Harnessing inexactness in scientific computing

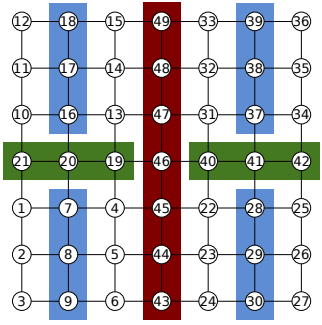
Lecture 6: direct methods for $Ax = b$

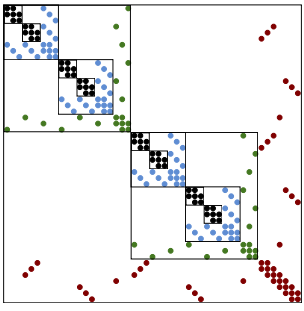
Theo Mary (CNRS)
theo.mary@lip6.fr
https://perso.lip6.fr/Theo.Mary/

Elisa Riccietti (ENS Lyon)
elisa.riccietti@ens-lyon.fr
https://perso.ens-lyon.fr/elisa.riccietti/

M2 course at ENS Lyon, 2024–2025

Slides available on course webpage





Dense systems

Sparse systems
- Fill-in characterization
- The multifrontal method
- Nested dissection and complexity
- Parallelism
- Memory
- Exercise

LU factorization

- Objective: given $A \in \mathbb{R}^{n \times n}$, compute $A = LU$ where U is upper triangular and L is unit lower triangular
- $\forall i, j \quad a_{ij} = \sum_{k=1}^{\min(i,j)} \ell_{ik} u_{kj}$
- Doolittle's formula:

for $k = 1 : n$ **do**
  for $j = k : n$ **do**
    $u_{kj} = a_{kj} - \sum_{i=1}^{k-1} \ell_{ki} u_{ij}$
  end for
  for $i = k + 1 : n$ **do**
    $\ell_{ik} = \left(a_{ik} - \sum_{j=1}^{k-1} \ell_{ij} u_{jk} \right) / u_{kk}$
  end for
end for

In-place LU factorization

- Overwrite upper triangular part of A with U (including diagonal) and lower triangular part (excluding diagonal) with L

```
for k = 1: n do
    for i = k + 1: n do
        a_ik = a_ik / a_kk
    end for
    for i = k + 1: n do
        for j = k + 1: n do
            a_ij ← a_ij − a_ik a_kj
        end for
    end for
end for
```

- $\text{flops}(LU) = \sum_{k=1}^{n-1} \left(n-k+2(n-k)^2\right) \approx 2\sum_{k=1}^{n-1}(n-k)^2 \approx \frac{2n^3}{3}$

Given a system $Ax = b$ and a computed solution $\widehat{x}$, we have

$$\eta_{\text{fwd}} = \frac{\|\widehat{x} - x\|}{\|\widehat{x}\|}$$

$$\eta_{\text{bwd}} = \min\{\varepsilon > 0 : \exists \Delta A,\ (A + \Delta A)\widehat{x} = b,\ \|\Delta A\| \leq \varepsilon \|A\|\}.$$

$$\kappa(A) = \|A\|\|A^{-1}\|$$

$$\eta_{\text{fwd}} \leq \kappa(A)\eta_{\text{bwd}}$$

$$\eta_{\text{bwd}} = \min\{\varepsilon > 0 : \exists \Delta A,\ (A + \Delta A)\widehat{x} = b,\ \|\Delta A\| \leq \varepsilon \|A\|\}.$$

We have the following formula

Rigal et Gaches, 1967

$$\eta_{\text{bwd}} = \frac{\|A\widehat{x} - b\|}{\|A\|\|\widehat{x}\|}$$

Proof:

- $\frac{\|A\widehat{x}-b\|}{\|A\|\|\widehat{x}\|} \leq \eta_{\text{bwd}}$
- $\eta_{\text{bwd}} \leq \frac{\|A\widehat{x}-b\|}{\|A\|\|\widehat{x}\|}$ (using $\Delta A = \frac{(b-A\widehat{x})\widehat{x}^T}{\|\widehat{x}\|_2^2}$)

Backward error bound (LU)

$$a_{ij} + \delta a_{ij} = \sum_{k=1}^{\min(i,j)} \widehat{\ell}_{ik}\widehat{u}_{kj}, \quad |a_{ij}| \le \gamma_n \sum_{k=1}^{\min(i,j)} |\widehat{\ell}_{ik}\widehat{u}_{kj}|$$

Theorem 9.3, Higham

$$A + \Delta A = \widehat{L}\widehat{U}, \quad |\Delta A| \le \gamma_n |\widehat{L}||\widehat{U}|$$

Backward error bound (LU)

$$a_{ij} + \delta a_{ij} = \sum_{k=1}^{\min(i,j)} \widehat{\ell}_{ik}\widehat{u}_{kj}, \quad |a_{ij}| \le \gamma_n \sum_{k=1}^{\min(i,j)} |\widehat{\ell}_{ik}\widehat{u}_{kj}|$$

Theorem 9.3, Higham

$$A + \Delta A = \widehat{L}\widehat{U}, \quad |\Delta A| \le \gamma_n |\widehat{L}||\widehat{U}|$$

Proof uses this more general result of the fundamental lemma:

Fundamental lemma

Let δ_k, $k = 1 : n$, such that $|\delta_k| \le u$ and $nu < 1$. Then for $\rho_i = \pm 1$

$$\prod_{k=1}^{n} (1 + \delta_k)^{\rho_i} = 1 + \theta_n, \quad |\theta_n| \le \gamma_n := \frac{nu}{1 - nu}.$$

Backward error bound ($Ax = b$)

Let $Ax = b$ be solved via the factorization $A = LU$ and the substitutions $Ly = b$ and $Ux = y$. We have the following results.

Theorem 8.5, Higham

$$(\widehat{L} + \Delta L)\widehat{y} = b, \quad |\Delta L| \leq \gamma_n |\widehat{L}|$$
$$(\widehat{U} + \Delta U)\widehat{x} = \widehat{y}, \quad |\Delta U| \leq \gamma_n |\widehat{U}|$$

Theorem 9.4, Higham

$$(A + \Delta A)\widehat{x} = b, \quad |\Delta A| \leq \gamma_{3n} |\widehat{L}||\widehat{U}| \Rightarrow \|\Delta A\| \leq \gamma_{3n} \||\widehat{L}||\widehat{U}|\|$$

- Solving $Ax = b$ via LU factorization is therefore stable as long as $\||\widehat{L}||\widehat{U}|\| \approx \|A\|$.
- Is that a reasonable assumption? In general, **NO!**

$$\ell_{ik} = \left(a_{ik} - \sum_{j=1}^{k-1} \ell_{ij} u_{jk} \right) / u_{kk}$$

$\Rightarrow$ u_{kk} $(= a_{kk})$ should not be too small!

$\Rightarrow$ **use pivoting:** $PAQ = LU$, where P and Q are permutation matrices

LU factorization with pivoting

```
for k = 1: n do
    Select next pivot a_rc
    Swap rows r and k
    Swap columns c and k
    for i = k + 1: n do
        a_ik = a_ik / a_kk
    end for
    for i = k + 1: n do
        for j = k + 1: n do
            a_ij ← a_ij − a_ik a_kj
        end for
    end for
end for
```

How to choose pivot a_{rc} at step k?

- Complete pivoting: choose maximum in the entire trailing matrix
  $|a_{rc}| = \max_{i=k+1:n,\ j=k+1:n} |a_{ij}|$
  $\Rightarrow$ **too expensive**
- Partial column pivoting: choose maximum in the kth column $|a_{rc}| = |a_{rk}| = \max_{i=k+1:n} |a_{ik}|$
  $\Rightarrow$ **widely used**, in practice the ratio $\rho_n = \frac{\|\widehat{L}\|\|\widehat{U}\|}{\|A\|}$ is almost always small ($\lesssim 10$).
  However there exists pathological matrices for which $\rho_n \propto 2^n$

Arithmetic intensity LU factorization

```
for k = 1: n do
    for i = k + 1: n do
        a_ik = a_ik / a_kk
    end for
    for i = k + 1: n do
        for j = k + 1: n do
            a_ij ← a_ij − a_ik a_kj
        end for
    end for
end for
```

- Arithmetic intensity: $\frac{\#\text{flops}}{\text{memory accesses}}$
- $a_{ij} \leftarrow a_{ij} - a_{ik}a_{kj} \Rightarrow n^3/3$ entries loaded from memory to cache $\Rightarrow$ arithmetic intensity not so good!

Arithmetic intensity LU factorization

for $k = 1 : n/b$ **do**
  Factorize $L_{kk}U_{kk} = A_{kk}$
  for $i = k+1 : n/b$ **do**
    Solve $A_{ik} = A_{ij}U_{kk}^{-1}$
    Solve $A_{ki} = L_{kk}^{-1}A_{ki}$
  end for
  for $i = k+1 : n/b$ **do**
    for $j = k+1 : n/b$ **do**
      $A_{ij} \leftarrow A_{ij} - A_{ik}A_{kj}$
    end for
  end for
end for

- Arithmetic intensity: $\frac{\#\text{flops}}{\text{memory accesses}}$
- $a_{ij} \leftarrow a_{ij} - a_{ik}a_{kj} \Rightarrow n^3/3$ entries loaded from memory to cache $\Rightarrow$ arithmetic intensity not so good!

$\Rightarrow$ **use blocking:** $A_{ij} \leftarrow A_{ij} - A_{ik}A_{kj}$ where A_{xy} are $b \times b$ blocks

$\Rightarrow$ $\frac{n^3}{3b}$ entries loaded $\Rightarrow$ intensity increased by a factor b

- Take b as large as possible such that 3 $b \times b$ blocks fit into the cache

Arithmetic intensity LU factorization

for $k = 1 : n/b$ **do**
  Factorize $L_{kk}U_{kk} = A_{kk}$
  for $i = k+1 : n/b$ **do**
    Solve $A_{ik} = A_{ij}U_{kk}^{-1}$
    Solve $A_{ki} = L_{kk}^{-1}A_{ki}$
  end for
  for $i = k+1 : n/b$ **do**
    for $j = k+1 : n/b$ **do**
      $A_{ij} \leftarrow A_{ij} - A_{ik}A_{kj}$
    end for
  end for
end for

- Arithmetic intensity: $\frac{\#\text{flops}}{\text{memory accesses}}$
- $a_{ij} \leftarrow a_{ij} - a_{ik}a_{kj} \Rightarrow n^3/3$ entries loaded from memory to cache $\Rightarrow$ arithmetic intensity not so good!

$\Rightarrow$ **use blocking:** $A_{ij} \leftarrow A_{ij} - A_{ik}A_{kj}$ where A_{xy} are $b \times b$ blocks

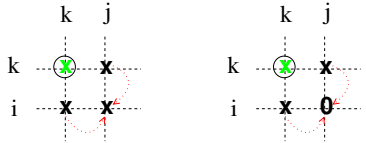
$\Rightarrow$ $\frac{n^3}{3b}$ entries loaded $\Rightarrow$ intensity increased by a factor b

- Take b as large as possible such that 3 $b \times b$ blocks fit into the cache
- Side-effect: if update is evaluated as $Temp = A_{ik}A_{kj}$, $A_{ij} \leftarrow A_{ij} - Temp$, then the constant in the error bound is reduced from n to $n/b + b$ (Lecture 2!)

Dense systems

Sparse systems
- Fill-in characterization
- The multifrontal method
- Nested dissection and complexity
- Parallelism
- Memory
- Exercise

- At step k, for $i, j > k$: $a'_{ij} = a_{ij} - a_{ik}a_{kj}$
- 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:

$$\begin{pmatrix} X & X & X & X & X \\ X & X & 0 & 0 & 0 \\ X & 0 & X & 0 & 0 \\ X & 0 & 0 & X & 0 \\ X & 0 & 0 & 0 & X \end{pmatrix} \quad 1 \leftrightarrow 5 \quad \begin{pmatrix} X & 0 & 0 & 0 & X \\ 0 & X & 0 & 0 & X \\ 0 & 0 & X & 0 & X \\ 0 & 0 & 0 & X & X \\ X & X & X & X & X \end{pmatrix}$$

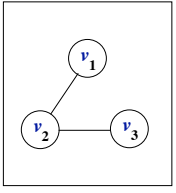
Challenges:

- How to predict where fill-in will occur?
- How to reduce the amount of fill-in? What is the complexity of the factorization?
- Knowing the fill-in, how to efficiently factorize the matrix? How to parallelize it?

Modeling fill-in: adjacency graphs

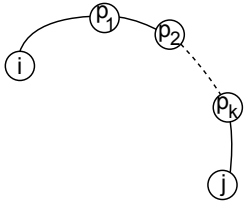
- The rows/columns and nonzeros of a given sparse matrix correspond (with natural labelling) to the vertices and edges, respectively, ofA a graph, called the **adjacency graph**.
- We will only consider on matrices with a symmetric pattern (numerical values can be unsymmetric), which means the adjacency graph is undirected.
- Matrices with unsymmetric pattern can still be handled by considering the pattern of $A + A^T$

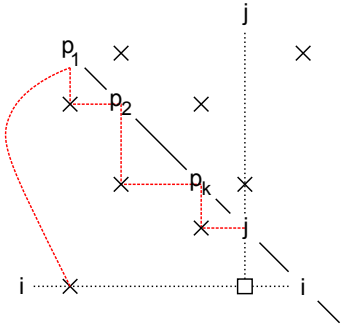
$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{pmatrix} & \times & \\ \times & \times & \times \\ & \times & \times \end{pmatrix} \end{array}$$



Fill path theorem (Rose, Tarjan, Lueker, 1976)

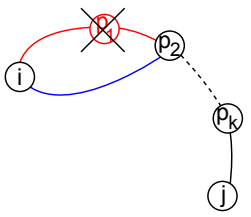
Let A be a symmetric pattern matrix, $G(A)$ its associated graph, and L the matrix of factors. Then $\ell_{ij} \neq 0$ iff there is a path in $G(A)$ between i and j such that all nodes in the path have indices smaller than both i and j.

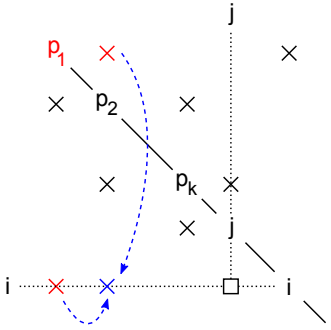




Fill path theorem (Rose, Tarjan, Lueker, 1976)

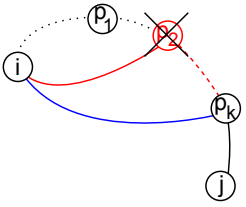
Let A be a symmetric pattern matrix, $G(A)$ its associated graph, and L the matrix of factors. Then $\ell_{ij} \neq 0$ iff there is a path in $G(A)$ between i and j such that all nodes in the path have indices smaller than both i and j.

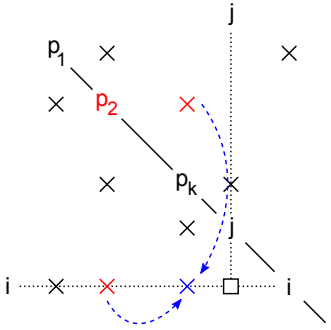




Fill path theorem (Rose, Tarjan, Lueker, 1976)

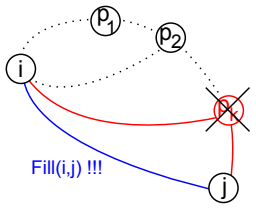
Let A be a symmetric pattern matrix, $G(A)$ its associated graph, and L the matrix of factors. Then $\ell_{ij} \neq 0$ iff there is a path in $G(A)$ between i and j such that all nodes in the path have indices smaller than both i and j.

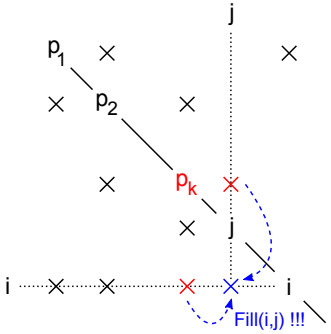




Fill path theorem (Rose, Tarjan, Lueker, 1976)

Let A be a symmetric pattern matrix, $G(A)$ its associated graph, and L the matrix of factors. Then $\ell_{ij} \neq 0$ 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.

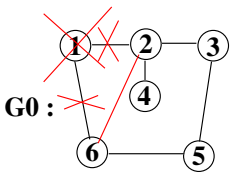


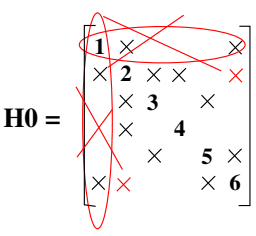


The elimination process in the graphs

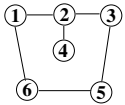
$G_U(V, E) \leftarrow$ undirected graph of A
for $k = 1 : n - 1$ **do**
  $V \leftarrow V - \{k\}$ ▷ remove vertex k
  $E \leftarrow E - \{(k, \ell) : \ell \in \mathrm{adj}(k)\} \cup \{(x, y) : x \in \mathrm{adj}(k) \text{ and } y \in \mathrm{adj}(k)\}$
  $G_k \leftarrow (V, E)$ ▷ for definition
end for

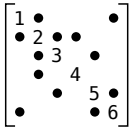
G_k are the so-called elimination graphs.

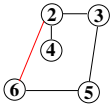


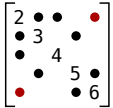


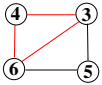
A sequence of elimination graphs

G_0: 

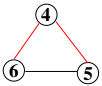
$A_0 =$ 

G_1: 

$A_1 =$ 

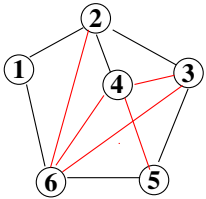
G_2: 

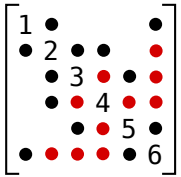
$A_2 =$ 

G_3: 

$A_3 =$ 

Filled graph $G(F) = G(L + U)$



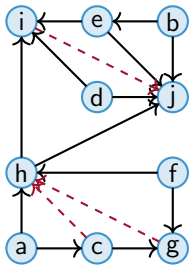


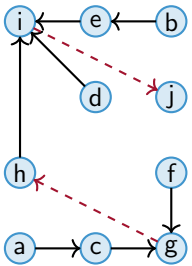
Dependency

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

How to represent dependencies in a compact way? Remove all redundant dependencies in $G(F)$ via transitive reduction.

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



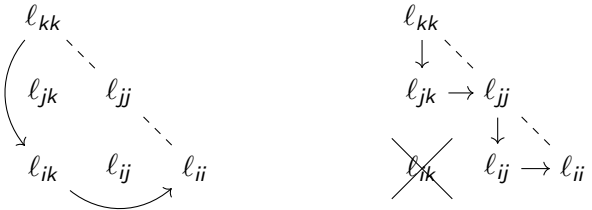


How can we easily identify redundant dependencies?

Proposition

Let $i > j > k$; if $k \to j$ and $k \to i$, necessarily $j \to i$

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



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

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).

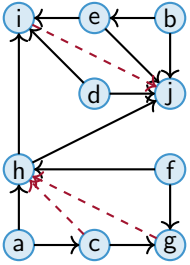
The Elimination Tree (Schreiber, 1982)

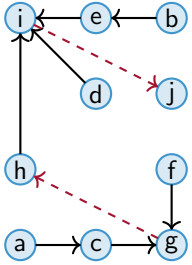
Let A be a sparse symmetric pattern matrix of order n with factor L. The elimination tree is defined to be the structure with n nodes $\{1, 2, ..., n\}$ such that the node p is the parent of node j if and only if

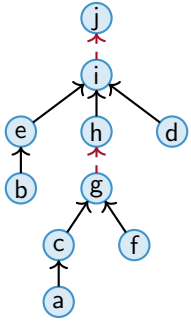
$$p = \min\{i > j \mid \ell_{ij} \neq 0\}.$$

The elimination tree

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







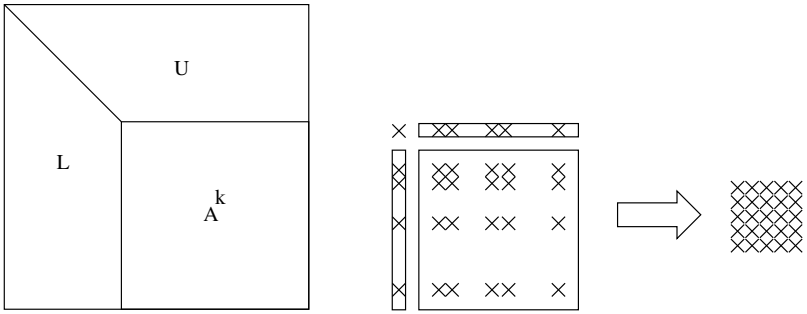
Topological order

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).

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

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 $\boxplus$ 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:

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

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

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

1 Frontal Matrix Assembly:

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

where F_i is the frontal matrix at node i of index set $\mathcal{I}_{F_i} = \{struct(L(i:n,i))\}$, j is the number of children of i in the elimination tree and $CB1, \cdots, CB_j$ are the Contribution Blocks (Schur complements) produced when processing the children (see next slide).

2 Frontal matrix factorization

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

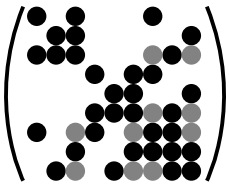
where

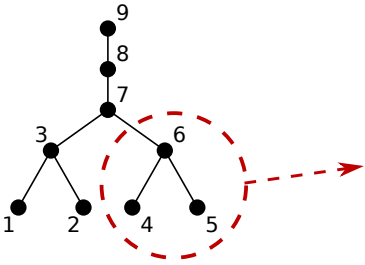
$$\begin{bmatrix} u_{11} \\ \vdots \\ u_{1m} \end{bmatrix} = \begin{bmatrix} f_{11} \\ \vdots \\ f_{1m} \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} \ell_{21} \\ \vdots \\ \ell_{m1} \end{bmatrix} = \begin{bmatrix} f_{21} \\ \vdots \\ f_{m1} \end{bmatrix} / u_{11}$$

$$CB_i = \begin{bmatrix} cb_{22} & & cb_{2m} \\ \ddots & \vdots & \vdots \\ cb_{m2} & \cdots & cb_{mm} \end{bmatrix} = \begin{bmatrix} f_{22} & & f_{2m} \\ \ddots & \vdots & \vdots \\ f_{m2} & \cdots & f_{mm} \end{bmatrix} - \begin{bmatrix} \ell_{21} \\ \vdots \\ \ell_{m1} \end{bmatrix} \cdot \begin{bmatrix} u_{12} \\ \vdots \\ u_{1m} \end{bmatrix}^T$$

The Multifrontal Method: example

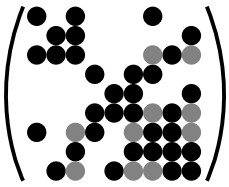
The elimination tree can be regarded as a graph of dependencies which defines where/how to assemble the elimination blocks and which variable to eliminate at each step.

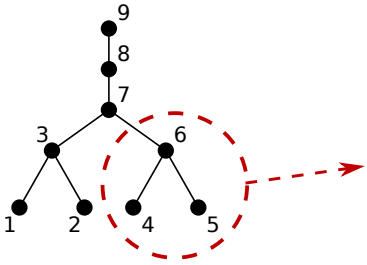




The Multifrontal Method: example

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.

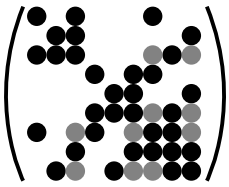


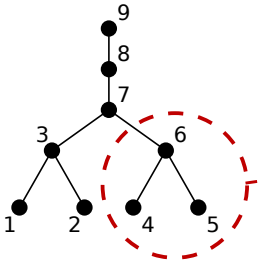


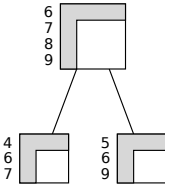
$$\begin{bmatrix} a_{44} & a_{46} & a_{47} \\ a_{64} & 0 & 0 \\ a_{74} & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} u_{44} & u_{46} & u_{47} \\ \ell_{64} & b_{66} & b_{67} \\ \ell_{74} & b_{76} & b_{77} \end{bmatrix}$$

The Multifrontal Method: example

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.

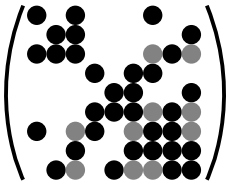


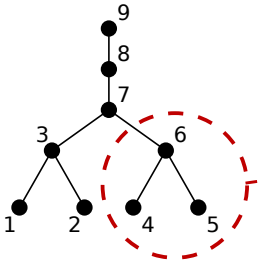


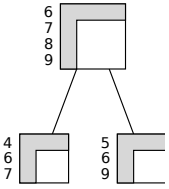


$$\begin{bmatrix} a_{55} & a_{56} & a_{59} \\ a_{65} & 0 & 0 \\ a_{95} & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} u_{55} & u_{56} & u_{59} \\ \ell_{65} & c_{66} & c_{69} \\ \ell_{95} & c_{96} & c_{99} \end{bmatrix}$$

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.







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

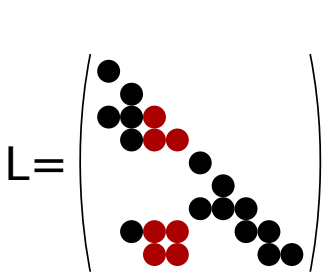
The whole factorization is recast into a sequence of partial dense factorizations of the type:

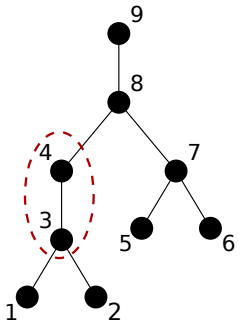
$$\begin{bmatrix} u_{11} \\ \vdots \\ u_{1m} \end{bmatrix} = \begin{bmatrix} f_{11} \\ \vdots \\ f_{1m} \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} \ell_{21} \\ \vdots \\ \ell_{m1} \end{bmatrix} = \begin{bmatrix} f_{21} \\ \vdots \\ f_{m1} \end{bmatrix} / u_{11}$$

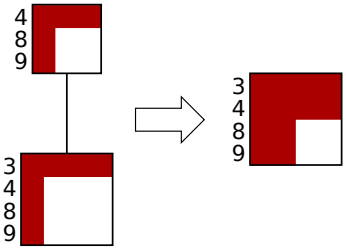
$$CB_i = \begin{bmatrix} cb_{22} & & cb_{2m} \\ \ddots & \vdots & \vdots \\ cb_{m2} & \cdots & cb_{mm} \end{bmatrix} = \begin{bmatrix} f_{22} & & f_{2m} \\ \ddots & \vdots & \vdots \\ f_{m2} & \cdots & f_{mm} \end{bmatrix} - \begin{bmatrix} \ell_{21} \\ \vdots \\ \ell_{m1} \end{bmatrix} \cdot \begin{bmatrix} u_{12} \\ \vdots \\ u_{1m} \end{bmatrix}^T$$

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

Tree Amalgamation

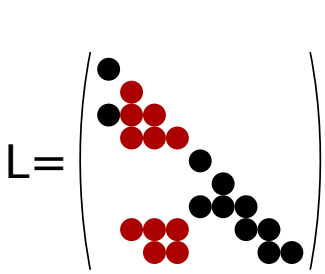


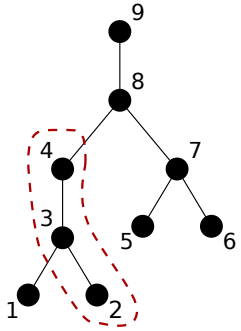


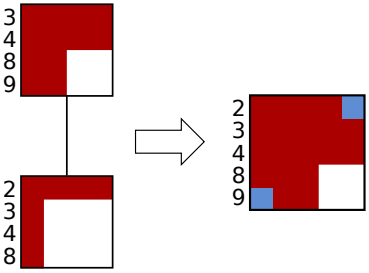


Amalgamation without fill-in consists in merging all the frontal matrices related to pivots whose columns in the factor L have the same structure. The subset of nodes containing this pivots is called a supernode. All the pivots in a supernode can thus be eliminated at once within the same frontal matrix

Tree Amalgamation

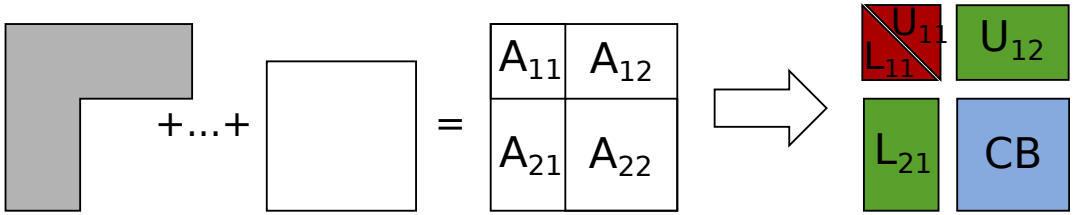






Amalgamation with fill-in is based on the same principle except that it groups together pivots whose column structure in L is not exactly the same. If the generated fill-in does not exceed a certain threshold, the extra cost is overcome by efficiency

After amalgamation:



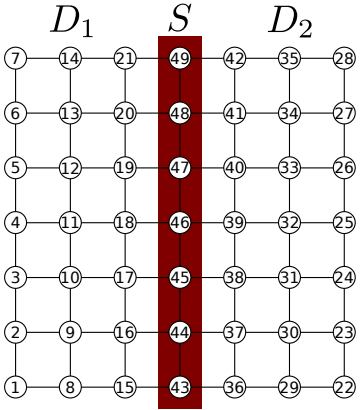
$$
\begin{aligned}
L_{11}U_{11} &= A_{11} \\
L_{21} &= A_{21}U_{11}^{-1} \\
U_{12} &= L_{11}^{-1}A_{12} \\
CB &= A_{22} - L_{21}U_{12}
\end{aligned}
$$

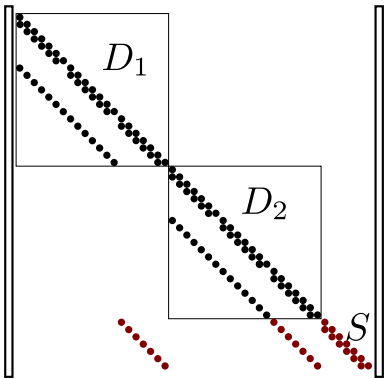
All the operations related to the frontal matrix can be done through Level-3 BLAS routines

Nested dissection

Remember, in the elimination graphs, when a node is eliminated, new edges are added that connect all its neighbors to each other.

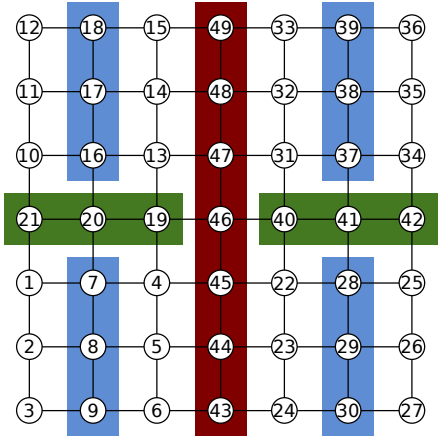
Assume a separator S is identified that partitions the domain into two subdomains D_1 and D_2. All the neighbors of nodes in D_1 (D_2) are either in D_1 (D_2) or in S. Thus, if D_1 is eliminated before D_2 and S, there cannot be any ℓ_{ij} where $v_i \in D_1$ and $v_j \in D_2$

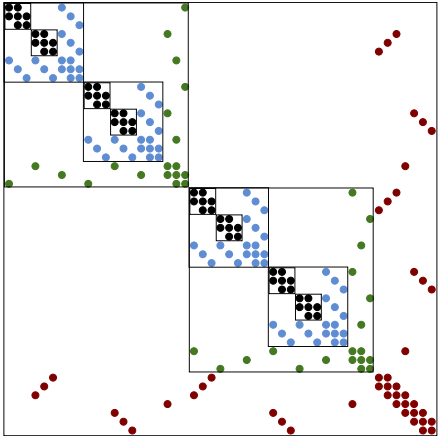




ND of a regular square mesh

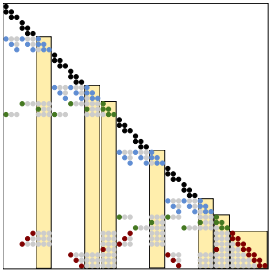
The nested dissection method aims at partitioning the domain so that the fill-in is only generated internally on each subdomain and on the interface by recursively computing bisectors.

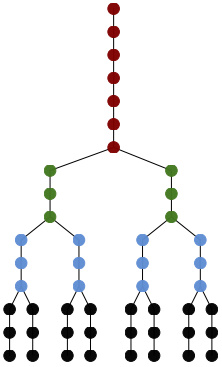


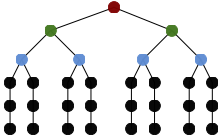


Nested dissection and elimination/assembly trees

The nested dissection method also produces an elimination tree: all the L columns of variables within the same separator have the same structure and can, therefore, be amalgamated into a supernode.





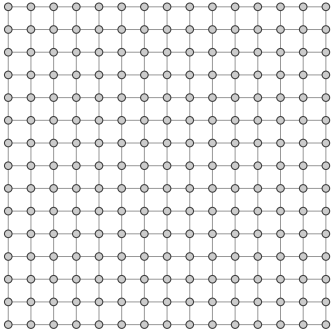


For this reason the nested dissection method provides an assembly tree implicitly defined by the tree of separators.

Complexity of the factorization with ND

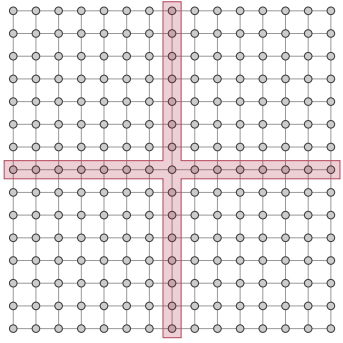
2D ND assumptions (George, 1973)

- 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



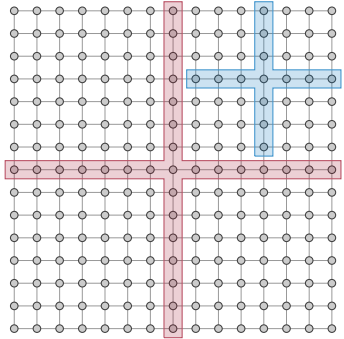
2D ND assumptions (George, 1973)

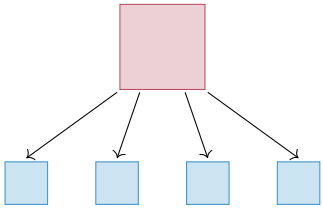
- 2D, square grid of size $N \times N$ and cross-shaped separators.
- The size of the separators/fronts is divided by 2 at every level starting at $2N$
- The number of nodes is multiplied by 4 at every level



2D ND assumptions (George, 1973)

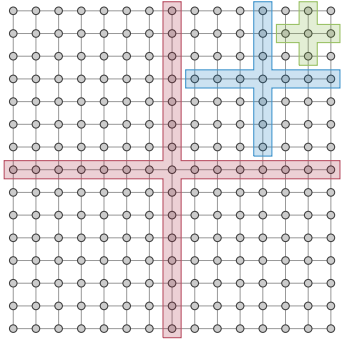
- 2D, square grid of size $N \times N$ and cross-shaped separators.
- The size of the separators/fronts is divided by 2 at every level starting at $2N$
- The number of nodes is multiplied by 4 at every level

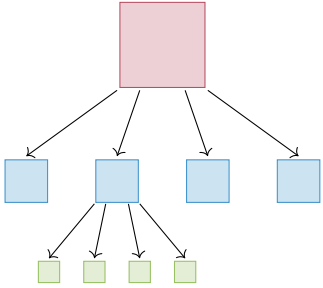




2D ND assumptions (George, 1973)

- 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





Flops

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

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

Factors size

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

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

Complexity of the factorization with ND

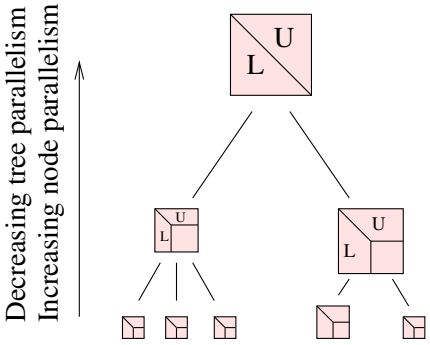
Regular problems (nested dissection)	2D $N \times N$ grid	3D $N \times N \times N$ grid
Nonzeros in original matrix	$O(N^2)$	???
Nonzeros in factors	$O(N^2 \log N)$	???
Floating-point ops	$O(N^3)$	???

- The complexities are naturally much better than if the system is handled as dense (e.g., N^6 flops) but still remain superlinear (i.e., more than N^2)
- The same analysis can be done on a 3D cubic domain of size $N \times N \times N$, with hypercross separators (union of two orthogonal planes)

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

Bound on parallelism

What is the best possible speedup we can achieve in the multifrontal method? Let's model this problem as follows:

1. We have an infinite number of processes
2. We have a regular 2D geometry partitioned with ND
3. The time is proportional to the flops.

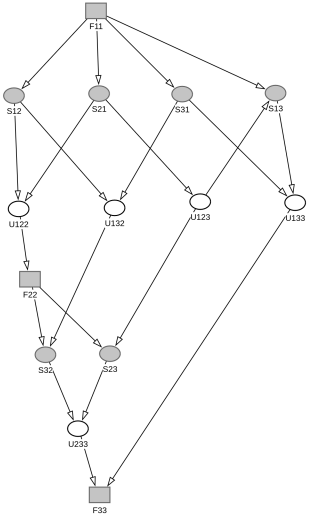
Under this assumption the sequential multifrontal factorization time is $O(N^3)$.

Bound on node parallelism

for $k = 1 : n/b$ **do**
  Factorize $L_{kk}U_{kk} = A_{kk} \rightarrow$ task F_{kk}
  for $i = k+1 : n/b$ **do**
    Solve $A_{ik} = A_{ij}U_{kk}^{-1} \rightarrow$ task S_{ik}
    Solve $A_{ki} = L_{kk}^{-1}A_{ki} \rightarrow$ task S_{ki}
  end for
  for $i = k+1 : n/b$ **do**
    for $j = k+1 : n/b$ **do**
      $A_{ij} \leftarrow A_{ij} - A_{ik}A_{kj} \rightarrow$ task U_{kij}
    end for
  end for
end for

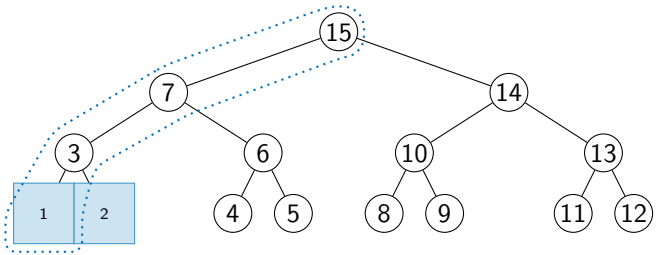
- Time sequential $\propto$ flops $\propto O(m^3)$
- Time parallel $\propto$ flops on critical path $\propto O(mb^3) = O(m)$

3×3 example:



Bound on tree 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

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

- Case 2: only node parallelism

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

- Case 3: tree and node parallelism

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

Bound on parallelism

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

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, time $\neq$ flops, etc.)

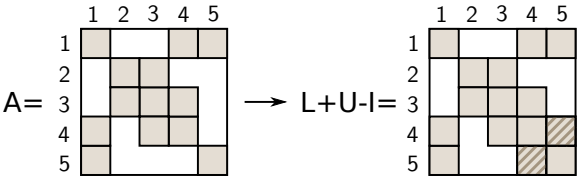
Postorder

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:

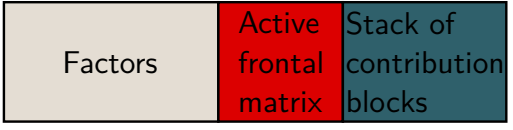
- In a sequential execution, a stack data structure can be used to store the contribution blocks. A contribution block is pushed on top of the stack when it is produced (i.e., upon factorization of the corresponding front) and popped from the top of the stack when it is assembled into the parent front
- 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.

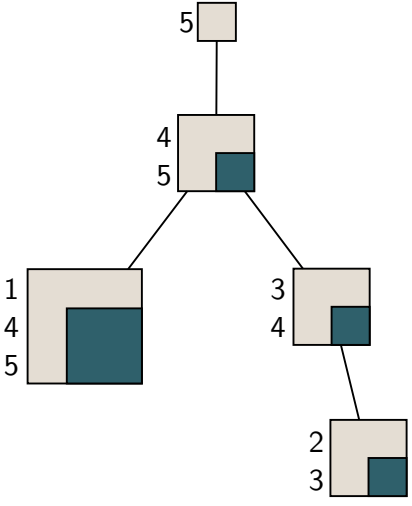
The multifrontal method: memory handling



Storage is divided into two parts:

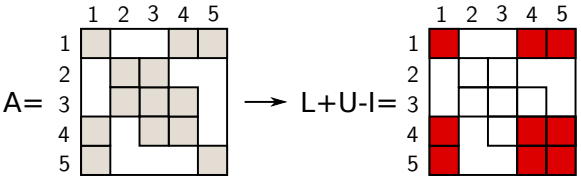
- Factors
- Active memory





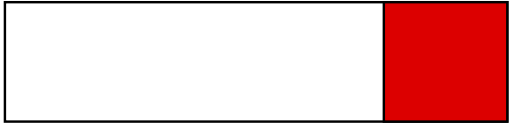
Elimination tree

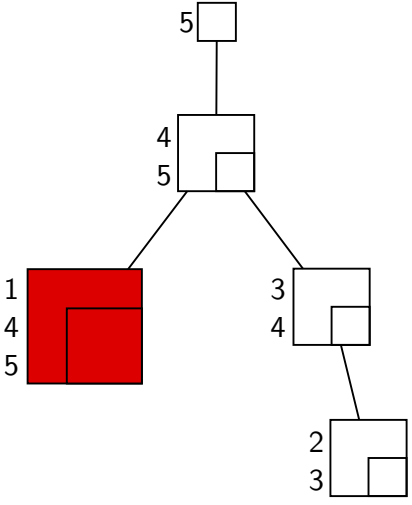
The multifrontal method: memory handling



Storage is divided into two parts:

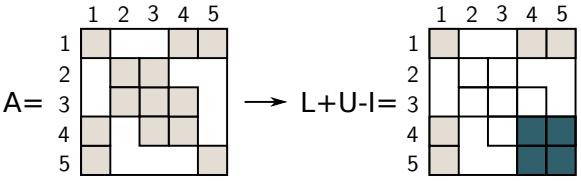
- Factors
- Active memory





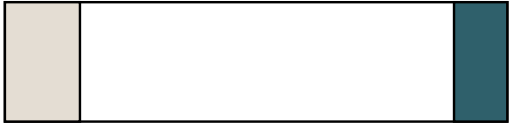
Elimination tree

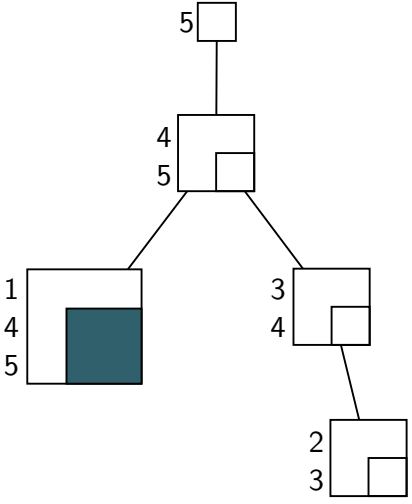
The multifrontal method: memory handling



Storage is divided into two parts:

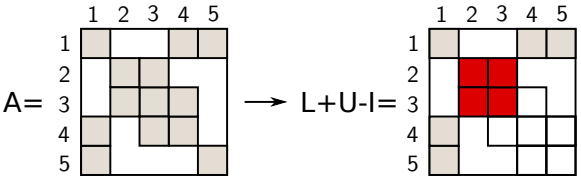
- Factors
- Active memory





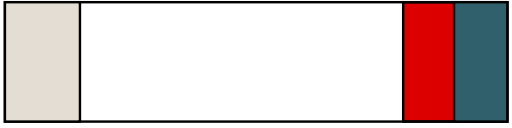
Elimination tree

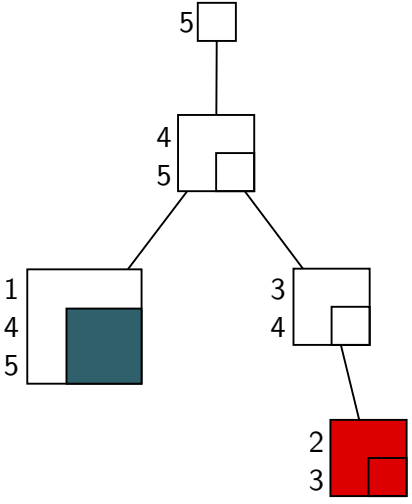
The multifrontal method: memory handling



Storage is divided into two parts:

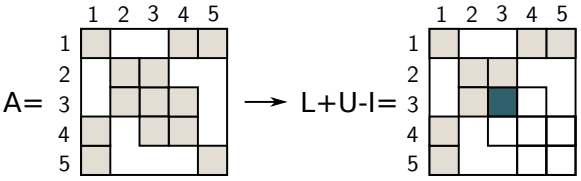
- Factors
- Active memory





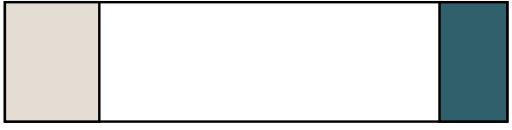
Elimination tree

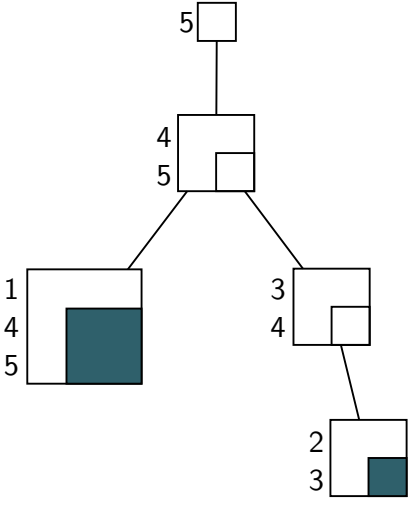
The multifrontal method: memory handling



Storage is divided into two parts:

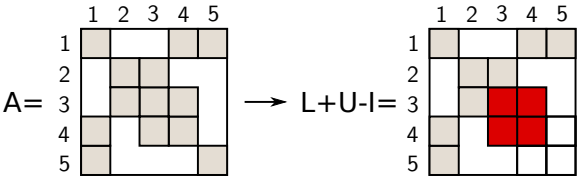
- Factors
- Active memory





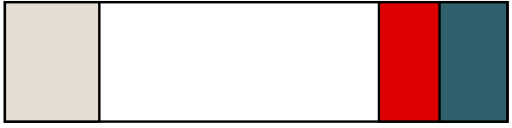
Elimination tree

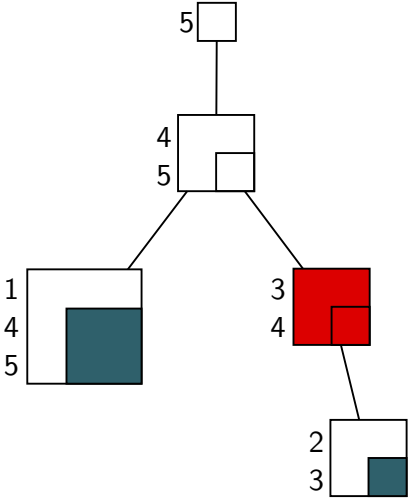
The multifrontal method: memory handling



Storage is divided into two parts:

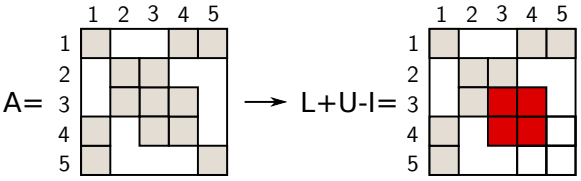
- Factors
- Active memory





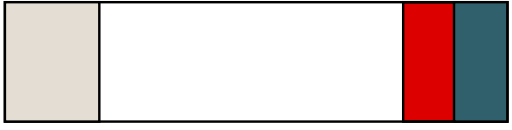
Elimination tree

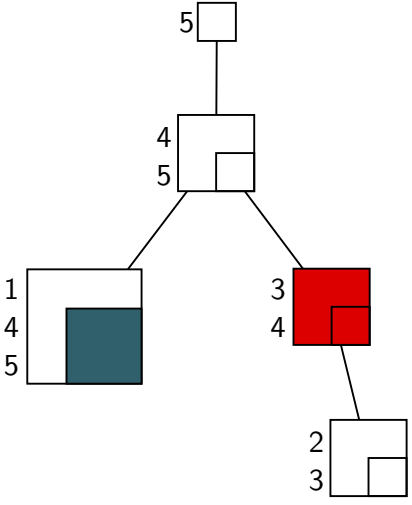
The multifrontal method: memory handling



Storage is divided into two parts:

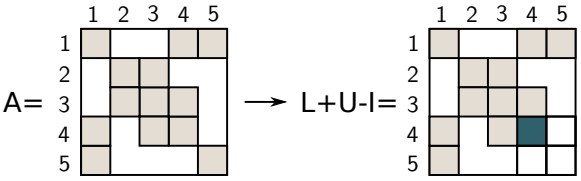
- Factors
- Active memory





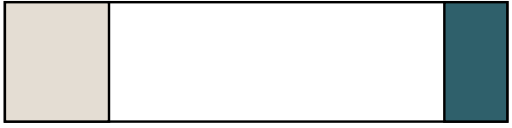
Elimination tree

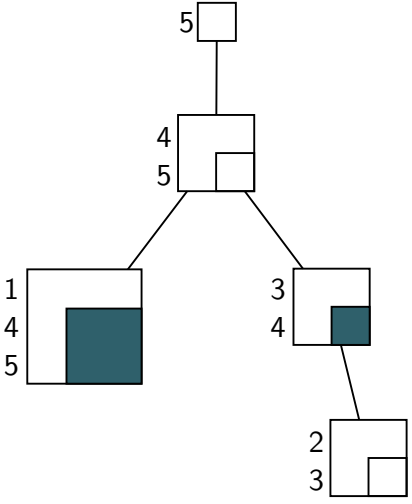
The multifrontal method: memory handling



Storage is divided into two parts:

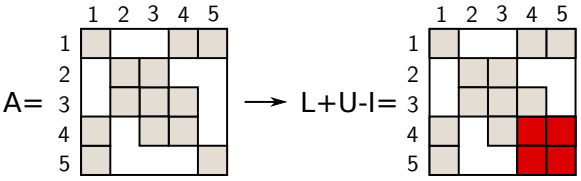
- Factors
- Active memory





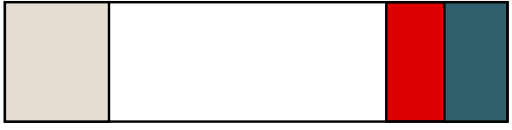
Elimination tree

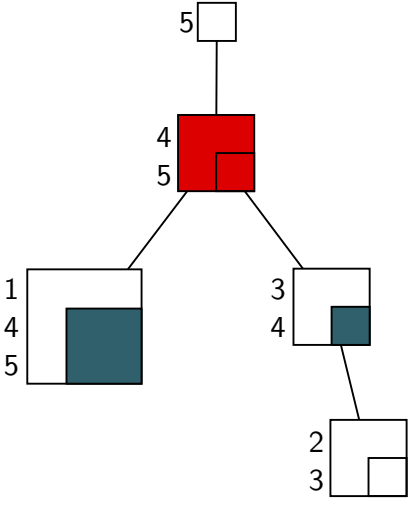
The multifrontal method: memory handling



Storage is divided into two parts:

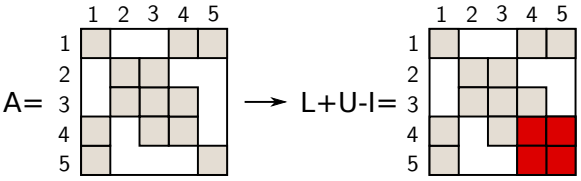
- Factors
- Active memory





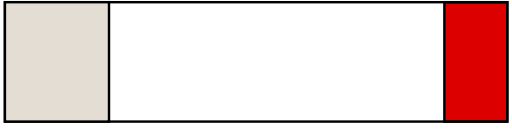
Elimination tree

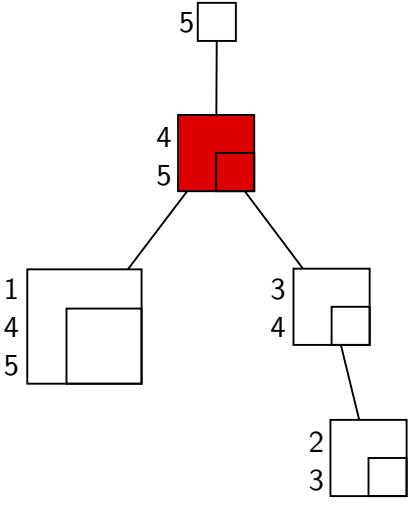
The multifrontal method: memory handling



Storage is divided into two parts:

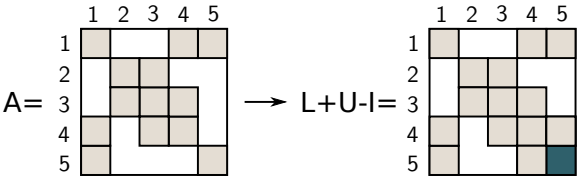
- Factors
- Active memory





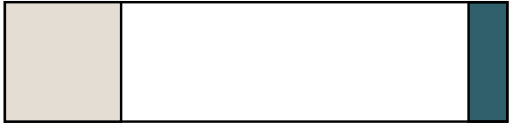
Elimination tree

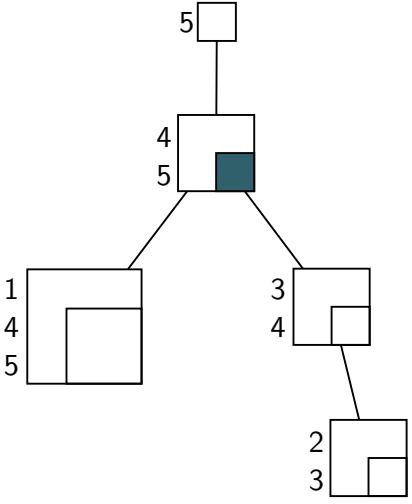
The multifrontal method: memory handling



Storage is divided into two parts:

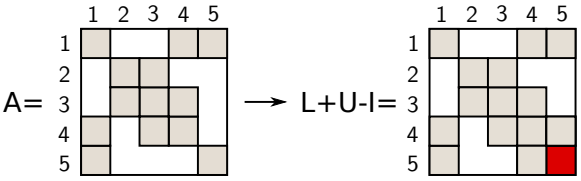
- Factors
- Active memory





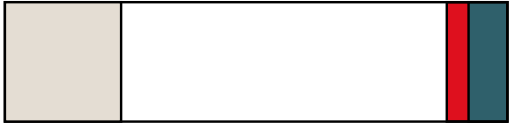
Elimination tree

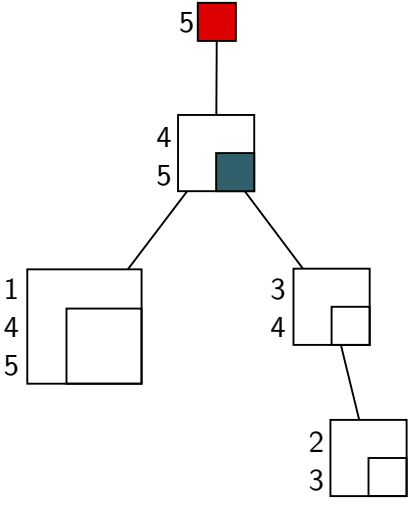
The multifrontal method: memory handling



Storage is divided into two parts:

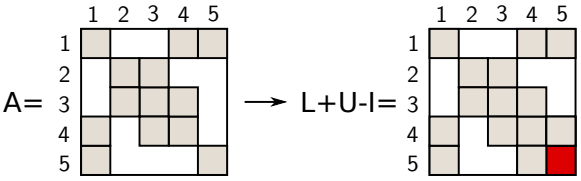
- Factors
- Active memory





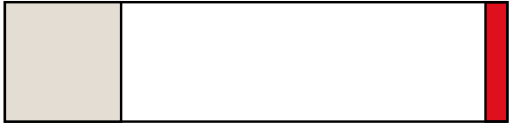
Elimination tree

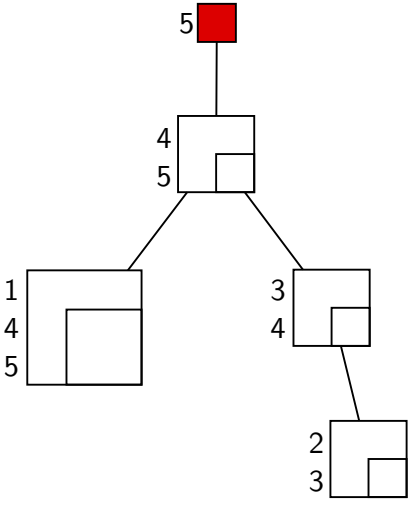
The multifrontal method: memory handling



Storage is divided into two parts:

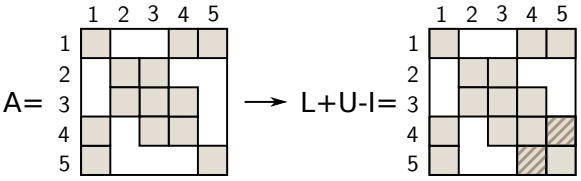
- Factors
- Active memory





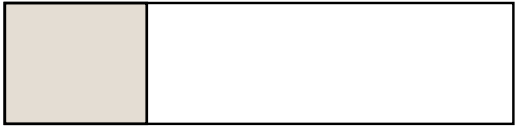
Elimination tree

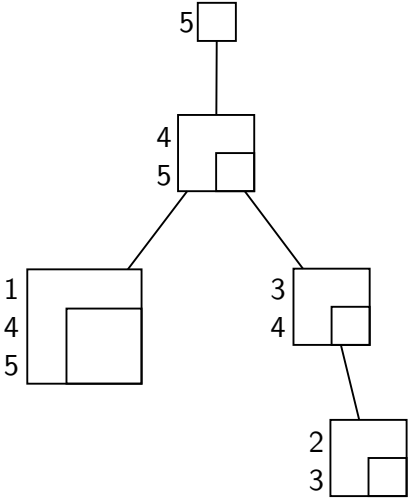
The multifrontal method: memory handling



Storage is divided into two parts:

- Factors
- Active memory

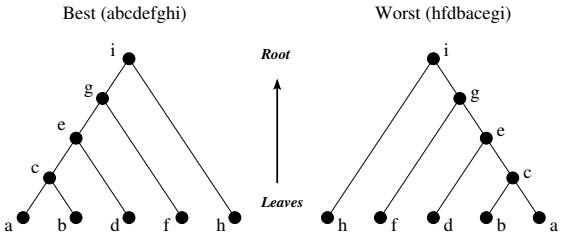




Elimination tree

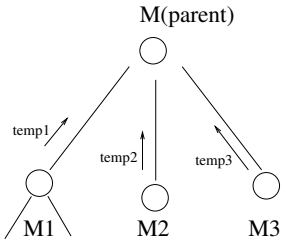
Postorder traversals: memory

Postorder provides a good data locality and better memory consumption that 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

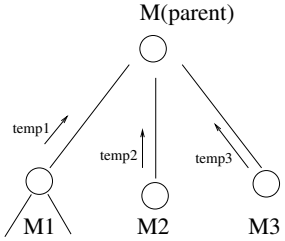
- M_i: memory peak for complete subtree rooted at i,
- CB_i: (temporary) contribution block produced at node i,
- m_i: memory for storing frontal matrix i,
- nc_i: number of children of node i.



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

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}

Theorem (Liu 1986)

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* $x_i - y_i$.

Proof. Assume a sequence S is optimal with $x_j - y_j < x_{j+1} - y_{j+1}$ for some j. Let the sequence S' be obtained by interchanging j and $j+1$. Let M_k and M'_k be the peaks at step k of S and S'. Then

- $M_k = M'_k$ for $k \neq j, j+1$
- $M'_j < M_{j+1}$ and $M'_{j+1} < M_{j+1}$

Theorem (Liu 1986)

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* $x_i - y_i$.

Proof. Assume a sequence S is optimal with $x_j - y_j < x_{j+1} - y_{j+1}$ for some j. Let the sequence S' be obtained by interchanging j and $j+1$. Let M_k and M'_k be the peaks at step k of S and S'. Then

- $M_k = M'_k$ for $k \neq j, j+1$
- $M'_j < M_{j+1}$ and $M'_{j+1} < M_{j+1}$

Corollary

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.

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.

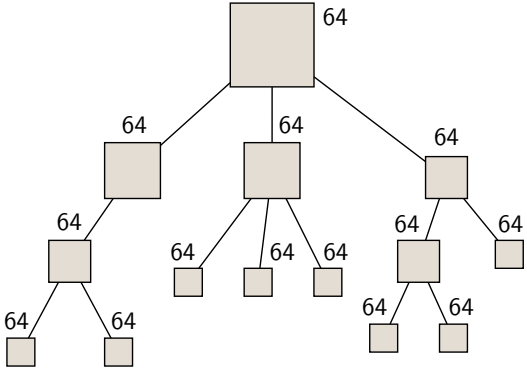
In order to asses the memory scalability we will use a metric called memory efficiency:

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

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:

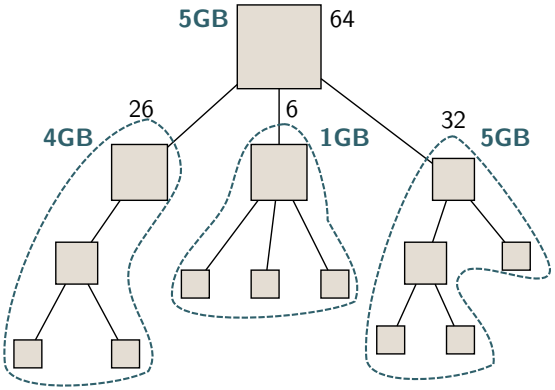


Optimal memory scalability ($S_{max}(p) = S_{seq}/p$) but no tree parallelism and prohibitive amounts of communications.

Example 2: proportional mapping

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

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



Dense systems

Sparse systems

- Fill-in characterization
- The multifrontal method
- Nested dissection and complexity
- Parallelism
- Memory
- Exercise

Exercise

- No MATLAB today! Just pen & paper (and then LaTeX!)
- Fill the ??? in the tables on slides 41 and 48
- Show your reasoning and calculations!
- Write it all up in LaTeX
- Bonus: assuming a regular 2D or 3D geometry partitioned with ND, how can we improve the constant γ_n in the backward error bound of LU factorization (slide 8) ?