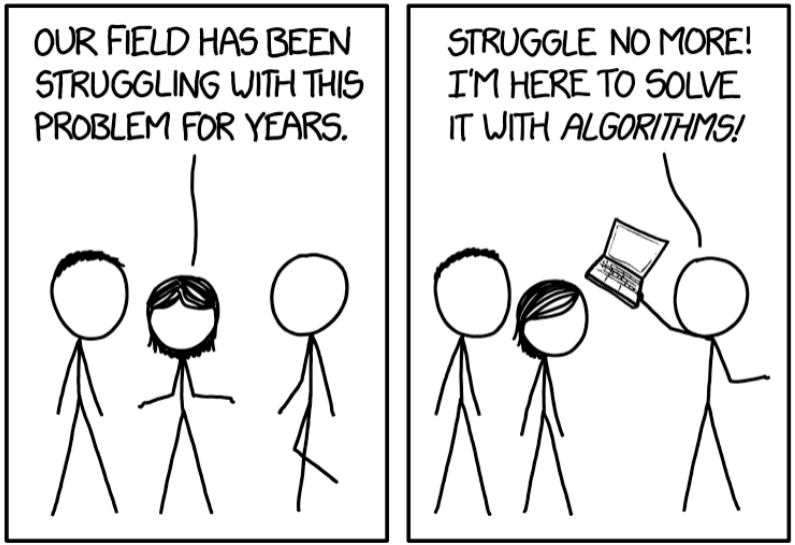
Data-sparse algorithms for structured matrices



Victor Minden
Stanford ICME

Dissertation defense

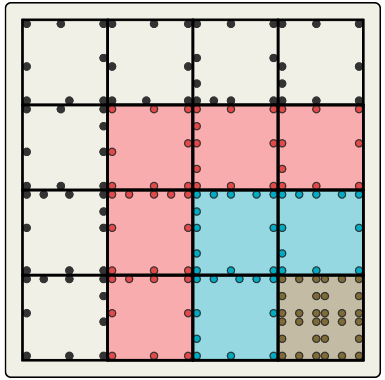
May 15, 2017

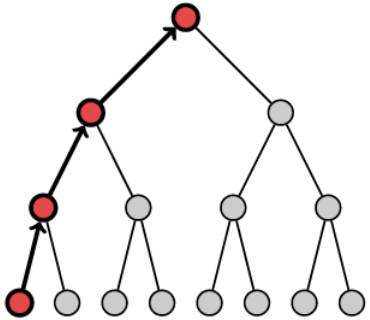
What I am going to talk about today

- My work: fast algorithms exploiting sparsity/related structure

- This talk
  - Part 1: a hierarchical factorization for the inverse operator of fast multipole method systems (e.g., discretizations of integral equations)

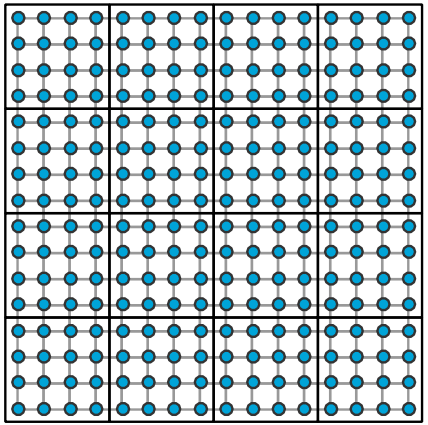
  - Part 2: using hierarchical factorizations to get a linear-complexity framework for Gaussian process maximum likelihood estimation in spatial statistics





Note: all references refer to thesis bibliography

Introduction



Introduction

- Many elliptic differential equations describing physical phenomena can be recast as integral equations with nice kernels

$$a(x)u(x) + \int_{\Omega} b(x)K(x-y)c(y)u(y)\,dy = f(x), \qquad x \in \Omega$$

- Examples
  - Laplace equation
  - Helmholtz equation
  - Lippmann-Schwinger
  - Stokes equation (vector-valued)

$$\Delta v(x) = 0, \qquad x \in \mathfrak{D}$$

$$v(x) = f(x), \quad x \in \partial\mathfrak{D}$$

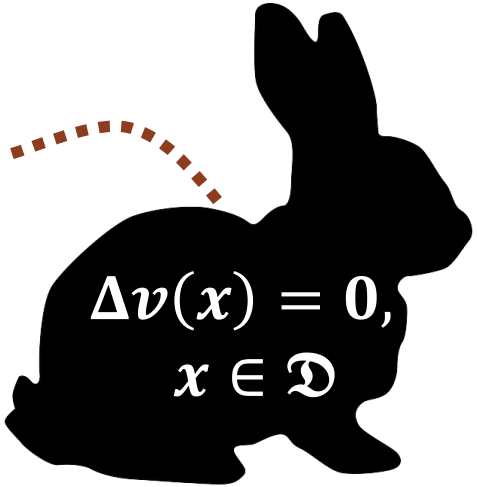
Introduction

- Let's simplify:

$$\int_{\Omega} K(x-y)u(y)\,dy = f(x), \qquad x \in \Omega$$

- $K(x-y) = K(r) = \ldots$
  - $1/|r|$ (3D)
  - $\log(|r|)$ (2D)
  - A derivative of one of these

$$v(x) = f(x),$$
$$x \in \partial\mathfrak{D}$$

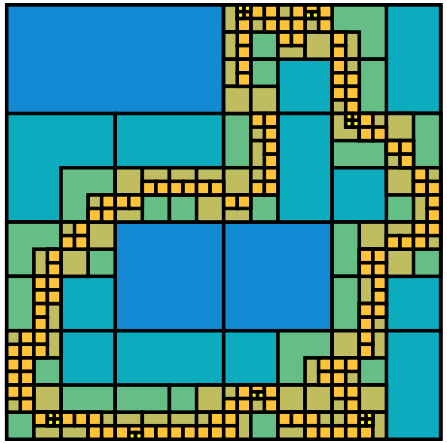


Discretization

- Discretization using standard methods leads to a many-body sum (assume uniform weights):

$$\sum_j K(x_i - x_j)u_j = f_i, \qquad i = 1, \dots, N$$

- Right: a discretization of the boundary of the subdomain, visualized in a quadtree containing the boundary points



Discretization

- Discretization using standard methods leads to a many-body sum (assume uniform weights):

$$\sum_j K(x_i - x_j)u_j = f_i, \qquad i = 1, \dots, N$$

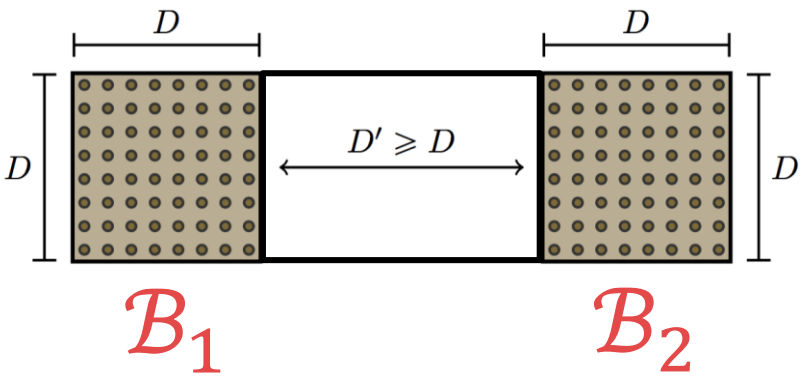
- Two different problems
  1. Given *u*, apply operator to get *f* (naïve cost $\mathcal{O}(N^2)$)
  2. Given *f*, solve system with operator to get *u* (naïve cost $\mathcal{O}(N^3)$)

- Tree codes [Barnes & Hut, 1986] or fast multipole method (FMM) [Greengard & Rokhlin, 1987 &1997] reduce cost of apply to $\mathcal{O}(N \log N)$ or $\mathcal{O}(N)$.

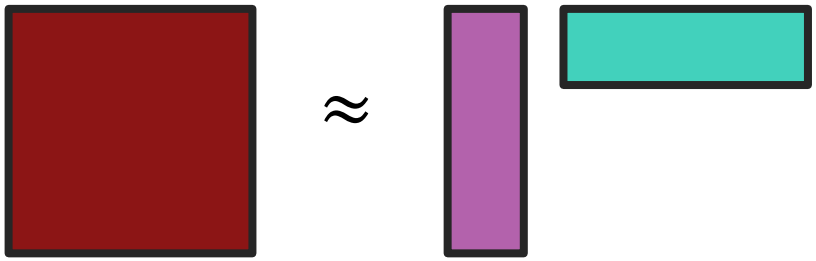
Tree codes and FMM

- For nice kernels from elliptic PDEs, well-separated interactions are compressible as low-rank factorization of off-diagonal block

$$\sum_j K(x_i - x_j)u_j = f_i, \qquad i = 1, \dots, N$$

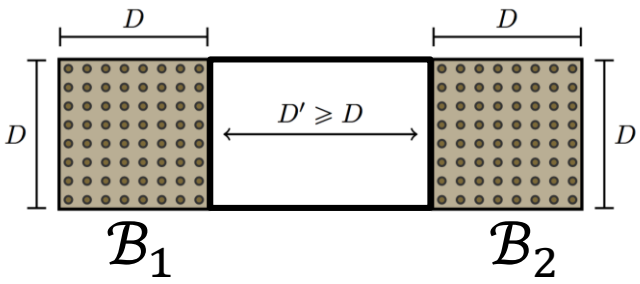


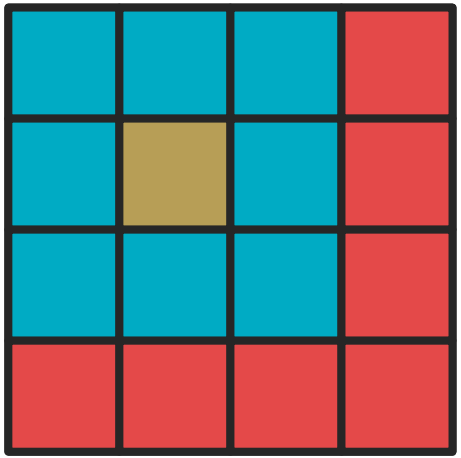
$$\mathrm{K}(\mathcal{B}_1, \mathcal{B}_2) \approx \mathrm{UV}^{\mathrm{T}}$$



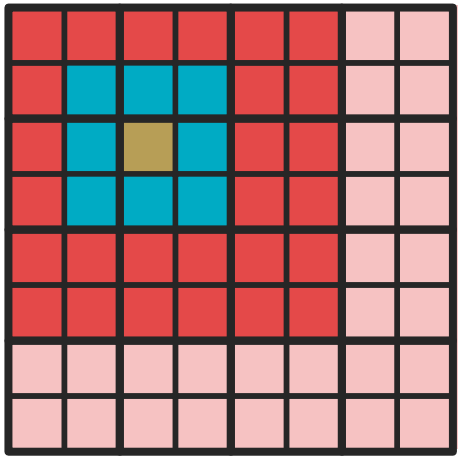
Tree codes and FMM

- Exploit far-field compression idea in a hierarchical fashion to apply operator quickly and approximately
- Below: square domain broken into subdomains recursively





One level of tree



Next lower level of tree

Solving systems

- So, we can use FMM to quickly apply K to u and get f.

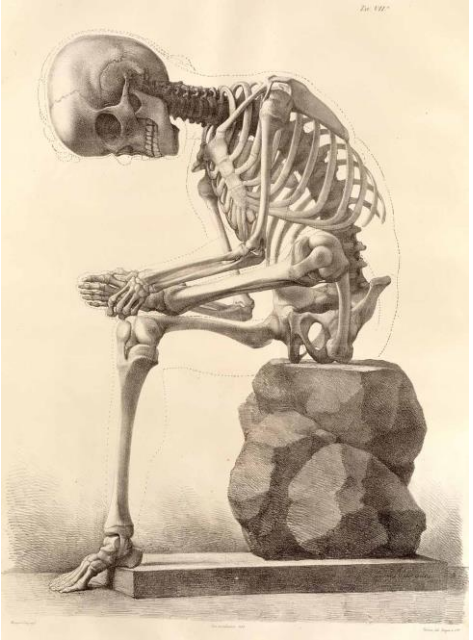
- What about the opposite problem, solving $Ku = f$ for u?

$$\sum_j K(x_i - x_j)u_j = f_i, \qquad i = 1, \dots, N$$

- Iterative methods (conjugate gradient, etc.) are $\mathcal{O}(N)$ or $\mathcal{O}(N \log N)$ per iteration with FMM, but often require many iterations (e.g., first-kind Fredholm equations) so need preconditioners or direct solvers

A recursive skeletonization factorization based on strong admissibility

WITH HO, DAMLE, YING

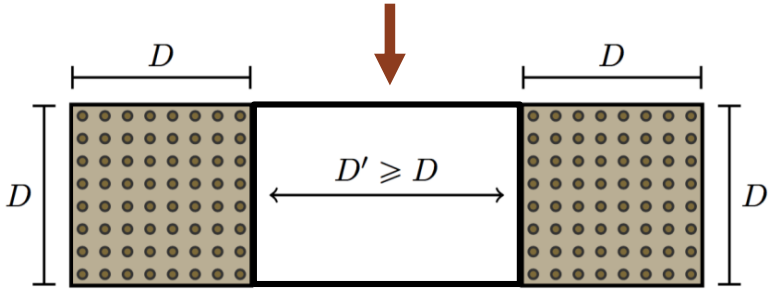


Hierarchical representations from skeletonization

- Want to solve $Ku = f$ efficiently by compressing certain blocks of K
- "Recursive skeletonization" and related literature has led to many nice hierarchical factorizations based on "weak admissibility"

- HSS / HBS matrices
  - [Martinsson & Rokhlin, 2005]
  - [Chandrasekaran et al., 2006 & 2007]
  - [Ho & Greengard, 2012]
  - [Xia et al., 2012]
  - [Gillman et al., 2012]
- HODLR matrices
  - [Martinsson, 2008]
  - [Ambikasaran & Darve, 2013]

Separation not enforced by HSS/HBS/HODLR, giving less compression



Our approach

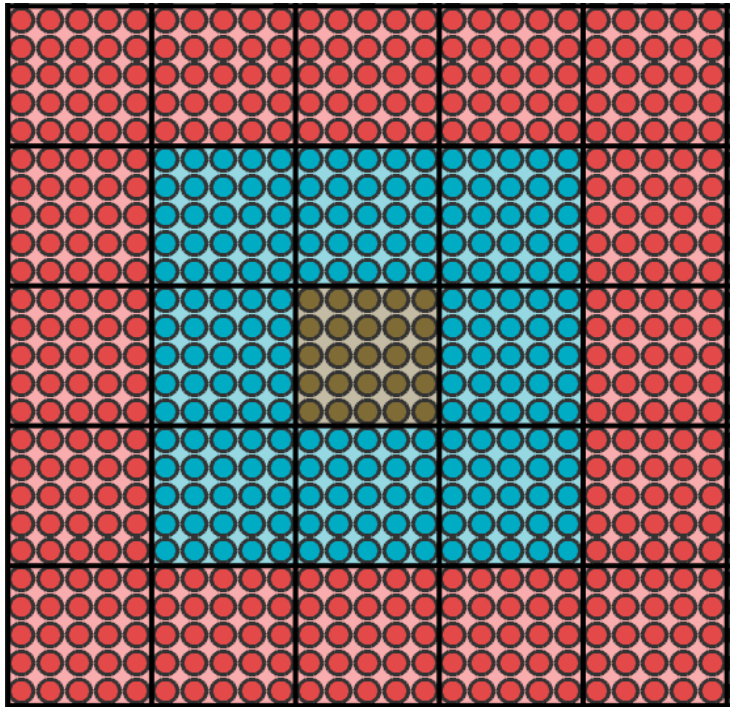
- Adapt recursive skeletonization [Martinsson & Rokhlin, 2005] to compress only well-separated interactions using the multiplicative formulation of skeletonization in [Ho & Ying, 2016].

- Result: approximate factorization of K^{-1} as the product of permutation matrices, block-unit-triangular matrices (with small blocks) and a block-diagonal matrix (with small blocks)

- Simple and fast way to apply K, K^{-1}, $K^{1/2}$, or $K^{-1/2}$, or compute log-det

- Related:
  - IFMM [Coulier et al., 2015] [Ambikasaran & Darve, 2014]
  - Compress-and-eliminate (sparse) [Sushnikova & Oseledets, 2016]
  - Strong $\mathcal{H}$-matrices [Hackbusch & collaborators, 1999-2002]

Bottom (first) level: quadtree and admissible neighbors



- **Brown**: box b
- **Blue**: near-field neighbors of b
- **Red**: far-field neighbors of b (lots of these)

$$\left[\begin{array}{c|c|c} K_{bb} & K_{bn} & K_{bf} \\ \hline K_{nb} & K_{nn} & K_{nf} \\ \hline K_{fb} & K_{fn} & K_{ff} \end{array}\right]$$

Interpolative decomposition

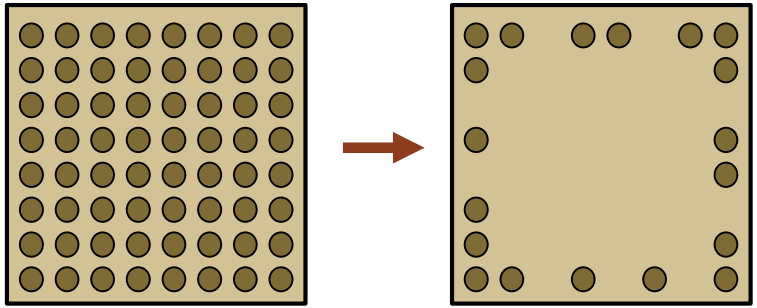
- By assumption, the interactions between **box b** and **far-field neighbors f** are low-rank.

- Compress these with an *interpolative decomposition* [Cheng et al., 2005], where box **b** is partitioned into small **"skeleton set" s** and larger **"redundant set" r**

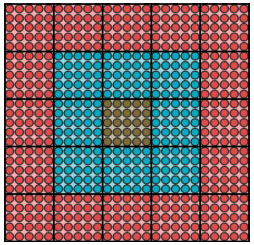
$$\left[\begin{array}{c|c|c} K_{bb} & K_{bn} & K_{bf} \\ \hline K_{nb} & K_{nn} & K_{nf} \\ \hline K_{fb} & K_{fn} & K_{ff} \end{array}\right]$$

$$b = s \cup r$$

$$K_{fr} \approx K_{fs} T$$



Block sparse elimination

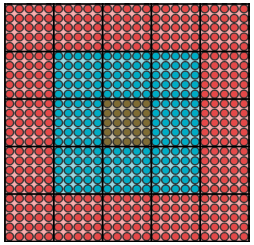


$$\left[\begin{array}{c|c|c} K_{bb} & K_{bn} & K_{bf} \\ \hline K_{nb} & K_{nn} & K_{nf} \\ \hline K_{fb} & K_{fn} & K_{ff} \end{array}\right] = \left[\begin{array}{cc|c|c} K_{rr} & K_{rs} & K_{rn} & K_{rf} \\ K_{sr} & K_{ss} & K_{sn} & K_{sf} \\ \hline K_{nr} & K_{ns} & K_{nn} & K_{nf} \\ \hline K_{fr} & K_{fs} & K_{fn} & K_{ff} \end{array}\right]$$

$$b = s \cup r$$

$$K_{fr} \approx K_{fs} T$$

Block sparse elimination

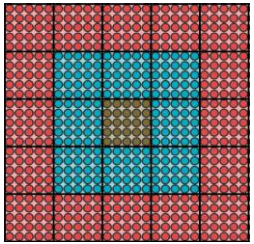


$$\left[\begin{array}{c|c|c} K_{bb} & K_{bn} & K_{bf} \\ \hline K_{nb} & K_{nn} & K_{nf} \\ \hline K_{fb} & K_{fn} & K_{ff} \end{array}\right] \approx \left[\begin{array}{cc|c|c} K_{rr} & K_{rs} & K_{rn} & T^T K_{sf} \\ K_{sr} & K_{ss} & K_{sn} & K_{sf} \\ \hline K_{nr} & K_{ns} & K_{nn} & K_{nf} \\ \hline K_{fs}T & K_{fs} & K_{fn} & K_{ff} \end{array}\right]$$

$$b = s \cup r$$

$$K_{fr} \approx K_{fs}T$$

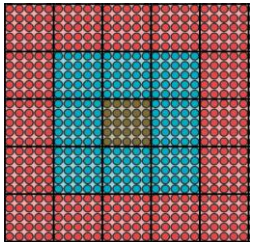
Block sparse elimination



$$\left[\begin{array}{cc|c|c} K_{rr} & K_{rs} & K_{rn} & T^T K_{sf} \\ K_{sr} & K_{ss} & K_{sn} & K_{sf} \\ \hline K_{nr} & K_{ns} & K_{nn} & K_{nf} \\ \hline K_{fs}T & K_{fs} & K_{fn} & K_{ff} \end{array}\right] \longrightarrow \left[\begin{array}{cc|c|c} X_{rr} & X_{rs} & X_{rn} & \\ X_{sr} & K_{ss} & K_{sn} & K_{sf} \\ \hline X_{nr} & K_{ns} & K_{nn} & K_{nf} \\ \hline & K_{fs} & K_{fn} & K_{ff} \end{array}\right]$$

- …and similarly for rows

Block sparse elimination



$$\left[\begin{array}{cc|c|c} X_{rr} & X_{rs} & X_{rn} & \\ X_{sr} & K_{ss} & K_{sn} & K_{sf} \\ \hline X_{nr} & K_{ns} & K_{nn} & K_{nf} \\ \hline & K_{fs} & K_{fn} & K_{ff} \end{array}\right] \longrightarrow \left[\begin{array}{c|c|c} X_{rr} & & \\ \hline & X_{ss} & X_{sn} & K_{sf} \\ \hline & X_{ns} & X_{nn} & K_{nf} \\ \hline & K_{fs} & K_{fn} & K_{ff} \end{array}\right]$$

- …and similarly for rows

After block Gaussian elimination, we have decoupled the redundant points and updated interactions between the skeleton points and near-field neighbors.

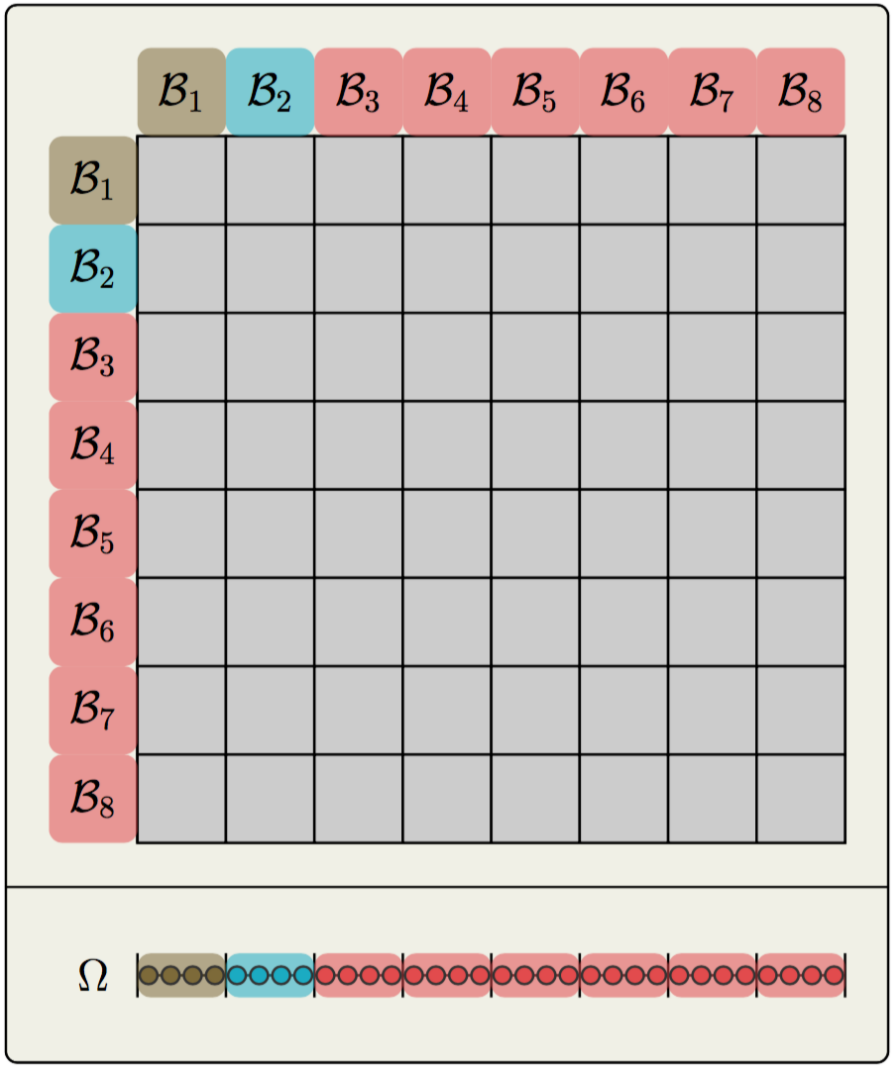
Block sparse elimination

$$\begin{bmatrix} K_{bb} & K_{bn} & K_{bf} \\ K_{nb} & K_{nn} & K_{nf} \\ K_{fb} & K_{fn} & K_{ff} \end{bmatrix} \approx V_b \left[\begin{array}{c|ccc} X_{rr} & & & \\ \hline & X_{ss} & X_{sn} & K_{sf} \\ & X_{ns} & X_{nn} & K_{nf} \\ & K_{fs} & K_{fn} & K_{ff} \end{array}\right] W_b$$

- Approximation error is controlled adaptively
- Skeleton set ***s*** is small
- The factors V_b and W_b are each products of block unit triangular matrices: easy to apply and invert!

$$W_b = \begin{bmatrix} I & X_{rr}^{-1}X_{rs} & X_{rr}^{-1}X_{rn} & \\ & I & & \\ & & I & \\ & & & I \end{bmatrix} \begin{bmatrix} I & & & \\ T & I & & \\ & & I & \\ & & & I \end{bmatrix}$$

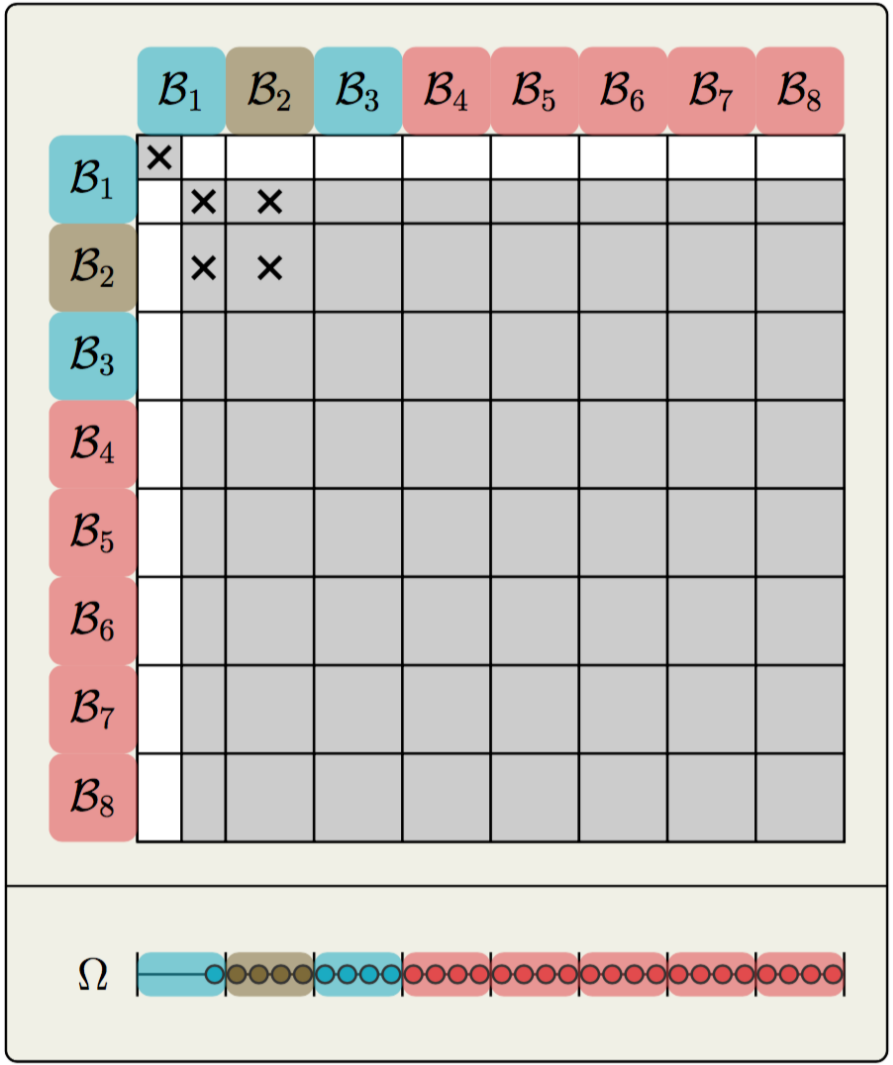
1D
Level 1
Before $\mathcal{B}_1$



The matrix

Points in the domain

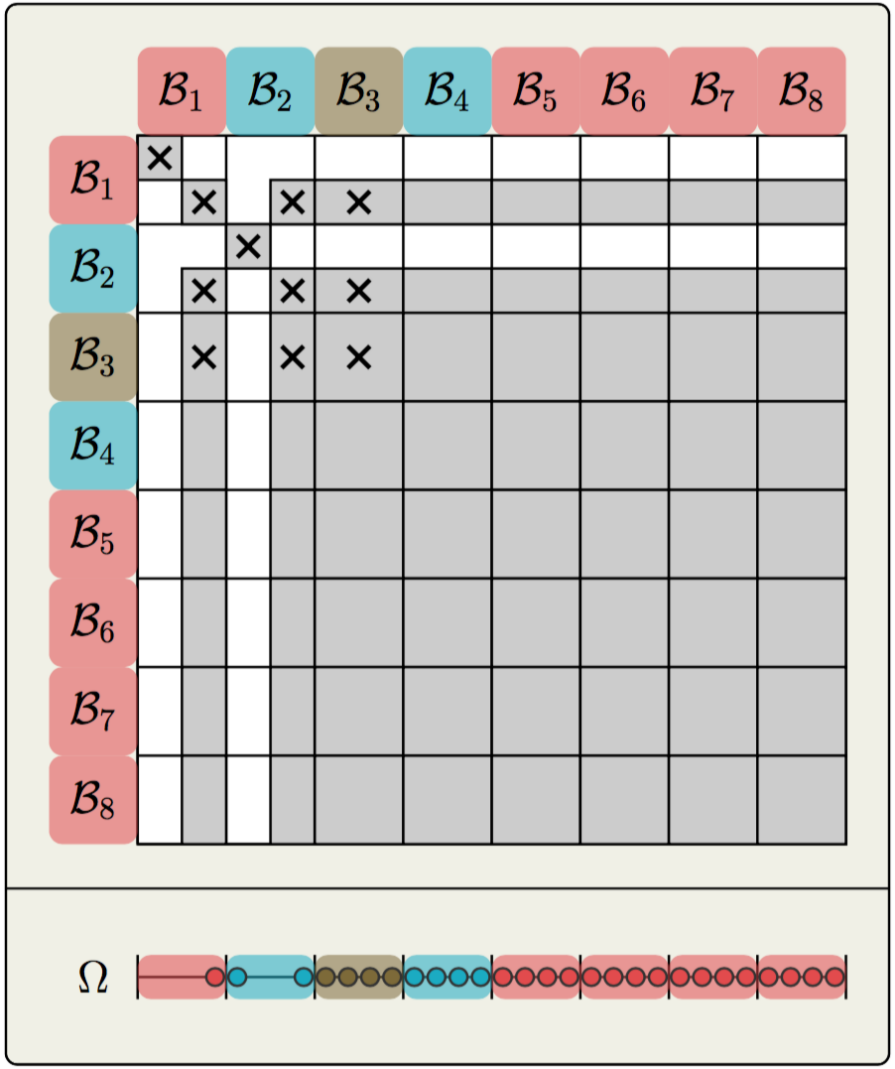
1D
Level 1
Before $\mathcal{B}_2$



The matrix

Points in the domain

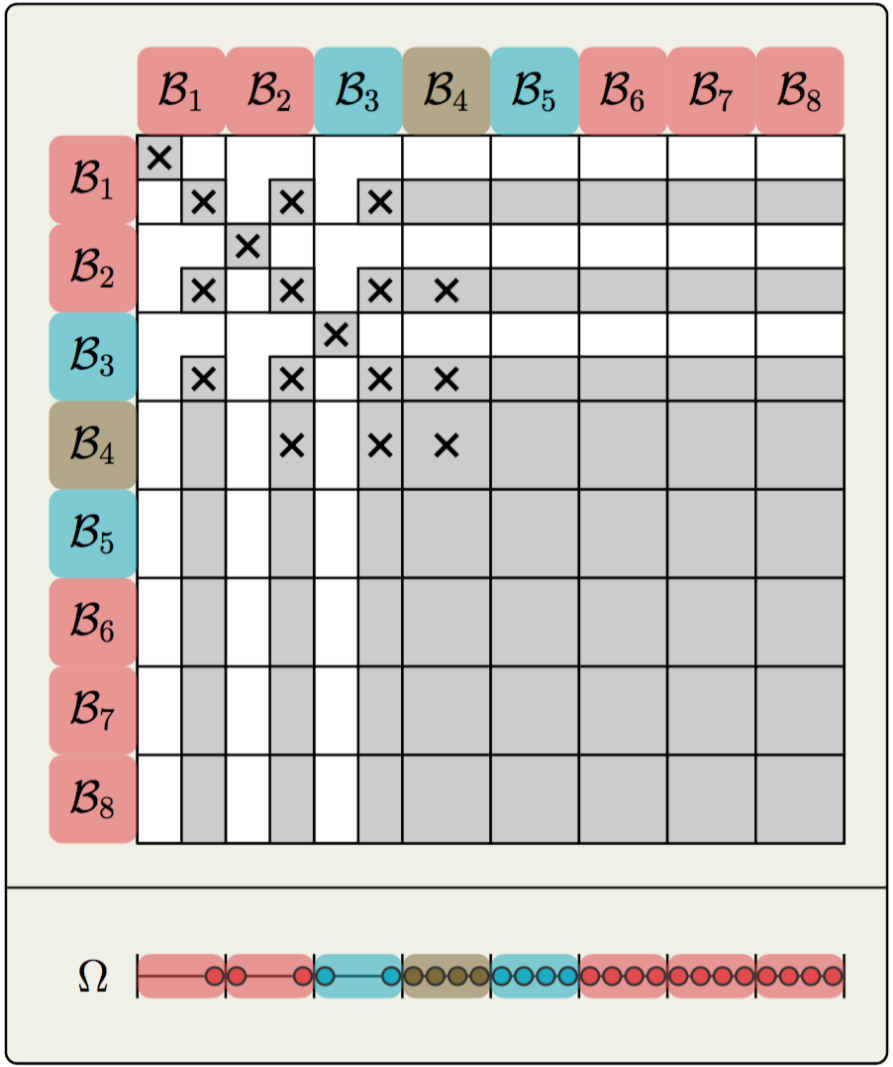
1D
Level 1
Before $\mathcal{B}_3$



The matrix

Points in the domain

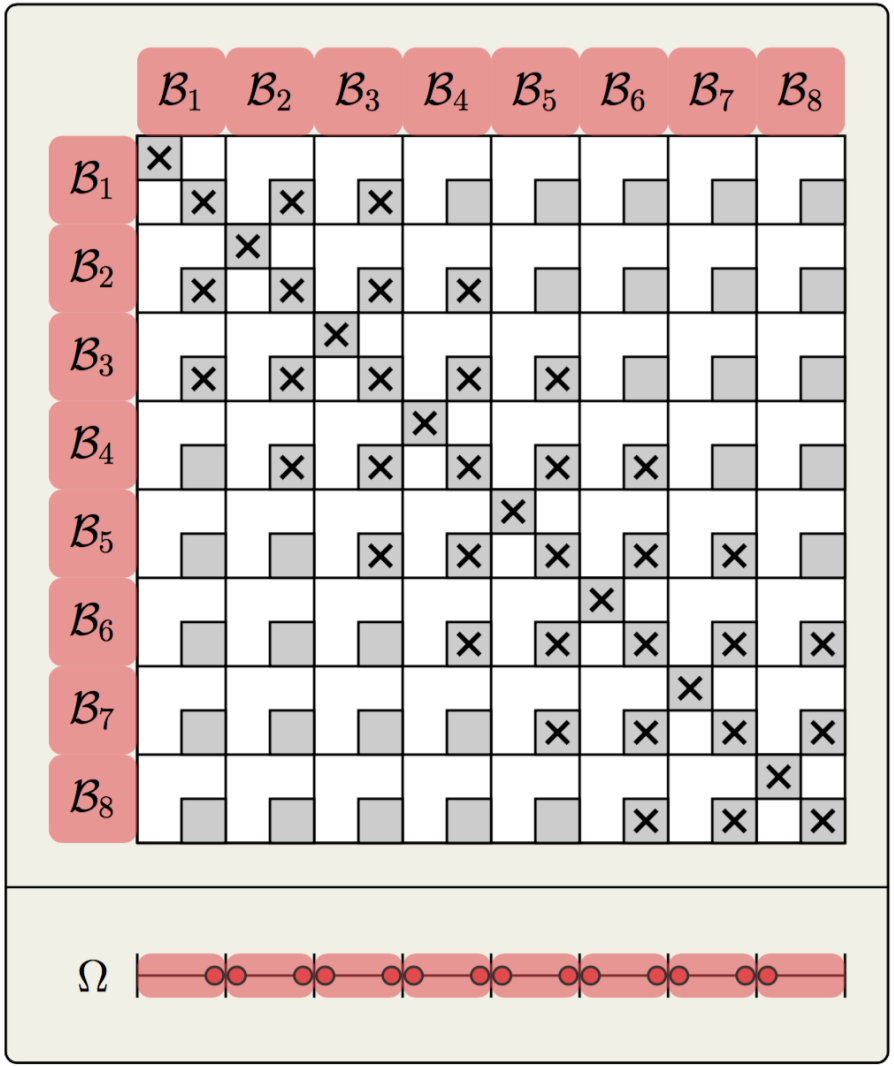
1D
Level 1
Before $\mathcal{B}_4$



The matrix

Points in the domain

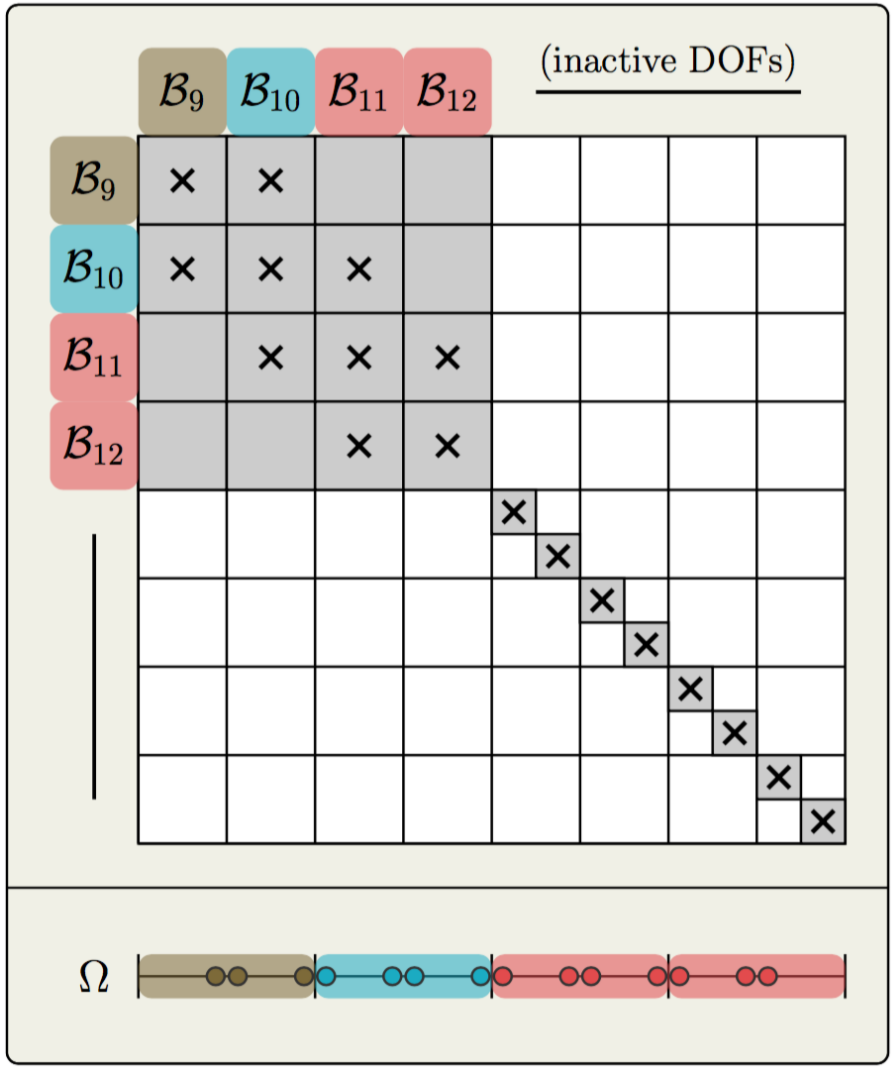
1D
Level 1
After $\mathcal{B}_8$



The matrix

Points in the domain

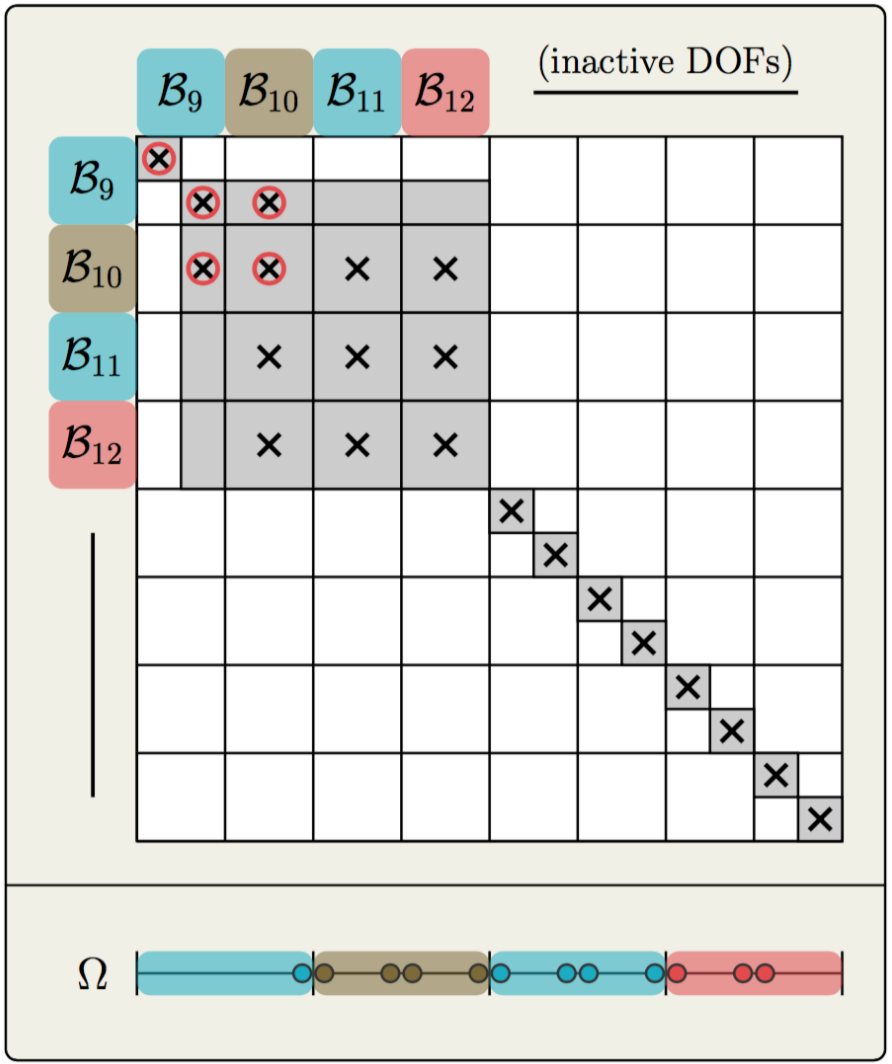
1D
Level 2
Before $\mathcal{B}_9$



The matrix

Points in the domain

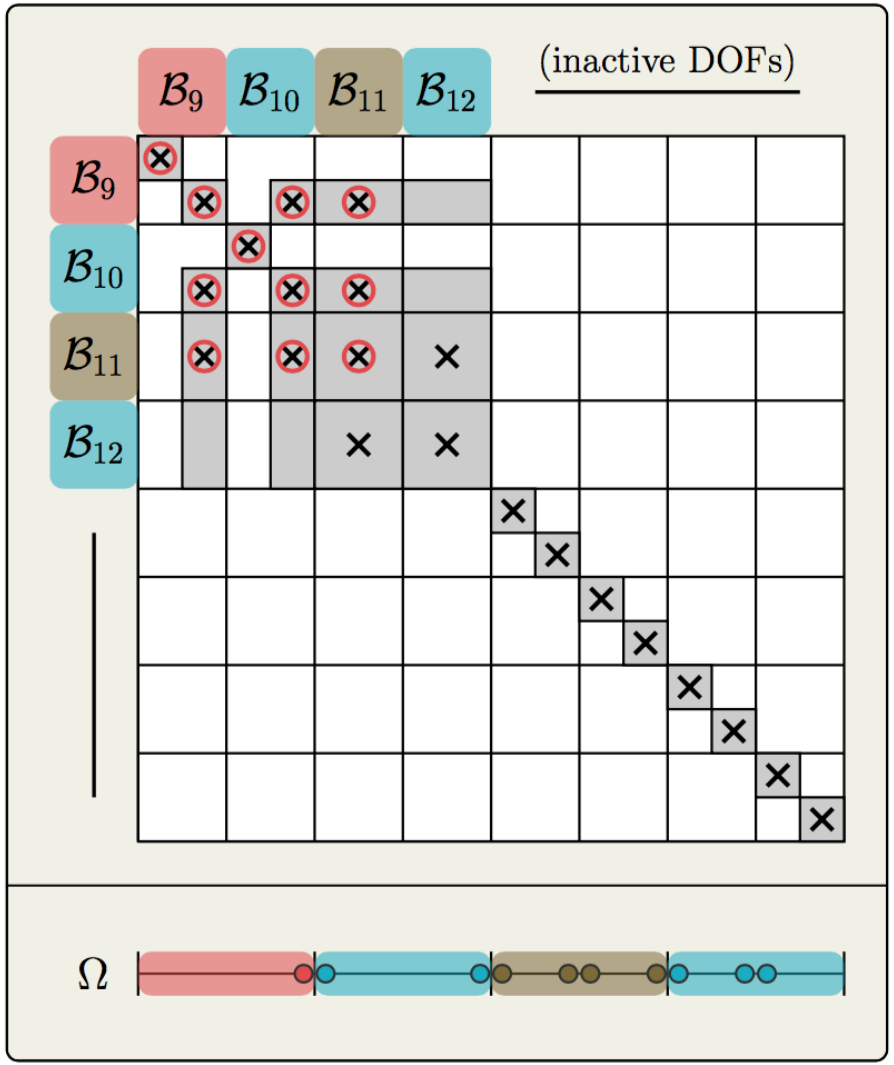
1D
Level 2
Before $\mathcal{B}_{10}$



The matrix

Points in the domain

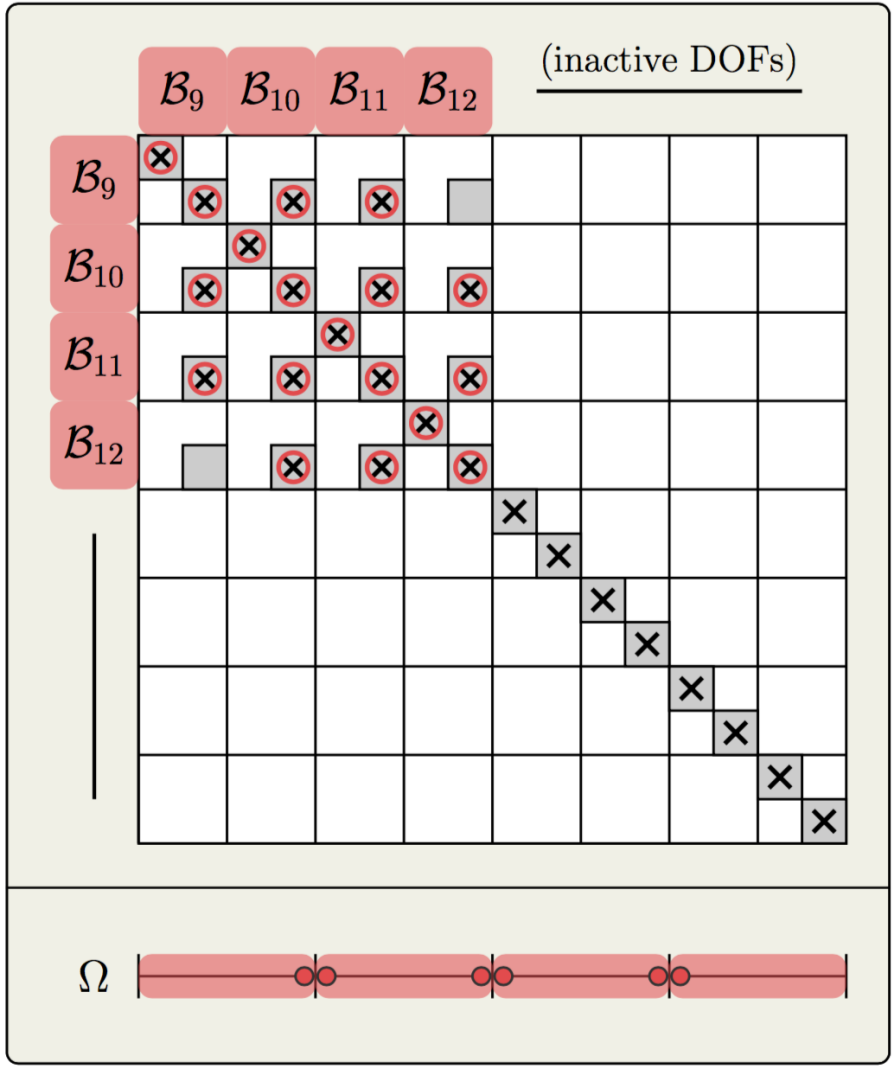
1D
Level 2
Before $\mathcal{B}_{11}$



The matrix

Points in the domain

1D
Level 2
After $\mathcal{B}_{12}$

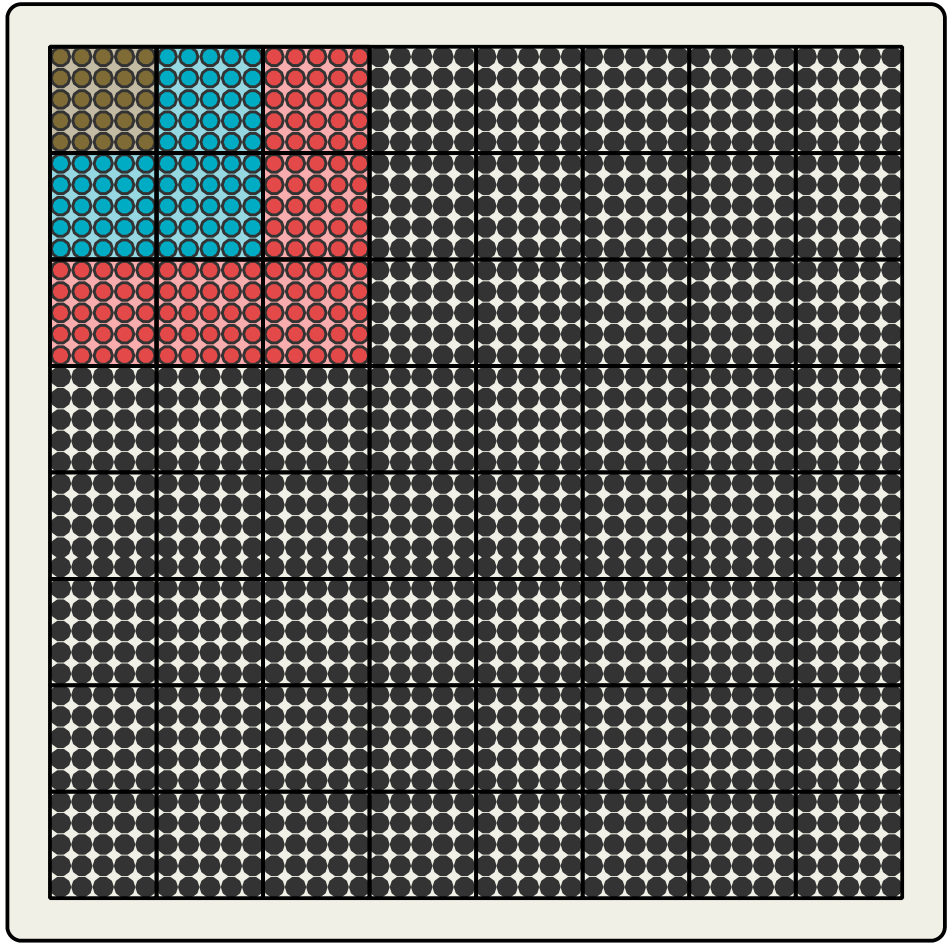


The matrix

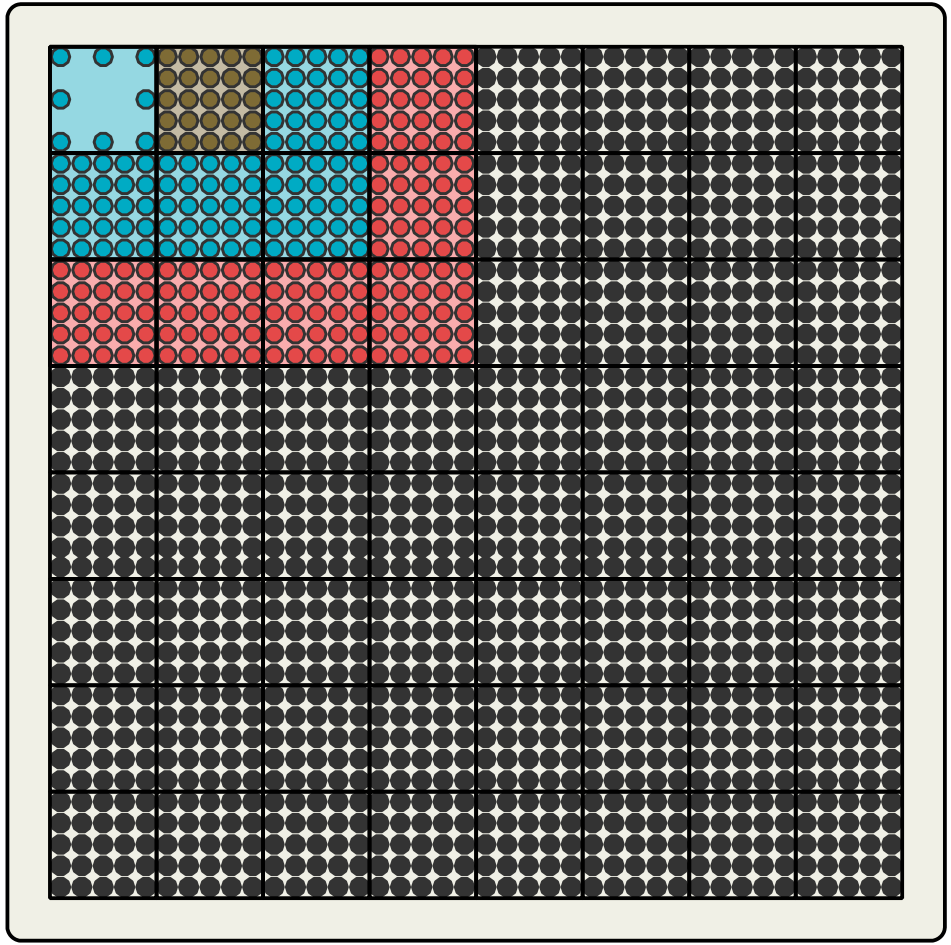
Finally: invert (or factor) diagonal blocks

Points in the domain

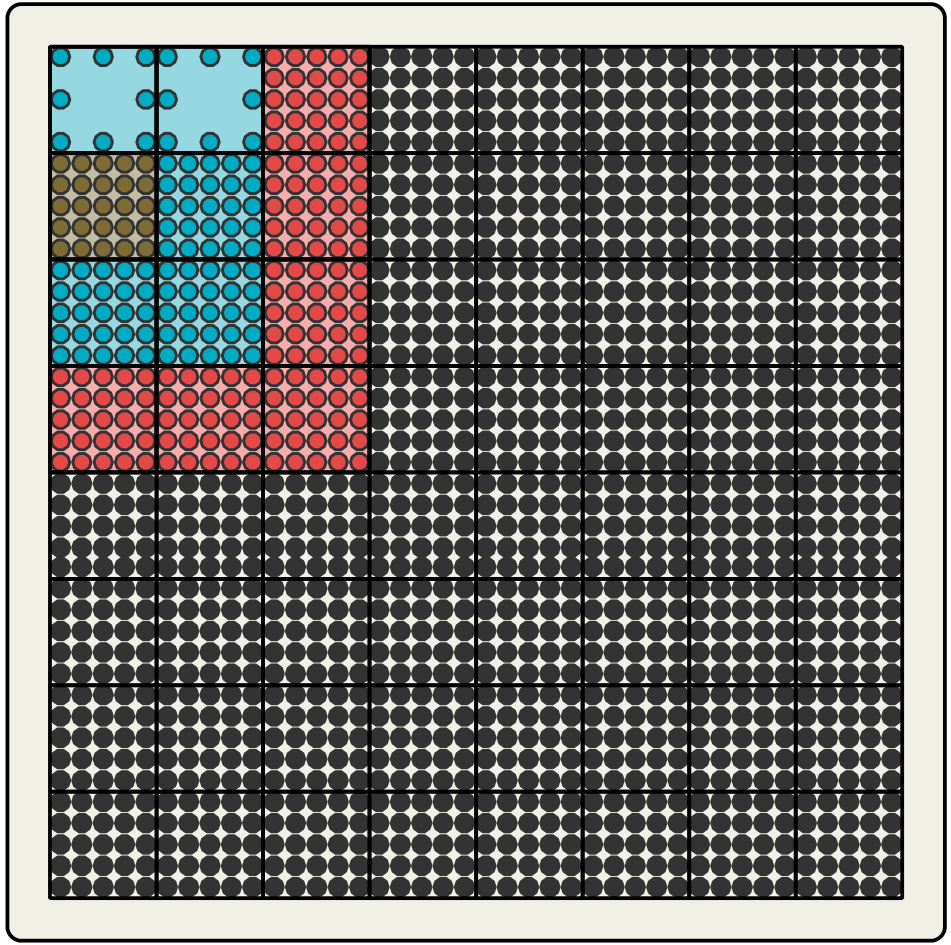
2D
Level 1
Before $\mathcal{B}_1$



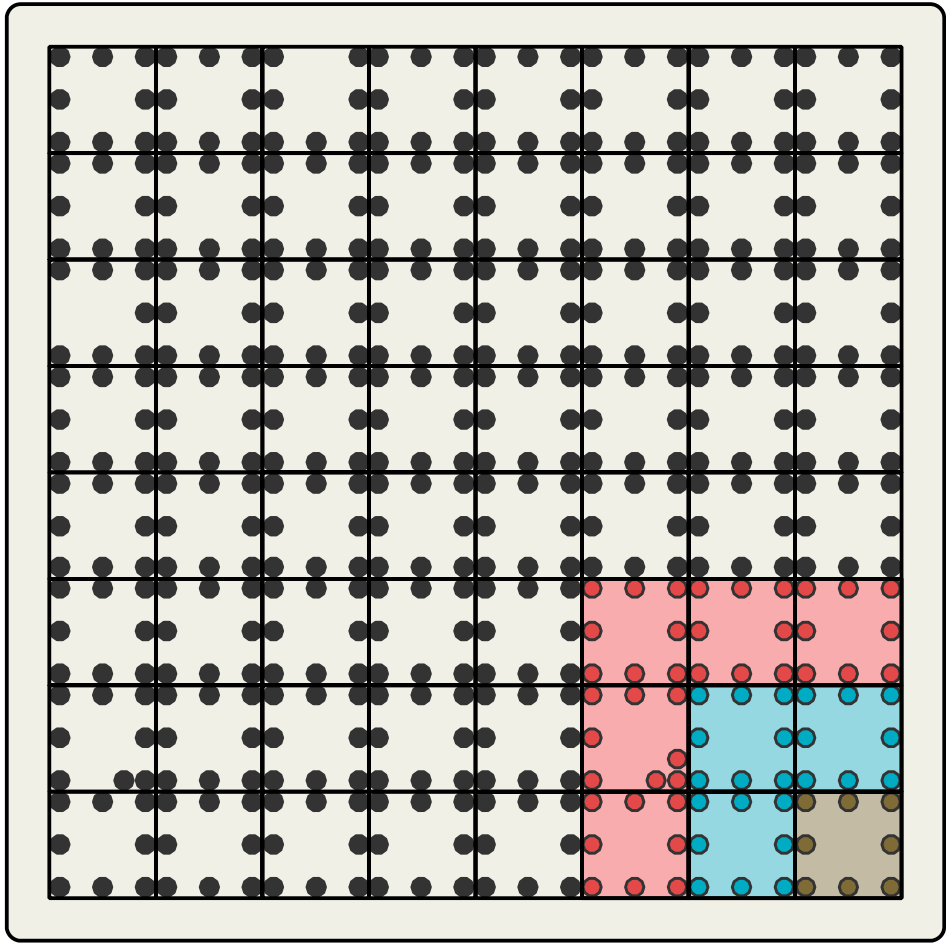
2D
Level 1
Before $\mathcal{B}_2$



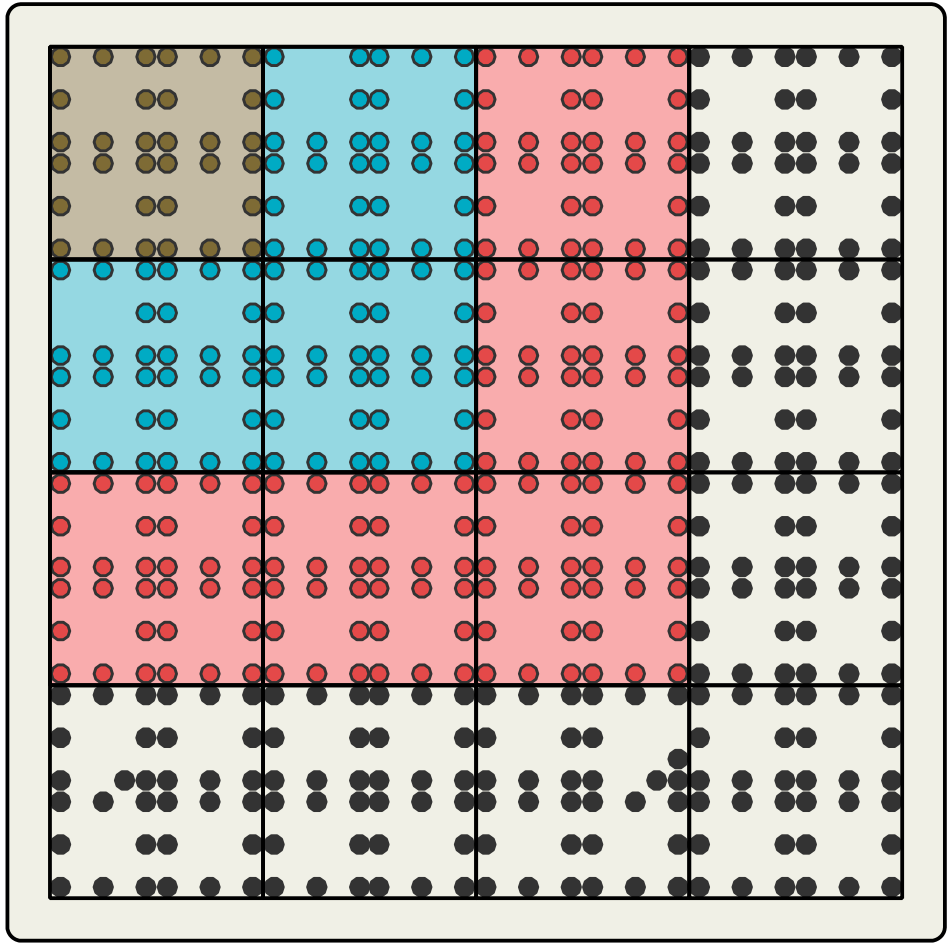
2D
Level 1
Before $\mathcal{B}_3$



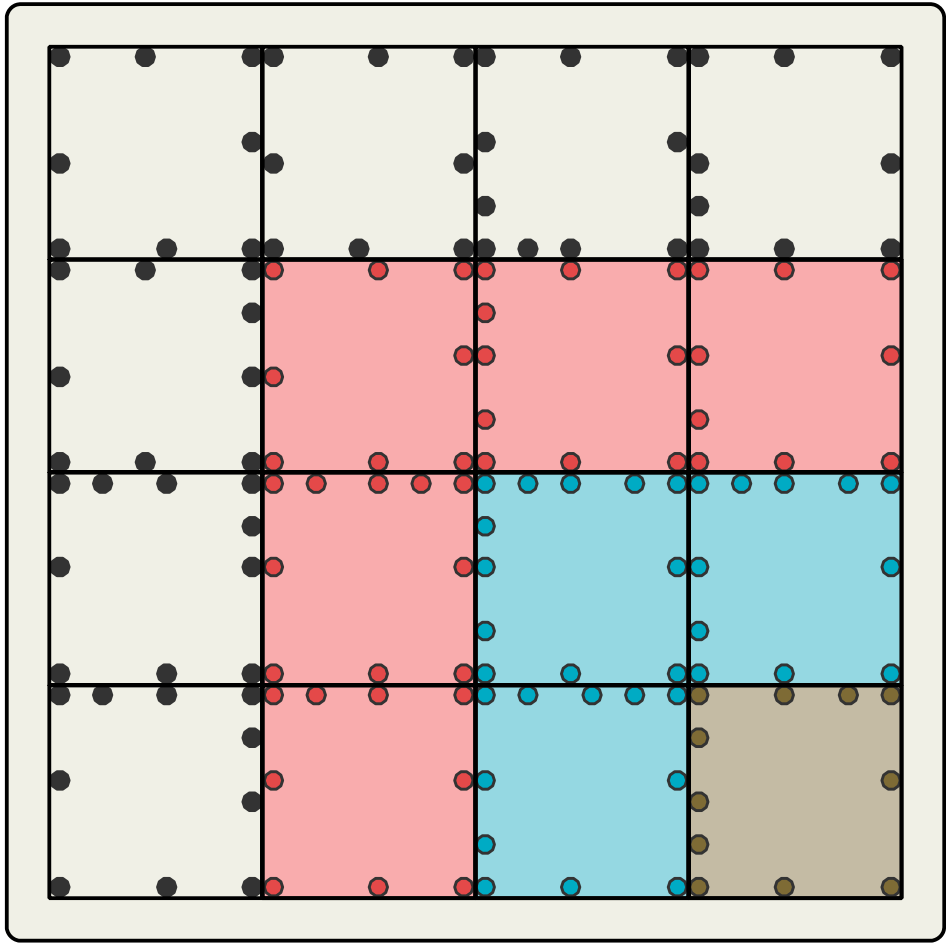
2D
Level 1
After $\mathcal{B}_{64}$



2D
Level 2
Before $\mathcal{B}_{65}$



2D
Level 2
After $\mathcal{B}_{80}$



What does a factorization look like?

- Forward operator is the product of many interlaced block unit triangular matrices, permutation matrices, and a block diagonal matrix

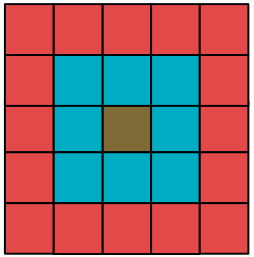
$$K \approx \left(\prod_{i\in[n]} V_i\right) P_t D P_t^* \left(\prod_{i\in[n]'} W_i\right) \equiv F$$

- Same holds for inverse operator

$$K^{-1} \approx \left(\prod_{i\in[n]} W_i^{-1}\right) P_t D^{-1} P_t^* \left(\prod_{i\in[n]'} V_i^{-1}\right) = F^{-1}$$

Complexity sketch

- For each box:
  1. Compute interpolative decomposition
  2. Perform block elimination



$$b = s \cup r$$

$$K_{fr} \approx K_{fs} T$$

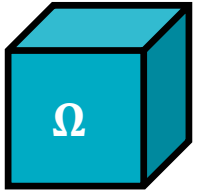
- If proxy surface used and ranks do not grow in N:

$\mathcal{O}(N)$ (linear complexity)

$$t_f = O(N) + \sum_{\ell=1}^{L-2} O(2^{d(L-\ell)} k_\ell^3)$$

$$\left[\begin{array}{c|c|c} X_{rr} & & \\ & X_{ss} & X_{sn} & K_{sf} \\ \hline & X_{ns} & X_{nn} & K_{nf} \\ \hline & K_{fs} & K_{fn} & K_{ff} \end{array}\right]$$

Some scaling results (3D unit cube)



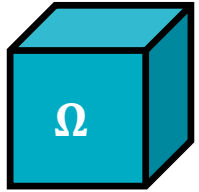
- Intel(R) Xeon(R) CPU E7-8890 @ 2.50GHz

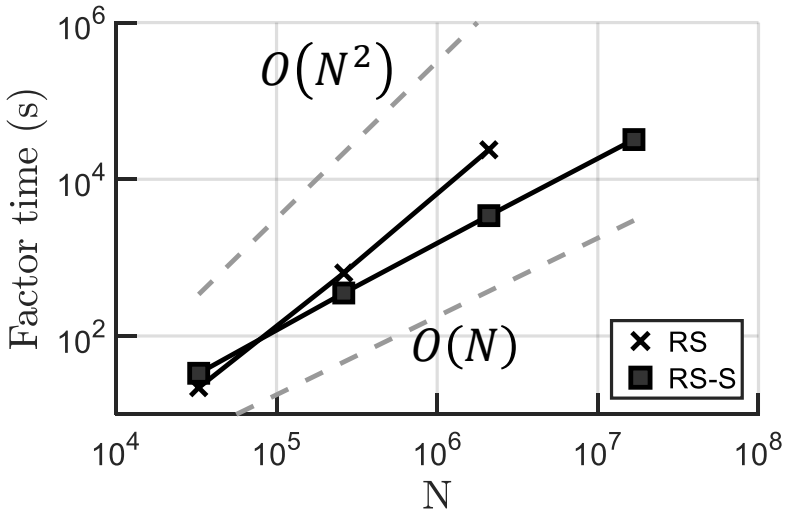
$$\int_{\Omega} \frac{1}{4\pi|x-y|} u(y)\, dy = f(x), \qquad x \in \Omega = [0,1]^3$$

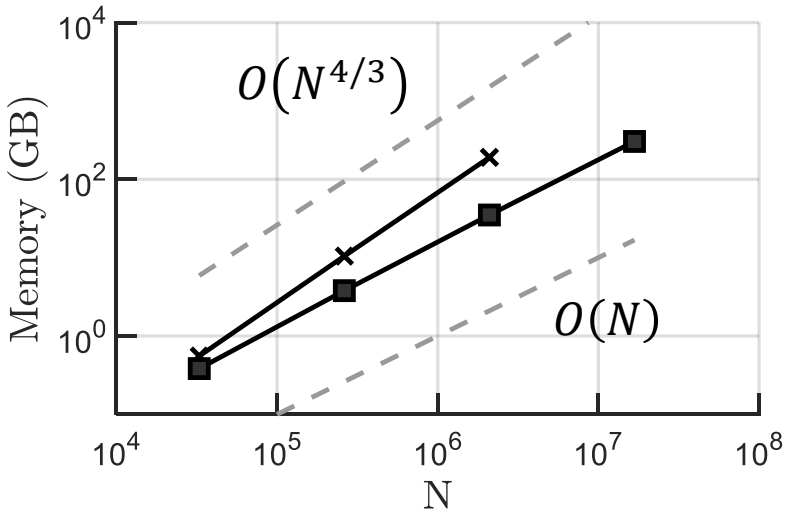
- Methods
  - Recursive skeletonization (RS)
  - Strong recursive skeletonization (RS-S)

3D results: lower accuracy (tolerance 10^{-3})

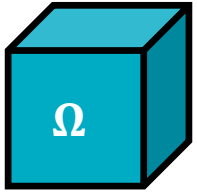
- Conjugate gradient converges to a relative residual norm of 10^{-12} in about 10 preconditioned iterations.



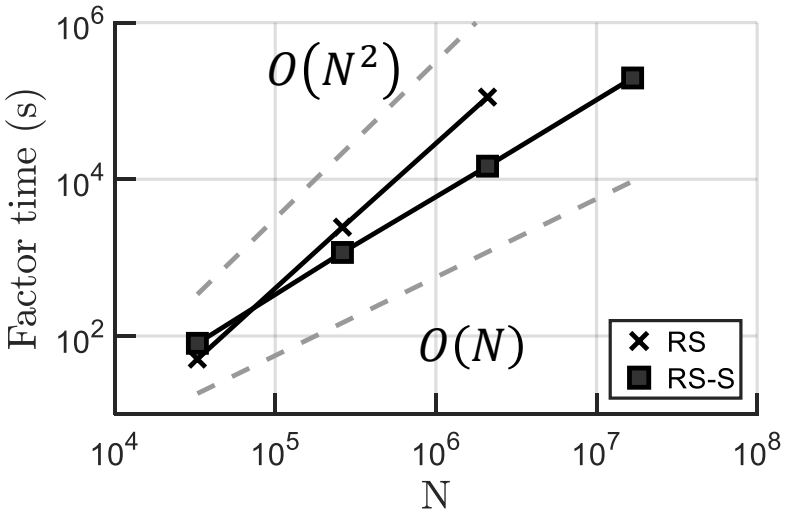


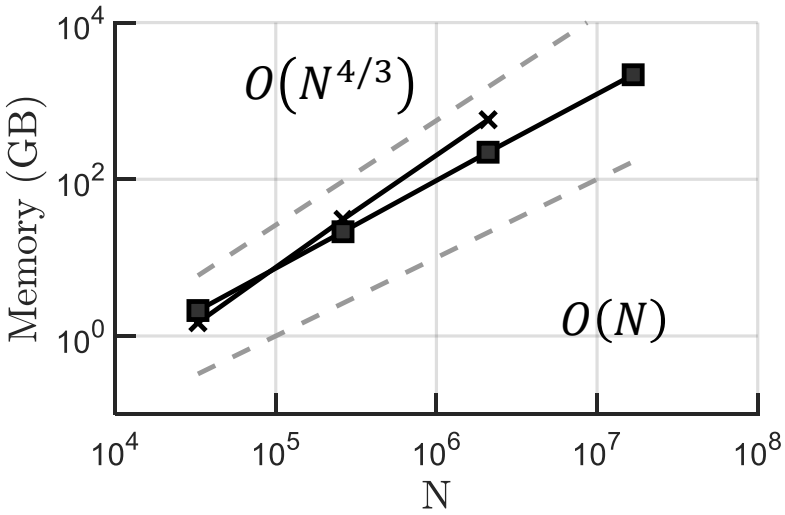


3D results: higher accuracy (tolerance 10^{-6})



- Conjugate gradient converges to a relative residual norm of 10^{-12} in about 3 preconditioned iterations.





Comments on results

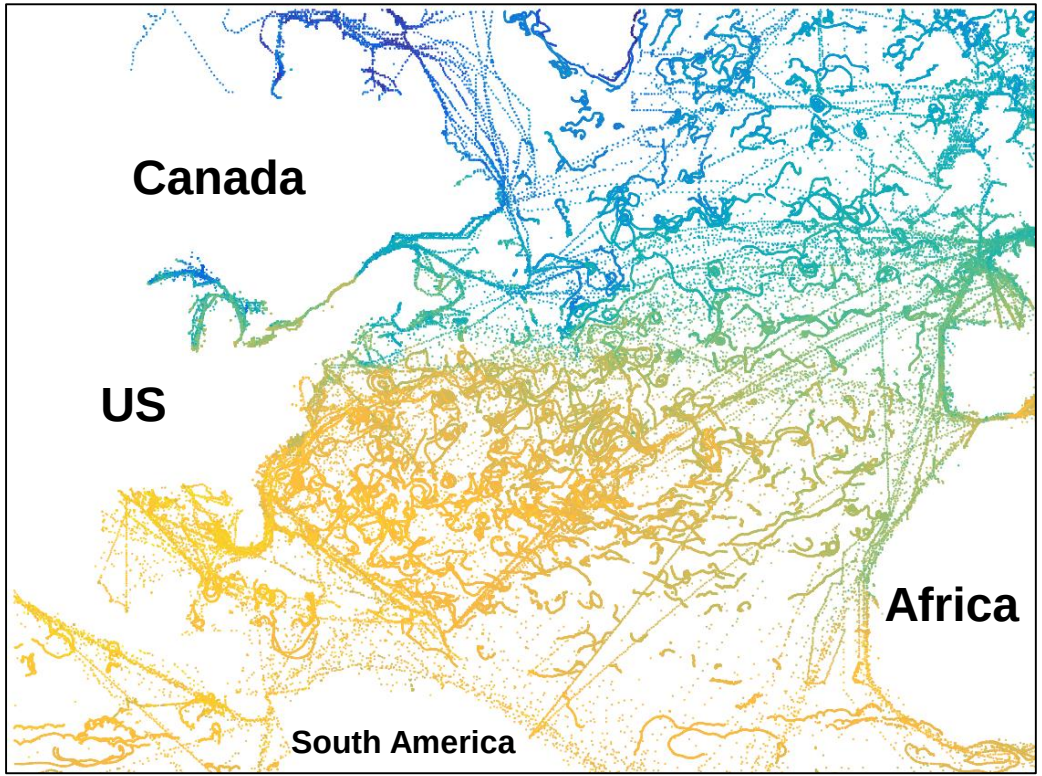
- Basic take-away
  - New factorization can be viewed as an “inverse” form of the FMM based on recursive skeletonization
  - Obtain an approximate multiplicative factorization of the inverse operator
  - Exhibits linear scaling, avoiding excess rank growth from compressing near-field
  - Simple to understand and to implement

Fast spatial Gaussian process MLE

WITH DAMLE, HO, YING

Real data: sea surface temperature (source: ICOADS)

- What is a good model for spatial data?
- Tobler's first law of geography
  - "Everything is related to everything else, but near things are more related than distant things"



The Gaussian process model (in 2D)

- Field is a function of space, $\mathcal{Z}: \mathbb{R}^2 \to \mathbb{R}$
- Finite-dimensional distributions are multivariate normal

$$(z_1, \ldots, z_N)^T \sim N(0, \Sigma), \text{ with } \Sigma_{ij} = k(x_i, x_j; \theta)$$

for any collection of observations $\{z_i\} = \{\mathcal{Z}(x_i)\}$ (simple kriging)

Applications of "kriging"

- Mining
- Hydrogeology
- Environmental science
- Natural resources

[Krige, 1951] [Zimmerman et al., 1998] [Bayraktar, 2005] [Goovaerts, 1997]

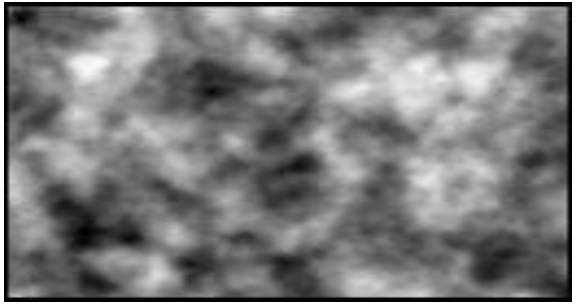
Maximum likelihood estimation for GPs (1)

Choice of kernel function is very important for GP regression

- Squared-exponential: $k(x, y; \theta) = \exp(-|x - y|_\theta^2)$
- Matérn kernel (one such): $k(x, y; \theta) = \left(1 + \sqrt{3}|x - y|_\theta\right) \exp(-\sqrt{3}|x - y|_\theta)$

Parameterize kernel for flexibility, e.g.,

$$|x - y|_\theta^2 = \frac{(x_1 - y_1)^2}{\theta_1^2} + \frac{(x_2 - y_2)^2}{\theta_2^2}$$



$\theta = [7, 10]$

$\theta = [30, 3]$

Maximum likelihood estimation for GPs (2)

So, assume GP model makes sense [Stein, 1999] and:

- Kernel $k(x, y; \theta)$ is specified up to θ (so Σ depends on θ)
- Have observation vector z= $[z_1, \ldots, z_N]^T$ with locations $\{x_i\} \subset \mathbb{R}^2$

Goal: efficient method for finding maximum likelihood estimate

$$\hat{\theta}_{MLE} = \operatorname{argmax} \ell(\theta)$$

where (up to an affine shift)

$$\ell(\theta) = -\mathrm{z}^{\mathrm{T}}\Sigma^{-1}z - \log|\Sigma|$$

$$\partial_{\theta_i}\ell(\theta) = z^T\Sigma^{-1}\Sigma_i\Sigma^{-1}z - \mathrm{Tr}(\Sigma^{-1}\Sigma_i)$$

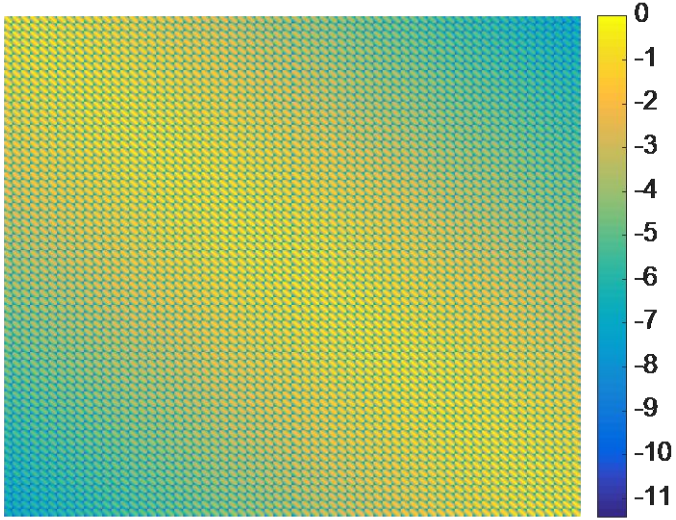
Our approach: black-box MLE

- We outline a simple efficient scheme for computing:
  - the log-likelihood (including $\log|\Sigma|$)
  - its gradient (including $\mathrm{Tr}(\Sigma^{-1}\Sigma_i)$)

- Then, use black-box optimization scheme (e.g., fminunc/fmincon)

- Some other approaches:
  - Sample average approximation
  - (Block) composite likelihood
  - Approximate by Gaussian Markov random field
  - Covariance tapering
  - Multi-level preconditioning + tapering

[Anitescu et al., 2012] [Eidsvik et al., 2014] [Vecchia, 1988] [Lindgren et al., 2011] [Furrer et al., 2006] [Castrillon-Candas et al., 2015]



Computing the objective function

- Computing $\ell(\theta) = -\mathrm{z}^{\mathrm{T}}\Sigma^{-1}z - \log|\Sigma|$ requires:
  1. Forming $\Sigma^{-1}z$ (by solving $\Sigma y = z$ for y)
  2. Computing $\log|\Sigma|$
- Both are efficient (linear complexity) using an appropriate skeletonization factorization of Σ (RS, RS-S, HIF)

$$\mathsf{F} \equiv \left(\prod_{i\in[n]} \mathsf{V}_i\right) \mathsf{P}_t \mathsf{D} \mathsf{P}_t^* \left(\prod_{i\in[n]'} \mathsf{W}_i\right) \qquad \log|\Sigma| \approx \log|F| = \log|D|$$

  - Previous work on hierarchical decompositions for Gaussian processes
    [Ambikasaran et al., 2016] [Ambikasaran et al., ArXiV]
    [Borme & Garcke, 2007] [Khoromskij et al., 2008]
    but no gradients

Computing the gradient

- The components of the gradient are more complicated

$$\partial_{\theta_i} \ell(\theta) = z^T \Sigma^{-1} \Sigma_i \Sigma^{-1} z - \mathrm{Tr}(\Sigma^{-1}\Sigma_i)$$

- One option: matrix peeling [Lin et al., 2011]
  - Compute a hierarchical representation of a black-box operator $\Sigma^{-1}\Sigma_i$ by applying it to structured random vectors, then extracting diagonal, then summing to get trace.

- A better option: selected sparse algebra (new)
  - Compute $\mathrm{Tr}(\Sigma^{-1}\Sigma_i)$ by intricately using the structure of factorizations of Σ and Σ_i
  - Use properties of trace

$$\begin{pmatrix} G_{1;11} & G_{1;12} \\ G_{1;21} & G_{1;22} \end{pmatrix}$$

$$\begin{pmatrix} G_{2;11} & G_{2;12} & & \\ G_{2;21} & G_{2;22} & & G_{1;12} \\ & & G_{2;33} & G_{2;34} \\ G_{1;21} & & G_{2;43} & G_{2;44} \end{pmatrix}$$

$$\begin{pmatrix} G_{3;11} & G_{3;12} & & & & & & \\ G_{3;21} & G_{3;22} & G_{2;12} & & & & & \\ & G_{2;21} & G_{3;33} & G_{3;34} & & & G_{1;12} & \\ & & G_{3;43} & G_{3;44} & & & & \\ & & & & G_{3;55} & G_{3;56} & & \\ & & & & G_{3;65} & G_{3;66} & & G_{2;34} \\ & G_{1;21} & & & & & G_{3;77} & G_{3;78} \\ & & & & G_{2;43} & & G_{3;87} & G_{3;88} \end{pmatrix}$$

A better method: selected sparse algebra using locality

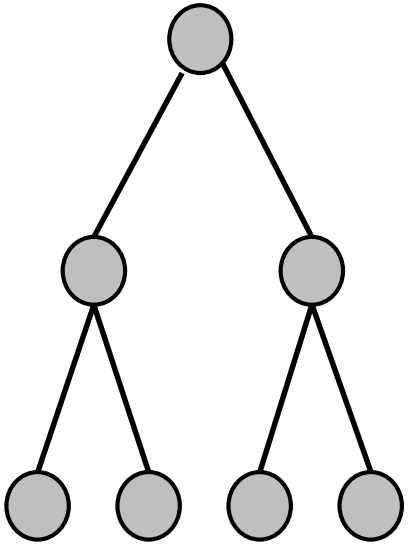
- Consider simpler case to start: drop product and look at $F \approx \Sigma$

$$\mathsf{F} \equiv \left(\prod_{i \in [n]} \mathsf{V}_i \right) \mathsf{P}_t \mathsf{D} \mathsf{P}_t^* \left(\prod_{i \in [n]'} \mathsf{W}_i \right)$$

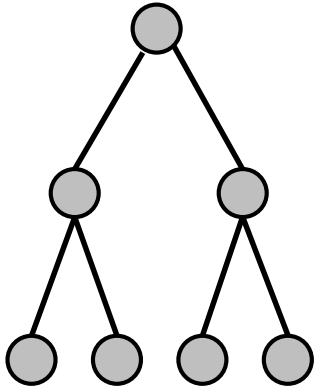
- Application of factorization *F* to a vector is two-stage:
  1. Apply a factor for each node from bottom-to-top

     (apply a block diagonal operator)

  2. Apply a factor for each node from top-to-bottom



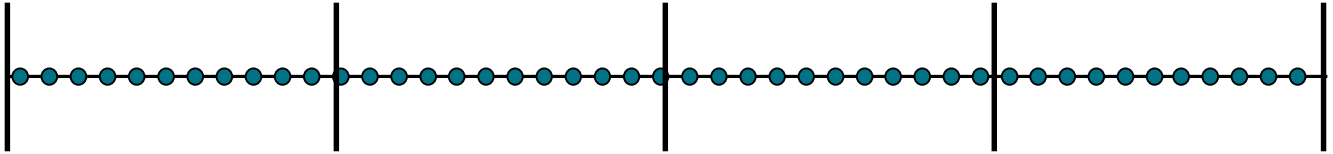
With sparse input, skip some factors on the way up

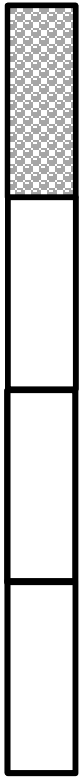


Left: tree

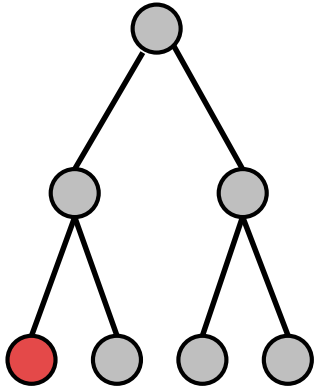
Right: vector

Below: domain





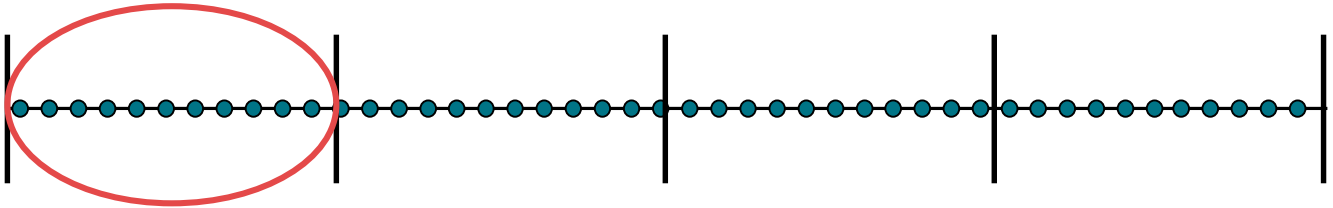
With sparse input, skip some factors on the way up

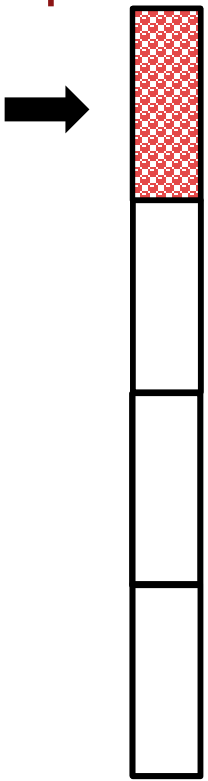


Left: tree

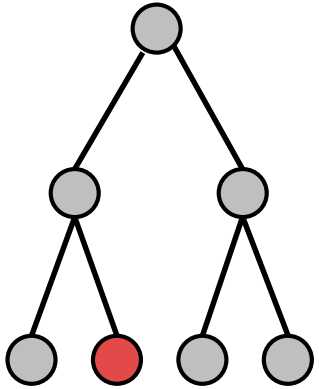
Right: vector

Below: domain





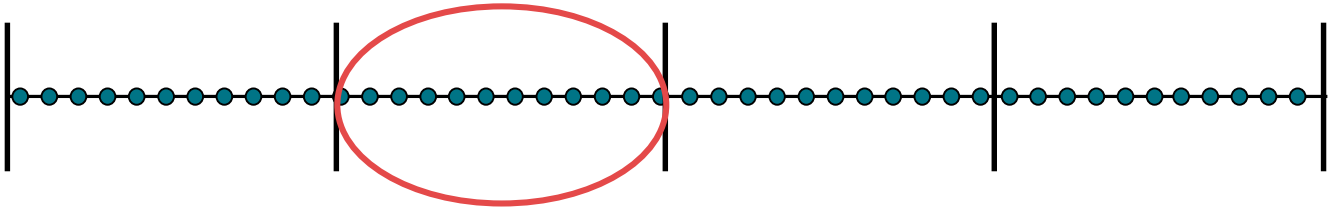
With sparse input, skip some factors on the way up

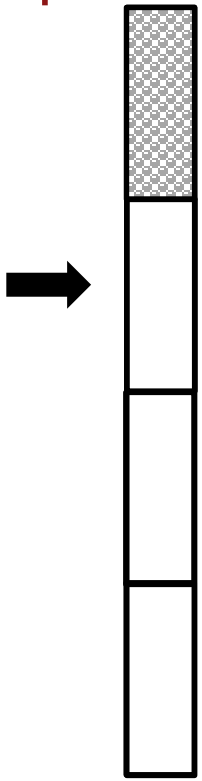


Left: tree

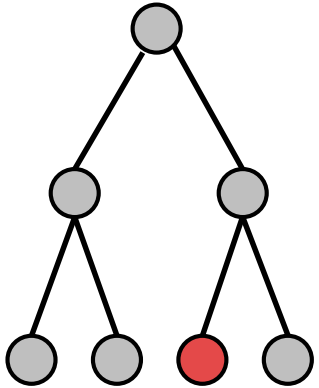
Right: vector

Below: domain





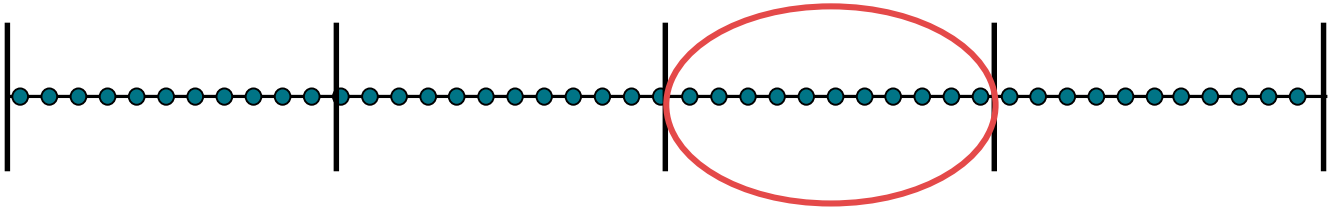
With sparse input, skip some factors on the way up

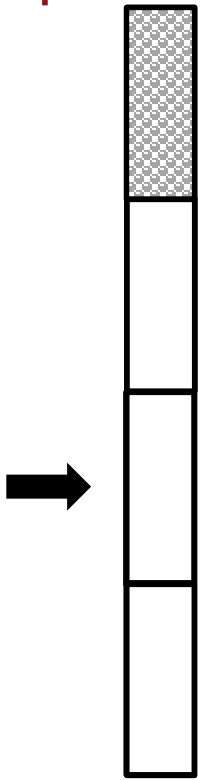


Left: tree

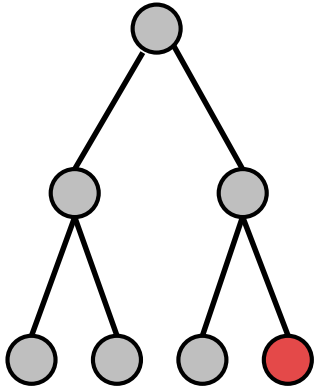
Right: vector

Below: domain





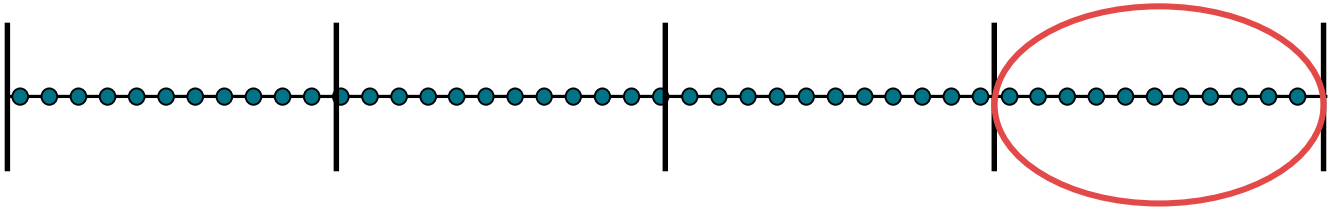
With sparse input, skip some factors on the way up

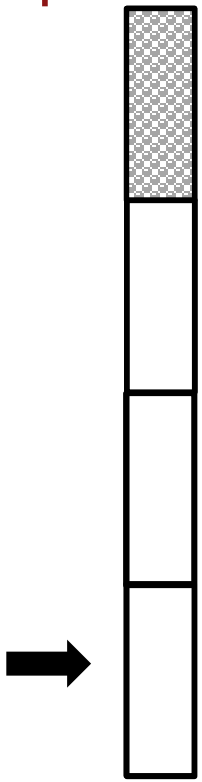


Left: tree

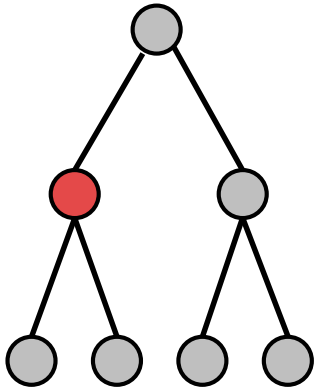
Right: vector

Below: domain





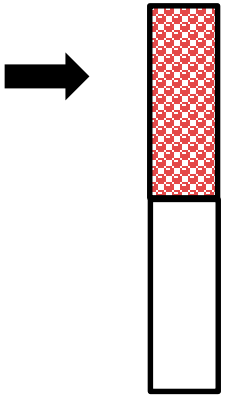
With sparse input, skip some factors on the way up

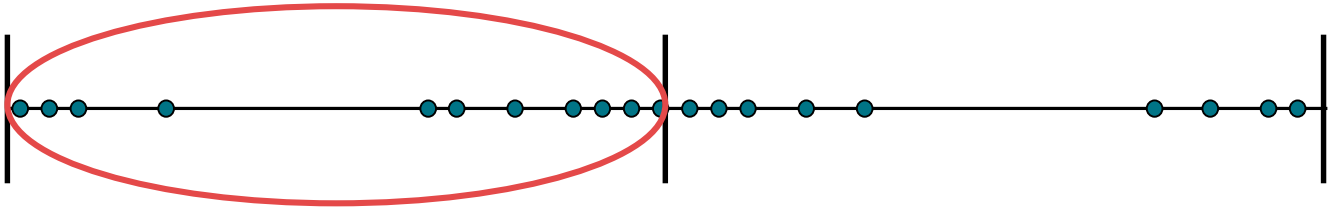


Left: tree

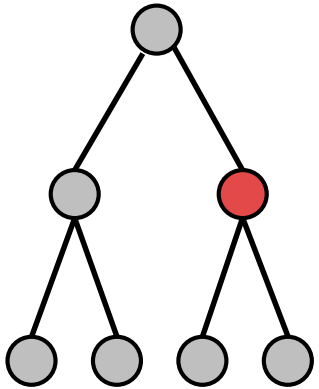
Right: vector

Below: domain





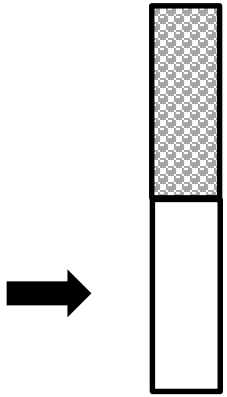
With sparse input, skip some factors on the way up

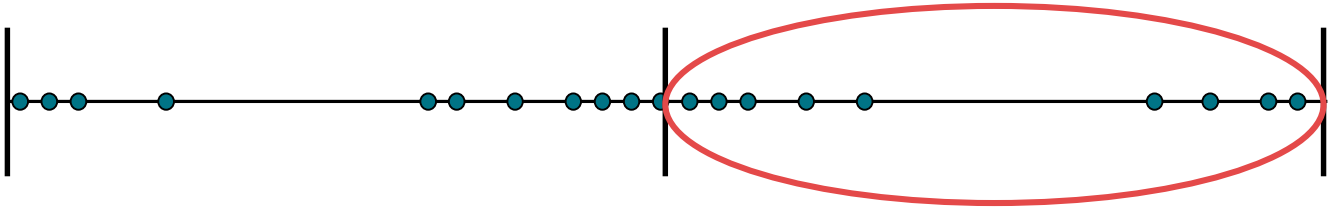


Left: tree

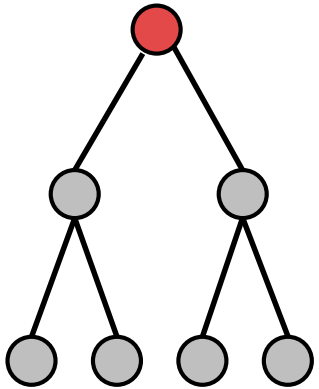
Right: vector

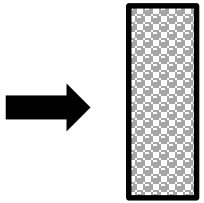
Below: domain





With sparse input, skip some factors on the way up

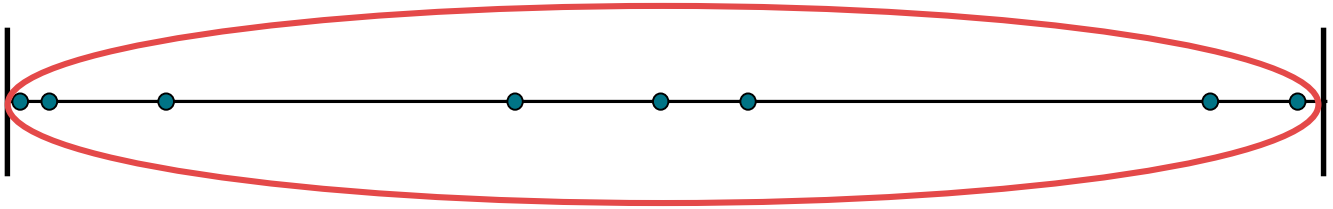




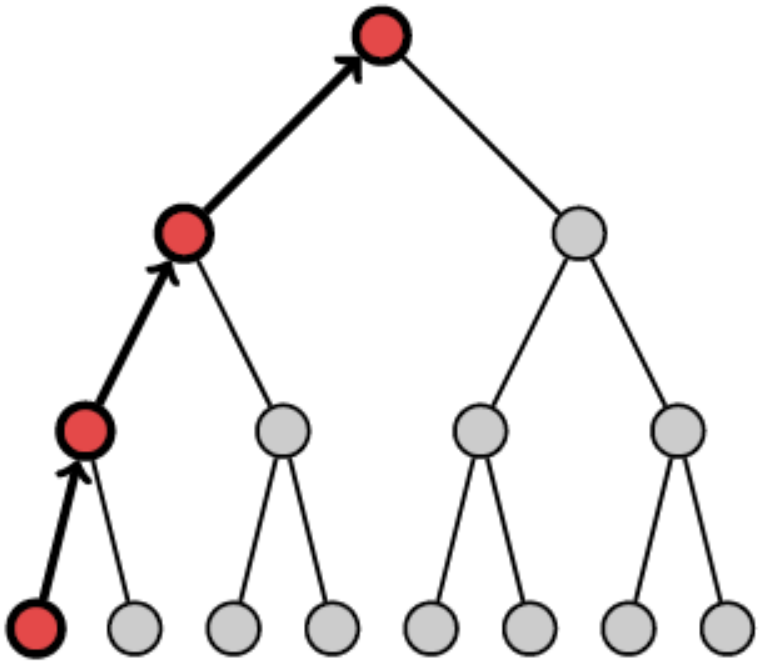
Left: tree

Right: vector

Below: domain

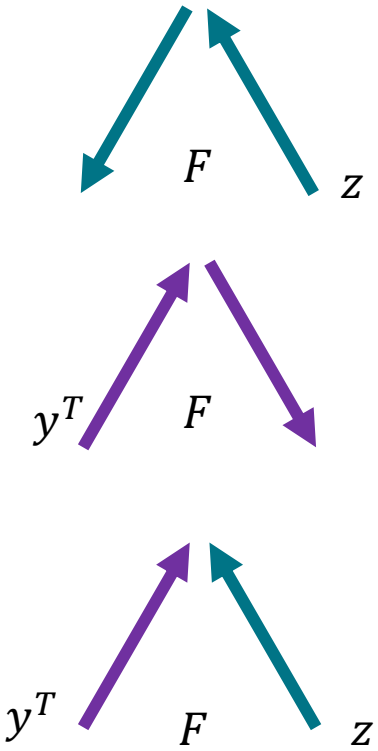


With sparse input, skip some factors on the way up



Result: easy to apply half of factorization to sparse vector

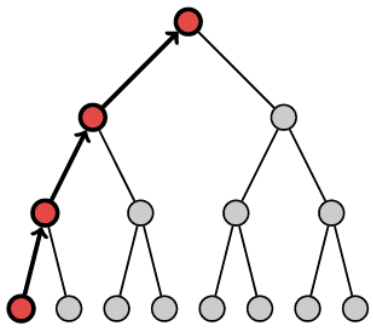
- Computing Fz for a vector z is two-stage:
  1. Walk up tree
  2. Walk down tree
- Computing $y^T F$ for a vector y is two-stage:
  1. Walk up tree
  2. Walk down tree
- Computing $y^T F z$ for vectors y and z is two stage:
  1. Walk up tree (from right)
  2. Walk up tree (from left)



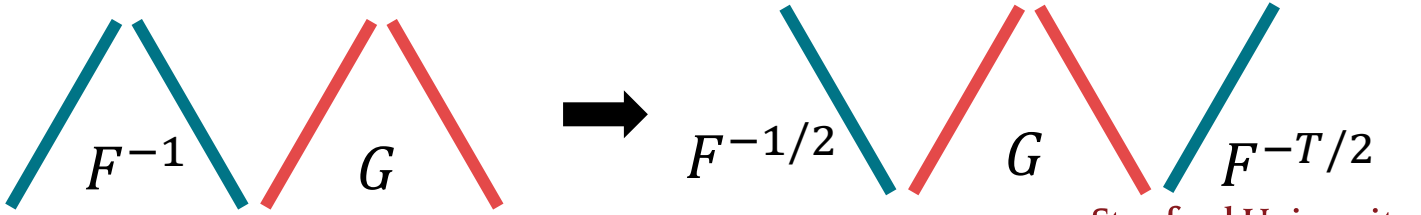
If sparse input and subselected output,
only apply few factors!

Selected sparse algebra (SSA) complexity

Operation	Complexity
Compute F_{ij}	$O(\log^3 N)$
Compute F_{ij}^{-1}	$O(\log^3 N)$
Compute diag (F^{-1})	$O(N\log^3 N)$
Compute $\mathrm{Tr}(F^{-1}G)$	$O(N\log^3 N)$



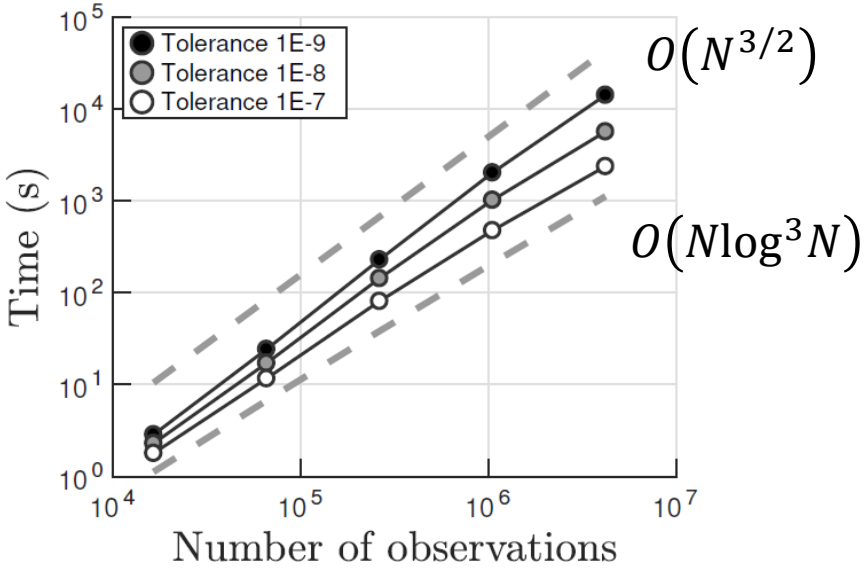
- Similar in spirit to SelInv [Lin et al., 2009] and FIND algorithm [Li et al., 2008] for sparse matrices.
- Does not give product trace directly, but similar idea using two trees

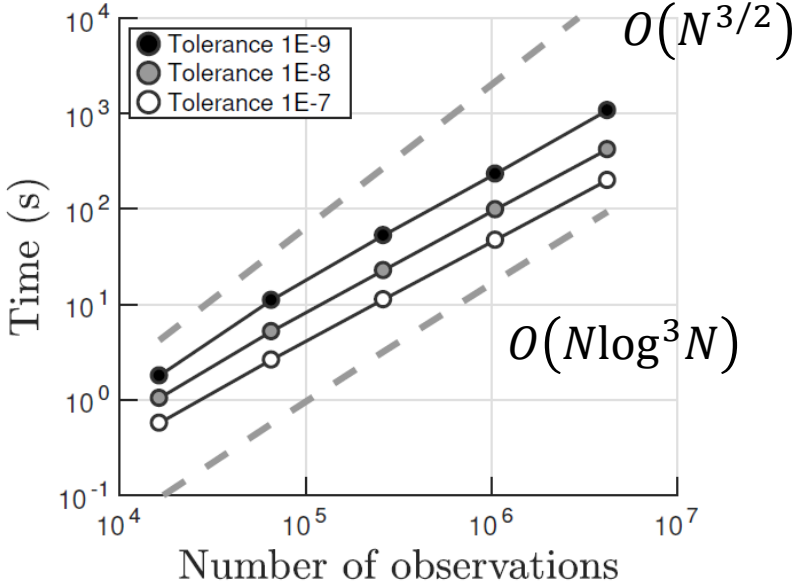


Results for SSA

$$k(x,y;\theta) = \left(1+\sqrt{3}|x-y|_\theta\right)\exp\left(-\sqrt{3}|x-y|_\theta\right)$$

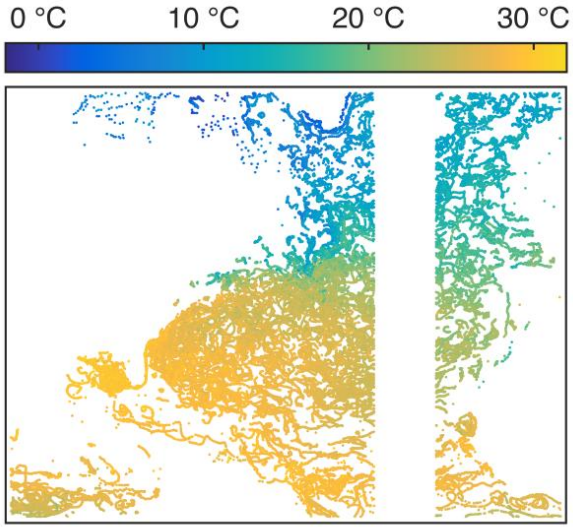
- Computing the product-trace term using SSA
- Gridded observations in 2D with Matérn family kernel with length parameters [7,10] and [70,100].

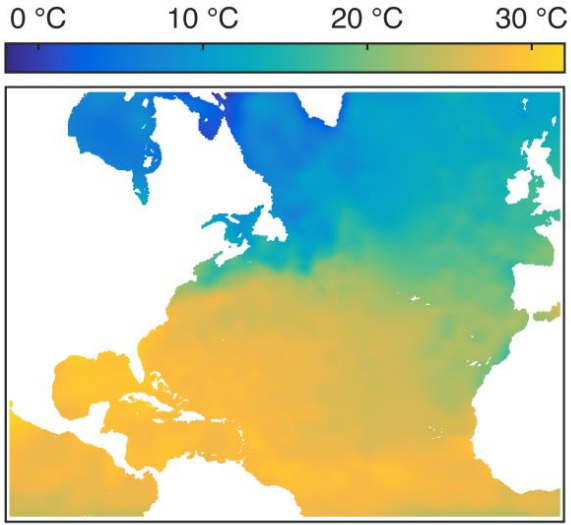




Gaussian process MLE with skeletonization & SSA

- Cost per iteration of black-box optimization: $O(pN\log^3 N)$
- Generated data: convergence in 5 to 7 quasi-Newton iterations
- Numerical differentiation stagnates: gradients are essential!





Comments on results

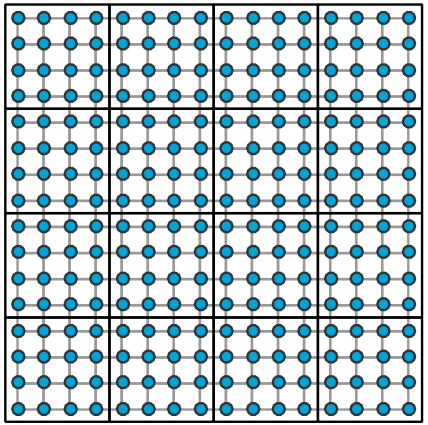
- Basic take-away
  - Hierarchical skeletonization-based factorizations are a natural choice for low-dimensional (spatial) Gaussian processes
  - Hardest (slowest) part is computing product trace for gradient

For $N = 512^2$
Time for SSA is 230 seconds = 7.1 minutes

For $N = 2048^2 = 16 * 512^2$
Time for SSA is 5900 seconds = 1.64 hours

Linear scaling

Final thoughts



Summary of contributions

- Strong recursive skeletonization factorization [Minden et al., 2017]
  - A (nearly) linear-time factorization scheme for solving systems with FMM matrices based on combining skeletonization with strong admissibility

- Fast spatial Gaussian process MLE with selected sparse algebra [Minden et al., ArXiV & thesis]
  - A (nearly) linear-time framework for computing the log-likelihood and its gradient in the context of black-box Gaussian process parameter estimation
  - A method for computing entries of the precision matrix or inverse of other rank-structured systems

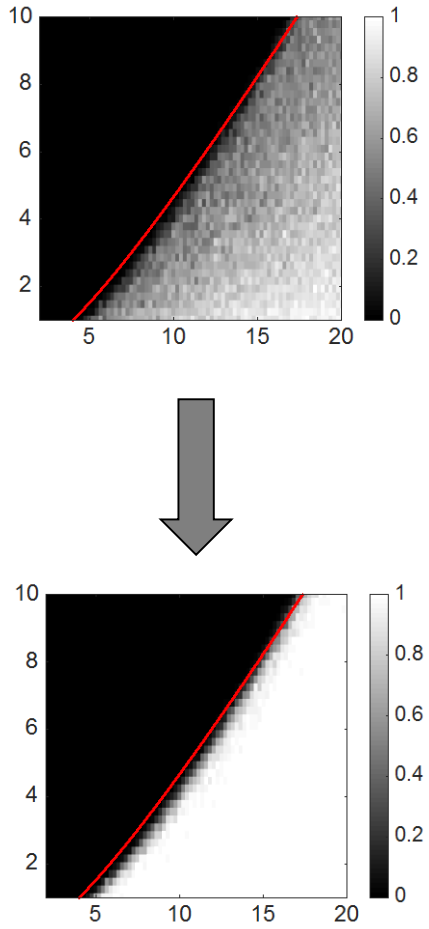
Things I didn't talk about today

- Updating skeletonization factorizations
  [Minden et al., 2016]

- Time-stepping Maxwell's equations via regularized Green's functions
  [Lo, Minden, & Colella, 2016]

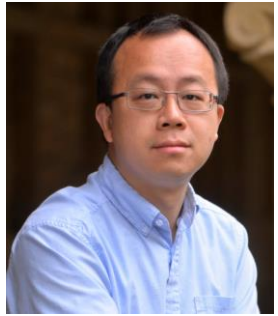
- Robust and efficient spectral clustering
  [Damle, Minden, & Ying, ArXiv]

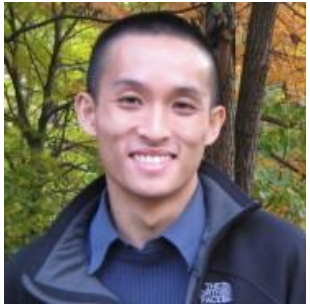
- Sparse canonical correlation analysis
  ongoing work with X. Suo (lead author) and B. Nelson



Thanks

- Professional
  - My advisor (Lexing), my committee (Eric, George, Sanjiva, Michael), my co-authors (Ken, Anil), ICME heads (Margot, Gianluca), ICME staff (Indira, Matt, Emily, Antoinette, Claudine, Brian, Judy, Karen), other mentors and letter writers (Dave, Phil)
- Family and friends
  - My family (mom, dad, brother, sisters, grandmothers, uncles, aunts, cousins), girlfriend (Anusha), ICME students (all but especially Austin, Sven, Ryan, Yingzhou, Zhiyu, Rikel, Xiaotong, Anjan, Ron, Nolan, Lan, Han, Sacha, Mike, Casey, Arun, Neel, Carson, Laura, Eileen, Brad, Cindy, Nurbek, Ruoxi, Dangna, Kari, Milinda, Fei), math post-docs (Yuehaw, Yuwei), friends from high school, college, and life (Alex, Devlin, Adam, Jafar, Sean, Rick, Nate, Dan, Julia, Tegan)



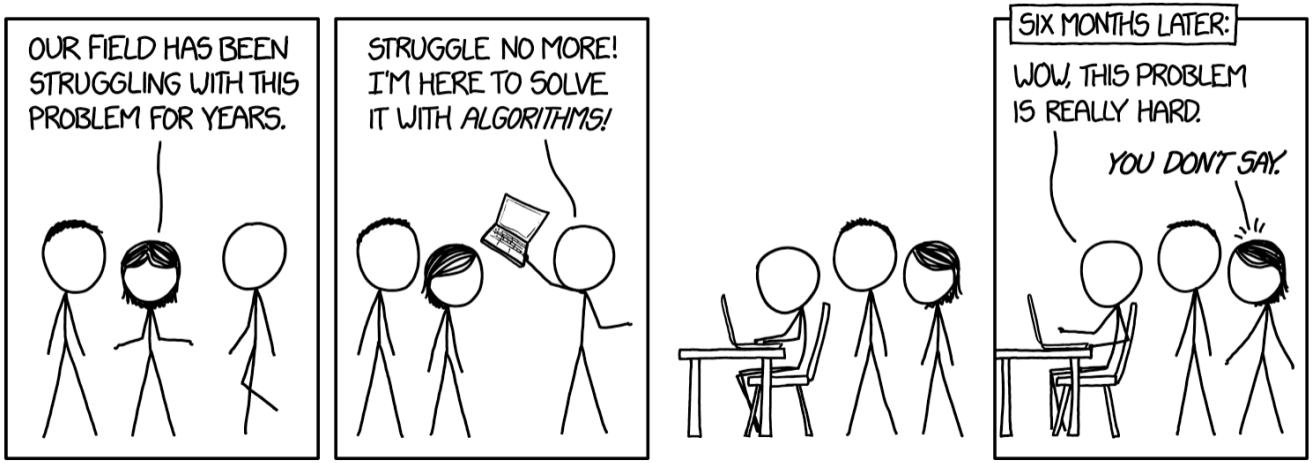


The idea lives not in one person's isolated individual consciousness – if it remains there only, it degenerates and dies. The idea begins to live, that is, to take shape, to develop, to find and renew its verbal expression, to give birth to new ideas, only when it enters into genuine dialogic relationships with other ideas, with the ideas of others.
-Mikhail Bakhtin

Funding

- V.M. is supported by a Stanford Graduate Fellowship and was previously supported by a U.S. Department of Energy Computational Science Graduate Fellowship under grant number DE-FG02-97ER25308.

Thanks for listening!



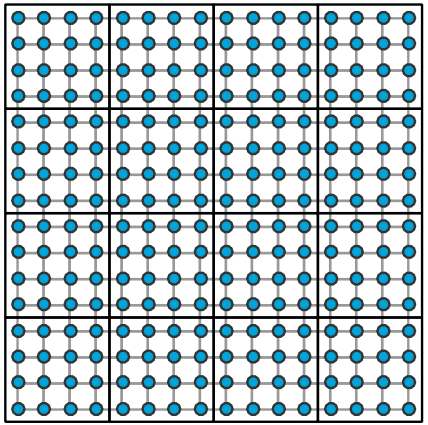
[Comic: R. Munroe, xkcd]

Suggested questions

- Can we get away with a randomized trace estimator instead of the actual product trace?

- Hierarchical matrices already give linear complexity solvers for integral equations, so how is strong skeletonization different?

Back-up slides



Some scaling results (2D unit square)

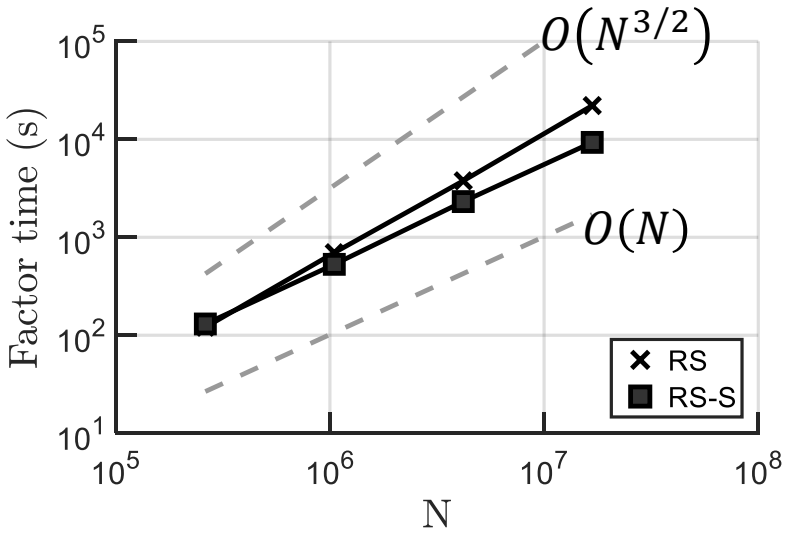
- Intel(R) Xeon(R) CPU E7-8890 @ 2.50GHz

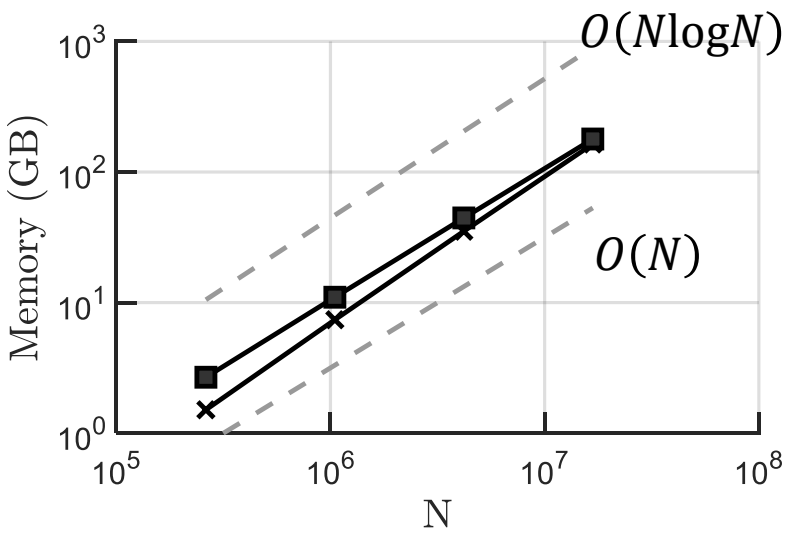
$$\int_{\Omega} \frac{-\log|x-y|}{2\pi} u(y)\, dy = f(x), \qquad x \in \Omega = [0,1]^2$$

- Methods
  - Recursive skeletonization (RS)
  - Strong recursive skeletonization (RS-S)

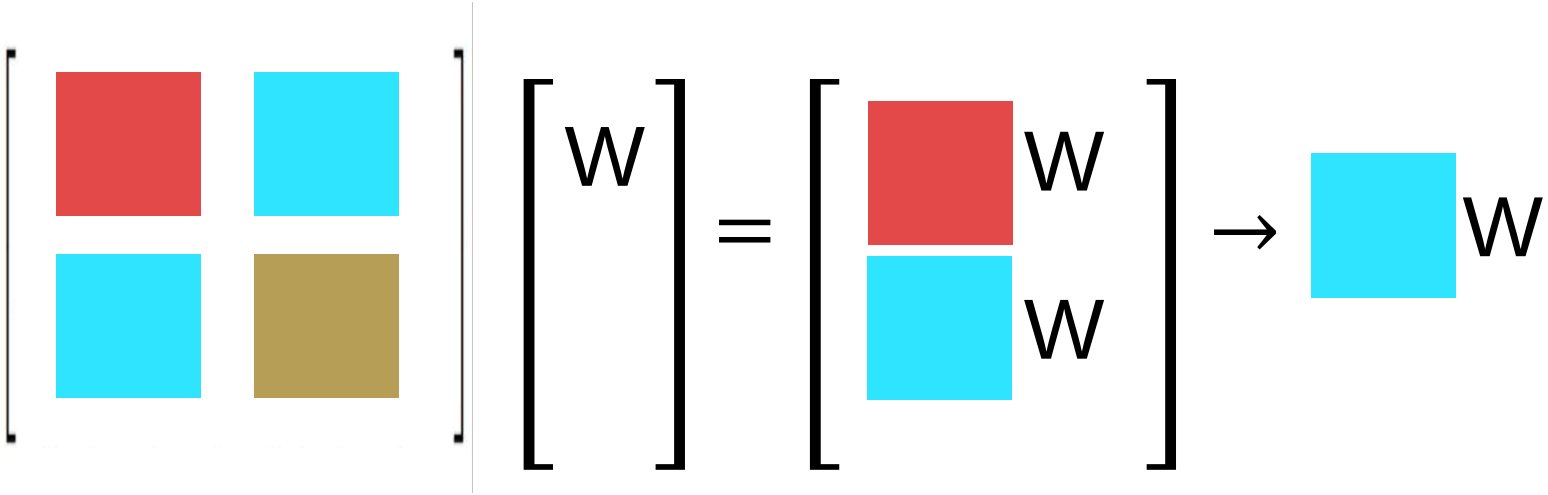
2D results: tolerance parameter 10^{-9}

- Seven digits of accuracy in direct solve, or conjugate gradient converges to a relative residual norm of 10^{-12} in about 2 preconditioned iterations.





Matrix peeling



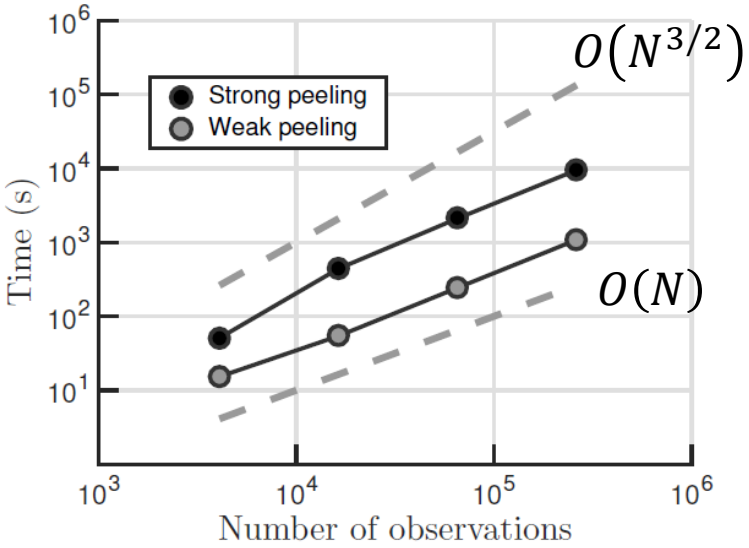
- Apply black-box operator to structured random vectors to get samples of range of off-diagonal blocks
- Use randomized SVD algorithm to get low-rank representation [Halko et al., 2011]

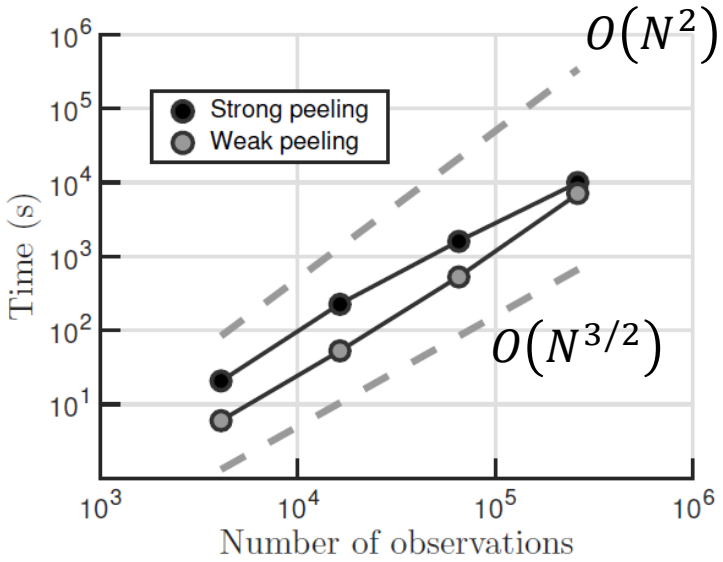
Results for peeling

$$K(x,y) = \exp\left(-\|x-y\|^2\right)$$

$$K(x,y) = \frac{1}{\Gamma(\nu)2^{\nu-1}}\left(\sqrt{2\nu}\cdot\|x-y\|\right)^{\nu} K_{\nu}\left(\sqrt{2\nu}\cdot\|x-y\|\right)$$

- Computing the product-trace term using peeling
- Gridded observations in 2D with squared-exponential kernel (left) and Matérn family kernel (right) at accuracy 1E-6





Converting a PDE to an integral equation

- Suppose $\Delta u = 0$ on D and $u = f$ on ∂D (Laplace BVP)
- Then, since $\int_D (u\Delta v - v\Delta u)dx = \int_{\partial D}\left(u\frac{\partial v(y)}{\partial n(y)} - \frac{\partial u(y)}{\partial n(y)}v\right)ds(y)$ for any nice *u* and *v* on *D*, we have for our u and for any x in D that

$$u(x) = \int_{\partial D}\left(G(x,y)\frac{\partial u(y)}{\partial n(y)} - \frac{\partial G(x,y)}{\partial n(y)}u(y)\right)ds(y)$$

- In reality, can usually use just one of these, giving for example

$$u(x) = \int_{\partial D} G(x,y)\phi(y)ds(y)$$

$$\mathrm{f}(z) = \int_{\partial D} G(z,y)\phi(y)ds(y)$$

The proxy trick for fast compression

- Far away point x_i, interior point x_j
- Use a Green's identity to express Green's function at x_j :

$$G(x_i - x_j) = \int_\Gamma \psi_i(y) G(x_j - y) ds(y)$$

- Similar "adjoint" idea holds as well by considering problem on complementary domain