Fast Direct Solvers for Integral Equations in 3D

Recent developments and challenges

Eduardo Corona[1]
P.G. Martinsson[2] Denis Zorin[1] Abtin Rahimian [1]

[1]Courant Institute of Mathematical Sciences, NYU
[2]University of Colorado at Boulder

June 28, 2014

Table of Contents

Table of Contents

Target problems

- 2D integral equations

$$a(x)q(x) + \int_\Gamma K(x,y)q(y)dS_y = f(x)$$

- Integral equations on surfaces in R^3
- Homogeneous PDE on a 3D domain Ω,e.g.,

$$\Delta u = 0 \text{ in } \Omega,\ u = f \text{ on } \partial\Omega.$$

leads to

$$\frac{1}{2}q(x) + \frac{1}{4\pi}\int_{\partial\Omega} \frac{\partial}{\partial\nu_y}\frac{1}{|x-y|}q(y)dS_y = -f(x)$$

- Discretization: $Aq = f$
- We begin with a simpler case, integral equations on a planar domain

The goal: **Fast** direct solvers

- We want compact representation of A^{-1} that can be applied fast.
- Work and storage for these factorizations must scale optimally ($O(N)$ or $O(N \log N)$).
- Why a **direct solver**? robust solution to ill-conditioned problems. Ideal for multiple right hand sides. Local perturbations of A, time-dependent problems, optimization.
- Time and memory overhead of inverse construction. However, the solve is really fast, better constants than FMM matvec.

Previous and related work

- Direct solvers for sparse matrices
- H-matrices and H^2-matrices (Hackbusch, Börm, Grasedyck, Bebenof...)
- HSS matrices (Gu, Chandrasekaran, Xia, Li...)
- Skeletonization-based methods (Martinsson, Rokhlin, Gillman, Young, Greengard, Ho)
- Hierarchical Interpolative Factorization (Ho,Ying)
- Sivaram's talk today (HODLR / Inverse FMM)

FMM / $\mathcal{H}$ / HSS: Well-separated $\Longrightarrow$ Low Rank

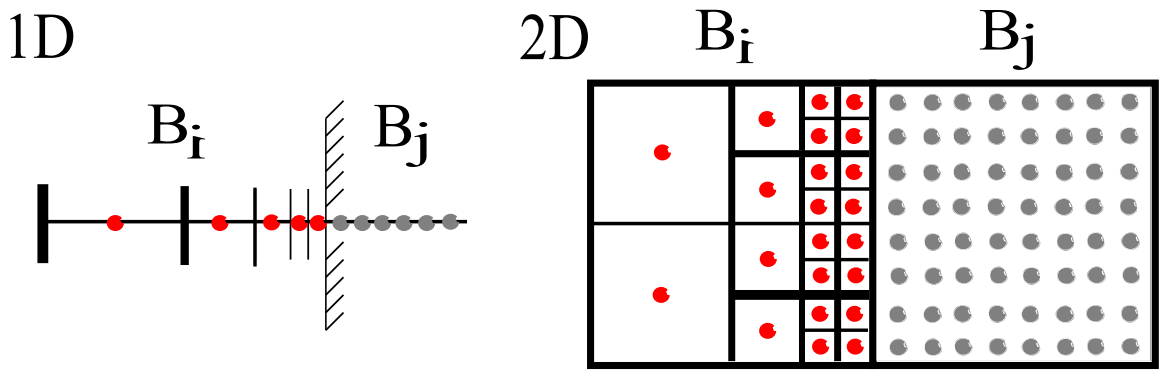
- Hierarchical division of our domain into boxes B (source / target tree data structures)
- Entries of block A_{ij} - kernel evaluations $\mathcal{K}(t_i, s_j)\omega_j$
- If sources and targets are **well-separated**, $\mathcal{K}(t_i, s_j)$ is smooth. A multipole expansion around center s_0 of B_j converges quickly:

$$\mathcal{K}(t_i, s_j) = \sum_{p=0}^{k} g_p(t_i - s_0) f_p(s_j - s_0) = \sum_{p=0}^{k} G_{ip} F_{pj}$$

And so A_{ij} is approximately low rank.

HSS: Not well-separated are still Low Rank

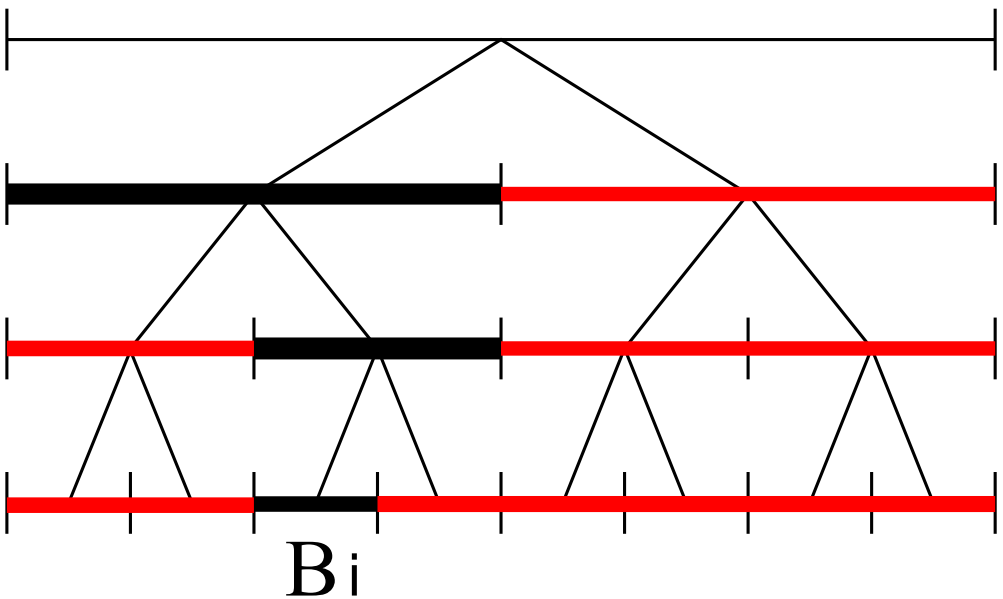
If B_i and B_j are **adjacent**, we can subdivide B_i into well-separated sets, whose interaction with B_j is constant rank.

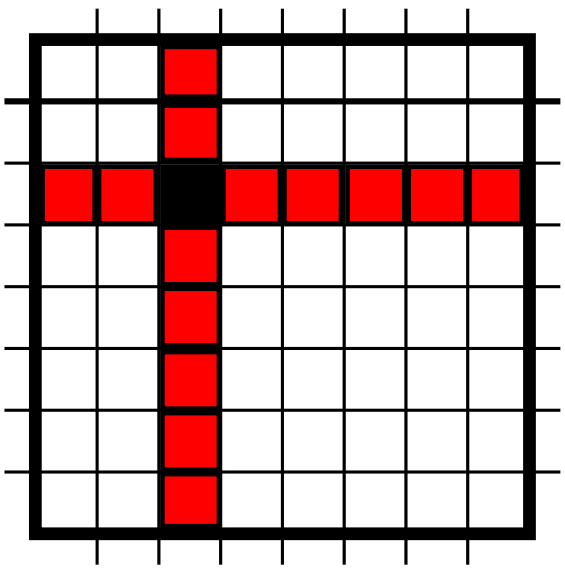


- In 1D, if B_i and B_j have n points, rank(A_{ij}) is $\mathbf{O(log(n))}$.
- In 2D, if B_i and B_j have n points, rank(A_{ij}) is $\mathbf{O(n^{1/2})}$.

Semi-Separable structure

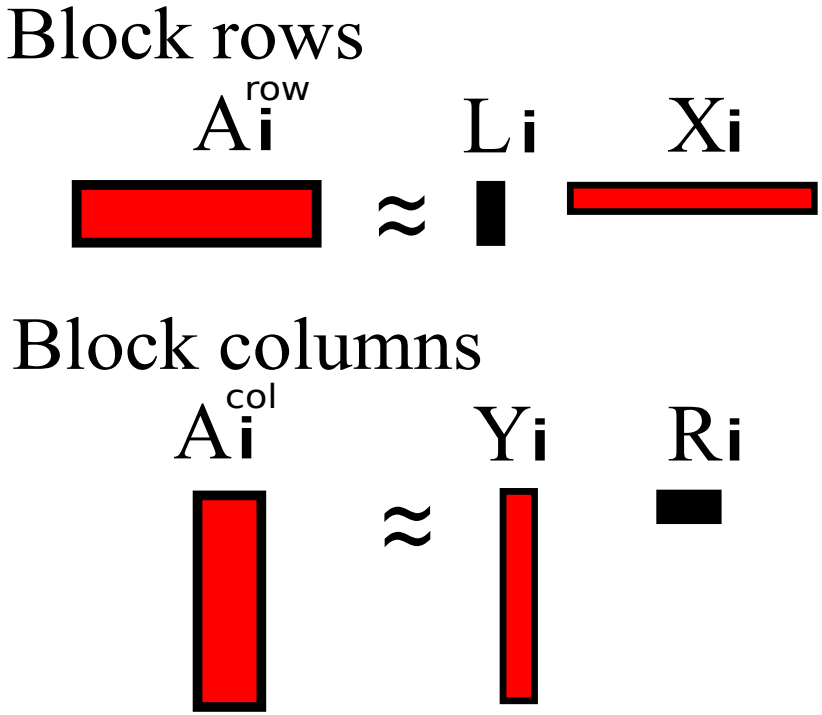
Interaction between points in a box B_i (black) and points outside (red) is $\approx$ low rank:

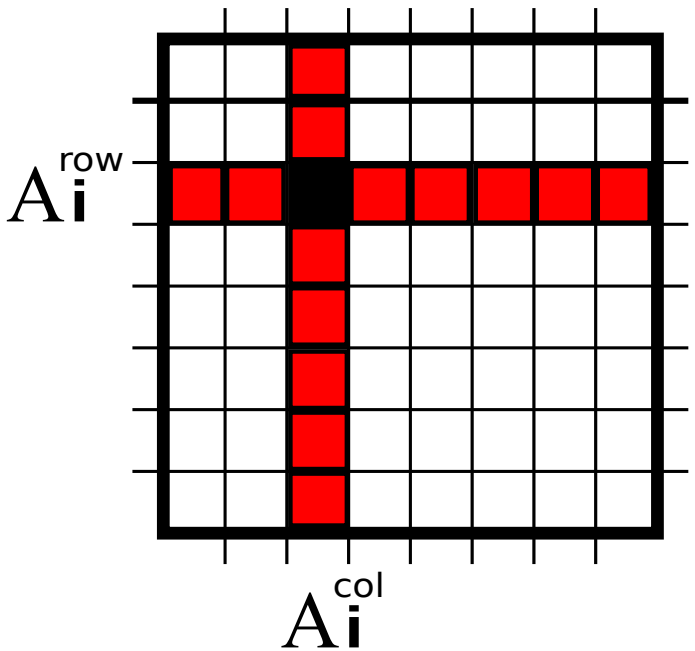




Semi-Separable structure

We define block rows $\mathcal{A}_i^{row}$ and block columns $\mathcal{A}_i^{col}$:

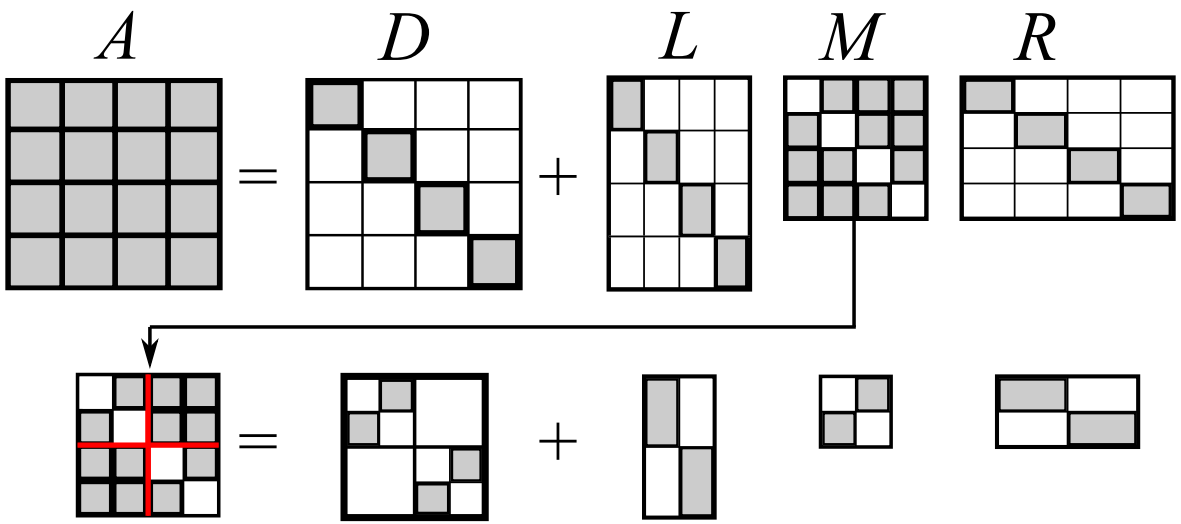




Hierarchical factorization of A

Hierarchical / Telescopic factorization: (Block-diagonal matrices)

$$\mathbf{A^d = D^d + L^d(D^{d-1} + L^{d-1}(D^{d-1} + \ldots(D^1 + L^1 D^0 R^1)\ldots)R^{d-1})R^d}$$



Note: FMM structure can be similarly laid out. D^ℓ are not block diagonal (neighbor or interaction list blocks)

Our strategy to obtain linear complexity:

In 2D, this leads to an $O(N^{3/2})$ algorithm for inversion.

1. We start with the inversion algorithm based on the 1D case.
2. **Skeleton sets have structure**
3. **Per-box operators defined on them are compressible:** The essential blocks of the inverse can be identified as local solution operators (scattering matrices).
4. We build operators in compressed form (low rank or 1D HSS), and replace dense matrix algebra appropriately.

Inverting a block-separable matrix

- Let $Z = F + LMR$. At the first step, $Z = A$, $F = D$.
- We solve the system $Zq = f$, with auxiliary variables $\phi = Rq$, $u = M\phi$

$$\begin{bmatrix} F & L & 0 \\ -R & 0 & I \\ 0 & -I & M \end{bmatrix} \begin{bmatrix} q \\ u \\ \phi \end{bmatrix} = \begin{bmatrix} f \\ 0 \\ 0 \end{bmatrix}$$

- Let $\tilde{R} = ERF^{-1}$, $\tilde{F} = F^{-1}(I - L\tilde{R})$ and $\tilde{L} = F^{-1}LE$
- Then $Z^{-1} = \tilde{F} + \tilde{L}(E + M)^{-1}\tilde{R}$, same form as Z!
- Apply recursively

Inverting block-separable matrix

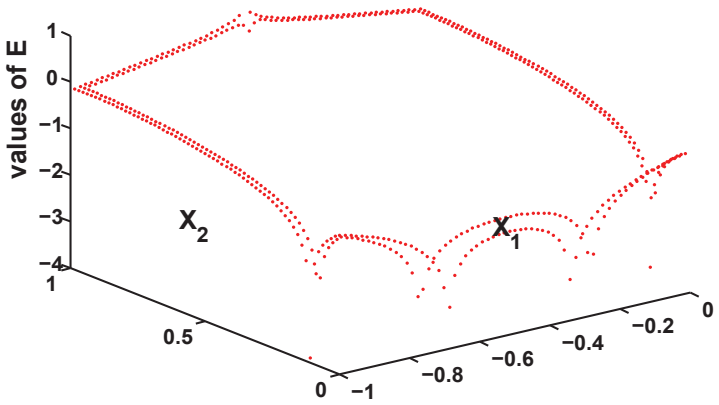
- Recursion formula for inverse, with $\tilde{A}^\ell = A^\ell + E^{\ell+1}$

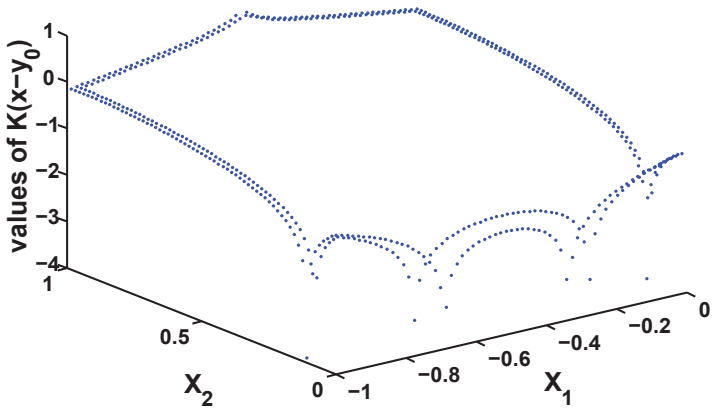
$$\left(\tilde{A}^\ell\right)^{-1} = \tilde{D}^\ell + \tilde{L}^\ell(\tilde{A}^{\ell-1})^{-1}\tilde{R}^\ell$$

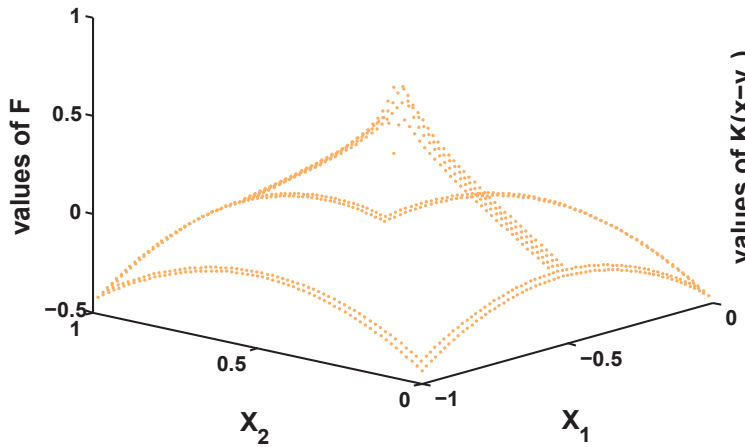
- If D, L and R are block diagonal, then so are $E, \tilde{R}, \tilde{L}$ and $\tilde{D}$. This means that these matrices can be computed inexpensively via independent computations that are local to each box.
- The factors in the inverse can be interpreted as follows:
  - $E^{-1} = RF^{-1}L$: *local solution operator* restricted to I^{sk}, maps potentials to charges.
  - $E + M$ maps charges on *the union of skeleton points* to potentials, adding diagonal Eq and off-diagonal Mq.

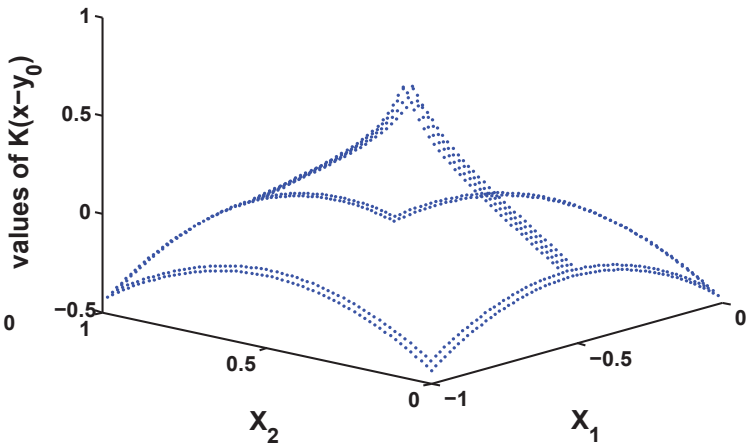
Operators E and F

- Why do we hope E and F compress well?









HSS Inversion Algorithm

for each box $\mathcal{B}_i$ in fine-to-coarse order **do**
  if $\mathcal{B}_i$ is a leaf **then**
    $F_i = D_i$
  else
    $$F_i = D_i + \begin{bmatrix} E_{c_1(i)} & \\ & E_{c_2(i)} \end{bmatrix}$$
  end if
  $\tilde{D}_{top} = F_{top}^{-1}$ {Direct inversion at the top level}
  if $\mathcal{B}_i$ below top-level **then**
    $E_i = (R_i F_i^{-1} L_i)^{-1}$
    $\tilde{R}_i = E_i R_i F_i^{-1}$
    $\tilde{D}_i = F_i^{-1}(I - L_i \tilde{R}_i)$
    $\tilde{L}_i = F_i^{-1} L_i E_i$
  end if
end for

How do we obtain a linear complexity algorithm?

Inverse Compression

- It does not suffice to compress the dense matrices above. We need to build the operators in compressed form as we go up the tree.
- Goal: compress and invert these operators in strictly better than $O(N/2^\ell)$ work.

Inverse matvec:

- Once compressed, all of these operators can be applied in $O((N/2^\ell)^{1/2})$ work, yielding a linear solve as well.

Compressed-block HSS strategy

Build essential blocks of the 2D HSS inverse in **compressed form**:

1. First, while building the binary tree, we compress the interpolation operators $T : I_i^{rs} \to I_i^{sk}$. This determines matrices L_i and R_i.
2. Go up the tree building inverse blocks F_i and E_i.
3. E_i depends on F_i, and for non-leaf boxes F_i depends on the E_i matrices for children. (formula shown by Gunnar for scattering matrices)
4. Build densely for small blocks, switch to compressed form at threshhold.

Matrix algebra we need

One-dimensional HSS and low-rank matrix algebra

- Dense-block HSS1D compression, inversion and matvec
- Fast addition and manipulation of HSS1D matrices:
- *Additional HSS compression routines*: LowRank to HSS1D, HSS1D Recompress (Xia)
- Randomized methods
- **NOT used:** HSS1D matrix-matrix product.

What have we achieved?

We have implemented this algorithm for the plane, and confirmed that work and storage scale linearly.

1. **Non Translation Invariant Kernels**: We compute and store one set of matrices (in compressed format) per box B_i. Work and storage scale linearly. $\rightarrow O(N)$
2. **Translation Invariant Kernels**: One set of matrices per level is computed. As a result, most of the work and storage scales sublinearly $\rightarrow O(N^{1/2} \log^2 N)$

Numerical experiments show outstanding performance for non-oscillatory and for low frequency oscillatory kernels, even at high target accuracies ($\varepsilon = 10^{-10}$).

Example: Lippman-Schwinger / Laplace kernel

Let A be an $N \times N$ matrix with entries:

$$A_{i,j} = \delta_{i,j} + h^2 b(x_i)\mathcal{G}(||x_i - x_j||)c(x_j) \tag{1}$$

Where $\{x_i\} \in [-1,1]^2$ are points on a regular grid with spacing h, b,c are given functions, and $\mathcal{G}(x)$ is the 2D Laplace kernel $\frac{1}{2\pi} log(x)$.
Then, it is a Nystrom discretization of:

$$\mathcal{A}[u](x) = u(x) + \int_\Omega b(x)\mathcal{G}(||x - y||)c(y)u(y)dy \tag{2}$$

Numerical Experiments: Setup

- Solving this equation then corresponds to the 0 frequency Lippman-Schwinger equation, which arises in scattering problems.
- If we pick b(x) and c(x) to be non-constant functions, the resulting matrix is **Non Translation Invariant**.
- Otherwise, $b(x) = c(y) = 1$ yields a **Translation Invariant** matrix.
- Given precision $\varepsilon = 10^{-10}$, we run scaling tests for inverse compression and Inverse matvec.
- Each test is run on one node of the NYU HPC Bowery cluster.

Translation-invariant inverse compression results

Compression time and memory usage for TI Laplace kernel.

N	HSS-D Time $O(N^{3/2})$	HSS-C Time $O(N)$	HSS-D Mem $O(N)$	HSS-C Mem $O(N)$
784	0.05 s	0.13 s	1.94 MB	1.75 MB
3136	0.21 s	0.98 s	9.04 MB	6.19 MB
12544	1.40 s	3.41 s	39.16 MB	19.03 MB
50176	9.68 s	10.76 s	163.19 MB	52.09 MB
200704	1.21 m	30.89 s	666.39 MB	151.41 MB
802816	9.20 m	1.59 m	2.61 GB	474.74 MB
3211264	1.19 hr	6.68 m	9.9359 GB	1.56 GB
12845056	9.28 hr	29.22 m	39.74 GB	5.29 GB

Non-translation-invariant inverse compression

Compression time and memory usage for NTI Laplace kernel.

N	HSS-D Time $O(N^{3/2})$	HSS-C Time $O(N)$	HSS-D Mem $O(N \log N)$	HSS-C Mem $O(N)$
784	0.11 s	0.17 s	4.68 MB	4.48 MB
3136	0.67 s	1.70 s	29.09 MB	25.24 MB
12544	4.50 s	8.32 s	159.59 MB	123.07 MB
50176	31.45 s	40.43 s	819.58 MB	538.51 MB
200704	3.79 m	3.23 m	3.72 GB	2.23 GB
802816	28.35 m	13.66 m	17.27 GB	9.23 GB
3211264	3.58 hr	54.795 m	70.99 GB	34.09 GB

Inverse Compression: HSS-C vs HSS-D

NTI case

- **Crossover point:** $N = 100,000$.
- **Inversion time:** By $N = 10^6$ our method is **4**$\times$ faster (55 min vs 4 hr).
- **Memory usage** It always uses less memory. By $N = 10^6$, it's about half (34 vs 71 GB)

TI case

- **Crossover point:** $N = 50,000$.
- It scales **sublinearly** for intermediate N
- **Inversion time** By $N = 10^7$, it is **19**$\times$ faster (29 min vs 9 hr)
- **Memory usage** By $N = 10^7$, **8**$\times$ less memory (5 vs 40 GB).

Inverse Apply (Solve) Timings

Average Time per Solve (**seconds**) for NTI and TI Laplace kernels.

N	NTI HSS-D $O(N \log N)$	NTI HSS-C $O(N)$	TI HSS-D $O(N \log N)$	TI HSS-C $O(N)$
784	0.0014	0.0018	0.0007	0.0011
3136	0.0064	0.0090	0.0031	0.0046
12544	0.0292	0.0362	0.0137	0.0162
50176	0.1320	0.1546	0.0590	0.0600
200704	0.5993	0.6772	0.2819	0.2512
802816	2.6611	2.8193	1.2709	1.0763
3211264	11.816	11.737	5.77296	4.5650
12845056	(52.468)	(48.8641)	25.8312	19.3619

Inverse Apply Observations

- For all cases, the Inverse Apply is **really fast**.
- There is little difference between HSS-C and HSS-D solve times. However, HSS-C provides a faster and more memory efficient inverse compression.
- For example: For $N = 3$ million, our method compresses the inverse in **7 minutes** using just **1.5 GB** of memory. Each solve then takes **5 seconds**.
- For the same example, the NTI case compresses the inverse in **55 minutes** using **34 GB** of memory. Each solve then takes **12 seconds**.

Table of Contents

Extension of HSS-C and related methods to 3D:

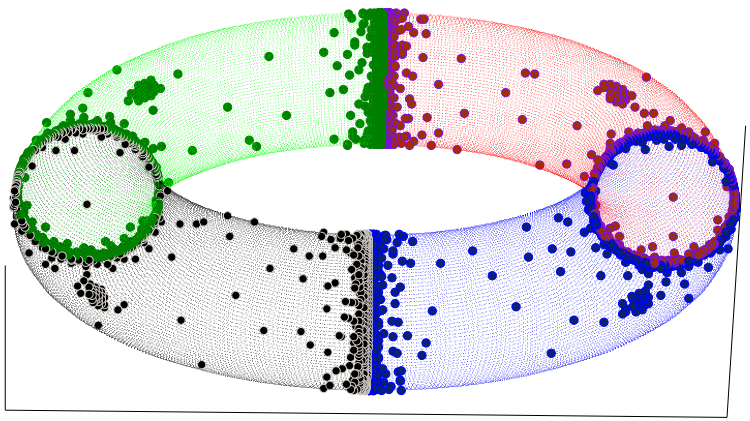
- On the plane, great performance and scaling, at high target accuracies ($\varepsilon = 10^{-10}$).
- Extending the 2D algorithms to parametric surfaces:
  - **Skeleton sets and equivalent surfaces are thicker**.
  - Boundary layers are not enough, have to find the rest of the skeleton sets adaptively.
  - **Ranks in compressed blocks** are also higher.
- Existing methods use a *lot* of memory, and experimental scaling is less satisfying
- So far, only feasible for low-mid accuracies ($\varepsilon = 10^{-4} - 10^{-6}$)

Why is inverse storage so expensive?

- As opposed to matrix compression / apply, the entries of A^{-1} are not kernel evaluations, and so they can't be computed on the fly.
- Blocks are stored for each box in the hierarchy, even in the presence of symmetries in the volume.
- Ranks of interactions grow considerably, and so contants grow.
- Even if a lot of memory is available, slow access to memory, and flop / memory might not be good.

Example: Skeleton sets on a torus (Laplace 3D Single Layer)

Skeleton sets for 4 boxes on a torus, for accuracy 10^{-6}. The number of layers required is higher than on the plane, and additional points appear inside the boxes.



Numerical results on the torus (Laplace 3D Single Layer)

Inverse compression time, memory usage and solve time for HSS-C ($\varepsilon = 10^{-6}$)

N	HSS-C Time	HSS-C Mem	HSS-C solve
16384	38 s	219 MB	0.03 s
65536	2.86 m	1.11 GB	0.17 s
262144	12.67 m	5.17 GB	0.79 s
1048576	1.31 h	22.28 GB	3.57 s
4194304	(8.13 h)	(95.99 GB)	(16 s)

If we can build the matrix, the apply remains quite fast. However, we quickly run out of memory, and this is more dramatic for high accuracy.

What can be done? Current and future work

We are currently working on representations of the inverse matrix that provide further compression, while maintaining high performance.

- At each level, we store a set of operators per box (e.g. a scattering or interpolation matrices). It might be possible to represent all such operators in terms of a reduced basis.
- Well-separated interactions can be mapped to proxy surfaces (as in the kernel independent FMM). Further compression may be more accesible for these interactions (especially if the kernel is translation invariant in 3D).

Tensor Train decomposition for integral equations

- The Tensor Train (TT) decomposition (Oseledets et al) is an extremely efficient numerical method to compress tensors.
- It overcomes curse of dimensionality: for many d dimensional tensors from applications, work and storage are $O(d)$.
- This decomposition can also be applied to an $2^L \times 2^L$ matrix, by reshaping it as a 2L dimensional tensor.
- Integral equations: it compresses all interactions (near and far) at a given tree level simultaneously. For volume integral equations, compression, inversion and storage are $O(\log N)$!

TT for volume integrals

- We applied the TT decomposition to the Laplace and low frequency Helmholtz single layer kernels in 2D and 3D volume (Lippmann Schwinger)
- Work and storage to compress the matrix A and its inverse theoretically scale like $O(\log N)$. Solve stage is $O(N \log N)$
- Experimental scaling looks even better.
- Inverse storage is tiny, often requiring only a few MBs.
- For example, for N=4194304, it takes only 12 seconds and 1 MB to compute the inverse for 2D Laplace ($\varepsilon = 10^{-6}$). The same example takes 71 seconds and 350 MB for HSS-C.
- Similar results for 3D Laplace.
- We can perform compression and inversion up to $N \sim 10^7 - 10^8$ for mid to high accuracies.

Challenge: Use TT to build boundary integral solvers in 3D

- It is possible to apply TT as is to boundary integral equations.
- For simple surfaces (e.g. torus, sphere), we get similar results as in the volume.
- For more complicated surfaces, compressing all interactions at a given level is too ambitious. Local interactions do not compress well if the geometry is complex.
- We are currently working on incorporating the TT as a tool in a hybrid method.
- We believe it can be used to compress far / well-separated interactions very successfully.