Exceptional service in the national interest

SAND2021-2556C

Photos placed in horizontal position with even amount of white space between photos and header

Hierarchical Low-rank Solvers for Large Sparse Linear Systems

Erik Boman, Sandia National Labs

SIAM CS&E, Mar. 5, 2021

Collaborators

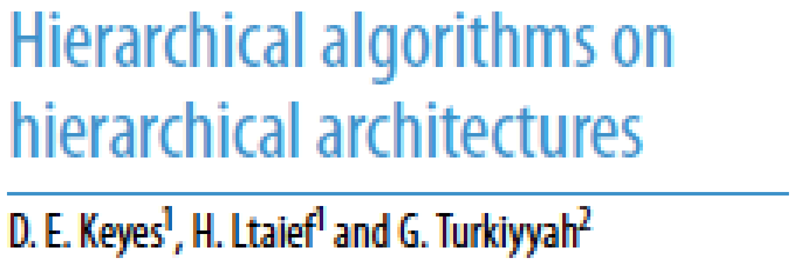
- Stanford:
  - Leopold Cambier , Chao Chen (now UT Austin), Eric Darve
- Sandia:
  - Siva Rajamanickam, Ray Tuminaro, Ichi Yamazaki

Motivation/goals

- Sparse direct factorization is robust but too expensive in 3D
- Want robust "black-box" approximate factorization
  - Use as preconditioner
  - Allow trade-off fill versus quality
- Current methods have limitations
  - AMG and DD are scalable but often not robust (e.g., indefinite, nonsym.)
  - Others (Incomplete factorizations, Sparse approximate inverses etc) are algebraic (black-box) but not scalable
- Hierarchical matrix methods could fill this gap
  - Many different algorithms
  - Will focus on a couple I have been involved with

Hierarchical algorithms+architectures

- Modern computers have hierarchical designs
  - Shared-memory nodes with slow interconnect
  - Each node may have multiple CPU, GPU, other hardware
  - Memory access is usually the bottleneck, not flops
- Challenge: Map hierarchical algorithms to hierarchical architectures



Hierarchical algorithms on hierarchical architectures

D. E. Keyes[1], H. Ltaief[1] and G. Turkiyyah[2]

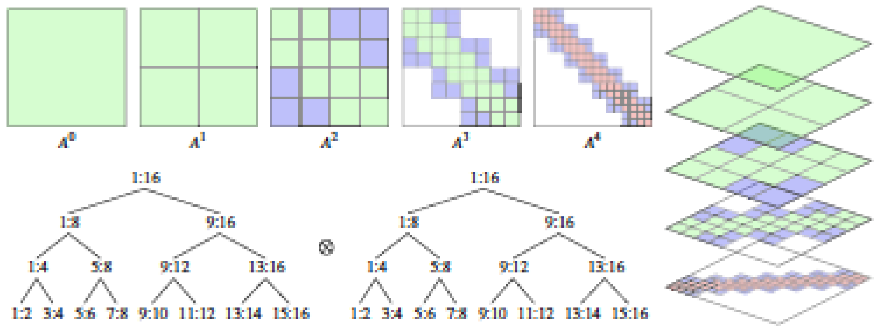


Figure 7. Recursive construction of an $\mathcal{H}$-matrix. Starting from the top-level row and column cluster trees are dual traversed, checking if a matrix block is admissible as a low-rank block (blue) or needs to be further subdivided (green) with its children checked at the next level. At the finest level, small inadmissible blocks are stored as dense blocks (red). (Online version in colour.)

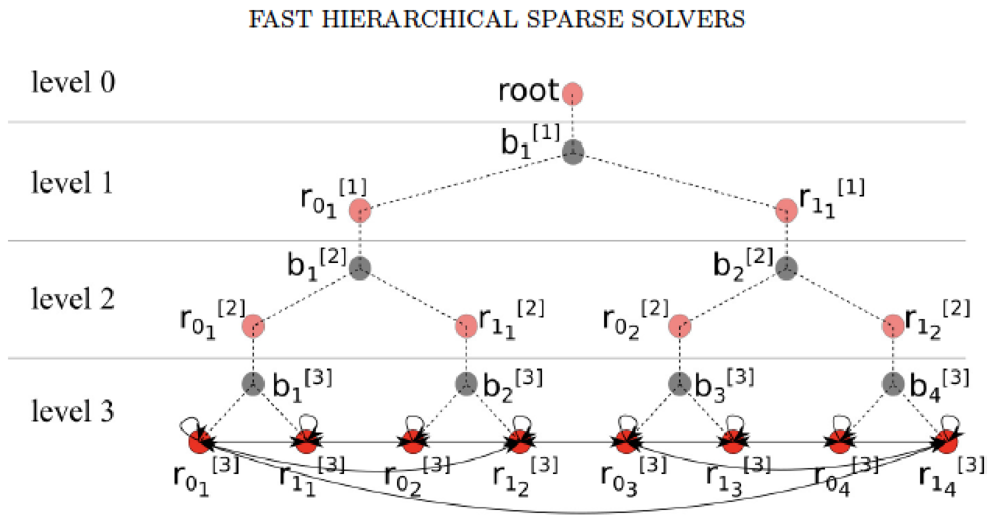
Context: Direct solvers

- Hierarchical matrix solvers are algebraic versions of FMM
  - “short range” (blocks near diagonal) must be accurate
  - “long range” (off-diagonal blocks) can be approximated by low-rank
  - Many variations: H, H^2, HSS, HODLR
- Hierarchical vs BLR (tiled)
  - Block (tiled) low-rank is “flat” not hierarchical
  - Asymptotically, hierarchical is better but hard to implement
    - BLR is simpler and often faster for moderate problem sizes
- Accelerate dense frontal calculations within sparse solvers
  - MUMPS: multifrontal with BLR
  - Strumpack: multifrontal with HSS, HODLR, HOD-BF
- Our focus: Other types of sparse solvers
  - Simpler, avoid the “extend-add” issue

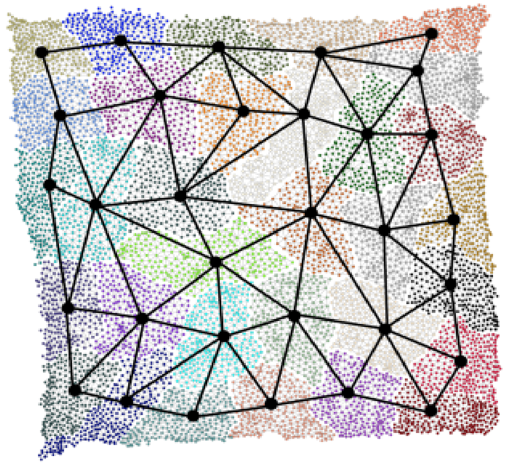
LoRaSp

LoRaSp is a hierarchical sparse solver (Pouransari et al., SISC'17)

- Approximate block Gaussian elimination
  - Similar to multilevel domain decomposition
  - Solve inexactly on subdomains using low-rank approx.
  - Recurse on remaining global problem
  - Small subdomains, treated as dense



ParH2

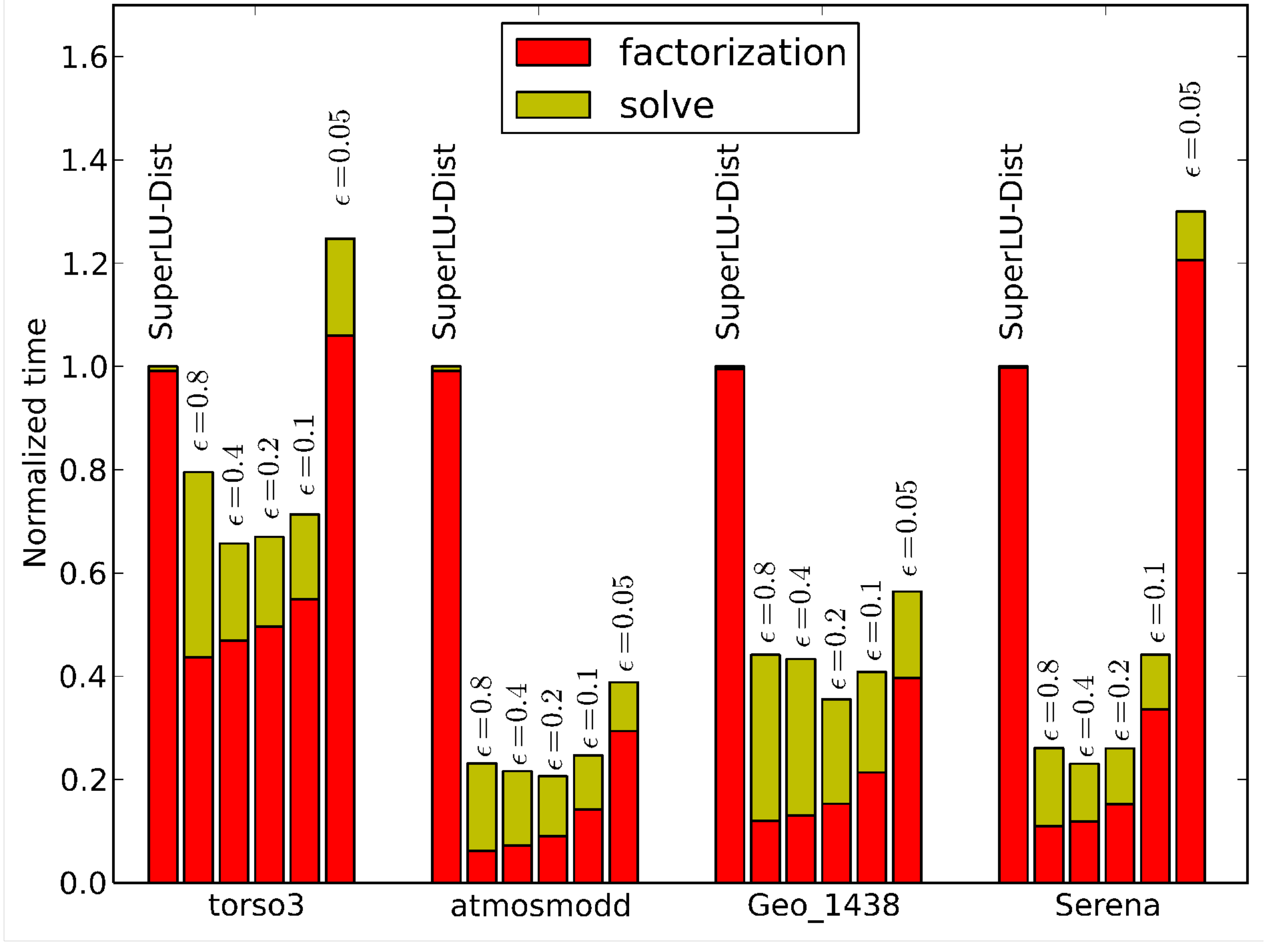


ParH2 (Chen et al., ParCo 2018, JCP 2019):

- Parallel (MPI) version of LoRaSp
  - Can solve much larger problems
- Improved/simplified derivation
  - Gaussian elimination with sparsification, no more red/black nodes
- Improved robustness with ”deferred compression”
  - Block diagonal scaling, aka “scale-and-compress”
  - Similar to Xia et al., (2017), also used in eSIF

Results: ParH2 vs. sparse direct.

Compare hierarchical solver as preconditioner vs. SuperLU-Dist direct solver on irregular problems from UF/SuiteSparse. Vary compression threshold epsilon. 16 processors (MPI ranks).



ParH2 Results:

- Ice sheet simulations of Antarctica
  - Modeled as Stokes' flow
  - Albany/Felix (now MALI) code

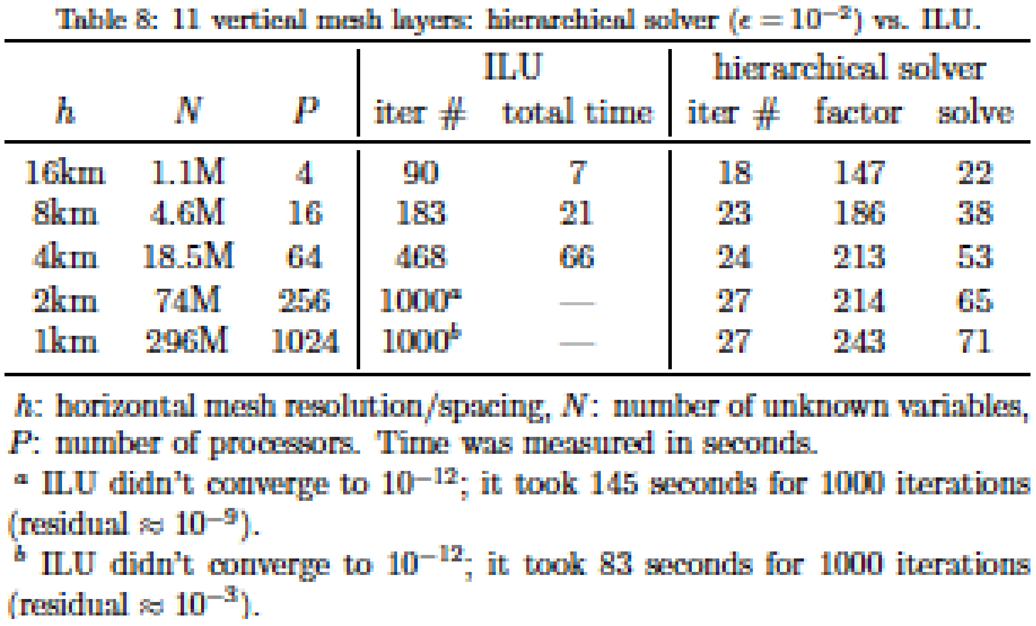


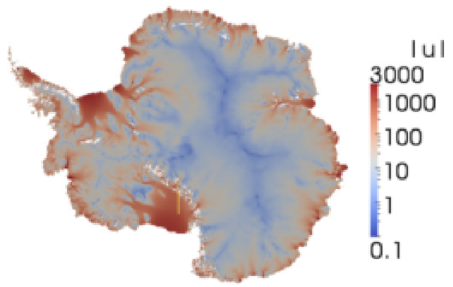
Table 8: 11 vertical mesh layers: hierarchical solver ($\epsilon = 10^{-2}$) vs. ILU.

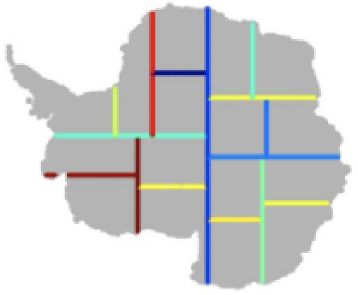
			ILU		hierarchical solver		
h	N	P	iter #	total time	iter #	factor	solve
16km	1.1M	4	90	7	18	147	22
8km	4.6M	16	183	21	23	186	38
4km	18.5M	64	468	66	24	213	53
2km	74M	256	1000[a]	—	27	214	65
1km	296M	1024	1000[b]	—	27	243	71

h: horizontal mesh resolution/spacing, N: number of unknown variables, P: number of processors. Time was measured in seconds.

[a] ILU didn't converge to 10^{-12}; it took 145 seconds for 1000 iterations (residual $\approx 10^{-9}$).

[b] ILU didn't converge to 10^{-12}; it took 83 seconds for 1000 iterations (residual $\approx 10^{-3}$).





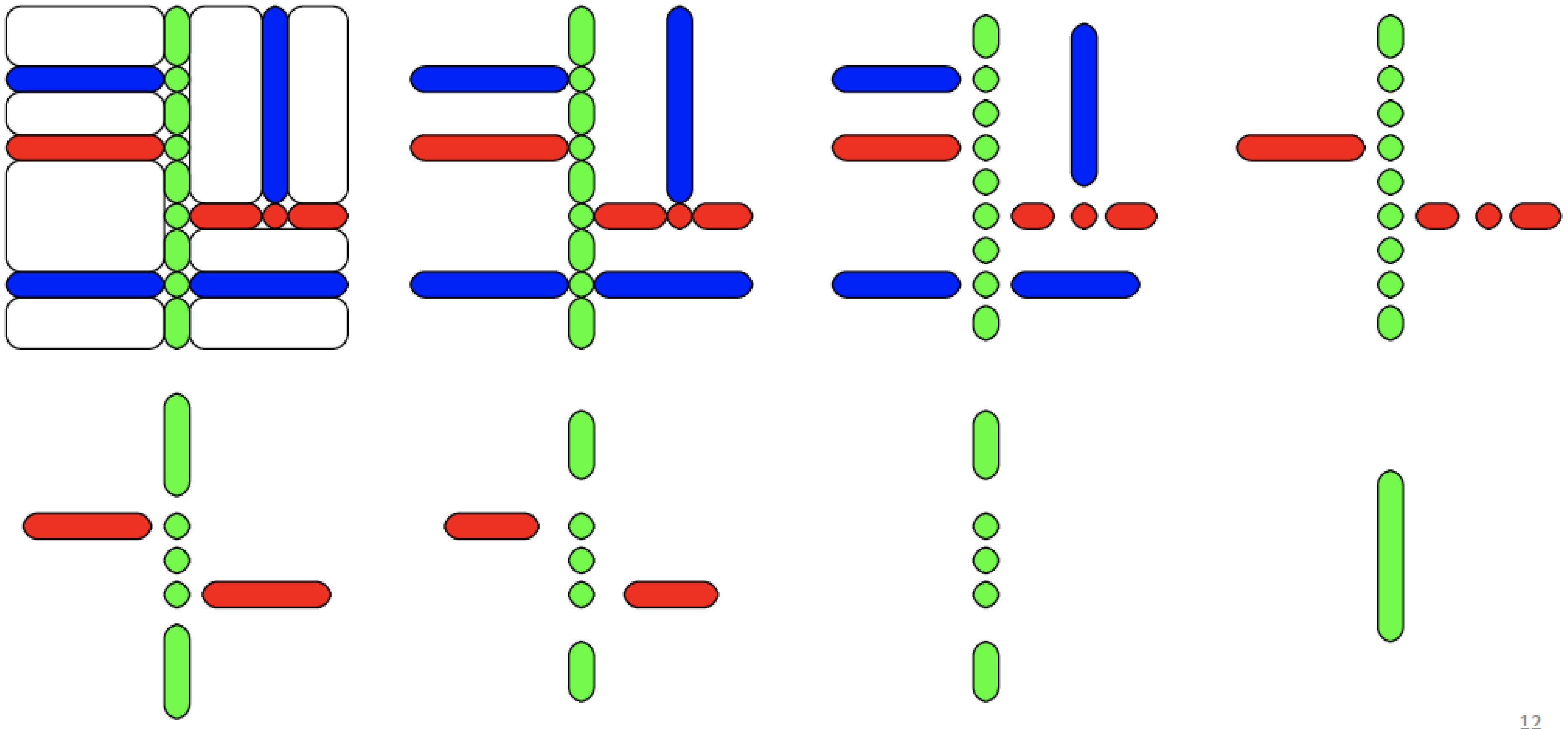
Images by Mauro Perego

SpaND overview

- SpaND is better sparse solver (Cambier et al, SIMAX 2020)
- Motivated by HIF (Ho, Ying 2016)
  - Several algorithmic improvements
  - Fully algebraic, supports unstructured problems
- Key ideas:
  - Use pattern of nested dissection sparse direct solvers
  - Compress on the fly, never form large dense frontal matrices
- Both serial and parallel codes available

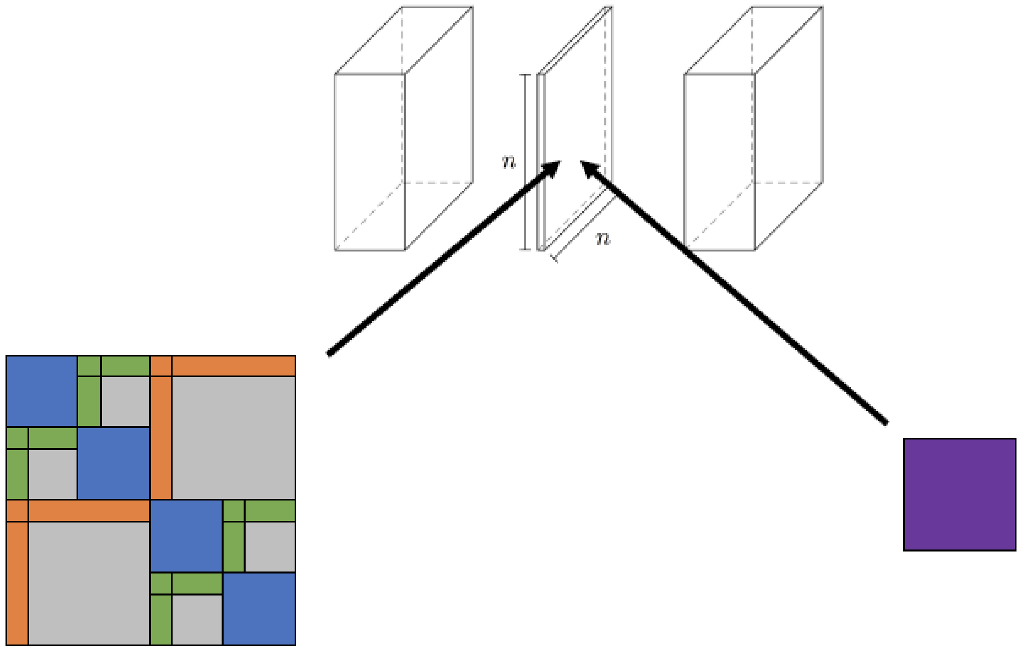
SpaND Summary

- Sparsify separators (low-rank compression) during elimination



Different from fast-algebra on dense

- Common approach: Fast algebra (H/HSS/BLR) on dense blocks
  - Ex: Strumpack, MUMPS, PasTix, etc.
- Instead we reduce the size of the separator blocks!



Sparsification via Low-rank Approx.

We need low-rank approximation of off-diagonal (rectangular) block.

1. Interpolative decomposition (ID)
   - Computed via RRQR (QRCP)
   - A.k.a. skeletonization
2. Orthogonal transform
   - Use RRQR or SVD
   - More stable, but may be more expensive

- For both methods there is a user parameter ε
  - Trade-off accuracy vs cost

Sparsification Step

- Block scaling, low-rank elimination, drop near-zero blocks

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

Block Cholesky improves stability

$$\underbrace{\begin{bmatrix} Q^{\top} & & \\ & I & \\ & & I \end{bmatrix} \begin{bmatrix} I & & A_{sn} \\ & A_{ww} & A_{wn} \\ A_{ns} & A_{nw} & A_{nn} \end{bmatrix} \begin{bmatrix} Q & & \\ & I & \\ & & I \end{bmatrix}}$$

Prefer orthogonal Q from RRQR

$$\begin{bmatrix} I & & & \varepsilon \\ & I & & W_{cn} \\ & & A_{ww} & A_{wn} \\ \varepsilon & W_{cn}^{\top} & A_{nw} & A_{nn} \end{bmatrix}$$

$\varepsilon \cong 0$,
So we drop it

Sparsification: Orthogonal version

(1) We start with

$$\begin{bmatrix} I & & A_{sn} \\ & A_{ww} & A_{wn} \\ A_{ns} & A_{nw} & A_{nn} \end{bmatrix}$$

(2) We then approximate

$$A_{sn} = Q_{sc} W_{cn} + \varepsilon$$
$$Q^\top s = f \cup c$$

(3) We end up with

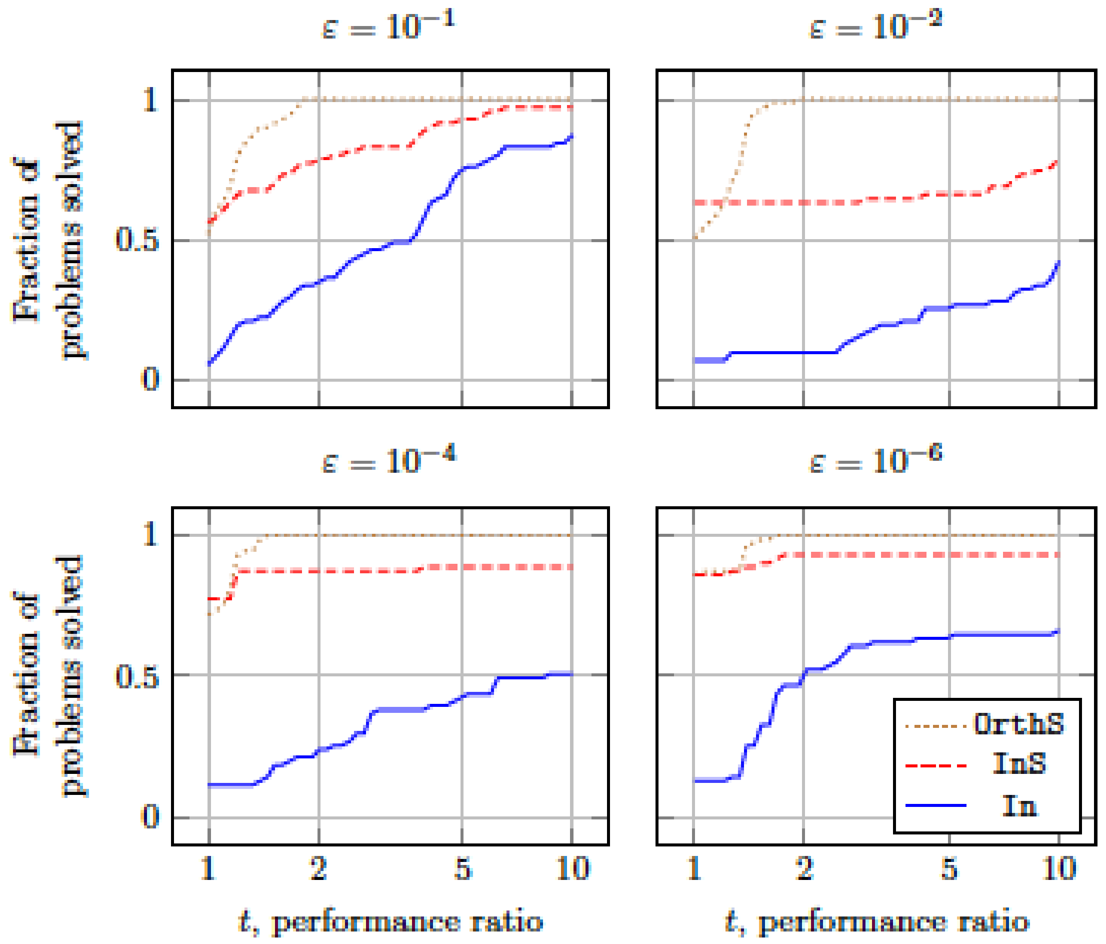
$$\begin{bmatrix} I & & & \varepsilon \\ & I & & W_{cn} \\ & & A_{ww} & A_{wn} \\ \varepsilon & W_{cn}^\top & A_{nw} & A_{nn} \end{bmatrix}$$

$$\begin{tikzcd}
f \arrow[loop left, "I"] \arrow[rr, leftrightarrow, bend right, "\mathcal{O}(\varepsilon)"'] & c \arrow[loop above, "I"] \arrow[r, leftrightarrow, "W_{cn}"] & n \arrow[loop above, "A_{nn}"] \arrow[r, leftrightarrow, "A_{nw}"] & w \arrow[loop right, "A_{ww}"]
\end{tikzcd}$$

6

Results: Performance Profile

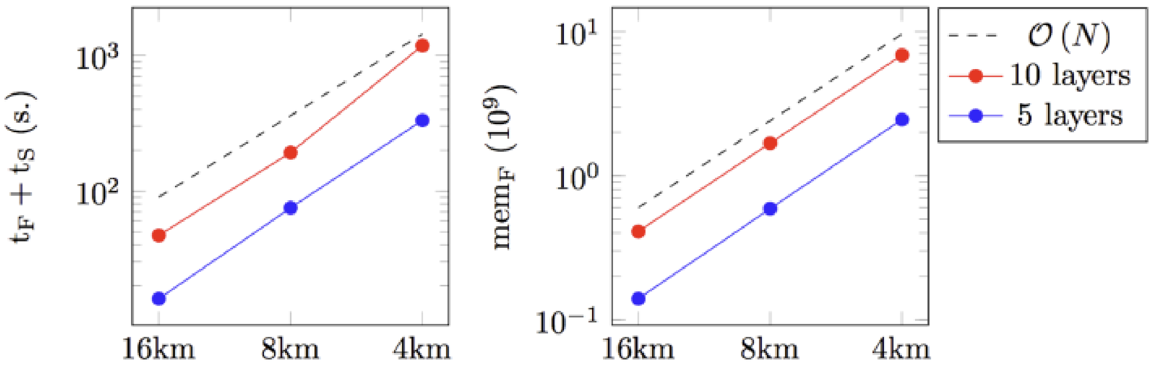
- Ran most SPD matrices in SuiteSparse collection
- OrthS version is very robust
  - Not sensitive to ε

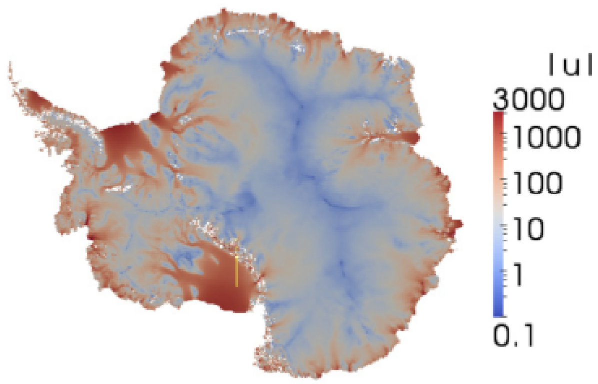


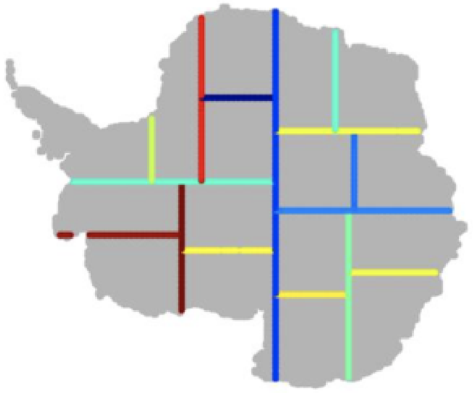
SpaND Results:

Ice-Sheet modeling $\kappa(A) > 10^{11}$

N	spaND t_F (s.)	t_S (s.)	n_{CG}	size_{Top}	mem_F (10^9)	Direct $t_F + t_S$ (s.)
5 layers						
629 544 (16 km)	13	3	7	76	0.14	22
2 521 872 (8 km)	55	20	8	89	0.59	206
10 096 080 (4 km)	217	115	10	100	2.45	1578
10 layers						
1 154 164 (16 km)	39	8	7	136	0.41	90
4 623 432 (8 km)	148	44	8	148	1.68	710
18 509 480 (4 km)	798	384	10	159	6.86	—



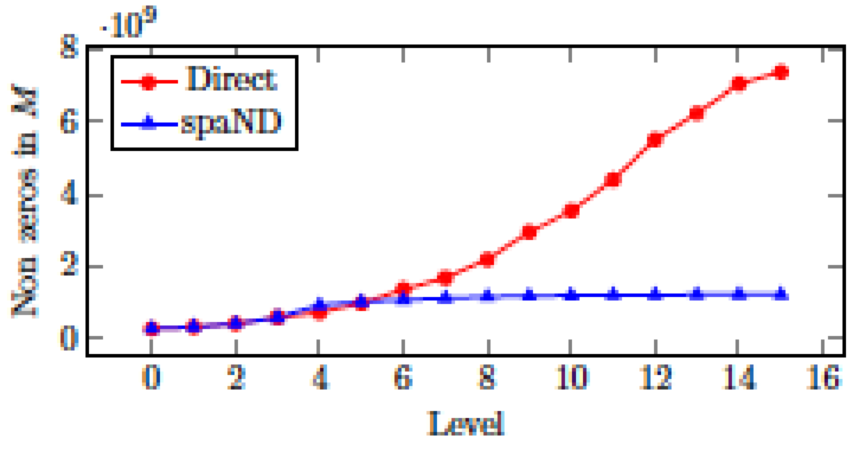


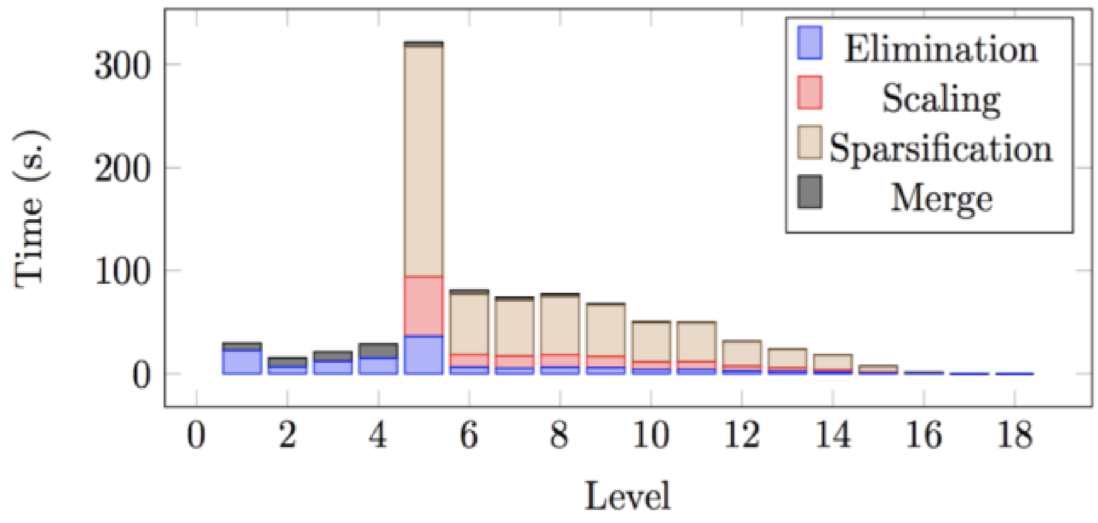


21

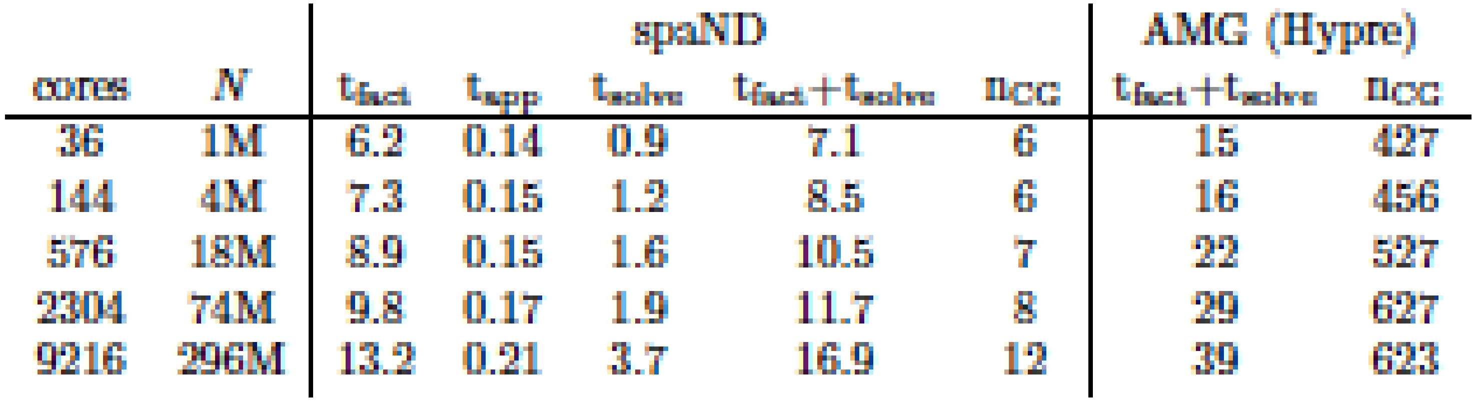
Profiling

- SpaND cost stays roughly constant per level
- Most expensive part is sparsification (RRQR)
- Skip sparsification on bottom levels
  - No benefit first levels
  - Pay the cost first time we sparsify





SpaND Parallel Results:



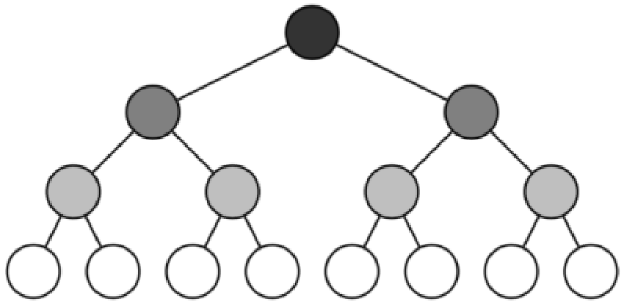
		spaND					AMG (Hypre)	
cores	N	t_{fact}	t_{app}	t_{solve}	$t_{fact}+t_{solve}$	n_{CG}	$t_{fact}+t_{solve}$	n_{CG}
36	1M	6.2	0.14	0.9	7.1	6	15	427
144	4M	7.3	0.15	1.2	8.5	6	16	456
576	18M	8.9	0.15	1.6	10.5	7	22	527
2304	74M	9.8	0.17	1.9	11.7	8	29	627
9216	296M	13.2	0.21	3.7	16.9	12	39	623

Table 4.1: Ice-sheet results, weak scalings, from 36 to 9216 cores.

- Parallel version by Cambier & Darve
  - Uses TaskTorrent task-parallel system
- Good weak scaling (not perfect)
- Faster than Hypre AMG on this problem
  - Default parameters for Hypre

Work in Progress

- Improved SpaND algorithm
  - Second order accurate version (Klockiewicz, upcoming talk)
- Mixed precision
  - Low-rank approximations can use lower precision (float32)
  - Expect largest savings in the factorization/setup phase
- GPU support
  - Dense linear algebra works well on GPU
  - Level-by-level, or task based execution?
  - May need batched BLAS/LAPACK



Conclusions

- Hierarchical solvers have shown great promise
  - Near-linear scaling on many PDE problems
  - Robust, often converges where other methods fail
  - Diversity of algorithms, no clear ”winner”
    - SpaND is very competitive IMHO

- Still mostly academic, why not popular in applications?
  - Focus has been on papers, not software
    - Software lags multigrid and domain decomposition
  - Algorithms are complicated
    - Difficult to implement efficiently
  - Active research area, algorithms change rapidly

References

- *SpaND: An Algebraic Sparsified Nested Dissection Algorithm Using Low-Rank Approximations*, L. Cambier, C. Chen, E.G. Boman, S. Rajamanickam, R.S. Tuminaro, E. Darve, SIMAX 41(2), 2020
- *A robust hierarchical solver for ill-conditioned systems with applications to ice sheet modeling*, C. Chen, L. Cambier, E.G. Boman, S. Rajamanickam, R.S. Tuminaro, E. Darve, JCP, v.396, 2019
- *A distributed-memory hierarchical solver for general sparse linear systems,* C. Chen, H. Pouransari, S. Rajamanickam, E.G. Boman, E. Darve, Parallel Computing, v.74, 2018

Backup

Sparsification 1: ID

(1) We start with

$$\begin{bmatrix} A_{ss} & & A_{sn} \\ & A_{ww} & A_{wn} \\ A_{ns} & A_{nw} & A_{nn} \end{bmatrix}$$

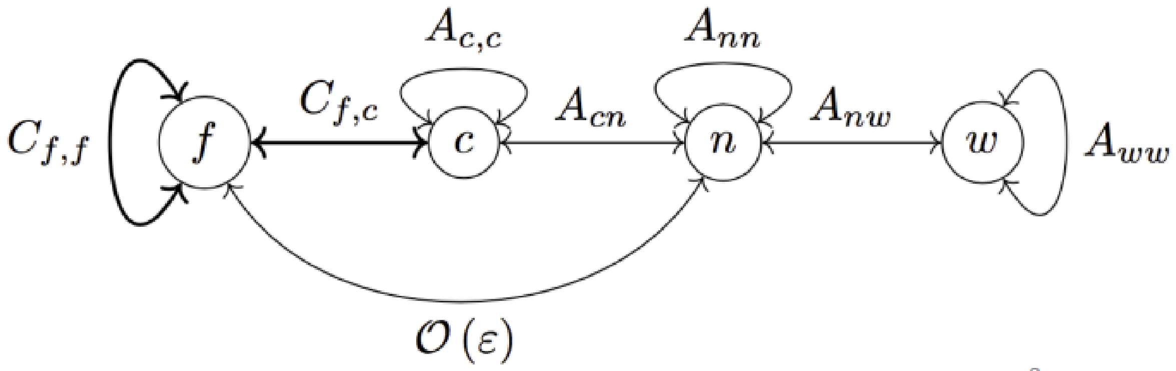
(2) We then approximate

$$A_{sn} = \begin{pmatrix} T_{fc} \\ I \end{pmatrix} A_{cn} + \varepsilon$$

$$s = f \cup c$$

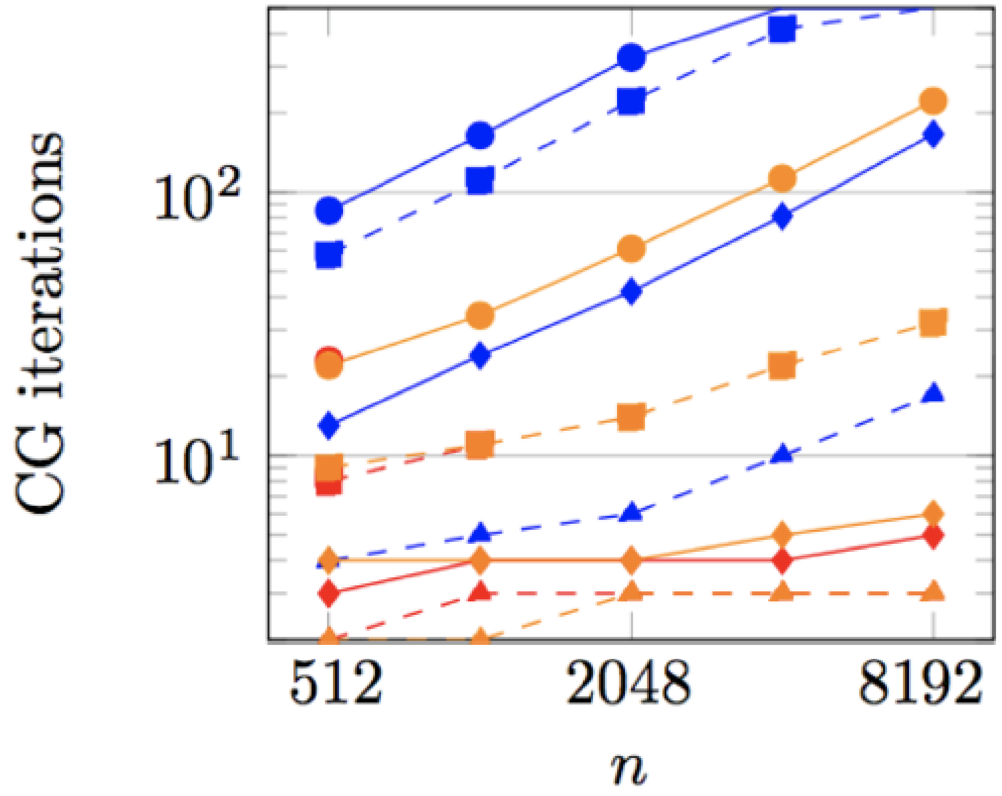
(3) We end up with

$$\begin{bmatrix} C_{ff} & C_{fc} & & \varepsilon \\ C_{cf} & A_{cc} & & A_{cn} \\ & & A_{ww} & A_{wn} \\ \varepsilon & A_{nc} & A_{nw} & A_{nn} \end{bmatrix}$$



8

Results: 2D Laplacians



Interpolative, no scaling

Interpolative, with scaling

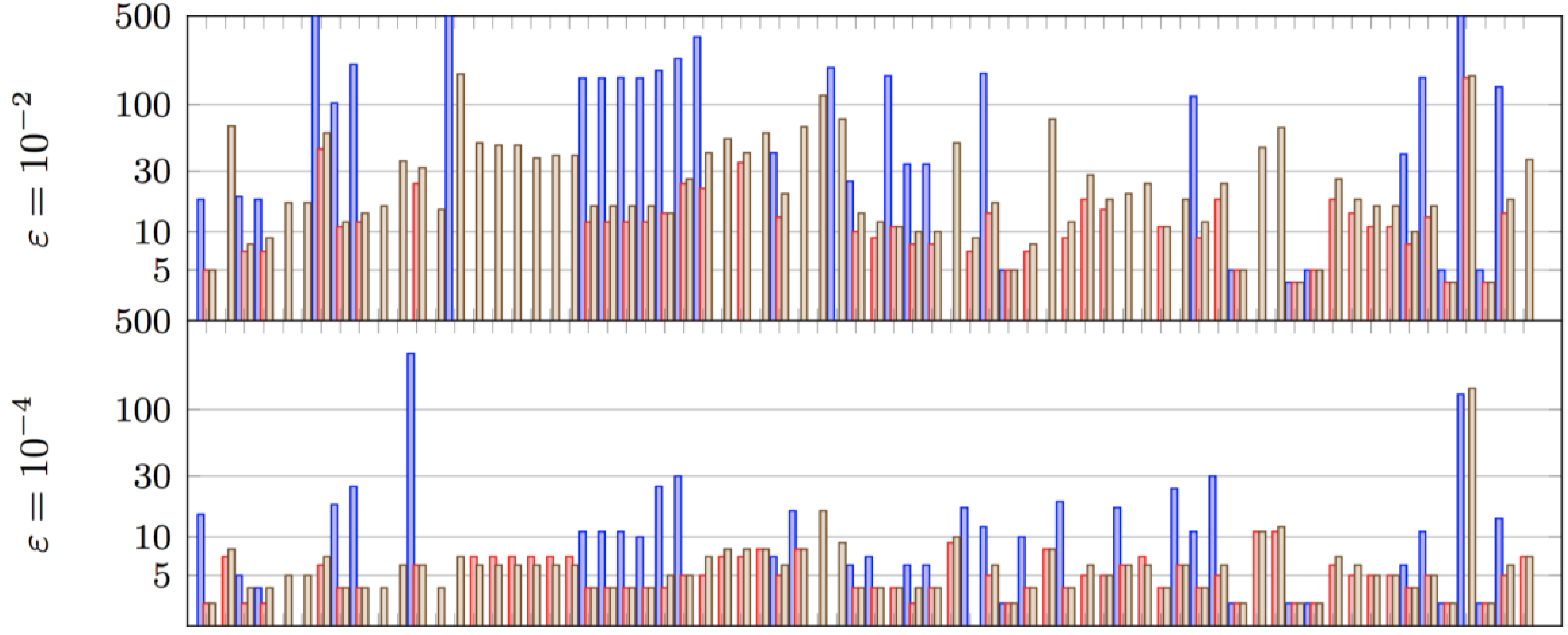
Orthogonal, with scaling

$$\varepsilon = 10^{-1} \to 10^{-6}$$

Results: SuiteSparse Collection

SPD problems from SuiteSparse

Interpolative, no scaling
Interpolative, with scaling
Orthogonal, with scaling

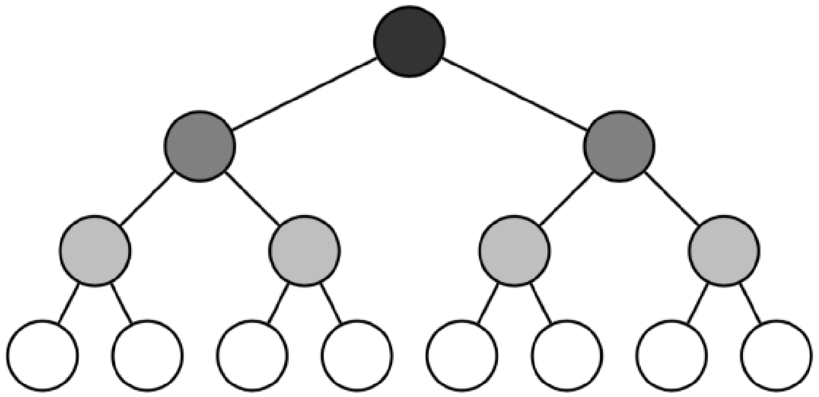


Parallel Approaches

We are exploring two approaches for parallel SpaND:

- Task-based
  - Dynamic scheduling of DAG on shared-memory systems
- Level-based
  - Process level-by-level, going up the tree
  - Need batched BLAS/LAPACK, many small operations in parallel
  - Use Kokkos library to run on both CPU and GPU

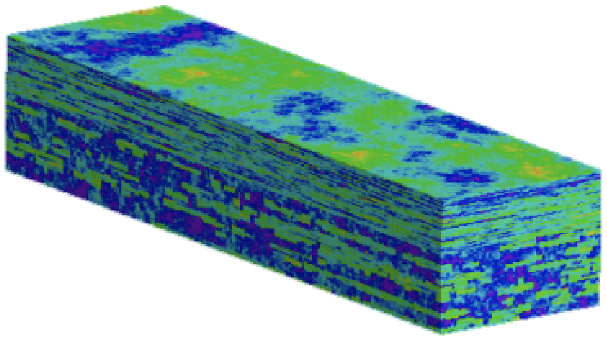
- This is work in progress.

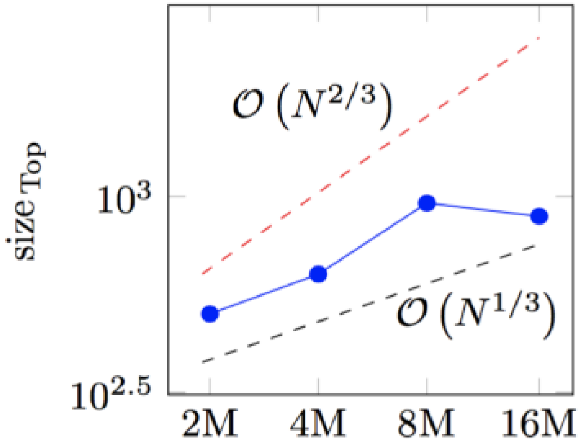


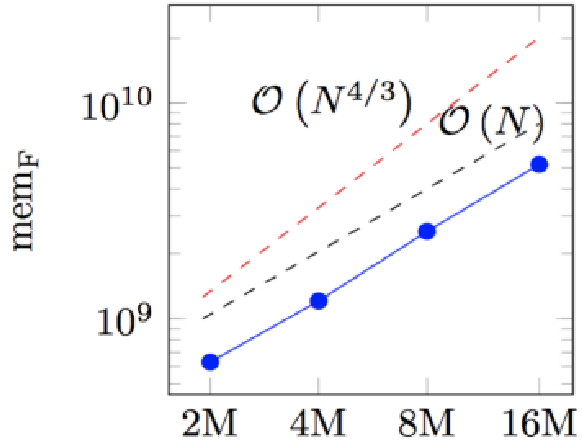
Results: SPE

The SPE problem

n	$N = n^3$	spaND $\mathrm{t_F}$ (s.)	$\mathrm{t_S}$ (s.)	$\mathrm{n_{CG}}$	$\mathrm{size_{Top}}$	$\mathrm{mem_F}$ (10^9)	Direct. $t_F + \mathrm{t_S}$ (s.)
128	2 097 152	61	23	12	502	0.63	686
160	4 096 000	175	46	13	634	1.21	—
200	8 000 000	287	158	16	962	2.54	—
252	16 003 008	963	369	16	890	5.19	—







Top separator block would be 32 GB without the sparsification!