A LINEAR-COMPLEXITY TENSOR BUTTERFLY ALGORITHM FOR COMPRESSING HIGH-DIMENSIONAL OSCILLATORY INTEGRAL OPERATORS

P. MICHAEL KIELSTRA*, TIANYI SHI†, HENGRUI LUO ‡, JIANLIANG QIAN §, AND YANG LIU†

Abstract. This paper presents a multilevel tensor compression algorithm called tensor butterfly algorithm for efficiently representing large-scale and high-dimensional oscillatory integral operators, including Green's functions for wave equations and integral transforms such as Radon transforms and Fourier transforms. The proposed algorithm leverages a tensor extension of the so-called complementary low-rank property of existing matrix butterfly algorithms. The algorithm partitions the discretized integral operator tensor into subtensors of multiple levels, and factorizes each subtensor at the middle level as a Tucker-type interpolative decomposition, whose factor matrices are formed in a multilevel fashion. For a d-dimensional ($d > 1$) integral operator discretized into a $2d$-mode tensor with n^{2d} entries, the overall CPU time and memory requirement scale as $O(n^d)$, in stark contrast to the $O(n^d \log n)$ complexity of existing matrix algorithms such as matrix butterfly algorithms and fast Fourier transforms (FFT), where n is the number of points per direction. When comparing with other tensor algorithms such as quantized tensor train (QTT), the proposed algorithm also shows superior CPU and memory performance for tensor contraction. Remarkably, the tensor butterfly algorithm can efficiently model high-frequency Green's function interactions between two unit cubes, each spanning 512 wavelengths per direction, which represent problems of scale over 512× larger than that existing butterfly algorithms can handle, with the same amount of computation resources. On the other hand, for a problem representing 64 wavelengths per direction, which is the largest size existing algorithms can handle, our tensor butterfly algorithm exhibits 200x speedups and 30× memory reduction comparing with existing ones. Moreover, the tensor butterfly algorithm also permits $O(n^d)$-complexity FFTs and Radon transforms up to $d = 6$ dimensions.

Key word. butterfly algorithm, tensor algorithm, Tucker decomposition, interpolative decomposition, quantized tensor train (QTT), fast Fourier transforms (FFT), fast algorithm, high-frequency wave equations, integral transforms, Radon transform, low-rank compression, Fourier integral operator, non-uniform FFT (NUFFT)

AMS subject classifications. 15A23, 65F50, 65R10, 65R20

1. Introduction.

Oscillatory integral operators (OIOs), such as Fourier transforms and Fourier integral operators [32, 7], are critical computational and theoretical tools for many scientific and engineering applications, such as signal and image processing, inverse problems and imaging, computer vision, quantum mechanics, and analyzing and solving partial differential equations (PDEs). The development of accurate and efficient algorithms for computing OIOs has profound impacts on the evolution of the pertinent research areas including, perhaps mostly remarkably, the invention of the fast Fourier transform (FFT) by Cooley and Tukey in 1965 and the invention of the fast multipole method (FMM) by Greengard and Rokhlin in 1987, both of which were listed among the ten most significant algorithms discovered in the 20th century. Among existing analytical and algebraic methods for OIOs, butterfly algorithms [53, 47, 37, 36, 58] represent an emerging class of multilevel matrix decompo-

*Department of Mathematics, University of California, Berkeley, CA, USA. Email: `pmkielstra@berkeley.edu`

†Applied Mathematics and Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA, USA. Email: {`tianyishi,liuyangzhuan`}`@lbl.gov`

‡Department of Statistics, Rice University, Houston, TX, USA. Email: `hrluo@rice.edu`

§Department of Mathematics and Department of CMSE, Michigan State University, East Lansing, MI, USA. Email: `jqian@msu.edu`

sition algorithms that have been proposed for Fourier transforms and Fourier integral operators [8, 70, 69], special function transforms [65, 4, 56], fast iterative [54, 53, 48] and direct [24, 43, 25, 26, 61, 44, 62] solution of surface and volume integral equations for wave equations, high-frequency Green's function ansatz for inhomogeneous wave equations [45, 41, 49], direct solution of PDE-induced sparse systems [42, 13], and machine learning for inverse problems [33, 35]. The (matrix) butterfly algorithms leverage the so-called complementary low-rank (CLR) property of the matrix representation of OIOs after proper row/column permutation. The CLR states that any submatrix with contiguous row and column index sets exhibits a low numerical rank if the number of the submatrix entries approximately equals the matrix size. These ranks are known as the butterfly ranks, which stay constant irrespective of the matrix sizes. This permits a multilevel sparse matrix decomposition requiring $O(n \log n)$ factorization time, application time, and storage units with n being the matrix size.

Despite their low asymptotic complexity, the matrix butterfly algorithms oftentimes exhibit relatively large prefactors, i.e., constant but high butterfly ranks, particularly for higher-dimensional OIOs. Examples include Green's functions for 3D high-frequency wave equations [61, 45], 3D Radon transforms for linear inverse problems [17], 6D Fourier–Bros–Iagolnitzer transforms for Wigner equations [15, 68], 6D Fourier transforms in diffusion magnetic resonance imaging [11] and plasma physics [18], 4D space-time transforms in quantum field theories [59, 50], and multi-particle Green's functions in quantum chemistry [21]. For these high-dimensional OIOs, the computational advantage of the matrix butterfly algorithms over other existing algorithms becomes significant only for very large matrices.

More broadly speaking, for large-scale multi-dimensional scientific data and operators, tensor algorithms are typically more efficient than matrix algorithms. Popular low-rank tensor compression algorithms include CANDECOMP/PARAFAC [30], Tucker [16], hierarchical Tucker [28], tensor train (TT) [57], and tensor network [12] decomposition algorithms. See references [34, 23] for a more complete review of available tensor formats and their applications. When applied to the representation of high-dimensional integral operators, tensor algorithms often leverage additional translational- or scaling-invariance property to achieve superior compression performance, including solution of quasi-static wave equations [67, 66, 22, 14], elliptic PDEs [3, 27], many-body Schrödinger equations [31], and quantum Fourier transforms (QFTs) [9]. That being said, most existing tensor decomposition algorithms will break down for OIOs due to their incapability to exploit the oscillatory structure of these operators; therefore, new tensor algorithms are called for.

In this paper, we propose a linear-complexity, low-prefactor tensor decomposition algorithm for large-scale and high-dimensional OIOs. This new tensor algorithm, henceforth dubbed the tensor butterfly algorithm, leverages the intrinsic CLR property of high-dimensional OIOs more effectively than the matrix butterfly algorithm, which is enabled by additional tensor properties such as translational invariance of free-space Green's functions and dimensional separability of Fourier transforms. The algorithm partitions the OIO tensor into subtensors of multiple levels, and factorizes each subtensor at the middle level as a Tucker-type interpolative decomposition, whose factor matrices are further constructed in a nested fashion. For a d-dimensional OIO (assuming a constant $d > 1$) discretized as a $2d$-mode tensor with n being the size per mode, the factorization time, application time, and storage cost scale as $O(n^d)$, and the resulting tensor factors have small multi-linear ranks. This is in stark contrast both to the $O(n^d \log n)$ scaling of existing matrix algorithms such as matrix butterfly algorithms and FFTs, and to the super-linear scaling of existing tensor algorithms. We mention that the linear complexity of the factorization time in our proposed algorithm is achieved via a simple random entry evaluation scheme, assuming that any arbitrary entry can be computed in $O(1)$ time. We remark that, for 3D high-frequency wave equations, the proposed tensor butterfly algorithm can handle $512\times$ larger discretized Green's function tensors than existing butterfly algorithms using the same amount of computation resources; on the other hand, for the largest sized tensor that can be handled by existing butterfly algorithms, our tensor butterfly algorithm is $200\times$ faster than existing ones. Moreover, we claim that the tensor butterfly algorithm instantiates the first linear-complexity implementation of high-dimensional FFTs for arbitrary input data.

1.1. Related Work. *Multi-dimensional butterfly algorithms* represent a version of matrix butterfly algorithms designed for high-dimensional OIOs [38, 10]. Instead of the traditional binary tree partitioning of the matrix rows/columns [53], these algorithms can be viewed as a modern version of [54] that permits quadtree and octree partitioning of the matrix rows/columns, which have been demonstrated on 2D and 3D OIOs. For a general d-dimensional OIO, the d-dimensional tree partitioning leads to a butterfly factorization with a d-fold reduction in the number of levels compared to the binary tree partitioning. That said, the binary tree based butterfly algorithms are easier to implement and exhibit very competitive overall costs comparing with the multi-dimensional butterfly algorithms. We note that both the multi-dimensional and binary tree-based butterfly algorithms are still matrix-based algorithms that scale as $O(n^d \log n)$, as opposed to the proposed tensor algorithm that scales as $O(n^d)$.

Quantized tensor train (QTT) algorithms, or simply TT algorithms, are tensor algorithms well-suited for very high-dimensional integral operators. They have been proposed to compress volume integral operators [14] arising from quasi-static wave equations and static PDEs with $O(\log n)$ memory and CPU complexities. However, for high-frequency wave equations, the QTT rank scales proportionally to the wave number [14] leading to deteriorated CPU and memory complexities (see our numerical results in Section section 4). Moreover, QTT has been proposed for computing FFT and QFT with $O(\log n)$ memory and CPU complexities [9]. However, after obtaining the QTT-compressed formats of both the volume-integral operator and the Fourier transform, the CPU complexity for contracting such a QTT compressed operator with arbitrary (i.e., non QTT-compressed) input data scales super-linearly. In contrast, our algorithm yields a linear CPU and memory complexity for the contraction operation.

1.2. Contents. In what follows, we first review the matrix low-rank decomposition and butterfly decomposition algorithms in section 2. In subsection 3.1, we introduce the Tucker-type interpolative decomposition algorithm as the building block for the proposed tensor butterfly algorithm detailed in subsection 3.2. The multilinear butterfly ranks for a few special cases are analyzed in subsection 3.2.1 and the complete complexity analysis is given in subsection 3.2.2. Section 4 shows a variety of numerical examples, including Green's functions for wave equations, Radon transforms, and uniform and non-uniform discrete Fourier transforms, to demonstrate the performance of matrix butterfly, tensor butterfly, Tucker and QTT algorithms.

1.3. Notations. Given a scalar-valued function $f(x)$, its integral transform is defined as

$$g(x) = \int_y K(x,y)f(y)dy \tag{1.1}$$

with an integral kernel $K(x,y)$. The indexing of a matrix $\mathbf{K}$ is denoted by $\mathbf{K}(i,j)$ or $\mathbf{K}(t,s)$, where i,j are indices and t,s are index sets. We use $\mathbf{K}^T$ to denote the transpose of matrix $\mathbf{K}$. For a sequence of matrices $\mathbf{K}_1,\ldots,\mathbf{K}_n$, the matrix product is

$$\prod_{i=1}^{n}\mathbf{K}_i=\mathbf{K}_1\mathbf{K}_2\ldots\mathbf{K}_n, \tag{1.2}$$

the vertical stacking (assuming the same column dimension) is

$$[\mathbf{K}_i]_i=[\mathbf{K}_1;\mathbf{K}_2;\ldots;\mathbf{K}_n], \tag{1.3}$$

and

$$\mathrm{diag}_i(\mathbf{K}_i)=\mathrm{diag}(\mathbf{K}_1,\mathbf{K}_2,\ldots,\mathbf{K}_n) \tag{1.4}$$

is a block diagonal matrix with $\mathbf{K}_i$ being the diagonal blocks. Given an L-level binary-tree partitioning $\mathcal{T}_t$ of an index set $t=\{1,2,\cdots,n\}$, any node τ at each level is a subset of t. The parent and children of τ are denoted by p_τ and τ^c $(c=1,2)$, respectively, and $\tau=\tau^1\cup\tau^2$.

A multi-index $\boldsymbol{i}=(i_1,\cdots,i_d)$ is a tuple of indices, and similarly a multi-set $\boldsymbol{\tau}=(\tau_1,\tau_2,\cdots,\tau_d)$ is a tuple of index sets. We define

$$\boldsymbol{\tau}_{k\leftarrow t}=(\tau_1,\tau_2,\cdots,\tau_{k-1},t,\tau_{k+1},\tau_{k+2},\cdots,\tau_d). \tag{1.5}$$

Given a tuple of nodes (i.e. a multi-set) $\boldsymbol{\tau}=(\tau_1,\tau_2,\cdots,\tau_d)$ and a multi-index $\boldsymbol{c}=(c_1,c_2,\cdots,c_d)$ with $c_i\in\{1,2\}$, the children of $\boldsymbol{\tau}$ are denoted $\boldsymbol{\tau}^{\boldsymbol{c}}=({\tau_1}^{c_1},{\tau_2}^{c_2},\cdots,{\tau_d}^{c_d})$ and the parents of τ_i, $i=1,2,\cdots,d$ can be simply written as $\boldsymbol{p_\tau}=(p_{\tau_1},p_{\tau_2},\cdots,p_{\tau_d})$. Similar to the above-described notations, we can replace the index i in $[\mathbf{K}_i]_i$ and $\mathrm{diag}_i(\mathbf{K}_i)$ with an index set τ, a multi-index $\boldsymbol{c}$, or a multi-set $\boldsymbol{\tau}$ assuming certain predefined index ordering.

Given complex-valued (or real-valued) functions $f(x)$ of d variables and integral operators $K(x,y)$, the tensor representations of their discretizations are respectively denoted by $\boldsymbol{\mathcal{F}}\in\mathbb{C}^{n_1\times n_2\times\cdots\times n_d}$ and $\boldsymbol{\mathcal{K}}\in\mathbb{C}^{m_1\times m_2\times\cdots\times m_d\times n_1\times n_2\times\cdots\times n_d}$, where $n_1,\cdots,n_d$ and $m_1,\cdots,m_d$ are sizes of discretizations for the corresponding variables. In this paper, we use *matricization* to denote the reshaping of $\boldsymbol{\mathcal{K}}$ into a $(\Pi_k m_k)\times(\Pi_k n_k)$ matrix, and the reshaping of $\boldsymbol{\mathcal{F}}$ into a $(\Pi_k n_k)\times 1$ matrix. The entries of $\boldsymbol{\mathcal{F}}$ and $\boldsymbol{\mathcal{K}}$ are denoted by $\boldsymbol{\mathcal{F}}(\boldsymbol{i})$ (or equivalently $\boldsymbol{\mathcal{F}}(i_1,i_2,\cdots,i_d)$) and $\boldsymbol{\mathcal{K}}(\boldsymbol{i},\boldsymbol{j})$, respectively. Similarly the subtensors are denoted by $\boldsymbol{\mathcal{F}}(\boldsymbol{\tau})$ (or equivalently $\boldsymbol{\mathcal{F}}(\tau_1,\tau_2,\cdots,\tau_d)$) and $\boldsymbol{\mathcal{K}}(\boldsymbol{\tau},\boldsymbol{\nu})$.

Given a d-mode tensor $\boldsymbol{\mathcal{F}}\in\mathbb{C}^{n_1\times n_2\times\cdots\times n_d}$, the mode-$j$ unfolding is denoted by $\mathbf{F}^{(j)}\in\mathbb{C}^{(\Pi_{k\neq j}n_k)\times n_j}$, the mode-$j$ tensor-matrix product of $\boldsymbol{\mathcal{F}}$ with a matrix $\mathbf{X}\in\mathbb{C}^{m\times n_j}$ is denoted by $\boldsymbol{\mathcal{Y}}=\boldsymbol{\mathcal{F}}\times_j\mathbf{X}$, or equivalently $\mathbf{Y}^{(j)}=\mathbf{F}^{(j)}\mathbf{X}^T$.

2. Review of Matrix Algorithms. We consider a d-dimensional OIO kernel $K(x,y)$ with $x,y\in\mathbb{R}^d$ discretized on point pairs x^i and y^j, $i=1,2,...,(m_1m_2\cdot\cdot\cdot m_d)$, $j=1,2,...,(n_1n_2\cdots n_d)$, where i (and similarly j) is the flattening of the corresponding multi-index $\boldsymbol{i}$. Such a discretization can be represented as a matrix $\mathbf{K}\in\mathbb{C}^{(m_1m_2\cdots m_d)\times(n_1n_2\cdots n_d)}$. When it is clear in the context, we assume that $m_k=n_k=n$ for $k=1,\ldots,d$. Throughout this paper, we assume that $\mathbf{K}$ (and its tensor representation) is never fully formed, but instead a function is provided to evaluate any matrix (or tensor) entry in $O(1)$ time. Next we review matrix compression algorithms for $\mathbf{K}$ including low-rank and butterfly algorithms.

2.1. Interpolative Decomposition. The interpolative decomposition (ID) algorithm [29, 39] is a matrix compression technique that constructs a low-rank decomposition whose factors contain original entries of the matrix. More specifically, consider the matrix $\mathbf{K}(\tau,\nu) \in \mathbb{C}^{m\times n}$, $\tau = \{1,2,\ldots,m\}$, $\nu = \{1,2,\ldots,n\}$, the column ID of $\mathbf{K}$ (the index sets τ and ν are omitted for clarity in context) is

$$\mathbf{K} \approx \mathbf{K}(:,\overline{\nu})\mathbf{V}, \tag{2.1}$$

where the skeleton matrix $\mathbf{K}(:,\overline{\nu})$ contains r skeleton columns indexed by $\overline{\nu} \subseteq \nu$ and the interpolation matrix $\mathbf{V}$ has bounded entries. Here the numerical rank r is chosen such that

$$\|\mathbf{K} - \mathbf{K}(:,\overline{\nu})\mathbf{V}\|_F^2 \leqslant O(\epsilon^2)\|\mathbf{K}\|_F^2 \tag{2.2}$$

for a prescribed relative tolerance ϵ. In practice, the column ID can be computed via rank-revealing QR decomposition with a relative tolerance ϵ [39]. Similarly, the row ID of the matrix $\mathbf{K}$ reads

$$\mathbf{K} \approx \mathbf{U}\mathbf{K}(\overline{\tau},:), \tag{2.3}$$

where the skeleton matrix $\mathbf{K}(\overline{\tau},:)$ contains r skeleton rows indexed by $\overline{\tau} \subseteq \tau$ and the interpolation matrix $\mathbf{U}$ has bounded entries. The row ID can be simply computed by the column ID of $\mathbf{K}^T$. Combining the column and row ID in (2.1) and (2.3) gives

$$\mathbf{K} \approx \mathbf{U}\mathbf{K}(\overline{\tau},\overline{\nu})\mathbf{V}. \tag{2.4}$$

It is straightforward to note that the memory and CPU complexities of ID scale as $O(nr)$ and $O(n^2r)$, respectively. The CPU complexity can be reduced to $O(nr^2)$ when properly selected proxy rows in (2.1) and columns in (2.3) are used in the rank-revealing QR. Common strategies of choosing proxy rows/columns (henceforth called *proxy index* strategies) for integral operators include evenly spaced or uniform random samples, and more generally the use of Chebyshev nodes and proxy surfaces (where new rows $K(\overline{x}, y^j)$ other than original rows of $\mathbf{K}$ are used with $\overline{x}$ denoting the proxies). However, for large OIOs (e.g., Green's functions of high-frequency wave equations discretized with a small number of points per wavelength), the rank r depends on the size n of the matrix; consequently, ID is not an efficient compression algorithm. Next, we review the matrix butterfly algorithm capable of achieving quasi-linear memory and CPU complexities for OIOs.

2.2. Matrix Butterfly Algorithm. For reasons discussed in subsection 1.1, we only consider the binary tree based matrix butterfly algorithm as the reference algorithm for the proposed tensor butterfly algorithm throughout this paper. Let $t^0 = \{1,2,\cdots,m\}$ and $s^0 = \{1,2,\cdots,n\}$. Without loss of generality, we assume that $m = n$. The L-level butterfly representation of the discretized OIO $\mathbf{K}(t^0,s^0)$ is based on two binary trees, $\mathcal{T}_{t^0}$ and $\mathcal{T}_{s^0}$, and the CLR property of the OIO takes the following form: at any level $0 \le l \le L$, for any node τ at level l of $\mathcal{T}_{t^0}$ and any node ν at level $L-l$ of $\mathcal{T}_{s^0}$, the subblock $\mathbf{K}(\tau,\nu)$ is numerically low-rank with rank $r_{\tau,\nu}$ bounded by a small number r called the butterfly rank [47, 36, 37, 58].

For any subblock $\mathbf{K}(\tau,\nu)$, the ID in (2.4) permits

$$\mathbf{K}(\tau,\nu) \approx \mathbf{U}_{\tau,\nu}\mathbf{K}(\overline{\tau},\overline{\nu})\mathbf{V}_{\tau,\nu}, \tag{2.5}$$

where the skeleton rows and columns are indexed by $\overline{\tau}$ and $\overline{\nu}$, respectively. It is worth noting that given a node ν, the selection of skeleton columns $\overline{\nu}$ depends on the node

τ. However, the notation $\bar{\cdot}$ does not reflect the dependency when it is clear in the context. By CLR, there are at most r skeleton rows and columns.

Without loss of generality, we assume that L is an even number so that $L^c = L/2$ denotes the middle level. At levels $l = 0, \ldots, L^c$, the interpolation matrices $\mathbf{V}_{\tau,\nu}$ are computed as follows:

At level $l = 0$, $\mathbf{V}_{\tau,\nu}$ are explicitly formed. While at level $0 < l \le L^c$, they are represented in a nested fashion. To see this, consider a node pair (τ, ν) at level $l > 0$ and let ν^1, ν^2 and p_τ be the children and parent of ν and τ, respectively. Let s be the ancestor of ν at level L^c of $\mathcal{T}_{s^0}$ and let $\mathcal{T}_s$ denote the subtree rooted at s.

By (2.4), we have

$$\begin{aligned}
\mathbf{K}(\tau,\nu) &= \begin{bmatrix}\mathbf{K}(\tau,\nu^1) & \mathbf{K}(\tau,\nu^2)\end{bmatrix} \\
&\approx \begin{bmatrix}\mathbf{K}(\tau,\overline{\nu^1}) & \mathbf{K}(\tau,\overline{\nu^2})\end{bmatrix} \begin{bmatrix}\mathbf{V}^s_{p_\tau,\nu^1} & \\ & \mathbf{V}^s_{p_\tau,\nu^2}\end{bmatrix} && (2.6) \\
&\approx \mathbf{K}(\tau,\overline{\nu})\mathbf{W}^s_{\tau,\nu} \begin{bmatrix}\mathbf{V}^s_{p_\tau,\nu^1} & \\ & \mathbf{V}^s_{p_\tau,\nu^2}\end{bmatrix}. && (2.7)
\end{aligned}$$

Here $\mathbf{W}^s_{\tau,\nu}$ and $\overline{\nu}$ are the interpolation matrix and skeleton columns from the ID of $\mathbf{K}(\tau, \overline{\nu^1} \cup \overline{\nu^2})$, respectively. $\mathbf{W}_{\tau,\nu}$ is henceforth referred to as the transfer matrix for ν in the rest of this paper. By CLR, $\mathbf{W}_{\tau,\nu}$ is of sizes at most $r \times 2r$. Note that we have added an additional superscript s to $\mathbf{V}_{p_\tau,\nu^c}$ and $\mathbf{W}_{\tau,\nu}$, for notation convenience in the later context. From (2.6), it is clear that the interpolation matrix $\mathbf{V}^s_{\tau,\nu}$ can be expressed in terms of its parent p_τ's and children ν^1, ν^2's interpolation matrices as

$$\mathbf{V}^s_{\tau,\nu} = \mathbf{W}^s_{\tau,\nu} \begin{bmatrix}\mathbf{V}^s_{p_\tau,\nu^1} & \\ & \mathbf{V}^s_{p_\tau,\nu^2}\end{bmatrix}. \tag{2.8}$$

Note that the interpolation matrices $\mathbf{V}^s_{\tau,\nu}$ at level $l = 0$ and transfer matrices $\mathbf{W}^s_{\tau,\nu}$ at level $0 < l \le L^c$ do not require the column ID on the full subblocks $\mathbf{K}(\tau,\nu)$ and $\mathbf{K}(\tau, \overline{\nu^1} \cup \overline{\nu^2})$, which would lead to at least an $O(mn)$ computational complexity.

In practice, one can select $O(r_{\tau,\nu})$ proxy rows $\hat{\tau} \subset \tau$ to compute $\mathbf{V}^s_{\tau,\nu}$ and $\mathbf{W}^s_{\tau,\nu}$ via ID as:

$$\mathbf{K}(\hat{\tau},\nu) \approx \mathbf{K}(\hat{\tau},\overline{\nu})\mathbf{V}^s_{\tau,\nu}, \quad l = 0, \tag{2.9}$$

$$\mathbf{K}(\hat{\tau},\overline{\nu^1} \cup \overline{\nu^2}) \approx \mathbf{K}(\hat{\tau},\overline{\nu})\mathbf{W}^s_{\tau,\nu}, \quad 0 < l \le L^c. \tag{2.10}$$

The viable choices for proxy rows have been discussed in several existing papers [45, 58, 61, 8].

At levels $l = L^c, \ldots, L$, the interpolation matrices $\mathbf{U}_{\tau,\nu}$ are computed by performing similar operations on $\mathbf{K}^T$. We only provide their expressions here and omit the redundant explanation. Let t be the ancestor of ν at level L^c of $\mathcal{T}_{t^0}$ and let $\mathcal{T}_t$ be the subtree rooted at t. At level $l = L$, $\mathbf{U}^t_{\tau,\nu}$ are explicitly formed. At level $L^c \le l < L$, only the transfer matrices $\mathbf{P}^t_{\tau,\nu}$ are computed from the column ID of $\mathbf{K}^T(\nu, \overline{\tau^1} \cup \overline{\tau^2})$ satisfying

$$\mathbf{U}^t_{\tau,\nu} = \begin{bmatrix}\mathbf{U}^t_{\tau^1,p_\nu} & \\ & \mathbf{U}^t_{\tau^2,p_\nu}\end{bmatrix} \mathbf{P}^t_{\tau,\nu}. \tag{2.11}$$

Combining (2.5), (2.8) and (2.11), the matrix butterfly decomposition can be

Meaning	Matrix butterfly	Tensor butterfly
Butterfly rank	r_m	r_t
Set/multi-set	τ, ν	$\boldsymbol{\tau}, \boldsymbol{\nu}$
k^{th} set of multi-set	-	τ_k, ν_k
Parent set/multi-set	p_τ	$\boldsymbol{p_\tau}$
Children set/multi-set	τ^c	$\boldsymbol{\tau^c}$
Root-level set/multi-set	t^0, s^0	$\boldsymbol{t}^0, \boldsymbol{s}^0$
Mid-level set/multi-set	t, s	$\boldsymbol{t}, \boldsymbol{s}$
Binary tree	$\mathcal{T}_{t^0}, \mathcal{T}_{s^0}$	$\mathcal{T}_{t_k^0}, \mathcal{T}_{s_k^0}$
Cardinality of leaf nodes	C_b^d	C_b
Cardinality of root nodes	n^d	n
Mid-level submatrix/subtensor	$\mathbf{K}(\bar{t}, \bar{s})$	$\boldsymbol{\mathcal{K}}(\bar{\boldsymbol{t}}, \bar{\boldsymbol{s}})$
Interpolation matrix	$\mathbf{V}^s_{\tau,\nu}, \mathbf{U}^t_{\tau,\nu}$	$\mathbf{V}^{\boldsymbol{s},k}_{\boldsymbol{\tau},\nu}, \mathbf{U}^{\boldsymbol{t},k}_{\tau,\boldsymbol{\nu}}$
Transfer matrix	$\mathbf{W}^s_{\tau,\nu}, \mathbf{P}^t_{\tau,\nu}$	$\mathbf{W}^{\boldsymbol{s},k}_{\boldsymbol{\tau},\nu}, \mathbf{P}^{\boldsymbol{t},k}_{\tau,\boldsymbol{\nu}}$
Interpolation factor	$\overline{\mathbf{U}}^t, \overline{\mathbf{V}}^s$	$\overline{\mathbf{U}}^{\boldsymbol{t},k}, \overline{\mathbf{V}}^{\boldsymbol{s},k}$
Transfer factor	$\overline{\mathbf{P}}^{t,s}_l, \overline{\mathbf{W}}^{t,s}_l$	$\overline{\mathbf{P}}^{\boldsymbol{t},\boldsymbol{s},k}_l, \overline{\mathbf{W}}^{\boldsymbol{t},\boldsymbol{s},k}_l$

Table 2.1: Notation comparison of the matrix butterfly algorithm in subsection 2.2 and the tensor butterfly algorithm in subsection 3.2. Note that the subscript k in τ_k, ν_k, in the tensor notations of the interpolation/transfer matrix and interpolation/transfer factor for dimension k, is dropped for simplicity throughout this paper.

expressed for each node pair (t, s) at level L^c of $\mathcal{T}_{t^0}$ and $\mathcal{T}_{s^0}$ as

$$\mathbf{K}(t, s) \approx \overline{\mathbf{U}}^t \Big(\prod_{l=1}^{L^c} \overline{\mathbf{P}}^{t,s}_l \Big) \mathbf{K}(\bar{t}, \bar{s}) \Big(\prod_{l=L^c}^{1} \overline{\mathbf{W}}^{t,s}_l \Big) \overline{\mathbf{V}}^s. \tag{2.12}$$

Here, $\bar{t}$ and $\bar{s}$ represent the skeleton rows and columns of the ID of $\mathbf{K}(t, s)$. The interpolation factors $\overline{\mathbf{U}}^t$ and $\overline{\mathbf{V}}^s$ in (2.12) are

$$\overline{\mathbf{U}}^t = \mathrm{diag}_\tau(\mathbf{U}^t_{\tau,s^0}), \quad \tau \text{ at level } L^c \text{ of } \mathcal{T}_t, \tag{2.13}$$

$$\overline{\mathbf{V}}^s = \mathrm{diag}_\nu(\mathbf{V}^s_{t^0,\nu}), \quad \nu \text{ at level } L^c \text{ of } \mathcal{T}_s, \tag{2.14}$$

and the transfer factors $\overline{\mathbf{P}}^{t,s}_l$ and $\overline{\mathbf{W}}^{t,s}_l$ for $l = 1, \ldots, L^c$ consist of transfer matrices $\mathbf{W}^s_{\tau,\nu}$ and $\mathbf{P}^s_{\tau,\nu}$:

$$\overline{\mathbf{W}}^{t,s}_l = \mathrm{diag}_\tau\Big(\big[\mathrm{diag}_\nu(\mathbf{W}^s_{\tau^c,\nu})\big]_c \Big), \quad \begin{array}{l} \tau \text{ at level } l-1 \text{ of } \mathcal{T}_{t^0}, \text{ and } t \subseteq \tau, \\ \nu \text{ at level } L^c - l \text{ of } \mathcal{T}_s; \end{array} \tag{2.15}$$

$$(\overline{\mathbf{P}}^{t,s}_l)^T = \mathrm{diag}_\nu\Big(\big[\mathrm{diag}_\tau((\mathbf{P}^t_{\tau,\nu^c})^T)\big]_c \Big), \quad \begin{array}{l} \tau \text{ at level } L^c - l \text{ of } \mathcal{T}_t, \\ \nu \text{ at level } l-1 \text{ of } \mathcal{T}_{s^0}, \text{ and } s \subseteq \nu. \end{array} \tag{2.16}$$

Here τ^c and ν^c with $c = 1, 2$ are children of τ and ν, respectively. For the ease of comparison with the tensor butterfly algorithm in subsection 3.2, we list some notations of the matrix butterfly algorithm in Table 2.1.

The CPU and memory requirement for computing the matrix butterfly decomposition can be briefly analyzed as follows. Note that we only need to analyze the costs for $\mathbf{V}^s_{\tau,\nu}$, $\mathbf{W}^s_{\tau,\nu}$ and $\mathbf{K}(\bar{t},\bar{s})$ as those for $\mathbf{U}^t_{\tau,\nu}$ and $\mathbf{P}^t_{\tau,\nu}$ are similar. By the CLR assumption, we assume that $r_{\tau,\nu} \leq r, \forall \tau, \nu$ for some constant r. Thanks to the use of the proxy rows and columns, the computation of one individual $\mathbf{V}^s_{\tau,\nu}$ and $\mathbf{W}^s_{\tau,\nu}$ by ID only operates on $O(r) \times O(r)$ matrices, hence its memory and CPU requirements are $O(r^2)$ and $O(r^3)$, respectively. In total, there are $O(2^{L^c})$ middle-level nodes s each having $O(2^{L^c})$ numbers of $\mathbf{V}^s_{\tau,\nu}$ and $O(L^c 2^{L^c})$ numbers of $\mathbf{W}^s_{\tau,\nu}$. Similarly, each $\mathbf{K}(\bar{t},\bar{s})$ requires $O(r^2)$ CPU and memory costs, and there are in total $O(2^L)$ middle-level node pairs (t,s). These numbers sum up to the overall $O(nr^2 \log n)$ memory and $O(nr^3 \log n)$ CPU complexities for matrix butterfly algorithms.

For d-dimensional discretized OIOs $\mathbf{K} \in \mathbb{C}^{(m_1 m_2 \cdots m_d)\times(n_1 n_2 \cdots n_d)}$ with $m_k = n_k = n$, we can assume that $n = C_b 2^L$ with some constant C_b. For the above-described binary-tree-based butterfly algorithm, the leaf nodes of the trees are of size C_b^d and this leads to a dL-level butterfly factorization. The memory and CPU complexities for this algorithm become $O(dn^d r^2 \log n)$ and $O(dn^d r^3 \log n)$, respectively. On the other hand, the multi-dimensional tree-based butterfly algorithm [38, 10] leads to a L-level factorization with $O(2^d n^d r^2 \log n)$ memory and $O(2^d n^d r^3 \log n)$ CPU complexities. In this paper, we only use the binary-tree-based algorithm as the baseline matrix butterfly algorithm. Despite their quasi-linear complexity for high-dimensional OIOs, the butterfly rank r is constant but high, leading to very large prefactors of these binary and multi-dimensional tree-based algorithms. In the following, we turn to tensor decomposition algorithms to reduce both the prefactor and asymptotic scaling of matrix butterfly algorithms. The proposed tensor decomposition explores additional tensor compressibility of high-dimensional OIOs such as translational invariance of free-space Green's functions and dimensional separability of Fourier transforms. As will be clear in the next section, the prefactor (dependent on the butterfly rank) can be reduced by leveraging Tucker decomposition for tensorization of the middle-level submatrices $\mathbf{K}(\bar{t},\bar{s})$ of (2.12). The Tucker decomposition is further factored out along each dimension in a nested fashion by simultaneously moving along the binary tree of that dimension and d binary trees of other dimensions. As a result, the number of transfer matrices becomes dominant only towards the middle level L^c, leading to a factor of $\log n$ reduction in the asymptotic complexity.

3. Proposed Tensor Algorithms.

In this section, we assume that the d-dimensional discretized OIO in section 2 is directly represented as a $2d$-mode tensor $\boldsymbol{\mathcal{K}} \in \mathbb{C}^{m_1 \times m_2 \times \cdots \times m_d \times n_1 \times n_2 \times \cdots \times n_d}$. We first extend the matrix ID algorithm in subsection 2.1 to its tensor variant, which serves as the building block for the proposed tensor butterfly algorithm.

3.1. Tucker-type Interpolative Decomposition.

Given the $2d$-mode tensor $\boldsymbol{\mathcal{K}}(\boldsymbol{\tau},\boldsymbol{\nu})$ with $\tau_k = \{1,2,\ldots,m_k\}$ and $\nu_k = \{1,2,\ldots,n_k\}$ for $k = 1,\ldots,d$, the proposed Tucker-type decomposition compresses each dimension independently via the column ID of the unfolding of $\boldsymbol{\mathcal{K}}$ along the k-th dimension,

$$\mathbf{K}^{(k)} \approx \mathbf{K}^{(k)}(:,\overline{\tau_k})\mathbf{U}^k, \quad \mathbf{K}^{(d+k)} \approx \mathbf{K}^{(d+k)}(:,\overline{\nu_k})\mathbf{V}^k, \quad k = 1,\ldots,d, \tag{3.1}$$

where $\mathbf{K}^{(k)} \in \mathbb{C}^{(\prod_{j\neq k} n_j)\times n_k}$ is the mode-k unfolding, or equivalently

$$\boldsymbol{\mathcal{K}} \approx \boldsymbol{\mathcal{K}}(\boldsymbol{\tau}_{k\leftarrow\overline{\tau_k}},\boldsymbol{\nu}) \times_k \mathbf{U}^k, \quad \boldsymbol{\mathcal{K}} \approx \boldsymbol{\mathcal{K}}(\boldsymbol{\tau},\boldsymbol{\nu}_{k\leftarrow\overline{\nu_k}}) \times_{d+k} \mathbf{V}^k, \quad k = 1,\ldots,d. \tag{3.2}$$

Here, $\overline{\tau_k}$ and $\overline{\nu_k}$ denote the skeleton indices along modes k and $d+k$ of $\boldsymbol{\mathcal{K}}$, respectively, while $\boldsymbol{\tau}_{k\leftarrow\overline{\tau_k}}$ and $\boldsymbol{\nu}_{k\leftarrow\overline{\nu_k}}$ denote multi-sets that replace τ_k and ν_k, respectively, with $\overline{\tau_k}$

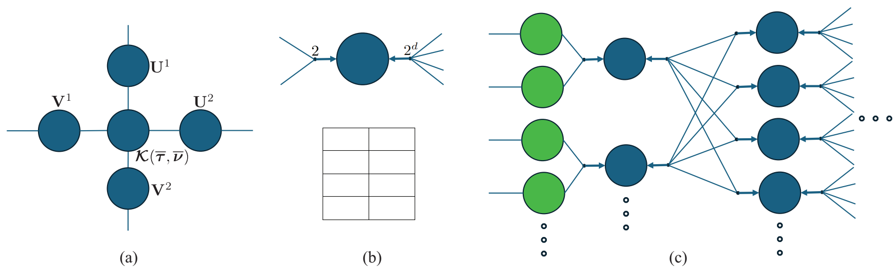

Fig. 3.1: Tensor diagrams for (a) the Tucker-ID decomposition of a 4-mode tensor, and (b) the matrix partitioner corresponding to a $2^d \times 2$ partitioning with $d = 2$ used in the tensor butterfly decomposition of a $2d$-mode tensor, such as $\left[\mathbf{W}^{\boldsymbol{s},k}_{\boldsymbol{\tau}^c,\nu}\right]_{\boldsymbol{c}}$ in (3.14) for fixed $\boldsymbol{s}, \boldsymbol{\tau}, k$ and ν, or $\left[\mathbf{P}^{\boldsymbol{t},k}_{\tau,\boldsymbol{\nu}^c}\right]_{\boldsymbol{c}}$ in (3.13) for fixed $\boldsymbol{t}, \boldsymbol{\nu}, k$ and τ. Here, each of the row and column dimensions is connected to a partitioning node. Each partitioning node has a parent edge with an arrow pointing to the dimension to be partitioned, and several children edges connected to the parent edge. The weight of the parent edge (i.e., the number of columns or rows of the matrix) equals the sum of the weights of the children edges. (c) The tensor diagram involving blocks $\mathbf{V}^{\boldsymbol{s},k}_{\boldsymbol{t}^0,\nu}$ (in green) and blocks $\left[\mathbf{W}^{\boldsymbol{s},k}_{\boldsymbol{\tau}^c,\nu}\right]_{\boldsymbol{c}}$ (in blue) for fixed $\boldsymbol{s}$ and k for the tensor butterfly decomposition of a $2d$-mode tensor.

and $\overline{\nu_k}$. Combining (3.2) for all dimensions yields the following proposed Tucker-type decomposition,

$$\boldsymbol{\mathcal{K}} \approx \boldsymbol{\mathcal{K}}(\overline{\boldsymbol{\tau}},\overline{\boldsymbol{\nu}})\left(\prod_{k=1}^{d} \times_k \mathbf{U}^k\right)\left(\prod_{k=1}^{d} \times_{d+k} \mathbf{V}^k\right), \tag{3.3}$$

where, $\overline{\boldsymbol{\tau}} = (\overline{\tau_1}, \overline{\tau_2}, \dots, \overline{\tau_d})$, $\overline{\boldsymbol{\nu}} = (\overline{\nu_1}, \overline{\nu_2}, \dots, \overline{\nu_d})$, the core tensor $\boldsymbol{\mathcal{K}}(\overline{\boldsymbol{\tau}},\overline{\boldsymbol{\nu}})$ is a subtensor of $\boldsymbol{\mathcal{K}}$, and $\mathbf{U}^k$ and $\mathbf{V}^k$ are the factor matrices for modes k and $d+k$, respectively.

See Figure 3.1(a) for the tensor diagram of (3.3) for a 4-mode tensor, which has the same diagram as other existing Tucker decompositions such as high-order singular value decompositions (HOSVD) [16]. However, unlike HOSVD that leads to orthonormal factor matrices, the proposed decomposition leads to factor matrices with bounded entries and the core tensor with the original tensor entries. Therefore, the proposed decomposition is named Tucker-type interpolative decomposition (Tucker-ID). It is worth noting that there exist several interpolative tensor decomposition algorithms [6, 51, 52, 60, 55]. However they either use original tensor entries in the factor matrices (instead of the core tensor) [51, 60, 6] or rely on a different tensor diagram [52]. Note that the structure-preserving decomposition in [55] is similar to Tucker-ID but relies on sketching instead of proxy indices for the construction. As will be seen in subsection 3.2, the Tucker-ID algorithm is a unique and essential building block of the tensor butterfly algorithm.

Just like HOSVD, one can easily show that if the approximations in (3.1) hold true up to a predefined relative compression tolerance ϵ as

$$\begin{aligned} &||\mathbf{K}^{(k)} - \mathbf{K}^{(k)}(:,\overline{\tau_k})\mathbf{U}^k||_F \le \epsilon ||\boldsymbol{\mathcal{K}}||_F, \quad k = 1,\dots,d,\\ &||\mathbf{K}^{(d+k)} - \mathbf{K}^{(d+k)}(:,\overline{\nu_k})\mathbf{V}^k||_F \le \epsilon ||\boldsymbol{\mathcal{K}}||_F, \quad k = 1,\dots,d, \end{aligned} \tag{3.4}$$

then the Tucker-ID of (3.3) satisfies

$$\left|\left|\mathcal{K}-\mathcal{K}(\overline{\boldsymbol{\tau}},\overline{\boldsymbol{\nu}})\left(\prod_{k=1}^{d}\times_k \mathbf{U}^k\right)\left(\prod_{k=1}^{d}\times_{d+k}\mathbf{V}^k\right)\right|\right|_F \leq \epsilon\sqrt{2d}||\mathcal{K}||_F. \tag{3.5}$$

The memory and CPU complexities of Tucker-ID can be briefly analyzed as follows. Assuming that $m_k = n_k = n$ and $\max_k |\overline{\tau}_k|$=$\max_k |\overline{\nu}_k| = r$ is a constant (we will discuss the case of non-constant r in subsection 3.2.3), the memory requirement is simply $O(drn + r^{2d})$, where the first and second term represent the storage units for the factor matrices and the core tensor, respectively. The CPU cost for naive computation of Tucker-ID is $O(drn^{2d} + r^{2d})$, where the first term represents the cost of rank-revealing QR of the unfolding matrices in (3.1), and the second term represents the cost forming the core tensor $\mathcal{K}(\overline{\boldsymbol{\tau}},\overline{\boldsymbol{\nu}})$. In practice, however, the unfolding matrices do not need to be fully formed and one can leverage the idea of proxy rows in subsection 2.2 to reduce the cost for computing the factor matrices to $O(dnr^{2d})$. We will explain this in more detail in the context of the proposed tensor butterfly decomposition algorithm.

Just like the matrix ID algorithm, Tucker-ID is also not suitable for representing large-sized OIOs as the rank r depends on the size n. That said, the Tucker-ID rank is typically significantly smaller than the matrix ID rank, as it exploits more compressibility properties across dimensions by leveraging e.g. translational-invariance or dimensional-separability properties of OIOs; see subsection 3.2.1 for a few of such examples. In what follows, we use Tucker-ID as the building block for constructing a linear-complexity tensor butterfly decomposition algorithm for large-sized OIOs.

3.2. Tensor Butterfly Algorithm. Consider a 2d-mode OIO tensor $\mathcal{K}(\boldsymbol{t}^0,\boldsymbol{s}^0)$ with $\boldsymbol{t}^0 = (t_1^0, t_1^0, \ldots, t_d^0)$, $\boldsymbol{s}^0 = (s_1^0, s_1^0, \ldots, s_d^0)$, $t_k^0 = \{1, 2, \ldots, m_k\}$, $s_k^0 = \{1, 2, \ldots, n_k\}$, $k = 1, 2, \ldots, d$. Without loss of generality, we assume that $m_k = n_k = n$. We further assume that each t_k^0 (and s_k^0) is binary partitioned with a tree $\mathcal{T}_{t_k^0}$ (and $\mathcal{T}_{s_k^0}$) of L levels for $k = 1, 2, \ldots, d$.

To start with, we first define the tensor CLR property as follows:

- For any level $0 \leq l \leq L^c$, any multi-set $\boldsymbol{\tau} = (\tau_1, \tau_2, \ldots, \tau_d)$ with $\tau_i, i \leq d$ at level l of $\mathcal{T}_{t_i^0}$, any multi-set $\boldsymbol{s} = (s_1, s_2, \ldots, s_d)$ with $s_i, i \leq d$ at level L^c of $\mathcal{T}_{s_i^0}$, any mode $1 \leq k \leq d$, and any node ν at level $L^c - l$ of $\mathcal{T}_{s_k}$, the mode-$(d+k)$ unfolding of the subtensor $\mathcal{K}(\boldsymbol{\tau}, \boldsymbol{s}_{k\leftarrow\nu})$ is numerically low-rank (with rank bounded by r), permitting an ID via (3.2):

$$\mathcal{K}(\boldsymbol{\tau}, \boldsymbol{s}_{k\leftarrow\nu}) \approx \mathcal{K}(\boldsymbol{\tau}, \boldsymbol{s}_{k\leftarrow\overline{\nu}}) \times_{d+k} \mathbf{V}^{\boldsymbol{s},k}_{\boldsymbol{\tau},\nu}. \tag{3.6}$$

- For any level $0 \leq l \leq L^c$, any multi-set $\boldsymbol{\nu} = (\nu_1, \nu_2, \ldots, \nu_d)$ with $\nu_i, i \leq d$ at level l of $\mathcal{T}_{s_i^0}$, any multi-set $\boldsymbol{t} = (t_1, t_2, \ldots, t_d)$ with $t_i, i \leq d$ at level L^c of $\mathcal{T}_{t_i^0}$, any mode $1 \leq k \leq d$, and any node τ at level $L^c - l$ of $\mathcal{T}_{t_k}$, the mode-k unfolding of the subtensor $\mathcal{K}(\boldsymbol{t}_{k\leftarrow\tau}, \boldsymbol{\nu})$ is numerically low-rank (with rank bounded by r), permitting an ID via (3.2):

$$\mathcal{K}(\boldsymbol{t}_{k\leftarrow\tau}, \boldsymbol{\nu}) \approx \mathcal{K}(\boldsymbol{t}_{k\leftarrow\overline{\tau}}, \boldsymbol{\nu}) \times_k \mathbf{U}^{\boldsymbol{t},k}_{\tau,\boldsymbol{\nu}}. \tag{3.7}$$

In essence, the tensor CLR in (3.6) and (3.7) investigates the unfolding of judiciously selected subtensors rather than the matricization used in the matrix CLR. Moreover, the tensor CLR requires fixing $d-1$ modes of the $2d$-mode subtensors to be of size $O(\sqrt{n})$ while changing the remaining $d+1$ modes with respect to l. Therefore each

ID computation can operate on larger subtensors compared to the matrix CLR. In subsection 3.2.1 we provide two examples, namely a free-space Green's function tensor and a high-dimensional Fourier transform, to explain why the tensor CLR is valid, and in subsection 3.2.2 we will see that the tensor CLR essentially reduces the quasilinear complexity of the matrix butterfly algorithm to linear complexity. Here, assuming that the tensor CLR holds true, we describe the tensor butterfly algorithm. We note that there may be alternative ways to define the tensor CLR different from (3.6) and (3.7), and we leave that as a future work. To avoid notation confusion, we list some notations of the tensor butterfly algorithm in Table 2.1.

In what follows, we focus on the computation of $\mathbf{V}^{\boldsymbol{s},k}_{\boldsymbol{\tau},\nu}$ (corresponding to the mid-level multi-set $\boldsymbol{s}$), as $\mathbf{U}^{\boldsymbol{t},k}_{\tau,\boldsymbol{\nu}}$ (corresponding to the mid-level multi-set $\boldsymbol{t}$) can be computed in a similar fashion. At level $l=0$, $\mathbf{V}^{\boldsymbol{s},k}_{\boldsymbol{\tau},\nu}$ are explicitly formed. At level $0<l\leq L^c$, they are represented in a nested fashion. Let $\boldsymbol{p_\tau}=(p_{\tau_1},p_{\tau_2},\ldots,p_{\tau_d})$ consist of parents of $\boldsymbol{\tau}=(\tau_1,\tau_2,\ldots,\tau_d)$ in (3.6).

By the tensor CLR property, we have

$$
\begin{aligned}
\boldsymbol{\mathcal{K}}(\boldsymbol{\tau},\boldsymbol{s}_{k\leftarrow\nu}) &\approx \boldsymbol{\mathcal{K}}(\boldsymbol{\tau},\boldsymbol{s}_{k\leftarrow\overline{\nu^1}\cup\overline{\nu^2}})\times_{d+k}\begin{bmatrix}\mathbf{V}^{\boldsymbol{s},k}_{\boldsymbol{p_\tau},\nu^1} & \\ & \mathbf{V}^{\boldsymbol{s},k}_{\boldsymbol{p_\tau},\nu^2}\end{bmatrix}\\
&\approx \boldsymbol{\mathcal{K}}(\boldsymbol{\tau},\boldsymbol{s}_{k\leftarrow\overline{\nu}})\times_{d+k}\left(\mathbf{W}^{\boldsymbol{s},k}_{\boldsymbol{\tau},\nu}\begin{bmatrix}\mathbf{V}^{\boldsymbol{s},k}_{\boldsymbol{p_\tau},\nu^1} & \\ & \mathbf{V}^{\boldsymbol{s},k}_{\boldsymbol{p_\tau},\nu^2}\end{bmatrix}\right).
\end{aligned}
\tag{3.8}
$$

Comparing (3.8) and (3.6), one realizes that the interpolation matrix $\mathbf{V}^{\boldsymbol{s},k}_{\boldsymbol{\tau},\nu}$ is represented as the product of the transfer matrix $\mathbf{W}^{\boldsymbol{s},k}_{\boldsymbol{\tau},\nu}$ and $\mathrm{diag}_c(\mathbf{V}^{\boldsymbol{s},k}_{\boldsymbol{p_\tau},\nu^c})$. Here, the transfer matrix $\mathbf{W}^{\boldsymbol{s},k}_{\boldsymbol{\tau},\nu}$ is computed as the interpolation matrix of the column ID of the mode-$(d+k)$ unfolding of $\boldsymbol{\mathcal{K}}(\boldsymbol{\tau},\boldsymbol{s}_{k\leftarrow\overline{\nu^1}\cup\overline{\nu^2}})$. As mentioned in section 3, in practice one never forms the unfolding matrix in full, but instead considers the unfolding of $\boldsymbol{\mathcal{K}}(\hat{\boldsymbol{\tau}},\hat{\boldsymbol{s}}_{k\leftarrow\overline{\nu^1}\cup\overline{\nu^2}})$, where $\hat{\boldsymbol{\tau}}=(\hat{\tau}_1,\hat{\tau}_2,\ldots,\hat{\tau}_d)$ and $\hat{\boldsymbol{s}}=(\hat{s}_1,\hat{s}_2,\ldots,\hat{s}_d)$; here $\hat{\tau}_i$ and $\hat{s}_i$ consist of $O(r)$ judiciously selected indices along modes i and $d+i$, respectively. Note that $\hat{s}_k$ is never used as it is replaced by $\overline{\nu^1}\cup\overline{\nu^2}$ in (3.8). The same proxy index strategy can be used to obtain $\mathbf{V}^{\boldsymbol{s},k}_{\boldsymbol{\tau},\nu}$ at the level $l=0$. For each $\mathbf{W}^{\boldsymbol{s},k}_{\boldsymbol{\tau},\nu}$ or $\mathbf{V}^{\boldsymbol{s},k}_{\boldsymbol{\tau},\nu}$, its computation requires $O(r^{2d+1})$ CPU time.

Similarly in (3.7), $\mathbf{U}^{\boldsymbol{t},k}_{\tau,\boldsymbol{\nu}}$ is explicitly formed at $l=0$ and constructed via the transfer matrix $\mathbf{P}^{\boldsymbol{t},k}_{\tau,\boldsymbol{\nu}}$ at level $0<l\leq L^c$:

$$
\begin{aligned}
\boldsymbol{\mathcal{K}}(\boldsymbol{t}_{k\leftarrow\tau},\boldsymbol{\nu}) &\approx \boldsymbol{\mathcal{K}}(\boldsymbol{t}_{k\leftarrow\overline{\tau^1}\cup\overline{\tau^2}},\boldsymbol{\nu})\times_{k}\begin{bmatrix}\mathbf{U}^{\boldsymbol{t},k}_{\tau^1,\boldsymbol{p_\nu}} & \\ & \mathbf{U}^{\boldsymbol{t},k}_{\tau^2,\boldsymbol{p_\nu}}\end{bmatrix}\\
&\approx \boldsymbol{\mathcal{K}}(\boldsymbol{t}_{k\leftarrow\overline{\tau}},\boldsymbol{\nu})\times_{k}\left(\mathbf{P}^{\boldsymbol{t},k}_{\tau,\boldsymbol{\nu}}\begin{bmatrix}\mathbf{U}^{\boldsymbol{t},k}_{\tau^1,\boldsymbol{p_\nu}} & \\ & \mathbf{U}^{\boldsymbol{t},k}_{\tau^2,\boldsymbol{p_\nu}}\end{bmatrix}\right).
\end{aligned}
\tag{3.9}
$$

Putting together (3.6), (3.7), (3.8) and (3.9), the proposed tensor butterfly decomposition can be expressed, for any multi-set $\boldsymbol{t}=(t_1,t_2,\ldots,t_d)$ with t_i at level L^c of $\mathcal{T}_{t_i^0}$ and any multi-set $\boldsymbol{s}=(s_1,s_2,\ldots,s_d)$ with s_i at level L^c of $\mathcal{T}_{s_i^0}$, by forming a Tucker-ID for the $(\boldsymbol{t},\boldsymbol{s})$ pair:

$$
\boldsymbol{\mathcal{K}}(\boldsymbol{t},\boldsymbol{s})\approx\boldsymbol{\mathcal{K}}(\overline{\boldsymbol{t}},\overline{\boldsymbol{s}})\left(\prod_{k=1}^{d}\times_k\left(\prod_{l=L^c}^{1}\overline{\mathbf{P}}^{\boldsymbol{t},\boldsymbol{s},k}_{l}\overline{\mathbf{U}}^{\boldsymbol{t},k}\right)\right)\left(\prod_{k=1}^{d}\times_{d+k}\left(\prod_{l=L^c}^{1}\overline{\mathbf{W}}^{\boldsymbol{t},\boldsymbol{s},k}_{l}\overline{\mathbf{V}}^{\boldsymbol{s},k}\right)\right).
\tag{3.10}
$$

Here, $\overline{\boldsymbol{t}}$ and $\overline{\boldsymbol{s}}$ represent the skeleton indices of the Tucker-ID of $\boldsymbol{\mathcal{K}}(\boldsymbol{t},\boldsymbol{s})$. The interpolation factors $\overline{\mathbf{U}}^{\boldsymbol{t},k}$ and $\overline{\mathbf{V}}^{\boldsymbol{s},k}$ in (3.10) are:

$$\overline{\mathbf{U}}^{\boldsymbol{t},k} = \mathrm{diag}_\tau(\mathbf{U}^{\boldsymbol{t},k}_{\tau,\boldsymbol{s}^0}), \quad \tau \text{ at level } L^c \text{ of } \mathcal{T}_{t_k}, \tag{3.11}$$

$$\overline{\mathbf{V}}^{\boldsymbol{s},k} = \mathrm{diag}_\nu(\mathbf{V}^{\boldsymbol{s},k}_{\boldsymbol{t}^0,\nu}), \quad \nu \text{ at level } L^c \text{ of } \mathcal{T}_{s_k}, \tag{3.12}$$

and the transfer factors $\overline{\mathbf{P}}_l^{\boldsymbol{t},\boldsymbol{s},k}$ and $\overline{\mathbf{W}}_l^{\boldsymbol{t},\boldsymbol{s},k}$ for $l = 1, \ldots, L^c$ are:

$$\overline{\mathbf{P}}_l^{\boldsymbol{t},\boldsymbol{s},k} = \mathrm{diag}_{\boldsymbol{\nu}}\Big(\Big[\mathrm{diag}_\tau(\mathbf{P}^{\boldsymbol{t},k}_{\tau,\boldsymbol{\nu}^c})\Big]_{\boldsymbol{c}} \Big), \quad \begin{array}{c} \tau \text{ at level } L^c - l \text{ of } \mathcal{T}_{t_k}, \\ \nu_i \text{ at level } l-1 \text{ of } \mathcal{T}_{s_i^0}, s_i \subseteq \nu_i, i \le d; \end{array} \tag{3.13}$$

$$\overline{\mathbf{W}}_l^{\boldsymbol{t},\boldsymbol{s},k} = \mathrm{diag}_{\boldsymbol{\tau}}\Big(\Big[\mathrm{diag}_\nu(\mathbf{W}^{\boldsymbol{s},k}_{\boldsymbol{\tau}^c,\nu})\Big]_{\boldsymbol{c}} \Big), \quad \begin{array}{c} \tau_i \text{ at level } l-1 \text{ of } \mathcal{T}_{t_i^0}, t_i \subseteq \tau_i, i \le d, \\ \nu \text{ at level } L^c - l \text{ of } \mathcal{T}_{s_k}. \end{array} \tag{3.14}$$

One can verify that when $d = 1$, the tensor butterfly algorithm (3.10) reduces to the matrix butterfly algorithm (2.12). But when $d > 1$, the tensor butterfly algorithm has a distinct algorithmic structure so that the corresponding computational complexity can be significantly reduced compared with the matrix butterfly algorithm. Detailed computational complexity analysis is provided in subsection 3.2.2.

To better understand the structure of the tensor butterfly in (3.10), (3.11), (3.12), (3.13), and (3.14), we describe its tensor diagram here. We first create the tensor diagram for a *matrix partitioner* as shown in Figure 3.1(b), which represents a $2^d \times 2$ block partitioning of a matrix such as $\Big[\mathbf{W}^{\boldsymbol{s},k}_{\boldsymbol{\tau}^c,\nu}\Big]_{\boldsymbol{c}}$ in (3.14) for fixed $\boldsymbol{s}, \boldsymbol{\tau}, k$ and ν, or $\Big[\mathbf{P}^{\boldsymbol{t},k}_{\tau,\boldsymbol{\nu}^c}\Big]_{\boldsymbol{c}}$ in (3.13) for fixed $\boldsymbol{t}, \boldsymbol{\nu}, k$ and τ. In Figure 3.1(b), each of the row and column dimensions is connected to a partitioning node. The row partitioning node has a parent edge with an arrow pointing to the row dimension to be partitioned, and 2 children edges connected to the parent edge. Similarly, the column partitioning node has a parent edge with an arrow pointing to the column dimension to be partitioned, and 2^d children edges connected to the parent edge. The weight of the parent edge (i.e., the number of columns and rows of the matrix) equals the sum of the weights of the children edges. The diagram in Figure 3.1(c) shows the connectivity for all $\mathbf{V}^{\boldsymbol{s},k}_{\boldsymbol{t}^0,\nu}$ (the green circles) and $\Big[\mathbf{W}^{\boldsymbol{s},k}_{\boldsymbol{\tau}^c,\nu}\Big]_{\boldsymbol{c}}$ (the blue circles) for fixed $\boldsymbol{s}$ and k. The multiplication or contraction of all matrices in Figure 3.1(c) results in $\mathbf{V}^{\boldsymbol{s},k}_{\boldsymbol{t},s_k}$ for all mid-level multi-sets $\boldsymbol{t}$, which are of course not explicitly formed.

As an example, consider an OIO representing the free-space Green's function interaction between two parallel facing unit square plates in Figure 3.2. The tensor is $\boldsymbol{\mathcal{K}}(\boldsymbol{i},\boldsymbol{j}) = K(x^{\boldsymbol{i}}, y^{\boldsymbol{j}}) = \frac{\exp(-\mathrm{i}\omega\rho)}{\rho}$ where $x^{\boldsymbol{i}} = (\frac{i_1}{n}, \frac{i_2}{n}, 0)$, $y^{\boldsymbol{j}} = (\frac{j_1}{n}, \frac{j_2}{n}, 1)$, $\rho = |x^{\boldsymbol{i}} - y^{\boldsymbol{j}}|$ and ω is the wavenumber. Here 1 represents the distance between the two plates. Consider an L=2-level tensor butterfly decomposition, with a total of 16 middle-level multi-set pairs. Let $(\boldsymbol{t},\boldsymbol{s})$ denote one middle-level multi-set pair with $\boldsymbol{t} = (t_1, t_2)$ and $\boldsymbol{s} = (s_1, s_2)$ as highlighted in orange in Figure 3.2(b). Their children are $t_1^1, t_2^1, t_1^2, t_2^2$ and $s_1^1, s_2^1, s_1^2, s_2^2$. Leveraging the representations in Figure 3.1(b)-(c), the full diagram for $\boldsymbol{\mathcal{K}}(\boldsymbol{t},\boldsymbol{s})$ consists of one 4-mode tensor $\boldsymbol{\mathcal{K}}(\overline{\boldsymbol{t}},\overline{\boldsymbol{s}})$ (highlighted in orange in Figure 3.2(a)), one transfer matrix per mode, and two factor matrices per mode. In addition, we plot the full connectivity for two other multi-set pairs (highlighted in green in Figure 3.2(a)). It is important to note that the factor matrices and transfer matrices are shared among the multi-set pairs.

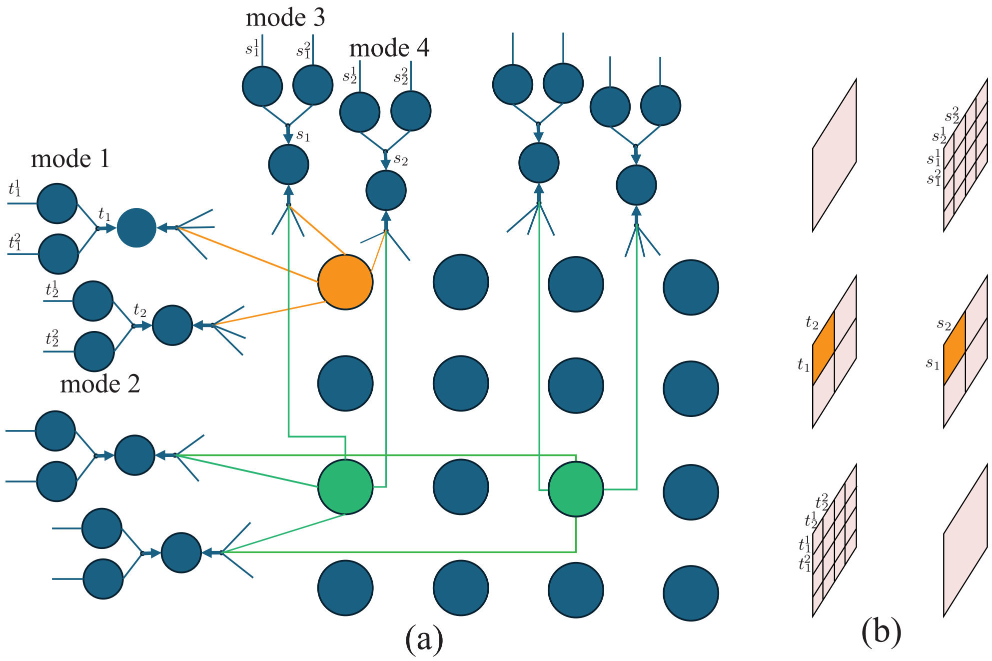

Fig. 3.2: (a) Tensor diagram for the tensor butterfly decomposition of $L = 2$ levels of a 4-mode OIO tensor representing (b) high-frequency Green's function interactions between parallel facing 2D unit squares. Only the full connectivity regarding three middle-level node pairs is shown (the two green circles and one orange circle in (a)). The orange circle in (a) represents the core tensor $\boldsymbol{\mathcal{K}}(\bar{\boldsymbol{t}}, \bar{\boldsymbol{s}})$ for a mid-level pair $(\boldsymbol{t}, \boldsymbol{s})$ with $\boldsymbol{t} = (t_1, t_2)$, $\boldsymbol{s} = (s_1, s_2)$ highlighted in orange in (b).

The proposed tensor butterfly algorithm is fully described in Algorithm 3.1 for a $2d$-mode tensor $\boldsymbol{\mathcal{K}} \in \mathbb{C}^{m_1 \times m_2 \times \cdots \times m_d \times n_1 \times n_2 \times \cdots \times n_d}$, which consists of three steps: (1) computation of $\mathbf{V}^{\boldsymbol{s},k}_{\boldsymbol{\tau},\nu}$ and $\mathbf{W}^{\boldsymbol{s},k}_{\boldsymbol{\tau},\nu}$ starting at Line 1, (2) computation of $\mathbf{U}^{\boldsymbol{t},k}_{\tau,\boldsymbol{\nu}}$ and $\mathbf{P}^{\boldsymbol{t},k}_{\tau,\boldsymbol{\nu}}$ starting at Line 17, and (3) computation of $\boldsymbol{\mathcal{K}}(\bar{\boldsymbol{t}}, \bar{\boldsymbol{s}})$ starting at Line 33. We note that, after each $\boldsymbol{\mathcal{K}}(\bar{\boldsymbol{t}}, \bar{\boldsymbol{s}})$ is formed, we leverage floating-point compression tools such as the ZFP software [40] to further compress it.

Once $\boldsymbol{\mathcal{K}}$ is compressed, any input tensor $\boldsymbol{\mathcal{F}} \in \mathbb{C}^{n_1 \times n_2 \times \cdots \times n_d \times n_v}$ can contract with it to compute $\boldsymbol{\mathcal{G}} = \boldsymbol{\mathcal{K}} \times_{d+1,d+2,\ldots,2d} \boldsymbol{\mathcal{F}}$. It is clear to see that the contraction is equivalent to matrix-matrix multiplication $\mathbf{G} = \mathbf{KF}$, where $\mathbf{G} \in \mathbb{C}^{\prod_k m_k \times n_v}$, $\mathbf{K} \in \mathbb{C}^{\prod_k m_k \times \prod_k n_k}$, and $\mathbf{F} \in \mathbb{C}^{\prod_k n_k \times n_v}$ are matricizations of $\boldsymbol{\mathcal{G}}$, $\boldsymbol{\mathcal{K}}$ and $\boldsymbol{\mathcal{F}}$, respectively, and n_v is the number of columns of $\mathbf{F}$. The contraction algorithm is described in Algorithm 3.2 which consists of three steps:

(1) Contraction with $\mathbf{V}^{\boldsymbol{s},k}_{\boldsymbol{\tau},\nu}$ and $\mathbf{W}^{\boldsymbol{s},k}_{\boldsymbol{\tau},\nu}$. For each level $l = 0, 1, \ldots, L^c$, one notices that, since the contraction operation for each multi-set $\boldsymbol{\tau}$ with τ_i at level l of $\mathcal{T}_{t_i^0}$ and the middle-level multi-set $\boldsymbol{s}$ is independent of each other, one needs a separate tensor $\boldsymbol{\mathcal{F}}_{\boldsymbol{\tau},\boldsymbol{s}}$ to store the contraction result for each multi-set pair $(\boldsymbol{\tau}, \boldsymbol{s})$. $\boldsymbol{\mathcal{F}}_{\boldsymbol{\tau},\boldsymbol{s}}$ can be computed by mode-by-mode contraction with the factor matrices $\overline{\mathbf{V}}^{\boldsymbol{s},k}$ for $l = 0$ (Line 6) and the transfer matrices $\operatorname{diag}_\nu(\mathbf{W}^{\boldsymbol{s},k}_{\boldsymbol{\tau},\nu})$ for $l > 0$ (Line 8).

(2) Contraction with $\boldsymbol{\mathcal{K}}(\bar{\boldsymbol{t}}, \bar{\boldsymbol{s}})$ at the middle level. Tensors at the middle level $\boldsymbol{\mathcal{F}}_{\boldsymbol{t},\boldsymbol{s}}$

are contracted with each subtensor $\mathcal{K}(\bar{t}, \bar{s})$ separately, resulting in tensors $\mathcal{G}_{t,s} = \mathcal{K}(\bar{t}, \bar{s}) \times_{d+1,d+2,\ldots,2d} \mathcal{F}_{t,s}$.

(3) Contraction with $\mathbf{U}^{t,k}_{\tau,\nu}$ and $\mathbf{P}^{t,k}_{\tau,\nu}$. As Step (1), for each level $l = L^c, L^{c-1}, \ldots, 0$, the contraction operation for each multi-set $\boldsymbol{\nu}$ with ν_i at level l of $\mathcal{T}_{s_i^0}$ and middle-level multi-set $\boldsymbol{t}$ is independent. At level $l > 0$, the contribution of tensors $\mathcal{G}_{t,\nu}$ is accumulated into $\mathcal{G}_{t,p_\nu}$ (Line 26); at level $l = 0$, the contraction results are stored in the final output tensor $\mathcal{G}(\boldsymbol{t}, 1 : n_v)$ (Line 24).

3.2.1. Rank Estimate. In this subsection, we use two specific high-dimensional examples, namely high-frequency free-space Green's functions for wave equations and uniform discrete Fourier transforms (DFTs) to investigate the matrix and tensor CLR properties, and compare the matrix and tensor butterfly ranks r_m and r_t, respectively. For the Green's function example, the tensor CLR property is a result of matrix CLR and translational invariance, and r_t is much smaller than r_m; for the DFT example, the tensor CLR property is a result of matrix CLR and dimensionality separability, and r_t is exactly the same as r_m of 1D DFTs. For more-general OIOs, such as analytical and numerical Green's functions for inhomogeneous media, Radon transforms, non-uniform DFTs, and general Fourier integral operators, rigorous rank analysis is non-trivial and we rely on numerical experiments in section 4 to demonstrate the efficacy of the tensor butterfly algorithm.

High-frequency Green's functions. We use an example similar to the one used in subsection 3.2. Consider an OIO representing the free-space Green's function interaction between two parallel-facing unit-square plates. The $n \times n \times n \times n$ tensor is

$$\mathcal{K}(\boldsymbol{i}, \boldsymbol{j}) = K(x^{\boldsymbol{i}}, y^{\boldsymbol{j}}) = \frac{\exp(-\mathrm{i}\omega\rho)}{\rho}, \tag{3.15}$$

where $x^{\boldsymbol{i}} = (\frac{i_1}{n}, \frac{i_2}{n}, 0)$, $y^{\boldsymbol{j}} = (\frac{j_1}{n}, \frac{j_2}{n}, \rho_{\min})$, ω is the wavenumber, and $\rho = |x^{\boldsymbol{i}} - y^{\boldsymbol{j}}|$. Here $\rho_{\min}$ represents the distance between the two plates assumed to be sufficiently large. In the high-frequency setting, $n = C_p\omega$ with a constant C_p independent of n and ω, and the grid size is $\delta_x = \delta_y = \frac{1}{n}$ per dimension. It has been well studied [53, 54, 20, 5] that for any multi-set pair $(\boldsymbol{\tau}, \boldsymbol{\nu})$ (assuming that each set of the multi-set $\boldsymbol{\tau}$ or $\boldsymbol{\nu}$ contains contiguous indices) leading to a subtensor $\mathcal{K}(\boldsymbol{\tau}, \boldsymbol{\nu})$ of sizes $m_1 \times m_2 \times n_1 \times n_2$ with $m_i, n_i \le n$, the numerical rank of its matricization $\mathbf{K} \in \mathbb{C}^{m_1 m_2 \times n_1 n_2}$ can be estimated as

$$r_m \approx \omega^2 a^2 \theta\phi + \Delta_\epsilon \approx \frac{\omega^2 a^2 n_1 n_2}{n^2 \rho_{\min}^2} + \Delta_\epsilon. \tag{3.16}$$

Here a is the radius of the sphere enclosing the target domain of physical sizes $m_1\delta_x \times m_2\delta_y$. $\theta \approx \frac{n_1}{n\rho_{\min}}$, $\phi \approx \frac{n_2}{n\rho_{\min}}$, and the product $\theta\phi$ represents the solid angle covered by the source domain as seen from the center of the target domain. Note that $\frac{\omega a}{\rho_{\min}}$ approximately represents the Nyquist sampling rate per direction needed in the source domain. The ϵ-dependent term $\Delta_\epsilon = O(\log \epsilon^{-1})$ according to analysis in [53, 54]. The matrix and tensor butterfly ranks can be estimated as follows:

- *Matrix butterfly rank:* Consider a matrix butterfly factorization of matricization of $\mathcal{K}$. By design, for any node pair at each level, $m_1 n_1 = m_2 n_2 = C_b n$, where C_b^2 represents the size of the leaf nodes. Therefore, the matrix butterfly rank can be estimated from (3.16) as

$$r_m \approx \frac{C_b^2}{2C_p^2\rho_{\min}^2} + \Delta_\epsilon. \tag{3.17}$$

Algorithm 3.1 Construction algorithm for the tensor butterfly decomposition of a $2d$-mode tensor $\boldsymbol{\mathcal{K}} \in \mathbb{C}^{m_1 \times m_2 \times \cdots \times m_d \times n_1 \times n_2 \times \cdots \times n_d}$

Input: A function to evaluate a $2d$-mode tensor $\boldsymbol{\mathcal{K}}(\boldsymbol{i}, \boldsymbol{j})$ for arbitrary multi-indices $(\boldsymbol{i}, \boldsymbol{j})$, binary partitioning trees of L levels $\mathcal{T}_{t_k^0}$ and $\mathcal{T}_{s_k^0}$ with roots $t_k^0 = \{1, 2, \ldots, m_k\}$ and $s_k^0 = \{1, 2, \ldots, n_k\}$, a relative compression tolerance ϵ.

Output: Tensor butterfly decomposition of $\boldsymbol{\mathcal{K}}$: (1) $\mathbf{V}^{\boldsymbol{s},k}_{\boldsymbol{\tau},\nu}$ at $l = 0$ and $\mathbf{W}^{\boldsymbol{s},k}_{\boldsymbol{\tau},\nu}$ at $1 \le l \le L^c$ of $k \le d$ for multi-set $\boldsymbol{\tau}$ with node τ_i at level l of $\mathcal{T}_{t_i^0}$, multi-set $\boldsymbol{s}$ with node s_i at level L^c of $\mathcal{T}_{s_i^0}$, and node ν at level $L^c - l$ of subtree $\mathcal{T}_{s_k}$, (2) $\mathbf{U}^{\boldsymbol{t},k}_{\tau,\boldsymbol{\nu}}$ at $l = 0$ and $\mathbf{P}^{\boldsymbol{t},k}_{\tau,\boldsymbol{\nu}}$ at $1 \le l \le L^c$ of $k \le d$ for multi-set $\boldsymbol{\nu}$ with node ν_i at level l of $\mathcal{T}_{s_i^0}$, multi-set $\boldsymbol{t}$ with node t_i at level L^c of $\mathcal{T}_{t_i^0}$, and node τ at level $L^c - l$ of subtree $\mathcal{T}_{t_k}$, and (3) subtensors $\boldsymbol{\mathcal{K}}(\bar{\boldsymbol{t}}, \bar{\boldsymbol{s}})$ at $l = L^c$.

(1) Compute $\mathbf{V}^{\boldsymbol{s},k}_{\boldsymbol{\tau},\nu}$ and $\mathbf{W}^{\boldsymbol{s},k}_{\boldsymbol{\tau},\nu}$:
for level $l = 0, \ldots, L^c$ **do**
    for multi-set $\boldsymbol{s} = (s_1, \ldots, s_d)$ with s_i at level L^c of $\mathcal{T}_{s_i^0}$ **do**
        for multi-set $\boldsymbol{\tau} = (\tau_1, \tau_2, \ldots, \tau_d)$ with τ_i at level l of $\mathcal{T}_{t_i^0}$ **do**
            for mode index $k = 1, \ldots, d$ **do**
                for node ν at level $L^c - l$ of $\mathcal{T}_{s_k}$ **do**
                    if $l = 0$ **then** ▷ Use (3.6) with proxies $\hat{\boldsymbol{\tau}}$, $\hat{\boldsymbol{s}}$ and tolerance ϵ
                        Compute $\mathbf{V}^{\boldsymbol{s},k}_{\boldsymbol{\tau},\nu}$ and $\overline{\nu}$ via mode-$(d+k)$ unfolding of $\boldsymbol{\mathcal{K}}(\hat{\boldsymbol{\tau}}, \hat{\boldsymbol{s}}_{k \leftarrow \nu})$
                    else ▷ Use (3.8) with proxies $\hat{\boldsymbol{\tau}}$, $\hat{\boldsymbol{s}}$ and tolerance ϵ
                        Compute $\mathbf{W}^{\boldsymbol{s},k}_{\boldsymbol{\tau},\nu}$ and $\overline{\nu}$ via mode-$(d+k)$ unfolding of $\boldsymbol{\mathcal{K}}(\hat{\boldsymbol{\tau}}, \hat{\boldsymbol{s}}_{k \leftarrow \overline{\nu^1} \cup \overline{\nu^2}})$
                    end if
                end for
            end for
        end for
    end for
end for
(2) Compute $\mathbf{U}^{\boldsymbol{t},k}_{\tau,\boldsymbol{\nu}}$ and $\mathbf{P}^{\boldsymbol{t},k}_{\tau,\boldsymbol{\nu}}$:
for level $l = 0, \ldots, L^c$ **do**
    for multi-set $\boldsymbol{t} = (t_1, \ldots, t_d)$ with t_i at level L^c of $\mathcal{T}_{t_i^0}$ **do**
        for multi-set $\boldsymbol{\nu} = (\nu_1, \nu_2, \ldots, \nu_d)$ with ν_i at level l of $\mathcal{T}_{s_i^0}$ **do**
            for mode index $k = 1, \ldots, d$ **do**
                for node τ at level $L^c - l$ of $\mathcal{T}_{t_k}$ **do**
                    if $l = 0$ **then** ▷ Use (3.7) with proxies $\hat{\boldsymbol{t}}$, $\hat{\boldsymbol{\nu}}$ and tolerance ϵ
                        Compute $\mathbf{U}^{\boldsymbol{t},k}_{\tau,\boldsymbol{\nu}}$ and $\overline{\tau}$ via mode-k unfolding of $\boldsymbol{\mathcal{K}}(\hat{\boldsymbol{t}}_{k \leftarrow \tau}, \hat{\boldsymbol{\nu}})$
                    else ▷ Use (3.9) with proxies $\hat{\boldsymbol{t}}$, $\hat{\boldsymbol{\nu}}$ and tolerance ϵ
                        Compute $\mathbf{P}^{\boldsymbol{t},k}_{\tau,\boldsymbol{\nu}}$ and $\overline{\tau}$ via mode-k unfolding of $\boldsymbol{\mathcal{K}}(\hat{\boldsymbol{t}}_{k \leftarrow \overline{\tau^1} \cup \overline{\tau^2}}, \hat{\boldsymbol{\nu}})$
                    end if
                end for
            end for
        end for
    end for
end for
(3) Compute $\boldsymbol{\mathcal{K}}(\bar{\boldsymbol{t}}, \bar{\boldsymbol{s}})$:
for multi-set $\boldsymbol{s} = (s_1, \ldots, s_d)$ with s_i at level L^c of $\mathcal{T}_{s_i^0}$ **do**
    for multi-set $\boldsymbol{t} = (t_1, \ldots, t_d)$ with t_i at level L^c of $\mathcal{T}_{t_i^0}$ **do**
        Compute $\boldsymbol{\mathcal{K}}(\bar{\boldsymbol{t}}, \bar{\boldsymbol{s}})$ and ZFP compress it
    end for
end for

Algorithm 3.2 Contraction algorithm for a tensor butterfly decomposition with an input tensor

Input: The tensor butterfly decomposition of a $2d$-mode tensor $\boldsymbol{\mathcal{K}} \in \mathbb{C}^{m_1\times m_2\times\cdots\times m_d\times n_1\times n_2\times\cdots\times n_d}$, and a (full) $d+1$-mode input tensor $\boldsymbol{\mathcal{F}} \in \mathbb{C}^{n_1\times n_2\times\cdots\times n_d\times n_v}$ where n_v denotes the number of columns of $\mathbf{F}^{(d+1)}$.
Output: The $d+1$-mode output tensor $\boldsymbol{\mathcal{G}} = \boldsymbol{\mathcal{K}} \times_{d+1,d+2,\dots,2d} \boldsymbol{\mathcal{F}}$ where $\boldsymbol{\mathcal{G}} \in \mathbb{C}^{m_1\times m_2\times\cdots\times m_d\times n_v}$.

(1) Multiply with $\mathbf{V}^{\boldsymbol{s},k}_{\boldsymbol{\tau},\nu}$ and $\mathbf{W}^{\boldsymbol{s},k}_{\boldsymbol{\tau},\nu}$:
for level $l = 0,\dots,L^c$ **do**
  for multi-set $\boldsymbol{s} = (s_1, s_2\dots, s_d)$ with s_i at level L^c of $\mathcal{T}_{s_i^0}$ **do**
    for multi-set $\boldsymbol{\tau} = (\tau_1, \tau_2, \dots, \tau_d)$ with τ_i at level l of $\mathcal{T}_{t_i^0}$ **do**
      if $l = 0$ **then**
        $\boldsymbol{\mathcal{F}}_{\boldsymbol{\tau},\boldsymbol{s}} = \boldsymbol{\mathcal{F}}(\boldsymbol{s}, 1:n_v)\prod_{k=1}^d \times_k \overline{\mathbf{V}}^{\boldsymbol{s},k}$
      else
        $\boldsymbol{\mathcal{F}}_{\boldsymbol{\tau},\boldsymbol{s}} = \boldsymbol{\mathcal{F}}_{\boldsymbol{p}_{\boldsymbol{\tau}},\boldsymbol{s}}\prod_{k=1}^d \times_k \mathrm{diag}_\nu(\mathbf{W}^{\boldsymbol{s},k}_{\boldsymbol{\tau},\nu})$ ▷ ν at level $L^c - l$ of $\mathcal{T}_{s_k}$
      end if
    end for
  end for
end for
(2) Contract with $\boldsymbol{\mathcal{K}}(\bar{\boldsymbol{t}}, \bar{\boldsymbol{s}})$:
for multi-set $\boldsymbol{t} = (t_1, t_2\dots, t_d)$ with t_i at level L^c of $\mathcal{T}_{t_i^0}$ **do**
  for multi-set $\boldsymbol{s} = (s_1, s_2\dots, s_d)$ with s_i at level L^c of $\mathcal{T}_{s_i^0}$ **do**
    ZFP decompress $\boldsymbol{\mathcal{K}}(\bar{\boldsymbol{t}}, \bar{\boldsymbol{s}})$ and compute $\boldsymbol{\mathcal{G}}_{\boldsymbol{t},\boldsymbol{s}} = \boldsymbol{\mathcal{K}}(\bar{\boldsymbol{t}}, \bar{\boldsymbol{s}}) \times_{d+1,d+2,\dots,2d} \boldsymbol{\mathcal{F}}_{\boldsymbol{t},\boldsymbol{s}}$
  end for
end for
(3) Multiply with $\mathbf{U}^{\boldsymbol{t},k}_{\tau,\boldsymbol{\nu}}$ and $\mathbf{P}^{\boldsymbol{t},k}_{\tau,\boldsymbol{\nu}}$:
for level $l = L^c,\dots,0$ **do**
  for multi-set $\boldsymbol{t} = (t_1, t_2, \dots, t_d)$ with t_i at level L^c of $\mathcal{T}_{t_i^0}$ **do**
    for multi-set $\boldsymbol{\nu} = (\nu_1, \nu_2, \dots, \nu_d)$ with ν_i at level l of $\mathcal{T}_{s_i^0}$ **do**
      if $l = 0$ **then** ▷ Compute and return $\boldsymbol{\mathcal{G}}$
        $\boldsymbol{\mathcal{G}}(\boldsymbol{t}, 1:n_v) = \boldsymbol{\mathcal{G}}_{\boldsymbol{t},\boldsymbol{\nu}}\prod_{k=1}^d \times_k \overline{\mathbf{U}}^{\boldsymbol{t},k}$
      else
        $\boldsymbol{\mathcal{G}}_{\boldsymbol{t},\boldsymbol{p}_{\boldsymbol{\nu}}} \mathrel{+}= \boldsymbol{\mathcal{G}}_{\boldsymbol{t},\boldsymbol{\nu}}\prod_{k=1}^d \times_k \mathrm{diag}_\tau(\mathbf{P}^{\boldsymbol{t},k}_{\tau,\boldsymbol{\nu}})$ ▷ τ at level $L^c - l$ of $\mathcal{T}_{t_k}$
      end if
    end for
  end for
end for

Here we have assumed $a = \frac{m_1}{\sqrt{2}n}$. Note that r_m is a constant independent of n, and therefore the matrix CLR property holds true.

- *Tensor butterfly rank:* Consider an L-level tensor butterfly factorization of $\boldsymbol{\mathcal{K}}$. We just need to check the tensor rank, e.g., the rank of the mode-4 unfolding of the corresponding subtensors at Step (1) of Algorithm 3.1, as the unfolding for the other modes can be investigated in a similar fashion. Figure 3.3(a) shows an example of $L = 2$, where the target and source domains are partitioned at $l = 0$ (top) and $l = L^c = 1$ (bottom) at Step (1) of Algorithm 3.1. Consider a multi-set pair $(\boldsymbol{\tau}, \boldsymbol{s}_{k\leftarrow\nu})$ with $k = 4$ required by the tensor CLR property in (3.6). Figure 3.3(a) highlights in orange one multi-set pair at $l = 0$ (top) and one multi-set pair at $l = L^c$ (bottom). Mode 4 is highlighted in green (in all subfigures of Figure 3.3), which needs to be skeletonized by ID. By (3.16), the rank of the matricization of

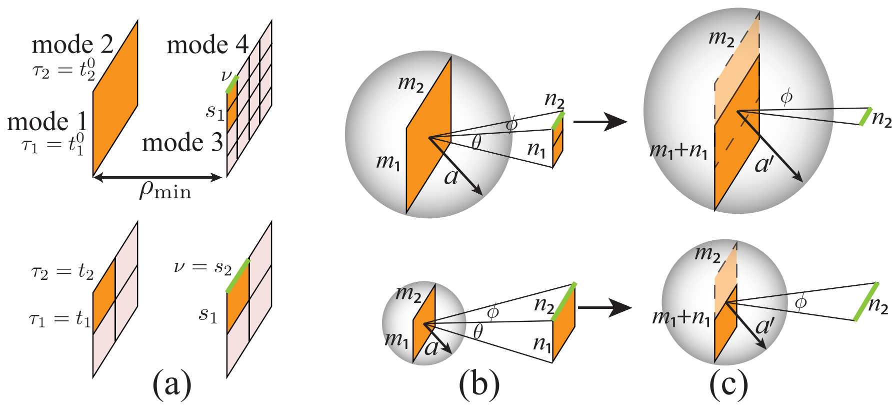

Fig. 3.3: Illustration of the tensor CLR property with $L = 2$ for a 4-mode tensor representing free-space Green's function interactions between parallel facing unit square plates. (a) The target and source domains are partitioned at $l = 0$ (top) and $l = L^c = 1$ (bottom) with a multi-set pair $(\boldsymbol{\tau}, \boldsymbol{s}_{k\leftarrow\nu})$ highlighted in orange for the skeletonization along mode 4. The sizes of the nodes are $|\tau_1| = m_1$,$|\tau_2| = m_1$, $|s_1| = n_1$ and $|\nu| = n_2$. (b) Illustration of the rank of the matricization of $\boldsymbol{\mathcal{K}}(\boldsymbol{\tau}, \boldsymbol{s}_{k\leftarrow\nu})$ used in the matrix butterfly algorithm. Here a is the radius of the sphere enclosing the target domain of physical sizes $m_1\delta_x \times m_2\delta_y$. $\theta \approx \frac{n_1}{n\rho_{\min}}$, $\phi \approx \frac{n_2}{n\rho_{\min}}$, and the product $\theta\phi$ represents the solid angle covered by the source domain as seen from the center of the target domain. (c) Illustration of the rank of the mode-4 unfolding of $\boldsymbol{\mathcal{K}}(\boldsymbol{\tau}, \boldsymbol{s}_{k\leftarrow\nu})$ used in the tensor butterfly algorithm. Here, a' is the radius of the sphere enclosing the enlarged target domain. The source domain is reduced to a line segment of length $n_2\delta_y$.

$\boldsymbol{\mathcal{K}}(\boldsymbol{\tau}, \boldsymbol{s}_{k\leftarrow\nu})$ is no longer a constant as the tensor butterfly algorithm needs to keep $n_1 = |s_1| = n/2^{L^c}$ (see Figure 3.3(b)). However, due to translational invariance of the free-space Green's function, i.e., $K(x^{\boldsymbol{i}}, y^{\boldsymbol{j}}) = K(\tilde{x}, \tilde{y})$, where $\tilde{x} = (0, \frac{i_2}{n}, 0)$, $\tilde{y} = (\frac{j_1 - i_1}{n}, \frac{j_2}{n}, \rho_{\min})$, the mode-4 unfolding of $\boldsymbol{\mathcal{K}}(\boldsymbol{\tau}, \boldsymbol{s}_{k\leftarrow\nu})$ is the matrix representing the Green's function interaction between an enlarged target domain of sizes $(m_1 + n_1)\delta_x \times m_2\delta_y$ and a source line segment of length $n_2\delta_y$. Therefore its rank (hence the tensor rank) can be estimated as

$$r_t \approx \omega a'\phi + \Delta_\epsilon \approx \frac{\omega a' n_2}{n\rho_{\min}} + \Delta_\epsilon \leq \frac{\sqrt{2}C_b}{C_p\rho_{min}} + \Delta_\epsilon, \tag{3.18}$$

where a' is the radius of the sphere enclosing the enlarged target domain and $\frac{\omega a'}{\rho_{\min}}$ approximately represents the Nyquist sampling rate on the source line segment. The last inequality is a result of $a' \approx \frac{m_1+n_1}{\sqrt{2}n} \leq \frac{\sqrt{2}m_1}{n}$ and $m_2 n_2 = C_b n$. Here, the critical condition $n_1 \leq m_1$ is a direct result of the setup of the tensor CLR in (3.6): $l \leq L^c$ and $n_1 = |s_1| = n/2^{L^c}$ (i.e., s_1 is fixed as the middle level set as l changes). One can clearly see from (3.18) that r_t is independent of n, and thus the tensor CLR property holds true.

We remark that the tensor butterfly rank r_t in (3.18) is significantly smaller than the matrix butterfly rank r_m in (3.17) with $r_t \approx 2\sqrt{r_m}$. One can perform similar analysis of r_m and r_t for different geometrical settings, such as a pair of well-separated 3D unit cubes, or a pair of co-planar 2D unit-square plates. We leave these exercises to the readers.

Discrete Fourier Transform. Our second example is the high-dimensional discrete Fourier transform (DFT) defined by

$$\mathcal{K}(\boldsymbol{i},\boldsymbol{j}) = \exp(2\pi \mathrm{i} x^{\boldsymbol{i}} \cdot y^{\boldsymbol{j}}) \tag{3.19}$$

with $x^{\boldsymbol{i}} = (i_1 - 1, i_2 - 1, \ldots, i_d - 1)$ and $y^{\boldsymbol{j}} = (\frac{j_1-1}{n}, \frac{j_2-1}{n}, \ldots, \frac{j_d-1}{n})$. We first notice that, since

$$\exp(2\pi \mathrm{i} x^{\boldsymbol{i}} \cdot y^{\boldsymbol{j}}) = \prod_{k=1}^{d} \exp\left(\frac{2\pi \mathrm{i}(i_k - 1)(j_k - 1)}{n}\right), \tag{3.20}$$

to carry out arbitrary high-dimensional DFTs one can simply perform 1D DFTs one dimension at a time (while fixing the indices of the other dimensions) by either 1D FFT or 1D matrix butterfly algorithms. We choose the 1D butterfly approach as our reference algorithm. For each node pair at dimension k discretized into a $m_k \times n_k$ matrix, we assume that $m_k n_k = C_b n$. It has been proved in [8, 70] that this leads to the matrix CLR property and each 1D DFT (fixing indices in other dimensions) can be computed by the matrix butterfly algorithm in $O(n \log n)$ time with a constant butterfly rank r_m. Overall this approach requires $O(dn^d \log n)$ operations.

In contrast, the tensor butterfly algorithm relies on direct compression of e.g., mode-k unfolding of subtensors $\mathcal{K}(\boldsymbol{\tau}, \boldsymbol{s}_{k \leftarrow \nu})$. Consider any submatrix $\mathbf{K}_{sub} \in \mathbb{C}^{m_k \times n_k}$ of this unfolding matrix $\mathbf{K}^{(k)}$; by fixing i_p and j_p with $p \neq k$, its entry is simply

$$\exp\left(\frac{2\pi \mathrm{i}(i_k - 1)(j_k - 1)}{n}\right)$$

scaled by a constant factor

$$\prod_{p \neq k} \exp\left(\frac{2\pi \mathrm{i}(i_p - 1)(j_p - 1)}{n}\right)$$

of modulus 1. Therefore the tensor butterfly rank is

$$r_t = \mathrm{rank}(\mathbf{K}^{(k)}) = \mathrm{rank}(\mathbf{K}_{sub}) = r_m. \tag{3.21}$$

The tensor CLR property thus holds true, and the tensor rank is exactly the same as the 1D butterfly algorithm per dimension. However, as we will see subsection 3.2.2, our tensor butterfly algorithm yields a linear instead of quasi-linear CPU complexity for high-dimensional DFTs.

3.2.2. Complexity Analysis. Here we provide an analysis of computational complexity and memory requirement of the proposed construction algorithm (Algorithm 3.1) and contraction algorithm (Algorithm 3.2), assuming that the tensor butterfly rank r_t is a small constant and $d > 1$. Recall that the $2d$-mode tensor $\mathcal{K}$ has size n and a binary tree ($\mathcal{T}_{t_k^0}$ or $\mathcal{T}_{s_k^0}$) of L levels along each mode k. $L^c = L/2$ denotes the middle level. We refer the readers to Table 2.1 to recall the notations of the multi-set, k^{th} set, mid-level subtensor, transfer matrix, and interpolation matrix, etc.

At Step (1) of Algorithm 3.1, each level $1 \leq l \leq L^c$ has $\#\boldsymbol{s} = O(\sqrt{n}^d)$, $\#\boldsymbol{\tau} = 2^{dl}$, $\#\nu = O(\sqrt{n}/2^l)$ for each mode $k \leq d$. Each $\mathbf{W}^{\boldsymbol{s},k}_{\boldsymbol{\tau},\nu}$ requires $O(r_t^2)$ storage, and

$O(r_t^{2d+1})$ computational time when proxy indices $\hat{\boldsymbol{\tau}}$, $\hat{\boldsymbol{s}}$ are being used. The storage requirement and computational cost for $\mathbf{W}^{\boldsymbol{s},k}_{\boldsymbol{\tau},\nu}$ are:

$$\text{mem}_W = \sum_{l=1}^{L^c} dO(\sqrt{n}^d)2^{dl}O(\sqrt{n}/2^l)O(r_t^2) = O(dn^d r_t^2), \tag{3.22}$$

$$\text{time}_W = \sum_{l=1}^{L^c} dO(\sqrt{n}^d)2^{dl}O(\sqrt{n}/2^l)O(r_t^{2d+1}) = O(dn^d r_t^{2d+1}). \tag{3.23}$$

One can easily verify that the computation and storage of $\mathbf{V}^{\boldsymbol{s},k}_{\boldsymbol{\tau},\nu}$ at $l = 0$ is less dominant than $\mathbf{W}^{\boldsymbol{s},k}_{\boldsymbol{\tau},\nu}$ at $l > 0$ and we skip its analysis.

At Step (2) of Algorithm 3.1, we have $\#\boldsymbol{s} = O(\sqrt{n}^d)$ and $\#\boldsymbol{t} = O(\sqrt{n}^d)$, and each $\boldsymbol{\mathcal{K}}(\bar{\boldsymbol{t}}, \bar{\boldsymbol{s}})$ requires $O(r_t^{2d})$ computation time and storage units (even if it is further ZFP compressed to reduce storage requirement), which adds up to

$$\text{mem}_K = O(\sqrt{n}^d)O(\sqrt{n}^d)O(r_t^{2d}) = O(n^d r_t^{2d}), \tag{3.24}$$

$$\text{time}_K = O(\sqrt{n}^d)O(\sqrt{n}^d)O(r_t^{2d}) = O(n^d r_t^{2d}). \tag{3.25}$$

Step (3) of Algorithm 3.1 has similar computational cost and memory requirement to Step (1) when contracting with the intermediate matrices $\mathbf{P}^{\boldsymbol{t},k}_{\tau,\boldsymbol{\nu}}$, with $\text{mem}_P \sim \text{mem}_W$ and $\text{time}_P \sim \text{time}_W$.

Overall, Algorithm 3.1 requires

$$\text{mem} = \text{mem}_W + \text{mem}_K + \text{mem}_P = O(n^d r_t^{2d}), \tag{3.26}$$

$$\text{time} = \text{time}_W + \text{time}_K + \text{time}_P = O(dn^d r_t^{2d+1}). \tag{3.27}$$

Following a similar analysis, one can estimate the computational cost of Algorithm 3.2 as $O(n^d r_t^{2d} n_v)$, which is essentially of the similar order as mem of Algorithm 3.1, except an extra factor n_v representing the size of the last dimension of the input tensor.

One critical observation is that the time and storage complexity of the tensor butterfly algorithm is *linear* in n^d with smaller ranks r_t, while that of the matrix butterfly algorithm is *quasi-linear* in n^d with much larger ranks r_m. This leads to a significantly superior algorithm, as will be demonstrated with the numerical results in section 4. That being said, one can verify that there is no difference between the two algorithms when $d = 1$.

3.2.3. Comparison with Tucker-ID and QTT. Here we make a comparison of the computational complexities of the matrix butterfly algorithm, tensor butterfly algorithm, Tucker-ID and QTT for several frequently encountered OIOs with $d = 2, 3$, namely Green's functions for high-frequency wave equations (where $d = 2$ represents two parallel facing unit square plates and $d = 3$ represents two separated unit cubes), Radon transforms (a type of Fourier integral operators), and DFT. We first summarize the computational complexities of the factorization and application of matrix and tensor butterfly algorithms in Table 3.1. Here we use r to denote the maximum rank of the submatrices or (unfolding and matricization of) subtensors associated with each algorithm. In other words, we drop the subscript of r_m and r_t in this subsection. We note that $r = O(1)$ for butterfly algorithms, and the computational complexity for matrix and tensor butterfly algorithms is, respectively, $O(dn^d \log n)$ and $O(dn^d)$, for all OIOs considered here.

Algorithm	Factor time $d=2$	Factor time $d=3$	Apply time $d=2$	Apply time $d=3$	r $d=2$	r $d=3$
Tensor butterfly	n^2	n^3	n^2	n^3	1	1
Matrix butterfly	$n^2 \log n$	$n^3 \log n$	$n^2 \log n$	$n^3 \log n$	1	1
Tucker-ID	n^4	$n^4 - n^{6*}$	n^4	$n^4 - n^{6*}$	n	n
QTT (Green&Radon)	$n^3 \log n$	$n^3 \log n$	$n^4 \log n$	$n^5 \log n$	n	n
QTT (DFT)	$\log n$	$\log n$	$n^2 \log n$	$n^3 \log n$	1	1

Table 3.1: CPU complexity of the tensor butterfly algorithm, matrix butterfly algorithm, Tucker-ID and QTT when applied to high-frequency Green's functions ($d = 2$ represents two parallel facing unit square plates and $d = 3$ represents two separated unit cubes), DFT and Radon transforms. Here we assume that tensor butterfly, matrix butterfly and Tucker-ID algorithms use proxy indices, and the QTT algorithm uses TT-cross. The big O notation is assumed. *: for $d = 3$, the complexity of Tucker-ID is n^6 for Radon transform and DFT, and n^4 for Green's function.

The Tucker-ID algorithm in subsection 3.1 (even with the use of proxy indices to accelerate the factorization), always leads to $r = O(n)$ for OIOs and hence almost always $O(n^{2d})$ factorization and application complexities (see Table 3.1). One exception is perhaps the Green's function for $d = 3$, where one can easily show that 4 out of the 6 unfolding matrices have a rank of $O(n)$ and the remaining 2 have a rank of $O(1)$, leading to the $O(n^4)$ computational complexity. Overall, we remark that Tucker-type decomposition algorithms are typically the least efficient tensor algorithms for OIOs.

The QTT algorithm, on the other hand, is a more subtle algorithm to compare with. Assuming that the maximum rank among all steps in QTT is r, we first summarize the computational complexities of the factorization and application of QTT. For factorization, we only consider the TT-cross type of algorithms, which yields the best known computational complexity among all TT-based algorithms. The computational complexity of TT-cross is $O(dr^3 \log n)$ [14, 57]. Once factorized, the application cost of the QTT factorization with a full input tensor is $O(dr^2 n^d \log n)$ [14]. This complexity can be reduced to $O(dr^2 r_i^2 \log n)$ when the input tensor is also in the QTT format with TT rank r_i. However, an arbitrary input tensor can have a TT rank up to $r_i = O(n^{d/2})$ (which leads to the same application cost as contraction with a full input tensor). Therefore in our comparative study, we stick with the $O(dr^2 n^d \log n)$ application complexity.

For high-frequency Green's functions and general-form Fourier integral operators (e.g. Radon transforms), the TT rank in general behaves as $r = O(n)$ [14], leading to a factorization cost of $O(dn^3 \log n)$ and an application cost of $O(dn^{2+d} \log n)$, as detailed in Table 3.1. It is worth mentioning that, treating DFTs as a special type of Fourier integral operators, QTT can achieve $r = O(1)$ when a proper bit-reversal ordering is used [9], leading to a factorization cost of $O(d \log n)$ and an application cost of $O(dn^d \log n)$, as shown in Table 3.1. In contrast, the proposed tensor butterfly algorithm can always yield $O(dn^d)$ factorization and $O(n^d)$ application costs.

4. Numerical Results. This section provides several numerical examples to demonstrate the accuracy and efficiency of the proposed tensor butterfly algorithm when applied to large-scale and high-dimensional OIOs including Green's function tensors for high-frequency Helmholtz equations (subsection 4.1), Radon transform tensors (subsection 4.2), and high-dimensional DFTs (subsection 4.3). We compare our algorithm with a few existing matrix and tensor algorithms including the matrix butterfly algorithm in subsection 2.2, the Tucker-ID algorithm in subsection 3.1, the QTT algorithm [57], the FFT algorithm implemented in the heFFTe package [1], and the non-uniform FFT (NUFFT) algorithm implemented in the FINUFFT package [2]. All of these algorithms except for Tucker-ID (sequential implementation in Fortran2008 via the ButterflyPACK package [46]) and FINUFFT (Python interface to the C backend with shared-memory parallelism) are tested in distributed-memory parallelism. The reference binary-tree-based matrix butterfly algorithm in subsection 2.2 is implemented in Fortran2008 with distributed-memory parallelism [47], available in the ButterflyPACK package [46]. The proposed tensor butterfly algorithm is also available in the ButterflyPACK package with distributed-memory parallelism (which will be described in detail in a future paper). The matrix and tensor butterfly algorithms leverage ZFP to further compress the middle-level submatrices and subtensors, respectively. It is worth noting that currently there is no single package that can both compute and apply the QTT decomposition in distributed-memory parallelism. In our tests, we perform the factorization using a distributed-memory TT code (fully Python) [63] that parallelizes a cross interpolation algorithm [19], and then we implement the distributed-memory QTT contraction via the CTF package (Python interface to the C++ backend) [64]. All experiments are performed using 4 CPU nodes of the Perlmutter machine at NERSC in Berkeley, where each node has two 64-core AMD EPYC 7763 processors and 128GB of 2133MHz DDR4 memory.

4.1. Green's functions for high-frequency Helmholtz equations. In this subsection, we consider the tensor discretized from 3D free-space Green's functions for high-frequency Helmholtz equations. Specifically, the tensor entry is

$$\boldsymbol{\mathcal{K}}(\boldsymbol{i},\boldsymbol{j}) = \frac{\exp(-\mathrm{i}\omega\rho)}{\rho}, \qquad \rho = |x^{\boldsymbol{i}} - y^{\boldsymbol{j}}|, \tag{4.1}$$

where ω represents the wave number. Two tests are performed: (1) A 4-way tensor representing the Green's function interaction between two parallel facing unit plates with distance 1, i.e., $x^{\boldsymbol{i}} = (\frac{i_1}{n}, \frac{i_2}{n}, 0)$, $y^{\boldsymbol{j}} = (\frac{j_1}{n}, \frac{j_2}{n}, 1)$, and $d = 2$. (2) A 6-way tensor representing the Green's function interaction between two unit cubes with the distance between their centers set to 2, i.e., $x^{\boldsymbol{i}} = (\frac{i_1}{n}, \frac{i_2}{n}, \frac{i_3}{n})$, $y^{\boldsymbol{j}} = (\frac{j_1}{n}, \frac{j_2}{n}, \frac{j_3}{n}+2)$, and $d = 3$. For both tests, the wave number is chosen such that the number of points per wave length is 4, i.e., $2\pi n/\omega = 4$ or $C_p = 2/\pi$. We first perform compression using the tensor butterfly, Tucker-ID and QTT algorithms, and then perform application/contraction using a random input tensor $\boldsymbol{\mathcal{F}}$. We also add results for the matrix butterfly algorithm using the corresponding matricization of $\boldsymbol{\mathcal{K}}$ and $\boldsymbol{\mathcal{F}}$.

Figure 4.1 (left) shows the factorization time, application time and memory usage of each algorithm using a compression tolerance $\epsilon = 10^{-6}$ for the parallel plate case. For QTT, we show the memory of the factorization (labeled as "QTT(Factor)") and application (labeled as "QTT(Apply)") separately. Note that although QTT factorization requires sub-linear memory usage, QTT contraction becomes super-linear due to the full QTT rank of the input tensor. Overall, we achieve the expected complexities listed in Table 3.1 for the butterfly and Tucker-ID algorithms. For QTT, however, instead of an $O(n)$ rank scaling, we observe an $O(n^{3/4})$ rank scaling, leading to slightly better complexities compared with Table 3.1. We leave this as a future investigation. That said, the tensor butterfly algorithm achieves the linear CPU and memory complexities for both factorization and application with a much smaller prefactor compared to all the other algorithms. Remarkably, the tensor butterfly al-

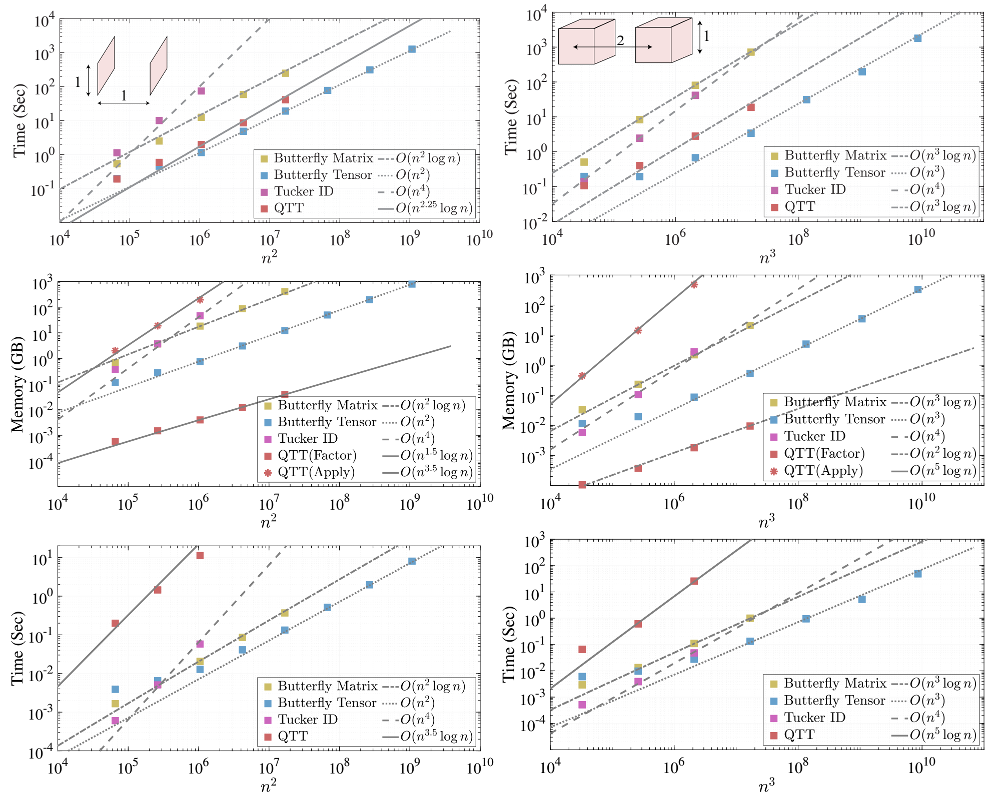

Fig. 4.1: Helmholtz equation: Computational complexity comparison among butterfly matrix, butterfly tensor, Tucker-ID and QTT for compressing (left) a 4-way Green's function tensor for interactions between two parallel 2D plates and (right) a 6-way Green's function tensor for interactions between two 3D cubes. The geometries are discretized with 4 points per wavelength. (Top): Factor time. (Middle): Factor and apply memory. (Bottom): Apply time. The largest data points correspond to 8192 wavelengths per direction for the 2D tests (left) and 512 wavelengths per direction for the 3D tests (right).

gorithm achieves a 30x memory reduction and 15x speedup, capable of handling 64x larger-sized tensors compared with the matrix butterfly algorithm.

Figure 4.1 (right) shows the factorization time, application time and memory usage of each algorithm using a compression tolerance $\epsilon = 10^{-2}$ for the cube case. Overall, we achieve the expected complexities listed in Table 3.1 for all four algorithms. The tensor butterfly algorithm achieves the linear CPU and memory complexities for both factorization and application with a much smaller prefactor compared to all the other algorithms. Remarkably, the tensor butterfly algorithm achieves a 30× memory reduction and 200x speedup, capable of handling 512× larger-sized tensors compared with the matrix butterfly algorithm. The largest data point $n = 2048$ corresponds to 512 wavelengths per physical dimension. The results in Figure 4.1 suggest the superiority of the tensor butterfly algorithm in solving high-frequency wave equations in 3D volumes and on 3D surfaces.

Next, we demonstrate the effect of changing compression tolerance ϵ for both test

n^d	ϵ	$r_{\min}$	r	error	T_f(sec)	T_a (sec)	Mem (MB)
16384^2	1E-02	5	8	1.49E-02	6.83E+01	1.16E+00	2.40E+04
16384^2	1E-03	6	10	2.19E-03	1.17E+02	1.89E+00	4.69E+04
16384^2	1E-04	7	11	1.84E-04	1.57E+02	2.80E+00	7.49E+04
16384^2	1E-05	8	12	3.46E-05	2.29E+02	4.03E+00	1.21E+05
16384^2	1E-06	9	13	9.26E-06	3.18E+02	5.92E+00	1.96E+05
512^3	1E-02	2	5	2.01E-02	1.18E+02	1.42E+00	1.19E+04
512^3	1E-03	2	6	1.18E-03	3.46E+02	4.08E+00	4.87E+04
512^3	1E-04	2	7	8.39E-05	6.26E+02	9.85E+00	1.49E+05
512^3	1E-05	3	8	9.21E-06	1.25E+03	2.40E+01	4.07E+05

Table 4.1: The technical data for a 4-way Green's function tensor of $n = 16384$ and a 6-way Green's function tensor of $n = 512$ for the Helmholtz equation using the proposed tensor butterfly algorithm of varying compression tolerance ϵ. The table shows the maximum rank r and minimum rank $r_{\min}$ across all ID operations, relative error in (4.2), factor time T_f, apply time T_a, and memory usage Mem.

cases in Table 4.1. Here the error is measured by

$$\text{error} = \frac{||\boldsymbol{\mathcal{K}} \times_{d+1,d+2,\ldots,2d} \boldsymbol{\mathcal{F}}_e - \boldsymbol{\mathcal{K}}_{BF} \times_{d+1,d+2,\ldots,2d} \boldsymbol{\mathcal{F}}_e||_F}{||\boldsymbol{\mathcal{K}} \times_{d+1,d+2,\ldots,2d} \boldsymbol{\mathcal{F}}_e||_F} \tag{4.2}$$

where $\boldsymbol{\mathcal{K}}_{BF}$ is the tensor butterfly representation of $\boldsymbol{\mathcal{K}}$, $\boldsymbol{\mathcal{F}}_e(\boldsymbol{j}) = 1$ for a small set of random entries $\boldsymbol{j}$ and 0 elsewhere. This way, $\boldsymbol{\mathcal{K}}$ does not need to be fully formed to compute the error. Table 4.1 shows the minimum rank ($r_{\min}$) and maximum rank (r), error, factorization time, application time and memory usage of varying ϵ, for $n = 16384, d = 2$ and $n = 512, d = 3$. We remark that the observed ranks clearly demonstrate $\Delta_\epsilon = O(\log \epsilon^{-1})$ in (3.18). Overall, the errors are close to the prescribed tolerances and the costs increase for smaller ϵ, as expected. We also note that keeping r as low as possible is critical in maintaining small prefactors of the tensor butterfly algorithm, particularly for higher dimensions.

4.2. Radon transforms. In this subsection, we consider 2D and 3D discretized Radon transforms similar to those presented in [8]. Specifically, the tensor entry is

$$\boldsymbol{\mathcal{K}}(\boldsymbol{i}, \boldsymbol{j}) = \exp(2\pi \mathrm{i} \phi(x^{\boldsymbol{i}}, y^{\boldsymbol{j}})) \tag{4.3}$$

with $x^{\boldsymbol{i}} = (\frac{i_1}{n}, \frac{i_2}{n}, \ldots, \frac{i_d}{n})$ and $y^{\boldsymbol{j}} = (j_1 - \frac{n}{2}, j_2 - \frac{n}{2}, \ldots, j_d - \frac{n}{2})$. For $d = 2$, we consider

$$\begin{aligned} \phi(x, y) &= x \cdot y + \sqrt{c_1^2 y_1^2 + c_2^2 y_2^2}, \\ c_1 &= (2 + \sin(2\pi x_1)\sin(2\pi x_2))/16, \\ c_2 &= (2 + \cos(2\pi x_1)\cos(2\pi x_2))/16. \end{aligned} \tag{4.4}$$

For $d = 3$, we consider

$$\begin{aligned} \phi(x, y) &= x \cdot y + c|y|, \\ c &= (3 + \sin(2\pi x_1)\sin(2\pi x_2)\sin(2\pi x_3))/100. \end{aligned} \tag{4.5}$$

We first perform compression using the matrix butterfly, tensor butterfly, and QTT algorithms, and then perform application/contraction using a random input tensor $\boldsymbol{\mathcal{F}}$.

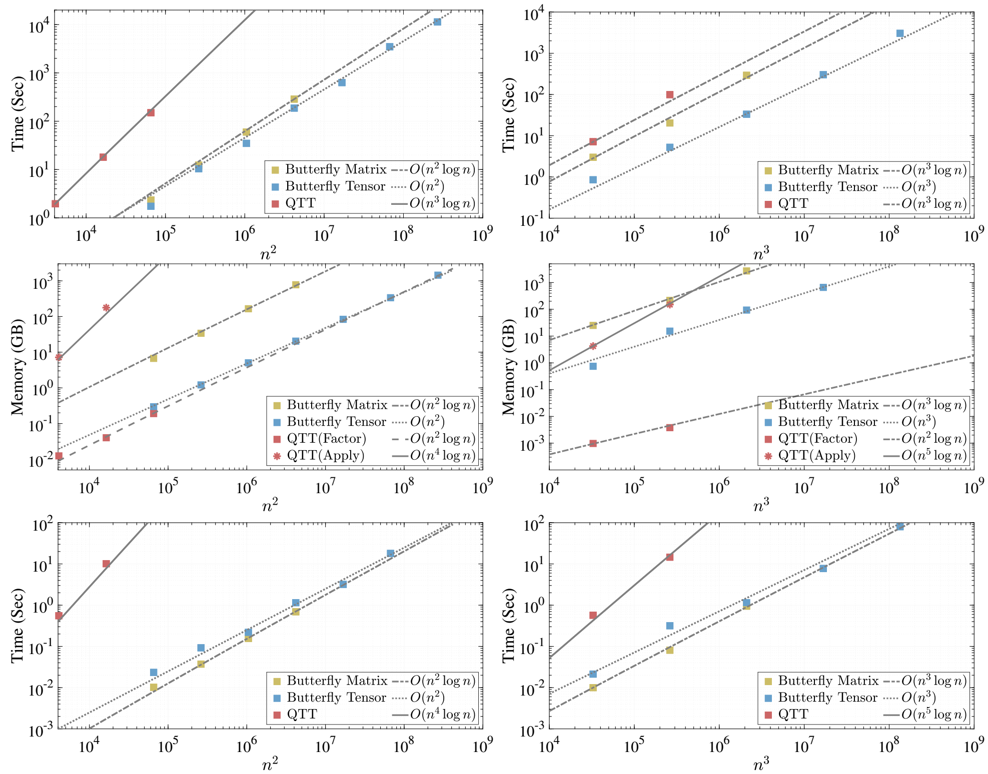

Fig. 4.2: Radon transforms: Computational complexity comparison among butterfly matrix, butterfly tensor and QTT for compressing (left) a 2D Radon transform tensor and (right) a 3D Radon transform tensor. (Top): Factor time. (Middle): Factor and apply memory. (Bottom): Apply time.

Figure 4.2 shows the factorization time, application time and memory usage of each algorithm using a compression tolerance $\epsilon = 10^{-3}$ for the 2D transform (left) and 3D transform (right). Overall, we achieve the expected complexities listed in Table 3.1 for all three algorithms. The QTT algorithm can only obtain the first 2 or 3 data points due to its high memory usage and large QTT ranks. In comparison, the tensor butterfly algorithm achieves the linear CPU and memory complexities for both factorization and application with a much smaller prefactor compared to all the other algorithms. Note that the Radon transform kernels in (4.4) and (4.5) are not translational invariant, but the tensor butterfly algorithm can still attain small ranks. As a result, the tensor butterfly algorithm can handle 64x larger-sized Radon transforms compared with the matrix butterfly algorithm, showing their superiority for solving linear inverse problems in tomography and seismic imaging.

Next, we demonstrate the effect of changing compression tolerance ϵ for both test cases in Table 4.2 with the error defined by (4.2). Table 4.2 shows the minimum and maximum ranks, error, factorization time, application time and memory usage of varying ϵ, for $n = 2048$ with $d = 2$ and $n = 128$ with $d = 3$, respectively. Overall, the errors are close to the prescribed tolerances and the costs increase for smaller ϵ, as expected. Just like the Green's function example, it is critical to keep r a low constant, particularly for higher dimensions.

n^d	ϵ	$r_{\min}$	r	error	T_f(sec)	T_a (sec)	Mem (MB)
2048^2	1E-02	4	18	2.04E-02	9.32E+01	7.20E-01	1.25E+04
2048^2	1E-03	4	20	1.51E-03	1.61E+02	1.28E+00	2.40E+04
2048^2	1E-04	4	22	1.49E-04	2.55E+02	2.05E+00	4.26E+04
2048^2	1E-05	4	23	2.45E-05	3.73E+02	3.12E+00	6.95E+04
128^3	1E-02	2	6	4.31E-02	3.89E+01	8.57E-01	1.59E+04
128^3	1E-03	2	8	1.00E-02	1.31E+02	3.74E+00	9.44E+04
128^3	1E-04	2	9	1.68E-03	2.42E+02	8.28E+00	2.38E+05
128^3	1E-05	2	11	1.48E-04	4.30E+02	2.05E+01	6.06E+05

Table 4.2: The technical data for a 4-way Radon transform tensor of $n = 2048$ in (4.4) and a 6-way Radon transform tensor of $n = 128$ in (4.5) using the proposed tensor butterfly algorithm of varying compression tolerance ϵ. The table shows the maximum rank r and minimum rank $r_{\min}$ across all ID operations, relative error in (4.2), factor time T_f, apply time T_a, and memory usage Mem..

4.3. High-dimensional discrete Fourier transform. Finally, we consider high-dimensional DFTs defined as

$$\boldsymbol{\mathcal{K}}(\boldsymbol{i}, \boldsymbol{j}) = \exp(2\pi \mathrm{i} x^{\boldsymbol{i}} \cdot y^{\boldsymbol{j}}), \tag{4.6}$$

where we choose $x^{\boldsymbol{i}} = (i_1 - 1, i_2 - 1, \ldots, i_d - 1)$ and $y^{\boldsymbol{j}} = (\frac{j_1-1}{n}, \frac{j_2-1}{n}, \ldots, \frac{j_d-1}{n})$ for uniform DFTs, and we choose $x^{\boldsymbol{i}}$ to be random (in the sense that $x_k^{\boldsymbol{i}} \in [0, n-1]$ for $k \leq d$ is a random number) and $y^{\boldsymbol{j}} = (\frac{j_1-1}{n}, \frac{j_2-1}{n}, \ldots, \frac{j_d-1}{n})$ for type-2 non-uniform DFTs. For high-dimensional DFTs with $d = 3, 4, 5, 6$, we perform compression using the tensor butterfly algorithms (with the bit-reversal ordering for each dimension), and perform application/contraction using a random input tensor $\boldsymbol{\mathcal{F}}$. In comparison, for $d = 3$ we perform FFT via the heFFTe package for the uniform DFT example and NUFFT via the FINUFFT package for the type-2 non-uniform DFT example.

Figure 4.3 shows the factorization time for the butterfly algorithm (or equivalently the plan creation time for heFFTe/FINUFFT), application time and memory usage of each algorithm using a compression tolerance $\epsilon = 10^{-3}$ (for butterfly and FINUFFT) for the uniform (left) and nonuniform (right) transforms. Overall, the tensor butterfly algorithm can obtain $O(n^d)$ CPU and memory complexities compared with the $O(n^d \log n)$ complexities of FFT and NUFFT. It is also worth mentioning that QTT can attain logarithmic-complexity uniform DFTs [9] when the input tensor $\boldsymbol{\mathcal{F}}$ is also in the QTT form with low TT ranks. However, for a general input tensor, the complexity of QTT falls back to $O(n^d \log n)$. Although the proposed tensor butterfly algorithm can obtain the best computational complexity among all existing algorithms, we observe that for the $d = 3$ case, FFT or NUFFT shows a memory usage similar to the tensor butterfly algorithm but much smaller prefactors for plan creation and application time. That said, the tensor butterfly algorithm provides a unique capability to perform higher dimensional DFTs (i.e., $d \geq 4$) with optimal asymptotic complexities.

5. Conclusion. We present a new tensor butterfly algorithm efficiently compressing and applying large-scale and high-dimensional OIOs, such as Green's functions for wave equations and integral transforms, including Radon transforms and Fourier transforms. The tensor butterfly algorithm leverages an essential tensor CLR

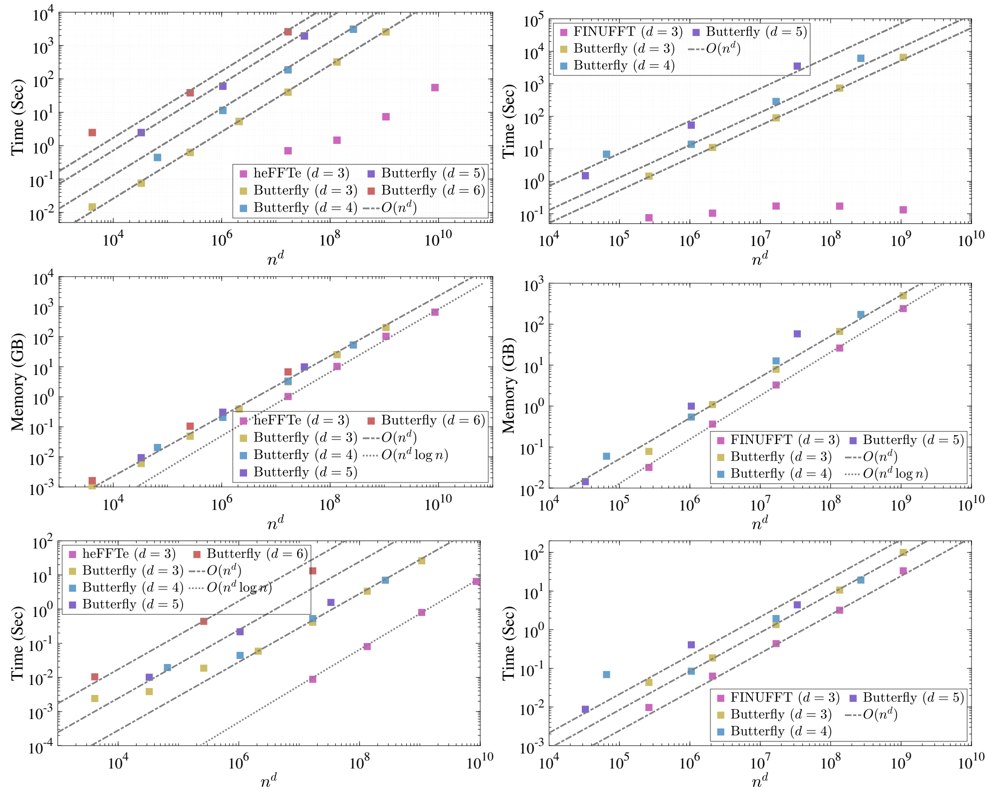

Fig. 4.3: Fourier transforms: Computational complexity of (left) butterfly tensor and heFFTe for compressing the high-dimensional DFT tensor and (right) butterfly tensor and FINUFFT for compressing the high-dimensional NUFFT tensor. (Top): Factor time of butterfly tensor and plan creation time for heFFTe/FINUFFT. (Middle): Factor memory. (Bottom): Apply time.

property to achieve both improved asymptotic computational complexities and lower leading constants. For the contraction of high-dimensional OIOs with arbitrary input tensors, the tensor butterfly algorithm achieves the optimal linear CPU and memory complexities; this is in huge contrast with both existing matrix algorithms and fast transform algorithms. The former includes the matrix butterfly algorithm, and the latter contains FFT, NUFFT, and other tensor algorithms such as Tucker-type decompositions and QTT. Nevertheless, all these algorithms exhibit higher asymptotic complexities and larger leading constants. As a result, the tensor butterfly algorithm can efficiently model high-frequency 3D Green's function interactions with over 512× larger problem sizes than existing butterfly algorithms; for the largest sized tensor that can be handled by existing algorithms, the tensor butterfly algorithm requires 200× less CPU time and 30× less memory than existing algorithms. Moreover, it can perform linear-complexity Radon transforms and DFTs with up to $d = 6$ dimensions. These OIOs are frequently encountered in the solution of high-frequency wave equations, X-ray and MRI-based inverse problems, seismic imaging and signal processing; therefore, we expect the tensor butterfly algorithm developed here to be both theoretically attractive and practically useful for many applications.

The limitation of the tensor butterfly algorithm is the requirement for a tensor grid, and hence its extension for unstructured meshes will be a future work. Also, the mid-level subtensors represent a memory bottleneck and need to be compressed with more efficient algorithms.

Acknowledgements. This research has been supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Mathematical Multifaceted Integrated Capability Centers (MMICCs) program, under Contract No. DE-AC02-05CH11231 at Lawrence Berkeley National Laboratory. This work was also supported in part by the NSF Mathematical Sciences Graduate Internship (NSF MSGI) program. This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231 using NERSC award ASCR-ERCAP0024170. Qian's research was partially supported by NSF grants 2152011 and 2309534 and an MSU SPG grant.

REFERENCES

[1] Alan Ayala, Stanimire Tomov, Piotr Luszczek, Sebastien Cayrols, Gerald Ragghianti, and Jack Dongarra. Analysis of the communication and computation cost of FFT libraries towards exascale. Technical Report ICL-UT-22-07, 2022-07 2022.

[2] Alexander H Barnett, Jeremy Magland, and Ludvig af Klinteberg. A parallel nonuniform fast Fourier transform library based on an "exponential of semicircle" kernel. *SIAM J. Sci. Comput.*, 41(5):C479–C504, 2019.

[3] David Joseph Biagioni. *Numerical construction of Green's functions in high dimensional elliptic problems with variable coefficients and analysis of renewable energy data via sparse and separable approximations*. PhD thesis, University of Colorado at Boulder, 2012.

[4] James Bremer, Ze Chen, and Haizhao Yang. Rapid application of the spherical harmonic transform via interpolative decomposition butterfly factorization. *SIAM J. Sci. Comput.*, 43(6):A3789–A3808, 2021.

[5] Ovidio M Bucci and Giorgio Franceschetti. On the degrees of freedom of scattered fields. *IEEE Trans. Antennas Propagat.*, 37(7):918–926, 1989.

[6] HanQin Cai, Keaton Hamm, Longxiu Huang, and Deanna Needell. Mode-wise tensor decompositions: Multi-dimensional generalizations of cur decompositions. *J. Mach. Learn. Res.*, 22(185):1–36, 2021.

[7] Emmanuel Candes, Laurent Demanet, and Lexing Ying. Fast computation of Fourier integral operators. *SIAM J. Sci. Comput.*, 29(6):2464–2493, 2007.

[8] Emmanuel Candès, Laurent Demanet, and Lexing Ying. A fast butterfly algorithm for the computation of Fourier integral operators. *SIAM Multiscale Model. Simul.*, 7(4):1727–1750, 2009.

[9] Jielun Chen and Michael Lindsey. Direct interpolative construction of the discrete Fourier transform as a matrix product operator. *arXiv preprint arXiv:2404.03182*, 2024.

[10] Ze Chen, Juan Zhang, Kenneth L Ho, and Haizhao Yang. Multidimensional phase recovery and interpolative decomposition butterfly factorization. *J. Comput. Phys.*, 412:109427, 2020.

[11] Jian Cheng, Dinggang Shen, Peter J Basser, and Pew-Thian Yap. Joint 6D kq space compressed sensing for accelerated high angular resolution diffusion MRI. In *Information Processing in Medical Imaging: 24th International Conference, IPMI 2015, Sabhal Mor Ostaig, Isle of Skye, UK, June 28-July 3, 2015, Proceedings*, pages 782–793. Springer, 2015.

[12] Andrzej Cichocki, Namgil Lee, Ivan Oseledets, Anh-Huy Phan, Qibin Zhao, Danilo P Mandic, et al. Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions. *Found. Trends Mach. Learn.*, 9(4-5):249–429, 2016.

[13] Lisa Claus, Pieter Ghysels, Yang Liu, Thái Anh Nhan, Ramakrishnan Thirumalaisamy, Amneet Pal Singh Bhalla, and Sherry Li. Sparse approximate multifrontal factorization with composite compression methods. *ACM Trans. Math. Softw.*, 49(3):1–28, 2023.

[14] Eduardo Corona, Abtin Rahimian, and Denis Zorin. A tensor-train accelerated solver for integral equations in complex geometries. *J. Comput. Phys.*, 334:145–169, 2017.

[15] Maurice A De Gosson. *The Wigner Transform*. World Scientific Publishing Company, 2017.

[16] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. A multilinear singular value decomposition. *SIAM J. Matrix Anal. Appl.*, 21(4):1253–1278, 2000.

[17] Stanley R Deans. *The Radon transform and some of its applications*. Courier Corporation,

2007.
[18] Gian Luca Delzanno. Multi-dimensional, fully-implicit, spectral method for the Vlasov–Maxwell equations with exact conservation laws in discrete form. *J. Comput. Phys.*, 301:338–356, 2015.
[19] Sergey Dolgov and Dmitry Savostyanov. Parallel cross interpolation for high-precision calculation of high-dimensional integrals. *Comput. Phys. Commun.Communications*, 246:106869, 2020.
[20] Björn Engquist and Lexing Ying. Fast directional multilevel algorithms for oscillatory kernels. *SIAM J. Sci. Comput.*, 29(4):1710–1737, 2007.
[21] Alexander L Fetter and John Dirk Walecka. *Quantum theory of many-particle systems*. Courier Corporation, 2012.
[22] Ilias I Giannakopoulos, Mikhail S Litsarev, and Athanasios G Polimeridis. Memory footprint reduction for the FFT-based volume integral equation method via tensor decompositions. *IEEE Trans. Antennas Propagat.*, 67(12):7476–7486, 2019.
[23] Lars Grasedyck, Daniel Kressner, and Christine Tobler. A literature survey of low-rank tensor approximation techniques. *GAMM-Mitteilungen*, 36(1):53–78, 2013.
[24] Han Guo, Jun Hu, and Eric Michielssen. On MLMDA/butterfly compressibility of inverse integral operators. *IEEE Antennas Wirel. Propag. Lett.*, 12:31–34, 2013.
[25] Han Guo, Yang Liu, Jun Hu, and Eric Michielssen. A butterfly-based direct integral-equation solver using hierarchical LU factorization for analyzing scattering from electrically large conducting objects. *IEEE Trans. Antennas Propag.*, 65(9):4742–4750, 2017.
[26] Han Guo, Yang Liu, Jun Hu, and Eric Michielssen. A butterfly-based direct solver using hierarchical LU factorization for Poggio-Miller-Chang-Harrington-Wu-Tsai equations. *Microw Opt Technol Lett.*, 60:1381–1387, 2018.
[27] Wolfgang Hackbusch and Boris N Khoromskij. Tensor-product approximation to multidimensional integral operators and Green's functions. *SIAM J. Matrix Anal. Appl.*, 30(3):1233–1253, 2008.
[28] Wolfgang Hackbusch and Stefan Kühn. A new scheme for the tensor representation. *J. Fourier Anal. Appl.*, 15(5):706–722, 2009.
[29] Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, January 2011.
[30] Richard A Harshman et al. Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis. *UCLA working papers in phonetics*, 16(1):84, 1970.
[31] Rui Hong, Ya-Xuan Xiao, Jie Hu, An-Chun Ji, and Shi-Ju Ran. Functional tensor network solving many-body Schrödinger equation. *Phys. Rev. B*, 105:165116, Apr 2022.
[32] L Hörmander. Fourier integral operators. i. In *Mathematics Past and Present Fourier Integral Operators*, pages 23–127. Springer, 1994.
[33] Yuehaw Khoo and Lexing Ying. Switchnet: a neural network model for forward and inverse scattering problems. *SIAM J. Sci. Comput.*, 41(5):A3182–A3201, 2019.
[34] Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.
[35] Matthew Li, Laurent Demanet, and Leonardo Zepeda-Núñez. Wide-band butterfly network: stable and efficient inversion via multi-frequency neural networks. *SIAM Multiscale Model. Simul.*, 20(4):1191–1227, 2022.
[36] Yingzhou Li and Haizhao Yang. Interpolative butterfly factorization. *SIAM J. Sci. Comput.*, 39(2):A503–A531, 2017.
[37] Yingzhou Li, Haizhao Yang, Eileen R Martin, Kenneth L Ho, and Lexing Ying. Butterfly factorization. *SIAM Multiscale Model. Simul.*, 13(2):714–732, 2015.
[38] Yingzhou Li, Haizhao Yang, and Lexing Ying. Multidimensional butterfly factorization. *Appl. Comput. Harmon. Anal.*, 44(3):737–758, 2018.
[39] E. Liberty, F. Woolfe, P.-G. Martinsson, V. Rokhlin, and M. Tygert. Randomized algorithms for the low-rank approximation of matrices. *Proc. Natl. Acad. Sci. USA*, 104:20167–20172, 2007.
[40] Peter Lindstrom. Fixed-rate compressed floating-point arrays. *IEEE Trans. Vis. Comput. Graph.*, 20(12):2674–2683, 2014.
[41] Yang Liu. A comparative study of butterfly-enhanced direct integral and differential equation solvers for high-frequency electromagnetic analysis involving inhomogeneous dielectrics. In *2022 3rd URSI Atlantic and Asia Pacific Radio Science Meeting (AT-AP-RASC)*, pages 1–4. IEEE, 2022.
[42] Yang Liu, Pieter Ghysels, Lisa Claus, and Xiaoye Sherry Li. Sparse approximate multifrontal

factorization with butterfly compression for high-frequency wave equations. *SIAM J. Sci. Comput.*, 0(0):S367–S391, 2021.

[43] Yang Liu, Han Guo, and Eric Michielssen. An HSS matrix-inspired butterfly-based direct solver for analyzing scattering from two-dimensional objects. *IEEE Antennas Wirel. Propag. Lett.*, 16:1179–1183, 2017.

[44] Yang Liu, Tianhuan Luo, Aman Rani, Hengrui Luo, and Xiaoye Sherry Li. Detecting resonance of radio-frequency cavities using fast direct integral equation solvers and augmented Bayesian optimization. *IEEE J. Multiscale Multiphysics Comput. Tech.*, 2023.

[45] Yang Liu, Jian Song, Robert Burridge, and Jianliang Qian. A fast butterfly-compressed Hadamard-Babich integrator for high-frequency Helmholtz equations in inhomogeneous media with arbitrary sources. *SIAM Multiscale Model. Simul.*, 21(1):269–308, 2023.

[46] Yang Liu and USDOE. ButterflyPACK, 11 2018. URL: https://github.com/liuyangzhuan/ButterflyPACK.

[47] Yang Liu, Xin Xing, Han Guo, Eric Michielssen, Pieter Ghysels, and Xiaoye Sherry Li. Butterfly factorization via randomized matrix-vector multiplications. *SIAM J. Sci. Comput.*, 43(2):A883–A907, 2021.

[48] Yang Liu and Haizhao Yang. A hierarchical butterfly LU preconditioner for two-dimensional electromagnetic scattering problems involving open surfaces. *J. Comput. Phys.*, 401:109014, 2020.

[49] W. Lu, J. Qian, and R. Burridge. Babich's expansion and the fast Huygens sweeping method for the Helmholtz wave equation at high frequencies. *J. Comput. Phys.*, 313:478–510, 2016.

[50] Axel Maas. Two and three-point Green's functions in two-dimensional Landau-gauge Yang-Mills theory. *Phys. Rev. D*, 75:116004, 2007.

[51] Michael W Mahoney, Mauro Maggioni, and Petros Drineas. Tensor-CUR decompositions for tensor-based data. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 327–336, 2006.

[52] Osman Asif Malik and Stephen Becker. Fast randomized matrix and tensor interpolative decomposition using countsketch. *Adv. Comput. Math.*, 46(6):76, 2020.

[53] Eric Michielssen and Amir Boag. Multilevel evaluation of electromagnetic fields for the rapid solution of scattering problems. *Microw Opt Technol Lett.*, 7(17):790–795, 1994.

[54] Eric Michielssen and Amir Boag. A multilevel matrix decomposition algorithm for analyzing scattering from large structures. *IEEE Trans. Antennas Propag.*, 44(8):1086–1093, 1996.

[55] Rachel Minster, Arvind K Saibaba, and Misha E Kilmer. Randomized algorithms for low-rank tensor decompositions in the Tucker format. *SIAM journal on mathematics of data science*, 2(1):189–215, 2020.

[56] Michael O'Neil, Franco Woolfe, and Vladimir Rokhlin. An algorithm for the rapid evaluation of special function transforms. *Appl. Comput. Harmon. A.*, 28(2):203 – 226, 2010. Special Issue on Continuous Wavelet Transform in Memory of Jean Morlet, Part I.

[57] Ivan V Oseledets. Tensor-train decomposition. *SIAM J. Sci. Comput.*, 33(5):2295–2317, 2011.

[58] Qiyuan Pang, Kenneth L. Ho, and Haizhao Yang. Interpolative decomposition butterfly factorization. *SIAM J. Sci. Comput.*, 42(2):A1097–A1115, 2020.

[59] Michael E Peskin. *An introduction to quantum field theory*. CRC press, 2018.

[60] Arvind K Saibaba. HOID: higher order interpolatory decomposition for tensors based on Tucker representation. *SIAM J. Matrix Anal. Appl.*, 37(3):1223–1249, 2016.

[61] Sadeed Bin Sayed, Yang Liu, Luis J. Gomez, and Abdulkadir C. Yucel. A butterfly-accelerated volume integral equation solver for broad permittivity and large-scale electromagnetic analysis. *IEEE Trans. Antennas Propagat.*, 70(5):3549–3559, 2022.

[62] Weitian Sheng, Abdulkadir C Yucel, Yang Liu, Han Guo, and Eric Michielssen. A domain decomposition based surface integral equation simulator for characterizing EM wave propagation in mine environments. *IEEE Trans. Antennas Propagat.*, 2023.

[63] Tianyi Shi, Daniel Hayes, and Jing-Mei Qiu. Distributed memory parallel adaptive tensor-train cross approximation. *arXiv preprint arXiv:2407.11290*, 2024.

[64] Edgar Solomonik, Devin Matthews, Jeff Hammond, and James Demmel. Cyclops tensor framework: Reducing communication and eliminating load imbalance in massively parallel contractions. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, pages 813–824. IEEE, 2013.

[65] Mark Tygert. Fast algorithms for spherical harmonic expansions, III. *J. Comput. Phys.*, 229(18):6181 – 6192, 2010.

[66] Mingyu Wang, Cheng Qian, Enrico Di Lorenzo, Luis J Gomez, Vladimir Okhmatovski, and Abdulkadir C Yucel. SuperVoxHenry: Tucker-enhanced and FFT-accelerated inductance extraction for voxelized superconducting structures. *IEEE Trans. Appl. Supercond.*, 31(7):1–11, 2021.

[67] Mingyu Wang, Cheng Qian, Jacob K White, and Abdulkadir C Yucel. VoxCap: FFT-accelerated and Tucker-enhanced capacitance extraction simulator for voxelized structures. *IEEE Trans. Microw. Theory Tech.*, 68(12):5154–5168, 2020.
[68] E. Wigner. On the quantum correction for thermodynamic equilibrium. *Phys. Rev.*, 40:749–759, Jun 1932.
[69] Haizhao Yang. A unified framework for oscillatory integral transforms: When to use NUFFT or butterfly factorization? *J. Comput. Phys.*, 388:103 – 122, 2019.
[70] Lexing Ying. Sparse Fourier transform via butterfly algorithm. *SIAM J. Sci. Comput.*, 31(3):1678–1694, 2009.