Rank-structured preconditioners for two- and three-dimensional integral and differential equations

Steffen Börm
with S. Christophersen and K. Reimer

University of Kiel, Germany

SIAM CSE, 17th of March, 2015

Overview

1. $\mathcal{H}^2$-matrices
2. $\mathcal{H}^2$-matrix arithmetic operations
3. Summary

Overview

1. $\mathcal{H}^2$-matrices
2. $\mathcal{H}^2$-matrix arithmetic operations
3. Summary

Model problem

Goal: Given an asymptotically smooth kernel function, solve

$$\int_\Omega g(x,y)u(y)\,dy = f(x) \qquad \text{for all } x \in \Omega.$$

Discretization leads to system $Gu = b$ with a matrix $G \in \mathbb{R}^{\mathcal{I}\times\mathcal{I}}$.

$$g_{ij} = \int_\Omega \varphi_i(x) \int_\omega g(x,y)\varphi_j(y)\,dy\,dx \quad \text{or}$$
$$g_{ij} = g(x_i, y_j) \qquad \text{for } i,j \in \mathcal{I}.$$

Problem: High accuracy requires large number of basis functions.

Idea: Approximate G by a matrix that can be represented efficiently.

Uniform $\mathcal{H}$-matrix

Idea: Locally approximate kernel function, e.g., by interpolation

$$g_{ij} = g(x_i, y_j) \approx \sum_{\nu=1}^{k} \sum_{\mu=1}^{k} \mathcal{L}_{\tau,\nu}(x_i) g(\xi_{\tau,\nu}, \xi_{\sigma,\mu}) \mathcal{L}_{\sigma,\mu}(y_j).$$

for clusters $\tau, \sigma \subseteq \Omega$, $x_i \in \tau$ and $y_j \in \sigma$.

Uniform $\mathcal{H}$-matrix

Idea: Locally approximate kernel function, e.g., by interpolation

$$g_{ij} = g(x_i, y_j) \approx \sum_{\nu=1}^{k} \sum_{\mu=1}^{k} \underbrace{\mathcal{L}_{\tau,\nu}(x_i)}_{=v_{\tau,i\nu}} \underbrace{g(\xi_{\tau,\nu}, \xi_{\sigma,\mu})}_{=s_{\tau\sigma,\nu\mu}} \underbrace{\mathcal{L}_{\sigma,\mu}(y_j)}_{=w_{\sigma,j\mu}}.$$

for clusters $\tau, \sigma \subseteq \Omega$, $x_i \in \tau$ and $y_j \in \sigma$.

Uniform $\mathcal{H}$-matrix

Idea: Locally approximate kernel function, e.g., by interpolation

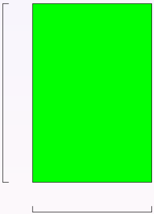
$$g_{ij} = g(x_i, y_j) \approx \sum_{\nu=1}^{k} \sum_{\mu=1}^{k} \underbrace{\mathcal{L}_{\tau,\nu}(x_i)}_{=v_{\tau,i\nu}} \underbrace{g(\xi_{\tau,\nu}, \xi_{\sigma,\mu})}_{=s_{\tau\sigma,\nu\mu}} \underbrace{\mathcal{L}_{\sigma,\mu}(y_j)}_{=w_{\sigma,j\mu}}.$$

for clusters $\tau, \sigma \subseteq \Omega$, $x_i \in \tau$ and $y_j \in \sigma$.

Block approximation by three-term factorization

$$G|_{\hat\tau\times\hat\sigma} \approx V_\tau S_{\tau\sigma} W_\sigma^*.$$

with $\hat\tau := \{i \in \mathcal{I} \;:\; x_i \in \tau\}$ and $\hat\sigma := \{j \in \mathcal{I} \;:\; y_j \in \sigma\}$.



Uniform $\mathcal{H}$-matrix

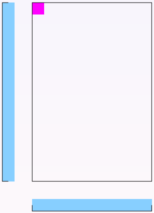
Idea: Locally approximate kernel function, e.g., by interpolation

$$g_{ij} = g(x_i, y_j) \approx \sum_{\nu=1}^{k} \sum_{\mu=1}^{k} \underbrace{\mathcal{L}_{\tau,\nu}(x_i)}_{=v_{\tau,i\nu}} \underbrace{g(\xi_{\tau,\nu}, \xi_{\sigma,\mu})}_{=s_{\tau\sigma,\nu\mu}} \underbrace{\mathcal{L}_{\sigma,\mu}(y_j)}_{=w_{\sigma,j\mu}}.$$

for clusters $\tau, \sigma \subseteq \Omega$, $x_i \in \tau$ and $y_j \in \sigma$.

Block approximation by three-term factorization

$$G|_{\hat{\tau}\times\hat{\sigma}} \approx V_\tau S_{\tau\sigma} W_\sigma^*.$$



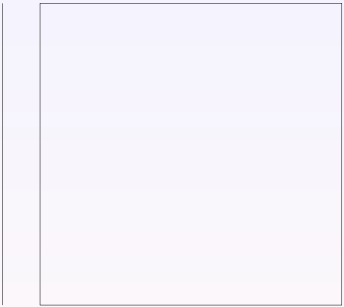
with $\hat{\tau} := \{i \in \mathcal{I} \ : \ x_i \in \tau\}$ and $\hat{\sigma} := \{j \in \mathcal{I} \ : \ y_j \in \sigma\}$.

Cluster bases: V_τ depends only on τ, W_σ only on σ.

Coupling matrices $S_{\tau\sigma}$ are $k \times k$-matrices.

Block tree

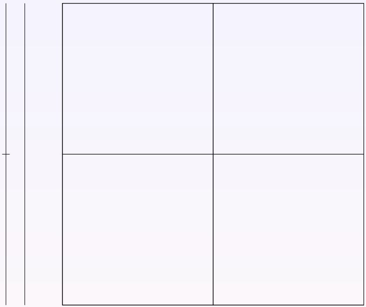
Problem: Since the kernel function is only locally smooth, split $\Omega \times \Omega$ into subdomains $\tau \times \sigma$ admitting an approximation.



1D case: $\tau \times \sigma$ admissible if $\operatorname{diam}(\tau) \leq \operatorname{dist}(\tau, \sigma)$.

Block tree

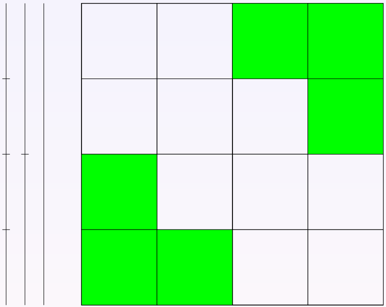
Problem: Since the kernel function is only locally smooth, split $\Omega \times \Omega$ into subdomains $\tau \times \sigma$ admitting an approximation.



1D case: $\tau \times \sigma$ admissible if $\operatorname{diam}(\tau) \leq \operatorname{dist}(\tau, \sigma)$.

Block tree

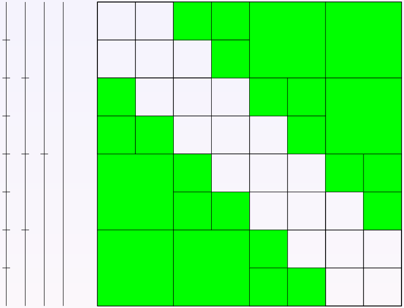
Problem: Since the kernel function is only locally smooth, split $\Omega \times \Omega$ into subdomains $\tau \times \sigma$ admitting an approximation.



1D case: $\tau \times \sigma$ admissible if $\operatorname{diam}(\tau) \leq \operatorname{dist}(\tau, \sigma)$.

Block tree

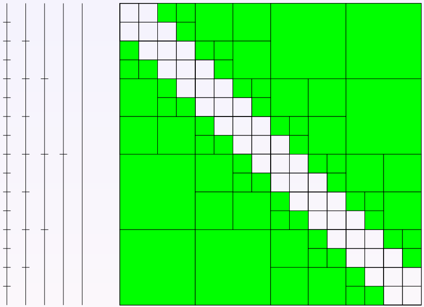
Problem: Since the kernel function is only locally smooth, split $\Omega \times \Omega$ into subdomains $\tau \times \sigma$ admitting an approximation.



1D case: $\tau \times \sigma$ admissible if $\mathrm{diam}(\tau) \leq \mathrm{dist}(\tau, \sigma)$.

Block tree

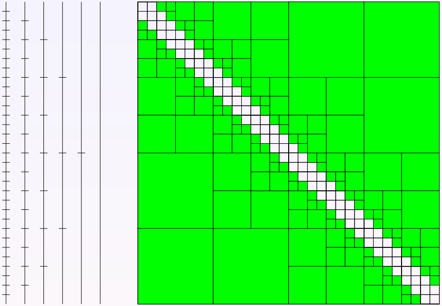
Problem: Since the kernel function is only locally smooth, split $\Omega \times \Omega$ into subdomains $\tau \times \sigma$ admitting an approximation.



1D case: $\tau \times \sigma$ admissible if $\mathrm{diam}(\tau) \leq \mathrm{dist}(\tau, \sigma)$.

Block tree

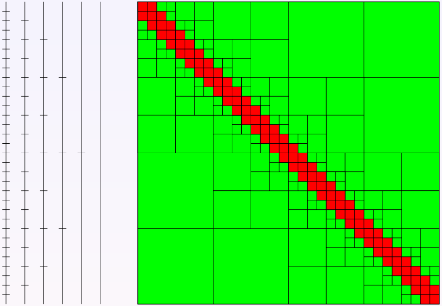
Problem: Since the kernel function is only locally smooth, split $\Omega \times \Omega$ into subdomains $\tau \times \sigma$ admitting an approximation.



1D case: $\tau \times \sigma$ admissible if $\mathrm{diam}(\tau) \leq \mathrm{dist}(\tau, \sigma)$.

Block tree

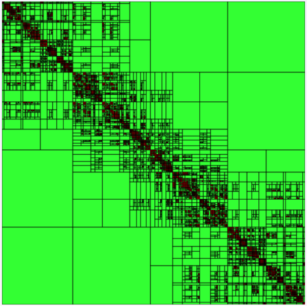
Problem: Since the kernel function is only locally smooth, split $\Omega \times \Omega$ into subdomains $\tau \times \sigma$ admitting an approximation.



1D case: $\tau \times \sigma$ admissible if $\operatorname{diam}(\tau) \leq \operatorname{dist}(\tau, \sigma)$.

Block tree

Problem: Since the kernel function is only locally smooth, split $\Omega \times \Omega$ into subdomains $\tau \times \sigma$ admitting an approximation.

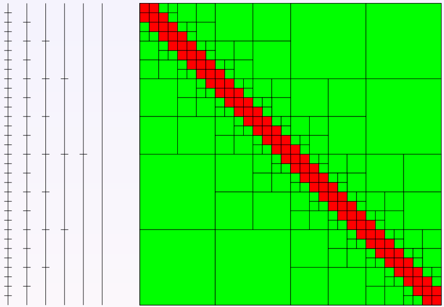


1D case: $\tau \times \sigma$ admissible if $\operatorname{diam}(\tau) \leq \operatorname{dist}(\tau, \sigma)$.

General case: Block structure can be significantly more involved.

Block tree

Problem: Since the kernel function is only locally smooth, split $\Omega \times \Omega$ into subdomains $\tau \times \sigma$ admitting an approximation.

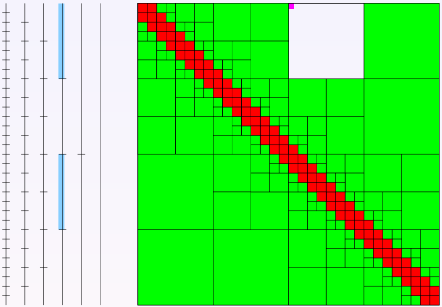


1D case: $\tau \times \sigma$ admissible if $\operatorname{diam}(\tau) \leq \operatorname{dist}(\tau, \sigma)$.

General case: Block structure can be significantly more involved.

Block tree

Problem: Since the kernel function is only locally smooth, split $\Omega \times \Omega$ into subdomains $\tau \times \sigma$ admitting an approximation.



1D case: $\tau \times \sigma$ admissible if $\mathrm{diam}(\tau) \leq \mathrm{dist}(\tau, \sigma)$.

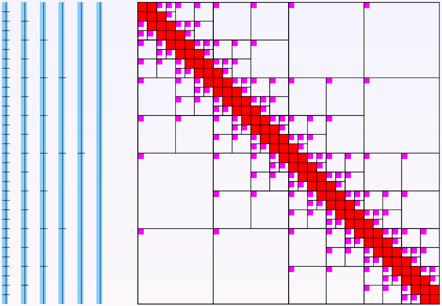
General case: Block structure can be significantly more involved.

Admissible blocks: Factorization

$$G|_{\hat\tau \times \hat\sigma} \approx V_\tau S_{\tau\sigma} W_\sigma^*.$$

Block tree

Problem: Since the kernel function is only locally smooth, split $\Omega \times \Omega$ into subdomains $\tau \times \sigma$ admitting an approximation.



1D case: $\tau \times \sigma$ admissible if $\operatorname{diam}(\tau) \leq \operatorname{dist}(\tau, \sigma)$.

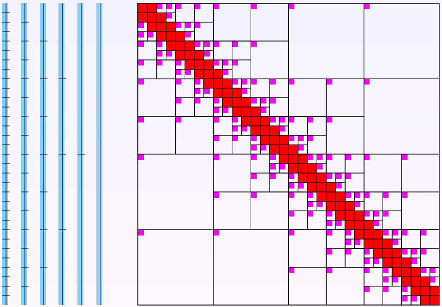
General case: Block structure can be significantly more involved.

Admissible blocks: Factorization

$$G|_{\hat{\tau} \times \hat{\sigma}} \approx V_\tau S_{\tau\sigma} W_\sigma^*.$$

Block tree

Problem: Since the kernel function is only locally smooth, split $\Omega \times \Omega$ into subdomains $\tau \times \sigma$ admitting an approximation.



1D case: $\tau \times \sigma$ admissible if $\operatorname{diam}(\tau) \leq \operatorname{dist}(\tau, \sigma)$.

General case: Block structure can be significantly more involved.

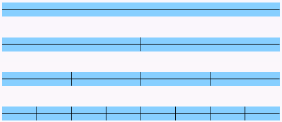
Admissible blocks: Factorization

$$G|_{\hat\tau \times \hat\sigma} \approx V_\tau S_{\tau\sigma} W_\sigma^*.$$

Result: $\mathcal{O}(nk)$ storage for coupling matrices $(S_{\tau\sigma})$, but $\mathcal{O}(nk \log n)$ for cluster bases (V_τ) and (W_σ).

Cluster basis

Challenge: Cluster basis $(V_\tau)_\tau$ requires storage $\sim nk \log n$.

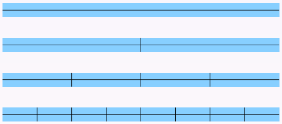


Cluster basis

Challenge: Cluster basis $(V_\tau)_\tau$ requires storage $\sim nk \log n$.

Idea: Interpolation in $\tau' \in \mathrm{sons}(\tau)$ reproduces $\mathcal{L}_{t,\nu}$.

$$v_{\tau,i\nu} = \mathcal{L}_{\tau,\nu}(x_i) = \sum_{\mu=1}^{k} \mathcal{L}_{\tau',\mu}(x_i)\mathcal{L}_{\tau,\nu}(\xi_{\tau',\mu})$$

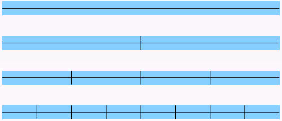


Cluster basis

Challenge: Cluster basis $(V_\tau)_\tau$ requires storage $\sim nk \log n$.

Idea: Interpolation in $\tau' \in \mathrm{sons}(\tau)$ reproduces $\mathcal{L}_{t,\nu}$.

$$v_{\tau,i\nu} = \mathcal{L}_{\tau,\nu}(x_i) = \sum_{\mu=1}^{k} \underbrace{\mathcal{L}_{\tau',\mu}(x_i)}_{=v_{\tau',i\mu}} \underbrace{\mathcal{L}_{\tau,\nu}(\xi_{\tau',\mu})}_{=e_{\tau',\mu\nu}}$$



Cluster basis

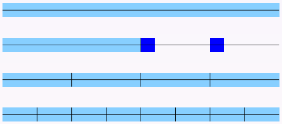
Challenge: Cluster basis $(V_\tau)_\tau$ requires storage $\sim nk \log n$.

Idea: Interpolation in $\tau' \in \mathrm{sons}(\tau)$ reproduces $\mathcal{L}_{t,\nu}$.

$$v_{\tau,i\nu} = \mathcal{L}_{\tau,\nu}(x_i) = \sum_{\mu=1}^{k} \underbrace{\mathcal{L}_{\tau',\mu}(x_i)}_{=v_{\tau',i\mu}} \underbrace{\mathcal{L}_{\tau,\nu}(\xi_{\tau',\mu})}_{=e_{\tau',\mu\nu}}$$

Transfer matrices can be stored instead of V_τ:

$$V_\tau = \begin{pmatrix} V_{\tau_1} E_{\tau_1} \\ V_{\tau_2} E_{\tau_2} \end{pmatrix}, \qquad \mathrm{sons}(\tau) = \{\tau_1, \tau_2\}.$$



Store V_τ only for leaves.

Cluster basis

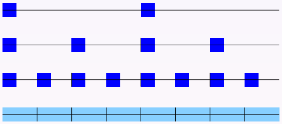
Challenge: Cluster basis $(V_\tau)_\tau$ requires storage $\sim nk \log n$.

Idea: Interpolation in $\tau' \in \mathrm{sons}(\tau)$ reproduces $\mathcal{L}_{t,\nu}$.

$$v_{\tau,i\nu} = \mathcal{L}_{\tau,\nu}(x_i) = \sum_{\mu=1}^{k} \underbrace{\mathcal{L}_{\tau',\mu}(x_i)}_{=v_{\tau',i\mu}} \underbrace{\mathcal{L}_{\tau,\nu}(\xi_{\tau',\mu})}_{=e_{\tau',\mu\nu}}$$

Transfer matrices can be stored instead of V_τ:

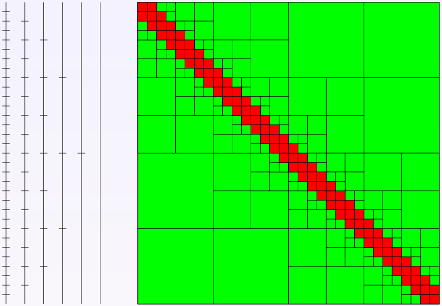
$$V_\tau = \begin{pmatrix} V_{\tau_1} E_{\tau_1} \\ V_{\tau_2} E_{\tau_2} \end{pmatrix}, \qquad \mathrm{sons}(\tau) = \{\tau_1, \tau_2\}.$$



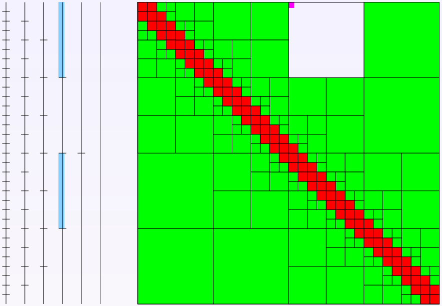
Store V_τ only for leaves.

Result: Storage $\sim nk$.

$\mathcal{H}^2$-matrix



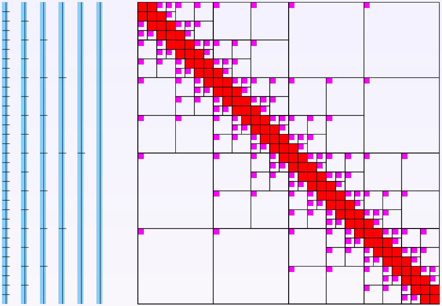
$\mathcal{H}^2$-matrix



Admissible blocks represented by coupling matrices

$$G|_{\hat{\tau}\times\hat{\sigma}} \approx V_\tau S_{\tau\sigma} W_\sigma^*.$$

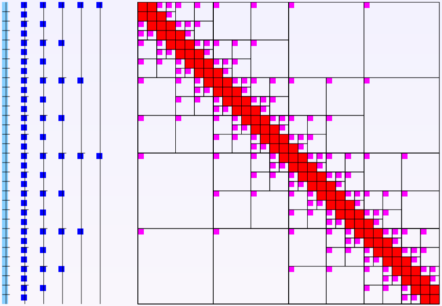
$\mathcal{H}^2$-matrix



Admissible blocks represented by coupling matrices

$$G|_{\hat{\tau}\times\hat{\sigma}} \approx V_\tau S_{\tau\sigma} W_\sigma^*.$$

$\mathcal{H}^2$-matrix



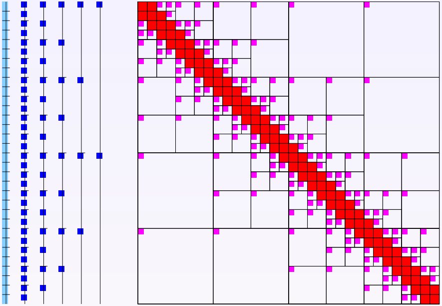
Admissible blocks represented by coupling matrices

$$G|_{\hat{\tau}\times\hat{\sigma}} \approx V_\tau S_{\tau\sigma} W_\sigma^*.$$

Cluster bases represented by transfer matrices

$$V_\tau|_{\hat{\tau}'\times k} = V_{\tau'} E_{\tau'}.$$

$\mathcal{H}^2$-matrix



Admissible blocks represented by coupling matrices

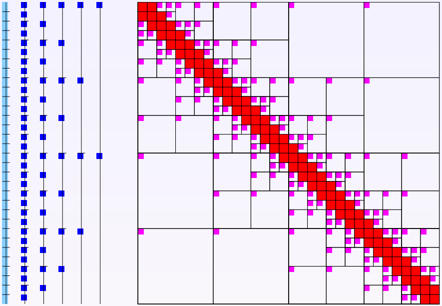
$$G|_{\hat{\tau}\times\hat{\sigma}} \approx V_\tau S_{\tau\sigma} W_\sigma^*.$$

Cluster bases represented by transfer matrices

$$V_\tau|_{\hat{\tau}'\times k} = V_{\tau'} E_{\tau'}.$$

Result: $\mathcal{H}^2$-matrix, storage $\mathcal{O}(nk)$.

$\mathcal{H}^2$-matrix



Admissible blocks represented by coupling matrices

$$G|_{\hat{\tau}\times\hat{\sigma}} \approx V_\tau S_{\tau\sigma} W_\sigma^*.$$

Cluster bases represented by transfer matrices

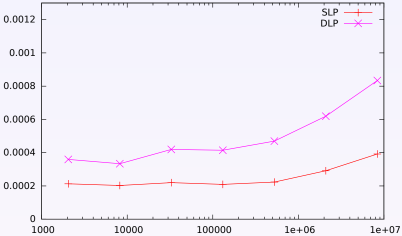
$$V_\tau|_{\hat{\tau}'\times k} = V_{\tau'} E_{\tau'}.$$

Result: $\mathcal{H}^2$-matrix, storage $\mathcal{O}(nk)$.

$\mathcal{H}^2$-matrices are the algebraic counterparts of fast multipole representations.

Experiment: Unit sphere

Example: Dirichlet problem on the unit sphere, direct formulation.

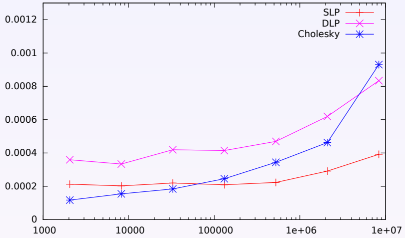


Time for setup:

Less than 1+2 hours for
more than 8 million triangles.

Experiment: Unit sphere

Example: Dirichlet problem on the unit sphere, direct formulation.



Time for setup:

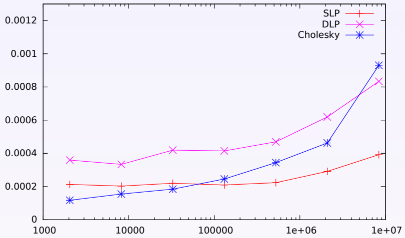
Less than 1+2 hours for more than 8 million triangles.

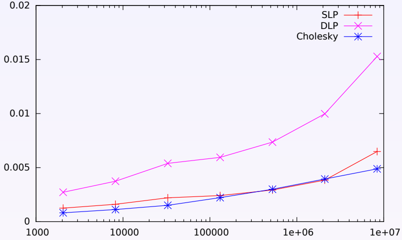
$\mathcal{H}$-Cholesky preconditioner:

More than 2 hours.

Experiment: Unit sphere

Example: Dirichlet problem on the unit sphere, direct formulation.





Time for setup:
Less than 1+2 hours for
more than 8 million triangles.

$\mathcal{H}$-Cholesky preconditioner:
More than 2 hours.

Storage for matrices:
Less than 55+128 GB for
more than 8 million triangles.

$\mathcal{H}$-Cholesky preconditioner:
More than 41 GB.

Overview

LR factorization

Goal: Given an $\mathcal{H}^2$-matrix G, compute its LR factorization $G = LR$.

Approach: If G is inadmissible, compute $G = LR$ directly.
Otherwise, use submatrices

$$\begin{pmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} R_{11} & R_{12} \\ & R_{22} \end{pmatrix}$$

LR factorization

Goal: Given an $\mathcal{H}^2$-matrix G, compute its LR factorization $G = LR$.

Approach: If G is inadmissible, compute $G = LR$ directly.
Otherwise, use submatrices

$$\begin{pmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} R_{11} & R_{12} \\ & R_{22} \end{pmatrix}$$

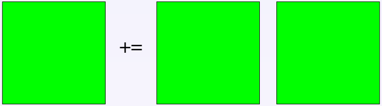
This is equivalent to

$$G_{11} = L_{11} R_{11},$$
$$G_{12} = L_{11} R_{12}, \qquad G_{21} = L_{21} R_{11},$$
$$G_{22} - L_{21} R_{12} = L_{22} R_{22}.$$

If we can compute $Z \leftarrow Z + \alpha XY$, we can find the LR factorization.

Multiplication

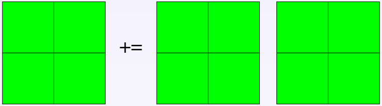
Goal: Perform update $Z|_{\hat{\tau}\times\hat{\rho}} \leftarrow Z|_{\hat{\tau}\times\hat{\rho}} + \alpha X|_{\hat{\tau}\times\hat{\sigma}} Y|_{\hat{\sigma}\times\hat{\rho}}$ efficiently.



Idea: If (τ, σ) and (σ, ρ) are subdivided, treat them by recursion.

Multiplication

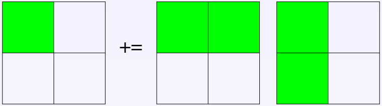
Goal: Perform update $Z|_{\hat{\tau}\times\hat{\rho}} \leftarrow Z|_{\hat{\tau}\times\hat{\rho}} + \alpha X|_{\hat{\tau}\times\hat{\sigma}} Y|_{\hat{\sigma}\times\hat{\rho}}$ efficiently.



Idea: If (τ, σ) and (σ, ρ) are subdivided, treat them by recursion.

Multiplication

Goal: Perform update $Z|_{\hat\tau\times\hat\rho} \leftarrow Z|_{\hat\tau\times\hat\rho} + \alpha X|_{\hat\tau\times\hat\sigma} Y|_{\hat\sigma\times\hat\rho}$ efficiently.

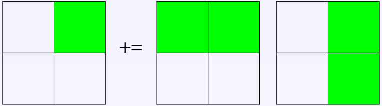


Idea: If (τ,σ) and (σ,ρ) are subdivided, treat them by recursion.

- $Z|_{\hat\tau_1\times\hat\rho_1} \leftarrow Z|_{\hat\tau_1\times\hat\rho_1} + \alpha X|_{\hat\tau_1\times\hat\sigma_1} Y|_{\hat\sigma_1\times\hat\rho_1} + \alpha X|_{\hat\tau_1\times\hat\sigma_2} Y|_{\hat\sigma_2\times\hat\rho_1}$

Multiplication

Goal: Perform update $Z|_{\hat\tau\times\hat\rho} \leftarrow Z|_{\hat\tau\times\hat\rho} + \alpha X|_{\hat\tau\times\hat\sigma} Y|_{\hat\sigma\times\hat\rho}$ efficiently.

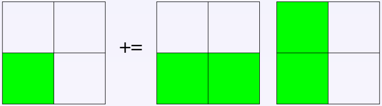


Idea: If (τ, σ) and (σ, ρ) are subdivided, treat them by recursion.

- $Z|_{\hat\tau_1\times\hat\rho_1} \leftarrow Z|_{\hat\tau_1\times\hat\rho_1} + \alpha X|_{\hat\tau_1\times\hat\sigma_1} Y|_{\hat\sigma_1\times\hat\rho_1} + \alpha X|_{\hat\tau_1\times\hat\sigma_2} Y|_{\hat\sigma_2\times\hat\rho_1}$
- $Z|_{\hat\tau_1\times\hat\rho_2} \leftarrow Z|_{\hat\tau_1\times\hat\rho_2} + \alpha X|_{\hat\tau_1\times\hat\sigma_1} Y|_{\hat\sigma_1\times\hat\rho_2} + \alpha X|_{\hat\tau_1\times\hat\sigma_2} Y|_{\hat\sigma_2\times\hat\rho_2}$

Multiplication

Goal: Perform update $Z|_{\hat{\tau}\times\hat{\rho}} \leftarrow Z|_{\hat{\tau}\times\hat{\rho}} + \alpha X|_{\hat{\tau}\times\hat{\sigma}} Y|_{\hat{\sigma}\times\hat{\rho}}$ efficiently.

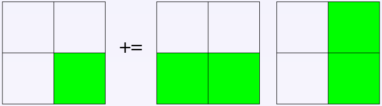


Idea: If (τ, σ) and (σ, ρ) are subdivided, treat them by recursion.

- $Z|_{\hat{\tau}_1\times\hat{\rho}_1} \leftarrow Z|_{\hat{\tau}_1\times\hat{\rho}_1} + \alpha X|_{\hat{\tau}_1\times\hat{\sigma}_1} Y|_{\hat{\sigma}_1\times\hat{\rho}_1} + \alpha X|_{\hat{\tau}_1\times\hat{\sigma}_2} Y|_{\hat{\sigma}_2\times\hat{\rho}_1}$
- $Z|_{\hat{\tau}_1\times\hat{\rho}_2} \leftarrow Z|_{\hat{\tau}_1\times\hat{\rho}_2} + \alpha X|_{\hat{\tau}_1\times\hat{\sigma}_1} Y|_{\hat{\sigma}_1\times\hat{\rho}_2} + \alpha X|_{\hat{\tau}_1\times\hat{\sigma}_2} Y|_{\hat{\sigma}_2\times\hat{\rho}_2}$
- $Z|_{\hat{\tau}_2\times\hat{\rho}_1} \leftarrow Z|_{\hat{\tau}_2\times\hat{\rho}_1} + \alpha X|_{\hat{\tau}_2\times\hat{\sigma}_1} Y|_{\hat{\sigma}_1\times\hat{\rho}_1} + \alpha X|_{\hat{\tau}_2\times\hat{\sigma}_2} Y|_{\hat{\sigma}_2\times\hat{\rho}_1}$

Multiplication

Goal: Perform update $Z|_{\hat{\tau}\times\hat{\rho}} \leftarrow Z|_{\hat{\tau}\times\hat{\rho}} + \alpha X|_{\hat{\tau}\times\hat{\sigma}} Y|_{\hat{\sigma}\times\hat{\rho}}$ efficiently.

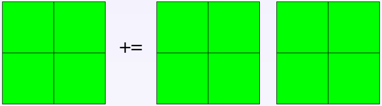


Idea: If (τ, σ) and (σ, ρ) are subdivided, treat them by recursion.

- $Z|_{\hat{\tau}_1\times\hat{\rho}_1} \leftarrow Z|_{\hat{\tau}_1\times\hat{\rho}_1} + \alpha X|_{\hat{\tau}_1\times\hat{\sigma}_1} Y|_{\hat{\sigma}_1\times\hat{\rho}_1} + \alpha X|_{\hat{\tau}_1\times\hat{\sigma}_2} Y|_{\hat{\sigma}_2\times\hat{\rho}_1}$
- $Z|_{\hat{\tau}_1\times\hat{\rho}_2} \leftarrow Z|_{\hat{\tau}_1\times\hat{\rho}_2} + \alpha X|_{\hat{\tau}_1\times\hat{\sigma}_1} Y|_{\hat{\sigma}_1\times\hat{\rho}_2} + \alpha X|_{\hat{\tau}_1\times\hat{\sigma}_2} Y|_{\hat{\sigma}_2\times\hat{\rho}_2}$
- $Z|_{\hat{\tau}_2\times\hat{\rho}_1} \leftarrow Z|_{\hat{\tau}_2\times\hat{\rho}_1} + \alpha X|_{\hat{\tau}_2\times\hat{\sigma}_1} Y|_{\hat{\sigma}_1\times\hat{\rho}_1} + \alpha X|_{\hat{\tau}_2\times\hat{\sigma}_2} Y|_{\hat{\sigma}_2\times\hat{\rho}_1}$
- $Z|_{\hat{\tau}_2\times\hat{\rho}_2} \leftarrow Z|_{\hat{\tau}_2\times\hat{\rho}_2} + \alpha X|_{\hat{\tau}_2\times\hat{\sigma}_1} Y|_{\hat{\sigma}_1\times\hat{\rho}_2} + \alpha X|_{\hat{\tau}_2\times\hat{\sigma}_2} Y|_{\hat{\sigma}_2\times\hat{\rho}_2}$

Multiplication

Goal: Perform update $Z|_{\hat\tau\times\hat\rho} \leftarrow Z|_{\hat\tau\times\hat\rho} + \alpha X|_{\hat\tau\times\hat\sigma} Y|_{\hat\sigma\times\hat\rho}$ efficiently.



Idea: If (τ,σ) and (σ,ρ) are subdivided, treat them by recursion.

- $Z|_{\hat\tau_1\times\hat\rho_1} \leftarrow Z|_{\hat\tau_1\times\hat\rho_1} + \alpha X|_{\hat\tau_1\times\hat\sigma_1} Y|_{\hat\sigma_1\times\hat\rho_1} + \alpha X|_{\hat\tau_1\times\hat\sigma_2} Y|_{\hat\sigma_2\times\hat\rho_1}$
- $Z|_{\hat\tau_1\times\hat\rho_2} \leftarrow Z|_{\hat\tau_1\times\hat\rho_2} + \alpha X|_{\hat\tau_1\times\hat\sigma_1} Y|_{\hat\sigma_1\times\hat\rho_2} + \alpha X|_{\hat\tau_1\times\hat\sigma_2} Y|_{\hat\sigma_2\times\hat\rho_2}$
- $Z|_{\hat\tau_2\times\hat\rho_1} \leftarrow Z|_{\hat\tau_2\times\hat\rho_1} + \alpha X|_{\hat\tau_2\times\hat\sigma_1} Y|_{\hat\sigma_1\times\hat\rho_1} + \alpha X|_{\hat\tau_2\times\hat\sigma_2} Y|_{\hat\sigma_2\times\hat\rho_1}$
- $Z|_{\hat\tau_2\times\hat\rho_2} \leftarrow Z|_{\hat\tau_2\times\hat\rho_2} + \alpha X|_{\hat\tau_2\times\hat\sigma_1} Y|_{\hat\sigma_1\times\hat\rho_2} + \alpha X|_{\hat\tau_2\times\hat\sigma_2} Y|_{\hat\sigma_2\times\hat\rho_2}$

Multiplication with a leaf block

Important case: What happens if one of the factors is not subdivided?

Multiplication with a leaf block

Important case: What happens if one of the factors is not subdivided?

Example: Let (σ, ρ) be an admissible block of Y.

$$X|_{\hat{\tau}\times\hat{\sigma}} Y|_{\hat{\sigma}\times\hat{\rho}} = X|_{\hat{\tau}\times\hat{\sigma}} (V_\sigma S_{\sigma\rho} W_\rho^*)$$

Multiplication with a leaf block

Important case: What happens if one of the factors is not subdivided?

Example: Let (σ, ρ) be an admissible block of Y.

$$X|_{\hat\tau\times\hat\sigma} Y|_{\hat\sigma\times\hat\rho} = X|_{\hat\tau\times\hat\sigma}(V_\sigma S_{\sigma\rho} W_\rho^*) = (X|_{\hat\tau\times\hat\sigma} V_\sigma S_{\sigma\rho}) W_\rho^*$$

Multiplication with a leaf block

Important case: What happens if one of the factors is not subdivided?

Example: Let (σ, ρ) be an admissible block of Y.

$$X|_{\hat\tau\times\hat\sigma} Y|_{\hat\sigma\times\hat\rho} = X|_{\hat\tau\times\hat\sigma}(V_\sigma S_{\sigma\rho} W_\rho^*) = (X|_{\hat\tau\times\hat\sigma} V_\sigma S_{\sigma\rho}) W_\rho^* = A_{\tau\rho} W_\rho^*.$$

Multiplication with a leaf block

Important case: What happens if one of the factors is not subdivided?

Example: Let (σ, ρ) be an admissible block of Y.

$$X|_{\hat{\tau}\times\hat{\sigma}} Y|_{\hat{\sigma}\times\hat{\rho}} = X|_{\hat{\tau}\times\hat{\sigma}}(V_\sigma S_{\sigma\rho} W_\rho^*) = (X|_{\hat{\tau}\times\hat{\sigma}} V_\sigma S_{\sigma\rho}) W_\rho^* = A_{\tau\rho} W_\rho^*.$$

Idea: $A_{\tau\rho}$ can be computed by fast matrix-vector multiplications.
We "only" need an efficient algorithm for low-rank updates

$$Z|_{\hat{\tau}\times\hat{\rho}} \leftarrow Z|_{\hat{\tau}\times\hat{\rho}} + \alpha A_{\tau\rho} W_\rho^*.$$

Multiplication with a leaf block

Important case: What happens if one of the factors is not subdivided?

Example: Let (σ, ρ) be an admissible block of Y.

$$X|_{\hat\tau\times\hat\sigma} Y|_{\hat\sigma\times\hat\rho} = X|_{\hat\tau\times\hat\sigma}(V_\sigma S_{\sigma\rho} W_\rho^*) = (X|_{\hat\tau\times\hat\sigma} V_\sigma S_{\sigma\rho}) W_\rho^* = A_{\tau\rho} W_\rho^*.$$

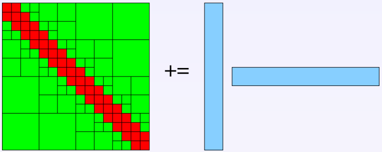
Idea: $A_{\tau\rho}$ can be computed by fast matrix-vector multiplications.
We “only” need an efficient algorithm for low-rank updates

$$Z|_{\hat\tau\times\hat\rho} \leftarrow Z|_{\hat\tau\times\hat\rho} + \alpha A_{\tau\rho} W_\rho^*.$$

General case: If (τ, σ) or (σ, ρ) are leaves, the product $X|_{\hat\tau\times\hat\sigma} Y|_{\hat\sigma\times\hat\rho}$ is always a low-rank matrix in factorized representation.

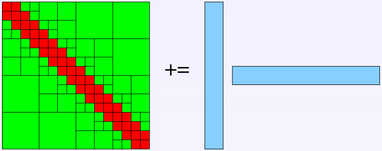
Low-rank update

Goal: Approximate $Z + AB^*$.



Low-rank update

Goal: Approximate $Z + AB^*$.

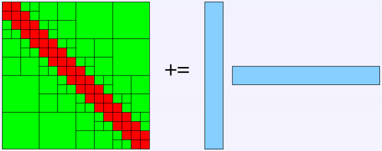


Idea: Result $Z + AB^*$ is an $\mathcal{H}^2$-matrix: for admissible (τ, σ) we have

$$
\begin{aligned}
(Z + AB^*)|_{\hat\tau\times\hat\sigma} &= V_\tau S_{\tau\sigma} W_\sigma^* + A|_{\hat\tau\times k} B|_{\hat\sigma\times k}^* \\
&= \underbrace{\begin{pmatrix} V_\tau & A|_{\hat\tau\times k} \end{pmatrix}}_{=:\widetilde{V}_\tau} \underbrace{\begin{pmatrix} S_{\tau\sigma} & \\ & I \end{pmatrix}}_{=:\widetilde{S}_{\tau\sigma}} \underbrace{\begin{pmatrix} W_\sigma & B|_{\hat\sigma\times k} \end{pmatrix}^*}_{=:\widetilde{W}_\sigma^*}
\end{aligned}
$$

Low-rank update

Goal: Approximate $Z + AB^*$.



Idea: Result $Z + AB^*$ is an $\mathcal{H}^2$-matrix: for admissible (τ, σ) we have

$$\begin{aligned}(Z + AB^*)|_{\hat\tau\times\hat\sigma} &= V_\tau S_{\tau\sigma} W_\sigma^* + A|_{\hat\tau\times k} B|_{\hat\sigma\times k}^* \\ &= \underbrace{\begin{pmatrix} V_\tau & A|_{\hat\tau\times k}\end{pmatrix}}_{=:\widetilde{V}_\tau} \underbrace{\begin{pmatrix} S_{\tau\sigma} & \\ & I \end{pmatrix}}_{=:\widetilde{S}_{\tau\sigma}} \underbrace{\begin{pmatrix} W_\sigma & B|_{\hat\sigma\times k}\end{pmatrix}^*}_{=:\widetilde{W}_\sigma^*}\end{aligned}$$

Challenge: Each update increases storage requirements, although actual numerical rank may be low. $\rightarrow$ Recompression required.

Recompression

Goal: Given an $\mathcal{H}^2$-matrix with unnecessarily high rank, construct a more efficient approximation.

Row basis: Find orthogonal $(Q_\tau)_\tau$ such that

$$Q_\tau Q_\tau^* G|_{\tau\times\sigma} \approx G|_{\tau\times\sigma} \qquad \text{for all admissible } (\tau,\sigma).$$

Recompression

Goal: Given an $\mathcal{H}^2$-matrix with unnecessarily high rank, construct a more efficient approximation.

Row basis: Find orthogonal $(Q_\tau)_\tau$ such that

$$Q_\tau Q_\tau^* G|_{\tau\times\sigma} \approx G|_{\tau\times\sigma} \qquad \text{for all admissible } (\tau,\sigma).$$

Idea: Due to $G|_{\tau\times\sigma} = V_\tau S_{\tau\sigma} W_\sigma^*$, it suffices to ensure

$$Q_\tau Q_\tau^* V_\tau Z_\tau^* \approx V_\tau Z_\tau^*,$$

where Z_τ is a small weight matrix. $\rightarrow$ Solve by SVD or RRQR.

Recompression

Goal: Given an $\mathcal{H}^2$-matrix with unnecessarily high rank, construct a more efficient approximation.

Row basis: Find orthogonal $(Q_\tau)_\tau$ such that

$$Q_\tau Q_\tau^* G|_{\tau\times\sigma} \approx G|_{\tau\times\sigma} \qquad \text{for all admissible } (\tau,\sigma).$$

Idea: Due to $G|_{\tau\times\sigma} = V_\tau S_{\tau\sigma} W_\sigma^*$, it suffices to ensure

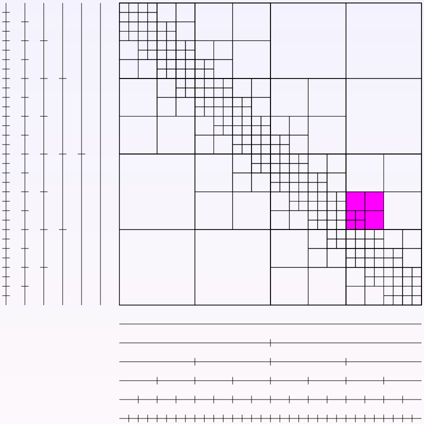
$$Q_\tau Q_\tau^* V_\tau Z_\tau^* \approx V_\tau Z_\tau^*,$$

where Z_τ is a small weight matrix. $\rightarrow$ Solve by SVD or RRQR.

Result: Cluster bases can be constructed in $\mathcal{O}(nk^2)$ operations, using $\mathcal{O}(nk)$ units of auxiliary storage, e.g., for $(Z_\tau)_\tau$.
Conversion of $\mathcal{H}^2$-matrix to new bases takes $\mathcal{O}(nk^2)$ operations.

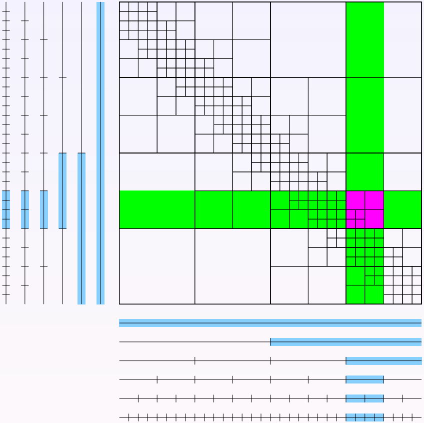
Local low-rank update

Goal: Approximate local update $Z|_{\hat{\tau}\times\hat{\sigma}} + AB^*$.



Local low-rank update

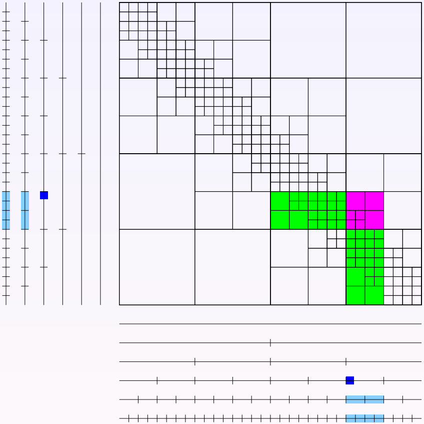
Goal: Approximate local update $Z|_{\hat{\tau}\times\hat{\sigma}} + AB^*$.



Cluster bases: Changes affect entire block rows and columns.

Local low-rank update

Goal: Approximate local update $Z|_{\hat{\tau}\times\hat{\sigma}} + AB^*$.

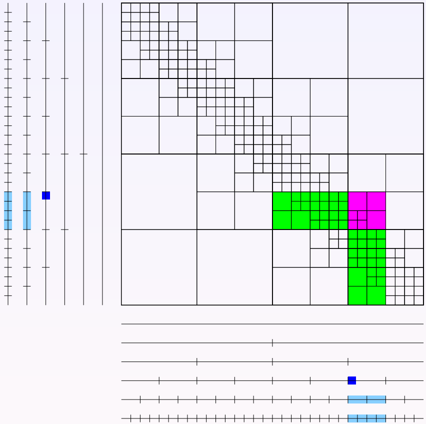


Cluster bases: Changes affect entire block rows and columns.

Solution: Use transfer matrices to limit the effect.

Local low-rank update

Goal: Approximate local update $Z|_{\hat\tau\times\hat\sigma} + AB^*$.



Cluster bases: Changes affect entire block rows and columns.

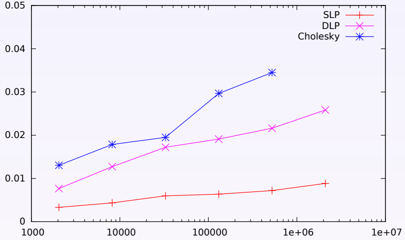
Solution: Use transfer matrices to limit the effect.

Result: Only $\mathcal{O}(k^2(\#\hat\tau + \#\hat\sigma))$ operations required.

Multiplication and factorization in $\mathcal{O}(nk^2 \log n)$ operations.

Experiment: Unit sphere

Example: Dirichlet problem on the unit sphere, direct formulation.

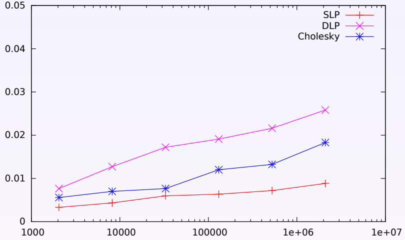


Time for setup:

Less than 6+15 hours for more than 2 million triangles.

Experiment: Unit sphere

Example: Dirichlet problem on the unit sphere, direct formulation.

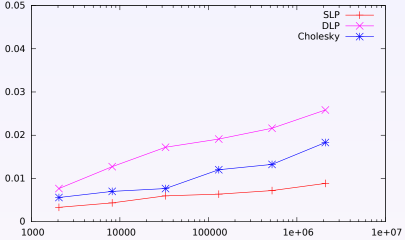


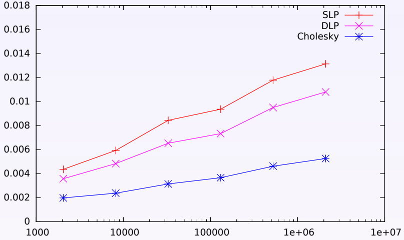
Time for setup:
Less than 6+15 hours for more than 2 million triangles.

$\mathcal{H}^2$-Cholesky preconditioner:
Less than 11 hours.

Experiment: Unit sphere

Example: Dirichlet problem on the unit sphere, direct formulation.





Time for setup:
Less than 6+15 hours for more than 2 million triangles.

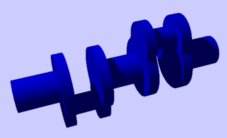
$\mathcal{H}^2$-Cholesky preconditioner:
Less than 11 hours.

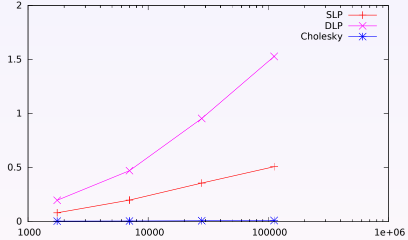
Storage for matrices:
Less than 28+23 GB for more than 2 million triangles.

$\mathcal{H}^2$-Cholesky preconditioner:
Less than 11 GB.

Experiment: Crank shaft

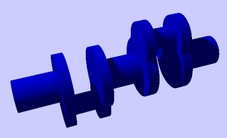
Example: Dirichlet problem on the NetGen "crank shaft" geometry, direct formulation.

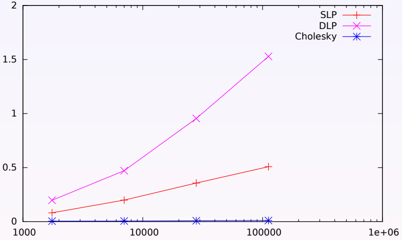


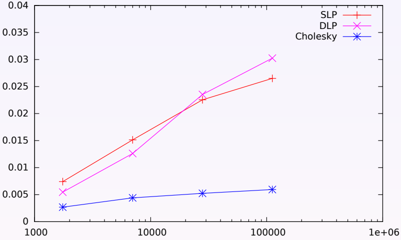


Experiment: Crank shaft

Example: Dirichlet problem on the NetGen "crank shaft" geometry, direct formulation.

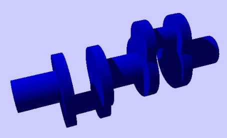


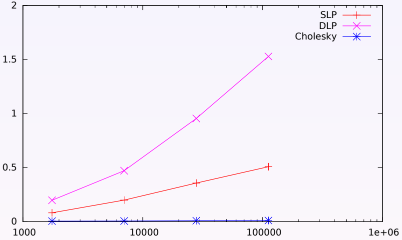


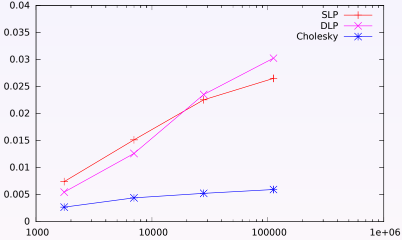


Experiment: Crank shaft

Example: Dirichlet problem on the NetGen "crank shaft" geometry, direct formulation.







Due to very high quadrature orders, the cost of constructing the $\mathcal{H}^2$-Cholesky preconditioner is almost negligible.

Summary

$\mathcal{H}^2$-matrices:

- Admissible blocks in factorized form $G|_{\hat{\tau}\times\hat{\sigma}} = V_\tau S_{\tau\sigma} W_\sigma^*$.
- Cluster bases in nested form $V_\tau = \begin{pmatrix} V_{\tau_1} E_{\tau_1} \\ V_{\tau_2} E_{\tau_2} \end{pmatrix}$.
- Storage requirements $\mathcal{O}(nk)$.

Arithmetic operations:

- Factorization expressed by multiplications
- Multiplications expressed by low-rank updates

Summary

$\mathcal{H}^2$-matrices:

- Admissible blocks in factorized form $G|_{\hat{\tau}\times\hat{\sigma}} = V_\tau S_{\tau\sigma} W_\sigma^*$.
- Cluster bases in nested form $V_\tau = \begin{pmatrix} V_{\tau_1} E_{\tau_1} \\ V_{\tau_2} E_{\tau_2} \end{pmatrix}$.
- Storage requirements $\mathcal{O}(nk)$.

Arithmetic operations:

- Factorization expressed by multiplications
- Multiplications expressed by low-rank updates
- Local low-rank update to $G|_{\hat{\tau}\times\hat{\sigma}}$ in $\mathcal{O}(k^2(\#\hat{\tau} + \#\hat{\sigma}))$.

Summary

$\mathcal{H}^2$-matrices:

- Admissible blocks in factorized form $G|_{\hat{\tau}\times\hat{\sigma}} = V_\tau S_{\tau\sigma} W_\sigma^*$.
- Cluster bases in nested form $V_\tau = \begin{pmatrix} V_{\tau_1} E_{\tau_1} \\ V_{\tau_2} E_{\tau_2} \end{pmatrix}$.
- Storage requirements $\mathcal{O}(nk)$.

Arithmetic operations:

- Factorization expressed by multiplications
- Multiplications expressed by low-rank updates, $\mathcal{O}(nk^2 \log n)$.
- Local low-rank update to $G|_{\hat{\tau}\times\hat{\sigma}}$ in $\mathcal{O}(k^2(\#\hat{\tau} + \#\hat{\sigma}))$.

Summary

$\mathcal{H}^2$-matrices:

- Admissible blocks in factorized form $G|_{\hat\tau\times\hat\sigma} = V_\tau S_{\tau\sigma} W_\sigma^*$.
- Cluster bases in nested form $V_\tau = \begin{pmatrix} V_{\tau_1} E_{\tau_1} \\ V_{\tau_2} E_{\tau_2} \end{pmatrix}$.
- Storage requirements $\mathcal{O}(nk)$.

Arithmetic operations:

- Factorization expressed by multiplications, $\mathcal{O}(nk^2 \log n)$.
- Multiplications expressed by low-rank updates, $\mathcal{O}(nk^2 \log n)$.
- Local low-rank update to $G|_{\hat\tau\times\hat\sigma}$ in $\mathcal{O}(k^2(\#\hat\tau + \#\hat\sigma))$.

Resources

References

- W. Hackbusch, B. N. Khoromskij, S. A. Sauter:
  On $\mathcal{H}^2$-matrices (2000)
- S. B., W. Hackbusch:
  Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices (2002)
- S. B.:
  Efficient numerical methods for non-local operators (2010)
- S. B., K. Reimer:
  Efficient arithmetic operations for rank-structured matrices based on hierarchical low-rank updates (2015)
- S. B., S. Christophersen:
  Approximation of integral operators by Green quadrature and nested cross approximation, arXiv preprint (2015)

Software

- H2Lib, open source, available at GitHub