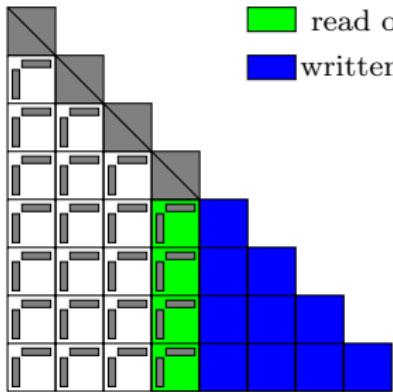
LL factorization

■ read once
■ written at each step

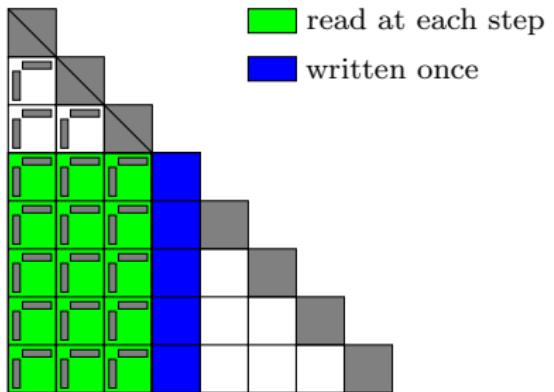
■ read at each step
■ written once

Right-looking Vs. Left-looking analysis (24 threads)

	FR time		BLR time	
	RL	LL	RL	LL
Update	338	336	110	67
Total	424	421	221	175



RL factorization

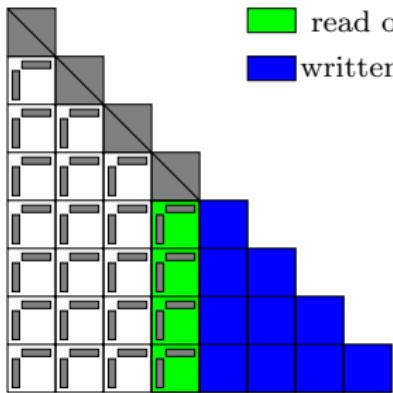


LL factorization

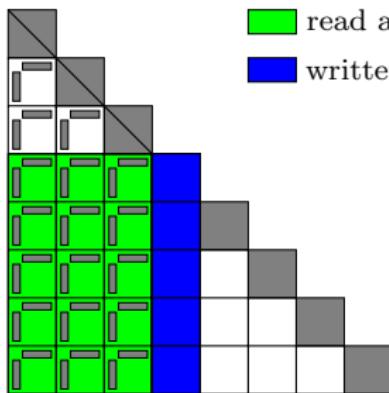
⇒ Lower volume of **memory transfers** in LL (more critical in MT)

Right-looking Vs. Left-looking analysis (24 threads)

	FR time		BLR time	
	RL	LL	RL	LL
Update	338	336	110	67
Total	424	421	221	175



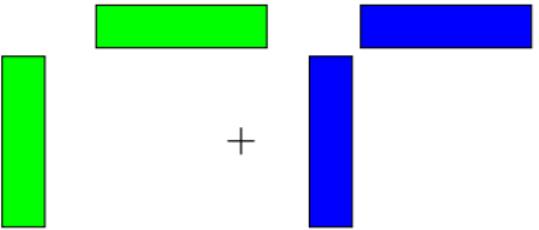
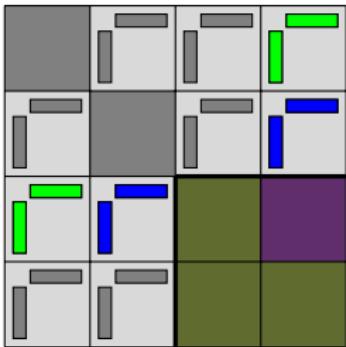
RL factorization



LL factorization

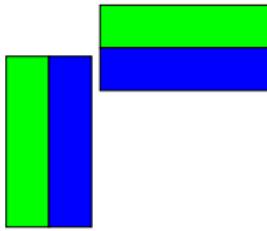
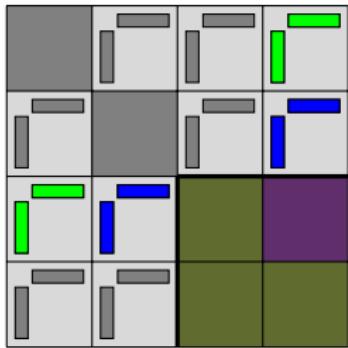
⇒ Lower volume of memory transfers in LL (more critical in MT)
Update is now less memory-bound: **1.9** gain becomes **2.4** in LL

LUAR variant: accumulation and recompression



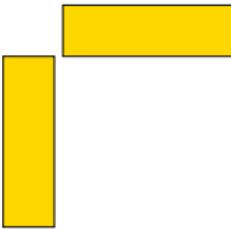
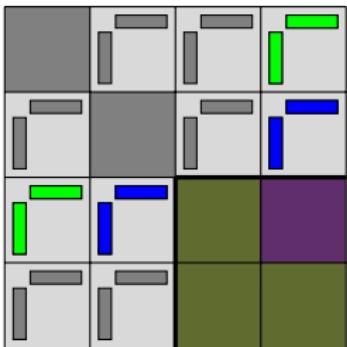
- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR

LUAR variant: accumulation and recompression



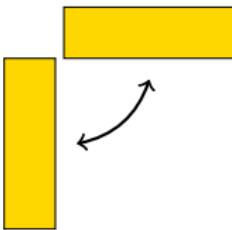
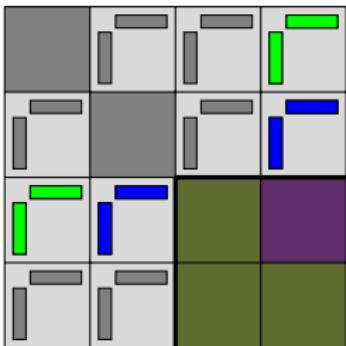
- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR
 - Better granularity in Update operations

LUAR variant: accumulation and recompression



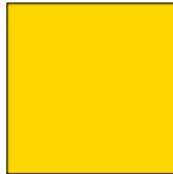
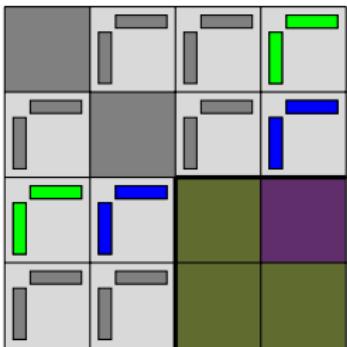
- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR
 - Better granularity in Update operations
 - Potential recompression \Rightarrow **asymptotic complexity reduction?**
 \Rightarrow Designed and compared several recompression strategies

LUAR variant: accumulation and recompression



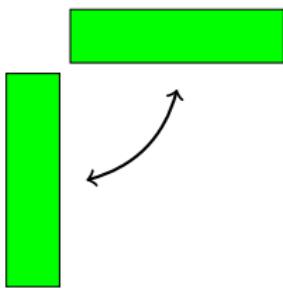
- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR
 - Better granularity in Update operations
 - Potential recompression \Rightarrow **asymptotic complexity reduction?**
 \Rightarrow Designed and compared several recompression strategies

LUAR variant: accumulation and recompression

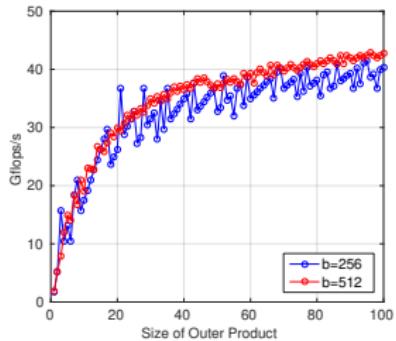


- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR
 - Better granularity in Update operations
 - Potential recompression \Rightarrow **asymptotic complexity reduction?**
 \Rightarrow Designed and compared several recompression strategies

Performance of Outer Product with LUA(R) (24 threads)

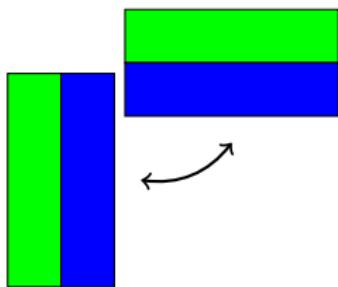


Outer Product benchmark

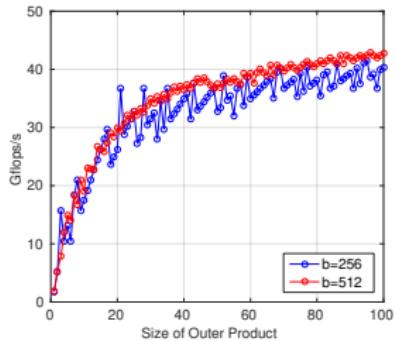


	LL	LUA	LUAR*
average size of Outer Product	16.5		
flops ($\times 10^{12}$)	Outer Product	3.8	
	Total	10.2	
time (s)	Outer Product	21	
	Total	175	

Performance of Outer Product with LUA(R) (24 threads)

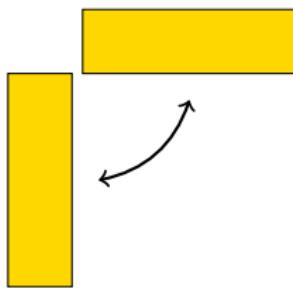


Outer Product benchmark

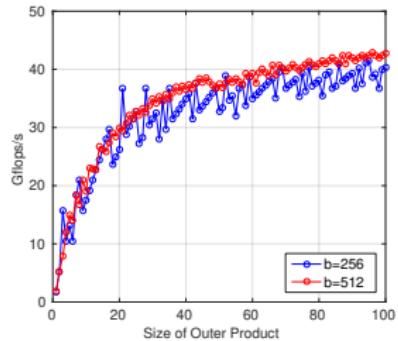


		LL	LUA	LUAR*
average size of Outer Product		16.5	61.0	
flops ($\times 10^{12}$)	Outer Product	3.8	3.8	
	Total	10.2	10.2	
time (s)	Outer Product	21	14	
	Total	175	167	

Performance of Outer Product with LUA(R) (24 threads)



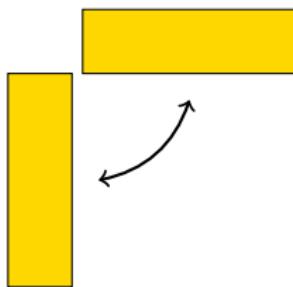
Outer Product benchmark



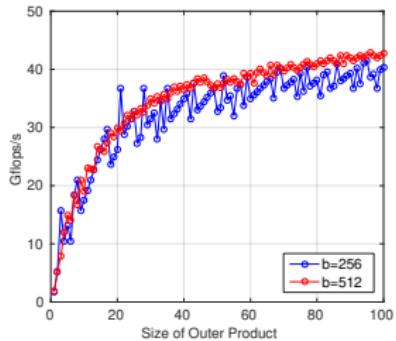
		LL	LUA	LUAR*
average size of Outer Product		16.5	61.0	32.8
flops ($\times 10^{12}$)	Outer Product	3.8	3.8	1.6
	Total	10.2	10.2	8.1
time (s)	Outer Product	21	14	6
	Total	175	167	160

* All metrics include the Recompression overhead

Performance of Outer Product with LUA(R) (24 threads)



Outer Product benchmark



		LL	LUA	LUAR*
average size of Outer Product		16.5	61.0	32.8
flops ($\times 10^{12}$)	Outer Product	3.8	3.8	1.6
	Total	10.2	10.2	8.1
time (s)	Outer Product	21	14	6
	Total	175	167	160

* All metrics include the Recompression overhead

Higher granularity and lower flops in Update:
⇒ **2.4** gain becomes **2.6**

Impact of machine properties on BLR: roofline model

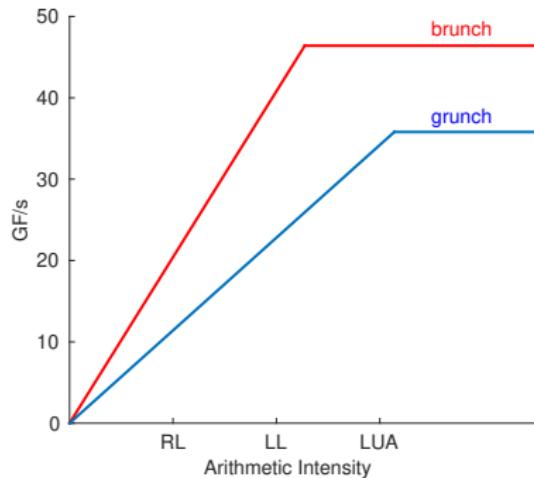
	specs		time (s) for BLR factorization		
	peak (GF/s)	bw (GB/s)	RL	LL	LUA
grunch (28 threads)	37	57	248	228	196
brunch (24 threads)	46	102	221	175	167

Impact of machine properties on BLR: roofline model

	specs		time (s) for BLR factorization		
	peak (GF/s)	bw (GB/s)	RL	LL	LUA
grunch (28 threads)	37	57	248	228	196
brunch (24 threads)	46	102	221	175	167

Arithmetic Intensity in BLR:

- **LL > RL** (lower volume of memory transfers)
- **LUA > LL** (higher granularities \Rightarrow more efficient cache use)

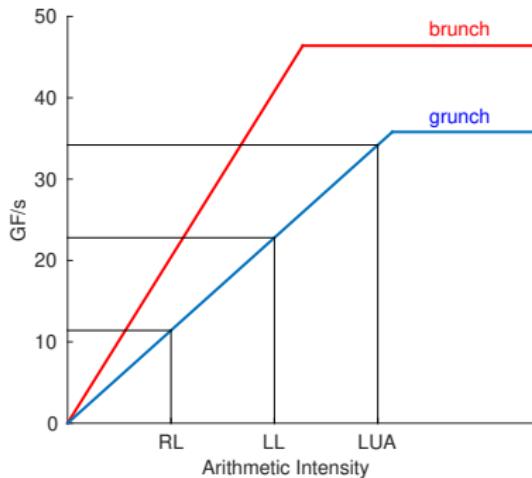


Impact of machine properties on BLR: roofline model

	specs peak (GF/s)	bw (GB/s)	time (s) for BLR factorization		
			RL	LL	LUA
grunch (28 threads)	37	57	248	228	196
brunch (24 threads)	46	102	221	175	167

Arithmetic Intensity in BLR:

- **LL > RL** (lower volume of memory transfers)
- **LUA > LL** (higher granularities \Rightarrow more efficient cache use)

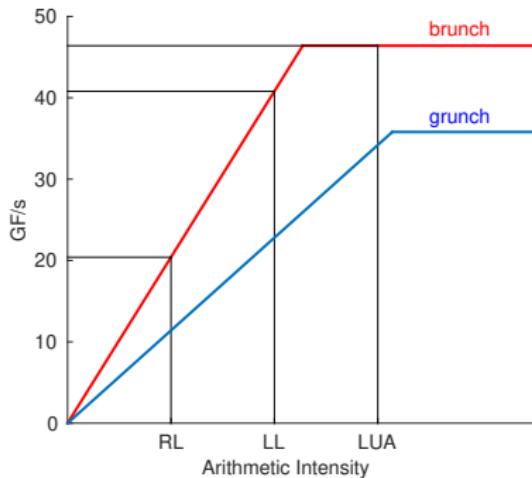


Impact of machine properties on BLR: roofline model

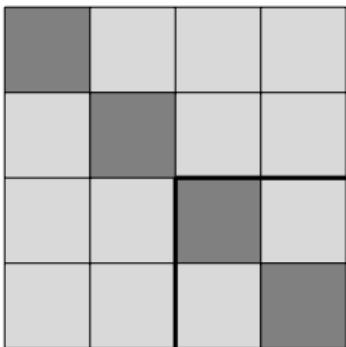
	specs peak (GF/s)	bw (GB/s)	time (s) for BLR factorization		
			RL	LL	LUA
grunch (28 threads)	37	57	248	228	196
brunch (24 threads)	46	102	221	175	167

Arithmetic Intensity in BLR:

- **LL > RL** (lower volume of memory transfers)
- **LUA > LL** (higher granularities \Rightarrow more efficient cache use)

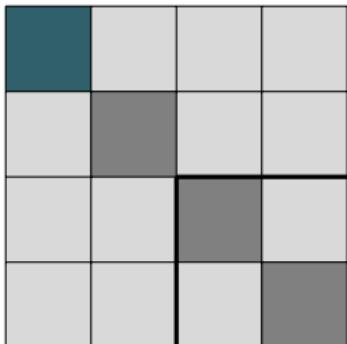


FCSU variant: compress before solve



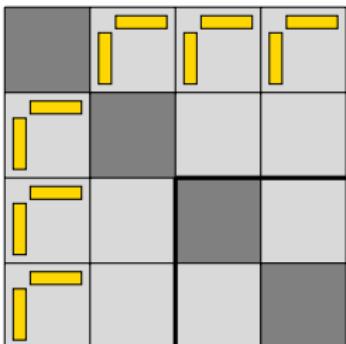
- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR
 - Better granularity in Update operations
 - Potential recompression \Rightarrow **asymptotic complexity reduction?**
 \Rightarrow Designed and compared several recompression strategies
- FCSU(+LUAR)

FCSU variant: compress before solve



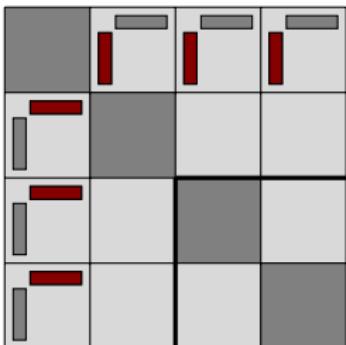
- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR
 - Better granularity in Update operations
 - Potential recompression \Rightarrow **asymptotic complexity reduction?**
 \Rightarrow Designed and compared several recompression strategies
- FCSU(+LUAR)

FCSU variant: compress before solve



- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR
 - Better granularity in Update operations
 - Potential recompression \Rightarrow **asymptotic complexity reduction?**
 \Rightarrow Designed and compared several recompression strategies
- FCSU(+LUAR)
 - Compress performed **before** Solve

FCSU variant: compress before solve



- FSCU (Factor, Solve, Compress, Update)
- FSCU+LUAR
 - Better granularity in Update operations
 - Potential recompression \Rightarrow asymptotic complexity reduction?
 \Rightarrow Designed and compared several recompression strategies
- FCSU(+LUAR)
 - Compress performed before Solve
 - Low-rank Solve \Rightarrow asymptotic complexity reduction?
 - On previous matrix: $160 \rightarrow 111s \Rightarrow 2.6$ gain becomes **3.7**

Variants improve asymptotic complexity

Exercise: prove that the variants improve the asymptotic complexity

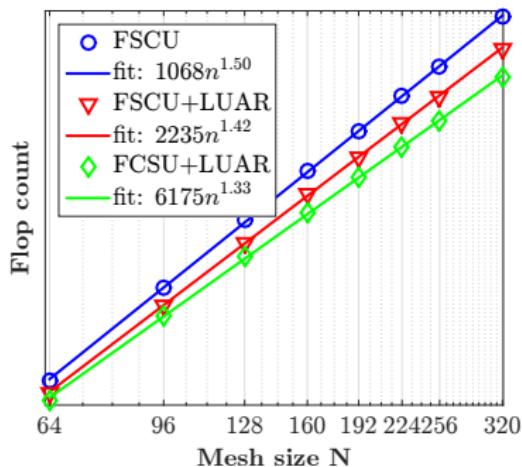
	FSCU	\rightarrow	FSCU+LUAR	\rightarrow	FCSU+LUAR
dense	$\mathcal{O}(m^{5/2}r^{1/2})$	\rightarrow	$\mathcal{O}(m^{7/3}r^{2/3})$	\rightarrow	$\mathcal{O}(m^2r)$
sparse (3D)	$\mathcal{O}(N^5r^{1/2})$	\rightarrow	$\mathcal{O}(N^{14/3}r^{2/3})$	\rightarrow	$\mathcal{O}(N^4r)$

Variants improve asymptotic complexity

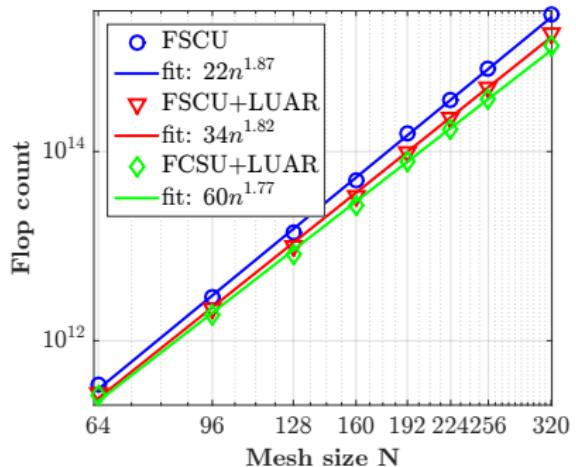
Exercise: prove that the variants improve the asymptotic complexity

FSCU	\rightarrow	FSCU+LUAR	\rightarrow	FCSU+LUAR
dense	$\mathcal{O}(m^{5/2}r^{1/2})$	$\rightarrow \mathcal{O}(m^{7/3}r^{2/3})$	$\rightarrow \mathcal{O}(m^2r)$	
sparse (3D)	$\mathcal{O}(N^5r^{1/2})$	$\rightarrow \mathcal{O}(N^{14/3}r^{2/3})$	$\rightarrow \mathcal{O}(N^4r)$	

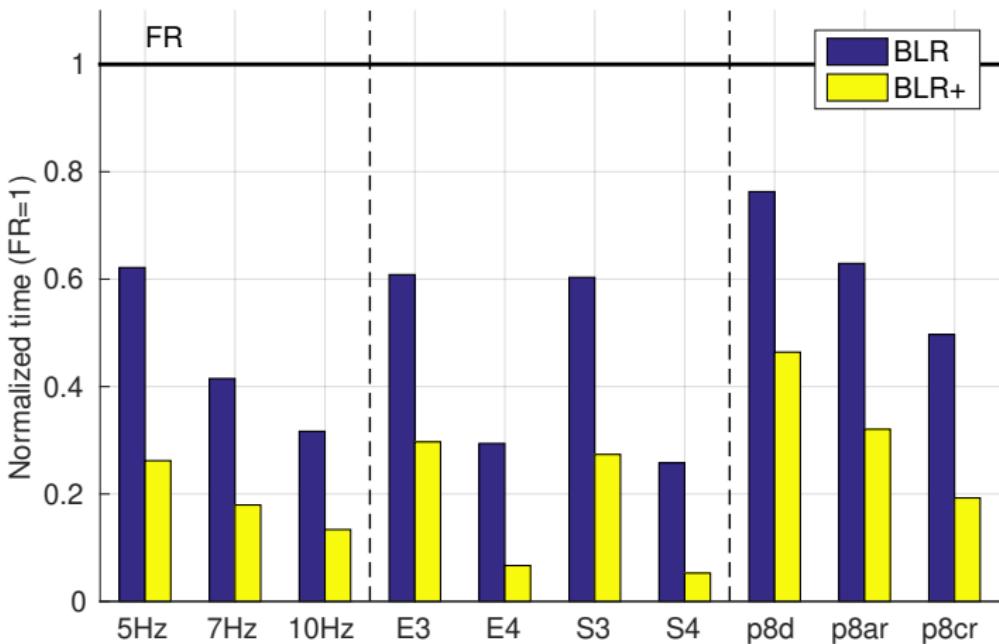
Poisson ($\varepsilon = 10^{-10}$)



Helmholtz ($\varepsilon = 10^{-4}$)



Multicore performance results (24 threads)



- “BLR”: FSCU, right-looking, node only multithreading
- “BLR+”: FCSU+LUAR, left-looking, node+tree multithreading

To go further on BLR

- Amestoy, Ashcraft, Boiteau, Buttari, L'Excellent, and Weisbecker, **Improving Multifrontal Methods by Means of Block Low-Rank Representations**, SIAM J. Sci. Comput., 2015,
<https://doi.org/10.1137/120903476>
- Amestoy, Buttari, L'Excellent, and Mary, **On the Complexity of the Block Low-Rank Multifrontal Factorization**, SIAM J. Sci. Comput., 2017, <https://doi.org/10.1137/16M1077192>
- Amestoy, Buttari, L'Excellent, and Mary, **Performance and Scalability of the Block Low-Rank Multifrontal Factorization on Multicore Architectures**, ACM Trans. Math. Soft., 2018,
<https://doi.org/10.1145/3242094>
- Amestoy, Buttari, L'Excellent, and Mary, **Bridging the Gap Between Flat and Hierarchical Low-Rank Matrix Formats: The Multilevel Block Low-Rank Format**, SIAM J. Sci. Comput., 2019,
<https://doi.org/10.1137/18M1182760>
- Amestoy, Brossier, Buttari, L'Excellent, Mary, Métivier, Miniussi, and Operto, **Fast 3D frequency-domain full waveform inversion with a parallel Block Low-Rank multifrontal direct solver: application to OBC data from the North Sea**, Geophysics, 2016,
<https://doi.org/10.1190/geo2016-0052.1>

References I

- [1] E. Agullo, P. R. Amestoy, A. Buttari, A. Guermouche, J.-Y. L'Excellent, and F.-H. Rouet. "Robust memory-aware mappings for parallel multifrontal factorizations". In: SIAM Journal on Scientific Computing 38.3 (2016), pp. C256–C279.
- [2] T. A. Davis, S. Rajamanickam, and W. M. Sid-Lakhdar. "A survey of direct methods for sparse linear systems". In: Acta Numerica 25 (2016), pp. 383–566.
- [3] I. S. Duff, A. M. Erisman, and J. K. Reid. Direct Methods for Sparse Matrices. London: Oxford University Press, 1986.
- [4] I. S. Duff, A. M. Erisman, and J. K. Reid. Direct Methods for Sparse Matrices, Second Edition. London: Oxford University Press, 2017.
- [5] I. S. Duff and J. Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. Tech. rep. RAL-TR-1999-030. Also CERFACS Report TR/PA/99/13. Rutherford Appleton Laboratory, 1999.
- [6] I. S. Duff and J. Koster. "On algorithms for permuting large entries to the diagonal of a sparse matrix". In: SIAM Journal on Matrix Analysis and Applications 22.4 (2001), pp. 973–996.
- [7] I. S. Duff and S. Pralet. "Strategies for scaling and pivoting for sparse symmetric indefinite problems". In: SIAM Journal on Matrix Analysis and Applications 27.2 (2005), pp. 313–340.

References II

- [8] I. S. Duff and S. Pralet. “Towards Stable Mixed Pivoting Strategies for the Sequential and Parallel Solution of Sparse Symmetric Indefinite Systems”. In: SIAM Journal on Matrix Analysis and Applications 29.3 (2007), pp. 1007–1024.
- [9] I. S. Duff and J. K. Reid. “The multifrontal solution of indefinite sparse symmetric linear systems”. In: ACM Transactions on Mathematical Software 9 (1983), pp. 302–325.
- [10] A. Geist and E. G. Ng. “Task scheduling for parallel sparse Cholesky factorization”. In: Int J. Parallel Programming 18 (1989), pp. 291–314.
- [11] J. A. George. “Nested dissection of a regular finite-element mesh”. In: SIAM Journal on Numerical Analysis 10.2 (1973), pp. 345–363.
- [12] J. R. Gilbert and J. W. H. Liu. “Elimination structures for unsymmetric sparse LU factors”. In: SIAM Journal on Matrix Analysis and Applications 14 (1993), pp. 334–352.
- [13] M. T. Heath, E. G. Ng, and B. W. Peyton. “Parallel Algorithms for Sparse Linear Systems”. In: SIAM Review 33 (1991), pp. 420–460.
- [14] J.-Y. L'Excellent. “Multifrontal Methods: Parallelism, Memory Usage and Numerical Aspects”. *Habilitation à diriger des recherches*. École normale supérieure de Lyon, Sept. 2012. URL:
<http://tel.archives-ouvertes.fr/tel-00737751>.

References III

- [15] J. W. H. Liu. “On the storage requirement in the out-of-core multifrontal method for sparse factorization”. In: ACM Transactions on Mathematical Software 12 (1986), pp. 127–148.
- [16] J. W. H. Liu. “The Role of Elimination Trees in Sparse Factorization”. In: SIAM Journal on Matrix Analysis and Applications 11 (1990), pp. 134–172.
- [17] W.-H. Liu and A. H. Sherman. “Comparative Analysis of the Cuthill-McKee and the Reverse Cuthill-McKee Ordering Algorithms for Sparse Matrices”. In: SIAM Journal on Numerical Analysis 13.2 (1976), pp. 198–213. doi: 10.1137/0713020. eprint: https://doi.org/10.1137/0713020/tmp/siam_sjnaam13_198.bib. URL: <https://doi.org/10.1137/0713020>.
- [18] S. V. Parter. “The Use of Linear Graphs in Gauss Elimination”. In: SIAM Review 3 (1961), pp. 119–130.
- [19] A. Pothen and C. Sun. “A Mapping Algorithm for Parallel Sparse Cholesky Factorization”. In: SIAM Journal on Scientific Computing 14(5) (1993), pp. 1253–1257.
- [20] D. J. Rose, R. E. Tarjan, and G. S. Lueker. “Algorithmic aspects of vertex elimination on graphs”. In: SIAM Journal on Computing 5.2 (1976), pp. 266–283.
- [21] R. Schreiber. “A new implementation of sparse Gaussian elimination”. In: ACM Transactions on Mathematical Software 8 (1982), pp. 256–276.