Reducing flops, communication and synchronization in sparse factorizations

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

25th International Domain Decomposition Conference
July 23-27, 2018, St John's, Newfoundland, Canada

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

Acknowledgments

This research was supported by the Exascale Computing Project (http://www.exascaleproject.org), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.

Project Number: 17-SC-20-SC

www.ExascaleProject.org

EXASCALE COMPUTING PROJECT

NNSA National Nuclear Security Administration

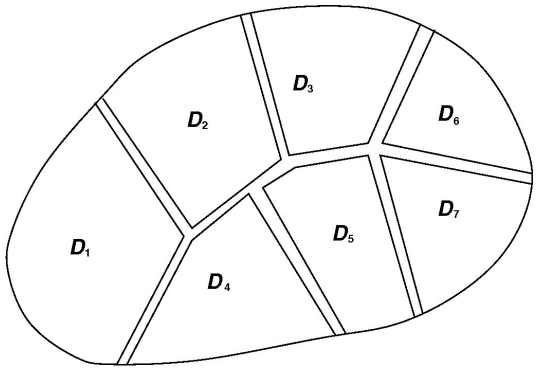
U.S. DEPARTMENT OF ENERGY | Office of Science

Domain Decomposition − a powerful algorithm framework for scalability

Divide-and-conquer paradigm . . .

- Divide entire problem (domain, graph) into subproblems (subdomains, subgraphs)
- Solve the subproblems
- Solve the interface problem (e.g., Schur complement)

Example: Schur-complement method



$$\left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right) = \left(\begin{array}{cccc|c} D_1 & & & & E_1 \\ & D_2 & & & E_2 \\ & & \ddots & & \vdots \\ & & & D_k & E_k \\ \hline F_1 & F_2 & \cdots & F_k & A_{22} \end{array}\right)$$

Schur-complement method

1. Reorder into 2x2 block system, A_{11} is block diagonal

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

2. Schur complement

$$S = A_{22} - A_{21} A_{11}^{-1} A_{12} = A_{22} - (U_{11}^{-T} A_{21}^{T})^{T} (L_{11}^{-1} A_{12}) = A_{22} - W \cdot G$$

where $A_{11} = L_{11} U_{11}$

S corresponds to interface (separator) variables, no need to form explicitly

3. Compute the solution

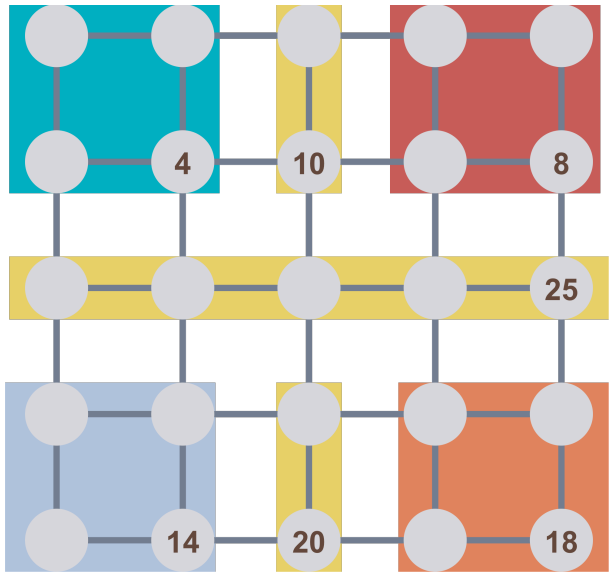
(1) $x_2 = S^{-1}(b_2 - A_{21} A_{11}^{-1} b_1)$ ← iterative solver

(2) $x_1 = A_{11}^{-1}(b_1 - A_{12} x_2)$ ← direct solver

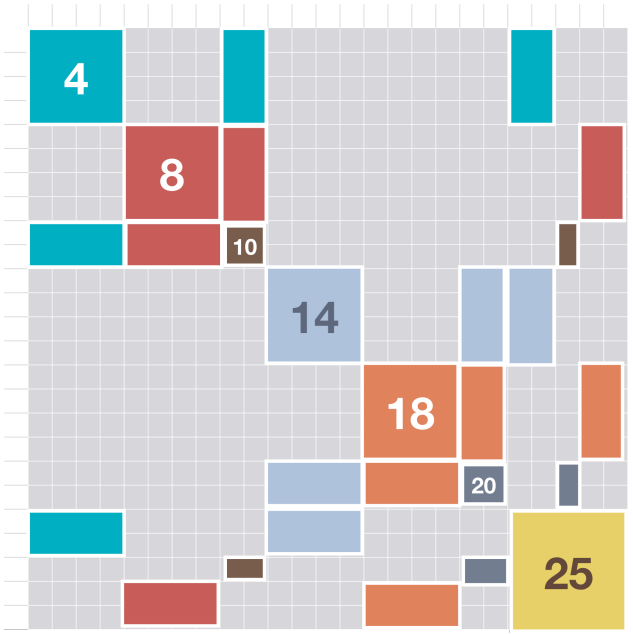
Graph Partitioning – general way to sub-divide problem

- Nested Dissection [George'78]

Geometry

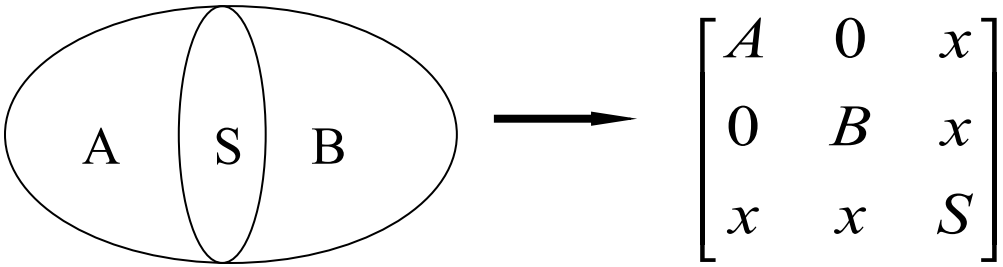


Matrix reordered



Graph Partitioning – general way to sub-divide problem

- Generalized nested dissection *(Lipton/Rose/Tarjan '79)*
- Sparse matrix ⇔ Graph:
  - nonzero A(i, j) correspond to an edge in G
  - Lead to geometry-oblivious algebraic solvers

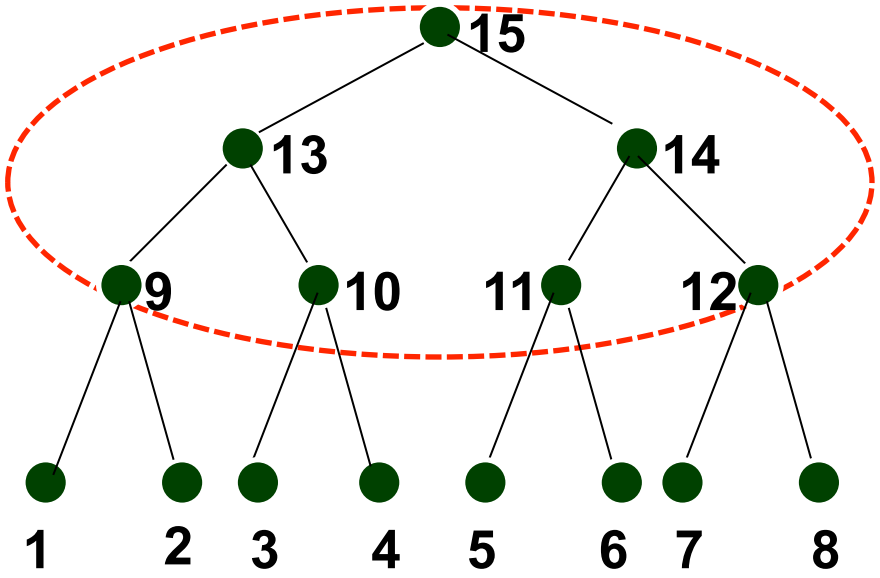


- Software: (Par)Metis, (PT)-Scotch, Chaco, ⋯
  - Recursively find the smallest possible separator S at each level

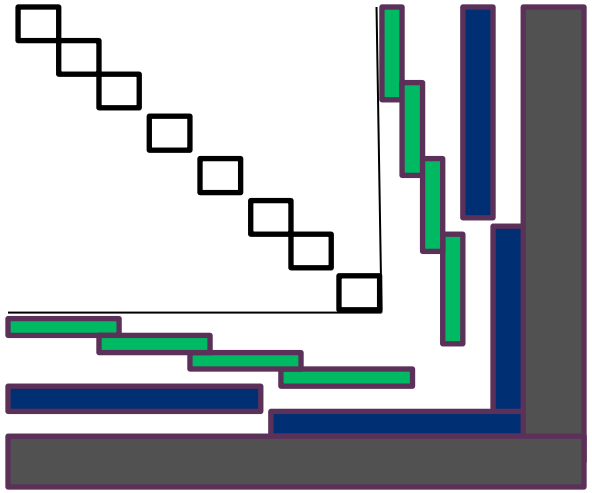
Graph partitioning for Schur-complement method

- Permute all the separators to the end

Separator tree



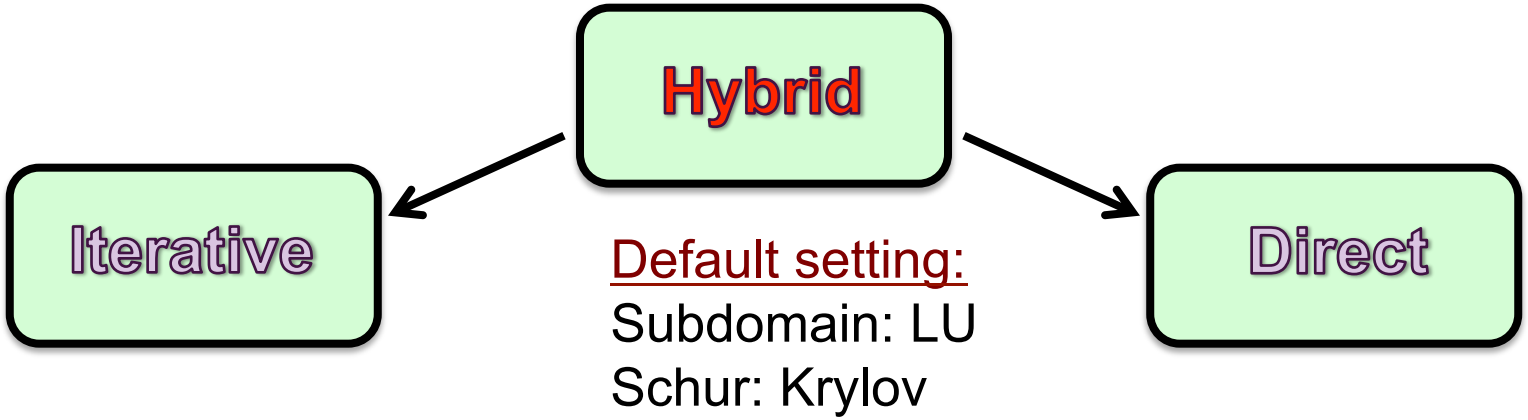
Matrix reordered



Multilevel parallelism is essential for scalability and robustness

PDSLin Schur-complement solver *(Yamazaki, Li)*

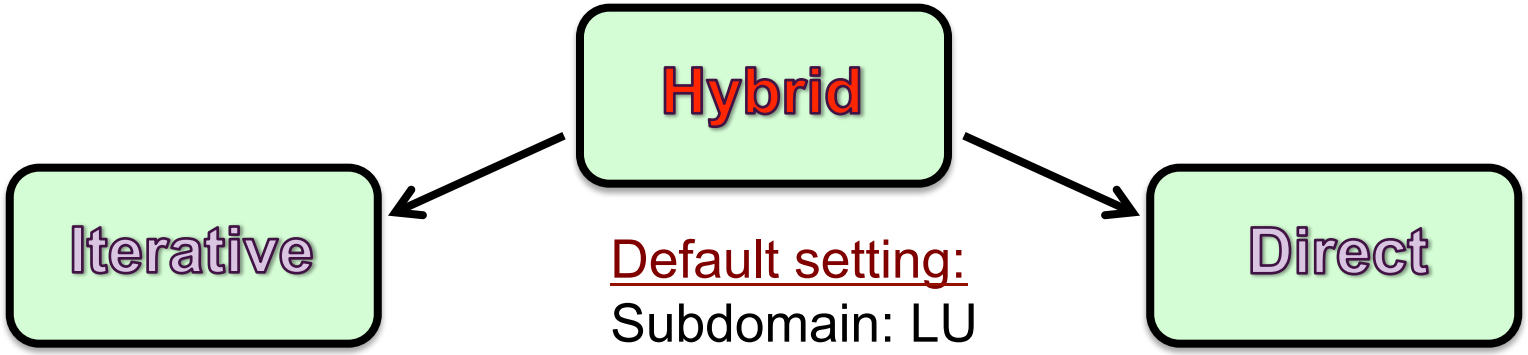
http://portal.nersc.gov/project/sparse/pdslin/



Other hybrid solvers:
HIPS (Henon/Saad}, MaPHyS (Agullo et al.), ShyLU (Boman et al.)

PDSLin Schur-complement solver *(Yamazaki, Li)*

http://portal.nersc.gov/project/sparse/pdslin/



Default setting:
Subdomain: LU
Schur: Krylov

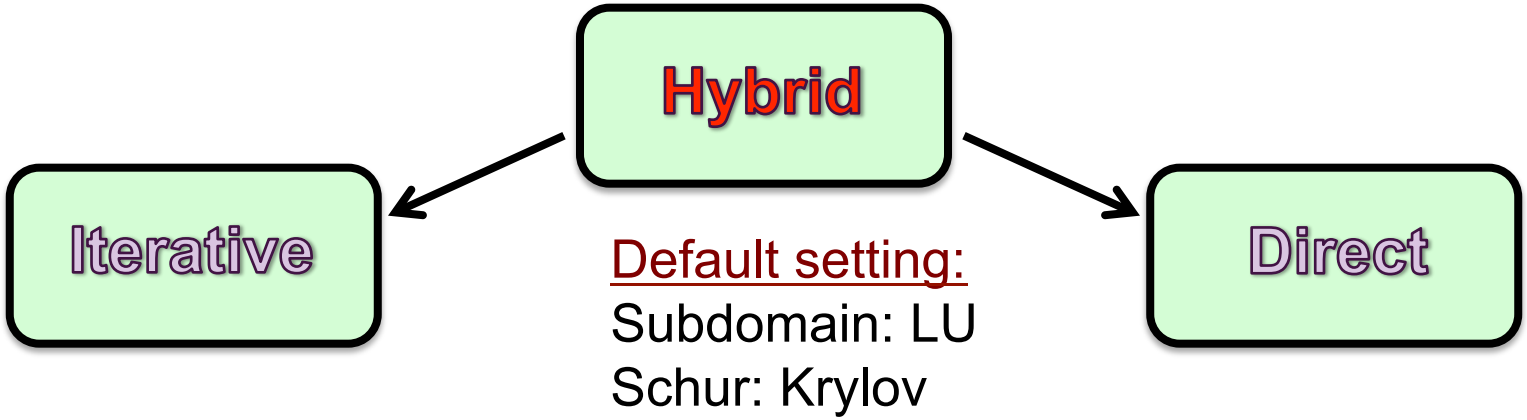
Options set by user:
1) num_doms = 0
Schur: Krylov
Or,
2) FGMRES Inner-Outer:
Subdomain: ILU
Schur: Krylov

Options set by user:
1) Subdomain: LU
Schur: LU
drop_tol = 0.0
Or,
2) num_doms = 1
Domain: LU

Other hybrid solvers:
HIPS (Henon/Saad}, MaPHyS (Agullo et al.), ShyLU (Boman et al.)

PDSLin Schur-complement solver *(Yamazaki, Li)*

http://portal.nersc.gov/project/sparse/pdslin



Applications:

- Interior eigenvalue problems in accelerator cavity design
  - 52.7M DOFs, 4.3B nonzeros
  - 128 subdomains, 2048 cores (16 cores per subdomain)
- Two-fluid MHD magnetic reconnection problems for Tokamak design *(Yuan, Li, Keyes, et al.)*
  - 4.2M DOFs

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, ...

Low-rank approximate factorization with quasi-linear arithmetic complexity

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

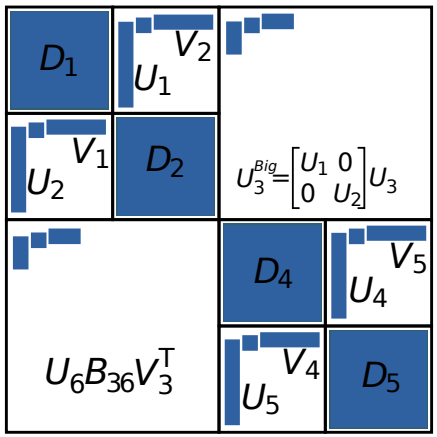
Sparse preconditioner – Ghysels, Li, Gorman, Rouet (IPDPS 17)
Adaptive sampling – Gorman, Chavez, Ghysels, Rouet, Li (SISC 2018, preprint)

STRUMPACK "inexact" direct solver

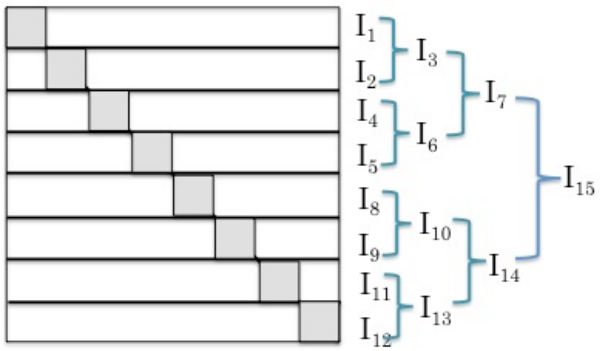
P. Ghysels, G. Chavez, C. Gorman, F.-H. Rouet, XL

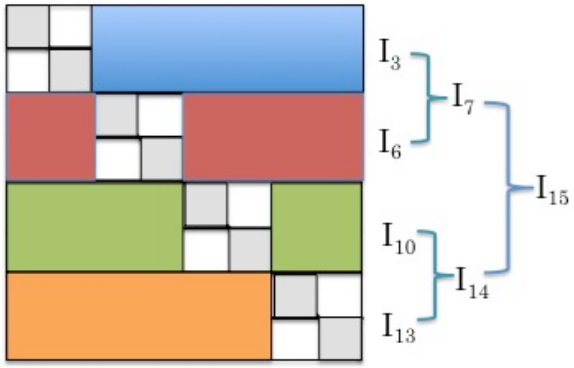
- **Dense:** exploit data-sparsity with low-rank compression.
- Hierarchical matrix algebra generalizes Fast Multipole
  - Diagonal block **("near field")** exact; off-diagonal block **("far field")** approximated via low-rank compression.
  - Hierarchically semi-separable (HSS), HODLR, etc.
  - **Nested bases + randomized sampling** to achieve linear scaling for sparse.
- Applications: PDEs, BEM methods, integral equations, machine learning, and structured matrices such as Toeplitz, Cauchy matrices.

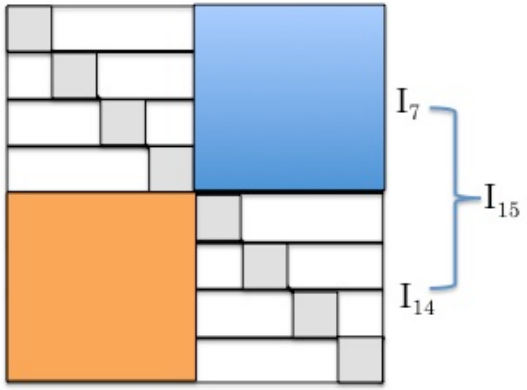
$$A \approx \begin{bmatrix} \begin{bmatrix} D_1 & U_1 B_1 V_2^T \\ U_2 B_2 V_1^T & D_2 \end{bmatrix} & U_3 B_3 V_6^T \\ U_6 B_6 V_3^T & \begin{bmatrix} D_4 & U_4 B_4 V_5^T \\ U_5 B_5 V_4^T & D_5 \end{bmatrix} \end{bmatrix}$$



Cluster tree





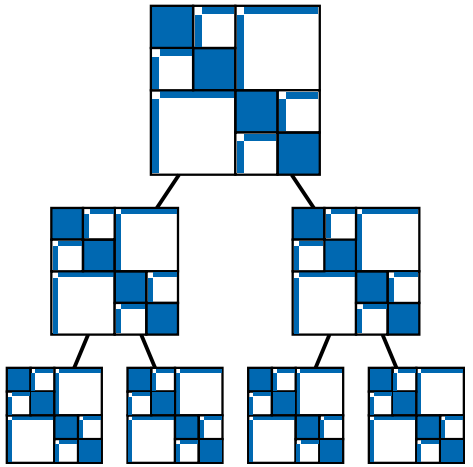


STRUMPACK "inexact" direct solver

P. Ghysels, G. Chavez, C. Gorman, F.-H. Rouet, XL

- **Sparse:** baseline is a sparse multifrontal direct solver.
- In addition to structural sparsity, further apply data-sparsity with low-rank compression:
  - O(N logN) flops, O(N) memory for 3D elliptic PDEs.
- Applications: 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"

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
   - "Hard restart" from scratch → Costly!
   - 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

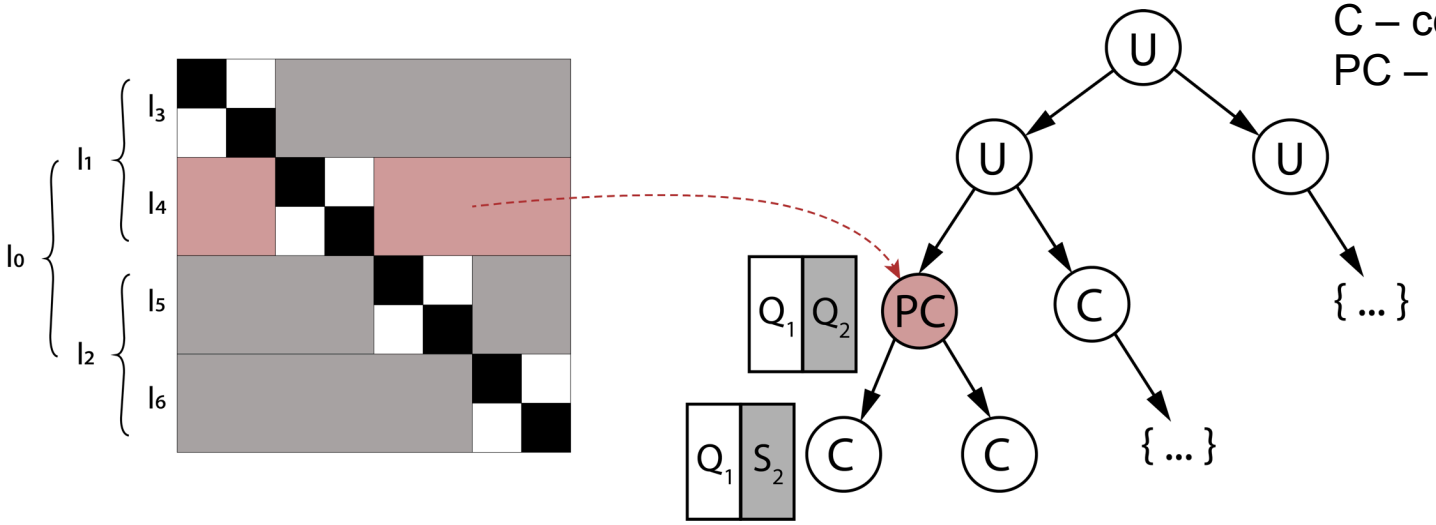
Increase sample size d, build Q incrementally (block variant)

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

$Q \leftarrow []$;
$S_1 \leftarrow A\,\Omega_1$;
$i \leftarrow 1$;
WHILE (error still large) {
  $Q_i \leftarrow QR(S_i)$ // Orthog. within current block
  $Q \leftarrow [\,Q \;\; Q_i\,]$
  $S_{i+1} \leftarrow A\,\Omega_{i+1}$ // 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

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



Recall:

- Only have S = AΩ
- At level ℓ :

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

Adaptive sampling: need good error estimation

- Goal: Bound error for A: $||(I - QQ^{*)}A||$,
- . . . but A is not available, can only use S

Need establish a stochastic relationship between

||A|| and ||S||

Stochastic norm estimation

Let $A \in \Re^{mxn}$, and $x \in \Re^{n}$ with $x_i \sim N(0,1)$. Consider SVD:

$$A = U\Sigma V^* = [U_1 \; U_2] \begin{bmatrix} \Sigma_r & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_1^* \\ V_2^* \end{bmatrix}$$

Define $\zeta = V^* x$, ζ is also a Gaussian random vector.

$$\|Ax\|_2^2 = \|\Sigma\zeta\|_2^2 = \zeta_1^2\sigma_1^2 + \cdots + \zeta_r^2\sigma_r^2$$

here, $\sigma_1 \geq \sigma_2 \cdots \geq \sigma_r > 0$ are positive singular values. Then:

$$\mathrm{E}\left[\|Ax\|_2^2\right] = \sigma_1^2 + \cdots + \sigma_r^2 = \|A\|_F^2$$

For d sample vectors: $\mathrm{E}\left[\|S\|_F^2\right] = d\|A\|_F^2$

STRUMPACK improved stability and performance
release 2.0.0

- New stopping criteria (block variant):

Let $[S_1 \;\; S_2] = [AR_1 \;\; AR_2]$, $Q \leftarrow QR(S_1)$

Absolute criterion:

$$\| (I - QQ^*)A \|_F \approx \frac{1}{\sqrt{d}} \| (I - QQ^*)S_2 \|_F \leq \varepsilon_a$$

Relative criterion:

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

- Results *(SISC preprint)*
  - Have enough samples (robustness), but not too many (performance)

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.

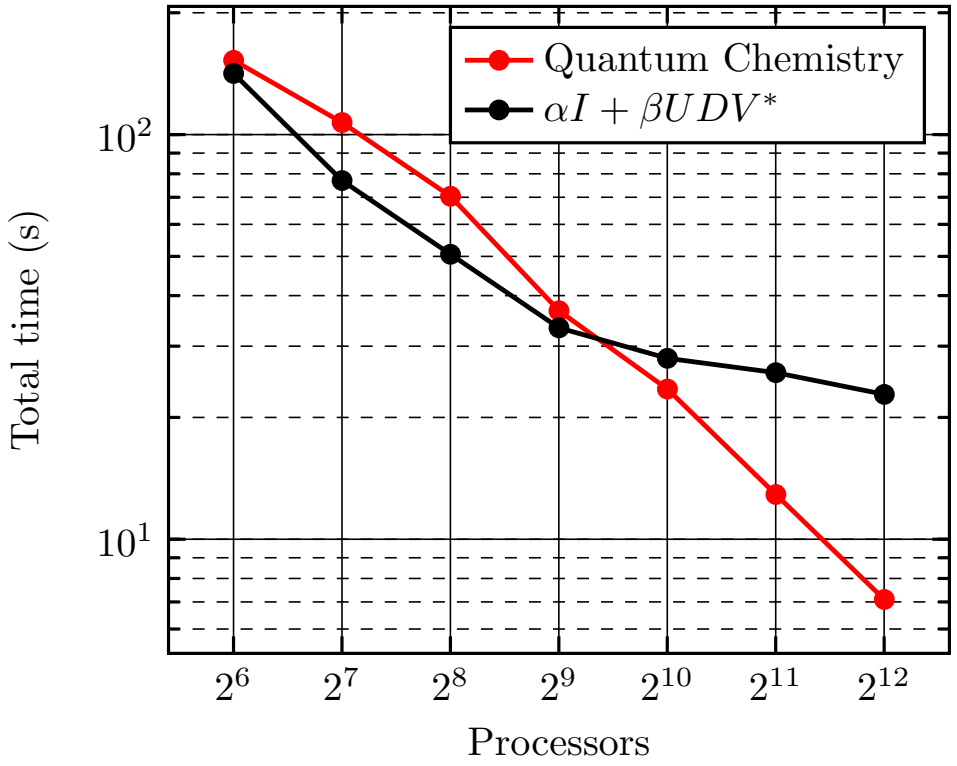
Adaptivity cost is small

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

		"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

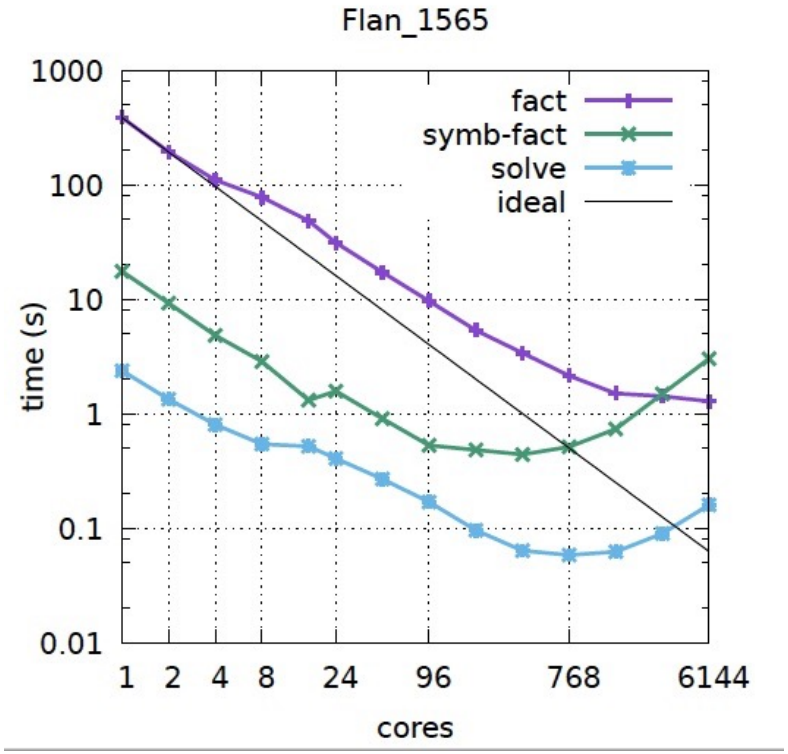
- Cray XC40 (Cori @ NERSC)
  - Quantum chemistry: $a_{i,i} = \frac{\pi^2}{6}$, $a_{i,j} = \frac{(-1)^{i-j}}{(i-j)^2 d^2}$ $N = 300k$
  - $A = I + UDV^*$, U,V of rank $r = 500$, $D_{k,k} = 2^{-53(k-1)/r}$, $N = 500{,}000$



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)

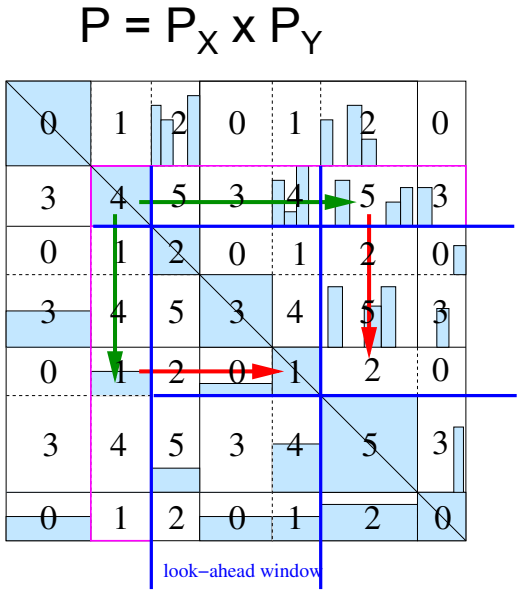
Communication-avoiding 3D sparse LU factorization

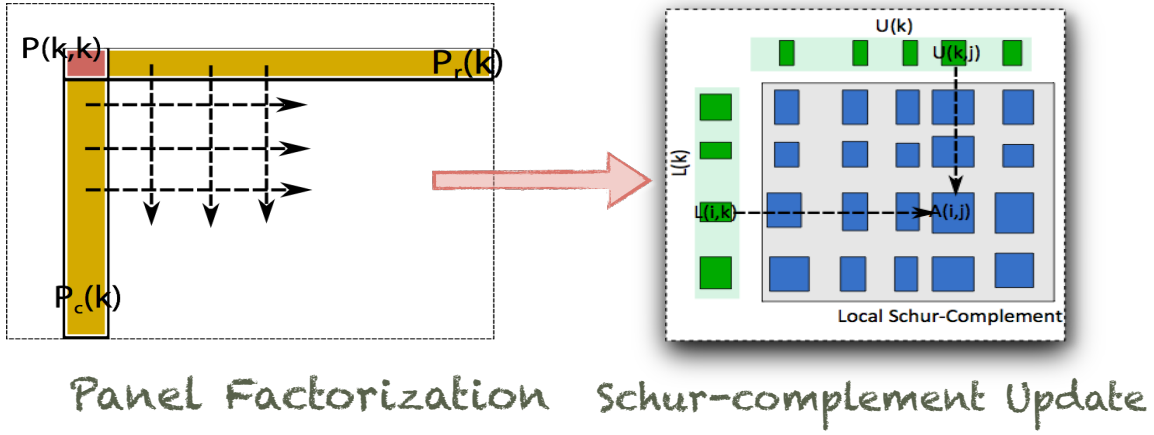
[P. Sao, XL, R. Vuduc, IPDPS'18]

(Sao thesis, 2018, Georgia Tech)

Review of 2D sparse factorization in SuperLU_DIST

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



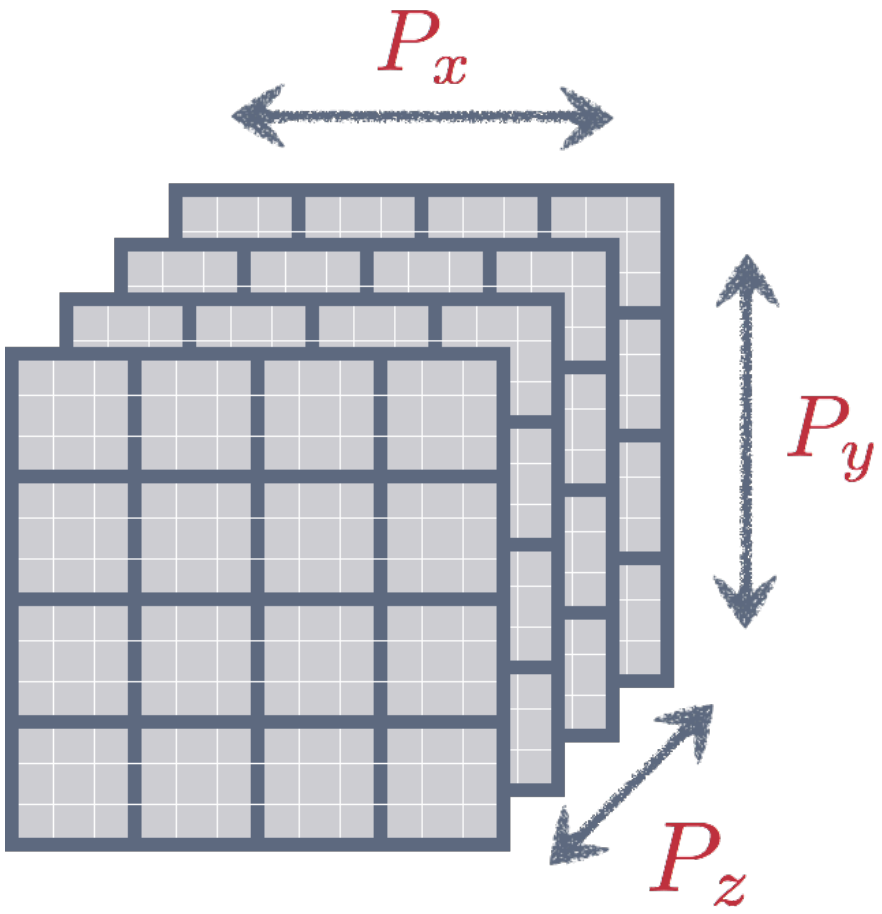


- Graph at step k+1 differs from step k

Limitations of 2D algorithm:

- Sequential Schur-complement update
- Fixed latency cost: Panel factorization on critical path

3D process grid



Overview of 3D algorithm

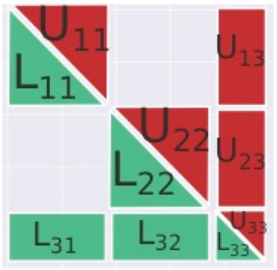
- All known "3D-LU" algorithms† are for **dense LU**. They reduce communication volume but increase latency.
- For sparse LU, we can **reduce** both the latency and bandwidth for "planar" problems **asymptotically**, and achieve constant-factor reductions for "non-planar" ones.
- There are other memory-for-communication techniques,‡ including **multifrontal methods**. We claim better memory and process scalability.

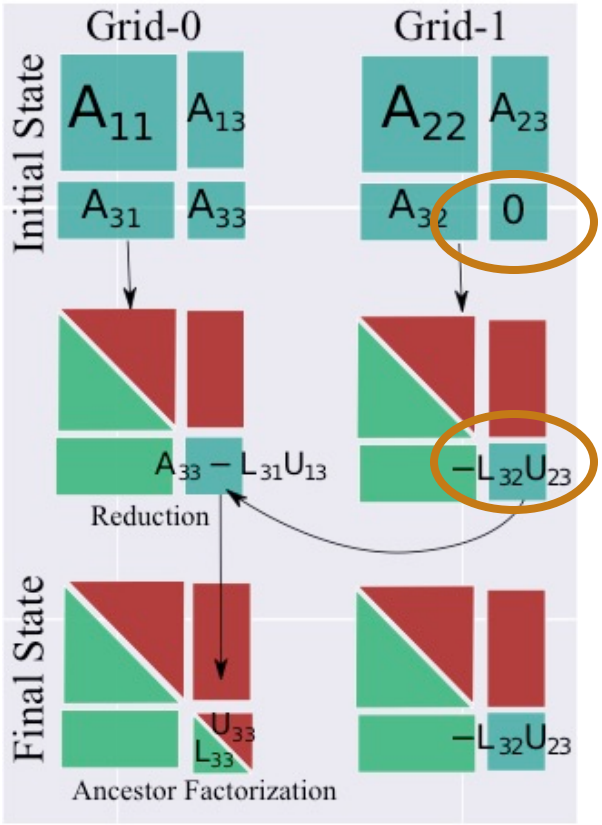
† Ashcraft (1991); Irony & Toledo (2002); Solomonik & Demmel (2011)
‡ Hulbert & Zmijewski (1991); Gupta et al. (1997)

3D sparse factorization

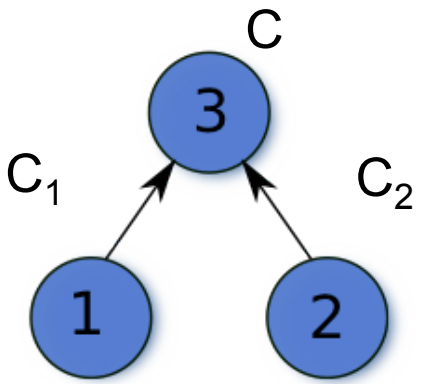
- Replicate Schur-complement on multiple 2D process grids
- P = P_{XY} x P_Z : factor P into P_Z slices of P_{XY} 2D process grids
- Trade extra memory for less communication, more parallelism.
  - Each Schur-complement block is split-updated by multiple processes
  - Reduction of Schur-complement of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

Nested Dissection



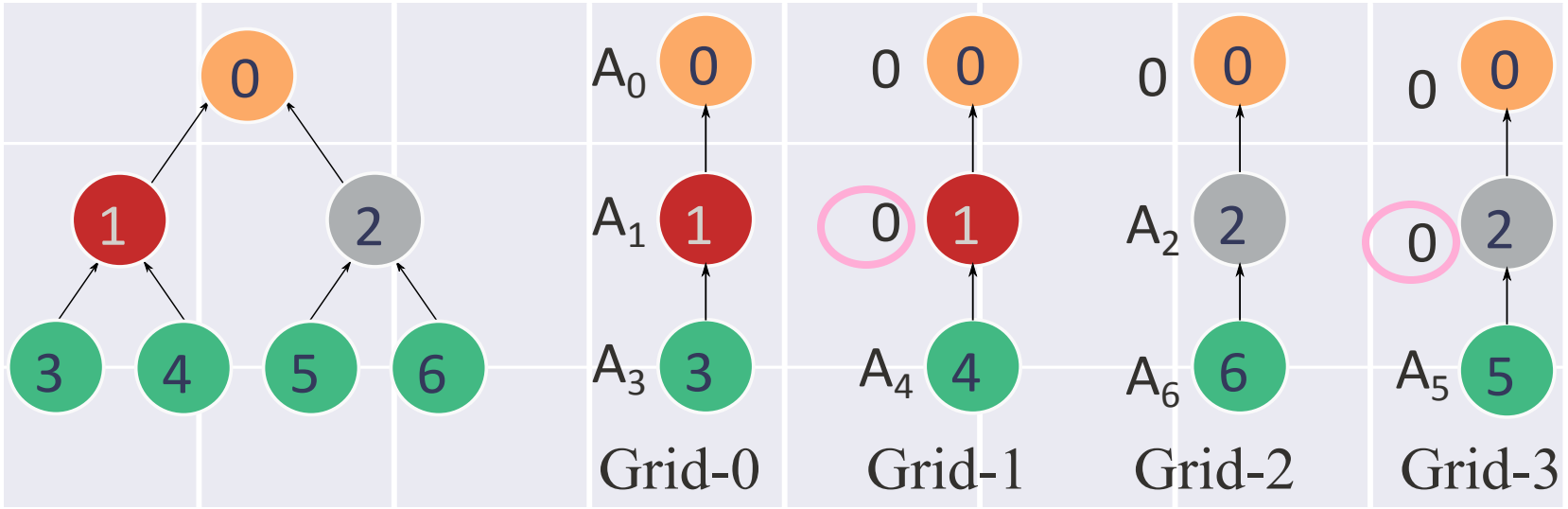


$P_Z = 2$
two slices of 2D grids



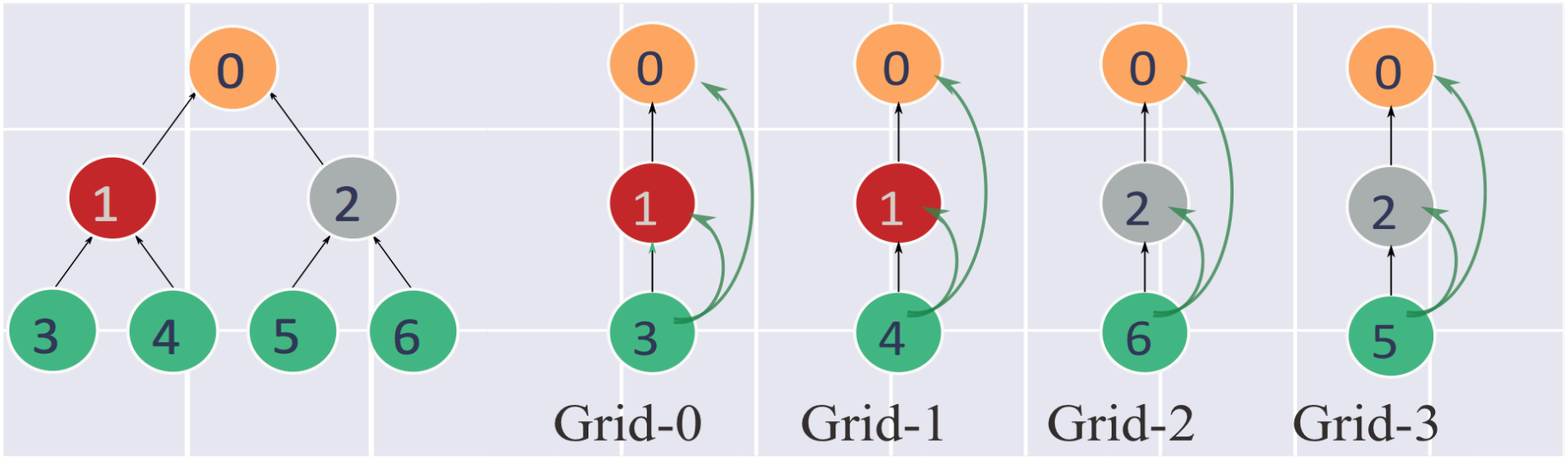
3D sparse factorization: $P_z = 4$

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



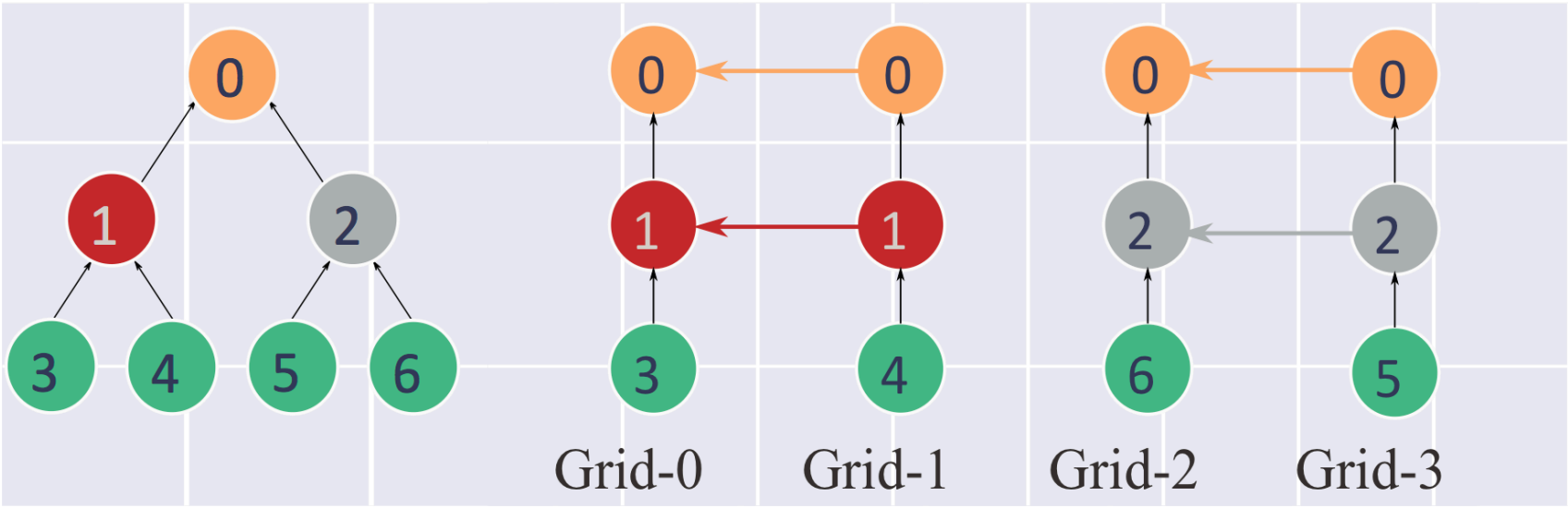
- Node 0 replicated on all process Grids
- Node 1 replicated on process Grids 0, 1
- Node 2 replicated on process Grids 2, 3

3D sparse factorization: $P_z = 4$



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

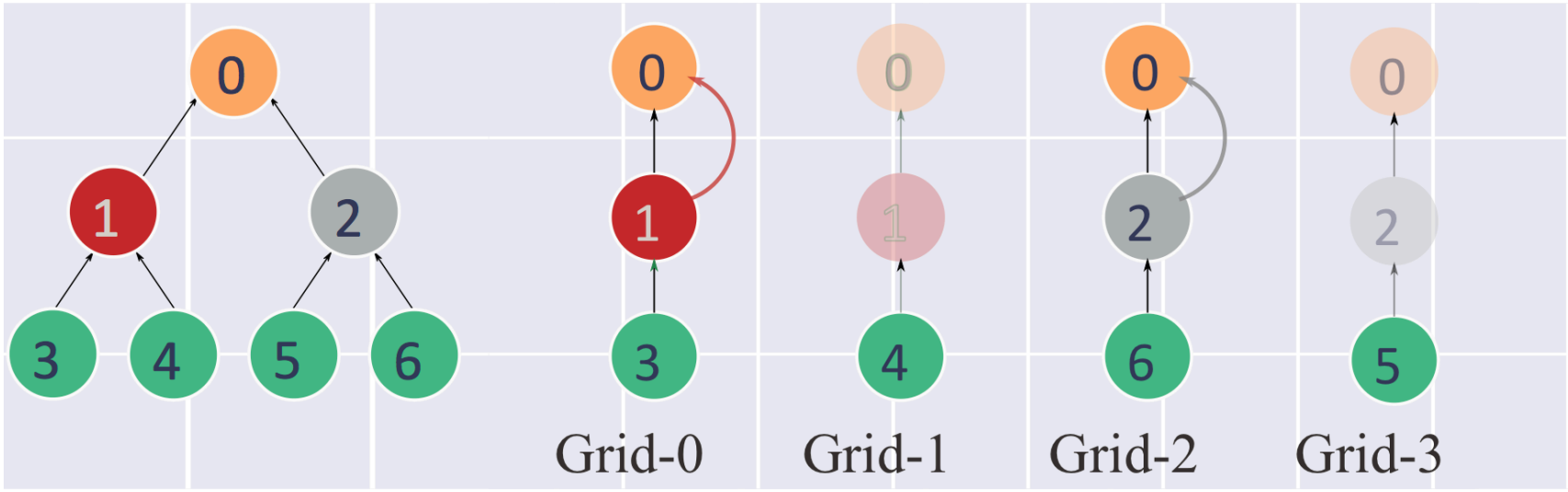
3D sparse factorization: $P_z = 4$



Ancestors reduce:

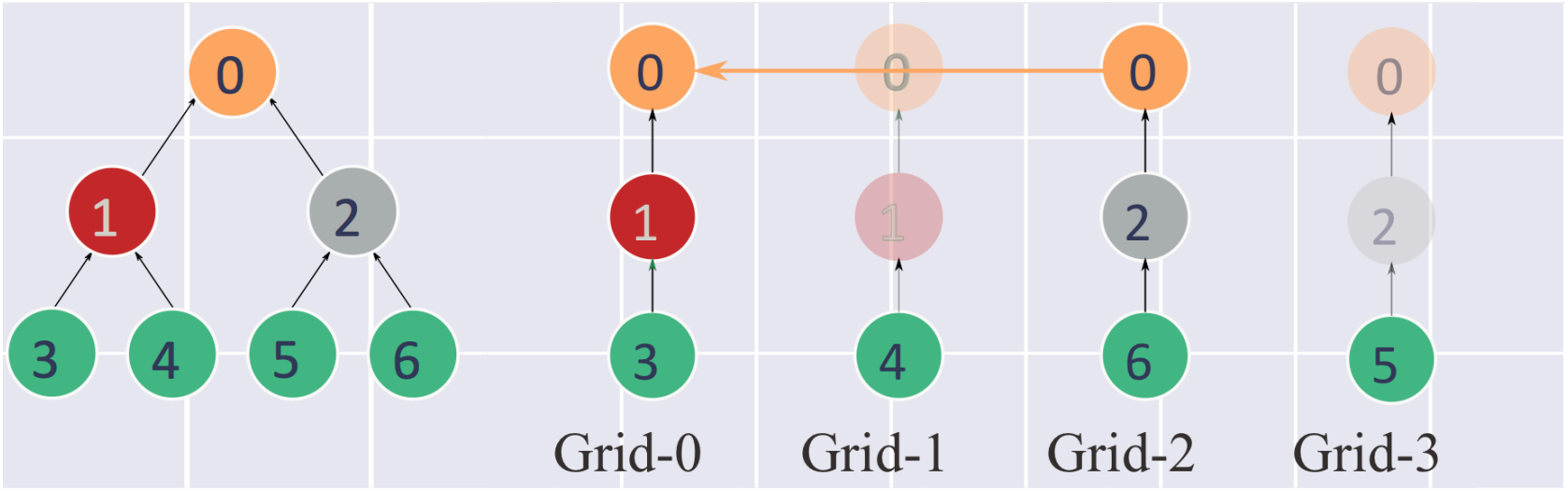
Grids 0-1 reduce to Grid 0
Grids 2-3 reduce to Grid 2

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)

- Count messages and volume

  {Latency, inverse-bandwidth} model
  - Point-to-point communication of a message of W words:

    t = alpha + W * beta
  - Broadcast / Reduction a message of W words among P processes:

    t = (alpha + W * beta) * log(P)

- Consider two cases

  (0) "Planar" geometries, e.g., PDE on a 2D domain

  (1) "Non-planar" case, e.g., 3D PDE

Planar graphs have good separators

Top-level separator size

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

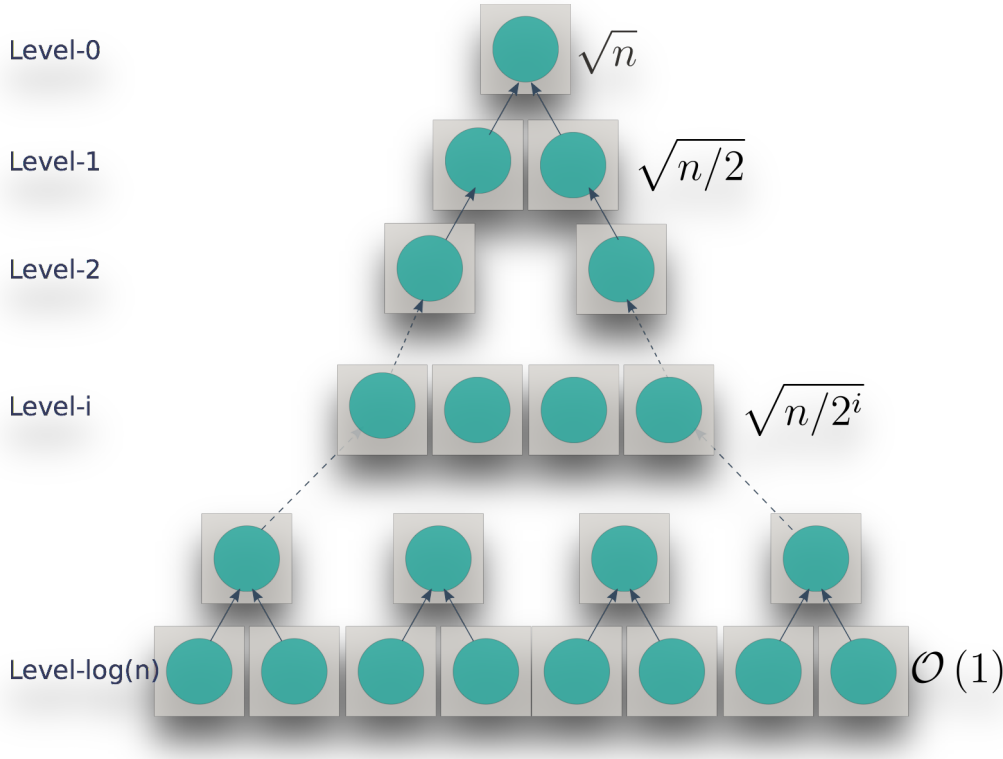
Memory of top-level separator

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

Total size of $L + U$

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

“Planar” geometry



George (1978), Lipton & Tarjan (1979)

Memory / process

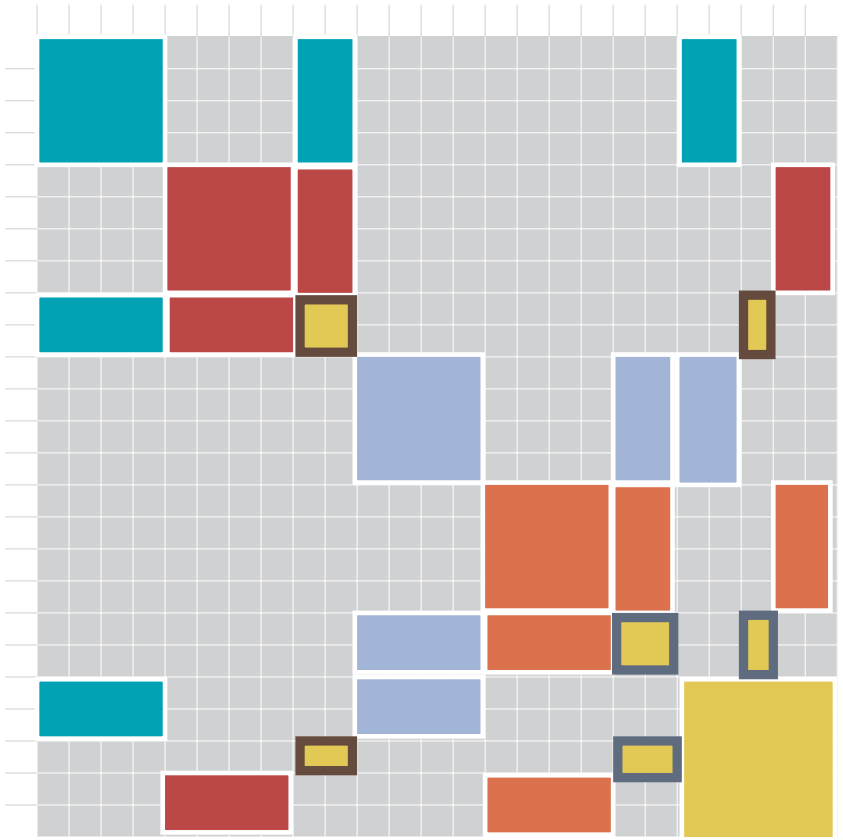
$$\frac{n \log n}{P}$$

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

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

⇒ **Only O(1) increase!**

Communication volume / process

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

Optimal tradeoff!

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

Latency (msgs) / process

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

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

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

Top-level separator size

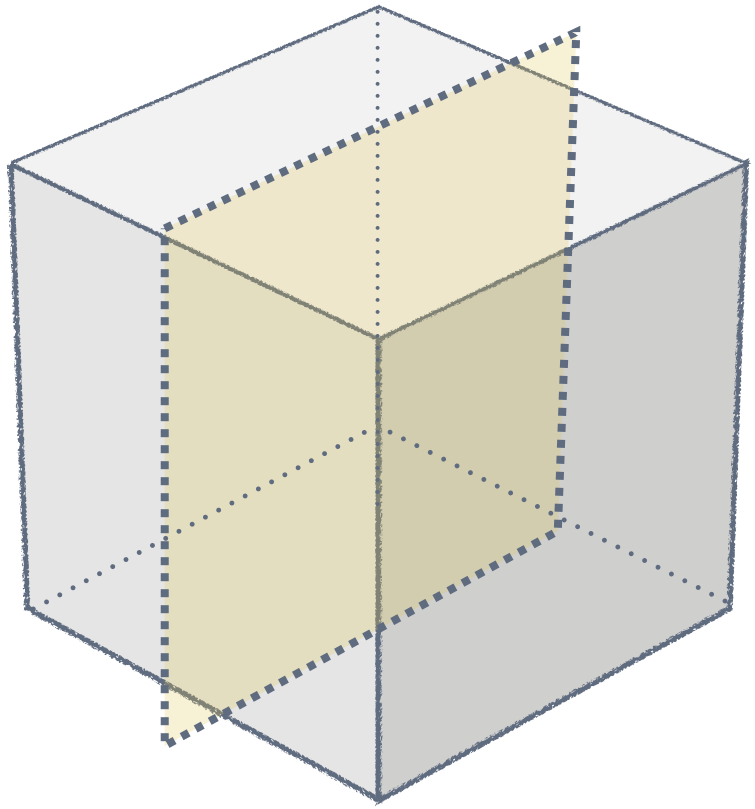
$$\mathcal{O}\left(n^{\frac{2}{3}}\right)$$

Memory of top-level separator

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

Total size of $L + U$

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



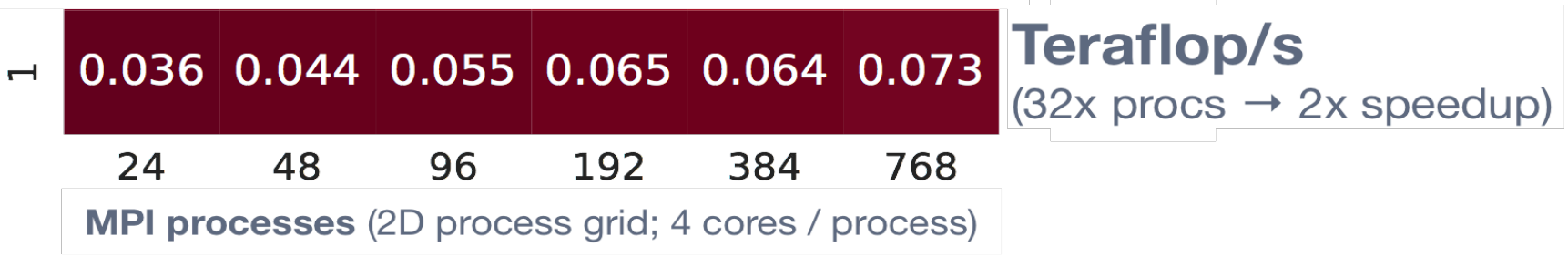
“Non-planar” (3D) geometry

Empirical study on Cray XC30 (edison @ NERSC)

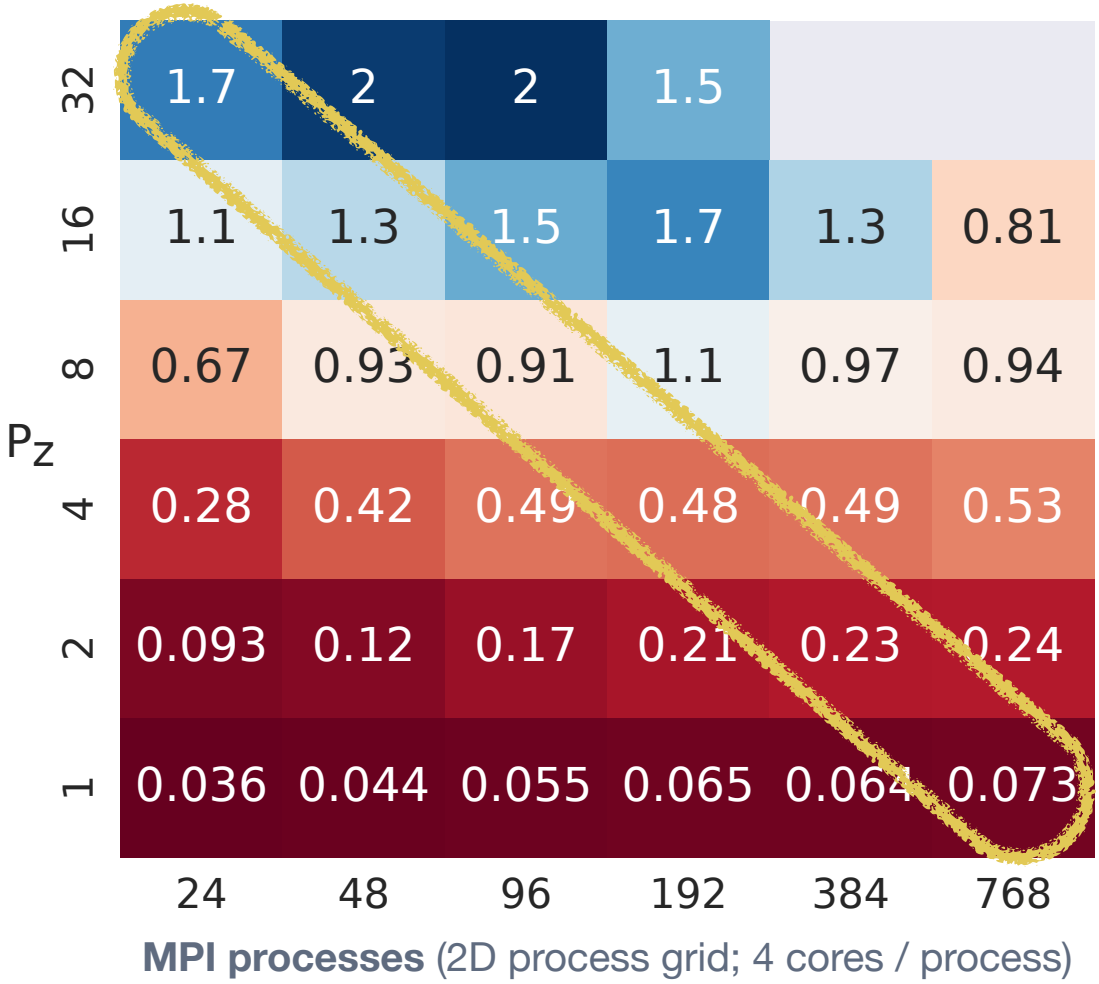
Intel IvyBridge processor: dual-socket, 12-core per socket

Baseline 2D algorithm: 10 – 55 seconds on 16 nodes.

Example: K2D5pt4096



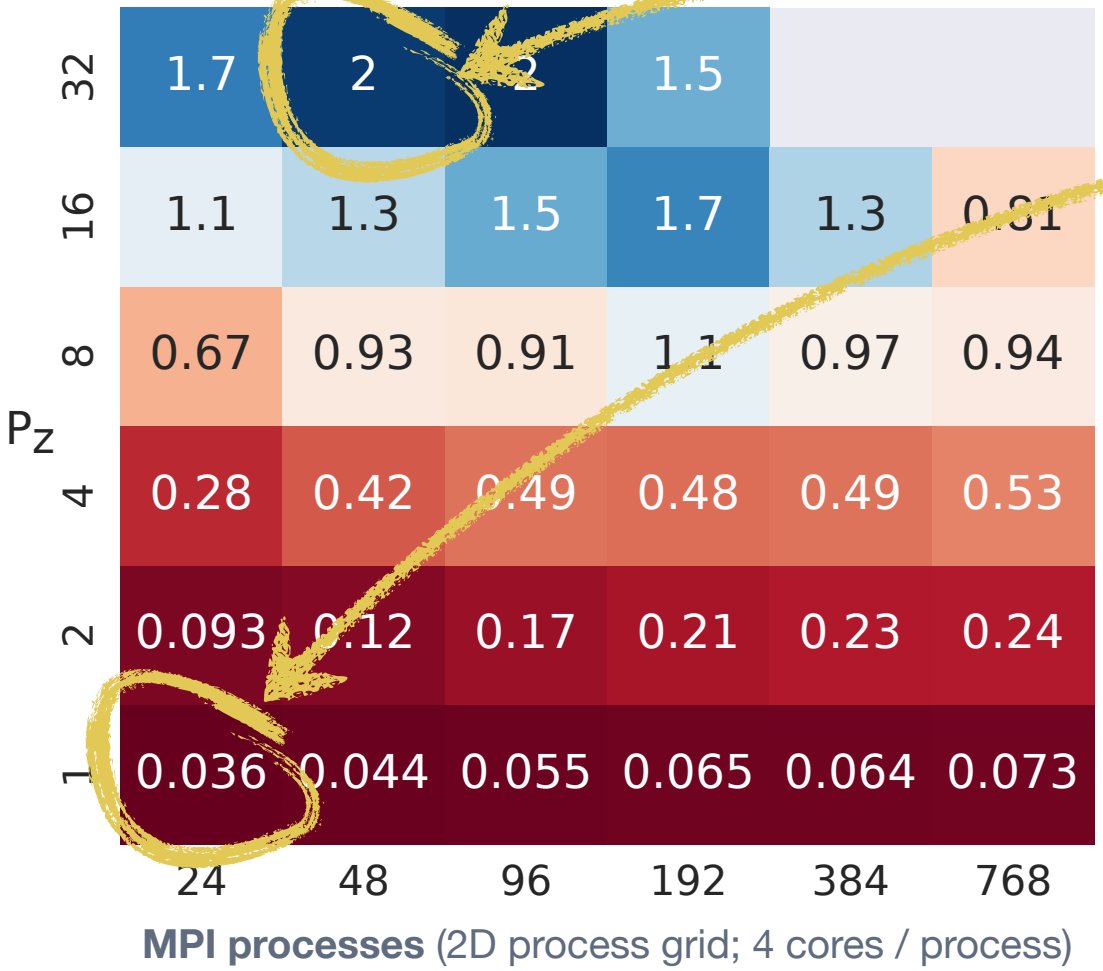
hpcgarage.org/hookem



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 on CPUs

Strong scale to 24,000 cores.

Compared to 2D algorithm:

	Comm. reduction relative to 2D alg. Theory	Empirical	Total Speedup	Memory overhead
Planar geometry	Volume $O\left(\sqrt{\log n}\right)$ Latency $O\left(\log n\right)$	4.7 x	27 x	30% @ P_Z=16
Non-planar	Volume O(1) Latency O(1)	3.7 x	3.3 x	2 x @ P_Z=16

Combining 3D algorithm with GPU acceleration

Sao, Vuduc, Li (JPDC preprint, 2018)

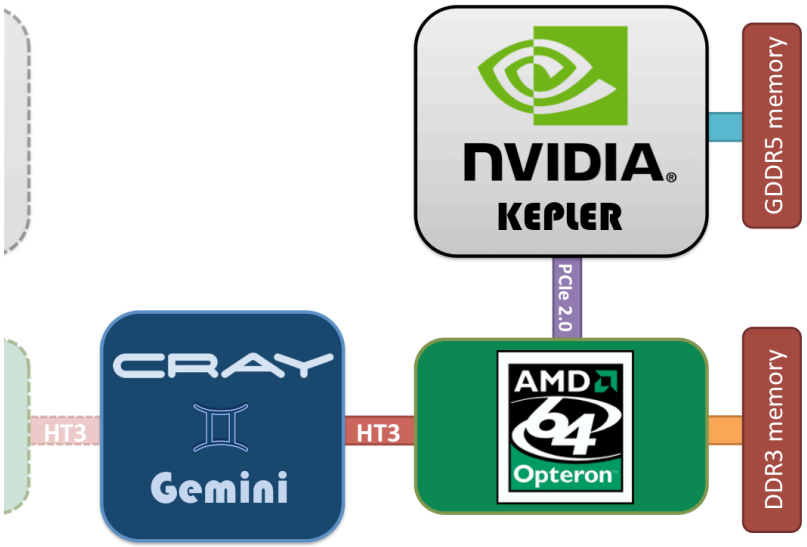
- Co-processor acceleration to reduce intra-node communication
  - *Sao, Vuduc, Li (EuroPar'14); Sao, Liu, Vuduc, Li (IPDPS'15)*
  - Offload Schur-complement computation to GPU

Combining 3D algorithm with GPU acceleration

Sao, Vuduc, Li (JPDC preprint, 2018)

- Co-processor acceleration to reduce intra-node communication
  - *Sao, Vuduc, Li (EuroPar'14); Sao, Liu, Vuduc, Li (IPDPS'15)*
  - Offload Schur-complement update to GPU
- Empirical study on Cray XK7 (titan @ OLCF)

  Each node: AMD Opteron processor (16 cores) + 1 Nvidia K20X GPU



Combining 3D algorithm with GPU acceleration

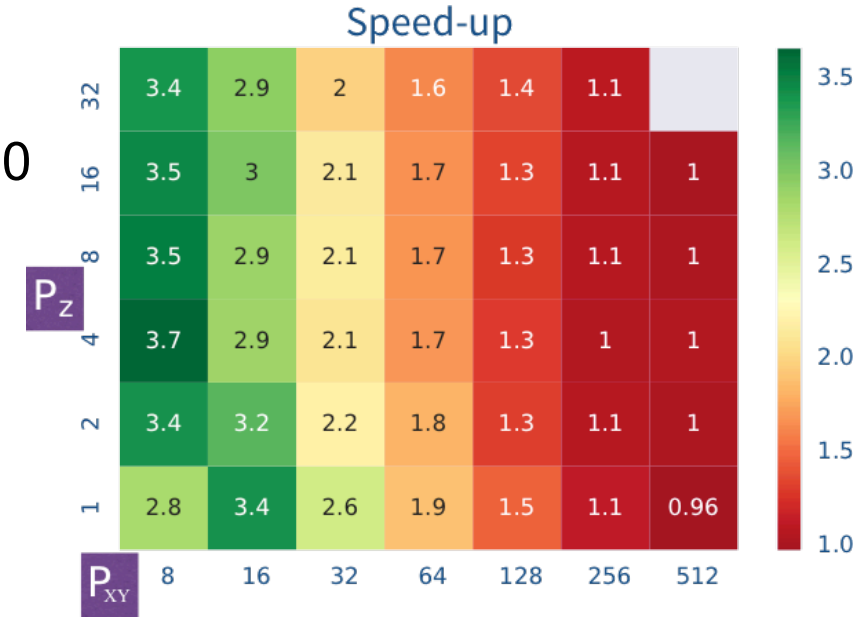
Sao, Vuduc, Li (JPDC preprint, 2018)

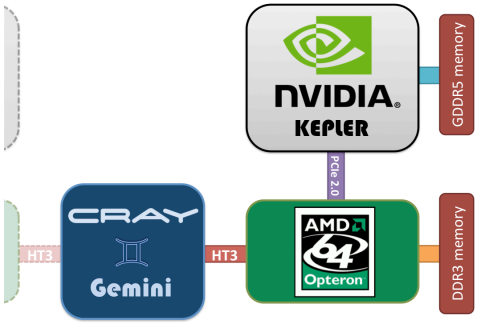
- Co-processor acceleration to reduce intra-node communication
  - *Sao, Vuduc, Li (EuroPar'14); Sao, Liu, Vuduc, Li (IPDPS'15)*
  - Offload Schur-complement update to GPU
- Empirical study on Cray XK7 (titan @ OLCF)

  Each node: AMD Opteron processor (16 cores) + 1 Nvidia K20X GPU

Speedup of combined 3D-CPU-GPU over 3D-CPU:

nlpkkt80



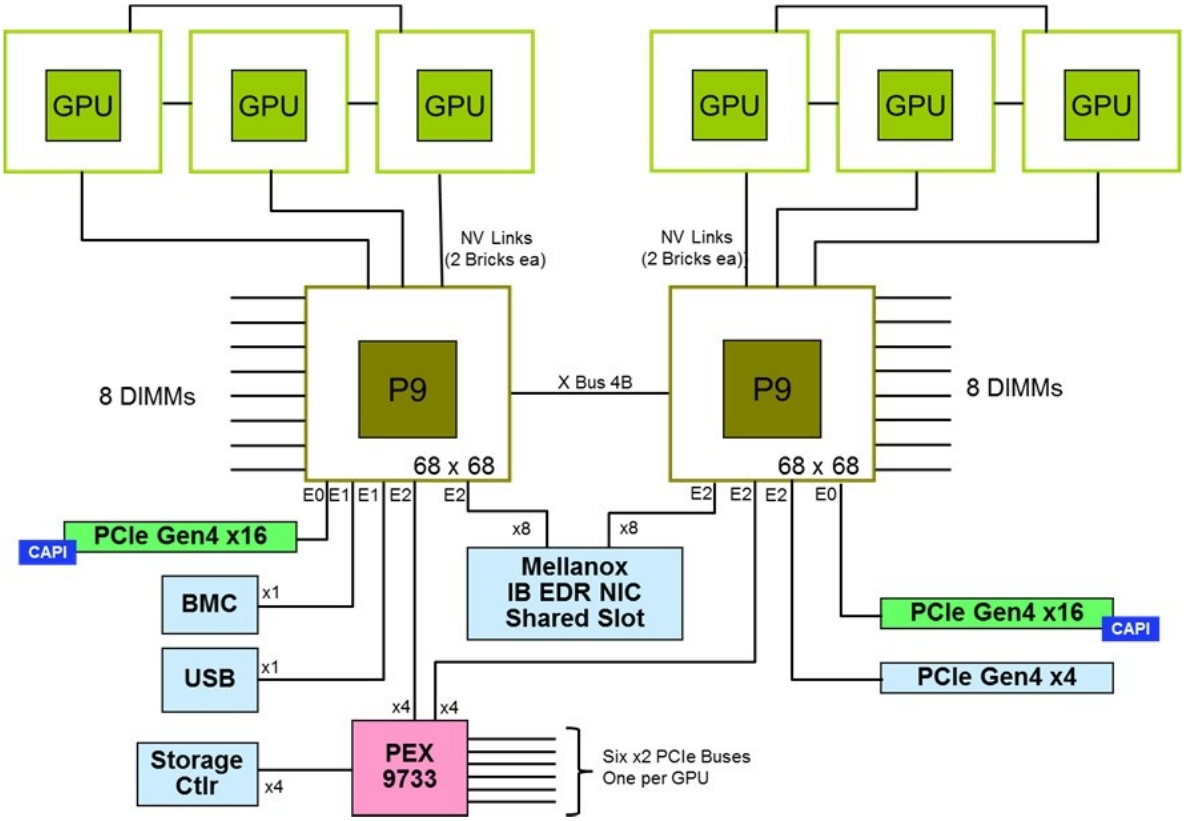


Looking forward: #1 machine in TOP500, June 2018

- summit @ OLCF: 187 PFlop/s

  Each node: 2 IBM Power9 processors (44 cores) + 6 Nvidia V100 GPUs

  Total 2,282,544 cores

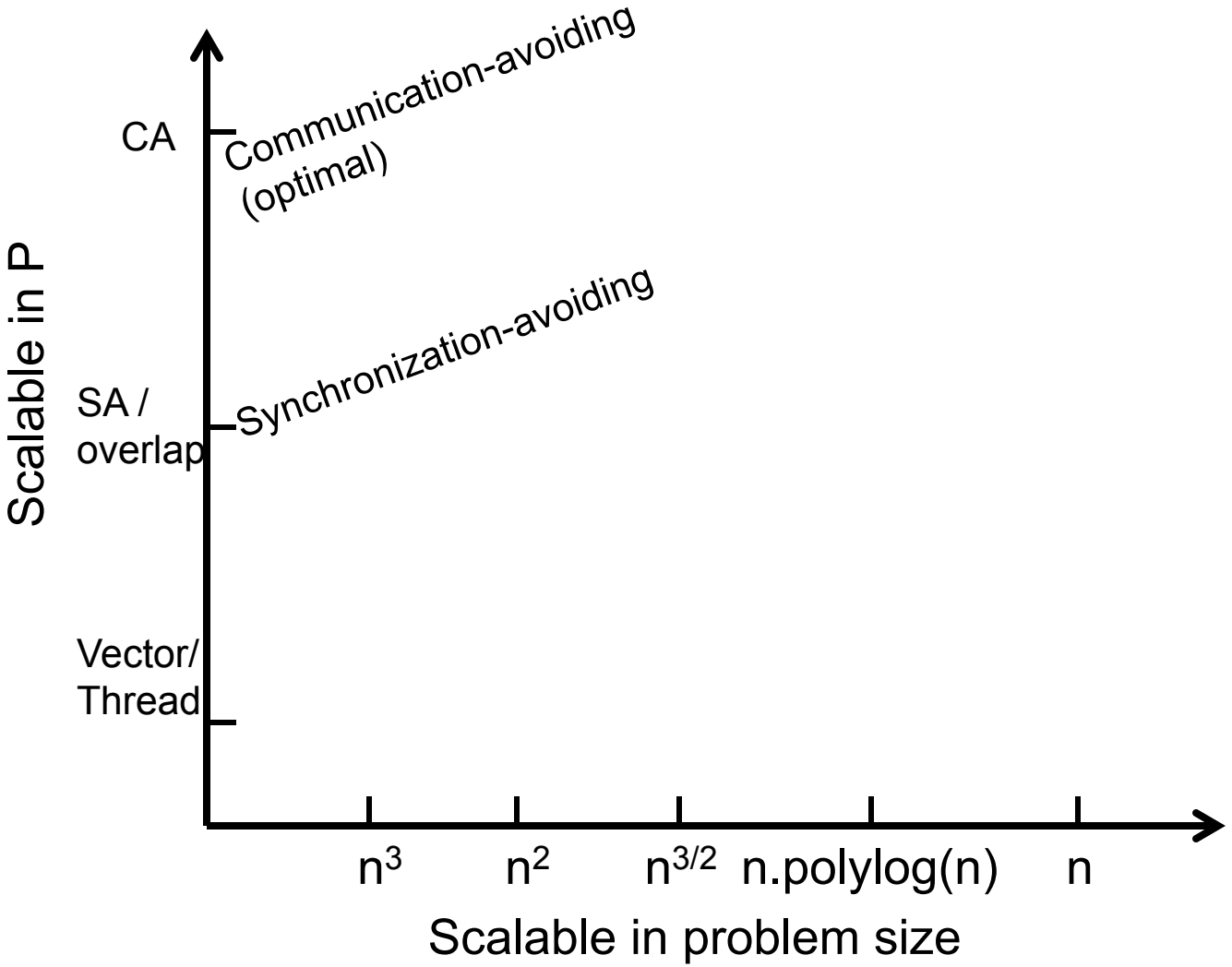


Final notes

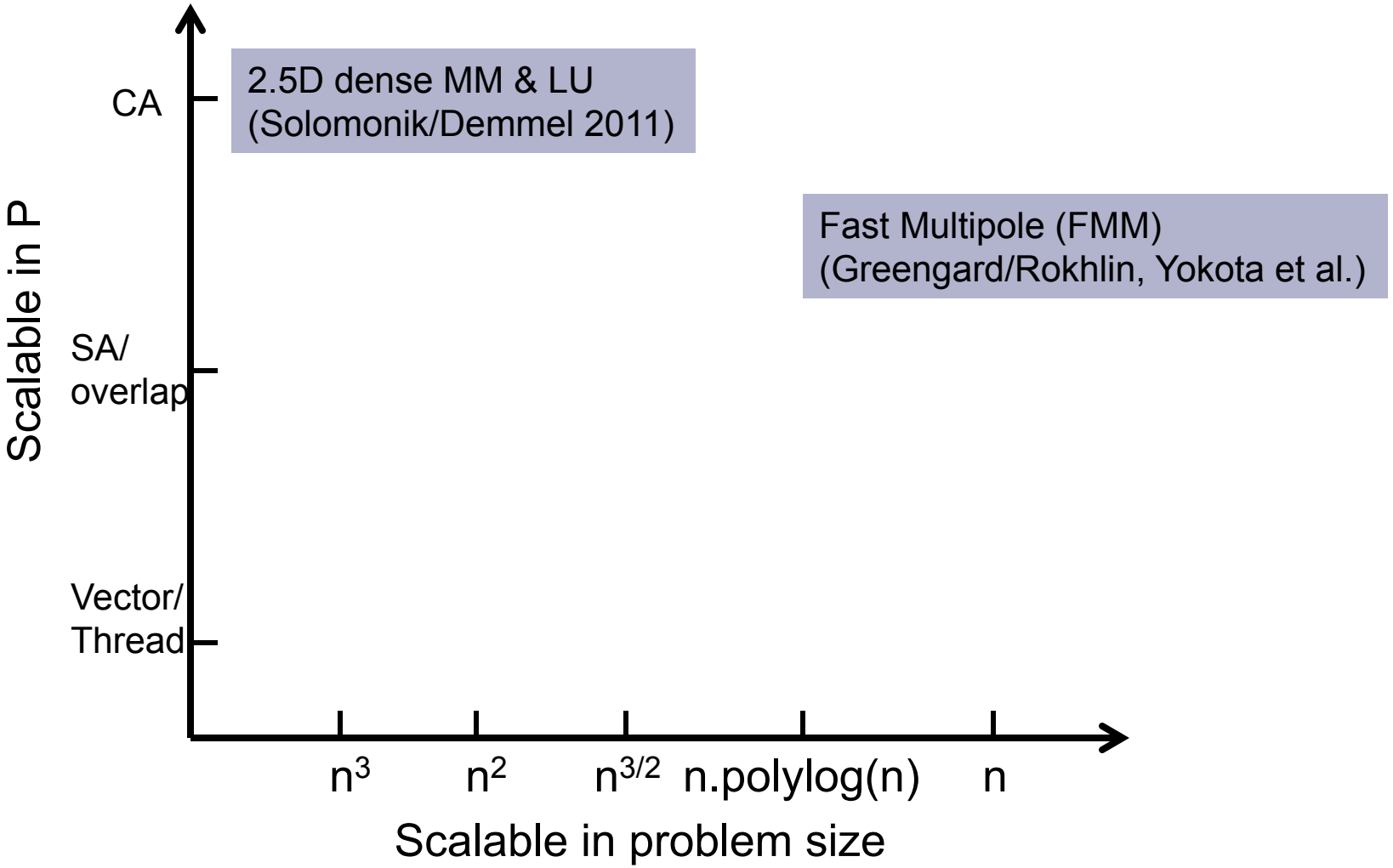
- Hybridization has become prevalent in algorithm and hardware
  - Needed to deliver scalability and robustness (esp. multiphysics problems)

- Hierarchical, multilevel are keys to algorithm scalability

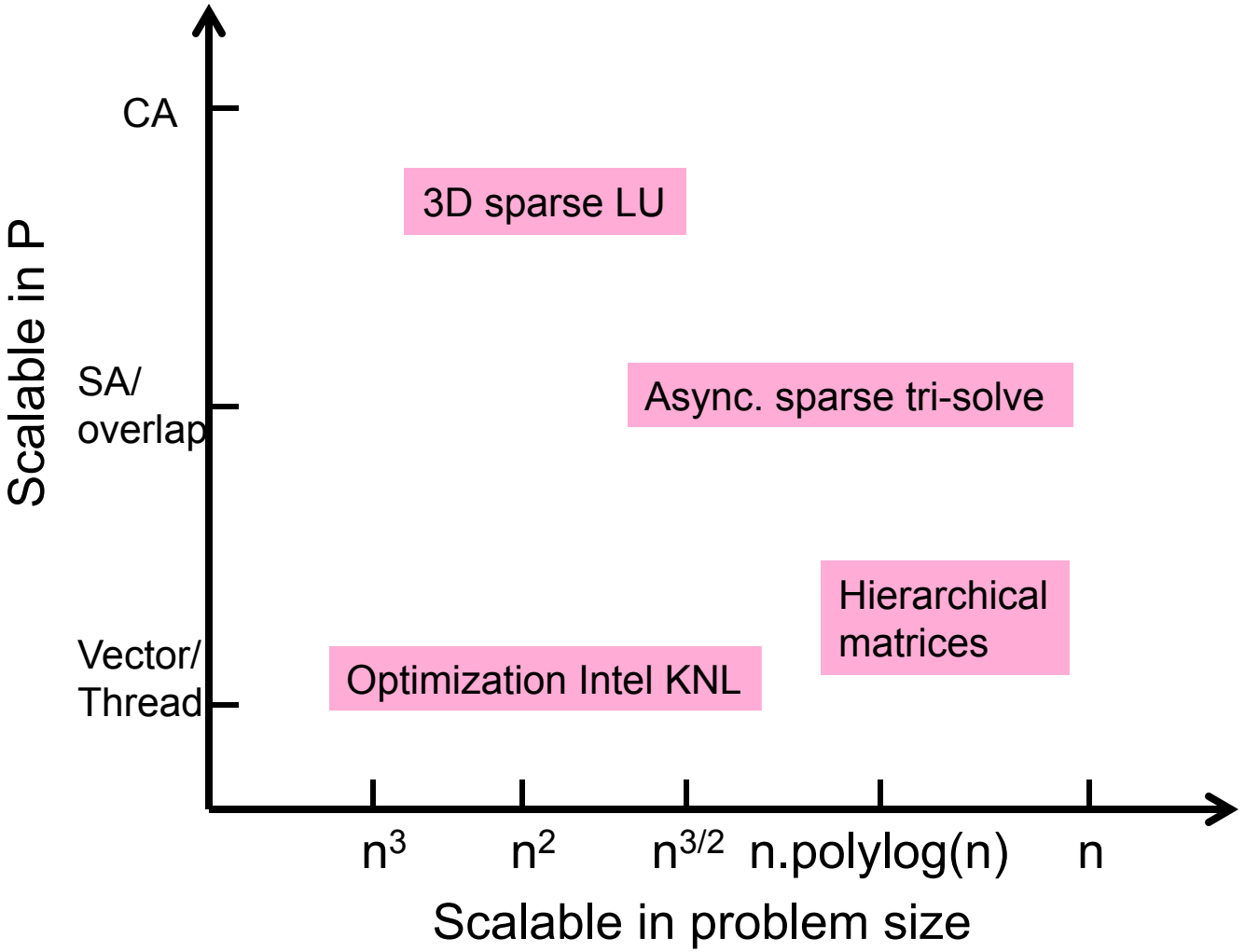
Algorithm scalability



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

THANK YOU

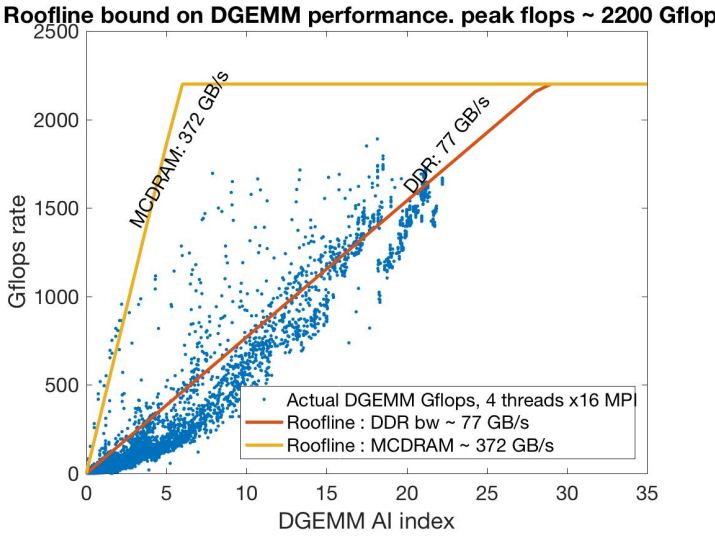
SuperLU factorization KNL optimization

80% faster on single KNL node (release 5.2.0)

- Replace small independent single-threaded MKL GEMMs by large multithreaded MKL GEMMs: **15-20% faster**
- Use new OpenMP features: **10-15% faster**
  - "task parallel" to reduce load imbalance
  - "nested parallel for" to increase parallelism
- Vectorizing Gather/Scatter: **10-20% faster**
  - Hardware support: Load Vector Indexed / Store Vector Indexed

```
#pragma omp simd     // vectorized Scatter
    for (i = 0; i < b; ++i) {
            nzval[ indirect2[i] ] = nzval[ indirect[i] ] -
tempv[i];
    }
```

- Reduce cache misses:
  - Use large pages
  - Malloc with Page-aligned address and CacheLine-aligned address



Y-TUNE collaboration:
Code generation of GEMM corner cases: small, non-square

Summary

- Explore new algorithms that require lower arithmetic complexity, communication, synchronization
  - STRUMPACK: “inexact” direct solver, preconditioner, based on hierarchical low rank structures: HSS, HODLR, etc.
  - SuperLU: new 3D algorithm to reduce communication.

- Refactor existing codes and implement new codes for current and next-generation machines (exascale in a few years)
  - Fully exploit manycore node architectures
    - Vectorization, multithreading, …
    - GPU accelerator
  - Reduce communication and synchronization

- Quarterly release of software

Acknowledgments

This research was supported by the Exascale Computing Project (http://www.exascaleproject.org), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.

Project Number: 17-SC-20-SC

What is a capable exascale computing system?

- Delivers 50× the performance of today's 20 PF systems, supporting applications that deliver high-fidelity solutions in less time and address problems of greater complexity
- Operates in a power envelope of 20–30 MW
- Is sufficiently resilient (perceived fault rate: ≤1/week)
- Includes a software stack that supports a broad spectrum of applications and workloads

Mitigate dense sampling cost

- HSS compression cost = sampling cost + $O(r^2 N)$
- Sampling cost:
  - Traditional matvec: $O(r N^2)$
  - FFT: $O(r N \log N)$ (e.g., Toeplitz)
  - FMM: $O(r N)$

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