A Tensor-Train accelerated solver for integral equations in complex geometries

Eduardo Corona[a], Abtin Rahimian[b], Denis Zorin[b]

[a]*Department of Mathematics, University of Michigan, Ann Arbor, MI 48109*
[b]*Courant Institute of Mathematical Sciences, New York University, New York, NY 10003*

Abstract

We present a framework using the Tensor Train decomposition (TT) to accurately and efficiently solve volume and boundary integral equations in three dimensions. We describe how the TT decomposition can be used as a hierarchical compression and inversion scheme for matrices arising from the discretization of integral equations. For a broad range of problems, computational and storage costs of the inversion scheme are extremely modest $\mathcal{O}(\log N)$ and once the inverse is computed, it can be applied in $\mathcal{O}(N \log N)$.

We analyze the TT ranks for hierarchically low rank matrices and discuss its relationship to commonly used hierarchical compression techniques such as FMM, HSS, and wavelets. We prove that the TT ranks are bounded for translation-invariant systems and argue that this behavior extends to non-translation invariant volume and boundary integrals.

For volume integrals, the TT decomposition provides an efficient direct solver requiring significantly less memory compared to other fast direct solvers. We present results demonstrating the remarkable performance of the TT-based solver when applied to both translation and non-translation invariant volume integrals in 3D.

For boundary integral equations, we demonstrate that using a TT decomposition to construct preconditioners for a Krylov subspace method leads to an efficient and robust solver with a small memory footprint. We test the TT preconditioners in the iterative solution of an exterior elliptic boundary value problem (Laplace) formulated as a boundary integral equation in complex, multiply connected geometries.

Keywords: Integral equations, Tensor Train decomposition, Preconditioned iterative solver, Complex Geometries, Fast Multipole Methods, Hierarchical matrix compression and inversion

1. Introduction

We aim to efficiently and accurately solve equations of the form

$$a\sigma(x) + \int_\Omega b(x)K(x,y)c(y)\sigma(y)\,d\Omega_y = f(x), \qquad \text{for all } x \in \Omega, \tag{IE}$$

where Ω is a domain in $\mathbb{R}^3$ (either a boundary or a volume). When $a \neq 0$, the integral equation is Fredholm of the second kind, which is the case for all equations presented in this work. A large class of physics problems, typically formulated as PDEs, may be cast in this form. This approach is generally advantageous, garnering as much gains in dimensionality reduction and conditioning as the physics of the problem allows. Generally, $K(x,y)$ is a *kernel function* derived from the fundamental solution associated with the relevant elliptic PDE—e.g. the Laplace, Helmholtz, or Stokes equations, etc. The kernel function is typically singular near the diagonal (as y approaches x) but is smooth otherwise. For the purposes of this paper, we also assume it is not highly oscillatory.

A discretization of Eq. (IE) produces a linear system of equations

$$\mathsf{A}\sigma = \mathsf{f}, \tag{LS}$$

Email addresses: `coronae@umich.edu` (Eduardo Corona), `arahimian@acm.org` (Abtin Rahimian), `dzorin@cs.nyu.edu` (Denis Zorin) where A is a dense $N \times N$ matrix. Krylov subspace methods such as GMRES [SS86] coupled with the rapid evaluation algorithms such as FMM [GR87] are widely used to solve this system of equations. However, the performance of the iterative solver is directly affected by the eigenspectrum of Eq. (IE).

The eigenspectrum of the system, while typically independent of the resolution of the discretization, can greatly vary, depending, in particular, on the geometric complexity of Ω and the kernel K. The cases when conditioning and spectral clustering are good are common, and the system is solved in a few iteration using a suitable iterative method. This however ceases to be the case for a number of problems of interest (e.g., when different parts of the boundary approaching each other closely). Such problems may either be solved by constructing an effective preconditioner for the iterative solver or by using *direct solvers*, in which the system is solved in fixed time independent of the distribution of the spectrum.

There have been a number of efforts to develop robust, fast direct solvers with linear complexity for systems given in Eq. (LS). When Ω is a contour in the plane, extremely efficient $\mathcal{O}(N)$ algorithms such as [MR05] exist. These algorithms may be extended to volumes in 2D and surfaces in 3D, producing direct solvers with complexity $\mathcal{O}\left(N^{3/2}\right)$ [Gil+12, HG12, Gil11]. More recently, approaches that aim to extend linear complexity to Hierarchically Semi-Separable (HSS) matrices have been developed [Cor+14, HY15]. Furthermore, a general inverse algorithm has been proposed for FMM matrices [AD14].

These approaches have many advantages, and when feasible, yield compressed representations of the inverse of A which can be efficiently applied, often many times faster than an FMM matrix-vector multiply. For 2D problems, this type of methods have excellent performance and remain practical even at high target accuracies, e.g 10^{-10}.

For volume or boundary integral equations in 3D, especially in complex geometries, algorithmic constants in the complexity of these methods grow considerably as a function of accuracy. In particular a large amount of memory is needed, limiting the range of practical target accuracies one can achieve for a given problem, and complicating efficient parallel implementation, due to the need to store and communicate large amounts of data. This also undermines their efficiency in terms of ratio of the size of the problem solved to the amount of memory and computation required by the solver [Wil+09].

These two solution schemes, i.e. simple iterative solvers (requiring only matrix-vector multiplication) and direct solvers, represent two extremes of the spectrum of *preconditioned iterative solvers*. In terms of computational cost and required memory, a direct solver produces an expensive preconditioner that requires a single solver iteration to achieve the desired accuracy. On the other extreme, a non-preconditioned iterative solver requires only the precomputation and storage of the compressed form of the matrix, but may require a large number of iterations for each solve. Within this spectrum lies preconditioned iterative solvers, where an approximate low accuracy inverse is used as a preconditioner, requiring less memory and precomputation time than a direct solver, at the expense of multiple iterations to achieve the desired accuracy.

By choosing the accuracy of the approximate inverse, the preconditioned solver approach makes it possible to find a "sweet spot" for a particular type of problem and let the practitioner strike a reasonable trade-off between precomputation time and solve time, within the available memory budget.

This paper describes how the tensor train decomposition (TT) [OT10] can be used as a hierarchical compression and inversion scheme for matrix A. We show that for a range of accuracies of the inverse, TT decomposition achieves significantly higher compression by finding a common basis for all interactions at a given level of the hierarchy. For volume and boundary integrals in simple geometries and for target accuracies up to 10^{-8}, TT provides a direct solver offering a significant advantage in memory consumption, at the expense of a moderate increase in computation time, compared to other fast direct solvers. Additionally, we demonstrate that using the TT framework to construct preconditioners for a Krylov subspace method leads to an efficient and robust solver with a small memory footprint for boundary integral equations in complex geometries.

The main features distinguishing the TT framework from existing fast direct solver techniques are:

- *Logarithmic scaling.* Given a target accuracy, the matrix compression and inversion steps based on rank-revealing techniques as well as the storage are observed to scale as $\mathcal{O}(\log N)$. This *sublinear* scaling, remarkably, implies that the relative cost of the computation stage prior to a solve (direct or preconditioned iterative) becomes negligible as N grows.

- *Memory efficiency.* One of the main issues of current fast direct solvers is that even when they retain linear scaling, they require significant storage per degree of freedom for problems in 3D. Due to its logarithmic scaling, the TT decomposition completely sidesteps this, requiring no more than 100 MB of memory for the compression and inversion of systems with a large number of unknowns ($N \sim 10^7$ to 10^8).
- *Fast matrix-vector and matrix-matrix applies.* If both of the operands are represented in the TT format, $\mathcal{O}\left(\log N\right)$ matrix-vector and matrix-matrix apply algorithms are available. Furthermore, $\mathcal{O}\left(N \log N\right)$ matrix-vector apply algorithm exists for the multiplication of a matrix in the TT format with a non-TT-compressed vector.

1.1. *Related work*

Solution of Eq. (LS) is computed with low computational complexity either iteratively leveraging rapid evaluation algorithms such as FMM combined with Krylov subspace methods or directly using fast direct solvers. At the heart of rapid evaluation or fast direct inversion algorithms lies the observation that, due to the properties of the underlying kernel, off-diagonal matrix blocks have low numerical rank. Using a hierarchical division of the integration domain Ω—represented by a tree data structure—these algorithms exploit this low rank property in a multi-level fashion.

1.1.1. *Iterative solvers for integral equations*

During the 1980s, the development of rapid evaluation algorithms for particle simulations such as the Fast Multiple Method [Rok85, GR87, GR97], Panel Clustering [HN89], and Barnes-Hut [BH86] as well as the development of Krylov subspace methods for general matrices such as GMRES [SS86] or Bi-CGSTAB [Vor92] provided $\mathcal{O}(kN)$ or $\mathcal{O}\left(kN \log N\right)$ frameworks for solving systems of the form Eq. (LS), where k is the required number of iterations in the iterative solver and directly depend on the conditioning of the problem.

1.1.2. *Direct solvers for integral equations*

To avoid the issues related to conditioning of the system, researchers focused on the development of fast direct solvers. Direct solvers also lead to substantial speedups in situations where the same equation is solved for multiple right-hand sides.

The work on fast direct solvers for HSS matrices can be traced back to [SR94, GL96], and more recently to [MR05], in which an optimal-complexity solver for boundary integrals in the plane was presented. Extensions of this solver to surfaces in 3D were developed in [Gil+12, HG12, Gil11] and in the related works of [Cha+06a, Cha+06b, Xia+10]. As we mentioned earlier, for 3D surfaces, the complexity of inversion in these algorithms is $\mathcal{O}\left(N^{3/2}\right)$. Since this increase is due to the growth in rank of off-diagonal interactions, additional compression is required to regain optimal complexity. Corona et al. [Cor+14] achieved this for volume integral equations in 2D by an additional level of hierarchical compression of the blocks in the HSS structure. While a similar approach can be applied to surfaces in 3D, it results in a significant increase in required memory for the inverse storage as well as longer computation times.

Ho and Ying [HY15] proposed an alternate approach using additional skeletonization levels and implemented it for both 2D volume and 3D boundary integral equations using standard direct solvers for sparse matrices in an augmented system. While the structure of the algorithms suggests linear scaling, for 3D problems the observed behavior is still above linear.

In a series of papers on $\mathcal{H}$- and $\mathcal{H}^2$-matrices, Hackbusch and coworkers constructed direct solvers for FMM-type matrices. The reader is referred to [Bör+03, Beb08, Bör10] for in depth surveys. The observed complexity for integral equation operators, as reported in [Bör10, Chapter 10], is $\mathcal{O}\left(N \log^4 N\right)$ for matrix compression, $\mathcal{O}\left(N \log^3 N\right)$ for inversion, and $\mathcal{O}\left(N \log^2 N\right)$ for solve time and memory usage, with relatively large constants.

More recently, a promising inverse FMM algorithm was introduced [AD14, Cou+15], demonstrating efficient performance and $\mathcal{O}(N)$ scaling for 2D and 3D volume computations.

1.1.3. Preconditioning techniques for integral equations

The convergence rate of GMRES is mainly controlled by the distribution of the eigenvalues in the complex plane [Nac+92, Ben02] and preconditioning techniques aim to improve the rate by clustering the eigenvalues away from zero. The distinguishing factors among preconditioning schemes are the setup cost, the cost of apply, the potential for parallelization, and, needless to say, their effectiveness in reducing the number of iterations. For excellent reviews on the general preconditioning techniques the reader is referred to [Ben02, Wat15]. Here, we focus on the preconditioners tailored for linear systems arising from the discretization of integral equations.

The bulk of preconditioning techniques for integral equations can be categorized as sparse approximate inverse (SPAI) or multi-level schemes. SPAI seeks to find a preconditioner M that satisfies $\min \|\mathsf{I} - \mathsf{M}\tilde{\mathsf{A}}\|_F$ subject to some constraint on the sparsity pattern of M—typically chosen *a priori*. Here $\tilde{\mathsf{A}}$ denotes an approximate to A to make the optimization process economical. Multi-level preconditioning methods include stationary iteration techniques like multigrid and single-grid low-accuracy inverse.

Apart from SPAI and multi-level methods, some authors used incomplete factorizations as preconditioner for integral equations [Wan+07, and references therein]. But because of their potential instabilities, difficulty of parallelization, lack of algorithmic scalability, and non-monotonic performance as a function of fill-ins [Ben02] they are less popular for integral equations.

Sparse approximate inverse preconditioners (SPAI). Motivated by their low computation and application cost and their potential for parallelism, SPAIs with sparsity and approximation based on geometric adjacency (e.g. FMM tree) are the popular choice for boundary integral equations [Vav92, Nab+94, TW96, Gra+96, TW97, Cha+97, Car+03, Car+05, Wan+07].

Vavasis [Vav92] introduced multiple SPAI schemes for boundary integrals. The schemes include *mesh-neighbor* and *hierarchical clustering*. In mesh-neighbor, the sparsity pattern for M is the FMM box and the matrix $\tilde{\mathsf{A}}$ is self and near neighbors. Hierarchical clustering improves $\tilde{\mathsf{A}}$ by including first-order multipoles from far boxes.

Later, different flavors of mesh-neighbor scheme were used by other authors in similar contexts [Nab+94, TW96, Pis+06]. For certain problems, mesh-neighbor is effective in reducing the number of iterations but it does not exhibit grid independence in any case and is most effective when the far interactions are negligible, e.g. [Pis+06]. Moreover, the effectiveness of SPAI preconditioners (including diagonal and mesh-neighbor), with sparsity pattern based on FMM adjacency and fixed number of points per leaf box, deteriorates with increased tree depth [Car+03, Car+05], because the preconditioner becomes sparser and fails to capture exchange of global information.

To improve the performance of the mesh-neighbor scheme, Tausch and White [TW97] attempted to incorporate the far field by including a first order multipole expansion in both M and $\tilde{\mathsf{A}}$, which requires solving a system of size $\mathcal{O}(\log N)$ for each set of target points in a box. The resulting preconditioner is not sparse but has constant blocks for far boxes and can be applied fast.

Carpentieri et al. [Car+03] and Giraud et al. [Gir+07] observed that the SPAI preconditioners are effective in clustering most of the eigenvalues but leave a few close to the origin and removing them needs problem-dependent parameter tuning. To remedy, these authors proposed low-rank updates to the preconditioner using the eigenvectors corresponding the smallest eigenvalues of MA.

Note that, there are other sparsification schemes that can perceivably be applied to integral equation. For example, Chan et al. [Cha+97] used discrete wavelet transform in the context of SPAI as an improved sparsification tool for PDE matrices.

Multi-level methods. These schemes were introduced to address the shortcomings of SPAI. Grama et al. [Gra+96] proposed a low-order and low-accuracy iterative inner solver as a multi-level preconditioner, which was very effective in reducing the number of iterations. However, the ill-conditioning of the system caused high number of inner iterations and consequently the scheme was not time effective. Carpentieri et al. [Car+05] pursued this direction further and used a mesh-neighbor SPAI as the preconditioner for the inner solver. Gürel and Malas [GM10] used a similar approach for solving electromagnetic scattering problems.

Authors have opted for SPAI or iterative multi-level methods mainly because these methods have $\mathcal{O}(N)$ complexity in time and memory by construction. Recently, leveraging randomized algorithm and fast direct solver schemes, preconditioners with competitive complexity and much better eigenspectrum clustering[1] have been proposed [Beb05, QB13, Yin14, Cou+15].

Bebendorf [Beb05] and Benedetti et al. [Ben+08] constructed compression schemes for boundary integral equations based on $\mathcal{H}$-matrix approximation. To solve the resulting system, a low accuracy $\mathcal{H}$-LU with accuracy ε_p was used as preconditioner for the iterative solver. The complexity of $\mathcal{H}$-LU is $\mathcal{O}(N|\log \varepsilon_p|^4 \log^2 N)$. In [Ben+08], the best speedup was achieved when using preconditioner with $\varepsilon_p = 10^{-1}$ and higher accuracies did not improve the time due to preconditioner setup and apply overhead.

Quaife and Biros [QB13] proposed FMM- and multigrid-based preconditioners for the second-kind formulation of the Laplace equation in 2D. They demonstrated that even with the exact inversion in constructing the mesh-neighbor preconditioner, GMRES still requires many iterations. To construct a better preconditioner, another level of neighbors were included and inverted using ILU(10^{-3}) combined with Sherman–Morrison–Woodbury formula. The preconditioner was further improved by including a rank $\mathcal{O}(\log N)$ approximation of the residual matrix $\mathsf{A} - \mathsf{A}_{\text{near}}$, where A_{near} denotes the sparsified matrix (approximately) inverted to construct the preconditioner.

Ying [Yin14] constructed a very effective preconditioner for the iterative solution of integral equation formulation of the Lippmann–Schwinger equation. The asymptotic setup and application time of the preconditioner as well as its memory requirement are similar to those of direct solvers but the preconditioner has a smaller prefactor. The proposed algorithm proceeds by numerically constructing sparsifying operators for the system matrix. The sparsifying operators are applied in linear time (by construction) and the resulting system can be inverted efficiently because of their local nature using nested dissection or multifrontal methods. The preconditioner is then the inverse of the sparsified system.

Coulier et al. [Cou+15] presented IFMM as a fast direct solver, where the matrix is converted to an extended sparse matrix and its sparse inverse is constructed by careful compression and redirection of the fill-in blocks resulting in $\mathcal{O}(N)$ complexity for the algorithm. To achieve high-accuracy solutions in a cost-effective way, the authors proposed using a low-accuracy IFMM as a preconditioner in GMRES and demonstrated its effectiveness in reducing the number of iterations and the computation time for different distribution of points in 3D.

1.1.4. Low-rank tensor approximation of linear operators

Tensor factorizations were originally designed to tackle high-dimensional problems in areas of physics such as quantum mechanics. To be able to perform computations for these problems, the curse of dimensionality needs to be addressed. The *tensor train decomposition*, is one of the tensor representation methods developed for this purpose. Other factorization methods include the CP (CANDECOMP/PARAFAC), Tucker, and Hierarchical Tucker [Hac+05] decompositions. For a general survey of techniques and applications, see [Gra+13, Gra10].

Interestingly, it was observed that schemes of this type can be useful for low-dimensional problems, recast in the tensor form. *Quantized-TT* (QTT) algorithms reshape vectors or matrices into higher dimensional tensors (i.e. *tensorize or quantize*) and then compute a TT decomposition with low tensor rank. This was first proposed in [Ose09, Ose10] and more generally discussed in [Kho11].

As we explain in §2 and Appendix A, the approximation of these tensorized arrays by a low-rank TT decomposition corresponds to an approximation of the original systems by sums of Kronecker products and can be seen as a generalization of [VLP92].

The observation that certain kinds of structured matrices may be efficiently represented using the QTT format has been made for Toeplitz matrices [Ols+06, Ose+11], the Laplace differential operator and its inverse [KK12, Ose10], general PDEs and eigenvalue problems [Kho11], convolution operators [Hac11], and the FFT [Dol+12], amongst others. In the context of boundary integral equations, it has been used to speed up the quadrature evaluation for BEM [Kho+01].

Given a linear system whose corresponding matrix can be efficiently represented with TT, there exist several algorithms to compute a TT representation for its inverse. In this work, we use the

[1] With a simple calculation, it can be seen that any preconditioned system that satisfies $\|\mathsf{I} - \mathsf{MA}\|_1 \le \varepsilon < 1$ has bounded condition number $\kappa_1(\mathsf{MA}) \le (1+\varepsilon)/(1-\varepsilon)$. Furthermore, Gershgorin's theorem implies that the eigenvalues of MA are within the ε-disc centered at 1 [GH97].

alternating minimal energy (AMEN) and the density matrix renormalization group (DMRG) methods as proposed in [OD12, DS13a, DS13b]. Another such method is the Newton–Hotelling–Schulz algorithm [Hac+08, Ols+08].

There are also various software toolboxes for low rank tensor approximation. We make use of the Matlab TT-Toolbox [Ose12] for all tensor computations discussed in this article.

1.2. Contributions and outline

We present the tensor train decomposition as an effective and memory-efficient framework to accelerate the solution of Eq. (LS). We present this algorithm as an alternative approach for the hierarchical compression and inversion of such linear operators and discuss its relationship to commonly used hierarchical compression techniques such as FMM, HSS, and wavelets. Furthermore, we demonstrate that the TT ranks are bounded for translation-invariant systems and we argue that this behavior extends to non-translation invariant volume and boundary integrals.

In §4, we introduce strategies to generalize the TT inversion process as a product of matrix factors in the TT form to achieve faster TT inversion. In §5.1, we present results demonstrating the astounding performance of the TT (in terms of inversion costs and solve times) when applied to both translation and non-translation invariant volume integrals in 3D, for which existing direct solvers are generally not practical. Variable coefficient elliptic PDE problems in three dimensions often arise when dealing with heterogeneous media in Stokes flow, electrostatics (Poisson–Boltzmann), or low frequency scattering (Lippmann–Schwinger). The results in §5.1 suggest that the TT decomposition might be highly effective in solving these problems quickly and accurately.

In §5.2, we test a TT-based preconditioner in the iterative solution of an exterior elliptic boundary value problem (Laplace) formulated as a boundary integral equation in complex, multiply connected geometries. The problems presented in that section are close to a broad range of boundary value problems with low frequency kernels, particularly for the simulation of rigid bodies in magnetostatics, electrostatics, and particulate Stokes flow. We compare the performance of the preconditioner with a Matlab implementations of preconditioners based on multigrid and Hierarchical Interpolative Factorization (HIF) [HY15]. We show that the TT decomposition approach, while providing speedups similar to that of HIF and other fast direct solvers, has significantly smaller and better scaling in the setup costs and memory footprint.

1.3. Scope and limitations

In this work we only consider linear systems from Fredholm integral equation of the second kind. Also, we will not consider adaptive hierarchical decomposition of the domain giving rise to non-uniform trees. Therefore the distribution of sample points in volume integral equations is isotropic in our examples. All our numerical experiments are performed sequentially in Matlab. Extension of some of the TT compression and inversion algorithms to obtain good parallel scaling is an interesting research direction in its own right.

The main limitation of the current framework as a hierarchical inversion tool of linear operators is the quartic[2] dependence of the algorithmic constant on the TT ranks in the inversion algorithm, see §4.1. In our experiments, when the linear system is non-translation invariant due to the geometry (the case of surfaces in 3D), the required TT ranks for a desired accuracy grow and slow down the inversion algorithm as well as the inverse apply for accuracies larger than 10^{-4}. The high TT ranks for these cases is due to the fact that, unlike other techniques, TT compresses all interactions at each level. While this is enormously advantageous in simple geometries, certain parts of the operator may become relatively incompressible in complex geometries (e.g. so-called self or near-interactions). We believe the TT decomposition may be extended to overcome this limitation but such extension is beyond the scope of this work.

2. Mathematical and algorithmic preliminaries

In this section, we briefly describe the tensor train (TT) decomposition and its application to the approximation of *tensorized* vectors consisting of samples of functions with hierarchically subdivided domain. Matrices arising from the discretization of Eq. (IE) in this setting are interpreted as

[2]Assuming the inverse has similar TT ranks to that of A, which is the case for systems arising from integral equations.

tensorized operators acting on such tensorized vectors. In this section as well as in §3, we outline how this interpretation allows us to effectively use the TT as a tool for hierarchical compression of integral operators.

2.1. *Nomenclature*

We use different typefaces to distinguish between different mathematical objects, namely we use:

- Roman letters for continuous functions: $f(x)$, $K(x,y)$;
- calligraphic letters for multidimensional arrays and tensors: $\mathcal{f}(i_1,\dots,i_d)$, $\mathcal{K}(i_1,j_1,\dots,i_d,j_d)$;
- sans-serif for vectors and matrices: $\mathsf{f}(i)$, $\mathsf{K}(i,j)$; and
- typewriter for the TT decompositions of tensors: $\mathtt{f}$, $\mathtt{K}$.

We use Matlab's notation for general array indexing and reshaping. Given a multi-index $(i_1,\dots,i_d)$, we will denote the corresponding one-dimensional index obtained by ordering multi-indices lexicographically, by placing a bar on top and removing commas between indices: $i = \overline{i_1 i_2 \cdots i_d}$. This mapping from multi-indices to one-dimensional index defines a conversion of a multidimensional array to a vector which we denote $\mathsf{b} = \mathrm{vec}(\mathcal{b})$, with $\mathsf{b}(\overline{i_1 i_2 \cdots i_d}) = \mathcal{b}(i_1,i_2,\dots,i_d)$.

2.2. *Tensor train decomposition*

The tensor train decomposition is a highly effective method for compact low-rank approximation of tensors [OT10]. For a d-dimensional tensor $\mathcal{A}(i_1,i_2,\dots,i_d), i_k \le n_k$, sampled at $N = \prod_{k=1}^d n_k$ points, the TT decomposition has the form

$$\mathtt{A}(i_1,i_2,\dots,i_d) := \sum_{\alpha_1,\dots,\alpha_{d-1}} \mathcal{G}_1(i_1,\alpha_1)\mathcal{G}_2(\alpha_1,i_2,\alpha_2)\dots\mathcal{G}_d(\alpha_{d-1},i_d), \tag{2.1}$$

where, each two- or three-dimensional $\mathcal{G}_k$ is known as a tensor core. The ranges of auxiliary indices $\alpha_k = 1,\dots,r_k$ determine the number of terms in the decomposition. We refer to r_k as the kth TT-rank, the analog of the rank in a low-rank factorization of a matrix. For algorithmic purposes, it is often useful to introduce dummy indices α_0 and α_d, and let the corresponding TT ranks $r_0 = r_d = 1$; in this way, we can view all cores as three-dimensional tensors.

For a tensor of dimension d, the kth *unfolding matrix* is defined as

$$\mathsf{A}^k(p_k,q_k) = \mathsf{A}^k(\overline{i_1 i_2 \cdots i_k}, \overline{i_{k+1}\cdots i_d}) = \mathcal{A}(i_1,i_2,\cdots,i_d) \quad \text{for} \quad k=1,\dots,d, \tag{2.2}$$

where $p_k = \overline{i_1 \cdots i_k}$ and $q_k = \overline{i_{k+1}\cdots i_d}$ are two flattened indices. Using Matlab's notation

$$\mathsf{A}^k = \mathtt{reshape}\left(\mathcal{A}, \prod_{\ell=1}^k n_\ell, \prod_{\ell=k+1}^d n_\ell\right). \tag{2.3}$$

The ranks of the TT decomposition are related to the ranks of unfolding matrices. A low-rank TT decomposition can be obtained by a sequence of low-rank approximations to A^k (e.g. using a sequence of truncated SVDs). More generally, given a low-rank approximation routine, a generic algorithm to compute the TT decomposition proceeds as in Algorithm 1.

Using the notation given in Algorithm 1, a low-rank decomposition of the unfolding matrix $\mathsf{A}^k \approx \mathsf{U}^k\mathsf{V}^k$ may be obtained by multiplying U_k by the first $k-1$ cores $\mathcal{G}_k$ already computed and setting $\mathsf{V}^k = \mathsf{V}_k$. Oseledets and Tyrtyshnikov [OT10] show that in SVD-based TT compression, for any tensor $\mathcal{A}$, when the low-rank decomposition error for the unfolding matrices is optimal for rank r_k,

$$\varepsilon_k = \left\|\mathsf{A}^k - \mathsf{U}^k\mathsf{V}^k\right\|_F = \min_{\mathrm{rank}(\mathsf{B})\le r_k} \left\|\mathsf{A}^k - \mathsf{B}\right\|_F, \tag{2.4}$$

the corresponding TT approximation $\mathtt{A}$ satisfies

$$\|\mathcal{A} - \mathtt{A}\|_F^2 \le \sum_{k=1}^{d-1} \varepsilon_k^2. \tag{2.5}$$

Algorithm 1 TT DECOMPOSITION.

```
Require: Tensor $\mathcal{A}$, and target accuracy $\varepsilon$
 1: $\mathsf{M}_1 = \mathsf{A}^1$                                  // First unfolding matrix
 2: $r_0 = 1$
 3: for $k = 1$ to $d-1$ do
 4:     $[\mathsf{U}_k, \mathsf{V}_k] = \texttt{lowrank\_approximation}(\mathsf{M}_k, \varepsilon)$
 5:     $r_k = \texttt{size}(\mathsf{U}_k, 2)$                     // $k$th TT rank
 6:     $\mathcal{G}_k = \texttt{reshape}(\mathsf{U}_k, [r_{k-1}, n_k, r_k])$     // $k$th TT core
 7:     $\mathsf{M}_{k+1} = \texttt{reshape}\left(\mathsf{V}_k, [r_k n_{k+1}, \prod_{\ell=k+2}^{d} n_\ell]\right)$  // $\mathsf{M}_{k+1}$ corresponds to the $(k+1)$th
                                                                   unfolding matrix of $\mathcal{A}$
 8: end for
 9: $\mathcal{G}_d = \texttt{reshape}(\mathsf{M}_d, [r_{d-1}, n_d, 1])$          // Set last core to the right factor in the
                                                                   low rank decomposition
10: return A
```

Given a prescribed upper bounds r_k for the TT ranks, there exists a unique optimal approximation in TT format $\mathtt{A}_{\text{optimal}}$ and the approximation $\mathtt{A}$ obtained by the SVD-based TT algorithm is quasi-optimal, i.e.

$$\|\mathcal{A} - \mathtt{A}\|_F \leq \sqrt{d-1}\left\|\mathcal{A} - \mathtt{A}_{\text{optimal}}\right\|_F . \tag{2.6}$$

The direct application of Algorithm 1, where the ranks r_k are obtained using a rank-revealing decomposition still leads to relatively high computational cost, $\mathcal{O}(N)$ or higher, which is exponential in the dimension d. The key advantage of the TT approximation is the multi-pass AMEN (Alternating Minimal Energy) Cross algorithm of [OT10], in which a low-TT-rank approximation is initially computed with fixed ranks and is improved upon by a series of passes through all cores.

The analysis and experiments in [DS13a, DS13b] show that both the approximation and the inverse compression AMEN algorithms exhibit monotonic, linear convergence. We observe this throughout all experiments presented in this work as well. The resulting algorithm typically scales linearly with dimension.

Quantized-TT: rank and mode size implications. While the algorithms and the analysis for the TT decomposition are generic, our work focuses on their application to function and kernel sampled in two or three dimensions by casting them as higher dimensional tensors. This type of TT decomposition is usually referred to as *Quantized*-TT or QTT. In the process of *tensorization*, QTT splits each dimension until each tensor mode n_k $(k = 1, \ldots, d)$ is very small in size. For instance, a one-dimensional vector of size 2^d is converted to a d-dimensional tensor with each mode of size 2 implying that $d \approx \log N$.

Computational Complexity and Memory Requirements. Because AMEN Cross and related TT rank revealing approaches proceed by enriching low-TT-rank approximations, all computations are performed on matrices of size $r_{k-1} n_k \times r_k$ or less. Performing an SVD on such matrices is $\mathcal{O}\left(r_{k-1}^2 n_k^2 r_k + r_k^3\right)$ [GVL12]. Other low-rank approximations such as the ID [Che+05] have similar complexity. As a consequence, computational complexity for this algorithm is bounded by $\mathcal{O}\left(r^3 d\right)$ or equivalently $\mathcal{O}\left(r^3 \log N\right)$, where $r = \max(r_k)$ is the maximal TT-rank that may be a function of sample size N and accuracy ε.

In some cases, as we will discuss in more detail below, the maximal TT-rank r can be bounded as a function of N. For differential and integral operators with non-oscillatory kernels, as well as their inverses, r typically stays constant or grows logarithmically with N [KK12, Ose10, Ose+11]. If this is the case, the overall complexity of computations is *sublinear* in N.

2.3. *Applying the QTT decomposition to function samples*

Our goal is to use the QTT decomposition to compress matrices in Eq. (LS). In this section, we cast QTT as an algorithm operating on a hierarchical partition of the data, providing a better understanding of its performance. In the next section, we formally prove that such decompositions indeed have low TT-ranks.

Let $f : \Omega \to \mathbb{R}$ be a function on a compact subset Ω of $\mathbb{R}^D$. We consider hierarchical partitions of Ω into disjoint sets; at each level of the partition, each subset is split into the same number of subsets n.

This partition can be viewed as a tree $\mathscr{T}$ with domains at different levels as nodes, and edges connecting a domain to all subdomains it is split into. We number levels from 2 to d, starting with the *finest* level—we use the first index for indexing samples in each domain. The domains at the finest level can be indexed with a multi-index $(i_2, \ldots i_d)$ where i_ℓ indicates which of n branches was taken at level ℓ. For each leaf domain we pick n_1 sample points $x_{i_1,i_2\ldots,i_d}$, adding an additional index $i_1 \leq n_1$ to the multi-index. Thus, we defined a tensor

$$f_{\mathscr{T}}(i_1, i_2, \ldots, i_d) = f(x_{i_1,i_2,\ldots,i_d}), \tag{2.7}$$

with each index corresponding to a level of the tree. Flattening this tensor yields a vector of samples $\mathsf{f} = \text{vec}(f_{\mathscr{T}})$.

If we compute the TT decomposition given in Eq. (2.1) for this d-tensor, we obtain an approximation to f as a sum of the terms of the form

$$\mathcal{G}_1(\alpha_0, i_1, \alpha_1)\mathcal{G}_2(\alpha_1, i_2, \alpha_2)\ldots\mathcal{G}_d(\alpha_{d-1}, i_d, \alpha_d), \tag{2.8}$$

where each factor $\mathcal{G}_\ell$ corresponds to a level of the hierarchy in $\mathscr{T}$.

QTT decomposition as a hierarchical adaptive filter. To gain further intuition about the compression of function samples using the QTT decomposition, it is instructive to consider how the decomposition operates step by step in Algorithm 1. The matrix compression is illustrated in Fig. 1 and is discussed later in this section, nevertheless the structure of decomposition shown in the lower part of the figure is identical for both vectors and matrices. At the first step, column c_j of the matrix M_1 consists of n_1 samples of the finest-level domain indexed by i_1. The low-rank factorization of this matrix with rank

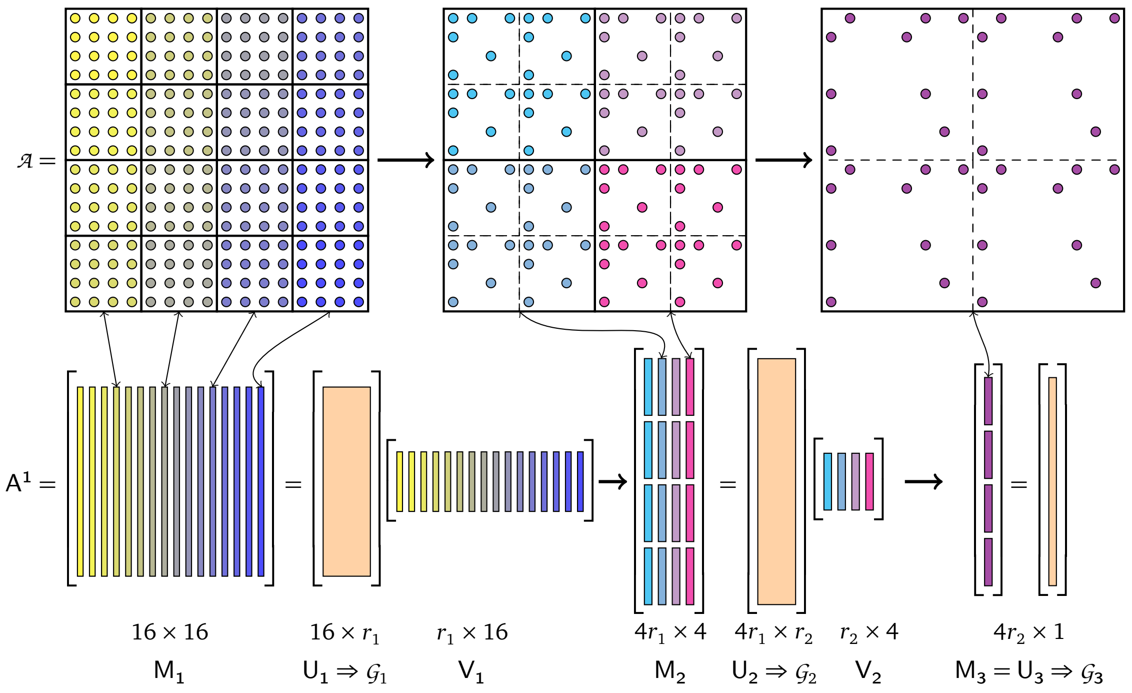

Figure 1: Steps for the TT decomposition given in Algorithm 1 for a matrix with $d = 3$. *For clarity, the colors of the blocks in the unfolding matrices match that of the blocks in the tree. The main steps of the algorithm are (i) computing a low rank decomposition $\mathsf{U}_k\mathsf{V}_k$ for the corresponding unfolding matrix M_k; (ii) taking U_k (in orange) as the kth TT core; and (iii) interpreting the right factor V_k as a matrix in level $k+1$ to form M_{k+1}. If the low rank decomposition used is interpolatory (as implied in the figure by the uniform subsampling of the tree nodes), this process corresponds to finding a hierarchical basis of matrix block entries.*

r_1 can be viewed as finding a basis of $r_1 \leq n_1$ vectors forming matrix V_1, such that all vectors c_j can be approximated by linear combinations of this set of row vectors with a given accuracy. If the decomposition is interpolatory, this corresponds to picking a set of rows in M_1 (that is, subsampling each leaf domain in the same way), such that remaining samples can be interpolated from these using the same $(n_1 - r) \times r$ interpolation operator.

At the next step, the subvectors for each tree node at the coarser level 2, are arranged into vectors, which form columns of the new matrix M_2, and compressed in the same manner. Thus, if interpolatory decomposition is used, each step of the process can be viewed as finding the optimal subsampling of the previous level and an interpolation matrix.

Computing a QTT decomposition of a signal can be viewed as a process of computing a set of *adaptive* filters, depending on the signal the QTT is applied to. These filters can be viewed as similar to the analysis low-pass filters in a wavelet transform. We discuss this analogy briefly, as it provides intuition why QTT is effective for integral operator matrices [Bey+91]. More details on QTT-based wavelets can be found in [OT11].

The simplest wavelet transform for a vector s_0 of length $N = 2^d$, the Haar transform, is determined by two filters, $\mathsf{L} = [\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}]$ and $\mathsf{H} = [\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}]$. These filters are applied to generate a sequence of smooth and detail vectors s_k and d_k, of lengths 2^{d-k}, $k = 0, \ldots, d$. A single step of the wavelet transform at a level ℓ amounts to splitting the signal into subvectors of length 2, arranging them into a $2 \times 2^{d-\ell-1}$ matrix, and multiplying this matrix on the left by row vectors L and H respectively, to obtain the vector of length $2^{d-\ell-1}$ of smooth coefficients $\mathsf{s}_{\ell+1}$ to be passed to the next level, and a vector of detail coefficients $\mathsf{d}_{\ell+1}$ of the same length.

The reconstruction proceeds using two synthesis filters $\widetilde{\mathsf{L}} = [\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}]$ and $\widetilde{\mathsf{H}} = [\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}]$, satisfying $\mathsf{L}^T\widetilde{\mathsf{L}} + \mathsf{H}^T\widetilde{\mathsf{H}} = \mathsf{I}$, which for the Haar wavelets coincide with L and H, respectively. Consequently, $\mathsf{s}_\ell = \widetilde{\mathsf{L}}\mathsf{s}_{\ell+1} + \widetilde{\mathsf{H}}\mathsf{d}_{\ell+1}$. We observe that if all detail coefficients are discarded, then the formula $\mathsf{s}_\ell = \widetilde{\mathsf{L}}\mathsf{s}_{\ell+1}$ is similar to the formula $\mathsf{M}_k = \mathsf{U}_k\mathsf{V}_k$, with $n_k = 2$, $r_k = 1$, and U_k playing the role of the low-pass reconstruction filter $\widetilde{\mathsf{L}}$ and V_k corresponding to smooth coefficients.

Thus, one can view the QTT decomposition as structurally similar to a discrete wavelet transform, with the following differences: (i) the reconstruction low-pass filters are computed adaptively on each level to minimize reconstruction error, (ii) all detail coefficients are discarded and effectively all information about the signal is encoded in the computed filters, (iii) the subsampling factor at each level is determined adaptively, instead of being fixed at 2. While the standard wavelet theory clearly does not apply in this setting, and separate analysis is needed (see §3), the wavelet analogy provides a connection to the previously proposed method [Bey+91] for compressing integral equation matrices.

Remark 2.1. *The inverse of flattening (i.e., reshaping the one-dimensional vector to a d-dimensional tensor) applied to a vector of samples of size $n_1 n^{d-1}$, yields a tensor, so formally we can start with a suitably-sized unorganized sequence of samples of a function and tensorize it. Implicitly, this defines a hierarchical partition of this set of samples; reordering the input vector changes the partition, and gives a different tensor, with a one-to-one correspondence between different partitions and permutations of the elements of the input vector.*

The ranks of TT factorizations of the resulting tensors strongly depend on the choice of permutation. A bad choice (e.g. with distant points grouped in leaf nodes) may yield large TT ranks. For instance, for $D > 1$, sampling $f\colon [0,1]^D \to \mathbb{R}$ on a regular grid and indexing according to the dictionary order will yield suboptimal ranks. To obtain good compression, the ordering needs to represent a geometrically meaningful partition.

2.4. *TT for samples of an integral kernel*

Consider the matrix A in Eq. (LS), with entries sampled from the integral kernel $K(x,y)$

$$\mathsf{A}(i,j) = K(x_i, y_j), \quad x_i, y_j \in \Gamma \subset \mathbb{R}^D \quad (D = 1, 2, \text{ or } 3). \tag{2.9}$$

for a set of M targets $\{x_i\}$ and N sources $\{y_j\}$. This is a particular instance of the setting described above, with $f = K$ and $\Omega = \Gamma \times \Gamma$.

Let the target and source trees on Γ be denoted by $\mathcal{T}_{\text{trg}}$ and $\mathcal{T}_{\text{src}}$. Both of the trees are of depth d and the number of children at all non-leaf levels of the trees is m and n respectively. There are respectively m_1 and n_1 points in the target and source tree leaf nodes and the total numbers of points are $M = m_1 m^{d-1}$ and $N = n_1 n^{d-1}$. Also, let $\mathcal{T}$ denote the product tree $\mathcal{T}_{\text{trg}} \times \mathcal{T}_{\text{src}}$, whose nodes at each level consist of pairs of source and target nodes.

Current fast solvers exploit the fact that matrix blocks representing interactions between well-separated target and source nodes at a given level are of low numerical rank. One can interpret $\mathcal{T}$ as a hierarchy of matrix-blocks, and in §3 we show that the TT structure for this hierarchy can be inferred from the standard hierarchical low rank structure.

At a given level ℓ (recall that levels are numbered starting at the finest) each node corresponds to a matrix block with row indices corresponding to a node of $\mathcal{T}_{\text{trg}}$ and column indices of a node in $\mathcal{T}_{\text{src}}$, indexed by integer coordinate pairs (i_k, j_k) with $k \le \ell$. Equivalently, we can consider block integer coordinates $b_k \in \{1, \cdots, m_k n_k\}$ for $\mathcal{T}$ such that $b_k = \overline{i_k j_k}$.

We can then apply the TT decomposition to the corresponding tensorized form of A, $\mathcal{A}_{\mathcal{T}}$, a d-dimensional tensor with entries defined as

$$\mathcal{A}_{\mathcal{T}}(b_1, b_2, \ldots, b_d) = \mathcal{A}_{\mathcal{T}}(\overline{i_1 j_1}, \overline{i_2 j_2}, \ldots, \overline{i_d j_d}) = \mathsf{A}(\overline{i_1 i_2 \cdots i_d}, \overline{j_1 j_2 \cdots j_d}) = K(x_i, y_j), \tag{2.10}$$

and obtain a TT factorization $\mathtt{A}$. Each core of $\mathtt{A}$, $\mathcal{G}_k(\alpha_{k-1}, \overline{i_k j_k}, \alpha_k)$ depends only on the pair of source and target tree indices at the corresponding level of the hierarchy. For matrix arithmetic purposes, such as the matrix-vector product algorithms in §4.2, the cores are sometimes reshaped as $m_k \times n_k$ matrices parametrized by α_{k-1} and α_k.

In Fig. 1, we illustrate the TT decomposition algorithm applied to a matrix A for binary source and target trees ($n = m = 2$) with depth $d = 3$ and four points in the leaf nodes $n_1 = m_1 = 4$, implying $N = M = 16$. In this example, the tree $\mathcal{T}$ is equivalent to a matrix-block quadtree.

3. Tensor-Train rank behavior of integral equation operators

In this section, we present an analysis of the TT ranks of QTT-compressed form of matrices obtained from integral kernels. As indicated in §1, integral equation formulation of PDEs often involves an integral kernel $K(r)$ with a singularity at $r = 0$, and are usually of the form

$$a\sigma(x) + \int_\Omega b(x) K(||x - y||) c(y) \sigma(y)\, d\Omega_y = f(x), \tag{IE}$$

where Ω is a domain in $\mathbb{R}^D$ for $D = 1, 2, 3$ (either a boundary or a volume). After we discretize the integral (e.g. with Nyström discretization using the appropriate quadrature), this becomes a linear system of the form

$$\sum_{j=1}^{N} w_j \Big[a\delta(x_i - y_j) + b(x_i) K(||x_i - y_j||) c(y_j) \Big] \sigma(y_j) = f(x_i), \tag{3.1}$$

where w_j denotes the appropriate quadrature weight. We can write the system in matrix form as

$$\mathsf{A}\sigma = \mathsf{f}, \tag{LS}$$

where $\mathsf{A} := a\mathsf{I} + \mathsf{BKC}$, in which B and C are diagonal matrices with entries $b(x_i)$ and $c(y_j)$, respectively.

3.1. Translation invariant kernels

If $b \equiv c \equiv 1$, then the kernel becomes translation invariant in $\mathbb{R}^D$. The extent the corresponding matrix inherits this property depends on the geometry of Γ and on how targets x_i and sources y_j are sampled. One example where we can exploit this to great advantage is when Ω is a box in $\mathbb{R}^D$ sampled on a uniform grid. To understand the rank behavior of the TT decomposition, we explore the relationship between the hierarchical low-rank structure exploited by the FMM, $\mathcal{H}$, or HSS matrices and the matrix-block low rank structure exploited by the TT decomposition.

We begin by recalling a common classification for pairs of boxes on $\mathcal{T}_{\text{trg}} \times \mathcal{T}_{\text{src}}$.

Definition 3.1. *A pair of boxes $(B_i, B_j) \in \mathcal{T}_{\text{trg}} \times \mathcal{T}_{\text{src}}$ is said to be well-separated if*

$$\text{dist}(B_i, B_j) \ge \max(\text{diam}(B_i), \text{diam}(B_j)). \tag{3.2}$$

Definition 3.2. *We define the far field $\mathscr{F}_\ell(B_i)$ of target box B_i as the subset of boxes in $\mathscr{T}_{\text{src}}$ at level ℓ such that (B_i, B_j) is well-separated. Similarly, we define its near field $\mathscr{N}_\ell(B_i)$ as the subset which is not well-separated.*

In a uniformly refined tree, $\left|\mathscr{N}_\ell(B_i)\right| \leq 3^D$ for all $B_i \in \mathscr{T}_{\text{trg}}$. For the case of adaptive trees, it is common practice to impose a level-restricted refinement, making the number of target boxes in the near field bounded even if neighbors at multiple levels are considered.

For all $B_j \in \mathscr{F}_\ell(B_i)$, given a desired precision ε, standard multipole estimates [GR87] show that for a broad class of integral kernels K the matrix block $\mathsf{A}_{i,j}$ corresponding to the evaluation of Eq. (3.1) with $(x_i, y_j) \in B_i \times B_j$ has numerical low rank $k_{i,j}$. This rank is bounded by a constant k_ε depending only on the kernel and ε.

In fact, for kernels that arise in integral formulations of elliptic PDEs, multipole expansions or Green's identities may be used to prove a stronger result: that the rank of a block with entries evaluated at $(x, y) \in S \times T$ is bounded by k_ε for any well-separated sets S and T. This implies that interactions between a box B and any subset of its far field have bounded rank.

Definition 3.3. *For a matrix A given in Eq. (LS) generated by the partitions of Ω corresponding to $\mathscr{T}_{\text{src}}$ and $\mathscr{T}_{\text{trg}}$, we say A is FMM-compressible if for a given accuracy ε, any matrix sub-block $A_{S,T}$ corresponding to evaluation at $(x, y) \in S \times T$ for well-separated sets S and T is such that* $\text{rank}_\varepsilon(\mathsf{A}_{S,T}) \leq k_\varepsilon$.

Theorem 3.4. *Let $K(r)$ be a translation-invariant kernel in Eq. (IE), for a box $\Omega \subset \mathbb{R}^D$. Let A be the corresponding matrix in Eq. (LS), sampled on a translation invariant grid. If A is FMM-compressible, then the tensorized form $\mathcal{A}_{\mathscr{T}}$ for the product tree $\mathscr{T} = \mathscr{T}_{\text{src}} \times \mathscr{T}_{\text{trg}}$ has bounded TT ranks*

$$r = \max(r_k) \leq k_\varepsilon^2 + 2D - 1. \tag{3.3}$$

Proof. In order to prove the theorem, we need to find an upper bound for the ε-rank of the unfolding matrices. This bound is then valid for the corresponding TT rank, as indicated in §2. Lets consider the unfolding matrix $\mathsf{A}^\ell_{\mathscr{T}}$ corresponding to interactions of boxes in level ℓ of the tree $\mathscr{T}$. As it was mentioned in §2.4 and Remark 2.1, columns of $\mathsf{A}^\ell_{\mathscr{T}}$ are vectorized matrix blocks $\{\text{vec}(\mathsf{A}_{i,j})\}$ comprising all near and far interactions between boxes at that level. We can permute the columns to place those corresponding to the near-field interactions first

$$\mathsf{A}^\ell_{\mathscr{T}} = \left[\mathsf{A}^{\text{near}}_{\mathscr{T}} \; \mathsf{A}^{\text{far}}_{\mathscr{T}}\right]. \tag{3.4}$$

The key observation is that, because boxes at a given level are translations of a reference box, due to *translation invariance*, if the sampling is the same across boxes, we need only consider inward and outward interactions for one box per level. This is also why fast solvers based on hierarchical structures such as HSS need only compute one set of matrices per level.

For near interactions, that means only one set of interactions between a representative box and some of its neighbors (including itself) is needed. This implies $\text{rank}(\mathsf{A}^{\text{near}}_{\mathscr{T}}) \leq 2\left|\mathscr{N}_\ell[B]\right| - 1$, since all columns in this subset are identical to the outgoing or incoming interactions of that representative box. For the case of a uniformly refined tree $\left|\mathscr{N}_\ell[B]\right| \leq 3^D$, and due to symmetries, $\text{rank}(\mathsf{A}^{\text{near}}_{\mathscr{T}}) \leq 2D - 1$.

For columns in $\mathsf{A}^{\text{far}}_{\mathscr{T}}$, we use an interpolative decomposition (ID) to compute a low rank approximation for the matrix blocks

$$\mathsf{A}_{i,j} \approx \mathsf{L}_i \mathsf{M}_{i,j} \mathsf{R}_j, \tag{3.5}$$

where L_i and R_j are interpolation matrices and $\mathsf{M}_{i,j}$ is a sub-block of $\mathsf{A}_{i,j}$ corresponding to skeleton rows and columns [Che+05, Mar+07, Tyr00]. Given that $|B_i| = \prod_{i \leq \ell} n_i, |B_j| = \prod_{j \leq \ell} m_j$, L_i is of size $|B_i| \times k_{i,j}$ and R_k is of size $k_{i,j} \times |B_j|$ and $\mathsf{M}_{i,j}$ of size $k_{i,j} \times k_{i,j}$. Substituting Eq. (3.5) for each vectorized column $A_{i,j}$, we have

$$\text{vec}(\mathsf{A}_{i,j}) = \text{vec}(\mathsf{L}_i \mathsf{M}_{i,j} \mathsf{R}_j) = (\mathsf{R}_j^T \otimes \mathsf{L}_i)\text{vec}(\mathsf{M}_{i,j}). \tag{3.6}$$

Due to translation invariance, we may construct a matrix of all far-field interactions with a model target box B, and apply an ID to obtain an interpolation matrix L and corresponding row skeleton set which are valid for all boxes at level ℓ.[3] An analogous computation for a model source box may

[3]Equivalent densities may be used to accelerate this computation, as it is done in [Cor+14] for HSS matrices.

be used to obtain R and the corresponding column skeleton set. By assumption, the corresponding ranks will be bounded by k_ε. As a consequence, the pre-factor on the vectorized interpolation formula in Eq. (3.6) becomes the same for all far-interaction blocks. If we define $\mathsf{M}^{\text{far}}_{\mathcal{T}}$ as the matrix with columns $\text{vec}(\mathsf{M}_{i,j})$, this means

$$\mathsf{A}^{\text{far}}_{\mathcal{T}} = (\mathsf{R}^T \otimes \mathsf{L})\mathsf{M}^{\text{far}}_{\mathcal{T}}. \tag{3.7}$$

This gives us a low rank decomposition of $\mathsf{A}^{\text{far}}_{\mathcal{T}}$, with rank at most k_ε^2.

Hence, the rank of the unfolding matrix $A^\ell_{\mathcal{T}}$ is bounded by a constant $k_\varepsilon^2 + 2D - 1$. □

Remark 3.5. *For an interval ($D = 1$), we can reduce this to $2k_\varepsilon + 2D - 2$ arguing that the matrix with blocks $M_{i,j}$ (with near-field interactions zeroed out) is block-Toeplitz. Experiments with unfolding matrices for $D = 2, 3$ and observation of the resulting TT ranks and column basis using an ID (as schematically depicted in Fig. 1) suggest a similar bound (with linear dependence on k_ε) is true for $D > 1$.*

3.2. *Non-translation invariant kernels*

The (discrete) integral operator can be non-translation invariant either due to nontrivial $b(x)$ or $c(y)$, e.g. in Lippmann–Schwinger, Poisson–Boltzmann or other variable coefficient elliptic equations, or due to the geometry of the discretization (in the sense that $\mathcal{T}_{\text{src}}$ and $\mathcal{T}_{\text{trg}}$ correspond to non-translation invariant partitions). For the former case, we can also conclude TT ranks are bounded as a direct corollary of Theorem 3.4.

Corollary 3.6. *Let $K(r)$ be a translation invariant kernel in Eq. (IE), for a box $\Omega \subset \mathbb{R}^D$. Let $\mathsf{A} := a\mathsf{I} + \mathsf{BKC}$ be the corresponding matrix, sampled on a translation invariant grid. Let K be FMM-compressible, with TT ranks bounded by the constant $r^{\mathsf{K}}_\varepsilon$. Further, let $b(x), c(y)$ both admit compact TT representations with rank bounds $r^{\mathsf{b}}_\varepsilon, r^{\mathsf{c}}_\varepsilon$. Then the tensorized form $\mathcal{A}_{\mathcal{T}}$ for the product tree $\mathcal{T} = \mathcal{T}_{\text{src}} \times \mathcal{T}_{\text{trg}}$ has bounded TT ranks*

$$r^{\mathsf{A}}_\varepsilon = \max(r_k) \leq r^{\mathsf{b}}_\varepsilon r^{\mathsf{K}}_\varepsilon r^{\mathsf{c}}_\varepsilon + 1. \tag{3.8}$$

Proof. By Theorem 3.4, we know that K being FMM-compressible ensures the corresponding tensorized version on $\mathcal{T}$ has bounded TT ranks. Regarding $b(x)$ and $c(y)$, by assumption, they are smooth, non-oscillatory functions, and therefore their TT ranks are expected be small (see [KS15, Big+14, Hac+05] for rank estimates of functions).

Further, we can readily observe that, the diagonal matrices B and C have TT structure essentially the same as the TT structure of b and c. Looking at the first unfolding matrix of the tensorized $\mathcal{B}$, if we ignore columns with all zero elements,

$$\mathsf{B}^1_{\mathcal{T}}(\overline{i_1 j_1}, \overline{i_2 j_2 \dots i_d j_d}) = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} b(x_1) & b(x_3) & \dots & b(x_{N-1}) \\ b(x_2) & b(x_4) & \dots & b(x_N) \end{bmatrix}, \tag{3.9}$$

where the second factor on the right hand side is the first unfolding matrix of b. Hence, the TT ranks of both decompositions are the same.

Following the structure of the TT matrix-matrix product algorithm, which is essentially identical to that of the TT compressed matrix-vector product in §4.2, it is apparent that the ranks of the tensorized form of BKC, before any rounding on the TT cores is performed, is equal to the product of the corresponding ranks (as the new auxiliary indices are a concatenation of those of each factor).

We can thus bound each the ranks of our matrix by the product of the corresponding ranks of B, K and C. Adding an identity matrix $a\mathsf{I}$, which is of rank 1 in TT form (a Kronecker product of identities), adds at most 1 to this bound. Taking a maximum over all TT ranks, we obtain the desired bound

$$r^{\mathsf{A}}_\varepsilon = \max(r_k) \leq r^{\mathsf{b}}_\varepsilon r^{\mathsf{K}}_\varepsilon r^{\mathsf{c}}_\varepsilon + 1. \tag{3.10}$$

□

Non-translation invariance due to complex geometry. The general and harder to analyze case is when the operator is non-translation invariant due to Γ. This is often because the geometry of Γ makes it impossible to partition it into a spatial hierarchy of translates. This is indeed the general case for linear systems coming from boundary integral equations defined on curves in $\mathbb{R}^2$ or surfaces in $\mathbb{R}^3$.

From our experiments with boundary integral operators defined on smooth surfaces in $\mathbb{R}^3$, which we present in §5.2, we observe that TT ranks are still bounded or slowly growing with problem size N, although they are generally much larger than the corresponding ranks for the volume in $\mathbb{R}^3$. Further, we have observed that compressing only far-field interactions at a given level leads to substantially reduced TT ranks.

Although further analysis and experimentation are needed, we conjecture that for kernels which are translation invariant in the volume, the rank of far-field interactions will remain bounded. Near and self interactions are evidently dependent on surface complexity, although we expect that if the discretization is refined enough for a smooth surface, a relatively small basis of columns of $\mathcal{A}_{\mathcal{T}}^{\text{near}}$ may still be found.

These findings suggest that a more general factorization based on TT compression of only far field interactions may be useful, especially for operators defined on complex geometries.

4. TT-compressed approximate inverses

In this section we describe how an approximate inverse of a matrix in the TT format can be constructed, and how a matrix in the TT format can be applied efficiently to an arbitrary vector. These are the two essential components of a direct solver based on TT factorization (respectively, corresponding to matrix factorization and solve in sparse solvers) and of a TT-based preconditioner, when the inverse is computed with low accuracy and applied at each iteration of an iterative solver.

4.1. *TT matrix inversion*

TT inversion methods provide efficient ways to compute a TT decomposition of A^{-1} given the TT decomposition of A. In these algorithms, the matrix equation for the TT cores of the inverse is solved iteratively. Starting from an initial guess for the inverse in the TT form, these algorithms proceed by iteratively cycling through the cores (freezing other cores) and solving a linear system to update the current core of A^{-1}. Since the TT ranks of the inverse are not known a priori, what distinguishes the inversion algorithms are the strategies employed to increase the ranks of the cores as needed to accelerate convergence to an accurate representation of the inverse. Further details about these algorithms can be found in Appendix B, Appendix C. and [OD12, DS13a, DS13b].

We note that, except for the case where the maximum TT rank r_{A} for matrix A is 1, there are no guarantees that the TT ranks of A^{-1} will be small even if TT ranks of A are small. In [Tyr10], for $r_{\mathsf{A}} = 2$, it is proven that $r_{\mathsf{X}} \leq \sqrt{N}$, and this inequality is shown to be sharp. Nonetheless, for the integral kernels considered in this work, extensive experimental evidence shows that they are within a small factor of each other. This observed correlation in rank behavior is also evident in the experimental scaling of the inverse compression and inverse apply, as shown in §5.

Inverting a matrix in the TT form. Given a representation of a matrix A in the TT form, we want to produce a TT representation for the inverse A^{-1}. The approximate inversion schemes have a common starting point where they consider the matrix equations $\mathsf{AX} = \mathsf{I}_{\mathsf{N}}$ or $\mathsf{AX}+\mathsf{XA} = 2\mathsf{I}_{\mathsf{N}}$ and extract a TT decomposition for X. If we vectorize $\mathsf{AX} = \mathsf{I}_{\mathsf{N}}$ using the identity for products of matrices, $\mathrm{vec}(\mathsf{ABC}) = (\mathsf{C}^{\mathsf{T}} \otimes \mathsf{A})\mathrm{vec}(\mathsf{B})$, we obtain

$$(\mathsf{I}_N \otimes \mathsf{A})\mathrm{vec}(\mathsf{X}) = \mathrm{vec}(\mathsf{I}_N). \tag{4.1}$$

In [OD12, DS13a, DS13b], DMRG (Density Matrix Renormalization Group) and AMEN (Alternating Minimal Energy) minimization methods are proposed.

Given an initial set of cores $\{\mathcal{W}_k\}_{k=1}^d$ and corresponding TT ranks $\{\rho_k\}_{k=1}^d$ for X, fixing all but $\mathcal{W}_k$, turns Eq. (4.1) into a reduced linear system, with a matrix of size $n_k\rho_{k-1}\rho_k \times n_k\rho_k\rho_{k-1}$. All methods cited above solve each one of these local systems in their descent step towards an accurate TT decomposition for X.

Preconditioning local systems. The condition number of the original linear system directly affects the performance of iterative solvers used to solve the local systems outlined above. When the original linear system is not well-conditioned, e.g. due to complex geometry [QB13], it is highly desirable to precondition them so that performance of the inversion algorithm does not degrade.

The matrices in each local system have tensor structure—as described in [DS13a, DS13b], and Appendix C—that can be exploited to construct preconditioners for each of these local systems. However, we found the block-Jacobi preconditioners available in the TT Toolbox to be ineffective for matrices obtained from integral equation formulations. More general preconditioners based on low-rank tensor approximation were too expensive to compute on the fly.

Instead, we solve for A^{-1} in a different form, using an auxiliary right preconditioner (i.e., bootstrap our approximate inverse computation with an initial, even coarser, inverse) M. Specifically, we apply the TT inversion algorithm to the matrix equation

$$\mathrm{vec}(\mathsf{AMY}) = (\mathsf{I}_N \otimes \mathsf{AM})\mathrm{vec}(\mathsf{Y}) = \mathrm{vec}(\mathsf{I}_N),$$

which implies $\mathsf{A}^{-1} = \mathsf{MY}$. The matrix M is computed in TT form, which allows for efficient matrix-matrix products. This preconditioner may be the inverse of a block-diagonal or block-sparse version of A (such as the sparsifying preconditioners in [QB13, Yin14]) or a low accuracy hierarchical factorization of A^{-1}.

In all experiments of §5.2, we compress a block-diagonal system for spheres and use it in the TT form to precondition the subsystems arising from the boundary integral equations defined on lattices of surfaces with spherical topology. This provides a significant acceleration for both the inversion algorithm and the resulting inverse apply.

Computational complexity. Most of the computational cost in the inversion algorithm lies in solving the local linear systems until the desired accuracy in TT approximation for the inverse is achieved. As mentioned in §2, these algorithms exhibit linear convergence similar to that of the AMEN compression algorithms, and so the number of cycles through the cores of A^{-1} is typically controlled by the maximum TT rank for the inverse with the desired target accuracy. For all examples considered in §5, experimental evidence suggests ranks are bounded.

Let the TT ranks of A be bounded by r_{A}, n denote the tensor mode size, and r_{X} bound the TT ranks of A^{-1}. The size of local systems is then bounded by $nr_{\mathsf{X}}^2 \times nr_{\mathsf{X}}^2$, and the cost of inverting them directly is at most $\mathscr{O}\left(n^3 r_{\mathsf{X}}^6\right)$. Using an iterative algorithm to solve local systems, the complexity for well-conditioned matrices goes down to $\mathscr{O}\left(r_{\mathsf{X}}^3 r_{\mathsf{A}} + r_{\mathsf{X}}^2 r_{\mathsf{A}}^2\right)$, i.e. the cost of applying the associated matrix in the tensor form. Since a system is solved for each of the d cores of A and A^{-1}, an estimate for the complexity of the whole algorithm is $\mathscr{O}\left((r_{\mathsf{X}}^3 r_{\mathsf{A}} + r_{\mathsf{X}}^2 r_{\mathsf{A}}^2)\log N\right)$.

4.2. *TT Matrix-vector products*

The second component needed by solver or a preconditioner is a matrix-vector product for a matrix represented in the TT format. In the cases where the vector can be compressed well in the TT form (e.g., for smooth data), additional benefit may be derived from computing matrix vector apply in this format.

TT Compressed matrix-vector Product. Let A be a matrix and b a vector with TT decompositions consisting of cores $\mathcal{G}_k^{\mathsf{A}}(\alpha_{k-1}, \overline{i_k j_k}, \alpha_k)$ and $\mathcal{G}_k^{\mathsf{b}}(\beta_{k-1}, j_k, \beta_k)$, respectively. Cores for a TT decomposition of the product $\mathsf{c} = \mathsf{Ab}$ is computed as

$$\mathcal{G}_k^{\mathsf{c}}(\overline{\alpha_{k-1}\beta_{k-1}}, i_k, \overline{\alpha_k \beta_k}) = \sum_{j_k} \mathcal{G}_k^{\mathsf{A}}(\alpha_{k-1}, \overline{i_k j_k}, \alpha_k)\mathcal{G}_k^{\mathsf{b}}(\beta_{k-1}, j_k, \beta_k). \tag{4.2}$$

That is, each core of C involves the contraction over the auxiliary index j_k, and a concatenation of the two pairs of auxiliary indices $(\alpha_{k-1}, \beta_{k-1})$ and (α_k, β_k). This is a special case of the matrix-matrix product in which b and its cores run only over the j_k indices, instead of a corresponding pair of row and column indices. If the TT ranks for the matrix and the vector are bounded by r_{A} and r_{b}, respectively, the overall computational complexity of this structured product is $\mathscr{O}\left(r_{\mathsf{A}}^2 r_{\mathsf{b}}^2 \log N\right)$.

Algorithm 2 TT MATRIX BY UNCOMPRESSED VECTOR PRODUCT.

```
1: Inputs: TT decomposition A with cores $\mathcal{G}_k$, column vector b
2: Output: vector $y = Ab$
3: Initialize $y_0 = b^T$, and $\alpha_0, \alpha_d$ as size 1 trivial indices.
4: for $k = 1$ to $d$ do
5:     Permute core dimensions and reshape as a matrix of size $r_k m_k \times r_{k-1} n_k$:
           $$M_k\left(\overline{\alpha_k i_k}, \overline{\alpha_{k-1} j_k}\right) = \mathcal{G}_k\left(\alpha_{k-1}, \overline{i_k j_k}, \alpha_k\right)$$
6:     Reshape $y_{k-1}$ to merge columns from the $n_k$ children of each source box $B$, indexed by $j_k$:
           $$b_k\left(\overline{\alpha_{k-1} j_k}, \overline{J_k^{BOX} I_{k-1}^{LCL}}\right) = y_{k-1}\left(\alpha_{k-1}, \overline{J_{k-1}^{BOX} I_{k-1}^{LCL}}\right)$$
7:     Obtain data for each target children, indexed by $i_k$:
           $$\phi_k = M_k b_k$$
8:     Permute $\phi_k$ (separate rows from the $m_k$ children of a target box $B$):
           $$y_k(\alpha_k, \overline{J_k^{BOX} I_k^{LCL}}) = \phi_k(\overline{\alpha_k i_k}, \overline{J_k^{BOX} I_{k-1}^{LCL}})$$
9: end for
10: return $y = y_d^T$
```

Matrix-vector product for an uncompressed vector. The algorithm for this case is given in Algorithm 2. In this case, the complexity of the algorithm increases to $\mathcal{O}\left(r_A^2 N \log N\right)$.

Given a TT decomposition for a matrix A, the algorithm proceeds by contracting one dimension at a time, applying the kth TT core. For efficiency, it is much faster to do this contraction as a matrix-vector operation and most of the work in Algorithm 2 is to prepare the operands for this in line 7. In the context of the matrix-block tree, sequentially contracting indices implies an upward pass through the tree, in which one level of the hierarchy is processed at a time, eliminating one source index j_k to compute the part of the matrix-vector product corresponding to the target index i_k. This upward pass produces a series of intermediate arrays indexed by $\{i_1, \ldots, i_k\}$ and $\{j_{k+1}, \ldots, j_d\}$. As remarked before, the first set determines local coordinates at each box of the target tree, and the second set corresponds to a box index at level k of the source tree. To reflect this, we use the following notation

$$I_k^{\text{LCL}} = \overline{i_1 \cdots i_k}, \quad J_k^{\text{BOX}} = \overline{j_{k+1} \cdots j_d}. \tag{4.3}$$

Notice that, by definition, $I_k^{\text{LCL}} = \overline{I_{k-1}^{\text{LCL}} i_k}$ and $J_{k-1}^{\text{BOX}} = \overline{j_k J_k^{\text{BOX}}}$.

When Algorithm 2 initializes, the vector that core $\mathcal{G}_1$ acts upon is the first unfolding matrix of b

$$b_1(j_1, \overline{j_2 \ldots j_d}) = b(\overline{j_1 \ldots j_d}) \tag{4.4}$$

For each $k > 1$, we reshape the result y_{k-1} from the previous step in line 6 in order to apply the core $\mathcal{G}_k$. For each source box B at level k, the n_k columns of size r_k from its children are merged.

The reshaped core M_k consists of $m_k \times n_k$ blocks each of size $r_{k+1} \times r_k$. By applying it to b_k, we obtain r_{k+1} results for each of the m_k boxes on $\mathcal{T}_{\text{trg}}$ at level k. In line 8, we separate each block row, so that the last column indices of y_k correspond to box indices in the target tree.

The matrix vector product in line 7 is between a matrix of size $r_{k+1} m_k \times r_k n_k$ and a vector of size $r_k n_k \times \frac{N}{n_k}$, and so it requires $2 m_k r_{k+1} r_k N$ operations. If $m_k = n_k = n$ and $r_k \leq r$ for all k, this computation is $\mathcal{O}\left(r^2 N\right)$. Since there are $d = \log_n N$ cores, the total computational cost is $\mathcal{O}\left(r^2 N \log N\right)$.

5. Numerical experiments

In this section, we present the results of a series of numerical experiments quantifying the performance of the TT decomposition and inversion algorithms discussed in §2 and §4. As our model problem, we use the linear system of equations arising from the Nyström discretization of volume and boundary integral operators in three dimensions, Eq. (IE) and Eq. (LS). For each kind of operator, we construct TT based accelerated solvers and compare them to some of the other existing alternatives.

The results in §5.1 demonstrate that for volume integral equations with non-oscillatory kernels, the TT inversion is cost-effective for moderate to high target accuracies ($\varepsilon \leq 10^{-6}$). Accordingly, we propose using the TT inversion combined with the fast matrix-vector algorithms (§4.2) as a fast direct solver. We examine the performance of the TT based direct solver on translation and non-translation invariant integrals.

In §5.2, we explore the application of the TT in inversion of matrices arising from boundary integral equations in complex geometries. For these systems, we demonstrate that using low-accuracy TT compression as a preconditioner for a Krylov subspace iterative methods such as GMRES is a cost effective approach with respect to time and memory requirements.

We make use of the Matlab TT-Toolbox [Ose12] for all tensor computations, and FMMLIB3D [GG11] as an accurate and fast matrix-vector apply. All experiments are performed serially on Intel Xeon E–2690 v2(3.0 GHz) nodes with 64 GB of memory.

In the experiments of this section, unless otherwise stated, we take the kernel $K(r)$ to be the free-space Laplace Green's function $K(r) = \frac{1}{4\pi r}$.

5.1. *Volume integral equations*

We test the performance of the TT decomposition as a volume integral solver for a box domain. We discretize the integral kernel on a regular grid with $N = 2^d$ points (for both sources and targets), indexing them according to successive bisection of the domain along each coordinate direction. This corresponds to a uniform binary tree with $d - 1$ levels. TT decompositions obtained for the resulting matrix and its inverse correspond to tensors of dimension d and mode sizes $n_k = 4$.

For problem sizes ranging from $N = 4\,096$ to $16\,777\,216$, we report the time it takes to compress and invert the matrix in the TT form, the resulting maximum TT ranks, and the storage requirements for the inverse. For the application of the TT inverse, we test both of the apply algorithms in §4.2. We report the time it takes to apply the TT inverse to a random, dense vector of size N (denoted as "solve") as well as the time it takes to compress a vector of size N sampled from a smooth function and then apply the inverse to it (denoted as "TT solve"). From the analysis in §4.2, we conclude that applying the TT inverse to compressed right-hand-sides ("TT solve") will be most advantageous if they have small maximum TT ranks.

As mentioned in §1, state-of-the-art direct solvers for HSS and other hierarchical matrices when applied to volume integrals in 3D exhibit above linear scaling, as well as significantly high setup and storage costs, limiting their practicality. This makes the TT an extremely attractive alternative in this setting. For example, for $N = 262\,144 = 64^3$, solving the translation invariant problem in §5.1.1 using the HIF solver for a target accuracy of $\varepsilon = 10^{-6}$ requires a setup time of 32 hours, as well as 40 GB of memory. Setting up the corresponding TT inverse takes 86 seconds, and requires only 2 MBs of memory (See Table 1). (We emphasize that the solve times for arbitrary right-hand sides still scale as $\mathscr{O}(N \log N)$).

5.1.1. *Translation invariant kernels in 3D*

We first consider the three dimensional unit box and solve the translation invariant system corresponding to the Laplace single-layer kernel. In Table 1, we report the results of this experiment. We set a target accuracy of algorithms involved to $\varepsilon = 10^{-6}$. For each experiment, we test the accuracy of both forward and inverse apply against random, dense vectors, obtaining residuals ranging from 1.5×10^{-7} to 5.5×10^{-7} and 1.0×10^{-6} to 1.3×10^{-6}, respectively.

Rank behavior and precomputation costs. We see in Table 1 that the maximum TT rank for the system matrix given a target accuracy is bounded, which is consistent with our argument in §3. As noted in §4, although we have no concrete estimate for the rank behavior of the inverse, in all cases considered in this work we observe that the maximum rank of the inverse is proportional to that

N	Time (sec)		Max Rank		Inverse Memory (MB)	Inverse Apply (sec)		
	Compress	Invert	Forward	Inverse		Solve	TT Solve Compress	& Apply
16^3	2.79	118.19	103	144	2.60	0.07	0.72	2.41
32^3	2.89	141.27	106	125	2.86	0.76	1.19	4.87
64^3	4.35	82.07	99	97	2.29	4.41	4.40	4.79
128^3	5.69	60.23	90	74	1.68	29.31	27.52	3.80
256^3	6.04	33.53	80	57	1.19	189.53	31.93	2.39

Table 1: TRANSLATION INVARIANT 3D VOLUME LAPLACE KERNEL. *Compression and Inversion times, maximum TT ranks, memory requirements and solve times for the TT decomposition algorithms applied to the 3D Laplace single-layer kernel. The problem sizes range from $N = 4096$ to 16777216, and the target accuracy for all algorithms is set to $\varepsilon = 10^{-6}$. Achieved accuracies for the solve match this target accuracy closely, ranging from 1.0×10^{-6} to 1.3×10^{-6}. For "TT Solve" we report the time required for the compression of the right-hand-side and the application of the TT inverse to the compressed vector. In this case, the right-hand-side is $f(x,y,z) = \phi(x)\phi(y)\phi(z)$ with $\phi(t) = \mathrm{diric}(t,10)$ (Dirichlet periodic* sinc *function), which is a smooth, oscillatory functions whose TT ranks are observed (see §5.1.3) to be bounded $(r \le 75)$ for the case considered in this experiment.*

of the original matrix. For a wide range of integral kernels in $1, 2$, and 3 dimensions, we in fact observe inverse ranks decrease with N. Moreover, even though additional levels (and corresponding tensor dimensions) are added as N increases, the decrease in TT ranks is substantial enough to bring down the inversion costs, as well as the storage requirements for the inverse in TT form, as shown in Table 1. We investigate this behavior in §5.1.3.

Since forward ranks are bounded, the time it takes to produce the TT factorization (Compress column in Table 1) displays logarithmic growth with N. In contrast, while we observe that the inversion time is also proportional to the maximum TT ranks, the inversion algorithm has a more complex relationship with the resulting rank distribution. Perhaps the most outstanding consequence of this is how economical the computation and storage of the inverse in the TT format is. For $N = 16777216 = 256^3$, with a target accuracy of $\varepsilon = 10^{-6}$, it takes only 34 seconds to compute the inverse using 1.2 MBs of storage.

When the target accuracy is increased to $\varepsilon = 10^{-8}$, the results of experiments exhibit similar rank behavior and scaling of the computational costs to those reported in Table 1. However, the maximum TT ranks for the matrix and its inverse increase to 200 and 300, respectively. The higher ranks imply higher algorithmic constant for compression, inversion, and apply steps but these costs stay quite economical for this case as well.

Inverse apply. As noted in §4.2, the application of a matrix in the TT form has computational complexity dependent on the structure of the operand. If the operand is compressible in the TT format, the inverse apply is $\mathcal{O}(\log N)$ and otherwise $\mathcal{O}(N \log N)$. In Table 1, we report timing for both types of right-hand-side. When the right-hand-side is compressible in TT form, we report timings for right-hand-side compression ("Compress") and TT matrix-vector multiply ("Apply") in the last two columns of the table.

We choose a tensor product of periodic sinc functions as a compressible right-hand-side. It is observed in §5.1.3, that these are smooth, oscillatory functions whose TT ranks are bounded $(r_\mathrm{b} \le 75)$ for the case considered in this experiment. As indicated in the complexity analysis in §4.2, computation for this inverse apply depends on both the ranks of the inverse and of the right-hand-side. Given rank bounds r_A for the matrix and r_b for the right-hand-side, the complexity is $\mathcal{O}\left(r_\mathrm{A}^2 r_\mathrm{b}^2 \log N\right)$. Here, the decrease in the inverse ranks of A brings down the cost of the apply.

5.1.2. Non-Translation invariant kernels in 3D

Here we test the ability of the TT decomposition to handle non-translation invariant kernels by choosing $b(x)$ and $c(y)$ in Eq. (IE) to Gaussians of the form

$$b(x) = 1 + e^{-(x-x_0)^T(x-x_0)}, \quad c(y) = 1 + e^{-(y-y_0)^T(y-y_0)}, \tag{5.1}$$

N	Time (sec)		Max Rank		Inverse Memory (MB)	Inverse Apply (sec)		
							TT Solve	
	Compress	Inverse	Forward	Inverse		Solve	Compress	& Apply
16^3	42.51	2510.38	386	209	5.33	0.14	0.73	5.65
32^3	62.67	1787.77	364	161	5.04	1.13	1.16	15.45
64^3	71.18	647.40	301	113	3.30	6.33	2.93	12.79
128^3	48.84	234.96	232	81	2.03	35.84	27.32	7.54
256^3	13.35	64.90	130	62	1.37	219.63	32.20	3.71

Table 2: Non-translation invariant 3D volume Laplace kernel. *Compression and Inversion times, maximum TT ranks, memory requirements and solve times for the TT decomposition algorithms applied to the non-translation invariant 3D Laplace single-layer kernel. Problem sizes range from* $N = 4\,096$ *to* $16\,777\,216$*, and target accuracy* $\varepsilon = 10^{-6}$*. Achieved accuracies for the solve match this target accuracy closely, ranging from* 1.2×10^{-6} *to* 2.0×10^{-6}*. Timings under "TT solve" include compression of right-hand-sides obtained from the sampling of* $f(x,y,z) = \phi(x)\phi(y)\phi(z)$ *with* $\phi(t) = \text{diric}(t,10)$ *(Dirichlet periodic sinc function) and the TT apply.*

N	Maximum forward (inverse) ranks				
	$K(r) = r$	$r^{1/2}$	$r^{-1/2}$	$\log(r)$	$\text{diric}(r,10)$
4 096	3 (5)	12 (51)	13 (59)	11 (52)	10 (10)
65 536	3 (5)	11 (47)	12 (55)	11 (46)	10 (10)
1 048 576	3 (5)	11 (45)	12 (49)	10 (41)	10 (10)
16 777 216	3 (5)	10 (42)	12 (44)	10 (38)	10 (10)

Table 3: Forward and inverse maximum TT ranks for 1D examples. *The behavior of the maximum TT ranks of matrices and their inverses (in parenthesis) for different integral kernels. The compression and inversion accuracies are both set to* $\varepsilon = 10^{-10}$.

as it was done in [Cor+14]. We report the results in Table 2. We again test the accuracy of both forward and inverse applies, obtaining residuals ranging from 1.1×10^{-6} to 1.4×10^{-6} and 1.2×10^{-6} to 2.0×10^{-6}, respectively.

For non-translation invariant kernels such as the one tested in Table 2, the ranks of the matrix is expected to increase as a function of the TT ranks of b and c ($r_{\mathrm{B}}, r_{\mathrm{C}} \simeq 30$, in this case). However, it's interesting to note that the ranks of the inverse do not seem to increase much compared to the corresponding translation-invariant case, Table 1. This is reflected in the performance of both kinds of inverse applies. Comparing the corresponding columns of Table 2 and Table 1, we observe that the difference in performance between both experiments decreases as N increases.

5.1.3. *Rank behavior of integral kernels and their inverses*

Throughout our experiments with volume integral equations of the form given in Eq. (IE), including those presented in §5.1.1 and §5.1.2 as well as analogous examples in one and two dimensions, we observed that while the maximum TT rank of the matrix A is relatively constant for different problem sizes N, the maximum TT rank of the corresponding inverse A^{-1} typically decreases.

While further analysis is needed to explain this phenomenon, we conducted a series of experiments for 1D kernels to better understand this behavior and confirm that it is not a side effect of different numerical algorithms we used. In Table 3, we report the maximum TT ranks for the matrices and their inverses arising from the discretization of different 1D kernels.

Separable kernels such as $K(r) = r$, $\cos(r)$, $\sin(r)$, or e^r and their inverses are known to have exact TT ranks, which is confirmed in our experiments. Periodic kernels like the Dirichlet periodic sinc, $\text{diric}(r,k)$, make the convolution circular, and result in circulant matrices. We observe that in this case, TT ranks for the matrix and its inverse are identical, and they remain stable as N grows. Non-periodic integrable kernels, like $K(r) = r^{1/2}$ or $\log(r)$ display TT rank behavior similar to that of volume integral experiments in §5.1.1 and §5.1.2. Ranks for the forward matrix decrease, albeit slowly, and ranks for the inverse show a steady decrease for the problem sizes tested. TT rank

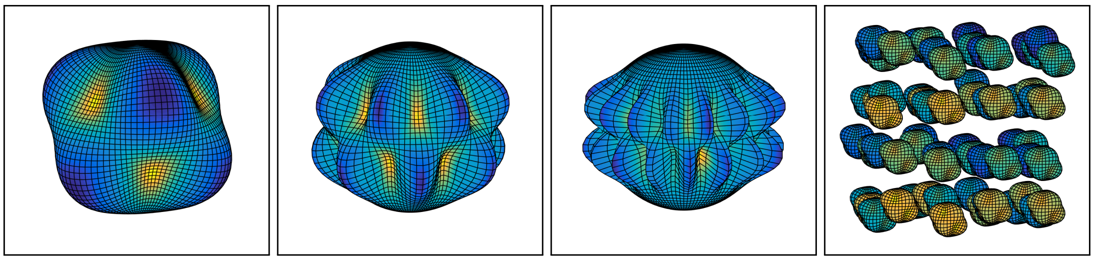

Figure 2: Example of test geometries. *(a)–(c) Example surfaces used in our experiments. The chart for surfaces is given by $\rho(\theta,\phi) = 1 + 0.5Y_{\ell m}(\theta,\phi)$ where $Y_{\ell m}$ denotes the spherical harmonics. The shading is the signed mean curvature. (d) A random lattice of* 64 *shapes (shading base color is varied to help better distinguish different surfaces).*

distributions seem to vary with smoothness and integrability, in accordance to estimates in [KS15].

We also note that, as is the case for the volume experiments in 3D, the steady decrease of inverse ranks results in a decrease in storage. Even though increasing N implies an increase in tensor dimensions (and in the number of cores), this effect does not seem to overcome the decrease in ranks or storage for the cases observed.

Note that, since this rank behavior is also present in 1D cases, it is not likely to be a side effect of how the matrix indices are tensorized (see Remark 2.1). Also since the same behavior is observed in a variety of smooth kernels, weakly singular nature of the kernel or the numerical quadrature are not expected to cause the rank behavior.

We also confirmed that different inverse algorithms (AMEN or DMRG) as well as direct compression for moderate problem sizes (using AMEN cross or TT-SVD as in Algorithm 1) yield almost identical rank distributions.

5.2. Boundary integral equations in complex geometries

Except for simple surfaces, it is generally the case that for a given target accuracy, applying the TT decomposition and inversion algorithms as described in the previous sections will yield TT ranks higher than in the volume cases described in §5.1. As indicated in §3, this is likely due to the loss of translation invariance, which makes self and near interactions less compressible.

As is the case for other fast direct solvers, the increase in TT ranks implies higher algorithmic constants, and so it becomes more practical to compute the TT compression and inversion at low target accuracies and use them as robust preconditioners for an iterative algorithm such as GMRES. In this section, we demonstrate the application of the TT decomposition as a cost effective and robust preconditioner for boundary integral equations.

5.2.1. Experiment setup

In order to build an example closer to boundary value problems encountered in applications, we pose an exterior Dirichlet problem for the Laplace equation on a multiply-connected complex domain with boundary Γ

$$\frac{1}{2}\sigma(x) + \int_\Gamma D(\|x-y\|)\sigma(y)\,d\Gamma_y = f(x), \tag{IE}$$

where $D(r)$ is the Laplace double-layer kernel in 3D. We modify this kernel by adding rank one modifications per surface to match the far-field decay [Kre+89]. For Γ, we choose an irregular cubic lattice of closed surfaces $\Gamma_i\,(i = 1,\ldots,q^3)$ of genus zero (spherical topology). Examples of such shapes and their distribution are shown in Fig. 2. We discretize each surface using a basis set of spherical harmonics of order p, and compute singular integrals using fast and spectrally accurate singular quadratures [GS02]. This setup, while being relevant to problems in electrostatics and fluid flow (particulate Stokes flow), enables us to study the effects of individual surface complexity as well as of interactions between surfaces on the performance of the TT preconditioner.

N	Time (sec)		Max Rank		Inverse Memory (MB)
	Compress	Invert	Forward	Inverse	
3 840	27.75	375.49	117	86 (49)	8.56
30 720	34.80	512.18	133	90 (49)	11.73
245 760	45.42	630.46	144	93 (50)	14.25
1 996 080	49.50	732.21	150	95 (50)	18.75

Table 4: TT PRECONDITIONER RANKS AND PRECOMPUTATION COSTS FOR THE BENCHMARK CASE. *Compression and Inversion times, maximum TT ranks, and memory requirements for the TT decomposition algorithms for benchmark case on a regular cubic lattice. The TT inverse is a product of two TT matrices, block-diagonal system for spherical surfaces* $\mathtt{M}$ *and the solution for the right-preconditioned system* $\mathtt{X}$*, as in §4. We include maximum ranks for both TT matrices, reporting those for block-diagonal* $\mathtt{M}$ *in parentheses, and reporting the aggregate memory requirements for both under Inverse Memory. The problem sizes range from* $N = 3\,840$ $(n = 8, p = 15)$ *to* $1\,996\,080$ $(n = 4096, p = 15)$*, and the target accuracy for all algorithms to construct the TT preconditioner is set to* $\varepsilon_p = 10^{-3}$*. Model surface has radius* $\rho = 1 + 0.5Y_{4,3}$.

In all experiments in this section, we compress a corresponding block-diagonal system M for spheres in TT form, effectively preconditioning the original linear system as in §4. The TT preconditioner is thus the product of two TT matrices $\mathtt{M}$ and $\mathtt{X}$. This provides a significant acceleration for the inversion algorithm, improving the conditioning of the local linear systems within it. It also provides an acceleration for the resulting inverse apply, as the resulting ranks of $\mathtt{M}$ and $\mathtt{X}$ are observed to be smaller than those of their product.

For the sake of comparison with the volume integral experiments in §5.1, in Table 4 we report results for matrix compression and preconditioner setup in TT form for $\varepsilon_p = 10^{-3}$, for a regular lattice of surfaces with spherical topology and radius $\rho = 1 + 0.5Y_{4,3}$. Unlike the cases in Table 1 and Table 2, maximum ranks for both the forward matrix $\mathtt{A}$ and the matrix $\mathtt{X}$ in the preconditioner show a relatively slow but steady increase with problem size N. As a consequence, both timings and inverse memory display sublinear growth with N. We note that in terms of rank behavior and the scaling of precomputation costs, the results presented in Table 4 are representative of all experiments presented in this section.

5.2.2. *Comparison with other preconditioners*

Multigrid, sparsifying preconditioners [QB13, Yin14], and hierarchical matrix preconditioners (using HSS-C, HIF, IFMM [Cou+15], or $\mathcal{H}$ inverse compression at low target accuracies) are a few options available for this type of problem.

We compare TT's setup costs (inversion time and memory requirements), its effectiveness in terms of iteration count for the iterative solver, and total solve time with those of multigrid and HIF. We use a two-level multigrid V-cycle, considered in [QB13], as preconditioner. Levels in this context are defined by the spherical harmonic order p. In our experiments, we choose $p_{\text{coarse}} = \lceil p/2 \rceil$ as the coarse level. We use the natural spectral truncation and padding for restriction and prolongation operators, and for smoothing, we use Picard iteration. The coarse-grid problems are solved iteratively using GMRES with tolerance ε_p. For HIF [HY15], we use a Matlab implementation kindly provided to us by Kenneth Ho and Lexing Ying.

For elliptic PDEs, multigrid provides an acceleration to the iterative solvers with almost negligible setup costs and storage requirements; however, its performance for integral equations is understood less. Fast direct solvers based on hierarchical matrix compression like HIF, on the other hand, have significantly large setup costs. Nevertheless, when the latter kind of preconditioners are affordable to construct for low accuracies, they present an effective preconditioner, reducing the number of iterations while incurring small cost associated with the application of the preconditioner, because of their quick apply.

Since the TT algorithms also provide a hierarchical decomposition of the inverse, we expect their performance to be similar to fast direct solvers such as HIF. We also anticipate that, due to its modest setup costs, it will enable the solution of problems with a large number of unknowns.

p	n	N	Unpreconditioned			Multigrid			TT			HIF		
			$\ell = 4$	8	16	4	8	16	4	8	16	4	8	16
15	8	3 840	129	401	1001	25 (140)	–	–	9	9	9	7	7	7
15	64	30 720	153	255	991	25 (140)	–	–	11	11	9	7	7	9
15	512	245 760	141	277	788	25 (140)	–	–	11	11	11	7	11	7
24	8	9 600	153	229	617	21 (92)	37 (180)	–	9	13	11	7	7	7
24	64	76 800	141	207	677	21 (114)	37 (180)	–	11	13	15	9	7	7
24	512	614 400	141	205	577	21 (103)	37 (180)	–	11	11	15	15	13	13
35	8	20 160	139	429	601	17 (16)	25 (84)	25 (140)	9	11	11	7	7	9
35	64	161 280	143	187	651	17 (16)	25 (73)	25 (140)	9	11	11	9	7	9
35	512	1 290 240	137	193	595	17 (16)	25 (79)	25 (140)	9	11	11	–	–	–

Table 5: MATRIX-VECTOR APPLY COUNT FOR THE BENCHMARK CASE. *Comparison of unpreconditioned GMRES and preconditioned GMRES using multigrid, TT, and HIF preconditioners for the exterior 3D Laplace problem over a regular lattice of n model surfaces each with radius* $\rho = 1 + 0.5Y_{\ell m}$ *where* $\ell = \{4, 8, 16\}$ *and* $m = \ell - 1$. *Surfaces are represented in spherical harmonics basis and discretized with* $2p(p+1)$ *collocation points. The relative residual tolerance for GMRES is set to* $\varepsilon = 10^{-8}$ *and GMRES is not restarted. For multigrid, the number of applies in the coarse GMRES solver (with* $\varepsilon_p = 10^{-2}$*) is reported in parentheses. TT and HIF approximate-inverse preconditioners are constructed with a target accuracy of* $\varepsilon_p = 10^{-3}$. *Empty entries correspond to experiments in which either the preconditioned GMRES failed to converge in* 100 *iterations or preconditioner setup costs were excessive.*

5.2.3. Benchmark

We test a cubic lattice of $q \times q \times q$ surfaces, for $n = q^3 \in \{8, 64, 512, 4096\}$, discretizing each surface using spherical harmonic basis of order $p \in \{15, 24, 35\}$, which requires $p+1$ collocation points in the latitude direction and $2p$ collocation points in the longitude direction. The total number of unknowns for each problem is then $N = 2p(p+1)n$.

In this initial experiment, we make all surfaces translations of a single shape on a regular lattice with spacing of 4, which implies the closest distance between surfaces is slightly smaller than their diameter. To control surface complexity, we make their radius $\rho(\theta, \phi) = 1 + 0.5Y_{\ell m}(\theta, \phi)$, where $Y_{\ell m}$ is the spherical harmonic function of order (ℓ, m).

For the matrix-vector apply, we use the FMMLIB3D library [GG11] for interactions between surfaces, and spectral quadrature for interactions within each surface [GS02]. We test three surfaces of increasing complexity by setting $\ell \in \{4, 8, 16\}$ and $m = \ell - 1$, shown in Fig. 2.

The results of the test with different preconditioner are reported in Table 5. For all problem sizes, we compare the number of matrix-vector applications for the unpreconditioned solve with those of the preconditioned solve with three different types of preconditioners (multigrid, TT, and HIF). Each approximate-inverse preconditioner is constructed for a target accuracy of $\varepsilon_p = 10^{-3}$ and is used within a GMRES solve with a tolerance of $\varepsilon = 10^{-8}$ for relative residual and no restart.

From the results in Table 5, it can be readily observed that in this configuration the condition number of the problem, implied by number of iterations of the unpreconditioned solve, mostly depends on the individual surface complexity and not the problem size.

As we increase surface complexity (controlled by ℓ), the average number of matrix-vector applies goes from 140 to 240 to 600, rendering the unpreconditioned solve impractical. The multigrid preconditioner is easy and inexpensive to setup, but, as it is mentioned in [QB13], its performance suffers when the geometry is not resolved in the coarse grid, i.e. p_{coarse} is not large enough. In Table 5, we observe that after the individual surface geometry is resolved in the coarse grid, multigrid leads to a reduction in the iteration counts. However, this requires unnecessary over-resolution in the fine grid that is not cause by the setup of the problem but by the preconditioner.

Furthermore, note that each multigrid cycle uses two matrix-vector applies at the fine level as well as a number of applies at the coarse level, and so the observed speedups with respect to the unpreconditioned solve are moderate. Additionally, for cases where both fine and coarse levels are resolved, such as $p = 35$ and $\ell = 4$, the number of coarse applies is still higher than the corresponding TT and HIF direct solvers at the coarse level ($p = 15$ and $\ell = 4$). Due to its mediocre performance, we choose not to further consider multigrid for our comparisons, focusing instead on

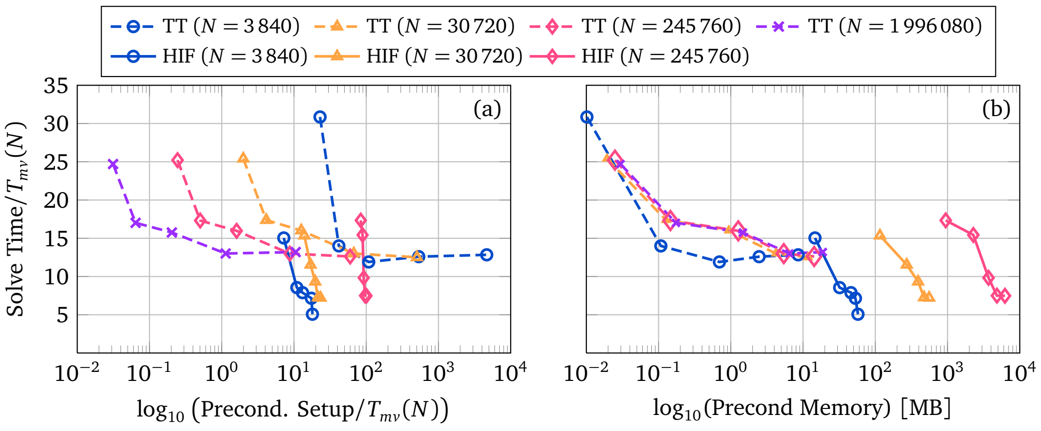

Figure 3: Solve time vs. preconditioner setup time and memory for the benchmark case. *Semi-log plots for solve times, normalized by the matvec time $T_{mv}(N)$, vs. the required inversion time, also normalized by $T_{mv}(N)$, and storage requirements (in MBs) for TT and HIF preconditioners for $p = 15$, $n = \{8, 64, 512, 4096\}$ and $\varepsilon_p = 10^{-1}$ to 10^{-3}. Model surface has radius $\rho = 1 + 0.5Y_{4,3}$. Because of $\log(N)$ complexity of TT compression and inversion, TT scheme becomes cheaper for larger problem sizes.*

comparing TT and HIF direct-solver preconditioners.

As expected, both TT and HIF display consistent performance across all cases considered and they display little to no dependence on surface complexity ℓ, lattice size n, and number of unknowns N. For both approaches, the cost of an iteration is one matrix-vector apply and one fast apply of the corresponding compressed low-accuracy inverse. For most examples considered, applying the preconditioner takes only a fraction of the matrix-vector apply, allowing for considerable speedups.

In the following section, we explore different aspects of TT and HIF preconditioners.

5.2.4. Comparison of direct-solver preconditioners

For a given model surface and problem size N, we report the setup costs (inversion time and storage) and the solve times using both direct solvers with inversion accuracies from $\varepsilon_p = 10^{-1}$ to 10^{-3}. In order to represent wall-clock time in a more meaningful way across experiments of different sizes, we normalize it in terms of the $\mathcal{O}(N)$ matrix-vector applies for the system matrix through FMM, hereinafter denoted by $T_{mv}(N)$. For $N > 500\,000$, high inversion costs generally prevent us from constructing the HIF preconditioner.

In Fig. 3, we plot solve times against setup costs (inversion time and storage) for both TT and HIF preconditioners. This allows us to identify distinct trade-offs in performance and efficiency, as well as to observe their scaling with respect to N and ε_p.

Solve time and speedup. As we increase the preconditioner accuracy ε_p, the number of iterations for the solve decreases while the cost of applying the corresponding preconditioner increases as TT and HIF ranks increase. Both direct solvers provide considerable speedup (the unpreconditioned solve takes about $140T_{mv}(N)$), with HIF mostly yielding higher speedups when available. The main reason behind this is that the HIF apply has a more favorable dependence on ε_p than the TT apply. This is evident by the fact that the rise in preconditioner application cost causes the TT solve time to plateau but it does not significantly affect the speedups yielded by HIF for the range of accuracies considered.

Inverse setup time. Fig. 3(a) shows the solve time versus the setup time as ε_p increases for different N. While HIF setup costs relative to $T_{mv}(N)$ increase as we increase the problem size, logarithmic scaling of the TT inverse makes it more efficient (relative to $T_{mv}(N)$) as N grows. This is one of the features that allows us to compute the TT for large N, and it implies that for sufficiently large problem ($N = 245\,760$ in this example), inverse setup can become cheaper than one matrix apply.

ε	Unprec. Matvecs	TT					HIF				
		$\varepsilon_p = 10^{-1}$	$10^{-1.5}$	10^{-2}	$10^{-2.5}$	10^{-3}	10^{-1}	$10^{-1.5}$	10^{-2}	$10^{-2.5}$	10^{-3}
10^{-4}	81	13.2	9.2	8.4	6.1	7.4	7.3	6.5	5.7	5.7	5.7
10^{-6}	107	19.4	14.8	13.3	11.1	10.6	13.5	10.9	8.0	5.7	5.7
10^{-8}	141	25.5	20.9	18.2	13.7	13.8	17.7	15.3	10.3	8.0	8.0
10^{-10}	165	33.2	25.4	23.0	17.2	17.0	22.4	16.6	12.7	10.3	10.3
10^{-12}	187	39.0	30.5	27.9	20.9	20.2	28.9	22.8	17.0	12.6	10.3

Table 6: SOLVE TIME FOR VARYING TARGET AND PRECONDITIONER ACCURACIES. *Comparison of solve times relative to $T_{mv}(N)$ for TT and HIF preconditioners for target accuracies $\varepsilon = 10^{-4}$ to 10^{-12} and preconditioner accuracies $\varepsilon_p = 10^{-1}$ to 10^{-3} for the model surface with radius $\rho = 1+0.5Y_{4,3}$, and problem size $N = 245\,760$ $(n = 512, p = 15)$. Matvec counts for the unpreconditioned solve are also included for reference.*

Inverse storage. Fig. 3(b) depicts the required storage for each preconditioner as ε_p increases. In this respect, TT is extremely efficient and memory requirements have logarithmic scaling with respect to N and do not exceed 100 MB. On the other hand, the HIF inverse displays $\mathcal{O}\left(N \log N\right)$ scaling with large prefactor, requiring 10s to 100s of GBs of memory to solve problems larger than a quarter million unknowns.

Dependence on surface complexity. As we increase the surface complexity by increasing the radial perturbation, we observe little to no difference in iteration counts (Table 5) as well as the solve times for both TT and HIF. Thus, both alleviate the increase in condition number, evident by the increase in the corresponding number of unpreconditioned iterations.

We do observe an increase in ranks and consequently in setup costs for the TT inverse. However, the difference in rank distributions sharply decreases with ε_p: maximum ranks for $(8,7)$ case are roughly 3× higher for $\varepsilon_p = 10^{-1}$ to less than 1.5× for $\varepsilon_p = 10^{-3}$. While there is also a slight increase in costs for HIF, it is predictably small due to the fact that it focuses on compressing far range interactions.

Effectiveness for different ε and ε_p. To investigate the robustness of the preconditioners for higher target accuracies, in Table 6, we report how solve times vary across a range of target and preconditioner accuracies. Overall, the solve times (relative to $T_{mv}(N)$) for both solvers seem to be proportional to $\log(\varepsilon)$, and their performance with respect to ε_p seems to replicate the case observed in Fig. 3 (which corresponds to $\varepsilon = 10^{-8}$). This indicates that these direct solvers are extremely reliable as preconditioners.

5.2.5. Perturbations of the benchmark

To further quantify the effectiveness of the TT preconditioner, we perform experiments in which we perturb the uniform cubic lattice considered in the benchmark. Given the ineffectiveness of the multigrid preconditioner presented above, we only focus on the HIF preconditioner for comparison. Since one expects the TT decomposition to exploit regularities and invariances in the geometry, this set of tests is aimed to measure its robustness when the regularity of the lattice is broken.

We construct each surface in the lattice with a radius of the form $\rho = 1 + rY_{\ell m}$, where r is a random number in $(0,1)$, and (ℓ, m) is also randomly chosen between $(4,3)$ and $(8,7)$. Additionally, we perturb the location of the center of each surface in a random direction by up to 50 percent of its diameter. The results of this test are shown in Fig. 4.

In Fig. 4, we observe similar behavior to the one seen in Fig. 3 for both solvers in terms of how solve times and setup costs behave as functions of the number of surfaces and preconditioner accuracy ε_p.

Comparing the corresponding data for the TT solver in these two figures, we observe a moderate increase in inverse setup times and storage for $n = 256, 4\,096$ $(N \geq 245\,760)$. This has an impact on the effective solve times in terms of matvecs, as the preconditioner apply is a bit more expensive.

We also observe that for both TT and HIF, $\varepsilon_p \simeq 10^{-1}$ seems to be much less effective than in the benchmark case. However, for $\varepsilon_p \leq 10^{-2}$, iteration counts and solve times in matvecs display similar behavior to the benchmark case.

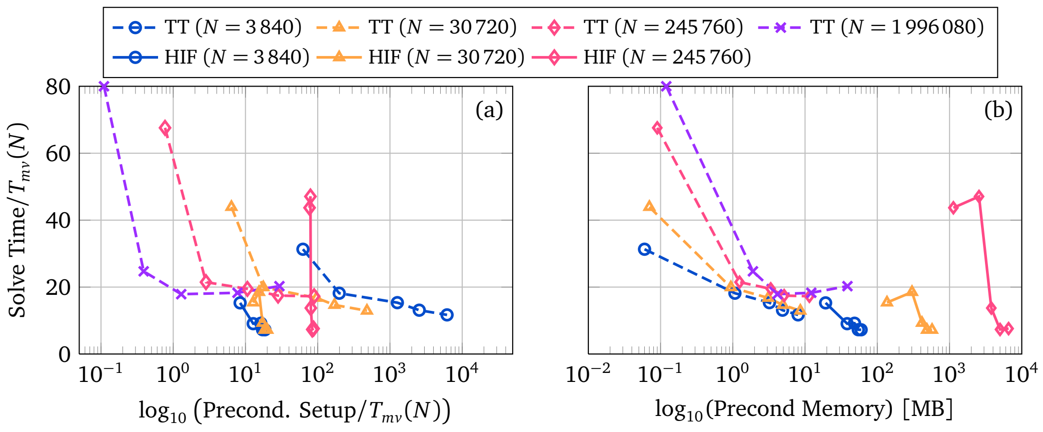

Figure 4: SOLVE VS. PRECONDITIONER SETUP AND MEMORY COSTS FOR IRREGULAR LATTICE AND SHAPES. *Semi-log plots for Solve times, normalized by matvec time $T_{mv}(N)$, for a given inversion time, also normalized by $T_{mv}(N)$, and storage requirements (in MBs) for TT and HIF preconditioners. Each model surface has radius $\rho = 1 + rY_{\ell,\ell-1}$, where ℓ is randomly chosen to be 4 or 8 and r is a random number in $(0, 1)$. The location of the surfaces is also randomly perturbed.*

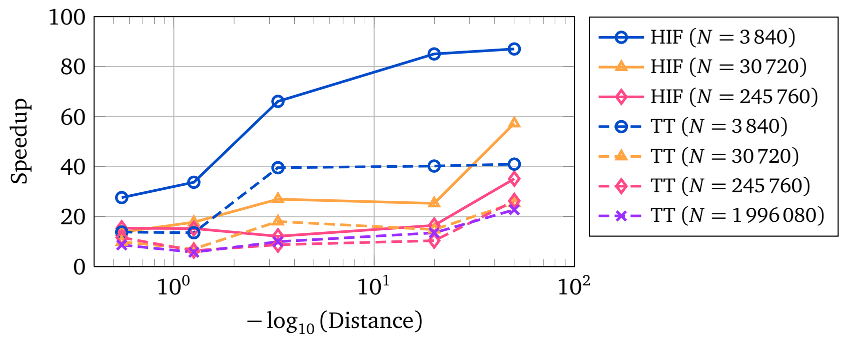

Figure 5: SOLVE SPEEDUP VS. LOG DISTANCE BETWEEN SURFACES. *Speedups, defined as the ratio between unpreconditioned and preconditioned solves and excluding the preconditioner setup time (see Fig. 6 for the setup costs), vs. log of the surface spacing in the lattice for TT and HIF preconditioners with $\varepsilon_p = 10^{-3}$.*

Distance between Surfaces. As mentioned in §1, it is well known that conditioning of second kind integral equations tends to deteriorate as surfaces come close to contact. In order to test robustness in performance of the TT and HIF preconditioners, we perform a series of tests, comparing speedups and setup costs as we draw surface centers in the lattice close to each other.

In Fig. 5, we report the speedups with respect to the average unpreconditioned solve, and plot them against the logarithm of the distance between surfaces in the lattice. Here, we use the number of unpreconditioned iterations as a surrogate for the problem conditioner number. As we draw the surfaces together, we observe that iteration counts and solve times for both preconditioners show a slight increase for low accuracy ($\varepsilon_p \simeq 10^{-1}$), becoming almost independent of distance for higher preconditioner accuracies ($\varepsilon_p \leq 10^{-2}$). This causes the effective speedup to increase as the unpreconditioned solve becomes more expensive, due to the increase in condition number.

In Fig. 6, we plot setup costs against the log of the distance between surfaces. Although displaying a sharp increase at first, preconditioner setup times increase at a pace much slower than the cost of the unpreconditioned solve, becoming more efficient. We again observe that as N increases, the TT preconditioner becomes more cost-effective, becoming just a fraction of an unpreconditioned solve for $N = 1\,966\,080$. The rate at which storage requirements increase for both solvers also slows down as we reduce the distance, which means that they both display robust behavior in spite of the added complexity.

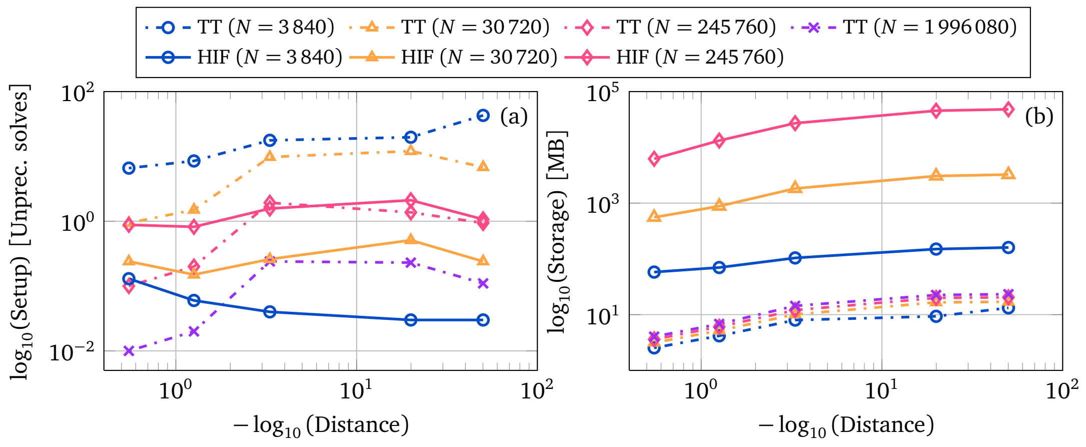

Figure 6: PRECONDITIONER SETUP AND MEMORY VS. LOG DISTANCE. *Log of setup costs (in unpreconditioned solves) and storage requirements vs. log of the surface spacing in the lattice for TT and HIF preconditioners with* $\varepsilon_p = 10^{-3}$.

5.2.6. Final remarks

The results presented here display the effectiveness of the TT preconditioner when applied to boundary integral equations in complex, multiply-connected geometries. Similar to the volume integral equations presented in §5.1, the TT decomposition is able to exploit regularities in the geometry and to retain robust performance and practical efficiency when these regularities are not present.

Both HIF and the TT preconditioners produce approximate inverses based on hierarchical compression and thus provide robust and reliable performance; substantially reducing iteration counts and solve times regardless of problem complexity. This is particularly remarkable for the tests in which surfaces come close to contact, as iteration counts and solve times for both solvers remain essentially unchanged in a regime where other classes of preconditioners become ineffective.

Between these two solvers, we observe a trade-off in terms of performance and efficiency: while the obtained speedups are generally higher for the HIF due to a faster inverse apply (although the difference decreases with problem size) the modest memory footprint and sublinear scaling of the setup cost for the TT make it extremely efficient, allowing the solution of problems with millions of unknowns. Hence, the choice between these and similar direct-solver preconditioner is problem and resource-dependent. For problems in fixed geometries involving a large number of right-hand-sides, the additional speedup provided by HIF might be desirable. For problems in moving geometries or with a large number of unknowns, TT provides an efficient, cost-effective preconditioner that can be cheaply updated.

6. Conclusions

Motivated by the ongoing challenges to produce memory-efficient and reliable fast solvers for integral equations in complex geometries, we presented a robust framework employing the Tensor Train decomposition to accelerate the solution of volume and boundary integral equations in three dimensions.

In the context of volume integral equations on a regularly sampled domain, even for problems with up to 10 million unknowns and for relatively high target accuracies ($\varepsilon = 10^{-6}$ to 10^{-10}), we are able to produce a compressed inverse in no more than a few minutes and store it using tens of MBs of memory, When compared to the current state-of-the-art direct solvers in three dimensions, TT is the only such fast direct solver to retain practical performance for large problem sizes.

In §5.2, we showed that the TT framework is applicable to matrices arising from the discretization of boundary integral equations in complex, multiply-connected geometries. Nonetheless, for a given target accuracy, the TT ranks are considerably higher than the volume integral case, rendering high target accuracies impractical. We thus proposed using a low target accuracy (e.g. ε_p between 10^{-1} and 10^{-3}) version of the TT-based inverse as a preconditioner for the GMRES. We compared its effectiveness and cost against two alternative preconditioners, a simple multigrid and a low-accuracy HIF inverse.

By virtue of being a multilevel, low-accuracy direct solver, the TT preconditioner matches the HIF in terms of reliable and robust performance across all examples presented. We observe a clear trade-off between these two solvers: while HIF generally provides higher speedups, modest setup costs and storage requirements for the TT make it extremely cost-effective. This is particularly the case for problems with a large number of unknowns, as setting up the TT preconditioner typically becomes comparable to one solve for the range of target and preconditioner accuracies presented. In §5.2, this allowed us to solve an external Dirichlet problem for the Laplace equation in an irregular domain composed of 4096 surfaces. The setup time for the TT preconditioner for this example is as small as a few FMM applies and has a memory footprint of less than 10 MBs.

There are several extensions of this work that can be pursued. Here, all presented examples used a uniform hierarchical partition of the domain, corresponding to a uniform tree. We believe it is possible to extend the current implementation of the TT algorithms to adaptive decompositions, using the connections between Tensor Train and other hierarchical decomposition techniques. Another research direction is generalizing of the TT algorithms to obtain effective parallel scaling. It would be interesting to see whether the compute-bound nature of the TT algorithms could be exploited to obtain good practical scaling.

7. Acknowledgments

We extend our thanks to Sergey Dolgov, Leslie Greengard, Ivan Oseledets, and Mark Tygert for stimulating conversations about various aspects of this work. We are also grateful to Michael O'Neil and Shravan Veerapaneni for critical reading of this work and providing valuable feedback.

Appendix A. Approximation with Kronecker products (Van Loan and Pitsianis)

In §2, we presented the application of the TT decomposition to a tensorized version of a matrix A, in which row and column indices are permuted to correspond to a matrix block hierarchy, encoded by the product tree $\mathscr{T}$. This means we are compressing a d-dimensional tensor with entries

$$\mathcal{A}_{\mathscr{T}}(\overline{i_1 j_1}, \ldots, \overline{i_d j_d}) = \mathsf{A}(\overline{i_1 \ldots i_d}, \overline{j_1 \ldots j_d}). \tag{A.1}$$

The TT decomposition exploits the fact that the unfolding matrices of $\mathcal{A}_{\mathscr{T}}$ are of low rank. One interesting question is what this means for the original matrix A.

This idea can be traced back to a paper by Van Loan and Pitsianis [VLP92] on the approximation of matrices with Kronecker products. That is, given an $M \times N$ matrix A, with $M = m_1 m_2$ and $N = n_1 n_2$, they consider the problem of finding the solution to

$$\min \|\mathsf{A} - \mathsf{B} \otimes \mathsf{C}\|_F , \tag{A.2}$$

with B of size $m_1 \times n_1$ and C of size $m_2 \times n_2$. A slight generalization of this is to attempt an approximation by the sum of r such products

$$\min \left\| \mathsf{A} - \sum_{k=1}^{r} \mathsf{B}_k \otimes \mathsf{C}_k \right\|_F , \tag{A.3}$$

where r is known as the Kronecker rank.

If we consider the corresponding partition of row and column indices $i = (i_1, i_2)$ and $j = (j_1, j_2)$, this means the approximation is a sum of terms $\mathsf{B}_k(i_1, j_1) \otimes \mathsf{C}_k(i_2, j_2)$, that is a separable approximation into two "levels" in the same sense we outlined for the TT algorithm in §2.

In [Hac+05], it is shown that if one rearranges the indices to produce the unfolding matrix $\mathsf{A}_{\mathscr{T}}(\overline{i_1 j_1}, \overline{i_2 j_2}) = \mathsf{A}(i, j)$, an equivalent problem is

$$\min \left\| \mathsf{A} - \sum_{k=1}^{r} \mathsf{B}_k \otimes \mathsf{C}_k \right\|_F = \min \left\| \mathsf{A}_{\mathscr{T}} - \sum_{k=1}^{r} \mathrm{vec}(\mathsf{B}_k)\mathrm{vec}(\mathsf{C}_k)^T \right\|_F . \tag{A.4}$$

That is, finding the best approximation of A with Kronecker products is equivalent to finding a rank r approximation of the corresponding unfolding matrix $\mathsf{A}_{\mathcal{T}}$. The best such approximation is given by the SVD of $\mathsf{A}_{\mathcal{T}} = \mathsf{U}\Sigma\mathsf{V}^\star$, truncated to r terms. Therefore, if this unfolding matrix is of low rank

$$\mathsf{A}_{\mathcal{T}} \approx \sum_{k=1}^{r} \sigma_k \mathsf{u}_k(\overline{i_1 j_1})\mathsf{v}_k(\overline{i_2 j_2})^\star, \qquad \mathsf{A} \approx \sum_{k=1}^{r} \sigma_k \mathsf{U}_k(i_1, j_1) \otimes \mathsf{V}_k(i_2, j_2), \tag{A.5}$$

with $\mathsf{U}_k = \texttt{reshape}(\mathsf{u}_k, [n_1, m_1])$ and $\mathsf{V}_k = \texttt{reshape}(\mathsf{v}_k, [n_2, m_2])$. Furthermore, $\{\mathsf{U}_k \otimes \mathsf{V}_k\}_{k=1}^r$ is an orthogonal set of matrices with respect to the Frobenius vector product, and this decomposition is generally known as a truncated Kronecker product SVD.

The TT decomposition can be interpreted as one of several possible multi-level generalizations of this matrix approximation problem. That is, we now have indices $i = \overline{i_1 i_2 \ldots i_d}$ and $j = \overline{j_1 j_2 \ldots j_d}$, and after performing the low rank approximation of the first unfolding matrix $\mathsf{A}^1_{\mathcal{T}}$, we have

$$\mathsf{A} \approx \sum_{\alpha_1=1}^{r_1} \mathsf{U}^1_{\alpha_1}(\overline{i_1 j_1}) \otimes \sigma_{\alpha_1} \mathsf{V}^1_{\alpha_1}(\overline{i_2 \ldots i_d}, \overline{j_2 \ldots i_d}) \tag{A.6}$$

Stacking the r_1 matrices $\sigma_{\alpha_1}\mathsf{V}^1_{\alpha_1}$ and merging the first index into $\overline{\alpha_1 i_2}$, we can perform the next round of Kronecker product approximation to obtain

$$\mathsf{A} \approx \sum_{\alpha_1=1}^{r_1} \mathsf{U}^1_{\alpha_1}(\overline{i_1 j_1}) \otimes \left(\sum_{\alpha_2}^{r_2} \mathsf{U}^2_{\alpha_1,\alpha_2}(\overline{i_2 j_2}) \otimes \sigma_{\alpha_2} \mathsf{V}^2_{\alpha_2}(\overline{i_3 \ldots i_d}, \overline{j_3 \ldots i_d}) \right). \tag{A.7}$$

We continue this procedure for each pair (i_k, j_k) until we obtain the TT decomposition.

Appendix B. TT decomposition as a hierarchical linear filter

In [OT11], the TT decomposition is interpreted as a subspace approach, and in the case of the SVD based TT decomposition (or more generally, when tensor cores are orthogonalized), as a fast method to compute a reduced orthogonal basis for structured tensors. We first show that the TT decomposition can be viewed as a linear filter with respect to each of its cores. We recall from §2 that the compression algorithm proceeds by computing a sequence of low rank decompositions of unfolding matrices. For the $(k-1)$th unfolding matrix, we have the low rank decomposition

$$\mathsf{A}^{k-1}_{\mathcal{T}}(\overline{i_1 \ldots i_{k-1}}, \overline{i_k \ldots i_d}) = \mathsf{U}_{k-1}(\overline{i_1 \ldots i_{k-1}}, \alpha_{k-1})\mathsf{V}_{k-1}(\alpha_{k-1}, \overline{i_k \ldots i_d}). \tag{B.1}$$

We then vectorize $\mathcal{A}_{\mathcal{T}}$, using the formula for products of matrices $\mathrm{vec}(\mathsf{ABC}) = (\mathsf{C}^T \otimes \mathsf{A})\mathrm{vec}(\mathsf{B})$, to obtain

$$\mathrm{vec}(\mathcal{A}_{\mathcal{T}}) = (\mathsf{I}_{m_{k-1}} \otimes \mathsf{U}_{k-1})\mathrm{vec}(\mathsf{V}_{k-1}), \tag{B.2}$$

where $\mathsf{I}_{m_{k-1}}$ is the identity of size $m_{k-1} = \prod_{q \geq k} n_q$, and so the left factor is in fact a block-diagonal matrix of m_{k-1} copies of U_{k-1}. If U_{k-1} is orthogonal this factor is orthogonal as well, and we also have that $\mathsf{V}_{k-1} = \mathsf{U}^\star_{k-1}\mathsf{A}^{k-1}_{\mathcal{T}}$ and $\mathrm{vec}(\mathsf{V}_{k-1}) = (\mathsf{I}_{m_{k-1}} \otimes \mathsf{U}^\star_{k-1})\mathrm{vec}(\mathcal{A}_{\mathcal{T}})$. Applying this same identity, we obtain

$$\mathrm{vec}(\mathcal{A}_{\mathcal{T}}) = (\mathsf{V}^T_{k-1} \otimes \mathsf{I}_{p_{k-1}})\mathrm{vec}(\mathsf{U}_{k-1}), \tag{B.3}$$

where $p_{k-1} = \prod_{q<k} n_q$, and again if V^T_{k-1} is orthogonal, $\mathsf{U}_{k-1} = (\mathsf{V}^T_{k-1})^\star \mathsf{A}^{k-1}_{\mathcal{T}}$ and $\mathrm{vec}(\mathsf{U}_{k-1}) = (\mathsf{V}^T_{k-1} \otimes \mathsf{I}_{p_{k-1}})^\star \mathrm{vec}(\mathcal{A}_{\mathcal{T}})$.

Since the TT compression algorithm can proceed via sweeps of low rank decompositions of unfoldings for V_k (left to right) or of U_k (right to left), we can iterate Eq. (B.2) and Eq. (B.3) separating one core at a time. Let $\mathcal{G}_k$ denote the kth core of the TT decomposition of A, and $\mathsf{G}_k = \texttt{reshape}(\mathcal{G}_k, [r_{k-1}n_k, r_k])$. If we apply Eq. (B.2) $k-1$ times, we obtain

$$\mathrm{vec}(\mathcal{A}_{\mathcal{T}}) = (\mathsf{I}_{m_1} \otimes \mathsf{G}_1)(\mathsf{I}_{m_2} \otimes \mathsf{G}_2)\ldots(\mathsf{I}_{m_{k-1}} \otimes \mathsf{G}_{k-1})\mathrm{vec}(\mathsf{V}_{k-1}). \tag{B.4}$$

If we apply Eq. (B.3), we can write an explicit formula for the kth core $\mathcal{G}_k$

$$\mathrm{vec}(\mathcal{A}_{\mathcal{T}}) = (\mathsf{I}_{m_1} \otimes \mathsf{G}_1)\ldots(\mathsf{I}_{m_{k-1}} \otimes \mathsf{G}_{k-1})(\mathsf{G}^T_d \otimes \mathsf{I}_{p_d r_{k-1}})\ldots(\mathsf{G}^T_{k+1} \otimes \mathsf{I}_{p_{k+1} r_{k-1}})\mathrm{vec}(\mathcal{G}_k) \tag{B.5}$$

Following notation from [DS13a], we denote the product of the $d-1$ matrix factors in Eq. (B.5) as $\mathcal{P}_{\neq k}(\mathsf{A})$. Its columns form a reduced tensor basis generated by $\{\mathsf{G}_q\}_{q\neq k}$, and $\mathrm{vec}(\mathcal{G}_k)$ corresponds to the coefficients that reconstruct A. If a correction step (e.g using QR) is implemented to make $\{\mathsf{G}_q\}_{q<k}$ and $\{\mathsf{G}_q^T\}_{q>k}$ orthogonal, then $\mathcal{P}_{\neq k}(\mathsf{A})$ is orthogonal as well, and we have

$$\mathrm{vec}(\mathcal{G}_k) = (\mathsf{G}_{k+1}^T \otimes \mathsf{I}_{p_{k+1}r_{k-1}})^\star \ldots (\mathsf{I}_{m_1} \otimes \mathsf{G}_1)^\star \mathrm{vec}(\mathcal{A}_{\mathcal{T}}) = \mathcal{P}_{\neq k}(\mathsf{A})^\star \mathrm{vec}(\mathcal{A}_{\mathcal{T}}) \tag{B.6}$$

Finally, we note that from Eq. (B.6) an identity for $\mathrm{vec}(\mathsf{A})$ can be readily found, as there exists a permutation matrix $\Pi_{\mathcal{T}}$ such that $\mathrm{vec}(\mathsf{A}) = \Pi_{\mathcal{T}}\mathrm{vec}(\mathcal{A}_{\mathcal{T}})$.

Appendix C. Setup of local linear systems for TT inversion algorithms

The approximate inversion seeks to find X that satisfies $\mathsf{A}\mathsf{X} = \mathsf{I}_\mathsf{N}$ or $\mathsf{A}\mathsf{X} + \mathsf{X}\mathsf{A} = 2\mathsf{I}_\mathsf{N}$ and to extract a TT decomposition for X. Given a set of proposed cores $\{\mathcal{W}_k\}_{k=1}^d$ for X and fixing all but $\mathcal{W}_k$, Eq. (B.5) provides an explicit formula to interpret the TT decomposition as an expansion in an orthonormal basis, with elements that depend on $\{\mathcal{W}_j\}_{j\neq k}$ and coefficients $\mathrm{vec}(\mathcal{W}_k)$. That is, we can write $\mathrm{vec}(\mathsf{X}) = \mathcal{P}_{\neq k}(\mathsf{X})\mathrm{vec}(\mathcal{W}_k)$ where $\mathcal{P}_{\neq k}(\mathsf{X})$ is orthogonal. Eq. (4.1) then becomes

$$(\mathsf{I}_N \otimes \mathsf{A})\mathcal{P}_{\neq k}(\mathsf{X})\mathrm{vec}(\mathcal{W}_k) = \mathrm{vec}(\mathsf{I}_N). \tag{C.1}$$

From this one may readily notice that fixing all cores but $\mathcal{W}_k$ yields an overdetermined linear system. Applying $\mathcal{P}_{\neq k}(\mathsf{X})^\star$, we obtain the equivalent reduced system

$$\mathcal{P}_{\neq k}(\mathsf{X})^\star(\mathsf{I}_N \otimes \mathsf{A})\mathcal{P}_{\neq k}(\mathsf{X})\mathrm{vec}(\mathcal{W}_k) = \mathcal{P}_{\neq k}(\mathsf{X})^\star\mathrm{vec}(\mathsf{I}_N). \tag{C.2}$$

The matrix in the linear system above is of size $n_k r_{k-1} r_k \times n_k r_k r_{k-1}$. Eq. (C.2) allows us to solve for each core $\mathcal{W}_k$ by projecting X onto this reduced basis in which $\mathcal{W}_k$ is the only degree of freedom. We note that this necessarily leaves the size and rank of $\mathcal{W}_k$ fixed, and as a consequence, some strategy must be implemented to increase ranks and accelerate convergence towards the solution.

- In DMRG methods, this issue is resolved by contracting two cores $\mathcal{W}_k, \mathcal{W}_{k+1}$ at a time into a supercore
$$\mathcal{S}_k(\alpha_{k-1}, \overline{i_k i_{k+1}}, \alpha_{k+1}) = \mathcal{W}_k(\alpha_{k-1}, i_k, \alpha_k)\mathcal{W}_{k+1}(\alpha_k, i_{k+1}, \alpha_{k+1}), \tag{C.3}$$
and solving the corresponding reduced system for $\mathcal{S}_k$. The kth rank is now free (it was contracted in merging both cores) and it will be determined when newly computed $\mathcal{S}_k$ is split into cores.

- In AMEN-type methods [DS13a, DS13b], the residual R of this system is approximated in TT form, and then it is used in an enrichment step to expand the reduced basis and allow for ranks to increase.

We also note that even though we present these reduced systems explicitly, in both DMRG and AMEN methods a recursive formula is employed to construct them as they cycle through the cores of X. In fact, it is shown in [OD12] that if $\mathcal{G}_k(\alpha_{k-1}, \overline{i_k j_k}, \alpha_k)$ is the kth core of the TT decomposition of $(\mathsf{I}_N \otimes \mathsf{A})$, then we can write the left-hand side of Eq. (C.2) in the following form

$$\sum_{\alpha,\gamma,j} \Psi_{k-1}(\alpha_{k-1}, \beta_{k-1}, \gamma_{k-1})\mathcal{G}_k(\alpha_{k-1}, \overline{i_k j_k}, \alpha_k)\Phi_k(\alpha_k, \beta_k, \gamma_k)\mathcal{W}_k(\gamma_{k-1}, \overline{j_k p_k}, \gamma_k), \tag{C.4}$$

where Ψ_{k-1} is a function of $\{\mathcal{G}_j\}_{j=1}^{k-1}$ and $\{\mathcal{W}_j\}_{j=1}^{k-1}$, and Φ_k a function of $\{\mathcal{G}_j\}_{j=k+1}^d$ and $\{\mathcal{W}_j\}_{j=k+1}^d$. Moreover, recursive formulas ψ and ϕ exist such that

$$\Psi_k = \psi(\Psi_{k-1}, \mathcal{G}_k, \mathcal{W}_k) \quad \text{and} \quad \Phi_{k+1} = \phi(\Phi_{k-1}, \mathcal{G}_k, \mathcal{W}_k). \tag{C.5}$$

Finally, we note that Eq. (C.4) may be interpreted as a compressed form of the matrix in the reduced linear system with a 3-dimensional TT decomposition with cores Ψ, $\mathcal{G}$, and Φ. If this linear system is relatively small, the matrix in Eq. (C.2) may be formed to solve this system densely. Otherwise, it is preferable to use the fast TT matrix vector apply on the tensor decomposition in Eq. (C.4) to solve this system using an iterative algorithm such as GMRES or Bi-CGSTAB.

References

[AD14] Sivaram Ambikasaran and Eric Darve. “The Inverse Fast Multipole Method”. In: *arXiv preprint arXiv:1407.1572* (2014), pp. 1–25.

[BH86] J. Barnes and P. Hut. “A hierarchical $O(N \log N)$ force-calculation algorithm”. In: *Nature* 324 (1986), p. 4.

[Beb08] M. Bebendorf. *Hierarchical matrices*. Vol. 63. Lecture Notes in Computational Science and Engineering. A means to efficiently solve elliptic boundary value problems. Berlin: Springer-Verlag, 2008, pp. xvi+290.

[Beb05] Mario Bebendorf. “Hierarchical LU decomposition-based preconditioners for BEM”. In: *Computing (Vienna/New York)*. Vol. 74. 3. 2005, pp. 225–247.

[Ben+08] I. Benedetti, M. H. Aliabadi, and G. Davì. “A fast 3D dual boundary element method based on hierarchical matrices”. In: *International Journal of Solids and Structures* 45.7-8 (2008), pp. 2355–2376.

[Ben02] Michele Benzi. “Preconditioning Techniques for Large Linear Systems: A Survey”. In: *Journal of Computational Physics* 182.2 (Nov. 2002), pp. 418–477.

[Bey+91] Gregory Beylkin, Ronald Coifman, and Vladimir Rokhlin. “Fast wavelet transforms and numerical algorithms I”. In: *Communications on pure and applied mathematics* 44.2 (1991), pp. 141–183.

[Big+14] D Bigoni, Allan P Engsig-Karup, and Youssef M Marzouk. “Spectral tensor-train decomposition”. In: *arXiv preprint arXiv:1405.5713* (2014).

[Bör10] S. Börm. *Efficient numerical methods for non-local operators*. Vol. 14. EMS Tracts in Mathematics. $\mathscr{H}^2$-matrix compression, algorithms and analysis. European Mathematical Society (EMS), Zürich, 2010, pp. x+432.

[Bör+03] S. Börm, L. Grasedyck, and W. Hackbusch. “Hierarchical matrices”. In: *Lecture notes* 21 (2003).

[Car+03] B. Carpentieri, I. S. Duff, and L. Giraud. “A Class of Spectral Two-Level Preconditioners”. In: *SIAM Journal on Scientific Computing* 25.2 (2003), pp. 749–765.

[Car+05] B. Carpentieri, I. S. Duff, L. Giraud, and G. Sylvand. “Combining Fast Multipole Techniques and an Approximate Inverse Preconditioner for Large Electromagnetism Calculations”. In: *SIAM Journal on Scientific Computing* 27.3 (2005), pp. 774–792.

[Cha+97] Tony F. Chan, W. P. Tang, and W. L. Wan. “Wavelet sparse approximate inverse preconditioners”. In: *BIT Numerical Mathematics* 37.3 (Sept. 1997), pp. 644–660.

[Cha+06a] S. Chandrasekaran, P. Dewilde, M. Gu, W. Lyons, and T. Pals. “A fast solver for HSS representations via sparse matrices”. In: *SIAM Journal on Matrix Analysis and Applications* 29.1 (2006), pp. 67–81.

[Cha+06b] S. Chandrasekaran, M. Gu, and T. Pals. “A fast ULV decomposition solver for hierarchically semiseparable representations”. In: *SIAM Journal on Matrix Analysis and Applications* 28.3 (2006), pp. 603–622.

[Che+05] Hongwei Cheng, Zydrunas Gimbutas, Per-Gunnar Martinsson, and Vladimir Rokhlin. “On the compression of low rank matrices”. In: *SIAM Journal on Scientific Computing* 26.4 (2005), pp. 1389–1404.

[Cor+14] E. Corona, P.G. Martinsson, and D. Zorin. “An $O(N)$ direct solver for integral equations on the plane”. In: *Applied and Computational Harmonic Analysis* (2014).

[Cou+15] Pieter Coulier, Hadi Pouransari, and Eric Darve. “The inverse fast multipole method: using a fast approximate direct solver as a preconditioner for dense linear systems”. In: *arXiv preprint arXiv:1508.01835* (2015).

[DS13a] S. Dolgov and D.V. Savostyanov. “Alternating minimal energy methods for linear systems in higher dimensions”. In: *Part I: SPD systems, arXiv preprint* 1301 (2013).

[DS13b] S. Dolgov and D.V. Savostyanov. “Alternating minimal energy methods for linear systems in higher dimensions. Part II: Faster algorithm and application to nonsymmetric systems”. In: *arXiv preprint arXiv:1304.1222* (2013).

[Dol+12] S. Dolgov, B. Khoromskij, and D. Savostyanov. “Superfast Fourier transform using QTT approximation”. In: *Journal of Fourier Analysis and Applications* 18.5 (2012), pp. 915–953.

[Gil11] A. Gillman. “Fast direct solvers for elliptic partial differential equations”. PhD thesis. University of Colorado, 2011.

[Gil+12] A. Gillman, P.M. Young, and P.G. Martinsson. “A direct solver with $O(N)$ complexity for integral equations on one-dimensional domains”. In: *Frontiers of Mathematics in China* (2012), pp. 1–31.

[GG11] Z Gimbutas and L Greengard. *FMMLIB3D 1.2, FORTRAN libraries for fast multiple method in three dimensions*. http://www.cims.nyu.edu/cmcl/fmm3dlib/fmm3dlib. 2011.

[Gir+07] L. Giraud, S. Gratton, and E. Martin. “Incremental spectral preconditioners for sequences of linear systems”. In: *Applied Numerical Mathematics* 57.11-12 (2007), pp. 1164–1180.

[GVL12] Gene H Golub and Charles F Van Loan. *Matrix computations*. Vol. 3. JHU Press, 2012.

[GS02] IG Graham and IH Sloan. “Fully discrete spectral boundary integral methods for Helmholtz problems on smooth closed surfaces in $\mathbb{R}^3$”. In: *Numerische Mathematik* 92.2 (2002), pp. 289–323.

[Gra+96] Ananth Grama, Vipin Kumar, and Ahmed Sameh. “Parallel Hierarchical Solvers and Preconditioners for Boundary Element Methods”. In: *Proceedings of the 1996 ACM/IEEE Conference on Supercomputing* 20.1 (1996), pp. 337–358.

[Gra10] L. Grasedyck. “Hierarchical low rank approximation of tensors and multivariate functions”. In: *Lecture notes of the Zürich summer school on Sparse Tensor Discretizations of High-Dimensional Problems* (2010).

[Gra+13] L. Grasedyck, D. Kressner, and C. Tobler. “A literature survey of low-rank tensor approximation techniques”. In: *GAMM-Mitteilungen* 36.1 (2013), pp. 53–78.

[GR87] L. Greengard and V. Rokhlin. “A fast algorithm for particle simulations”. In: *Journal of Computational Physics* 73.2 (1987), pp. 325–348.

[GR97] L. Greengard and V. Rokhlin. “A new version of the fast multipole method for the Laplace equation in three dimensions”. In: *Acta numerica, 1997*. Vol. 6. Acta Numer. Cambridge: Cambridge Univ. Press, 1997, pp. 229–269.

[GL96] Leslie Greengard and June-Yub Lee. “A direct adaptive Poisson solver of arbitrary order accuracy”. In: *Journal of Computational Physics* 125.2 (1996), pp. 415–424.

[GH97] Marcus J. Grote and Thomas Huckle. “Parallel Preconditioning with Sparse Approximate Inverses”. In: *SIAM Journal on Scientific Computing* 18.3 (1997), pp. 838–853.

[GM10] Levent Gürel and Tahir Malas. *Iterative Near-Field Preconditioner for the Multilevel Fast Multipole Algorithm*. 2010.

[Hac11] W. Hackbusch. “Tensorisation of vectors and their efficient convolution”. In: *Numerische Mathematik* 119.3 (2011), pp. 465–488.

[HN89] W. Hackbusch and Z.P. Nowak. “On the fast matrix multiplication in the boundary element method by panel clustering”. In: *Numerische Mathematik* 54.4 (1989), pp. 463–491.

[Hac+05] W. Hackbusch, B. Khoromskij, and E. Tyrtyshnikov. “Hierarchical Kronecker tensor-product approximations”. In: *Journal of Numerical Mathematics jnma* 13.2 (2005), pp. 119–156.

[Hac+08] W. Hackbusch, B. Khoromskij, and E. Tyrtyshnikov. “Approximate iterations for structured matrices”. In: *Numerische Mathematik* 109.3 (2008), pp. 365–383.

[HY15] Kenneth L Ho and Lexing Ying. “Hierarchical interpolative factorization for elliptic operators: integral equations”. In: *Communications on Pure and Applied Mathematics* (2015).

[HG12] K.L. Ho and L. Greengard. “A fast direct solver for structured linear systems by recursive skeletonization”. In: *SIAM Journal on Scientific Computing* 34.5 (2012), pp. 2507–2532.

[KS15] V Kazeev and Ch Schwab. “Approximation of Singularities by Quantized-Tensor FEM”. In: (2015).

[KK12] V.A. Kazeev and B. Khoromskij. “Low-rank explicit QTT representation of the Laplace operator and its inverse”. In: *SIAM Journal on Matrix Analysis and Applications* 33.3 (2012), pp. 742–758.

[Kho11] B. Khoromskij. “$O(d \log n)$-quantics approximation of N-d tensors in high-dimensional numerical modeling”. In: *Constructive Approximation* 34.2 (2011), pp. 257–280.

[Kho+01] B. Khoromskij, S. Sauter, and A. Veit. “Fast quadrature techniques for retarded potentials based on TT/QTT tensor approximation”. In: *Computational Methods in Applied Mathematics Comput. Methods Appl. Math.* 11.3 (2001), pp. 342–362.

[Kre+89] Rainer Kress, V Maz’ya, and V Kozlov. *Linear integral equations*. Vol. 82. Springer, 1989.

[Mar+07] Per-Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert. “On interpolation and integration in finite-dimensional spaces of bounded functions”. In: *Communications in Applied Mathematics and Computational Science* 1.1 (2007), pp. 133–142.

[MR05] P.G. Martinsson and V. Rokhlin. “A fast direct solver for boundary integral equations in two dimensions”. In: *Journal of Computational Physics* 205.1 (2005), pp. 1–23.

[Nab+94] K. Nabors, T. Korsmeyer, and J. White. “Multipole-Accelerated Preconditioned Iterative Methods For Solving Three-Dimensional Mixed First And Second Kind Integral Equations”. In: *SIAM Journal on Scientific and Statistical Computing* 15.3 (1994), pp. 713–735.

[Nac+92] Noël M. Nachtigal, Satish C. Reddy, and Lloyd N. Trefethen. “How Fast are Nonsymmetric Matrix Iterations?” In: *SIAM Journal on Matrix Analysis and Applications* 13.3 (1992), pp. 778–795. DOI: 10.1137/0613049.

[Ols+06] V. Olshevsky, I.V. Oseledets, and E. Tyrtyshnikov. “Tensor properties of multilevel Toeplitz and related matrices”. In: *Linear algebra and its applications* 412.1 (2006), pp. 1–21.

[Ols+08] V. Olshevsky, I.V. Oseledets, and E. Tyrtyshnikov. “Superfast inversion of two-level Toeplitz matrices using Newton iteration and tensor-displacement structure”. In: *Recent Advances in Matrix and Operator Theory*. Springer, 2008, pp. 229–240.

[Ose09] I.V. Oseledets. “Tensors inside of matrices give logarithmic complexity”. In: *Preprint* 4 (2009).

[Ose10] I.V. Oseledets. “Approximation of $2^d \times 2^d$ Matrices Using Tensor Decomposition”. In: *SIAM Journal on Matrix Analysis and Applications* 31.4 (2010), pp. 2130–2145.

[Ose12] I.V. Oseledets. *TT-Toolbox 2.2*. 2012.

[OD12] I.V. Oseledets and S.V. Dolgov. “Solution of linear systems and matrix inversion in the TT-format”. In: *SIAM Journal on Scientific Computing* 34.5 (2012), A2718–A2739.

[OT10] I.V. Oseledets and E. Tyrtyshnikov. “TT-cross approximation for multidimensional arrays”. In: *Linear Algebra and its Applications* 432.1 (2010), pp. 70–88.

[OT11] I.V. Oseledets and E. Tyrtyshnikov. “Algebraic wavelet transform via quantics tensor train decomposition”. In: *SIAM Journal on Scientific Computing* 33.3 (2011), pp. 1315–1328.

[Ose+11] I.V. Oseledets, E. Tyrtyshnikov, and N. Zamarashkin. “Tensor-train ranks for matrices and their inverses”. In: *Comput. Methods Appl. Math.* 11.3 (2011), pp. 394–403.

[Pis+06] Davy Pissoort, Eric Michielssen, Dries Vande Ginste, and Femke Olyslager. “A rank-revealing preconditioner for the fast integral-equation-based characterization of electromagnetic crystal devices”. In: *Microwave and Optical Technology Letters* 48.4 (Apr. 2006), pp. 783–789.

[QB13] Bryan Quaife and George Biros. "On preconditioners for the Laplace double-layer in 2D". In: *arXiv preprint arXiv:1308.1937* 1 (Aug. 2013), pp. 1–26.

[Rok85] V. Rokhlin. "Rapid solution of integral equations of classical potential theory". In: *J. Comput. Phys.* 60.2 (1985), pp. 187–207.

[SS86] Y. Saad and M.H. Schultz. "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems". In: *SIAM Journal on Scientific and Statistical Computing* 7.3 (1986), pp. 856–869.

[SR94] P. Starr and V. Rokhlin. "On the numerical solution of two-point boundary value problems. II". In: *Comm. Pure Appl. Math.* 47.8 (1994), pp. 1117–1159. DOI: 10.1002/cpa.3160470806.

[TW96] Johannes Tausch and Jacob White. "Preconditioning first and second kind integral formulations of the capacitance problem". In: *Methods* 4 (1996), p. 96.

[TW97] Johannes Tausch and Jacob White. "Preconditioning and fast summation techniques for first-kind boundary integral equations". In: *IMACS International Symposium on Iterative Methods in Scientific Computation*. 1997.

[Tyr00] Eugene Tyrtyshnikov. "Incomplete cross approximation in the mosaic-skeleton method". In: *Computing* 64.4 (2000), pp. 367–380.

[Tyr10] Eugene Tyrtyshnikov. "Tensor ranks for the inversion of tensor-product binomials". In: *Journal of computational and applied mathematics* 234.11 (2010), pp. 3170–3174.

[VLP92] C.F. Van Loan and N. Pitsianis. *Approximation with Kronecker products*. Tech. rep. Cornell University, 1992.

[Vav92] Stephen A. Vavasis. "Preconditioning for Boundary Integral Equations". In: *SIAM Journal on Matrix Analysis and Applications* 13.3 (1992), pp. 905–925.

[Vor92] H.A. Van der Vorst. "Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems". In: *SIAM Journal on Scientific and Statistical Computing* 13.2 (1992), pp. 631–644.

[Wan+07] Yin Wang, Jeonghwa Lee, and Jun Zhang. "A short survey on preconditioning techniques for large-scale dense complex linear systems in electromagnetics". In: *International Journal of Computer Mathematics* 84.8 (2007), pp. 1211–1223.

[Wat15] A. J. Wathen. "Preconditioning". In: *Acta Numerica* 24.April (2015), pp. 329–376.

[Wil+09] Samuel Williams, Andrew Waterman, and David Patterson. "Roofline: an insightful visual performance model for multicore architectures". In: *Communications of the ACM* 52.4 (2009), pp. 65–76.

[Xia+10] J. Xia, S. Chandrasekaran, M. Gu, and X.S. Li. "Fast algorithms for hierarchically semiseparable matrices". In: *Numerical Linear Algebra with Applications* 17.6 (2010), pp. 953–976.

[Yin14] Lexing Ying. "Sparsifying preconditioner for the lippmann-schwinger equation". In: *Multiscale Modeling & Simulation* 13.2 (Jan. 2014), pp. 1–18.