A General Sparsified Nested Dissection Algorithm with a Task-Based Runtime System

Léopold Cambier*

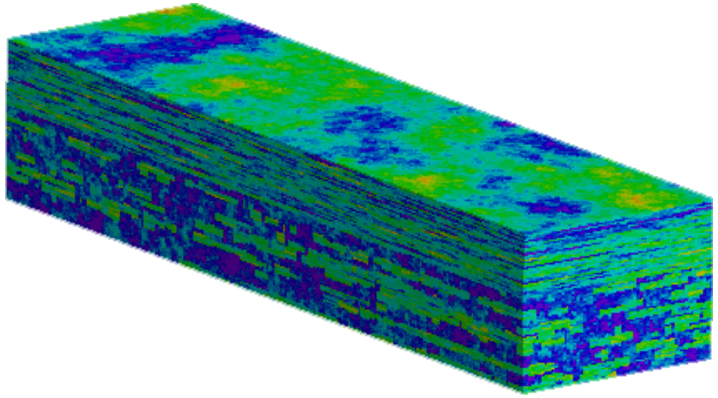
with Y. Qian*, B. Klockiewicz*, E. Darve*,
C. Chen†, E. Boman‡, S. Rajamanickam‡, R. Tuminaro‡

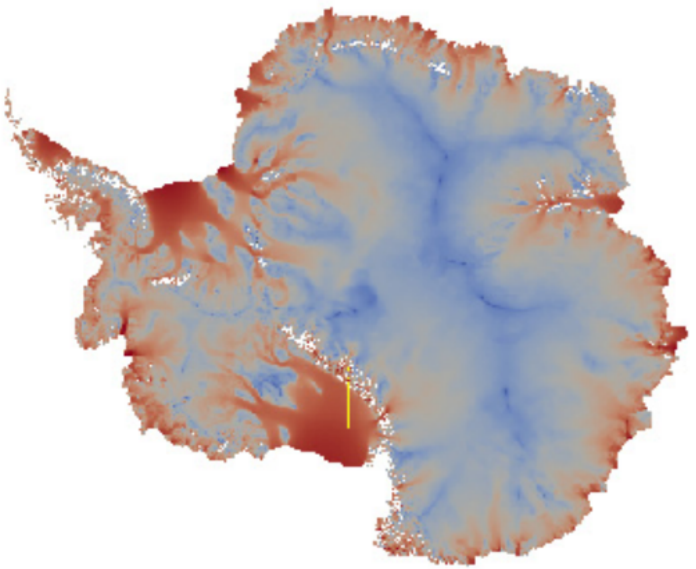
* Stanford, † UT Austin, ‡ Sandia

February 2020

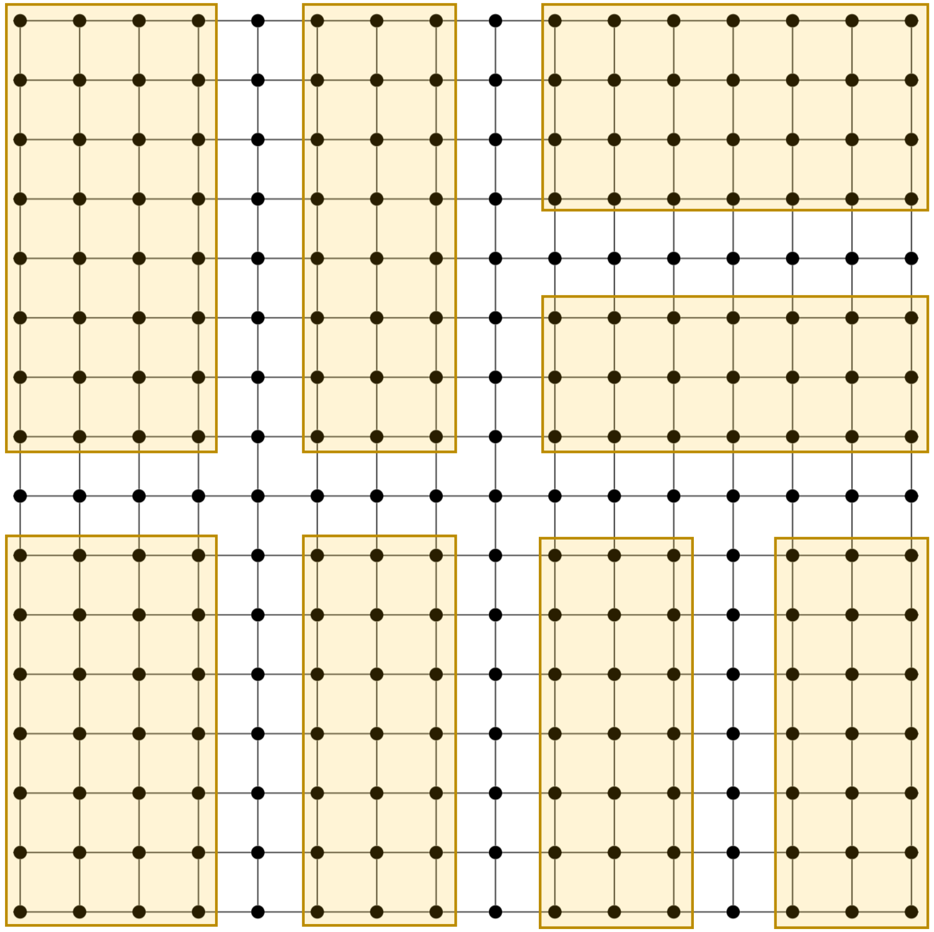
Problem and Motivation

- Ax = b, A is sparse and (typically) from PDE's
- Generic and algebraic (ε only) and scalable (multilevels, O(N log N))
- Parallel, using a task-based runtime system

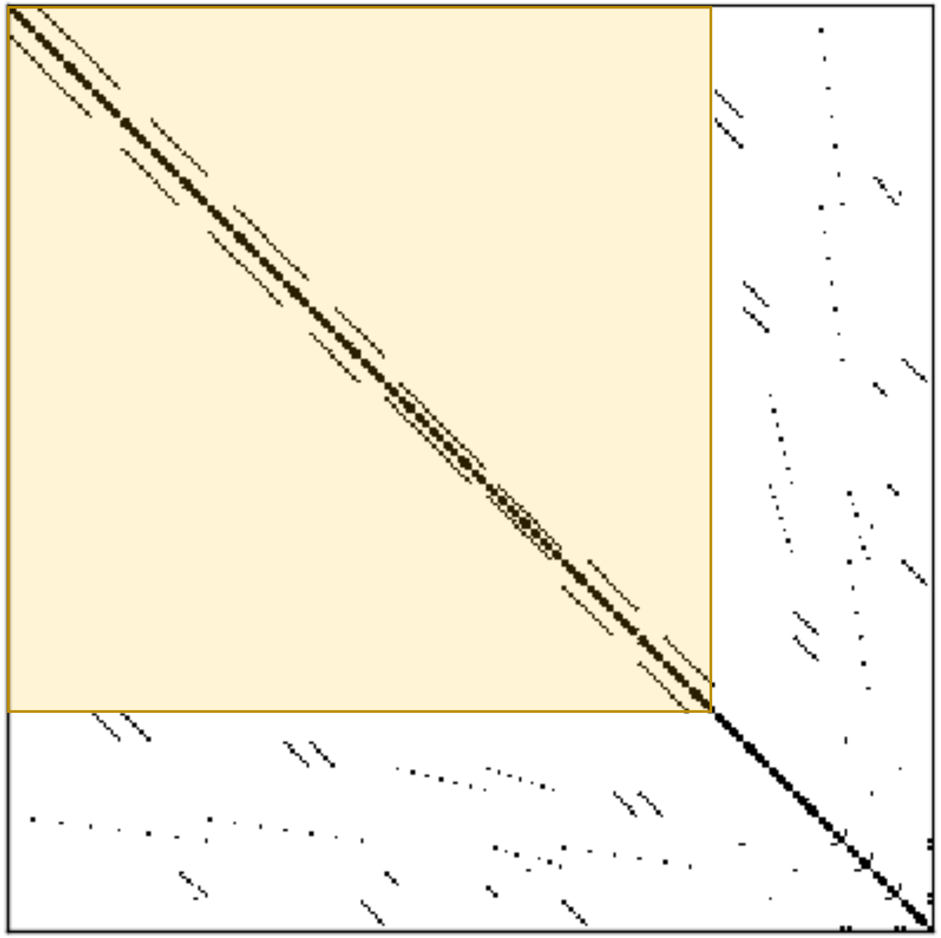




Sparse Linear Systems with Nested Dissection

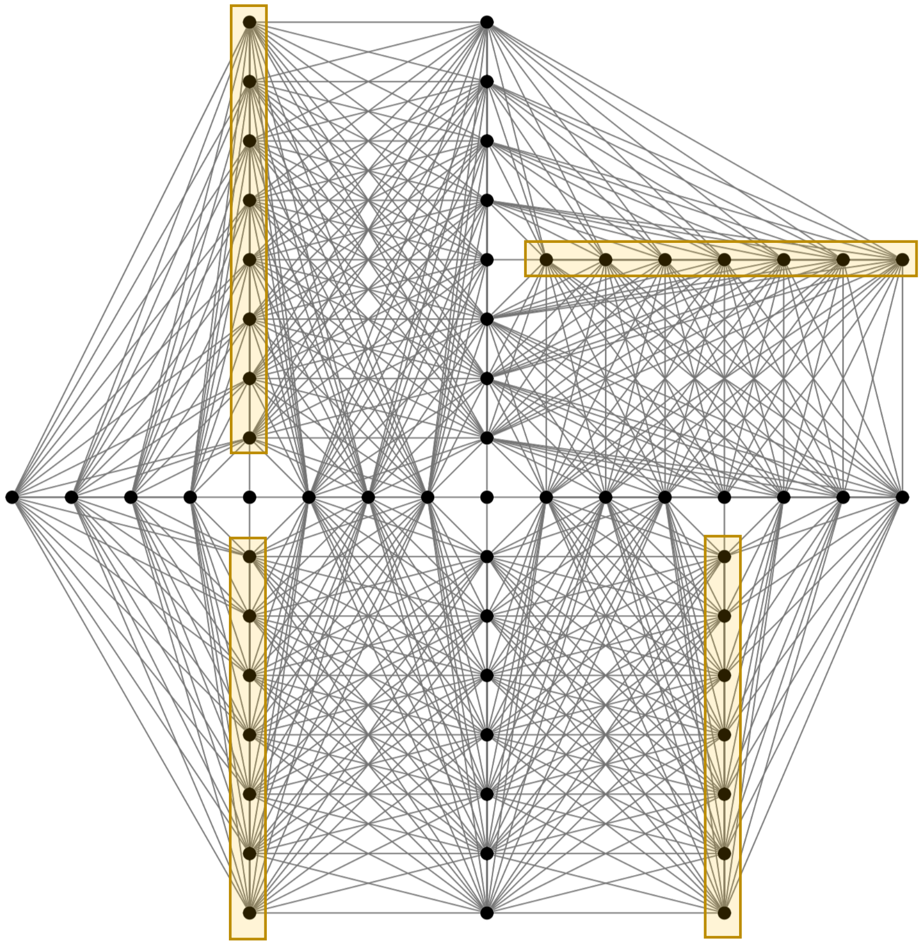


Matrix graph, nodes = unknowns

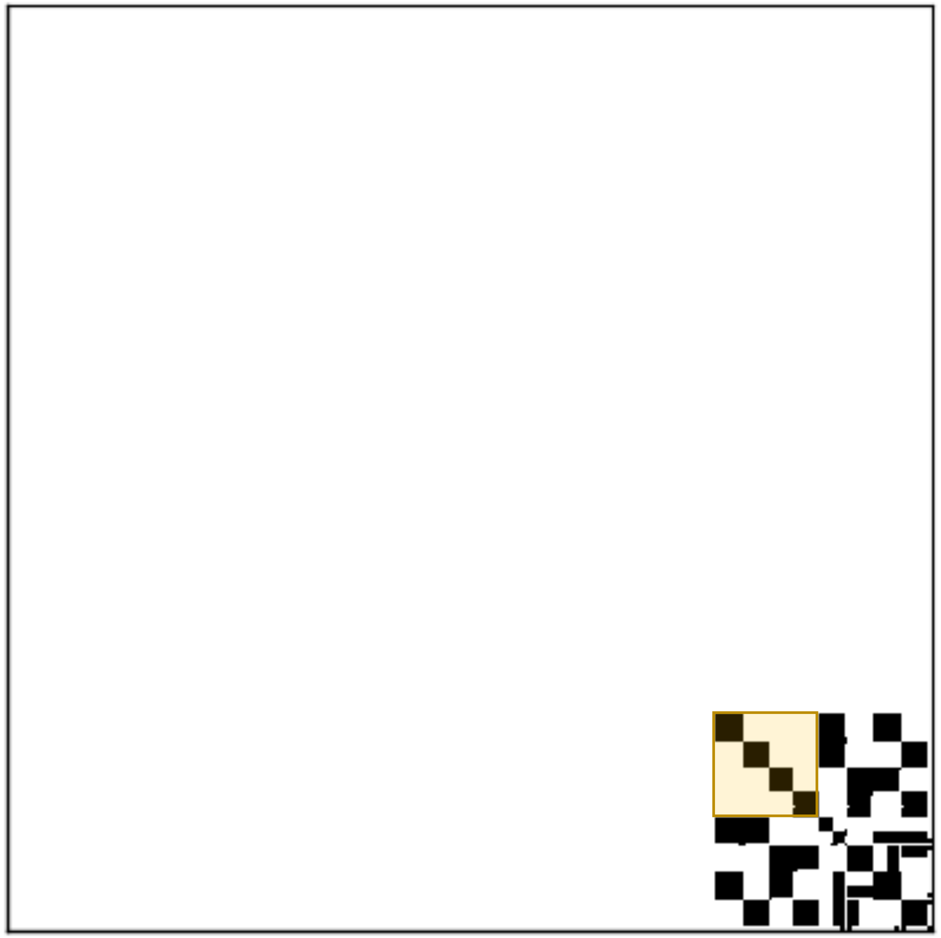


Matrix

Sparse Linear Systems with Nested Dissection

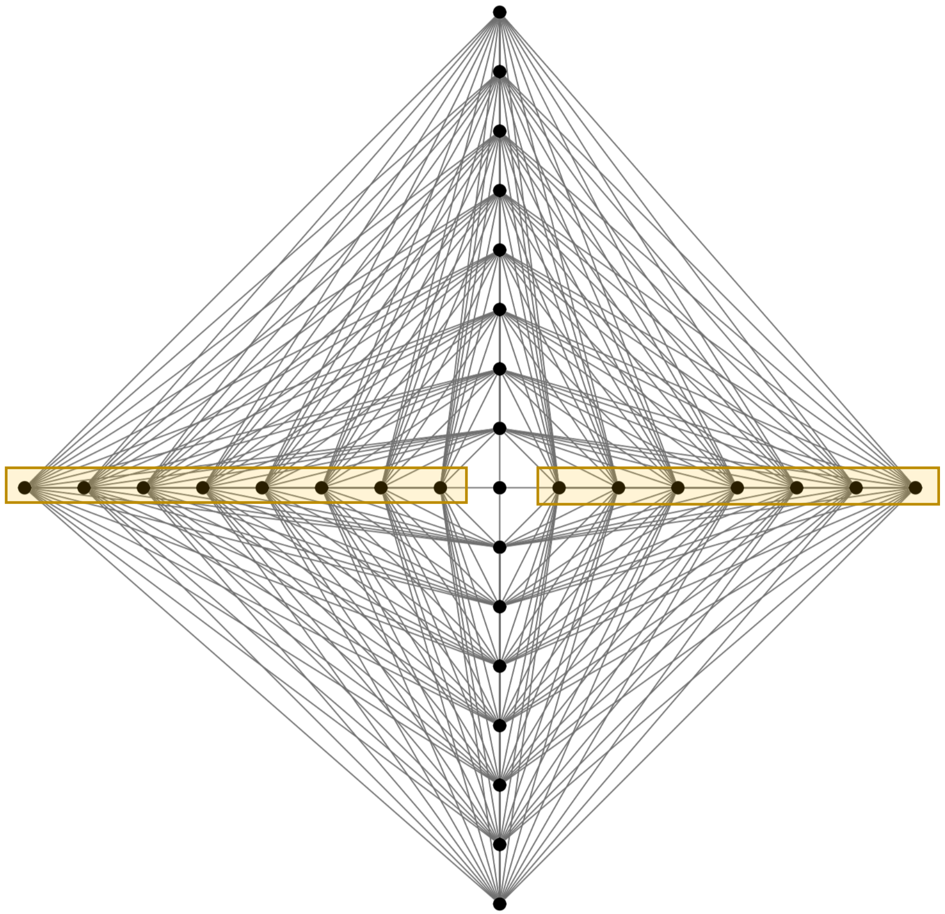


Matrix graph, nodes = unknowns

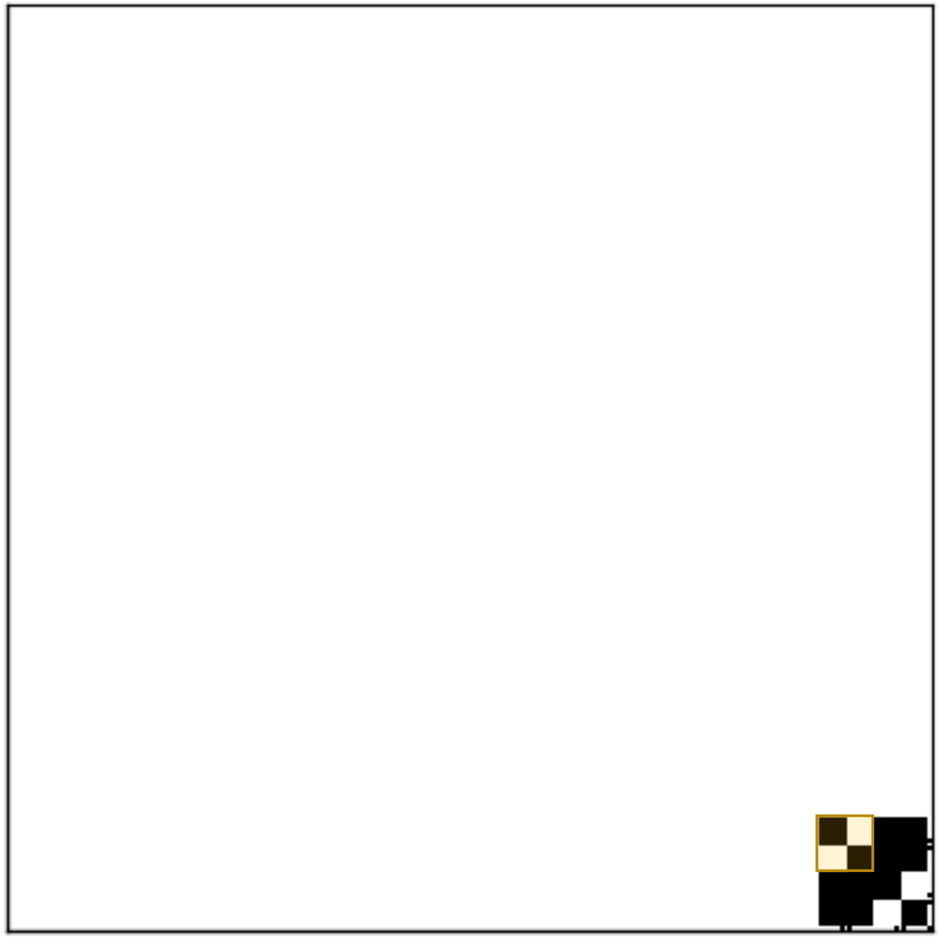


Matrix

Sparse Linear Systems with Nested Dissection



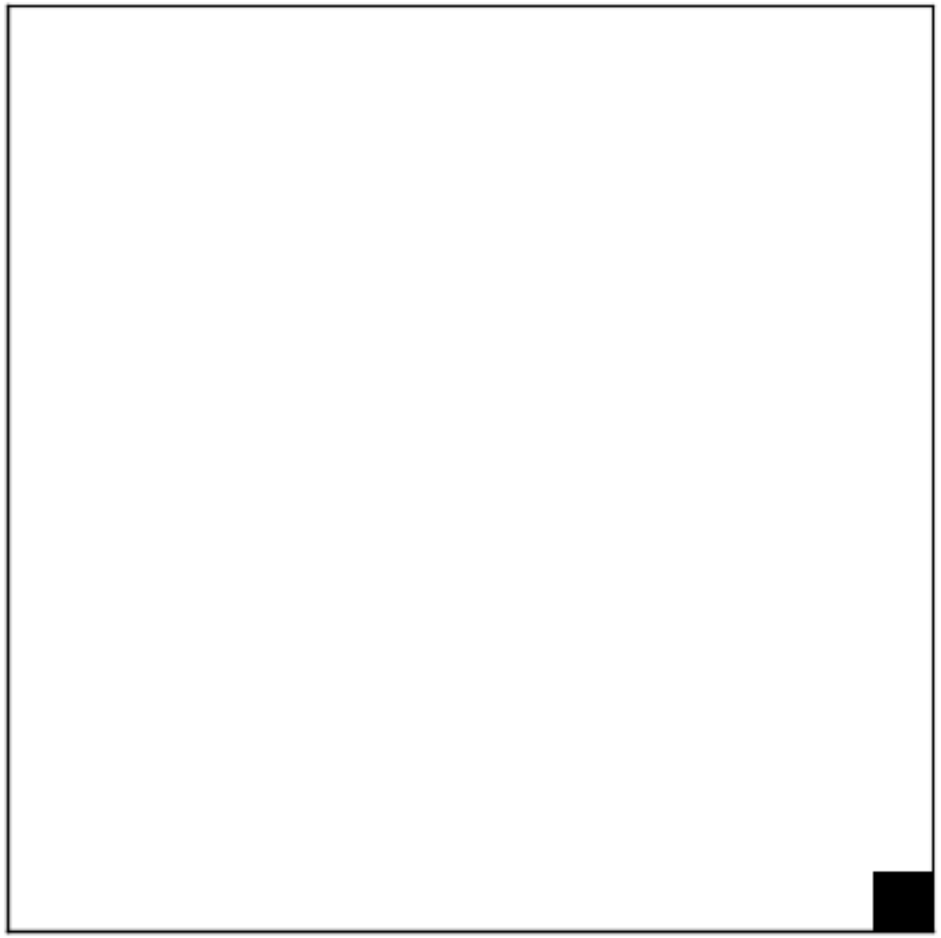
Matrix graph, nodes = unknowns



Matrix

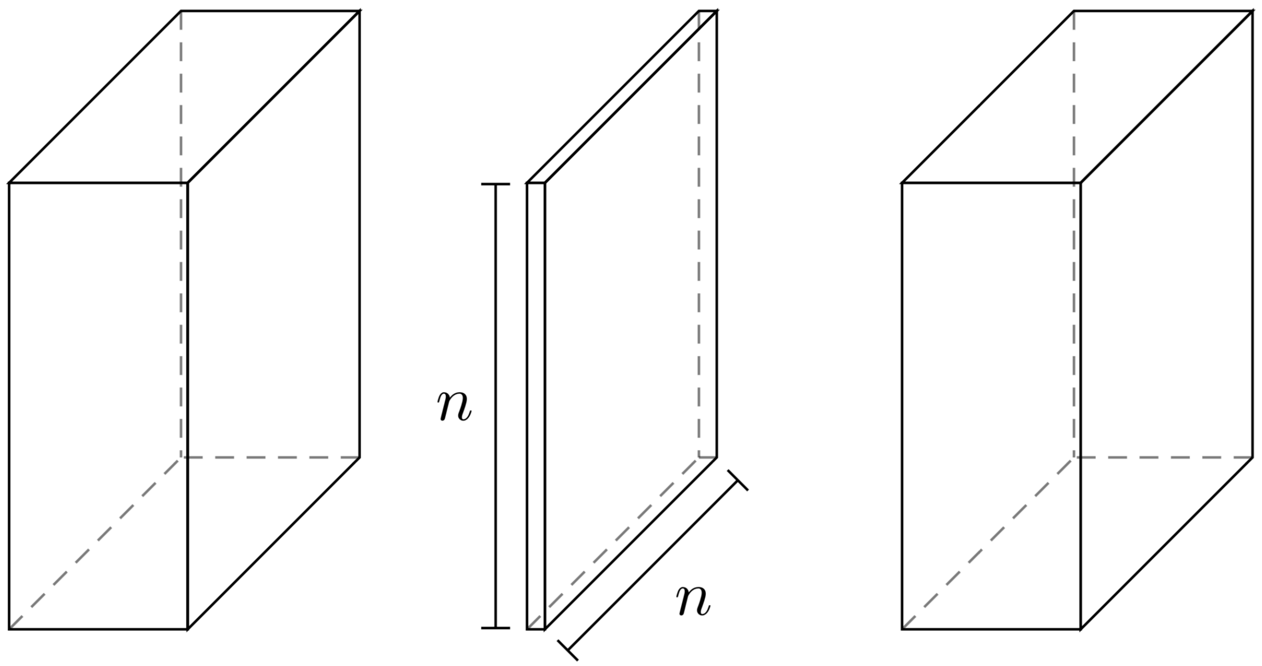
Sparse Linear Systems with Nested Dissection

Matrix graph, nodes = unknowns



Matrix

Nested Dissection is $O(N^2)$ in 3D

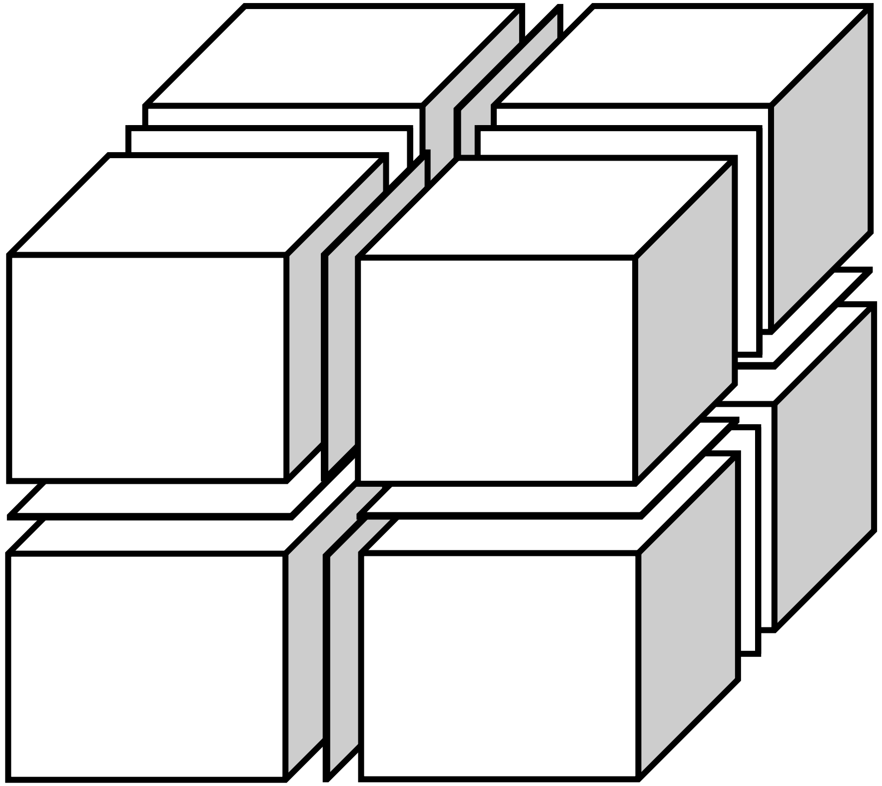


Too much fill-in!

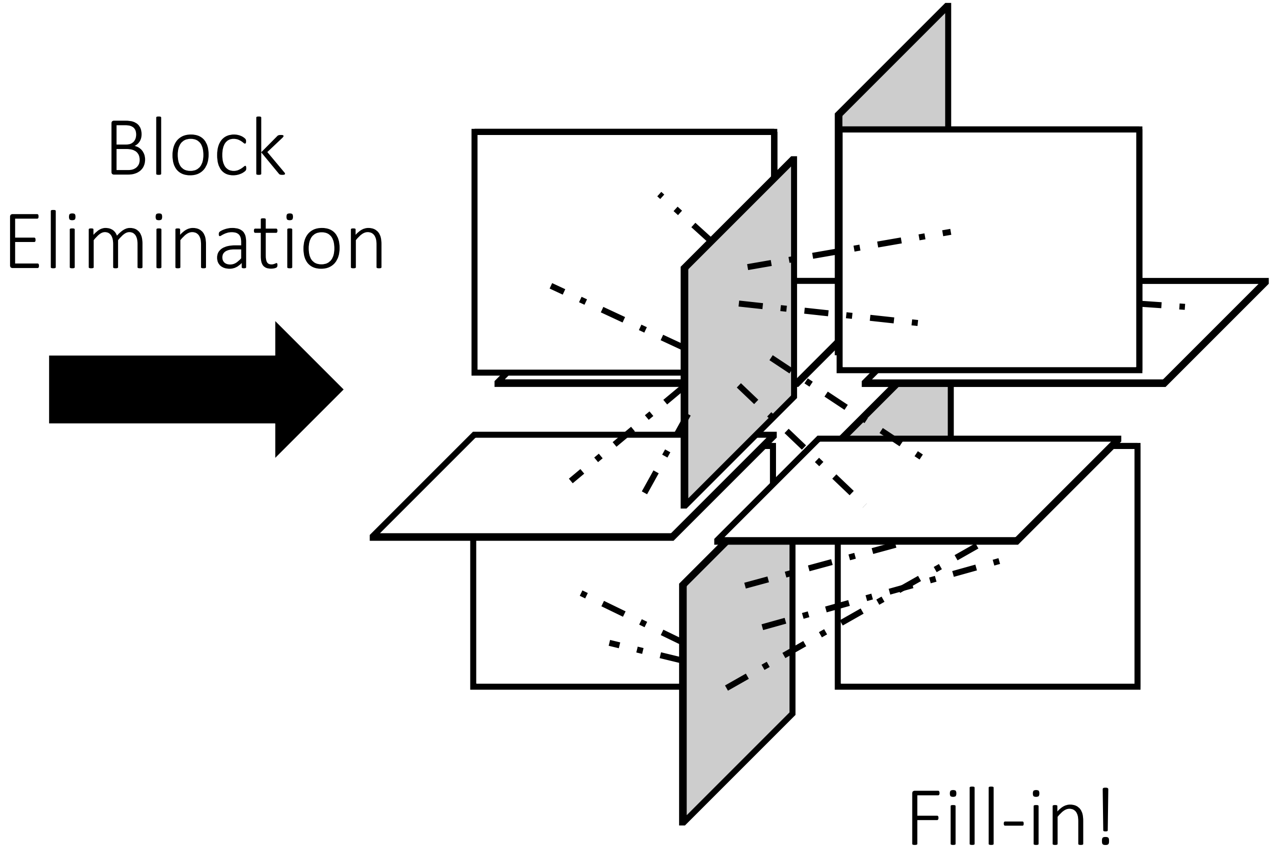
Factoring $n^2 = N^{\frac{2}{3}}$ takes $O(N^2)$

Nested Dissection Ellimination

$$L^{-1}\begin{bmatrix} A_{pp} & A_{pn} & \\ A_{np} & A_{nn} & A_{nw} \\ & A_{wn} & A_{ww} \end{bmatrix} U^{-1} = \begin{bmatrix} I & & \\ & A_{nn} - A_{ns}A_{ss}^{-1}A_{sn} & A_{nw} \\ & A_{wn} & A_{ww} \end{bmatrix}$$

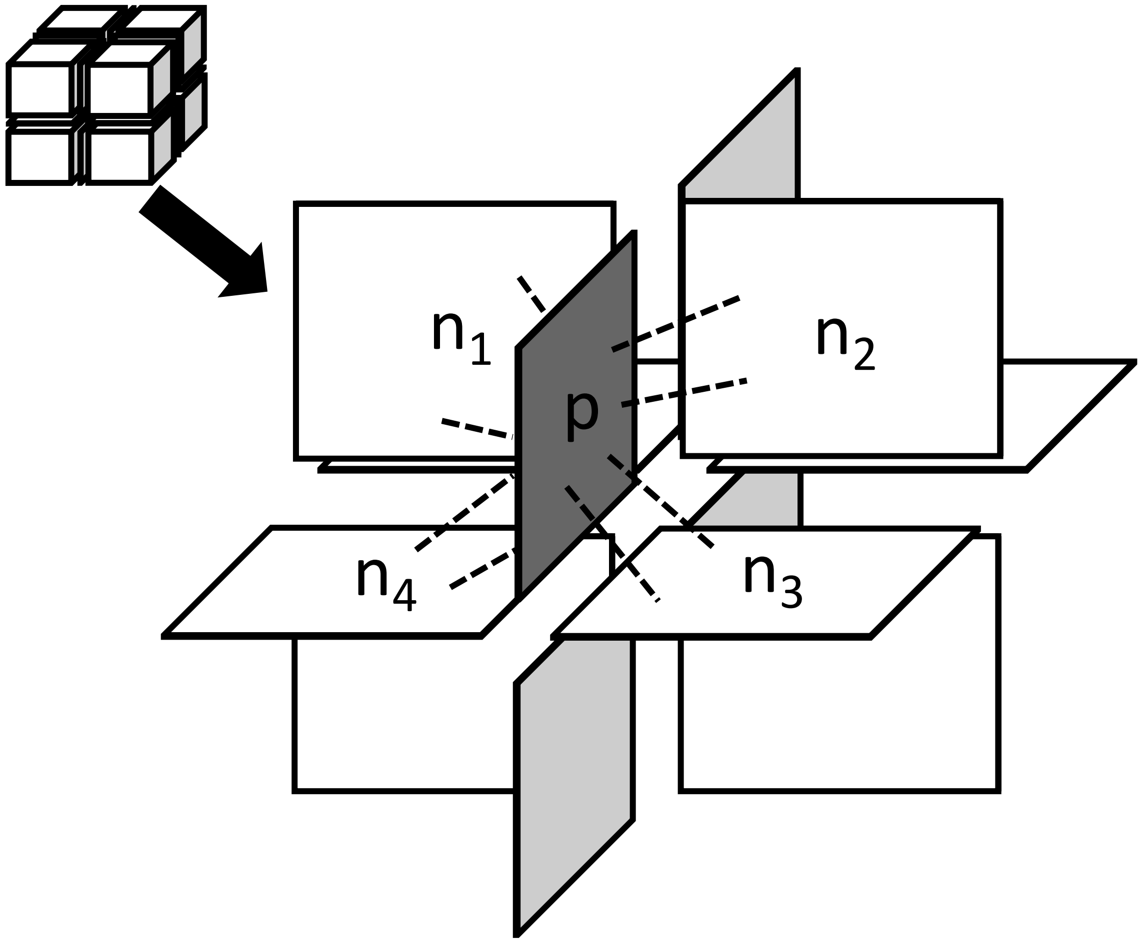


Matrix graph, sets = clusters of vertices

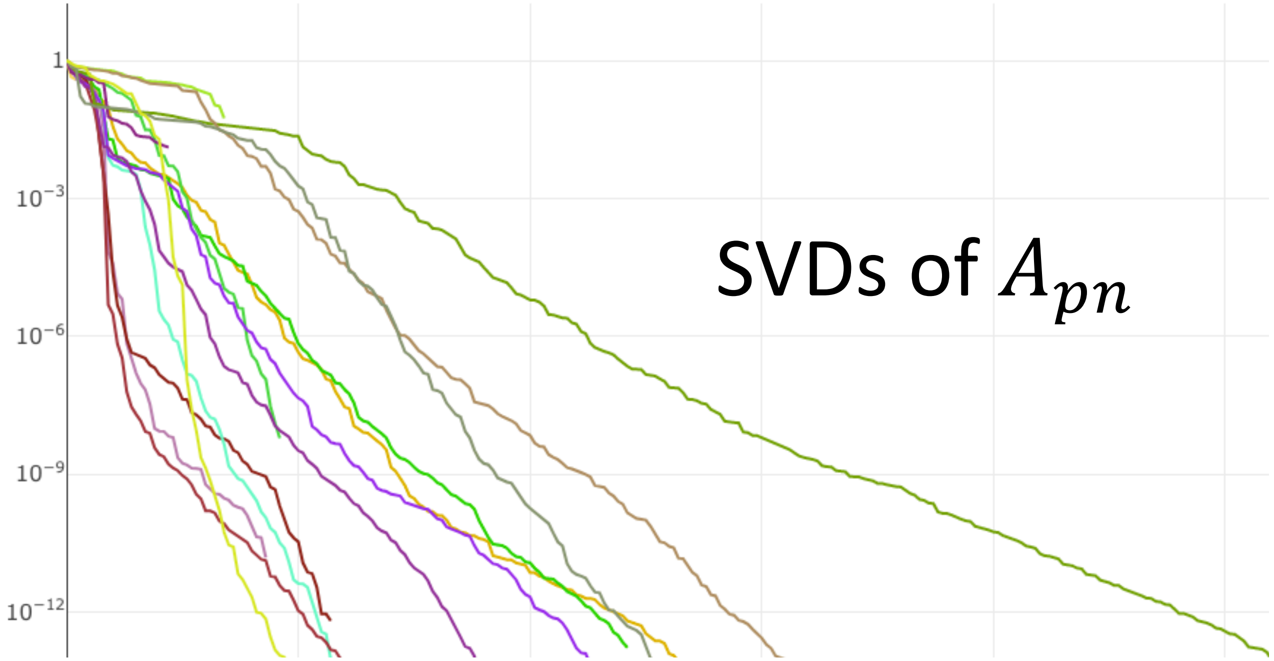


Sparsification

$$A_{pn} = [A_{pn_1} \quad A_{pn_2} \quad A_{pn_3} \quad \ldots]$$
$$= Q_c W_{cn} + Q_f W_{fn}$$
$$\approx Q_c W_{cn}$$



Matrix graph, sets = clusters of vertices



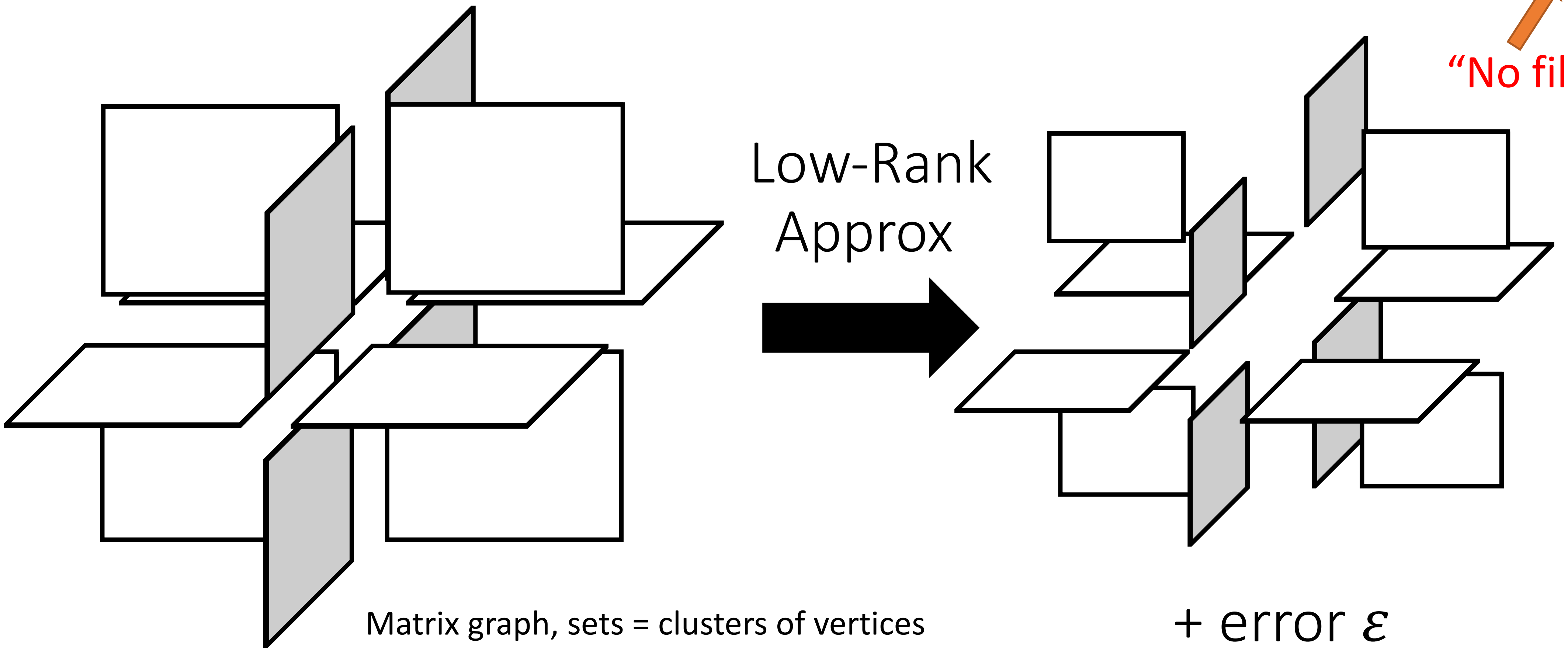
SVDs of A_{pn}

Sparsification

"Eliminated"

$$\begin{bmatrix} Q_p^T & \\ & I \end{bmatrix} \begin{bmatrix} I & A_{pn} \\ A_{np} & A_{nn} \end{bmatrix} \begin{bmatrix} Q_p & \\ & I \end{bmatrix} = \begin{bmatrix} I & & \times \\ & I & W_{cn} \\ \times & W_{nc} & A_{nn} \end{bmatrix}$$

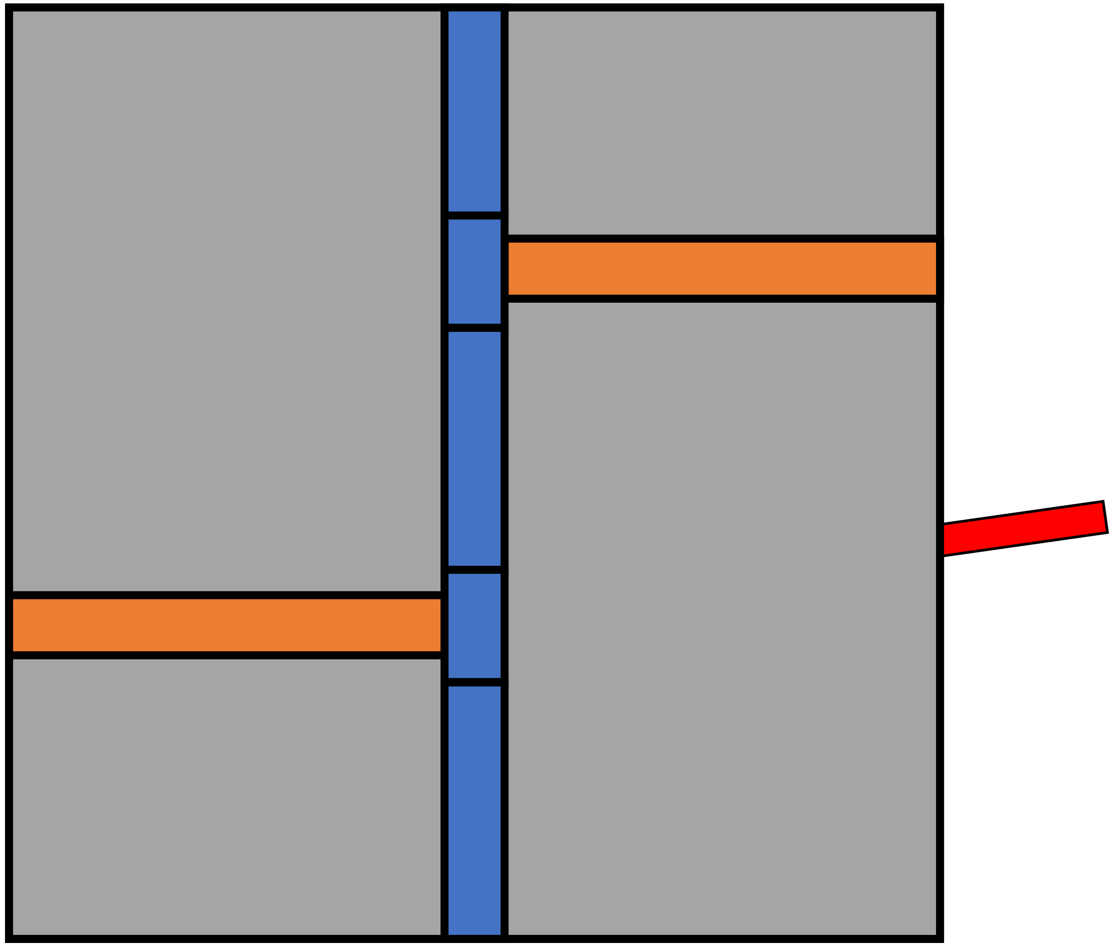
"No fill-in!"



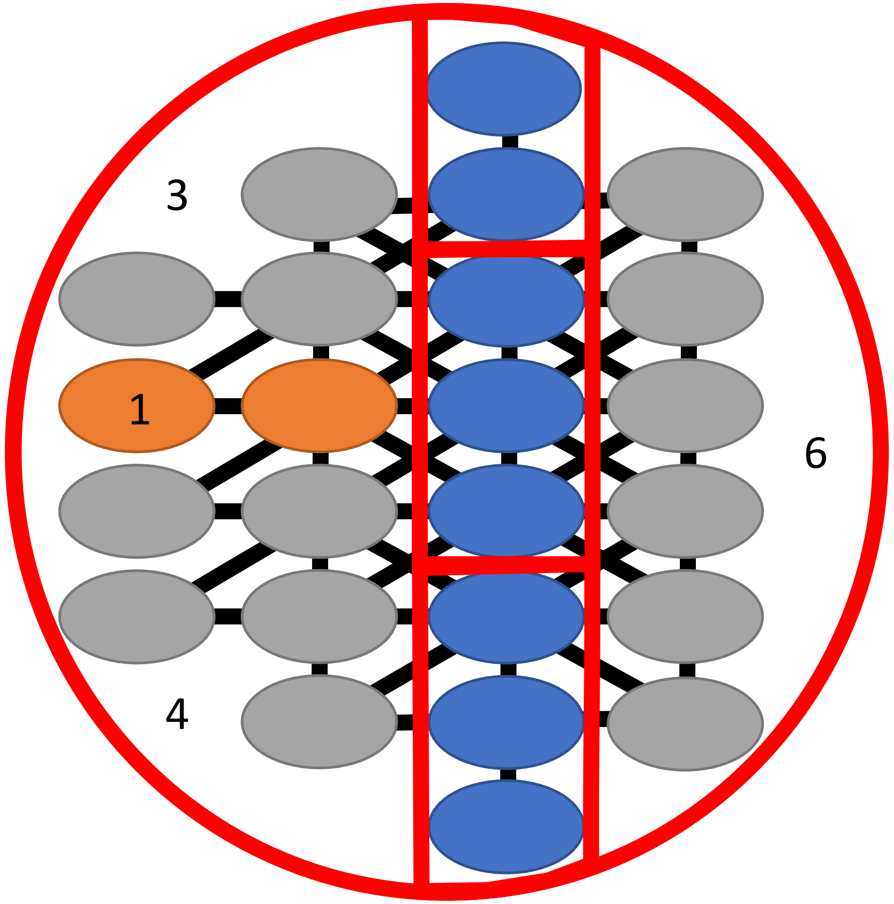
Matrix graph, sets = clusters of vertices

HIF: Ho, Kenneth L., and Lexing Ying. "Hierarchical interpolative factorization for elliptic operators: differential equations." *Communications on Pure and Applied Mathematics* 69.8 (2016): 1415-1451.

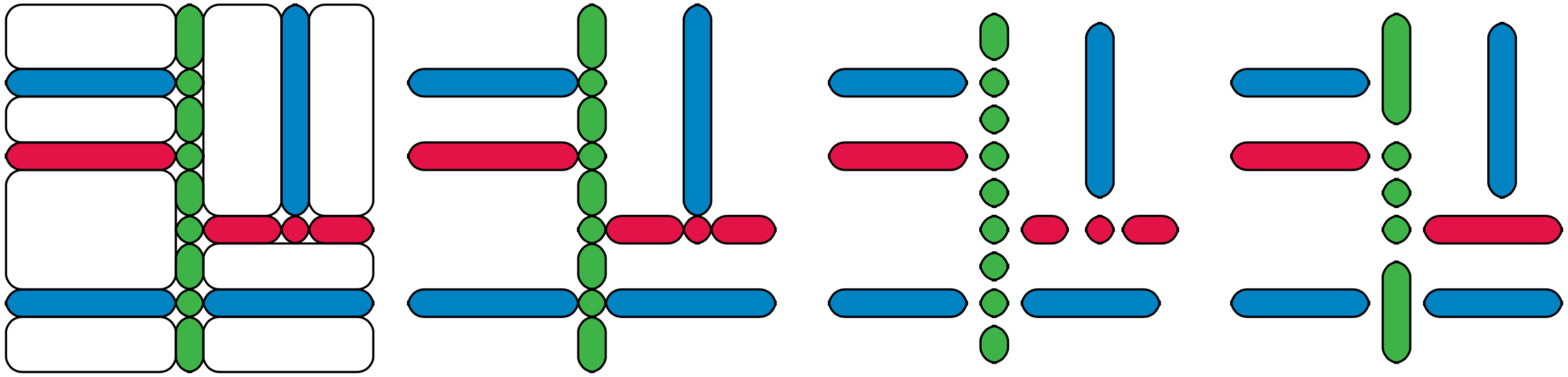
Building interfaces



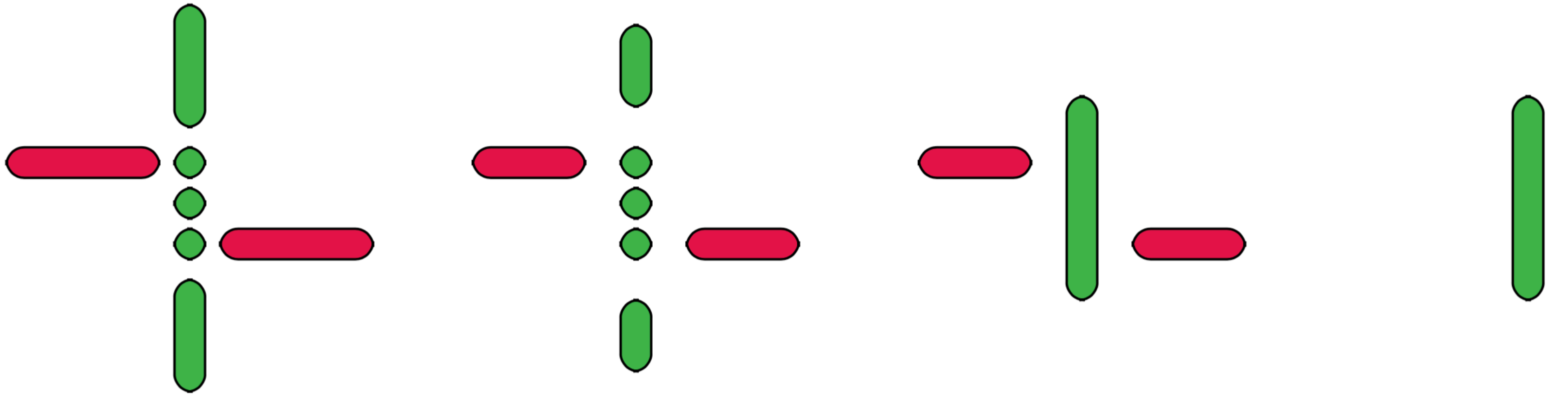
Matrix graph, sets = clusters of vertices



Eliminate ↦ Scale ↦ Sparsify ↺



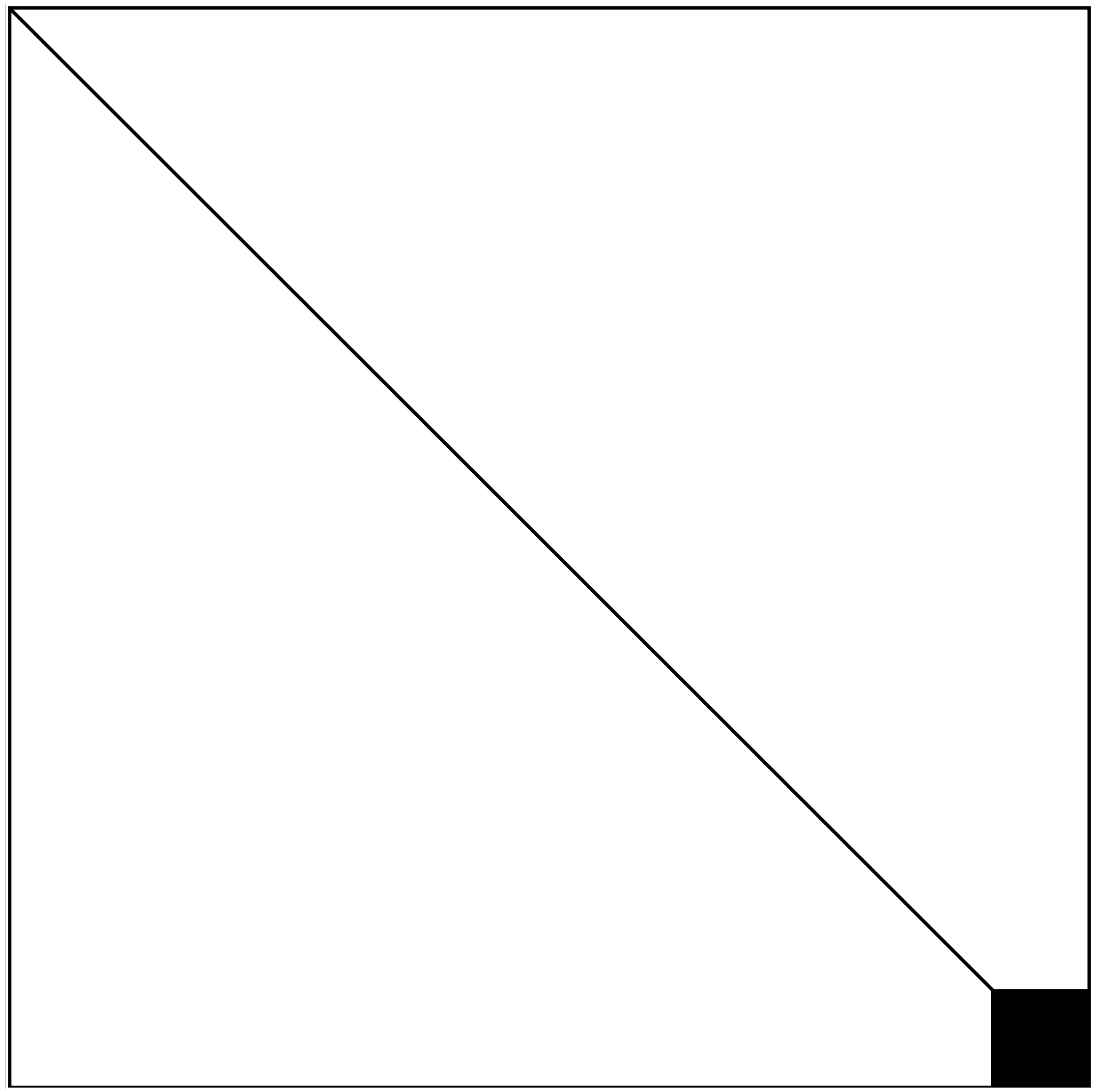
(g) A, original graph (h) After $E_1^\top$ (i) After $S_1^\top Q_1$ (j) After merge



(k) After $E_2^\top$ (l) After $S_2^\top Q_2$ (m) After merge (n) After $E_3^\top$

Matrix graph,
sets = clusters
of vertices

Eliminate ↦ Scale ↦ Sparsify ↺



General spaND

Block scaling matters!

For level k = 1, …, L

- Eliminate interiors (LL^T, LDL^T, PLU, PLUQ)

$$L^{-1}\begin{bmatrix} A_{pp} & A_{pn} & \\ A_{np} & A_{nn} & A_{nw} \\ & A_{wn} & A_{ww} \end{bmatrix} U^{-1} = \begin{bmatrix} I & & \\ & A_{nn} - A_{ns}A_{ss}^{-1}A_{sn} & A_{nw} \\ & A_{wn} & A_{ww} \end{bmatrix}$$

Fill-in;
limited by separators

- Scale interfaces (LL^T, LDL^T, PLU, PLUQ)

$$\begin{bmatrix} L^{-1} & \\ & I \end{bmatrix}\begin{bmatrix} A_{pp} & A_{pn} \\ A_{np} & A_{nn} \end{bmatrix}\begin{bmatrix} U^{-1} & \\ & I \end{bmatrix} = \begin{bmatrix} I & L^{-1}A_{pn} \\ A_{np}U^{-1} & A_{nn} \end{bmatrix}$$

- Sparsify interfaces (RRQR)

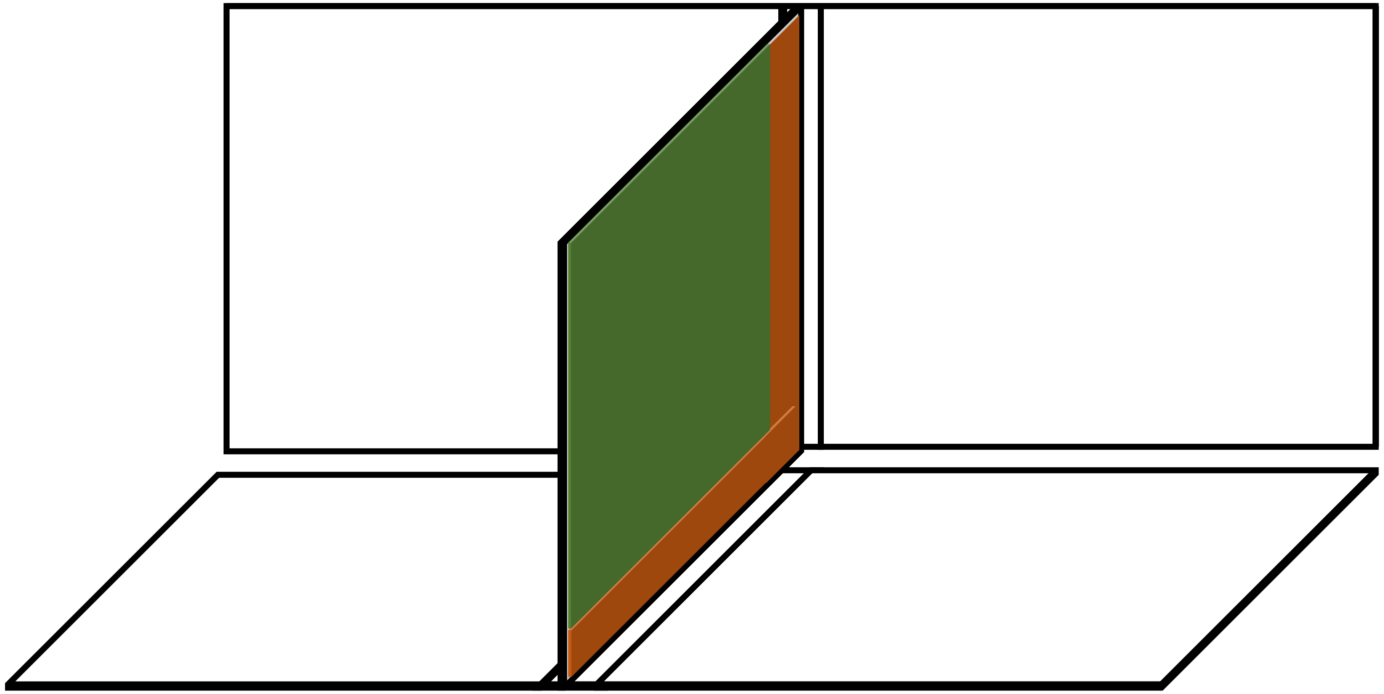
$$\begin{bmatrix} Q_p^T & \\ & I \end{bmatrix}\begin{bmatrix} I & A_{pn} \\ A_{np} & A_{nn} \end{bmatrix}\begin{bmatrix} Q_p & \\ & I \end{bmatrix} = \begin{bmatrix} I & & \varepsilon \\ & I & W_{cn} \\ \varepsilon & W_{nc} & A_{nn} \end{bmatrix} \approx \begin{bmatrix} I & & \\ & I & W_{cn} \\ & W_{nc} & A_{nn} \end{bmatrix}$$

No fill-in!

- Merge clusters

Separator sizes ?

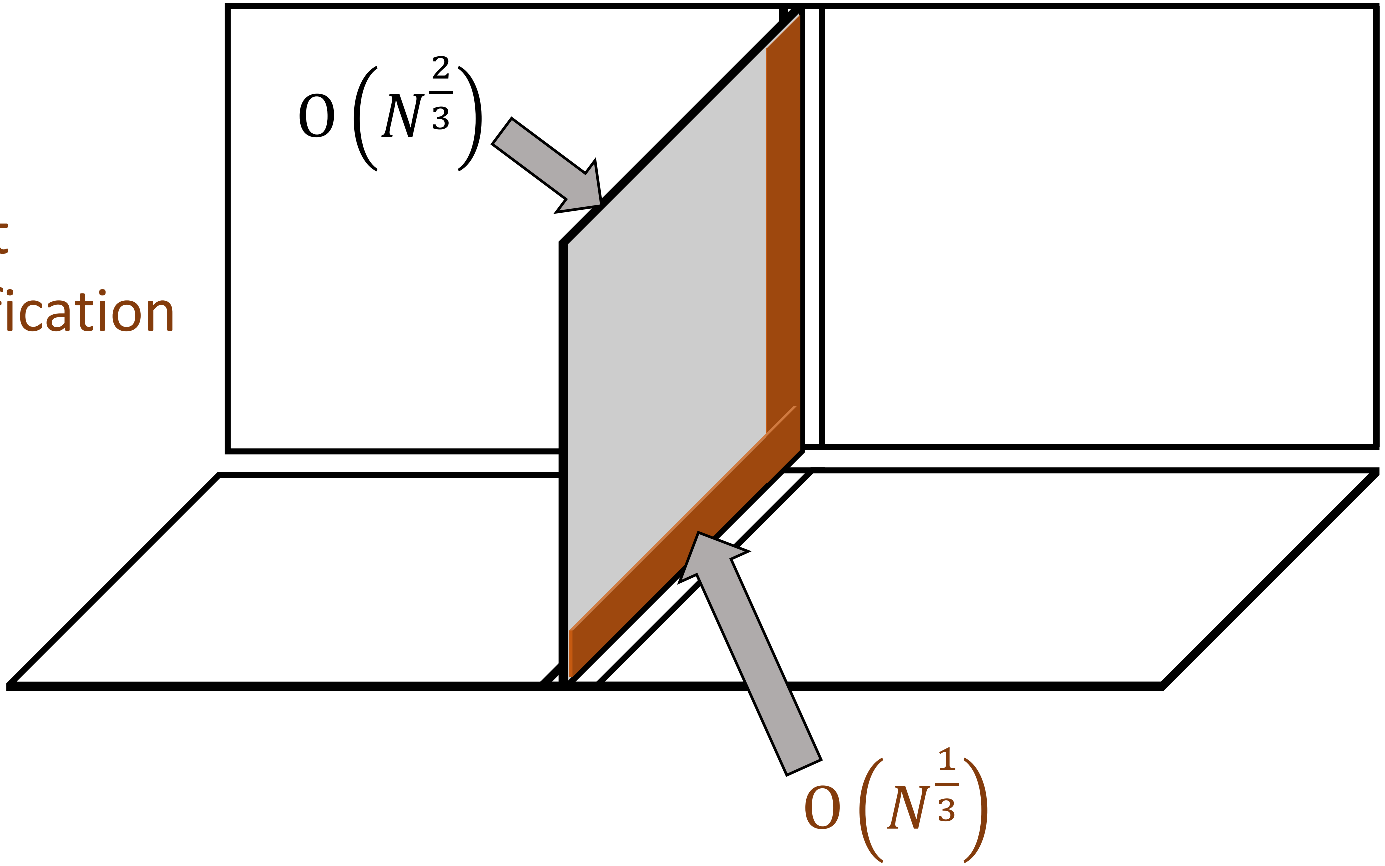
Brown = original connections
to other nodes = do not sparsify



Green = fill-in's to other nodes = sparsifies well

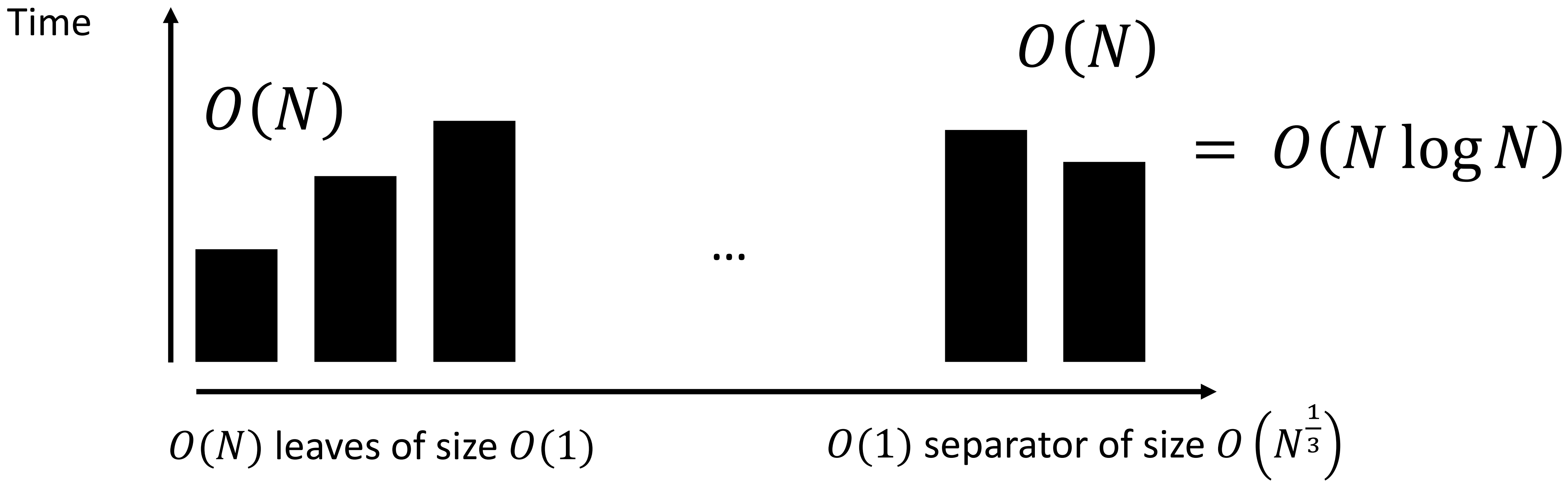
Separator sizes: $O\left(N^{\frac{2}{3}}\right) \Rightarrow O\left(N^{\frac{1}{3}}\right)$

Brown = left
after sparsification



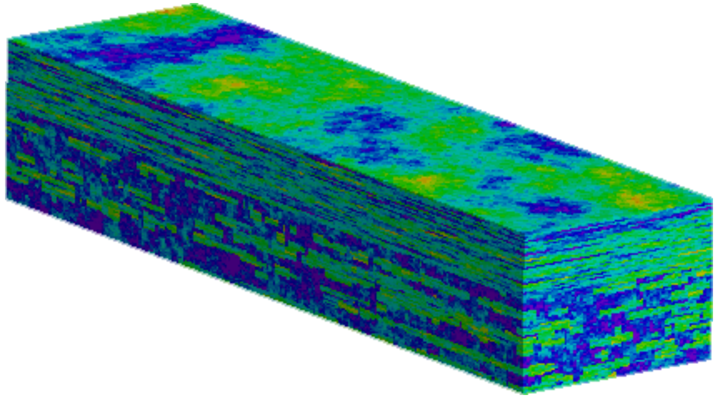
spaND is $O(N\log\mathrm{N})$ in 3D

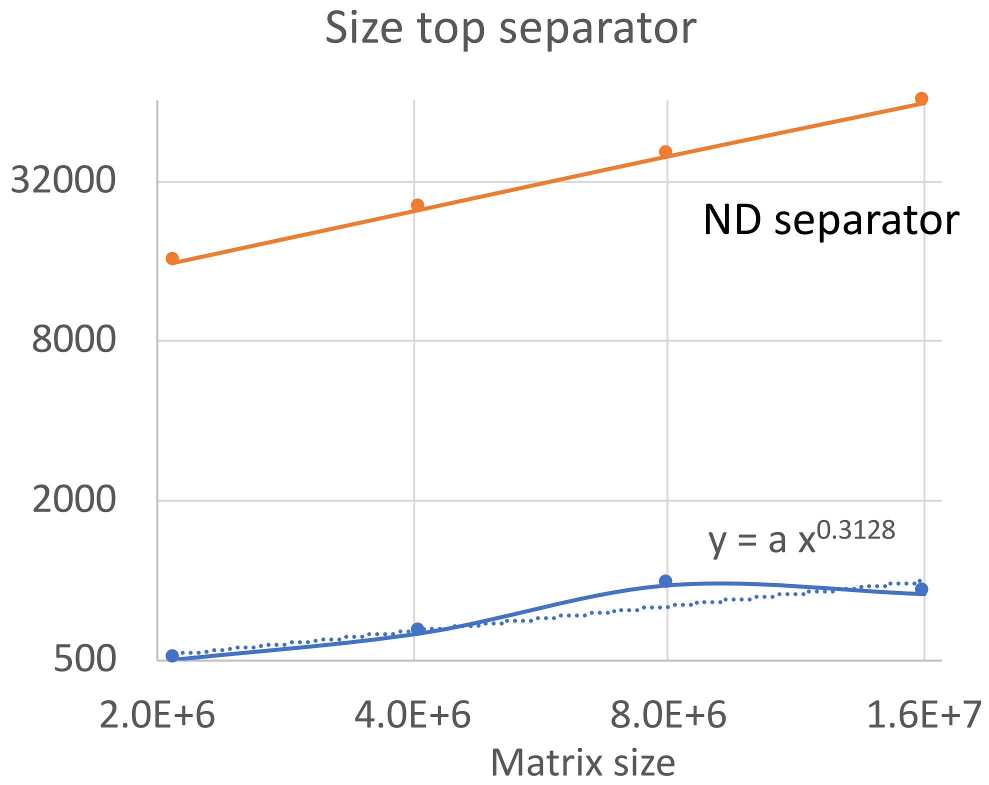
<u>If separators</u> $N^{\frac{2}{3}} \rightarrow N^{\frac{1}{3}}$

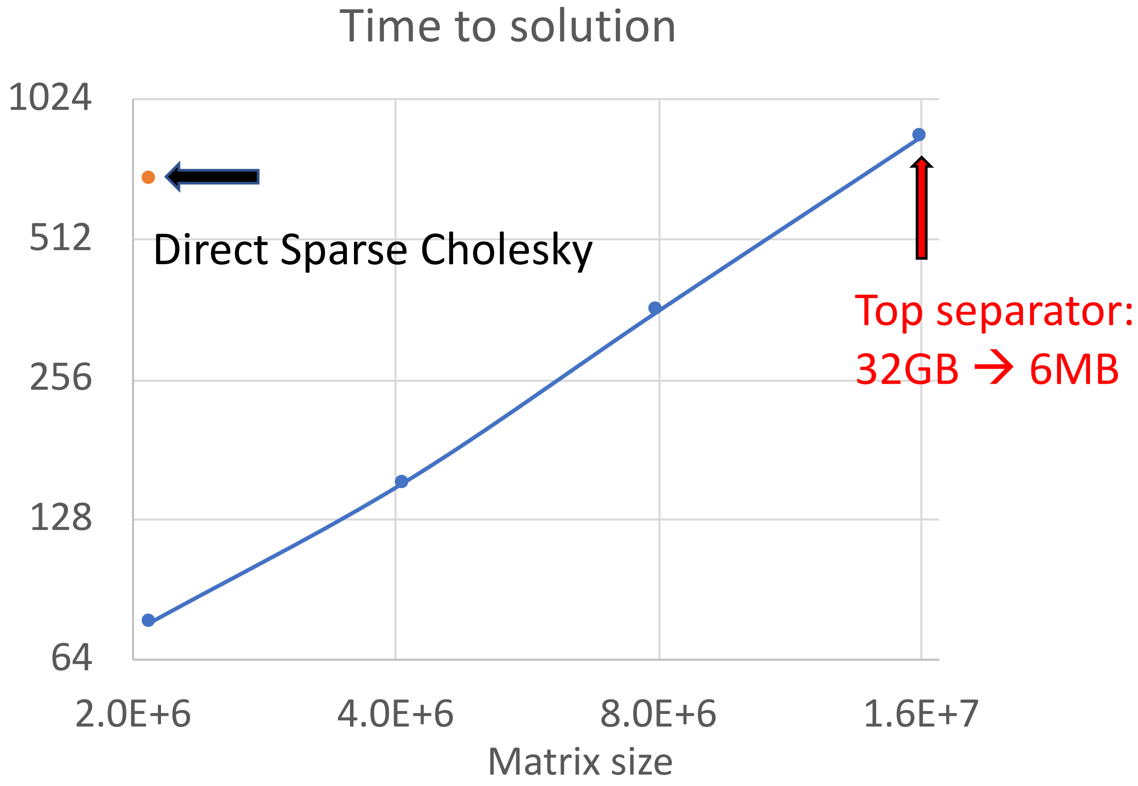


Separators $\approx O\left(N^{\frac{1}{3}}\right)$ on 3D-like problems

(SPE10, 3D $\nabla \cdot (a(x)\nabla u)$ Poisson-like, SPD)

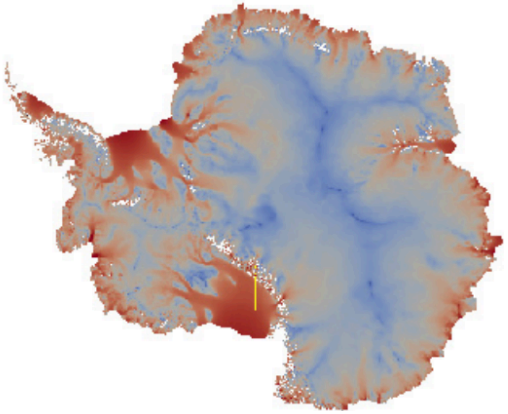




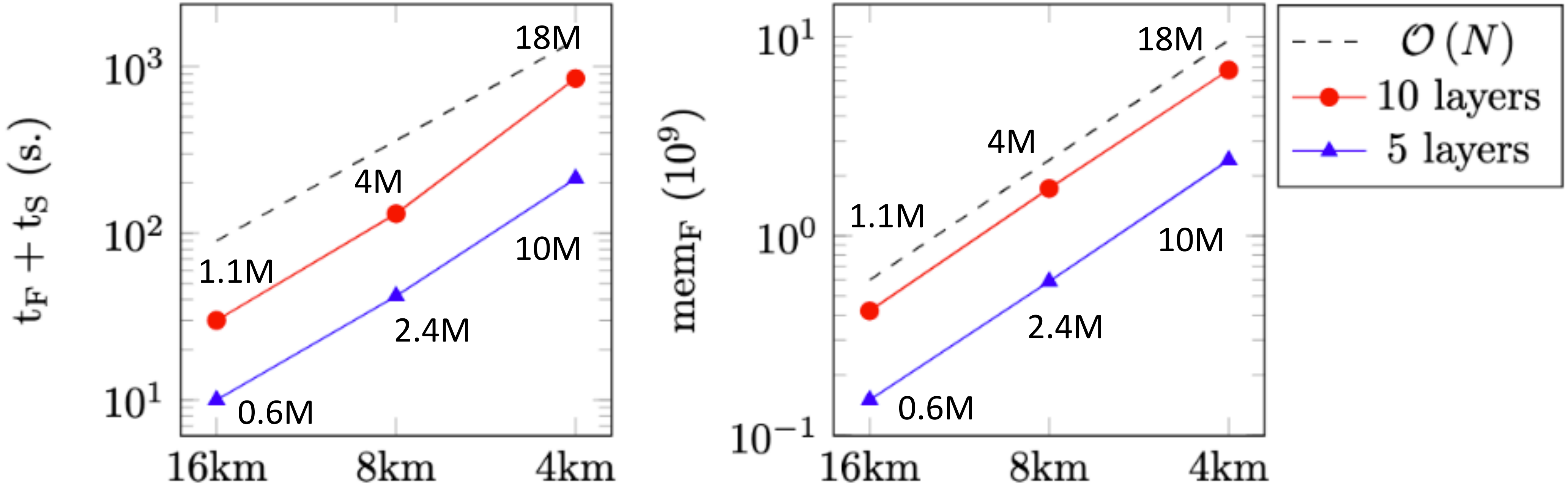


M. Christie, M. Blunt, et al., Tenth SPE comparative solution project: A comparison of upscaling techniques, in SPE reservoir simulation symposium, Society of Petroleum Engineers, 2001.

Separators ≈ $O(1)$ on 2D-like problems



2D FEM, SPD, Very ill-conditioned (k(A) > 10^{11})



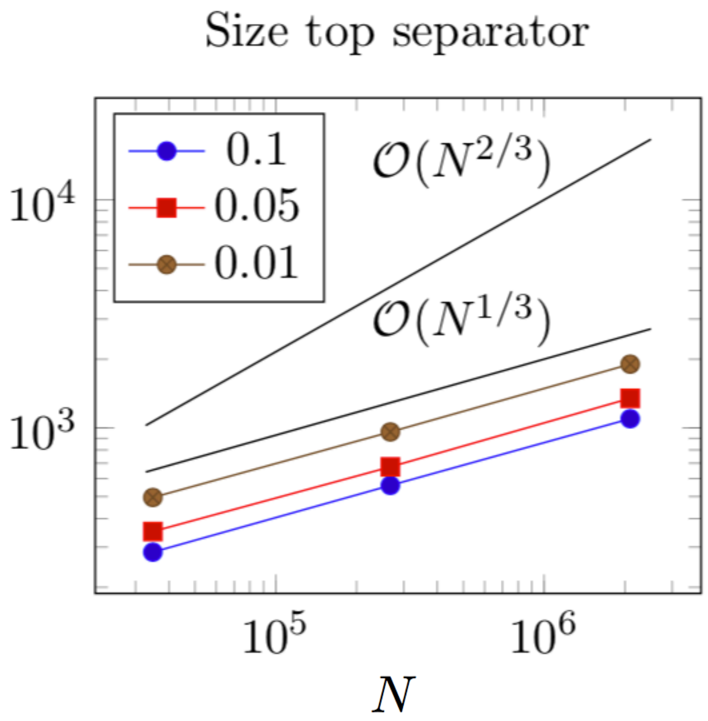
Separators drop to 78 (smallest) to 159 (largest) → O(1) on 2-D like problems

K. Tezaur, M. Perego, A. G. Salinger, R. S. Tuminaro, and S. F. Price, ALBANY/FELIX: a parallel, scalable and robust, finite element, first-order Stokes approximation ice sheet solver built for advanced analysis, Geoscientific Model Development (Online),8 (2015).

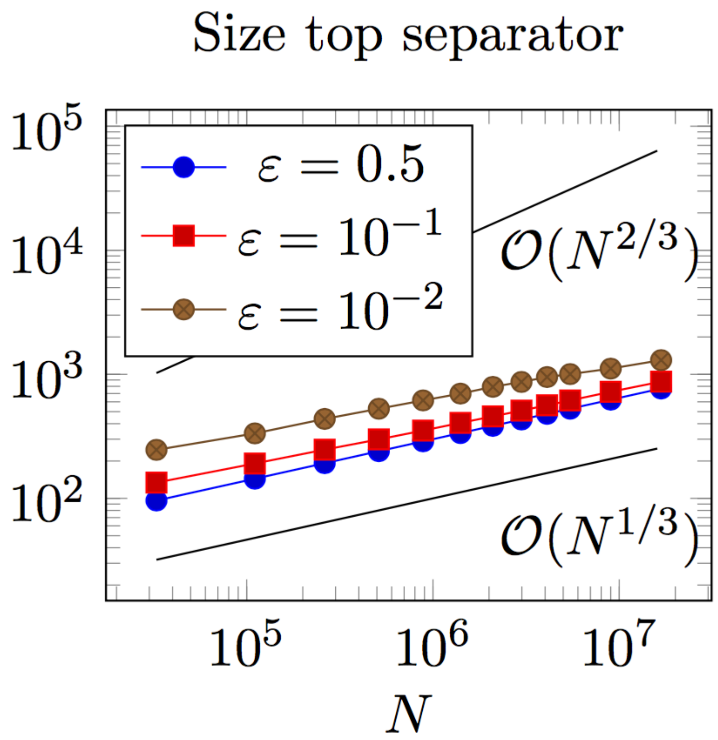
Holds for non-elliptic PDE's as well

Biot problem, 3D FEM
coupled pressure/displacement PDE

$$A = \begin{bmatrix} K & B \\ B^\top & -C \end{bmatrix}, K \succ 0, C \succ 0$$

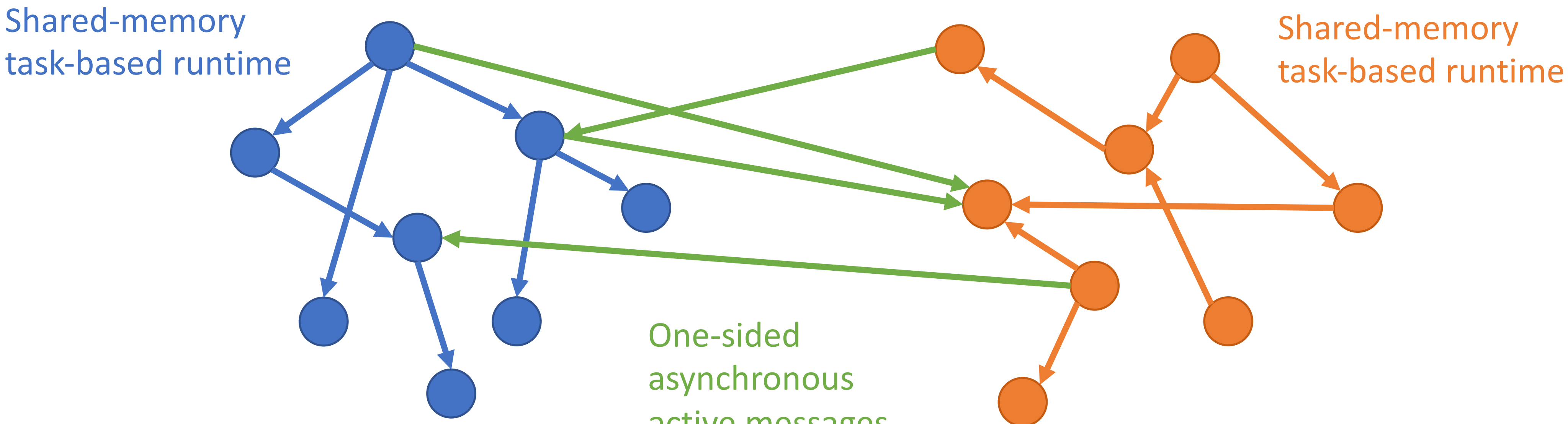


Advection-diffusion
3D FD $a\Delta u + b\nabla u = f$,
Dirichlet, $a = 10^{-2}, b = 1$



TaskTorrent

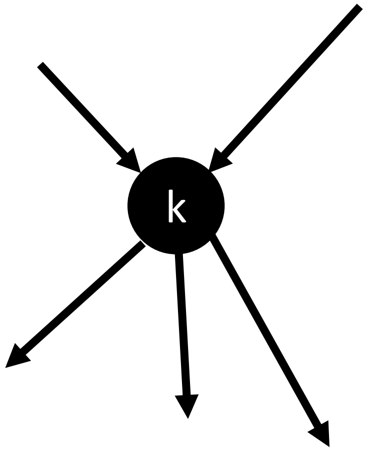
https://github.com/leopoldcambier/tasktorrent



TaskTorrent

Parametrized task graph

- Tasks == functions(k)
  1. Number of incoming dependencies
  2. Computational routine
  3. Fulfill outgoing dependencies
- Tasks fulfill deps on other tasks
  - Locally
  - Remotely
    - Asynchronous one-sided communications
    - No waiting on receiver
    - Never blocking




```
tf = Taskflow<int>();


tf.set_n_deps_in([](int k) {
    return ndeps(k);
});


tf.set_task([](int k) {
    domath(k);
    tf.fulfill_promise(otherk);
    am->send(otherrank, data,
        anotherk)
});
```

```
comm = Communicator()

am = comm.make_active_msg([&](data d, int k) {
   copy(data, somewhere)
   tf.fulfill_promise(k);
});
```

TaskTorrent

Parametrized task graph

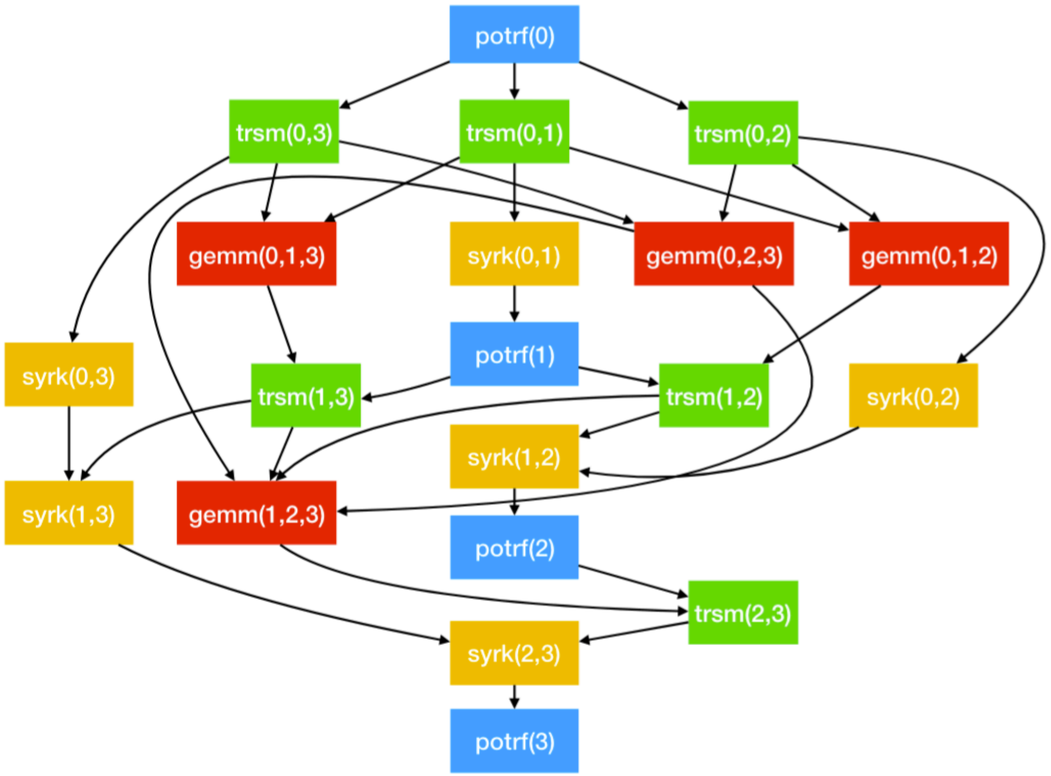
- Distributed tasks creation/exploration
- 100% asynchronous execution (no wait)
- Data and dependencies are separated
    - Automatic dependency tracking
    - Non-blocking data/fulfill “pushes”
- No sequential semantic (code looks different)

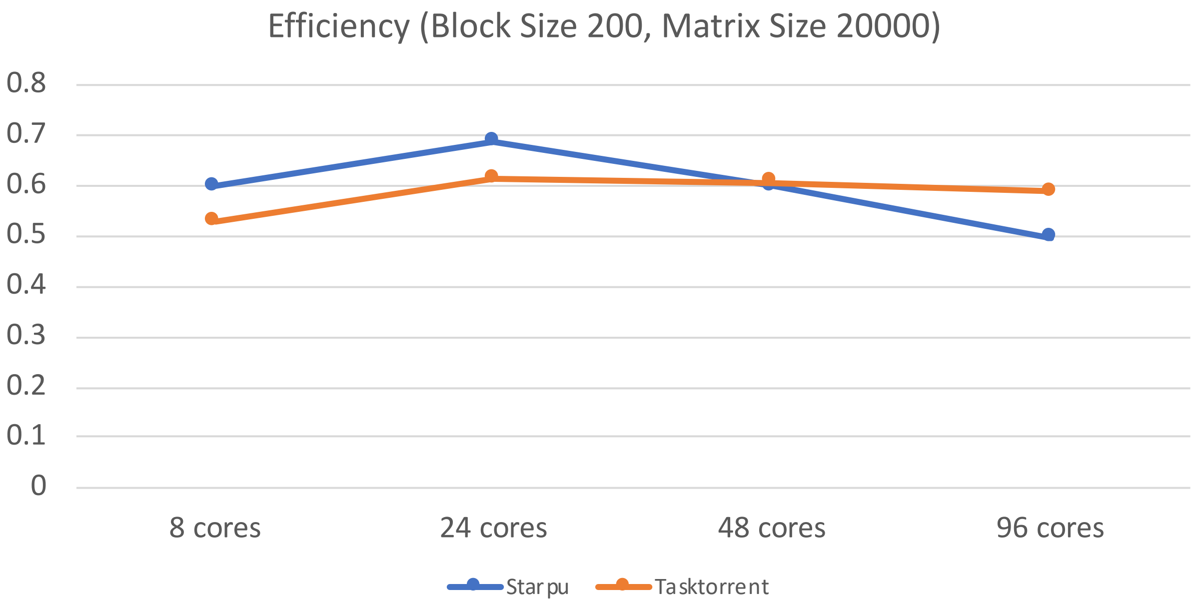
Comparison with other runtime

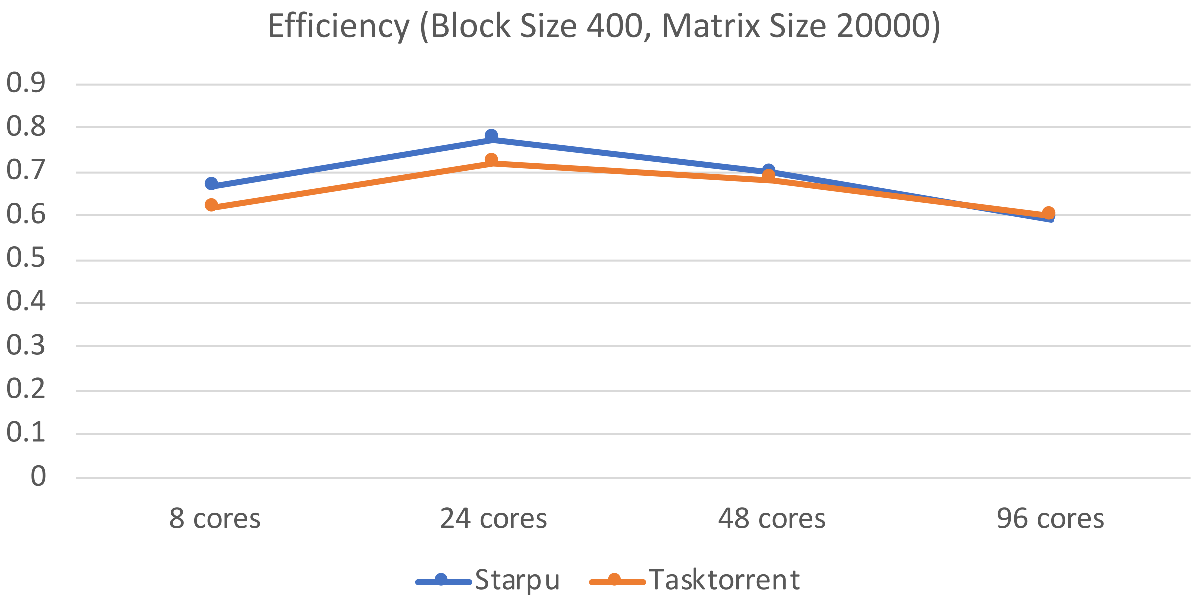
- StarPU:
  - Tasks are sequentially inserted, dependencies == data
- Legion:
  - Sequential semantic
- Parsec:
  - Parametrized task graph, dependencies and data
- Lapack/Scalapack:
  - No dynamic runtime

StarPU: http://starpu.gforge.inria.fr/
Legion: https://legion.stanford.edu/overview/index.html
Parsec: http://icl.utk.edu/parsec/

Dense Cholesky

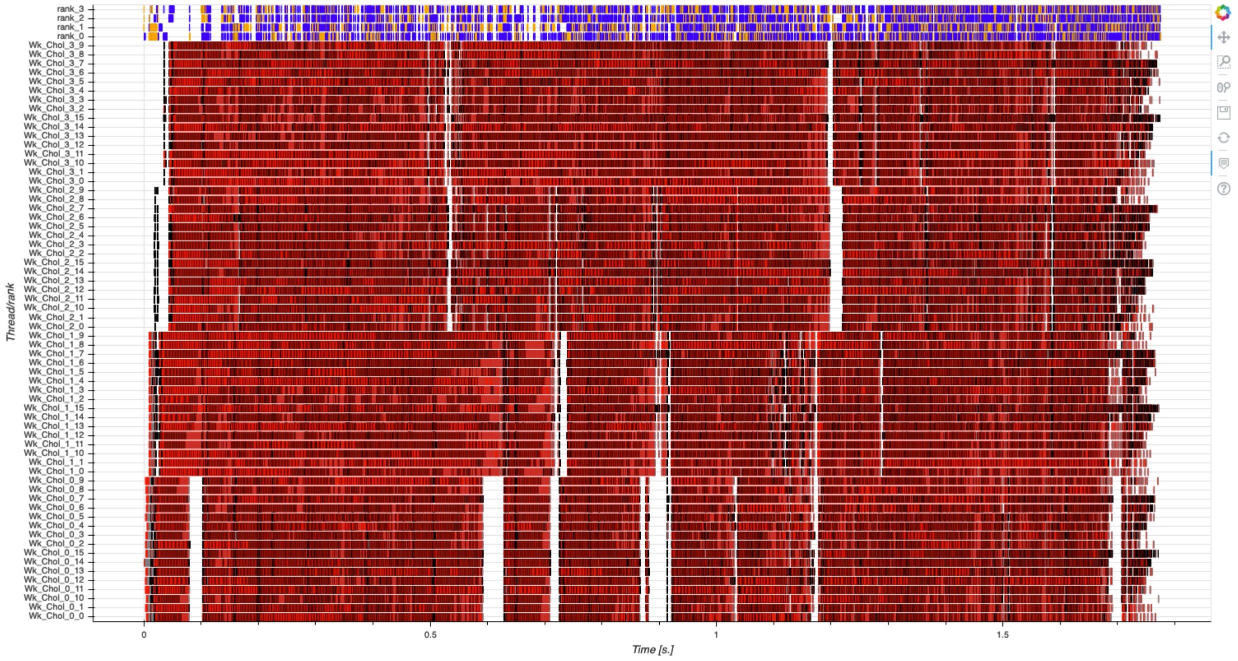




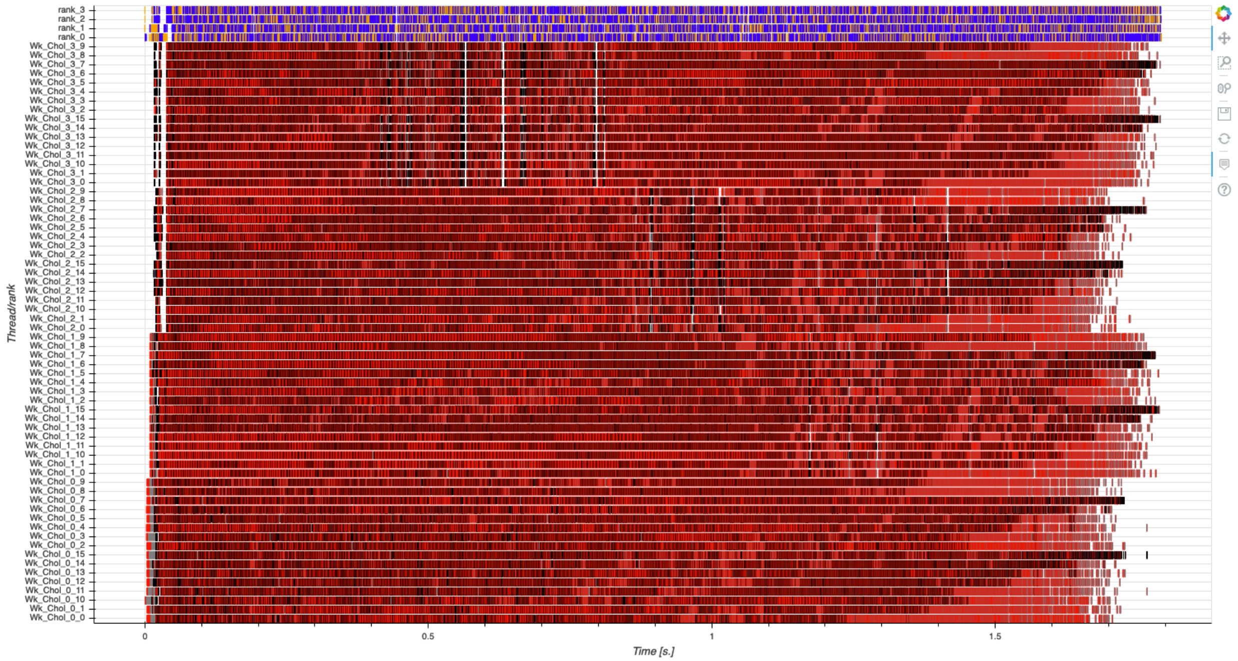


With Yizhou Qian

Tasks priorities matter



No priorities

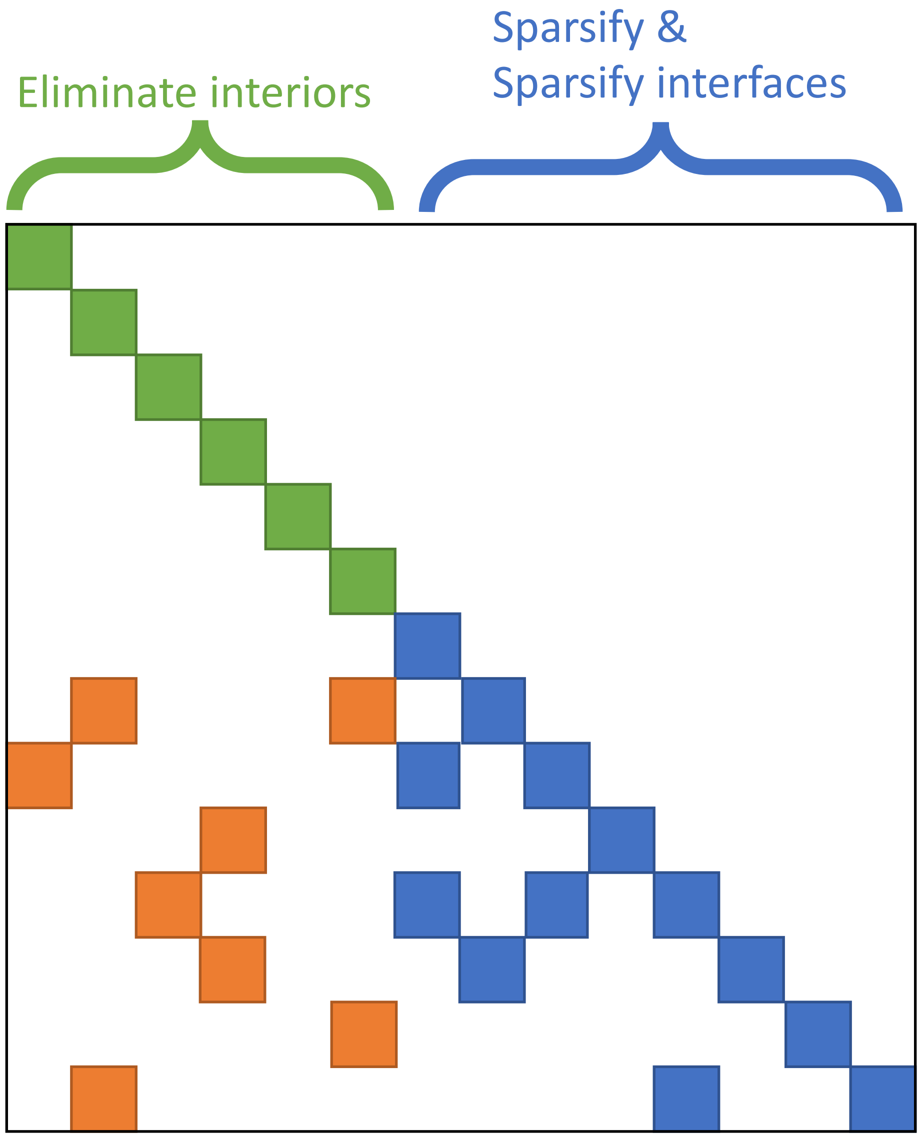


Row-based priorities

With Yizhou Qian

spaND

At a given level…



spaND

At a given level…

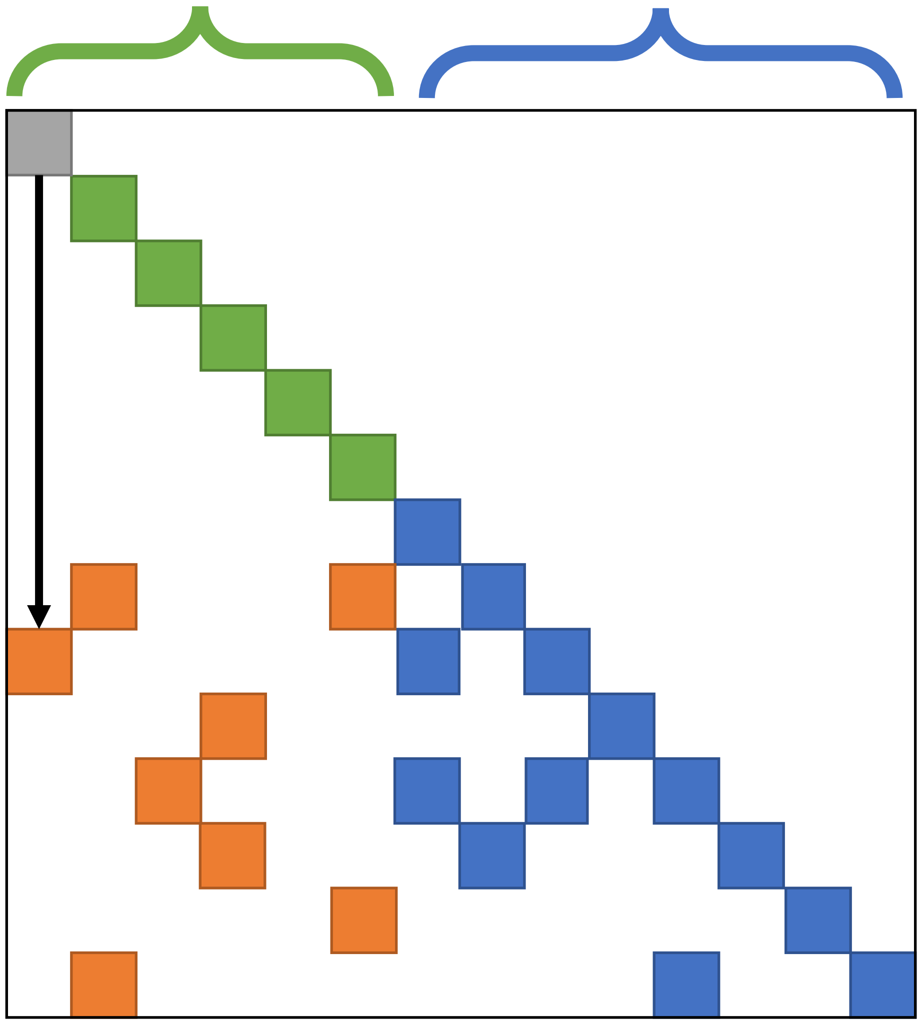
Eliminate leaves

$$L^{-1}\begin{bmatrix} A_{pp} & A_{pn} & \\ A_{np} & A_{nn} & A_{nw} \\ & A_{wn} & A_{ww} \end{bmatrix} U^{-1}$$

$$= \begin{bmatrix} I & & \\ & A_{nn} - A_{ns}A_{ss}^{-1}A_{sn} & A_{nw} \\ & A_{wn} & A_{ww} \end{bmatrix}$$

Eliminate interiors

Sparsify &
Sparsify interfaces



potrf(k) → trsm(k,i)

$$L_{ii} = chol(A_{ii})$$

$$A_{ij} \leftarrow L_{ii}^{-1}A_{ij}$$

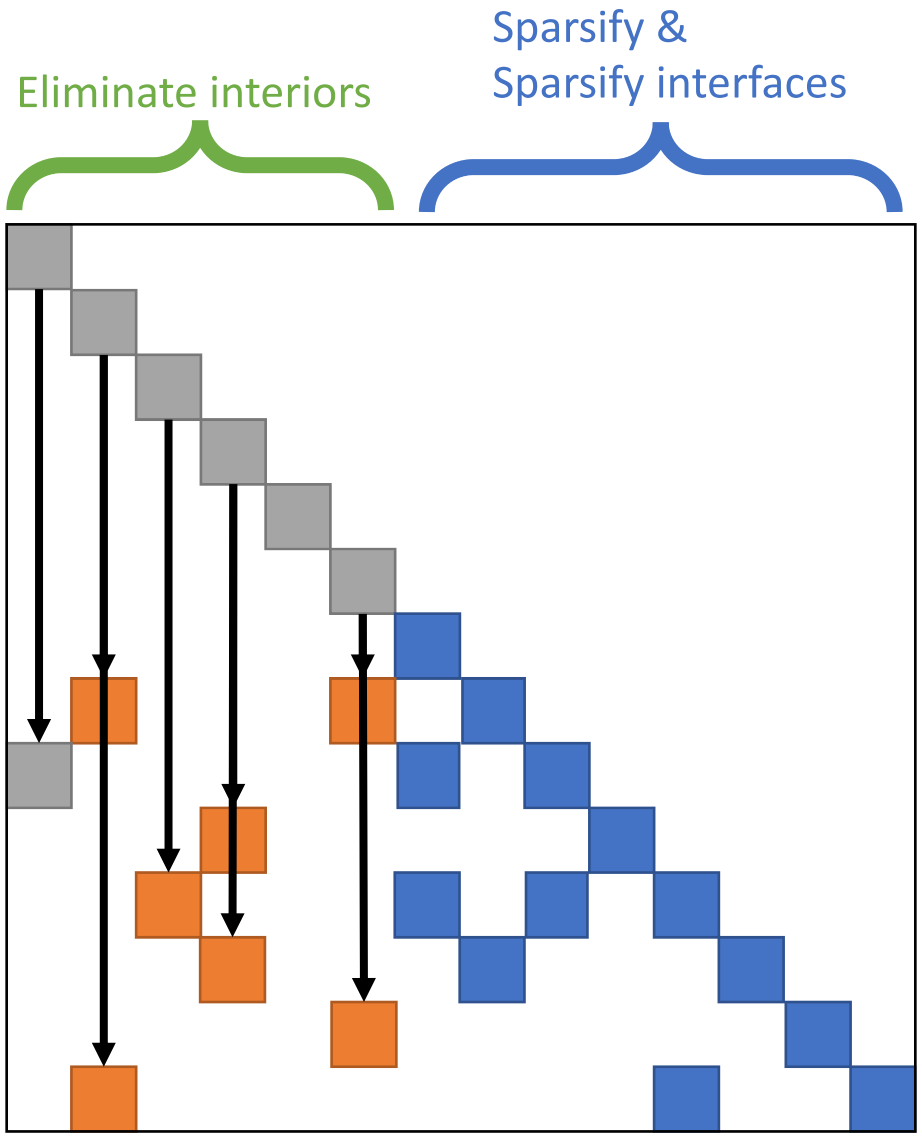
spaND

At a given level…

Eliminate leaves

$$L^{-1}\begin{bmatrix} A_{pp} & A_{pn} & \\ A_{np} & A_{nn} & A_{nw} \\ & A_{wn} & A_{ww} \end{bmatrix} L^{-\top}$$

$$= \begin{bmatrix} I & & \\ & A_{nn} - A_{ns}A_{ss}^{-1}A_{sn} & A_{nw} \\ & A_{wn} & A_{ww} \end{bmatrix}$$



potrf(k) → trsm(k,i)

$$L_{ii} = chol(A_{ii})$$

$$A_{ij} \leftarrow L_{ii}^{-1} A_{ij}$$

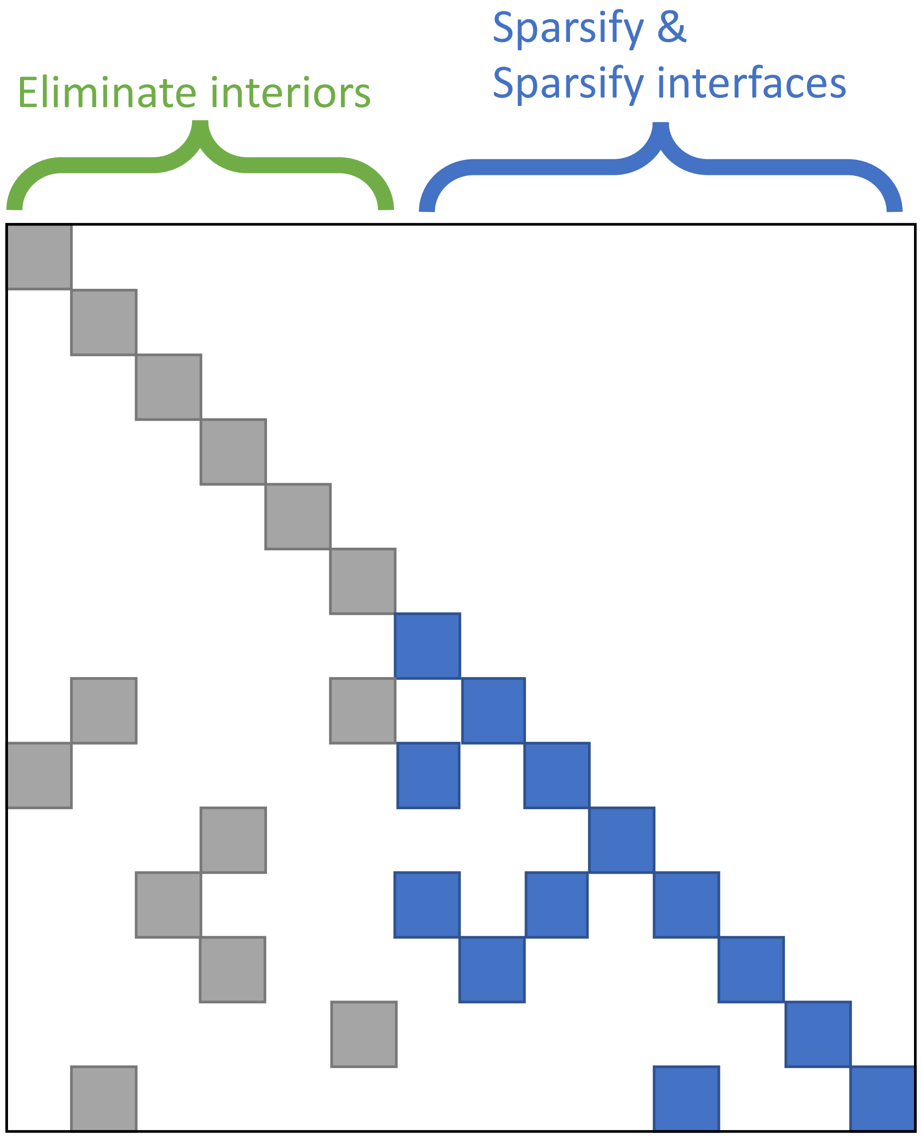
spaND

At a given level…

Eliminate leaves

$$L^{-1}\begin{bmatrix} A_{pp} & A_{pn} & \\ A_{np} & A_{nn} & A_{nw} \\ & A_{wn} & A_{ww} \end{bmatrix} L^{-\top}$$

$$= \begin{bmatrix} I & & \\ & A_{nn} - A_{ns}A_{ss}^{-1}A_{sn} & A_{nw} \\ & A_{wn} & A_{ww} \end{bmatrix}$$



potrf(k) → trsm(k,i)

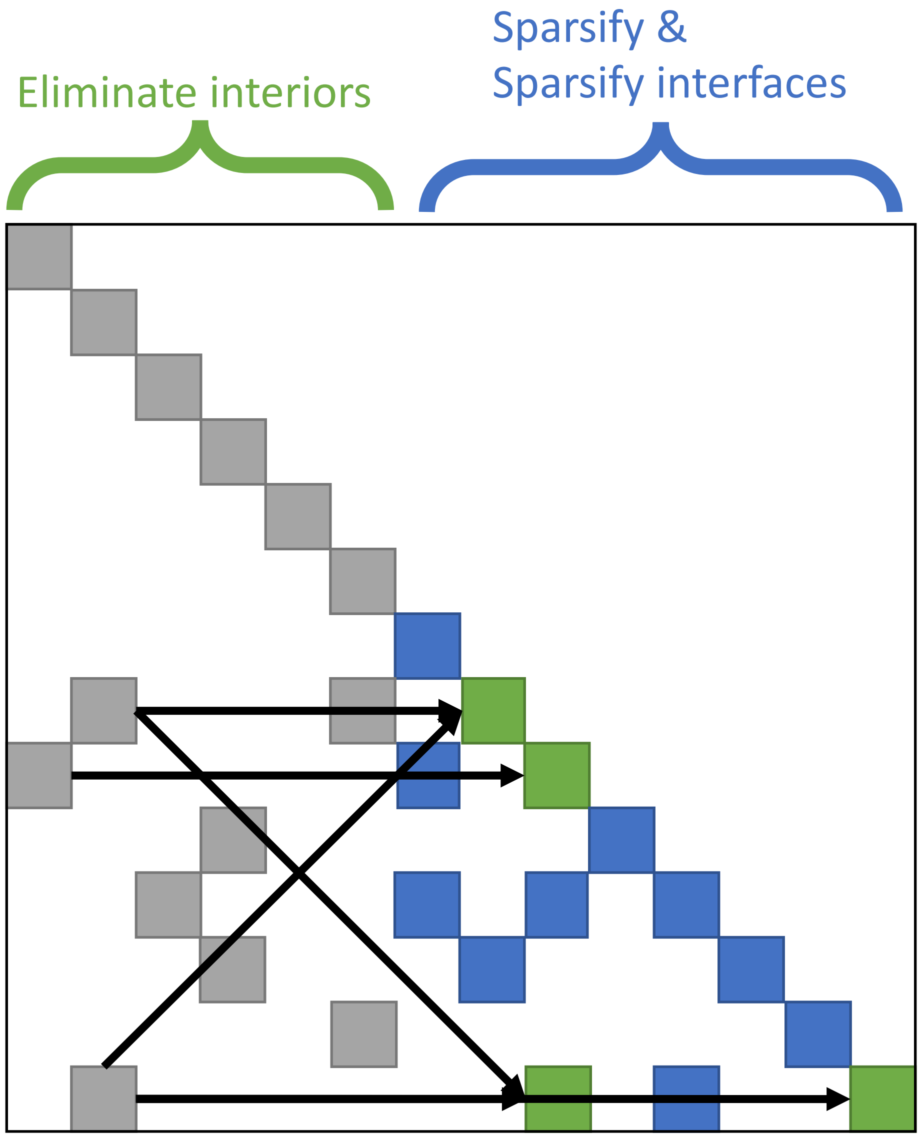
spaND

At a given level…

Eliminate leaves

$$L^{-1}\begin{bmatrix} A_{pp} & A_{pn} & \\ A_{np} & A_{nn} & A_{nw} \\ & A_{wn} & A_{ww} \end{bmatrix} L^{-\top}$$

$$= \begin{bmatrix} I & & \\ & A_{nn} - A_{ns}A_{ss}^{-1}A_{sn} & A_{nw} \\ & A_{wn} & A_{ww} \end{bmatrix}$$



potrf(k) → trsm(k,i)
trsm(k,i) & trsm(k,j) → gemm(i,j,k)

$$A_{ij} \mathrel{-}= L_{ik}L_{jk}^{\top}$$

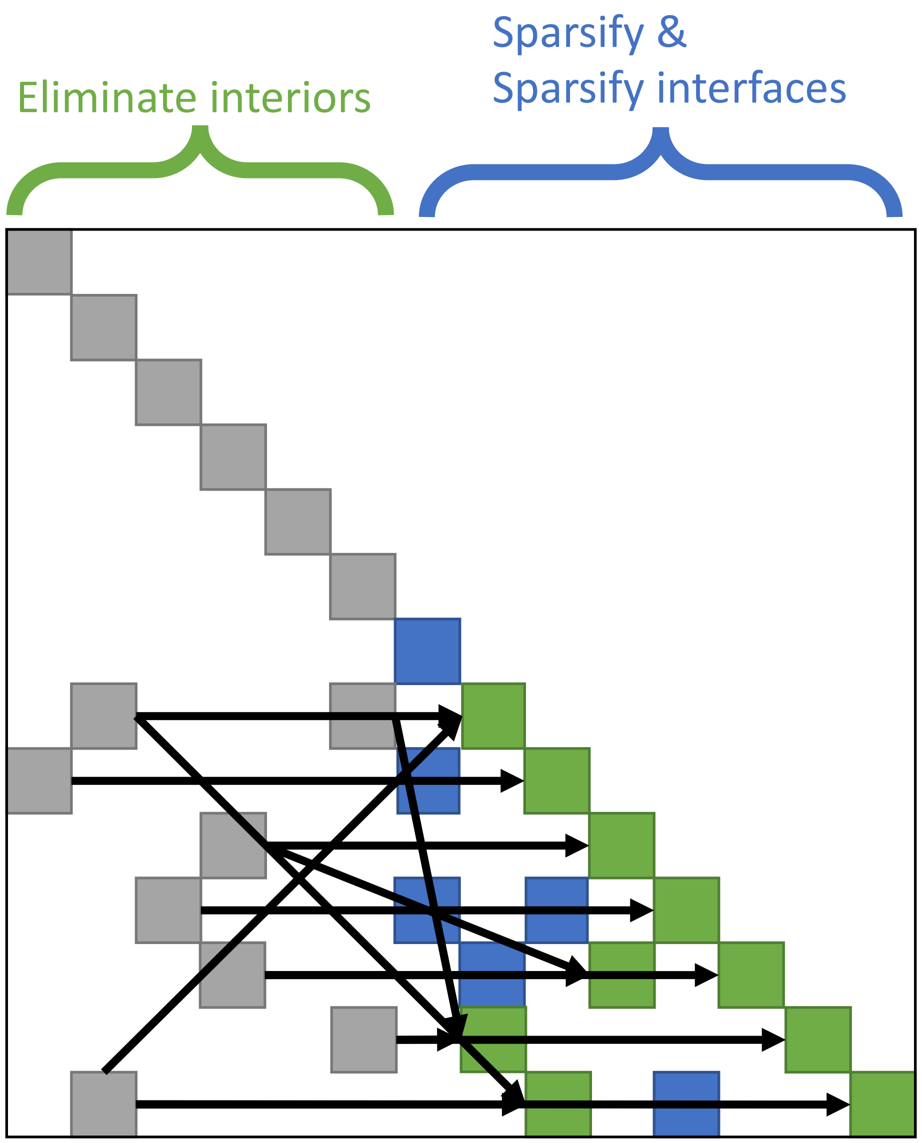
spaND

At a given level…

Eliminate leaves

$$L^{-1}\begin{bmatrix} A_{pp} & A_{pn} & \\ A_{np} & A_{nn} & A_{nw} \\ & A_{wn} & A_{ww} \end{bmatrix} L^{-\top}$$

$$= \begin{bmatrix} I & & \\ & A_{nn} - A_{ns}A_{ss}^{-1}A_{sn} & A_{nw} \\ & A_{wn} & A_{ww} \end{bmatrix}$$



potrf(k) → trsm(k,i)
trsm(k,i) & trsm(k,j) → gemm(i,j,k)

$$A_{ij} -= L_{ik}L_{jk}^{\top}$$

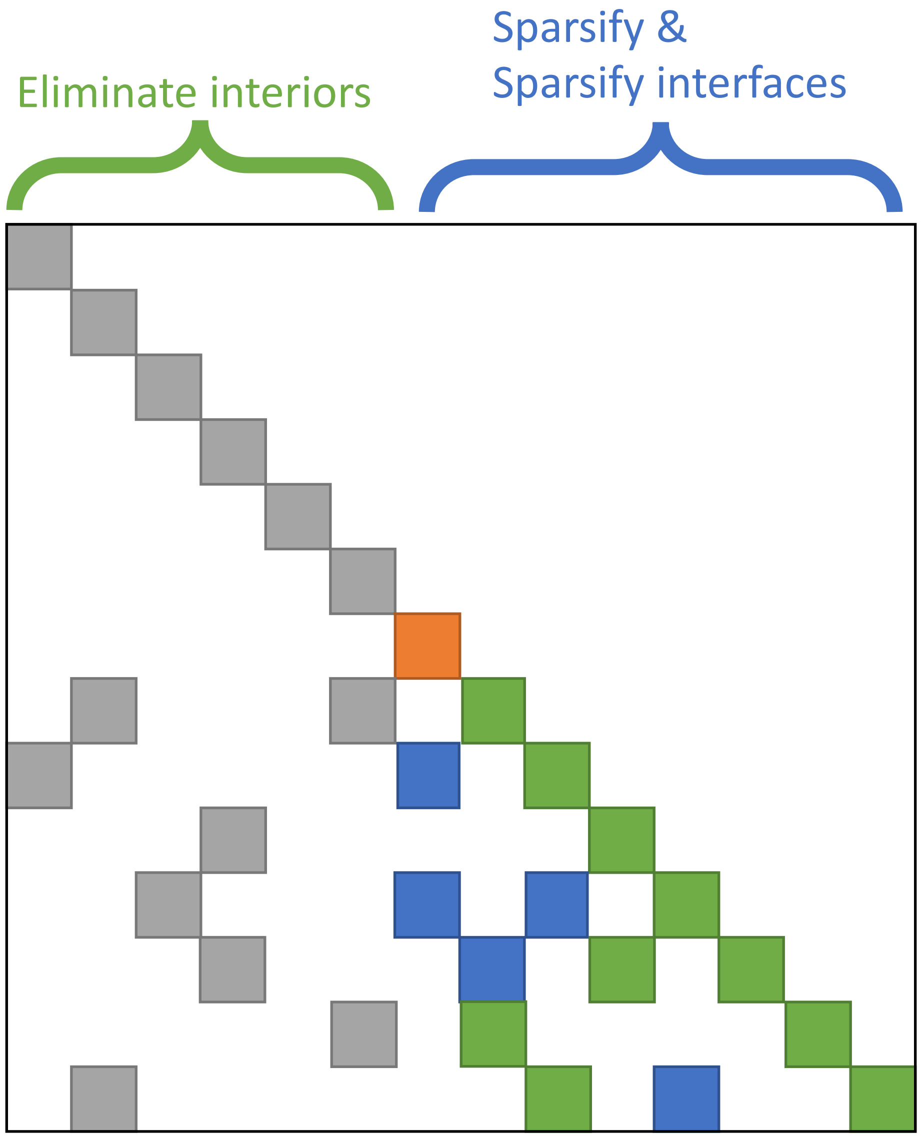
spaND

At a given level…

Scale interfaces

$$L^{-1}\begin{bmatrix} A_{pp} & A_{pn} & \\ A_{np} & A_{nn} & A_{nw} \\ & A_{wn} & A_{ww} \end{bmatrix} L^{-\top}$$

$$= \begin{bmatrix} I & L_{pp}^{-1}A_{pn} & \\ A_{np}L_{pp}^{-1} & A_{nn} & \\ & A_{wn} & A_{ww} \end{bmatrix}$$



potrf(k) → trsm(k,i)
trsm(k,i) & trsm(k,j) → gemm(i,j,k)
potrf(k) → trsm(k,i) & trsm(j,k)

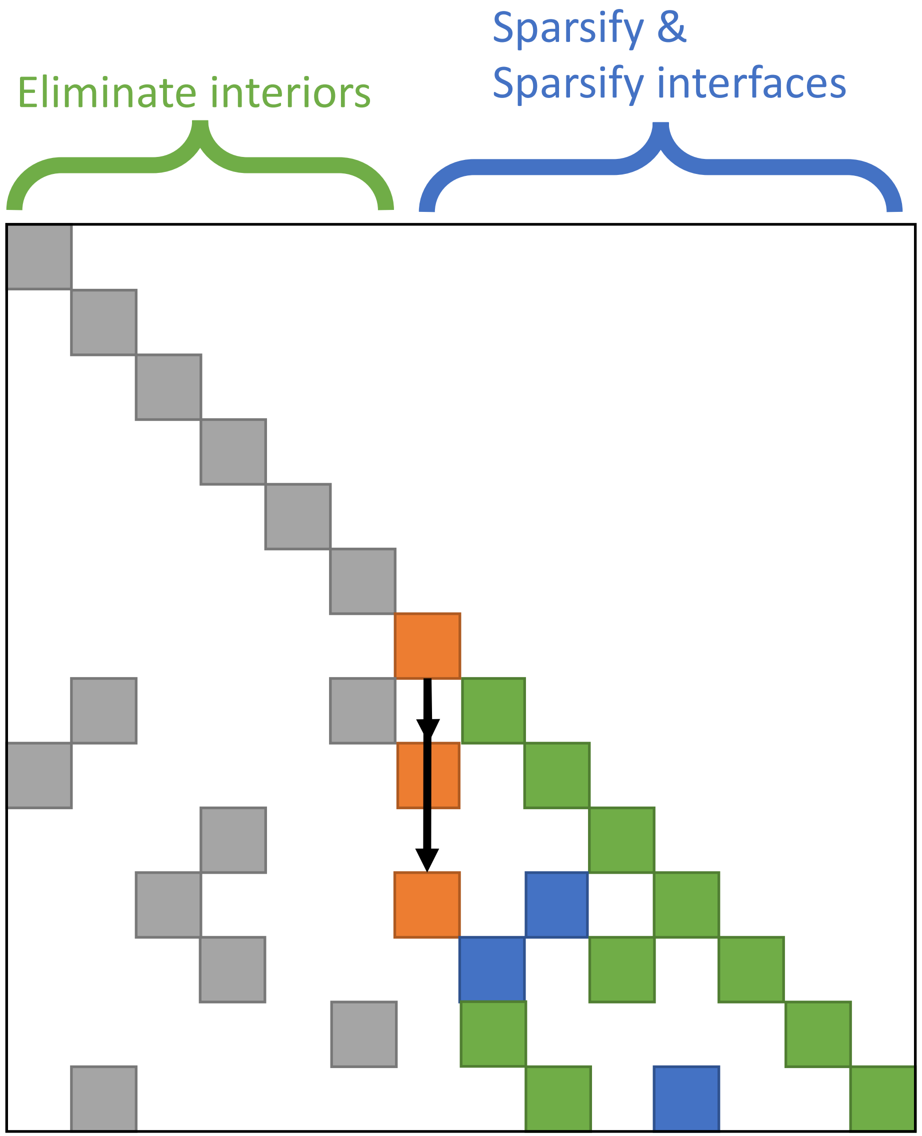
$$L_{ii} = chol\left(A_{ij}\right)$$

spaND

At a given level…

Scale interfaces

$$L^{-1}\begin{bmatrix} A_{pp} & A_{pn} & \\ A_{np} & A_{nn} & A_{nw} \\ & A_{wn} & A_{ww} \end{bmatrix} L^{-\top} = \begin{bmatrix} I & L_{pp}^{-1}A_{pn} & \\ A_{np}L_{pp}^{-1} & A_{nn} & \\ & A_{wn} & A_{ww} \end{bmatrix}$$



potrf(k) → trsm(k,i)
trsm(k,i) & trsm(k,j) → gemm(i,j,k)
potrf(k) → trsm(k,i) & trsm(j,k)

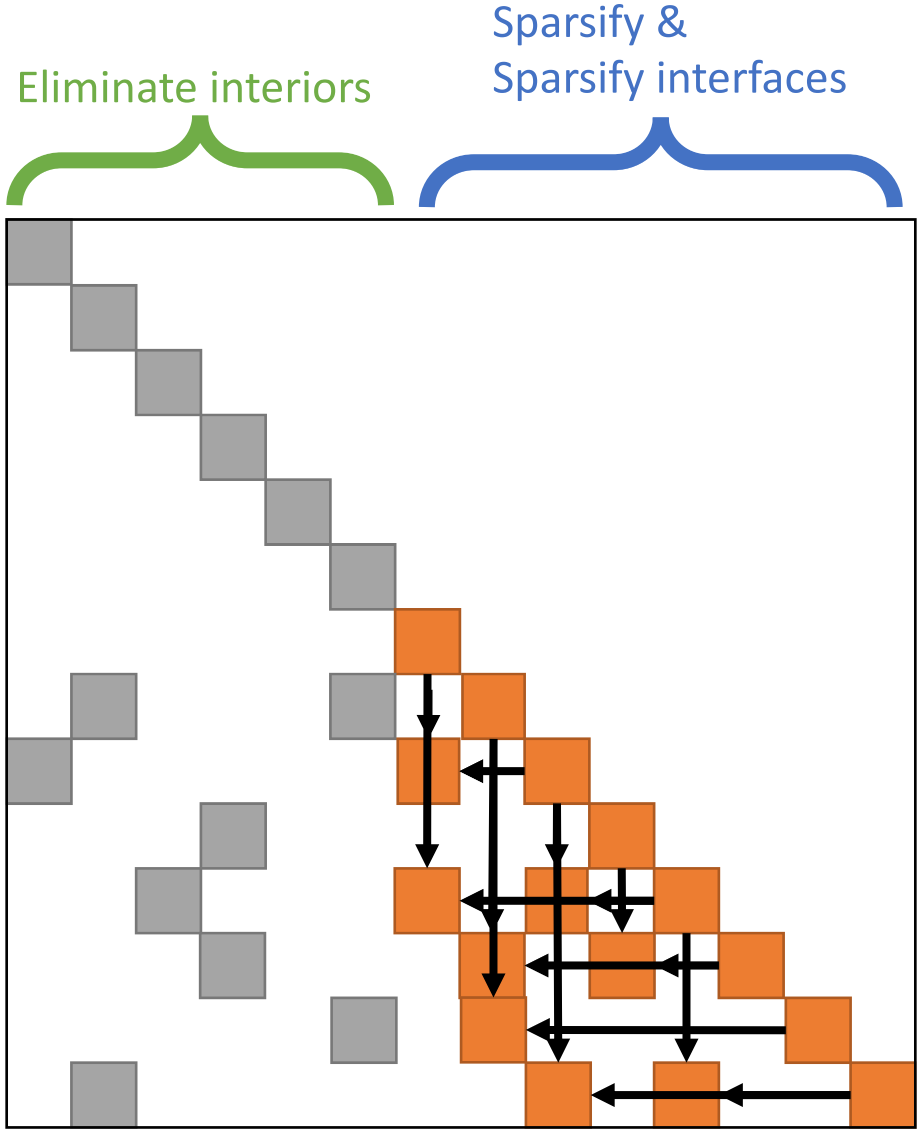
$$L_{ii} = chol(A_{ij})$$
$$A_{ij} \leftarrow L_{ii}^{-1} A_{ij} L_{jj}^{-1}$$

spaND

At a given level…

Scale interfaces

$$L^{-1}\begin{bmatrix} A_{pp} & A_{pn} & \\ A_{np} & A_{nn} & A_{nw} \\ & A_{wn} & A_{ww} \end{bmatrix} L^{-\top}$$
$$= \begin{bmatrix} I & L_{pp}^{-1}A_{pn} & \\ A_{np}L_{pp}^{-1} & A_{nn} & \\ & A_{wn} & A_{ww} \end{bmatrix}$$

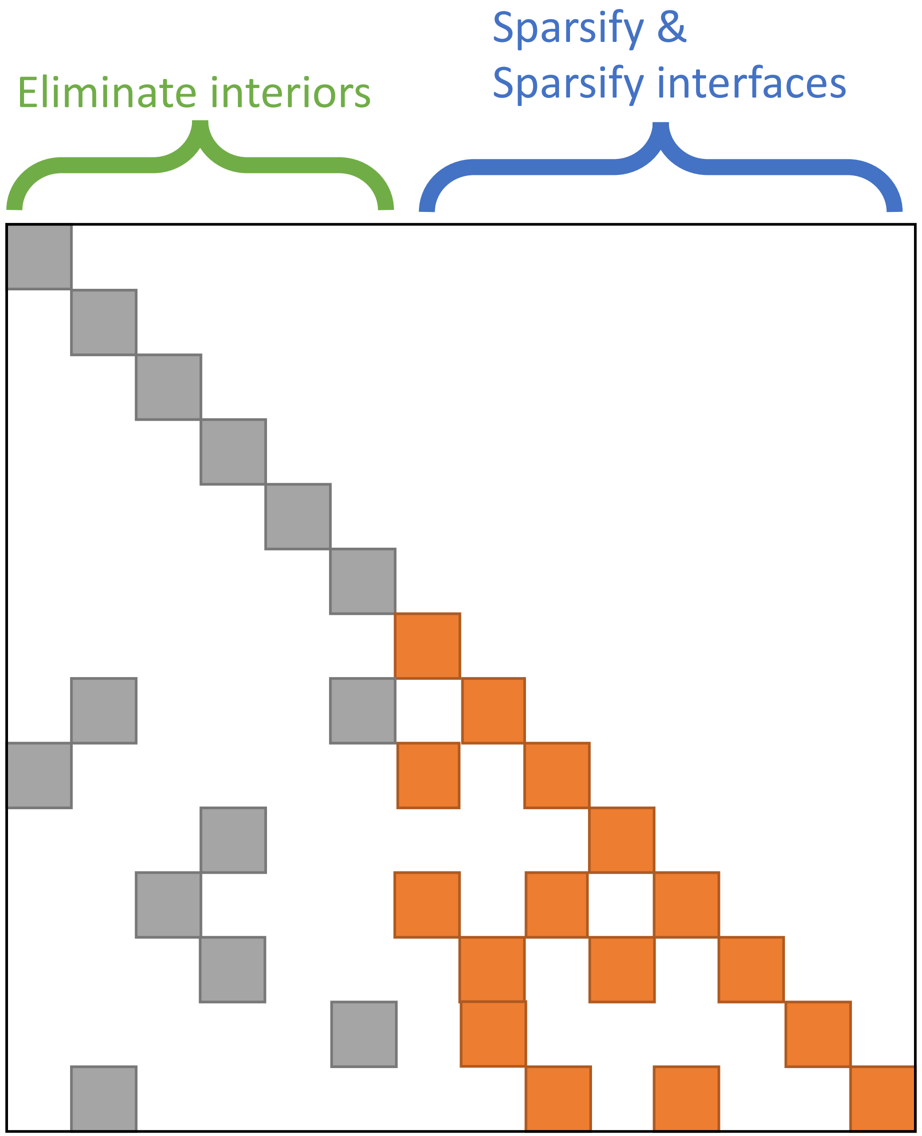


potrf(k) → trsm(k,i)
trsm(k,i) & trsm(k,j) → gemm(i,j,k)
potrf(k) → trsm(k,i) & trsm(j,k)

$$L_{ii} = chol(A_{ij})$$
$$A_{ij} \leftarrow L_{ii}^{-1} A_{ij} L_{jj}^{-1}$$

spaND

At a given level…



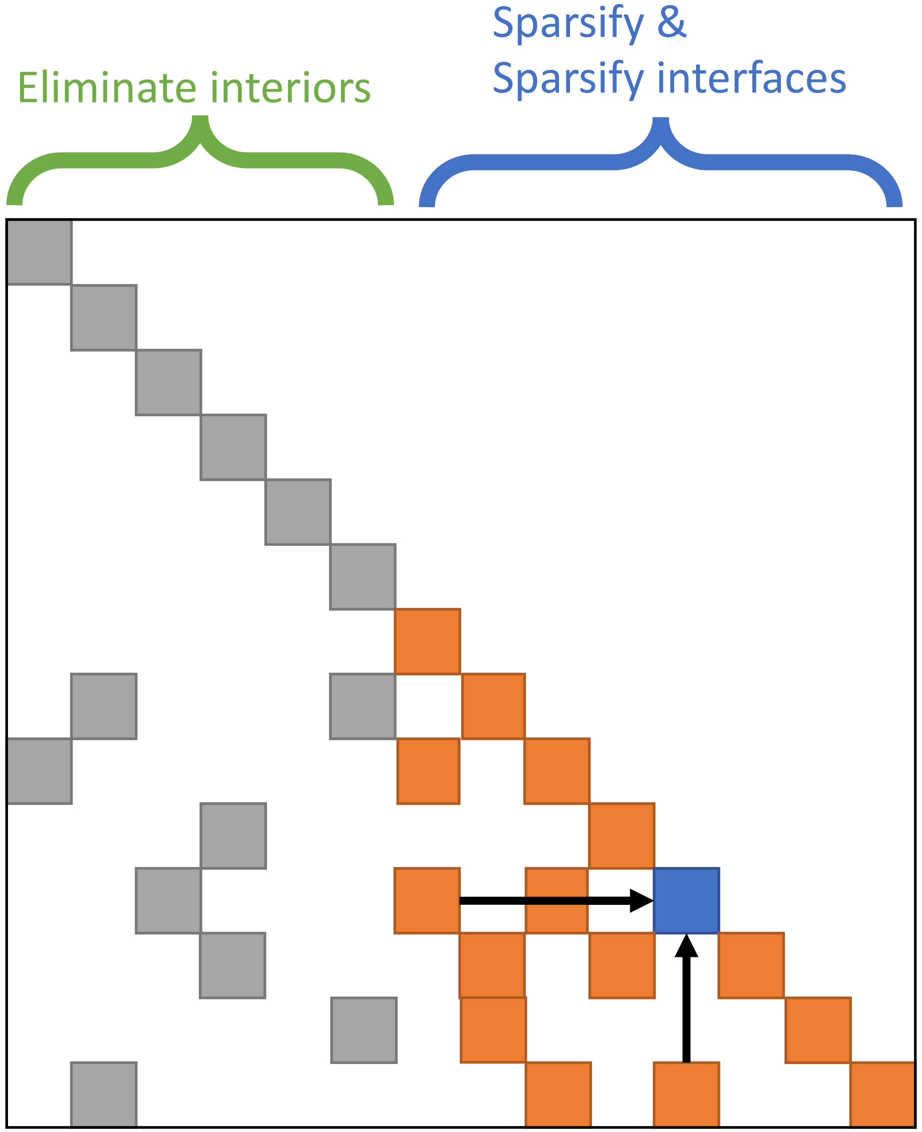
potrf(k) → trsm(k,i)
trsm(k,i) & trsm(k,j) → gemm(i,j,k)
potrf(k) → trsm(k,i) & trsm(j,k)

spaND

At a given level...

Sparsify interfaces

$$\begin{bmatrix} Q_p^T & \\ & I \end{bmatrix}\begin{bmatrix} I & A_{pn} \\ A_{np} & A_{nn}\end{bmatrix}\begin{bmatrix} Q_p & \\ & I\end{bmatrix} = \begin{bmatrix} I & & \varepsilon \\ & I & W_{cn} \\ \varepsilon & W_{nc} & A_{nn}\end{bmatrix}$$



potrf(k) → trsm(k,i)
trsm(k,i) & trsm(k,j) → gemm(i,j,k)
potrf(k) → trsm(k,i) & trsm(j,k)
trsm(j,i) & trsm(i,j) → rrqr(i), rrqr(j)

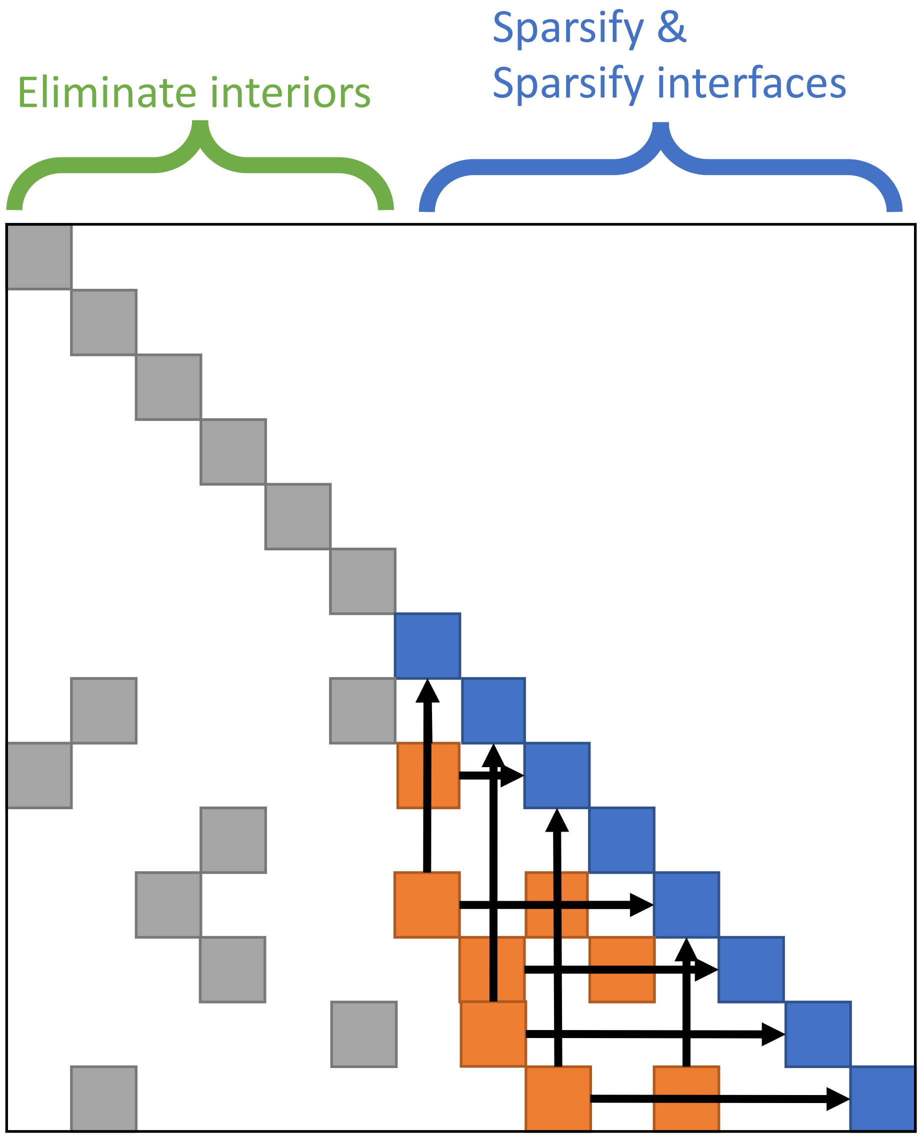
$$Q_c W_c \approx [A_{sn_1} \quad \dots \quad A_{sn_k}]$$

spaND

At a given level…

Sparsify interfaces

$$\begin{bmatrix} Q_p^T & \\ & I \end{bmatrix} \begin{bmatrix} I & A_{pn} \\ A_{np} & A_{nn} \end{bmatrix} \begin{bmatrix} Q_p & \\ & I \end{bmatrix} = \begin{bmatrix} I & & \varepsilon \\ & I & W_{cn} \\ \varepsilon & W_{nc} & A_{nn} \end{bmatrix}$$



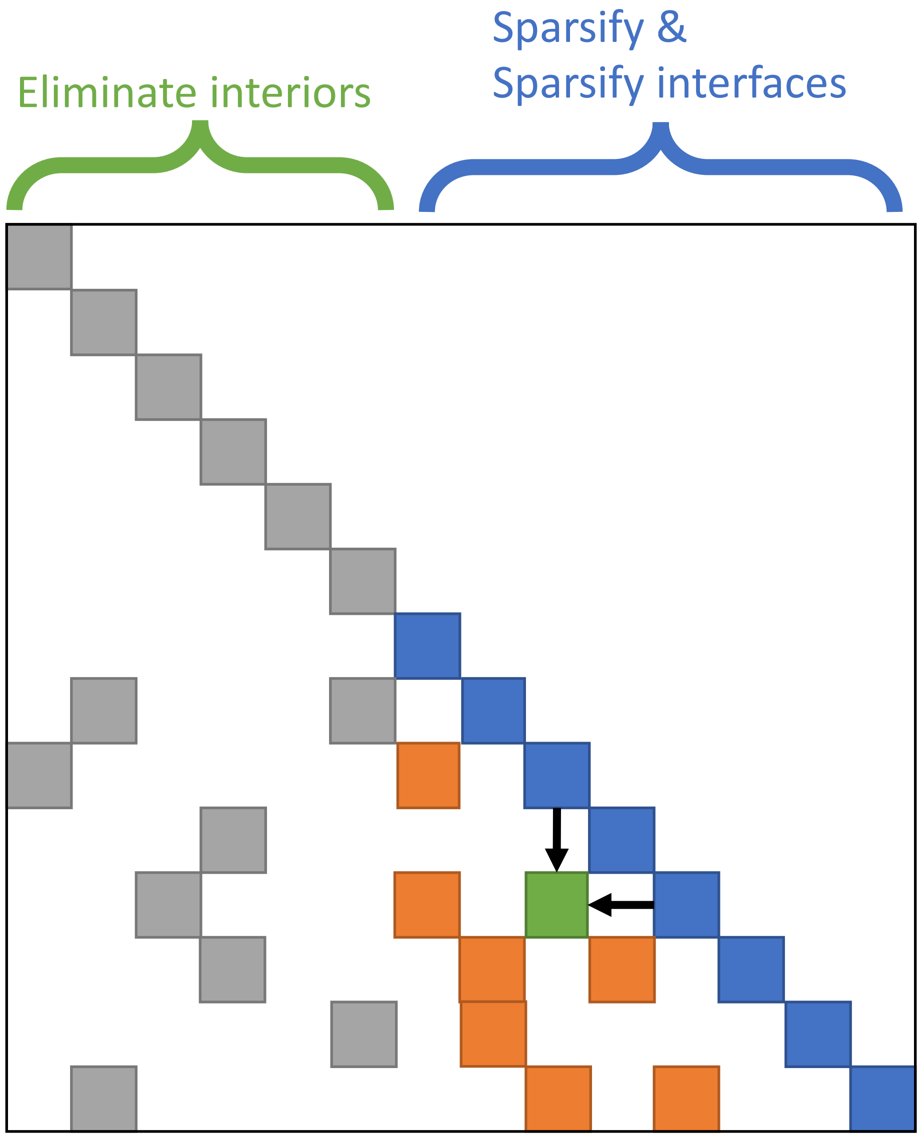
potrf(k) → trsm(k,i)
trsm(k,i) & trsm(k,j) → gemm(i,j,k)
potrf(k) → trsm(k,i) & trsm(j,k)
trsm(j,i) & trsm(i,j) → rrqr(i), rrqr(j)

spaND

At a given level…

Sparsify interfaces

$$\begin{bmatrix} Q_p^T & \\ & I \end{bmatrix} \begin{bmatrix} I & A_{pn} \\ A_{np} & A_{nn} \end{bmatrix} \begin{bmatrix} Q_p & \\ & I \end{bmatrix} = \begin{bmatrix} I & & \varepsilon \\ & I & W_{cn} \\ \varepsilon & W_{nc} & A_{nn} \end{bmatrix}$$



potrf(k) → trsm(k,i)
trsm(k,i) & trsm(k,j) → gemm(i,j,k)
potrf(k) → trsm(k,i) & trsm(j,k)
trsm(j,i) & trsm(i,j) → rrqr(i), rrqr(j)
rrqr(i) & rrqr(j) → ormqr(i,j)

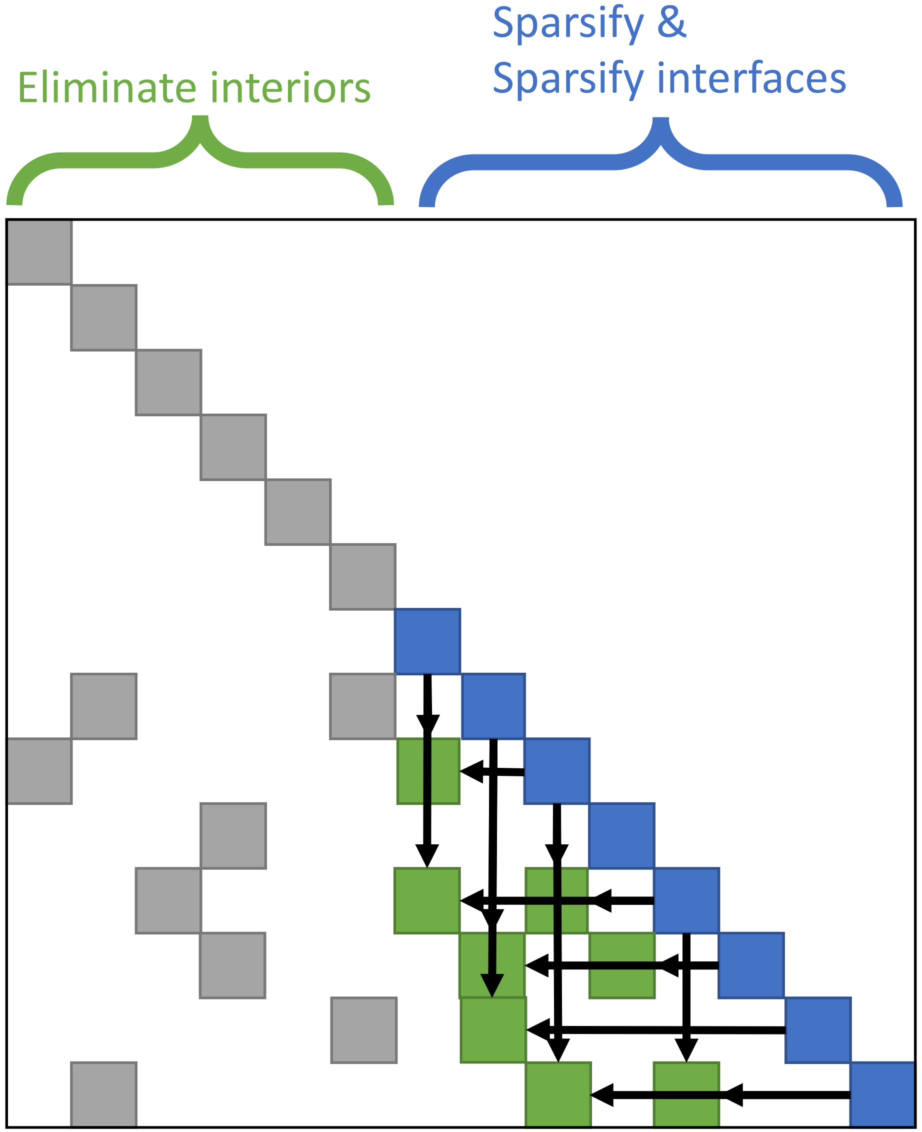
$$A_{ij} \leftarrow Q_{ci}^{\mathsf{T}} A_{ij} Q_{cj}^{\mathsf{T}}$$

spaND

At a given level…

Sparsify interfaces

$$\begin{bmatrix} Q_p^T & \\ & I \end{bmatrix} \begin{bmatrix} I & A_{pn} \\ A_{np} & A_{nn} \end{bmatrix} \begin{bmatrix} Q_p & \\ & I \end{bmatrix} = \begin{bmatrix} I & & \varepsilon \\ & I & W_{cn} \\ \varepsilon & W_{nc} & A_{nn} \end{bmatrix}$$

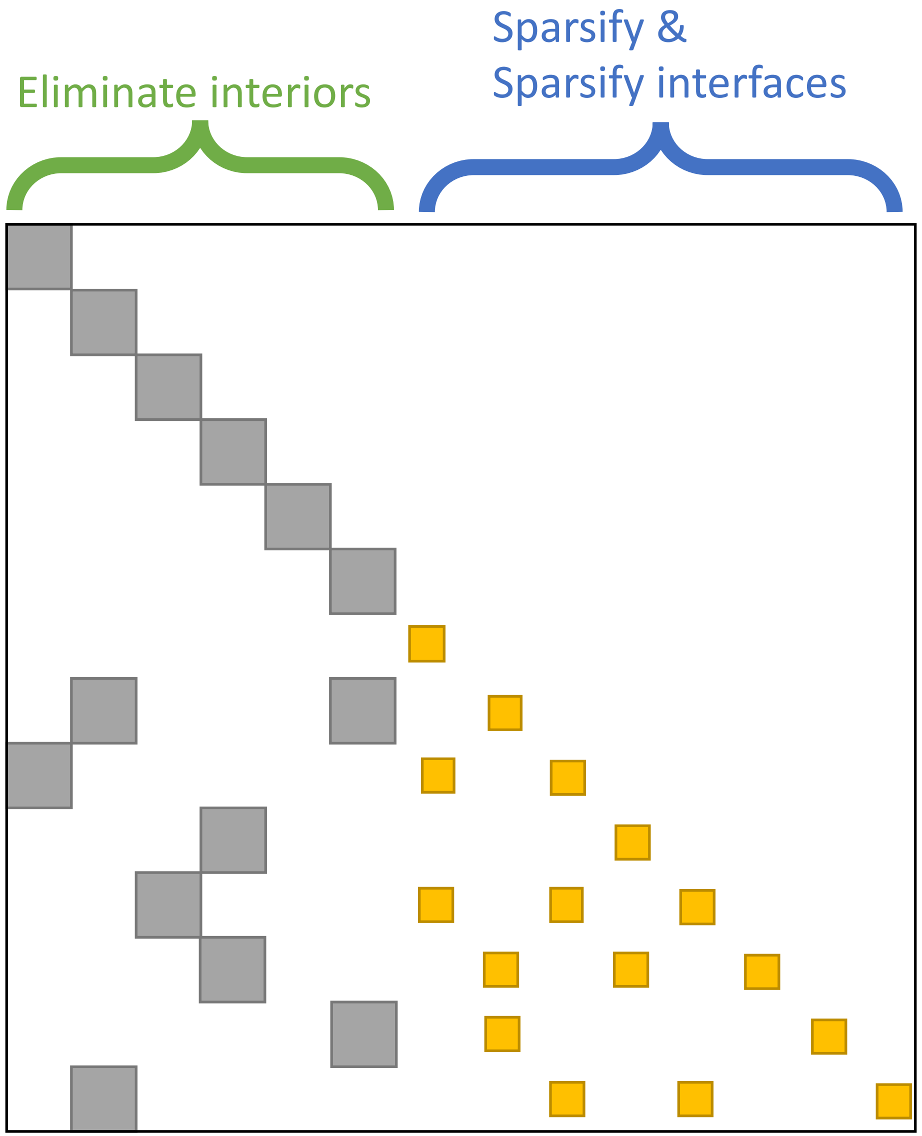


potrf(k) → trsm(k,i)
trsm(k,i) & trsm(k,j) → gemm(i,j,k)
potrf(k) → trsm(k,i) & trsm(j,k)
trsm(j,i) & trsm(i,j) → rrqr(i), rrqr(j)
rrqr(i) & rrqr(j) → ormqr(i,j)

$$A_{ij} \leftarrow Q_{ci}^{\top} A_{ij} Q_{cj}^{\top}$$

spaND

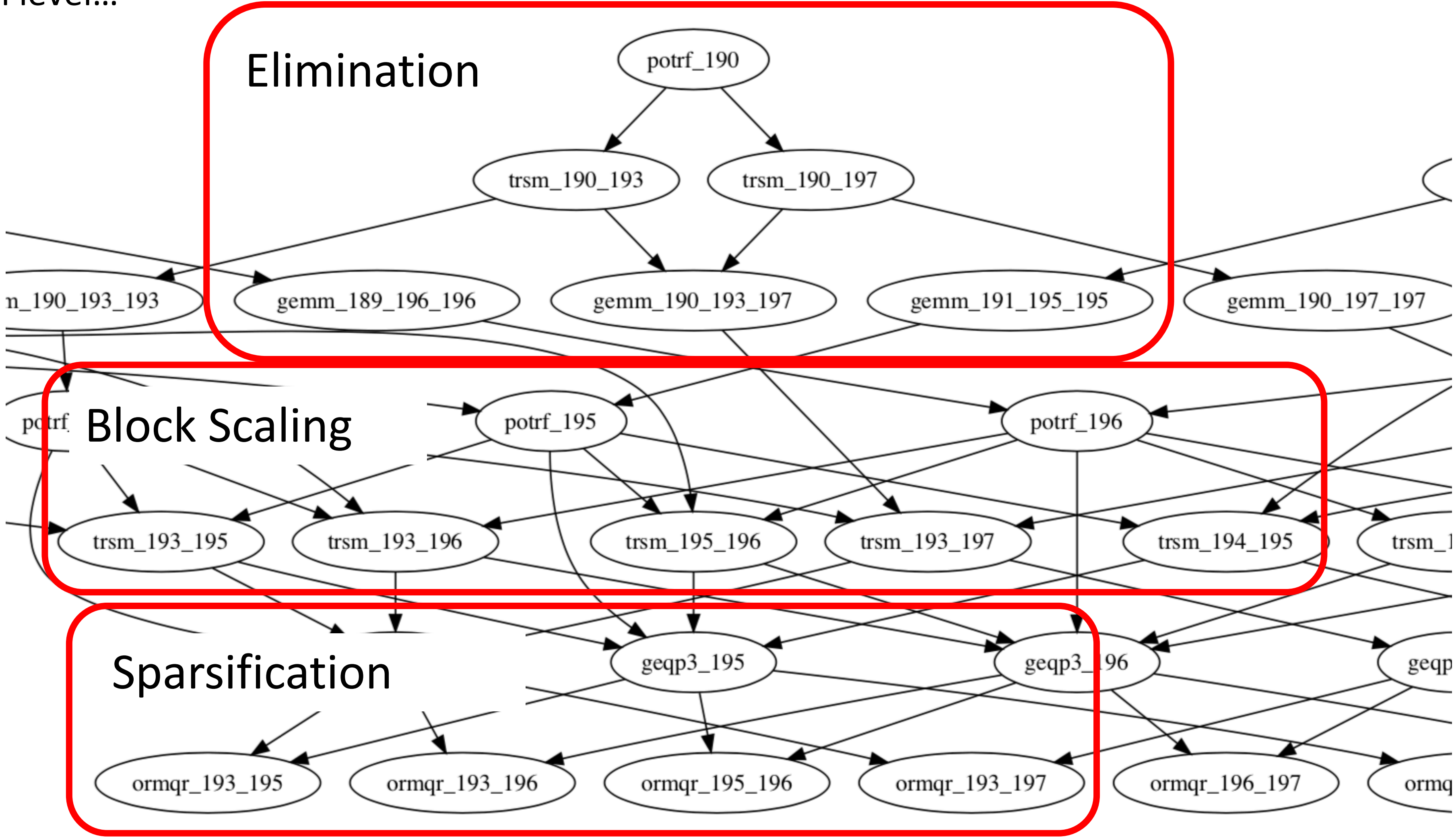
At a given level…



potrf(k) → trsm(k,i)
trsm(k,i) & trsm(k,j) → gemm(i,j,k)
potrf(k) → trsm(k,i) & trsm(j,k)
trsm(j,i) & trsm(i,j) → rrqr(i), rrqr(j)
rrqr(i) & rrqr(j) → ormqr(i,j)

spaND

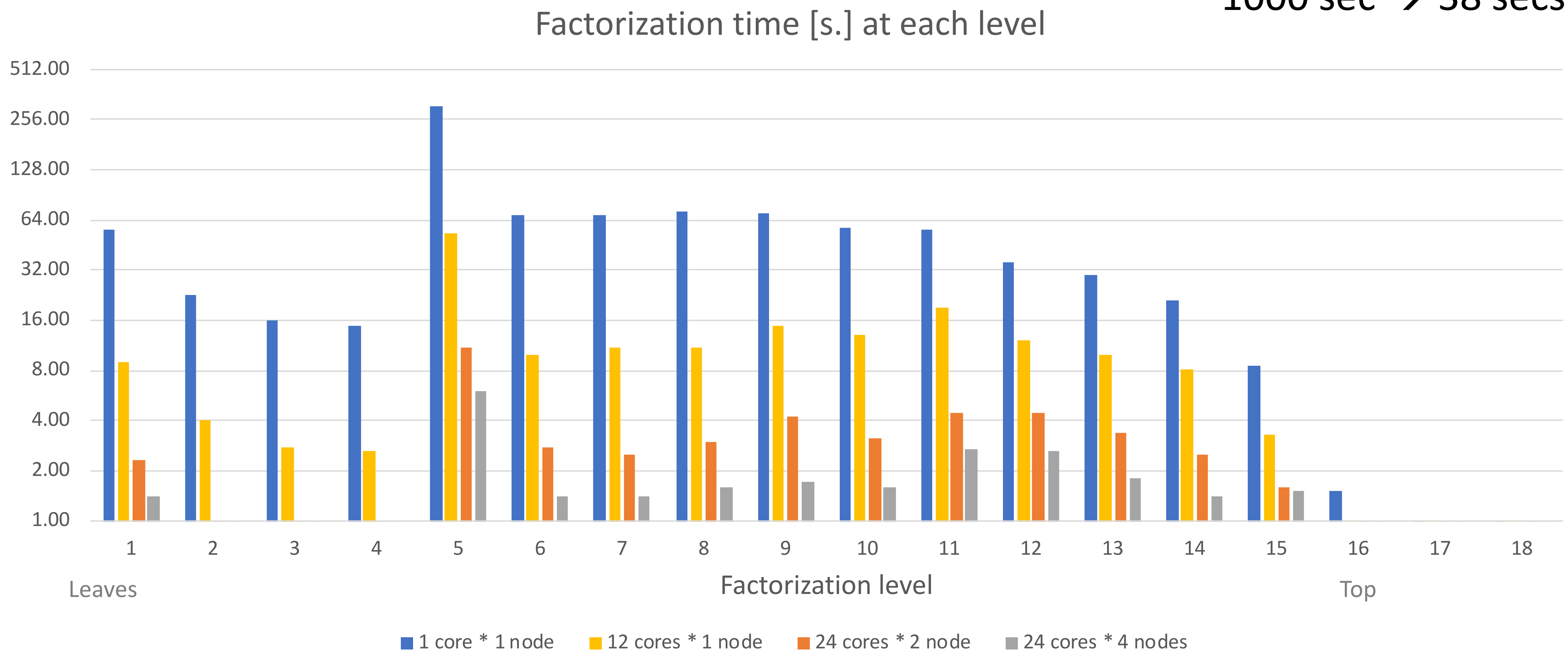
At a given level…



spaND

16M, 7-points stencil-like

Total Fact. time ≈
1000 sec → 38 secs



Conclusions and Future Work

- General algebraic algorithm, works on large class of problems
- Preliminary parallel task-based version with TaskTorrent (https://github.com/leopoldcambier/tasktorrent) runtime

Future work

- spaND + ttor: improved mapping block → rank
  - Hierarchical partitioning w/ minimization of communication cost between levels
- spaND + ttor: more scalable RRQR
  - Top can cost up to O(N)

Acknowledgements & references

- Initial spaND work with Chao Chen, Eric Darve, Erik Boman, Ray Tuminaro, Siva Rajamanickam
- TaskTorrent work with Eric Darve and Yizhou Qian
- Support from Sandia National Lab (spaND) & Total (spaND & TaskTorrent)

- spaND: Cambier, Léopold, et al. "An algebraic sparsified nested dissection algorithm using low-rank approximations." *arXiv preprint arXiv:1901.02971* (2019). To appear in SIMAX
- spaND w/ near nullspace preservation: Klockiewicz, Bazyli, and Eric Darve. "Sparse hierarchical preconditioners using piecewise smooth approximations of eigenvectors." *arXiv preprint arXiv:1907.03406* (2019).
- original spaND (HIF): Ho, Kenneth L., and Lexing Ying. "Hierarchical interpolative factorization for elliptic operators: differential equations." *Communications on Pure and Applied Mathematics* 69.8 (2016): 1415-1451.
- HIF + Block scaling: Feliu-Fabà, Jordi, Kenneth L. Ho, and Lexing Ying. "Recursively Preconditioned Hierarchical Interpolative Factorization for Elliptic Partial Differential Equations." *arXiv preprint arXiv:1808.01364* (2018).
- TaskTorrent: `https://github.com/leopoldcambier/tasktorrent`