Titre: Title:	A Split Fast Fourier Transform Algorithm for Block Toeplitz Matrix-Vector Multiplication
Auteur: Author:	Alexandre Siron
Date:	2024
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Siron, A. (2024). A Split Fast Fourier Transform Algorithm for Block Toeplitz Matrix-Vector Multiplication [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/59194/

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

A Split Fast Fourier Transform Algorithm for Block Toeplitz Matrix-Vector Multiplication

ALEXANDRE SIRON
Département de génie physique

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie physique

Août 2024

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

A Split Fast Fourier Transform Algorithm for Block Toeplitz Matrix-Vector Multiplication

présenté par **Alexandre SIRON**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Maksim SKOROBOGATIY, président
Sean MOLESKY, membre et directeur de recherche
Juan Nicolás QUESADA MEJIA, membre

DEDICATION

I dedicate this thesis to my family - even if it is not as easy to hang on the wall as a poster. Thank you for your encouragement and your trust. Thank you for supporting my decision to study on the other side of the planet.

To my girlfriend, who not only gave me unconditional emotional support throughout the entire writing process of this thesis, but also spent countless hours reading it.

To my professor, I am honored to be your first student. I am grateful we could share this journey together.

To my fellow lab colleagues, for those leaving at the end of the summer, it was a pleasure sharing this white board with you. For the others, I am eager to continue sharing it with you in the years to come.

Finally, to Lyne Dénommé, I would like to thank you for guiding me through the winding maze of Polytechnique's administration procedures.

ACKNOWLEDGEMENTS

I would first and foremost like to express my gratitude to my professor, Sean Molesky. This project would not have been possible without your unwavering guidance and support during my master's degree.

I would also like to acknowledge and give my sincerest thanks to Paul Virally and Manon Beaufort for their careful reading and feedback on this thesis. I am especially grateful to Paul Virally for his support and guidance through the use of different packages used in this project.

This work was supported by the Québec Ministry of Economy of Innovation through the Québec Quantum Photonics Program, the National Sciences and Engineering Research Council of Canada via Discovery Grant RGPIN-2023-05818, and the Canada First Research Excellence Fund through its collaboration with the Institut de Valorisation des Données (IVADO).

RÉSUMÉ

La modélisation des phénomènes physiques invariants par translation, dans une base vectorielle spatiale, entraine des systèmes d'équations qui peuvent être représentés par des matrices de Toeplitz par bloc multi-niveaux. C'est notamment le cas des phénomènes liés à l'électromagnétisme ou l'acoustique. La résolution d'équations impliquant ces matrices requiert généralement une méthode de résolution itérative, nécessitant un grand nombre de multiplications de matrice de Toeplitz par bloc avec des vecteurs. Ainsi, pour simuler des problèmes physiques de grande taille, il est indispensable d'améliorer l'efficacité de ces opérations. Il existe un grand nombre de méthodes réalisant ces multiplications, mais chacune présente des limites majeures. Les plus prometteuses de ces méthodes sont celle par insertion de matrices circulantes et celle par décomposition tensorielle en paquet. Malheureusement, ces méthodes présentent des temps de calculs et des besoins en mémoire qui augmentent rapidement avec la taille du système ou le rang du tenseur. Dans le cas de vecteurs sources quelconques, le calcul de la décomposition tensorielle est plus coûteux que la multiplication directe. Aussi, lorsque l'on étend l'insertion de matrices circulantes aux matrices de Toeplitz par bloc en d dimensions, l'insertion entraine l'introduction d'un grand nombre de coefficients redondants. Dans ce cas, seul une proportion de $1/2^d$ du vecteur traité dans le calcul contient des informations utiles. Cette redondance conduit rapidement à une surcharge de mémoire et à une complexité opérationnelle élevée.

L'algorithme présenté dans ce mémoire a pour objectif de répondre à ce problème d'inefficacité en reportant au plus tard les insertions et en avançant au plus tôt les projections. Cette stratégie permet de calculer les transformées de Fourier sur un nombre réduit de coefficients, diminuant ainsi la complexité par un facteur de $d/\left(2-2^{-d+1}\right)$ et la demande en mémoire maximale de $2/\left((d+1)2^{-d+1}\right)$. Pour une multiplication matrice-vecteur en trois dimensions, le ratio entre la complexité théorique de la méthode standard par insertion de matrices circulantes et celle de la version améliorée de cette méthode tend vers $12/7$ lorsque la taille de la matrice de Toeplitz par bloc tend vers l'infini pour chaque niveau. De plus, l'algorithme utilise une propriété de la transformée de Fourier rapide afin de séparer le vecteur transformé en deux vecteurs de coefficients d'indices paires et impaires respectivement, et ce pour chaque transformée de Fourier. Cela résulte en une structure arborescente des tâches de l'algorithme. Cette structure permet une meilleure flexibilité dans la gestion de la mémoire et de la vitesse de calcul grâce à la possibilité d'appliquer différentes stratégies de parallélisation impliquant plusieurs GPU. Ainsi, l'utilisation simultanée de différents GPU, chacun

assigné à une branche, peut améliorer la vitesse de calcul au détriment d'une utilisation de mémoire accrue. Avec un unique GPU, le ratio d'allocation de mémoire entre les méthodes standard et améliorée tend vers $4/3$ pour les matrices et vecteurs en trois dimensions, lorsque les tailles de chaque niveau tendent vers l'infini. Dans les cas où la matrice de Toeplitz par blocs est symétrique ou antisymétrique, c'est-à-dire que chaque bloc de chaque niveau est symétrique ou antisymétrique et de Toeplitz, une méthode d'insertion adaptée permet de réduire l'allocation mémoire. Le ratio théorique de mémoire allouée entre les deux méthodes devient alors $(2^d + 1)/(d + 2)$. Ces symétries apparaissent dans des théories physiques qui présentent une réciprocité entre les sources et les receveurs, en particulier en électromagnétisme, en acoustique et en mécanique quantique.

Les ratios de temps de calculs mesurés entre la méthode par insertion standard et la méthode par insertion séparée indiquent que l'algorithme proposé est plus efficace que les attendus théoriques tant qu'elle n'a pas atteint sa capacité mémoire. Une analyse supplémentaire des temps de calculs consécutifs pour la méthode par insertion séparée révèle que les écarts aux attendus théoriques sont en partie causés par les méthodes utilisées par le package « FFTW.jl » pour gérer les calculs de transformées de Fourier rapides. En effet, ce package emploie automatiquement différentes méthodes pour le calcul des transformées de Fourier rapides selon les tailles des matrices et vecteurs multiniveaux. Sachant que les vecteurs à transformer n'ont pas la même taille dans les deux méthodes, cela explique en partie les écarts observés.

ABSTRACT

Modeling translationally invariant physics, such as electromagnetism, in a spatial basis typically results in systems of equations involving multi-level block-Toeplitz matrices. These matrices often necessitate iterative solution methods, which in turn require numerous block-Toeplitz matrix-vector multiplications. Therefore, enhancing the efficiency of these multiplications is essential for simulating a wide range of large-scale physical systems. While various methods for performing these multiplications currently exist, each comes with major drawbacks. Specifically, the most promising methods—circulant embedding and tensor train decomposition—suffer from poor scalability with respect to system size or tensor rank. For general source vectors, computing the tensor decomposition is more computationally intensive than direct multiplication. When extended to d-dimensional block-Toeplitz matrices, embedding leads to the introduction of a large number of redundant coefficients, so that only $1/2^d$ of the vector treated in computation contains useful information. This scaling quickly leads to memory overload and high operational complexity.

The algorithm introduced in this thesis aims to address this latter inefficiency through lazy embeddings and eager projections, deferring embeddings as late as possible and performing projections as early as possible. This strategy allows for the computation of Fourier transforms on a reduced number of coefficients, lowering complexity by a factor of $d/\left(2-2^{-d+1}\right)$ and peak memory usage by a factor of $2/\left((d+1)2^{-d+1}\right)$. For three-dimensional matrix-vector multiplications, the theoretical complexity ratio of the standard circulant embedding method over the enhanced version approaches $12/7$ as the size of the block-Toeplitz matrix tends to infinity for each level. Additionally, the algorithm exploits a property of the fast Fourier transform (FFT) to divide the transformed vector into even and odd coefficients, following a tree branch structure. This structure offers greater flexibility in managing memory and computational speed through various parallelization strategies along each branch using several GPUs. Running these branches independently on separate GPUs can boost computational speed at the expense of increased memory usage. On a single GPU, the memory usage ratio between the standard and enhanced methods tends to $4/3$ for three-dimensional matrices and vectors as the sizes of each level increase. In cases where the Toeplitz data is entirely symmetric or anti-symmetric—meaning each block at each level is a symmetric or anti-symmetric Toeplitz matrix——a clever embedding can reduce memory consumption. The theoretical memory ratio between the two methods then becomes $(2^d+1)/(d+2)$. These symmetries occur in any physical theory exhibiting reciprocity between sources and receivers,

and, in particular, arise in electromagnetics, acoustics, and quantum mechanics.

The measured time complexity ratios between the split and standard circulant embedding methods indicate that the proposed algorithm usually outperforms theoretical expectations until the machine memory limits. Further analysis of the consecutive computational times of the split circulant embedding method revealed that the discrepancies were due to the "FFTW.jl" package's method for handling FFT computations. Indeed, this package employs different methods for FFT computation depending on the sizes of the multi-level matrices. The discrepancy in computational time ratios is partly due to the fact that the two methods perform FFTs on different vector sizes.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

(F)FT	(Fast) Fourier Transform
DFT	Discrete Fourier Transform
IFFT	Inverse Fourier Transform
(Q)TT	(Quantized) Tensor Train
CDP	CANDECOMP/PARAFAC
CPU	Central Processing Unit
GPU	Graphics Processing Unit
RAM	Random Access Memory
PDE	Partial Differential Equation
SVD	Singular Value Decomposition
PML	Perfectly Matched Layer
DMRG	Density Matrix Renormalization Group

CHAPTER 1 INTRODUCTION

Multi-level block Toeplitz matrices are widely applicable across mathematics, physics, and engineering. In mathematics, these matrices are used in compressed sensing [1], cryptography [2], and modeling multi-dimensional random processes [3]. In physics, block Toeplitz matrices play a crucial role in multi-dimensional wave propagation equations, particularly in acoustics [4], [5] and electromagnetics [6], [7]. Additionally, they are employed in signal processing [8]. Due to their extensive range of applications, mathematicians and physicists have developed various techniques to optimize operations involving these matrices. This thesis focuses its application of Toeplitz matrix-vector multiplication to simulations of electromagnetic fields, where the Green's function matrix exhibits a block Toeplitz structure due to the translational invariance of these fields. Nevertheless, the algorithm discussed herein is broadly applicable to all the aforementioned applications.
Our research project aims to enhance the existing circulant embedding method for Toeplitz matrix-vector multiplication. This introduction will present the mathematical concepts underpinning this method, highlighting its limitations and identifying targets for improvement. We will begin by introducing key mathematical concepts, including the Fast Fourier Transform (FFT) (section 1.1.1), Toeplitz and block Toeplitz matrices (section 1.1.2), and the properties of circulant matrices (section 1.1.3). Following this, we will explain the circulant embedding method and delineate its limitations (section 1.2) in order to state the research objectives (section 1.3). Finally, we will provide an outline of the thesis, detailing the organization of its chapters (section 1.4).

Notations

Throughout this thesis, matrices are labeled in capital letters, vectors in bold letters and scalar coefficients are labeled in small letters.
Complexities are given in $O(f(n))$, where O is the order of approximation symbol in the asymptotic notation.

1.1 Mathematical background

To understand the circulant embedding method, it is essential to grasp its two key components: the transformation into Fourier space and the circulant embedding of a Toeplitz ma-

trix. The Fourier transform can be computed in various ways, but the most efficient method employs the Cooley-Tukey fast Fourier transform algorithm [9]. This approach leverages the properties of circulant matrices in Fourier space, reducing the complexity of the multiplication to a linear operation.

1.1.1 Fast Fourier transform algorithm and its properties

Most of the reordering-based methods for Toeplitz matrix-vector multiplication are performed in Fourier space. As such, the Discrete Fourier Transform (DFT) matrix can be used to perform this transformation through a linear operation. In a N-dimensional linear space, the DFT matrix Ω_N is defined to be:

$$\Omega_N = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & ... & ... & 1 \\ 1 & \omega & \omega^2 & \omega^3 & ... & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & ... & \omega^{2(N-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & ... & \omega^{3(N-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \omega^{3(N-1)} & ... & \omega^{(N-1)^2} \end{bmatrix},$$

where $\omega = e^{-\frac{2\pi i}{N}}$.

The naive approach to the Fourier transform would be to compute the matrix multiplication, which has $O(N^2)$ complexity. Thus, if we denote $\mathbf{f}$ to be the vector to transform and $\widehat{\mathbf{f}}$ its Fourier transform, $\widehat{\mathbf{f}} = \Omega_N \mathbf{f} = \left(\sum_{n=0}^{N-1} f_n e^{-2\pi \frac{k}{N} n}\right)_{k=1,...N}$. The fast Fourier transform algorithm for a vector of length $N = 2^l$ uses the following property:

$$\widehat{\mathbf{f}} = \Omega_N \mathbf{f} = \begin{bmatrix} I_{N/2} & D_{N/2} \\ I_{N/2} & -D_{N/2} \end{bmatrix} \begin{bmatrix} \Omega_{N/2} & 0 \\ 0 & \Omega_{N/2} \end{bmatrix} \begin{bmatrix} \mathbf{f}_{\text{even}} \\ \mathbf{f}_{\text{odd}} \end{bmatrix},$$

where $D_{N/2} = \text{diag}\left(1, \omega, ..., \omega^{N/2}\right)$, $\mathbf{f}_{\text{even}}$ and $\mathbf{f}_{\text{odd}}$ are the vectors containing only the coefficients with even indices of $\mathbf{f}$ and the odd ones respectively, and $I_{N/2}$ is the identity matrix of size $N/2$.

After this first decomposition, the half sized $\Omega_{N/2}$ matrix can be decomposed in the same way. Thus, for $l \in \mathbb{N}^*$, the DFT matrix multiplication can be broken down into smaller sub-multiplications. This divide and conquer method requires only $O(N \log N)$ operations to compute the Fourier transform of a vector. For sizes that are not powers of two, specific alternative methods exist, with for instance, the prime factor algorithm [10], adapted for products of relatively prime numbers. The most general case is the $N = N_1 N_2$ split with

arbitrary N_1 and N_2 using the Cooley-Tukey algorithm [9].

1.1.2 Toeplitz and multi-level block-Toeplitz matrices

In this subsection, we will consider only square matrices, as these are the only types encountered in our physical context (see Section 3.1).
Let T be a square matrix of size and of complex coefficients, with $n \in \mathbb{N}^*$. T is said to be Toeplitz if its coefficients are the same along each diagonal. Thus,

$$T = \begin{bmatrix} t_0 & t_{-1} & t_{-2} & ... & ... & t_{-n+1} \\ t_1 & t_0 & t_{-1} & \ddots & & \vdots \\ t_2 & t_1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & t_{-1} & t_{-2} \\ \vdots & & \ddots & t_1 & t_0 & t_{-1} \\ t_{n-1} & ... & ... & t_2 & t_1 & t_0 \end{bmatrix}.$$

Alternatively, one can write that $t_{i,j} = t_{i-j}$ for all $i, j = 1, ..., n$. Therefore, a square Toeplitz matrix is uniquely determined by the $2n-1$ coefficients on its first row and column $\{t_k\}_{k=1-n,...,0,...,n-1}$.
A matrix A is block Toeplitz if it is composed of sub-matrices or sub-blocks that are constant along each diagonal:

$$A = \begin{bmatrix} A_0 & A_{-1} & A_{-2} & ... & ... & A_{-n+1} \\ A_1 & A_0 & A_{-1} & \ddots & & \vdots \\ A_2 & A_1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & A_{-1} & A_{-2} \\ \vdots & & \ddots & A_1 & A_0 & A_{-1} \\ A_{n-1} & ... & ... & A_2 & A_1 & A_0 \end{bmatrix}.$$

A multi-level block Toeplitz of dimension 2 is a block Toeplitz matrix such that each of its blocks is a Toeplitz sub-matrix. Hence,

$$A_k = \begin{bmatrix} a_{k,0} & a_{k,-1} & a_{k,-2} & ... & ... & a_{k,-n+1} \\ a_{k,1} & a_{k,0} & a_{k,-1} & \ddots & & \vdots \\ a_{k,2} & a_{k,1} & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & a_{k,-1} & a_{k,-2} \\ \vdots & & \ddots & a_{k,1} & a_{k,0} & a_{k,-1} \\ a_{k,n-1} & ... & ... & a_{k,2} & a_{k,1} & a_{k,0} \end{bmatrix}, \quad \text{for each } k = 1-n, ..., 0, ..., n-1.$$

The dimension of the multi-level block Toeplitz matrix corresponds to the number of layers of sub-matrices, the one dimensional case being the simple Toeplitz matrix.

1.1.3 Circulant matrices and properties

Let $C \in \mathcal{M}_n(\mathbb{C}), n \in \mathbb{N}^*$. C is said to be circulant if its coefficients along a given row (or column) are obtained through a circular permutation of the previous row. Alternatively, a row is obtained by shifting the coefficients of the previous row to the right. Thus,

$$C = \begin{bmatrix} c_0 & c_{n-1} & c_{n-2} & ... & ... & c_1 \\ c_1 & c_0 & c_{n-1} & \ddots & & \vdots \\ c_2 & c_1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & c_{n-1} & c_{n-2} \\ \vdots & & \ddots & c_1 & c_0 & c_{n-1} \\ c_{n-1} & ... & ... & c_2 & c_1 & c_0 \end{bmatrix}.$$

A circulant matrix is uniquely determined by the n coefficients of its first column $\{c_k\}_{k=0,...n-1}$. As such, we denote $\mathcal{C}(c_0, c_1, ..., c_{n-1})$ to be the circulant matrix with first column $(c_0, c_1, ..., c_{n-1})$. Note that all circulant matrices are Toeplitz. To understand the value of circulant matrices in the context of Toeplitz matrix multiplication, let us diagonalize the arbitrary circulant matrix C. Define

$$J = \begin{bmatrix} 0 & 1 & 0 & ... & 0 \\ 0 & 0 & 1 & ... & 0 \\ \vdots & & & \ddots & \vdots \\ 0 & & & & 1 \\ 1 & 0 & 0 & ... & 0 \end{bmatrix},$$

an operator that circularly permutes the coefficients of a vector by one step. As n of these permutations consecutively put the coefficients back to their place, $J^n = I_n$, where I_n stands for the identity matrix of size n. Hence, $P = X^n - 1$ is a nullifying polynomial of order n for J so it is its characteristic polynomial. Therefore, it is diagonalisable and its eigenvalues are the n^{th} roots of unity in $\mathbb{C}$: $\omega = e^{i\frac{2\pi}{n}}$.

Let $k \in [\![0, n-1]\!]$ and define $X_k = \begin{bmatrix} 1 \\ \omega^k \\ \omega^{2k} \\ \vdots \\ \omega^{k(n-1)} \end{bmatrix}$ and note that: $JX_k = \begin{bmatrix} \omega^k \\ \omega^{2k} \\ \omega^{3k} \\ \vdots \\ 1 \end{bmatrix} = \omega^k X_k$

Therefore, X_k is the eigenvector associated with the eigenvalue ω^k of J for all $k = 0, ..., n-1$. The $\{X_k\}$ thus form an eigenbasis for J.

One can check that $\mathcal{C}(c_0, c_1, ..., c_{n-1}) = \sum_{j=0}^{n-1} c_j J^j$. Hence, C is a linear combination of the $\{J^k\}_{k=0,...,n-1}$ and the $\{X_k\}$s form also an eigenbasis of C and its eigenvalues are $\lambda_k = \sum_{j=0}^{n-1} c_j \omega^{jk}$, associated to the eigenvectors $\sum_{j=0}^{n-1} c_j X_{jk}$. The normalized transformation matrix is then

$$U = \frac{1}{\sqrt{n}} \begin{bmatrix} 1 & 1 & ... & 1 \\ 1 & \omega & ... & \omega^n \\ \vdots & \vdots & & \vdots \\ 1 & \omega^{n-1} & ... & \omega^{(n-1)^2} \end{bmatrix},$$

which is exactly the DFT matrix of size n: $C = \Omega_n D \Omega_n^\dagger$ where $D = \text{diag}(\sum_{k=0}^{n-1} c_j \omega^{kj})_{j=0,...,n-1}$.

1.2 The circulant embedding method with Fourier transformed vector convolution

An efficient way of multiplying a matrix with a vector is through diagonalization. Indeed, multiplication by the diagonal matrix of eigenvalues is very efficient. However, in general, performing the required basis transformation is very costly. As stated in the previous section, the diagonalisation of a circulant matrix C is equivalent to performing a discrete Fourier transform of each of its columns. Thus, one can check that

$$C\mathbf{x} = \Omega_n D \Omega_n^\dagger \mathbf{x} = \mathcal{F}^{-1}\left(\mathcal{F}(\mathbf{c}) \odot \mathcal{F}(\mathbf{x})\right), \tag{1.1}$$

where $\mathbf{c}$ is the generator vector of C, i.e. its first column vector, $\odot$ is the element-wise vector multiplication and $\mathcal{F}$ is the Fourier transform operation. Using the fast Fourier transform al-

gorithm at each transform allows one to perform the one dimensional circulant matrix-vector multiplication in $O(2n\log(n)+n)$ instead of the $O(n^2)$ of the naive method. The complexity related to the preparation of the matrix information is not counted in the operational complexity throughout the thesis since the matrix-vector multiplication is placed in the context of iterative methods for which several vectors are multiplied by the same matrix (see section 3.1). Thus, the complexity of the one preparation of the matrix data is negligible compared to the transformations on numerous different vectors. The spectral properties of the circulant matrix do not directly extend to the Toeplitz matrices. However, it is possible to embed a Toeplitz matrix into a circulant matrix and thus take advantage of the above property. The smallest possible embedding of a $n \times n$ Toeplitz matrix is into a $(2n-1)\times(2n-1)$ circulant one. Here, we will present a method which embeds by doubling the size of the initial matrix. Intuitively, the initial Toeplitz matrix is placed on the top left and bottom right and the diagonals between the two are extended. Thus, for a $n \times n$ Toeplitz matrix T defined as in section 1.1.2,

$$C = \begin{bmatrix} T & S \\ S & T \end{bmatrix},$$

where S is a Toeplitz matrix defined as $S_{i,j} = s_{i-j}$ with $s_i = \begin{cases} t_{i-n}, & \text{if } i > 0 \\ t_{i+n}, & \text{if } i < 0 \end{cases}$ and s_0 chosen arbitrarily.

Then, $C = \mathcal{C}(t_0, t_1, ..., t_{n-1}, s_0, t_{-n+1}, ..., t_{-1})$. The matrix vector product then becomes $C\tilde{\mathbf{x}}$ where $\tilde{\mathbf{x}}$ is the $2n$-long vector $\begin{bmatrix} x_1 & ... & x_n & 0 & ... & 0 \end{bmatrix}^T$.

The computational gains provided by embedding overcome the associated increase in vector size, qualifying the circulant embedding method as the most efficient to multiply dense Toeplitz matrices with a vector [11]. For one dimensional matrices, the complexity ratio between the naive method and the circulant embedding one is

$$R = \frac{C_{\text{naive}}}{C_{\text{embed}}} = \frac{n^2}{2n\log n + n},$$

$R = 145.9$ for $n = 2^{10}$, and $R = 498.0$ for $n = 2^{12}$.

This embedding can be performed on each layer of a multi-level block Toeplitz matrix to turn it into a multi-level block circulant matrix. Each embedding also requires doubling the size of the vector $\mathbf{x}$ to make its size fit with the multi-level matrix. The final step to get the result of the matrix-vector product consists in extracting the relevant data of the embedded and multiplied vector. In the article, this final step is called the "projection", as the embedded set carrying the multi-level circulant problem is effectively projected back to its former set.

For example, consider the 2-level 2×2 block Toeplitz matrix $T = \begin{bmatrix} T_0 & T_{-1} \\ T_1 & T_0 \end{bmatrix}$ with $T_i = \begin{bmatrix} t_{0,i} & t_{-1,i} \\ t_{1,i} & t_{0,i} \end{bmatrix}$ for all $i \in \{-1, 0, 1\}$

Also consider the two level vector $\mathbf{x} = \begin{bmatrix} \begin{bmatrix} x_{0,0} \\ x_{0,1} \end{bmatrix} \\ \begin{bmatrix} x_{1,0} \\ x_{0,1} \end{bmatrix} \end{bmatrix}$.

First, embed T along all the dimension into a 2-level block circulant matrix C

$$C = \begin{bmatrix} T_0 & T_{-1} & S_0 & T_1 \\ T_1 & T_0 & T_{-1} & S_0 \\ S_0 & T_1 & T_0 & T_{-1} \\ T_{-1} & S_0 & T_1 & T_0 \end{bmatrix},$$

with $T_i = \begin{bmatrix} t_{0,i} & t_{-1,i} & s_{0,i} & t_{1,i} \\ t_{1,i} & t_{0,i} & t_{-1,i} & s_{0,i} \\ s_{0,i} & t_{1,i} & t_{0,i} & t_{-1,i} \\ t_{-1,i} & s_{0,i} & t_{1,i} & t_{0,i} \end{bmatrix}$ for all $i \in \{-1, 0, 1\}$ and where the $s_{0,i}$ and S_0 are arbitrary. Then, complete the vector $\mathbf{x}$ with additional zeros to make it fit the size of the fully embedded matrix C.

$$\tilde{\mathbf{x}} = \begin{bmatrix} \begin{bmatrix} x_{0,0} \\ x_{0,1} \\ 0 \\ 0 \end{bmatrix} & \begin{bmatrix} x_{1,0} \\ x_{0,1} \\ 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \end{bmatrix}^T.$$

Then, compute the multiplication, according to equation 1.1 and get the result vector $\tilde{\mathbf{y}}$ of size $4 \times 1 \times 4$.

Finally, project back the vector onto the original space: $\mathbf{x} = \begin{bmatrix} \begin{bmatrix} \tilde{y}_{0,0} \\ \tilde{y}_{0,1} \end{bmatrix} \\ \begin{bmatrix} \tilde{y}_{1,0} \\ \tilde{y}_{0,1} \end{bmatrix} \end{bmatrix}$.

1.3 Research objectives

For a d-dimensional block Toeplitz matrix-vector multiplication, the circulant embedding method doubles the size of each level of the multi-level vector, resulting in useful information held in only $\frac{1}{2^d}$ of the total vector (see Figure 1.1). That is, the more dimensions, the more redundant data added in the vector. In the case where all the embeddings along each of

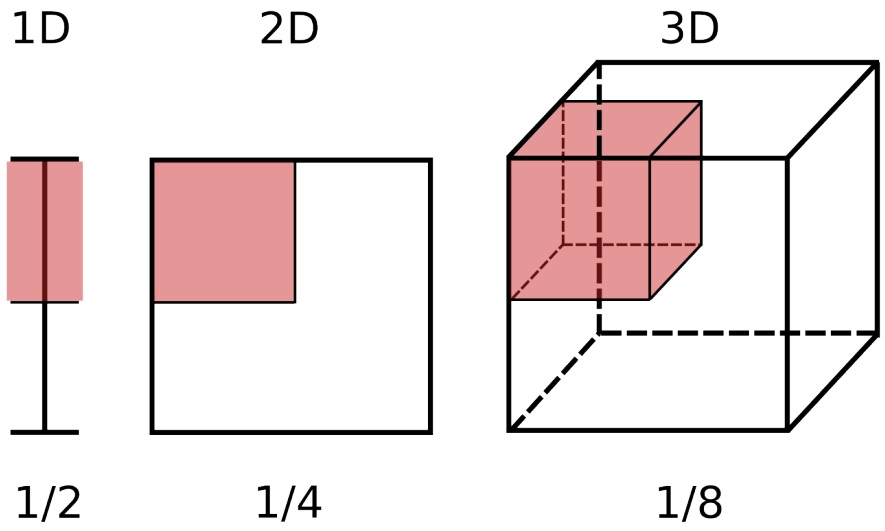

Figure 1.1 Growth of redundant coefficients in the embedded vector with the number of dimensions.

the dimensions are performed consecutively, each Fourier and inverse Fourier transform are computed on a fully "extended" vector. Moreover, the data redundancy significantly reduces the memory capacity of the simulation. Our objective is to reduce the complexity and data redundancy introduced by embedding in order to accelerate the computation of block Toeplitz matrix-vector products for applications in electromagnetics. In this context, the algorithm presented herein reorders the tasks of a "naive" implementation of the method, postponing each embedding for as long as possible in order to reduce the complexity. The algorithm also proposes a solution to better handle the memory capacity issue, proposing a tunable parallelization of the computation throughout the different levels.

1.4 Outline of the thesis

Before developing the solution proposed in the algorithm presented in the article (section 4), we will review the history of the tools used in the circulant embedding method (section

2.1) and then explore an alternative solution to perform the block Toeplitz matrix-vector product (section 2.2). Next, we will outline the physical context that motivated this problem to thoroughly identify all the constraints it entails (section 3.1). This will help us state the objectives that the algorithm will have to fulfill (section 3.2). Then, the algorithm will be presented in an article submitted to *IEEE* on the 25^{th} of June 2024 and still being reviewed at the time this thesis is submitted. After presenting the algorithm, we will analyse complementary results to better understand the cause of the discrepancies between the theoretical expectations and the computational time measures observed in the results of the article (section 5.1). In that chapter, we will also give our conclusions on the algorithm performance compared to the alternative method stated in section 2.2 and then give additional details on the tunability of the parallelization for memory management purposes, sections 5.2 and 5.3 respectively.

CHAPTER 2 LITERATURE REVIEW

Existing literature provides several methods to efficiently perform block Toeplitz matrix-vector multiplication, depending on the characteristics of the involved data. These methods can be divided into two categories: reordering based methods and tensor decomposition based methods. Among the reordering methods, none of them are as powerful on the same scope of general dense block Toeplitz matrices as the circulant embedding method described in section 1.2. The exception to this statement is Cariow's work on one dimensional Toeplitz matrices, which claims better complexity than the circulant embedding in Fourier space [12]. However, the method does not seem applicable to large multi-dimensional block Toeplitz structures. Tensor decomposition methods display a greater variety of solutions to the multiplication problem by the numerous types of existing decomposition (rank based, by matrix multiplication, etc).

The following chapter provides a comprehensive review of existing multi-level block Toeplitz matrix-vector multiplication methods. It begins with section 2.1 providing the historical context of the mathematical tools used in the circulant embedding method, specifically Toeplitz matrices and the FFT algorithm. This background aims to help the reader understand the development and evolution of these methods. Additionally, alternative methods based on tensor decomposition, as found in the literature, will be explored (section 2.2). The most promising of them will be further detailed in order to highlight its advantages and limitations in comparison to circulant embedding within the physical context outlined in section 3.1.

2.1 History of the multi-level Toeplitz-vector multiplication methods by Fourier transform

The circulant embedding method in Fourier space—as stated in section 1.2—is presently the most efficient reordering based method to perform the matrix-vector product of block Toeplitz matrices. The idea of embedding Toeplitz matrices into circulant ones for performance purposes was established by Dembo, Marllow and Shepp (1989), in the context of covariance estimation of Gaussian processes [13]. Lee reached the same conclusion from a different research angle in 1986, canonically associating polynomials to the Toeplitz generator vector before applying convolutions in Fourier space to perform the multiplication [14]. These two papers are the first to properly setout the circulant embedding method for fast Toeplitz-vector multiplication. In the following section, we will explore the historical context

that led to the naming of Toeplitz matrices and dive in the development of the fast Fourier transform algorithm.

2.1.1 Historical context of Toeplitz matrices

The history of Otto Toeplitz's contribution to Toeplitz matrices is described in section 1.2 of the reference [15]. Even though Toeplitz matrices' applications span over a wide variety of areas, Toeplitz's context was purely mathematical. Otto Toeplitz (1881-1940) never properly studied Toeplitz operators T as we define them today but only studied Laurent operators L. As such, Toeplitz matrices are the finite version of Laurent matrices. Otto Toeplitz published the majority of its results on the theory of forms [16]. In this article, he developed a spectral theory of Toeplitz and Laurent linear forms on $\ell^2(\mathbb{Z})$. The only remark he made on finite Laurent operators is in a bottom page note at page 355, claiming that there was no difference between the finite and the infinite cases in the theory of forms. Otto Toeplitz probably did not see interest in developing the finite Laurent operator over the infinite case. It turns out that mathematical applications to Laurent operator are quite limited, the only notable application being the representation of the spectral theorem of normal operators. On the other hand, Toeplitz operators offers a rich theory in pure analysis and a wide variety of applications, with operator used, for instance, in problems of trigonometric moments in mechanics [17], for Wiener filters in signal processing [18], or in the Wiener-Hopf equation as a decomposition method of singular functions [19].
Ultimately, it is likely the formalism developed by Toeplitz and his work on Laurent forms in [16] that led to his name being associated with the widely used finite Laurent operators.

2.1.2 The fast Fourier transform algorithm (FFT)

Paternity of the fast Fourier transform algorithm (FFT) is debated. This part aims to develop the genesis of the method's design, highly based on the research from Heideman et al. [20].

Historical context of the literature

The algorithm got its name from Cooley and Tukey's paper on numerical analysis in 1965 [9]. They present it as a mean of computing the discrete Fourier transform and prove its complexity in $N \log N$, faster than the N^2 standard method. They claim that Good's work on the prime factor algorithm (1958) [21] was their main source of inspiration for the development

of the FFT. The work from Danielson and Lanczos in 1942 [22]—who refer to Runge (1905) [23]—could also be an indirect influence on Cooley-Tukey algorithm. Danelson, Lanczos and Runge propose a form of doubling algorithm that computes the FT of two N-points subsequence to get 2N-point Fourier transformed vector. This algorithm is not as general as Cooley-Tukey: while Danielson-Lanczos can only double the original sequence length, Cooley-Tukey allows any multiple of the original vector or sample. On the other hand, Goldstine (1977) [24] claims that an algorithm—very similar to the FFT—has been developed by Karl Friedrich Gauss and presented in a posthumous publication in 1866 [25]. This paper was presumably written in 1805, which predates Fourier's work on harmonic analysis.
Before Cooley-Tukey, Runge's algorithm was the most popular to perform discrete Fourier transforms. This method can be labeled today as a "pruned" FFT method [26] and was meant to compute FT for a small number of harmonics, using computational tables. At that time, Gauss's algorithm remained unnoticed for several reasons. First, Gauss uses outdated notations to present his algorithm in an article written in latin, due to the fact that his paper was published more than sixty years after the presumed writing period. For that same reason, the use of real trigonometric functions instead of complex exponentials did not help associating this algorithm to the FFT afterwards. Also, the lack of notice from Goldstine and from Burkhardt (1899-1916) [27]—who also mention Gauss's algorithm—did not help acknowledging his contribution to the FFT.

Gauss's algorithm

Gauss presents his algorithm in the context of orbital mechanics, treating the problem of determining the trajectory of asteroids from a finite number of observations. He wanted to use interpolation to fit N observations to a general continuous function f not exclusively even or odd, using the sum of cosine series and sine series with $\lfloor \frac{N}{2} \rfloor$ harmonics. He defines N_1 equally spaced samples and computes the m harmonics of the Fourier series. Then, he defines another set of N_1 samples, offset from the original sample by $\frac{1}{N_2}$, such that $N = N_1 N_2$. Another Fourier is computed on this new combined set of samples and Gauss realized that the coefficients of the original samples are different when the new shifted samples are added. In modern terms, he discovered that his waveform was under-sampled and the error he observed was due to aliasing of high frequency harmonics. To solve this problem, he measured N_2 sets of N_1 equally spaced samples to form a total of $N = N_1 N_2$ equally spaced samples. Finite Fourier series of the set of N samples are then computed starting with the coefficients of each N_2 sets of length N_1, shifted equally from the origin. These operations are then repeated with the N_1 sets of length N_2. A final trigonometric identity is used to convert the coefficients to finite Fourier series.

DFT of the samples:

$$C(k) = \sum_{n=0}^{N-1} X(n) W_N^{nk}, \qquad X(n) = f\left(\frac{n}{N}\right) \text{ and } W_N = e^{-\frac{2\pi i}{N}},$$

Define $n = N_2 n_1 + n_2$ and $k = k_1 + N_1 k_2$ for $n_1, k_1 = 0, ..., N_1 - 1$ and $n_2, k_2 = 0, ..., N_2 - 1$, such that

$$C(k_1 + N_1 k_2) = \underbrace{\sum_{n_2=0}^{N_2-1} \left(\underbrace{\sum_{n_1=0}^{N_1-1} X(N_2 n_1 + n_2) W_{N_1}^{n_1 k_1} W_N^{n_2 k_1}}_{N_2 \text{length-} N_1 \text{DFT corrected by} W_N} \right) W_{N_2}^{n_2 k_2}}_{N_1 \text{length-} N_2 \text{DFT}}$$

This complex exponential form of the Gauss's algorithm is equivalent to the Cooley-Tukey algorithm with the correction factor W_N called a "twiddle factor".
Finally, Gauss's algorithm is as powerful and as general as the FFT from Cooley-Tukey. It seems that the two algorithm were developed independently from one another at two different periods due to the low impact of Gauss's paper on the area of harmonics analysis.

2.2 An alternative view: the tensor decomposition

In the field of computational methods for Toeplitz-vector multiplication, the literature can be broadly categorized into two main approaches: reordering-based methods and those involving tensor decomposition techniques. Both of these approaches have been developed to efficiently handle the inherent structure of Toeplitz matrices. The method proposed in the article (section 4.3) falls within the reordering category. It employs a technique based on the circulant embedding method and optimizes the computational process by reordering the tasks of the successive embeddings and projections of each dimension. The following section will delve into the details of the tensor decomposition techniques [28], [29], [30]. It will explore how these methods can be applied to Toeplitz matrices and compare one of them [31] with the circulant embedding approach commonly used in reordering-based methods. By examining its strengths and limitations, we aim to identify aspects of the presented tensor decomposition method that could be integrated into the reordering-based method proposed in this article. This comparison will highlight potential enhancements for the algorithm presented in section 4.3.

2.2.1 Overview on different types of tensor decomposition

Several methods of decomposition of a d-dimensional (or d-mode) tensor exist in literature. These decompositions aim to cut a computationally expensive tensor into smaller or cheaper ones, depending on the purpose of the data stored inside. These approaches can be sorted into two major sub-categories: tensor rank-based decompositions, and matrix product-based decompositions.

Tensor rank or CANDECOMP/PARAFAC decomposition (CDP)

The CDP based decomposition is often referred as the "classical method" of tensor decomposition [28]. It consists in splitting a tensor into a sum of rank 1 tensors.
A N-way tensor $\mathcal{X} \in \mathbb{C}^{L_1 \times L_2 \times ... \times L_N}$ is rank one if it can be written as the outer product of N vectors, i.e., $\mathcal{X} = a(1) \otimes a(2) \otimes ... \otimes a(N)$, where $\otimes$ represents the vector outer product. This means that each element of the tensor is the product of the corresponding vector elements:

$$x_{i_1 i_2 ... i_N} = a^{(1)}_{i_1} a^{(2)}_{i_2} ... a^{(N)}_{i_N}, \text{ for all } 1 \leq i_n \leq L_n$$

As such, the CDP approximation can be written as: $\mathcal{X} \approx [\![A, B, C]\!] = \sum_{r=1}^{R} a_r b_r c_r$, where $R > 0$ is the rank of the tensor. Elementwise, $x_{ijk} \approx \sum_{r=1}^{R} a_{ir} b_{jr} c_{kr}$.

The rank of a tensor is defined as the smallest number of rank one tensors that generate it as their sum.

Example in 3-dimensional tensor:
Define $\mathcal{X}$ in frontal slice matrices as $\mathcal{X}[:,:,1] = X_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ and $\mathcal{X}[:,:,2] = X_2 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$.
The CP decomposition of $\mathcal{X}$ is $\mathcal{X} = [\![A, B, C]\!]$,
where $A = \begin{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 1 \end{bmatrix} & \begin{bmatrix} 1 \\ -1 \end{bmatrix} \end{bmatrix}$, $B = \begin{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 1 \end{bmatrix} & \begin{bmatrix} 1 \\ 1 \end{bmatrix} \end{bmatrix}$, $C = \begin{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} & \begin{bmatrix} 1 \\ 1 \end{bmatrix} & \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{bmatrix}$.
The main limitation of the CDP decomposition is the need of an iterative method to compute it, due to the fact there is no finite algorithm that determines the rank of a tensor. Hence, there is no way to know in advance how many rank one tensors are needed to have a good enough approximation. The only known method uses the convergence to the original tensor $\text{Min}_{\widehat{\mathcal{X}}} ||\mathcal{X} - \widehat{\mathcal{X}}||$ such that $\widehat{\mathcal{X}} = \sum_{r=1}^{R} \lambda_r . a_r \otimes b_r \otimes c_r$, where λ is a vector containing weight

parameters.

Tucker decomposition

The Tucker decomposition was introduced for the first time in 1963 by Tucker in [32]. It breaks up a tensor into a smaller tensor of same dimensions, multiplied by a matrix along each mode. As such, a 3D tensor $\mathcal{X} \in \mathbb{C}^{I \times J \times K}$ decomposes into

$$\mathcal{X} \approx \mathcal{G} \times_1 A \times_2 B \times_3 C = \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{pqr} a_p \otimes b_q \otimes c_r$$

where $A \in \mathbb{C}^{I \times P}$, $B \in \mathbb{C}^{J \times Q}$, $C \in \mathbb{C}^{K \times R}$ and $\mathcal{G} \in \mathbb{C}^{P \times Q \times R}$.
Elementwise, the Tucker decomposition for 3-way tensors is

$$x_{ijk} \approx \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{pqr} a_{ip} b_{jq} c_{kr} \quad \text{for } i = 1, ..., I, j = 1, ..., J, k = 1, ..., K$$

By allowing different ranks for the decomposition at each mode, the Tucker decomposition method offers significantly more flexibility than the CDP decomposition. This flexibility makes it a preferred tool for modeling multi-way data with a relatively small number of components for each mode. Additionally, the Tucker decomposition is less affected by noise compared to CDP decomposition. However, this method is considerably more computationally expensive—in terms of both complexity and memory—making it feasible only for small-sized tensors, as it involves computing the core tensor and factor matrices.

2.2.2 The quantized tensor train decomposition and its properties

Among various matrix product and lower mode decompositions, the tensor train (TT) decomposition exhibits particularly useful properties for multi-level matrix-vector multiplication and multi-level Fourier transforms (section 2.2.3). Therefore, it is valuable to compare this method with the standard circulant embedding approach in the context of electromagnetic field simulations.
In this section we will define the tensor train decomposition in the general case and explore its relevant properties for matrix-vector multiplication. Then, we will develop the complete procedure to obtain the TT decomposition of a 3-mode tensor before defining the specific case of quantized tensor train decomposition.

Tensor train decomposition

Let $\mathcal{T}$ be a d-mode tensor, $\mathcal{T} \in \mathbb{C}^{I_1 \times ... \times I_d}$. This tensor decomposes in the TT format into d matrices or tensors $\mathcal{Q}^{(n)} \in \mathbb{C}^{r_{n-1} \times I_n \times r_n}$. Elementwise,

$$t_{i_{I_1},...,i_{I_d}} = \sum_{s_1=1}^{r_1} ... \sum_{s_{d-1}=1}^{r_{d-1}} q^{(1)}_{1,i_1,s_1} q^{(2)}_{s_1,i_2,s_2} ... q^{(d)}_{s_{d-1},i_d,1}$$

where $\{r_n\}_{n=0,...,d}$ are the compression ranks with $r_0 = r_d = 1$. This can be written more compactly as a matrix product:

$$\mathcal{T} = \mathcal{G}_1[i_1]...\mathcal{G}_d[i_d]$$

When $I_1 = ... = I_d = n$ and for $r = \text{Max}_i\{r_i\}$ the maximal TT-rank of $\mathcal{T}$, this format allows one to store n^d elements using $O(dnr^2)$ of memory unit.
Example in 3D: Consider the 3-way tensor $\mathcal{T} \in \mathbb{R}^3 \times \mathbb{R}^4 \times \mathbb{R}^5$ defined as $\mathcal{T}[i_1, i_2, i_3] = i_1 + i_2 + i_3$
Its TT-format is $\mathcal{T}[i_1, i_2, i_3] = \mathcal{G}_1[i_1]\mathcal{G}_2[i_2]\mathcal{G}_3[i_3]$, where

$$\mathcal{G}_1[i_1] = \begin{bmatrix} i_1 & 1 \end{bmatrix}, \quad \mathcal{G}_2[i_2] = \begin{bmatrix} 1 & 0 \\ i_2 & 1 \end{bmatrix} \quad \text{and} \quad \mathcal{G}_3[i_3] = \begin{bmatrix} 1 \\ i_3 \end{bmatrix}$$

One can check that $\mathcal{G}_1[i_1]\mathcal{G}_2[i_2]\mathcal{G}_3[i_3] = i_1 + i_2 + i_3 = \mathcal{T}[i_1, i_2, i_3]$.
The main advantages of the TT decomposition are its low memory cost compared to the cost of a full low rank tensor (dnr^2 versus n^d), and the low cost associated with transformations from regular tensors to the TT-format. Furthermore, one notable property of the TT format is the ability to perform elementary operations like additions and element-wise product directly on its cores without the need to convert them back into full tensors. These incentives strongly favour TT decomposition, compared to more classic formats, for fast multi-level block-Toeplitz matrix-vector multiplication.

Addition in the TT format

Define three d-mode tensors $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ such that $\mathcal{C} = \mathcal{A} + \mathcal{B}$.
Hence, $\mathcal{C}[i_1, ..., i_d] = \mathcal{A}[i_1, ..., i_d] + \mathcal{B}[i_1, ..., i_d]$
Consider $C_k[i_k] = \begin{bmatrix} A_k[i_k] & 0 \\ 0 & B_k[i_k] \end{bmatrix}$ for $k = 2, ..., d-1$,
$C_1[i_1] = \begin{bmatrix} A_1[i_1] & B_1[i_1] \end{bmatrix}$ and $C_d[i_d] = \begin{bmatrix} A_d[i_d] \\ B_d[i_d] \end{bmatrix}$.

As such, $\prod_{k=1}^{d} C_k[i_k] = \mathcal{A}[i_1, ..., i_d] + \mathcal{B}[i_1, ..., i_d] = \mathcal{C}[i_1, ..., i_d]$, so the $\{C_k\}$s are the TT cores of $\mathcal{C}$ and $\text{rank}(\mathcal{C}) = \text{rank}(\mathcal{A}) + \text{rank}(\mathcal{B})$. In essence, the addition in TT-format is only the independent sum of the TT-cores of each tensor. The complexity of this operation is then $O(dn(\text{rank}(\mathcal{A}) + \text{rank}(\mathcal{B}))^2)$. [30]

Hadamard product in the TT format

First, note that—as a matrix product—multiplication by a scalar in the TT format can be achieved by multiplying one of the cores by the scalar. In that case, the tensor rank does not change and its complexity is $O(dr)$. [30]
Now, define three d-mode tensors $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ such that $\mathcal{C} = \mathcal{A} \odot \mathcal{B}$, where $\odot$ stands for the Hadamard product, i.e the element-wise product: $A \odot B = (a_{i,j}b_{i,j})_{i,j}$.
$\mathcal{C}[i_1, ..., i_d] = \mathcal{A}[i_1, ..., i_d] \odot \mathcal{B}[i_1, ..., i_d]$
Define $C_k[i_k] = A_k[i_k] \otimes B_k[i_k]$, where $\otimes$ is the Kronecker product. As such, one can check that $\prod_{k=1}^{d} C_k[i_k] = \mathcal{A}[i_1, ..., i_d] \odot \mathcal{B}[i_1, ..., i_d] = \mathcal{C}[i_1, ..., i_d]$. Therefore, the $\{C_k\}$s are the TT cores of $\mathcal{C}$ and $\text{rank}(\mathcal{C}) = \text{rank}(\mathcal{A})\text{rank}(\mathcal{B})$. Thus, the Hadamard product is performed by grouping the cores corresponding to the same TT-ranks together. The complexity of the Hadamard product in TT format is then $O(dn\text{rank}(\mathcal{A})^2\text{rank}(\mathcal{B})^2)$ [30].

Tensor train decomposition of a 3-mode tensor

Consider a 3-mode tensor $\mathcal{T}$ of size $n_1 \times n_2 \times n_3$. To perform tensor train decomposition, begin by unfolding the tensor at mode 1, i.e. if $T_1, T_2, ..., T_{n_3}$ are its $n_1 \times n_2$ third dimensional cut matrices, then the unfolded tensor at mode 1 is the $n_1 \times n_2 n_3$ matrix:

$$\mathcal{T}_{(1)} = \begin{bmatrix} T_1 & T_2 & ... & T_{n_3} \end{bmatrix}$$

Compute the rank r_1 Singular Value Decomposition (SVD) approximation of $\mathcal{T}_{(1)}$ to obtain $\mathcal{T}_{(1)} \approx U_1 \Sigma_1 V_1^\dagger$, with U_1 an $n_1 \times r_1$ matrix and $\Sigma_1 V_1^\dagger$ an $r_1 \times n_2 n_3$ matrix. Reshape $\Sigma_1 V_1^\dagger$ matrix into a matrix R of size $r_1 n_2 \times n_3$ and compute its rank r_2 SVD approximate: $R \approx U_2 \Sigma_2 V_2^\dagger$, with U_2 a matrix of size $r_1 n_2 \times r_2$ and $\Sigma_2 V_2^\dagger$ a matrix of size $r_2 \times n_3$. The TT cores of $\mathcal{T}$ are then defined as

$$\mathcal{G}_1 = \text{reshape}(U_1, (1, n_1, r_1))$$

$$\mathcal{G}_2 = \text{reshape}(U_2, (r_1, n_2, r_2))$$

$$\mathcal{G}_3 = \text{reshape}(\Sigma_2 V_2^\dagger, (r_2, n_3, 1))$$

where the reshape function reorder every coefficient of its first argument matrix and replace them in an array whose size is given by the second argument tuple. The filling of coefficients follows the order of the modes (primarily along each line, then each column, third dimension, and so on...). As can be verified directly, $\mathcal{T} = \mathcal{G}_1 \times \mathcal{G}_2 \times \mathcal{G}_3$—the product between tensors being the product of each of the layer matrices, perpendicular to the third dimension. That is

$$\mathcal{T}(i_1, i_2, i_3) = \sum_{\alpha_2=1}^{r_2} \sum_{\alpha_1=1}^{r_1} G_3(i_3, \alpha_2, 1) G_2(\alpha_2, i_2, \alpha_1) G_1(1, \alpha_1, i_1),$$

for all $i_1 = 1, ..., n_1$; $i_2 = 1, ..., n_2$; $i_3 = 1, ..., n_3$.

Intuitively, tensor train decomposition re-writes the tensor into a smaller (optimised) basis, defined multiplicatively, via SVD approximation along each of its mode.

Quantized tensor train decomposition

When working with the FFT, it is often preferable to choose powers of 2 as vector and matrix sizes in order to maximize the algorithm's performance. Thus, we consider $n = 2^l$, $l > 0$.
It is possible to further break down a tensor along each binary dimension. Consider a d-mode tensor $\mathcal{T}$ of size $n_1 \times ... \times n_d$ where $n_k = 2^{l_k}$ for all $k = 1, ..., d$. Prior to performing tensor train decomposition, reshape $\mathcal{T}$ into a $\prod_{k=1}^{d} l_k$-dimensional array of size $2 \times 2 \times ... \times 2$ by considering the binary coding of each index:

$$i_k = \sum_{\kappa=1}^{l_k} 2^{l_k - \kappa_k} i_{\kappa_k}, \quad \text{where} \;\; i_k = 0, ..., n_k - 1 \quad \text{and} \quad i_{\kappa_k} \in \{0, 1\} \;\; \text{for all} \;\; \kappa_k = 1, ..., l_k$$

Thus, the tensor train decomposition can perform SVDs on each of the modes of the new $2 \times 2 \times ... \times 2$ tensor, giving $\prod_{k=1}^{d} l_k$ QTT-cores to the initial d-dimensional array. This operation that further breaks down a tensor along each binary dimension is known as "quantization".
For instance, consider the 4×4 matrix A

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

By rewriting the indices of its coefficients, we can reshape A into the 4-modes tensor of size $2 \times 2 \times 2 \times 2$ $\mathcal{A}$ (since $l_1 l_2 = 2 \times 2 = 4$).

$$\mathcal{A} = \left[\begin{bmatrix} a_{001,001} & a_{001,010} \\ a_{010,001} & a_{010,010} \\ a_{011,001} & a_{011,010} \\ a_{100,001} & a_{100,010} \end{bmatrix} \begin{bmatrix} a_{001,011} & a_{001,100} \\ a_{010,011} & a_{010,100} \\ a_{011,011} & a_{011,100} \\ a_{100,011} & a_{100,100} \end{bmatrix} \right]$$

One can then compute the TT decomposition on the new $\prod_{k=1}^{d} l_k$-mode tensor to get its $\prod_{k=1}^{d} l_k$ TT cores. This method is refered to as the Quantized Tensor Train (QTT) decomposition. Kazeev et al. give a detailed procedure on how to easily perform the QTT decomposition on Toeplitz and circulant matrices [31]. Consider a one-level Toeplitz matrix T of size $n \times n$ generated by the vector $\mathbf{x}$ of size $2n$, such that:

$$T = \begin{bmatrix} \mathbf{x}_{n+1} & \mathbf{x}_n & \dots & \mathbf{x}_3 & \mathbf{x}_2 \\ \mathbf{x}_{n+2} & \ddots & \ddots & \ddots & \mathbf{x}_3 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \mathbf{x}_{2n-1} & \ddots & \ddots & \ddots & \mathbf{x}_n \\ \mathbf{x}_{2n} & \mathbf{x}_{2n-1} & \dots & \mathbf{x}_{n+2} & \mathbf{x}_{n+1} \end{bmatrix},$$

where $\mathbf{x}_1$ is an arbitrary value, to make the number of parameters even.
For a given multi-dimensional vector $\mathbf{y}$, the method developed by Kazeev et al. takes the QTT decomposition of the outer product $\mathbf{x}\mathbf{y}^\dagger$ and returns the QTT-cores of the result of the $T\mathbf{y}$ multiplication.
The exact computation of the matrix-vector multiplication becomes costly as the size of the matrices and vectors increases. Therefore, Kazeev et al. use an iterative method of approximation with rank truncation known as "Density Matrix Renormalization Group (DMRG)" and presented by Oseledets in 2011 [33].

2.2.3 Performance and limitations of the method

In their paper, Kazeev et al. compare the computational time of this method to the full matrix FFT method (they do not specify if this full matrix method includes a circulant embedding of the Toeplitz matrix) and a QTT format with the Fourier transform applied directly on the cores. The method used to perform a Fourier transform on QTT cores is explained in [34]. Here is a quick overview of that method.
Define a d-dimensional vector $\mathbf{x}$ with QTT cores $X_{k_1}, ..., X_{k_d}$, where the $\{k_i\}$s are the binary

expansion of the dimension k for all $k = 1, ..., d$. Then, define the 2^d-long vector

$$\mathbf{w}_d^T = \begin{bmatrix} 1 & ... & 1 & 1 & \omega_d & \omega_d^2 & ... & \omega_d^{2^{d-1}-1} \end{bmatrix}$$

To compute the Fourier transform of the k^{th} dimensional matrix, initialize $\mathbf{x}_d = \mathbf{x} = X_{d,k_1}, ..., X_{d,k_d}$, the first index corresponding to the vector and the second to the TT-core. For $D = d, d-1, ..., 1$, take $\tilde{\mathbf{x}}_D = X_{D,k_1}...X_{D,k_{D-1}}\tilde{X}_{D,k_D}$ where $\tilde{X}_{D,0} = X_{D,0} + X_{D,1}$ and $\tilde{X}_{D,0} = X_{D,0} - X_{D,1}$. Then, compute $z_D = w_D \odot \tilde{\mathbf{x}}_D$ directly on the TT-cores using the property of Hadamard product in TT format. Set $\mathbf{x}_{D-1} = z_D = Z^{(D)}_{D,k_1}...Z^{(D)}_{D,k_D}$. The Fourier transform of $\mathbf{x}$ along the k^{th} dimension is then given by the TT-cores $Y^{(p)_{j_p}} = \left(Z^{(d-p+1)}_{j_p}\right)^T$ for $p = 1, ..., d$, which is simply the bit reversed version of the output.

Simulation results

The results from the simulation conducted by Kazeev et al. [31] match the expectation from the complexity of the tensor train transformations. Indeed, the QTT decomposition method presented only outperforms the full matrix method by Fourier transform when the length of the vector is sufficiently long compared to its rank. Thus, the crossing point of the measured computational time between the two methods in one dimension is located at $n = 2^{16}$ for $r = 5$, $n = 2^{18}$ for $r = 15$ and $n = 2^{22}$ for $r = 40$. These crossings correspond to a "rank ratio" $\frac{r}{n}$ of 7.6×10^{-5}, 5.7×10^{-5} and 9.5×10^{-6} respectively, which corresponds to highly correlated data.
Kazeev et al. specify that from $r = 10$, the computation of the exact result becomes too expensive and should encourage the DMRG approximated methods. This can be used in addition with truncation techniques [33]. Note that for the first crossing point determination, an exact method is used to compare with the full vector method, while the other two crossing points are obtained via an approximated and truncated rank version of the method.

Theoretical complexity analysis

Let us go through the whole process of the multi-level block Toeplitz matrix-vector product by QTT format decomposition and evaluate its cost in terms of peak memory and complexity in terms of number of needed multiplications.
Consider a d-dimensional $n_k \times n_k$ Toeplitz matrix, with $n_k = 2^{l_k}$ for each level $k = 1, ..., d$. The generator vector $\mathbf{x}$ of the Toeplitz matrix is extracted without multiplication from the Toeplitz matrix and the same goes for the outer product $\mathbf{x}\mathbf{y}^\dagger$. From there, one would need to convert the obtained matrix into QTT-format in $O(dlr^4)$ multiplications, where r is the maximum

TT-rank of the TT-cores of $\mathbf{x}\mathbf{y}^\dagger$. Denote $r_x = \text{rank}(\mathbf{x})$, $r_y = \text{rank}(\mathbf{y})$ and r_z the maximum of the QTT ranks of the result of the matrix-vector multiplication. Kazeev et al. claim that the complexity of one iteration of the DMRG method is $O\left(dlr_z^3 + dlr_x r_y r_z (r_x + r_y + r_z)\right)$, where $l = \text{Max}_{k=1,\dots,d} l_k$. Finally, we can conclude that the total complexity of the method is in $O(\alpha dlr^3(3r+1) + dlR^4)$, where α is some proportionality factor that depends on the number of iteration of the DMRG method before convergence, $r = \text{Max}\{r_x, r_y, r_z\}$ and R is the maximum TT-rank of the outer product $\mathbf{x}\mathbf{y}^\dagger$. Memorywise, the peak memory is achieved right before the convolution $\mathbf{x} * \mathbf{y}$, where the variables stored are the vectors $\mathbf{x}$ and $\mathbf{y}$ and the tensor $\mathbf{S}$. Therefore, the total peak memory value is $dl(r_x^2 + r_y^2 + 4)$ as $\text{rank}(\mathbf{S}) = 2$.

Complexity and memory storage comparison with the circulant embedding method

In comparison, the circulant embedding method in Fourier space—which is the standard method for high rank multi-level block Toeplitz matrix vector multiplication—performs the matrix-vector product in $C_{\text{embed}} = 2^{d+1} dn^d \log_2(2n) + (2n)^d$ elementary multiplications (detailed computation in section 3). We will compare that method with one iteration of the DMRG method.
Hence, if $n_k = n$ for all $k = 1, ..., d$, and for $R = \text{Max}\{r, r_x, r_y, r_z\}$,

$$C_{\text{embed}} \leq C_{\text{QTT}}$$

$\Leftrightarrow$

$$2^{d+1} dn^d \log 2(2n) + 2^d n^d \leq dlr_z^3 + dlr_x r_y r_z (r_x + r_y + r_z)$$

$\Leftrightarrow$

$$2^{d(l+1)} \left(2\frac{l+1}{l} + \frac{1}{dl}\right) \leq r_z^3 + r_x r_y r_z (r_x + r_y + r_z) \leq r^3(3r+1)$$

For a laptop scale simulation with for instance $d = 3$ and $l = 8$, $2^{d(l+1)} \left(2\frac{l+1}{l} + \frac{1}{dl}\right) = 3.08 \times 10^8$. Therefore, $C_{\text{embed}} = C_{\text{QTT}}$ for $r \approx 100$ with $n = 2^8 = 256$.
Memorywise, the storage required for the circulant embedding method is $M_{\text{embed}} = (2n)^d = 2^{d(l+1)}$. On the other hand, $M_{\text{QTT}} = dl(r_x^2 + r_y^2) \leq 2dlr^2$. Hence, in the same laptop scale simulation, $M_{\text{embed}} = 1.34 \times 10^8$ such that $M_{\text{embed}} \leq M_{\text{QTT}}$ for $r \geq 1672$.

2.2.4 Conclusion on the tensor decomposition method for multi-level Toeplitz-vector multiplication

The tensor train decomposition method provides a significant reduction in memory consumption for multi-level matrices and vectors, assuming sufficiently low TT-ranks. Among

the options described above, the tensor train format has the benefit of being stable by basic operations like tensor addition and Hadamard product, while also reducing the size of each elementary operations by acting on the smaller TT cores of the tensor. The quantized tensor train decomposition offers an even finer decomposition for sizes that can be expressed in powers of 2. Moreover, solutions exist to perform efficient Toeplitz matrix-vector multiplication via a QTT convolution of the outer product of the generating Toeplitz vector and the source vector. However, the QTT method has two major limitations compared to the circulant embedding one in the context of solving wave equation by successive iterations. First, the initial SVD used in the QTT decomposition is computationally intensive, akin to performing several matrix-vector multiplications, rendering the method impractical without prior knowledge of the data. However, for highly structured input vector data, where sufficient information is available to approximate the TT cores of the outer product between the generator vector and the input vector without explicit computation, this method can be highly efficient. Then, the crossing point between the time complexity of the QTT decomposition technique and the embedding method—determined by Kazeev et al. [31]—is located at large matrix-vector size (near the limit of laptop-scale simulations in three dimensions: $n = 2^{22}$) and for very low tensor rank ($r \geq 40$). Above $r = 10$, the exact computation method becomes too expensive and requires more advanced rank truncation and iterative resolution by approximation techniques. For our case of the electromagnetic Green's function the input vector is a polarization current which can have a low rank under certain conditions. These conditions could be explored in future work. Second, each multiplication in QTT format increases the tensor rank of the result of the multiplication. Knowing that in any iterative solution method matrix-vector products would need to be performed successively (see Section 3.1), the use of the QTT format would seemingly lead to increasing the tensor rank (until convergence). The potential gains of the method would hence be limited by the fact that the memory needed to store a vector in QTT format increases quadraticly with tensor rank, and that the time complexity of reconstruction scales as a 4-order polynomial. These limitations naively impose strong limits on potential applications. Particularly, in the context of iterative resolution of wave propagation equations, the target vector would likely need to be highly structured in order for tensor decomposition methods to offer any benefit.

CHAPTER 3 DISCUSSION OF THE RESEARCH PROJECT

The present chapter illustrates why the acceleration of multi-level block Toeplitz matrix-vector multiplication is important for modelling physical systems. This direct connection to engineering applications rests as the core motivation for conceiving the algorithm described in chapter 4. It begins by exploring the physical origins of block Toeplitz data structures within translationally invariant physics (section 3.1,1). Next, the formalism of scattering theory will be developed to derive the equations we aim to solve using iterative methods (section 3.1.2). Subsequently, we will present the specific equations related to electromagnetism within the same scattering theory framework (section 3.1.3). Finally, we will derive the objectives that the algorithm must achieve based on the conditions established by the physical context (section 3.2).

3.1 Physical context of the block Toeplitz matrix-vector multiplication in electromagnetism

3.1.1 Translational invariance and block Toeplitz structure.

Translational invariance is a core pillar of physics: for a theory to be sensible it must not depend on the distance to an arbitrarily selected point in a homogeneous space. Conceptually, a quantity is invariant under translation if the laws and equations that govern the quantity are the same at all points in space. As such, any translationally invariant quantity $\mathcal{Q}$ can be written as $\mathcal{Q}(\mathbf{r}_1 - \mathbf{r}_2)$, such that

$$\mathcal{Q}\left((\mathbf{r}_1 + \mathbf{a}) - (\mathbf{r}_2 + \mathbf{a})\right) = \mathcal{Q}(\mathbf{r}_1 - \mathbf{r}_2).$$

On a cell grid, this property manifests itself in a consistent relationship between the values of two adjacent cells along any given axis, which is identical for all other pairs of adjacent cells along the same axis. Similarly, the relationship between two cells separated by c cells is the same as that between any other two cells separated by the same distance along the same axis with respect to the field $\mathcal{Q}$. Consequently, for a one-dimensional cell line grid, if the values of the field $\mathcal{Q}$ is approximated as a matrix G, with the coefficient $G_{i,j}$ representing the influence of cell i on cell j concerning the field $\mathcal{Q}$, the resulting matrix will resemble the top half of Figure 3.1.

In this kind of matrix, the diagonal coefficients $G_{i,i}$ represent the self-interactions (0-step) for

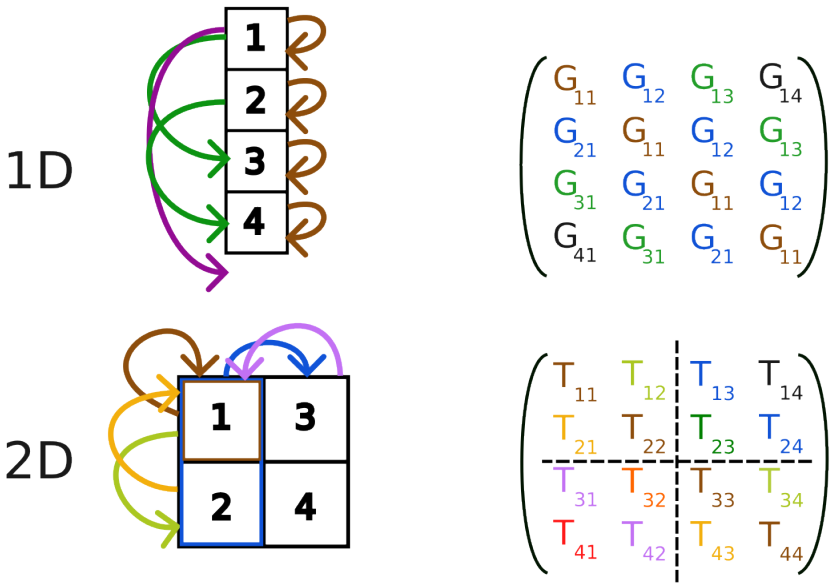

Figure 3.1 Translational invariance and block Toeplitz structure.

each cell of the line towards the quantity $\mathcal{Q}$. These interactions are labeled in brown in Figure 3.1. The 1-step interactions up and down, labeled in blue, correspond to the coefficients on the upper and lower diagonal and similarly for any of k-step interactions corresponding to the k^{th}.

In a two dimensional grid, each column—with the cells inside interacting with each other as described above—has the same type of interaction with the other columns. On the bottom half of Figure 3.1, the blue column has self interactions labeled in brown which corresponds to the top left block of coefficients and is equal to the second column self interactions coefficients, located at the bottom right block. The interactions between the columns (labeled in blue and pink) are represented by the two remaining blocks, which are not necessarily equal for non symmetric phenomena. Thus, any matrix storing the cell interactions of the translationally invariant quantity $\mathcal{Q}$ in a two dimensional grid displays a block-Toeplitz structure of $n_y \times n_y$ blocks of $n_x \times n_x$ matrices, n_x and n_y being the length of the cell along the x and y axis.

A three dimensional grid would then add another level to the data structure, each of the block-Toeplitz being an element contained in the total matrix and forming itself a Toeplitz structure due to the interactions between each layer of the third dimension.

3.1.2 Application in scattering theory

The context from which we approached the block Toeplitz matrix-vector multiplication problem has a strong connection with scattering theory. Its more general definition is related to the studies of the behaviour of waves, fields or particles near localized non-uniformities. This set of non-uniformities is denoted as the scattering center or scatterer. The medium of the incident wave, before the scatterer, and the scattered wave, after the scatterer, is considered homogeneous and linear [35]. i, s and t subscripts are used to denote incident, scattered, and total quantities respectively.

For any free space physics, solutions can be derived explicitly by the means of solving a PDE. As such, the scattering theory approach is to define a scattering term $\mathbf{X}$ that represents some part of the physics that we are not able to solve explicitly but that can be expressed easily. This approach is specifically appealing when the matrix-vector multiplication with $\mathbf{X}$ is easy to compute. Let $\mathbf{M}_0$ denote the operator of the linear system of equation in free space physics, easy to solve, i.e invertible. Let $\mathbf{f}_t$ be the combined solution of the free space and scatterer space physics and $\mathbf{p}_i$ be some source vector. The wave equation in free space is hence given by:

$$(\mathbf{M}_0)\,\mathbf{f}_i = \frac{i}{k_0}\mathbf{p}_i.$$

In accordance with the assumption of the scattering theory, the wave equation in the scattered medium is:

$$(\mathbf{M}_0 - \mathbf{X})\,\mathbf{f}_t = \frac{i}{k_0}\mathbf{p}_i. \tag{3.1}$$

Take $\mathbf{G}_0$ to denote the inverse matrix of the free space equation matrix $\mathbf{M}_0$. Combining the definition $\mathbf{f}_t = \mathbf{f}_i + \mathbf{f}_s$ with the free space wave equation,

$$\mathbf{f}_s = \mathbf{G}_0\mathbf{X}\mathbf{f}_t.$$

Therefore,

$$\mathbf{f}_t = \mathbf{f}_i + \mathbf{f}_s,$$

$$\text{and } \; (\mathbf{Id} - \mathbf{G}_0\mathbf{X})\,\mathbf{f}_t = \mathbf{f}_i.$$

Let $\mathbf{f}_0$ to denote a possible free solution entering though the boundary of the computational domain such that $\mathbf{f}_i = \mathbf{f}_0 + \mathbf{G}_0\mathbf{p}_i$, and $\mathbf{p}_0 = -\mathbf{X}\mathbf{f}_0$. Hence, with (3.2), we get

$$(\mathbf{Id} - \mathbf{X}\mathbf{G}_0)\,\mathbf{p}_s = -\mathbf{X}\mathbf{f}_0 + \mathbf{X}\mathbf{G}_0\mathbf{p}_i$$

$$(\mathbf{Id} - \mathbf{XG}_0)\,\mathbf{p}_t = \mathbf{p}_0 + \mathbf{p}_i$$

Now that the equation of the induced densities is set, the question of the invertibility of the operator $(\mathbf{Id} - \mathbf{XG}_0)$ should be asked.
Any linear operator $\mathbf{A}$ in a Hilbert space $\mathcal{H}$ can be decomposed into a Hermitian and skew-Hermitian component $\mathbf{A} = \mathbf{A}^s + i\mathbf{A}^a$ where

$$\mathbf{A}^s = \frac{\mathbf{A} + \mathbf{A}^\dagger}{2}, \qquad \mathbf{A}^a = \frac{\mathbf{A} - \mathbf{A}^\dagger}{2i}$$

Then, we know that $\mathbf{A}^a$ is either positive definite or negative definite so its kernel is empty, making $\mathbf{A}^a$ invertible. Furthermore, the anti-symmetric component of $\mathbf{X}$ is positive definite due to the properties of the dissipation of power inside the scatterer [36]. Thus, the anti-symmetric component of $\mathbf{X}^{-1} - \mathbf{G}_0$ is positive definite, within the scatterer, so the full operator is invertible within the scatterer. Finally, as by [36] $\mathbf{X}$ is invertible inside the scatterer, $\mathbf{Id} - \mathbf{XG}_0$ is invertible. As such, solving for the induced quantities (combination of the incident and scattered) reduces to the matrix-vector product $(\mathbf{Id} - \mathbf{XG}_0)^{-1}(\mathbf{p}_0 + \mathbf{p}_i)$. Thus, to solve for $\mathbf{p}_t$ one can either go through direct inversion or by substitution, which in general becomes prohibitively expensive for large systems. Alternatively, the evaluation of $(\mathbf{Id} - \mathbf{XG}_0)^{-1}$ can be done via iterative methods which effectively mimic the description of the inverse given by power series expansion:

$$(\mathbf{Id} - \mathbf{XG}_0)^{-1} \approx \mathbf{Id} + \mathbf{XG}_0 + \mathbf{XG}_0\mathbf{XG}_0 + ...$$

Therefore, being able to perform the matrix-vector product of $\mathbf{XG}_0$ together with $\mathbf{p}_0 + \mathbf{p}_i$ as efficiently as possible is the most crucial aspect of trying to optimize the computation of such simulations. Moreover, the scattering problem to solve is direct, which means that we are trying to determine the properties of the scattered wave based on the incident wave and the properties of the scatterer.

3.1.3 Physical model in electromagnetics

Consider the medium outside of the scatterer to be free space and we only add a scattering operator $\mathbf{X}$ term—depending on X_ϵ and X_μ the linear relative permittivity and permeability respectively—inside the scatterer. Thus, the quantities considered through the prism of scattering theory are bare currents, those induced by scattering and the combination of the two. Thus,

$$\frac{i}{k_0}\begin{bmatrix}\mathbf{j_s}\\ \mathbf{m_s}\end{bmatrix} = \begin{bmatrix}z^{-1}\mathbf{X_{je}} & z\mathbf{X_{jh}}\\ z^{-1}\mathbf{X_{me}} & z\mathbf{X_{mh}}\end{bmatrix}\begin{bmatrix}\mathbf{e_t}\\ \mathbf{h_t}\end{bmatrix}$$

$$\frac{i}{k_0}\mathbf{p_s} = \mathbf{X}\mathbf{f_t} \tag{3.2}$$

where k_0 is the free space wave vector component $2\pi/\lambda$. The electromagnetic field and polarization current densities are treated as six component vectors $\mathbf{f} = \{\mathbf{e}, \mathbf{h}\}$ and $\mathbf{p} = \{\mathbf{j}, \mathbf{m}\}$ and z is the impedance of free space $\sqrt{\mu_0/\epsilon_0}$. It defines the bound polarization for current densities induced by the total electromagnetic field, induced and scattered.

Now, starting from the assumptions of the scattering theory, let's derive the block Toeplitz equation from the Maxwell equations. As a reminder, the local Maxwell's equations in free space for the induced current and magnetic density are:

$$\nabla_c \mathbf{h}_i - \frac{\partial}{\partial t}\mathbf{d}_i = \mathbf{j}_i$$

$$\nabla_c \mathbf{e}_i + \frac{\partial}{\partial t}\mathbf{b}_i = -\mathbf{m}_i$$

where ∇_c is the curl operator. Hence,

$$\nabla_c \mathbf{h}_i + \frac{i}{k_0}\mathbf{Z}^{-1}\mathbf{e}_i = \mathbf{j}_i$$

$$\nabla_c \mathbf{e}_i - ik_0\mathbf{Z}\mathbf{h}_i = -\mathbf{m}_i$$

Therefore,

$$-\mathbf{Z}^{-1}\mathbf{e}_i + \frac{i}{k_0}\nabla_c \mathbf{h}_i = \frac{i}{k_0}\mathbf{j}_i$$

$$-\frac{i}{k_0}\nabla_c \mathbf{e}_i - \mathbf{Z}\mathbf{h}_i = \frac{i}{k_0}\mathbf{m}_i$$

which can be written as

$$\mathbf{M}_0\mathbf{f}_i = \frac{i}{k_0}\mathbf{p}_i$$

where $\mathbf{M}_0 = -\begin{bmatrix} \mathbf{Z}^{-1} & -\frac{i}{k_0}\nabla_c \\ \frac{i}{k_0}\nabla_c & \mathbf{Z} \end{bmatrix}$ is the vacuum Maxwell operator [37]. Hence, the $\mathbf{G}_0$ matrix introduced in the previous subsection is then the vacuum Green's function operator.
Note that due to the local property within the scatterer, the scattering term $\mathbf{X}$ is diagonal in the context of electromagnetism. Indeed, the local parameters of the scatterer at cell c_1 do not affect those at cell c_2, regardless of their proximity. Consequently, the scattering term $\mathbf{X}$, which represents the influence of the medium's properties on the free space wave propagation equation, is confined to the diagonal coefficients, only corresponding to self-interacting cell

terms. This localization simplifies the computation of the $\mathbf{XG}_0$ matrix multiplication for electromagnetic wave simulations.

3.1.4 Green's functions and finite differences

This section provides a comparative analysis of two approaches for solving wave equations in electromagnetism: the Green's function method and the finite difference method applied to the standard Maxwell partial differential equations (PDEs). We will evaluate the range of problems each method can address, examine the structure and sparsity of their respective data representations, and ultimately assess their computational complexity.

The Green's function method

Consider the problem:

$$\begin{cases} \mathcal{L}\mathbf{u} = \mathbf{f} \\ \mathcal{D}\mathbf{u} = 0 \end{cases}$$

where $\mathcal{L}$ and $\mathcal{D}$ are linear differential operators for the wave equation and the boundary conditions respectively.
Note that in the case of electromagnetic wave propagation equations, $\mathbf{f}$ is a source vector and $\mathcal{L} = \nabla^2 + k$.
The Green's function G is defined such that $\mathcal{L}G = \delta$ where δ is the Dirac's delta function. The unique solution for the wave equation and the boundary conditions in 1D is then of the form $u(x) = \int_0^l f(s)G(x,s)ds$. Therefore, the Green's function is uniquely determined through the set of wave equations and boundary conditions. Unlike the method consisting in solving the PDE, the Green's function method does not require local variation of the equations at the boundary of the computational domain.

Example in 1D
Consider the electromagnetic wave equation in 1D with Neumann type boundary conditions:

$$\begin{cases} (\nabla^2 + k)u(x) = f(x) \\ u(0) = 0 \\ u(l) = 0 \end{cases}$$

Hence, we solve for the Green's function $\frac{\partial^2}{\partial x^2}G + k^2 G = \delta(x - \xi)$ with the same boundary conditions $\frac{\partial}{\partial x}G(0) = 0$ and $\frac{\partial}{\partial x}G(l) = 0$. To be able to respect the boundary condition at

$x = 0$ and $x = l$ and the wave equation, set $G = \begin{cases} A\cos(kx) & \text{for } x < \xi \\ B\cos(k(x-l)) & \text{for } x > \xi \end{cases}$
Finally, the two boundary conditions uniquely determine the constants A and B, on the left and on the right of the forcing ξ respectively:

$$G(x,\xi) = \begin{cases} \frac{\cos(kx)cos(k(\xi-l))}{k\sin(kl)} & \text{for } x < \xi \\ \frac{\cos(k(x-l))\cos(k\xi)}{k\sin(kl)} & \text{for } x > \xi \end{cases}$$

The solution of the problem is then given by $u(x) = \int_0^l G(x,\xi) f(\xi) d\xi$.

Data structure: sparsity and translational invariance

As stated in section 3.1.1, the translational invariance of the physics generates a block Toeplitz data structure as soon as the cell grid remains homogeneous, i.e each cell interact with the same number of cells in each direction. Indeed, Fig. 3.2 shows that a non-homogeneous grid breaks the block Toeplitz structure.

The translational invariance of the physical system encompasses both the governing wave equation and its boundary conditions. Specifically, boundary conditions such as Dirichlet (which set the wave amplitude to zero at the boundary) or Neumann (which set the derivative of the wave amplitude to zero at the boundary) alter the equation at the boundary cells of the computational domain, thereby disrupting the translational invariance at coefficients corresponding to these boundary cells. To preserve the block Toeplitz structure of the Green's function operator and fully exploit its properties, it is necessary to only apply boundary conditions that maintain this structure. This can be achieved by using either outgoing boundary conditions at infinity (e.g., Sommerfeld radiation condition or the Bayliss-Gunzburger-Turkel condition [38]), periodic boundary conditions [39], or a combination of both.

By limiting to problems that exhibit translational invariance across the entire computational domain, the Green's function approach results in a dense matrix but with block Toeplitz data structure. In contrast, the finite difference method yields a sparse matrix where nonzero coefficients are limited to a few diagonals around the main diagonal, reflecting interactions confined to neighboring cells. However, the finite difference operator does not possess any particular data structure, and thus, the inherent properties of the data cannot be leveraged to optimize matrix-vector multiplications.

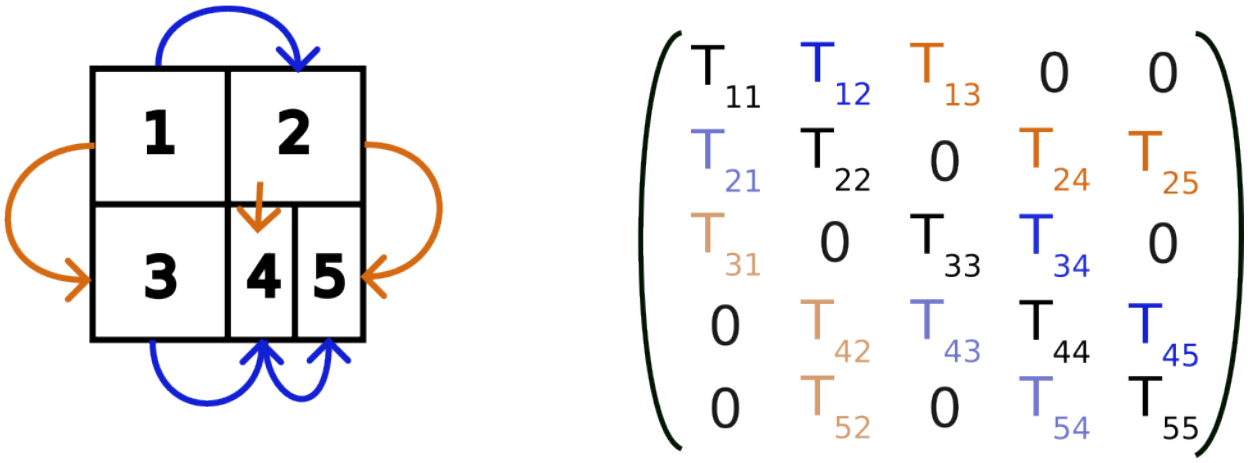

Figure 3.2 Representation of a non-homogeneous cell grid in translational invariant physics with its associated matrix of cell interactions. One step horizontal interactions are represented in dark blue from left to right and light blue in the other direction. One step vertical interactions are represented in orange from top to down and light orange in the other direction.

Complexity analysis in 3D

Consider a 3-dimensional computational grid of size $L \times L \times L$ with $L \geq 5$. In the first method, we consider the standard wave propagation equation with some wall type boundary condition (Dirichlet or Neumann) at the edge of the domain : $(\nabla^2 + k)\mathbf{u} = \mathbf{f}$. Given that the equation is a second-order differential equation, the finite difference method with second-order approximation requires each cell to interact with its two neighbors along all six directions in three-dimensional space. Indeed,

$$f''(x_0) \approx \frac{-f(x_{-2})/12 + 4/3f(x_{-1}) - 5/2f(x_0) + 4/3f(x_1) - f(x_2)/12}{h_x^2}$$

Hence, each row of the finite element matrix would have 25 nonzero coefficients. The total number of nonzero coefficients is then $25(L')^3$, where $L' > L$ is the total number of cells along one axis of the total computational domain including boundary cells (e.g. PML cells). Since the location of the nonzero coefficients are known, the number of multiplications of the matrix-vector product can be reduced to the number of nonzero coefficients.

On the other hand, the Green's function method with outgoing and/or periodic boundary condition would exhibit a dense 3-level block Toeplitz structure of size $L^3 \times L^3$ with intermediate blocks of size $L^2 \times L^2$ and the most inner block would be of size $L \times L$. The complexity of the matrix-vector product using the split circulant embedding method presented in 4.3 is

then $2(2^3-1)L^3(2\log_2(L)+1)+2^3L^3 = 14L^3(2\log_2(L)+1)+8L^3$
Finally, for the sake of argument, take $L' = L = 2^{10}$. Then, the finite difference method performs one matrix-vector product in 25×2^{10} elementary multiplications whereas the Green's function split circulant embedding method requires 302×2^{10}, approximately 12 times more.

Analysis caveats

Despite the finite difference method's apparent efficiency due to the high sparsity of its cells interactions matrix, the actual time complexity of the two methods are more comparable for several reasons. First, the implementation of open boundary conditions necessitates the addition of an extra "boundary layer" in the computational domain (e.g. PML), which increases the size of the computational domain needed with the finite difference approach. Second, PDE-based methods generally have poorer conditioning compared to an integral equation based methods. For instance, when simulating electromagnetic waves around sharp edges, finite difference methods require a finer grid to capture the non-differentiable solutions near these features, which particularly occurs in nanophotonics. In contrast, integral equation methods tend to smooth out the solutions, thereby reducing the need for higher resolution locally.

Additionally, the Green's function method offers advantages in terms of numerical accuracy. Numerical errors in solving differential equations arise from two sources: discretization of the operator and discretization of space [40]. The Green's function approach can minimize the error from discretizing the operator with minimal additional cost [41]. In contrast, the finite difference method, which relies on derivatives, is more susceptible to numerical errors around sharp boundaries, where it may produce inaccuracies at points of divergence.

3.2 Objectives of the article with respect to the constraints of the physical context

The context presented in the previous section implies successive multiplications of the same multi-level block Toeplitz matrix with the result vector. As such, the multiplication method would have to fit the constraint of non-increasing parameters involved in the complexity or memory storage. Indeed, if either of the two increases at each iteration, a slow convergence method would lead most of the time to exceeded memory capacity or excessive computational time. As such, when considering methods using a special tensor storage format, the method

would need non-increasing complexity parameters. Therefore, we will try to define an algorithm optimizing the circulant embedding method, which has non-increasing complexity or memory. The article presented in this thesis (section 4), presents an algorithm that deals with this problem. Then, the theoretical complexity analysis compares the enhanced circulant embedding method to the standard one. Finally, the ratio of real computational time measurements are displayed for laptop scale matrix-vector products.

CHAPTER 4 ARTICLE 1: A SPLIT FAST FOURIER TRANSFORM ALGORITHM FOR BLOCK TOEPLITZ MATRIX-VECTOR MULTIPLICATION

The article presented in this section was submitted to IEEE under the *Computational Physics* category on June 25, 2024. It was authored by Sean Molesky and Alexandre Siron.

4.1 Abstract

Numeric modeling of electromagnetics and acoustics frequently entails matrix-vector multiplication with block Toeplitz structure. When the corresponding block Toeplitz matrix is not highly sparse, e.g. when considering the electromagnetic Green function in a spatial basis, such calculations are often carried out by performing a multilevel embedding that gives the matrix a fully circulant form. While this transformation allows the associated matrix-vector multiplication to be computed via Fast Fourier Transforms (FFTs) and diagonal multiplication, generally leading to dramatic performance improvements compared to naive multiplication, it also adds unnecessary information that increases memory consumption and reduces computational efficiency. As an improvement, we propose a lazy embedding, eager projection, branching algorithm that, for dimensionality d, asymptotically reduces the number of needed computations $\propto d/\left(2-2^{-d+1}\right)$ and peak memory usage $\propto 2/\left((d+1)2^{-d}+1\right)$, generally, and $\propto \left(2^d+1\right)/\left(d+2\right)$ for a fully symmetric or skew-symmetric systems. The structure of the algorithm suggests several simple approaches for parallelization of large block Toeplitz matrix-vector products across multiple devices and adds flexibility in memory and task management.

4.2 Introduction

Translationally invariant physics, when considered in a spatial basis, leads to matrix representations with (multi-level) block Toeplitz structure, Fig. 4.1. Through scattering theory—wherein a complex physical system is described in terms of its free space properties and "scattering" effects caused by previously unaccounted for interactions [15]—the need to efficiently compute block Toeplitz matrix-vector products accordingly arises in a variety of physical contexts [1]. For example, block Toeplitz matrices appear frequently in simulation and device design problems for electromagnetic [2, 3] and acoustic waves [4], as well as the

analysis of multi-dimensional discrete random processes [5]. More directly, in many of these settings (e.g. the simulation of nanophotonic devices utilizing Green functions [16]) a sort of scattering phenomena is efficiently described by a highly sparse operation (e.g. the added interaction is local in space) and the primary bottle-neck of iterative solution methods is the speed of block Toeplitz matrix-vector multiplication [19]. The standard method of computing block Toeplitz matrix-vector products—motivated a number of perspectives [6, 7]—is to perform a circulant embedding for each level of the Toeplitz structure. That is, the Toeplitz matrix-vector method of "extending the diagonal bands",

$$\underbrace{\begin{pmatrix} t_{11} & t_{12} \\ t_{21} & t_{11} \end{pmatrix}}_{n\times n \text{ Toeplitz}} \rightarrow \underbrace{\left(\begin{array}{cc|cc} t_{11} & t_{12} & s_0 & t_{21} \\ t_{21} & t_{11} & t_{12} & s_0 \\ \hline s_0 & t_{21} & t_{11} & t_{12} \\ t_{12} & s_0 & t_{21} & t_{11} \end{array}\right)}_{2n\times 2n \text{ circulant}},$$

is carried out iteratively. Through this procedure, the full matrix acquires a circulant form, and multiplication with the resulting embedded vector is carried out via the FFT approach to convolution [7]. Notably, each embedding doubles the size of the corresponding Toeplitz level, and hence overall size of the system, by the inclusion of additional zero coefficients: taking d as the number of levels, the embedding results in a vector space that is 2^d times as large. These padding coefficients occupy an increasingly large fraction of the total system information as the dimension (number of levels) of the Toeplitz structure increases [Fig. 4.1], leading to inefficient memory utilization and computation.

Two major categories of alternatives have been previously suggested to this procedure. The first class focuses on how block Toeplitz matrix-vector multiplication happens mechanistically, examining how successive coefficient reorderings may be used to minimize the number of arithmetic operations [8]. These methods, to the best of our understanding, have yet to offer substantial speed improvements over the usual FFT approach for large systems, and can not be simply implemented with optimized functions from standard libraries. The second class focuses on the acceleration of block Toeplitz (or Toeplitz) matrix-vector mulplications through the use of matrix or tensor product decompositions, e.g. via Vandermonde matrices [9], QTT tensor formatting [10], or trigonometric transformation [11]. While extremely powerful in certain settings [10], these approach generally require some degree of approximation, are more computationally expensive than circulant embedding for arbitrary vectors, and rely on specialized code bases. Similarly, proposals to improve upon contemporary FFT algorithms, either under specific conditions [12] or by providing a functional equivalent [13], are, at present, only beneficial for sparse vectors. However, if progress is made in this area,

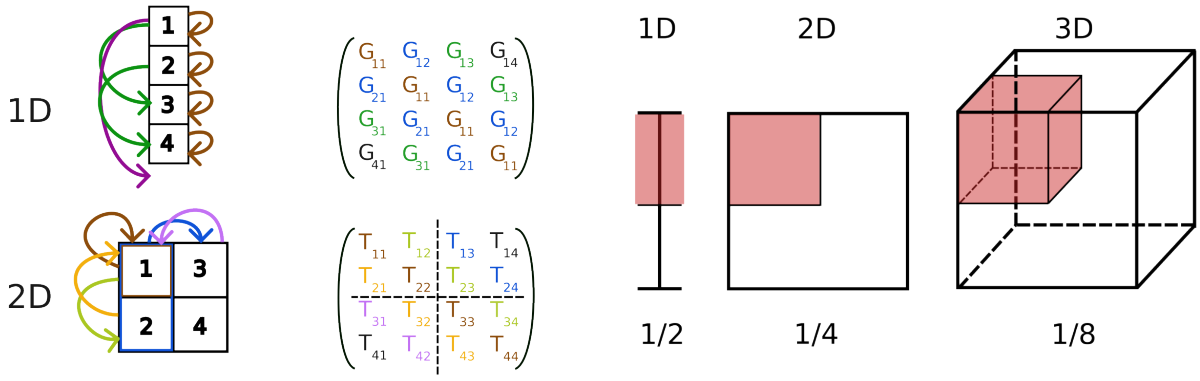

Figure 4.1 Left: **Translational invariance and block Toeplitz structure.** The schematic displays the distribution of interaction coefficients for translationally invariant physics of a pair of simple grids. 0-step (self-interactions) are labeled in brown, vertical 1-step interactions in yellow and light green, horizontal 1-step interactions in blue and pink, 2-step interactions in darker green, and 3-step interactions in purple. Right: **Block Toeplitz embedding and data padding.** The cartoon illustrates the dilution of input vector data in the standard circulant embedding procedure as the dimensionality (level) of block Toeplitz structure increases. Input vector data is is indicated as the coloured portion of each sub-figure.

it should be directly applicable to the algorithm presented below. Here, recognizing that the data layout of a block Toeplitz structure allows for the ordering of embedding, Fourier transformation, and projection to be partially interchanged, we present an algorithm that performs block Toeplitz matrix-vector multiplication by dividing each dimension into a pair of branches. In this "divide and conquer" approach, circulant embeddings are replaced by the creation of phase modified vector copies containing even and odd Fourier coefficients (section 4.3.1). This change enables a lazy evaluation strategy that can substantially reduce memory pressure, $\propto \left(2^d + 1\right) / \left(d + 2\right)$ in the case of electromagnetics, and operational complexity. Moreover, the partitioned nature of the algorithm presents several possibilities for tunable parallelization to increase computational speed, or handle large block Toeplitz systems. An accompanying implementation used for all presented results, written in Julia, is available from https://github.com/alsirc/SplitFFT_lazyEmbed. As the procedure is straightforward (section 4.3.2), and FFT dominated, similar performance should be simply achievable in any major scientific computing language.

4.3 Algorithm

Conceptually, the proposed algorithm is based on an observation about how the padding elements of a given level of embedding are incorporated into subsequent FFTs. Working

from the finest level of Toeplitz structure (numbers) to the coarsest (outer matrix blocks), the added zeros are only mixed with the input vector information once the FFT of the particular level has been completed. Analogously, as soon as the inverse FFT of a particular level has been carried out, half of the elements no longer influence the final matrix-vector product: they are obviated by subsequent projection, Fig. 4.1 right. As such, by postponing each embedding step until it is necessary, and performing each projection as soon as possible, required memory and computation can be reduced.

The properties described under subsection 4.3.1 indicate how lazy (postponed) embedding and eager (as soon as possible) projection are achieved by a division of the input vector at each level into "original" and phase-modified "child" copies that carry even and odd Fourier coefficients. Via this branching structure, the overall matrix-vector operation is partitioned into smaller calculations, which can be leveraged to reduce memory pressure—information is only calculated once it is needed to continue evaluation of the topmost (all-even) branch Fig. 4.2. The structure also offers possibilities for additional parallelization. For instance, in a multi-device setting, the linearity of the Fourier transform could be used to perform parallel matrix-vector multiplication in a number of ways: a branch may either communicate and merge, decreasing overall computation, or continue to some other predefined termination point, decreasing communication overhead. To handle larger systems, the decomposition structure of the Fourier transform could be used to further divide the number of coefficients treated at any one time. That is, $\mathbf{v}$ could be pre-partitioned into nearly independent even and odd parts.

4.3.1 Splitting and merging of vector coefficients

At each level, splitting into even and odd indices is accomplished using the following property. Let $\mathbf{v}$ be a vector of size n. Define $\tilde{\mathbf{v}}$ as $\tilde{\mathbf{v}} = [\mathbf{v}, 0]$, a vector of size $2n$, and P as the diagonal phase shift operator, $P = \text{diag}\left[1, \ldots, e^{i\pi k/n}, \ldots, -1\right]$. $\mathcal{F}(\mathbf{v})$ and $\mathcal{F}(P\mathbf{v})$ are then, respectively, the even and odd coefficients of $\mathcal{F}(\tilde{\mathbf{v}})$, where $\mathcal{F}$ denotes the Fourier transform.

Proof. Let v_k denote coefficients of the vectors $\mathbf{v}$, and $\tilde{v}_k$ the coefficients of $\tilde{\mathbf{v}}$.

$$\mathcal{F}(\tilde{\mathbf{v}}) = \left(\frac{1}{2n}\sum_{k=0}^{2n-2}\tilde{v}_k\tilde{\omega}^{lk}\right)_{l=0,\ldots n-1},$$

where $\tilde{\omega} = \exp(-\frac{i\pi}{n})$, and

$$\mathcal{F}(\mathbf{v}) = \left(\frac{1}{n}\sum_{k=0}^{n-1} V_k \omega^{lk}\right)_{l=0,\ldots,n-1} = \left(\frac{1}{n}\sum_{k=0}^{n-1} V_k \tilde{\omega}^{2lk}\right)_{l=0,\ldots,n-1},$$

where $\omega = \exp(-\frac{2i\pi}{n})$. Similarly,

$$\mathcal{F}(P\mathbf{v}) = \frac{1}{\sqrt{n}}\left(\sum_{k=0}^{n-1}\frac{\omega^{lk}}{\sqrt{n}}\exp\left(i\frac{l\pi}{n}\right)\right)_{l=0,\ldots,n-1} \times \mathbf{v}$$

$$= \frac{1}{n}(\sum_{k=o}^{n-1}\tilde{\omega}^{(2l+1)k} V_k)_{l=0,\ldots n-1}.$$

Therefore,

$$\mathcal{F}(\tilde{\mathbf{v}}) = \mathcal{F}(\tilde{\mathbf{v}}_{\text{even}}) + \mathcal{F}(\tilde{\mathbf{v}}_{\text{odd}}) = \mathcal{F}(\mathbf{v}) + \mathcal{F}(P\mathbf{v}).$$

□

Through this separation, the circulant embedding operation at a given level of the block Toeplitz structure is transformed into a branching operation, with each branch retaining the size of the initial vector.

After the diagonal multiplication step, branches are merged in reverse order—the projection counterpart to embedding by branching. Using the equivalence presented above, accounting for the difference in relative normalization factors, merging is accomplished via the formula

$$\tilde{\mathbf{v}} = \mathcal{F}^{-1}\left(\frac{1}{2}(\mathcal{F}(\mathbf{v}_{\text{even}}) + \overline{P}\mathcal{F}(\mathbf{v}_{\text{odd}}))\right).$$

4.3.2 Procedure

Making use of the above properties, the proposed algorithm follows the pseudo-code presented under Algorithm 1. Additional details on the Julia implementation available via https://github.com/alsirc/SplitFFT_lazyEmbed are given in the appendix. The protocol proceeds recursively by nested calls to the *toeMulBrn* function, which organizes the splitting, merging, and possible diagonal multiplication operations that occur on a specific level, or dimension, d. As visualized in Fig.4.2, the first task completed by *toeMulBrn* is to compute the FFT along the current level. *toeMulBrn* then creates a phase shifted copy of the transformed vector by applying the *sptBrn* function, and launches two progeny branches, respectively containing even and odd Fourier coefficients. Overall coordination is ensured by

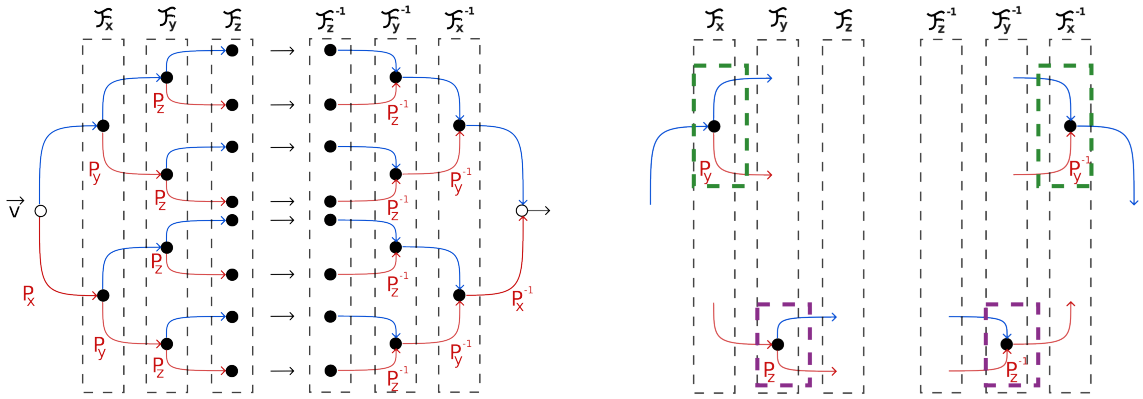

Figure 4.2 Left: **Branching form of proposed algorithm.** The left panel depicts the branching tree of data flow in 3D to perform block Toeplitz matrix-vector multiplication in the proposed algorithm, Alg. 1. Black dots are FFT tranformations, P labelled arrows represent odd (phase modified) Fourier coefficients, and black horizontal arrows are diagonal multiplications with block Toeplitz matrix data. Right: **Branch execution in recursive implementation.** The schematic shows the particular parts of the data flow that are controlled by a given branch (bId) in reference to Alg. 1. Two different levels of *toeMulBrn* are displayed, green and purple boxes. Each level launches two function calls, and then merges the retruned results. Control is then returned to the calling level.

passing each of these calls a branch identifier, *bId*: the even branch inherits the identifier of its caller, while the odd branch is assigned a new identifier by *nxtId*. The function then waits for these calls to return, merges their results, using *mrgBrn*, and performs an inverse FFT, *IFFT*, before returning. If the maximal level of block Toeplitz structure is reached, $d = d_{\max}$ then *toeMulBrn* launches no further calls, and instead computes the diagonal multiplication of its given vector with the appropriate Toeplitz data, $T\,[bId]$, via *mulBrn*. *toeMulBrn* then computes the inverse FFT, *IFFT*, and returns.

Algorithm 1: block Toeplitz multiplication

Require $d \geq 0$
Ensure $y = Tv$
$y \leftarrow v$
if $d > 0$ **then**
  {FFT along the d^{th} dimension}
  $v \leftarrow FFT_d(v)$
end
if $d < d_{max}$ **then**
  {create phase shifted vector (splitting)}
  $v_{\text{child}} \leftarrow sptBrn(v)$
  {launch next branches (recursive call)}
  $v \leftarrow toeMulBrn(T, d+1, bId, v)$
  $v_{\text{child}} \leftarrow toeMulBrn(T, d+1, nxtId(T, d), v_{\text{child}})$
  {merge v (even coeffs.) with v_{child} (odd coeffs.)}
  $v \leftarrow mrgBrn(T, v, d+1, P, v_{\text{child}})$
else
  {diagonal multiplication of v with Toeplitz data}
  $v \leftarrow mulBrn(v, T[bId])$
end
if $d > 0$ **then**
  {inverse FFT along the d^{th} dimension}
  $v \leftarrow iFFT_d(v)$
end

4.3.3 Operational complexity and memory usage

Consider a d-dimensional vector v of total size $s = n^d$ where $d > 0$. As before, let $\tilde{v}$ be the fully embedded image of v (along all dimensions), so that the total length of $\tilde{v}$ is $2^d s$. The complexity of a FFT as used in a given level of Alg. 1 is $m_2 \log_2(m_1)$ where m_1 is the total length of the vector to be transformed and m_2 is the product of its sizes along the dimensions to be transformed [12]. The complexity of the standard circulant embedding method is therefore

$$C_{\text{embed}} = \underbrace{2^{d+1} s \log_2(2^d s)}_{\text{forward and inverse FFTs}} + \underbrace{2^d s}_{\text{multiplication step}}$$

At the l^{th} level, $1 < l < d$, of Alg. 1 there are 2^l forward FFTs and 2^l inverse FFTs, each along one dimension. Including the additional diagonal multiplications involved with the phase shifting and multiplication with the Toeplitz data, the operational complexity of

Alg. 1 is thus

$$C_{\text{split}} = \underbrace{2\sum_{l=1}^{d} 2^l s \log_2(n)}_{\text{forward and inverse FFts}} + \underbrace{2^d s}_{\text{multiplication step}} + \underbrace{2\sum_{l=0}^{d-1} 2^l s}_{\text{phase shift on odd branches}} = 2(2^d-1)s(2\log_2(n)+1)+2^d s$$

Take $R_c = C_{\text{embed}}/C_{\text{split}}$ to be operational complexity ratio between the two methods. Combining the above,

$$R_c = \frac{2^{d+1} s \log_2(2^d s) + 2^d s}{2(2^d-1)s(2\log_2(n)+1)+2^d s} = \frac{d\log_2(2n)+1}{(1-2^{-d})(2\log_2(n)+1)+1},$$

so that $R_c \to d/\left(2-2^{-d+1}\right)$ as $n \to \infty$.

In terms of memory usage, the standard circulant method requires an overall doubling in vector size for each level (dimension) of block Toeplitz structure. The memory required for both the Toeplitz data and input vector is, consequently, $M_{v,\text{embed}} = 2^d s = M_{T,\text{embed}}$, with s as above. For Alg. 1, a lazy evaluation strategy leads peak memory usage to occur when the first branch (top of Fig. 4.2) is processed up to multiplication with its corresponding Toeplitz data, leading to a peak allocation of $M_{v,\text{split}} = (d+1)s$ elements. For the Toeplitz information, the full embedded vector of Fourier transformed data is split into 2^d parts of s coefficients each, so $M_{T,\text{split}} = 2^d s$. Comparing these results, the peak memory ratio R_m of the two approaches, in full generality, is

$$R_m = \frac{M_{v,\text{embed}} + M_{T,\text{embed}}}{M_{v,\text{split}} + M_{T,\text{split}}} = \frac{2^{d+1} s}{(d+1)\, s + 2^d s} = \frac{2}{(d+1)\, 2^{-d} + 1}$$

Because electromagnetics, acoustics, and quatum mechanics typically exhibit reciprocity between sources and receivers [14], there are a number of settings of physical interest wherein system matrices are both block Toeplitz and block symmetric (resp. block skew-symmetric). In such cases, the Fourier transform of the vector containing the generating elements of the associated Toeplitz matrix is mirror symmetric (resp. skew-symmetric) about the middle index of each dimension, and thus can be fully stored in a vector of size s. In such cases, the peak memory ratio between the two methods becomes

$$R_{m,sym} = \frac{2^d + 1}{(d+2)}. \tag{4.1}$$

More precisely, if $\mathbf{v}$ is a vector with coefficients $(v_k)_{k=0,n-1}$ such that $v_i = v_{n-i}$ for $i = 1, ..n-1$, as happens in the case of a symmetric block Toplitz matrix, then its Fourier coefficents $\mathbf{f}$ ex-

hibit the symmetry $f_i = f_{n/2+i}$. Hence, only half the coefficients must be stored. Inductively, an equivalent reduction occurs at each level, and single vector's worth of Toeplitz data is required. The antisymmetric case is analogous as $v_i = -v_{n-i}$ implies that $f_{i+n/2} = -f_i - 2f_0$.

Example of coefficient symmetry for two levels

$$\begin{bmatrix} f^{00} & f^{01} & f^{02} & f^{03} & 0 & f^{03} & f^{02} & f^{01} \\ f^{10} & f^{11} & f^{12} & f^{13} & 0 & f^{13} & f^{12} & f^{11} \\ f^{20} & f^{21} & f^{22} & f^{23} & 0 & f^{23} & f^{22} & f^{21} \\ f^{30} & f^{31} & f^{32} & f^{33} & 0 & f^{33} & f^{32} & f^{31} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ f^{30} & f^{31} & f^{32} & f^{33} & 0 & f^{33} & f^{32} & f^{31} \\ f^{20} & f^{21} & f^{22} & f^{23} & 0 & f^{23} & f^{22} & f^{21} \\ f^{10} & f^{11} & f^{12} & f^{13} & 0 & f^{13} & f^{12} & f^{11} \end{bmatrix}$$

Table 4.1 Asymptiotic theoretic complexity, Eq. 4.1, and peak memory ratios, Eq. 4.1 and 4.1.

Dimensions	2	3	4	5	6
R_c	1.33	1.71	2.13	2.58	3.05
R_m	1.14	1.33	1.52	1.68	1.80
Rm, sym	1.25	1.80	2.83	4.71	8.13

4.4 Results

As analytic complexity calculations must necessarily overlook many, potentially influential, practical implementation details, we directly measure the relative performance of the split and standard circulant embedding algorithms by performing block Toeplitz matrix-vector multiplication for various dimension numbers and vector sizes.

For convenience, the length of each dimension is assumed to be consistent, but this is not a limitation of either method (or code). Our presented results focus on lengths corresponding to powers of 2, as FFTs are most efficient for these lengths. Plots of cases where the sizes are products of primes and plots of consecutive natural numbers are available in the appendix. All results were obtained through local computation on a laptop with 16 GiB of RAM, and an AMD® Ryzen 7 5800hs processor. Although the Julia implementation can be run on GPU, for simplicity, only parallelized, multi-threaded, CPU computations are compared.

The coefficients used for the matrix vectors and the input vector are complex numbers with random real and imaginary part with mean value equal to 0 and a parametrizable variance. The number of points for each plot is limited by the memory capacity of the machine used for the computation.

The results obtained do not clearly show the tendencies predicted by operational complexity calculations. Namely, Alg. 1 is generally faster than expected, and only converges to the predictions of Eq. (4.1) as the memory limit of the test machine is reached. The broad uncertainty observed (see appendix) suggests that the statistical data captured by BenchmarkTools.jl is independent and uncorrelated. These differences in measured performance, compared to Eq. (4.1), and large variance of results, may be caused by several factors. First, all FFTs are computed using the FFTW package [17], which utilizes different methods depending on the structure of the vector to be transformed, and has an internal thread management system that is not controlled by Julia. For a fixed input vector, the two methods perform FFTs of different sizes, and smaller sizes are known to be comparatively faster than expected based theoretical complexity [18]. Second, the standard circulant embedding method employs full thread parallelization on fully embedded vectors at all steps. Conversely, Alg. 1 consistently applies thread parallelization at the size of the input vector and makes greater use of the Julia task scheduler. For small numbers of tasks, and small loops, launch overhead becomes a meaningful percentage of the benchmarks. Parallelization of small loops can also lead to highly variable memory writes and reads, which could explain the spread of measured ratios.

4.5 Outlook

In this article, we have described how the standard circulant embedding method for multi-level block Toeplitz matrix-vector multiplication can be improved by reordering embeddings and projections with respect to their associated Fourier transformation level. Concretely, by utilizing a lazy embedding / eager projection strategy, for d levels of Toeplitz structure, we have shown that operational complexity can be reduced by a factor of $d/\left(2-2^{-d+1}\right)$, and peak memory usage by a factor of $2/\left((d+1)\,2^{-d}+1\right)$. As the size of the matrix-vector product tends to infinity, for 3 levels of structure, these ratios tend to 12/7 and 4/3 respectively. For a fully symmetric (resp. skew-symmetric) matrix peak memory reduction improves to a factor of $\left(2^d+1\right)/\left(d+2\right)$—9/5 for $d=3$. In simulation, the wall clock time ratio between the two methods is better than theoretical expectations in almost all cases. The proposed algorithm also suggests several way in which additional parallelization could be introduced to handle the large scattering problems that occur in acoustics and electromagnetics. In fact,

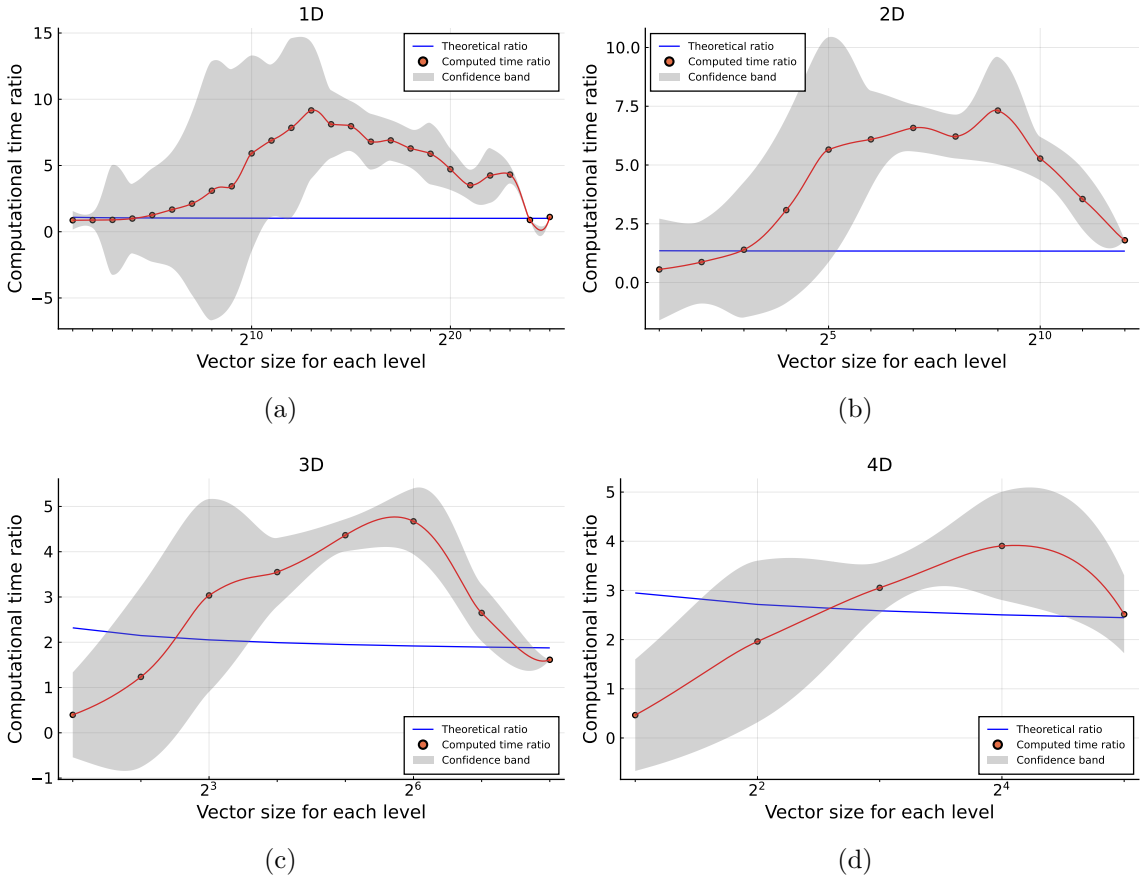

Figure 4.3 **Comparison of wall clock and operational complexity ratios.** The figure depicts the wall clock time ratio of the Julia implementation of Alg. 1 (https://github.com/alsirc/SplitFFT_lazyEmbed) compared to standard circulant embedding for vectors with lengths corresponding to powers of 2. In moving from panel (a) to (d), the dimensionality of block Toeplitz structure is increased from 1 (Toeplitz matrix) to 4.

a slightly modified version of the (GPU enabled) Julia implementation is already being used in the GilaElectromagnetics package (https://github.com/moleskySean/GilaElectromagnetics.jl), currently under development by the authors. The algorithm relies only on (highly optimized) standard library functions, and should be easily adaptable to almost any relevant context.

4.6 Acknowledgment

The authors acknowledge Justin Cardona and Paul Virally for their careful rereading and feedback on this article. The first author especially thanks Paul Virally for his support and guidance through the use of different packages used in this project. This work was supported by the Québec Ministry of Economy of Innovation through the Québec Quantum Phonotnics Program, the National Sciences and Engineering Reasearch Council of Canada via Discovery Grant RGPIN-2023-05818, and the Canada First Research Excellence Fund through its collaboration with the Institut de Valorisation des Données (IVADO).

Bibliography

[1] C. Jelich, M. Karimi, N. Kessissoglou, S. Marburg, "Efficient solution of block Toeplitz systems with multiple right-hand sides arising from a periodic boundary element formulation", *Engineering Analysis with Boundary Elements*, vol. 130, pp. 135-144, 2015.

[2] S. A. Goreinov, D. V. Savostyanov, E. E. Tyrtyshnikov, "Tensor and Toeplitz Structures Applied to Direct and Inverse 3D Electromagnetic Problems", *PIERS Proceedings*, Institute of Numerical Mathematics, Russian Academy of Sciences, 2009.

[3] B. E. Barrowes, F. L. Teixeira and J. A. Kong, *Fast Algorithm for matrix-vector multiply of asymmetric multilevel block Toeplitz matrices in 3D scattering.* Massachusetts, U.S., Massachusetts Institute of Technology, 2001.

[4] M. Karimi, P. Croacker, N. Kessissoglou, "Boundary element solution for periodic acoustic problems", *Journal of Sound and Vibration*, School of Mechanical and Manufacturing Engineering, UNSW Australia, Sydney, Australia, 2015

[5] P. A. Voois, "A Theorem on the Asymptotic Eigenvalue Distribution of Toeplitz-Block-Toeplitz Matrices", *IEEE Transactions on Signal Processing*, vol. 44, no. 7, 1996.

[6] P.J.S.G. Ferreira, M. E. Domínguez, "Trading-off matrix size and matrix structure: Handling Toeplitz equations by embedding on a larger circulant set", *Digital Signal Processing*, Vol. 20, pp. 1711-1722, 2010.

[7] D. Lee, "Fast Multiplication of a Recursive Block Toeplitz Matrix by a Vector and its Application", *Journal of Complexity*, Vol. 2, pp. 295-305, 1986.

[8] A. Carriow and M. Gliszczyński, *The Fast algorithms to compute matrix-vector products for Toeplitz and Hankel matrices.* Szczecin, PL, West Pomeranian University of Technology, 2012.

[9] I. Gohberg, V. Olshevsky, "Fast Algorithms with Preprocessing for Matrix-Vector Mutliplication Problems", Ramat Aviv, Israel, Tel Aviv University, *Journal of Complexity*, Vol.10, pp. 411-427, 1993.

[10] V. A. Kazeev, B. N. Khoromskij, E. E. Tyrtyshnikov, "Multilevel Toeplitz matrices generated by tensor-structured vectors and convolution with logarithmic complexity", Max-Plank-Institute für Mathematik, *Naturwissenshaften Leipzig*, 2011.

[11] G. Heinig, R. Rost, "Representations of Toeplitz-plus-Hankel matrices using trigonometric transformations with application to fast matrix-vector multiplication", *Linear Algebra and its Applications*, Vol. 275-276, pp. 225-248, 1998.

[12] X. Wen, M. Sandler, "Calculation of radix-2 discrete multiresolution Fourier transform", *Signal Processing*, vol. 87, no. 10, pp. 2455-2460, 2007.

[13] Z. Hu, H. Wan, "A Novel Generic Fast Fourier Transform Pruning Technique and Complexity Analysis", *IEEE Transaction on signal processing*, vol. 53, no. 1, 2005.

[14] J.D. Jackson, *Electrodynamics*, *The Optics Encyclopedia* (eds T.G. Brown, K. Creath, H. Kogelnik, M.A. Kriss, J. Schmit and M.J. Weber), 2007. https://doi.org/10.1002/9783527600441.oe014

[15] P. D. Lax, R. S. Phillips, *Scattering Theory*, *Academic Press, Inc.*, 1990.

[16] T. Søndergaard, "Modeling of plasmonic nanostructures: Green's function integral equation methods", *Physica Status Solidi*, vol. 244, Issue 10, 2007.

[17] M. Frigo, S. G. Johnson, "The Design and Implementation of FFTW3", *Proc. IEEE*, vol. 93, no. 2, pp. 216–231, 2005.

[18] Y. Dotsenko, S. S. Baghsorkhi, B. Lloyd, N. K. Govindaraju, "Auto-tuning of fast fourier transform on graphics processors", *ACM SIGPLAN Notices*, vol. 46, no. 8, pp. 257-266, 2011.

[19] A. CHAI, M. MOSCOSO, G. PAPANICOLAOU "Imaging strong localized scatterers with sparsity promoting optimization", *SIAM Journal on Imaging Sciences*, vol. 7, no. 2, pp. 1358-1387, 2014.

4.7 Appendix

4.7.1 Julia code structure and input variables explanation

The code for the algorithm is composed of two files: *toeUtilities.jl* and *toeMatVecProd.jl*. The first file contains all the Julia structures used by the algorithm; the second contains all the functions that process the input data. The code is launched running *toeUtilities.jl*.

The first structure of *toeUtilities.jl* contains the condensed information about the Toeplitz matrix required to compute the multiplication. The first field is an array containing the sizes of all the levels of the matrix. For instance, a 3-level block-Toeplitz of size 2^{20} on its two outermost layers and 2^{10} for its innermost layer, shall have dimInf = $[2^{20}, 2^{20}, 2^{10}]$. The second field corresponds to the coefficients of the matrix organised in a list of 2^d arrays — where d is the number of levels — such that each element of the list can be multiplied by the corresponding split vector (see tree-branch structure [Fig. 4.2]). The matrix data should be rearranged to fit this format. Below is a code snippet performing this operation:

```
# enter the number of level of the block-Toeplitz
lvls = ???

# enter the full Fourier transformed block-Toeplitz
# matrix of type Array{ComplexFXX, lvls}
toeFFT = ???
eoDim = ^(2, lvls)
toeInf = Array{Array{ComplexF64}}(undef, eoDim)
not_a_filter = Tuple([Colon() for i in 1:2^(lvls - 1)])

# write Toeplitz data
for eoItr in 0:(eoDim - 1)

    # odd / even branch extraction
    toeInf[eoItr + 1] = Array{ComplexF64}(undef, dimInf...)

    # first division is along smallest stride -> largest binary division
        toeInf[eoItr + 1][not_a_filter...] .= toeFFT[Tuple([(1 +
 mod(div(eoItr, 2^(lvls - d)), 2)):2:(2*dimInf[d] - 1 +
 mod(div(eoItr, 2^(lvls - d)), 2)) for d in 1:lvls])...]
```

```
    # verify that all values are numeric.
    if maximum(isnan.(toeInf[eoItr + 1])) == 1 ||
                maximum(isinf.(toeInf[eoItr + 1])) == 1
                error("Fourier information contains non-numeric values.")
    end
end
```

The last three fields of the structure are pre-computed forward and reverse FFTs — using the plan_fft and plan_ifft functions from FFTW package — and the array containing the phase shift operators. The latter can be constructed with $P_j = \left(e^{\frac{-i\pi k}{\text{dimInf}[j]}}\right)_{k\in 0,...\text{dimInf}[j]-1}$, for all $j \in 1, ...\text{lvls}$. For the purpose of this article, these arrays are computed and generated outside the main function not to include their computation time in the total measured time needed to process the algorithm. The second structure manages the different GPU kernels and threads. The third structure manages the different processors that should be used to perform the product. The first field is a tuple of two booleans. The first one is a flag that determines whether the CPU does at least half the work using all available threads, and the other is a flag to either run the code on GPU or on CPU. The second coordinate contains the runtime options for running the GPU kernels.

Finally, the multiplication is launched by the toeMul function throught the toeMulBrn function on the second file. The entries required for that function to perform the multiplication are:

- toeOpr, that uses the toeDat structure defined above and contains all the information needed by the program

- vOrg is the vector multiplied by the block-Toeplitz matrix. It has to share the same tensor structure as the block-Toeplitz matrix such that [size(vOrg)...] == dimInf is true.

Note that the result of the multiplication will then be stored in this same vOrg variable in order to save memory.

4.7.2 Extra plots with sizes in product of prime numbers and for a certain range of numbers

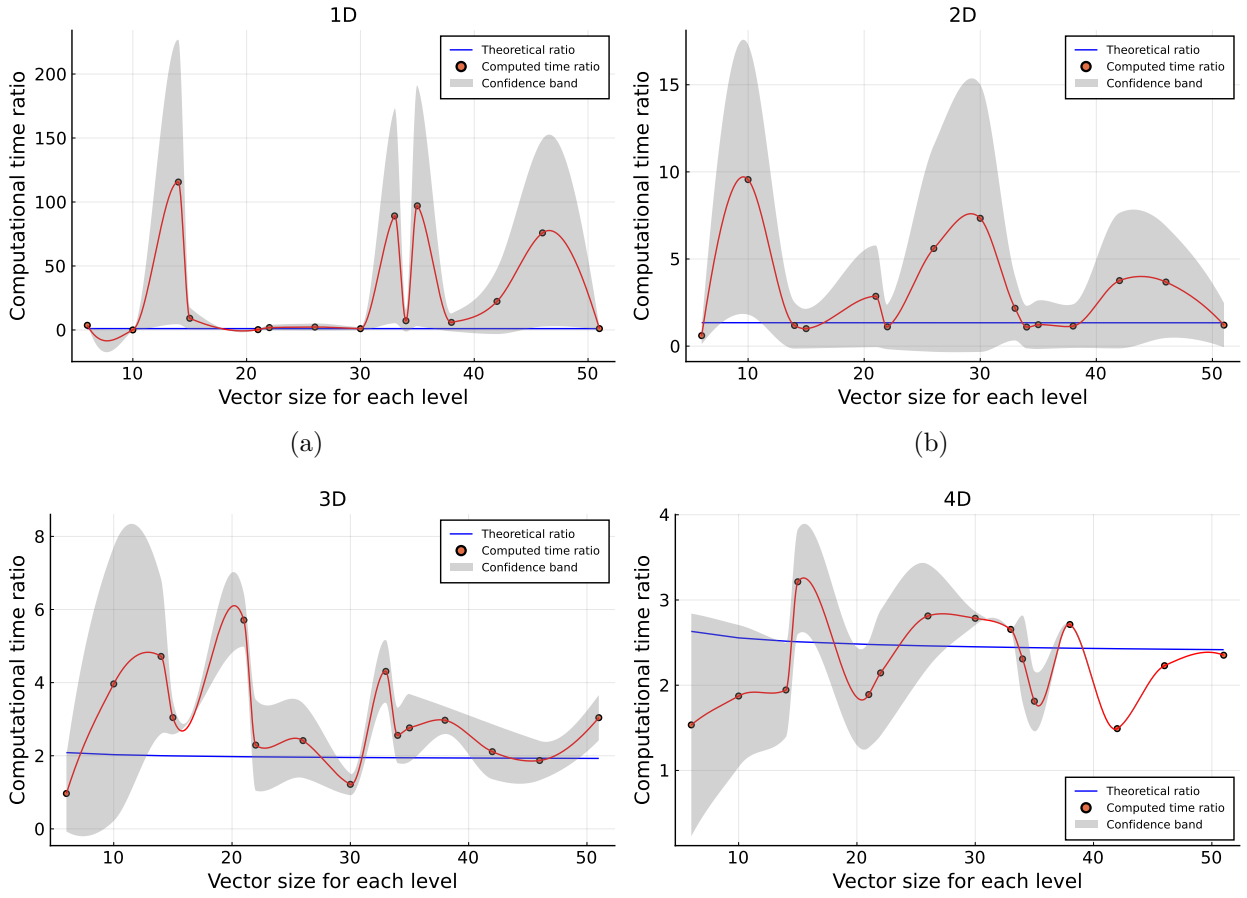

Figure 4.4 Computation time ratio of the two methods with vector length in product of two prime numbers up to 51 with uncertainty band in gray for one to four dimensions (resp. (a) to (d)).

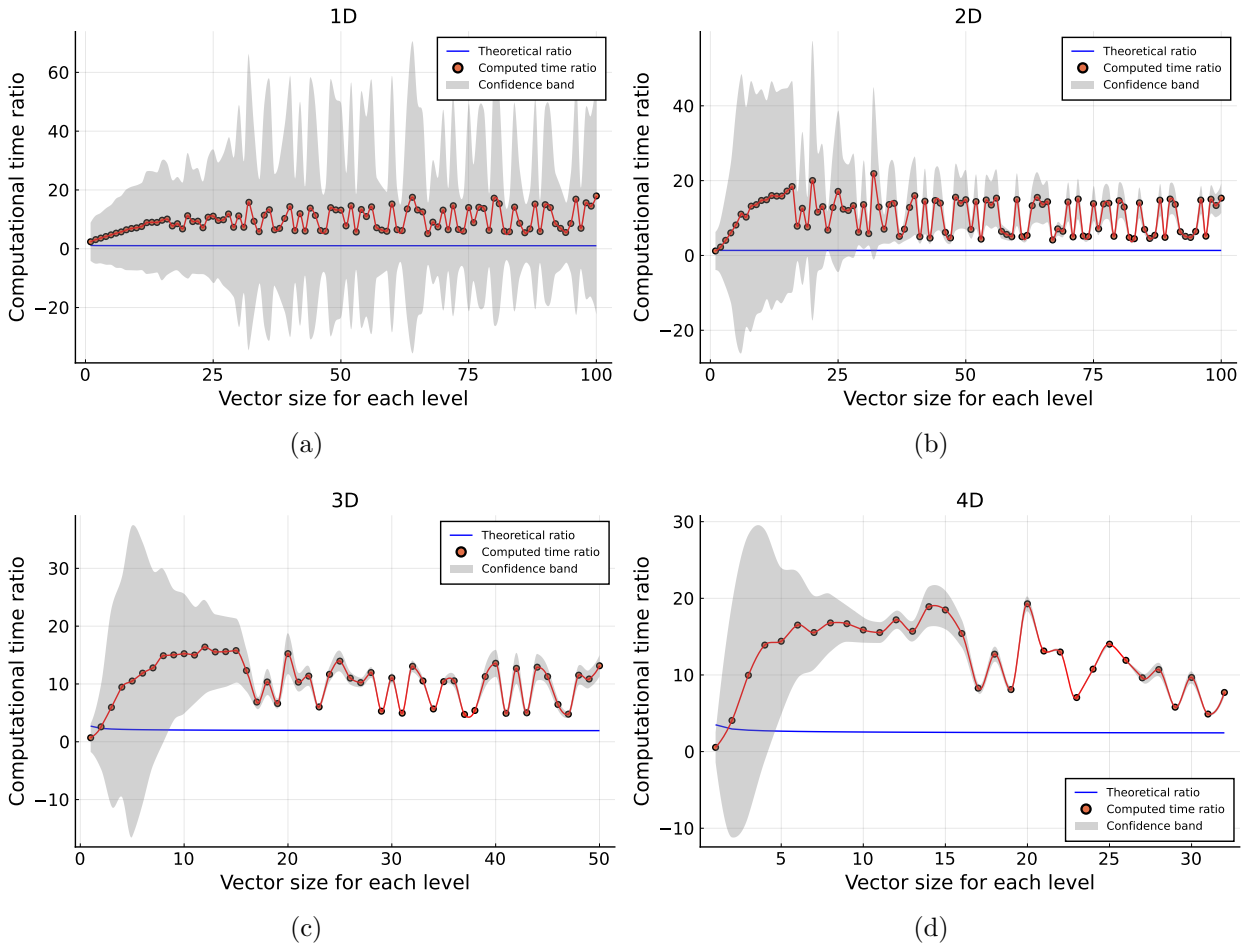

Figure 4.5 Computation time ratio of the two methods with vector length from 1 to 100 with uncertainty band in gray and smooth curve on mean values for one and two dimensions, from 1 to 50 for three dimensions and from 1 to 32 in four dimensions (resp. (a) to (d)).

CHAPTER 5 GENERAL DISCUSSION

This chapter will provide a critical analysis of the split circulant embedding algorithm presented in section 4.3. We will begin by examining the discrepancies between the theoretical and measured time complexities (section 5.1). Following this, we will compare the theoretical performance of the algorithm with that of the tensor decomposition method discussed in the literature review in section 2.2. Additionally, we will explore the tunability offered by the algorithm for GPU parallelization in the 3-dimensional case (section 5.3).

5.1 Complement to the results analysis

The results observed in the article for the time complexity ratio of the two methods display an important difference compared to the theoretical complexity ratio. Namely, the split circulant embedding method performs better than the theoretical expectations until the memory limit of the test machine is reached. Moreover, each mean value point is wrapped in a broad uncertainty, showing that the data captured by the *BenchmarkTools.jl* package is uncorrelated. Two hypotheses were proposed to try to explain this phenomenon.
The first possibility is that the internal task scheduler from the *FFTW.jl* package does not cooperate properly with the Julia task scheduler, applying the different task to each threads automatically, without the possibility to manually assign the threads for the Fourier transforms. The second proposal to explain the differences with the theoretical complexity is that the real time complexity of the fast Fourier transform on sizes in powers of 2 performed by the *FFTW.jl* package does not follow the theoretical $O(N\log N)$ but has some adjustment that could explain the convex shape of the result (increasing and then decreasing). This would be visible in the time ratio since for a d-dimensional vector of size $N = n^d$, the split circulant embedding performs 2^{d+1} FFTs and IFFTs on N-long vectors, while the standard circulant embedding performs d FFTs and IFFTs on a larger $2^d N$-long vector.

5.1.1 Single thread computation

The first hypothesis can simply be checked by running the same vector size trials on a single thread to see if significant change are observed.

The required simulations are essentially the same as those that were required to produce the results presented in the article for vector lengths in powers of 2 with different random

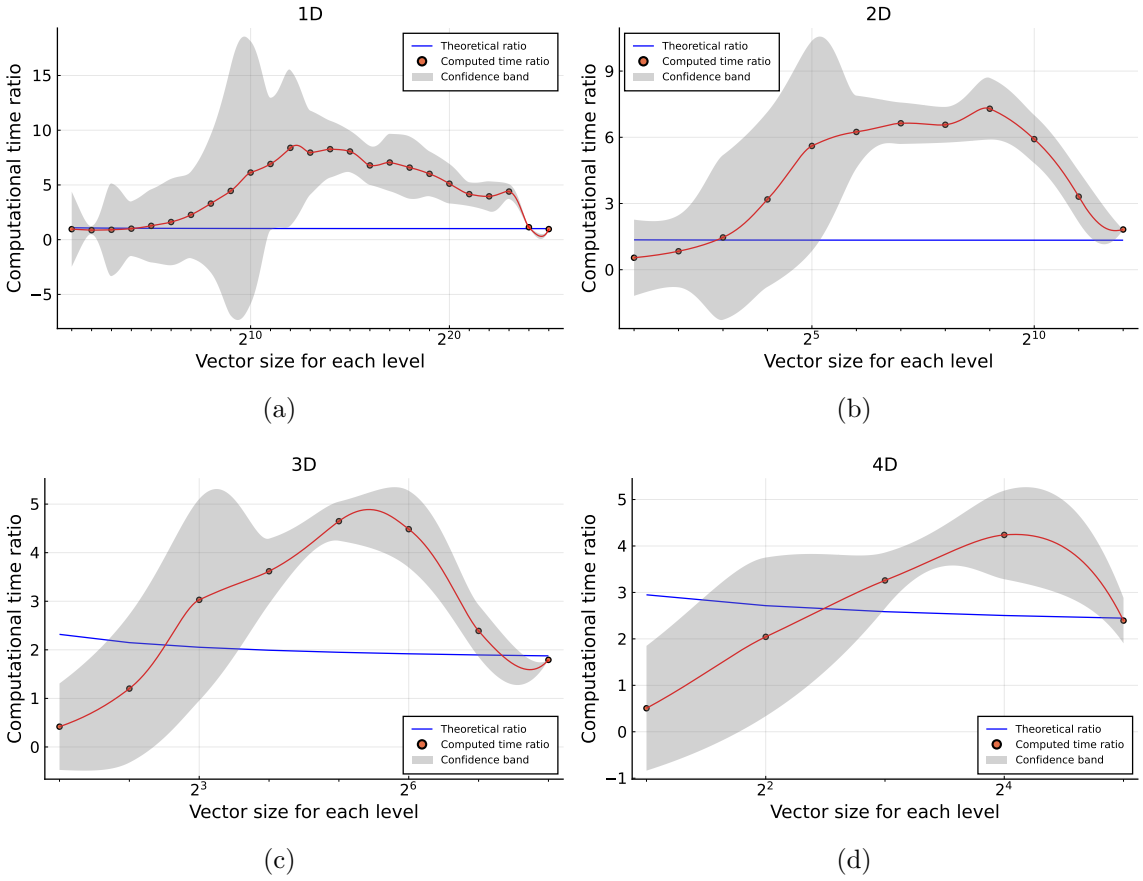

Figure 5.1 **Comparison of wall clock and operational complexity ratios run on a single thread.** The figure depicts the wall clock time ratio of the Julia implementation of Alg. 1 (https://github.com/alsirc/SplitFFT_lazyEmbed) compared to standard circulant embedding for vectors with lengths corresponding to powers of 2. In moving from panel (a) to (d), the dimensionality of block Toeplitz structure is increased from 1 (Toeplitz matrix) to 4.

coefficients (now run on a single thread). The striking similarity between figures 4.3 and 5.1 shows that the proposal of an independent task scheduler for the *FFTW.jl* package has to be rejected, or at least if there is, its influence is not the main cause of the variation we observe in the results of the time complexity ratio.

5.1.2 Computational time ratio of two consecutive powers of 2-long vector

To check the second assumption, one could plot the time complexity ratio between two consecutive sizes in powers of two and compare it with the theoretical consecutive ratio. We will therefore consider the ratio of two consecutive computational times from the results in section 4.3 and only for the split circulant embedding method.
Thus, we define the theoretical ratio for two consecutive vector sizes as $R_{\text{consec.}} = C_{\text{split}}(2s)/C_{\text{split}}(s)$ where C_{split} is the theoretical complexity of the split circulant embedding method determined in section 4.3. Therefore,

$$R_{\text{consec.}} = \frac{2\left(2^d-1\right)(2n)^d(2\log_2(2n)+1)+(4n)^d}{2\left(2^d-1\right)n^d(2\log_2(n)+1)+(2n)^d} = 2^d\frac{2\left(2^d-1\right)n^d(2\log_2(n)+3)+(2n)^d}{2\left(2^d-1\right)n^d(2\log_2(n)+1)+(2n)^d}$$

The difference between the theoretical and measured consecutive computational time ratios obtained in Figure 5.2 does not follow the theoretical ratio. As such, we can conclude that the measured complexity of the FFT performed by the *FFTW.jl* package does not exactly follow the theoretical $N\log N$ trend, and this difference is clearly one of the causes of variation between the measured and theoretical complexity ratios presented in the article. Indeed, the two methods perform different FFTs for the same input matrix and vector. While the standard circulant embedding method computes full vector Fourier transforms for each level, the split circulant embedding computes smaller FFTs, which are quicker to compute relative to the size of the vector. However, the variation observed in Fig. 4.3 does not fully explain why the lazy embedding algorithm is generally faster than expected; the peak ratio seen in Fig. 4.3 is reached at 2^{14}, the consecutive time ratio at 2^{14}, is nearly 1, in Fig. 5.2.

5.2 Time complexity and memory comparison with the tensor decomposition method

Knowing the complexity of the split circulant embedding method, it is relevant to compare it to the tensor decomposition method developed in the literature review in section 2.2 and determine the cases where the one is more efficient than the other in terms of complexity or memory capacity. We will consider one iteration of the DMRG method developed by Kazeev

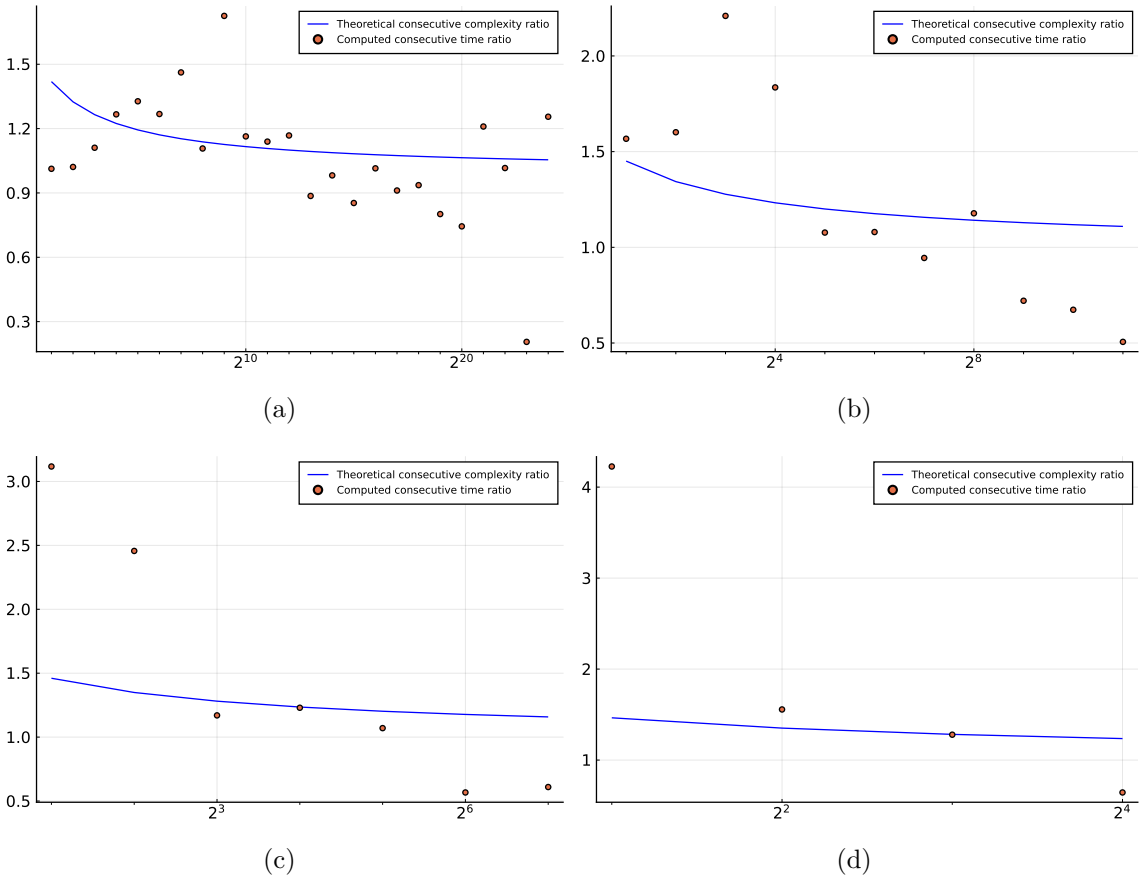

Figure 5.2 **Comparison of wall clock and operational complexity ratios between two consecutive vector size in powers of 2.** The figure depicts the wall clock time ratio of the Julia implementation of Alg. 1 (https://github.com/alsirc/SplitFFT_lazyEmbed) of two consecutive points with lengths corresponding to powers of 2. In moving from panel (a) to (d), the dimensionality of block Toeplitz structure is increased from 1 (Toeplitz matrix) to 4.

et al. [31] to determine the approximate maximal rank for which this method would perform better than the split circulant embedding, provided that the QTT decomposition of the outer product $\mathbf{x}\mathbf{y}^\dagger$ is known.

However, the iterations of DMRG method do not seem to be the bottleneck of the QTT decomposition method for Toeplitz matrix-vector multiplication since the operational complexity of the conversion into QTT format is $O(dlR^4)$, where R is the maximum of the TT-ranks. While the ranks of the iteration of the DMRG method can be reduces through different means of approximation, R stands for the exact maximum TT-rank. Furthermore, the first SVD computed for the QTT decomposition of the tensor is more expensive than the matrix multiplication ($m^2n + n^3$ multiplications for a $m \times n$ matrix). For the QTT decomposition to be relevant in the context of electromagnetic simulations in scattering theory (section 3.1), one would need to know enough information on the source vector to be able to provide a QTT decomposition of the outer product without computing it, thus reducing the complexity of the conversion below the one of the matrix-vector multiplication. The conditions for low rank source vector and the method of approximation used to perform such QTT decomposition could be the object of future research.

We recall that for a d-dimensional vectors of length $n = 2^l$ along each of its dimension,

$$C_{\text{split}} = 2\left(2^d - 1\right) n^d (2\log_2(n) + 1) + (2n)^d = 2\left(2^d - 1\right) 2^{ld}(2ld + 1) + 2^{d(l+1)}$$

is the theoretical complexity for the split FFT method and that

$$C_{\text{tensor}} = dlr_z^3 + dlr_x r_y r_z (r_x + r_y + r_z) \leq dlr^3(3r + 1)$$

is the one for the quantized tensor train decomposition method (section 2.2.2) where $r = \text{Max}\{r_x, r_y, r_z\}$.

For $d = 3$, the tensor decomposition is the more efficient when

$$C_{\text{tensor}} \leq C_{\text{split}} \iff 3lr^3(3r + 1) \leq 14 \times 8^l(6l + 1) + 8^{l+1} \iff r^3(3r + 1) \leq \frac{8^l}{3l}(84l + 22)$$

This inequality defines a maximal rank $r_{\max}$ for which the tensor decomposition has lower complexity than the split FFT method at a given size $n = 2^l$. One can compare this rank to the maximum rank allowed by the vector size in the QTT format in 3D: $n^d = 2^{dl} = 8^l$. Hence, we define a rank ratio as $r_{\text{ratio}} = r_{\max}/8^l$ to better emphasize how low this rank is compared to the current vector size.

Tables 5.1 and 5.2 show that the maximum QTT rank is very low. From $l = 4$—which would

Table 5.1 Maximal QTT rank for the QTT decomposition to require fewer operations than the split FFT method and its ratio with the maximum possible rank for vector size from 2 to 2^8.

ℓ	1	2	3	4	5	6	7	8
$r_{\max}$	3	5	8	14	23	39	67	112
$r_{\text{ratio}}(\%)$	37.5	7.8	1.56	0.34	0.07	1.45e-2	3.24e-3	6.71e-4

Table 5.2 Maximal QTT rank for the QTT decomposition to require fewer operations than the split FFT method and its ratio with the maximum possible rank for vector size from 2^9 to 2^{16}.

ℓ	9	10	11	12	13	14	15	16
$r_{\max}$	189	318	535	899	1512	2543	4275	7188
$r_{\text{ratio}}(\%)$	1.40e-4	2.96e-5	6.22e-6	1.31e-6	2.75e-7	5.81e-8	1.24e-8	3.01e-9

correspond to a computational domain of a $16 \times 16 \times 16$ cube—the rank ratio defined above is less than 0.34% and drops rapidly as the size increases in powers of 2.

Regarding the memory capacity, we recall that the peak memory storage of the tensor decomposition method for 3-dimensional vectors is

$$M_{\text{tensor}} = 3l\left(r_x^2 + r_y^2 + 4\right) \leq 3l\left(r^2 + 4\right)$$

For the split FFT method,

$$M_{\text{split}} = 4 \times 8^l + 8^{l+1} = 12 \times 8^l$$

Therefore,

$$M_{\text{tensor}} \leq M_{\text{split}} \iff 3l\left(2r^2 + 4\right) \leq 12 \times 8^l \iff r^2 + 2 \leq 2 \times 8^l/l$$

As $r > 0$, the inequality is equivalent to $r \leq \sqrt{\frac{2}{l}8^l - 2}$. We can then build the same tables for the memory capacity comparison.

The tables 5.3 and 5.4 show similar results as the table for theoretical time complexity. Thus, the DMRG method for QTT decomposed vectors performs better in terms of peak memory storage than the split FFT method only for very low tensor rank. One can notice that the difference is slightly less significant compared to the complexity comparison case, which em-

Table 5.3 Maximal QTT rank for the QTT decomposition to store fewer coefficients at peak than the split FFT method and its ratio with the maximum possible rank for vector size from 2 to 2^8.

ℓ	1	2	3	4	5	6	7	8
$r_{\max}$	3	7	18	45	114	295	774	2048
$r_{\text{ratio}}(\%)$	37.5	10.9	3.52	1.10	0.35	0.11	3.69e-2	1.22e-2

Table 5.4 Maximal QTT rank for the QTT decomposition to store fewer coefficients at peak than the split FFT method and its ratio with the maximum possible rank for vector size from 2^9 to 2^{16}.

ℓ	9	10	11	12	13	14	15	16
$r_{\max}$	5.46e3	1.47e4	3.95e4	1.07e5	2.91e5	7.93e5	2.17e6	5.93e6
$r_{\text{ratio}}(\%)$	4.07e-3	1.36e-3	4.60e-4	1.56e-4	5.29e-5	1.80e-5	6.16e-6	2.11e-6

phasizes the fact that possible memory savings is the primary benefit of the QTT format, more so than the complexity of its elementary operations.

5.3 Details on the GPU parallelization possibilities in 3 dimensions

In this section, we will develop the parallelization possibilities of the algorithm as this point is only briefly discussed in the article. Indeed, the even and odd splitting in the algorithm causes a tree-branch structure that spans over 2^d branches—d being the number of levels in the multi-level block Toeplitz structure—and then recombines to the end result (section 4.4). This tree-branch structure gives tunability in the memory management, allowing the user to parallelize the computation on different GPUs for each dimension to trade some memory capacity for speed.
Consider for instance the 3-dimensional case whose tree-branch schematic is presented below in Figure 5.3.

The setup that offers the lowest memory usage is a lazy evaluation strategy. In that case, the different splits and recombinations follow the alphabetical order in Figure 5.3 and the peak memory of the algorithm is reached when the maximum number of branches is started, i.e right before the first recombination (at node I in the 3D case). The memory storage of the vector is then $(d+1)s$, where $s = n^d$ is the total length of the vector.
Alternatively, one can choose to parallelize the computation by considering the vector at node A and the one at node B as two separated matrix-vector multiplications. Thus, one

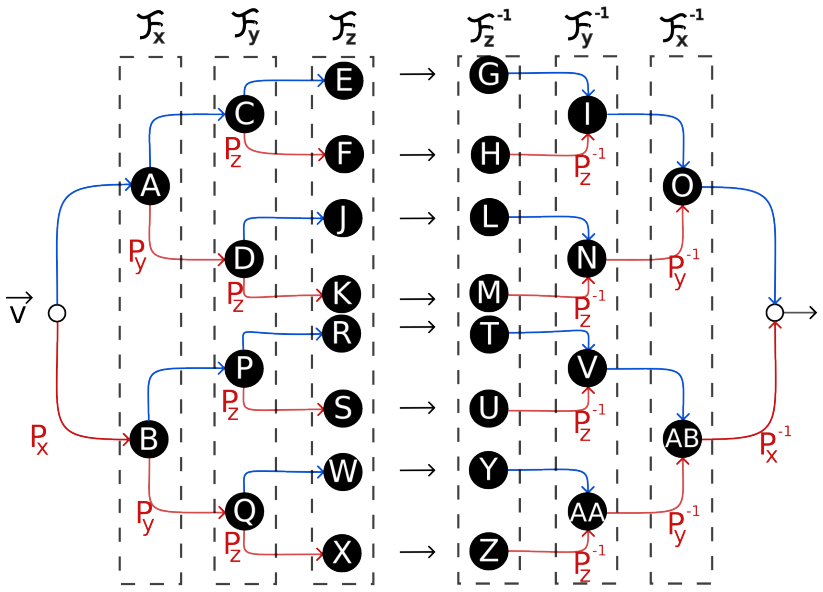

Figure 5.3 Tree-branch figure with labeled node such that the order of the splitting and projection in case of a single computational unit follows the alphabetical order.

could run the two branches simultaneously on a pair of GPUs and compute the overall result with a single communication step, merging the O and AB nodes. Utilizing this strategy effectively cuts the "wait" time experienced by original parent branch in half. However, this basic strategy introduces data redundancy as each GPU must still hold a peak of $d+1$ vectors in memory. For a two GPU parallelization in 3D, the peak memory would be reached before first recombination at each GPU so a total of ds coefficients each for a total of $2ds$ coefficients. The storage required for the vector containing the matrix data is not impacted. Another set of GPUs can be implemented for extra computational speed as long as there are dimensions, i.e splits in the tree-branch to parallelize. Hence, for maximum computational speed, one could use up to 2^d GPUs to parallelize all the multiplication steps on each GPU so that each GPU run its own split up multiplication (at E, F, J, K, R, S, W, X in 3D in Figure 5.3).

To generalize, for a d-dimensional Toeplitz matrix-vector multiplication, it is possible to use g GPUs, with $g \in \{2^k, k \in [\![0, d]\!]\}$, to speed up the computation at the cost of memory capacity. The system would then need to store $g(d+2-\log_2(g))s$ coefficients for a vector of length $s = n^d$. However, these parallelizations strategies are quite naive. More advanced and better suited parallelization strategies could be explored in future works.

CHAPTER 6 CONCLUSION

6.1 Summary of works

Modelling translationally invariant physics (e.g. electromagnetism) in a spatial basis generally leads to systems of equations involving multi-level block-Toeplitz matrices, which typically require iterative solution methods (section 3.1), and hence the computation of many block-Toeplitz matrix-vector products. As such, accelerating block-Toeplitz matrix-vector products is crucial for efficiently simulating a range of large-scale physics. While several methods presently exist for performing such multiplications, it is almost certain that they are sub-optimal. Specifically, the most promising approaches of circulant embedding (section 1.2) and tensor train decomposition (section 2.2) scale poorly with either system size or tensor rank. For a general source vector, computing the tensor decomposition is more expensive than simply performing multiplication. When extended to d-dimensional block Toeplitz matrices, the embedded vector contains a ratio of $\frac{2^d 1}{2^d}$ redundant coefficients, resulting in memory overload and time complexity inefficiency, as some of the Fourier transforms can be computed on fewer of these redundant coefficients.

The main idea of the algorithm presented in this thesis is to mitigate the later issue via lazy embeddings and eager projections, meaning that embeddings are postponed as late as possible, and projections are done as early as possible. These alterations allow for the required Fourier transforms to be computed on a reduced number of coefficients. For three-dimensional matrix-vector multiplication, the theoretical complexity ratio of the standard circulant embedding over the enhanced version tends to 12/7 as the sizes of the block Toeplitz matrices increase to infinity at each level. Moreover, the algorithm leverages a property of the fast Fourier transform (FFT) to split the transformed vector into even and odd coefficients, following a tree branch structure (Figure 4.2), which offers additional tunability in managing memory capacity and computational speed through branches parallelization on different GPUs. By running branches independently on different GPUs, one can gain computational speed at the cost of increased memory usage (section 5.3). When running on a single GPU, the memory usage ratio between the standard and split circulant embedding methods tends to 4/3 for a 3-dimensional matrix and vector as the sizes at each level tend to infinity. Whenever the Toeplitz data is completely symmetric or anti-symmetric—meaning that each of its block at each level is a Toeplitz symmetric or anti-symmetric matrix—then a clever embedding can reduce its memory storage. The

theoretical memory ratio between the two methods then becomes $\frac{2^d+1}{d+2}$. This situation occurs in the physical context of electromagnetic wave due to the reciprocity property between sources and receivers. This algorithm is still being used in the *GilaElectromagnetics.jl* package to accelerate simulations of electromagnetic waves in scattering media (see `https://github.com/moleskySean/GilaElectromagnetics.jl/tree/main`).

The measured time complexity ratio between the two versions of the circulant embedding method demonstrates better performance for the split circulant embedding method than predicted by theoretical complexity analysis, although it eventually converges to the theoretical ratio as the computational machine's memory limit is reached (Figure 4.3). Further analysis of the consecutive computational times of the split circulant embedding method revealed that some discrepancies were due to the switching method used by the "FFTW.jl" package for FFT computations as matrix and vector sizes vary (section 5.1).

6.2 Limitations

Despite the improvements over the standard circulant embedding for laptop-scale Toeplitz matrix-vector multiplications, the results remain erratic with a wide standard deviation for each statistical result, making the computational time of the method unpredictable. To improve predictability, it is essential to properly characterize the observed discrepancies between theoretical complexity and computational time. Rewriting the algorithm in a different programming language and using alternative packages could help identify the causes of these discrepancies.

6.3 Future research

The analysis of the alternative method based on tensor train decomposition suggests that further memory capacity and time complexity gains may be possible for highly structured input source vectors. It is not unreasonable to guess that such situations actually do occur somewhat frequently in electromagnetic field simulations. From this vantage, the identification of physical conditions that lead to strongly correlated tensor data, and the development of methods to obtain the TT-decomposition of such data without explicit computation, stand as interesting directions for further study. Achieving such an efficient method is essential for fully leveraging tensor train decomposition in wave scattering simulations. Finally, time measurement on different setup of GPU parallelization can be performed to determine the best

parallelization strategy according to the number of dimensions and the sizes of each block. One promising strategy would be to create additional artificial divisions of the data along the outermost data dimension following the FFT algorithm (see section 1.1). Indeed, using block matrix inversion, the Fourier transform can be written as

$$\mathbf{\Omega} f = \begin{bmatrix} \mathbf{\Omega} & \mathbf{0} \\ \mathbf{0} & \mathbf{\Omega} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{D} \\ \mathbf{D}^{-1} & -\mathbf{I} \end{bmatrix} \mathbf{P} f$$

where $\mathbf{\Omega}$ is the (variable size) Fourier transform, $\mathbf{D}$ is a diagonal matrix, and $\mathbf{P}$ is the permutation matrix grouping the even and odd indices of a vector. One could apply this division on outside of the current split circulant embedding algorithm to break the work that needs to be done into independent streams. That is, the input vector can be formatted to enable each GPU to work independently on a sub-collection of indices. However, this would require additional pre-processing and post-processing of the data to be implemented.

REFERENCES

[1] J. Haupt *et al.*, "Toeplitz compressed sensing matrices with applications to sparse channel estimation," *IEEE transactions on information theory*, vol. 56, no. 11, pp. 5862–5875, 2010.

[2] S. Pal, K. S. Pandian, and K. C. Ray, "Fpga implementation of stream cipher using toeplitz hash function," in *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, 2014, pp. 1834–1838.

[3] P. A. Voois, "A theorem on the asymptotic eigenvalue distribution of toeplitz-block-toeplitz matrices," *IEEE transactions on signal processing*, vol. 44, no. 7, pp. 1837–1841, 1996.

[4] X. Antoine, C. Chniti, and K. Ramdani, "On the numerical approximation of high-frequency acoustic multiple scattering problems by circular cylinders," *Journal of Computational Physics*, vol. 227, no. 3, pp. 1754–1771, 2008.

[5] M. Karimi, P. Croaker, and N. Kessissoglou, "Boundary element solution for periodic acoustic problems," *Journal of Sound and vibration*, vol. 360, pp. 129–139, 2016.

[6] B. E. Barrowes, F. L. Teixeira, and J. A. Kong, "Fast algorithm for matrix–vector multiply of asymmetric multilevel block-toeplitz matrices in 3-d scattering," *Microwave and Optical technology letters*, vol. 31, no. 1, pp. 28–32, 2001.

[7] S. Goreinov, D. Savostyanov, and E. Tyrtyshnikov, "Tensor and toeplitz structures applied to direct and inverse 3d electromagnetic problems," *Progress In Electromagnetics Research*, vol. 1897, 2009.

[8] M. Oudin and J. P. Delmas, "Asymptotic generalized eigenvalue distribution of block multilevel toeplitz matrices," *IEEE Transactions on Signal Processing*, vol. 57, no. 1, pp. 382–387, 2008.

[9] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of computation*, vol. 19, no. 90, pp. 297–301, 1965.

[10] I. J. Good, "The interaction algorithm and practical fourier analysis," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 20, no. 2, pp. 361–372, 1958.

[11] P. J. Ferreira and M. E. Domínguez, "Trading-off matrix size and matrix structure: Handling toeplitz equations by embedding on a larger circulant set," *Digital Signal Processing*, vol. 20, no. 6, pp. 1711–1722, 2010.

[12] A. Cariow and M. Gliszczyński, "Fast algorithms to compute matrix-vector products for toeplitz and hankel matrices," *Electrical Review*, vol. 88, no. 8, pp. 166–171, 2012.

[13] A. Dembo, C. L. Mallows, and L. A. Shepp, "Embedding nonnegative definite toeplitz matrices in nonnegative definite circulant matrices, with application to covariance estimation," *IEEE Transactions on Information Theory*, vol. 35, no. 6, pp. 1206–1212, 1989.

[14] D. Lee, "Fast multiplication of a recursive block toeplitz matrix by a vector and its application," *Journal of Complexity*, vol. 2, no. 4, pp. 295–305, 1986.

[15] N. Nikolski, *Toeplitz matrices and operators*. Cambridge University Press, 2020, vol. 182.

[16] O. Toeplitz, "Zur theorie der quadratischen und bilinearen formen von unendlichvielen veränderlichen: I. teil: Theorie der l-formen," *Mathematische Annalen*, vol. 70, pp. 351–376, 1911.

[17] V. I. Korobov and G. M. Sklyar, "Time-optimal control and the trigonometric moment problem," *Mathematics of the USSR-Izvestiya*, vol. 35, no. 1, p. 203, 1990.

[18] T. Moir, "Toeplitz matrices for lti systems, an illustration of their application to wiener filters and estimators," *International Journal of Systems Science*, vol. 49, no. 4, pp. 800–817, 2018.

[19] H. Bart, I. Gohberg, and M. Kaashoek, "Wiener-hopf integral equations, toeplitz matrices and linear systems," in *Toeplitz Centennial: Toeplitz Memorial Conference in Operator Theory, Dedicated to the 100th Anniversary of the Birth of Otto Toeplitz, Tel Aviv, May 11–15, 1981*. Springer, 1982, pp. 85–135.

[20] M. T. Heideman, D. H. Johnson, and C. S. Burrus, "Gauss and the history of the fast fourier transform," *Archive for history of exact sciences*, pp. 265–277, 1985.

[21] I. J. Good, "The interaction algorithm and practical fourier analysis," *Journal of the Royal Statistical Society Series B: Statistical Methodology*, vol. 20, no. 2, pp. 361–372, 1958.

[22] G. C. Danielson and C. Lanczos, "Some improvements in practical fourier analysis and their application to x-ray scattering from liquids," *Journal of the Franklin Institute*, vol. 233, no. 5, pp. 435–452, 1942.

[23] C. Runge, "Über die zerlegung einer empirischen funktion in sinuswellen," *Z. Math. Phys.*, vol. 52, pp. 117–123, 1905.

[24] H. H. Goldstine, *A History of Numerical Analysis from the 16th through the 19th Century*. Springer Science & Business Media, 2012, vol. 2.

[25] C. F. Gauss, "Nachlass: Theoria interpolationis methodo nova tractata," *Carl Friedrich Gauss Werke*, vol. 3, pp. 265–327, 1866.

[26] J. Markel, "Fft pruning," *IEEE transactions on Audio and Electroacoustics*, vol. 19, no. 4, pp. 305–311, 1971.

[27] H. Burkhardt, *Trigonometrishe interpolation*. Teubner, 1899-1916, vol. Encyklopädie der Mathematishen Wissenshaften, vol. 2.

[28] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review, Volume 51, Issue 3, page 455-500*, 2008.

[29] D. O. Vervliert, Nico and L. De Lathauwer, "Exploiting efficient representations in large-scale tensor decompositions," *SIAM J. SCI. COMPUT. Vol. 41, No. 2, pp. A789–A815*, 2019.

[30] A. Rodomanov, "Introduction to the tensor train decomposition and its applications in machine learning," seminar on Applied Linear Algebra, *Higher School of Economics*, Moscow, Russia, 14 March 2016.

[31] K. B. N. Kazeev, Vladimir A. and E. E. Tyrtyshnikov, "Multilevel toeplitz matrices generated by tensor-structured vectors and convolution with logarithmic complexity," *SIAM Journal on Scientific Computing Volume 35, Issue 3, pages A1511 - A1536*, 2013.

[32] L. Tucker, "Implications of factor analysis of three-way matrices for measurement of change," *Problems in Measuring Change, vol. 15, pp.122-137*, 1963.

[33] I. V. Oseledets, "Tensor-train decomposition," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2295–2317, 2011.

[34] S. Dolgov, B. Khoromskij, and D. Savostyanov, "Multidimensional fourier transform in logarithmic complexity using qtt approximation," *Preprint*, vol. 18, 2011.

[35] F. Frezza, F. Mangini, and N. Tedeschi, "Introduction to electromagnetic scattering: tutorial," *J. Opt. Soc. Am. A*, vol. 35, no. 1, pp. 163–173, Jan 2018. [Online]. Available: https://opg.optica.org/josaa/abstract.cfm?URI=josaa-35-1-163

[36] L. D. Landau and E. M. Lifshitz, *Statistical Physics: Volume 5*. Elsevier, 2013, vol. 5.

[37] R. Bocker and B. Frieden, "Solution of the maxwell field equations in vacuum for arbitrary charge and current distributions using the methods of matrix algebra," *IEEE Transactions on Education*, vol. 36, no. 4, pp. 350–356, 1993.

[38] A. Bayliss, M. Gunzburger, and E. Turkel, "Boundary conditions for the numerical solution of elliptic equations in exterior regions," *SIAM Journal on Applied Mathematics*, vol. 42, no. 2, pp. 430–451, 1982.

[39] G. Makov and M. C. Payne, "Periodic boundary conditions in ab initio calculations," *Physical Review B*, vol. 51, no. 7, p. 4014, 1995.

[40] D. Madier, *Practical finite element analysis for mechanical engineers*. FEA Academy, 2020, vol. 147.

[41] A. G. Polimeridis *et al.*, "Directfn: Fully numerical algorithms for high precision computation of singular integrals in galerkin sie methods," *IEEE Transactions on Antennas and Propagation*, vol. 61, no. 6, pp. 3112–3122, 2013.