Reducing communication, flops in sparse factorizations
Part II

X. Sherry Li
xsli@lbl.gov
Lawrence Berkeley National Laboratory

Woudschoten Conference, Oct. 3-5, 2018

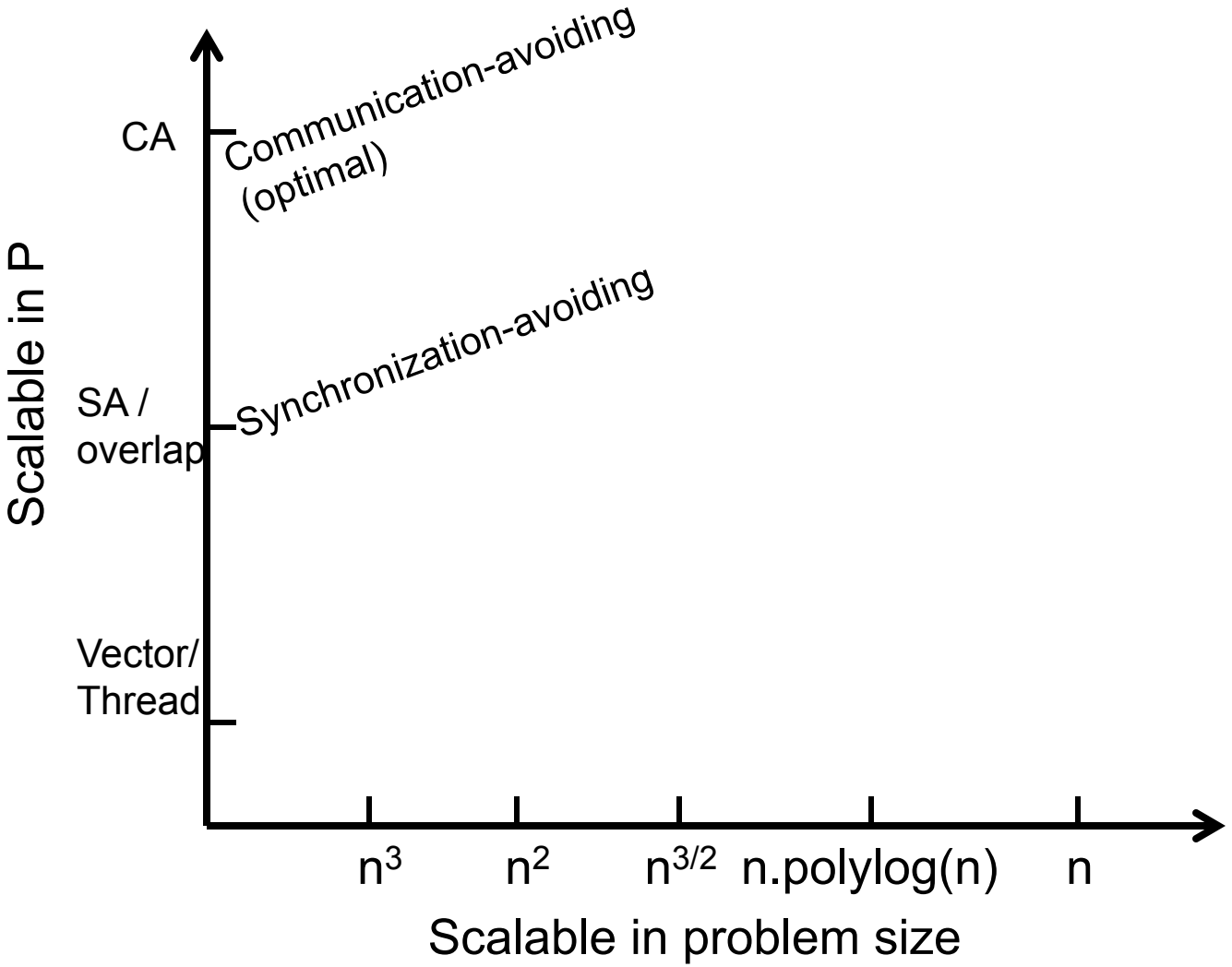
Collaborators

- Gustavo Chavez, LBNL
- Pieter Ghysels, LBNL
- Chris Gorman, UC Santa Barbara
- Yang Liu, LBNL
- Francois-Henry Rouet, LSTC
- Piyush Sao, Georgia Tech
- Rich Vuduc, Georgia Tech

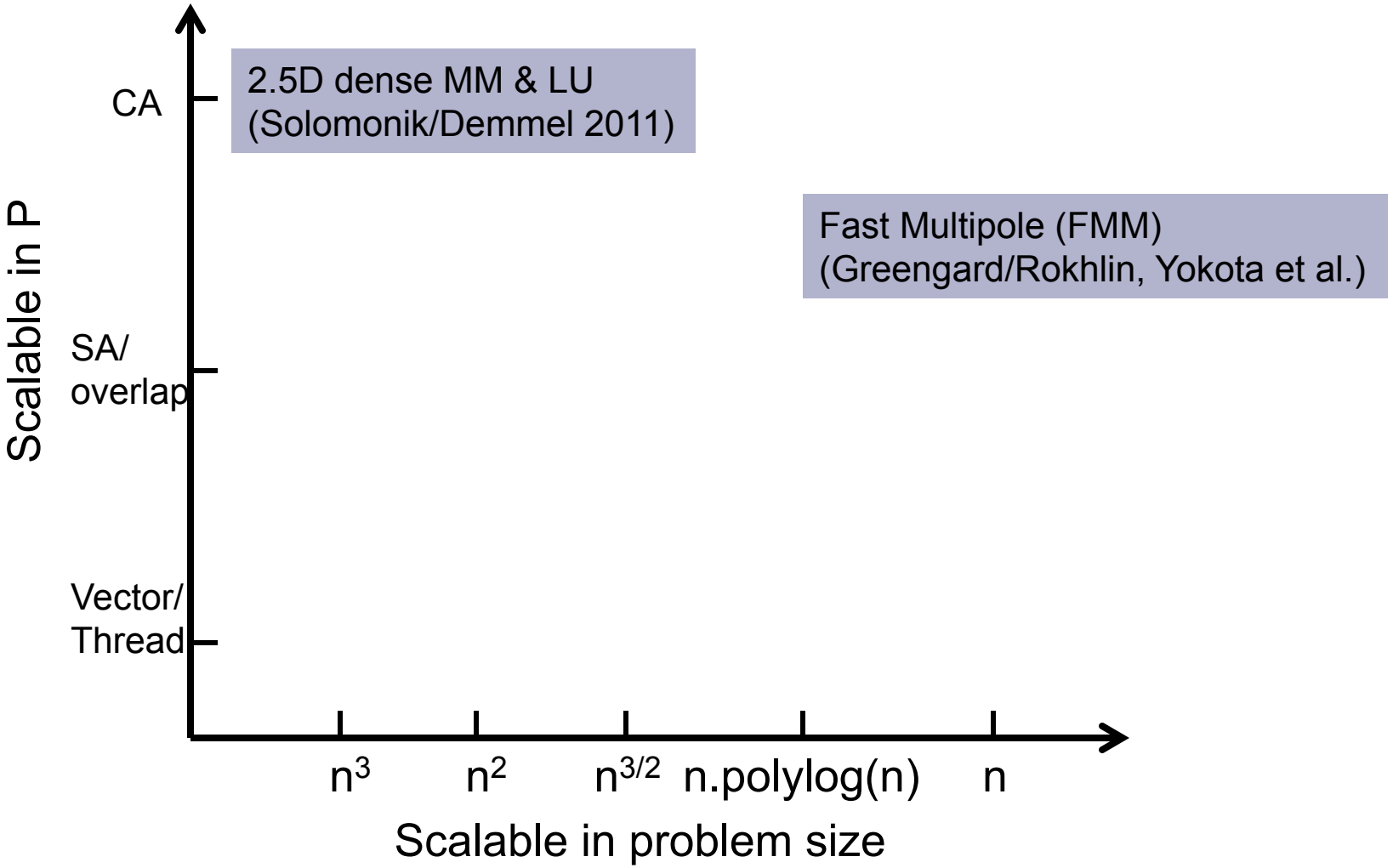
Goals of scalable algorithms

- Scalable w.r.t. problem size
  - Holy Grail: O(n) operations
- Scalable w.r.t. machine size
  - Holy Grail: perfect load balance, minimum communication
- Architecture-aware
  - Vectorization, multithreading, accelerators, ...

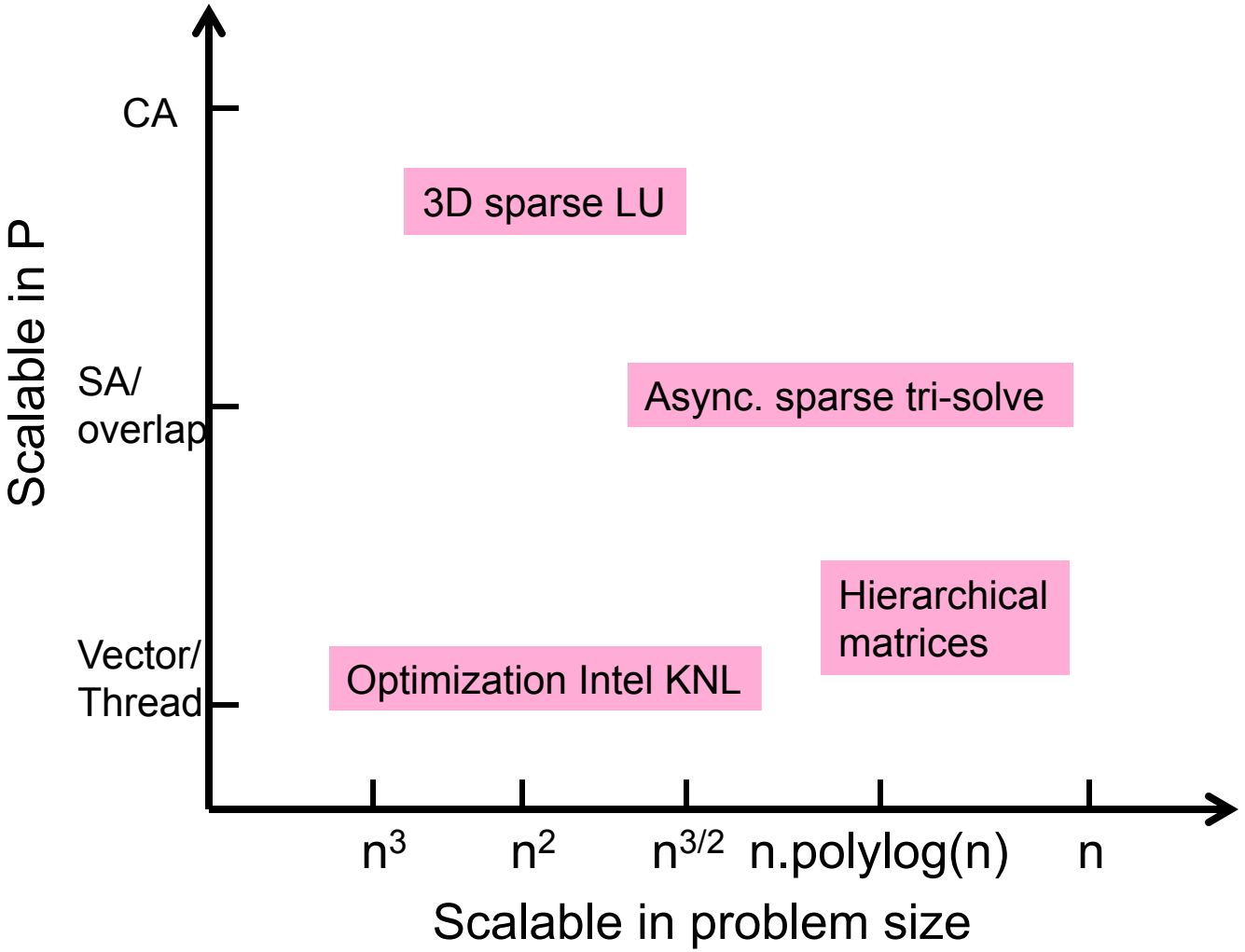
Algorithm scalability



Algorithm scalability



Future work


CA
SA/
overlap
Vector/
Thread
Scalable in P
3D sparse LU
Async. sparse tri-solve
Hierarchical
matrices
Optimization Intel KNL
n3
n2
n3/2
n.polylog(n)
n
Scalable in problem size

Latency-bandwidth model to analyze communication

- Time to send a message of length n is roughly

$$\begin{aligned} \text{Time} &= \text{latency} + n * \text{time\_per\_word} \\ &= \text{latency} + n / \text{bandwidth} \end{aligned}$$

Called "α−β model" and written as:

$$\text{Time} = \alpha + \beta \times n$$

- Usually α >> β >> time per flop

→Can do hundreds or thousands of flops with the time one message

- One long message is cheaper than many short ones

$$\alpha + n*\beta \ << \ n*(\alpha + 1*\beta)$$

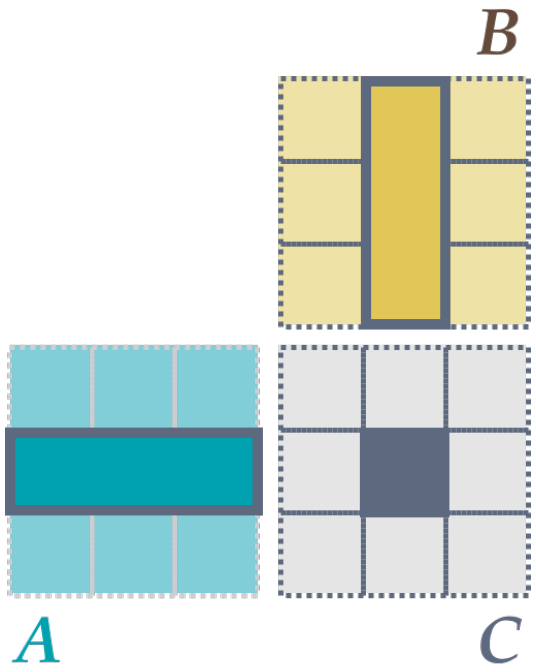
Communication-avoiding 3D sparse LU factorization

(P. Sao, XL, R. Vuduc, IPDPS'18)

(Sao's thesis, 2018, Georgia Tech)

Communication-avoiding idea

For matrix multiplication, $C \mathrel{+}= A \cdot B$*, on* P *processors*



$$\Longrightarrow$$

$\sqrt{P} \times \sqrt{P}$ process grid

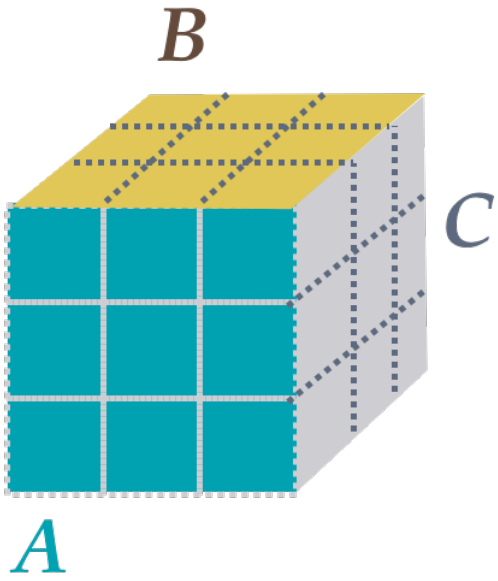
$\Theta\left(\frac{n^2}{P}\right)$ elems per proc

$\Omega\left(\sqrt{P} + \frac{n^2}{\sqrt{P}}\right)$ comm time

Attained by Cannon's algorithm (1969), for instance

Communication-avoiding idea

For matrix multiplication, $C \mathrel{+}= A \cdot B$*, on* P *processors*



$P^{\frac{1}{3}} \times P^{\frac{1}{3}} \times P^{\frac{1}{3}}$ process grid

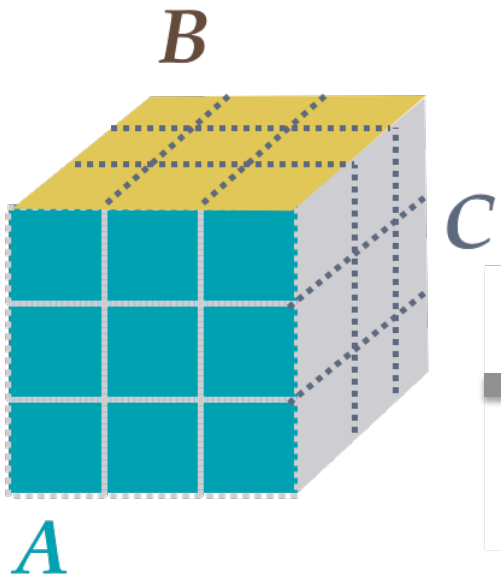
$\Theta\left(\frac{n^2}{P^{\frac{2}{3}}}\right)$ elems per proc (replicate)

Idea: Use a 3-D process grid and replicate

Dekel et al. (1981); Agarwal et al. (1995); + more

Communication-avoiding idea

For matrix multiplication, $C \mathrel{+}= A \cdot B$*, on* P *processors*



$P^{\frac{1}{3}} \times P^{\frac{1}{3}} \times P^{\frac{1}{3}}$ process grid

$\Theta\left(\frac{n^2}{P^{\frac{2}{3}}}\right)$ elems per proc (replicate)

Memory$^{(3D)}$ = Memory$^{(2D)}$ x $P^{1/3}$

CommTime$^{(3D)}$ ≈ CommTime$^{(2D)}$ / $P^{1/6}$

Trades extra memory for less communication

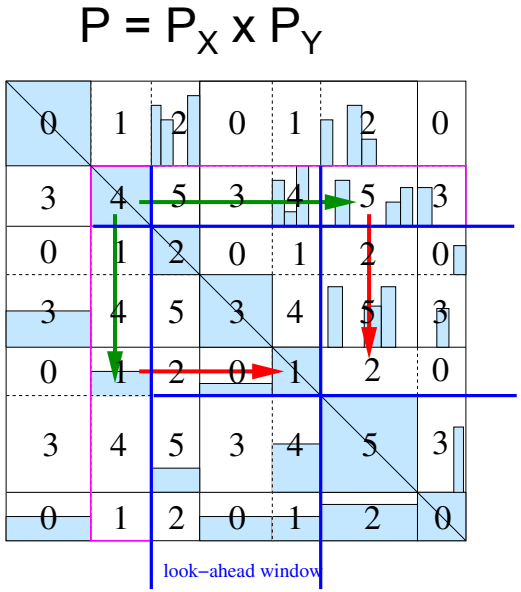
Overview of 3D algorithms

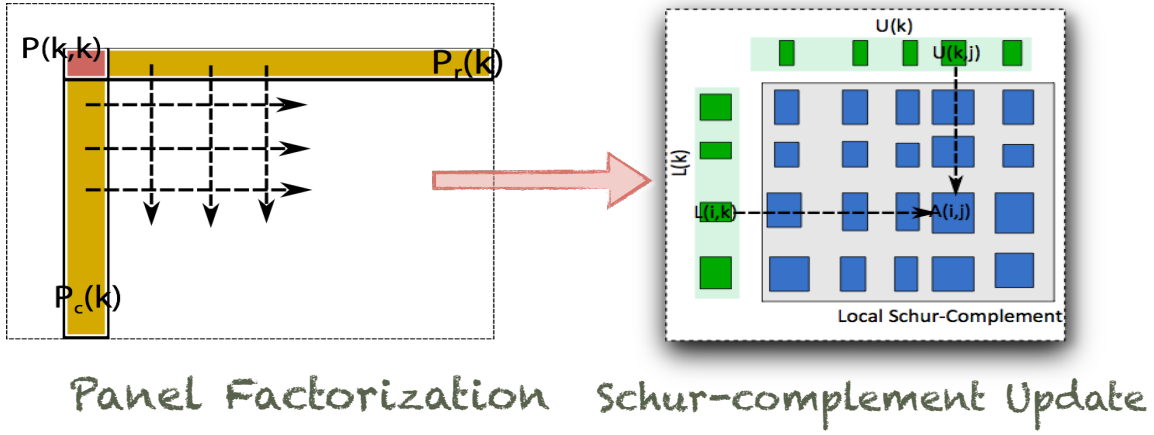
- All known "3D-LU" algorithms† are for **dense LU**. They reduce communication volume but increase latency.
- For sparse LU, we can reduce **both the latency and bandwidth** for "planar" problems **asymptotically**, and achieve constant-factor reduction for "non-planar" ones.
- Note: **multifrontal method** is among other memory-for-communication techniques.‡

† Ashcraft (1991); Irony & Toledo (2002); Solomonik & Demmel (2011)
‡ Hulbert & Zmijewski (1991); Gupta et al. (1997)

Review of 2D sparse factorization in SuperLU_DIST

XL, J. Demmel, J. Gilbert, L. Grigori, Y. Liu, P. Sao, Meiyue Shao, I. Yamazaki



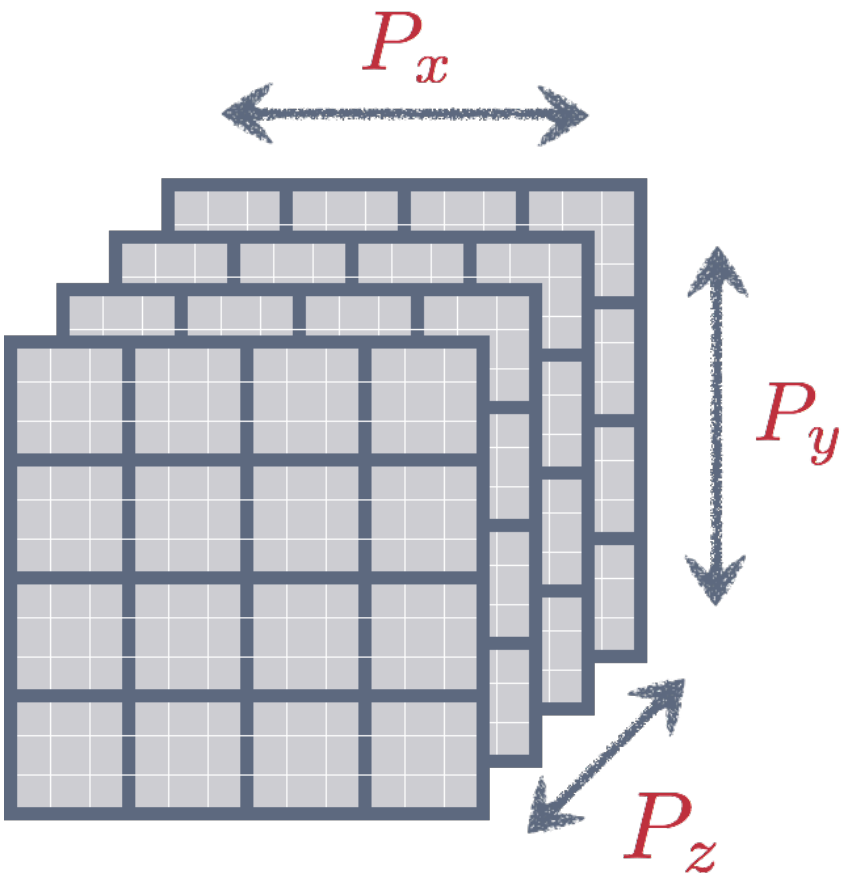


- Graph at step k+1 differs from step k
- Panel factorization on critical path

Limitations of 2D algorithm:

- Sequential Schur-complement update
- Fixed latency cost: O(n)

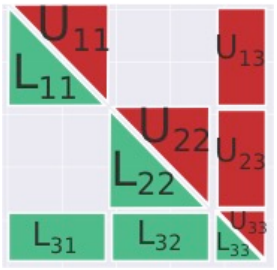
3D process grid

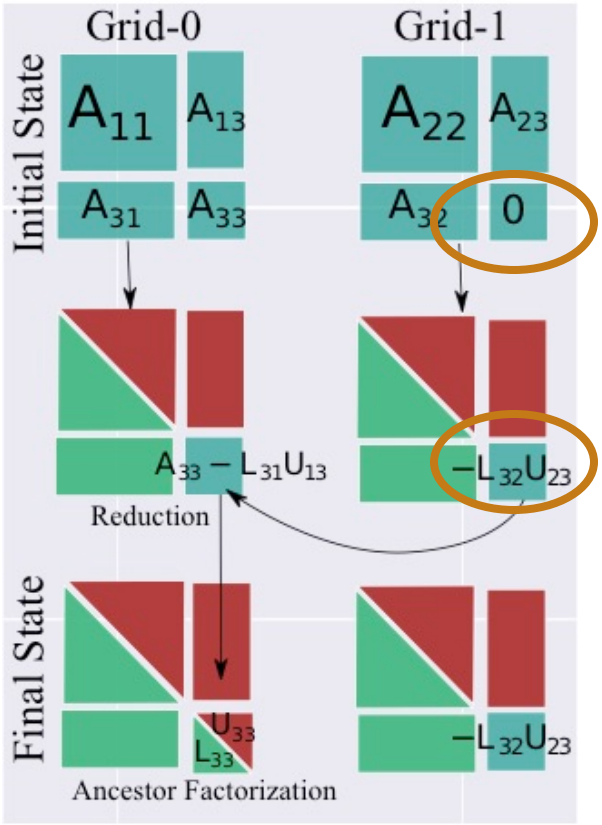


3D sparse factorization

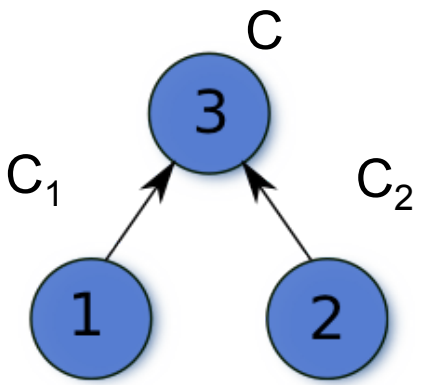
- Replicate Schur-complement on multiple 2D process grids
- P = P_{XY} x P_Z : P_Z slices of P_{XY} 2D process grids
- Trade extra memory for less communication, more parallelism.
  - Each Schur-complement block is split-updated by multiple processes
  - Reduction of Schur-complement of common ancestor C = C_1 + C_2

Nested Dissection



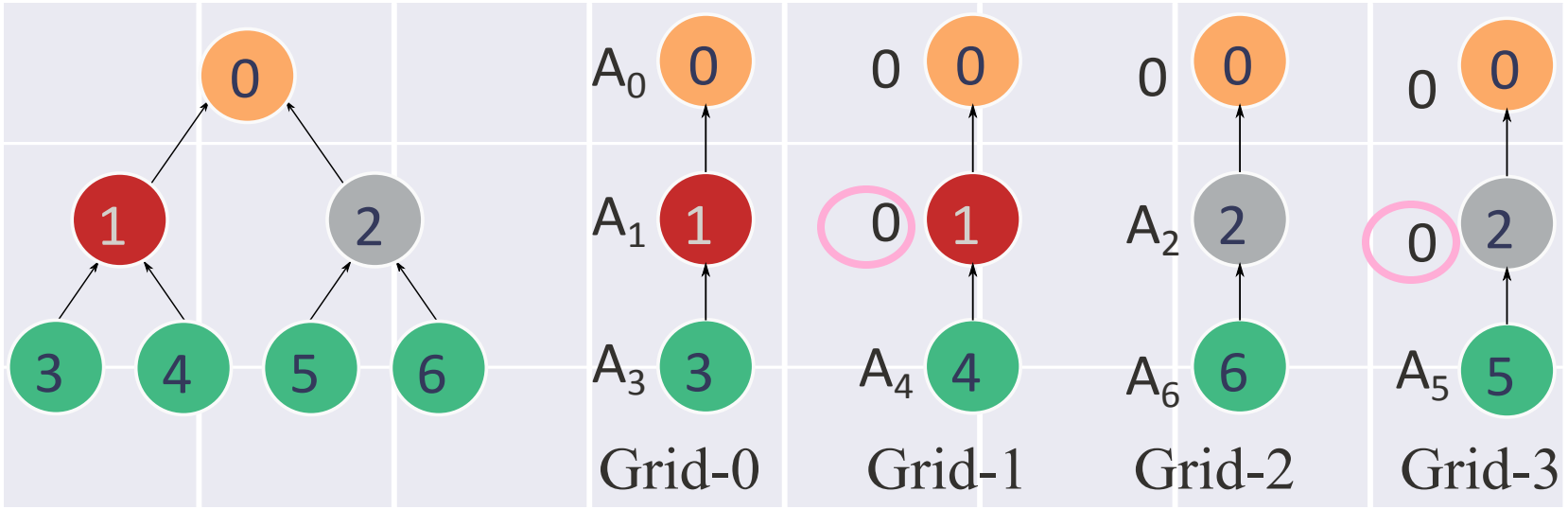


P_Z = 2
two slices of 2D grids



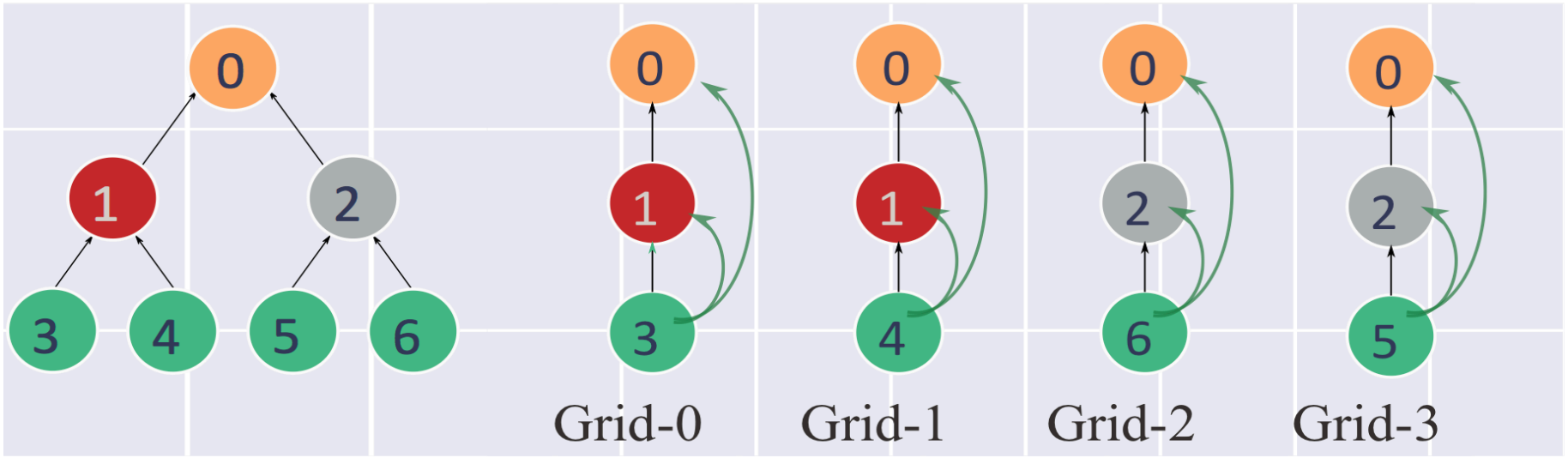
3D sparse factorization: $P_z = 4$

4 slices of 2D grids, top 2 levels use replication



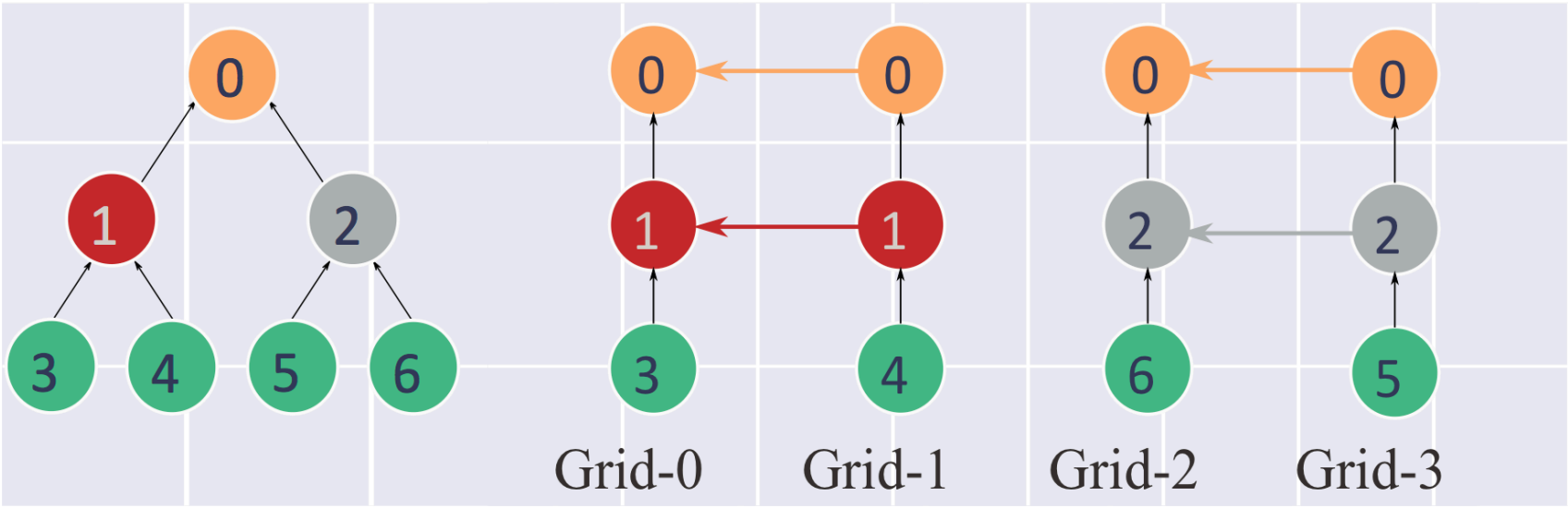
- Node 0 replicated on all process Grids
- Node 1 replicated on process Grids 0, 1
- Node 2 replicated on process Grids 2, 3

3D sparse factorization: $P_z = 4$



Factorize leaf subtrees & do (partial) Schur updates on ancestral copies

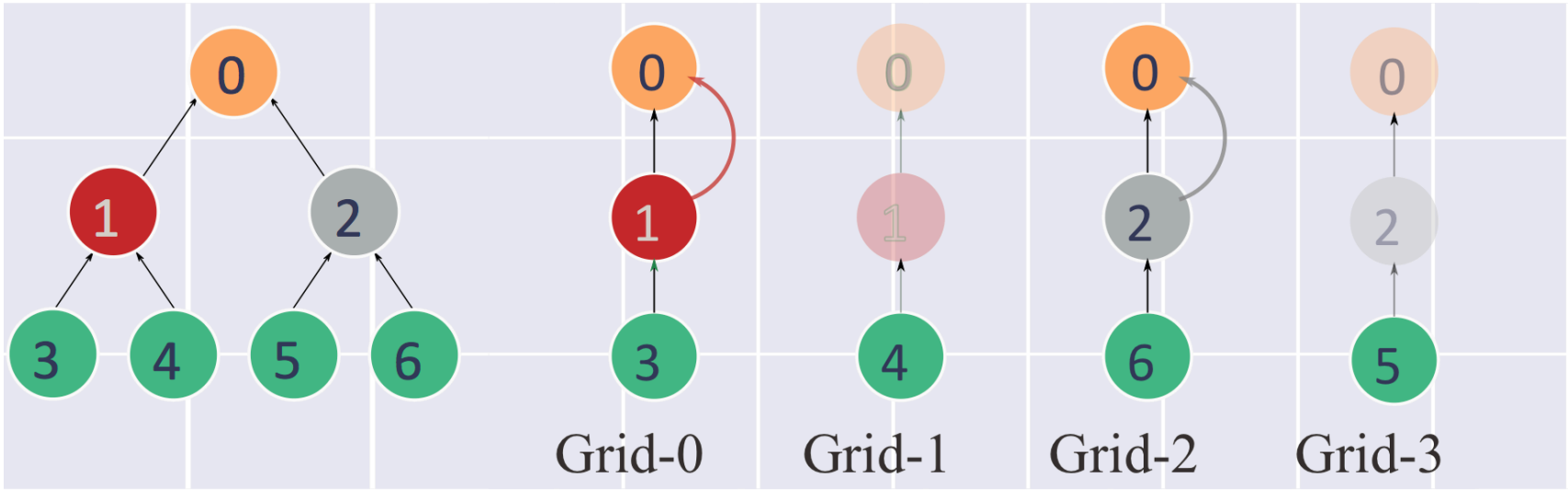
3D sparse factorization: $P_z = 4$



Ancestors reduce:

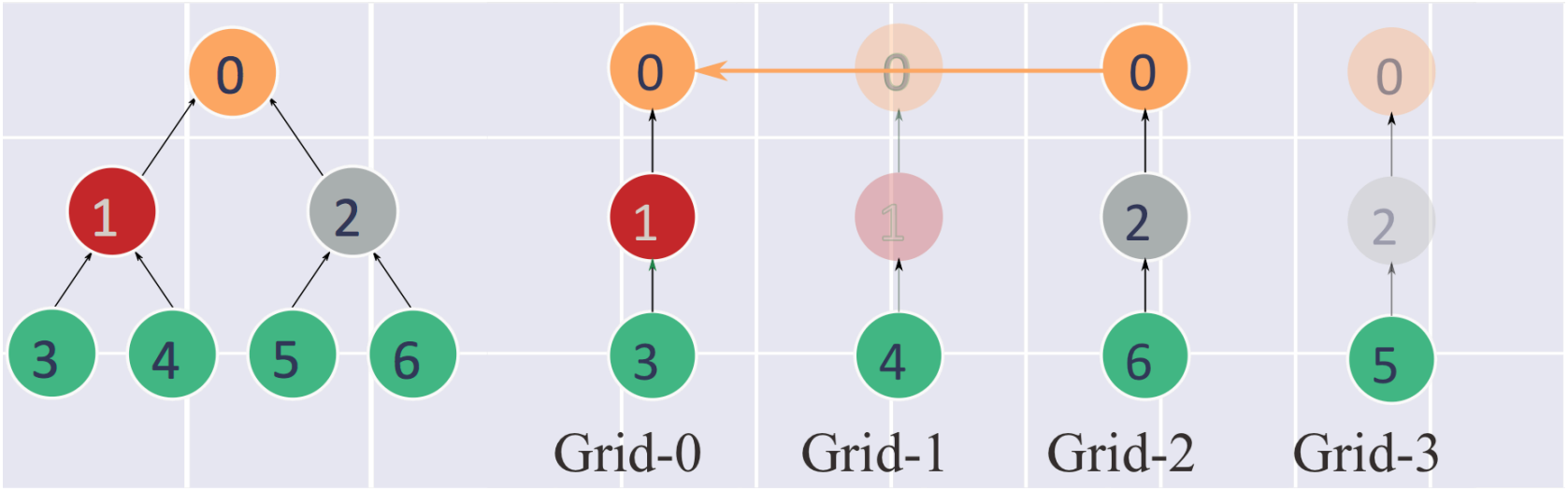
Grids 0-1 reduce to Grid 0
Grids 2-3 reduce to Grid 2

3D sparse factorization: $P_z = 4$



Repeat: Factorization and (partial) Schur update

$P_z = 4$



Reduce

Restriction:
***P_z* = power of two**

Communication analysis (along critical path)

Not possible to analyze general sparse matrices.

Possible for model problems:

- **“Planar”** geometries, i.e., PDE on a 2D domain
- **“Non-planar”** case, i.e., PDE on a 3D domain

Planar graphs have good separators

Top-level separator size

$$\mathcal{O}(\sqrt{n})$$

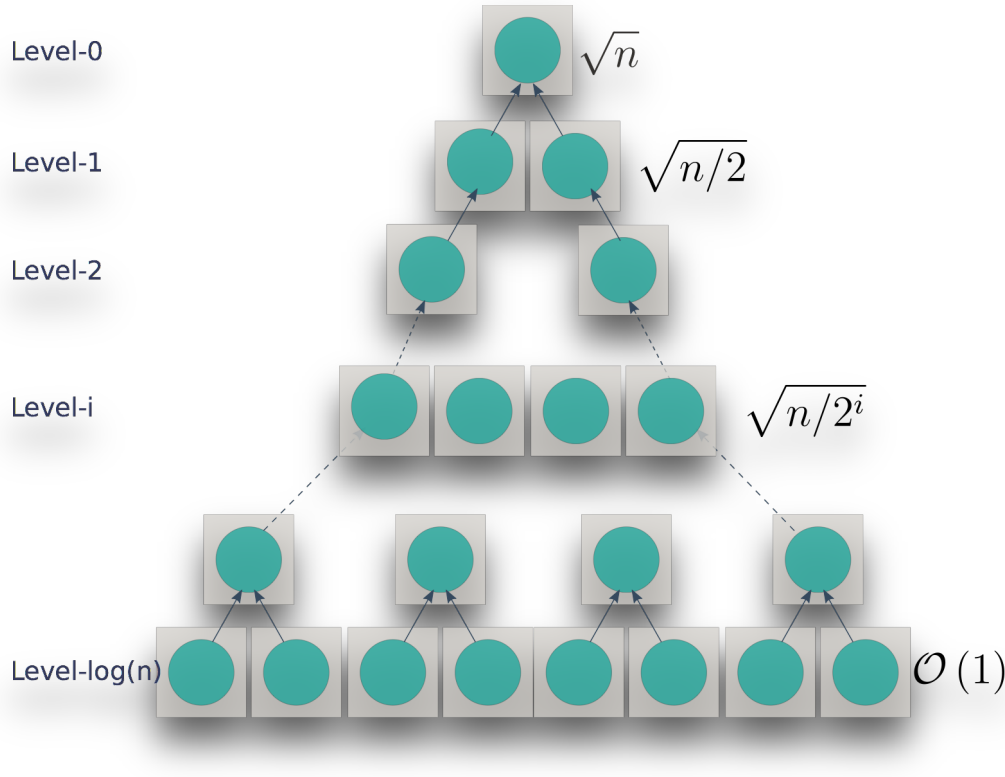
Memory of top-level separator

$$\mathcal{O}(n)$$

Total size of $L + U$

$$\mathcal{O}(n \log n)$$

“Planar” geometry



E.g. George (1978), Lipton & Tarjan (1979)

Memory / process

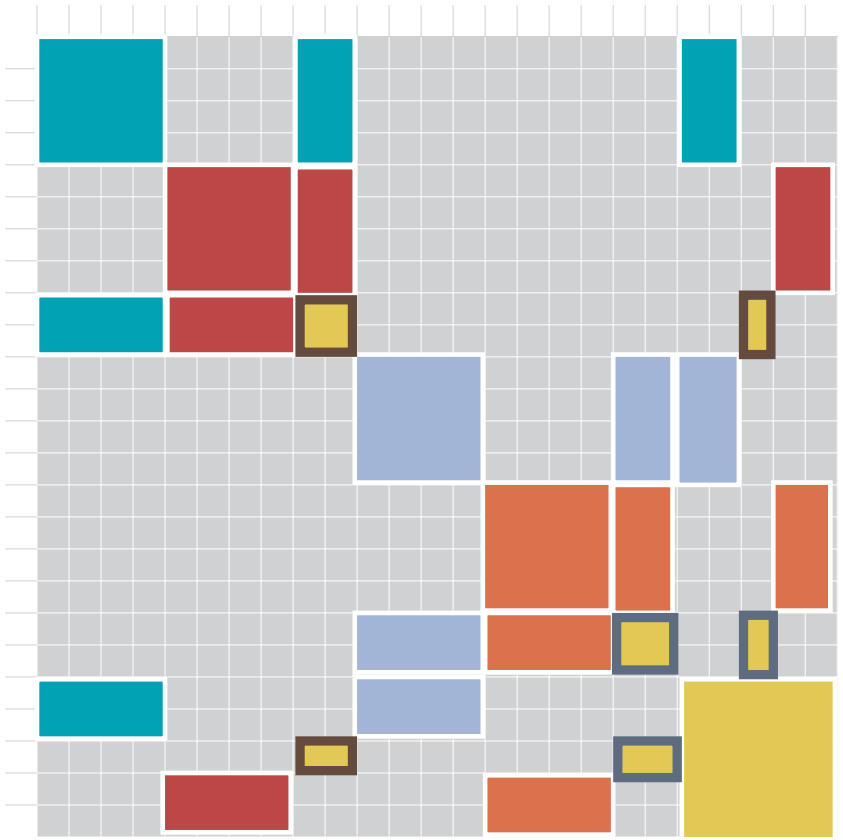
$$\frac{n \log n}{P}$$

Communication volume / process

$$\approx M\sqrt{P} = \frac{n \log n}{\sqrt{P}}$$

Latency (msgs) / process

$$n$$



“Planar” geometry + 2D algorithm (all “big-O”)

“Planar” geometry + 3D algorithm

Memory / process

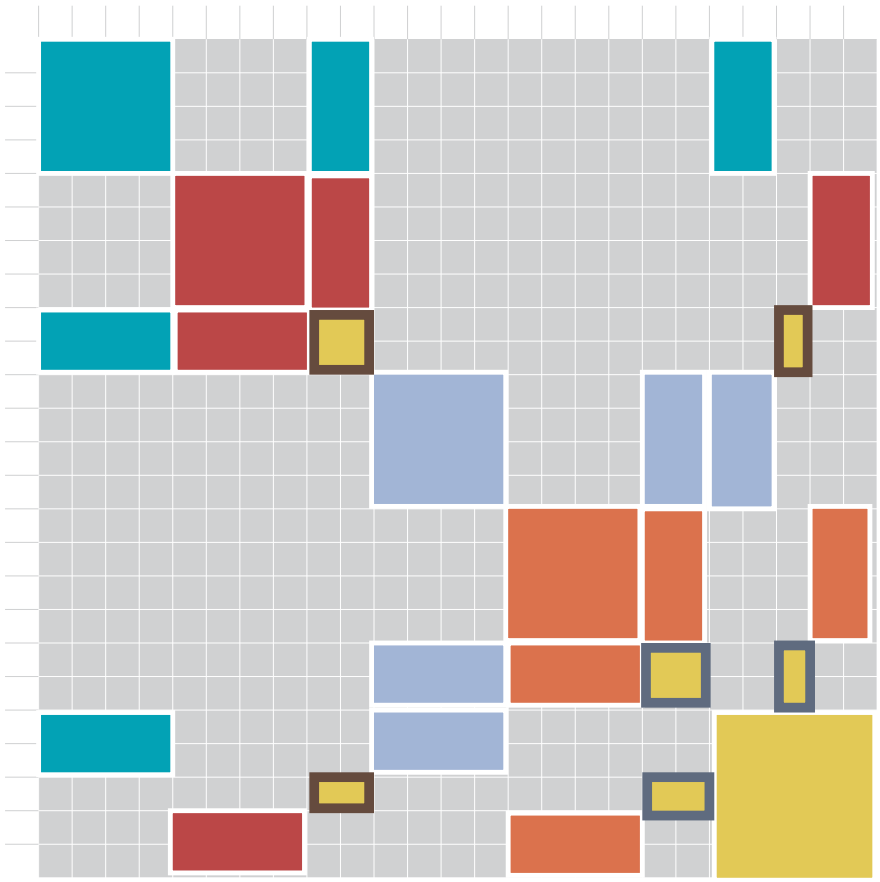
??

Communication volume / process

??

Latency (msgs) / process

??



“Planar” geometry + 3D algorithm?

Memory / process

$$\mathrm{M}^{(2D)} \cdot \left(1 + \frac{P_z}{\log n}\right)$$

Communication volume / process

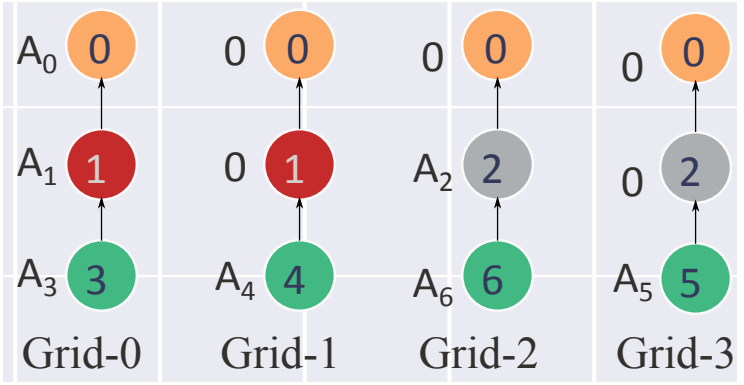
??

Latency (msgs) / process

??

“Planar” geometry + 3D algorithm?

Replication



Memory / process

$$M^{(2D)} \cdot \left(1 + \frac{P_z}{\log n}\right)$$

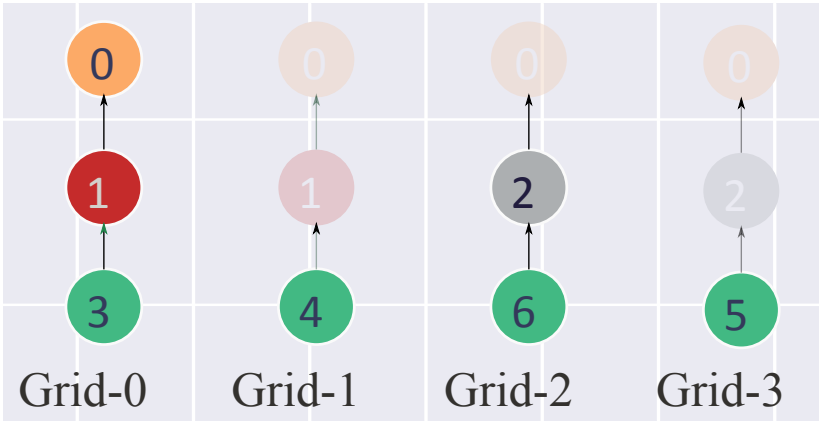
Communication volume / process

$$W^{(2D)} \cdot \left(\frac{1}{\sqrt{P_z}} + \frac{\sqrt{P_z}}{\log n}\right)$$

Latency (msgs) / process

??

Smaller subproblems lower in the tree…



“Planar” geometry + 3D algorithm?

Memory / process

$$M^{(2D)} \cdot \left(1 + \frac{P_z}{\log n}\right)$$

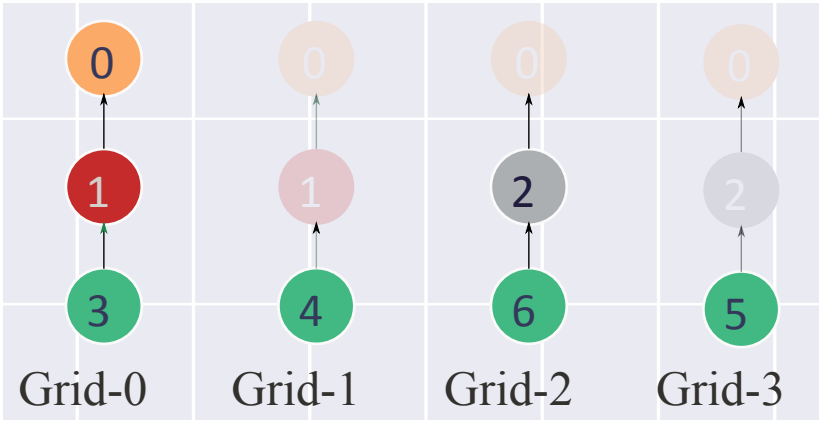
Communication volume / process

$$W^{(2D)} \cdot \left(\frac{1}{\sqrt{P_z}} + \frac{\sqrt{P_z}}{\log n}\right)$$

Latency (msgs) / process

??

… but fewer participants toward the top.



“Planar” geometry + 3D algorithm?

Memory / process

$$\mathrm{M}^{(2D)} \cdot \left(1 + \frac{P_z}{\log n}\right)$$

Communication volume / process

$$\mathrm{W}^{(2D)} \cdot \left(\frac{1}{\sqrt{P_z}} + \frac{\sqrt{P_z}}{\log n}\right) \Longrightarrow \frac{1}{\sqrt{\log n}} \text{ reduction}$$

Optimal tradeoff!

$$P_z = \Theta(\log n)$$

Latency (msgs) / process

??

“Planar” geometry + 3D algorithm?

Memory / process

$$M^{(2D)} \cdot \left(1 + \frac{P_z}{\log n}\right)$$

⇒ **Only O(1) increase!**

Communication volume / process

$$W^{(2D)} \cdot \left(\frac{1}{\sqrt{P_z}} + \frac{\sqrt{P_z}}{\log n}\right) \Longrightarrow \frac{1}{\sqrt{\log n}} \text{ reduction}$$

Optimal tradeoff!

$$P_z = \Theta(\log n)$$

Latency (msgs) / process

??

“Planar” geometry + 3D algorithm?

Memory / process

$$\mathrm{M}^{(2D)} \cdot \left(1 + \frac{P_z}{\log n}\right)$$

Communication volume / process

$$\mathrm{W}^{(2D)} \cdot \left(\frac{1}{\sqrt{P_z}} + \frac{\sqrt{P_z}}{\log n}\right)$$

Latency (msgs) / process

$$\frac{\mathrm{L}^{(2D)}}{P_z}$$

Optimal tradeoff!

$\frac{1}{\log n}$ reduction

“Planar” geometry + 3D algorithm?

“Non-planar” geometry + 3D algorithm: O(1) improvement

Top-level separator size

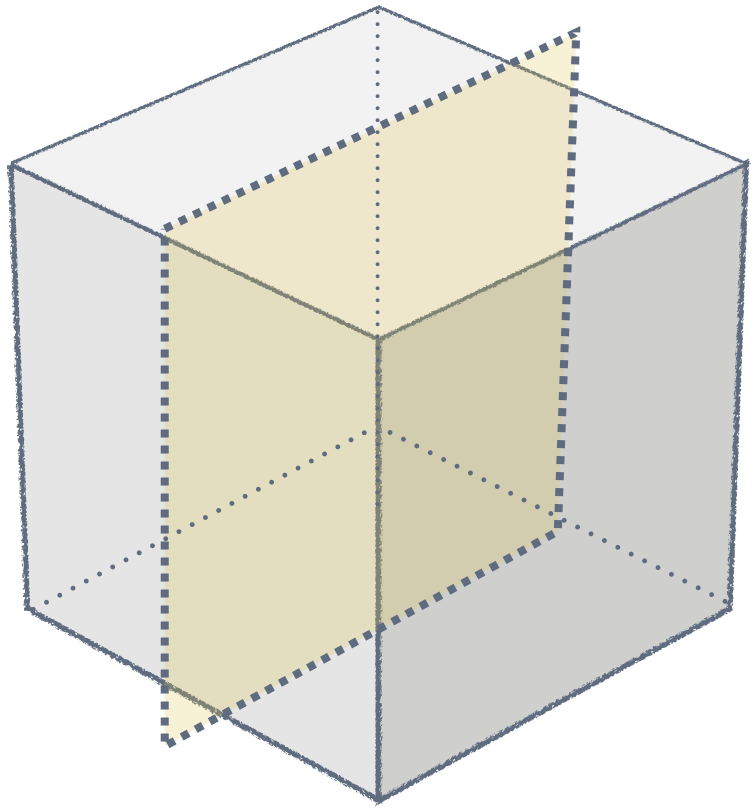
$$\mathcal{O}\left(n^{\frac{2}{3}}\right)$$

Memory of top-level separator

$$\mathcal{O}\left(n^{\frac{4}{3}}\right)$$

Total size of $L + U$

$$\mathcal{O}\left(n^{\frac{4}{3}}\right)$$



“Non-planar” (3D) geometry

“Non-planar” geometry + 3D algorithm

Memory / process

$$\mathrm{M}^{(2D)} \cdot P_z$$

⇒ **O(1) more memory**

Communication volume / process

$$\mathrm{W}^{(2D)} \cdot \mathcal{O}\left(\frac{1}{{P_z}^{\frac{4}{3}}} + \sqrt{P_z}\right)$$

⇒ **O(1) improvement (only)**

Latency (msgs) / process

$$\mathrm{L}^{(2D)} \cdot \mathcal{O}\left(\frac{1}{{P_z}^{\frac{2}{3}}} + \frac{1}{n^{\frac{1}{3}}}\right)$$

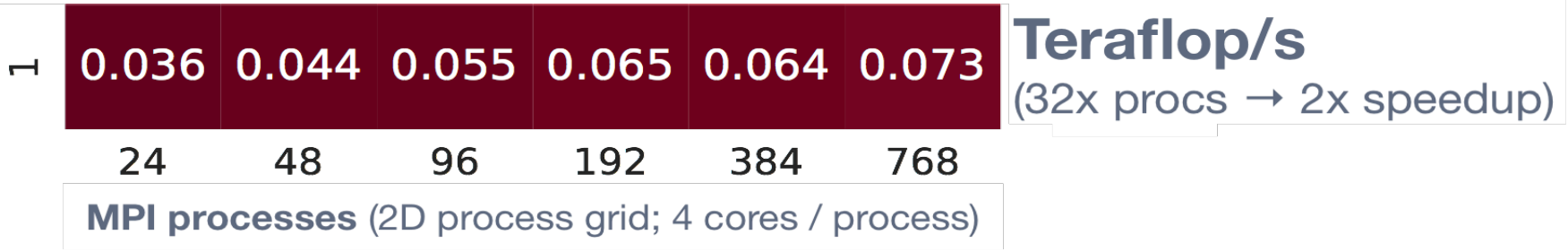
⇒ **O(1) improvement (only)**

“Non-planar” (3D) geometry + 3D algorithm

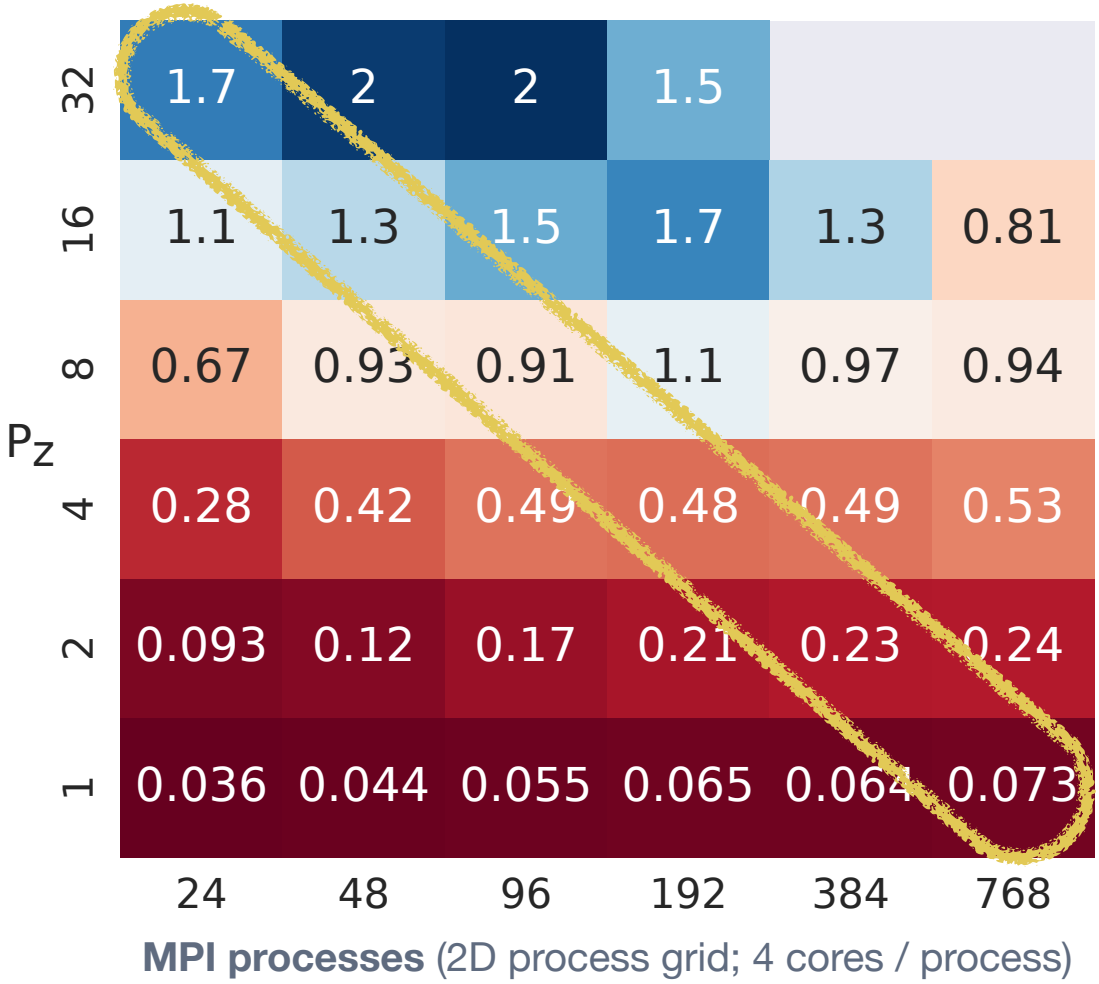
Actual performance on Cray XC30 (edison @ NERSC)

Example: K2D5pt4096

2D algorithm (strong scaling)



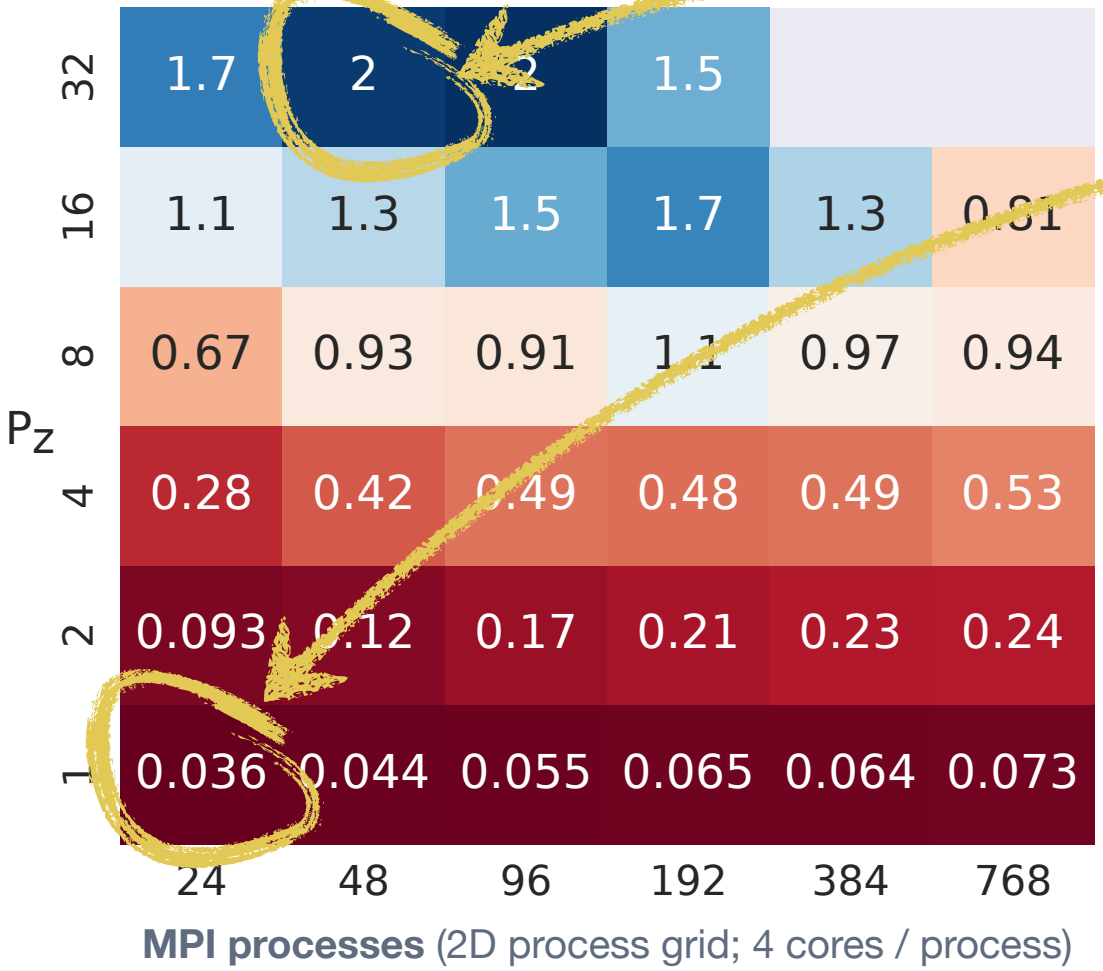
hpcgarage.org/hookem



2D to 3D:
→ 23x speedup

Teraflop/s
(32x procs → 2x speedup)

hpcgarage.org/hookem



64x procs
→ 55x speedup

Teraflop/s
(32x procs → 2x speedup)

Summary of 3D sparse LU

- Planar graph: asymptotically lower communication
  - Reduce communication volume by a factor of O(sqrt(log n))
  - Reuce latency by a factor of O(log n)

  - How far from optimal ?

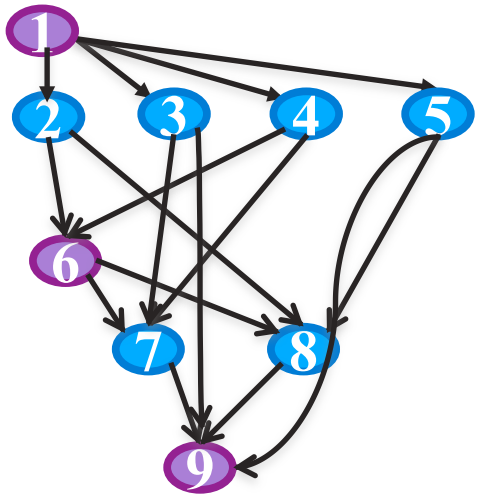
- Non-planar graph: constant-factor improvement
  - Top separator dimension $n^{2/3}$ dominates entire LU factor size $n^{4/3}$

- Strong scale to 24,000 cores. Compared to 2D algorithm:
  - Planar: up to 27x faster, 30% more memory @ $P_z = 16$
  - Non-planar: up to 3.3x faster, 2x more memory @ $P_z = 16$

About extra memory ...

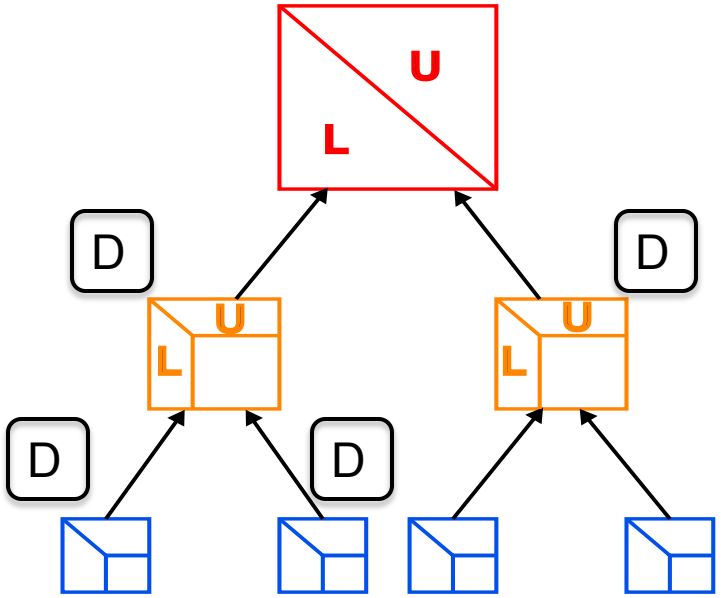
... no more than multifrontal method

DAG based Supernodal: SuperLU



$S^{(j)} \leftarrow (S^{(j)} - D^{(k1)}) - D^{(k2)}) - \ldots$

Tree based Multifrontal:
STRUMPACK, MUMPS



$S^{(j)} \leftarrow S^{(j)} - ((D^{(k1)}) + D^{(k2)}) + \ldots)$

Approximate factorization with quasi-linear arithmetic complexity

(P. Ghysels, G. Chavez, C. Gorman, Y. Liu, L.)

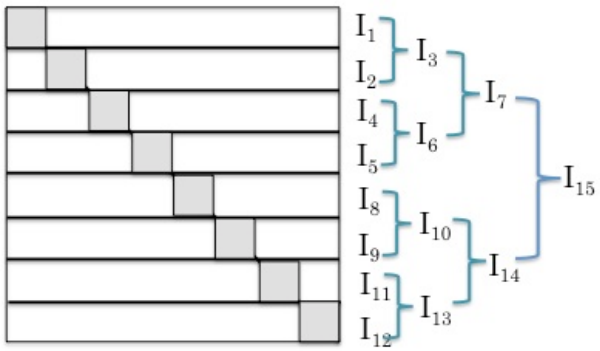
Software: STRUMPACK
http://portal.nersc.gov/project/sparse/strumpack

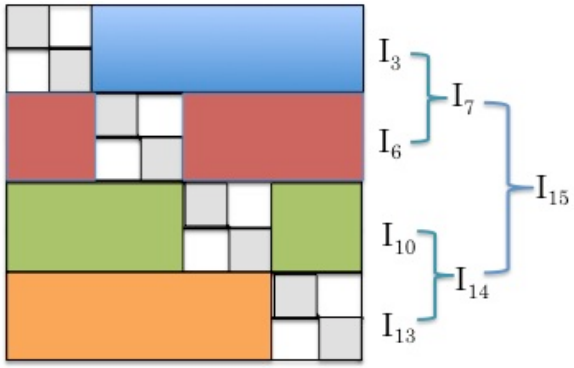
Hierarchical matrix approximation

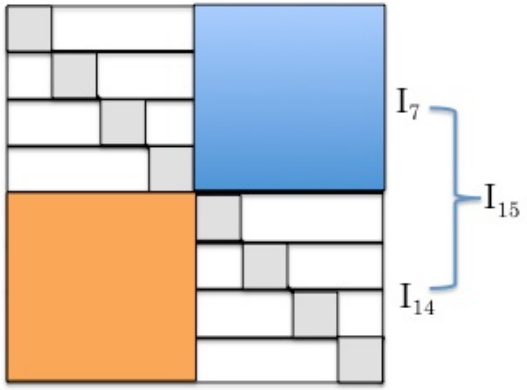
- **Dense:** exploit data-sparsity with low-rank compression.
- Same mathematical foundation as Fast Multipole (FMM, Greengard-Rokhlin'87), but in matrix form:
  - Diagonal block **("near field")** exact
  - off-diagonal block **("far field")** approximated via low-rank compression
- Applications: BEM methods, integral equations, machine learning, and structured matrices such as Toeplitz, Cauchy matrices.

$$A \approx \begin{bmatrix} \begin{bmatrix} D_1 & U_1 B_1 V_2^T \\ U_2 B_2 V_1^T & D_2 \end{bmatrix} & U_3 B_3 V_6^T \\ U_6 B_6 V_3^T & \begin{bmatrix} D_4 & U_4 B_4 V_5^T \\ U_5 B_5 V_4^T & D_5 \end{bmatrix} \end{bmatrix}$$

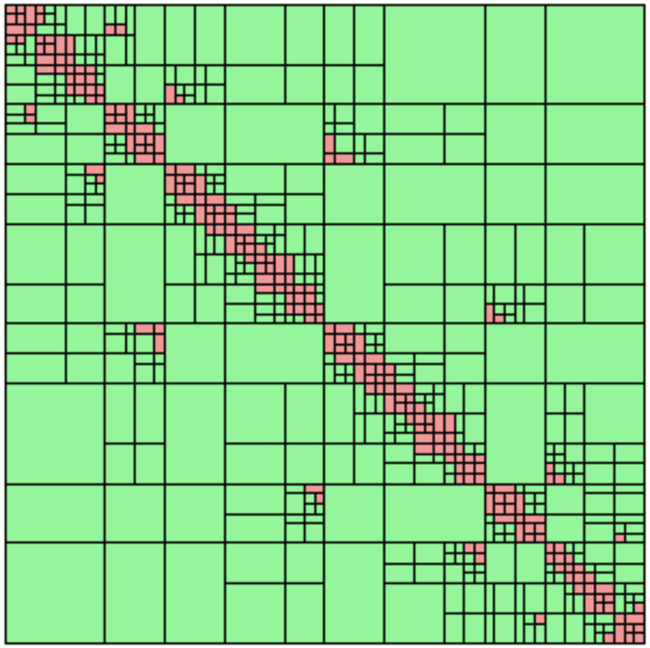
Cluster tree



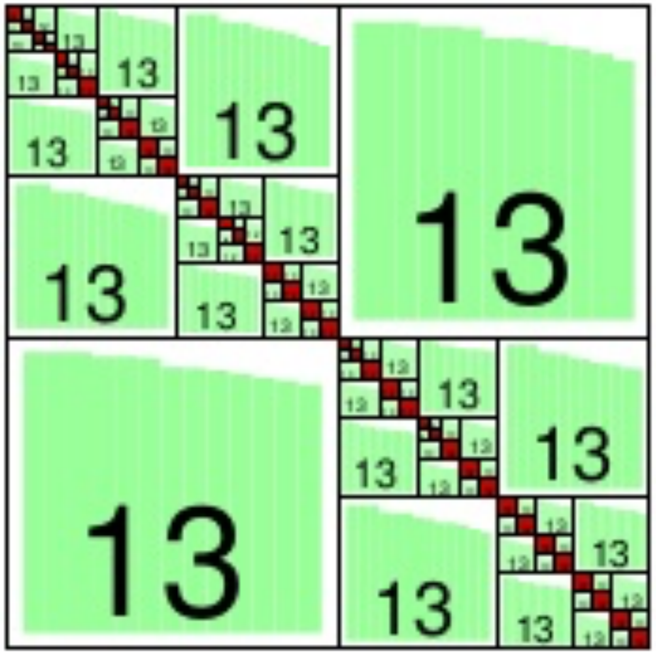




Hierarchical matrix formats

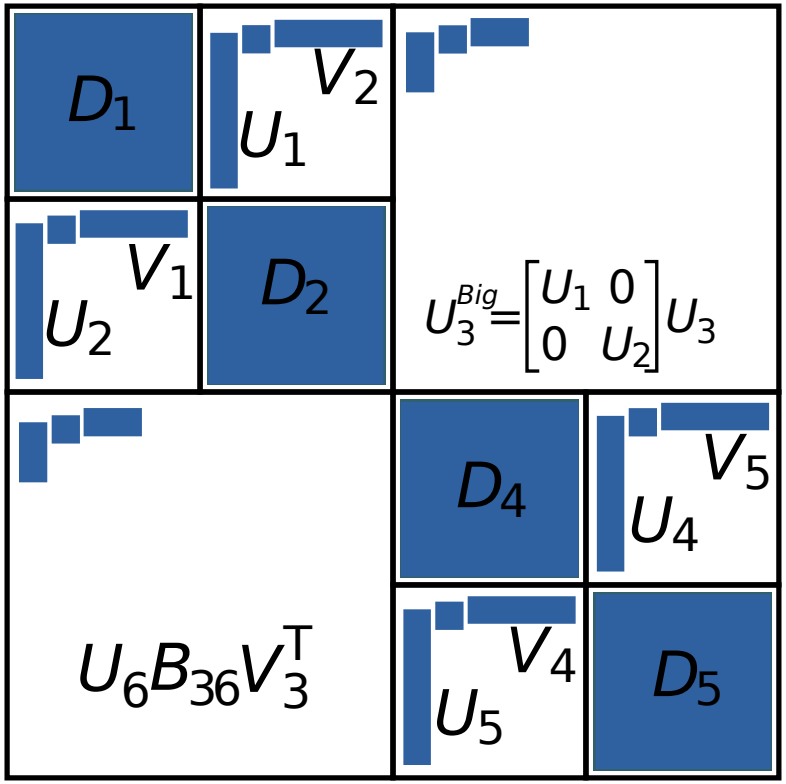


H-matrix (W. Hackbusch et al.)
O(r N log N)



HSS matrix (J. Xia et al.)
O(r N)

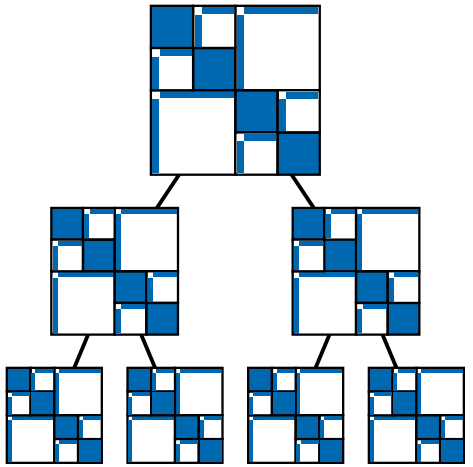
Nested bases − HSS (hierarchically semi-separable matrix)



Embedding HSS in sparse multifrontal

- **Sparse:** baseline is a sparse multifrontal direct solver.
- In addition to structural sparsity, further apply data-sparsity to dense frontal matrices:
  - O(N logN) flops, O(N) memory for 3D elliptic PDEs
- Applications: “inexact” direct solver for PDEs, algebraic preconditioner

multifrontal



Low rank compression via Randomized Sampling

Approximate range of A:

1. Pick random matrix $\Omega_{nx(k+p)}$, k target rank, p small, e.g. 10
2. Sample matrix S = A Ω (tall-skinny)
3. Compute Q = ON-basis(S) via rank-revealing QR

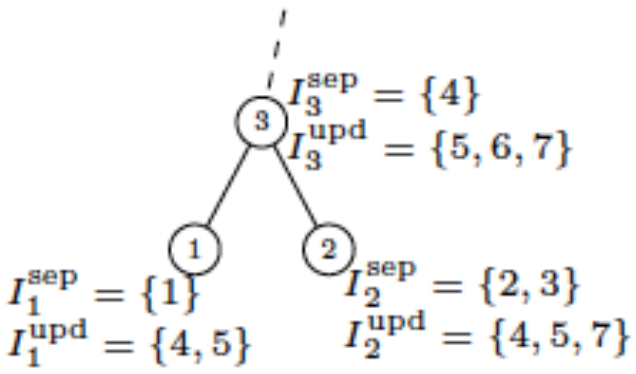
Accuracy (in 2-norm) [Halko/Martinsson/Tropp '11]

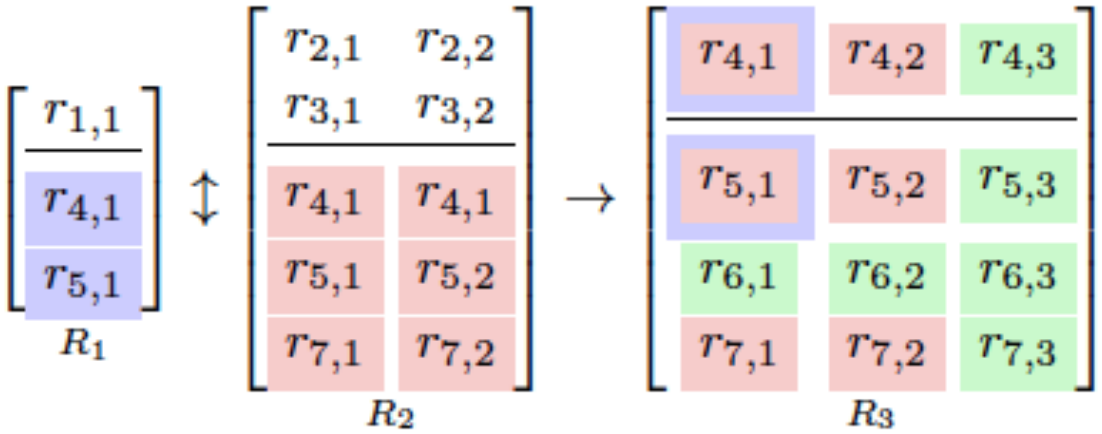
- On average: $$\mathrm{E}[\| A - QQ^* A \|] = \left(1 + \frac{4\sqrt{k+p}}{p-1}\sqrt{\min\{m,n\}}\right)\sigma_{k+1}$$
- Probabilistic bound: with probability ≥ 1 − 3 x10^{-p},
$$\| A - QQ^* A \| \leq (1 + 9\sqrt{k+p}\sqrt{\min\{m,n\}})\sigma_{k+1}$$

Benefits:

- Matrix-free: only need matvec
- When embedded in sparse frontal solver, simplifies "extend-add"

RS simplifies “extend-add” of update matrices





HSS compression via RS [Martinsson 2011, Xia 2013]

- R random matrix with d = r +p columns
  - r is the estimated maximum rank, p is oversampling parameter
- Random sampling of matrix A (unsymmetric)
  - S^r = A R, columns of S^r span the column space of A
  - S^c = A^* R, columns of S^c span the row space of A

- Only sample off-diagonal blocks at each level (Hankel blocks)
  Block diagonal matrix at level ℓ : $D^{(l)} = diag(D_1, D_2, \ldots, D_q)$

$$S^{(l)} = (A - D^{(l)})R = S^r - D^{(l)}R$$

- Rank-revealing QR on $S^{(l)}$

Practical issues

- Need ε-rank: $||A-QQ^*A|| \leq \varepsilon$
- Non-decay singular spectrum
- Dense sampling is costly using traditional matvec

Solutions:

1. Gradually increase sample size
   - Built-in automatic strategy → not to re-do already-compressed blocks
     →Need good error estimation
2. Use faster matvec in sampling: FMM, FFT, *H*-matrix, ⋯

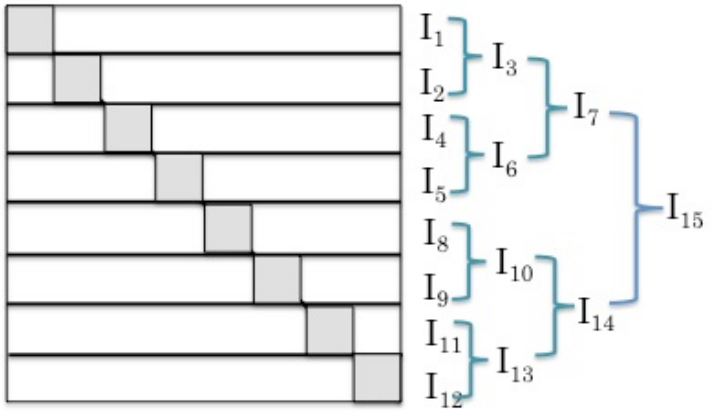
Adaptive sampling for robustness and performance

Increase sample size d, build Q incrementally (block variant)

$[S_1 \quad S_2 \quad S_3 \quad \cdots]$

$Q \leftarrow []$;
$S_1 \leftarrow A\,\Omega_1$;
$i \leftarrow 1$;
WHILE (error still large) {
  $Q_i \leftarrow QR(S_i)$ // Orthog. within current block
  $Q \leftarrow [\,Q \quad Q_i\,]$
  $S_{i+1} \leftarrow A\,\Omega_{i+1}$ // New samples
  $S_{i+1} \leftarrow (I - QQ^*)\,S_{i+1}$ // Orthog. to previous Q
  Compute error;
  $i \leftarrow i+1$
}

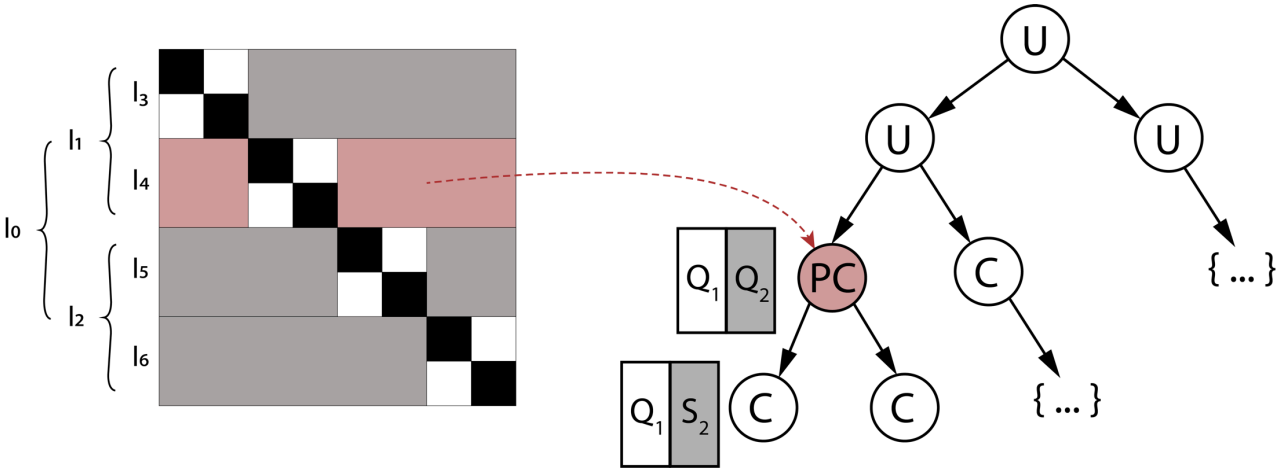
Adaptive sampling in HSS tree



Recall:

- Only have S = AΩ
- At level ℓ :

$$S^{(l)} = (A - D^{(l)})R = S^r - D^{(l)}R$$



Node state:
U – untouched
C – compressed
PC – partially compressed

STRUMPACK improved stability and performance
release 2.0.0

- Establish relation between ||S|| and ||A|| (C. Gorman)
  - For random vector x: E($||Ax||_2^2$) = $||A||_F^2$
  - For d random vectors S = AX: E($||S||_F^2$) = d $||A||_F^2$

- New stopping criteria (block variant): [Q_1, S_2]

$$\frac{\|(I-QQ^*)A\|_F}{\|A\|_F} \approx \frac{\|(I-QQ^*)S_2\|_F}{\|S_2\|_F} \leq \varepsilon$$

- Results
  - Have enough samples (robustness), but not too many (performance)
  - C. Gorman, G. Chavez, P. Ghysels, F.-H. Rouet, X.S. Li, https://arxiv.org/abs/1810.04125

BERKELEY LAB

Adaptivity example – constant singular value (worst case)

$$A = \alpha I + UDV^*, \; U,V \; rank = 120, \; D_{k,k} = 1$$

$\varepsilon_r \backslash \varepsilon_a$	1e-2	1e-4	1e-6	1e-8	1e-10	1e-12	1e-14
1e-1	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128
1e-2	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128
1e-3	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128
1e-4	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128
1e-5	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128
1e-6	120; 128	120; 128	120; 128	120; 128	120; 768	120; 768	120; 768
1e-7	120; 128	120; 128	120; 128	120; 128	120; 768	120; 768	120; 768
1e-8	120; 128	120; 128	120; 128	120; 128	120; 768	120; 768	120; 768

Table 8: Size: 100000; Alpha: 100000; Rank: 120; Decay-value: 0; d-start: 16; d-add: 16. These numbers are "(Computed HSS Rank); (Random Samples Used)". Here, we are using $\mathrm{fl}\left[\left(I - Q_1 Q_1^*\right) S_2\right] = \left(I - Q_1 Q_1^*\right)^2 S_2$, with the products computed intelligently.

Adaptivity example – constant singular value (worst case)

$$A = I + UDV^*,\ U,V\ rank = 120,\ D_{k,k} = 1$$

$\varepsilon_r \backslash \varepsilon_a$	1e-2	1e-4	1e-6	1e-8	1e-10	1e-12	1e-14
1e-1	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128
1e-2	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128
1e-3	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128
1e-4	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128
1e-5	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128
1e-6	120; 128	120; 128	120; 128	120; 128	120; 768	120; 768	120; 768
1e-7	120; 128	120; 128	120; 128	120; 128	120; 768	120; 768	120; 768
1e-8	120; 128	120; 128	120; 128	120; 128	120; 768	120; 768	120; 768

Table 8: Size: 100000; Alpha: 100000; Rank: 120; Decay-value: 0; d-start: 16; d-add: 16. These numbers are "(Computed HSS Rank); (Random Samples Used)". Here, we are using $\mathrm{fl}\left[\left(I - Q_1 Q_1^*\right) S_2\right] = \left(I - Q_1 Q_1^*\right)^2 S_2$, with the products computed intelligently.

Adaptivity cost is small

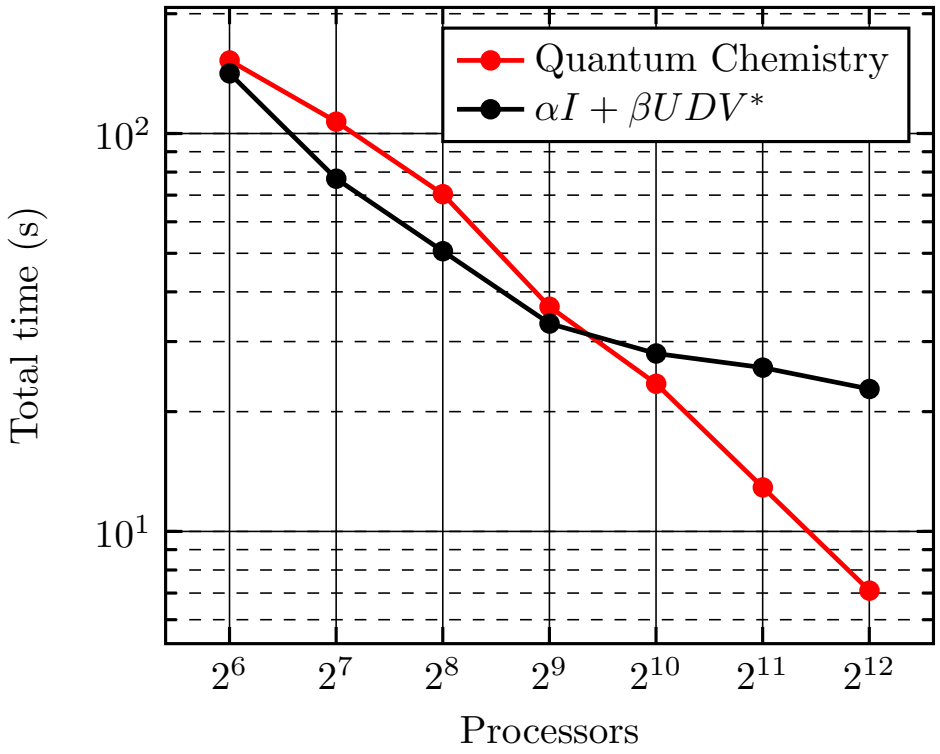
$$A = I + UDV^*,\ U,V\ rank = 1200,\ D_{k,k} = 2^{-53(k-1)/r},\ N = 60000,\ P = 1024$$

		Known rank	Adaptive	Hard restart
d0 = 128 dd = 64	Compression time (sec)	36.5	37.2	100.3
	HSS rank	1162	1267	1165
	# Increments	0	17	4

STRUMPACK dense scaling

- Cray XC40 (Cori @ NERSC)
  Quantum chemistry: $a_{i,i} = \frac{\pi^2}{6}$, $a_{i,j} = \frac{(-1)^{i-j}}{(i-j)^2 d^2}$ $n = 300k$

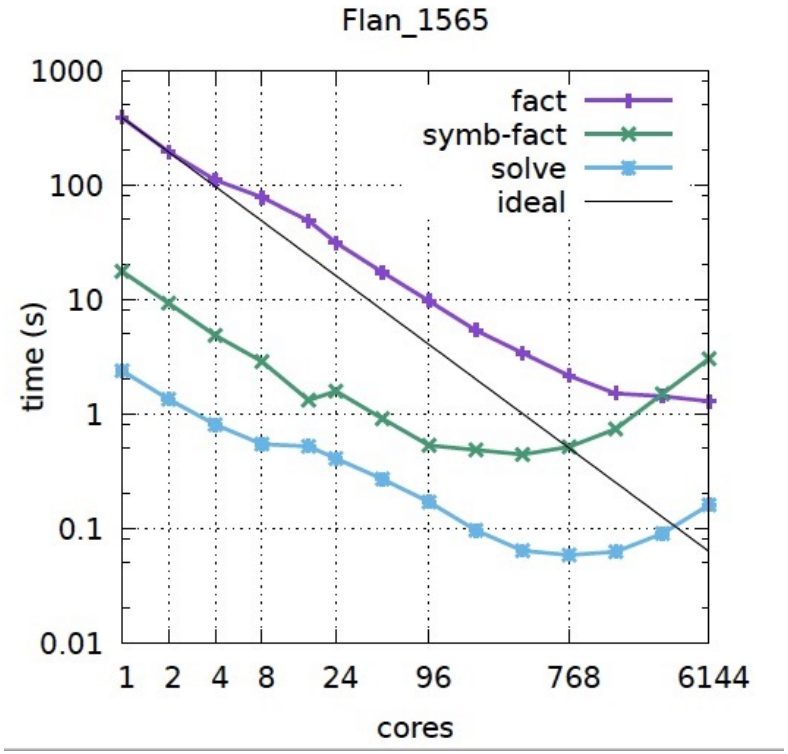
  UDV: n = 500,000, rank = 500



STRUMPACK sparse scaling [IPDPS 2017]

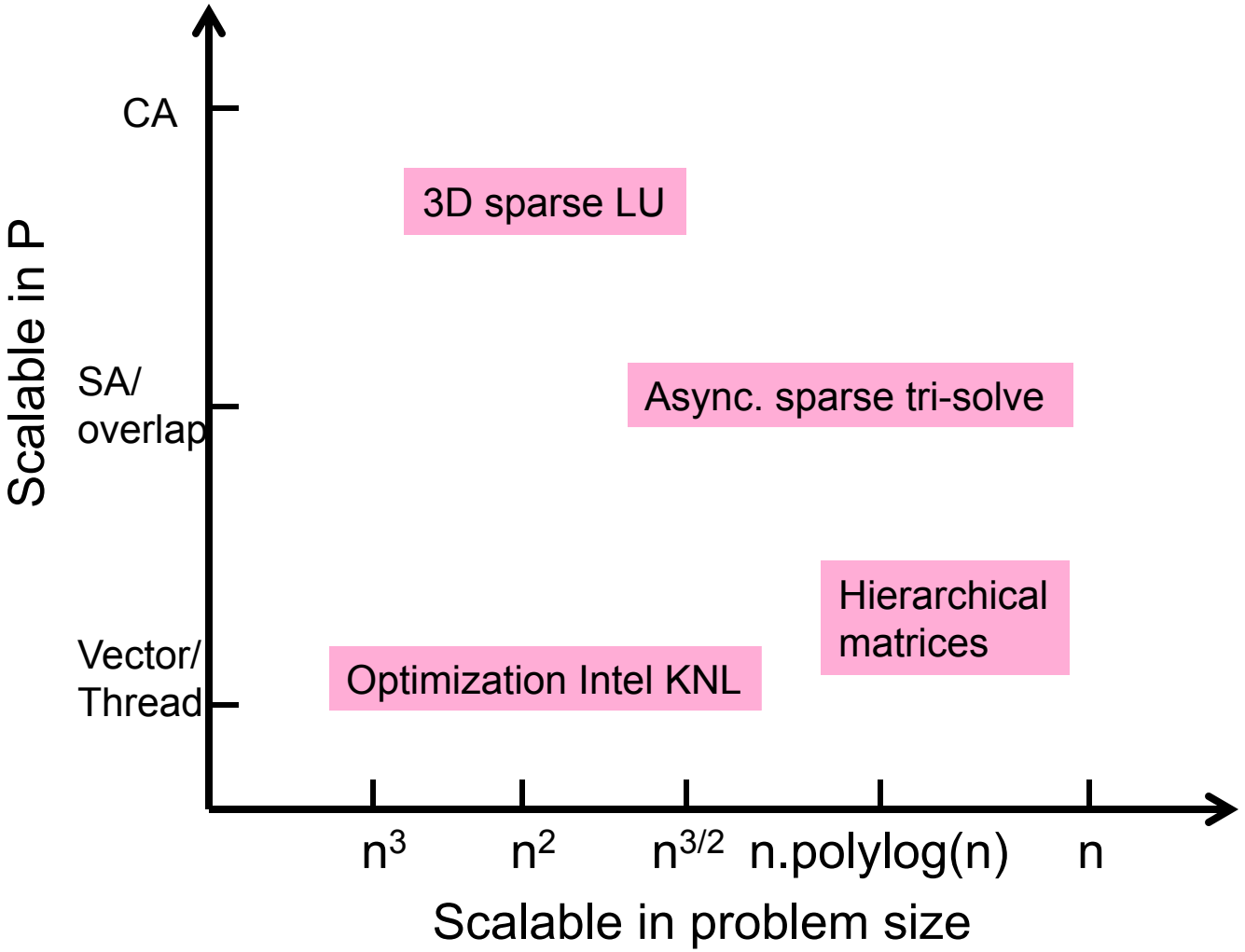
Matrix from SuiteSparse Matrix Collection:

Flan_1565 (N= 1,564,794, NNZ = 114,165,372)



- Flat MPI on nodes with 2 12-core Intel Ivy Bridge, 64GB (NERSC Edison)
- Fill-reducing reordering (ParMetis) has poor scalability, quality decreases

Future work


CA
SA/
overlap
Vector/
Thread
Scalable in P
3D sparse LU
Async. sparse tri-solve
Hierarchical
matrices
Optimization Intel KNL
n3
n2
n3/2
n.polylog(n)
n
Scalable in problem size

THANK YOU

Mitigate dense sampling cost

- Kernel Ridge Regression for classification in machine learning [IPDPS ParLearning workshop 2018]

$$\text{Kernel matrix: } K_{ij} := \exp\left(-\frac{1}{2}\frac{\| x_i - x_j \|^2}{h^2}\right), \text{ need solve } w := (K + \lambda I)^{-1} y \leftarrow \text{use HSS}$$

- Use general *H*-matrix to perform sampling for HSS construction.
- Preprocessing: data clustering (reordering) affect off-diagonal rank
  - Recursive two-means, KD-tree, PCA, etc.

SUSY: 4.5M, dimension=8. COVTYPE: 0.5M, dimension=54

	SUSY		COVTYPE	
Cores	32	512	32	512
$\mathcal{H}$ construction	173.7	18.3	36.5	32.2
HSS construction	3344.4	726.7	432.3	239.7
$\longrightarrow$ Sampling	2993.5	662.1	305.2	178.4
$\longrightarrow$ Other	350.9	64.6	127.1	61.3
Factorization	14.2	3.3	26.5	4.6
Solve	0.5	0.3	0.5	0.4