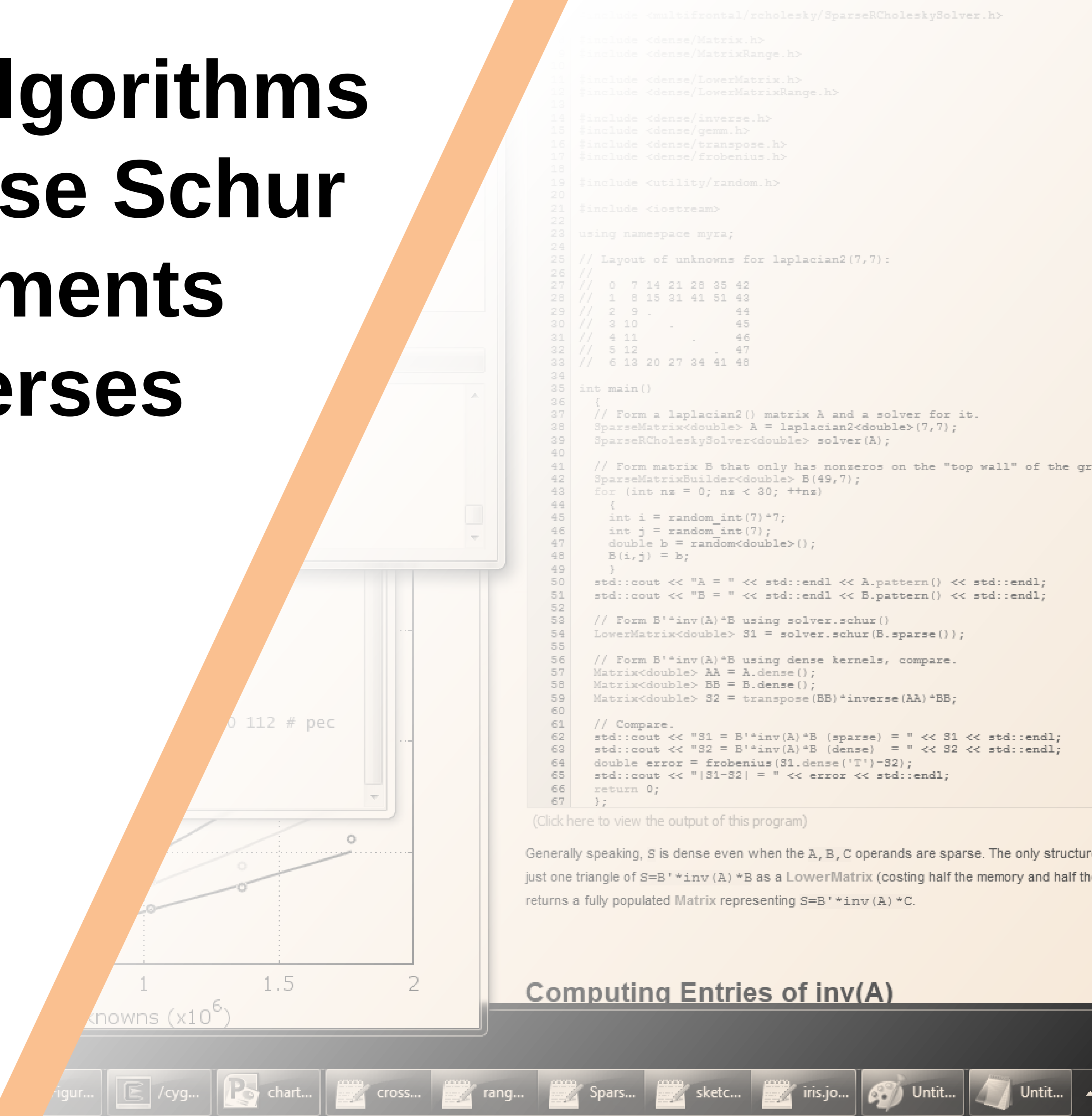
Direct Algorithms for Sparse Schur Complements and Inverses

Dr. Ryan Chilton

MyraMath

myracore.com



Outline

Examine some less common sparse direct algorithms:

- Partial linear solution.
- Schur complements.
- Sampling the inverse operator.

Apply them as “frontends” for low-rank skeletonization:

- Cross approximation.
- Range estimation.
- Ritz projection.

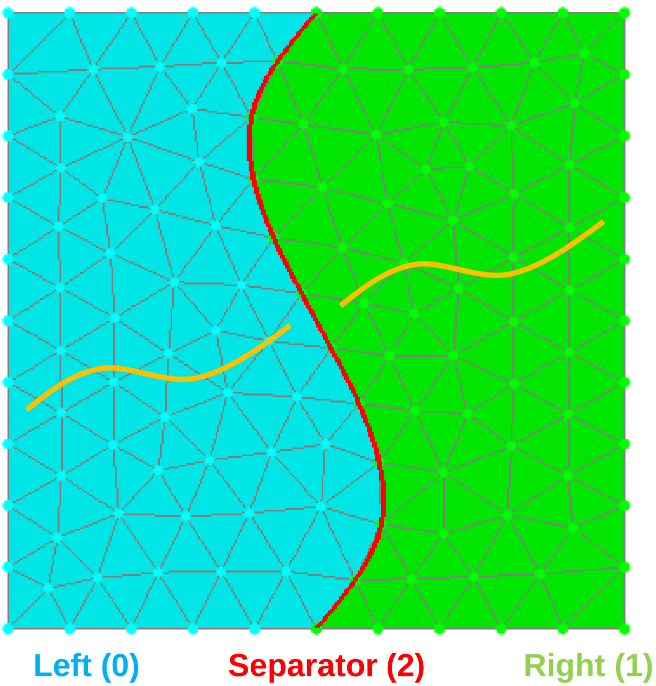
Motivations: fast direct solvers for FE-BI’s and FE-DDM’s.

Refresher: Factor $A=LL^T$

Reorder: Left(0), Right(1), Separator(2). $A_{01} = A_{10}$ = all zero!

Right looking. Factor A_{00}/A_{11}, schur downdate A_{22}, factor.

FEM mesh:

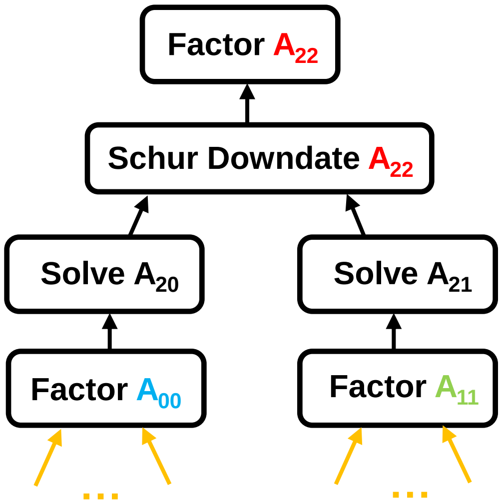


Reordered matrix:

$$\begin{bmatrix} A_{00} & 0 & A_{02} \\ 0 & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{bmatrix}$$

Separator induces these zeroes. They can't fill-in!

Algorithm steps:



Note A_{00} and A_{11} also sparse, apply idea recursively.

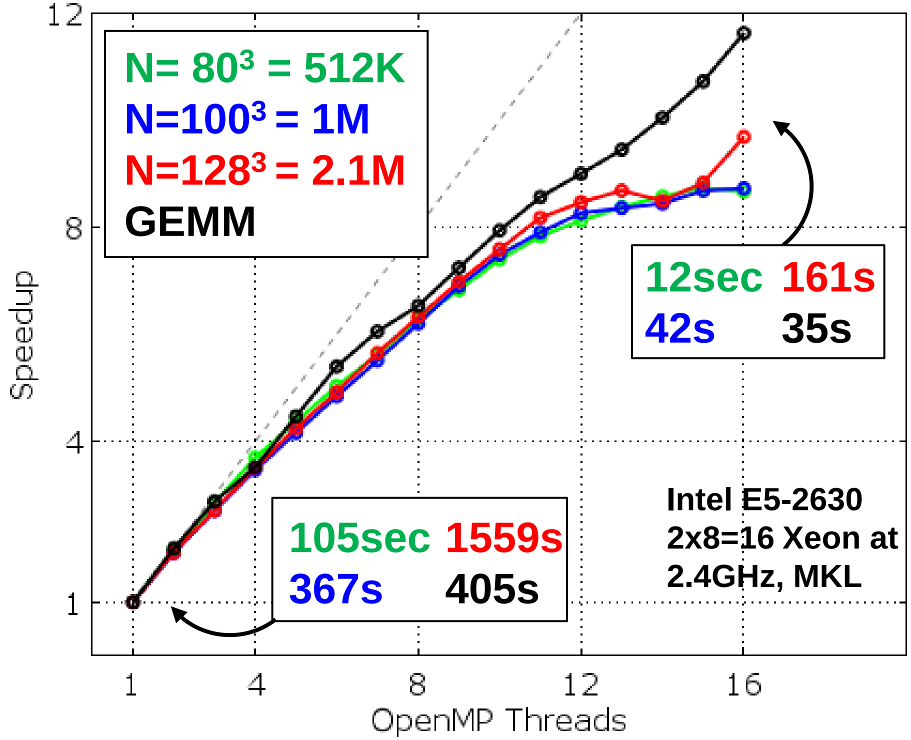
Leads to a tree of operations, eliminating from bottom up.

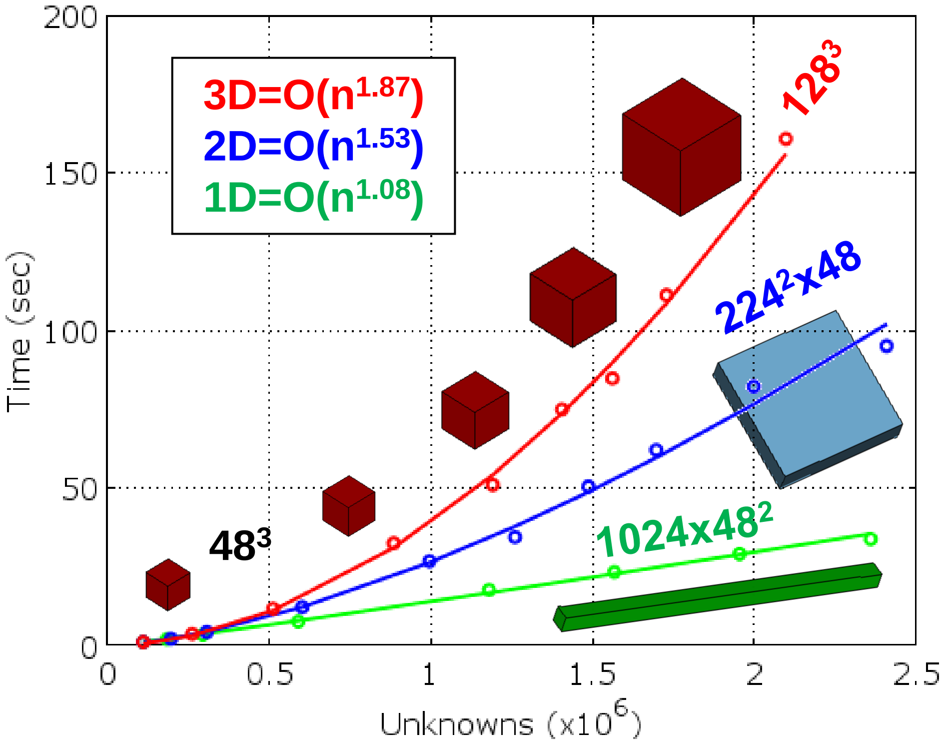
myracore.com

e pluribus rapidum

Selected profiling data.

Example problem under study: I x J x K brick (N = IJK)



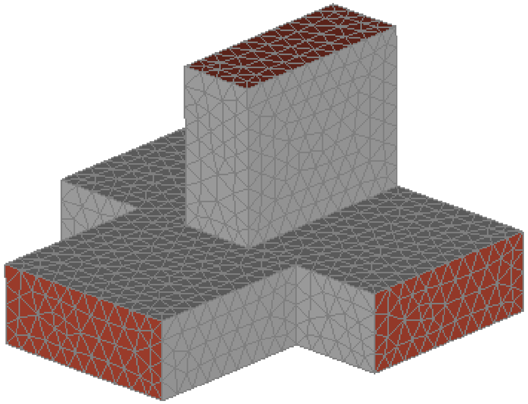


Discrete graph laplacian (7-point): well understood spectrum.

Structured grid: easy to reorder using nested dissection.

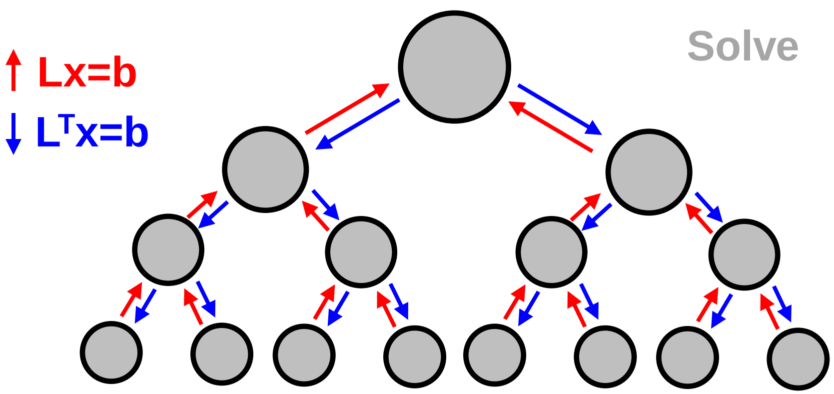
Partial solution $x=R_i^T A^{-1} R_j b$

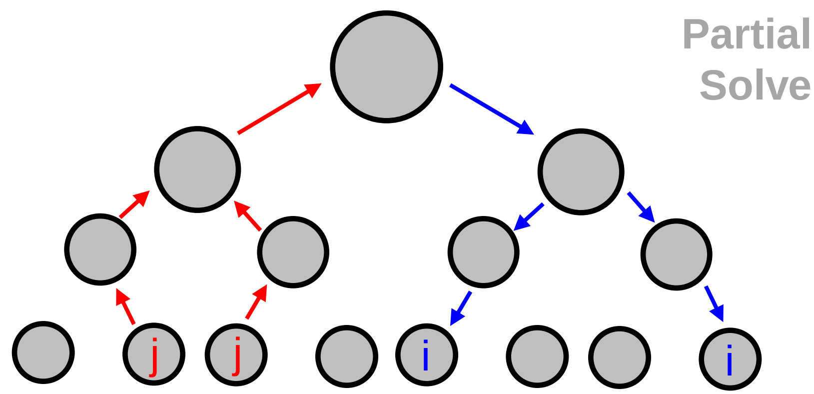
In plain english: only b(j) nonzero, only x(i) is needed.



$$= \frac{1}{\sqrt{2}} \begin{bmatrix} 0 & 0 & +1 & +1 \\ 0 & 0 & +1 & -1 \\ +1 & +1 & 0 & 0 \\ +1 & -1 & 0 & 0 \end{bmatrix}$$

Many engineering QoI's use only *boundary-valued* b and x.





$O(n^{4/3})$ time, like $x=A^{-1}b$. Only $O(n^{2/3})$ space per RHS, not $O(n)$.

Schur complement S=B^{T}A^{-1}B

Concept: form "saddle system" of A and B, then "quit" early.

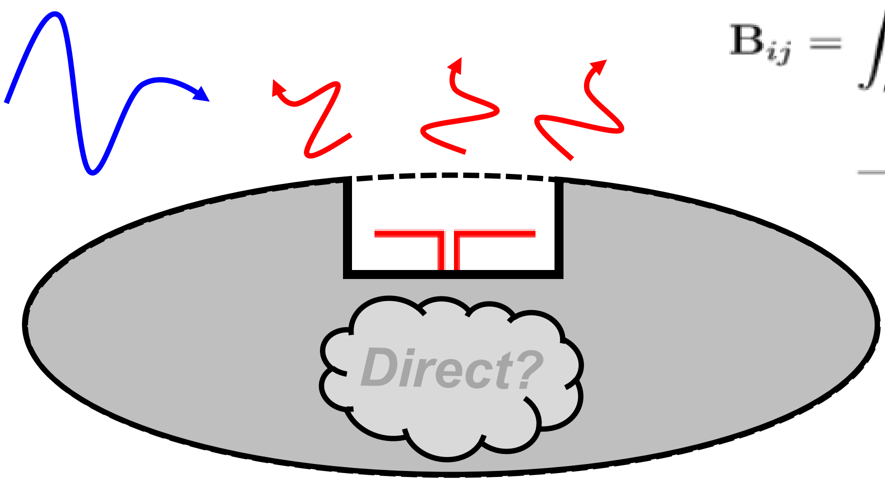
$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}^T & 0 \end{bmatrix} = \begin{bmatrix} \mathbf{L} & 0 \\ (\mathbf{L}^{-1}\mathbf{B})^{\mathbf{T}} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{I} & 0 \\ 0 & \mathbf{B}^T\left(\mathbf{L}\mathbf{L}^T\right)^{-1}\mathbf{B} \end{bmatrix} \begin{bmatrix} \mathbf{L}^T & \mathbf{L}^{-1}\mathbf{B} \\ 0 & \mathbf{I} \end{bmatrix}$$

Arise from FE-BI hybrids, eg scattering from apertures.

$$\mathbf{A}_{ij} = \int_V \left(\nabla \times \vec{w}_i \cdot \mu_r^{-1} \nabla \times \vec{w}_j - k^2 \vec{w}_i \cdot \epsilon_r \vec{w}_j\right) dv$$

$$\mathbf{B}_{ij} = \int_S \nabla \bullet (\hat{n} \times \vec{w}_i) \cdot \int_{S'} \nabla' \bullet (\hat{n}' \times \vec{w}_j) \cdot g(\vec{r}, \vec{r}')\, dS' dS$$

$$- k^2 \int_S (\hat{n} \times \vec{w}_i) \bullet \int_{S'} (\hat{n}' \times \vec{w}_j) \cdot g(\vec{r}, \vec{r}')\, dS' dS$$



$$\begin{bmatrix} \mathbf{A}_{ee} & \mathbf{A}_{em} \\ \mathbf{A}_{me} & \mathbf{A}_{mm} + 2\mathbf{B}_{mm} \end{bmatrix} \begin{bmatrix} \mathbf{e} \\ \mathbf{m} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{h}_{inc} \end{bmatrix}$$

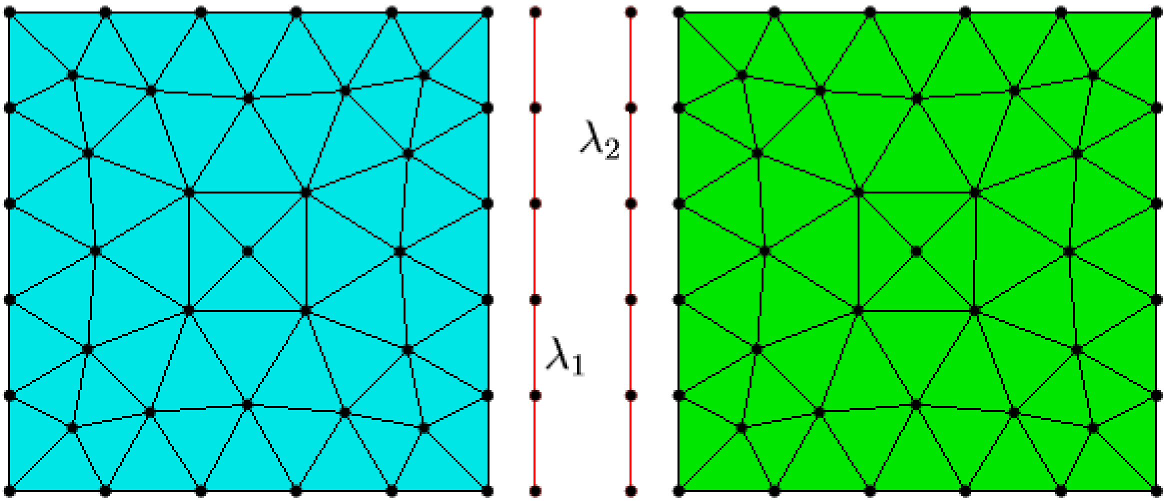
$$\begin{bmatrix} \mathbf{A}_{mm} - \mathbf{A}_{em}^T \mathbf{A}_{ee}^{-1} \mathbf{A}_{em} + 2\mathbf{B}_{mm} \end{bmatrix} \begin{bmatrix} \mathbf{m} \end{bmatrix} = \begin{bmatrix} \mathbf{h}_{inc} \end{bmatrix}$$

Sampling the inverse Z(i,j), $Z=A^{-1}$

Closely related to Schur complement, $Z(i,j) = R_i^T \cdot A^{-1} \cdot R_j$

Arise in FETI/DDM, iterate/exchange fields at boundaries.

$\mathbf{A}_1^{-1}$



$\mathbf{A}_2^{-1}$

$$\begin{bmatrix} \mathbf{I} & \mathbf{I} - 2\alpha \underline{\mathbf{R}_2^T \mathbf{A}_2^{-1} \mathbf{R}_2} \\ I - 2\alpha \underline{\mathbf{R}_1^T \mathbf{A}_1^{-1} \mathbf{R}_1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} 2\alpha R_1^T \mathbf{A}_1^{-1} \mathbf{f}_1 \\ 2\alpha R_2^T \mathbf{A}_2^{-1} \mathbf{f}_2 \end{bmatrix}$$

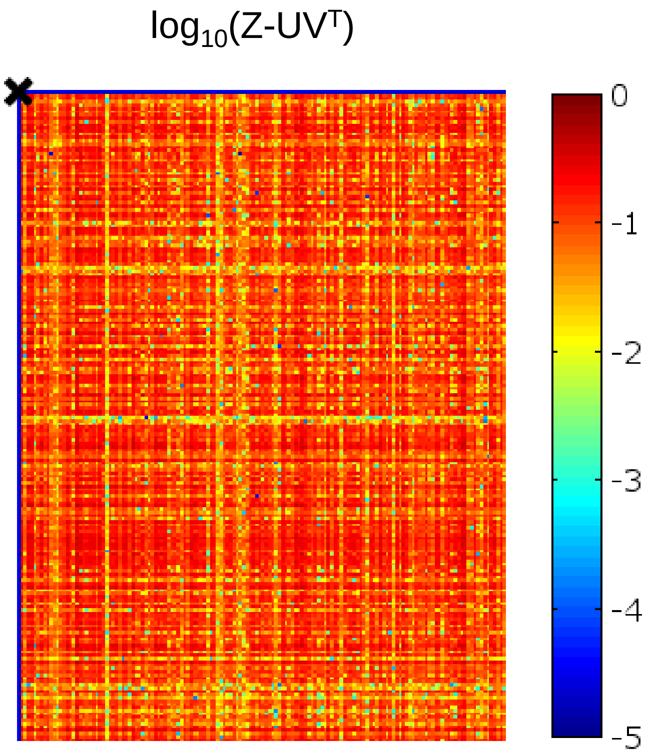
Direct?

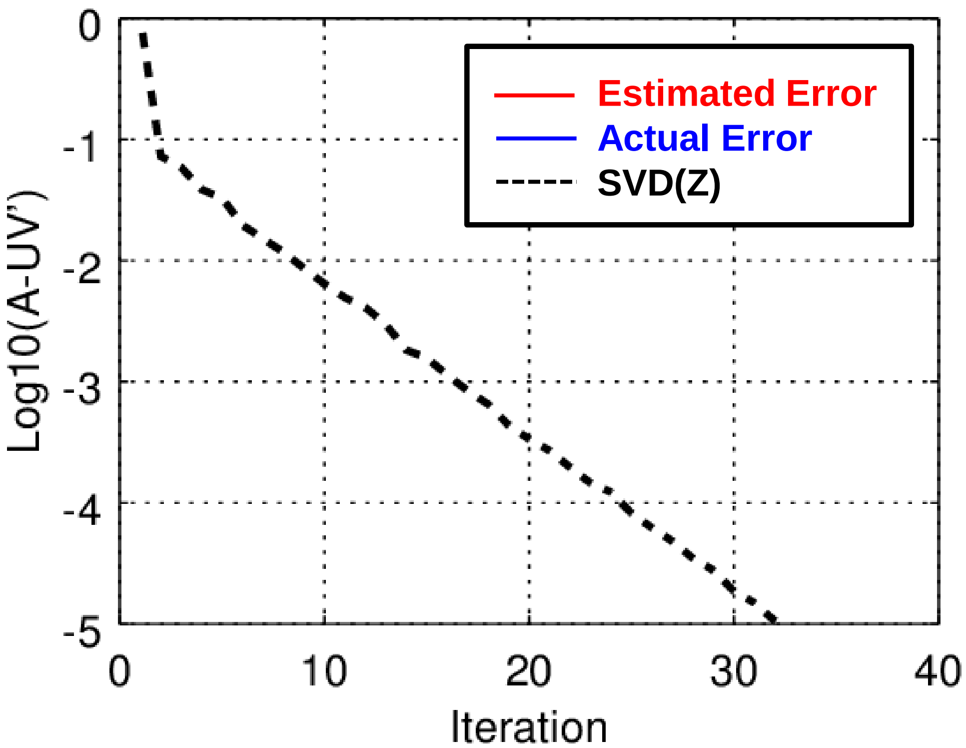
Scatter, solve, gather. Scatter, solve, gather.

Tabulating Z(i,j) opens up reuse/preconditioning options.

Cross Approximating Z(i,j) [1/2]

Alternately sample row/column with largest error modulus.



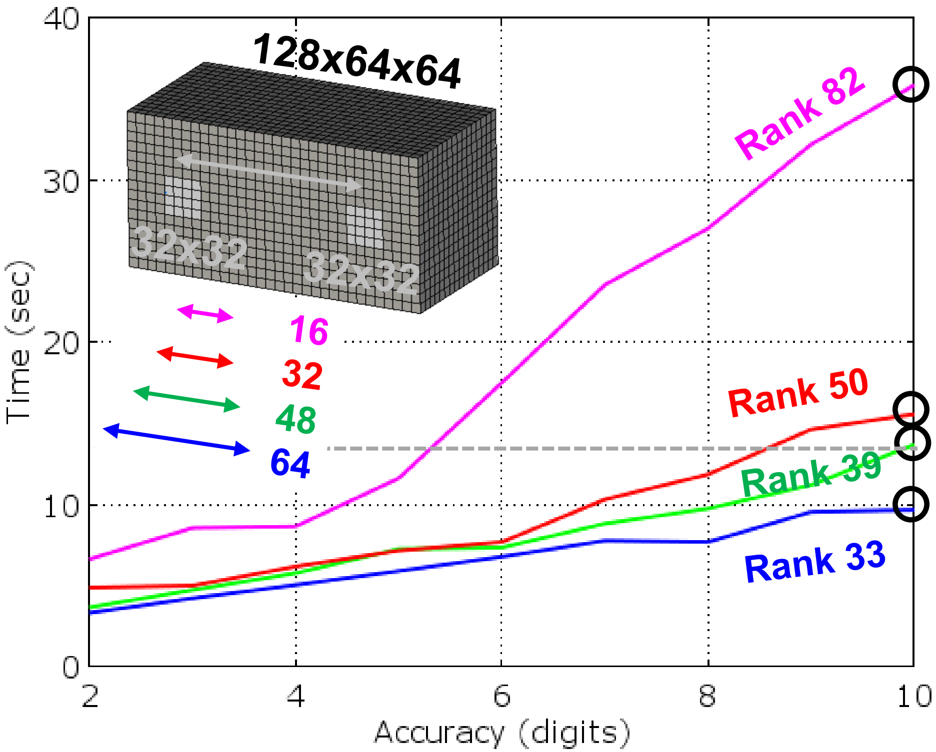


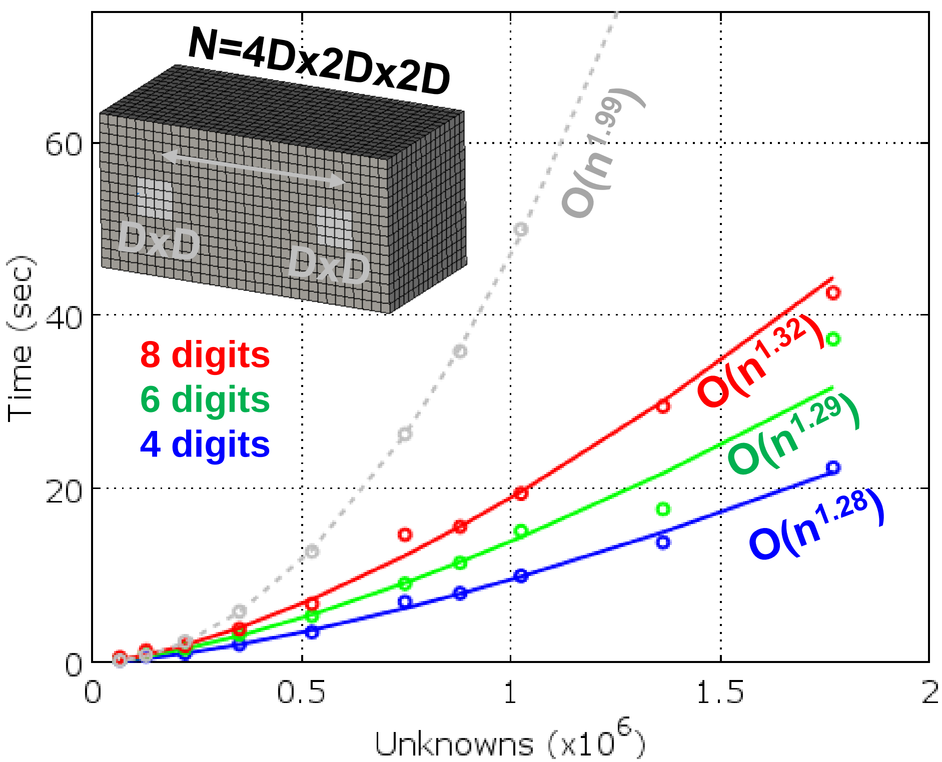
Key idea: partialsolve() can efficiently extract rows/columns:

```
c = Z([i],j) = solver.partialsolve([i],j,x=1.0,'Left')
r = Z(i,[j]) = solver.partialsolve(i,[j],x=1.0,'Right')
```

Cross Approximating Z(i,j) [2/2]

Beats solver.inverse() at large N, especially at low rank/tol.





But in parallel the gap narrows, BLAS3 vs BLAS1 effects.

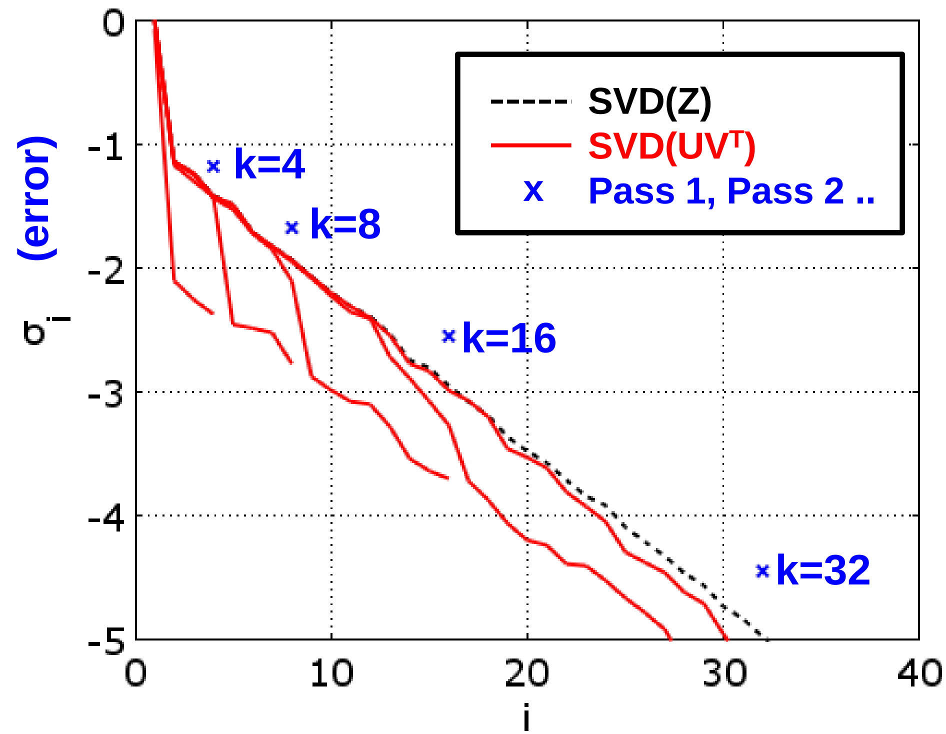
Range estimation of Z(i,j) [1/2]

Apply action of Z to random vectors X, form image Y=ZX.

If Z has rapidly decaying σ's, Y probably spans range(Z).

```
// Find Q = span(Z)
X = rand(Z.cols,k)
Y = Z.apply(X)
[Q,R,π] = QR(Y,0)

// Build k-SVD from Q
W = Z'.apply(Q)
[U,Σ,V] = svd(W,0)
Z≈(QU)·Σ·(V)
```

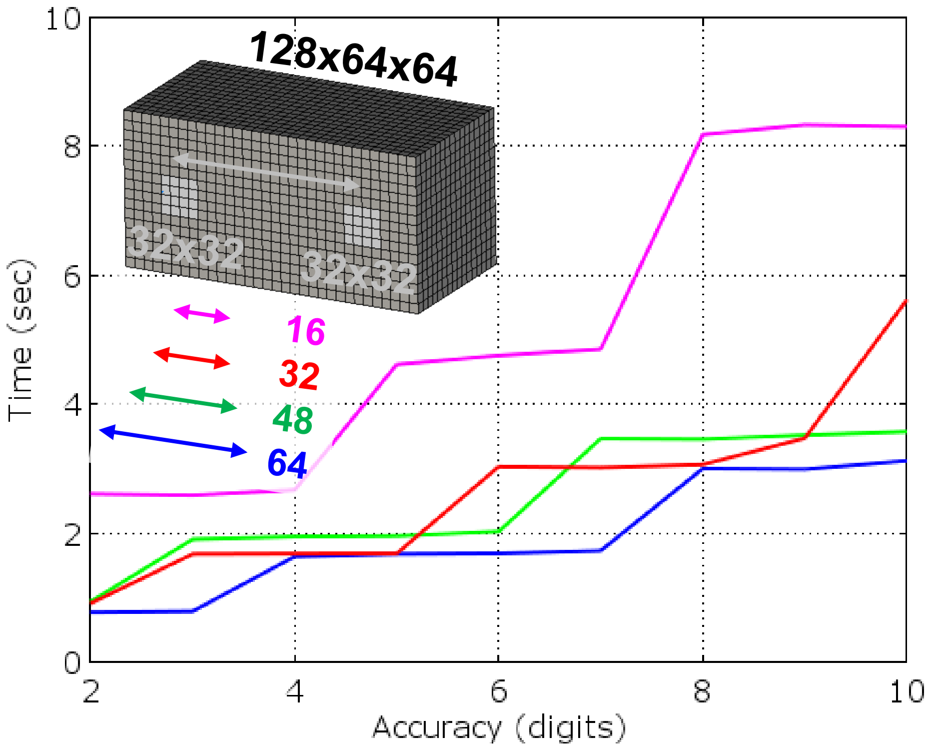


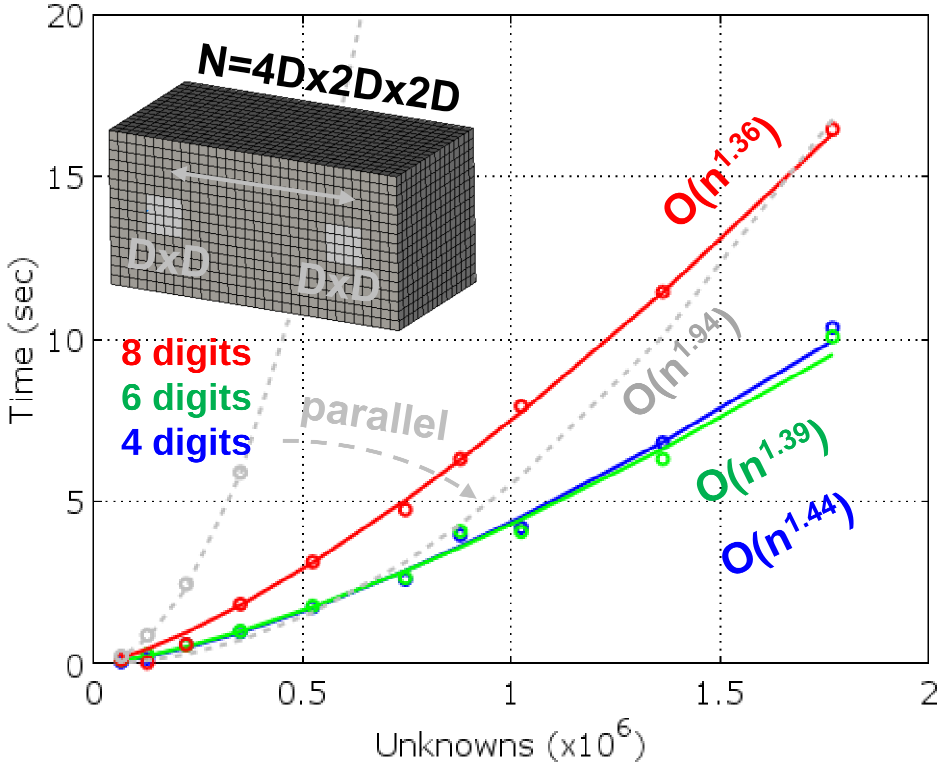
Key idea: partialsolve() can efficiently apply Y=Z(i,j)·X:

```
Y = Z([i],[j])*X = solver.partialsolve([i],[j],X,'Left')
```

Range estimation of Z(i,j) [2/2]

All the same problem instances as before (sizes,shapes).



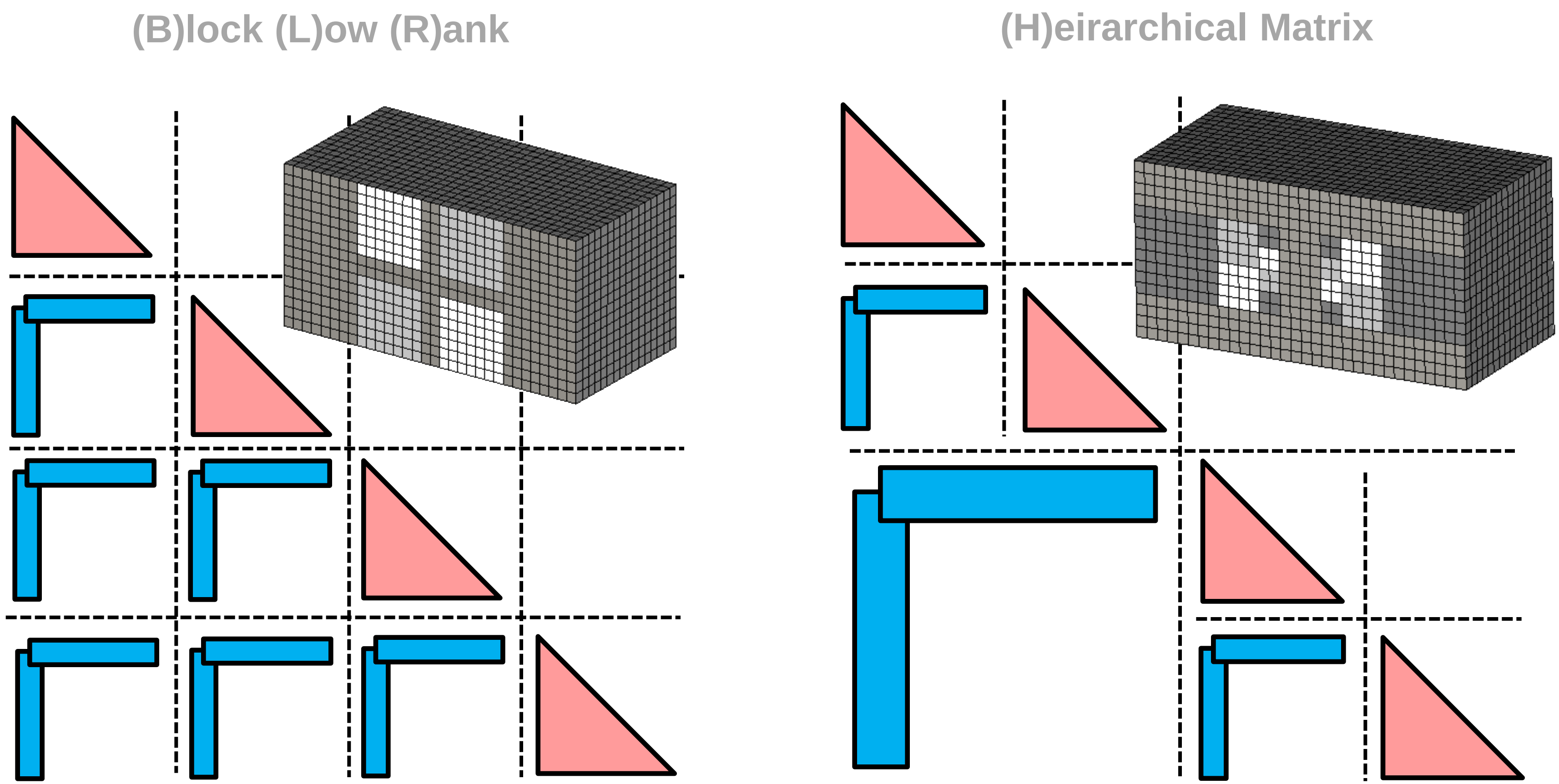


Availability of all forcing data up front leads to speedup.

Can be faster than parallel solver.inverse(), even at modest N.

Ritz Projection of Z(i,j) [1/3]

What about approximating *more* than just one block?

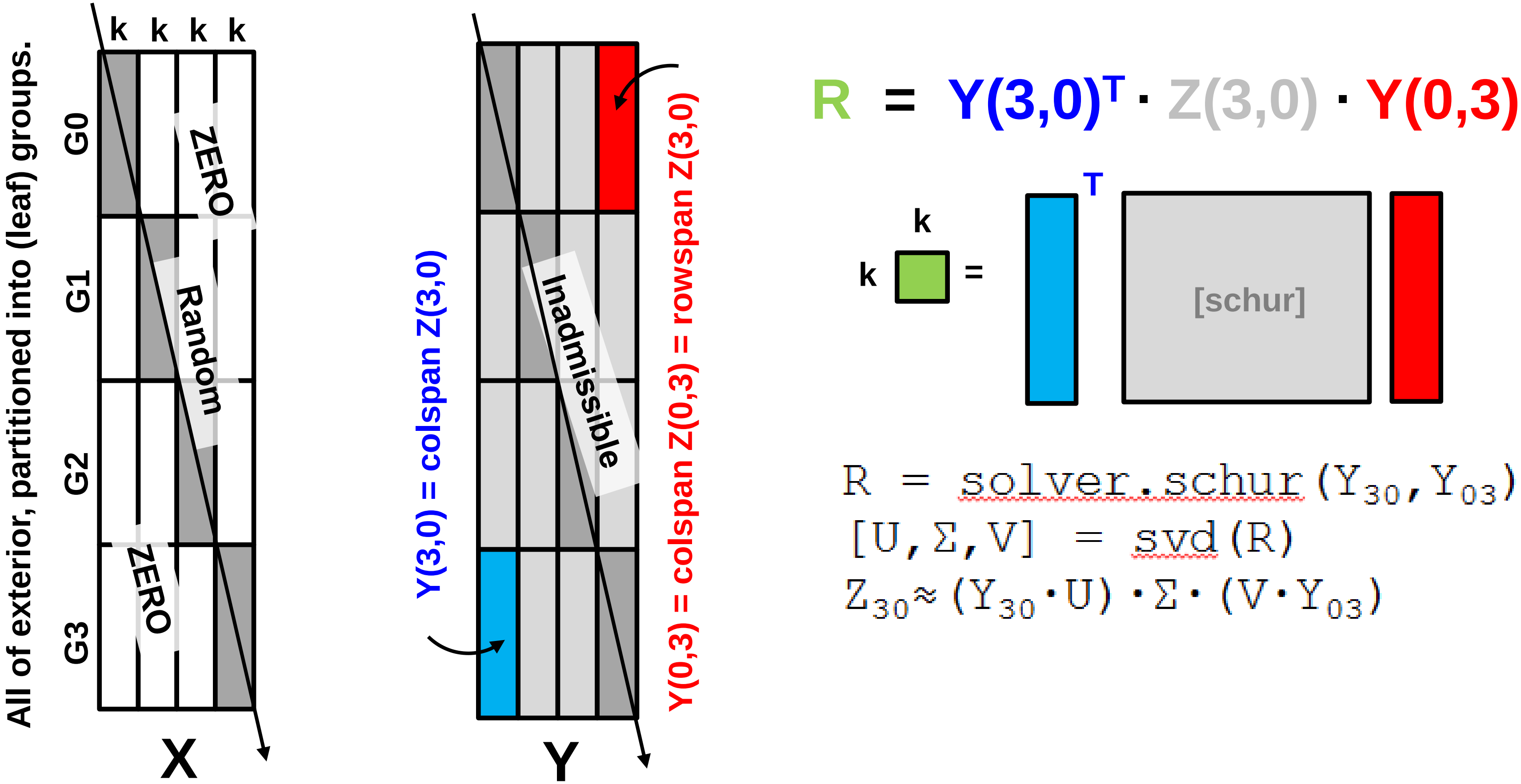


Optimization(BLR)/amortization(H) opportunities do exist.

Ritz Projection of Z(i,j) [2/3]

First pass: find row/column spans using "fat" partialsolve()

```
Y = partialsolve([all],[all],X)
```



$R = \text{solver.schur}(Y_{30}, Y_{03})$

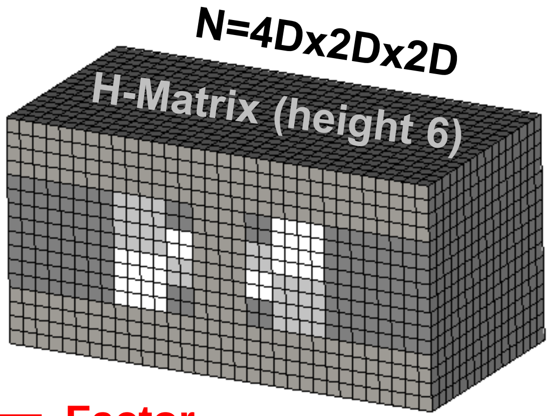
$[U, \Sigma, V] = \text{svd}(R)$

$Z_{30} \approx (Y_{30} \cdot U) \cdot \Sigma \cdot (V \cdot Y_{03})$

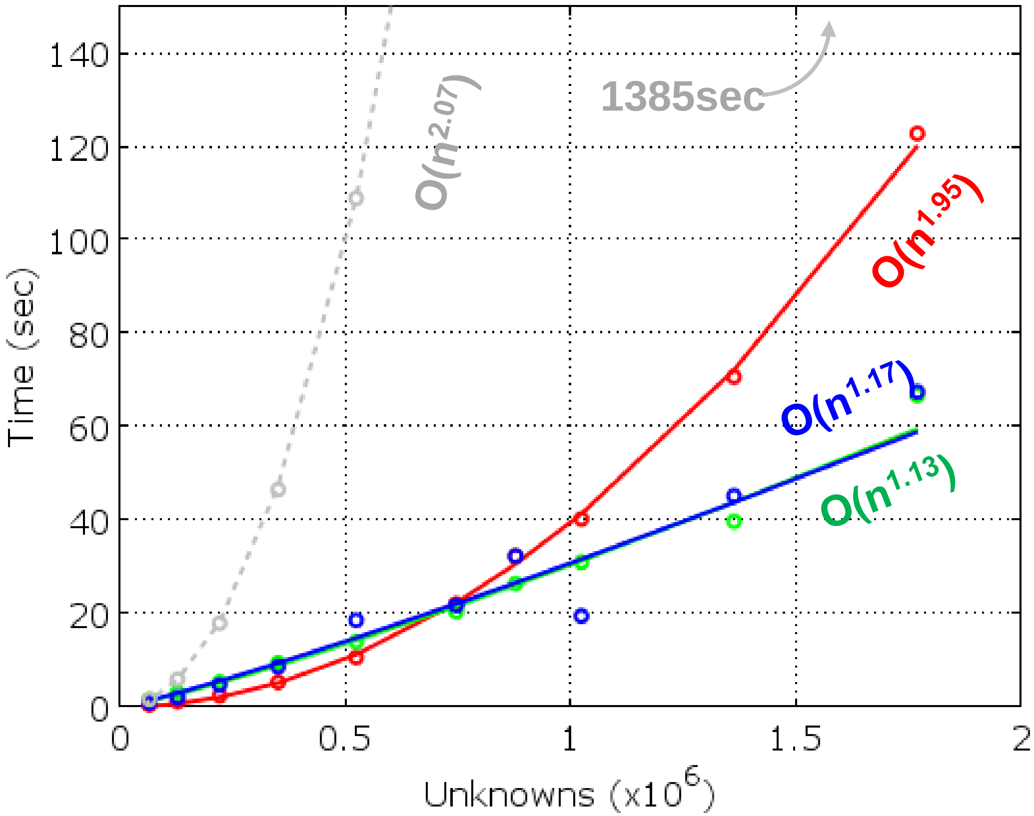
Second pass: Ritz projection using solver.schur(), k-SVD

Ritz Projection of Z(i,j) [3/3]

Fill an H-matrix representation of Z restricted to boundary.



— Factor
— Form Y [partialsolve]
— Form B [schur,QR,SVD]
---- Form Z [inverse]



Algorithm quickly furnishes all (admissible) blocks.

Can form H-matrix of $S=B^TA^{-1}B$ with a few minor changes.

Wrapping Up

Examined several uncommon sparse direct algorithms:

Partial linear solution: $x=R_i^T A^{-1} R_j b$ (sparse b, sifted x)

Schur complements: $B^T A^{-1} B$, $B^T A^{-1} C$, all sparse

Sampling the inverse operator: $Z(i,j) = R_i A^{-1} R_j$

Used them as "frontends" for low-rank/skeletonization:

Cross approximation: partialsolve() can extract row/column

Range estimation: partialsolve() can apply Z(i,j) quickly

Ritz projection: schur()+partialsolve(), amortization over blocks

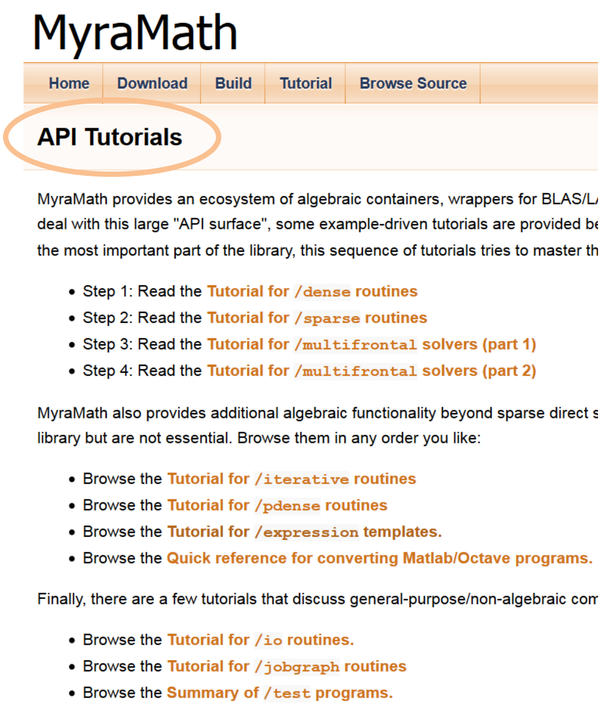
Essential tools for FEBI/DDM methods (sparsity+lowrank).

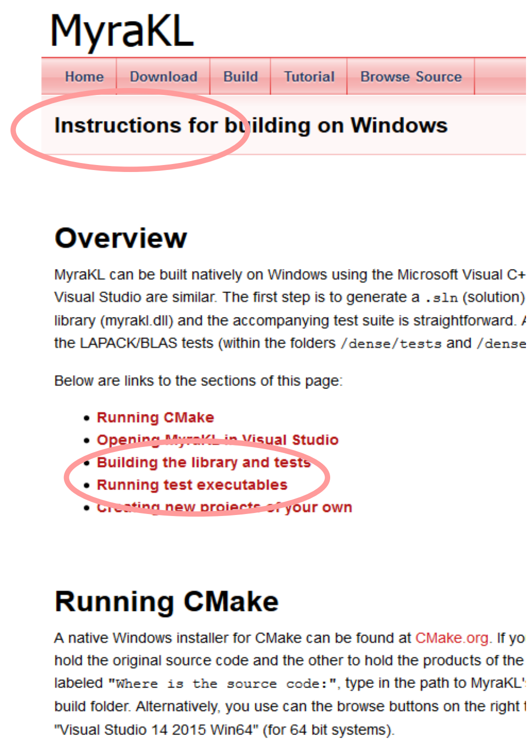
Contact: myracore.com

e pluribus rapidum

MyraMath: sparse factor/solve/schur/inverse/partialsolve.

MyraKL: BLAS/LAPACK API for MyraMath, or use MKL.





Free software (GPL), or dual license (info@myracore.com)