GCA-$\mathcal{H}^2$ matrix compression for electrostatic simulations

Steffen Börm, Sven Christophersen, and Jessica Gördes

Christian-Albrechts-Universität zu Kiel

12th International Conference on
Scientific Computing in Electrical Engineering,
Taormina, Sicily, 2018

Overview

Overview

Laplace's equation

Task: Given a domain Ω and a surface potential σ, we want to compute the potential φ satisfying

$$\begin{aligned} -\Delta\varphi(x) &= 0 && \text{for all } x \in \Omega, \\ \varphi(x) &= \sigma(x) && \text{for all } x \in \partial\Omega. \end{aligned}$$

Standard approach: Finite element discretization, e.g., with piecewise linear functions.

- ✔ Implementation relatively simple.
- ✔ Fast solvers available, e.g., multigrid methods.
- ✗ Requires volume mesh.
- ✗ Large number of degrees of freedom.

Boundary integral method

Green's equation: For all $x \in \Omega$, we have

$$\varphi(x) = \int_{\partial\Omega} g(x,y) \frac{\partial\varphi}{\partial n}(y)\, dy - \int_{\partial\Omega} \frac{\partial g}{\partial n_y}(x,y)\varphi(y)\, dy.$$

Problem: We need the Neumann values $\partial\varphi/\partial n$ on the boundary.

Boundary integral method

Green's equation: For all $x \in \Omega$, we have

$$\varphi(x) = \int_{\partial\Omega} g(x,y) \frac{\partial\varphi}{\partial n}(y)\, dy - \int_{\partial\Omega} \frac{\partial g}{\partial n_y}(x,y)\varphi(y)\, dy.$$

Problem: We need the Neumann values $\partial\varphi/\partial n$ on the boundary.

Green's boundary integral: For all $x \in \partial\Omega$, we have

$$\frac{1}{2}\varphi(x) = \int_{\partial\Omega} g(x,y) \frac{\partial\varphi}{\partial n}(y)\, dy - \int_{\partial\Omega} \frac{\partial g}{\partial n_y}(x,y)\varphi(y)\, dy$$

provided that the boundary $\partial\Omega$ is smooth.

Boundary integral method

Green's equation: For all $x \in \Omega$, we have

$$\varphi(x) = \int_{\partial\Omega} g(x,y) \frac{\partial\varphi}{\partial n}(y)\, dy - \int_{\partial\Omega} \frac{\partial g}{\partial n_y}(x,y)\varphi(y)\, dy.$$

Problem: We need the Neumann values $\partial\varphi/\partial n$ on the boundary.

Green's boundary integral: For all $x \in \partial\Omega$, we have

$$\frac{1}{2}\varphi(x) + \int_{\partial\Omega} \frac{\partial g}{\partial n_y}(x,y)\varphi(y)\, dy = \int_{\partial\Omega} g(x,y) \frac{\partial\varphi}{\partial n}(y)\, dy$$

provided that the boundary $\partial\Omega$ is smooth.

Boundary integral method

Green's equation: For all $x \in \Omega$, we have

$$\varphi(x) = \int_{\partial\Omega} g(x,y) \frac{\partial\varphi}{\partial n}(y)\, dy - \int_{\partial\Omega} \frac{\partial g}{\partial n_y}(x,y)\sigma(y)\, dy.$$

Problem: We need the Neumann values $\partial\varphi/\partial n$ on the boundary.

Green's boundary integral: For all $x \in \partial\Omega$, we have

$$\frac{1}{2}\sigma(x) + \int_{\partial\Omega} \frac{\partial g}{\partial n_y}(x,y)\sigma(y)\, dy = \int_{\partial\Omega} g(x,y) \frac{\partial\varphi}{\partial n}(y)\, dy$$

provided that the boundary $\partial\Omega$ is smooth.

Boundary integral method

Green's equation: For all $x \in \Omega$, we have

$$\varphi(x) = \int_{\partial\Omega} g(x,y) \frac{\partial\varphi}{\partial n}(y)\, dy - \int_{\partial\Omega} \frac{\partial g}{\partial n_y}(x,y)\sigma(y)\, dy.$$

Problem: We need the Neumann values $\partial\varphi/\partial n$ on the boundary.

Green's boundary integral: For all $x \in \partial\Omega$, we have

$$\frac{1}{2}\sigma(x) + \int_{\partial\Omega} \frac{\partial g}{\partial n_y}(x,y)\sigma(y)\, dy = \int_{\partial\Omega} g(x,y) \frac{\partial\varphi}{\partial n}(y)\, dy$$

provided that the boundary $\partial\Omega$ is smooth.

Approach: Solve boundary integral equation to get Neumann values.

Boundary element method

Goal: Solve boundary integral equation numerically.

Collocation: Choose collocation points $(x_i)_{i=1}^n$, find approximation μ_h with

$$\int_{\partial\Omega} g(x_i, y)\mu_h(y)\,dy = \frac{1}{2}\sigma(x_i) + \int_{\partial\Omega} \frac{\partial g}{\partial n_y}(x_i, y)\sigma(y)\,dy.$$

Alternative: Galerkin discretization offers better stability, but the implementation is more challenging.

Result: Linear system of dimension n.

Boundary element method

Goal: Solve boundary integral equation numerically.

Collocation: Choose collocation points $(x_i)_{i=1}^n$, find approximation μ_h with

$$\int_{\partial\Omega} g(x_i, y)\mu_h(y)\,dy = \frac{1}{2}\sigma(x_i) + \int_{\partial\Omega} \frac{\partial g}{\partial n_y}(x_i, y)\sigma(y)\,dy.$$

Alternative: Galerkin discretization offers better stability, but the implementation is more challenging.

Result: Linear system of dimension n.

Comparison to finite elements:

- ✗ Matrix assembly requires singular integrals.
- ✓ Requires only surface mesh.
- ✓ Only $n \sim h^{-2}$ instead of $n \sim h^{-3}$ degrees of freedom.
- ✓ Works for interior and exterior domain problems.
- ✗ Matrix dense instead of sparse.

Boundary element method

Goal: Solve boundary integral equation numerically.

Collocation: Choose collocation points $(x_i)_{i=1}^n$, find approximation μ_h with

$$\int_{\partial\Omega} g(x_i, y)\mu_h(y)\,dy = \frac{1}{2}\sigma(x_i) + \int_{\partial\Omega} \frac{\partial g}{\partial n_y}(x_i, y)\sigma(y)\,dy.$$

Alternative: Galerkin discretization offers better stability, but the implementation is more challenging.

Result: Linear system of dimension n.

Comparison to finite elements:

- ✗ Matrix assembly requires singular integrals.
- ✓ Requires only surface mesh.
- ✓ Only $n \sim h^{-2}$ instead of $n \sim h^{-3}$ degrees of freedom.
- ✓ Works for interior and exterior domain problems.
- ✓ Matrix dense instead of sparse. $\rightarrow$ Data-sparse approximation.

Boundary element method

Goal: Solve boundary integral equation numerically.

Collocation: Choose collocation points $(x_i)_{i=1}^n$, find approximation μ_h with

$$\int_{\partial\Omega} g(x_i, y)\mu_h(y)\,dy = \frac{1}{2}\sigma(x_i) + \int_{\partial\Omega} \frac{\partial g}{\partial n_y}(x_i, y)\sigma(y)\,dy.$$

Alternative: Galerkin discretization offers better stability, but the implementation is more challenging.

Result: Linear system of dimension n.

Comparison to finite elements:

- ✔ Matrix assembly requires singular integrals. $\rightarrow$ Accelerate using GPUs
- ✔ Requires only surface mesh.
- ✔ Only $n \sim h^{-2}$ instead of $n \sim h^{-3}$ degrees of freedom.
- ✔ Works for interior and exterior domain problems.
- ✔ Matrix dense instead of sparse. $\rightarrow$ Data-sparse approximation.

Overview

Degenerate kernels

Degenerate kernels have the form

$$g(x,y) \approx \sum_{\nu=1}^{k} a_\nu(x) b_\nu(y)$$

and lead to

$$g_{ij} \approx \sum_{\nu=1}^{k} \underbrace{a_\nu(x_i)}_{=:a_{i\nu}} \underbrace{\int_{\partial\Omega} b_\nu(y)\varphi_j(y)\,dy}_{=:b_{j\nu}}$$

Low-rank approximation of G yields

- ✓ Storage $2nk$ instead of n^2.
- ✓ Matrix-vector multiplication in $4nk$ operations instead of $2n^2$.

Degenerate kernels

Degenerate kernels have the form

$$g(x,y) \approx \sum_{\nu=1}^{k} a_\nu(x) b_\nu(y)$$

and lead to

$$g_{ij} \approx \sum_{\nu=1}^{k} \underbrace{a_\nu(x_i)}_{=:a_{i\nu}} \underbrace{\int_{\partial\Omega} b_\nu(y)\varphi_j(y)\,dy}_{=:b_{j\nu}}, \qquad G \approx AB^T.$$

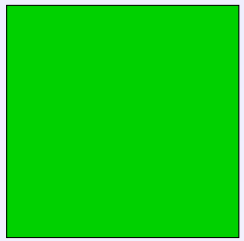
Low-rank approximation of G yields

- ✓ Storage $2nk$ instead of n^2.
- ✓ Matrix-vector multiplication in $4nk$ operations instead of $2n^2$.

Degenerate kernels

Degenerate kernels have the form

$$g(x,y) \approx \sum_{\nu=1}^{k} a_\nu(x) b_\nu(y)$$



and lead to

$$g_{ij} \approx \sum_{\nu=1}^{k} \underbrace{a_\nu(x_i)}_{=:a_{i\nu}} \underbrace{\int_{\partial\Omega} b_\nu(y)\varphi_j(y)\,dy}_{=:b_{j\nu}}, \qquad G \approx AB^T.$$

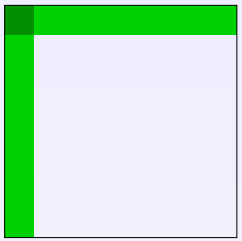
Low-rank approximation of G yields

- ✓ Storage $2nk$ instead of n^2.
- ✓ Matrix-vector multiplication in $4nk$ operations instead of $2n^2$.

Degenerate kernels

Degenerate kernels have the form

$$g(x,y) \approx \sum_{\nu=1}^{k} a_\nu(x) b_\nu(y)$$



and lead to

$$g_{ij} \approx \sum_{\nu=1}^{k} \underbrace{a_\nu(x_i)}_{=:a_{i\nu}} \underbrace{\int_{\partial\Omega} b_\nu(y)\varphi_j(y)\,dy}_{=:b_{j\nu}}, \qquad G \approx AB^T.$$

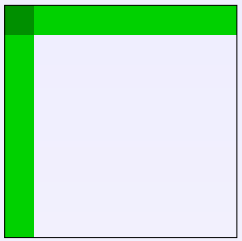
Low-rank approximation of G yields

- ✓ Storage $2nk$ instead of n^2.
- ✓ Matrix-vector multiplication in $4nk$ operations instead of $2n^2$.

Degenerate kernels

Degenerate kernels have the form

$$g(x,y) \approx \sum_{\nu=1}^{k} a_\nu(x) b_\nu(y)$$



and lead to

$$g_{ij} \approx \sum_{\nu=1}^{k} \underbrace{a_\nu(x_i)}_{=:a_{i\nu}} \underbrace{\int_{\partial\Omega} b_\nu(y)\varphi_j(y)\,dy}_{=:b_{j\nu}}, \qquad G \approx AB^T.$$

Low-rank approximation of G yields

- ✓ Storage $2nk$ instead of n^2.
- ✓ Matrix-vector multiplication in $4nk$ operations instead of $2n^2$.
- ✗ Sufficiently accurate only if g is "smooth enough".

Local approximation

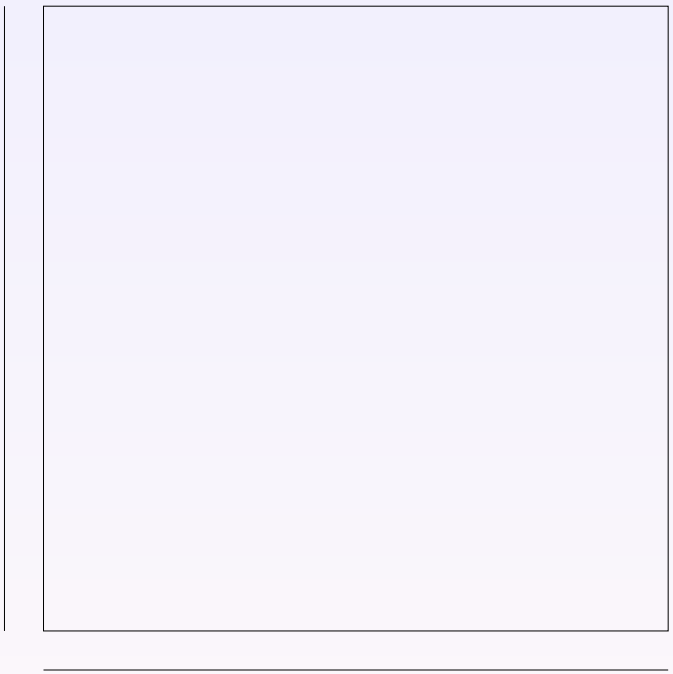
Problem: Degenerate kernels only exist if the kernel function is "smooth".

Approach: Apply approximation only in subdomains that are far from the singularity on the diagonal.

Local approximation

Problem: Degenerate kernels only exist if the kernel function is "smooth".

Approach: Apply approximation only in subdomains that are far from the singularity on the diagonal.



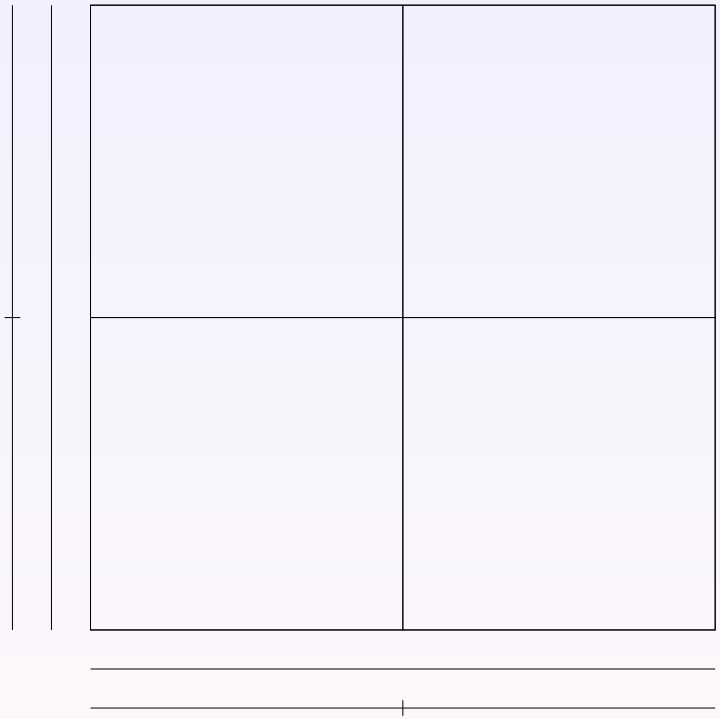
Algorithm:

- Split surface into subsets ("clusters")

Local approximation

Problem: Degenerate kernels only exist if the kernel function is "smooth".

Approach: Apply approximation only in subdomains that are far from the singularity on the diagonal.



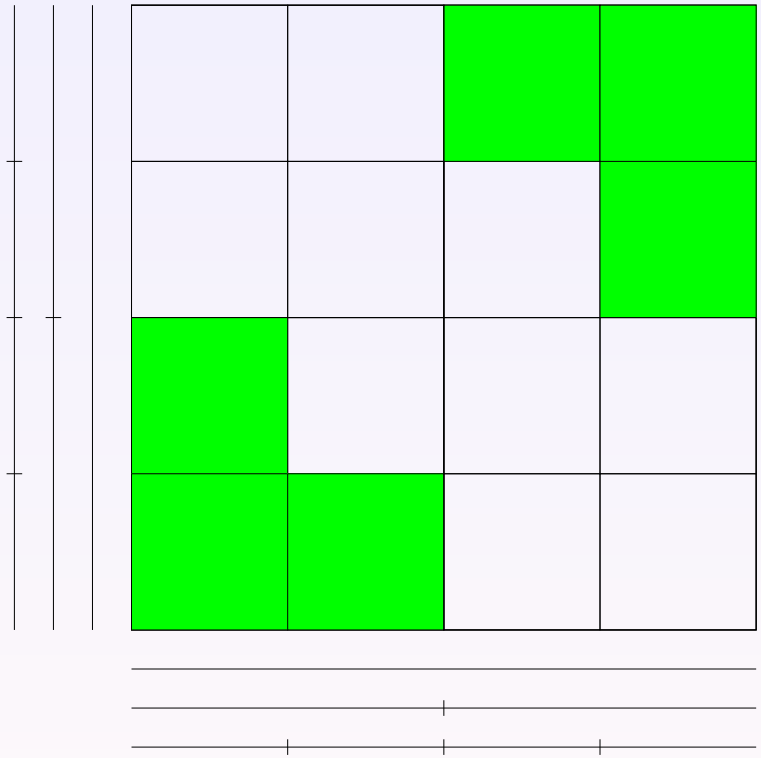
Algorithm:

- Split surface into subsets ("clusters")

Local approximation

Problem: Degenerate kernels only exist if the kernel function is "smooth".

Approach: Apply approximation only in subdomains that are far from the singularity on the diagonal.



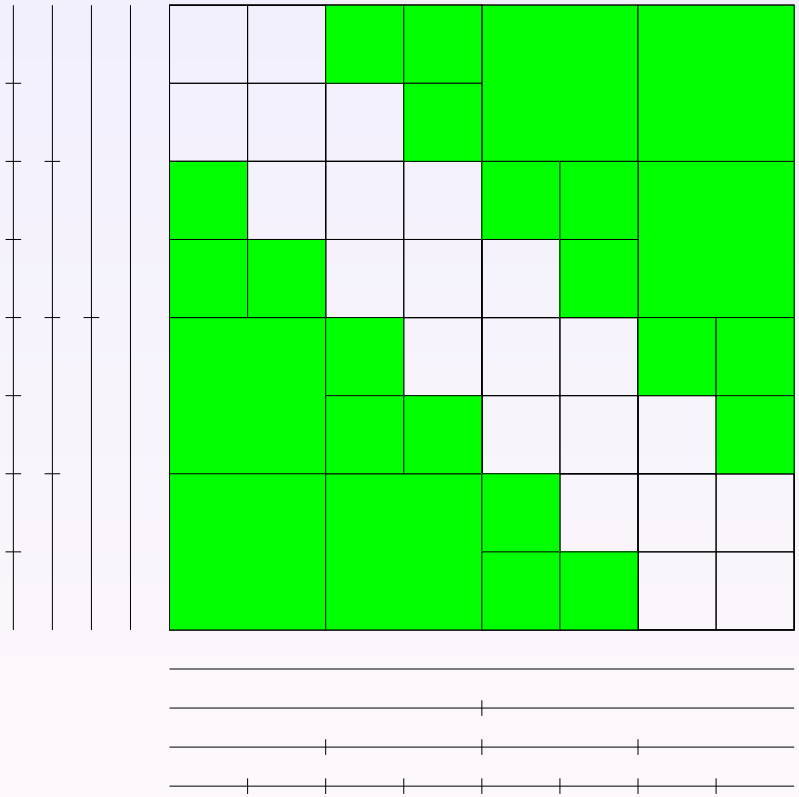
Algorithm:

- Split surface into subsets ("clusters")

Local approximation

Problem: Degenerate kernels only exist if the kernel function is "smooth".

Approach: Apply approximation only in subdomains that are far from the singularity on the diagonal.



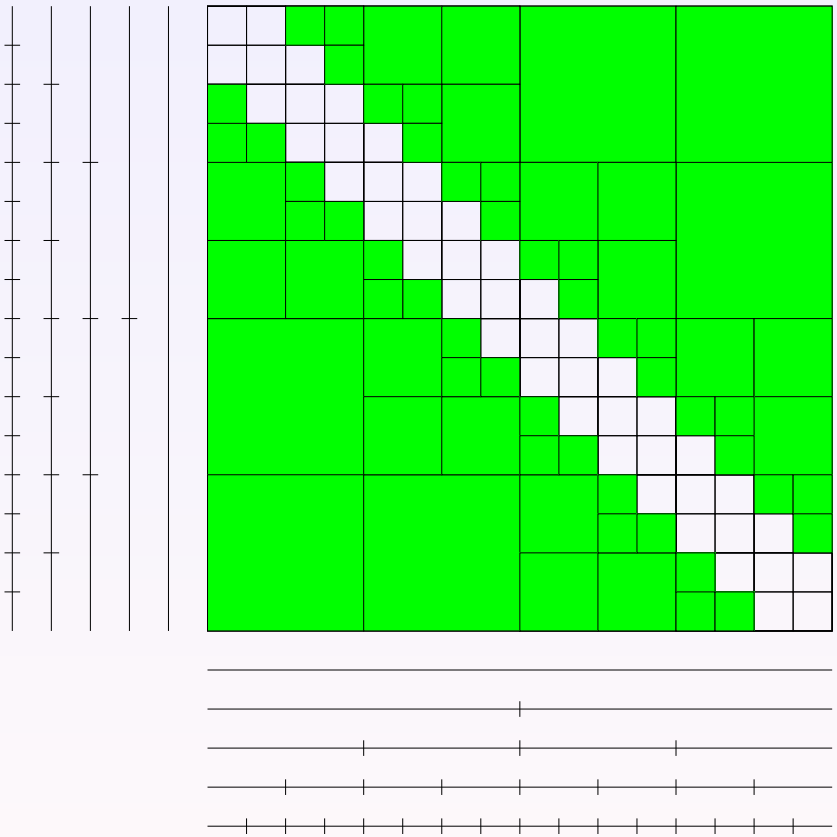
Algorithm:

- Split surface into subsets ("clusters")

Local approximation

Problem: Degenerate kernels only exist if the kernel function is "smooth".

Approach: Apply approximation only in subdomains that are far from the singularity on the diagonal.



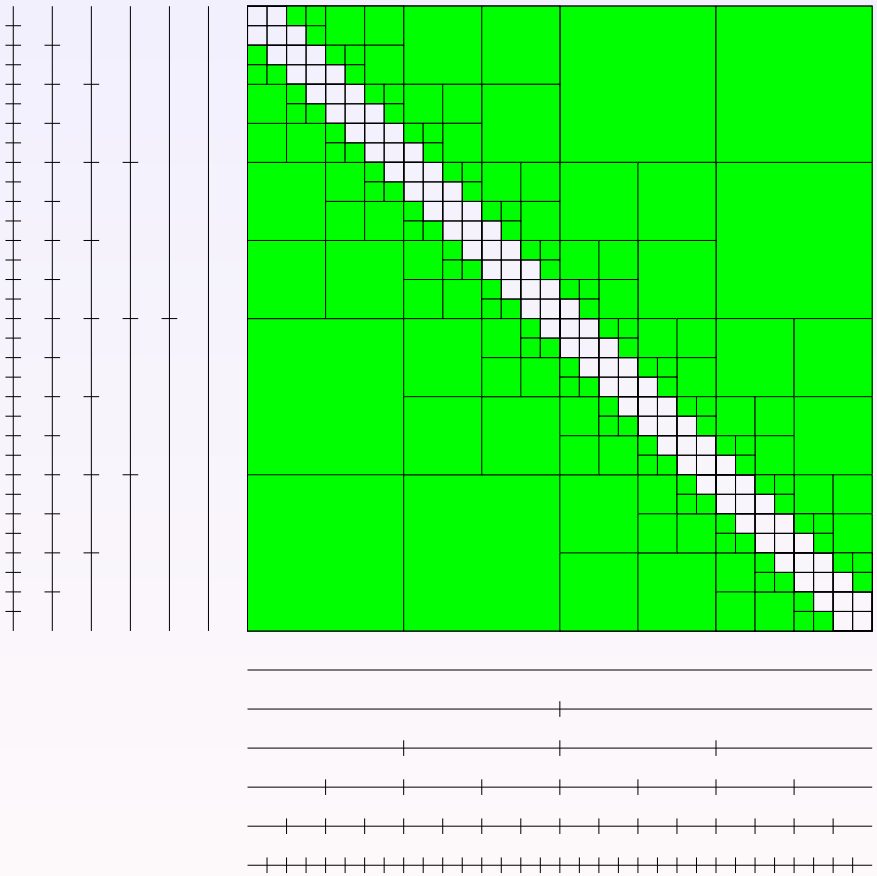
Algorithm:

- Split surface into subsets ("clusters")

Local approximation

Problem: Degenerate kernels only exist if the kernel function is "smooth".

Approach: Apply approximation only in subdomains that are far from the singularity on the diagonal.



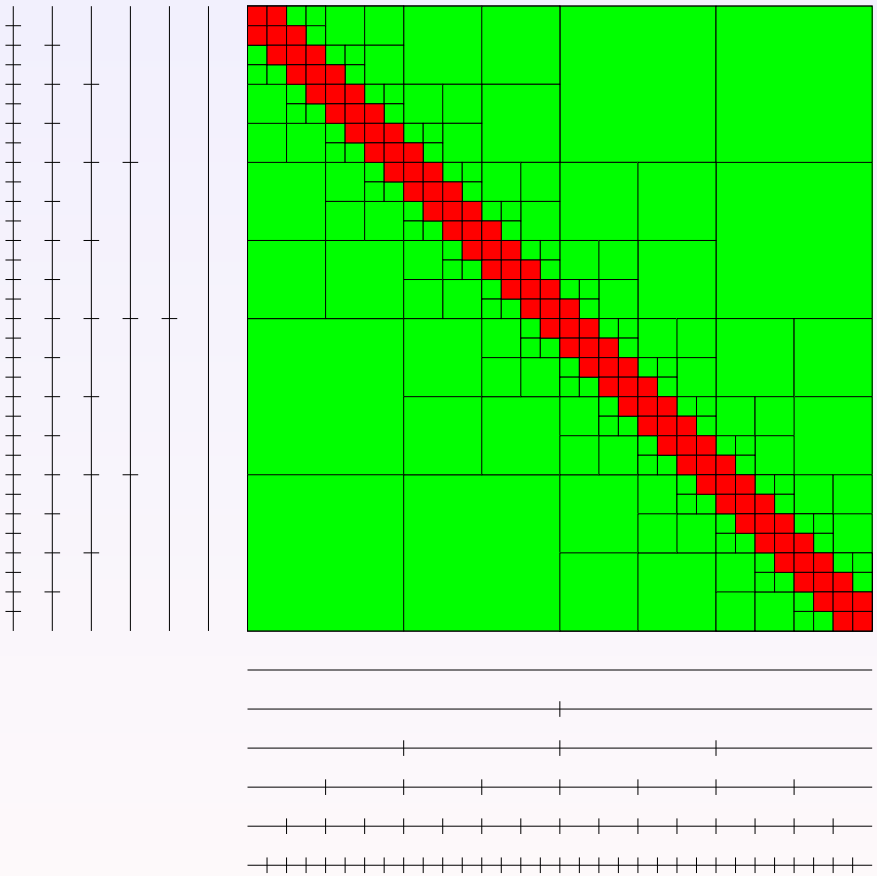
Algorithm:

- Split surface into subsets ("clusters")

Local approximation

Problem: Degenerate kernels only exist if the kernel function is "smooth".

Approach: Apply approximation only in subdomains that are far from the singularity on the diagonal.



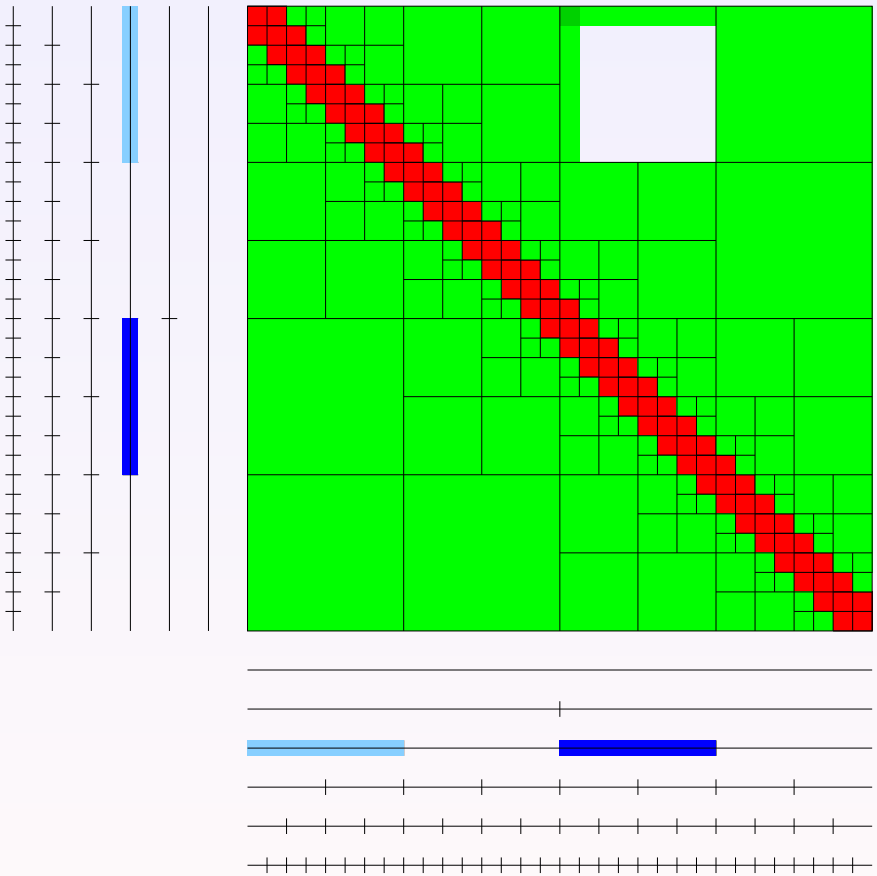
Algorithm:

- Split surface into subsets ("clusters")
- until remainder is small enough.

Local approximation

Problem: Degenerate kernels only exist if the kernel function is "smooth".

Approach: Apply approximation only in subdomains that are far from the singularity on the diagonal.



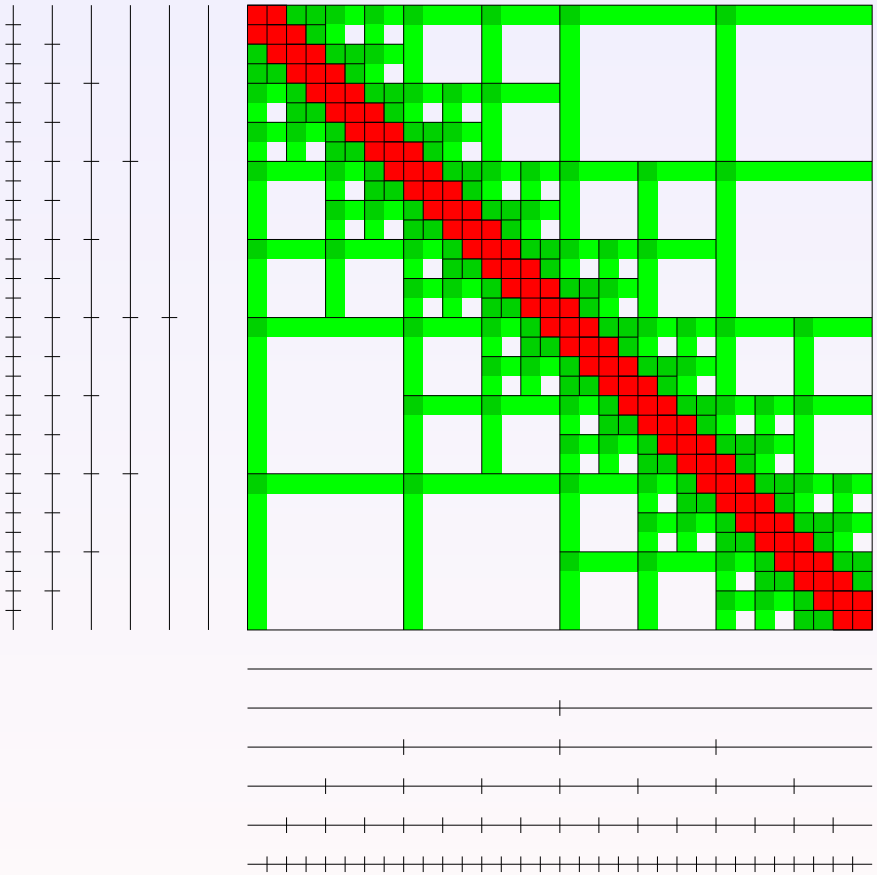
Algorithm:

- Split surface into subsets ("clusters")
- until remainder is small enough.
- Apply local low-rank approximations

Local approximation

Problem: Degenerate kernels only exist if the kernel function is "smooth".

Approach: Apply approximation only in subdomains that are far from the singularity on the diagonal.

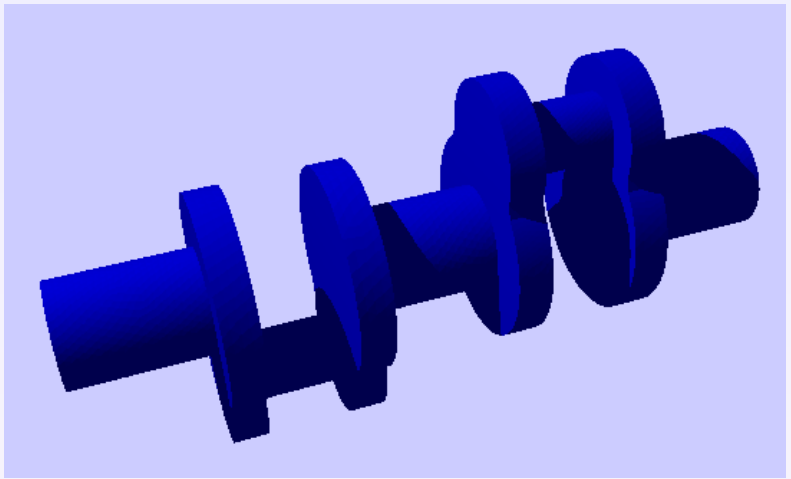


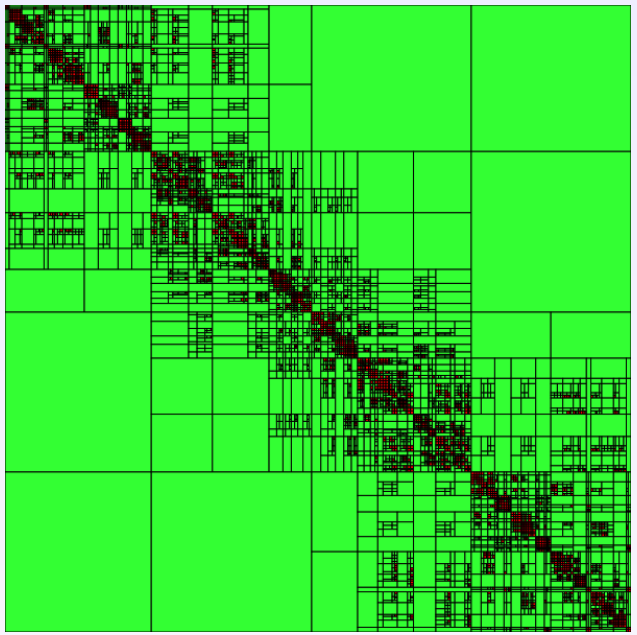
Algorithm:

- Split surface into subsets ("clusters")
- until remainder is small enough.
- Apply local low-rank approximations
- to obtain global matrix approximation.

Hierarchical matrix

Result: Blockwise low-rank approximation.





Properties of hierarchical matrices:

- ✔ Storage $\mathcal{O}(nk \log n)$.
- ✔ Matrix-vector multiplication in $\mathcal{O}(nk \log n)$ operations.
- ✔ Approximate inversion, LR factorization in $\mathcal{O}(nk^2 \log^2 n)$ operations.

Low-rank approximation

Question: How to construct local low-rank approximations?

Analytic methods:

- Taylor expansion (e.g., Hackbusch/Nowak, Sauter).
- Multipole expansion (e.g., Rokhlin, Greengard/Rokhlin).
- Interpolation (e.g., Lage, Tausch, B.).

Algebraic methods:

- Rank-revealing LR or QR factorization (e.g., Cai/Chow/Saad, Xia/Chandrasekaran/Gu).
- Adaptive cross approximation (e.g., Bebendorf/Rjasanow).
- Randomized sampling (e.g., Martinsson).

Low-rank approximation

Question: How to construct local low-rank approximations?

Analytic methods:

- Taylor expansion (e.g., Hackbusch/Nowak, Sauter).
- Multipole expansion (e.g., Rokhlin, Greengard/Rokhlin).
- Interpolation (e.g., Lage, Tausch, B.).
- Green quadrature approximation (B./Christophersen/Gördes).

Algebraic methods:

- Rank-revealing LR or QR factorization (e.g., Cai/Chow/Saad, Xia/Chandrasekaran/Gu).
- Adaptive cross approximation (e.g., Bebendorf/Rjasanow).
- Randomized sampling (e.g., Martinsson).

Low-rank approximation

Question: How to construct local low-rank approximations?

Analytic methods:

- Taylor expansion (e.g., Hackbusch/Nowak, Sauter).
- Multipole expansion (e.g., Rokhlin, Greengard/Rokhlin).
- Interpolation (e.g., Lage, Tausch, B.).
- Green quadrature approximation (B./Christophersen/Gördes).

Algebraic methods:

- Rank-revealing LR or QR factorization (e.g., Cai/Chow/Saad, Xia/Chandrasekaran/Gu).
- Adaptive cross approximation (e.g., Bebendorf/Rjasanow).
- Randomized sampling (e.g., Martinsson).

Hybrid methods: Combination of analytic and algebraic.

Green quadrature

Green's representation formula for harmonic functions in $\omega \subseteq \mathbb{R}^3$:

$$\varphi(x) = \int_{\partial\omega} g(x,z) \frac{\partial\varphi}{\partial n}(z)\, dz - \int_{\partial\omega} \frac{\partial g}{\partial n_z}(x,z)\varphi(z)\, dz$$

Green quadrature

Green's representation formula for harmonic functions in $\omega \subseteq \mathbb{R}^3$:

$$\varphi(x) = \int_{\partial\omega} g(x,z) \frac{\partial\varphi}{\partial n}(z)\, dz - \int_{\partial\omega} \frac{\partial g}{\partial n_z}(x,z)\varphi(z)\, dz$$

Idea: For $y \notin \omega$, $z \mapsto g(z,y)$ is harmonic, so the equation applies.

Green quadrature

Green's representation formula for harmonic functions in $\omega \subseteq \mathbb{R}^3$:

$$g(x,y) = \int_{\partial\omega} g(x,z)\frac{\partial g}{\partial n_z}(z,y)\,dz - \int_{\partial\omega} \frac{\partial g}{\partial n_z}(x,z)g(z,y)\,dz$$

Idea: For $y \notin \omega$, $z \mapsto g(z,y)$ is harmonic, so the equation applies.

Green quadrature

Green's representation formula for harmonic functions in $\omega \subseteq \mathbb{R}^3$:

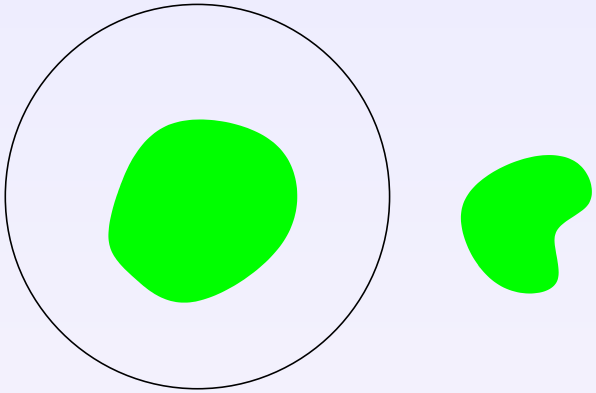
$$g(x,y) = \int_{\partial\omega} g(x,z)\frac{\partial g}{\partial n_z}(z,y)\,dz - \int_{\partial\omega} \frac{\partial g}{\partial n_z}(x,z)g(z,y)\,dz$$

Idea: For $y \notin \omega$, $z \mapsto g(z,y)$ is harmonic, so the equation applies.

Quadrature: If the boundary $\partial\omega$ is sufficiently far from x and y, we can approximate the integral by a sum.

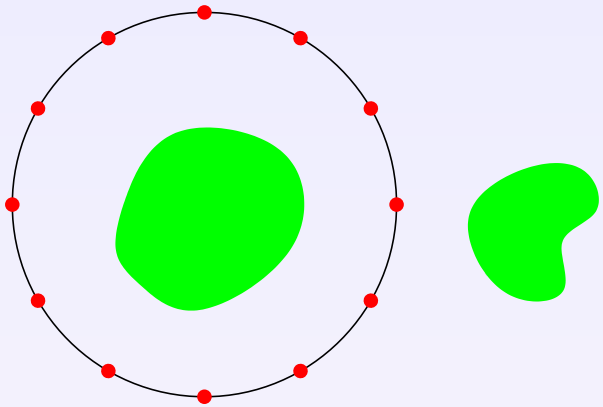
$$g(x,y) \approx \sum_{\nu=1}^{k} w_\nu g(x,z_\nu)\frac{\partial g}{\partial n_z}(z_\nu,y) - w_\nu \frac{\partial g}{\partial n_z}(x,z_\nu)g(z_\nu,y)$$

Properties



Admissibility: Ensure that $\tau, \sigma \in \mathbb{R}^3$ are well separated.

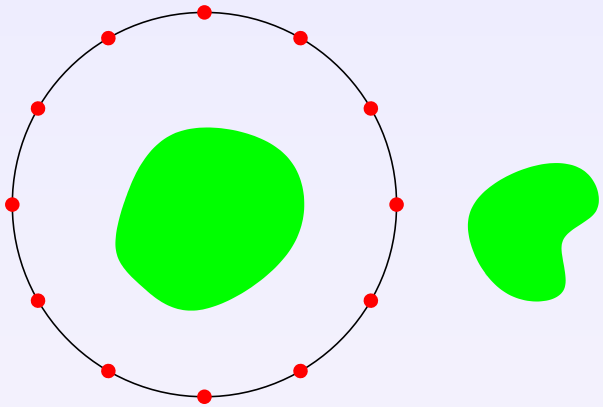
Properties



Admissibility: Ensure that $\tau, \sigma \in \mathbb{R}^3$ are well separated.

Quadrature: Equidistant points for circles or Gauss points for boxes.

Properties



Admissibility: Ensure that $\tau, \sigma \in \mathbb{R}^3$ are well separated.

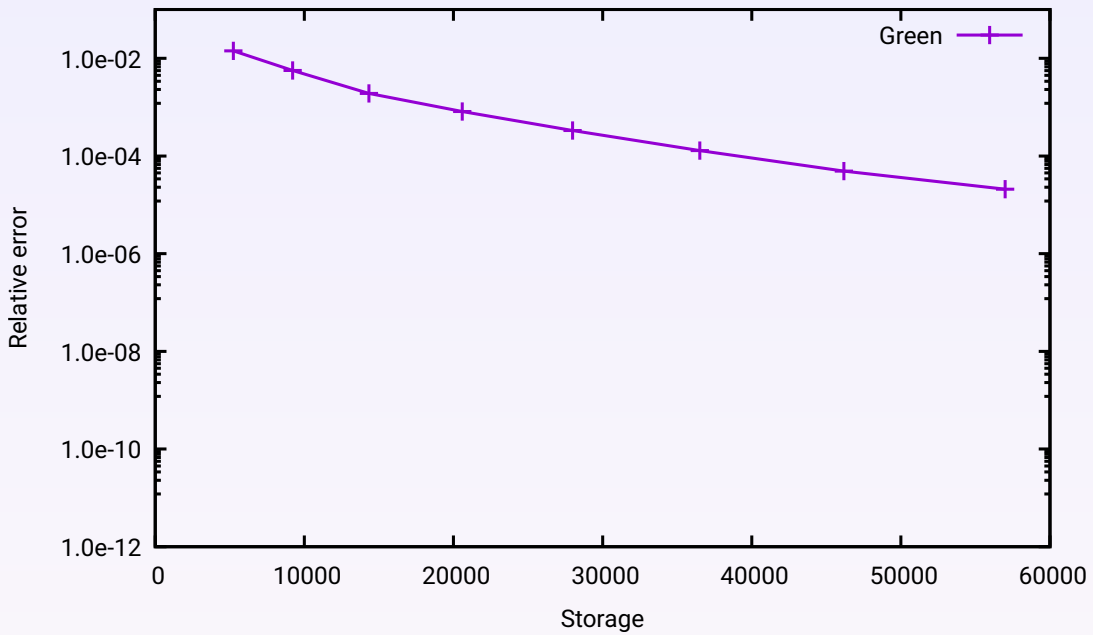
Quadrature: Equidistant points for circles or Gauss points for boxes.

Comparison to other techniques:

- Unlike Taylor expansion or interpolation, only $k \sim m^2$ terms required for m-th order approximation.
- Unlike FMM, no special expansion required.
- Unlike "multipole without multipoles" (Biros/Ying/Zorin), no least-squares problems have to be solved.
- Rigorous mathematical proof.

Efficiency

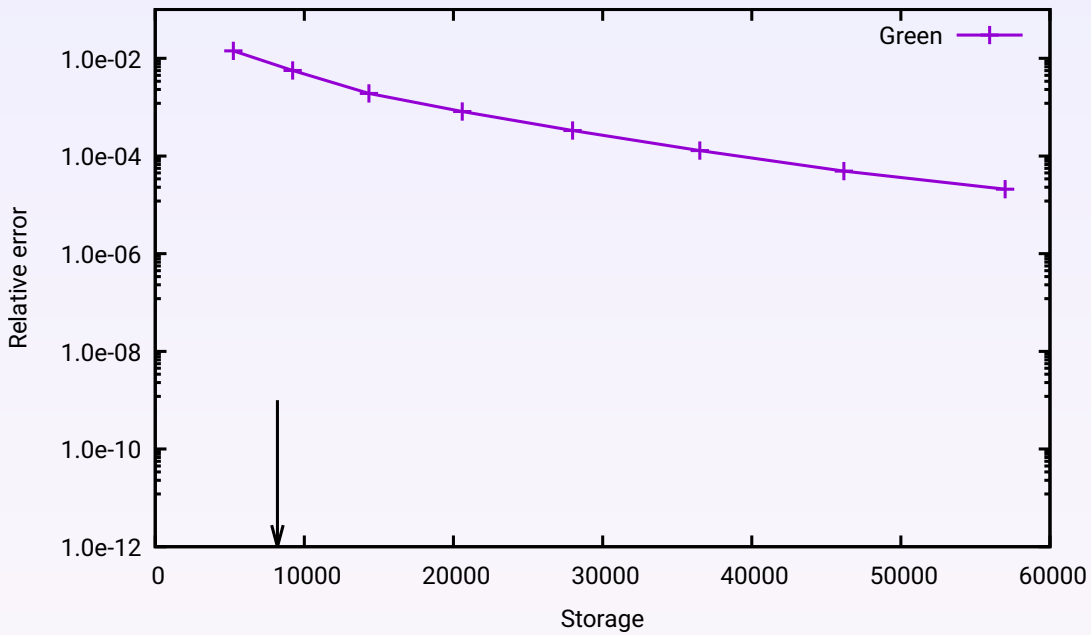
Experiment: Approximate boundary element matrix with different quadrature orders.



Result: Poor accuracy

Efficiency

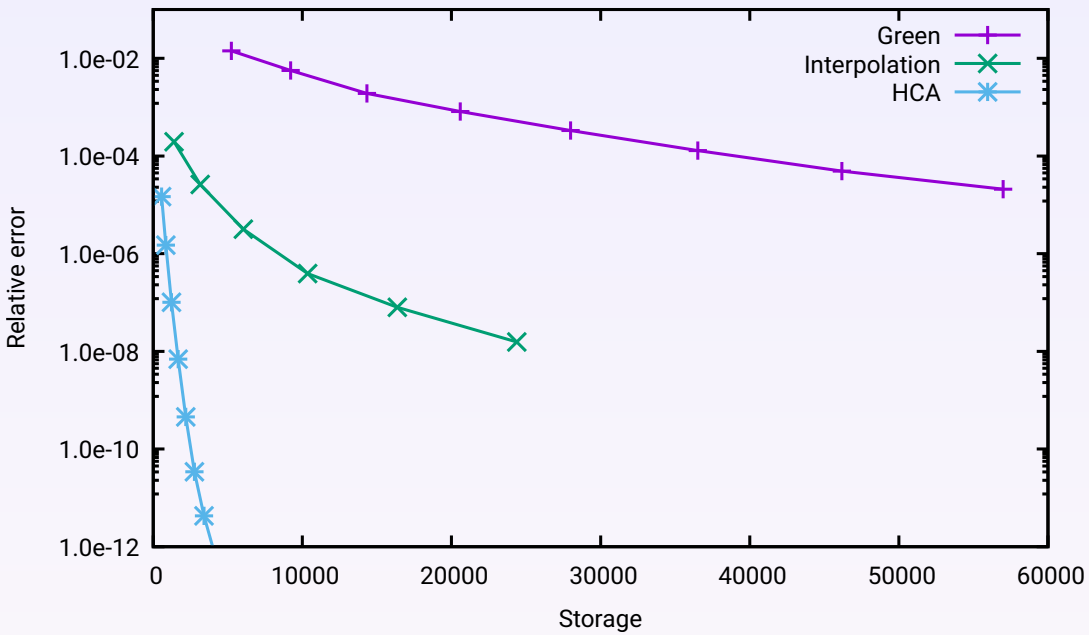
Experiment: Approximate boundary element matrix with different quadrature orders.



Result: Poor accuracy, almost no compression

Efficiency

Experiment: Approximate boundary element matrix with different quadrature orders.



Result: Poor accuracy, almost no compression, competing methods far superior.

Overview

Dirichlet vs Neumann

$$g(x,y) \approx \sum_{\nu=1}^{k} w_\nu g(x,z_\nu) \underbrace{\frac{\partial g}{\partial n_z}(z_\nu,y)}_{\text{Neumann values}} - w_\nu \frac{\partial g}{\partial n_z}(x,z_\nu) \underbrace{g(z_\nu,y)}_{\text{Dirichlet values}}$$

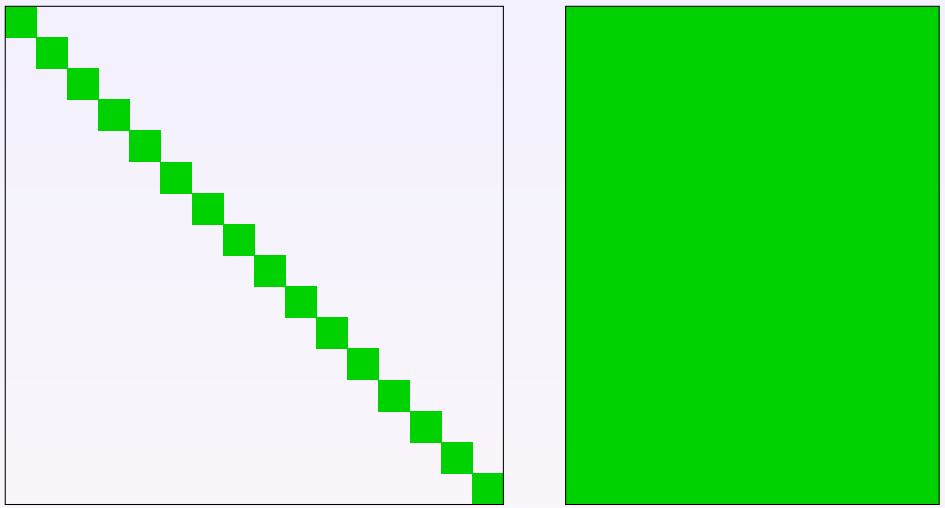
Observation: Quadrature approximation includes both Dirichlet and Neumann values of $z \mapsto g(z,y)$, although both are not independent.
$\rightarrow$ Number of terms at least twice as high as necessary.

Goal: Eliminate unnecessary terms from the approximation.

Cross approximation

Goal: Find low-rank approximation of a matrix C.

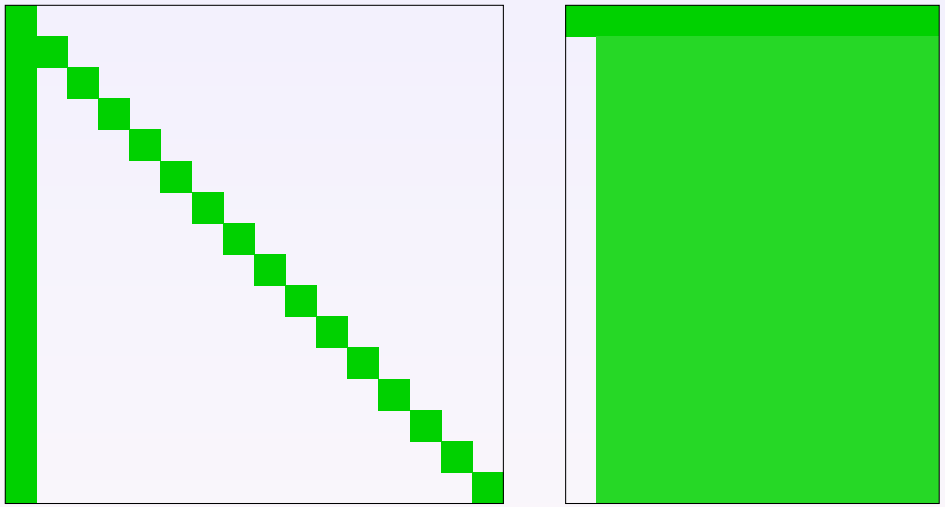
Idea: Perform LR factorization (aka Gaussian elimination)



Cross approximation

Goal: Find low-rank approximation of a matrix C.

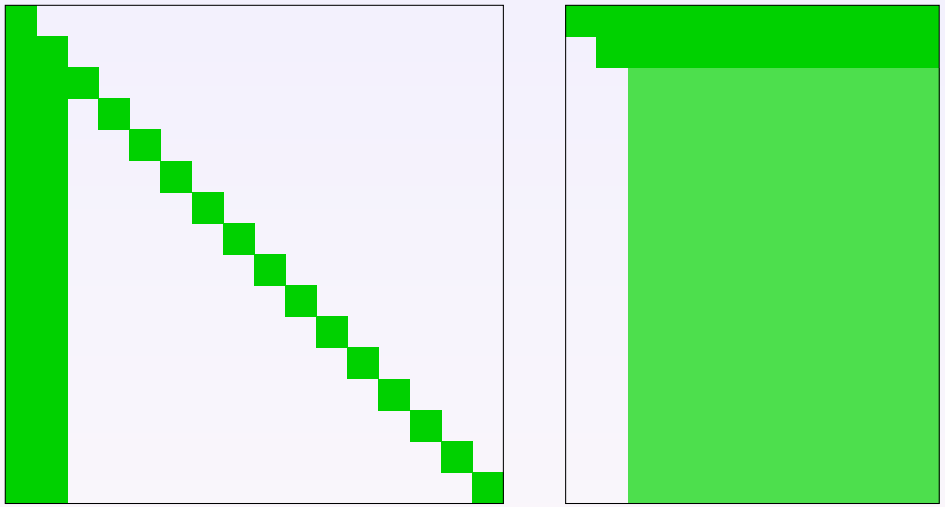
Idea: Perform LR factorization (aka Gaussian elimination)



Cross approximation

Goal: Find low-rank approximation of a matrix C.

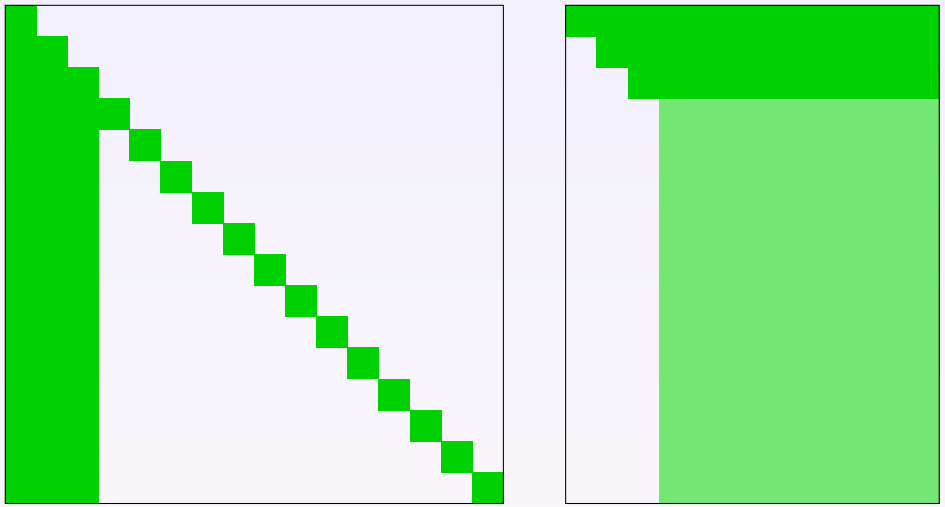
Idea: Perform LR factorization (aka Gaussian elimination)



Cross approximation

Goal: Find low-rank approximation of a matrix C.

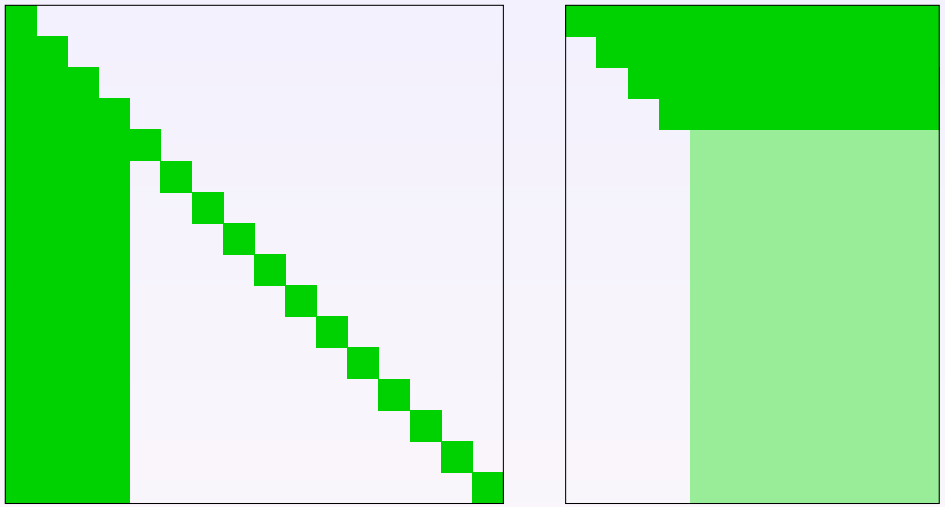
Idea: Perform LR factorization (aka Gaussian elimination)



Cross approximation

Goal: Find low-rank approximation of a matrix C.

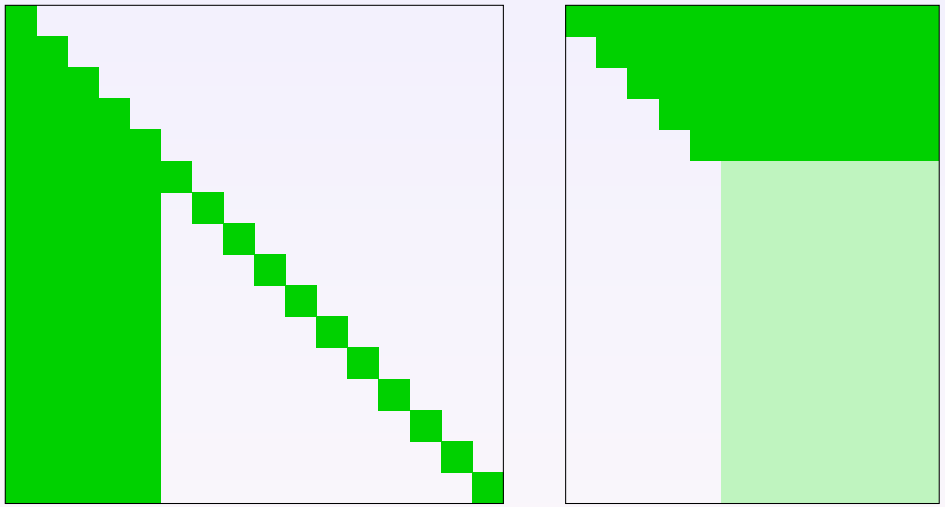
Idea: Perform LR factorization (aka Gaussian elimination)



Cross approximation

Goal: Find low-rank approximation of a matrix C.

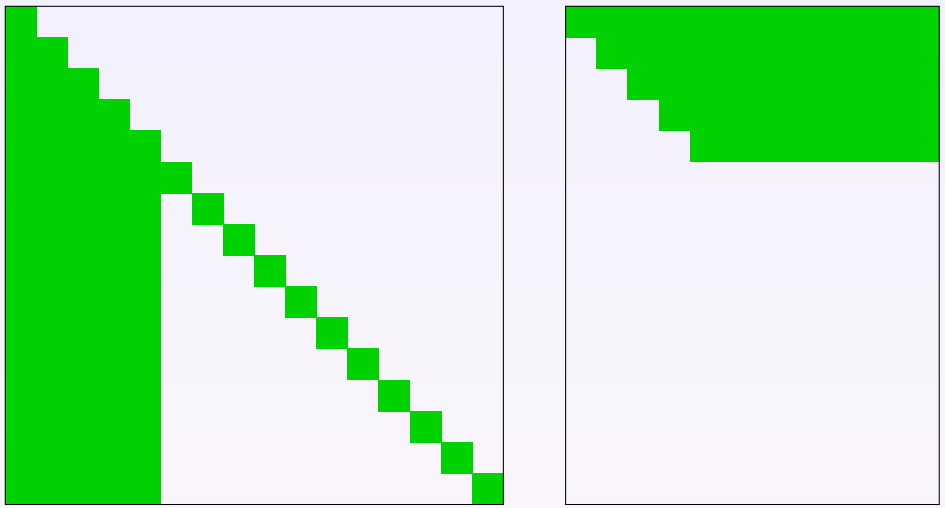
Idea: Perform LR factorization (aka Gaussian elimination)



Cross approximation

Goal: Find low-rank approximation of a matrix C.

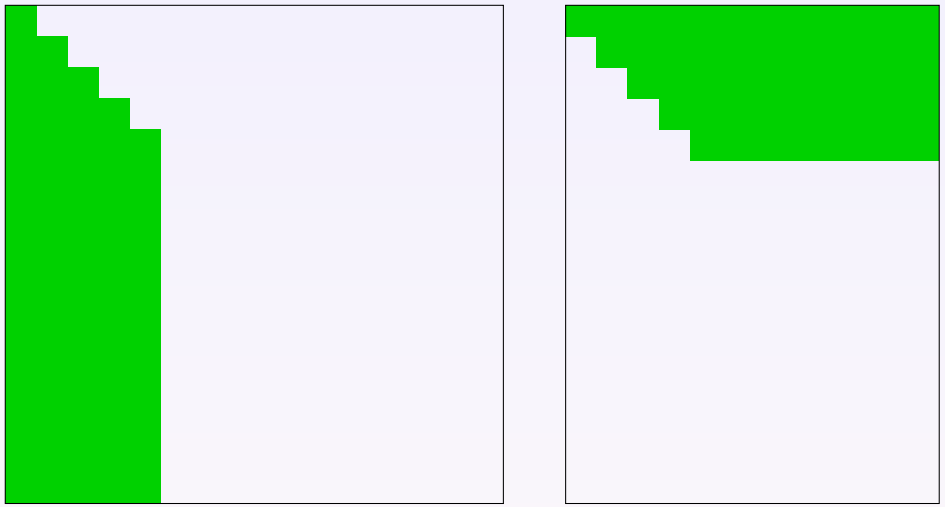
Idea: Perform LR factorization (aka Gaussian elimination) and remove remainder as soon as it is negligible.



Cross approximation

Goal: Find low-rank approximation of a matrix C.

Idea: Perform LR factorization (aka Gaussian elimination) and remove remainder as soon as it is negligible.



Result: Low-rank approximation of (theoretically) guaranteed accuracy.

Algebraic interpolation

Cross approximation yields

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} \approx \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} \begin{pmatrix} R_{11} & R_{12} \end{pmatrix}$$

Algebraic interpolation

Cross approximation yields

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} \approx \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} \begin{pmatrix} R_{11} & R_{12} \end{pmatrix} = \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} L_{11}^{-1} L_{11} \begin{pmatrix} R_{11} & R_{12} \end{pmatrix}$$

Algebraic interpolation

Cross approximation yields

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} \approx \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} \begin{pmatrix} R_{11} & R_{12} \end{pmatrix} = \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} L_{11}^{-1} L_{11} \begin{pmatrix} R_{11} & R_{12} \end{pmatrix}$$

$$= \begin{pmatrix} I \\ L_{21} L_{11}^{-1} \end{pmatrix} \begin{pmatrix} L_{11} R_{11} & L_{11} R_{12} \end{pmatrix}$$

Algebraic interpolation

Cross approximation yields

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} \approx \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} \begin{pmatrix} R_{11} & R_{12} \end{pmatrix} = \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} L_{11}^{-1} L_{11} \begin{pmatrix} R_{11} & R_{12} \end{pmatrix}$$

$$= \begin{pmatrix} I \\ L_{21} L_{11}^{-1} \end{pmatrix} \begin{pmatrix} L_{11} R_{11} & L_{11} R_{12} \end{pmatrix} = V \begin{pmatrix} C_{11} & C_{12} \end{pmatrix}.$$

Algebraic interpolation

Cross approximation yields

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} \approx \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} \begin{pmatrix} R_{11} & R_{12} \end{pmatrix} = \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} L_{11}^{-1} L_{11} \begin{pmatrix} R_{11} & R_{12} \end{pmatrix}$$
$$= \begin{pmatrix} I \\ L_{21} L_{11}^{-1} \end{pmatrix} \begin{pmatrix} L_{11} R_{11} & L_{11} R_{12} \end{pmatrix} = V \begin{pmatrix} C_{11} & C_{12} \end{pmatrix}.$$

Observation: We have found a matrix V that allows us to recover the entire matrix from its first k rows.

Algebraic interpolation

Cross approximation yields

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} \approx \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} \begin{pmatrix} R_{11} & R_{12} \end{pmatrix} = \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} L_{11}^{-1} L_{11} \begin{pmatrix} R_{11} & R_{12} \end{pmatrix}$$
$$= \begin{pmatrix} I \\ L_{21} L_{11}^{-1} \end{pmatrix} \begin{pmatrix} L_{11} R_{11} & L_{11} R_{12} \end{pmatrix} = V \begin{pmatrix} C_{11} & C_{12} \end{pmatrix}.$$

Observation: We have found a matrix V that allows us to recover the entire matrix from its first k rows.

Generalization: Given a matrix $C \in \mathbb{R}^{t \times s}$ that can be approximated with rank k, we find $\tau \subseteq t$ and $V \in \mathbb{R}^{t \times \sigma}$ with

$$C \approx VC|_{\tau \times s}, \qquad \#\tau = k.$$

Green cross approximation

The quadrature approximation

$$\underbrace{g(x_i, y_j)}_{=g_{ij}} \approx \sum_{\nu=1}^{k} \underbrace{w_\nu g(x_i, z_\nu)}_{=a_{i\nu}} \underbrace{\frac{\partial g}{\partial n_z}(z_\nu, y_j)}_{=b_{j\nu}} \underbrace{-w_\nu \frac{\partial g}{\partial n_z}(x_i, z_\nu)}_{=a_{i+k,\nu}} \underbrace{g(z_\nu, y_j)}_{=b_{j+k,\nu}}$$

can be written as $G|_{t\times s} \approx AB^T$.

Cross approximation applied to A yields $\tau \subseteq t$ and $V \in \mathbb{R}^{t\times\tau}$ with

$$A \approx VA|_{\tau\times 2k}$$

Green cross approximation

The quadrature approximation

$$\underbrace{g(x_i, y_j)}_{=g_{ij}} \approx \sum_{\nu=1}^{k} \underbrace{w_\nu g(x_i, z_\nu)}_{=a_{i\nu}} \underbrace{\frac{\partial g}{\partial n_z}(z_\nu, y_j)}_{=b_{j\nu}} \underbrace{-w_\nu \frac{\partial g}{\partial n_z}(x_i, z_\nu)}_{=a_{i+k,\nu}} \underbrace{g(z_\nu, y_j)}_{=b_{j+k,\nu}}$$

can be written as $G|_{t\times s} \approx AB^T$.

Cross approximation applied to A yields $\tau \subseteq t$ and $V \in \mathbb{R}^{t\times\tau}$ with

$$A \approx VA|_{\tau\times 2k}, \qquad G|_{t\times s} \approx AB^T$$

Green cross approximation

The quadrature approximation

$$\underbrace{g(x_i, y_j)}_{=g_{ij}} \approx \sum_{\nu=1}^{k} \underbrace{w_\nu g(x_i, z_\nu)}_{=a_{i\nu}} \underbrace{\frac{\partial g}{\partial n_z}(z_\nu, y_j)}_{=b_{j\nu}} \underbrace{-w_\nu \frac{\partial g}{\partial n_z}(x_i, z_\nu)}_{=a_{i+k,\nu}} \underbrace{g(z_\nu, y_j)}_{=b_{j+k,\nu}}$$

can be written as $G|_{t\times s} \approx AB^T$.

Cross approximation applied to A yields $\tau \subseteq t$ and $V \in \mathbb{R}^{t\times\tau}$ with

$$A \approx VA|_{\tau\times 2k}, \qquad G|_{t\times s} \approx AB^T \approx VA|_{\tau\times 2k} B^T$$

Green cross approximation

The quadrature approximation

$$\underbrace{g(x_i, y_j)}_{=g_{ij}} \approx \sum_{\nu=1}^{k} \underbrace{w_\nu g(x_i, z_\nu)}_{=a_{i\nu}} \underbrace{\frac{\partial g}{\partial n_z}(z_\nu, y_j)}_{=b_{j\nu}} \underbrace{-w_\nu \frac{\partial g}{\partial n_z}(x_i, z_\nu)}_{=a_{i+k,\nu}} \underbrace{g(z_\nu, y_j)}_{=b_{j+k,\nu}}$$

can be written as $G|_{t\times s} \approx AB^T$.

Cross approximation applied to A yields $\tau \subseteq t$ and $V \in \mathbb{R}^{t\times\tau}$ with

$$A \approx VA|_{\tau\times 2k}, \qquad G|_{t\times s} \approx AB^T \approx VA|_{\tau\times 2k} B^T \approx VG|_{\tau\times s}.$$

Green cross approximation

The quadrature approximation

$$\underbrace{g(x_i, y_j)}_{=g_{ij}} \approx \sum_{\nu=1}^{k} \underbrace{w_\nu g(x_i, z_\nu)}_{=a_{i\nu}} \underbrace{\frac{\partial g}{\partial n_z}(z_\nu, y_j)}_{=b_{j\nu}} \underbrace{-w_\nu \frac{\partial g}{\partial n_z}(x_i, z_\nu)}_{=a_{i+k,\nu}} \underbrace{g(z_\nu, y_j)}_{=b_{j+k,\nu}}$$

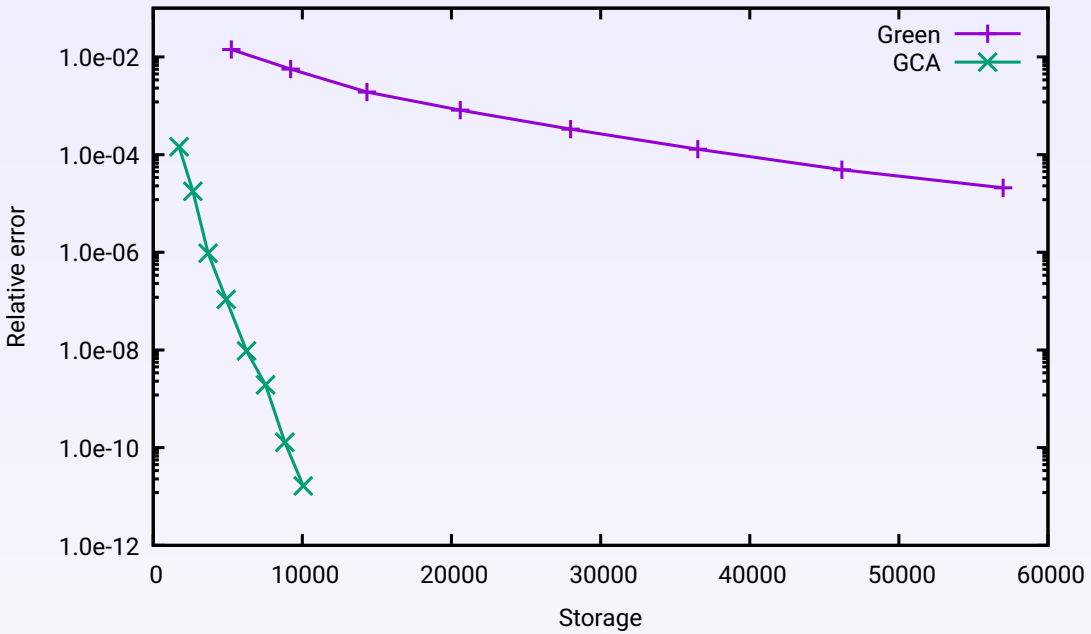
can be written as $G|_{t\times s} \approx AB^T$.

Cross approximation applied to A yields $\tau \subseteq t$ and $V \in \mathbb{R}^{t\times\tau}$ with

$$A \approx VA|_{\tau\times 2k}, \qquad G|_{t\times s} \approx AB^T \approx VA|_{\tau\times 2k} B^T \approx VG|_{\tau\times s}.$$

Accuracy: Green quadrature only used to construct V.
Block approximation results from algebraic interpolation of original matrix.

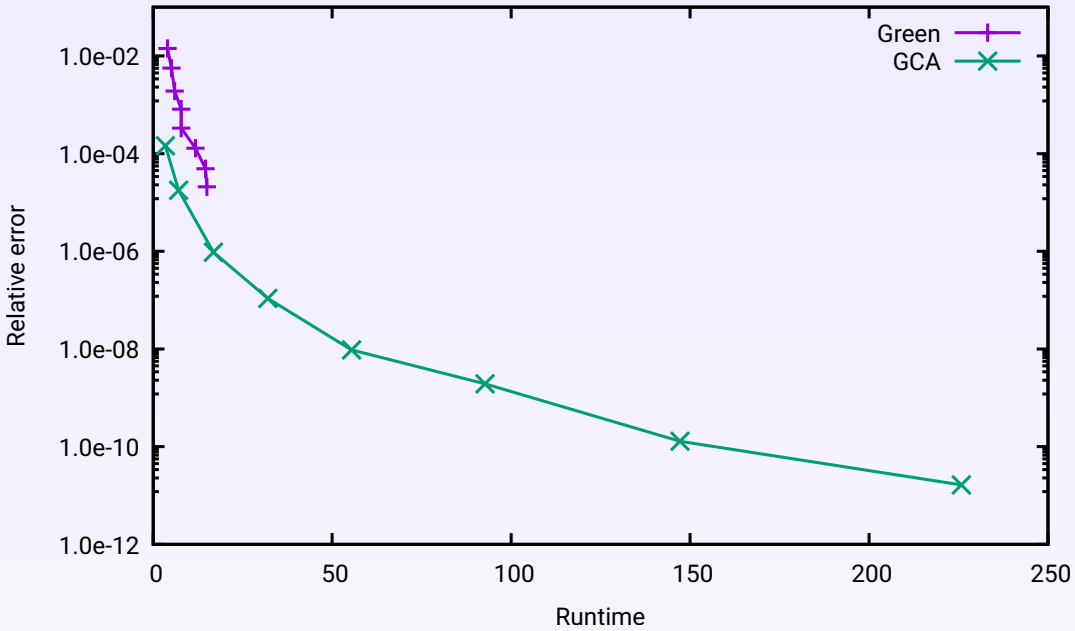
Experiment: Green cross approximation



Observations:

- ✔ Cross approximation significantly improves the efficiency.

Experiment: Green cross approximation



Observations:

- ✔ Cross approximation significantly improves the efficiency.
- ✗ Cross approximation quite slow for higher accuracies.

Hierarchical cross approximation

Symmetry: We can repeat the GCA procedure with columns instead of rows to get $\sigma \subset s$ and $W \in \mathbb{R}^{s \times \sigma}$ with

$$G|_{t \times s} \approx G|_{t \times \sigma} W^T$$

Hierarchical cross approximation

Symmetry: We can repeat the GCA procedure with columns instead of rows to get $\sigma \subset s$ and $W \in \mathbb{R}^{s \times \sigma}$ with

$$G|_{t \times s} \approx G|_{t \times \sigma} W^T \approx V G|_{\tau \times \sigma} W^T.$$

Hierarchical cross approximation

Symmetry: We can repeat the GCA procedure with columns instead of rows to get $\sigma \subset s$ and $W \in \mathbb{R}^{s \times \sigma}$ with

$$G|_{t \times s} \approx G|_{t \times \sigma} W^T \approx V G|_{\tau \times \sigma} W^T.$$

Hierarchy: If $t = t_1 \dot{\cup} t_2$ and τ_1, τ_2 and V_1, V_2 have already been computed, we have

$$G|_{t \times s} = \begin{pmatrix} G|_{t_1 \times s} \\ G|_{t_2 \times s} \end{pmatrix}$$

Hierarchical cross approximation

Symmetry: We can repeat the GCA procedure with columns instead of rows to get $\sigma \subset s$ and $W \in \mathbb{R}^{s \times \sigma}$ with

$$G|_{t \times s} \approx G|_{t \times \sigma} W^T \approx V G|_{\tau \times \sigma} W^T.$$

Hierarchy: If $t = t_1 \dot{\cup} t_2$ and τ_1, τ_2 and V_1, V_2 have already been computed, we have

$$G|_{t \times s} = \begin{pmatrix} G|_{t_1 \times s} \\ G|_{t_2 \times s} \end{pmatrix} \approx \begin{pmatrix} V_1 G|_{\tau_1 \times s} \\ V_2 G|_{\tau_2 \times s} \end{pmatrix}$$

Hierarchical cross approximation

Symmetry: We can repeat the GCA procedure with columns instead of rows to get $\sigma \subset s$ and $W \in \mathbb{R}^{s \times \sigma}$ with

$$G|_{t \times s} \approx G|_{t \times \sigma} W^T \approx V G|_{\tau \times \sigma} W^T.$$

Hierarchy: If $t = t_1 \dot{\cup} t_2$ and τ_1, τ_2 and V_1, V_2 have already been computed, we have

$$G|_{t \times s} = \begin{pmatrix} G|_{t_1 \times s} \\ G|_{t_2 \times s} \end{pmatrix} \approx \begin{pmatrix} V_1 G|_{\tau_1 \times s} \\ V_2 G|_{\tau_2 \times s} \end{pmatrix} = \begin{pmatrix} V_1 & \\ & V_2 \end{pmatrix} \begin{pmatrix} G|_{\tau_1 \times s} \\ G|_{\tau_2 \times s} \end{pmatrix}$$

Hierarchical cross approximation

Symmetry: We can repeat the GCA procedure with columns instead of rows to get $\sigma \subset s$ and $W \in \mathbb{R}^{s \times \sigma}$ with

$$G|_{t \times s} \approx G|_{t \times \sigma} W^T \approx V G|_{\tau \times \sigma} W^T.$$

Hierarchy: If $t = t_1 \dot{\cup} t_2$ and τ_1, τ_2 and V_1, V_2 have already been computed, we have

$$G|_{t \times s} = \begin{pmatrix} G|_{t_1 \times s} \\ G|_{t_2 \times s} \end{pmatrix} \approx \begin{pmatrix} V_1 G|_{\tau_1 \times s} \\ V_2 G|_{\tau_2 \times s} \end{pmatrix} = \begin{pmatrix} V_1 & \\ & V_2 \end{pmatrix} G|_{(\tau_1 \dot{\cup} \tau_2) \times s}$$

Hierarchical cross approximation

Symmetry: We can repeat the GCA procedure with columns instead of rows to get $\sigma \subset s$ and $W \in \mathbb{R}^{s \times \sigma}$ with

$$G|_{t \times s} \approx G|_{t \times \sigma} W^T \approx V G|_{\tau \times \sigma} W^T.$$

Hierarchy: If $t = t_1 \dot{\cup} t_2$ and τ_1, τ_2 and V_1, V_2 have already been computed, we have

$$G|_{t \times s} = \begin{pmatrix} G|_{t_1 \times s} \\ G|_{t_2 \times s} \end{pmatrix} \approx \begin{pmatrix} V_1 G|_{\tau_1 \times s} \\ V_2 G|_{\tau_2 \times s} \end{pmatrix} = \begin{pmatrix} V_1 & \\ & V_2 \end{pmatrix} \widehat{G},$$

Hierarchical cross approximation

Symmetry: We can repeat the GCA procedure with columns instead of rows to get $\sigma \subset s$ and $W \in \mathbb{R}^{s \times \sigma}$ with

$$G|_{t \times s} \approx G|_{t \times \sigma} W^T \approx V G|_{\tau \times \sigma} W^T.$$

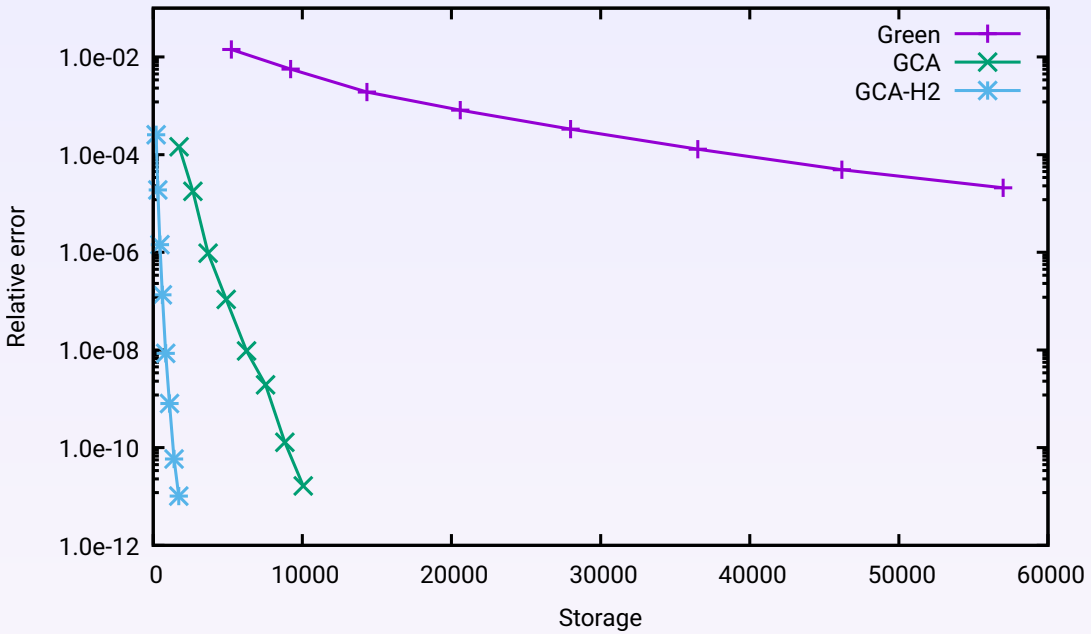
Hierarchy: If $t = t_1 \dot{\cup} t_2$ and τ_1, τ_2 and V_1, V_2 have already been computed, we have

$$G|_{t \times s} = \begin{pmatrix} G|_{t_1 \times s} \\ G|_{t_2 \times s} \end{pmatrix} \approx \begin{pmatrix} V_1 G|_{\tau_1 \times s} \\ V_2 G|_{\tau_2 \times s} \end{pmatrix} = \begin{pmatrix} V_1 & \\ & V_2 \end{pmatrix} \widehat{G},$$

so we can apply the GCA procedure to the significantly smaller matrix $\widehat{G} := G|_{\hat{\tau} \times s}$, $\hat{\tau} := \tau_1 \cup \tau_2$, to find $\tau \subseteq \hat{\tau}$ and $\widehat{V} \in \mathbb{R}^{\hat{\tau} \times \tau}$ with

$$\widehat{G} \approx \widehat{V} \widehat{G}|_{\tau \times s}, \qquad G|_{t \times s} \approx \begin{pmatrix} V_1 & \\ & V_2 \end{pmatrix} \widehat{V} G|_{\tau \times s}.$$

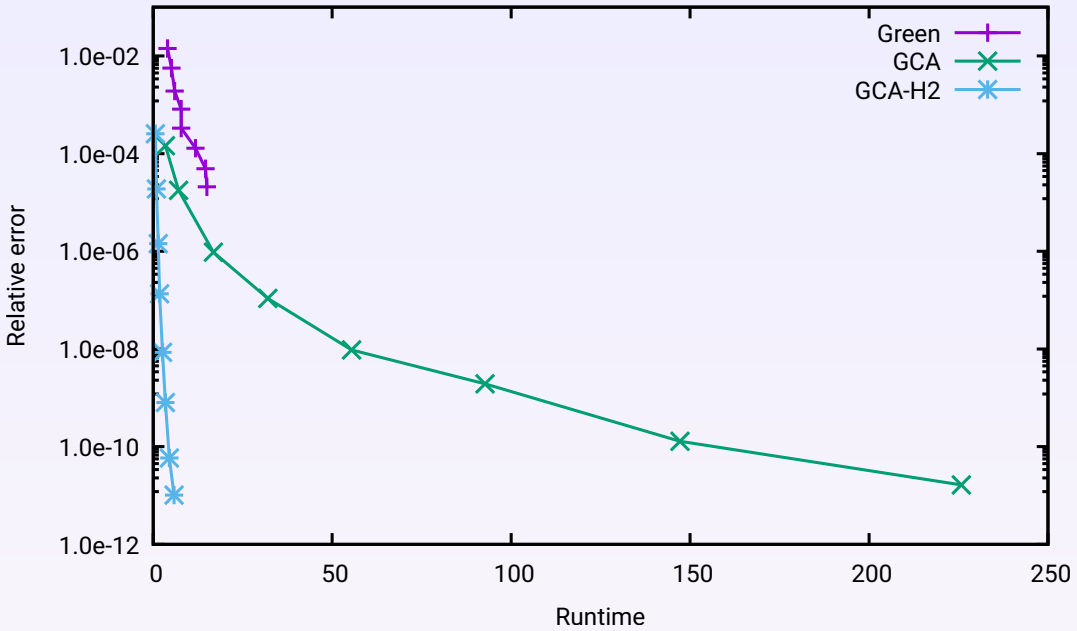
Experiment: Green cross approximation for $\mathcal{H}^2$-matrices



Observations:

- ✔ Additional hierarchy improves efficiency even further.

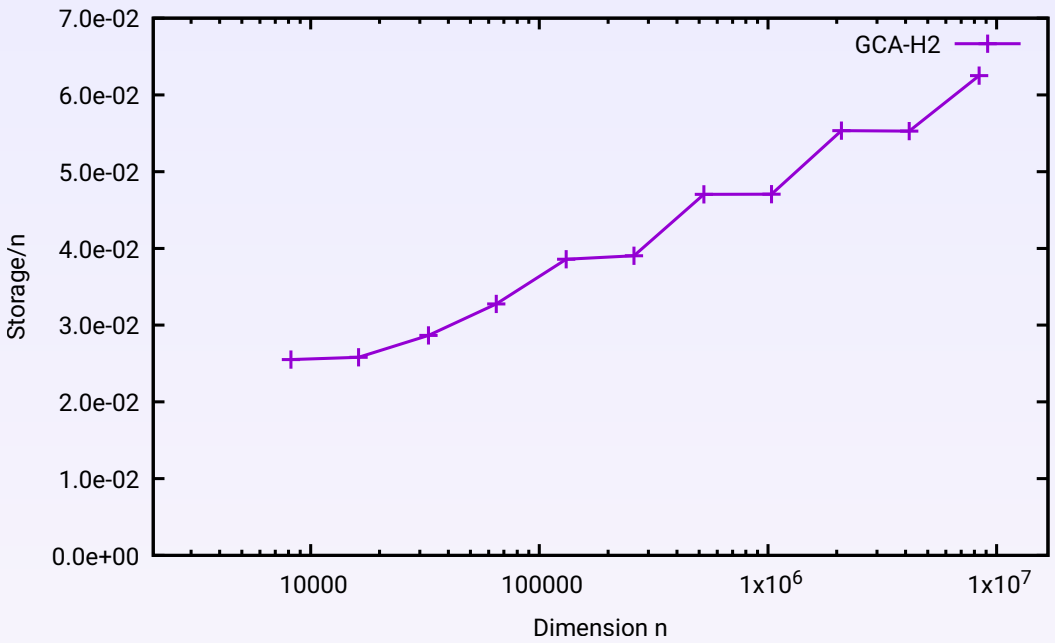
Experiment: Green cross approximation for $\mathcal{H}^2$-matrices



Observations:

- ✔ Additional hierarchy improves efficiency even further.
- ✔ Additional hierarchy greatly decreases the runtime.

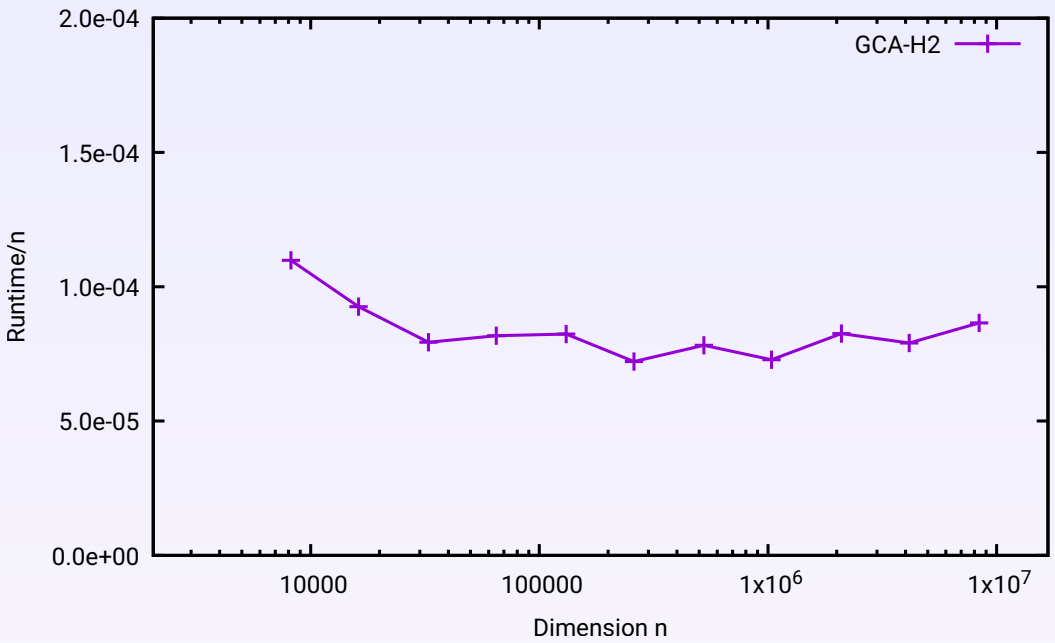
Experiment: Scaling



Observations:

- Storage appears to grow like $\mathcal{O}(n \log n)$.
- $n = 1\,036\,800$ requires 48 GB, $n = 8\,388\,608$ requires 512 GB.

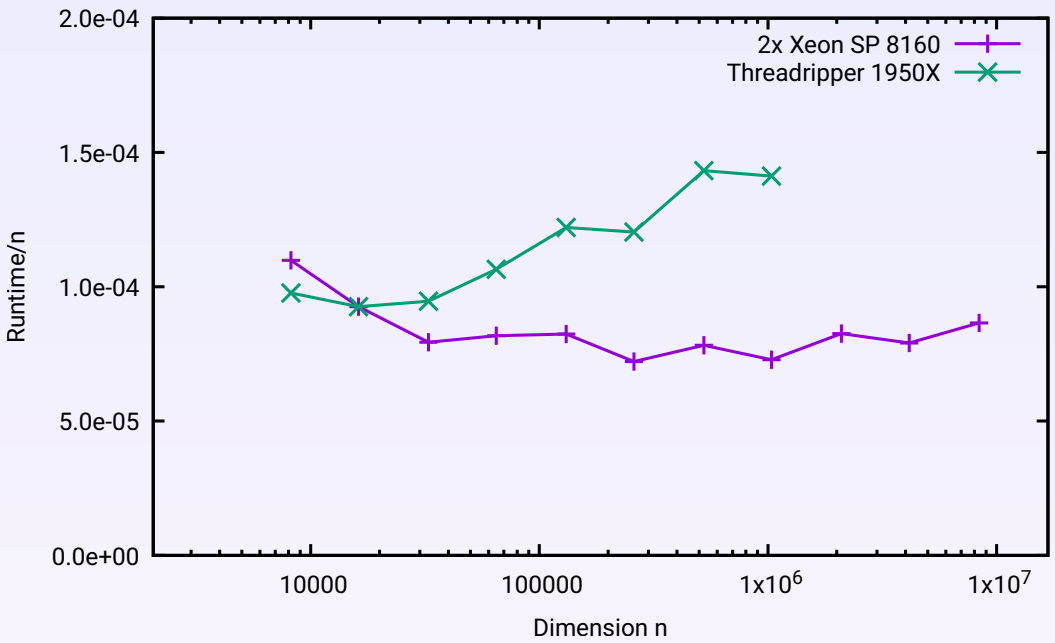
Experiment: Scaling



Observations:

- Storage appears to grow like $\mathcal{O}(n \log n)$.
- $n = 1\,036\,800$ requires 48 GB, $n = 8\,388\,608$ requires 512 GB.
- $n = 1\,036\,800$ requires 76 seconds to set up, 120 seconds to solve.
- $n = 8\,388\,608$ requires 730 seconds to set up, 2020 seconds to solve.

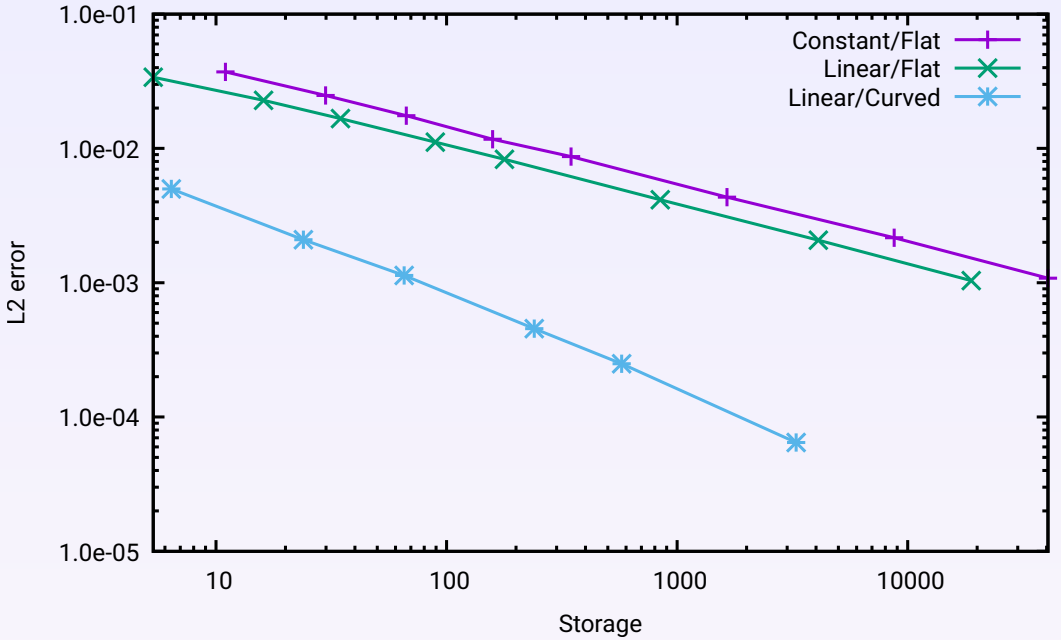
Experiment: Scaling



Observations:

- Storage appears to grow like $\mathcal{O}(n \log n)$.
- $n = 1\,036\,800$ requires 48 GB, $n = 8\,388\,608$ requires 512 GB.
- $n = 1\,036\,800$ requires 76 seconds to set up, 120 seconds to solve.
- $n = 8\,388\,608$ requires 730 seconds to set up, 2020 seconds to solve.

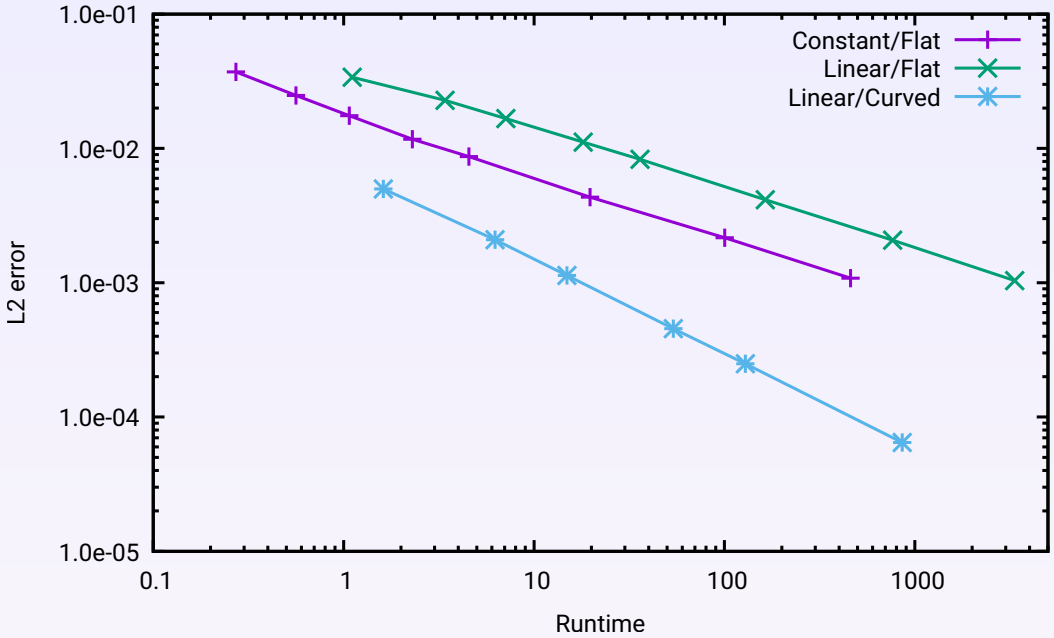
Experiment: Linear basis functions



Observations:

- ✔ Linear basis functions with curved triangles are very efficient.

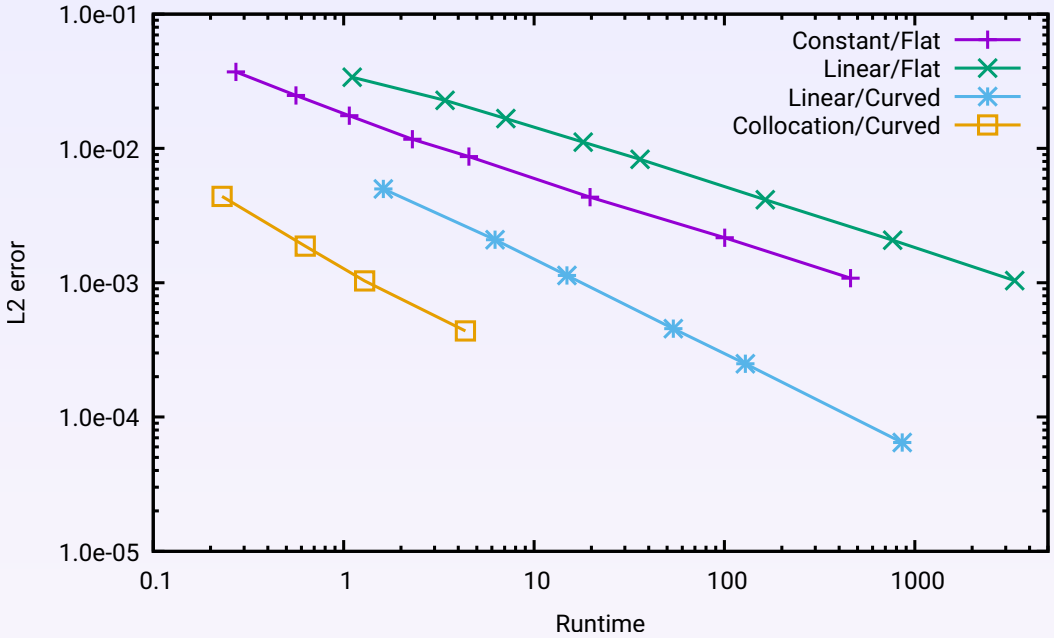
Experiment: Linear basis functions



Observations:

- ✔ Linear basis functions with curved triangles are very efficient.
- ✔ GCA-$\mathcal{H}^2$ compression works also for higher-order discretization.

Experiment: Linear basis functions



Observations:

- ✔ Linear basis functions with curved triangles are very efficient.
- ✔ GCA-$\mathcal{H}^2$ compression works also for higher-order discretization.

Overview

GPU computing

Goal: Speed up computation using graphics processors (GPUs).

Architectural differences between GPUs and CPUs:

- Larger number of arithmetic units
  (up to 3 840 compared to $28x16 = 448$),
- Larger memory bandwidth
  (up to 550 GB/s compared to 60 GB/s),
- Designed for large numbers of instances of the same short and simple shader program, not for general applications.
- Vector programming using the SIMT model.
- Limited on-board memory.

SIMT programming

Single Instruction, Multiple Threads:

- Program split into threads
  (each essentially a program counter and data registers).
- Threads are combined into warps.
- Every cycle, one warp and one instruction in the program is chosen.
  The instruction is executed by all threads that want to execute it,
  while all other threads do nothing.

Problem: If all threads in a warp want to execute different instructions, most threads do nothing in each cycle ("control flow divergence").

Consequence: Uniformity good, adaptivity bad.

GCA on GPUs

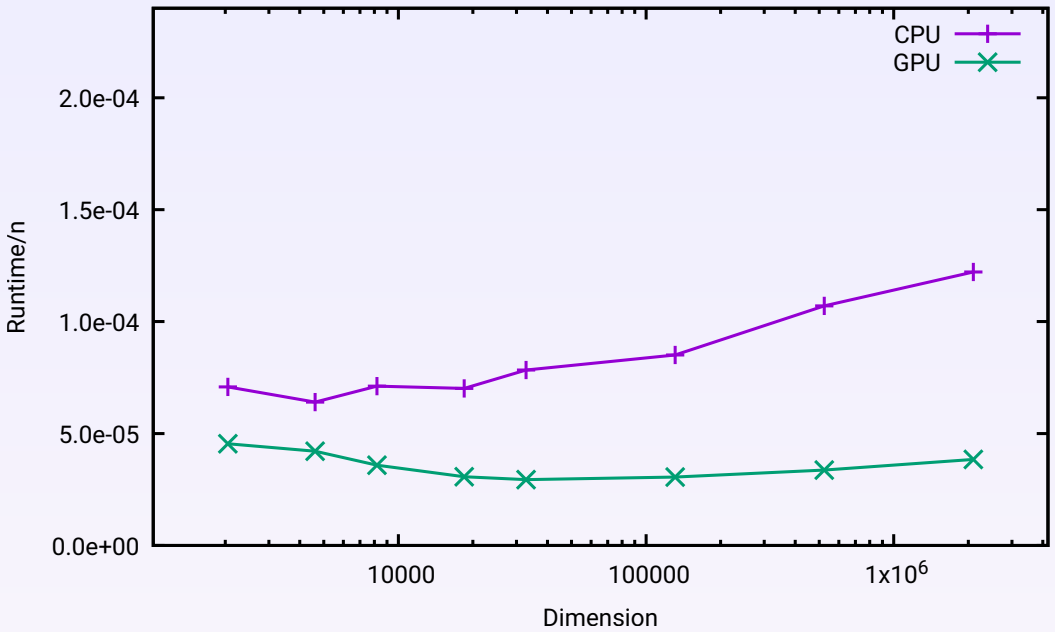
Idea: Since numerical quadrature is a compute-intensive, but simple task that requires only a small amount of data, it is well-suited for GPUs.

Implementation requires careful choices:

- Handle adaptive cross approximation on the CPU.
- Collect similar integrals in batches processed on the GPU.
- Send new batches to the GPU while others are processed.

Asynchronous execution: CPU prepares lists of integrals to be computed, GPU may process integrals in a different order than issued.

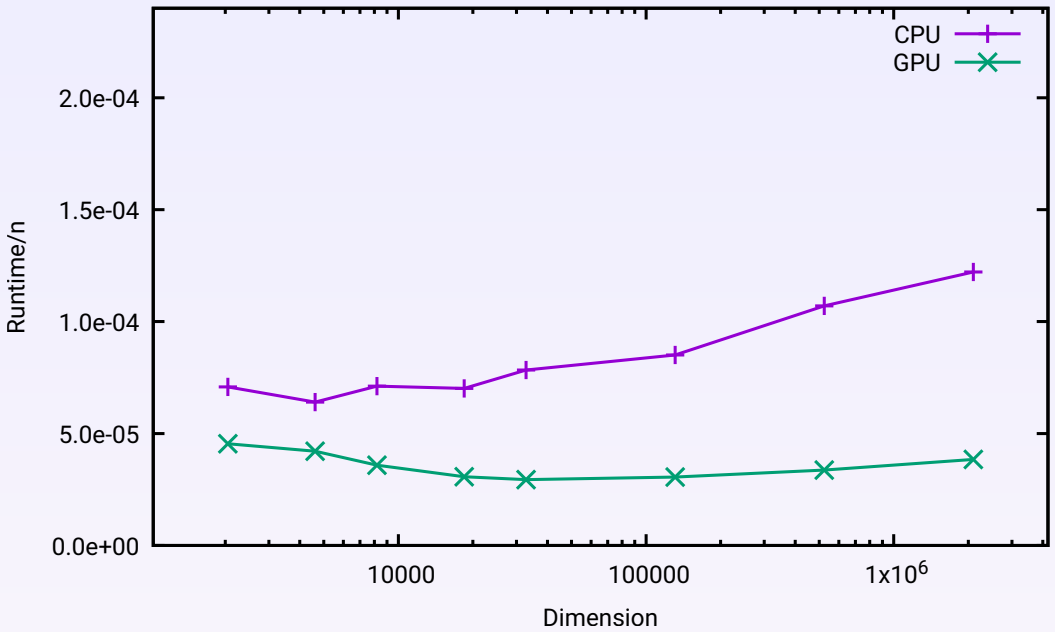
Experiment: GPU acceleration



Observations:

- ✔ Even compared to a highly optimized CPU implementation, the GPU offers significant advantages.

Experiment: GPU acceleration



Observations:

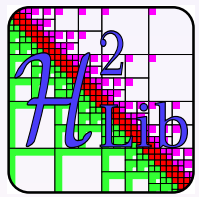
- ✔ Even compared to a highly optimized CPU implementation, the GPU offers significant advantages.
- ✔ The scaling behaviour for the GPU appears to be almost linear.

Summary

Compression algorithms like GCA-$\mathcal{H}^2$ allow boundary element applications to handle very large meshes on standard workstations.

Graphics processors can significantly reduced computing times, but obtaining optimal performance is challenging.

Work in progress: Preconditioners, distributed-memory parallelization, high-frequency scattering



`http://www.h2lib.org`