HIERARCHICAL TUCKER LOW-RANK MATRICES: CONSTRUCTION AND MATRIX-VECTOR MULTIPLICATION*

YINGZHOU LI† AND JINGYU LIU‡

Abstract. In this paper, a hierarchical Tucker low-rank (HTLR) matrix is proposed to approximate non-oscillatory kernel functions in linear complexity. The HTLR matrix is based on the hierarchical matrix, with the low-rank blocks replaced by Tucker low-rank blocks. Using high-dimensional interpolation as well as tensor contractions, algorithms for the construction and matrix-vector multiplication of HTLR matrices are proposed admitting linear and quasi-linear complexities respectively. Numerical experiments demonstrate that the HTLR matrix performs well in both memory and runtime. Furthermore, the HTLR matrix can also be applied on quasi-uniform grids in addition to uniform grids, enhancing its versatility.

Key words. fast algorithms, $\mathcal{H}$-matrix, integral equations, Tucker decomposition

1. Introduction.

This paper considers the integral equation (IE) taking the form

$$a(\boldsymbol{x})u(\boldsymbol{x}) + \int_\Omega k(\boldsymbol{x},\boldsymbol{y})u(\boldsymbol{y})\mathrm{d}\boldsymbol{y} = f(\boldsymbol{x}), \quad x \in \Omega, \tag{1.1}$$

where $\Omega = [0,1]^d$ is the unit box in $\mathbb{R}^d$. Here the *kernel function* $k(\boldsymbol{x},\boldsymbol{y})$ as well as the function $a(\boldsymbol{x})$ in (1.1) are given. In some cases, the function $u(\boldsymbol{x})$ is known, and we would like to evaluate the integral equation (1.1) to obtain $f(\boldsymbol{x})$. This is known as the forward evaluation or the application. In other cases, the function $f(\boldsymbol{x})$ is known and $u(\boldsymbol{x})$ is unknown. We would like to solve the integral equation (1.1) to obtain $u(\boldsymbol{x})$. This is known as the backward evaluation or the inversion. Both forward and backward evaluations are of great importance in practice.

Discretizing (1.1) using typical approaches leads to a dense linear system

$$\boldsymbol{A}\boldsymbol{u} = \boldsymbol{f}, \tag{1.2}$$

where $\boldsymbol{A} \in \mathbb{R}^{N\times N}$ is a dense matrix and N is the number of discretization points. For instance, the Nyström scheme of (1.1) on a uniform grid with n points in each direction yields the following linear system of size $N = n^d$:

$$a(\boldsymbol{x}_i)u_i + \sum_{j=1}^{N} K_{i,j}h^d u_j = f(\boldsymbol{x}_i), \quad i = 1,\ldots,N, \tag{1.3}$$

in which $u_i \approx u(\boldsymbol{x}_i)$ approximates the value on the grid points. The matrix $\boldsymbol{K}$ can be viewed as a kernel matrix, with the exception of a modification on the diagonal. Its entries are defined as follows:

$$K_{i,j} \coloneqq \begin{cases} k(\boldsymbol{x}_i,\boldsymbol{x}_j), & j \neq i, \\ \int_{\Omega_i} k(\boldsymbol{x}_i,\boldsymbol{y})\mathrm{d}\boldsymbol{y}/h^d, & j = i, \end{cases} \tag{1.4}$$

*Submitted to the editors DATE.

Funding: This research was supported by the National Natural Science Foundation of China (NSFC) under grant numbers 12271109, the Science and Technology Commission of Shanghai Municipality (STCSM) under grant numbers 22TQ017 and 24DP2600100, and the Shanghai Institute for Mathematics and Interdisciplinary Sciences (SIMIS) under grant number SIMIS-ID-2024-(CN).

†School of Mathematical Sciences, Fudan University; Shanghai Key Laboratory for Contemporary Applied Mathematics, Fudan University(yingzhouli@fudan.edu.cn).

‡School of Mathematical Sciences, Fudan University(jyliu22@m.fudan.edu.cn).

where the integral is computed numerically. The product $K_{i,j}h^d$ is the approximated integral of $k(\boldsymbol{x}_i, \cdot)$ on a square domain Ω_j with width $h = 1/n$ and center $\boldsymbol{x}_j$. The discretization (1.3) can also be regarded as a collocation method using piecewise-constant basis functions whose supports are $\{\Omega_i\}$, except that the off-diagonal entries use an approximation of the integral.

1.1. Related Work. Typically, dense direct methods for solving (1.2) such as the LU factorization take the $\mathcal{O}(N^3)$ time complexity and $\mathcal{O}(N^2)$ storage complexity, which becomes prohibitive as N increases to moderately large values.

Fast algorithms have been designed to reduce both the time and storage complexity in addressing (1.2). The key insight is the observation that off-diagonal submatrices of $\boldsymbol{A}$ in (1.3) are numerically low-rank if the kernel function k is smooth and not highly oscillatory away from the diagonal. Such kernel functions are commonly encountered in practice, including the Green's function for elliptic PDEs and low frequency wave equations, as well as Gaussian kernels. This *low-rank property* brings the possibility to enhance both storage efficiency and computational performance for the matrix. This idea can be traced back to the *Barnes–Hut algorithm* [3] (also known as the tree-code) and *fast multipole methods* (FMM) [8, 15, 21, 49], which accelerate the matrix-vector multiplication for kernel matrices. Besides, Hackbusch and his collaborators have introduced *hierarchical matrices* [22, 25] (also known as $\mathcal{H}$-matrices and $\mathcal{H}^2$-matrices). These matrices achieve quasi-linear or linear complexity for most matrix algebraic operations including matrix-vector multiplication, matrix-matrix multiplication, matrix LU factorization, etc [6, 22, 23, 24, 25, 35]. Notably, $\mathcal{H}$-matrices and $\mathcal{H}^2$-matrices can be viewed as the algebraic version of the Barnes–Hut algorithm and FMM respectively.

Hierarchical block-separable (HBS) matrices (also referred to as *hierarchical semi-separable*, HSS, matrices) [40] represent another class of fast algorithms that accelerate kernel matrix operations. These matrices are closely related to $\mathcal{H}^2$-matrix under weak admissibility condition. Various HBS matrix factorization algorithms are proposed to achieve quasi-linear or linear complexity for matrix-vector multiplication and solving linear systems [9, 12, 17, 19, 48]. Another family of fast algorithms factorizes the matrix $\boldsymbol{A}$ as a product of block sparse lower and upper triangular matrices, where each block could be applied or inverted in $\mathcal{O}(1)$ complexity. Algorithms in this category include *recursive skeletonization factorization* [31, 41] and *hierarchical interpolative factorization* [32]. Two key techniques employed for algorithms in this family are the *interpolative decomposition* [10] and the *proxy surface* [39]. When the kernel function is smooth without singularity, such as in the case of Gaussian kernels, the low-rank approximation can be applied more aggressively, extending to those diagonal blocks as well [46, 47].

Though aforementioned fast algorithms achieve low complexity with respect to the matrix size, they still face the challgenge of the *curse of dimensionality* (CoD). When the problem dimension d increases, the prefactor in the complexity with respect to the size of each direction scales exponentially with d. To address this issue, tensor low-rank decompositions are commonly used techniques. In [34], the authors provide a comprehensive introduction to tensor decompositions such as the *CANDECOMP/PARAFAC* (CP) decomposition and the *Tucker decomposition*, both of which can be viewed as a generalization of the matrix singular value decomposition (SVD). Tensor decompositions could be computed by various numerical algorithms, including *high-order SVD* (HOSVD) [13], *higher-order orthogonal iteration* (HOOI) [14], *sequentially truncated higher order SVD* (STHOSVD) [45], and their randomized versions [30, 42]. Besides, *tensor train*, *tensor ring*, and *tensor network* are another

family of tensor decompositions widely used in practice, especially in computational physics and chemistry [11, 43].

Blending the structure of hierarchical decomposition and tensor decomposition has the potential to yield linear or quasi-linear fast algorithms with improved prefactors. *Hierarchical Kronecker tensor-product* (HKT) approximation [16, 26, 27, 28, 29] combines these two techniques, resulting om a quasi-linear representation of some high-dimensional integral and elliptic operators. Recently, *tensor butterfly algorithm* [33] has been proposed to represent high-dimensional oscillatory integral operators, which combines the *butterfly factorization* [36, 37, 38] and Tucker decomposition, further enhancing computational efficiency in high-dimensional contexts.

1.2. Contributions. In this paper, a *hierarchical Tucker low-rank* (HTLR) matrices is defined by replacing the low-rank blocks in an $\mathcal{H}$-matrix with *Tucker low-rank* (TLR) matrices. The Tucker decomposition enables TLR matrices to mitigate the CoD compared to conventional low-rank matrices, resulting in lower memory requirements and faster computational runtime. Our analysis demonstrates that only $\mathscr{O}(N)$ storage is needed to store an HTLR matrix of size N. Linear construction and quasi-linear application algorithms are proposed for HTLR matrices. In the construction algorithm, the TLR matrices are generated via multidimensional interpolations. A theoretical error bound is established for a specific class of kernel functions. The application algorithm uses tensor contractions for matrix-vector multiplication, achieving an improvement in the prefactor of the dominant complexity compared to $\mathcal{H}$-matrices. While the HTLR matrix is first introduced and discussed in the context of problems with a uniform grid discretization, we also present its application on a quasi-uniform grid discretization. Numerical experiments are conducted across various settings. The results not only support our complexity analysis but also offer compelling evidence for the efficiency of HTLR matrices.

1.3. Organization. The rest of the paper is organized as follows. Section 2 introduces the notations used throughout the paper and provides a brief review of $\mathcal{H}$-matrices. In Section 3, HTLR matrices are introduced for problems on uniform grids. We demonstrate that HTLR matrices only require linear storage complexity. The application of HTLR matrices on a quasi-uniform grid is also discussed there. The construction and application algorithms of HTLR matrices, as well as their complexity analysis, are described in detail in Section 4. Section 5 presents numerical results for two-dimensional and three-dimensional problems to demonstrate the performance of HTLR matrices. Finally, Section 6 concludes the paper with a discussion on future work.

2. Preliminaries.

2.1. Notations. For a positive integer N, the index set $\{1, \ldots, N\}$ is denoted by $[N]$. The notation $|\cdot|$ denotes either the number of elements in a set or the area of a domain. Vectors are denoted by boldface lowercase letters, e.g., $\boldsymbol{a}$, matrices are denoted by boldface capital letters, e.g., $\boldsymbol{A}$, and tensors (with order greater than 2) are denoted by boldface Euler script letters, e.g., $\boldsymbol{\mathcal{A}}$. We use MATLAB notations throughout the paper. For example, the i-th entry of a vector is denoted by $\boldsymbol{a}_i$ or $\boldsymbol{a}(i)$. The submatrix of a matrix $\boldsymbol{A}$ corresponding to row and column index sets τ and σ are denoted by $\boldsymbol{A}_{\tau,\sigma}$ or $\boldsymbol{A}(\tau, \sigma)$. A colon is used to indicate all entries of a dimension. For example, the j-th column of a matrix $\boldsymbol{A}$ is denoted by $\boldsymbol{A}_{:,j}$ or $\boldsymbol{A}(:, j)$. The conjugate transpose of a matrix $\boldsymbol{A}$ is denoted by $\boldsymbol{A}^*$. A matrix $\boldsymbol{U}$ is *orthonormal* if its columns form an orthonormal set, i.e., $\boldsymbol{U}^*\boldsymbol{U} = \boldsymbol{I}$.

We make use of multi-indices on tensor structures. For instance, when dealing with a tensor grid $\{(x_{1;i_1}, x_{2;i_2})\}_{i_1,i_2=1}^{n}$, we use $\boldsymbol{i} = (i_1, i_2)$ as the index for each point and represent it by $\boldsymbol{x}_{\boldsymbol{i}} = (x_{1;i_1}, x_{2;i_2})$. The one-to-one mapping between a multi-index and its linear order is given by $\boldsymbol{i} \leftrightarrow i_1 + (i_2 - 1)n$, and the grid is also denoted as $\{\boldsymbol{x}_{\boldsymbol{i}}\}_{\boldsymbol{i}=1}^{n^2}$. Additionally, we denote $|\boldsymbol{i}| \coloneqq i_1 + i_2$ and $\boldsymbol{i}! \coloneqq i_1 i_2$. These notations and their corresponding meanings can be extended analogously to d-dimensional cases.

Our notations and terminologies of tensors are consistent with [34]. The mode-ℓ product (contraction) of a tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ and a matrix $\boldsymbol{U} \in \mathbb{R}^{m \times n_\ell}$ is denoted as $\boldsymbol{\mathcal{A}} \times_\ell \boldsymbol{U} \in \mathbb{R}^{n_1 \times \cdots \times n_{\ell-1} \times m \times n_{\ell+1} \times \cdots \times n_d}$. The contraction of two tensors $\boldsymbol{\mathcal{A}}$ and $\boldsymbol{\mathcal{B}}$ is represented by $\boldsymbol{\mathcal{A}} \times_{\boldsymbol{a},\boldsymbol{b}} \boldsymbol{\mathcal{B}}$, where $\boldsymbol{a}$ and $\boldsymbol{b}$ are vectors specifying the dimensions in $\boldsymbol{\mathcal{A}}$ and $\boldsymbol{\mathcal{B}}$ to be contracted.

The *Tucker decomposition* of a tensor $\boldsymbol{\mathcal{A}}$ with *core tensor* $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{p_1 \times \cdots \times p_d}$ and *factored matrices* $\{\boldsymbol{U}_\ell \in \mathbb{R}^{n_\ell \times p_\ell}\}_{\ell=1}^{d}$ is defined as follows:

$$\boldsymbol{\mathcal{A}} \coloneqq \mathrm{Tucker}(\boldsymbol{\mathcal{G}}, \{\boldsymbol{U}_\ell\}_{\ell=1}^{d}) \coloneqq \boldsymbol{\mathcal{G}} \times_1 \boldsymbol{U}_1 \times_2 \cdots \times_d \boldsymbol{U}_d. \tag{2.1}$$

The vector $\boldsymbol{p} = (p_1, \ldots, p_d)$ is referred to as the *Tucker rank* or simply, the *rank* of $\boldsymbol{\mathcal{A}}$. When $p_1 = \cdots = p_d = p$, we also say p is the rank of $\boldsymbol{\mathcal{A}}$. In this article, we often consider $2d$-order tensors where the first and last d dimensions correspond to points in two different tensor grids. In such cases, the notation

$$\mathrm{Tucker}\big(\boldsymbol{\mathcal{G}}, \{\boldsymbol{U}_\ell\}_{\ell=1}^{d}, \{\boldsymbol{V}_\ell\}_{\ell=1}^{d}\big) \coloneqq \boldsymbol{\mathcal{G}} \times_1 \boldsymbol{U}_1 \times_2 \cdots \times_d \boldsymbol{U}_d \times_{d+1} \boldsymbol{V}_1 \times_{d+2} \cdots \times_{2d} \boldsymbol{V}_d$$

is used to represent the Tucker decomposition.

It is often advantageous to enforce the factored matrices to be orthonormal, which can be satisfied by most Tucker decomposition algorithms, such as those presented in [13, 45]. In particular, given a Tucker computation (2.1), this can be achieved through a series QR factorizations on each factored matrix, followed by a modification to the core tensor. When QR factorization with column pivoting (QRCP) is adapted, we can further compress the Tucker rank. Assuming that all $p_\ell = p$ and $n_\ell = n$, then the total complexity of the orthogonalization is $\mathcal{O}(dp^2 n + dp^{d+1})$.

2.2. Hierarchical Matrices. The definition of $\mathcal{H}$-matrices is based on the concept of cluster tree, a tree encoding the partition information of the grid points.

Definition 2.1 (Cluster tree). *Let I be the index set associated with the grid points from the discretization of IE* (1.1)*. A tree T_I is said to be a cluster tree corresponding to I if the following conditions hold:*

1. *Every node in T_I is a subset of I.*
2. *The root of T_I is I.*
3. *Each node $\tau \in \mathsf{T}_I$ of T_I is not empty.*
4. *For every $\tau \in \mathsf{T}_I$, let* $\mathrm{children}(\tau)$ *be the set consisting of its children. Then, either* $\mathrm{children}(\tau) = \emptyset$ *(τ is a leaf node) or it forms a partition of τ, i.e.,*

$$\tau = \bigsqcup_{\tau' \in \mathrm{children}(\tau)} \tau'.$$

Here we use $\sqcup$ to denote the union of pairwise disjoint sets.

Typically, each point $\boldsymbol{x}_i$ is associated with a computational domain Ω_i, which can be viewed as the support of its corresponding piecewise-constant function and satisfies the non-overlapping condition $|\Omega_i \cap \Omega_j| = 0$ for $j \neq i$. It can be obtained from the Voronoi cells [2] associated with the grid points. In particular, when consdiering a

uniform grid, Ω_i is a square domain centered at $\boldsymbol{x}_i$. For a indexset τ associated with the points $\{\boldsymbol{x}_i\}_{i\in\tau}$, we define $\Omega_\tau := \cup_{i\in\tau}\Omega_i$.

When Ω is a unit box $[0,1]^d$ and the points are obtained from a uniform grid, the cluster tree can be constructed using a recursive 2^d uniform partition. In the construction of a cluster tree, a stopping criterion is typically imposed to prevent the successive partition, ensuring that the number of indices in a leaf node is neither too small nor exceeds a user-specified value, denoted as N_0. Specifically, when $|\tau| \le N_0$, we assign τ be the leaf node and halt further partition on it.

EXAMPLE 2.1 (A cluster tree on 2D uniform grid). *Suppose $\Omega = [0,1]^2$ and I is associated with discretization points given by*

$$\boldsymbol{x_i} = \left((i_1-1)h + \frac{h}{2}, (i_2-1)h + \frac{h}{2}\right),\ \boldsymbol{i} = (i_1,i_2),\ 1 \le i_1, i_2 \le n,$$

where $n \in \mathbb{N}_+$ and $h = 1/n$. Consequently, the corresponding computational domain for each point is defined as $\Omega_{\boldsymbol{i}} = [(i_1-1)h, i_1h] \times [(i_2-1)h, i_2h]$. The cluster tree T_I is a quadtree and its leaf nodes are shown in Figure 2.1.

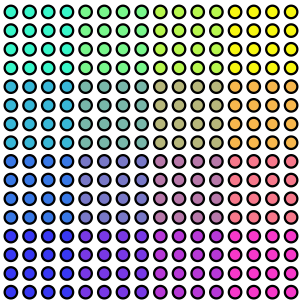


Fig. 2.1: Leaf nodes in the cluster tree in Example 2.1. Here $n = 16$, $N = n^2 = 256$ and $N_0 = 16$. Different colors represent different leaf nodes.

The structure of $\mathcal{H}$-matrices is determined by its admissibility condition, a criterion for whether the interaction matrix of two domains can be regarded as low-rank. There are typically two types of admissibility conditions: weak admissibility condition and strong admissibility condition. Figure 2.2 demonstrates them respectively.

DEFINITION 2.2 (Weak admissibility). *Two nodes τ and σ in T_I are weakly admissible if Ω_τ and Ω_σ are non-overlapping.*

DEFINITION 2.3 (Strong admissibility). *Two nodes τ and σ in T_I are strongly admissible if*

$$\max\{\operatorname{diam}(\Omega_\tau), \operatorname{diam}(\Omega_\sigma)\} \le \eta \operatorname{dist}(\Omega_\tau, \Omega_\sigma),$$

where $\operatorname{diam}(\Omega_\tau) := \max_{\boldsymbol{x},\boldsymbol{y}\in\Omega_\tau} \|\boldsymbol{x}-\boldsymbol{y}\|$ is the diameter of Ω_τ and $\operatorname{dist}(\Omega_\tau,\Omega_\sigma) := \min_{\boldsymbol{x}\in\Omega_\tau,\boldsymbol{y}\in\Omega_\sigma} \|\boldsymbol{x}-\boldsymbol{y}\|$ is the distance between Ω_τ and Ω_σ. Here $\eta > 0$ is a predefined hyperparameter. In this case Ω_τ and Ω_σ are said to be well-separated.

When the kernel function in IE (1.1) is smooth except the diagonal and not highly oscillatory, such as the Green's function of elliptic PDEs, the main distinction between

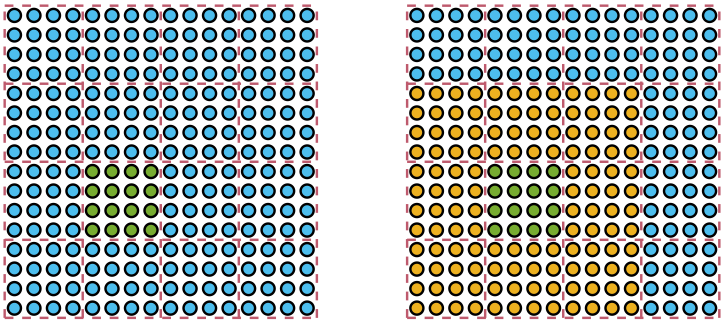


Fig. 2.2: Weak admissibility (left) and strong admissibility with parameter $\eta = \sqrt{2}$ (right) in Example 2.1. The grid points we consider is colored by green. The admissible grid points with respect to them are colored by blue while the inadmissible ones are colored by yellow. The dashed domain is the computational domain Ω_τ for each index set τ.

weak and strong admissibility lies in whether the rank of the interaction matrix associated with two admissible nodes can be bounded by a constant independent of the matrix size. Weak admissibility provides a straightforward hierarchical structure, sometimes referred to as *hierarchical off-diagonal low-rank* (HODLR) matrices [1] since all off-diagonal blocks of the matrices are regarded as low-rank. However, the rank of these low-rank matrices may increase mildly with the matrix size [19, 31]. In contrast, strong admissibility ensures a numerically constant rank but results in a more complex $\mathcal{H}$-matrix structure [6, 25].

Given a cluster tree T_I and an admissibility condition, the block cluster tree, defined as follows, provides a partition of $I \times I$ and specifies which block possesses a low-rank representation. The $\mathcal{H}$-matrix is then defined using the block cluster tree by compressing the submatrices of admissible leafs into low-rank matrices.

DEFINITION 2.4 (Block cluster tree). *Suppose T_I is a cluster tree associated with the index set I. The block cluster tree $\mathsf{T}_{I\times I}$, associated with T_I and an admissibility condition, is a tree whose nodes are subsets of $I \times I$, which is constructed using the following procedure starting from the root node $(\tau, \sigma) = (I, I)$:*

- *If τ and σ are admissible, then (τ, σ) is an admissible leaf node.*
- *If* $\mathrm{children}(\tau) = \emptyset$ *or* $\mathrm{children}(\sigma) = \emptyset$*, then (τ, σ) is an inadmissible leaf node.*
- *If* $\mathrm{children}(\tau) \neq \emptyset$ *and* $\mathrm{children}(\sigma) \neq \emptyset$*, then (τ, σ) is a non-leaf node with children nodes given by $(\tau', \sigma') \in \mathrm{children}(\tau) \times \mathrm{children}(\sigma)$.*

Particularly, it can be directly verified that a block cluster tree is also a cluster tree.

DEFINITION 2.5 (Hierarchical matrix, $\mathcal{H}$-matrix, [5]). *Let I be an index set of the grid points from the discretization of the IE (1.1) and $\mathsf{T}_{I\times I}$ be a block cluster tree. An $\mathcal{H}$-matrix of rank r, denoted by $\boldsymbol{A}^{\mathcal{H}}$, is a matrix on $\mathsf{T}_{I\times I}$ such that for every leaf $(\tau, \sigma) \in \mathsf{T}_{I\times I}$, the following conditions hold:*

- *If (τ, σ) is an admissible leaf node, then $\boldsymbol{A}^{\mathcal{H}}_{\tau,\sigma}$ is a low-rank matrix with rank at most r, i.e., $\boldsymbol{A}^{\mathcal{H}}_{\tau,\sigma} = \boldsymbol{U}\boldsymbol{G}\boldsymbol{V}^*$ for some $\boldsymbol{U} \in \mathbb{R}^{|\tau|\times r}$, $\boldsymbol{V} \in \mathbb{R}^{|\sigma|\times r}$ and $\boldsymbol{G} \in \mathbb{R}^{r\times r}$.*
- *If (τ, σ) is an inadmissible leaf node, then $\boldsymbol{A}^{\mathcal{H}}_{\tau,\sigma} \in \mathbb{R}^{|\tau|\times|\sigma|}$ is a dense matrix.*
- *If (τ, σ) is a non-leaf node, then $\boldsymbol{A}^{\mathcal{H}}_{\tau,\sigma}$ is a block matrix with blocks $\boldsymbol{A}^{\mathcal{H}}_{\tau',\sigma'}$, where $(\tau', \sigma') \in \mathrm{children}(\tau, \sigma)$.*

The structures of corresponding $\mathcal{H}$-matrices under weak and strong admissibility (with $\eta = \sqrt{2}$) of Example 2.1 are shown in Figure 2.3.

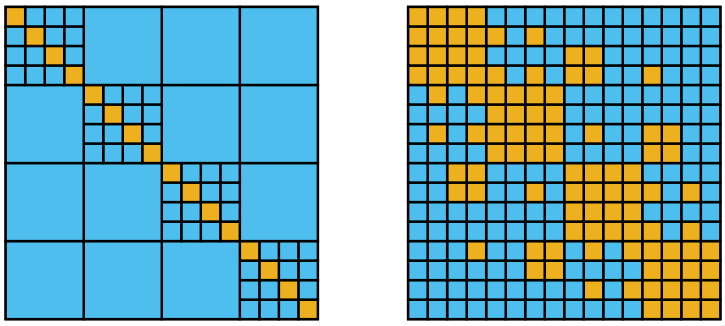


Fig. 2.3: $\mathcal{H}$-matrices corresponding to Example 2.1 under weak admissibility (left) and strong admissibility with parameter $\eta = \sqrt{2}$ (right) where $n = 16$, $N = n^2 = 256$ and $N_0 = 4^2$. The yellow submatrices are dense while the blue ones are low-rank. The weak admissibility implies that there is only one dense submatrix (the diagonal block) in each row block. On the other hand, under strong admissibility, each row block has at most 9 dense submatrices.

Compared to the typical $\mathcal{O}(N^2)$ storage complexity of a complete dense matrix, the storage cost of an $\mathcal{H}$-matrix is significantly lower, specifically $\mathcal{O}(rN \log N)$ [18, 22]. However, as we will demonstrate in Section 3.1, for a given accuracy, the upper bound of the desired rank r in $\mathcal{H}$-matrices typically grows exponentially with respect to the dimension d, taking the form $r = p^d$. This exponential dependence can lead to storage challgenges for moderately large d.

3. Hierarchical Tucker Low-Rank Matrices. When considering interaction matrices between two different tensor grids, Tucker decomposition is more appropriate for the low-rank compression. By combing this idea with the $\mathcal{H}$-matrix framework, in this section we propose the HTLR matrix, which achieves linear storage complexity. Additionally, we illustrate how the HTLR matrix can be effectively applied to a quasi-uniform grid.

3.1. Tucker Decomposition from High-Dimensional Interpolation. Interpolation is a widely-used technique for obtaining low-rank representations of kernel matrices [7, 15]. We first review the high-dimensional interpolation on a regular domain and then apply this method to construct the Tucker decomposition of interaction matrices between two tensor grids.

We begin with the one-dimensional case. Suppose $f\colon [a,b] \to \mathbb{R}$ is a function to be interpolated on p Chebyshev points given by

$$\xi_t = \frac{b-a}{2} \cos\left(\frac{(2t-1)\pi}{2p}\right) + \frac{b+a}{2}, \quad 1 \le t \le p.$$

The interpolant of f is defined as $\mathscr{J}_{[a,b]}[f](x) \coloneqq \sum_{t=1}^{p} f(\xi_t) L_{[a,b];t}(x)$ where

$$L_{[a,b];t}(x) \coloneqq \prod_{j=1,\ j\neq t}^{p} \frac{x-\xi_t}{\xi_j - \xi_t}$$

is the Lagrange interpolation function on the interval $[a, b]$.

In the d-dimensional case, suppose f is a function defined on the box $B = \prod_{\ell=1}^{d}[a_\ell, b_\ell]$, the interpolant of f using Chebyshev tensor points is given by

$$\mathscr{J}_B[f](\boldsymbol{x}) \coloneqq \sum_{\boldsymbol{t}\in[p]^d} f(\boldsymbol{\xi}_{\boldsymbol{t}}) L_{B;\boldsymbol{t}}(\boldsymbol{x}), \tag{3.1}$$

where $\boldsymbol{\xi}_{\boldsymbol{t}} = (\xi_{1;t_1}, \ldots, \xi_{d;t_d})$ is the interpolation point in B indexed by the multi-index $\boldsymbol{t} = (t_1, \ldots, t_d)$ and p is the number of interpolation points along each dimension. The Lagrange interpolation function $L_{B;\boldsymbol{t}}(\boldsymbol{x})$ is defined as the product of Lagrange interpolation functions associated the interval on each dimension, i.e., $L_{B;\boldsymbol{t}}(\boldsymbol{x}) \coloneqq \prod_{\ell=1}^{d} L_{[a_\ell,b_\ell];t_\ell}(x_\ell)$. The upper bound of the approximation error is given by the following lemma.

LEMMA 3.1 (Approximate error of the d-dimensional interpolation, [23] Lemma B.7). *Suppose f is a function on $B = \prod_{\ell=1}^{d}[a_\ell, b_\ell]$ which has $(p+1)$-th continuous derivatives, then for the Chebyshev interpolation* (3.1)*, we have*

$$\|\mathscr{J}_B[f] - f\|_{\infty,B} \leq 2\Lambda_p^{d-1} \sum_{\ell=1}^{d} \left(\frac{b_\ell - a_\ell}{4}\right)^{p+1} \frac{\|\partial_\ell^{p+1} f\|_{\infty,B}}{(p+1)!}, \tag{3.2}$$

where Λ_p is the Lebesgue constant and satisfies the estimate $\Lambda_p \sim 2\log(p)/\pi$ (See [44], Theorem 15.2).

Let $k(\boldsymbol{x}, \boldsymbol{y})$ be the kernel function and $B_\tau = \prod_{\ell=1}^{d}[a_\ell, b_\ell]$ and $B_\sigma = \prod_{\ell=1}^{d}[c_\ell, d_\ell]$ be two disjoint boxes in $\mathbb{R}^d$. By applying (3.1) twice on $\boldsymbol{x}$ and $\boldsymbol{y}$, we obtain

$$k(\boldsymbol{x}, \boldsymbol{y}) \approx \sum_{\boldsymbol{t}\in[p]^d} \sum_{\boldsymbol{s}\in[p]^d} k(\boldsymbol{\xi}_{\boldsymbol{t}}, \boldsymbol{\eta}_{\boldsymbol{s}}) L_{B_\tau;\boldsymbol{t}}(\boldsymbol{x}) L_{B_\sigma;\boldsymbol{s}}(\boldsymbol{y}). \tag{3.3}$$

Suppose $\{\boldsymbol{x}_{\boldsymbol{i}} = (x_{1;i_1}, \ldots, x_{d;i_d})\} \subset B_\tau$ and $\{\boldsymbol{y}_{\boldsymbol{j}} = (y_{1;j_1}, \ldots, y_{d;j_d})\} \subset B_\sigma$ are two set of points, each containing n^d points. The interaction matrix $\boldsymbol{K} \in \mathbb{R}^{n^d \times n^d}$ of them is defined by $\boldsymbol{K}(\boldsymbol{i}, \boldsymbol{j}) = k(\boldsymbol{x}_{\boldsymbol{i}}, \boldsymbol{y}_{\boldsymbol{j}})$. Utilizing the interpolation scheme (3.3), each entry $\boldsymbol{K}(\boldsymbol{i}, \boldsymbol{j})$ can be approximated by

$$\boldsymbol{K}(\boldsymbol{i}, \boldsymbol{j}) \approx \sum_{\boldsymbol{t}\in[p]^d} \sum_{\boldsymbol{s}\in[p]^d} k(\boldsymbol{\xi}_{\boldsymbol{t}}, \boldsymbol{\eta}_{\boldsymbol{s}}) L_{B_\tau;\boldsymbol{t}}(\boldsymbol{x}_{\boldsymbol{i}}) L_{B_\sigma;\boldsymbol{s}}(\boldsymbol{y}_{\boldsymbol{j}}). \tag{3.4}$$

By exploiting the tensor form of the points as well as the Lagrange interpolation functions, we denote

$$\begin{aligned} \boldsymbol{U}_\ell(i_\ell, \mu_\ell) &= L_{[a_\ell,b_\ell];\mu_\ell}(x_{\ell;i_\ell}), \quad \boldsymbol{U}_\ell \in \mathbb{R}^{n\times p}, \quad 1 \leq \ell \leq d, \\ \boldsymbol{V}_\ell(j_\ell, \nu_\ell) &= L_{[c_\ell,d_\ell];\nu_k}(y_{\ell;j_\ell}), \quad \boldsymbol{V}_\ell \in \mathbb{R}^{n\times p}, \quad 1 \leq \ell \leq d, \\ \boldsymbol{G}(\boldsymbol{t}, \boldsymbol{s}) &= k(\boldsymbol{\xi}_{\boldsymbol{t}}, \boldsymbol{\eta}_{\boldsymbol{s}}), \quad \boldsymbol{G} \in \mathbb{R}^{p^d \times p^d}. \end{aligned} \tag{3.5}$$

Substituting these into (3.4), we can reformulate it as

$$\boldsymbol{K} \approx \boldsymbol{U}\boldsymbol{G}\boldsymbol{V}^*, \tag{3.6}$$

where $\boldsymbol{U} = \boldsymbol{U}_d \otimes \cdots \otimes \boldsymbol{U}_1 \in \mathbb{R}^{n^d \times p^d}$ and $\boldsymbol{V} = \boldsymbol{V}_d \otimes \cdots \otimes \boldsymbol{V}_1 \in \mathbb{R}^{n^d \times p^d}$. The decomposition (3.6) serves as the low-rank representation of admissible leaf nodes in

$\mathcal{H}$-matrices [22, 23], requiring a storage of $2n^d p^d + p^{2d}$. Typically, the matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ are required to be orthonormal, which can be achieved through QR factorizations on them, along with a modification of $\boldsymbol{G}$. The overall complexity of this process is $\mathscr{O}(p^{2d}n^d + p^{3d})$. The matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ are called *basis matrices* and $\boldsymbol{G}$ is called the *core matrix* of the low-rank decomposition.

However, because of the inherent tensor structure of points, it is more advantageous to represent (3.4) using a Tucker low-rank decomposition, rather than constructing $\boldsymbol{U}$ and $\boldsymbol{V}$ explicitly. Let $\boldsymbol{\mathcal{K}}$ and $\boldsymbol{\mathcal{G}}$ are $2d$-order tensors defined by

$$\boldsymbol{\mathcal{K}}(i_1, \dots, i_d, j_1, \dots, j_d) = \boldsymbol{K}(\boldsymbol{i}, \boldsymbol{j}), \quad \boldsymbol{\mathcal{G}}(t_1, \dots, t_d, s_1, \dots, s_d) = \boldsymbol{G}(\boldsymbol{t}, \boldsymbol{s}).$$

Using (3.5), we can express (3.4) as the following Tucker decomposition:

$$\boldsymbol{\mathcal{K}} \approx \mathrm{Tucker}\big(\boldsymbol{\mathcal{G}}, \{\boldsymbol{U}_\ell\}_{\ell=1}^d, \{\boldsymbol{V}_\ell\}_{\ell=1}^d\big). \tag{3.7}$$

Compared with (3.6), the Tucker low-rank structure (3.7) maintains the same accuracy while revealing the essense of interpolation on tensor grids. It is also more data-efficient, requiring only $2dnp + p^{2d}$ storage. If we ignore the common cost p^{2d} on the core part ($\boldsymbol{G}$ or $\boldsymbol{\mathcal{G}}$), the Tucker low-rank matrix (3.7) exhibits greater effectiveness, as the rest complexity, $2dnp$, scales linearly with the dimension d. To meet the requirement of orthonormal factored matrices, we perform an additional orthogonalization step after the interpolation, as discussed in the end of Section 2.1. A detailed algorithm is postponed till Section 4.1. The overall complexity is $\mathscr{O}(dp^2 n + dp^{2d+1})$. Notably, the dominant term, $dp^2 n$, exhibits linear dependence on d.

Remark 3.2. This paper frequently treats a matrix as a tensor, or vice versa. To maintain conciseness, we adopt the agreement that when dealing with two tensor grids, the interaction can be represented as either a matrix or a $2d$-order tensor. The relationship between the two is given by $\boldsymbol{\mathcal{K}}(i_1, \dots, i_d, j_1, \dots, j_d) = \boldsymbol{K}(\boldsymbol{i}, \boldsymbol{j})$. At times, we may use matrix notation and tensor notation interchangeably to simplify our illustrations, and this usage should be correctly inferred from the context.

We end this section with an error estimate for the interpolation (3.3), which provides an estimate of the numerical rank of the Tucker decomposition for a specific class of kernel functions. One important property of them is the *asymptotical smoothness*.

DEFINITION 3.3 (Asympotically smoothness, [5]). *A kernel function $k\colon \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is said to be asymptotically smooth if there exists constants $C_{\mathrm{as}}, \gamma > 0$ depending only on k satisfying*

$$|\partial_{\boldsymbol{x}}^{\boldsymbol{\alpha}} \partial_{\boldsymbol{y}}^{\boldsymbol{\beta}} k(\boldsymbol{x}, \boldsymbol{y})| \le C_{\mathrm{as}}\big(\boldsymbol{\alpha} + \boldsymbol{\beta}\big)! \gamma^{|\boldsymbol{\alpha}+\boldsymbol{\beta}|} \frac{|k(\boldsymbol{x}, \boldsymbol{y})|}{\|\boldsymbol{x} - \boldsymbol{y}\|^{|\boldsymbol{\alpha}|+|\boldsymbol{\beta}|}}, \quad \forall \boldsymbol{x} \neq \boldsymbol{y},\ \boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathbb{N}^d. \tag{3.8}$$

where $\partial_{\boldsymbol{x}}^{\boldsymbol{\alpha}} \coloneqq \partial_{x_1}^{\alpha_1} \cdots \partial_{x_d}^{\alpha_d}$ and $\partial_{\boldsymbol{y}}^{\boldsymbol{\beta}} \coloneqq \partial_{y_1}^{\beta_1} \cdots \partial_{y_d}^{\beta_d}$ denote the multi-dimensional partial differential operators.

For example, it could be verified that kernels $k(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|^{-a}$ and $k(\boldsymbol{x}, \boldsymbol{y}) = \log(\|\boldsymbol{x} - \boldsymbol{y}\|)$ are asymptotically smooth [23].

THEOREM 3.4 (Interpolation error for asymptotically smooth kernels). *Suppose k is an asymptotically smooth kernel defined on two boxes $B_\tau = \prod_{\ell=1}^d [a_\ell, b_\ell]$ and $B_\sigma = \prod_{\ell=1}^d [c_\ell, d_\ell]$ in $\mathbb{R}^d$. If $\mathrm{dist}(B_\tau, B_\sigma) \ge \eta \max\{\mathrm{diam}(B_\tau), \mathrm{diam}(B_\sigma)\}$, then the approximation error $r(\boldsymbol{x}, \boldsymbol{y})$ of the Chebyshev interpolation* (3.3)*, defined by*

$$r(\boldsymbol{x}, \boldsymbol{y}) = \sum_{\boldsymbol{t}, \boldsymbol{s} \in [p]^d} k(\boldsymbol{\xi}_{\boldsymbol{t}}, \boldsymbol{\eta}_{\boldsymbol{s}}) L_{B_\tau; \boldsymbol{t}}(\boldsymbol{x}) L_{B_\sigma; \boldsymbol{s}}(\boldsymbol{y}) - k(\boldsymbol{x}, \boldsymbol{y}),$$

has the following estimate

$$\|r(\boldsymbol{x},\boldsymbol{y})\|_{\infty,B_\tau\times B_\sigma} \leq \frac{4C_{\mathrm{as}}\gamma^{p+1}\Lambda_p^{2d-1}d}{(4\eta)^{p+1}}\|k(\boldsymbol{x},\boldsymbol{y})\|_{\infty,B_\tau\times B_\sigma}, \tag{3.9}$$

where C_{as} and γ are the constants in (3.8) *and Λ_p is the Lebesgue constant in* (3.2).

Proof. The desired estimate follows directly from Lemma 3.1 and the property outlined in (3.8). ∎

From the estimate of the Lebesgue constant Λ_p, for a fixed dimension d, the growth rate of the numerator in (3.9) with respect to p can be considered negligible compared to the denominator when $4\eta > \gamma$ (a condition that is usually satisfied). Consequently, Theorem 3.4 demonstrates that for an asymptotically smooth kernel, if B_τ and B_σ are well-separated, the relative approximate error of Chebyshev interpolation decreases nearly exponentially as the interpolation order p increases. Additionally, we deduce from (3.9) that in this case, the numerical rank of the interaction matrix remains independent of its size.

3.2. Hierarchical Tucker Low-Rank Matrices. Motived by the Tucker low-rank structure discussed in Section 3.1, we propose the definition for hierarchical Tucker low-rank matrices. Since our previous discussion relies on the tensor structure of the grid, we focus on the discretization (1.3) and require the cluster tree and block cluster tree to inherit the "tensor property" as well.

Definition 3.5 (Tensor node cluster tree). *Let $I = \{\boldsymbol{i}\colon \boldsymbol{i} \in [n]^d\}$ be the index set associated with tensor grid points from the discretization* (1.3)*. A cluster tree T_I is called a tensor node cluster tree if every node $\tau \in \mathsf{T}_I$ can be expressed in the form $\tau = \tau_1 \times \cdots \times \tau_d$, where each τ_ℓ is the index set corresponding to the ℓ-th dimension.*

Definition 3.6 (Tensor node block cluster tree). *Suppose T_I is a tensor node cluster tree with root index I. The tensor node block cluster tree is defined as the block cluster tree $\mathsf{T}_{I\times I}$ corresponding to T_I and an admissibility condition.*

One main property of the tensor node cluster tree is that for every node, the corresponding points exhibit a tensor structure and its computational domain is a d-dimensional box. Therefore, the submatrices can be approximated using Tucker low-rank decomposition.

Definition 3.7 (Hierarchical Tucker low-rank matrices, HTLR matrices). *Suppose $\mathsf{T}_{I\times I}$ is the tensor node block cluster tree with root I. We define a matrix $\boldsymbol{A}^{\mathrm{HTLR}}$ as an HTLR matrix (on $\mathsf{T}_{I\times I}$) of rank p if for every leaf $(\tau,\sigma) \in \mathsf{T}_{I\times I}$, the following conditions hold:*

- *If (τ,σ) is an admissible leaf, then $\boldsymbol{A}^{\mathrm{HTLR}}_{\tau,\sigma}$ is a Tucker low-rank (TLR) matrix of rank at most p, i.e., $\boldsymbol{A}^{\mathrm{HTLR}}_{\tau,\sigma} = \mathrm{Tucker}\big(\boldsymbol{\mathcal{G}}, \{\boldsymbol{U}_\ell\}_{\ell=1}^d, \{\boldsymbol{V}_\ell\}_{\ell=1}^d\big)$ for some $\boldsymbol{U}_\ell \in \mathbb{R}^{|\tau_\ell|\times p}$, $\boldsymbol{V}_\ell \in \mathbb{R}^{|\sigma_\ell|\times p}$ and $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{p\times\cdots\times p}$.*
- *If (τ,σ) is an inadmissible leaf, then $\boldsymbol{A}^{\mathrm{HTLR}}_{\tau,\sigma}$ is a dense matrix.*
- *If (τ,σ) is a non-leaf node, then $\boldsymbol{A}^{\mathrm{HTLR}}_{\tau,\sigma}$ is a block matrix with blocks $\boldsymbol{A}^{\mathrm{HTLR}}_{\tau',\sigma'}$, where $(\tau',\sigma') \in \mathrm{children}(\tau,\sigma)$.*

From the relationship (3.6) and (3.7), the conventional low-rank representation of rank p^d achieves the same accuracy as the TLR representation of rank p when both are derived from high-dimensional interpolation. Using the storage cost of TLR matrices discussed in Section 3.1, the storage complexity for an HTLR matrix is linear, which is stated in the following proposition.

PROPOSITION 3.8 (Storage of HTLR matrices under weak admissibility). *For a fixed dimension $d \geq 2$ and $N_0 = n_0^d$. Let $I = \{\boldsymbol{i} \colon \boldsymbol{i} \in [n]^d\}$ be the index set corresponding to the tensor grid points from the discretization* (1.3) *and $N = n^d$. Suppose the tensor node cluster tree T_I is constructed by a 2^d partition whose every leaf node contains fewer than N_0 points and $\mathsf{T}_{I\times I}$ is the corresponding tensor node block cluster tree under weak admissibility condition. If $\boldsymbol{A}^{\mathrm{HTLR}}$ is an HTLR matrix of rank p and $p \leq n_0 \leq 2p$, then the storage required for $\boldsymbol{A}^{\mathrm{HTLR}}$ is $\mathscr{O}(p^d N)$, where the prefactor depends only on d.*[1]

Proof. Withou loss of generality, assume $N = n^d = 2^{dL} n_0^d$, where L is the maximum level in T_I. For each level $0 \leq \ell \leq L$, note that the matrix size corresponding to each node, the number of nodes, inadmissible and admissible blocks of a node are bounded by $N/2^{d\ell}$, $2^{d\ell}$ and 1, $2^d - 1$ respectively. Therefore, the storage of an HTLR matrix of size N, denoted by $S^{\mathrm{HTLR}}(N)$, can be computed as

$$\begin{aligned} S^{\mathrm{HTLR}}(N) &= \sum_{\ell=1}^{L} 2^{d\ell}(2^d - 1)\big(2dn_0 2^{L-\ell} p + p^{2d}\big) + 2^{dL}(N/2^{dL})^2 \\ &\leq 8dpn_0 \frac{N}{n_0^d} + p^{2d}\frac{N}{n_0^d} + n_0^d N \leq \left(16dp^{2-d} + p^d + 2^d p^d\right) N = \mathscr{O}\big(p^d N\big), \end{aligned} \tag{3.10}$$

where the prefactor depends only on d. ∎

Since the only difference between an $\mathcal{H}$-matrix and an HTLR matrix is the representation of low-rank submatrices, by a discussion analogous to that in the proof of Proposition 3.8, the storage of an $\mathcal{H}$-matrix of size N with rank p^d is

$$\begin{aligned} S^{\mathcal{H}}(N) &\leq \left(2^d d^{-1} p^d \log\big(N/N_0\big) + p^d + 2^d p^d\right) N \\ &= \mathscr{O}\big(p^d N \log_2(N) + p^d N\big). \end{aligned} \tag{3.11}$$

Comparing (3.10) and (3.11), we conclude that HTLR matrices always requires lower storage than $\mathcal{H}$-matrices. Furthermore, examining the distinct components of the exact storage complexity expression reveals that the first term for HTLR matrices has a smaller coefficient that depends only on the dimension d. In contrast, the coefficient for $\mathcal{H}$-matrices grows exponentially with d and logarithmically with N. Therefore, HTLR matrix also exhibits better asymptotic performance. Finally, it is remarkable that for the HTLR matrix, the main cost is the storage of Tucker core tensors, while that of the $\mathcal{H}$-matrix is the storage of basis matrices.

Similar results and dissussions can be obtained under strong admissibility. The only difference is the prefactor hidden in the $\mathscr{O}$ notation.

PROPOSITION 3.9 (Storage of HTLR matrices under strong admissibility). *Under the same condition in Proposition 3.8 except that $\mathsf{T}_{I\times I}$ is constructed under the strong admissibility condition with $\eta = \sqrt{d}$. If $\boldsymbol{A}^{\mathrm{HTLR}}$ is an HTLR matrix of rank p, then the storage required for $\boldsymbol{A}^{\mathrm{HTLR}}$ is $\mathscr{O}(p^d N)$.*

Proof. The calculations are similar to that in the proof of Proposition 3.8, the only difference is that the number of inadmissible and admissible blocks of a node is bounded by 3^d and $6^d - 3^d$, respectively. ∎

[1] From now on, for every complexity estimate, the prefactor is assumed to depend only on d.

3.3. HTLR Matrices on the Quasi-Uniform Grid. The HTLR matrix proposed in Section 3.2 requires a (uniform) tensor grid. However, in certain scenarios, functions are sampled from a quasi-uniform grid instead. In this section we show how HTLR matrices can be appied to the IE (1.1) using a discretization on the quasi-uniform grid. As with the non-uniform fast Fourier transform [4, 20], the key tool is the interpolation between the tensor grid and the quasi-uniform grid.

Suppose quasi-uniform grid points are given by $\{\boldsymbol{x}_i\}_{i=1}^N$ with non-overlapping computational domains $\{\Omega_i^{\mathrm{qu}}\}_{i=1}^N$,satisfying $\Omega = \cup_{i=1}^N \Omega_i^{\mathrm{qu}}$. An example of this can be seen in the triangulation of Ω from the finite element method, where each $\boldsymbol{x}_i$ is the the center of the triangular domain, as illustrated in Figure 3.1. The discretization in this case is expressed as follows:

$$a(\boldsymbol{x}_i)u_i + \sum_{j=1}^{N} K_{i,j}\left|\Omega_j^{\mathrm{qu}}\right|u_j = f(\boldsymbol{x}_i), \quad i = 1, \ldots, N,$$

where

$$K_{i,j} \coloneqq \begin{cases} k(\boldsymbol{x}_i, \boldsymbol{x}_j), & j \neq i, \\ \int_{\Omega_j} k(\boldsymbol{x}_i, \boldsymbol{y})\mathrm{d}\boldsymbol{y}/\left|\Omega_j^{\mathrm{qu}}\right|, & j = i. \end{cases}$$

This formulation can be viewed as applying the collection method using piecewise-constant basis functions $\left\{\mathbf{1}_{\Omega_i^{\mathrm{qu}}}(\boldsymbol{x})\right\}_i$.

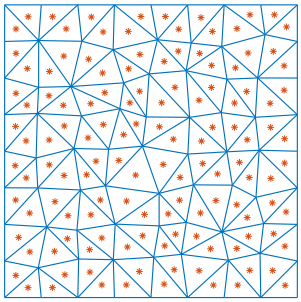


Fig. 3.1: Triangulation of $[0,1]^2$. The edges of each triangle are colored blue, and the centers are colored orange and marked with star "$*$".

Throughout this section, we assume the vector $\boldsymbol{u}$ in the forward computation is sampled from a smooth function $u(\boldsymbol{x})$, i.e., $u_i = u(\boldsymbol{x}_i)$. To utilize the HTLR matrix, a finer uniform tensor grid is prepared. Let m denote the number of points in each direction, resulting in the uniform grid denoted by $\{\hat{\boldsymbol{x}}_t\}_{t=1}^{m^d}$ with computational domains $\{\Omega_t^{\mathrm{uni}}\}_{t=1}^{m^d}$. Define $M = m^d$ as the number of uniform grid points and let $\hat{\boldsymbol{A}}$ represent the matrix defined by (1.3) on the uniform grid. We introduce two interpolation matrices $\boldsymbol{T} : \mathbb{R}^M \to \mathbb{R}^N$ and $\boldsymbol{S} : \mathbb{R}^N \to \mathbb{R}^M$, that facilitate the relationship between the quasi-uniform grid and the uniform grid, such that the following approximation holds:

$$\boldsymbol{A}\boldsymbol{u} \approx \boldsymbol{T}\hat{\boldsymbol{A}}\boldsymbol{S}\boldsymbol{u}, \tag{3.12}$$

Thus, the forward evaluation $\boldsymbol{A}\boldsymbol{u}$ can be approximately computed as $\boldsymbol{T}\hat{\boldsymbol{A}}\boldsymbol{S}\boldsymbol{u}$. The equation (3.12) can be interpreted as the following three steps: We first interpolate

a function on the quasi-uniform grid to the uniform tensor grid by multiplying the interpolation matrix $\boldsymbol{S}$. Next, we perform the matrix-vector multiplication on the uniform tensor grid using $\hat{\boldsymbol{A}}$. Finally, we interpolate the result from the uniform grid to the quasi-uniform grid by acting $\boldsymbol{T}$ on it. Since $\hat{\boldsymbol{A}}$ can be compressed into an HTLR matrix, its matrix-vector multiplication can be computed efficiently (see Section 4.2). On the other hand, as will be shown later, the matrices $\boldsymbol{T}$ and $\boldsymbol{S}$ are local interpolation matrix and, therefore, sparse. By combing these observations, we conclude that the matrix-vector multiplication can be performed efficiently on the quasi-uniform grid. If we replace $\hat{\boldsymbol{A}}$ in (3.12) with its HTLR approximation $\hat{\boldsymbol{A}}^{\mathrm{HTLR}}$, this can also be interpreted as a data-sparse representation of $\boldsymbol{A}$.

We discuss the choice of M and the construction of $\boldsymbol{T}$ and $\boldsymbol{S}$. The selection of M is a trade-off between accuracy and efficiency: A large M improves the accuracy but may reduce the efficiency. Typically, $M \approx \rho^d N$ is sufficient where $\rho \geq 1$ is a small constant. Relevant numerical results are presented in Section 5.4. To construct the interpolation matrices, we adopt the perspective of the Garlarkin method. Every vector $\boldsymbol{u} \in \mathbb{R}^N$ on the quasi-uniform grid $\{\boldsymbol{x}_i\}_{i=1}^N$ is interpreted as a piecewise-constant function defined by

$$u^{\mathrm{qu}}(\boldsymbol{x}) \coloneqq \sum_{i=1}^{N} u_i \mathbf{1}_{\Omega_i^{\mathrm{qu}}}(\boldsymbol{x}).$$

For a given uniform tensor grid $\{\hat{\boldsymbol{x}}_t\}_{t=1}^M$, the task is to determine the coefficients $\{\hat{u}_t\}$ such that

$$u^{\mathrm{qu}}(\boldsymbol{x}) \approx u^{\mathrm{uni}}(\boldsymbol{x}) \coloneqq \sum_{t=1}^{M} \hat{u}_t \mathbf{1}_{\Omega_t^{\mathrm{uni}}}(\boldsymbol{x}). \tag{3.13}$$

By taking the inner product with $\mathbf{1}_{\Omega_t^{\mathrm{uni}}}(\boldsymbol{x})$ on both sides, we obtain

$$\hat{u}_t = \sum_{i=1}^{N} \frac{|\Omega_t^{\mathrm{uni}} \cap \Omega_i^{\mathrm{qu}}|}{|\Omega_t^{\mathrm{uni}}|} u_i,$$

which means that the interpolation matrix from quasi-uniform grid to uniform grid is given by $\boldsymbol{S}_{t,i} = |\Omega_t^{\mathrm{uni}} \cap \Omega_i^{\mathrm{qu}}|/|\Omega_t^{\mathrm{uni}}|$. Similarly, the the interpolation matrix $\boldsymbol{T}$ from uniform grid to quasi-uniform grid is given by $\boldsymbol{T}_{i,t} = |\Omega_i^{\mathrm{qu}} \cap \Omega_t^{\mathrm{uni}}|/|\Omega_i^{\mathrm{qu}}|$. Since each region only intersects with few neighboring regions, both $\boldsymbol{T}$ and $\boldsymbol{S}$ are sparse, with the number of nonzeros approximately $\mathcal{O}(\max\{M, N\}) = \mathcal{O}(M)$.

4. Construction and Application of HTLR Matrices. This section focuses on the construction and application (matrix-vector multiplication) of HTLR matrices. Generally speaking, operations adhere follow a similar framework to those of $\mathcal{H}$-matrices, with the exception of a distinct step related to admissible leaf nodes. Throughout this section, we assume that $\mathsf{T}_{I\times I}$ is a given tensor node block cluster tree associated with a uniform grid.

4.1. Construction of HTLR Matrices. The construction of the HTLR matrix $\boldsymbol{A}^{\mathrm{HTLR}}$ directly follows the process outlined in Definition 3.7. More specifically, let (τ, σ) be the node we are currently working on. This can be categorized into three cases:

- (τ, σ) is an admissible leaf node: In this case, computational domains Ω_τ and Ω_σ are non-overlapping and $\boldsymbol{A}_{\tau,\sigma}$ is a TLR matrix constructed from the

kernel function k. Specifically, we first construct the TLR matrix $\boldsymbol{A}_{\tau,\sigma} = \mathrm{Tucker}\big(\boldsymbol{\mathcal{G}}, \{\boldsymbol{U}_\ell\}_{\ell=1}^d, \{\boldsymbol{V}_\ell\}_{\ell=1}^d\big)$ by the d-dimensional interpolation of the kernel matrix as (3.5), followed by a sequential QR to have each $\boldsymbol{U}_\ell$ and $\boldsymbol{V}_\ell$ orthonormal, then multiply the core tensor $\boldsymbol{\mathcal{G}}$ by h^d and $\boldsymbol{R}$ factors from the QR factorization.

- (τ,σ) is an inadmissible leaf node: We direct construct the dense matrix for $\boldsymbol{A}_{\tau,\sigma}^{\mathrm{HTLR}}$, i.e.,

$$\boldsymbol{A}_{\tau,\sigma}^{\mathrm{HTLR}} = \Big(a(\boldsymbol{x}_i)\delta_{i,j} + K_{i,j}h^d\Big)_{i\in\tau, j\in\sigma}, \tag{4.1}$$

  where $\delta_{i,j}$ is the Kronecker notation such that $\delta_{i,j} = 1$ when $i = j$ and $\delta_{i,j} = 0$ when $i \neq j$.
- (τ,σ) is a non-leaf node: The construction is performed recursively for each child $(\tau',\sigma') \in \mathrm{children}(\tau,\sigma)$.

The entire procedure is summarized as in Algorithm 4.1, which starts from the root (I, I) of $\mathsf{T}_{I\times I}$.

Algorithm 4.1 Construct$\big(a(\boldsymbol{x}), k(\boldsymbol{x},\boldsymbol{y}), (\tau,\sigma), p\big)$.

Input: Functions $a(\boldsymbol{x})$ and $k(\boldsymbol{x},\boldsymbol{y})$ in (1.1), current node (τ,σ), target rank p
Output: The HTLR matrix representation of the submatrix $\boldsymbol{A}_{\tau,\sigma}^{\mathrm{HTLR}}$

if (τ,σ) is an admissible leaf **then**
  Let Ω_τ and Ω_σ be the associated domains and $\{\boldsymbol{x_i}\}$ and$\{\boldsymbol{y_j}\}$ be the tensor points corresponding to τ and σ
  Compute the core tensor $\boldsymbol{\mathcal{G}}$ and factored matrices $\{\boldsymbol{U}_\ell\}$ and $\{\boldsymbol{V}_\ell\}$ by Chebyshev interpolation as in (3.5)
  for $\ell = 1,\ldots,d$ **do**
    Compute the QR factorization $\boldsymbol{U}_\ell = \boldsymbol{W}_\ell\boldsymbol{R}_\ell$ and $\boldsymbol{V}_\ell = \boldsymbol{Q}_\ell\boldsymbol{T}_\ell$
    Update $\boldsymbol{U}_\ell \leftarrow \boldsymbol{W}_\ell$ and $\boldsymbol{V}_\ell \leftarrow \boldsymbol{Q}_\ell$
  end for
  Update $\boldsymbol{\mathcal{G}} \leftarrow (h^d\boldsymbol{\mathcal{G}}) \times_1 \boldsymbol{R}_1 \times_2 \cdots \times_d \boldsymbol{R}_d \times_{d+1} \boldsymbol{T}_1 \times_{d+2} \cdots \times_{2d} \boldsymbol{T}_d$
  Set $\boldsymbol{A}_{\tau,\sigma}^{\mathrm{HTLR}} = \mathrm{Tucker}\big(\boldsymbol{\mathcal{G}}, \{\boldsymbol{U}_\ell\}_{\ell=1}^d, \{\boldsymbol{V}_\ell\}_{\ell=1}^d\big)$.
else if (τ,σ) is an inadmissible leaf **then**
  Form $\boldsymbol{A}_{\tau,\sigma}^{\mathrm{HTLR}}$ by (4.1)
else
  for $(\tau',\sigma') \in \mathrm{children}((\tau,\sigma))$ **do**
    $\boldsymbol{A}_{\tau',\sigma'}^{\mathrm{HTLR}} = \mathrm{Construct}\big(a(\boldsymbol{x}), k(\boldsymbol{x},\boldsymbol{y}), (\tau',\sigma'), p\big)$
  end for
end if

The leading complexity in Algorithm 4.1 arises from the orthogonalization of factored matrices, which admits

$$\sum_{\ell=1}^{L} \mathscr{O}\Big(2^{d\ell}\big(2dn_0 2^{L-\ell}p^2 + 2dp^{2d+1}\big)\Big) = \mathscr{O}\big(p^{3-d}N + p^{d+1}N\big).$$

The dominant step in this process is the contraction with tensor $\boldsymbol{\mathcal{G}}$. On the other hand, for $\mathcal{H}$-matrices, the leading complexity also stems from the orthogonalization

of low-rank factors:

$$\sum_{\ell=1}^{L} \mathscr{O}\Big(2^{d\ell}(2^d-1)\big(2^{d(L-\ell)}n_0^d p^{2d}+p^{3d}\big)\Big) = \mathscr{O}\big(p^{2d}N\log N + p^{2d}N\big).$$

However, in this case, the dominant step is the QR factorization of basis matrices $\boldsymbol{U}$ and $\boldsymbol{V}$. Therefore, the total construction complexity of HTLR matrices and $\mathcal{H}$-matrices are $\mathscr{O}(p^{d+1}N)$ and $\mathscr{O}\big(p^{2d}N\log_2(N)\big)$ respectively, indicating that the construction of HTLR matrices is more efficient. The main difference is that, for HTLR matrices, we deal with d matrices of size $n \times p$, while for $\mathcal{H}$-matrices we need to construct and orthogonalize a matrix of size $n^d \times p^d$.

4.2. Application of HTLR Matrices. Using the hierarchical structure, the application, or matrix-vector multiplication, of HTLR matrices can also be computed efficiently. Starting at (I, I), let (τ, σ) be the current node, and we aim to update

$$\boldsymbol{f}_\tau \leftarrow \boldsymbol{f}_\tau + \boldsymbol{A}^{\mathrm{HTLR}}_{\tau,\sigma}\boldsymbol{u}_\sigma. \tag{4.2}$$

There are three cases depending on the type of the node (τ, σ):

- (τ, σ) is an admissible leaf: In this case, assume the TLR representation of $\boldsymbol{A}^{\mathrm{HTLR}}_{\tau,\sigma}$ is denoted by $\boldsymbol{A}^{\mathrm{HTLR}}_{\tau,\sigma} = \mathrm{Tucker}\big(\boldsymbol{\mathcal{G}}, \{\boldsymbol{U}_\ell\}_{\ell=1}^d, \{\boldsymbol{V}_\ell\}_{\ell=1}^d\big)$. The matrix-vector multiplication is calculated via

  $$\boldsymbol{f}_\tau \leftarrow \boldsymbol{f}_\tau + (\boldsymbol{U}_d \otimes \cdots \otimes \boldsymbol{U}_1)\boldsymbol{G}(\boldsymbol{V}_d \otimes \cdots \otimes \boldsymbol{V}_1)^*\boldsymbol{u}_\sigma, \tag{4.3}$$

  where $\boldsymbol{G}$ is obatained by reshaping $\boldsymbol{\mathcal{G}}$ into a matrix (cf. Remark 3.2). The product of Kronecker product of a matrix series and vector is computed as follows. Consider the calculation of $\boldsymbol{w} = (\boldsymbol{V}_d \otimes \cdots \otimes \boldsymbol{V}_1)^*\boldsymbol{u}$ where $\boldsymbol{V}_\ell \in \mathbb{R}^{n_\ell \times p_\ell}$ and $\boldsymbol{u} \in \mathbb{R}^{n_1 \cdots n_d}$. By reshaping $\boldsymbol{u}$ to a d-dimensional tensor $\boldsymbol{\mathcal{U}}$, the tensorized result $\boldsymbol{\mathcal{W}}$ can be obtained through contractions along each dimension, expressed as

  $$\boldsymbol{\mathcal{W}} = \boldsymbol{\mathcal{U}} \times_1 \boldsymbol{V}_1^* \times_2 \cdots \times_d \boldsymbol{V}_d^*.$$

  Using these tensors, the multiplication of matrix $\boldsymbol{h} = \boldsymbol{G}\boldsymbol{w}$ can be reformulated as a tensor contraction $\boldsymbol{\mathcal{H}} = \boldsymbol{\mathcal{G}} \times_{[d+1,\ldots,2d],[1,\ldots,d]} \boldsymbol{\mathcal{W}}$. Finally, the computation of $\boldsymbol{f} = (\boldsymbol{U}_d \otimes \cdots \otimes \boldsymbol{U}_1)\boldsymbol{h}$ is analogous to the application of $(\boldsymbol{V}_d \otimes \cdots \otimes \boldsymbol{V}_1)^*$. Suppose the matrices $\boldsymbol{U}_\ell$ and $\boldsymbol{V}_\ell$ are of size $n \times p$, and $\boldsymbol{G}$ are of size $p^d \times p^d$, the computation complexity of (4.3) is $\mathscr{O}\big(dpn^d + p^{2d}\big)$. This is more efficient than the conventional low-rank representation in $\mathcal{H}$-matrices with $r = p^d$, whose complexity is $\mathscr{O}\big(p^d n^d + p^{2d}\big)$.
- (τ, σ) is an inadmissible leaf node: For this case, the update (4.2) is computed directly since $\boldsymbol{A}^{\mathrm{HTLR}}_{\tau,\sigma}$ is a small dense matrix.
- (τ, σ) is a non-leaf node: The update can be reduced to each children node, i.e., we perform $\boldsymbol{f}_{\tau'} \leftarrow \boldsymbol{f}_{\tau'} + \boldsymbol{A}^{\mathrm{HTLR}}_{\tau',\sigma'}\boldsymbol{u}_{\sigma'}$ for each $(\tau', \sigma') \in \mathrm{children}(\tau, \sigma)$.

Algorithm 4.2 and Proposition 4.1 provide the pseudocode as well as the complexity results of HTLR matrix-vector multiplication.

PROPOSITION 4.1 (Complexity of HTLR application). *For both weak and strong admissible conditions, the matrix-vector multiplication complexity for a rank-p HTLR matrix is* $\mathscr{O}(pN\log N + p^d N)$.

Algorithm 4.2 HMultVec$\big(\boldsymbol{A}^{\mathrm{HTLR}}, (\tau,\sigma), \boldsymbol{u}, \boldsymbol{f}\big)$.

Input: HTLR matrix $\boldsymbol{A}^{\mathrm{HTLR}}$, current node (τ,σ), vectors $\boldsymbol{u}$ and $\boldsymbol{f}$.
Output: $\boldsymbol{f}_\tau \leftarrow \boldsymbol{f}_\tau + \boldsymbol{A}^{\mathrm{HTLR}}_{\tau,\sigma}\boldsymbol{u}_\sigma$.

if (τ,σ) is an admissible leaf **then**
  Let $\boldsymbol{A}^{\mathrm{HTLR}}_{\tau,\sigma} = \mathrm{Tucker}\big(\boldsymbol{\mathcal{G}}, \{\boldsymbol{U}_\ell\}_{\ell=1}^d, \{\boldsymbol{V}_\ell\}_{\ell=1}^d\big)$
  Reshape $\boldsymbol{u}_\sigma$ to a d-dimensional tensor $\boldsymbol{\mathcal{U}}_\sigma$
  $\boldsymbol{\mathcal{W}}_\sigma = \boldsymbol{\mathcal{U}}_\sigma \times_1 \boldsymbol{V}_1^* \times_2 \cdots \times_d \boldsymbol{V}_d^*$
  $\boldsymbol{\mathcal{H}}_\tau = \boldsymbol{\mathcal{G}} \times_{[d+1,\ldots,2d],[1,\ldots,d]} \boldsymbol{\mathcal{W}}_\sigma$
  $\boldsymbol{\mathcal{F}}_\tau = \boldsymbol{\mathcal{H}}_\tau \times_1 \boldsymbol{U}_1 \times_2 \cdots \times_d \boldsymbol{U}_d$
  Reshape $\boldsymbol{\mathcal{F}}_\tau$ to a vector and add it into $\boldsymbol{f}_\tau$
else if (τ,σ) is an inadmissible leaf **then**
  Compute $\boldsymbol{f}_\tau \leftarrow \boldsymbol{f}_\tau + \boldsymbol{A}^{\mathrm{HTLR}}_{\tau,\sigma}\boldsymbol{u}_\sigma$
else
  for $(\tau',\sigma') \in \mathrm{children}((\tau,\sigma))$ **do**
    HMultVec$\big(\boldsymbol{A}^{\mathrm{HTLR}}, (\tau',\sigma'), \boldsymbol{u}, \boldsymbol{f}\big)$
  end for
end if

For a rank-p^d $\mathcal{H}$-matrix of size N, the complexity of matrix-vector multiplication of is $\mathscr{O}(p^d N \log N + p^d N)$. Specifically, for both HTLR matrices and $\mathcal{H}$-matrices, the computations involving dense matrices in inadmissible leaves incur the same cost, $\mathscr{O}(p^d N)$. Similarly, the complexity for computations associated with core matrices or tensors in admissible leaves is also $\mathscr{O}(p^d N)$. However, when it comes to the computation of factored or basis matrices, the complexity for HTLR matrices is reduced to $\mathscr{O}(pN \log N)$, whereas the complexity for $\mathcal{H}$-matrices is $\mathscr{O}(p^d N \log N)$. From the analysis, we conclude that HTLR matrices have a significantly smaller prefactor in the leading complexity, and are more efficient than $\mathcal{H}$-matrices when applied to a vector.

5. Numerical Results. We apply HTLR matrices and $\mathcal{H}$-matrices to several examples to evaluate their efficiency. Two types of kernels are considered:

- Gaussian Kernel: $k(\boldsymbol{x},\boldsymbol{y}) = \exp\big(-\|\boldsymbol{x}-\boldsymbol{y}\|^2/(2\sigma^2)\big)$. The Gaussian kernel is smooth and has no singularity. In our experiments, the bandwidth σ is set to be $\sqrt{d}$ where d is the dimension of the problem.
- Single Layer Potential (SLP) Kernel: $k(\boldsymbol{x},\boldsymbol{y}) = -\log\big(\|\boldsymbol{x}-\boldsymbol{y}\|\big)/(2\pi)$ for $d=2$ and $k(\boldsymbol{x},\boldsymbol{y}) = \|\boldsymbol{x}-\boldsymbol{y}\|/(4\pi)$ for $d=3$. The SLP kernel has singularity on the diagonal and is smooth everywhere else.

All algorithms are implemented in MATLAB R2023b. The experiments are carried out on a server with an Intel Gold 6226R CPU at 2.90 GHz and 1000.6 GB of RAM.

For each example, the following notations are adopted: We use t_{c} and t_{a} to denote the runtime (in seconds) for the construction and application, and m_{h} to denote the memory cost (in GB). The approximated relative error of the fast matrix-vector multiplication is defined by

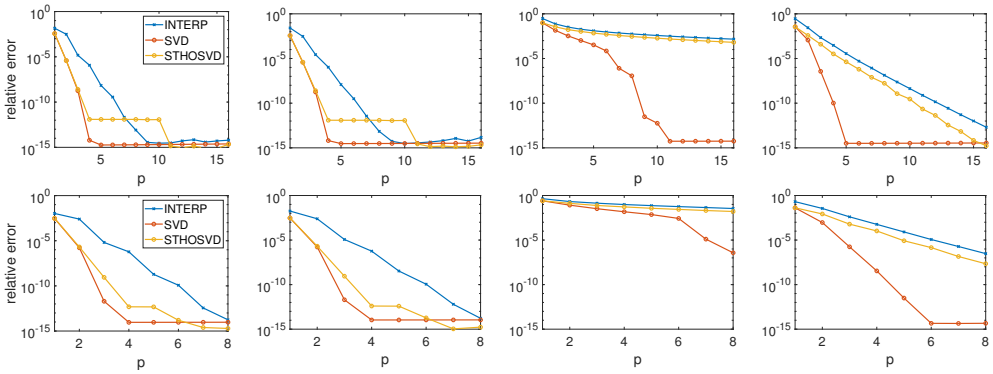
$$e_{\mathrm{a;r}} = \frac{\|\tilde{\boldsymbol{f}}(I_r,:) - \boldsymbol{f}(I_r,:)\|_2}{\|\boldsymbol{f}(I_r,:)\|_2},$$

where $\boldsymbol{f} = \boldsymbol{A}\boldsymbol{u}$ and $\tilde{\boldsymbol{f}} = \tilde{\boldsymbol{A}}\boldsymbol{u}$ are the exact matrix-vector multiplication and the approximated result corresponding to the HTLR or $\mathcal{H}$-matrix respectively. The index set I_r is randomly sampled from $[N]$ and contains $|I_r| = 1000$. Based on the results,

$e_{\mathrm{a;r}}$ serves as a good estimate for the exact relative error. Vector $\boldsymbol{u} \in \mathbb{R}^N$ is generated randomly (Section 5.2 and Section 5.3) or evaluated from a specific function (Section 5.4). Unless specified, we set the threshold of leaf nodes and Tucker rank to $N_0 = 16^2$ and $p = 8$ for 2D problem, and $N_0 = 5^3$ and $p = 4$ for 3D problems. In Section 5.1, we discuss the tensor low-rank representations of submatrices corresponding to these kernels under different cases. Sections 5.2 and 5.3 demonstrate results on uniform grids in 2D and 3D, respectively. Finally, in Section 5.4, we illustrate the application of HTLR matrices on quasi-uniform grids.

5.1. Exploring Tensor Low-Rank Representations of Kernel Functions. We numerically explore the Tucker low-rankness of interaction matrices, with a comparison to conventional low-rank matrices. Two configurations of domains for the interaction matrices are examined: Neighbor domains, which consist of two adjacent subdomains, and well-separated domains, which are characterized by two strongly admissible subdomains. To construct a low-rank decomposition, we use the following approaches: For both types of low-rank structures, interpolation (INTERP) method discussed in Section 3.1 is used to create a low-rank decomposition. Besides, we apply SVD to compute conventional low-rank decomposition and the STHOSVD [45] to compute the Tucker decomposition. The accuracy of the compression is measured using the relative error with respect to the Frobenius norm.

In the 2D case, we consider following domains: $\Omega_1 = [0, h] \times [0, h]$, $\Omega_2 = [h, 2h] \times [0, h]$ and $\Omega_3 = [2h, 3h] \times [0, h]$ where $h = 0.25$. The domains Ω_1 and Ω_2 are neighbors, while Ω_1 and Ω_3 are well-separated. Each domain is discretized with 32 points in each direction. The rank p in each direction ranges from 1 to 16, and the rank of the SVD equals to p^2. Similarly, the domains in 3D are given by $\Omega_1 = [0, h] \times [0, h] \times [0, h]$, $\Omega_2 = [h, 2h] \times [0, h] \times [0, h]$ and $\Omega_3 = [2h, 3h] \times [0, h] \times [0, h]$ for $h = 0.25$. Each domain is discretized with 16 points in each direction, and p ranges from 1 to 8 with the rank of the SVD equals to p^3.



(a) Gaussian kernel and neighbor domains.

(b) Gaussian kernel and well-separated domains.

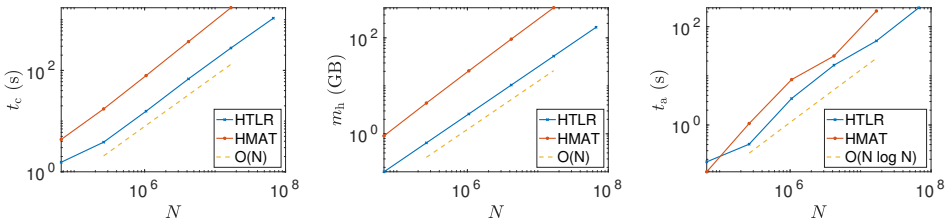
(c) SLP kernel and neighbor domains.

(d) SLP kernel and well-separated domains.

Fig. 5.1: Relative approximation errors of Gaussian and SLP kernel. The size of the interaction matrix is $32^2 \times 32^2$ for 2D problems (top) and $16^3 \times 16^3$ for 3D problems (bottom).

The results of different low-rank decompositions are plotted in Figure 5.1. For well-separated domains (Figure 5.1b and 5.1d), all three methods exhibit a rapid decay in error as p increases, regardless of the kernel. In this scenario, the interpolation method is sufficient to achieve an accurate low-rank decomposition. However, when two domains are neighbors (as depicted in Figure 5.1a and 5.1c), the decay rates of the relative error exhibit dependence on the kernels. For the Gaussian kernel, both low-rank structures still work well, with the interpolation-based construction yielding satisfactory results. In contrast, for the SLP kernel, in contrast, the Tucker low-rank format proves unsuitable. Even with the application of STHOSVD, the relative error decreases very slowly as the rank increases. Consequently, in the subsequent sections, we will adopt SLP kernel with strong admissibility condition and Gaussian kernel with weak admissibility condition.

5.2. Uniform Grid in Two Dimensions. This section provides experiments of HTLR matrices on 2D uniform grids. Let $\Omega = [0,1]^2$ be the unit square and $a(\boldsymbol{x}) \equiv 0$ in (1.1).



(a) Construction time. (b) Memory. (c) Application time

Fig. 5.2: Time and memory costs of HTLR matrices and $\mathcal{H}$-matrices for the Gaussian kernel under weak admissibility in 2D.

We focus on the Gaussian kernel under the weak admissibility condition first. We discretize the problem with n points in each dimension, where n ranges from 256 to 8192. Figure 5.2 plots the construction and application time as well as the memory usage of HTLR matrices and $\mathcal{H}$-matrices. The data point of $\mathcal{H}$-matrix of size 8192^2 is missing due to the memory limitation.

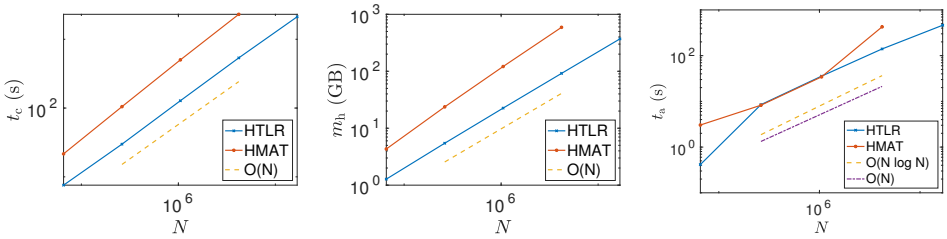
It is evident that both t_c and m_h scales as $\mathcal{O}(N)$, as predicted while the application time fluncates but asymptotically follows the $\mathcal{O}(N \log N)$ complexity. Detailed information is given in Table 5.1. Since the low-rank components in HTLR matrices and $\mathcal{H}$-matrices are both constructed by interpolation, they exhibit the same level of accuracy. Therefore, we only report the error for HTLR matrices. Compared with $\mathcal{H}$-matrices, the construction runtime of HTLR matrices is, on average, 4 to 5 times faster and the memory usage is 8 to 10 times lower. These advantages become increasingly pronounced as the size N grows. Notably, the error $e_{a;r}$ remains stable as N increases. This indicates that, when dealing with the Gaussian kernel, empolying weak admissibility with a constant rank is sufficient to achieve the desired accuracy.

Next we consider the SLP kernel under the strong admissibility condition. In this scenario, n ranges from 256 to 4096. Figure 5.3 illustrates the comparison between HTLR matrices and $\mathcal{H}$-matrices. The data point of $\mathcal{H}$-matrix of size 4096^2 is absent due to the memory limit.

As anticipated, the complexities of $\mathcal{O}(N)$ for construction and storage are strictly

N	p	t_{c} (s)	speedup	m_{h} (GB)	memory saving	$e_{\mathrm{a;r}}$
256^2	8	1.5e+00	2.8×	1.6e-01	5.6×	1.3e-11
512^2	8	3.8e+00	4.5×	6.5e-01	6.8×	2.1e-11
1024^2	8	1.6e+01	5.0×	2.6e+00	7.9×	1.5e-10
2048^2	8	6.8e+01	5.4×	1.0e+01	9.1×	2.0e-11
4096^2	8	2.8e+02	6.2×	4.1e+01	10.2×	1.8e-11
8192^2	8	1.1e+03	-	1.7e+02	-	2.3e-11

Table 5.1: Numerical results of HTLR matrices for the Gaussian kernel under weak admissibility in 2D.



(a) Construction time. (b) Memory. (c) Application time

Fig. 5.3: Time and memory costs of HTLR matrices and $\mathcal{H}$-matrices for the SLP kernel under strong admissibility in 2D.

followed. However, the application time exhibits a $\mathscr{O}(N)$ complexity instead of the expected $\mathscr{O}(N \log N)$. As discussed following Proposition 4.1, this discrepancy may arise because the prefactor associated with the $\mathscr{O}(N \log N)$ term is smaller than that of the $\mathscr{O}(N)$ term. Consequently, when the size of the matrix is not sufficiently large, the $\mathscr{O}(N \log N)$ term does not dominate the complexity estimate. Additionally, the application time does not follow the anticipated complexity initially. This behavior occurs because, for smaller matrix sizes, the number of admissible blocks is relatively small compared to the matrix size, resulting in a runtime that scales between $\mathscr{O}(N)$ and $\mathscr{O}(N^2)$.

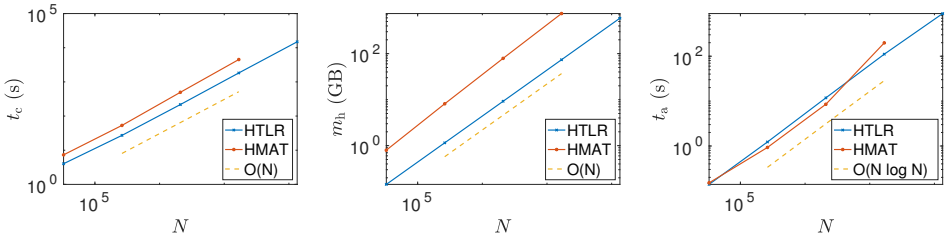
Table 5.2 presents the results. The construction runtime of HTLR matrices is 3 to 4 times faster and the memory usage is 4 to 5 times lower. However, the advantage here is not as obvious as it is under the weak admissibility condition, mainly because of the increase in the number of dense blocks. Moreover, the relative error does not exhibit significant variation as the matrix size N increases, which is consistent with Theorem 3.4. Therefore, strong admissible HTLR matrices prove to be more applicable in this case, as they demonstrate superior performance in both time efficiency and memory usage.

5.3. Uniform Grid in Three Dimensions. The example in this section is the 3D analogue of Section 5.2, where $\Omega = [0, 1]^3$ with $a(\boldsymbol{x}) \equiv 0$.

When the kernel function is the Gaussian, n ranges from 32 to 512. Figure 5.4

N	p	t_{c} (s)	speedup	m_{h} (GB)	memory saving	$e_{\mathrm{a;r}}$
256^2	8	7.6e+00	2.9×	1.3e+00	3.4×	1.1e-07
512^2	8	3.0e+01	3.5×	5.4e+00	4.3×	8.1e-08
1024^2	8	1.3e+02	3.9×	2.3e+01	5.4×	1.2e-07
2048^2	8	5.4e+02	4.3×	9.2e+01	6.4×	5.3e-08
4096^2	8	2.1e+03	-	3.7e+02	-	4.4e-08

Table 5.2: Numerical results of HTLR matrices for the SLP kernel under strong admissibility in 2D.



(a) Construction time. (b) Memory. (c) Application time

Fig. 5.4: Time and memory costs of HTLR matrices and $\mathcal{H}$-matrices for the Gaussian kernel under weak admissibility in 3D.

plots corresponding results and detalied data can be found in Table 5.3. The data point of $\mathcal{H}$-matrix of size 512^3 is missing due to the memory limitation.

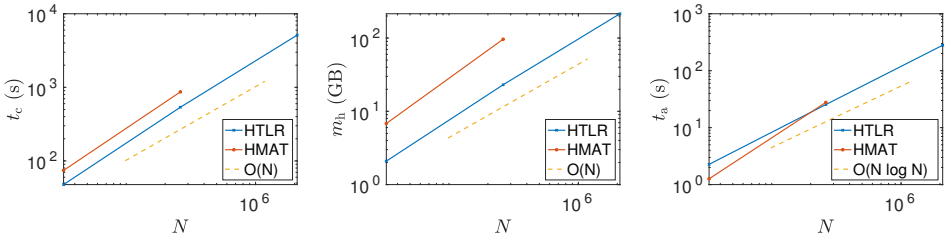
It is straightforward that the $\mathcal{O}(N)$ complexity for construction and memory, as well as the $\mathcal{O}(N \log N)$ complexity for application, are nearly strictly upheld. Similar to the 2D case, the proposed HTLR matrices demonstrate greater efficiency than $\mathcal{H}$-matrices, achieving nearly double the speed in construction and offering a memory savings of 7 to 10 times. The memory cost of a HTLR matrix of size 512^3 is nearly 590 GB, which is still less than that of a $\mathcal{H}$-matrix of size 256^3 (744 GB). However, as illustrated in Figure 5.4c, the advantage in application is not immediately noticeable until N reaches a moderately large size. Further increasing the rank could potentially highlight the benefits of HTLR matrices. Further increasing the rank could reveal the benefits of HTLR matrices. Additionally, as indicated in the last column of Table 5.3, the error in this scenario remains small even when a constant rank is employed.

As for the SLP kernel, we discretize the problem with n points in each dimension for for $n = 32$, 64 and 128 respectively. The corresponding results are summarized in Figure 5.5 and Table 5.4. The data point of $\mathcal{H}$-matrices when $N = 128^3$ is missing due to the memory limitation.

The estimated complexities are consistent with our numerical observations, indicating that HTLR matrices are efficient in both construction (1.6 times faster) and memory usage (3 to 4 times lower). Like the 2D case, the application error under strong admissibility exhibits only mild variation as the matrix size N increases, which

N	p	t_{c} (s)	speedup	m_{h} (GB)	memory saving	$e_{\mathrm{a;r}}$
32^3	4	4.0e+00	1.8×	1.4e-01	5.5×	2.3e-05
64^3	4	2.7e+01	2.0×	1.2e+00	7.1×	7.4e-06
128^3	4	2.2e+02	2.3×	9.2e+00	8.6×	8.6e-06
256^3	4	1.8e+03	2.5×	7.4e+01	10.1×	1.2e-05
512^3	4	1.5e+04	-	5.9e+02	-	5.4e-05

Table 5.3: Relative application error of HTLR matrices and $\mathcal{H}$-matrices for the Gaussian kernel under weak admissibility in 3D.



(a) Construction time. (b) Memory. (c) Application time

Fig. 5.5: Time and memory costs of HTLR matrices and $\mathcal{H}$-matrices for the SLP kernel under strong admissibility in 3D.

ensures that it remains an accurate and stable approximation of the matrix.

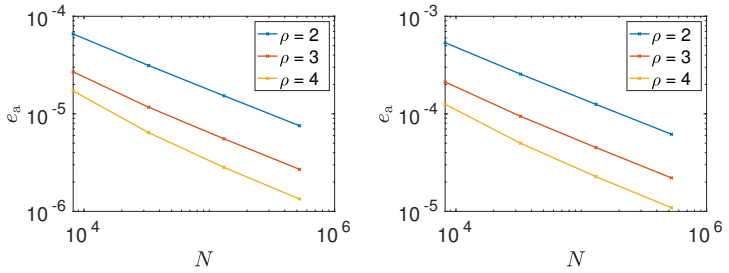
N	p	t_{c} (s)	speedup	m_{h} (GB)	memory saving	$e_{\mathrm{a;r}}$
32^3	4	4.8e+01	1.6×	2.1e+00	3.3×	1.9e-04
64^3	4	5.4e+02	1.6×	2.3e+01	4.2×	2.6e-04
128^3	4	5.1e+03	-	2.1e+02	-	3.1e-04

Table 5.4: Relative application error of HTLR matrices and $\mathcal{H}$-matrices for the SLP kernel under strong admissibility in 3D.

5.4. Quasi-Uniform Grid in Two Dimensions. In this section, we present two examples to illustrate the application of HTLR matrices on quasi-uniform grids. More specifically, we consider the same domain and kernel functions as discussed in Section 5.2. The discretization points are given by the triangulation of Ω where each $\boldsymbol{x}_i$ is the the center of the triangular domain (See Figure 3.1). The matrix size N is 8192, 32768, 131072 and 524288. Let M be the number of points of the uniform grid and $\boldsymbol{A}$ and $\hat{\boldsymbol{A}}$ be the corresponding matrices of quasi-uniform and uniform grid respectively, as discussed in Section 3.3. In this case, $\hat{\boldsymbol{A}}$ is represented by its HTLR matrix approximation. we define the oversampling ratio $\rho = \sqrt{2M/N}$, where the

factor 2 comes from the fact that 1 square domain corresponds to 2 triangular domains. The underlying function $u(\boldsymbol{x})$ of vector $\boldsymbol{u}$ is selected to be

$$u(\boldsymbol{x}) = u(x_1, x_2) = 1 + 0.5\mathrm{e}^{-(x_1-0.3)^2-(x_2-0.6)^2} + \sin(5x_1x_2).$$



(a) Gaussian kernel. (b) SLP kernel.

Fig. 5.6: Relative application error of HTLR matrices for Gaussian and SLP kernel in 2D. Left: Gaussian kernel. Right: SLP kernel.

Figure 5.6 describes the approximated relative application error of $\hat{\boldsymbol{A}}$ for different sizes. Weak and strong admissibility are adopted for Gaussian kernel and SLP kernel respectively. For each N, a larger ρ leads to a smaller error, which aligns with our intuition. Conversely, for a fixed ρ, the error decreases as N increases. This is due to the fact that both the quasi-uniform and uniform grids become finer with larger N Particularly, when $N = 524288$ and $\rho = 2$, the errors for the Gaussian kernel and SLP kernel are approximately 10^{-5} and 10^{-4} respectively. In comparison, Tables 5.1 and 5.2 indicate that the application errors of the HTLR matrix at this size are about 10^{-10} and 10^{-7}, suggesting that the dominant error in this case arises from interpolation. We claim that choosing a small ρ (for example, $\rho = 2$) is sufficient for many practical application, as the corresponding curves have demonstrate farily good performance.

6. Conclusion and Future Work. In this paper, we introduce hierarchical Tucker low-rank matrices and present the corresponding algorithms for construction and matrix-vector multiplication. When the underlying discretization exhibits a tensor structure, HTLR matrices are generally more efficient than $\mathcal{H}$-matrices. We establish that the complexity for storage and construction is $\mathcal{O}(N)$, while the complexity for application is $\mathcal{O}(N \log N)$. Compared to $\mathcal{H}$-matrices, the prefactors of the dominant terms are smaller and usually exhibit a linear dependence on the dimension rather than an exponential dependence. Furthermore, we demonstrate the application of HTLR matrices on quasi-uniform grids, enhancing their applicability. We also discuss the relevance of Tucker low-rank structures. Numerical results show that HTLR matrices usually save 3 to 10 times in memory and provide speedups of 2 to 6 times compared to $\mathcal{H}$-matrices.

There are several future directions to consider. One potential avenue is to develop additional algebraic operations for HTLR matrices, including matrix addition, multiplication, and LU decomposition. Furthermore, exploring the parallelization of these algorithms could enhance their efficiency and scalability, enabling them to accommodate larger problem sizes effectively.

REFERENCES

[1] A. Aminfar, S. Ambikasaran, and E. Darve, *A fast block low-rank dense solver with applications to finite-element matrices*, J. Comput. Phys., 304 (2016), pp. 170–188, https://doi.org/10.1016/j.jcp.2015.10.012.

[2] F. Aurenhammer, *Voronoi diagrams—a survey of a fundamental geometric data structure*, ACM Comput. Surv., 23 (1991), pp. 345–405, https://doi.org/10.1145/116873.116880.

[3] J. Barnes and P. Hut, *A hierarchical $O(N \log N)$ force-calculation algorithm*, Nature, 324 (1986), pp. 446–449, https://doi.org/10.1038/324446a0.

[4] A. H. Barnett, J. Magland, and L. af Klinteberg, *A parallel nonuniform fast fourier transform library based on an "exponential of semicircle" kernel*, SIAM J. Sci. Comput., 41 (2019), pp. C479–C504, https://doi.org/10.1137/18M120885X.

[5] M. Bebendorf, *Hierarchical Matrices: A means to efficiently solve elliptic boundary value problems*, Lect. Notes Comput. Sci. Eng. 63, Springer Berlin, 1st ed., 2008.

[6] S. Börm, L. Grasedyck, and W. Hackbusch, *Introduction to hierarchical matrices with applications*, Eng. Anal. Boundary Elem., 27 (2003), pp. 405–422, https://doi.org/10.1016/S0955-7997(02)00152-2.

[7] E. Candès, L. Demanet, and L. Ying, *A fast butterfly algorithm for the computation of fourier integral operators*, Multiscale Model. Simul., 7 (2009), pp. 1727–1750, https://doi.org/10.1137/080734339.

[8] J. Carrier, L. Greengard, and V. Rokhlin, *A fast adaptive multipole algorithm for particle simulations*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 669–686, https://doi.org/10.1137/0909044.

[9] S. Chandrasekaran, M. Gu, and T. Pals, *A fast ULV decomposition solver for hierarchically semiseparable representations*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 603–622, https://doi.org/10.1137/S0895479803436652.

[10] H. Cheng, Z. Gimbutas, P.-G. Martinsson, and V. Rokhlin, *On the compression of low rank matrices*, SIAM J. Sci. Comput., 26 (2005), pp. 1389–1404, https://doi.org/10.1137/030602678.

[11] A. Cichocki, N. Lee, I. Oseledets, A.-H. Phan, Q. Zhao, and D. P. Mandic, *Tensor networks for dimensionality reduction and large-scale optimization: Part 1 Low-rank tensor decompositions*, Found. Trends Mach. Learn., 9 (2016), pp. 249–429, https://doi.org/10.1561/2200000059.

[12] E. Corona, P.-G. Martinsson, and D. Zorin, *An $O(N)$ direct solver for integral equations on the plane*, Appl. Comput. Harmon. Anal., 38 (2015), pp. 284–317, https://doi.org/10.1016/j.acha.2014.04.002.

[13] L. De Lathauwer, B. De Moor, and J. Vandewalle, *A multilinear singular value decomposition*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1253–1278, https://doi.org/10.1137/S0895479896305696.

[14] L. De Lathauwer, B. De Moor, and J. Vandewalle, *On the best rank-1 and rank-$(R_1, R_2, \ldots, R_N)$ approximation of higher-order tensors*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1324–1342, https://doi.org/10.1137/S0895479898346995.

[15] W. Fong and E. Darve, *The black-box fast multipole method*, J. Comput. Phys., 228 (2009), pp. 8712–8725, https://doi.org/10.1016/j.jcp.2009.08.031.

[16] I. P. Gavrilyuk, W. Hackbusch, and B. N. Khoromskij, *Hierarchical tensor-product approximation to the inverse and related operators for high-dimensional elliptic problems*, Computing, 74 (2005), pp. 131–157, https://doi.org/10.1007/s00607-004-0086-y.

[17] A. Gillman, P. M. Young, and P.-G. Martinsson, *A direct solver with $O(N)$ complexity for integral equations on one-dimensional domains*, Front. Math. China, 7 (2012), pp. 217–247, https://doi.org/10.1007/s11464-012-0188-3.

[18] L. Grasedyck and W. Hackbusch, *Construction and arithmetics of $\mathcal{H}$-matrices*, Computing, 70 (2003), pp. 295–334, https://doi.org/10.1007/s00607-003-0019-1.

[19] L. Greengard, D. Gueyffier, P.-G. Martinsson, and V. Rokhlin, *Fast direct solvers for integral equations in complex three-dimensional domains*, Acta Numer., 18 (2009), pp. 243–275, https://doi.org/10.1017/S0962492906410011.

[20] L. Greengard and J.-Y. Lee, *Accelerating the nonuniform fast Fourier transform*, SIAM Rev., 46 (2004), pp. 443–454, https://doi.org/10.1137/S003614450343200X.

[21] L. Greengard and V. Rokhlin, *A fast algorithm for particle simulations*, J. Comput. Phys., 135 (1997), pp. 280–292, https://doi.org/10.1006/jcph.1997.5706.

[22] W. Hackbusch, *A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: Introduction to $\mathcal{H}$-matrices*, Computing, 62 (1999), pp. 89–108, https://doi.org/10.1007/s006070050015.

[23] W. Hackbusch, *Hierarchical matrices: Algorithms and analysis*, Springer Ser. Comput. Math.

49, Springer, 1st ed., 2015.
[24] W. Hackbusch and S. Börm, *Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices*, Computing, 69 (2002), pp. 1–35, https://doi.org/10.1007/s00607-002-1450-4.
[25] W. Hackbusch and B. N. Khoromskij, *A sparse $\mathcal{H}$-matrix arithmetic. Part II: Application to multi-dimensional problems*, Computing, 64 (2000), pp. 21–47, https://doi.org/10.1007/PL00021408.
[26] W. Hackbusch and B. N. Khoromskij, *Low-rank Kronecker-product approximation to multi-dimensional nonlocal operators. Part I. Separable approximation of multi-variate functions*, Computing, 76 (2006), pp. 177–202, https://doi.org/10.1007/s00607-005-0144-0.
[27] W. Hackbusch and B. N. Khoromskij, *Low-rank Kronecker-product approximation to multi-dimensional nonlocal operators. Part II. HKT representation of certain operators*, Computing, 76 (2006), pp. 203–225, https://doi.org/10.1007/s00607-005-0145-z.
[28] W. Hackbusch and B. N. Khoromskij, *Tensor-product approximation to multidimensional integral operators and Green's functions*, SIAM J. Matrix Anal. Appl., 30 (2008), pp. 1233–1253, https://doi.org/10.1137/060657017.
[29] W. Hackbusch, B. N. Khoromskij, and E. E. Tyrtyshnikov, *Hierarchical Kronecker tensor-product approximations*, J. Numer. Math., 13 (2005), pp. 119–156, https://doi.org/doi:10.1515/1569395054012767.
[30] B. Hashemi and Y. Nakatsukasa, *RTSMS: Randomized Tucker with single-mode sketching*, arXiv.org, (2023), https://doi.org/10.48550/arXiv.2311.14873.
[31] K. L. Ho and L. Greengard, *A fast direct solver for structured linear systems by recursive skeletonization*, SIAM J. Sci. Comput., 34 (2012), pp. A2507–A2532, https://doi.org/10.1137/120866683.
[32] K. L. Ho and L. Ying, *Hierarchical interpolative factorization for elliptic operators: Integral equations*, Commun. Pure Appl. Math., 69 (2016), pp. 1314–1353, https://doi.org/10.1002/cpa.21577.
[33] P. M. Kielstra, T. Shi, H. Luo, J. Qian, and Y. Liu, *A linear-complexity tensor butterfly algorithm for compressing high-dimensional oscillatory integral operators*, Multiscale Model. Simul., (2025), pp. 864–893, https://doi.org/10.1137/24M1707821.
[34] T. G. Kolda and B. W. Bader, *Tensor decompositions and applications*, SIAM Rev., 51 (2009), pp. 455–500, https://doi.org/10.1137/07070111X.
[35] Y. Li, J. Poulson, and L. Ying, *Distributed-memory $\mathcal{H}$-matrix algebra I: Data distribution and matrix-vector multiplication*, CSIAM Trans. Appl. Math., 2 (2021), pp. 431–459, https://doi.org/https://doi.org/10.4208/csiam-am.2020-0206.
[36] Y. Li and H. Yang, *Interpolative butterfly factorization*, SIAM J. Sci. Comput., 39 (2017), pp. A503–A531, https://doi.org/10.1137/16M1074941. 10.1137/16M1074941.
[37] Y. Li, H. Yang, E. R. Martin, K. L. Ho, and L. Ying, *Butterfly factorization*, Multiscale Model. Simul., 13 (2015), pp. 714–732, https://doi.org/10.1137/15M1007173.
[38] Y. Li, H. Yang, and L. Ying, *Multidimensional butterfly factorization*, Appl. Comput. Harmon. Anal., 44 (2018), pp. 737–758, https://doi.org/10.1016/J.ACHA.2017.04.002.
[39] P.-G. Martinsson, *Fast direct solvers for elliptic PDEs*, Fast direct solvers for elliptic partial differential equations, Society for Industrial Applied Mathematics, 2019.
[40] P.-G. Martinsson and V. Rokhlin, *A fast direct solver for boundary integral equations in two dimensions*, J. Comput. Phys., 205 (2005), pp. 1–23, https://doi.org/10.1016/j.jcp.2004.10.033.
[41] V. Minden, K. L. Ho, A. Damle, and L. Ying, *A recursive skeletonization factorization based on strong admissibility*, Multiscale Model. Simul., 15 (2017), pp. 768–796, https://doi.org/10.1137/16M1095949.
[42] R. Minster, A. K. Saibaba, and M. E. Kilmer, *Randomized algorithms for low-rank tensor decompositions in the Tucker format*, SIAM J. Math. Data Sci., 2 (2020), pp. 189–215, https://doi.org/10.1137/19M1261043.
[43] I. V. Oseledets, *Tensor-train decomposition*, SIAM J. Sci. Comput., 33 (2011), pp. 2295–2317, https://doi.org/10.1137/090752286.
[44] L. N. Trefethen, *Approximation theory and approximation practice*, SIAM, 2019.
[45] N. Vannieuwenhoven, R. Vandebril, and K. Meerbergen, *A new truncation strategy for the higher-order singular value decomposition*, SIAM J. Sci. Comput., 34 (2012), pp. A1027–A1052, https://doi.org/10.1137/110836067.
[46] R. Wang, Y. Li, and E. F. Darve, *On the numerical rank of radial basis function kernels in high dimensions*, SIAM J. Matrix Anal. Appl., 39 (2018), pp. 1810–1835, https://doi.org/10.1137/17M1135803.
[47] R. Wang, Y. Li, M. W. Mahoney, and E. Darve, *Block basis factorization for scalable kernel evaluation*, SIAM J. Matrix Anal. Appl., 40 (2019), pp. 1497–1526, https://doi.org/

10.1137/18M1212586.

[48] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, *Fast algorithms for hierarchically semiseparable matrices*, Numer. Linear Algebra Appl., 17 (2010), pp. 953–976, https://doi.org/10.1002/nla.691.

[49] L. Ying, G. Biros, and D. Zorin, *A kernel-independent adaptive fast multipole algorithm in two and three dimensions*, J. Comput. Phys., 196 (2004), pp. 591–626, https://doi.org/10.1016/j.jcp.2003.11.021.