iccs 2004

June 6-9, 2004
Krakow, Poland

A Fast Multifrontal Solver for Non-Linear Multi-Physics Problems

[A. Bertoldo, M. Bianco, G. Pucci]   {cyberto, bianco1, geppo} @dei.unipd.it

DEI - UNIVERSITY OF PADOVA
ACG
ADVANCED COMPUTING GROUP

Area of interest

Simulation of porous media under high temperature

Physical model

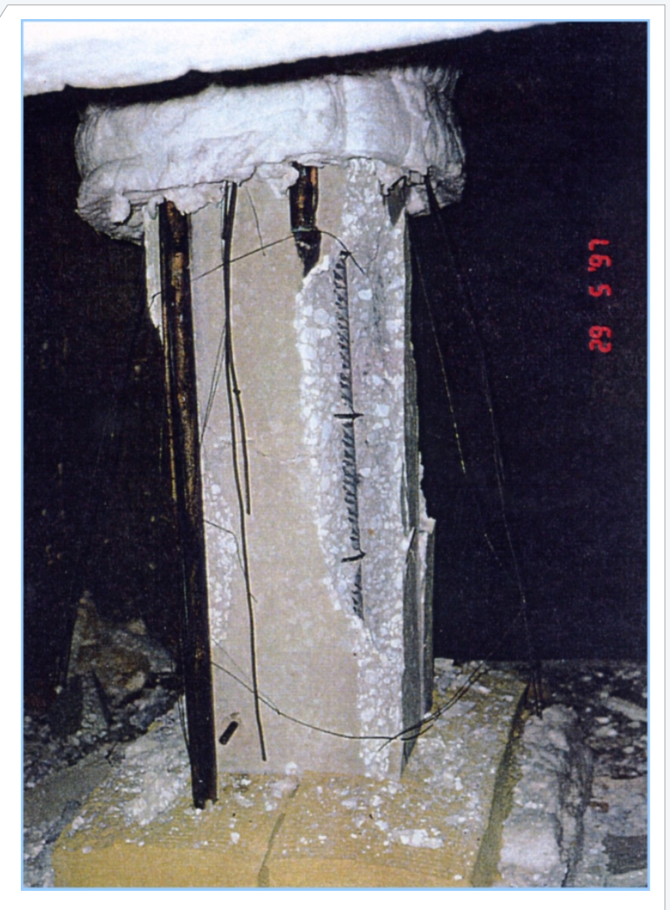
Non-linear coupled multi-physics problem

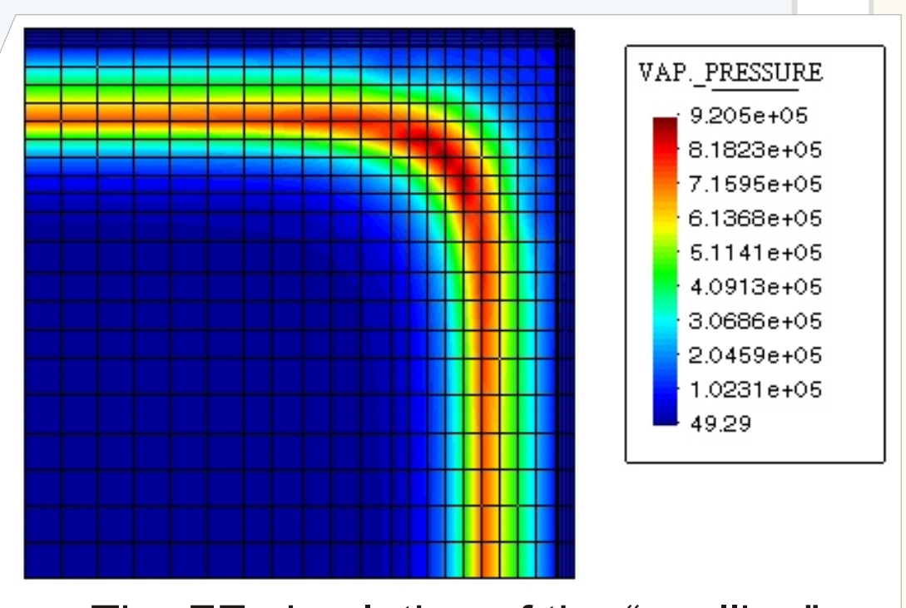
Mathematical model

Large non-linear systems of PDEs

Linearization   FEM

LARGE LINEAR SYSTEMS


The “spalling” phenomenon in a concrete-made pillar after a simulated fire.



The FE simulation of the “spalling” phenomenon in a (section of) concrete-made pillar in case of fire

Our Goal

For each time step, and for each Newton-Raphson iteration, we have to solve the linear system $\mathbf{A}\mathbf{u} = \mathbf{b}$ where:

$$a_{ij} = \sum_{\substack{e \in \{1,\dots,\eta\}:\\ \exists k,l:\\ \upsilon_e(k)=i\\ \upsilon_e(l)=j}} a^{[e]}_{k,l} \qquad b_i = \sum_{\substack{e \in \{1,\dots,\eta\}:\\ \exists k:\\ \upsilon_e(k)=i}} b^{[e]}_k$$

starting from η linear systems $\mathbf{A}^{[e]}\mathbf{u}^{[e]} = \mathbf{b}^{[e]}$ of the elements, and an index map $\upsilon_e(\cdot)$ from element-local to global indexes

Main features of our solver

- Frontal solvers are **direct methods** since they first transform the system using Gaussian elimination or **LU decomposition**, then get the final solution using forward and backward substitution.
- They do not operate on the completely assembled linear system, but rather **interleave** assembly phases with elimination phases.
- They require **low memory** space and can exploit efficient **dense linear algebra kernels**.

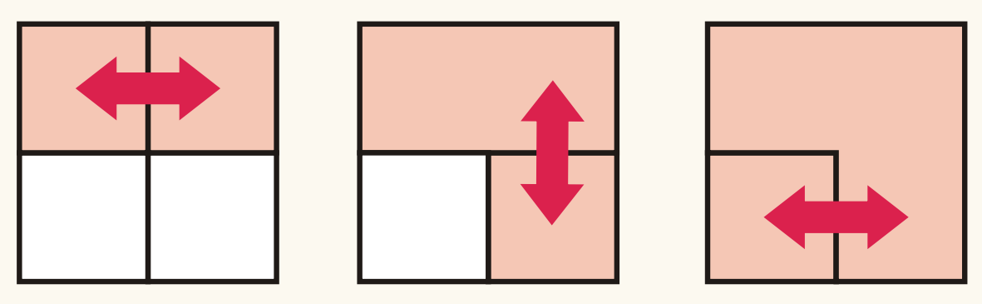
Other specific features:

- **Multifrontal** assembly/elimination strategy
- **Implicit minimum degree** pivoting
- Symbolic preprocessing phase
- **Super-assembly** phase
- Blocked LU decomposition for elimination

The multifrontal approach

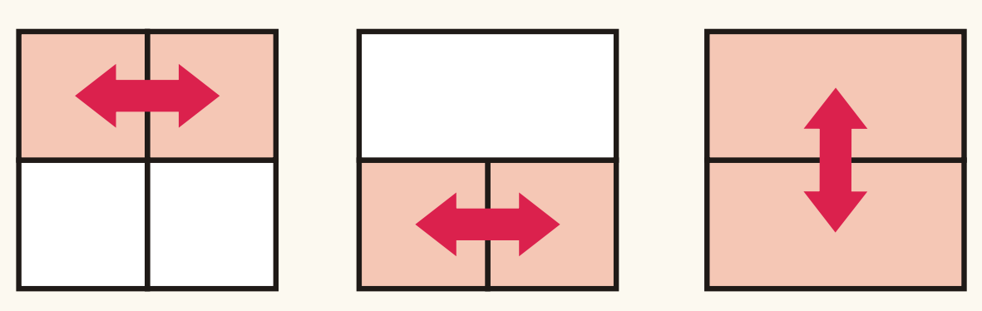
Unifrontal

Sequential assembly strategy



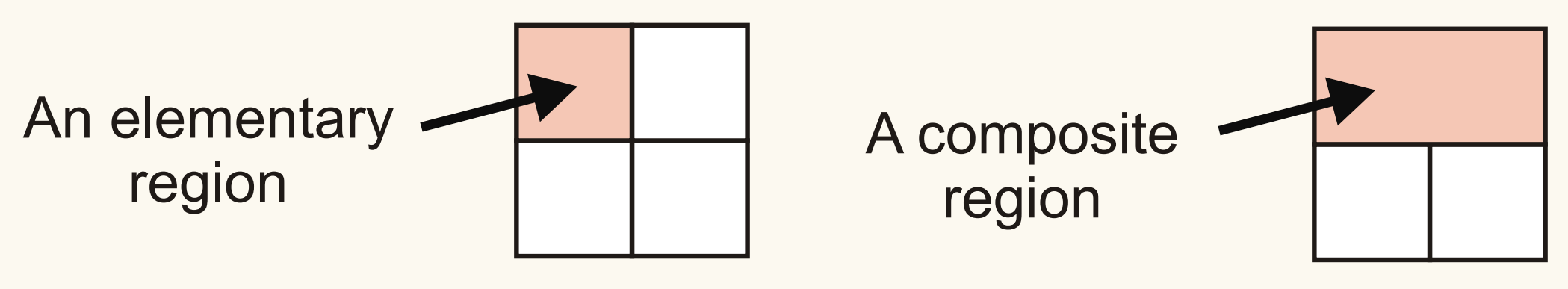
Multifrontal

Recursive assembly strategy in a bottom-up fashion



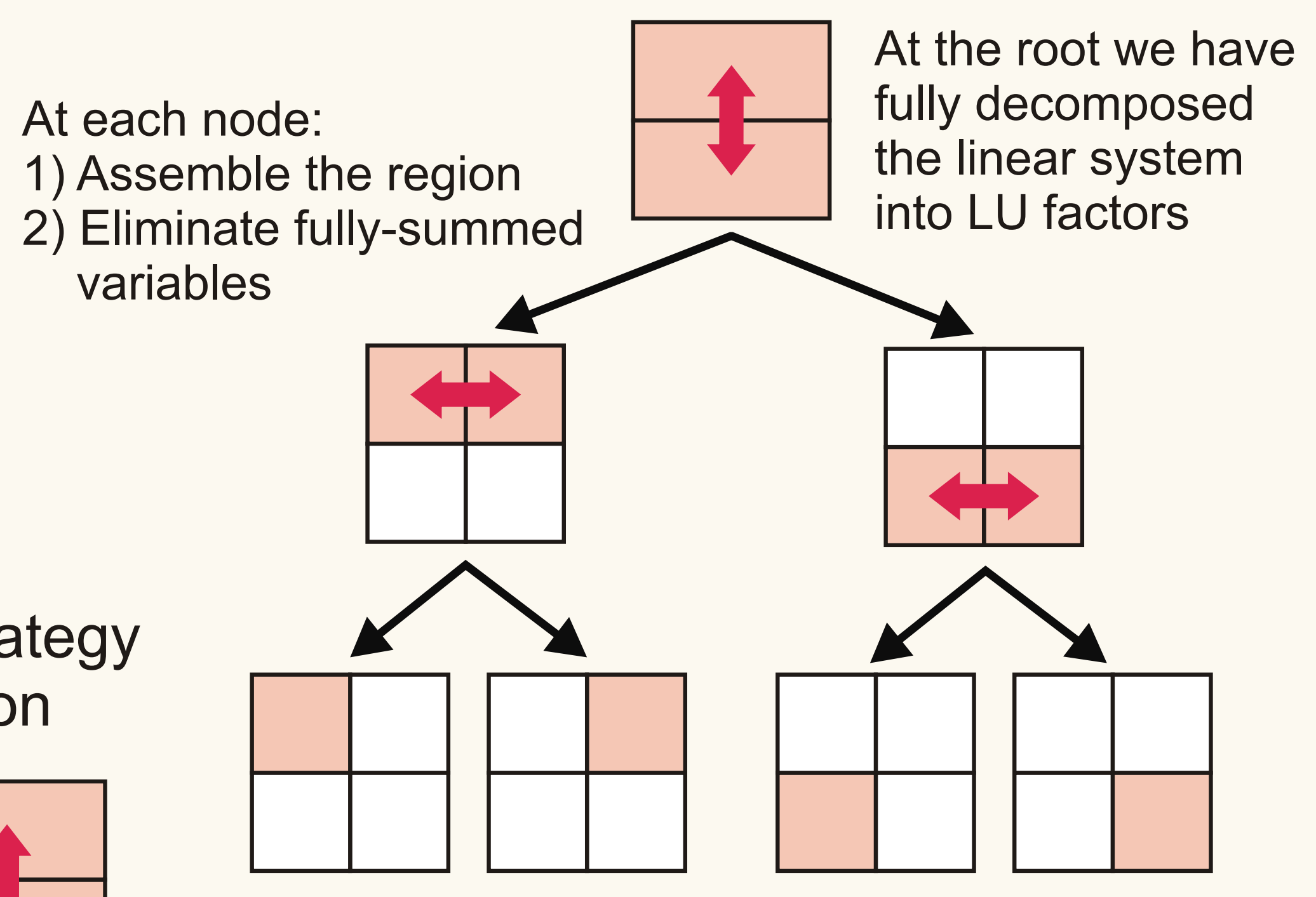
Regions

Regions can be both elementary and composite. The former are obtained from the finite element formulation. The latter are unions of two component regions from an assembly phase.

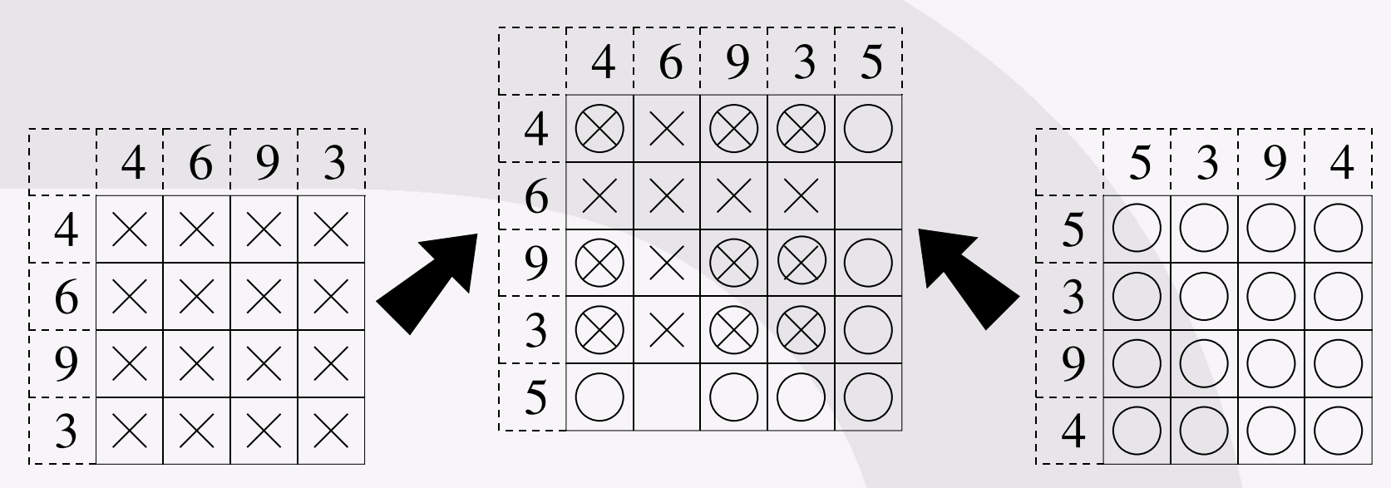


Mesh region E ⟷ $\mathbf{A}^{[E]}\mathbf{u}^{[E]} = \mathbf{b}^{[E]}$

The assembly tree


At each node:
1) Assemble the region
2) Eliminate fully-summed variables
At the root we have fully decomposed the linear system into LU factors
Leaves: The assembly phase gets elements from the FEM formulation

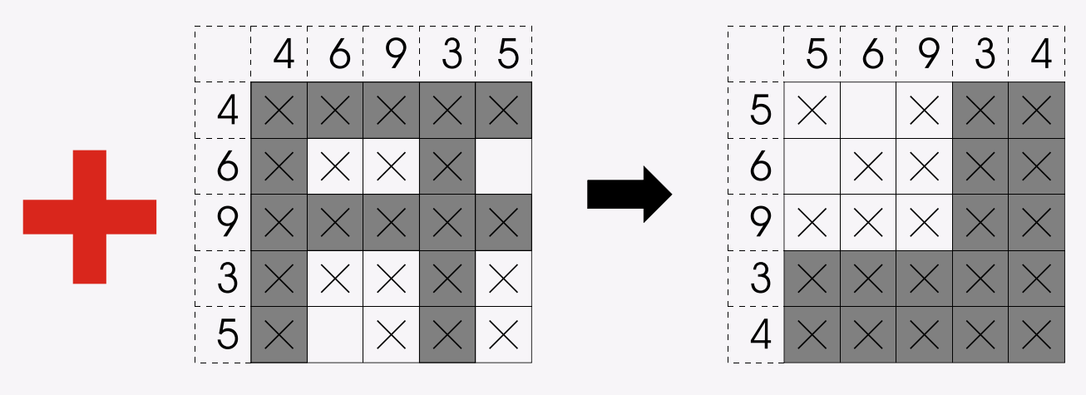
Ia. Assembly phase



Merge the two reduced components into a new composite region

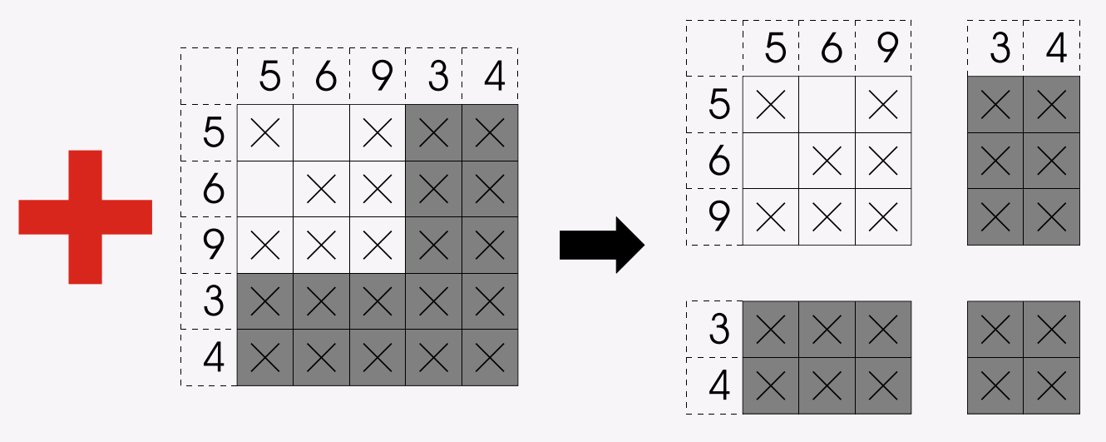
$$\mathbf{A}^{[t]} = \bar{\mathbf{A}}^{[\mathrm{lx}(t)]} \oplus \bar{\mathbf{A}}^{[\mathrm{rx}(t)]}$$

Ib. Swap phase



Pack FS rows and columns at the bottom-right corner of the non-reduced region

Ic. Copy phase



Copy FS blocks into temporary buffers

= Super-assembly phase

II. Elimination phase

$$\mathbf{A}^{[t]} = \left[\begin{array}{c|c} \mathbf{N} & \mathbf{R} \\ \hline \mathbf{C} & \mathbf{S} \end{array}\right] = \left[\begin{array}{c|c} \mathbf{N}' & \mathbf{U}' \\ \hline \mathbf{0} & \mathbf{U} \end{array}\right] \cdot \left[\begin{array}{c|c} \mathbf{I} & \mathbf{0} \\ \hline \mathbf{L}' & \mathbf{L} \end{array}\right]$$

Eliminate FS variables using a blocked UL decomposition

where

$$\mathbf{U}' = \mathbf{R}\mathbf{L}^{-1} \qquad \mathbf{L}' = \mathbf{U}^{-1}\mathbf{C} \qquad \mathbf{N}' = \mathbf{N} - \mathbf{U}'\mathbf{L}'$$

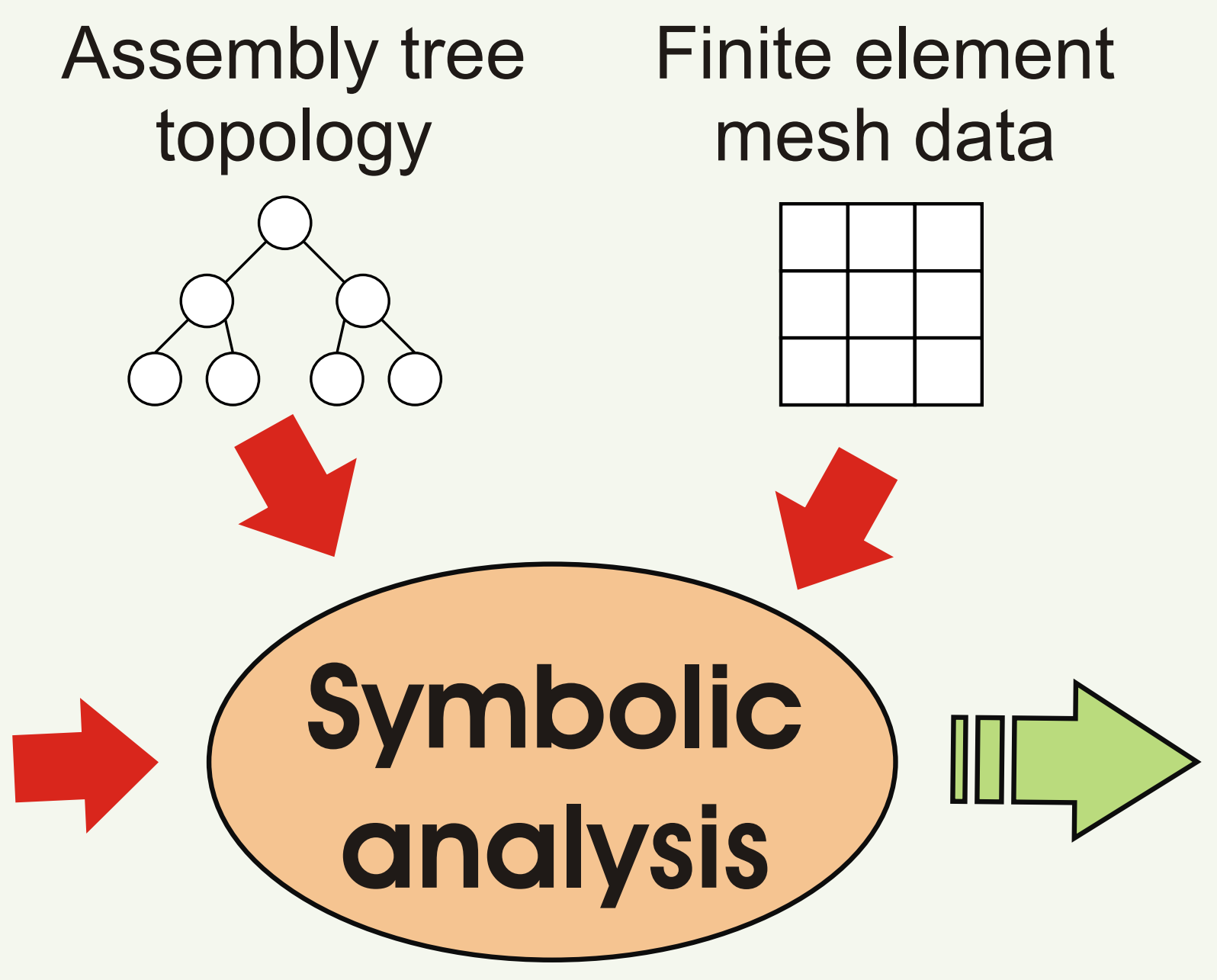
BLAS Level 3

III. Strip phase

Strip factors away for subsequent use in the final backward and forward substitution

Computation properties

- The rows and columns that become fully-summed at each node of the assembly tree are the same at every iteration of the solver for a given mesh. Therefore, we can compute the related information only once at the beginning of the simulation.
- As a consequence of the previous point, for each region, the position of each variable in the blocks $\mathbf{N}$, $\mathbf{C}$, $\mathbf{R}$, and $\mathbf{S}$ is the same at every iteration of the solver for a given mesh.
- We do not really need to explicitly assemble $\mathbf{A}^{[t]}$, rather, we can keep its sub-matrices $\mathbf{N}$, $\mathbf{C}$, $\mathbf{R}$, and $\mathbf{S}$ within separate buffers $\mathbf{B_N}$, $\mathbf{B_C}$ $\mathbf{B_R}$ and $\mathbf{B_S}$, which are obtained directly from the mesh elements for elementary regions, and from the Schür complements $\bar{\mathbf{A}}^{[\mathrm{rx}(t)]}$ and $\bar{\mathbf{A}}^{[\mathrm{lx}(r)]}$ for composite regions.



Symbolic data

These data are computed once at the beginning of the computation, but used at each iteration to perform the super-assembly phase.

For each assembly tree node t:

f_t: dimension of $\mathbf{A}^{[t]}$ (*front size*)

n_t: number of FS variables in $\mathbf{A}^{[t]}$

$\upsilon_t(i)$: $\{1,\dots,f_t\} \to \{1,\dots,n\}$
$i \mapsto$ variable at position i in $\mathbf{A}^{[t]}$

$$\gamma^{[t]}_l(k) = \begin{cases} h & \text{if } \exists h < \infty : \\ & \upsilon_t(k) = \upsilon_{\mathrm{lx}(t)}(h); \\ \infty & \text{otherwise.} \end{cases}$$

$$\gamma^{[t]}_r(k) = \begin{cases} h & \text{if } \exists h < \infty : \\ & \upsilon_t(k) = \upsilon_{\mathrm{rx}(t)}(h); \\ \infty & \text{otherwise.} \end{cases}$$

Algorithm of super-assembly phase

```
for $j = f_t - n_t + 1$ to $f_t$ do
    for $i = f_t - n_t + 1$ to $f_t$ do
        $\mathbf{B_S}(i - f_t + n_t, j - f_t + n_t) \leftarrow$
            $\bar{\mathbf{A}}^{[\mathrm{lx}(t)]}(\gamma^{[t]}_l(i), \gamma^{[t]}_l(j)) +$
            $\bar{\mathbf{A}}^{[\mathrm{rx}(t)]}(\gamma^{[t]}_r(i), \gamma^{[t]}_r(j))$
    endfor
endfor
for $j = f_t - n_t + 1$ to $f_t$ do
    for $i = 1$ to $f_t - n_t$ do
        $\mathbf{B_R}(i, j - f_t + n_t) \leftarrow$
            $\bar{\mathbf{A}}^{[\mathrm{lx}(t)]}(\gamma^{[t]}_l(i), \gamma^{[t]}_l(j)) +$
            $\bar{\mathbf{A}}^{[\mathrm{rx}(t)]}(\gamma^{[t]}_r(i), \gamma^{[t]}_r(j))$
    endfor
endfor
```

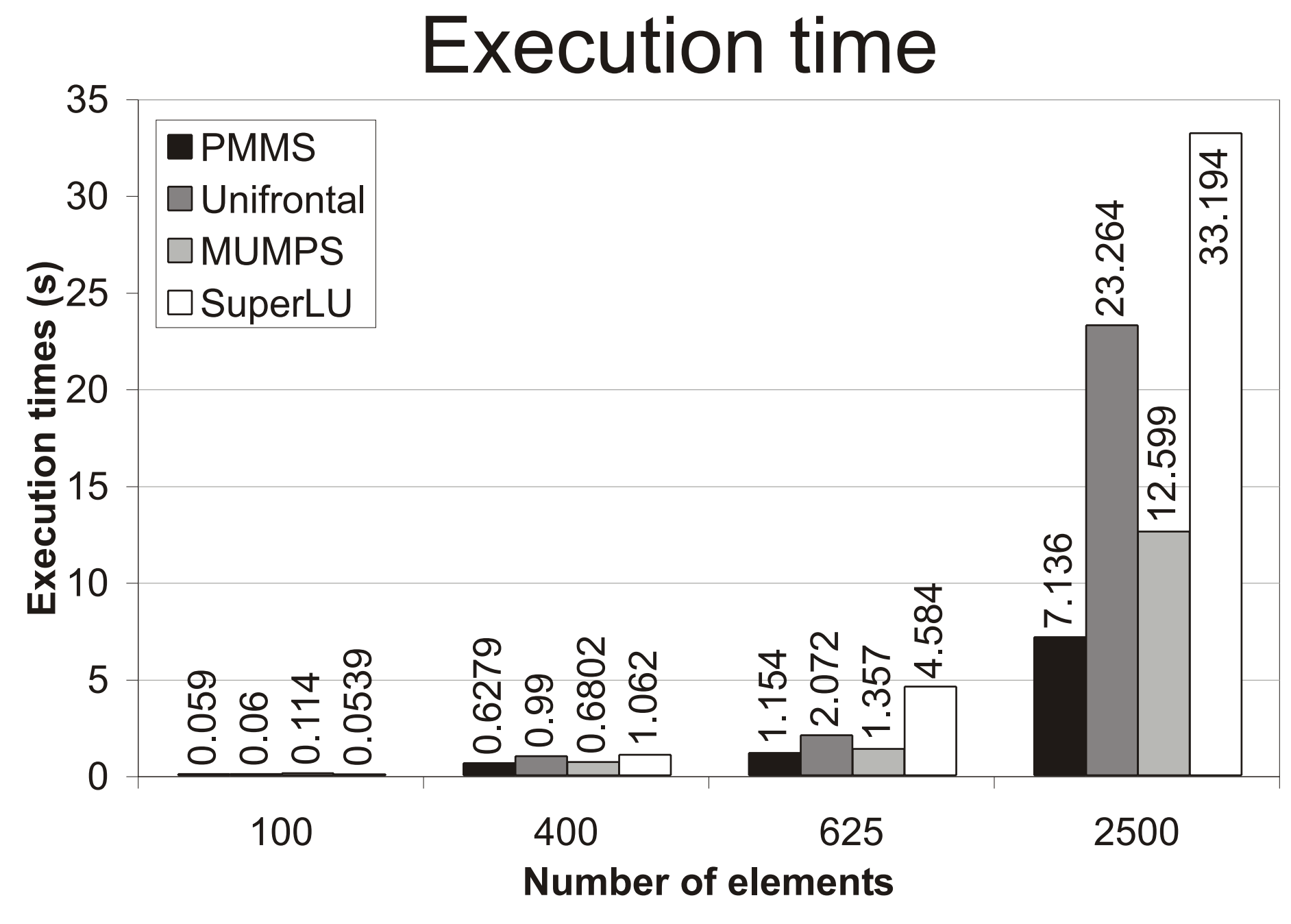
```
for $j = 1$ to $f_t - n_t$ do
    for $i = f_t - n_t + 1$ to $f_t$ do
        $\mathbf{B_C}(i - f_t + n_t, j) \leftarrow$
            $\bar{\mathbf{A}}^{[\mathrm{lx}(t)]}(\gamma^{[t]}_l(i), \gamma^{[t]}_l(j)) +$
            $\bar{\mathbf{A}}^{[\mathrm{rx}(t)]}(\gamma^{[t]}_r(i), \gamma^{[t]}_r(j))$
    endfor
endfor
for $j = 1$ to $f_t - n_t$ do
    for $i = 1$ to $f_t - n_t$ do
        $\mathbf{B_N}(i, j) \leftarrow$
            $\bar{\mathbf{A}}^{[\mathrm{lx}(t)]}(\gamma^{[t]}_l(i), \gamma^{[t]}_l(j)) +$
            $\bar{\mathbf{A}}^{[\mathrm{rx}(t)]}(\gamma^{[t]}_r(i), \gamma^{[t]}_r(j))$
    endfor
endfor
```

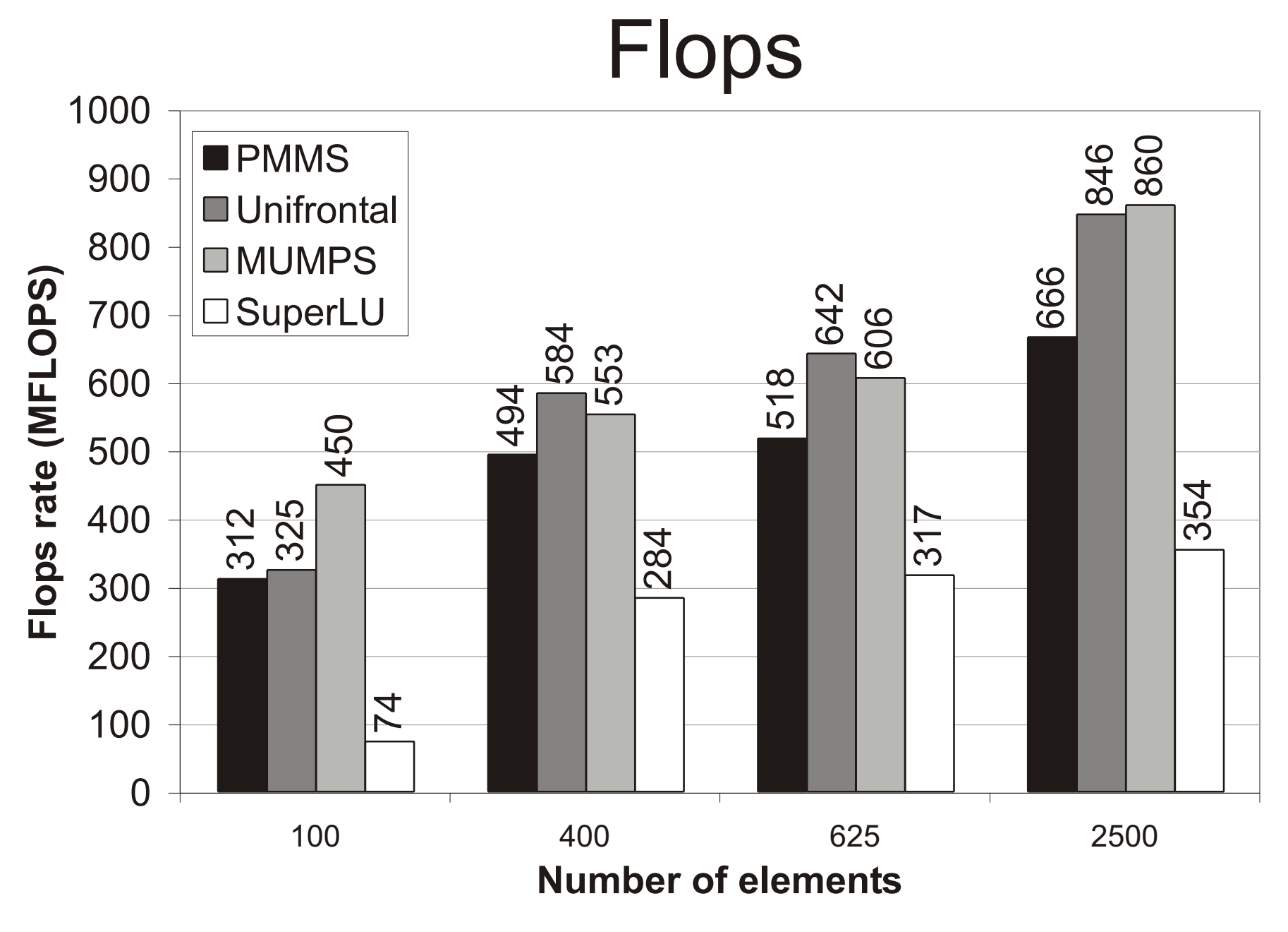
- Symbolic data are used to perform the super-assembly phase
- Sub-matrices $\mathbf{N}$, $\mathbf{C}$, $\mathbf{R}$ and $\mathbf{S}$ of $\mathbf{A}^{[t]}$ are directly computed from $\bar{\mathbf{A}}^{[\mathrm{lx}(t)]}$ and $\bar{\mathbf{A}}^{[\mathrm{rx}(t)]}$
- Sub-matrices are placed directly into buffers $\mathbf{B_N}$, $\mathbf{B_R}$, $\mathbf{B_C}$, and $\mathbf{B_S}$ to save space and reduce memory movement operations
- Buffer $\mathbf{B_N}$ reuses as much memory space as possible from the one previously allocated to $\bar{\mathbf{A}}^{[\mathrm{lx}(t)]}$
- To avoid comparisons inside loops, whenever one of the two source indexes returned by functions $\gamma^{[t]}_l$ or $\gamma^{[t]}_r$ yields ∞, then the indexed matrix entry returns zero
- To ensure correctness, the fourth loop must be executed last

Performance Results

- PMMS is our multifrontal solver
- SuperLU version 3.0
- MUMPS version 4.3

- IBM Power3@375MHz with 4 GB mem
- HPM Toolkit for performance measurements
- FE square meshes with 100, 400, 625, and 2500 square 8-node elements





PMMS is faster than Unifrontal solver, but they have the same solving kernel	The multifrontal assembly scheme is more efficient
PMMS is faster than both SuperLU and MUMPS for all significant problem sizes	The super-assembly phase and the use of BLAS boosts the computation
The larger is the problem size, the faster is PMMS with respect to the other solvers	For larger test cases we expect a bigger performance improvement
MUMPS and Unifrontal solver exhibit larger flop rates than PMMS does	They are better tuned but their algorithm has higher complexity

MIUR Center for Science and Application of Advanced Computing Paradigms at the University of Padova, Italy

Department of Information Engineering University of Padova, Italy

International Centre for Mechanical Sciences, Udine, Italy

Consorzio Roma Ricerche Roma, Italy
Ente per le Nuove tecnologie, l'Energia e l'Ambiente, Roma, Italy