SPARSE FACTORIZATIONS AND SCALABLE ALGORITHMS FOR DIFFERENTIAL AND INTEGRAL OPERATORS

A DISSERTATION
SUBMITTED TO THE INSTITUTE FOR COMPUTATIONAL &
MATHEMATICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Yingzhou Li

May 2017

This dissertation is online at: http://purl.stanford.edu/dm425vb5993

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Lexing Ying, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Leonid Ryzhik

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Chao Yang

Approved for the Stanford University Committee on Graduate Studies.

Patricia J. Gumport, Vice Provost for Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format. An original signed hard copy of the signature page is on file in University Archives.

Preface

This dissertation presents sparse factorizations and scalable algorithms for elliptic differential operators and Fourier integral operators (FIOs). The former operators are solved by the distributed-memory hierarchical interpolative factorization (DHIF) whereas the later operators are addressed by the butterfly factorization.

By exploiting locality and certain low-rank properties of the elliptic differential operators, the hierarchical interpolative factorization achieves quasi-linear complexity for factorizing the discrete positive definite elliptic differential operator and linear complexity for solving the associated linear system. In this dissertation, the DHIF is introduced as a scalable and distributed-memory implementation of the hierarchical interpolative factorization. The DHIF organizes the processes in a hierarchical structure and keep the communication as local as possible. The computation complexity is $O\left(\frac{N \log N}{P}\right)$ and $O\left(\frac{N}{P}\right)$ for constructing and applying the DHIF, respectively, where N is the size of the problem and P is the number of processes. The communication complexity is $O\left(\sqrt{P} \log^3 P\right) \alpha + O\left(\frac{N^{2/3}}{\sqrt{P}}\right) \beta$ where α is the latency and β is the inverse bandwidth. Extensive numerical examples are performed on the NERSC Edison system with up to 8192 processes. The numerical results agree with the complexity analysis and demonstrate the efficiency and scalability of the DHIF.

The butterfly factorization is a data-sparse approximation for the FIOs as well

as other matrices that satisfy a complementary low-rank property. The factorization can be constructed efficiently if either fast algorithms for applying the matrix and its adjoint are available or the entries of the matrix can be sampled individually. For an $N \times N$ matrix, the resulting factorization is a product of $O(\log N)$ sparse matrices, each with $O(N)$ non-zero entries. Hence, it can be applied rapidly in $O(N \log N)$ operations. Numerical results are provided to demonstrate the effectiveness of the butterfly factorization and its construction algorithms. For the kernel matrices of multidimensional FIOs, for which the complementary low-rank property is usually not satisfied due to a singularity at the origin, we extend this factorization by combining it with either a polar coordinate transformation or a multiscale decomposition of the integration domain to overcome the singularity. Numerical results are provided to demonstrate the efficiency of the proposed algorithms as well.

Acknowledgement

This dissertation would not have been possible without the help and support from many people.

I would like to thank my adviser Lexing Ying for his guidance and encouragement. In the past five years, he teaches me how to solve problems, how to ask questions and how to think as an applied mathematician. His thoughts have deeply influenced both my academic life and my everyday life.

I am grateful to Eric Darve, Lenya Ryzhik, and Chao Yang for serving on my thesis committee and for providing valuable discussion and suggestion.

I would like to thank my collaborators Jack Poulson and Haizhao Yang for being great teachers and constant supporters. I would also like to thank my other collaborators: Kenneth Ho, Lin Lin, Michael Mahoney, Eileen Martin, Victor Minden, Ruoxi Wang, and Liming Zhang. Without their help, my achievements would not be possible.

Through all the past five years, the faculty and staff of the institute for computational & mathematical engineering provide a great environment for me to do my research. I also wish to thank the people I have worked with at LBNL for making a wonderful place to work.

Finally, I want to thank my parents, Zhigang Li and Xiaohua Zhou, for their love,

care and encouragement. Without them, I would be nothing.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1) Many physical models are described by linear partial differential equations and/or integral equations.

2) One of the important problems in scientific computing and applied mathematics is to develop efficient and scalable algorithms for differential and integral operators.

Most computational problems from real applications cannot be evaluated or solved explicitly. Numerical methods provide a powerful hammer to tackle these practical problems.

Differential operators are usually discretized by local schemes, for instance, finite difference method and finite element method. The resulting equation is a sparse linear system,

$$Au = f,$$

where A is a sparse matrix of size $N \times N$ with $O(N)$ non-zeros, u and f are vectors of length N, and N is the total number of discretization points. Due to the sparsity, the forward application of the matrix can be efficiently calculated in $O(N)$ operations.

However, solving the sparse linear system naïvely costs $O(N^3)$ operations, which is intractable for large N. Even if the sparse matrix is carefully permuted, solving it for three-dimensional problems still costs $O(N^2)$ operations [33]. Therefore, fast and scalable algorithms are required for solving these large scale sparse matrices.

On the other hand side, integral operator is usually discretized via Nyström method, collocation method, and Galerkin method. The resulting equation is a dense linear system,

$$Ku = f,$$

where K is a dense kernel matrix of size $N \times N$, u and f are vectors of length N, and N is the total number of discretization points. Applying a dense matrix of size N to a vector costs $O(N^2)$ operations, whereas solving for u costs $O(N^3)$ operations. In this case, both applying the matrix and solving the linear system need to be accelerated.

In this dissertation, we consider two specific classes of operators, elliptic partial differential operators with rough coefficients and Fourier integral operators.

Rough coefficient elliptic partial differential operators

Rough coefficient elliptic partial differential operators are of the form

$$-\nabla \cdot (a(x)\nabla u(x)) + b(x)u(x) = f(x), \quad x \in \Omega \subset \mathbb{R}^d \tag{1.1}$$

with appropriate boundary conditions on $\partial\Omega$, where $u(x)$ is the unknown function, $a(x) > 0, b(x), f(x)$ are given functions. Such equations are of fundamental importance in science and engineering and encompass many PDEs from classical physics, e.g., the Laplace equations, the Stockes equations, and the low-frequency time-harmonic Helmholtz and Maxwell equations, etc. Solving the associated sparse linear systems

efficiently would have significant impact in practice. Hierarchical interpolative factorization (HIF) proposed in [50] is an efficient and accurate way to factorize this linear system. Chapter 2 proposes a distributed-memory HIF implementation to address this issue. The sparse matrix A can be represented as a product of a sequence of sparse lower or upper triangular matrices. The inversion of any of these sparse triangular matrices has the same sparsity pattern as the matrix itself. Therefore, the inverse of A is representable as a reverse order product of the inverses of these sparse lower or upper triangular matrices. Solving the linear system $Au = f$ can be calculated via a sequence of sparse matrix vector multiplications, and costs $O(N \log N)$ operations. As many of these sparse matrices can be generated, applied, and inverted independently and simultaneously, our implementation achieves almost perfect parallelization: given P processes, each process runs in $O(\frac{N}{P} \log N)$ operations in the algorithm.

Fourier integral operators

Fourier integral operators (FIOs) are defined as

$$u(x) = \int_{\mathbb{R}^d} a(x,\xi) e^{2\pi\imath\Phi(x,\xi)} \widehat{f}(\xi)\, \mathrm{d}\xi, \tag{1.2}$$

where $a(x,\xi)$ is an amplitude function, $\Phi(x,\xi)$ is a phase function that is smooth in (x,ξ) for $\xi \neq 0$ and obeys the homogeneity condition of degree 1 in ξ, $\widehat{f}(\xi)$ is the Fourier transform of the input $f(x)$. An especially important example of an FIO is the solution operator to the scalar wave equation with variable but smooth sound speeds. For small times, the solution operator is a sum of two FIOs with smooth phases and amplitudes. Since the FIOs are dense, their forward applications require fast and scalable algorithms. Chapter 3 proposes an efficient multiscale butterfly algorithm with low pre-factor. Chapter 4 first proposes a sparse factorization for the

operator for one-dimensional problems. In the same chapter, combining techniques developed in Chapter 3, we obtain three multidimensional sparse factorizations to accelerate the application of FIOs in multidimensional problems. The key in all butterfly factorizations is to factorize the kernel matrix K as a product of $O(\log N)$ sparse matrices, each of which contains $O(N)$ non-zeros, where N is the size of K. Once the factorization is available, the application of K costs $O(N \log N)$ operations. The application of each sparse matrix can be fully parallelized. Communications are needed for redistributing the vector according to the sparsity patterns. Due to the special block pattern in the sparse matrices, each process communicates with a constant number of other processes in the redistribution.

Chapter 2

Elliptic PDEs and hierarchical interpolative factorization

2.1 Background

This chapter proposes an efficient distributed-memory algorithm for solving elliptic partial differential equations (PDEs) of the form,

$$-\nabla \cdot (a(x)\nabla u(x)) + b(x)u(x) = f(x) \quad x \in \Omega \subset \mathbb{R}^3, \tag{2.1}$$

with a certain boundary condition, where $a(x) > 0$, $b(x)$ and $f(x)$ are given functions and $u(x)$ is an unknown function. Since this elliptic equation is of fundamental importance to several problems in physical sciences, solving (2.1) effectively has a significant impact in practice. Discretizing this with local schemes such as the finite difference or finite element methods leads to a sparse linear system,

$$Au = f, \tag{2.2}$$

where $A \in \mathbb{R}^{N\times N}$ is a sparse symmetric matrix with $O(N)$ non-zero entries with N being the number of the discretization points, and u and f are the discrete approximations of the functions $u(x)$ and $f(x)$, respectively. For many practical applications, one often needs to solve (2.1) on a sufficient fine mesh for which N can be very large, especially for three-dimensional (3D) problems. Hence, there is a practical need for developing fast and parallel algorithms for the efficient solution of (2.1).

2.1.1 Related work

A great deal of effort in the field of scientific computing has been devoted to the efficient solution of (2.2). Beyond the $O(N^3)$ complexity naïve matrix inversion approach, one can classify the existing fast algorithms into the following groups.

The first one consists of the sparse direct algorithms, which take advantage of the sparsity of the discrete problem. The most noticeable example in this group is the nested dissection multifrontal method (MF) method [38, 33, 66]. By carefully exploring the sparsity and the locality of the problem, the multifrontal method factorizes the matrix A (and thus A^{-1}) as a product of sparse lower and upper triangular matrices. For 3D problems, the factorization step costs $O(N^2)$ operations while the application step takes $O(N^{4/3})$ operations. Many parallel implementations [3, 4, 77] of the multifrontal method were proposed and they typically work quite well for problem of moderate size. However, as the problem size goes beyond a couple of millions, most implementations, including the distributed-memory ones, hit severe bottlenecks in memory consumption. Another closely related method is the supernodal method [32, 42]. Asymptotically, it behaves similar as the multifrontal method.

The second group consists of iterative solvers [14, 79, 80, 37], including famous algorithms such as the conjugate gradient (CG) method and the multigrid method.

Each iteration of these algorithms typically takes $O(N)$ steps and hence the overall cost for solving (2.2) is proportional to the number of iterations required for convergence. For problems with smooth coefficient functions $a(x)$ and $b(x)$, the number of iterations typically remains rather small and the optimal linear complexity is achieved. However, if the coefficient functions lack regularity or have high contrast, the iteration number typically grows quite rapidly as the problem size increases.

The third group contains the methods based on structured matrices [11, 10, 12, 22]. These methods, for example the $\mathcal{H}$-matrix [43, 45], the $\mathcal{H}^2$-matrix [44], and the hierarchically semi-separable matrix (HSS) [21, 93], are shown to have efficient approximations of linear or quasi-linear complexity for the matrices A and A^{-1}. As a result, the algebraic operations of these matrices are of linear or quasi-linear complexities as well. More specifically, the recursive inversion and the rocket-style inversion [1] are two popular methods for the inverse operation. For distributed-memory implementations, however, the former lacks parallel scalability [55, 57] while the latter demonstrates scalability only for 1D and 2D problems [1]. For 3D problems, these methods typically suffer from large prefactors that make them less efficient for practical large-scale problems.

A recent group of methods explore the idea of integrating the MF method with the hierarchical matrix [69, 92, 91, 90, 39, 88, 47] or block low-rank matrix [81, 82, 2, 87] approach in order to leverage the efficiency of both methods. Instead of directly applying the hierarchical matrix structure to the 3D problems, these methods apply it to the representation of the frontal matrices (i.e., the interactions between the lower dimensional fronts). These methods are of linear or quasi-linear complexities in theory with much smaller prefactors. However, due to the combined complexity, the implementation is highly non-trivial and quite difficult for parallelization [67, 94].

More recently, the hierarchical interpolative factorization (HIF) [50, 51, 73, 62] is proposed as a new way for solving elliptic PDEs and integral equations. As compared to the multifrontal method, the HIF includes an extra step of skeletonizing the fronts in order to reduce the size of the dense frontal matrices. Based on the key observation that the number of skeleton points on each front scales linearly as the one-dimensional fronts, the HIF factorizes the matrix A (and thus A^{-1}) as a product of sparse matrices that contains only $O(N)$ non-zero entries in total. In addition, the factorization and application of the HIF are of complexities $O(N \log N)$ and $O(N)$, respectively, for N being the total number of degrees of freedom (DOFs) in (2.2). In practice, the HIF shows significant saving in terms of computational resources required for 3D problems.

2.1.2 Contribution

This chapter proposes the first *distributed-memory hierarchical interpolative factorization* (DHIF) for solving very large scale problems. In a nutshell, the DHIF organizes the processes in an octree structure in the same way that the HIF partitions the computation domain. In the simplest setting, each leaf node of the computation domain is assigned a single process. Thanks to the locality of the operator in (2.1), this process only communicates with its neighbors and all algebraic computations are local within the leaf node. At higher levels, each node of the computation domain is associated with a process group that contains all processes in the subtree starting from this node. The computations are all local within this process group via parallel dense linear algebra and the communications are carried out between neighboring process groups. By following this octree structure, we make sure that both the communication and computations in the distributed-memory HIF are evenly distributed. As a result,

the distributed-memory HIF implementation achieves $O\left(\frac{N \log N}{P}\right)$ and $O\left(\frac{N}{P}\right)$ parallel complexity for constructing and applying the factorization, respectively, where N is the number of DOFs and P is the number of processes.

We have performed extensive numerical tests. The numerical results support the complexity analysis of the distributed-memory HIF and suggest that the DHIF is a scalable method up to thousands of processes and can be applied to solve large scale elliptic PDEs.

2.1.3 Organization

The rest of this chapter is organized as follow. In Section 2.2, we introduce the basic tools needed, and review the sequential HIF. Section 2.3 presents the DHIF as a parallel extension of the sequential HIF for 3D problems. Complexity analyses for memory usage, computation time and communication volume are given at the end of this section. The numerical results detailed in Section 2.4 show that the DHIF is applicable to large scale problems and achieves parallel scalability up to thousands of processes. Finally, Section 2.5 concludes with some extra discussions on future work.

2.2 Preliminaries

This section reviews the basic tools and the sequential HIF. First, we start with the notations that are widely used throughout this chapter.

2.2.1 Notations

In this chapter, we adopt MATLAB notations for simple representation of submatrices. For example, given a matrix A and two index sets, s_1 and s_2, $A(s_1, s_2)$ represents

the submatrix of A with the row indices in s_1 and column indices in s_2. The next two examples explore the usage of MATLAB notation ":". With the same settings, $A(s_1, :)$ represents the submatrix of A with row indices in s_1 and all columns. Another usage of notation ":" is to create regularly spaced vectors for integer values i and j, for instance, $i : j$ is the same as $[i, i + 1, i + 2, \ldots, j]$ for $i \leqslant j$.

In order to simplify the presentation, we consider the problem (2.1) with periodic boundary condition and assume that the domain $\Omega = [0, 1)^3$, and is discretized with a grid of size $n \times n \times n$ for $n = 2^L m$, where $L = O(\log n)$ and $m = O(1)$ are both integers. In the rest of this chapter, $L + 1$ is known as the number of levels in the hierarchical structure and L is the level number of the root level. We use $N = n^3$ to denote the total number of DOFs, which is the dimension of the sparse matrix A in (2.2). Furthermore, each grid point $\mathbf{x_j}$ is defined as

$$\mathbf{x_j} = h\mathbf{j} = h(j_1, j_2, j_3) \tag{2.3}$$

where $h = 1/n$, $\mathbf{j} = (j_1, j_2, j_3)$ and $0 \leqslant j_1, j_2, j_3 < n$.

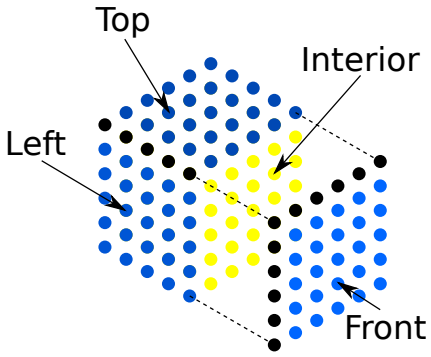

Figure 2.1: Cell structure: top, front, left, and interior points are indicated by arrows; bottom, back, and right points are not plotted in the figure; the black dots denote the edge points; the dash line indicates that the front frame is pulled away in order to show the interior points.

In order to fully explore the hierarchical structure of the problem, we recursively bipartite each dimension of the grid into $L+1$ levels. Let the leaf level be level 0 and the root level be level L. At level ℓ, a cell indexed with $\mathbf{j}$ is of size $m2^\ell \times m2^\ell \times m2^\ell$ and each point in the cell is in the range, $\big[m2^\ell j_1 + (0 : m2^\ell - 1)\big] \times \big[m2^\ell j_2 + (0 : m2^\ell - 1)\big] \times \big[m2^\ell j_3 + (0 : m2^\ell - 1)\big]$, for $\mathbf{j} = (j_1, j_2, j_3)$ and $0 \leqslant j_1, j_2, j_3 < 2^{L-\ell}$. $C_{\mathbf{j}}^\ell$ denotes the grid point set of the cell at level ℓ indexed with $\mathbf{j}$.

A cell $C_{\mathbf{j}}^\ell$ owns three faces: top, front, and left. Each of these three faces contains the grid points on the first frame in the corresponding direction. For example, the front face contains the grid points in $\big[m2^\ell j_1 + (0 : m2^\ell - 1)\big] \times \big[m2^\ell j_2\big] \times \big[m2^\ell j_3 + (0 : m2^\ell - 1)\big]$. Besides these three in-cell faces (top, front, and left) that are owned by a cell, each cell is also adjacent to three out-of-cell faces (bottom, back, right) owned by its neighbors. Each of these three faces contains the grid points on the next to the last frame in the corresponding dimension. As a result, these faces contain DOFs that belong to adjacent cells. For example, the bottom face of $C_{\mathbf{j}}^\ell$ contains the grid points in $\big[m2^\ell(j_1 + 1)\big] \times \big[m2^\ell j_2 + (0 : m2^\ell - 1)\big] \times \big[m2^\ell j_3 + (0 : m2^\ell - 1)\big]$. These six faces are the surrounding faces of $C_{\mathbf{j}}^\ell$. One also defines the interior of $C_{\mathbf{j}}^\ell$ to be $I_{\mathbf{j}}^\ell = \big[m2^\ell j_1 + (1 : m2^\ell - 1)\big] \times \big[m2^\ell j_2 + (1 : m2^\ell - 1)\big] \times \big[m2^\ell j_3 + (1 : m2^\ell - 1)\big]$ for the same $\mathbf{j} = (j_1, j_2, j_3)$ and $0 \leqslant j_1, j_2, j_3 < 2^{L-\ell}$. Figure 2.1 gives an illustration of a cell, its faces, and its interior. These definitions and notations are summarized in Table 2.1. Also included here are some notations used for the processes, which will be introduced later.

2.2.2 Sparse elimination

Suppose that A is a symmetric matrix. The row/column indices of A are partitioned into three sets $I \bigcup F \bigcup R$ where I refers to the interior point set, F refers to the

Notation	Description
n	Number of points in each dimension of the grid
N	Number of points in the grid
h	Grid gap size
ℓ	Level number in the hierarchical structure
L	Level number of the root level in the hierarchical structure
$\mathbf{e}_1$, $\mathbf{e}_2$, $\mathbf{e}_3$	Unit vector along each dimension
$\mathbf{0}$	Zero vector
$\mathbf{j}$	Triplet index $\mathbf{j} = (j_1, j_2, j_3)$
$\mathbf{x_j}$	Point on the grid indexed with $\mathbf{j}$
Ω	The set of all points on the grid
$C^\ell_\mathbf{j}$	Cell at level ℓ with index $\mathbf{j}$
$\mathcal{C}^\ell$	$\mathcal{C}^\ell = \{C^\ell_\mathbf{j}\}_\mathbf{j}$ is the set of all cells at level ℓ
$\mathcal{F}^\ell_\mathbf{j}$	Set of all surrounding faces of cell $C^\ell_\mathbf{j}$
$\mathcal{F}^\ell$	Set of all faces at level ℓ
$I^\ell_\mathbf{j}$	Interior of $C^\ell_\mathbf{j}$
$\mathcal{I}^\ell$	$\mathcal{I}^\ell = \{I^\ell_\mathbf{j}\}_\mathbf{j}$ is the set of all interiors at level ℓ
Σ^ℓ	The set of active DOFs at level ℓ
$\Sigma^\ell_\mathbf{j}$	The set of active DOFs at level ℓ with process group index $\mathbf{j}$
$p^\ell_\mathbf{j}$, p^ℓ	The process group at level ℓ with/without index $\mathbf{j}$

Table 2.1: Commonly used notations

surrounding face point set, and R refers to the rest point set. We further assume that there is no interaction between the indices in I and the ones in R. As a result, one can write A in the following form

$$A = \begin{bmatrix} A_{II} & A_{FI}^T & \\ A_{FI} & A_{FF} & A_{RF}^T \\ & A_{RF} & A_{RR} \end{bmatrix}. \tag{2.4}$$

Let the LDL^T decomposition of A_{II} be $A_{II} = L_I D_I L_I^T$, where L_I is lower triangular matrix with unit diagonal. According to the block Gaussian elimination of A

given by (2.4), one defines the *sparse elimination* to be

$$S_I^T A S_I = \begin{bmatrix} D_I & & \\ & B_{FF} & A_{RF}^T \\ & A_{RF} & A_{RR} \end{bmatrix}, \tag{2.5}$$

where $B_{FF} = A_{FF} - A_{FI} A_{II}^{-1} A_{FI}^T$ is the associated Schur complement and the explicit expressions for S_I is

$$S_I = \begin{bmatrix} L_I^{-T} & -A_{II}^{-1} A_{FI}^T & \\ & I & \\ & & I \end{bmatrix}. \tag{2.6}$$

The sparse elimination removes the interaction between the interior points I and the corresponding surrounding face points F and leaves A_{RF} and A_{RR} untouched. We call the entire point set, $I \bigcup F \bigcup R$, the *active point set*. Then, after the sparse elimination, the interior points are decoupled from other points, which is conceptually equivalent to eliminate the interior points from the active point set. After this, the new active point set can be regarded as $F \bigcup R$.

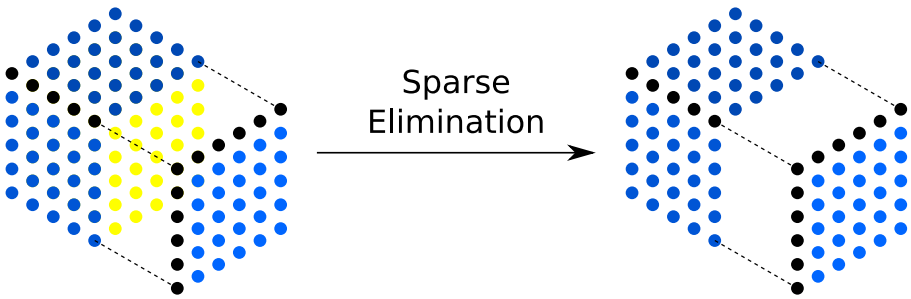

Figure 2.2: Sparse elimination: the interior points are eliminated after the sparse elimination; the rest points are not all plotted in the figure.

Figure 2.2 illustrates the impact of the sparse elimination. The dots in the figure represent the active points. Before the sparse elimination (left), edge points, face

points and interior points are active while after the sparse elimination (right) the interior points are eliminated from the active point set.

2.2.3 Skeletonization

Skeletonization is a tool for eliminating redundant point set from a symmetric matrix that has low-rank off-diagonal blocks. The key step in skeletonization uses the interpolative decomposition [23, 71] of low-rank matrices.

Let A be a symmetric matrix of the form,

$$A = \begin{bmatrix} A_{FF} & A_{RF}^T \\ A_{RF} & A_{RR} \end{bmatrix}, \tag{2.7}$$

where A_{RF} is a numerically low-rank matrix. The interpolative decomposition of A_{RF} is (up to a permutation)

$$A_{RF} = \begin{bmatrix} A_{R\bar{\bar{F}}} & A_{R\hat{F}} \end{bmatrix} \approx \begin{bmatrix} A_{R\hat{F}}T_F & A_{R\hat{F}} \end{bmatrix}, \tag{2.8}$$

where T_F is the interpolation matrix, $\widehat{F}$ is the skeleton point set, $\bar{\bar{F}}$ is the redundant point set, and $F = \widehat{F} \bigcup \bar{\bar{F}}$. Applying this approximation to A results

$$A \approx \left[\begin{array}{cc|c} A_{\bar{\bar{F}}\bar{\bar{F}}} & A_{\hat{F}\bar{\bar{F}}}^T & T_F^T A_{R\hat{F}}^T \\ A_{\hat{F}\bar{\bar{F}}} & A_{\hat{F}\hat{F}} & A_{R\hat{F}}^T \\ \hline A_{R\hat{F}}T_F & A_{R\hat{F}} & A_{RR} \end{array}\right], \tag{2.9}$$

and be symmetrically factorized as

$$S_{\bar{\bar{F}}}^T Q_F^T A Q_F S_{\bar{F}} \approx S_{\bar{\bar{F}}}^T \left[\begin{array}{cc|c} B_{\bar{\bar{F}}\bar{F}} & B_{\hat{F}\bar{\bar{F}}}^T & \\ B_{\hat{F}\bar{\bar{F}}} & A_{\hat{F}\hat{F}} & A_{R\hat{F}}^T \\ \hline & A_{R\hat{F}} & A_{RR} \end{array}\right] S_{\bar{\bar{F}}} = \left[\begin{array}{cc|c} D_{\bar{\bar{F}}} & & \\ & B_{\hat{F}\hat{F}} & A_{R\hat{F}}^T \\ \hline & A_{R\hat{F}} & A_{RR} \end{array}\right], \tag{2.10}$$

where

$$B_{\bar{\bar{F}}\bar{F}} = A_{\bar{\bar{F}}\bar{F}} - T_F^T A_{\hat{F}\bar{\bar{F}}} - A_{\hat{F}\bar{\bar{F}}}^T T_F + T_F^T A_{\hat{F}\hat{F}} T_F, \tag{2.11}$$

$$B_{\hat{F}\bar{\bar{F}}} = A_{\hat{F}\bar{\bar{F}}} - A_{\hat{F}\hat{F}} T_F, \tag{2.12}$$

$$B_{\hat{F}\hat{F}} = A_{\hat{F}\hat{F}} - B_{\hat{F}\bar{F}} B_{\bar{\bar{F}}\bar{F}}^{-1} B_{\hat{F}\bar{\bar{F}}}^T. \tag{2.13}$$

The factor Q_F is generated by the block Gaussian elimination, which is defined to be

$$Q_F = \left[\begin{array}{cc|c} I & & \\ -T_F & I & \\ \hline & & I \end{array}\right]. \tag{2.14}$$

Meanwhile, the factor $S_{\bar{\bar{F}}}$ is introduced in the sparse elimination:

$$S_{\bar{\bar{F}}} = \left[\begin{array}{cc|c} L_{\bar{\bar{F}}}^{-T} & -B_{\bar{\bar{F}}\bar{F}}^{-1} B_{\hat{F}\bar{\bar{F}}}^T & \\ & I & \\ \hline & & I \end{array}\right] \tag{2.15}$$

where $L_{\bar{\bar{F}}}$ and $D_{\bar{\bar{F}}}$ come from the LDL^T factorization of $B_{\bar{\bar{F}}\bar{F}}$, i.e., $B_{\bar{\bar{F}}\bar{F}} = L_{\bar{\bar{F}}} D_{\bar{\bar{F}}} L_{\bar{\bar{F}}}^T$. Similar to what happens in Section 2.2.2, the skeletonization eliminates the redundant point set $\bar{\bar{F}}$ from the active point set.

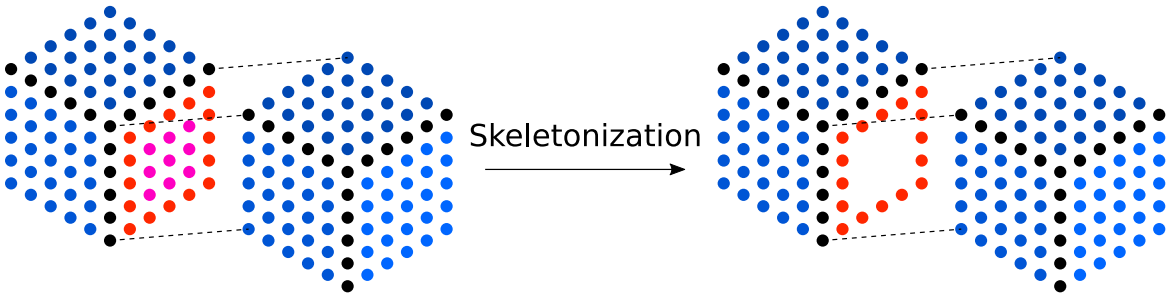

Figure 2.3: Skeletonization: the working face is colored by red and pink; red points are the skeleton points on the face whereas pink points are the redundant points on the face; skeletonization eliminates the redundant points from the active point set.

The point elimination idea of the skeletonization is illustrated in Figure 2.3. Before the skeletonization (left), the edge points, interior points, skeleton face points (red) and redundant face points (pink) are all active, while after the skeletonization (right) the redundant face points are eliminated from the active point set.

2.2.4 Sequential hierarchical interpolative factorization

This section reviews the sequential hierarchical interpolative factorization (HIF) for 3D elliptic problems (2.1) with the periodic boundary condition. Without loss of generality, we discretize (2.1) with the seven-point stencil on a uniform grid, which is defined in Section 2.2.1. The discrete system is

$$\begin{aligned} &\frac{1}{h^2}\left(a_{\mathbf{j}-\frac{1}{2}\mathbf{e}_1}+a_{\mathbf{j}+\frac{1}{2}\mathbf{e}_1}+a_{\mathbf{j}-\frac{1}{2}\mathbf{e}_2}+a_{\mathbf{j}+\frac{1}{2}\mathbf{e}_2}+a_{\mathbf{j}-\frac{1}{2}\mathbf{e}_3}+a_{\mathbf{j}+\frac{1}{2}\mathbf{e}_3}\right)u_{\mathbf{j}} \\ &-\frac{1}{h^2}\left(a_{\mathbf{j}-\frac{1}{2}\mathbf{e}_1}u_{\mathbf{j}-\mathbf{e}_1}+a_{\mathbf{j}+\frac{1}{2}\mathbf{e}_1}u_{\mathbf{j}+\mathbf{e}_1}+a_{\mathbf{j}-\frac{1}{2}\mathbf{e}_2}u_{\mathbf{j}-\mathbf{e}_2}+a_{\mathbf{j}+\frac{1}{2}\mathbf{e}_2}u_{\mathbf{j}+\mathbf{e}_2}\right. \\ &\qquad\left.+a_{\mathbf{j}-\frac{1}{2}\mathbf{e}_3}u_{\mathbf{j}-\mathbf{e}_3}+a_{\mathbf{j}+\frac{1}{2}\mathbf{e}_3}u_{\mathbf{j}+\mathbf{e}_3}\right)+b_{\mathbf{j}}u_{\mathbf{j}}=f_{\mathbf{j}} \end{aligned} \tag{2.16}$$

at each grid point $\mathbf{x}_{\mathbf{j}}$ for $\mathbf{j}=(j_1,j_2,j_3)$ and $0\leqslant j_1,j_2,j_3<n$, where $a_{\mathbf{j}}=a(\mathbf{x}_{\mathbf{j}})$, $b_{\mathbf{j}}=b(\mathbf{x}_{\mathbf{j}})$, $f_{\mathbf{j}}=f(\mathbf{x}_{\mathbf{j}})$, and $u_{\mathbf{j}}$ approximates the unknown function $u(x)$ at $\mathbf{x}_{\mathbf{j}}$. The

corresponding linear system is

$$Au = f \tag{2.17}$$

where A is a sparse symmetric matrix. Further if $b > 0$, A is SPD matrix.

We first introduce the notion of active and inactive DOFs.

- A set Σ of DOFs of A is called **active** if $A_{\Sigma\Sigma}$ is not a diagonal matrix or $A_{\bar{\Sigma}\Sigma}$ is a non-zero matrix;
- A set Σ of DOFs of A is called **inactive** if $A_{\Sigma\Sigma}$ is a diagonal matrix and $A_{\bar{\Sigma}\Sigma}$ is a zero matrix.

Here $\bar{\Sigma}$ refers to the complement of the set Σ. Sparse elimination and skeletonization provide concrete examples of active and inactive DOFs. For example, sparse elimination turns the indices I from active DOFs of A to inactive DOFs of $\widetilde{A} = S_I^T A S_I$ in (2.5). Skeletonization turns the indices $\bar{\bar{F}}$ from active DOFs of A to inactive DOFs of $\widetilde{A} = S_{\bar{\bar{F}}}^T Q_F^T A Q_F S_{\bar{F}}$ in (2.10).

With these notations, the sequential HIF in [50] is summarized as follows. A more illustrative representation of the sequential HIF is given on the left column of Figure 2.5.

- **Preliminary.** Let $A^0 = A$ be the sparse symmetric matrix in (2.17), Σ^0 be the initial active DOFs of A, which includes all indices.
- **Level ℓ for $\ell = 0, \ldots, L-1$.**
  - **Preliminary.** Let A^ℓ denote the matrix before any elimination at level ℓ. Σ^ℓ is the corresponding active DOFs. Let us recall the notations in Section 2.2.1. $C_{\mathbf{j}}^\ell$ denotes the active DOFs in the cell at level ℓ indexed

with $\mathbf{j}$. $\mathcal{F}_{\mathbf{j}}^{\ell}$ and $I_{\mathbf{j}}^{\ell}$ denote the surrounding faces and interior active DOFs in the corresponding cell, respectively.

– **Sparse Elimination.** We first focus on a single cell at level ℓ indexed with $\mathbf{j}$, i.e., $C_{\mathbf{j}}^{\ell}$. To simplify the notation, we drop the superscript and subscript for now and introduce $C = C_{\mathbf{j}}^{\ell}$, $I = I_{\mathbf{j}}^{\ell}$, $F = \mathcal{F}_{\mathbf{j}}^{\ell}$, and $R = R_{\mathbf{j}}^{\ell}$. Based on the discretization and previous level eliminations, the interior active DOFs interact only with itself and its surrounding faces. The interactions of the interior active DOFs and the rest DOFs are empty and the corresponding matrix is zero, $A^{\ell}(R, I) = 0$. Hence, by applying sparse elimination, we have,

$$S_I^T A^{\ell} S_I = \begin{bmatrix} D_I & & \\ & B_{FF}^{\ell} & \left(A_{RF}^{\ell}\right)^T \\ & A_{RF}^{\ell} & A_{RR}^{\ell} \end{bmatrix}, \tag{2.18}$$

where the explicit definitions of B_{FF}^{ℓ} and S_I are given in the discussion of sparse elimination. This factorization eliminates I from the active DOFs of A^{ℓ}.

Looping over all cells $C_{\mathbf{j}}^{\ell}$ at level ℓ, we obtain

$$\widetilde{A}^{\ell} = \left(\prod_{I \in \mathcal{I}^{\ell}} S_I\right)^T A^{\ell} \left(\prod_{I \in \mathcal{I}^{\ell}} S_I\right), \tag{2.19}$$

$$\widetilde{\Sigma}^{\ell} = \Sigma^{\ell} \backslash \bigcup_{I \in \mathcal{I}^{\ell}} I. \tag{2.20}$$

Now all the active interior DOFs at level ℓ are eliminated from Σ^{ℓ}.

– **Skeletonization.** Each face at level ℓ not only interacts within its own cell but also interacts with faces of neighbor cells. Since the interaction between

any two different faces is low-rank, this leads us to apply skeletonization. The skeletonization for any face $F \in \mathcal{F}^\ell$ gives,

$$S_{\bar{\bar{F}}}^T Q_F^T \widetilde{A}^\ell Q_F S_{\bar{\bar{F}}} = \left[\begin{array}{cc|c} \widetilde{D}_{\bar{\bar{F}}} & & \\ & \widetilde{B}^\ell_{\hat{F}\hat{F}} & \left(\widetilde{A}^\ell_{R\hat{F}}\right)^T \\ \hline & \widetilde{A}^\ell_{R\hat{F}} & \widetilde{A}^\ell_{RR} \end{array}\right], \tag{2.21}$$

where $\widehat{F}$ is the skeleton DOFs of F, $\bar{\bar{F}}$ is the redundant DOFs of F, and R refers to the rest DOFs. Due to the elimination from previous levels, $|F|$ scales as $O(m2^\ell)$ and $\widetilde{A}^\ell_{RF}$ contains a non-zero submatrix of size $O(m2^\ell) \times O(m2^\ell)$. Therefore, the interpolative decomposition can be formed efficiently. Readers are referred to Section 2.2.3 for the explicit forms of each matrix in (2.21).

Looping over all faces at level ℓ, we obtain

$$\begin{aligned} A^{\ell+1} &\approx \left(\prod_{F\in\mathcal{F}^\ell} S_{\bar{\bar{F}}} Q_F\right)^T \widetilde{A}^\ell \left(\prod_{F\in\mathcal{F}^\ell} S_{\bar{\bar{F}}} Q_F\right) \\ &= \left(\prod_{F\in\mathcal{F}^\ell} S_{\bar{\bar{F}}} Q_F\right)^T \left(\prod_{I\in\mathcal{I}^\ell} S_I\right)^T A^\ell \left(\prod_{I\in\mathcal{I}^\ell} S_I\right) \left(\prod_{F\in\mathcal{F}^\ell} S_{\bar{\bar{F}}} Q_F\right) \\ &= \left(W^\ell\right)^T A^\ell W^\ell, \end{aligned} \tag{2.22}$$

where $W^\ell = \left(\prod_{I\in\mathcal{I}^\ell} S_I\right)\left(\prod_{F\in\mathcal{F}^\ell} S_{\bar{\bar{F}}} Q_F\right)$. The active DOFs for the next level is now defined as,

$$\Sigma^{\ell+1} = \widetilde{\Sigma}^\ell \backslash \bigcup_{F\in\mathcal{F}^\ell} \bar{\bar{F}} = \Sigma^\ell \backslash \left(\left(\bigcup_{I\in\mathcal{I}^\ell} I\right) \bigcup \left(\bigcup_{F\in\mathcal{F}^\ell} \bar{\bar{F}}\right)\right). \tag{2.23}$$

- **Level L.** Finally, A^L and Σ^L are the matrix and active DOFs at level L. Up

to a permutation, A^L can be factorized as

$$\begin{aligned} A^L &= \begin{bmatrix} A^L_{\Sigma^L\Sigma^L} & \\ & D_R \end{bmatrix} = \begin{bmatrix} L_{\Sigma^L} & \\ & I \end{bmatrix} \begin{bmatrix} D_{\Sigma^L} & \\ & D_R \end{bmatrix} \begin{bmatrix} L^T_{\Sigma^L} & \\ & I \end{bmatrix} \\ &:= \big(W^L\big)^{-T} D \big(W^L\big)^{-1}. \end{aligned} \tag{2.24}$$

Combining all these factorization results

$$\begin{aligned} A \approx &\big(W^0\big)^{-T} \cdots \big(W^{L-1}\big)^{-T} \big(W^L\big)^{-T} \\ &D \big(W^L\big)^{-1} \big(W^{L-1}\big)^{-1} \cdots \big(W^0\big)^{-1} \equiv F \end{aligned} \tag{2.25}$$

and

$$A^{-1} \approx W^0 \cdots W^{L-1} W^L D^{-1} \big(W^L\big)^T \big(W^{L-1}\big)^T \cdots \big(W^0\big)^T = F^{-1}. \tag{2.26}$$

A^{-1} is factorized into a multiplicative sequence of matrices W^ℓ and each W^ℓ corresponding to level ℓ is again a multiplicative sequence of sparse matrices, S_I, $S_{\bar{F}}$ and Q_F. Due to the fact that any S_I, $S_{\bar{F}}$ or Q_F contains a small non-trivial (i.e., neither identity nor empty) matrix of size $O(\frac{N^{1/3}}{2^{L-\ell}}) \times O(\frac{N^{1/3}}{2^{L-\ell}})$, the overall complexity for strong and applying W^ℓ is $O(N/2^\ell)$. Hence the application of the inverse of A is of $O(N)$ computation and memory complexity.

2.3 Distributed-memory hierarchical interpolative factorization

This section describes our main contribution, the algorithm for the distributed-memory HIF.

2.3.1 Process tree

For simplicity, assume that there are 8^L processes. We introduce a *process tree* that has $L+1$ levels and resembles the hierarchical structure of the computation domain. Each node of this process tree is called a *process group*. First at the leaf level, there are 8^L leaf process groups denoted as $\{p_{\mathbf{j}}^0\}_{\mathbf{j}}$. Here $\mathbf{j} = (j_1, j_2, j_3)$, $0 \leqslant j_1, j_2, j_3 < 2^L$ and the superscript 0 refers to the leaf level (level 0). Each group at this level only contains a single process. Each node at level 1 of the process tree is constructed by merging 8 leaf processes. More precisely, we denote the process group at level 1 as $p_{\mathbf{j}}^1$ for $\mathbf{j} = (j_1, j_2, j_3)$, $0 \leqslant j_1, j_2, j_3 < 2^{L-1}$, and $p_{\mathbf{j}}^1 = \bigcup_{\lfloor \mathbf{j}_c/2 \rfloor = \mathbf{j}} p_{\mathbf{j}_c}^0$. Similarly, we recursively define the node at level ℓ as $p_{\mathbf{j}}^\ell = \bigcup_{\lfloor \mathbf{j}_c/2 \rfloor = \mathbf{j}} p_{\mathbf{j}_c}^{\ell-1}$. Finally, the process group $p_{\mathbf{0}}^L$ at the root includes all processes. Figure 2.4 illustrates the process tree. Each cube in the process tree is a process group.

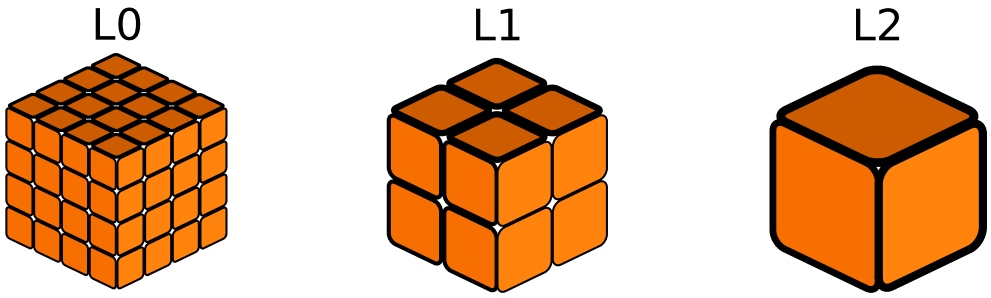

Figure 2.4: Process tree: 64 processes are organized in the process tree.

2.3.2 Distributed-memory method

Same as in Section 2.2.4, we define the $n \times n \times n$ grid on $\Omega = [0,1)^3$ for $n = m2^L$, where $m = O(1)$ is a small positive integer and $L = O(\log N)$ is the level number of the root level. Discretizing (2.1) with seven-point stencil on the grid provides the linear system $Au = f$, where A is a sparse $N \times N$ symmetric matrix, $u \in \mathbb{R}^N$ is the unknown function at grid points, and $f \in \mathbb{R}^N$ is the given function at grid points.

Given the process tree (Section 2.3.1) with 8^L processes and the sequential HIF structure (Section 2.2.4), the construction of the distributed-memory hierarchical interpolative factorization (DHIF) consists of the following steps.

- **Preliminary.** Construct the process tree with 8^L processes. Each process group $p_{\mathbf{j}}^0$ owns the data corresponding to cell $C_{\mathbf{j}}^0$ and the set of active DOFs in $C_{\mathbf{j}}^0$ is denoted as $\Sigma_{\mathbf{j}}^0$, for $\mathbf{j} = (j_1, j_2, j_3)$ and $0 \leqslant j_1, j_2, j_3 < 2^L$. Set $A^0 = A$ and let the process group $p_{\mathbf{j}}^0$ own $A^0(:, \Sigma_{\mathbf{j}}^0)$, which is a sparse tall-skinny matrix with $O(N/P)$ non-zero entries.

- **Level ℓ for $\ell = 0, \ldots, L-1$.**

  - **Preliminary.** Let A^ℓ denote the matrix before any elimination at level ℓ. $\Sigma_{\mathbf{j}}^\ell$ denotes the active DOFs owned by the process group $p_{\mathbf{j}}^\ell$ for $\mathbf{j} = (j_1, j_2, j_3)$, $0 \leqslant j_1, j_2, j_3 < 2^{L-\ell}$, and the non-zero submatrix of $A^\ell(:, \Sigma_{\mathbf{j}}^\ell)$ is distributed among the process group $p_{\mathbf{j}}^\ell$ using the two-dimensional block-cyclic distribution.

  - **Sparse Elimination.** The process group $p_{\mathbf{j}}^\ell$ owns $A^\ell(:, \Sigma_{\mathbf{j}}^\ell)$, which is sufficient for performing sparse elimination for $I_{\mathbf{j}}^\ell$. To simplify the notation, we define $I = I_{\mathbf{j}}^\ell$ as the active interior DOFs of cell $C_{\mathbf{j}}^\ell$, $F = \mathcal{F}_{\mathbf{j}}^\ell$ as the

surrounding faces, and $R = R^\ell_{\mathbf{j}}$ as the rest active DOFs. Sparse elimination at level ℓ within the process group $p^\ell_{\mathbf{j}}$ performs essentially

$$S_I^T A^\ell S_I = \begin{bmatrix} D_I & & \\ & B^\ell_{FF} & \left(A^\ell_{RF}\right)^T \\ & A^\ell_{RF} & A^\ell_{RR} \end{bmatrix}, \tag{2.27}$$

where $B^\ell_{FF} = A^\ell_{FF} - A^\ell_{FI}\left(A^\ell_{II}\right)^{-1}\left(A^\ell_{FI}\right)^T$,

$$S_I = \begin{bmatrix} \left(L^\ell_I\right)^{-T} & -\left(A^\ell_{II}\right)^{-1}\left(A^\ell_{FI}\right)^T & \\ & I & \\ & & I \end{bmatrix} \tag{2.28}$$

with $L^\ell_I D_I \left(L^\ell_I\right)^T = A^\ell_{II}$. Since $A^\ell(:, \Sigma^\ell_{\mathbf{j}})$ is owned locally by $p^\ell_{\mathbf{j}}$, both A^ℓ_{FI} and A^ℓ_{II} are local matrices. All non-trivial (i.e., neither identity nor empty) submatrices in S_I are formed locally and stored locally for application. On the other hand, updating on A^ℓ_{FF} requires some communication in the next step.

– **Communication after sparse elimination.** After all sparse eliminations are performed, some communication is required to update A^ℓ_{FF} for each cell $C^\ell_{\mathbf{j}}$. For the problem (2.1) with the periodic boundary conditions, each face at level ℓ is the surrounding face of exactly two cells. The owning process groups of these two cells need to communicate with each other to apply the additive updates, a submatrix of $-A^\ell_{FI}\left(A^\ell_{II}\right)^{-1}\left(A^\ell_{FI}\right)^T$. Once all communications are finished, the parallel sparse elimination does the rest

of the computation, which can be conceptually denoted as,

$$\begin{aligned}\widetilde{A}^{\ell} &= \left(\prod_{I\in\mathcal{I}^{\ell}} S_I\right)^T A^{\ell} \left(\prod_{I\in\mathcal{I}^{\ell}} S_I\right),\\ \widetilde{\Sigma}^{\ell}_{\mathbf{j}} &= \Sigma^{\ell}_{\mathbf{j}} \backslash \bigcup_{I\in\mathcal{I}^{\ell}} I,\end{aligned} \tag{2.29}$$

for $\mathbf{j} = (j_1, j_2, j_3), 0 \leqslant j_1, j_2, j_3 < 2^{L-\ell}$.

– **Skeletonization.** For each face F owned by $p^{\ell}_{\mathbf{j}}$, the corresponding matrices $\widetilde{A}^{\ell}(:, F)$ is stored locally. Similar to the parallel sparse elimination part, most operations are local at the process group $p^{\ell}_{\mathbf{j}}$ and can be carried out using the parallel dense linear algebra efficiently. By forming a parallel interpolative decomposition (ID) for $\widetilde{A}^{\ell}_{RF} = \begin{bmatrix}\widetilde{A}^{\ell}_{R\hat{F}} T^{\ell}_F & \widetilde{A}^{\ell}_{R\hat{F}}\end{bmatrix}$, the parallel skeletonization can be, conceptually, written as,

$$S_{\bar{F}} Q_F \widetilde{A}^{\ell} (Q_F)^T \left(S_{\bar{F}}\right)^T \approx \left[\begin{array}{cc|c} D_{\bar{F}} & & \\ & \widetilde{B}^{\ell}_{\hat{F}\hat{F}} & \widetilde{A}^{\ell}_{R\hat{F}} \\ \hline & \widetilde{A}^{\ell}_{R\hat{F}} & \widetilde{A}^{\ell}_{RR} \end{array}\right], \tag{2.30}$$

where the definitions of Q_F and $S_{\bar{F}}$ are given in the discussion of skeletonization. Since $\widetilde{A}^{\ell}_{\bar{F}\bar{F}}$, $\widetilde{A}^{\ell}_{\hat{F}\bar{F}}$, $\widetilde{A}^{\ell}_{\hat{F}\hat{F}}$ and T^{ℓ}_F are all owned by $p^{\ell}_{\mathbf{j}}$, it requires only local operations to form

$$\begin{aligned}\widetilde{B}^{\ell}_{\bar{F}\bar{F}} =& \widetilde{A}^{\ell}_{\bar{F}\bar{F}} - \left(T^{\ell}_F\right)^T \widetilde{A}^{\ell}_{\hat{F}\bar{F}} - \left(\widetilde{A}^{\ell}_{\hat{F}\bar{F}}\right)^T T^{\ell}_F + \left(T^{\ell}_F\right)^T \widetilde{A}^{\ell}_{\hat{F}\hat{F}} T^{\ell}_F,\\ \widetilde{B}^{\ell}_{\hat{F}\bar{F}} =& \widetilde{A}^{\ell}_{\hat{F}\bar{F}} - \widetilde{A}^{\ell}_{\hat{F}\hat{F}} T^{\ell}_F,\\ \widetilde{B}^{\ell}_{\hat{F}\hat{F}} =& \widetilde{A}^{\ell}_{\hat{F}\hat{F}} - \widetilde{B}^{\ell}_{\hat{F}\bar{F}} \left(\widetilde{B}^{\ell}_{\bar{F}\bar{F}}\right)^{-1} \left(\widetilde{B}^{\ell}_{\hat{F}\bar{F}}\right)^T.\end{aligned} \tag{2.31}$$

Similarly, $L_{\bar{\bar{F}}}$, which is the LDL^T factor of $\widetilde{B}_{\bar{\bar{F}}\bar{\bar{F}}}$, is also formed within the process group $p_{\mathbf{j}}^{\ell}$. Moreover, since non-trivial blocks in Q_F and $S_{\bar{\bar{F}}}$ are both local, this implies that the applications of Q_F and $S_{\bar{\bar{F}}}$ are local operations. As a result, the parallel skeletonization factorizes A^{ℓ} conceptually as,

$$\begin{aligned} A^{\ell+1} \approx & \left(\prod_{F\in\mathcal{F}^{\ell}} S_{\bar{\bar{F}}} Q_F\right)^T \widetilde{A}^{\ell} \left(\prod_{F\in\mathcal{F}^{\ell}} S_{\bar{\bar{F}}} Q_F\right) \\ = & \left(\prod_{F\in\mathcal{F}^{\ell}} S_{\bar{\bar{F}}} Q_F\right)^T \left(\prod_{I\in\mathcal{I}^{\ell}} S_I\right)^T A^{\ell} \left(\prod_{I\in\mathcal{I}^{\ell}} S_I\right) \left(\prod_{F\in\mathcal{F}^{\ell}} S_{\bar{\bar{F}}} Q_F\right) \end{aligned} \tag{2.32}$$

and we can define

$$\begin{aligned} W^{\ell} = & \left(\prod_{I\in\mathcal{I}^{\ell}} S_I\right) \left(\prod_{F\in\mathcal{F}^{\ell}} S_{\bar{\bar{F}}} Q_F\right), \\ \Sigma_{\mathbf{j}}^{\ell+1/2} = & \widetilde{\Sigma}_{\mathbf{j}}^{\ell} \setminus \bigcup_{F\in\mathcal{F}^{\ell}} \bar{\bar{F}} \\ = & \Sigma_{\mathbf{j}}^{\ell} \setminus \left(\left(\bigcup_{F\in\mathcal{F}^{\ell}} \bar{\bar{F}}\right) \bigcup \left(\bigcup_{I\in\mathcal{I}^{\ell}} I\right)\right). \end{aligned} \tag{2.33}$$

We would like to emphasize that the factors W^{ℓ} are evenly distributed among the process groups at level ℓ and that all non-trivial blocks are stored locally.

– **Merging and Redistribution.** Towards the end of the factorization at level ℓ, we need to merge the process groups and redistribute the data associated with the active DOFs in order to prepare for the work at level $\ell+1$. For each process group at level $\ell+1$, $p_{\mathbf{j}}^{\ell+1}$, for $\mathbf{j} = (j_1, j_2, j_3)$, $0 \leqslant j_1, j_2, j_3 < 2^{L-\ell-1}$, we first form its active DOF set $\Sigma_{\mathbf{j}}^{\ell+1}$ by merging $\Sigma_{\mathbf{j}_c}^{\ell+1/2}$ from all its children $p_{\mathbf{j}_c}^{\ell}$, where $\lfloor \mathbf{j}_c/2 \rfloor = \mathbf{j}$. In addition, $A^{\ell+1}(:, s_{\mathbf{j}}^{\ell+1})$

is separately owned by $\left\{p^{\ell}_{\mathbf{j}_c}\right\}_{\lfloor \mathbf{j}_c/2 \rfloor = \mathbf{j}}$. A redistribution among $p^{\ell+1}_{\mathbf{j}}$ is needed in order to reduce the communication cost for future parallel dense linear algebra. Although this redistribution requires a global communication among $p^{\ell+1}_{\mathbf{j}}$, the complexities for message and bandwidth are bounded by the cost for parallel dense linear algebra. Actually, as we shall see in the numerical results, its cost is far lower than that of the parallel dense linear algebra.

- **Level L Factorization.** The parallel factorization at level L is quite similar to the sequential one. After factorizations from all previous levels, $A^L(\Sigma^L_{\mathbf{0}}, \Sigma^L_{\mathbf{0}})$ is distributed among $p^L_{\mathbf{0}}$. A parallel LDL^T factorization of $A^L_{\Sigma^L_{\mathbf{0}}\Sigma^L_{\mathbf{0}}} = A^L(\Sigma^L_{\mathbf{0}}, \Sigma^L_{\mathbf{0}})$ among the processes in $p^L_{\mathbf{0}}$ results

$$\begin{aligned} A^L &= \begin{bmatrix} A^L_{\Sigma^L_{\mathbf{0}}\Sigma^L_{\mathbf{0}}} & \\ & D_R \end{bmatrix} \\ &= \begin{bmatrix} L^L_{\Sigma^L_{\mathbf{0}}} & \\ & I \end{bmatrix} \begin{bmatrix} D^L_{\Sigma^L_{\mathbf{0}}} & \\ & D_R \end{bmatrix} \begin{bmatrix} \left(L^L_{\Sigma^L_{\mathbf{0}}}\right)^T & \\ & I \end{bmatrix} = \left(W^L\right)^{-T} D \left(W^L\right)^{-1}. \end{aligned} \tag{2.34}$$

Consequently, we forms the DHIF for A and A^{-1} as

$$\begin{aligned} A \approx &\left(W^0\right)^{-T} \cdots \left(W^{L-1}\right)^{-T} \left(W^L\right)^{-T} \\ &D \left(W^L\right)^{-1} \left(W^{L-1}\right)^{-1} \cdots \left(W^0\right)^{-1} \equiv F \end{aligned} \tag{2.35}$$

and

$$A^{-1} \approx W^0 \cdots W^{L-1} W^L D^{-1} \left(W^L\right)^T \left(W^{L-1}\right)^T \cdots \left(W^0\right)^T = F^{-1}. \tag{2.36}$$

The factors, W^ℓ are evenly distributed among all processes and the application of F^{-1} is basically a sequence of parallel dense matrix-vector multiplications.

In Figure 2.5, we illustrate an example of DHIF for problem of size $24\times24\times24$ with $m = 6$ and $L = 2$. The computation is distributed on a process tree with $4^3 = 64$ processes. Particularly, Figure 2.5 highlights the DOFs owned by process groups involving $p^0_{(0,1,0)}$, i.e., $p^0_{(0,1,0)}$, $p^1_{(0,0,0)}$, and $p^2_{(0,0,0)}$. Yellow points denote interior active DOFs, blue and brown points denote face active DOFs, and black points denote edge active DOFs. Meanwhile, we also have unfaded and faded groups of points. Unfaded points are owned by the process groups involving $p^0_{(0,1,0)}$. In other words, $p^0_{(0,1,0)}$ is the owner for part of the unfaded points. The faded points are owned by other process groups. In the second row and the forth row, we also see faded brown points, which indicates the required communication to process $p^0_{(0,1,0)}$. Here Figure 2.5 works through two levels of the elimination processes of the DHIF step by step.

2.3.3 Complexity analysis

2.3.3.1 Memory complexity

There are two places in the distributed algorithm that require heavy memory usage. The first one is to store the original matrix A and its updated version A^ℓ for each level ℓ. As we mentioned above in the parallel algorithm, A^ℓ contains at most $O(N)$ non-zeros and they are evenly distributed on P processes as follows. At level ℓ, there are $8^{L-\ell}$ cells, and empirically each of which contains $O\left(\frac{N^{1/3}}{2^{L-\ell}}\right)$ active DOFs. Meanwhile, each cell is evenly owned by a process group with 8^ℓ processes. Hence, $O\left(\left(\frac{N^{1/3}}{2^{L-\ell}}\right)^2\right)$ non-zero entries of $A^\ell(:, s^\ell_{\mathbf{j}})$ is evenly distributed on process group $p^\ell_{\mathbf{j}}$ with 8^ℓ processes. Overall, there are $O\left(8^{L-\ell}\cdot\frac{N^{2/3}}{4^{L-\ell}}\right) = O(N\cdot2^{-\ell})$ non-zero entries in

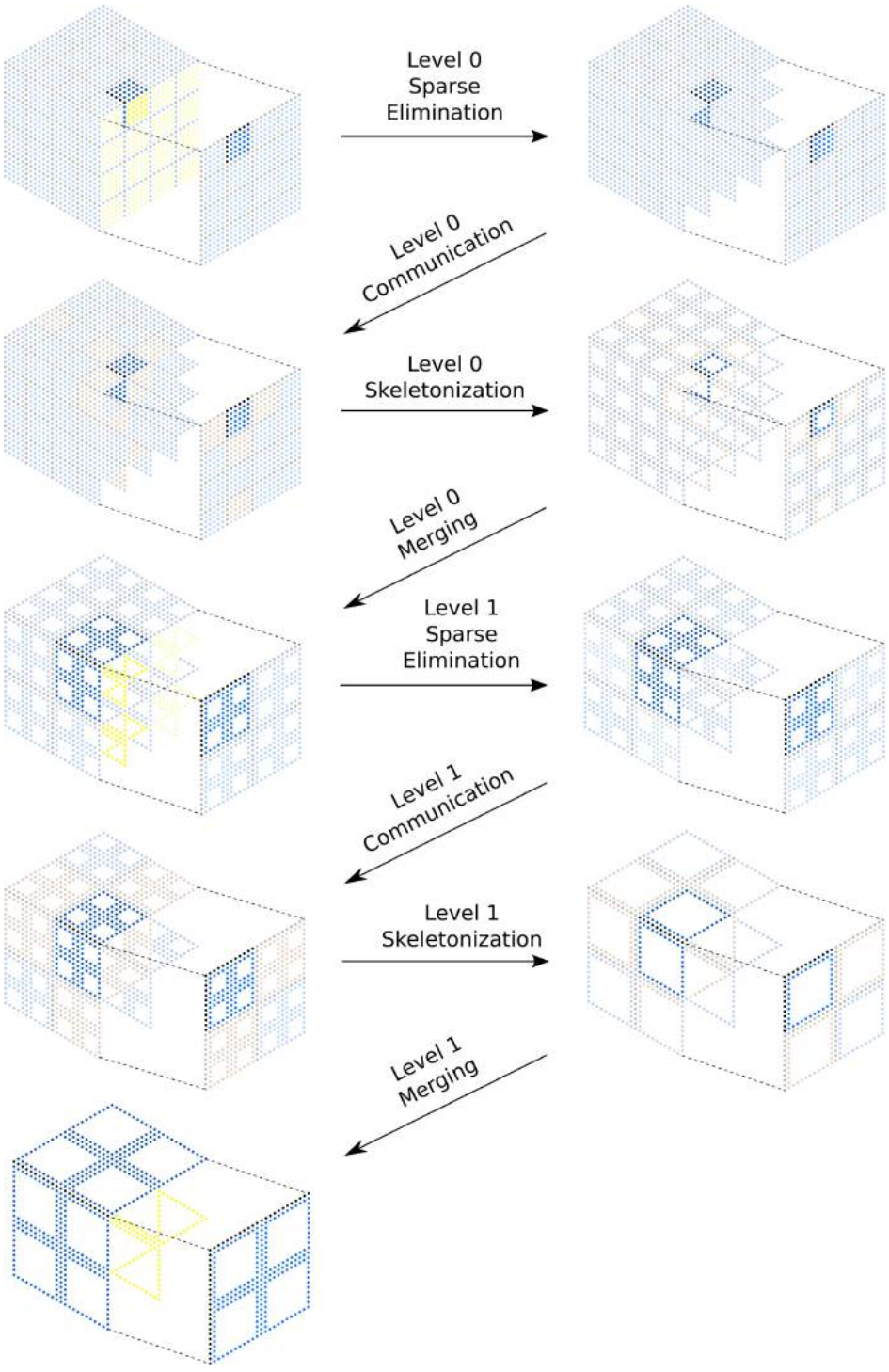

Figure 2.5: Distributed-memory hierarchical interpolative factorization

A^ℓ evenly distributed on $8^{L-\ell} \cdot 8^\ell = P$ processes, and each process owns $O\left(\frac{N}{P} \cdot 2^{-\ell}\right)$ data for A^ℓ. Moreover, the factorization at level ℓ does not rely on the matrix $A^{\ell'}$ for $\ell' < \ell - 1$. Therefore, the memory cost for storing A^ℓs is $O(\frac{N}{P})$ for each process.

The second place is to store the factors W^ℓ. It is not difficult to see that the memory cost for each W^ℓ is the same as A^ℓ. Only non-trivial blocks in S_I, Q_F, and $S_{\bar{F}}$ require storage. Since each of these non-trivial blocks is of size $O\left(\frac{N^{1/3}}{2^{L-\ell}}\right) \times O\left(\frac{N^{1/3}}{2^{L-\ell}}\right)$ and evenly distributed on 8^ℓ processes, the overall memory requirement for each W^ℓ on a process is $O\left(\frac{N}{P} \cdot 2^{-\ell}\right)$. Therefore, $O\left(\frac{N}{P}\right)$ memory is required on each process to store all W^ℓs.

2.3.3.2 Computation complexity

The majority of the computation work goes to the construction of S_I, Q_F and $S_{\bar{F}}$. As stated in the previous section, at level ℓ, each non-trivial dense matrix in these factors is of size $O\left(\frac{N^{1/3}}{2^{L-\ell}}\right) \times O\left(\frac{N^{1/3}}{2^{L-\ell}}\right)$. The construction adopts the matrix-matrix multiplication, the interpolative decomposition (pivoting QR), the LDL^T factorization, and the triangular matrix inversion. Each of these operation is of cubic computation complexities and the corresponding parallel computation cost over 8^ℓ processes is $O\left(\frac{N}{P}\right)$. Since there is only a constant number of these operations per process at a single level, the total computational complexity across all $O(\log N)$ levels is $O\left(\frac{N \log N}{P}\right)$.

The application computational complexity is simply the complexity of applying each non-zero entries in W^ℓs once, hence, the overall computational complexity is the same as the memory complexity $O\left(\frac{N}{P}\right)$.

2.3.3.3 Communication complexity

The communication complexity is composed of three parts: the communication in the parallel dense linear algebra, the communication after sparse elimination, and the merging and redistribution step within DHIF. It is clear to see that the communication cost for the second part is bounded by either of the rest. Hence, we will simply derive the communication cost for the first and third parts. Here, we adopt the simplified communication model, $T_{comm} = \alpha + \beta$, where α is the latency, and β is the inverse bandwidth.

At level ℓ, the parallel dense linear algebra involves the matrix-matrix multiplication, the ID, the LDL^T factorization, and the triangular matrix inversion for matrices of size $O\left(\frac{N^{1/3}}{2^{L-\ell}}\right) \times O\left(\frac{N^{1/3}}{2^{L-\ell}}\right)$. All these basic operations are carried out on a process group of size 8^ℓ. Following the discussion in [8], the communication cost for these operations are bounded by $O\left(\ell^3\sqrt{8^\ell}\right)\alpha + O\left(\frac{N^{2/3}}{4^{L-\ell}8^\ell}\ell\right)\beta$. Summing over all levels, one can control the communication cost of the parallel dense linear algebra part by

$$O\left(\sqrt{P}\log^3 P\right)\alpha + O\left(\frac{N^{2/3}}{P^{2/3}}\right)\beta. \tag{2.37}$$

On the other hand, the merging and redistribution step at level ℓ involves $8^{\ell+1}$ processes and redistributes matrices of size $O\left(\frac{N^{1/3}}{2^{L-\ell}} \cdot 8\right) \times O\left(\frac{N^{1/3}}{2^{L-\ell}} \cdot 8\right)$. The current implementation adopts the MPI routine MPI_AllToAll to handle the redistribution on a 2D process mesh. Further, we assume the all-to-all communication sends and receives long messages. The standard upper bound for the cost of this routine is $O\left(\sqrt{8^{\ell+1}}\right)\alpha + O\left(\frac{N^{2/3}}{4^{L-\ell}\sqrt{8^{\ell+1}}} \cdot 64\right)\beta$ [83]. Therefore, the over all cost is

$$O\left(\sqrt{P}\right)\alpha + O\left(\frac{N^{2/3}}{\sqrt{P}}\right)\beta. \tag{2.38}$$

The complexity of the latency part is not scalable. However empirically, the cost for this communication is relatively small in the actual running time.

2.4 Numerical results

Here we present a couple of numerical examples to demonstrate the parallel efficiency of the distributed-memory HIF. The algorithm is implemented in C++11 and all inter-process communication is expressed via the Message Passing Interface (MPI). The distributed-memory dense linear algebra computation is done through the Elemental library [78]. All numerical examples are performed on Edison at the National Energy Research Scientific Computing center (NERSC). The numbers of processes used are always powers of two, ranging from 1 to 8192. The memory allocated for each process is limited to 2GB.

All numerical results are measured in two ways: the strong scaling and weak scaling. The strong scaling measurement fixes the problem size, and increases the number of processes. For a fixed problem size, let T_P^S be the running time of P processes. The strong scaling efficiency is defined as,

$$E^S = \frac{T_1^S}{P \cdot T_P^S}. \tag{2.39}$$

In the case that T_1^S is not available, e.g., the fixed problem can not fit into the single process memory, we adopts the first available running time, T_m^S, associating with the smallest number of processes, m, as a reference. And the modified strong scaling efficiency is,

$$E^S = \frac{m \cdot T_m^S}{P \cdot T_P^S}. \tag{2.40}$$

The weak scaling measurement fixes the ratio between the problem size and the number of processes. For a fixed ratio, we define the weak scaling efficiency as,

$$E^W = \frac{T_m^W}{T_P^W}, \tag{2.41}$$

where T_m^W is the first available running time with m processes, and T_P^W is the running time of P processes.

Notation	Explanation
ϵ	Relative precision of the ID
N	Total number of DOFs in the problem
e_s	Relative error for solving, $\|(I - F^{-1}A)x\| / \|x\|$, where x is a Gaussian random vector
$\|\Sigma_L\|$	Number of remaining active DOFs at the root level
m_f	Maximum memory required to perform the factorization in GB across all processes
t_f	Time for constructing the factorization in seconds
E_f^S	Strong scaling efficiency for factorization time
t_s	Time for applying F^{-1} to a vector in seconds
E_s^S	Strong scaling efficiency for application time
n_{iter}	Number of iterations to solve $Au = f$ with GMRES with F^{-1} being a preconditioner to a tolerance of 10^{-12}

Table 2.2: Notations for the numerical results of DHIF

The notations used in the following tables and figures are explained in Table 2.2. For simplicity, all examples are defined over $\Omega = [0, 1)^3$ with periodic boundary condition, discretized on a uniform grid, $n \times n \times n$, with n being the number of points in each dimension and $N = n^3$. The PDEs defined in (2.1) is discretized using the second-order central difference method with seven-point stencil, which is the same as (2.16). Octrees are adopted to partition the computation domain with the block size at leaf level bounded by 64.

Example 1. We first consider the problem in (2.1) with $a(x) \equiv 1$ and $b(x) \equiv 0.1$. The relative precision of the ID is set to be $\epsilon = 10^{-3}$.

As shown in Table 2.3, given the tolerance $\epsilon = 10^{-3}$ the relative error remains well below this for all N and P. The number of skeleton points on the root level, $|\Sigma_L|$, grows linearly as the one-dimensional problem size increases. The empirical linear scaling of the root skeleton size strongly supports the quasi linear scaling for the factorization, linear scaling for the application, and linear scaling for memory cost. The column labeled with m_f in Table 2.3, or alternatively Figure 2.6c, illustrates the perfect strong scaling for the memory cost. Since the bottleneck for most parallel algorithms is the memory cost, this point is especially important in practice. Perfect distribution of the memory usage allows us to solve very large problem on a massive number of processes, even through the communication penalty on massive parallel computing would be relatively large. The factorization time and application time show good scaling up to thousands of processes. Figure 2.6a and Figure 2.6b present the strong scaling plot for the running time of factorization and application respectively. Together with Figure 2.6d, which illustrates the timing for each part of the factorization, we conclude that the communication cost beside the parallel dense linear algebra (labeled with "El") remains small comparing to the cost of the parallel dense linear algebra. It is the parallel dense linear algebra part that stops the strong scaling. As it is also well know that parallel dense linear algebra achieves good weak scaling, so does our DHIF implementation. Finally, the last column of Table 2.3 shows the number of iterations for solving $Au = f$ using the GMRES algorithm with a relative tolerance of 10^{-12} and with the DHIF as a preconditioner. As the numbers in the entire column are equal to 6, this shows that DHIF serves an excellent preconditioner with the iteration number almost independent of the problem size.

N	P	e_s	$\|s_L\|$	m_f	t_f	E_f^S	t_s	E_s^S	n_{iter}
32^3	1	4.84e-04	3440	1.92e-01	4.85e+00	100%	1.36e-01	100%	6
	2	5.26e-04	3440	9.60e-02	2.60e+00	93%	6.65e-02	103%	6
	4	3.78e-04	3440	4.80e-02	1.45e+00	84%	3.47e-02	98%	6
	8	4.93e-04	3440	2.40e-02	8.38e-01	72%	1.99e-02	85%	6
	16	3.97e-04	3440	1.20e-02	5.83e-01	52%	1.31e-02	65%	6
	32	7.33e-04	3440	6.03e-03	4.35e-01	35%	1.47e-02	29%	6
64^3	2	5.92e-04	7760	9.07e-01	3.87e+01	100%	6.08e-01	100%	6
	4	5.98e-04	7760	4.54e-01	2.36e+01	82%	2.99e-01	102%	6
	8	5.59e-04	7760	2.27e-01	1.48e+01	65%	1.61e-01	94%	6
	16	6.30e-04	7760	1.13e-01	1.03e+01	47%	9.52e-02	80%	6
	32	5.89e-04	7760	5.68e-02	5.34e+00	45%	6.88e-02	55%	6
	64	5.45e-04	7760	2.84e-02	2.67e+00	45%	4.10e-02	46%	6
	128	5.43e-04	7760	1.42e-02	1.52e+00	40%	3.43e-02	28%	6
	256	6.29e-04	7760	7.14e-03	1.27e+00	24%	2.69e-02	18%	6
128^3	16	6.19e-04	16208	9.77e-01	1.43e+02	100%	8.24e-01	100%	6
	32	5.98e-04	16208	4.89e-01	7.40e+01	97%	4.37e-01	94%	6
	64	5.85e-04	16208	2.44e-01	3.87e+01	92%	2.26e-01	91%	6
	128	6.23e-04	16208	1.22e-01	2.11e+01	85%	1.40e-01	74%	6
	256	6.14e-04	16208	6.12e-02	1.00e+01	89%	9.76e-02	53%	6
	512	5.96e-04	16208	3.06e-02	5.80e+00	77%	1.98e-01	13%	6
	1024	5.86e-04	16208	1.54e-02	3.46e+00	65%	6.13e-02	21%	6
256^3	128	6.18e-04	33104	1.01e+00	2.24e+02	100%	9.18e-01	100%	6
	256	6.11e-04	33104	5.07e-01	1.19e+02	94%	4.88e-01	94%	6
	512	6.06e-04	33104	2.53e-01	6.33e+01	88%	2.85e-01	81%	6
	1024	6.25e-04	33104	1.27e-01	3.19e+01	88%	1.86e-01	62%	6
	2048	6.18e-04	33104	6.35e-02	2.44e+01	57%	1.58e-01	36%	6
	4096	6.16e-04	33104	3.18e-02	1.27e+01	55%	1.73e-01	17%	6
	8192	6.14e-04	33104	1.60e-02	1.16e+01	30%	4.14e-01	3%	6
512^3	1024	6.16e-04	66896	1.03e+00	3.32e+02	100%	1.08e+00	100%	6
	2048	6.15e-04	66896	5.16e-01	1.84e+02	90%	6.53e-01	82%	6
	4096	6.14e-04	66896	2.58e-01	9.55e+01	87%	4.90e-01	55%	6
	8192	6.13e-04	66896	1.29e-01	5.58e+01	74%	4.58e-01	29%	6
1024^3	8192	6.15e-04	134480	1.04e+00	4.67e+02	100%	1.48e+00	100%	6

Table 2.3: Example 1. Numerical results for DHIF

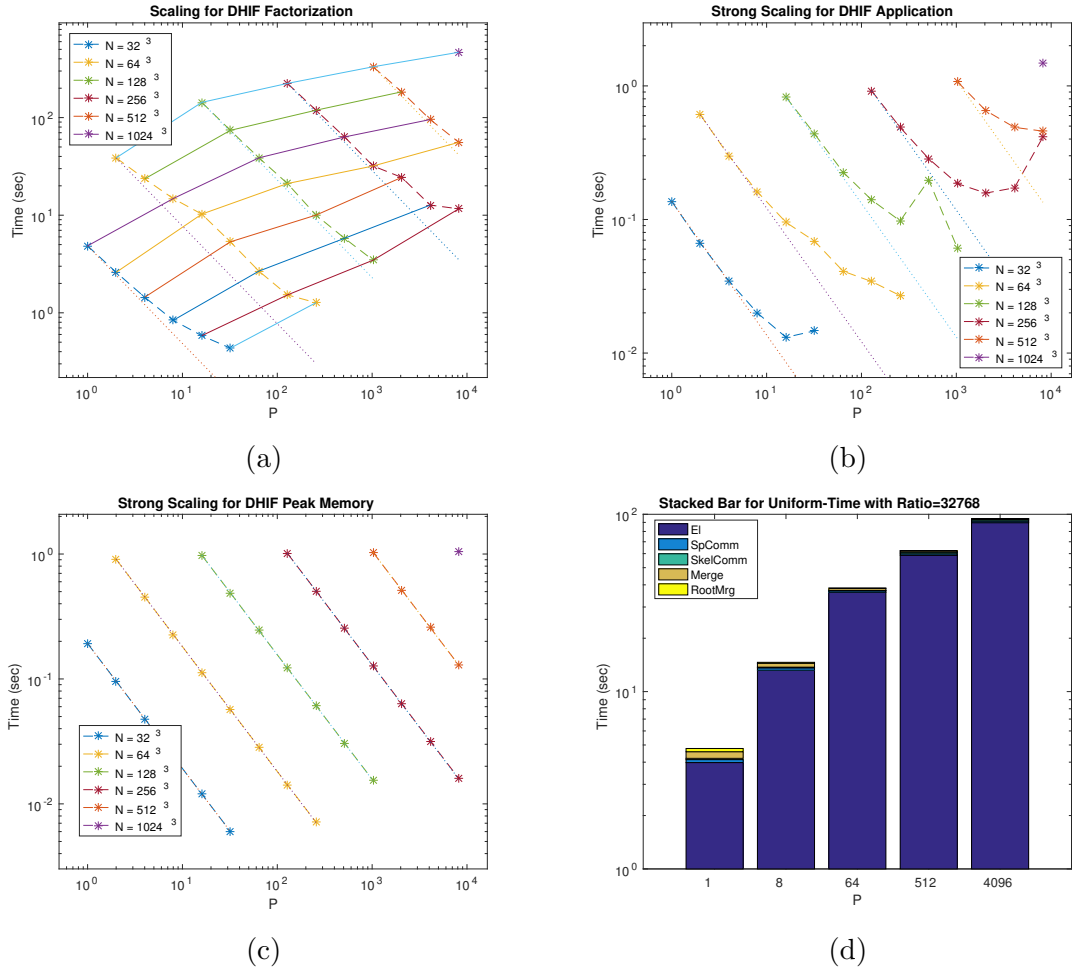

Figure 2.6: Example 1. (a) is the scaling plot for the DHIF factorization time, the solid lines indicate the weak scaling results, the dashed lines are the strong scaling results, and the dotted lines are the reference lines for perfect strong scaling, the line style apples to all figures below; (b) is the strong scaling for the DHIF application time; (c) is the strong scaling for the DHIF peak memory usage; (d) shows a stacked bar plot for factorization time for fixed ratio between the problem size and the number of processes.

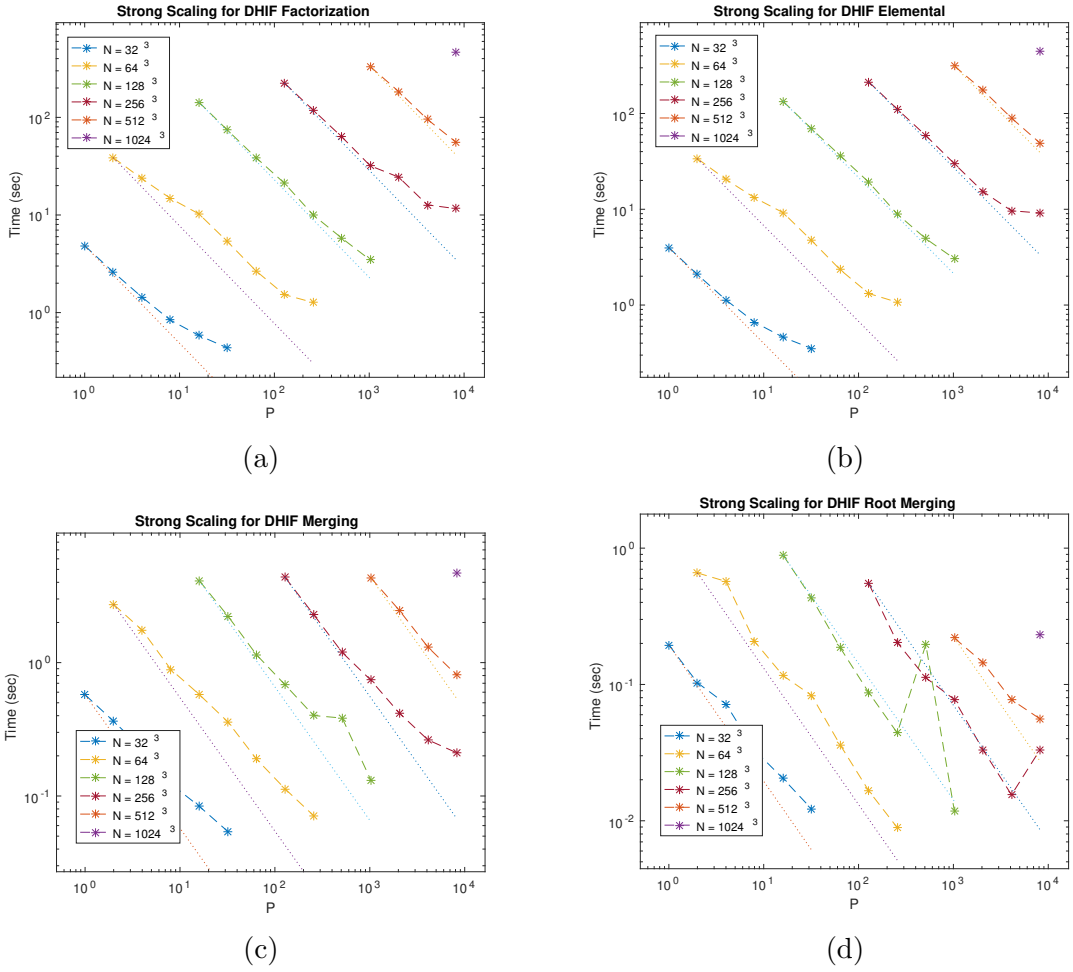

Figure 2.7: Example 1. (a) is the strong scaling plot for the DHIF factorization time; (b) is the strong scaling for the Elemental time; (c) is the strong scaling for the merging time excluding the root level; (d) is the strong scaling for the merging time at root level.

Example 2. This example is a problem of (2.1) with high-contrast random field $a(x)$ and $b(x) \equiv 0.1$. The high-contrast random field $a(x)$ is defined as follows,

1. Generate uniform random value $a_{\mathbf{j}}$ between 0 and 1 for each discretization point;
2. Convolve the random value $a_{\mathbf{j}}$ with an isotropic three-dimensional Gaussian with standard deviation 1;
3. Quantize the field via

$$a_{\mathbf{j}} = \begin{cases} 0.1, & a_{\mathbf{j}} \leqslant 0.5 \\ 1000, & a_{\mathbf{j}} > 0.5 \end{cases} . \tag{2.42}$$

The given tolerance is set to be 10^{-5}.

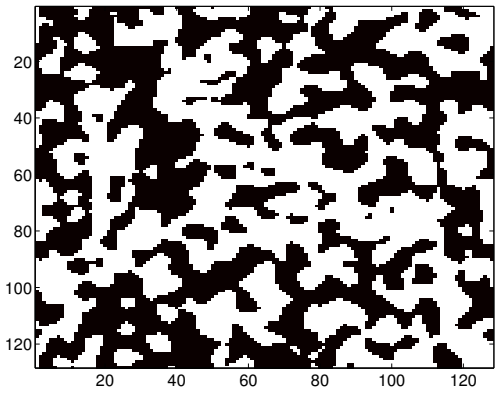

Figure 2.8: A slice in a random field realization of size 128^3.

Figure 2.8 shows a slice in a realization of the random field. The corresponding matrix A is clearly of high-contrast. Solving such a problem is harder than example 1 due to the raise of the condition number. The performance results of our algorithm are presented in Table 2.4. As we expect, the relative error for solving is lower than that in Table 2.3 and the number of iterations in GMRES is higher.

Table 2.4, Figure 2.9 and Figure 2.10 demonstrate the efficiency of the DHIF for high-contrast random field. Almost all comments regarding the numerical results in

N	P	e_s	$\lvert s_L\rvert$	m_f	t_f	E_f^S	t_s	E_s^S	n_{iter}
32^3	1	3.02e-03	3865	2.00e-01	5.80e+00	100%	1.34e-01	100%	7
	2	3.39e-03	3632	9.31e-02	2.48e+00	117%	6.69e-02	100%	7
	4	2.69e-03	3934	5.13e-02	1.72e+00	84%	3.75e-02	89%	7
	8	3.18e-03	3660	2.37e-02	9.50e-01	76%	2.20e-02	76%	7
	16	3.13e-03	3693	1.24e-02	6.22e-01	58%	1.32e-02	63%	7
	32	3.00e-03	3744	6.42e-03	4.83e-01	38%	1.49e-02	28%	7
64^3	2	3.29e-03	8580	9.45e-01	4.33e+01	100%	6.15e-01	100%	7
	4	3.13e-03	8938	4.94e-01	2.91e+01	74%	3.10e-01	99%	7
	8	3.09e-03	9600	2.51e-01	1.98e+01	55%	1.68e-01	91%	7
	16	3.07e-03	8919	1.19e-01	1.27e+01	43%	9.86e-02	78%	7
	32	3.09e-03	9478	6.59e-02	6.99e+00	39%	7.89e-02	49%	7
	64	3.18e-03	9111	3.03e-02	3.17e+00	43%	4.90e-02	39%	7
	128	3.02e-03	9419	1.58e-02	2.15e+00	31%	3.31e-02	29%	7
	256	3.03e-03	9349	7.97e-03	1.60e+00	21%	3.66e-02	13%	7
128^3	16	3.16e-03	19855	1.07e+00	2.11e+02	100%	8.89e-01	100%	7
	32	3.09e-03	20487	5.58e-01	1.18e+02	90%	4.86e-01	91%	7
	64	3.06e-03	21345	2.78e-01	6.43e+01	82%	2.45e-01	91%	7
	128	3.10e-03	20344	1.37e-01	3.39e+01	78%	1.34e-01	83%	7
	256	3.07e-03	21152	7.43e-02	1.76e+01	75%	1.10e-01	51%	7
	512	3.07e-03	20779	3.51e-02	8.46e+00	78%	8.80e-02	32%	7
	1024	3.04e-03	21361	1.76e-02	5.38e+00	61%	6.31e-02	22%	7
256^3	128	3.11e-03	42420	1.14e+00	4.15e+02	100%	1.04e+00	100%	7
	256	3.12e-03	43828	5.91e-01	2.12e+02	98%	5.77e-01	90%	8
	512	3.11e-03	44126	2.90e-01	1.25e+02	83%	3.86e-01	67%	7
	1024	3.08e-03	43302	1.46e-01	6.31e+01	82%	2.12e-01	61%	7
	2048	3.09e-03	44131	7.78e-02	3.43e+01	76%	1.86e-01	35%	7
	4096	3.10e-03	43691	3.71e-02	1.96e+01	66%	2.28e-01	14%	7
	8192	3.10e-03	43952	1.85e-02	2.05e+01	32%	4.03e-01	4%	7
512^3	1024	3.11e-03	88070	1.16e+00	6.37e+02	100%	1.22e+00	100%	7
	2048	3.11e-03	88577	6.11e-01	3.47e+02	92%	6.84e-01	89%	8
	4096	3.11e-03	88757	3.03e-01	1.89e+02	84%	5.31e-01	58%	7
	8192	3.11e-03	85877	1.50e-01	1.02e+02	78%	6.20e-01	25%	7
1024^3	8192	3.11e-03	177323	1.18e+00	9.35e+02	100%	1.95e+00	100%	8

Table 2.4: Example 2. Numerical results for DHIF

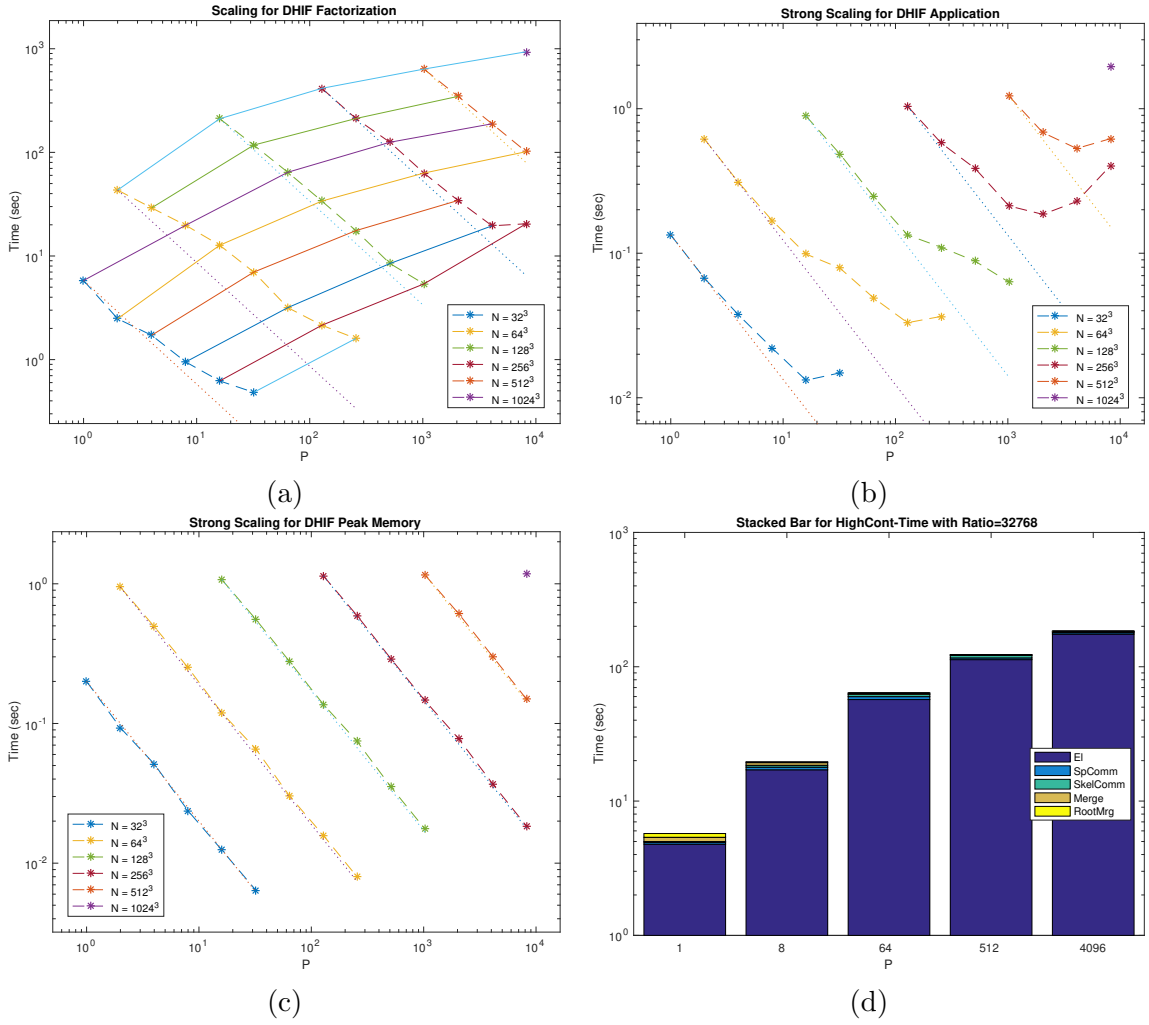

Figure 2.9: Example 2. (a) provides a scaling plot for DHIF factorization time; (b) is the strong scaling for DHIF application time; (c) is the strong scaling for DHIF peak memory usage; (d) shows a stacked bar plot for factorization time for fixed ratio between problem size and number of processes.

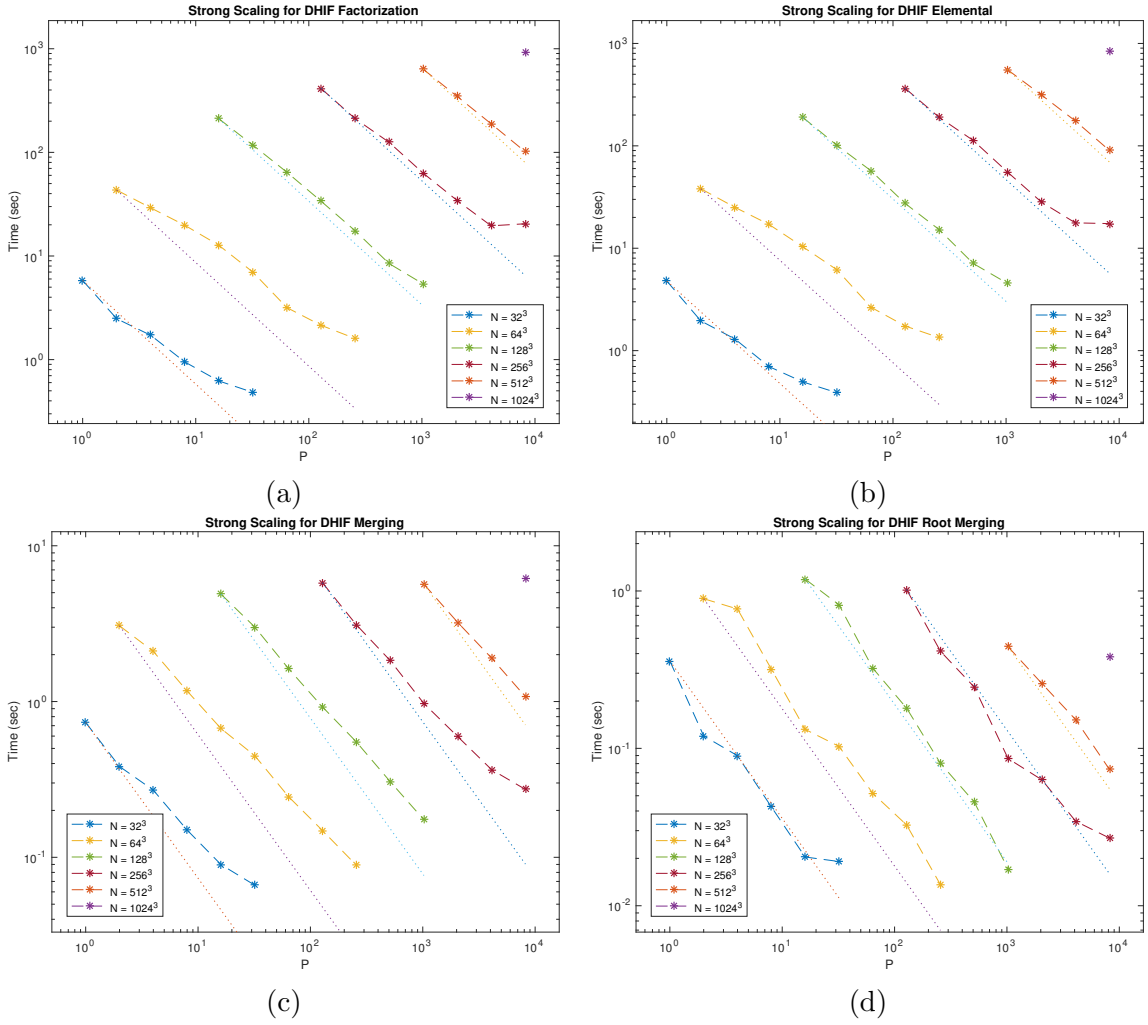

Figure 2.10: Example 2. (a) is the strong scaling plot for the DHIF factorization time; (b) is the strong scaling for the Elemental time; (c) is the strong scaling for the merging time excluding the root level; (d) is the strong scaling for the merging time at root level.

Example 1 apply here. To focus on the difference between Example 1 and Example 2, the most noticeable difference is about the relative error, e_s. Though we give a higher relative precision $\epsilon = 10^{-5}$, the relative error for Example 2 is about $3 \cdot 10^{-3}$, which is about ten times larger than e_s in Example 1. The reason for the decrease of accuracy is most likely the increase of the condition number for Example 2. This also increases the number of iterations in GMRES. However, both e_s and n_{iter} remain roughly constant for varying problem sizes. This means that DHIF still serves as a robust and efficient solver and preconditioner for such problems. Another difference is the number of skeleton points on the root level, $|\Sigma_L|$. Due to the fact that the field $a(x)$ is random, and different rows in Table 2.4 actually adopts different realizations, the small fluctuation of $|\Sigma_L|$ for the same N and different P is expected. Overall $|\Sigma_L|$ still grows linearly as $n = N^{1/3}$ increases. This again supports the complexity analysis given above.

Example 3. This example is a problem of (2.1) with constant $a(x) = 1$ and $b(x) = -(2\pi\kappa)^2$, where κ is the number of wavelengths. In this case, (2.1) becomes Helmholtz equation. As problem becomes more oscillatory, the DOFs per wavelength is fixed to be 8. The given tolerance is set to be 10^{-6} while the accuracy for GMRES is reduced to be 10^{-8}.

Table 2.5, Figure 2.11 and Figure 2.12 demonstrate the efficiency of the DHIF for Helmholtz problems. There are two major observations from this numerical example. First, for Helmholtz problem, given the same tolerance, the solving accuracy actually decreases as the problem size growths, which means that the domain contains more wavelengths. Second, the size of the skeleton, indicated by $|s_L|$, increases super-linear as problem size growths. Both of these observations are caused by the wave property. The ranks in the low-rank submatrices are no longer constant but depend on the

N	P	e_s	$\lvert s_L\rvert$	m_f	t_f	E_f^S	t_s	E_s^S	n_{iter}
32^3	1	5.60e-07	5264	3.17e-01	1.03e+01	100%	2.55e-01	100%	2
	2	6.04e-07	5264	1.59e-01	5.62e+00	92%	1.22e-01	105%	2
	4	6.83e-07	5264	7.93e-02	3.14e+00	82%	6.27e-02	102%	2
	8	7.29e-07	5264	3.97e-02	1.80e+00	72%	3.53e-02	91%	2
	16	7.21e-07	5264	1.99e-02	1.19e+00	54%	2.14e-02	74%	2
	32	9.33e-07	5264	9.95e-03	8.47e-01	38%	3.50e-02	23%	2
64^3	2	1.45e-05	14008	1.81e+00	1.58e+02	100%	1.11e+00	100%	2
	4	1.31e-05	14008	9.03e-01	8.92e+01	89%	5.69e-01	98%	2
	8	1.87e-05	14008	4.52e-01	5.26e+01	75%	3.04e-01	92%	2
	16	1.79e-05	14008	2.26e-01	3.73e+01	53%	1.73e-01	80%	2
	32	1.23e-05	14008	1.13e-01	2.02e+01	49%	1.23e-01	57%	2
	64	1.46e-05	14008	5.65e-02	1.04e+01	48%	7.76e-02	45%	2
	128	1.20e-05	14008	2.83e-02	4.78e+00	52%	5.33e-02	33%	2
	256	1.96e-05	14008	1.42e-02	3.49e+00	35%	8.87e-02	10%	2
128^3	32	4.28e-05	34968	1.11e+00	3.90e+02	100%	8.78e-01	100%	3
	64	4.79e-05	34968	5.53e-01	1.99e+02	98%	4.49e-01	98%	3
	128	5.82e-05	34968	2.77e-01	1.06e+02	92%	2.56e-01	86%	3
	256	4.86e-05	34968	1.38e-01	6.01e+01	81%	2.10e-01	52%	3
	512	4.55e-05	34968	6.93e-02	3.28e+01	74%	1.68e-01	33%	3
	1024	4.01e-05	34968	3.47e-02	1.68e+01	73%	1.74e-01	16%	3
	2048	4.20e-05	34968	1.74e-02	1.32e+01	46%	1.56e-01	9%	3
256^3	256	1.03e-03	81136	1.24e+00	8.60e+02	100%	1.18e+00	100%	4
	512	1.06e-03	81136	6.19e-01	4.55e+02	94%	6.74e-01	88%	4
	1024	1.17e-03	81136	3.10e-01	2.59e+02	83%	4.51e-01	66%	4
	2048	1.24e-03	81136	1.55e-01	1.46e+02	73%	3.42e-01	43%	4
	4096	1.17e-03	81136	7.77e-02	8.29e+01	65%	3.78e-01	20%	4
	8192	1.25e-03	81136	3.89e-02	5.45e+01	49%	4.36e-01	8%	4
512^3	2048	2.90e-03	208528	1.35e+00	1.74e+03	100%	1.59e+00	100%	5
	4096	2.74e-03	208528	6.77e-01	9.75e+02	89%	1.20e+00	66%	5
	8192	2.37e-03	208512	3.39e-01	5.85e+02	74%	9.99e-01	40%	5

Table 2.5: Example 3. Numerical results for DHIF

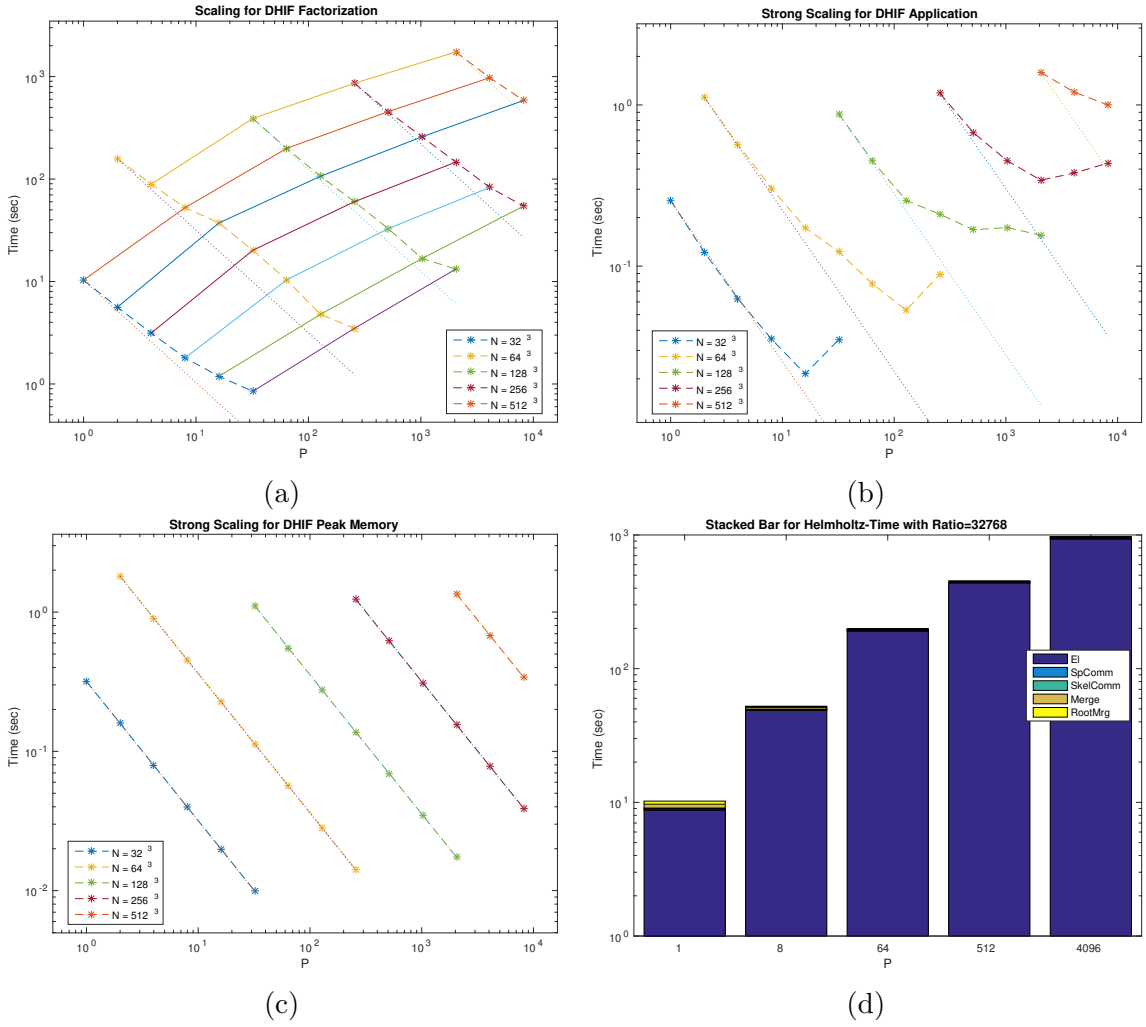

Figure 2.11: Example 3. (a) provides a scaling plot for DHIF factorization time; (b) is the strong scaling for DHIF application time; (c) is the strong scaling for DHIF peak memory usage; (d) shows a stacked bar plot for factorization time for fixed ratio between problem size and number of processes.

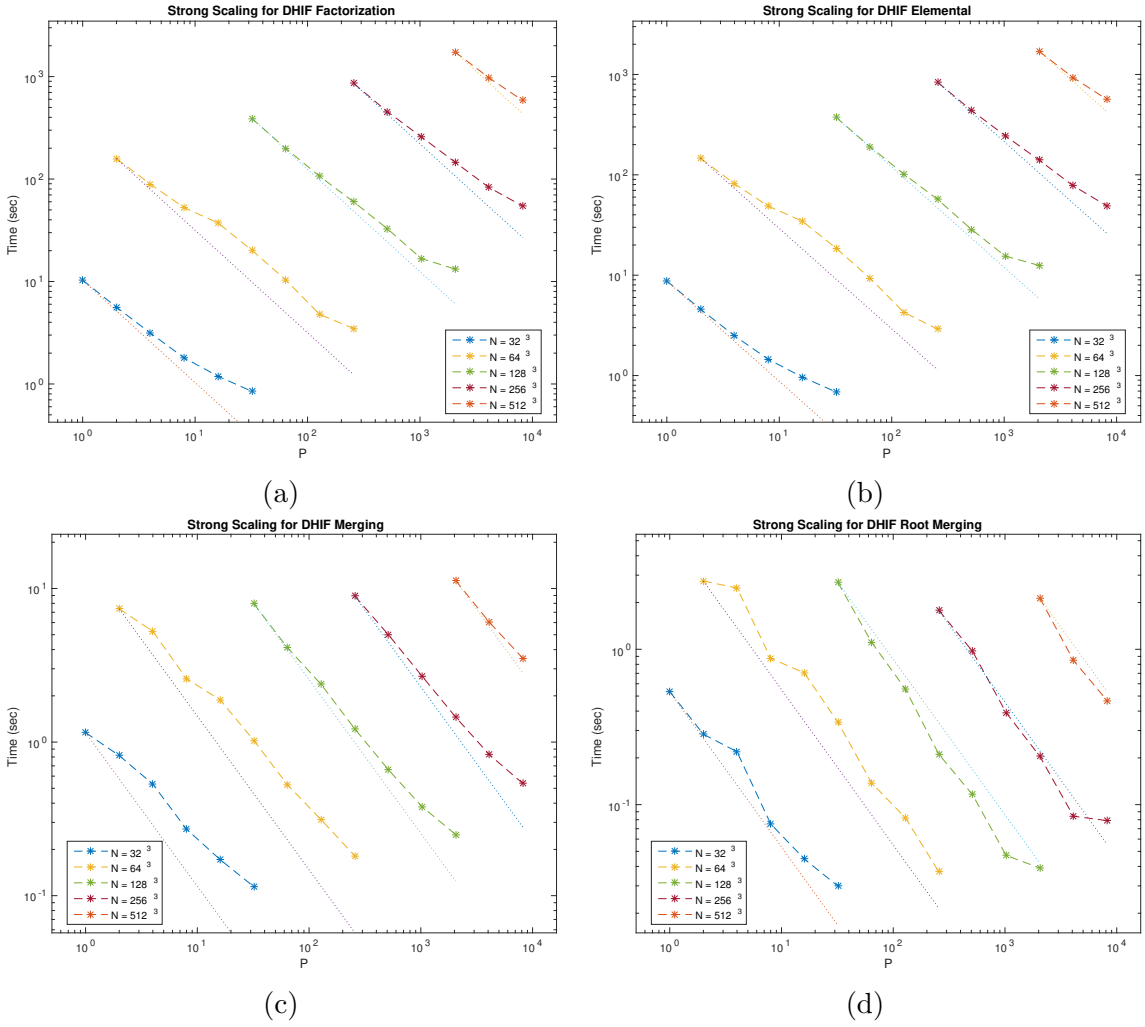

Figure 2.12: Example 3. (a) is the strong scaling plot for the DHIF factorization time; (b) is the strong scaling for the Elemental time; (c) is the strong scaling for the merging time excluding the root level; (d) is the strong scaling for the merging time at root level.

problem size. Finally, the number of iterations in GMRES also growths mildly as problem size increases. According to the numerical results, we claim that HIF and DHIF are still efficient preconditioners for Helmholtz problems.

Example 4. This example provides a concrete comparison between the proposed DHIF and multigrid method (hypre [24]). The problem behaves similar as example 2 without randomness, (2.1) with high-contrast field $a(x)$ and $b(x) \equiv 0.1$. The high-contrast field $a(x)$ is defined as follows,

$$a(\mathbf{x}) = \begin{cases} 1000, & \sum_{i=1}^{3} \lfloor \frac{x_i n}{7} \rfloor \equiv 0 \pmod 2 \\ 0.1, & \sum_{i=1}^{3} \lfloor \frac{x_i n}{7} \rfloor \equiv 1 \pmod 2 \end{cases}, \tag{2.43}$$

where n is the number of grid points on each dimension.

We adopt GMRES iterative method in both DHIF and hypre to solve the elliptic problem to a relative error 10^{-12}. The given tolerance in DHIF is set to be 10^{-4}. And SMG interface in hypre is used as preconditioner for the problem on regular grids. The numerical results for DHIF and hypre are given in Table 2.6.

		DHIF			hypre		
N	P	$t_{setup}(sec)$	$t_{solve}(sec)$	n_{iter}	$t_{setup}(sec)$	$t_{solve}(sec)$	n_{iter}
64^3	8	15.27	18.10	21	0.29	9.67	67
	64	2.46	3.45	21	1.47	17.37	60
128^3	64	29.20	24.53	22	1.78	140.90	394
	512	3.93	4.41	22	2.11	113.57	455
256^3	512	59.66	26.33	21	4.11	258.22	492
	4096	11.58	6.78	21	8.97	191.15	375

Table 2.6: Numerical results for DHIF and hypre. t_{setup} is the setup time which is identical to t_f in previous examples for DHIF, t_{solve} is the total iterative solving time using GMRES, n_{iter} is the number of iterations in GMRES.

As we can read from Table 2.6, there are a few advantages of DHIF over hypre in the given settings. First, the solving time of DHIF is faster than hypre's SMG except for on small problems with small numbers of processes. And the number of iterations grows as the problem size grows in hypre, while it remains almost the same in DHIF. In truly large problems, the advantages of DHIF are more pronounced. Second, the scalability of DHIF appears to be better than that of hypre's SMG. Finally, DHIF only requires powers of two numbers of processes, whereas hypre's SMG requires powers of eight for 3D problems.

2.5 Conclusion

In this chapter, we introduced the distributed-memory hierarchical interpolative factorization (DHIF) for solving discretized elliptic partial differential equations in 3D. The computational and memory complexity for DHIF are

$$O\left(\frac{N \log N}{P}\right) \quad \text{and} \quad O\left(\frac{N}{P}\right), \tag{2.44}$$

respectively, where N is the total number of DOFs and P is the number of processes. The communication cost is

$$O\left(\sqrt{P} \log^3 P\right) \alpha + O\left(\frac{N^{2/3}}{\sqrt{P}}\right) \beta, \tag{2.45}$$

where α is the latency, and β is the inverse bandwidth. Not only the factorization is efficient, the application can also be done in $O\left(\frac{N}{P}\right)$ operations. Numerical examples in Section 2.4 illustrate the efficiency and parallel scaling of the algorithm. The results show that DHIF can be used both as a direct solver and as an efficient preconditioner

for iterative solvers.

We have described the algorithm using the periodic boundary condition in order to simplify the presentation. However, the implementation can be extended in a straightforward way to problems with other type of boundary conditions. The discretization adopted here is the standard Cartesian grid. For more general discretizations such as finite element methods on unstructured meshes, one can generalize the current implementation by combining with the idea proposed in [82].

Here we have only considered the parallelization of the HIF for differential equations. As shown in [50], the HIF is also applicable to solving integral equations with non-oscillatory kernels. Parallelization of this case is also of practical importance.

Chapter 3

Oscillatory integral operator and butterfly algorithm

3.1 Background

This chapter is concerned with the rapid application of Fourier integral operators (FIOs), which are defined as

$$(Lf)(x) = \int_{\mathbb{R}^d} a(x,\xi) e^{2\pi\imath\Phi(x,\xi)} \widehat{f}(\xi)\, \mathrm{d}\xi, \tag{3.1}$$

where

- $a(x,\xi)$ is an amplitude function that is smooth both in x and ξ,
- $\Phi(x,\xi)$ is a phase function that is smooth in (x,ξ) for $\xi \neq 0$ [1] and obeys the homogeneity condition of degree 1 in ξ, namely, $\Phi(x,\lambda\xi) = \lambda\Phi(x,\xi)$ for each $\lambda > 0$;

[1] $\Phi(x,\xi)$ is allowed to be singular at $\xi = 0$. The (possible) singularity at $\xi = 0$ is application dependent, e.g., see the example of a generalized Radon transform in the numerical results section.

- $\widehat{f}(\xi)$ is the Fourier transform of the input function $f(x)$ defined by

$$\widehat{f}(\xi) = \int_{\mathbb{R}^d} e^{-2\pi\imath x\cdot\xi} f(x)dx.$$

The computation of Fourier integral operators appears quite often in the numerical solution of wave equations and related applications in computational geophysics [31, 53, 85, 96]. In a typical setting, it is often assumed that the problem is periodic (i.e., $a(x,\xi)$, $\Phi(x,\xi)$, and $f(x)$ are all periodic in x) or the function $f(x)$ decays sufficiently fast so that one can embed the problem in a sufficiently large periodic cell. To simplify the discussion, we restrict to the case $d = 2$. A simple discretization in two dimensions considers functions $f(x)$ given on a Cartesian grid

$$X = \left\{ x = \left(\frac{n_1}{n}, \frac{n_2}{n}\right), 0 \leqslant n_1, n_2 < n \text{ with } n_1, n_2 \in \mathbb{Z} \right\} \tag{3.2}$$

in a unit square and defines the discrete Fourier integral operator by

$$(Lf)(x) = \sum_{\xi\in\Omega} a(x,\xi) e^{2\pi\imath\Phi(x,\xi)} \widehat{f}(\xi), \quad x \in X,$$

where n is the number of discretization points on each dimension, $N = n^2$ denotes the total number of discretization points,

$$\Omega = \left\{ \xi = (n_1, n_2), -\frac{n}{2} \leqslant n_1, n_2 < \frac{n}{2} \text{ with } n_1, n_2 \in \mathbb{Z} \right\}, \tag{3.3}$$

and $\widehat{f}(\xi)$ is the discrete Fourier transform of $f(x)$,

$$\widehat{f}(\xi) = \frac{1}{n^2} \sum_{x\in X} e^{-2\pi\imath x\cdot\xi} f(x).$$

In most examples, $a(x,\xi)$ is numerically low-rank in the joint X and Ω domain [16, 30, 95] and its numerical treatment is relatively easy. Therefore, we will simplify the problem by assuming $a(x,\xi) = 1$ in the following algorithmic description and analysis. Under this assumption, the discrete FIO discussed in this chapter takes the following form:

$$(Lf)(x) = \sum_{\xi\in\Omega} e^{2\pi\imath\Phi(x,\xi)}\widehat{f}(\xi), \quad x \in X. \tag{3.4}$$

A direct computation of (3.4) takes $O(n^4)$ operations, which is quadratic in the number of DOFs $N = n^2$. Hence, a practical need is to design efficient and accurate algorithms to evaluate (3.4).

3.1.1 Related work

An earlier method for the rapid computation of general FIOs is the algorithm for two-dimensional problems proposed in [16]. This method starts by partitioning the frequency domain Ω into $O(\sqrt{n})$ wedges of equal angle. The summation (3.4) restricted to each wedge is then factorized into two components, both of which can be handled efficiently. The first one has a low-rank structure that leads to an $O(N\log N)$ fast computation, while the second one is a non-uniform Fourier transform which can be evaluated in $O(N\log N)$ steps with the algorithms developed in [5, 34]. Summing the computational cost over all $O(\sqrt{n})$ wedges gives an $O(N^{1.25}\log N)$ computational cost.

The butterfly algorithm proposed in [72, 75] gives rise a way to apply (3.4) for one-dimensional problems. These algorithms consist of two stages: the off-line stage and the on-line stage. In the off-line stage, it conducts simultaneously a top down traversal of a tree associated with domain X and a bottom up traversal of another

tree associated with domain Ω to recursively compress all low-rank submatrices (see Figure 3.2 for an example of necessary submatrices). In the end, a dense kernel matrix is factorized as a multiplication of $O(\log N)$ sparse matrices, each of which has $O(N)$ non-zeros. This typically takes $O(N^2)$ operations for the off-line stage. In the on-line stage, simply multiplying the sparse matrices to a given input vector $g \in \mathbb{C}^N$ costs $O(N \log N)$ operations.

Shortly after, an algorithm with strict quasilinear complexity for general FIOs for two-dimensional problems was proposed in [17] using the framework in [72, 75]. This approach introduces a polar coordinate transformation in the frequency domain to remove the singularity of $\Phi(x, \xi)$ at $\xi = 0$, proves the existence of low-rank separated approximations between certain pairs of spatial and frequency domains, and implements the low-rank approximations with oscillatory Chebyshev interpolations. The resulting algorithm evaluates (3.4) with $O(N \log N)$ operations and $O(N)$ memory. Both are essentially linear in terms of the number of DOFs. Inspired by these butterfly algorithms, more variants of the butterfly algorithm were designed to efficiently address other closely related problems, e.g., FIOs with the parallel butterfly algorithm [76], the sparse Fourier transforms [97], the numerical solutions of acoustic wave equations [31].

Another related research direction aims at sparse representations of the FIOs under modern basis functions from harmonic analysis. Such a sparse representation allows fast matrix-vector products in the transformed domain. Local Fourier transforms [7, 13, 25], wavelet-packet transforms [54], curvelet transforms [15, 18, 20, 19], wave atom frames [28, 29], wave packet frames [6, 27] have been investigated for the purpose of operator sparsification. In spite of favorable asymptotic behaviors, the actual representations of the FIOs typically have a large pre-factor in terms of both the

computational time and the memory requirement. This makes them less competitive compared to the approaches in [72, 16, 17, 75].

3.1.2 Organization

The rest of this chapter is organized as follows. Section 3.2 defines the complementary low-rank property, and reviews several butterfly algorithms and low-rank approximations. Section 3.3 proves a multiscale low-rank approximation that is essential to the multiscale butterfly algorithm. Section 3.4 combines the results of the previous two sections and describes the multiscale butterfly algorithm in detail. In Section 3.5, numerical results of several examples are provided to demonstrate the efficiency of the multiscale butterfly algorithm. Finally, we conclude this chapter with some discussion on parallelization in Section 3.6.

3.2 Low-rank approximations and butterfly algorithms

This section first summarizes the complementary low-rank property, which is the core to all butterfly algorithms and butterfly factorizations. We then briefly review the classical butterfly algorithm and the polar version. In this section, X and Ω refer to two general sets of N points, respectively. We assume the points in these two sets are distributed quasi-uniformly in their domains.

3.2.1 Complementary low-rank property

For a matrix, the rows are typically indexed by a set of points, say X, and the columns by another set of points, say Ω. Both X and Ω are often point sets in $\mathbb{R}^d$ for some dimension d. Associated with X and Ω are two trees T_X and T_Ω, respectively and both trees are assumed to have the same depth $L = O(\log N)$, with the top level being level 0 and the bottom one being level L.

Definition 3.2.1. *A matrix K of size $N \times N$ is said to satisfy the* **complementary low-rank property** *if for any level ℓ, any node A in T_X at level ℓ, and any node B in T_Ω at level $L-\ell$, the submatrix $K_{A,B}$, obtained by restricting K to the rows indexed by the points in A and the columns indexed by the points in B, is numerically low-rank. More precisely, for any ϵ, there exists a constant r_ϵ and two sets of functions $\{\alpha_t^{AB}(x)\}_{1\leqslant t\leqslant r_\epsilon}$ and $\{\beta_t^{AB}(\xi)\}_{1\leqslant t\leqslant r_\epsilon}$ such that the following holds*

$$\left| K(x,\xi) - \sum_{t=1}^{r_\epsilon} \alpha_t^{AB}(x)\beta_t^{AB}(\xi) \right| \leqslant \epsilon, \quad \forall x \in A, \forall \xi \in B. \tag{3.5}$$

The number r_ϵ is called the ϵ-separation rank.

The exact forms of the functions $\{\alpha_t^{AB}(x)\}_{1\leqslant t\leqslant r_\epsilon}$ and $\{\beta_t^{AB}(\xi)\}_{1\leqslant t\leqslant r_\epsilon}$ of course depend on the problem to which the butterfly algorithm is applied. In many applications, one can even show that the rank is only bounded polynomially in $\log(1/\epsilon)$ and is independent of N. While it is straightforward to generalize the concept of the complementary low-rank property to a matrix with different row and column dimensions, the following discussion is restricted to the square matrices for simplicity.

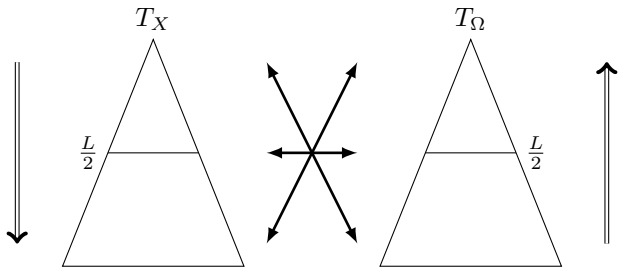

Figure 3.1: Trees of the row and column indices. Left: T_X for the row indices X. Right: T_Ω for the column indices Ω. The interaction between $A \in T_X$ and $B \in T_\Omega$ starts at the root of T_X and the leaves of T_Ω.

A simple yet important example is the Fourier matrix K of size $N \times N$, where

$$X = \Omega = \{0, \ldots, N-1\},$$
$$K = \left(\exp(2\pi\imath jk/N)\right)_{0 \leqslant j,k < N}.$$

Here the trees T_X and T_Ω are generated by bisecting the sets X and Ω recursively. Both trees have the same depth $L = \log_2 N$. For each pair of nodes $A \in T_X$ and $B \in T_\Omega$ with A at level ℓ and B at level $L - \ell$, the numerical rank of the submatrix $K_{A,B}$ for a fixed precision ϵ is bounded by a number that is independent of N and scales linearly with respect to $\log(1/\epsilon)$ [75].

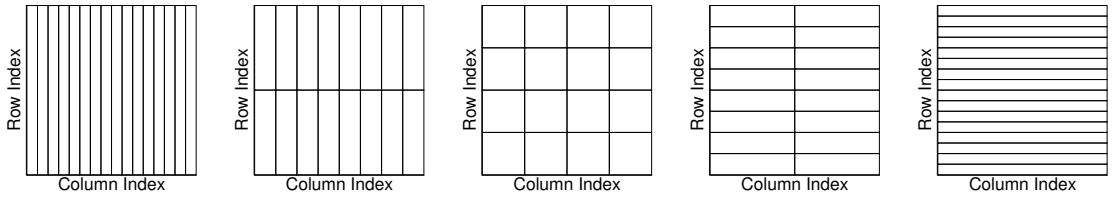

Figure 3.2: Hierarchical decomposition of the row and column indices of a 16×16 matrix. The trees T_X and T_Ω have roots containing 16 column and row indices and leaves containing a single column and row index. The rectangles above indicate the submatrices satisfying the complementary low-rank property.

The concept of complementary low-rank property can be directly applied to FIOs in one dimension. In higher dimensions, the Fourier transform is a class of FIOs that satisfies the complementary low-rank property. For other general FIOs with dimension higher than 1, they typically have a singularity at the origin $\xi = 0$ in the Ω domain and the complementary low-rank property does not hold at the range close to the origin. More effort is needed to show a special complementary low-rank property, which involves either special transform of the domain or special partition of the domain.

3.2.2 Butterfly algorithm

In this section, we review the butterfly algorithm for kernels that satisfy the complementary low-rank property.

Given an input $\{g(\xi), \xi \in \Omega\}$, the goal is to compute the potentials $\{u(x), x \in X\}$ defined by

$$u(x) = \sum_{\xi \in \Omega} K(x, \xi) g(\xi), \quad x \in X,$$

where $K(x, \xi)$ is a kernel function. For FIOs in (3.4), $K(x, \xi) = e^{2\pi\imath\Phi(x,\xi)}$ and $g(\xi) = \widehat{f}(\xi)$. Let $D_X \supset X$ and $D_\Omega \supset \Omega$ be two square domains containing X and Ω respectively. The main data structure of the butterfly algorithm is a pair of quadtrees T_X and T_Ω as in the complementary low-rank property. Having D_X as its root box, the tree T_X is built by recursive dyadic partitioning of D_X until each leaf box contains at most a certain number of points. The tree T_Ω is constructed by recursively partitioning in the same way. With the convention that a root node is at level 0, a leaf node is at level $L = O(\log N)$ under the quasi-uniformity condition about the point distributions. Throughout, we shall use A and B to denote the square boxes of

T_X and T_Ω with ℓ_A and ℓ_B denoting their levels, respectively.

For a given square B in D_Ω, define $u^B(x)$ to be the *restricted potential* over the sources $\xi \in B$

$$u^B(x) = \sum_{\xi \in B} K(x,\xi) g(\xi).$$

The low-rank property gives a compact expansion for $\{u^B(x)\}_{x\in A}$ as summing (3.5) over $\xi \in B$ with weights $g(\xi)$ gives

$$\left| u^B(x) - \sum_{t=1}^{r_\epsilon} \alpha_t^{AB}(x) \left(\sum_{\xi \in B} \beta_t^{AB}(\xi) g(\xi) \right) \right| \leqslant \left(\sum_{\xi \in B} |g(\xi)| \right) \epsilon, \quad \forall x \in A.$$

Therefore, if one can find coefficients $\{\delta_t^{AB}\}_{1\leqslant t\leqslant r_\epsilon}$ obeying

$$\delta_t^{AB} \approx \sum_{\xi \in B} \beta_t^{AB}(\xi) g(\xi), \quad 1 \leqslant t \leqslant r_\epsilon, \tag{3.6}$$

then the restricted potential $\{u^B(x)\}_{x\in A}$ admits a compact expansion

$$\left| u^B(x) - \sum_{t=1}^{r_\epsilon} \alpha_t^{AB}(x) \delta_t^{AB} \right| \leqslant \left(\sum_{\xi \in B} |g(\xi)| \right) \epsilon, \quad \forall x \in A.$$

A key point of the butterfly algorithm is that for each pair (A, B), the number of terms in the expansion is independent of N.

Computing $\{\delta_t^{AB}\}_{1\leqslant t\leqslant r_\epsilon}$ by means of (3.6) for all pairs A, B is not efficient when B is a large box because for each B there are many paired boxes A. The butterfly algorithm, however, comes with an efficient way for computing $\{\delta_t^{AB}\}_{1\leqslant t\leqslant r_\epsilon}$ recursively. The general structure of the algorithm consists of a top down traversal of T_X and a bottom up traversal of T_Ω, carried out simultaneously.

1. Construct the trees T_X and T_Ω with root nodes D_X and D_Ω.

2. Let A be the root of T_X. For each leaf box B of T_Ω, construct the expansion coefficients $\{\delta_t^{AB}\}_{1\leqslant t\leqslant r_\epsilon}$ for the potential $\{u^B(x)\}_{x\in A}$ by simply setting

$$\delta_t^{AB} = \sum_{\xi\in B} \beta_t^{AB}(\xi) g(\xi), \quad 1 \leqslant t \leqslant r_\epsilon. \tag{3.7}$$

3. For $\ell = 1, 2, \ldots, L$, visit level ℓ in T_X and level $L-\ell$ in T_Ω. For each pair (A, B) with $\ell_A = \ell$ and $\ell_B = L-\ell$, construct the expansion coefficients $\{\delta_t^{AB}\}_{1\leqslant t\leqslant r_\epsilon}$ for the potential $\{u^B(x)\}_{x\in A}$ using the low-rank representation constructed at the previous level ($\ell = 0$ is the initialization step). Let P be A's parent and C be a child of B. Throughout, we shall use the notation $C \succ B$ when C is a child of B. At level $\ell - 1$, the expansion coefficients $\{\delta_s^{PC}\}_{1\leqslant s\leqslant r_\epsilon}$ of $\{u^C(x)\}_{x\in P}$ are readily available and we have

$$\left| u^C(x) - \sum_{s=1}^{r_\epsilon} \alpha_s^{PC}(x)\delta_s^{PC} \right| \leqslant \left(\sum_{\xi\in C} |g(\xi)| \right) \epsilon, \quad \forall x \in P.$$

Since $u^B(x) = \sum_{C\succ B} u^C(x)$, the previous inequality implies that

$$\left| u^B(x) - \sum_{C\succ B} \sum_{s=1}^{r_\epsilon} \alpha_s^{PC}(x)\delta_s^{PC} \right| \leqslant \left(\sum_{\xi\in B} |g(\xi)| \right) \epsilon, \quad \forall x \in P.$$

Since $A \subset P$, the above approximation is of course true for any $x \in A$. However, since $\ell_A + \ell_B = L$, the sequence of restricted potentials $\{u^B(x)\}_{x\in A}$ also has a low-rank approximation of size r_ϵ, namely,

$$\left| u^B(x) - \sum_{t=1}^{r_\epsilon} \alpha_t^{AB}(x)\delta_t^{AB} \right| \leqslant \left(\sum_{\xi\in B} |g(\xi)| \right) \epsilon, \quad \forall x \in A.$$

Combining the last two approximations, we obtain that $\{\delta_t^{AB}\}_{1\leqslant t\leqslant r_\epsilon}$ should obey

$$\sum_{t=1}^{r_\epsilon} \alpha_t^{AB}(x)\delta_t^{AB} \approx \sum_{C>B}\sum_{s=1}^{r_\epsilon} \alpha_s^{PC}(x)\delta_s^{PC}, \quad \forall x \in A. \tag{3.8}$$

This is an over-determined linear system for $\{\delta_t^{AB}\}_{1\leqslant t\leqslant r_\epsilon}$ when $\{\delta_s^{PC}\}_{1\leqslant s\leqslant r_\epsilon, C>B}$ are available. Instead of computing $\{\delta_t^{AB}\}_{1\leqslant t\leqslant r_\epsilon}$ with a least-square method, the butterfly algorithm typically uses an efficient linear transformation approximately mapping $\{\delta_s^{PC}\}_{1\leqslant s\leqslant r_\epsilon, C>B}$ into $\{\delta_t^{AB}\}_{1\leqslant t\leqslant r_\epsilon}$. The actual implementation of this step is very much application-dependent.

4. Finally, $\ell = L$ and set B to be the root node of T_Ω. For each leaf box $A \in T_X$, use the constructed expansion coefficients $\{\delta_t^{AB}\}_{1\leqslant t\leqslant r_\epsilon}$ to evaluate $u(x)$ for each $x \in A$,

$$u(x) = \sum_{t=1}^{r_\epsilon} \alpha_t^{AB}(x)\delta_t^{AB}. \tag{3.9}$$

A schematic illustration of this algorithm in two dimension is provided in Figure 3.3. We would like to emphasize that the strict balance between the levels of the target boxes A and source boxes B maintained throughout this procedure is the key to obtain the accurate low-rank separated approximations.

However, as been mentioned in Section 3.2.1, many important multidimensional FIO kernel matrices fail to satisfy the complementary low-rank property in the entire domain $X \times \Omega$. The rest of this chapter will address this issue.

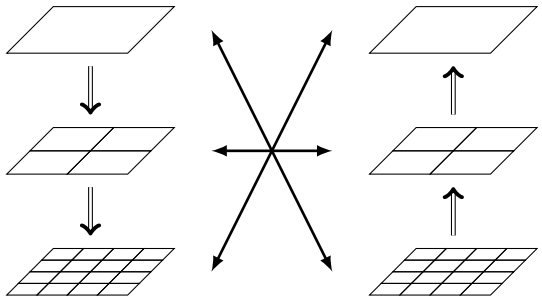

Figure 3.3: Hierarchical domain trees of the 2D butterfly algorithm. Left: T_X for the spatial domain D_X. Right: T_Ω for the frequency domain D_Ω. The interactions between subdomains $A \subset D_X$ and $B \subset D_\Omega$ are represented by left right arrow lines.

3.2.3 Polar low-rank approximations and polar butterfly algorithm

When the dimension is higher than 1, the phase function $\Phi(x, \xi)$ is usually singular at $\xi = 0$, and the numerical rank of the kernel $e^{2\pi\imath\Phi(x,\xi)}$ in a domain near or containing $\xi = 0$ is typically large. Hence, in general, $K(x, \xi) = e^{2\pi\imath\Phi(x,\xi)}$ does not satisfy the complementary low-rank property over the domain $X \times \Omega$ with quadtree structures T_X and T_Ω. To fix this problem, the polar butterfly algorithm introduces a scaled polar transformation on Ω,

$$\xi = (\xi_1, \xi_2) = \frac{\sqrt{2}}{2} n p_1 \cdot (\cos 2\pi p_2, \sin 2\pi p_2), \tag{3.10}$$

for $\xi \in \Omega$ and $p = (p_1, p_2) \in [0, 1]^2$. We use p to denote a point in the polar coordinate and P for the set of all points p transformed from $\xi \in \Omega$. This transformation gives rise to a new phase function $\Psi(x, p)$ in variables x and p satisfying

$$\Psi(x, p) = \frac{1}{n}\Phi(x, \xi(p)) = \frac{\sqrt{2}}{2}\Phi\left(x, (\cos 2\pi p_2, \sin 2\pi p_2)\right) \cdot p_1, \tag{3.11}$$

where the last equality comes from the fact that $\Phi(x,\xi)$ is homogeneous of degree 1 in ξ. This new phase function $\Psi(x,p)$ is smooth in the entire domain $X \times P$ and the FIO in (3.4) takes the new form

$$u(x) = \sum_{p\in P} e^{2\pi\imath n\Psi(x,p)} g(p), \quad x \in X. \tag{3.12}$$

The transformation (3.10) ensures that $X \times P \subset [0,1]^2 \times [0,1]^2$. By partitioning $[0,1]^2$ recursively, we can construct two quadtrees T_X and T_P of depth $L = O(\log n)$ for X and P, respectively. The following theorem is a rephrased version of Theorem 3.1 in [17] that shows analytically the complementary low-rank property of $e^{2\pi\imath n\Psi(x,p)}$ in the $X \times P$ domain.

Theorem 3.2.2. *Suppose A is a node in T_X at level ℓ and B is a node in T_P at level $L-\ell$. Given an FIO kernel function $e^{2\pi\imath n\Psi(x,p)}$ with a real-analytic phase function in the joint variables x and p, there exist $\epsilon_0 > 0$ and $n_0 > 0$ such that for any positive $\epsilon \leqslant \epsilon_0$ and $n \geqslant n_0$, there exist r_ϵ pairs of functions $\{\alpha_t^{A,B}(x), \beta_t^{A,B}(p)\}_{1\leqslant t\leqslant r_\epsilon}$ satisfying*

$$\left| e^{2\pi\imath n\Psi(x,p)} - \sum_{t=1}^{r_\epsilon} \alpha_t^{A,B}(x)\beta_t^{A,B}(p) \right| \leqslant \epsilon,$$

for $x \in A$ and $p \in B$ with $r_\epsilon \lesssim \log^4(1/\epsilon)$.

The polar butterfly algorithm

Based on Theorem 3.2.2, the polar butterfly algorithm traverses upward in T_P and downward in T_X simultaneously and visits the low-rank submatrices $K_{A,B} = \{K(x_i,p_j) = e^{2\pi\imath n\Psi(x_i,p_j)}\}_{x_i\in A, p_j\in B}$ for pairs (A,B) in $T_X \times T_P$. Replacing notations Ω by P and ξ by p in the above algorithm results the polar butterfly algorithm. The algorithm is asymptotically very efficient: for a given input vector $g(p)$ for $p \in P$, it

evaluates (3.12) in $O(N \log N)$ operations with $O(N)$ memory. We refer the readers to [17] for a detailed description of this algorithm.

Though having optimal complexity, this polar-Cartesian transformation comes with several drawbacks, which results in a large pre-factor of the computational complexity. First, due to the polar grid in the frequency domain, the points in P for the butterfly algorithm are irregularly distributed and a separate Chebyshev interpolation matrix is required for the evaluation at each point. In order to avoid the memory bottleneck from storing these interpolation matrices, the polar butterfly algorithm generates these interpolation matrices on-the-fly during the evaluation. This turns out to be expensive in the operation count. Second, since the amplitude and phase functions are often written in the Cartesian coordinates, the polar butterfly algorithm applies the polar-Cartesian transformation for each kernel evaluation. Finally, in order to maintain a reasonable accuracy, the polar butterfly algorithm divides the frequency domain into multiple parts and applies the same butterfly algorithm to each part separately. This also increases the actual running time by a non-trivial constant factor.

Those drawbacks of the polar butterfly algorithm motivate us to propose a new multiscale butterfly algorithm using a Cartesian grid both in the spatial and frequency domain.

3.3 Multiscale low-rank approximations

In order to introduce the new multiscale butterfly algorithm that significantly reduces the pre-factor, one would require the existence of the following low-rank separated

representation

$$e^{2\pi\imath\Phi(x,\xi)} \approx \sum_{t=1}^{r_\epsilon} \alpha_t^{AB}(x)\beta_t^{AB}(\xi)$$

for any pair of boxes A and B such that $\ell_A + \ell_B = L$. If the frequency domain B is well-separated from the origin $\xi = 0$ in a relative sense, one can prove a low-rank separated representation.

In order to make it more precise, for two given squares $A \subset X$ and $B \subset \Omega$, we introduce a new function called the residue phase function

$$R^{AB}(x,\xi) := \Phi(x,\xi) - \Phi(c_A,\xi) - \Phi(x,c_B) + \Phi(c_A,c_B), \tag{3.13}$$

where c_A and c_B are the centers of A and B respectively. Using this new definition, the kernel can be written as

$$e^{2\pi\imath\Phi(x,\xi)} = e^{2\pi\imath\Phi(c_A,\xi)}e^{2\pi\imath\Phi(x,c_B)}e^{-2\pi\imath\Phi(c_A,c_B)}e^{2\pi\imath R^{AB}(x,\xi)}. \tag{3.14}$$

Below is our main theorem in this chapter. It provides theoretical support to the low-rank approximations we used in the multiscale butterfly algorithm. In this theorem, w_A and w_B denote the side lengths of A and B, respectively; $\operatorname{dist}(B,0)$ denotes the distance between the square B and the origin 0 in the frequency domain. The distance is given by $\operatorname{dist}(B,0) = \min_{\xi\in B}\|\xi - 0\|$. Throughout this dissertation, when we write $O(\cdot)$, $\lesssim$ and $\gtrsim$, the implicit constant is independent of n and ϵ.

Theorem 3.3.1. *Suppose $\Phi(x,\xi)$ is a phase function that is real analytic for x and ξ away from $\xi = 0$. There exists positive constants ϵ_0 and n_0 such that the following is true. Let A and B be two squares in X and Ω, respectively, obeying $w_A w_B \leqslant 1$ and*

dist$(B, 0) \geqslant \frac{n}{4}$. *For any positive* $\epsilon \leqslant \epsilon_0$ *and* $n \geqslant n_0$, *there exists an approximation*

$$\left| e^{2\pi\imath R^{AB}(x,\xi)} - \sum_{t=1}^{r_\epsilon} \tilde{\alpha}_t^{AB}(x)\tilde{\beta}_t^{AB}(\xi) \right| \leqslant \epsilon$$

for $x \in A$ *and* $\xi \in B$ *with* $r_\epsilon \lesssim \log^4(\frac{1}{\epsilon})$. *Moreover,*

- *when* $w_B \leqslant \sqrt{n}$, *the functions* $\{\tilde{\beta}_t^{AB}(\xi)\}_{1\leqslant t\leqslant r_\epsilon}$ *can all be chosen as monomials in* $(\xi - c_B)$ *with a degree not exceeding a constant times* $\log^2(1/\epsilon)$,
- *and when* $w_A \leqslant 1/\sqrt{n}$, *the functions* $\{\tilde{\alpha}_t^{AB}(x)\}_{1\leqslant t\leqslant r_\epsilon}$ *can all be chosen as monomials in* $(x - c_A)$ *with a degree not exceeding a constant times* $\log^2(1/\epsilon)$.

Proof. Since $w_A w_B \leqslant 1$, we either have $w_A \leqslant 1/\sqrt{n}$ or $w_B \leqslant \sqrt{n}$ or both.

Let us first consider the case $w_B \leqslant \sqrt{n}$. Then

$$\begin{aligned} R^{AB}(x,\xi) &= \Phi(x,\xi) - \Phi(c_A,\xi) - \Phi(x,c_B) + \Phi(c_A,c_B) \\ &= [\Phi(x,\xi) - \Phi(c_A,\xi)] - [\Phi(x,c_B) - \Phi(c_A,c_B)] \\ &= H(x,\xi) - H(x,c_B), \end{aligned}$$

where $H(x,\xi) := \Phi(x,\xi) - \Phi(c_A,\xi)$. The function $R^{AB}(x,\xi)$ inherits the smoothness from $\Phi(x,\xi)$. Applying the multi-variable Taylor expansion of degree k in ξ centered at c_B gives

$$R^{AB}(x,\xi) = \sum_{1\leqslant|i|<k} \frac{\partial_\xi^i H(x,c_B)}{i!}(\xi - c_B)^i + \sum_{|i|=k} \frac{\partial_\xi^i H(x,\xi^*)}{i!}(\xi - c_B)^i, \tag{3.15}$$

where ξ^* is a point in the segment between c_B and ξ. Here $i = (i_1, i_2)$ is a multi-index with $i! = i_1! i_2!$, and $|i| = i_1 + i_2$. Let us first choose the degree k so that the second sum in (3.15) is bounded by $\epsilon/(4\pi)$. For each i with $|i| = k$, the definition of $H(x,\xi)$

gives

$$\partial_\xi^i H(x,\xi^*) = \sum_{|j|=1} \partial_x^j \partial_\xi^i \Phi(x^*,\xi^*)(x-c_A)^j,$$

for some point x^* in the segment between c_A and x. Using the fact that $\Phi(x,\xi)$ is real-analytic over $|\xi|=1$ gives that there exists a radius R such that

$$|\partial_x^j \partial_\xi^i \Phi(x,\xi)| \leqslant Ci!j!\frac{1}{R^{|i+j|}} = Ci!j!\frac{1}{R^{k+1}},$$

for ξ with $|\xi|=1$. Here the constant C is independent of k. Since $\Phi(x,\xi)$ is homogeneous of degree 1 in ξ, a scaling argument shows that

$$|\partial_x^j \partial_\xi^i \Phi(x^*,\xi^*)| \leqslant Ci!j!\frac{1}{R^{k+1}|\xi^*|^{k-1}}.$$

Since $\operatorname{dist}(B,0) \geqslant n/4$ and $w_A w_B \leqslant 1$, we have

$$\left|\frac{\partial_\xi^i H(x,\xi^*)}{i!}(\xi-c_B)^i\right| \leqslant \frac{2Ci!j!}{i!}\frac{1}{R^{k+1}|\xi^*|^{k-1}} w_A w_B^k \leqslant \frac{2C}{R^{k+1}}\left(\frac{4}{\sqrt{n}}\right)^{k-1}.$$

Combining this with (3.15) gives

$$\begin{aligned}&\left|R^{AB}(x,\xi) - \sum_{1\leqslant|i|<k}\frac{\partial_\xi^i H(x,c_B)}{i!}(\xi-c_B)^i\right| \\ =&\left|\sum_{|i|=k}\frac{\partial_\xi^i H(x,\xi^*)}{i!}(\xi-c_B)^i\right| \leqslant \frac{2C(k+1)}{R^{k+1}}\left(\frac{4}{\sqrt{n}}\right)^{k-1}.\end{aligned}$$

Therefore, for a sufficient large $n_0(R)$, if $n > n_0(R)$, choosing $k = k_\epsilon = O(\log(1/\epsilon))$ ensures that the difference is bounded by $\epsilon/(4\pi)$.

The special case $k = 1$ results in the following bound for $R^{AB}(x, \xi)$

$$|R^{AB}(x, \xi)| \leqslant \frac{4C}{R^2}.$$

To simplify the notation, we define

$$R^{AB}_{\epsilon}(x, \xi) := \sum_{1 \leqslant |i| < k_\epsilon} \frac{\partial^i_\xi H(x, c_B)}{i!} (\xi - c_B)^i,$$

i.e., the first sum on the right hand side of (3.15) with $k = k_\epsilon$. The choice of k_ϵ together with (3.15) implies the bound

$$|R^{AB}_{\epsilon}(x, \xi)| \leqslant \frac{4C}{R^2} + \epsilon.$$

Since $R^{AB}_{\epsilon}(x, \xi)$ is bounded, a direct application of Lemma 3.2 of [17] gives

$$\left| e^{2\pi\imath R^{AB}_{\epsilon}(x,\xi)} - \sum_{p=0}^{d_\epsilon} \frac{(2\pi\imath R^{AB}_{\epsilon}(x, \xi))^p}{p!} \right| \leqslant \epsilon/2, \tag{3.16}$$

where $d_\epsilon = O(\log(1/\epsilon))$. Since $R^{AB}_{\epsilon}(x, \xi)$ is a polynomial in $(\xi - c_B)$, the sum in (3.16) is also a polynomial in $(\xi - c_B)$ with degree bounded by $k_\epsilon d_\epsilon = O(\log^2(1/\epsilon))$. Since our problem is in 2D, there are at most $O(\log^4(1/\epsilon))$ possible monomial in $(\xi - c_B)$ with degree bounded by $k_\epsilon d_\epsilon$. Grouping the terms with the same multi-index in ξ results in an $O(\log^4(1/\epsilon))$ term ϵ-accurate separated approximation for $e^{2\pi\imath R^{AB}_{\epsilon}(x,\xi)}$ with the factors $\{\tilde{\beta}^{AB}_t(\xi)\}_{1 \leqslant t \leqslant r_\epsilon}$ being monomials of $(\xi - c_B)$.

Finally, from the inequality $|e^{\imath a} - e^{\imath b}| \leqslant |a - b|$, it is clear that a separated approximation for $e^{2\pi\imath R^{AB}_{\epsilon}(x,\xi)}$ with accuracy $\epsilon/2$ is also one for $e^{2\pi\imath R^{AB}(x,\xi)}$ with accuracy $\epsilon/2 + \epsilon/2 = \epsilon$. This completes the proof for the case $w_B \leqslant \sqrt{n}$.

The proof for the case $w_A \leqslant 1/\sqrt{n}$ is similar. The only difference is that we now group with

$$R^{AB}(x,\xi) = [\Phi(x,\xi) - \Phi(x,c_B)] - [\Phi(c_A,\xi) - \Phi(c_A,c_B)]$$

and apply the multivariable Taylor expansion in x centered at c_A instead. This results an $O(\log^4(1/\epsilon))$ term ϵ-accurate separated approximation for $e^{2\pi\imath R^{AB}(x,\xi)}$ with the factors $\{\tilde{\alpha}_t^{AB}(x)\}_{1\leqslant t\leqslant r_\epsilon}$ being monomials of $(x - c_A)$. □

Though the above proof is constructive, it is cumbersome to construct the separated approximation this way. On the other hand, the proof shows that when $w_B \leqslant \sqrt{n}$, the ξ-dependent factors in the low-rank approximation of $e^{2\pi\imath R^{AB}(x,\xi)}$ can be monomials in $(\xi - c_B)$. Similarly, when $w_A \leqslant 1/\sqrt{n}$, the x-dependent factors are monomials in $(x - c_A)$. This suggests to use Chebyshev interpolation in x when $w_A \leqslant 1/\sqrt{n}$ and in ξ when $w_B \leqslant \sqrt{n}$. For this purpose, we associate with each box a Chebyshev grid as follows.

For a fixed integer q, the Chebyshev grid of order q on $[-1/2, 1/2]$ is defined by

$$\left\{ z_i = \frac{1}{2}\cos\left(\frac{i\pi}{q-1}\right) \right\}_{0\leqslant i\leqslant q-1}.$$

A tensor-product grid *adapted to a square* with center c and side length w is then defined via shifting and scaling as

$$\{c + w(z_i, z_j)\}_{i,j=0,1,\ldots,q-1}$$

In what follows, M_t^B is the 2D Lagrange interpolation polynomial on the Chebyshev grid adapted to the square B (i.e., using $c = c_B$ and $w = w_B$).

Theorem 3.3.2. *Let A and B be as in Theorem 3.3.1. Then for any $\epsilon \leqslant \epsilon_0$ and $n \geqslant n_0$ where ϵ_0 and n_0 are the constants in Theorem 3.3.1, there exists $q_\epsilon \lesssim \log^2(1/\epsilon)$ such that*

- *when $w_B \leqslant \sqrt{n}$, the Lagrange interpolation of $e^{2\pi\imath R^{AB}(x,\xi)}$ in ξ on a $q_\epsilon \times q_\epsilon$ Chebyshev grid $\{g_t^B\}_{1\leqslant t\leqslant r_\epsilon}$ adapted to B obeys*

$$\left| e^{2\pi\imath R^{AB}(x,\xi)} - \sum_{t=1}^{r_\epsilon} e^{2\pi\imath R^{AB}(x,g_t^B)} M_t^B(\xi) \right| \leqslant \epsilon, \quad \forall x \in A, \forall \xi \in B, \tag{3.17}$$

- *when $w_A \leqslant 1/\sqrt{n}$, the Lagrange interpolation of $e^{2\pi\imath R^{AB}(x,\xi)}$ in x on a $q_\epsilon \times q_\epsilon$ Chebyshev grid $\{g_t^A\}_{1\leqslant t\leqslant r_\epsilon}$ adapted to A obeys*

$$\left| e^{2\pi\imath R^{AB}(x,\xi)} - \sum_{t=1}^{r_\epsilon} M_t^A(x) e^{2\pi\imath R^{AB}(g_t^A,\xi)} \right| \leqslant \epsilon, \quad \forall x \in A, \forall \xi \in B. \tag{3.18}$$

Both (3.17) *and* (3.18) *provide a low-rank approximation with $r_\epsilon = q_\epsilon^2 \lesssim \log^4(1/\epsilon)$ terms.*

The proof for this follows exactly the one of Theorem 3.3 of [17].

Finally, we are ready to construct the low-rank approximation for the kernel $e^{2\pi\imath\Phi(x,\xi)}$, i.e.,

$$e^{2\pi\imath\Phi(x,\xi)} \approx \sum_{t=1}^{r_\epsilon} \alpha_t^{AB}(x)\beta_t^{AB}(\xi). \tag{3.19}$$

When $w_B \leqslant \sqrt{n}$, we multiply (3.17) with $e^{2\pi\imath\Phi(c_A,\xi)} e^{2\pi\imath\Phi(x,c_B)} e^{-2\pi\imath\Phi(c_A,c_B)}$, which gives that $\forall x \in A, \forall \xi \in B$

$$\left| e^{2\pi\imath\Phi(x,\xi)} - \sum_{t=1}^{r_\epsilon} e^{2\pi\imath\Phi(x,g_t^B)} \left(e^{-2\pi\imath\Phi(c_A,g_t^B)} M_t^B(\xi) e^{2\pi\imath\Phi(c_A,\xi)} \right) \right| \leqslant \epsilon.$$

In terms of the notations in (3.19), the expansion functions are given by

$$\alpha_t^{AB}(x) = e^{2\pi\imath\Phi(x,g_t^B)}, \quad \beta_t^{AB}(\xi) = e^{-2\pi\imath\Phi(c_A,g_t^B)} M_t^B(\xi) e^{2\pi\imath\Phi(c_A,\xi)}, \quad 1 \leqslant t \leqslant r_\epsilon. \quad (3.20)$$

This is a special interpolant of the function $e^{2\pi\imath\Phi(x,\xi)}$ in the ξ variable, which prefactors the oscillation, performs the interpolation, and then remodulates the outcome. When $w_A \leqslant 1/\sqrt{n}$, multiply (3.18) with $e^{2\pi\imath\Phi(c_A,\xi)} e^{2\pi\imath\Phi(x,c_B)} e^{-2\pi\imath\Phi(c_A,c_B)}$ and obtain that $\forall x \in A, \forall \xi \in B$

$$\left| e^{2\pi\imath\Phi(x,\xi)} - \sum_{t=1}^{r_\epsilon} \left(e^{2\pi\imath\Phi(x,c_B)} M_t^A(x) e^{-2\pi\imath\Phi(g_t^A,c_B)} \right) e^{2\pi\imath\Phi(g_t^A,\xi)} \right| \leqslant \epsilon.$$

The expansion functions are now

$$\alpha_t^{AB}(x) = e^{2\pi\imath\Phi(x,c_B)} M_t^A(x) e^{-2\pi\imath\Phi(g_t^A,c_B)}, \quad \beta_t^{AB}(\xi) = e^{2\pi\imath\Phi(g_t^A,\xi)}, \quad 1 \leqslant t \leqslant r_\epsilon. \quad (3.21)$$

Due to the presence of the demodulation and remodulation steps in the definitions (3.20) and (3.21), we refer to them as *oscillatory Chebyshev interpolations*.

3.4 Multiscale butterfly algorithm

In this section, we combine the multiscale low-rank approximations described in Section 3.3 with the butterfly algorithm in Section 3.2.2 to propose a multiscale butterfly algorithm for the FIOs in two dimensions. Extension to higher dimensions is straightforward.

To deal with the singularity of the kernel $\Phi(x,\xi)$ at $\xi = 0$, we hierarchically decompose the frequency domain into a union of non-overlapping Cartesian coronas

with a common center $\xi = 0$ (see Figure 3.4). More precisely, define

$$\Omega_j = \left\{(n_1, n_2) : \frac{n}{2^{j+1}} < \max(|n_1|, |n_2|) \leqslant \frac{n}{2^j}\right\} \cap \Omega$$

for $j = 1, \ldots, \log n - s$, where s is just a small constant integer. The domain $\Omega_d = \Omega \backslash \cup_j \Omega_j$ is the remaining square grid at the center of constant size. Following this decomposition of the frequency domain, one can write (3.4) accordingly as

$$(Lf)(x) = \sum_j \left(\sum_{\xi \in \Omega_j} e^{2\pi\imath\Phi(x,\xi)} \widehat{f}(\xi) \right) + \sum_{\xi \in \Omega_d} e^{2\pi\imath\Phi(x,\xi)} \widehat{f}(\xi). \tag{3.22}$$

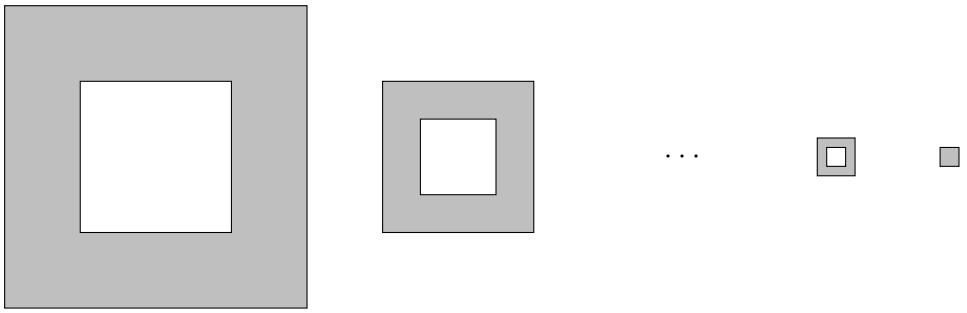

Figure 3.4: This figure shows the frequency domain decomposition of Ω. Each sub-domain Ω_j, $j = 1, \ldots, \log n - s$, is a corona and Ω_d is a small square domain near the origin.

The kernel function of (3.22) is smooth in each sub-domain Ω_j and a classical butterfly algorithm as described in Section 3.2.2 can be applied to evaluate the contribution from Ω_j. In contrast to the polar butterfly algorithm that works in the polar coordinates for Ω, we refer to this one as the *single-scale butterfly algorithm*. For the center square Ω_d, since it contains only a constant number of points, a direct summation is used. Because of the multiscale nature of the frequency domain decomposition,

we refer to this algorithm as **the multiscale butterfly algorithm**. As we shall see, the computational and memory complexity of the multiscale butterfly algorithm are still $O(N \log N)$ and $O(N)$, respectively. On the other hand, the pre-factors are much smaller, since the multiscale butterfly is based on the Cartesian grids and requires no polar-Cartesian transformation.

3.4.1 Single-scale butterfly algorithm

To make it more explicit, let us first consider the interaction between (X, Ω_1), with the multiscale low-rank approximation implemented using the oscillatory Chebyshev interpolation discussed in Section 3.3.

1. *Preliminaries.* Construct two quadtrees T_X and T_{Ω_1} for X and Ω_1 by uniform hierarchical partitioning. Let b be a constant greater than or equal to 4 and define $n_1 = n$.

2. *Initialization.* For each square $A \in T_X$ of width $1/b$ and each square $B \in T_{\Omega_1}$ of width b, the low-rank approximation functions are

$$\alpha_t^{AB}(x) = e^{2\pi\imath\Phi(x, g_t^B)}, \tag{3.23}$$

$$\beta_t^{AB}(\xi) = e^{-2\pi\imath\Phi(c_A, g_t^B)} M_t^B(\xi) e^{2\pi\imath\Phi(c_A, \xi)}, \quad 1 \leqslant t \leqslant r_\epsilon. \tag{3.24}$$

   Hence, we can define the expansion weights $\{\delta_t^{AB}\}_{1 \leqslant t \leqslant r_\epsilon}$ with

$$\delta_t^{AB} := \sum_{\xi \in B} \beta_t^{AB}(\xi) \widehat{f}(\xi) = e^{-2\pi\imath\Phi(c_A, g_t^B)} \sum_{\xi \in B} \left(M_t^B(\xi) e^{2\pi\imath\Phi(c_A, \xi)} \widehat{f}(\xi) \right). \tag{3.25}$$

3. *Recursion.* Go up in tree T_{Ω_1} and down in tree T_X at the same time until we reach the level such that $w_B = \sqrt{n_1}$. At each level, visit all the pairs

(A, B). We apply the Chebyshev interpolation in variable ξ and still define the approximation functions given in (3.23). Let $\{\delta_s^{PC}\}_{1\leqslant s\leqslant r_\epsilon}$ denote the expansion coefficients available in previous steps, where P is A's parent, C is a child of B, and s indicates the Chebyshev grid points in previous domain pairs. We define the new expansion coefficients $\{\delta_t^{AB}\}_{1\leqslant t\leqslant r_\epsilon}$ as

$$\delta_t^{AB} := e^{-2\pi\imath\Phi(c_A,g_t^B)} \sum_{C>B} \sum_{s=1}^{r_\epsilon} M_t^B(g_s^C) e^{2\pi\imath\Phi(c_A,g_s^C)} \delta_s^{PC}, \tag{3.26}$$

where we recall that the notation $C \succ B$ means that C is a child of B.

4. *Switch.* For the levels visited, the Chebyshev interpolation is applied in variable ξ, while the interpolation is applied in variable x for levels $l > \log(n_1)/2$. Hence, we are switching the interpolation method at this step. Now we are still working on level $l = \log(n_1)/2$ and the same domain pairs (A, B) in the last step. Let δ_s^{AB} denote the expansion weights obtained by Chebyshev interpolation in variable ξ in the last step. Correspondingly, $\{g_s^B\}_s$ are the grid points in B in the last step. We take advantage of the interpolation in variable x in A and generate grid points $\{g_t^A\}_{1\leqslant t\leqslant r_\epsilon}$ in A. Then we can define new expansion weights

$$\delta_t^{AB} := \sum_{s=1}^{r_\epsilon} e^{2\pi\imath\Phi(g_t^A,g_s^B)} \delta_s^{AB}.$$

5. *Recursion.* Go up in tree T_{Ω_1} and down in tree T_X at the same time until we reach the level such that $w_B = n_1/b$. We construct the approximation functions by Chebyshev interpolation in variable x as follows:

$$\alpha_t^{AB}(x) = e^{2\pi\imath\Phi(x,c_B)} M_t^A(x) e^{-2\pi\imath\Phi(g_t^A,c_B)}, \quad \beta_t^{AB}(\xi) = e^{2\pi\imath\Phi(g_t^A,\xi)}. \tag{3.27}$$

We define the new expansion coefficients $\{\delta_t^{AB}\}_{1\leqslant t\leqslant r_\epsilon}$ as

$$\delta_t^{AB} := \sum_{C>B} e^{2\pi\imath\Phi(g_t^A,c_C)} \sum_{s=1}^{r_\epsilon} \left(M_s^P(g_t^A) e^{-2\pi\imath\Phi(g_s^P,c_C)} \delta_s^{PC} \right), \tag{3.28}$$

where again P is A's parent and C is a child box of B.

6. *Termination.* Finally, we reach the level that $w_B = n_1/b$. For each B on this level and for each square $A \in T_X$ of width b/n_1, we apply the approximation functions given by (3.27) and obtain

$$u^B(x) := e^{2\pi\imath\Phi(x,c_B)} \sum_{t=1}^{r_\epsilon} \left(M_t^A(x) e^{-2\pi\imath\Phi(g_t^A,c_B)} \delta_t^{AB} \right) \tag{3.29}$$

for each $x \in A$. Finally, summing over all B on this level, we have

$$u^{\Omega_1}(x) := \sum_B u^B(x) \tag{3.30}$$

for each $x \in A$.

We would like to emphasize that the center part of the tree T_{Ω_j} is always empty since Ω_j is a corona. Accordingly, the algorithm skips this empty part.

For a general Ω_j, the interaction between (X, Ω_j) follows a similar algorithm, except that we replace Ω_1 with Ω_j, $u^{\Omega_1}(x)$ with $u^{\Omega_j}(x)$, n_1 with $n_j = n/2^{j-1}$, and stop at the level that $w_B = n_j/b$.

Finally, (3.4) is evaluated via

$$(Lf)(x) = u^{\Omega_d}(x) + \sum_j u^{\Omega_j}(x). \tag{3.31}$$

The multiscale butterfly algorithm can be adapted to non-uniform grid points in

X and Ω. The only difference is that the oscillatory Chebyshev interpolation for low-rank approximation uses non-uniform grid points. In this case, we have to generate different interpolation matrices when we visit different leaf domain pairs $A \times B$, i.e., either A is a leaf box of T_X or B is a leaf box of T_Ω. This will end up with extra operation and memory requirement. But careful calculation shows that the overall operation and memory complexity remains the same.

3.4.2 Complexity analysis

The cost of evaluating the term of Ω_d takes at most $O(n^2)$ steps since $|\Omega_d| = O(1)$. Let us now consider the cost of the terms associated with $\{\Omega_j\}$.

For the interaction between X and Ω_1, the computation consists of two parts: the recursive evaluation of $\{\delta_t^{AB}\}$ and the final evaluation of $u^{\Omega_1}(x)$. The recursive part takes $O(q^3 n^2 \log n)$ since there are at most $O(n^2 \log n)$ pairs of squares (A, B) and the evaluation of $\{\delta_t^{AB}\}$ for each pair takes $O(q^3)$ steps via dimension-wise Chebyshev interpolation. The final evaluation of $u^{\Omega_1}(x)$ clearly takes $O(q^2 n^2)$ steps as we spend $O(q^2)$ on each point $x \in X$.

For the interaction between X and Ω_j, the analysis is similar. The recursive part takes now $O(q^3 n_j^2 \log n_j)$ steps (with $n_j = n/2^{j-1}$) as there are at most $O(n_j^2 \log n_j)$ pairs of squares involved. The final evaluation still takes $O(q^2 n^2)$ steps.

Summing these contributions together results in the total computational complexity

$$O(q^3 n^2 \log n) + O(q^2 n^2 \log n) = O(q^3 n^2 \log n) = O(r_\epsilon^{3/2} n^2 \log n).$$

The multiscale butterfly algorithm is also highly efficient in terms of memory as the Cartesian butterfly algorithm is applied sequentially to evaluate (3.29) for each Ω_j. Although the overall memory complexity is still $O(n^2)$, the peak memory could

be significantly reduced since only $\frac{1}{b^2}$ memory of the original Cartesian butterfly algorithm is used to evaluate the FIO in Ω_j.

3.5 Numerical results

This section presents several numerical examples to demonstrate the effectiveness of the multiscale butterfly algorithm introduced above. In truth, FIOs usually have non-constant amplitude functions. Nevertheless, the main computational difficulty is the oscillatory phase term. We refer to [17] for detailed fast algorithms to deal with non-constant amplitude functions. Our MATLAB implementation can be found on the authors' personal homepages. The numerical results were obtained on a desktop with a 3.5 GHz CPU and 32 GB of memory. Let $\{u^d(x), x \in X\}$, $\{u^m(x), x \in X\}$ and $\{u^p(x), x \in X\}$ be the results of a discrete FIO computed by a direct matrix-vector multiplication, the multiscale butterfly algorithm and the polar butterfly algorithm [17], respectively. To report on the accuracy, we randomly select a set S of 256 points from X and evaluate the relative errors of the multiscale butterfly algorithm and the polar butterfly algorithm by

$$\epsilon^m = \sqrt{\frac{\sum_{x\in S}|u^d(x) - u^m(x)|^2}{\sum_{x\in S}|u^d(x)|^2}} \text{ and } \epsilon^p = \sqrt{\frac{\sum_{x\in S}|u^d(x) - u^p(x)|^2}{\sum_{x\in S}|u^d(x)|^2}}. \tag{3.32}$$

According to the description of the multiscale butterfly algorithm in Section 3.4, we recursively divide Ω into $\Omega_j, j = 1, 2, \ldots, \log n - s$, where s is 5 in the following examples. This means that the center square Ω_d is of size $2^5 \times 2^5$ and the interaction from Ω_d is evaluated via a direct matrix-vector multiplication. Suppose q_ϵ is the number of Chebyshev points in each dimension. There is no sense to use butterfly algorithms to construct $\{\delta_t^{AB}\}$ when the number of points in B is fewer than q_ϵ^2.

Hence, the recursion step in butterfly algorithms starts from the squares B that are a couple of levels away from the bottom of T_Ω such that each square contains at least q_ϵ^2 points. Similarly, the recursion stops at the squares in T_X that are the same number of levels away from the bottom. In the following examples, we start from level $\log n - 3$ and stop at level 3 (corresponding to $b = 2^3$ defined in Section 3.4) which matches with q_ϵ (4 to 11).

In order to make a fair comparison, we compare the MATLAB versions of the polar butterfly algorithm and the multiscale butterfly algorithm. Hence, the running time of the polar butterfly algorithm here is slower than the one in [17], which was implemented in C++.

Example 1. This example is a generalized Radon transform whose kernel is given by

$$\begin{aligned} \Phi(x,\xi) &= x\cdot\xi + \sqrt{c_1^2(x)\xi_1^2 + c_2^2(x)\xi_2^2}, \\ c_1(x) &= (2+\sin(2\pi x_1)\sin(2\pi x_2))/3, \\ c_2(x) &= (2+\cos(2\pi x_1)\cos(2\pi x_2))/3. \end{aligned} \tag{3.33}$$

We assume the amplitude of this example is a constant 1. Now the FIO models an integration over ellipses where $c_1(x)$ and $c_2(x)$ are the axis lengths of the ellipse centered at the point $x \in X$. Table 3.1 summarize the results of this example given by the polar butterfly algorithm and the multiscale butterfly algorithm.

Example 2. Next, we provide an FIO example with a smooth amplitude function,

$$u(x) = \sum_{\xi\in\Omega} a(x,\xi) e^{2\pi\imath\Phi(x,\xi)} \widehat{f}(\xi), \tag{3.34}$$

Multiscale Butterfly			Polar Butterfly			
n, q_ϵ	ϵ^m	$T_m(sec)$	n, q_ϵ	ϵ^p	$T_p(sec)$	T_p/T_m
256,5	7.89e-02	6.96e+01	256,5	4.21e-02	4.84e+02	6.96e+00
512,5	9.01e-02	3.62e+02	512,5	5.54e-02	2.34e+03	6.46e+00
1024,5	9.13e-02	1.81e+03	1024,5	4.26e-02	1.14e+04	6.31e+00
2048,5	9.47e-02	8.79e+03	2048,5	-	-	-
256,7	6.95e-03	8.20e+01	256,7	5.66e-03	5.97e+02	7.28e+00
512,7	8.43e-03	4.16e+02	512,7	5.89e-03	2.82e+03	6.79e+00
1024,7	8.45e-03	2.03e+03	1024,7	4.84e-03	1.35e+04	6.64e+00
2048,7	8.42e-03	1.04e+04	2048,7	-	-	-
256,9	3.90e-04	1.10e+02	256,9	8.25e-04	7.74e+02	7.04e+00
512,9	3.42e-04	5.39e+02	512,9	6.78e-04	3.57e+03	6.61e+00
1024,9	7.61e-04	2.74e+03	1024,9	4.18e-04	1.67e+04	6.09e+00
2048,9	4.82e-04	1.25e+04	2048,9	-	-	-
256,11	2.15e-05	1.84e+02	256,11	3.69e-05	1.15e+03	6.27e+00
512,11	1.89e-05	8.60e+02	512,11	5.53e-05	5.10e+03	5.93e+00
1024,11	1.96e-05	4.27e+03	1024,11	2.042e-05	2.30e+04	5.39e+00
2048,11	1.50e-05	1.82e+04	2048,11	-	-	-

Table 3.1: Comparison of the multiscale butterfly algorithm and the polar butterfly algorithm for the phase function in (3.33). T_m is the running time of the multiscale butterfly algorithm; T_a is the running time of the polar butterfly algorithm; and T_m/T_p is the speedup factor.

where the amplitude and phase functions are given by

$$a(x,\xi) = (J_0(2\pi\rho(x,\xi)) + \imath Y_0(2\pi\rho(x,\xi)))e^{-\pi\imath\rho(x,\xi)},$$
$$\Phi(x,\xi) = x\cdot\xi + \rho(x,\xi),$$
$$\rho(x,\xi) = \sqrt{c_1^2(x)\xi_1^2 + c_2^2(x)\xi_2^2},$$
$$c_1(x) = (2+\sin(2\pi x_1)\sin(2\pi x_2))/3,$$
$$c_2(x) = (2+\cos(2\pi x_1)\cos(2\pi x_2))/3.$$

Here, J_0 and Y_0 are Bessel functions of the first and second kinds. We refer to [16] for more details of the derivation of these formulas. As discussed in [17], we compute the low rank approximation of the amplitude functions $a(x,\xi)$ first:

$$a(x,\xi) \approx \sum_{t=1}^{s_\epsilon} g_t(x)h_t(\xi).$$

In the second step, we apply the multiscale butterfly algorithm to compute

$$u_t(x) = \sum_{\xi\in\Omega} e^{2\pi\imath\Phi(x,\xi)}\widehat{f}(\xi)h_t(\xi),$$

and sum up all $g_t(x)u_t(x)$ to evaluate

$$u(x) = \sum_t g_t(x)u_t(x).$$

Table 3.2 summarizes the results of this example given by the direct method and the multiscale butterfly algorithm.

Note that the accuracy of the multiscale butterfly algorithm is well controlled by the number of Chebyshev points q_ϵ. This indicates that our algorithm is numerically

n, q_ϵ	ϵ^m	$T_d(sec)$	$T_m(sec)$	T_d/T_m
256,7	5.10e-03	3.78e+03	6.07e+02	6.23e+00
512,7	7.29e-03	3.71e+04	3.50e+03	1.06e+01
1024,7	6.16e-03	6.42e+05	1.70e+04	3.77e+01
256,9	4.49e-04	2.34e+03	7.88e+02	2.97e+00
512,9	4.04e-04	3.66e+04	4.64e+03	7.90e+00
1024,9	3.88e-04	6.21e+05	2.17e+04	2.86e+01
256,11	1.86e-05	2.48e+03	1.33e+03	1.86e+00
512,11	1.80e-05	3.60e+04	6.94e+03	5.18e+00
1024,11	2.39e-05	5.96e+05	2.83e+04	2.11e+01

Table 3.2: Numerical results given by the multiscale butterfly algorithm for the FIO in (3.34). T_d is the running time of the direct evaluation; T_m is the running time of the multiscale butterfly algorithm; and T_d/T_m is the speedup factor.

stable. Another observation is that the relative error improves on average by a factor of 12 every time q_ϵ is incremented by a factor of 2. As we can see in those tables, for a fixed kernel and a fixed q_ϵ, the accuracy is almost independent of n. In fact, as we keep on incrementing q_ϵ, the relative error decreases until it reaches the machine precision, independent of n. Hence, in practical applications, one can increase the value of q_ϵ until a desired accuracy is reached. In the comparison in Table 3.1, the multiscale butterfly algorithm and the polar butterfly algorithm use $q_\epsilon = \{5, 7, 9, 11\}$ and achieve comparable accuracy. Meanwhile, as we observed from Table 3.1, the relative error decreasing rate of the multiscale butterfly algorithm is larger than the decreasing rate of the polar butterfly algorithm. This means if a high accuracy is desired, the multiscale butterfly algorithm requires a smaller q_ϵ to achieve it comparing to the polar butterfly algorithm.

The second concern about the algorithm is the asymptotic complexity. From the T_m column of Table 3.1 and 3.2, we see that T_m almost quadrupled when the problem size doubled under the same q_ϵ. According to this, we are convinced that the empirical running time of the multiscale butterfly algorithm follows the $O(n^2 \log n)$ asymptotic

complexity. Note that the speedup factor over the polar butterfly algorithm is about 6 and the multiscale butterfly algorithm obtains better accuracy. This makes the multiscale butterfly algorithm quite attractive to practitioners who are interested in evaluating an FIO with a large n.

Example 3. Extending the multiscale butterfly algorithm to higher dimensions is straightforward. There are two main modifications: higher dimensional multiscale domain decomposition and Chebyshev interpolation. In three dimensions, the frequency domain is decomposed into cubic shells instead of coronas. The kernel interpolation is applied on a three dimensional Chebyshev grids. We apply our three-dimensional multiscale butterfly algorithm to a simple example integrating over spheres with different radii. We assume a constant amplitude function and the kernel function is given by

$$\begin{aligned} &\Phi(x,\xi) = x\cdot\xi + c(x)\sqrt{\xi_1^2+\xi_2^2+\xi_3^2}, \\ &c(x) = (3+\sin(2\pi x_1)\sin(2\pi x_2)\sin(2\pi x_3))/4. \end{aligned} \tag{3.35}$$

Table 3.3 summarizes the results of this example given by the direct method and the multiscale butterfly algorithm.

n, q_ϵ	ϵ^m	$T_d(sec)$	$T_m(sec)$	T_d/T_m
64,5	9.41e-02	1.82e+04	2.50e+03	7.31e+00
128,5	7.57e-02	6.21e+05	2.42e+04	2.57e+01
256,5	8.23e-02	3.91e+07	2.35e+05	1.66e+02
64,7	1.20e-02	1.83e+04	7.32e+03	2.50e+00
128,7	1.03e-02	6.03e+05	4.48e+04	1.35e+01
256,7	8.13e-03	4.39e+07	3.81e+05	1.15e+02

Table 3.3: Numerical results given by the multiscale butterfly algorithm for the phase function in (3.35).

3.6 Conclusion and remarks on parallelization

A simple and efficient multiscale butterfly algorithm for evaluating FIOs is introduced in this chapter. This method hierarchically decomposes the frequency domain into multiscale coronas in order to avoid possible singularity of the phase function $\Phi(x, \xi)$ at $\xi = 0$. A single-scale butterfly algorithm is applied to evaluate the FIO over each corona. Many drawbacks of the original butterfly algorithm based on a polar-Cartesian transform in [17] can be avoided. The new multiscale butterfly algorithm has an $O(N \log N)$ operation complexity with a smaller pre-factor, while keeping the same $O(N)$ memory complexity.

The multiscale butterfly algorithm is also a scalable algorithm and can be efficiently parallelized. Assume that we are given P processes together with a problem of size N. The multiscale butterfly algorithm decomposes the frequency domain into $O(\log N)$ multiscale coronas, and the evaluation on each corona is achieved through a single-scale butterfly algorithm. Naturally, the single-scale butterfly algorithm on each corona can be efficiently parallelized on $O(P/\log N)$ processes via the parallel butterfly algorithm [76]. Combining the complexity analysis in Section 3.4.2 and [76], we conclude that the computational cost for the parallel butterfly algorithm on each corona is $O(\frac{N}{P} \log^2 N + \beta \frac{N}{P} \log N \log P + \alpha \log P)$, where α is the message latency and β is the inverse bandwidth. Direct evaluation on the center disk of Ω can be naïvely parallelized with $O(\frac{N}{P} \log N)$ operations and zero communication. Putting all pieces together gives rise to a parallel multiscale butterfly algorithm with computational cost $O(\frac{N}{P} \log^3 N + \beta \frac{N}{P} \log^2 N \log P + \alpha \log N \log P)$.

Chapter 4

Butterfly factorization

4.1 Introduction

As we mentioned earlier in Section 3.1, the butterfly algorithms [17, 97, 76] apply the kernel matrix to a vector in quasi-linear time without precomputation. However, these algorithms, including the one proposed in the previous chapter, rely on the analytic properties of the kernel. When such information is not available, we are forced to fall back to algorithms in [72, 75] with $O(N^2)$ precomputation cost.

A natural question is whether it is possible to reduce the cost of the precomputation phase if the analytic properties of the kernel are not directly accessible. The following two cases are quite common in applications:

(i) Only black-box routines for computing Kg and K^*g in $O(N \log N)$ operations are given;

(ii) Only a black-box routine for evaluating any entry of the matrix K in $O(1)$ operations is given.

To answer this question, we propose in this chapter the **butterfly factorization**, which, more precisely, represents K as a product of a sequence of sparse matrices:

$$K \approx U^L G^{L-1} \cdots G^h M^h (H^h)^* \cdots (H^{L-1})^* (V^L)^*, \tag{4.1}$$

where the depth $L = O(\log N)$ of T_X and T_Ω is assumed to be even, $h = L/2$ is a middle level index, and all factors are sparse matrices with $O(N)$ nonzero entries.

The construction of the butterfly factorization proceeds as follows in two stages. The first stage is to construction a preliminary middle level factorization that is associated with the middle level of T_X and T_Ω

$$K \approx U^h M^h (V^h)^*, \tag{4.2}$$

where U^h and V^h are block diagonal matrices and M^h is a weighted permutation matrix. In the first case, this is achieved by applying K to a set of $O(N^{1/2})$ structured random vectors and then applying the random singular value decomposition (SVD) to the result. This typically costs $O(N^{3/2} \log N)$ operations. In the second case, (4.2) is built via the random sampling method proposed in [36, 95] for computing approximate SVDs. This random sampling needs to make the assumption that the columns and rows of middle level blocks of K to be incoherent with respect to the delta functions and it typically takes only $O(N^{3/2})$ operations in practice.

Once the middle level factorization (4.2) is available, the second stage is a sequence of truncated SVDs that further factorize each of U^h and V^h into a sequence of sparse matrices, resulting in the final factorization (4.1). The operation count of this stage is $O(N^{3/2})$ and the total memory complexity for constructing butterfly factorization is $O(N^{3/2})$.

When the butterfly factorization (4.1) is constructed, the cost of applying K to a given vector $g \in \mathbb{C}^N$ is $O(N \log N)$ because (4.1) is a sequence of $O(\log N)$ sparse matrices, each with $O(N)$ non-zero entries.

This work is motivated by problems that require repeated applications of a butterfly algorithm. In several applications, such as inverse scattering [85, 96] and fast spherical harmonic transform (SHT) [86], the butterfly algorithm is called repeatedly either in an iterative process of minimizing some regularized objective function or to a large set of different input vectors. Therefore, it becomes important to reduce the constant prefactor of the butterfly algorithm to save actual runtime. For example in [17], Chebyshev interpolation is applied to recover low-rank structures of submatrices with a sufficiently large number of interpolation points. The recovered rank is far from the optimum. Hence, the prefactor of the corresponding butterfly algorithm in [17] is large. The butterfly factorization can further compress this butterfly algorithm to obtain nearly optimal low-rank approximations resulting in a much smaller prefactor, as will be shown in the numerical results. Therefore, it is more efficient to construct the butterfly factorization using this butterfly algorithm and then apply the butterfly factorization repeatedly. In this sense, the butterfly factorization can be viewed as a compression of certain butterfly algorithms.

Another important application is the computation of a composition of several FIOs. A direct method to construct the composition takes $O(N^3)$ operations, while the butterfly factorization provides a data-sparse representation of this composition in $O(N^{3/2} \log N)$ operations, once the fast algorithm for applying each FIO is available. After the construction, the application of the butterfly factorization is independent of the number of FIOs in the composition, which is significant when the number of FIOs is large.

Recently, there has also been a sequence of papers on recovering a structured matrix via applying it to (structured) random vectors. For example, the random SVD algorithms [46, 63, 89] recover a low-rank approximation to an unknown matrix when it is numerically low-rank. The work in [70] constructs a sparse representation for an unknown HSS matrix. More recently, [64] considers the more general problem of constructing a sparse representation of an unknown $\mathcal{H}$-matrix. To our best knowledge, the present work is the first to address such matrix recovery problem if the unknown matrix satisfies the complementary low-rank property.

4.1.1 Organization

The rest of this chapter is organized as follows. Section 4.2 briefly reviews some basic tools that shall be used repeatedly in later sections. Section 4.3 describes in detail the butterfly factorization in one dimension and its construction algorithm. Section 4.4 proposes several fast algorithms for multidimensional operators. The corresponding numerical examples are provided right after the description of each algorithm. Section 4.5 discusses the parallelism issue for the butterfly factorizations.

4.2 Preliminaries

For a matrix $Z \in \mathbb{C}^{m\times n}$, we define a rank-$r$ approximate singular value decomposition (SVD) of Z as

$$Z \approx U_0 \Sigma_0 V_0^*,$$

where $U_0 \in \mathbb{C}^{m\times r}$ is unitary, $\Sigma_0 \in \mathbb{R}^{r\times r}$ is diagonal, and $V_0 \in \mathbb{C}^{n\times r}$ is unitary. A straightforward method to obtain the optimal rank-r approximation of Z is to compute its truncated SVD, where U_0 is the matrix with the first r left singular vectors,

Σ_0 is a diagonal matrix with the first r singular values in decreasing order, and V_0 is the matrix with the first r right singular vectors.

A typical computation of the truncated SVD of Z takes $O(mn\min(m,n))$ operations, which can be quite expensive when m and n are large. Therefore, a lot of research has been devoted to faster algorithms for computing approximate SVDs, especially for matrices with fast decaying singular values. In Sections 4.2.1 and 4.2.2, we will introduce two random algorithms for computing approximate SVDs for numerically low-rank matrices Z: the first one [46] is based on applying the matrix to random vectors while the second one [36, 95] relies on sampling the matrix entries randomly.

Once an approximate SVD $Z \approx U_0\Sigma_0 V_0^*$ is computed, it can be written in several equivalent ways, each of which is convenient for certain purposes. First, one can write

$$Z \approx USV^*,$$

where

$$U = U_0\Sigma_0,\ S = \Sigma_0^{-1} \text{ and } V^* = \Sigma_0 V_0^*. \tag{4.3}$$

This construction is analogous to the well-known CUR decomposition [68] in the sense that the left and right factors in both factorization methods inherit similar singular values of the original numerical low-rank matrix. Here, the middle matrix S in (4.3) can be carefully constructed to ensure numerical stability, since the singular values in Σ_0 can be computed to nearly full relative precision.

As we shall see, sometimes it is also convenient to write the approximation as

$$Z \approx UV^*$$

where

$$U = U_0 \text{ and } V^* = \Sigma_0 V_0^*, \tag{4.4}$$

or

$$U = U_0\Sigma_0 \text{ and } V^* = V_0^*. \tag{4.5}$$

Here, one of the factors U and V share the singular values of Z.

4.2.1 SVD via random matrix-vector multiplication

One popular approach is the random algorithm in [46] that reduces the cubic complexity to $O(rmn)$ complexity. We briefly review this following [46] for constructing a rank-r approximation SVD $Z \approx U_0\Sigma_0 V_0^*$ below.

Algorithm 4.2.1. *Randomized SVD*

1. *Generate two tall skinny random Gaussian matrices $R_{col} \in \mathbb{C}^{n\times(r+p)}$ and $R_{row} \in \mathbb{C}^{m\times(r+p)}$, where $p = O(1)$ is an additive oversampling parameter that increases the approximation accuracy.*

2. *Apply the pivoted QR factorization to ZR_{col} and let Q_{col} be the matrix of the first r columns of the Q matrix. Similarly, apply the pivoted QR factorization to Z^*R_{row} and let Q_{row} be the matrix of the first r columns of the Q matrix.*

3. *Generate a tiny middle matrix $M = (R_{row}^*Q_{col})^\dagger R_{row}^* Z R_{col}(Q_{row}^* R_{col})^\dagger$ and compute its rank-r truncated SVD: $M \approx U_M\Sigma_M V_M^*$, where $(\cdot)^\dagger$ denotes the pseudo inverse.*

4. *Let $U_0 = Q_{col}U_M$, $\Sigma_0 = \Sigma_M$, and $V_0^* = V_M^* Q_{row}^*$. Then $Z \approx U_0\Sigma_0 V_0^*$.*

The dominant complexity comes from the application of Z to $O(r)$ random vectors. If fast algorithms for applying Z are available, the quadratic complexity can be further reduced.

Once the approximate SVD of Z is ready, the equivalent forms in (4.3), (4.4), and (4.5) can be constructed easily. Under the condition that the singular values of Z decay sufficiently rapidly, the approximation error of the resulting rank-r is nearly optimal with an overwhelming probability. Typically, the additive over-sampling parameter $p = 5$ is sufficient to obtain an accurate rank-r approximation of Z.

For most applications, the goal is to construct a low-rank approximation up to a fixed relative precision ϵ, rather than a fixed rank r. The above procedure can then be embedded into an iterative process that starts with a relatively small r, computes a rank-r approximation, estimates the error probabilistically, and repeats the steps with doubled rank $2r$ if the error is above the threshold ϵ [46].

4.2.2 SVD via random sampling

The above algorithm relies only on the product of the matrix $Z \in \mathbb{C}^{m\times n}$ or its transpose with given random vectors. If one is allowed to access the individual entries of Z, the following random sampling method for low-rank approximations introduced in [36, 95] can be more efficient. This method only visits $O(r)$ columns and rows of Z and hence only requires $O(r^2(m+n))$ operations and $O(r(m+n))$ memory.

Here, we adopt the standard notation for a submatrix: given a row index set I and a column index set J, $Z_{I,J} = Z(I,J)$ is the submatrix with entries from rows in I and columns in J; we also use " : " to denote the entire columns or rows of the matrix, i.e., $Z_{I,:} = Z(I,:)$ and $Z_{:,J} = Z(:,J)$. With these handy notations, we briefly introduce the random sampling algorithm to construct a rank-r approximation

of $Z \approx U_0 \Sigma_0 V_0^*$.

Algorithm 4.2.2. *Randomized sampling for low-rank approximation*

1. *Let Π_{col} and Π_{row} denote the important columns and rows of Z that are used to form the column and row bases. Initially $\Pi_{col} = \varnothing$ and $\Pi_{row} = \varnothing$.*

2. *Randomly sample rq rows and denote their indices by S_{row}. Let $I = S_{row} \cup \Pi_{row}$. Here $q = O(1)$ is a multiplicative oversampling parameter. Perform a pivoted QR decomposition of $Z_{I,:}$ to get*

$$Z_{I,:}P = QR,$$

   where P is the resulting permutation matrix and $R = (r_{ij})$ is an $O(r) \times n$ upper triangular matrix. Define the important column index set Π_{col} to be the first r columns picked within the pivoted QR decomposition.

3. *Randomly sample rq columns and denote their indices by S_{col}. Let $J = S_{col} \cup \Pi_{col}$. Perform a pivoted LQ decomposition of $Z_{:,J}$ to get*

$$PZ_{:,J} = LQ,$$

   where P is the resulting permutation matrix and $L = (l_{ij})$ is an $m \times O(r)$ lower triangular matrix. Define the important row index set Π_{row} to be the first r rows picked within the pivoted LQ decomposition.

4. *Repeat steps 2 and 3 a few times to ensure Π_{col} and Π_{row} sufficiently sample the important columns and rows of Z.*

5. *Apply the pivoted QR factorization to* $Z_{:,\Pi_{col}}$ *and let* Q_{col} *be the matrix of the first* r *columns of the* Q *matrix. Similarly, apply the pivoted QR factorization to* $Z^*_{\Pi_{row},:}$ *and let* Q_{row} *be the matrix of the first* r *columns of the* Q *matrix.*

6. *We seek a middle matrix* M *such that* $Z \approx Q_{col} M Q^*_{row}$*. To solve this problem efficiently, we approximately reduce it to a least-squares problem of a smaller size. Let* S_{col} *and* S_{row} *be the index sets of a few extra randomly sampled columns and rows. Let* $J = \Pi_{col} \cup S_{col}$ *and* $I = \Pi_{row} \cup S_{row}$*. A simple least-squares solution to the problem*

$$\min_M \|Z_{I,J} - (Q_{col})_{I,:} M (Q^*_{row})_{:,J}\|$$

gives $M = (Q_{col})^\dagger_{I,:} Z_{I,J} (Q^*_{row})^\dagger_{:,J}$*, where* $(\cdot)^\dagger$ *stands for the pseudo-inverse.*

7. *Compute an SVD* $M \approx U_M \Sigma_M V^*_M$*. Then the low-rank approximation of* $Z \approx U_0 S_0 V_0^*$ *is given by*

$$U_0 = Q_{col} U_M, \quad \Sigma_0 = \Sigma_M, \quad V_0^* = V_M^* Q^*_{row}. \tag{4.6}$$

We have not been able to quantify the error and success probability rigorously for this procedure at this point. On the other hand, when the columns and rows of K are incoherent with respect to "delta functions" (i.e., vectors that have only one significantly larger entry), this procedure works well in our numerical experiments. Here, a vector u is said to be incoherent with respect to a vector v if $\mu = |u^T v|/(\|u\|_2 \|v\|_2)$ is small. In the typical implementation, the multiplicative oversampling parameter q is equal to 3 and Steps 2 and 3 are iterated no more than three times. These parameters are empirically sufficient to achieve accurate low-rank approximations and are used

through out numerical examples.

As we mentioned above, for most applications the goal is to construct a low-rank approximation up to a fixed relative error ϵ, rather than a fixed rank. This process can also be embedded into an iterative process to achieve the desired accuracy.

4.3 One-dimensional butterfly factorization

This section presents the butterfly factorization algorithm for a matrix $K \in \mathbb{C}^{N\times N}$ discretized from one-dimensional problems. For simplicity let $X = \Omega = \{1, \ldots, N\}$. The trees T_X and T_Ω are complete binary trees with $L = \log_2 N - O(1)$ levels. We assume that L is an even integer and the number of points in each leaf node of T_X and T_Ω is bounded by a uniform constant.

At each level ℓ, $\ell = 0, \ldots, L$, we denote the ith node at level ℓ in T_X as A_i^ℓ for $i = 0, 1, \ldots, 2^\ell - 1$ and the jth node at level $L-\ell$ in T_Ω as $B_j^{L-\ell}$ for $j = 0, 1, \ldots, 2^{L-\ell}-1$. These nodes naturally partition K into $O(N)$ submatrices $K_{A_i^\ell, B_j^{L-\ell}}$. For simplicity, we write $K_{i,j}^\ell := K_{A_i^\ell, B_j^{L-\ell}}$, where the superscript is used to indicate the level (in T_X). The butterfly factorization utilizes rank-r approximations of all submatrices $K_{i,j}^\ell$ with $r = O(1)$.

The butterfly factorization of K is built in two stages. In the first stage, we compute a rank-r approximations of each submatrix $K_{i,j}^h$ at the level $\ell = h = L/2$ and then organize them into an initial factorization:

$$K \approx U^h M^h (V^h)^*,$$

where U^h and V^h are block diagonal matrices and M^h is a weighted permutation matrix. This is referred as the **middle level factorization** and is described in detail

in Section 4.3.1.

In the second stage, we recursively factorize $U^\ell \approx U^{\ell+1}G^\ell$ and $(V^\ell)^* \approx (H^\ell)^*(V^{\ell+1})^*$ for $\ell = h, h+1, \ldots, L-1$, since U^ℓ and $(V^\ell)^*$ inherit the complementary low-rank property from K, i.e., the low-rank property of U^ℓ comes from the low-rank property of $K^\ell_{i,j}$ and the low-rank property of V^ℓ results from the one of $K^{L-\ell}_{i,j}$. After this recursive factorization, one reaches at the butterfly factorization of K

$$K \approx U^L G^{L-1} \cdots G^h M^h (H^h)^* \cdots (H^{L-1})^* (V^L)^*, \tag{4.7}$$

where all factors are sparse matrices with $O(N)$ nonzero entries. We refer to this stage as the **recursive factorization** and it is discussed in detail in Section 4.3.2.

4.3.1 Middle level factorization

The first step of the middle level factorization is to compute a rank-r approximation to every $K^h_{i,j}$. Recall that we consider one of the following two cases.

Assumption 4.3.1.

(i) Only black-box routines for computing Kg and K^*g in $O(N \log N)$ operations are given;

(ii) Only a black-box routine for evaluating any entry of the matrix K in $O(1)$ operations is given.

The actual computation of this step proceeds differently depending on which case is under consideration. Through the discussion, $m = 2^h = O(N^{1/2})$ is the number of nodes in the middle level $h = L/2$ and we assume without loss of generality that N/m is an integer.

- In the first case, the rank-r approximation of each $K^h_{i,j}$ is constructed with the SVD algorithm via random matrix-vector multiplication in Section 4.2.1. This requires us to apply $K^h_{i,j}$ and its adjoint to random Gaussian matrices of size $(N/m)\times(r+p)$, where r is the desired rank and p is an oversampling parameter. In order to take advantage of the fast algorithm for multiplying K, we construct a matrix C of size $N \times m(r+p)$. C is partitioned into an $m \times m$ blocks with each block C_{ij} for $i, j = 0, 1, \ldots, m-1$ of size $(N/m) \times (r+p)$. In additional, C is block-diagonal and its diagonal blocks are random Gaussian matrices. This is equivalent to applying each $K^h_{i,j}$ to the same random Gaussian matrix C_{jj} for all i. We then use the fast algorithm to apply K to each column of C and store the results. Similarly, we form another random block diagonal matrix R similar to C and use the fast algorithm of applying K^* to R. This is equivalent to applying each $(K^h_{i,j})^*$ to an $(N/m) \times (r+p)$ Gaussian random matrix R_{ii} for all $j = 0, 1, \ldots, m-1$. With $K^h_{i,j}C_{jj}$ and $(K^h_{i,j})^*R_{ii}$ ready, we can compute the rank-r approximate SVD of $K^h_{i,j}$ following the procedure described in Section 4.2.1.

- In the second case, it is assumed that an arbitrary entry of K can be calculated in $O(1)$ operations. We simply apply the SVD algorithm via random sampling in Section 4.2.2 to each $K^h_{i,j}$ to construct a rank-r approximate SVD.

In either case, once the approximate SVD of $K^h_{i,j}$ is ready, it is transformed in the form

$$K^h_{i,j} \approx U^h_{i,j} S^h_{i,j} (V^h_{j,i})^*$$

following (4.3). We would like to emphasize that the columns of $U^h_{i,j}$ and $V^h_{j,i}$ are scaled with the singular values of the approximate SVD so that they keep track of

the importance of these columns in approximating $K^h_{i,j}$.

After calculating the approximate rank-r factorization of each $K^h_{i,j}$, we assemble these factors into three block matrices U^h, M^h and V^h as follows:

$$
\begin{aligned}
K \approx & \begin{pmatrix}
U^h_{0,0}S^h_{0,0}(V^h_{0,0})^* & U^h_{0,1}S^h_{0,1}(V^h_{1,0})^* & \cdots & U^h_{0,m-1}S^h_{0,m-1}(V^h_{m-1,0})^* \\
U^h_{1,0}S^h_{1,0}(V^h_{0,1})^* & U^h_{1,1}S^h_{1,1}(V^h_{1,1})^* & & U^h_{1,m-1}S^h_{1,m-1}(V^h_{m-1,1})^* \\
\vdots & & \ddots & \\
U^h_{m-1,0}S^h_{m-1,0}(V^h_{0,m-1})^* & U^h_{m-1,1}S^h_{m-1,1}(V^h_{1,m-1})^* & & U^h_{m-1,m-1}S^h_{m-1,m-1}(V^h_{m-1,m-1})^*
\end{pmatrix} \\
= & \begin{pmatrix}
U^h_0 & & & \\
& U^h_1 & & \\
& & \ddots & \\
& & & U^h_{m-1}
\end{pmatrix}
\begin{pmatrix}
M^h_{0,0} & M^h_{0,1} & \cdots & M^h_{0,m-1} \\
M^h_{1,0} & M^h_{1,1} & & M^h_{1,m-1} \\
\vdots & & \ddots & \\
M^h_{m-1,0} & M^h_{m-1,1} & & M^h_{m-1,m-1}
\end{pmatrix}
\begin{pmatrix}
(V^h_0)^* & & & \\
& (V^h_1)^* & & \\
& & \ddots & \\
& & & (V^h_{m-1})^*
\end{pmatrix} \\
= & U^h M^h (V^h)^*,
\end{aligned}
\tag{4.8}
$$

where

$$
\begin{aligned}
U^h_i &= \begin{pmatrix} U^h_{i,0} & U^h_{i,1} & \cdots & U^h_{i,m-1} \end{pmatrix} \in \mathbb{C}^{(N/m)\times mr}, \\
V^h_j &= \begin{pmatrix} V^h_{j,0} & V^h_{j,1} & \cdots & V^h_{j,m-1} \end{pmatrix} \in \mathbb{C}^{(N/m)\times mr},
\end{aligned}
\tag{4.9}
$$

and $M^h \in \mathbb{C}^{(m^2r)\times(m^2r)}$ is a weighted permutation matrix. Each submatrix $M^h_{i,j}$ is itself an $m \times m$ block matrix with block size $r \times r$ where all blocks are zero except that the (j,i) block is equal to the diagonal matrix $S^h_{i,j}$. It is obvious that there are only $O(N)$ nonzero entries in M^h. See Figure 4.1 for an example of a middle level factorization of a 64×64 matrix with $r = 1$.

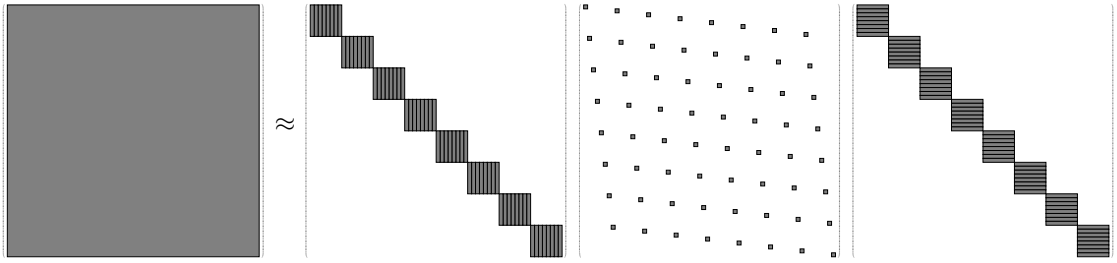

Figure 4.1: The middle level factorization of a 64×64 complementary low-rank matrix $K \approx U^3 M^3 (V^3)^*$ assuming $r = 1$. Grey blocks indicate nonzero blocks. U^3 and V^3 are block-diagonal matrices with 8 blocks. The diagonal blocks of U^3 and V^3 are assembled according to Equation (4.9) as indicated by black rectangles. M^3 is a 8×8 block matrix with each block $M^3_{i,j}$ itself an 8×8 block matrix containing diagonal weights matrix on the (j, i) block.

4.3.2 Recursive factorization

We will recursively factorize

$$U^\ell \approx U^{\ell+1} G^\ell \tag{4.10}$$

for $\ell = h, h+1, \ldots, L-1$ and

$$(V^\ell)^* \approx (H^\ell)^* (V^{\ell+1})^* \tag{4.11}$$

for $\ell = h, h+1, \ldots, L-1$. After these recursive factorizations, we can obtain the following butterfly factorization by substituting these factorizations into (4.8):

$$K \approx U^L G^{L-1} \cdots G^h M^h (H^h)^* \cdots (H^{L-1})^* (V^L)^*. \tag{4.12}$$

4.3.2.1 Recursive factorization of U^h

Each factorization at level ℓ in (4.10) results from the low-rank property of $K_{i,j}^{\ell}$ for $\ell \geqslant L/2$. When $\ell = h$, recall that

$$U^h = \begin{pmatrix} U_0^h & & & \\ & U_1^h & & \\ & & \ddots & \\ & & & U_{m-1}^h \end{pmatrix}$$

and

$$U_i^h = \begin{pmatrix} U_{i,0}^h & U_{i,1}^h & \cdots & U_{i,m-1}^h \end{pmatrix}$$

with each $U_{i,j}^h \in \mathbb{C}^{(N/m)\times r}$. We split U_i^h and each $U_{i,j}^h$ into halves by row, i.e.,

$$U_i^h = \begin{pmatrix} U_i^{h,t} \\ \hline U_i^{h,b} \end{pmatrix} \text{ and } U_{i,j}^h = \begin{pmatrix} U_{i,j}^{h,t} \\ \hline U_{i,j}^{h,b} \end{pmatrix},$$

where the superscript t denotes the top half and b denotes the bottom half of a matrix. Then we have

$$U_i^h = \left(\begin{array}{cccc} U_{i,0}^{h,t} & U_{i,1}^{h,t} & \cdots & U_{i,m-1}^{h,t} \\ \hline U_{i,0}^{h,b} & U_{i,1}^{h,b} & \cdots & U_{i,m-1}^{h,b} \end{array}\right). \tag{4.13}$$

Notice that, for each $i = 0, 1, \ldots, m-1$ and $j = 0, 1, \ldots, m/2-1$, the columns of

$$\begin{pmatrix} U_{i,2j}^{h,t} & U_{i,2j+1}^{h,t} \end{pmatrix} \text{ and } \begin{pmatrix} U_{i,2j}^{h,b} & U_{i,2j+1}^{h,b} \end{pmatrix} \tag{4.14}$$

in (4.13) are in the column space of $K^{h+1}_{2i,j}$ and $K^{h+1}_{2i+1,j}$, respectively. By the complementary low-rank property of the matrix K, $K^{h+1}_{2i,j}$ and $K^{h+1}_{2i+1,j}$ are numerical low-rank. Hence $\begin{pmatrix} U^{h,t}_{i,2j} & U^{h,t}_{i,2j+1} \end{pmatrix}$ and $\begin{pmatrix} U^{h,b}_{i,2j} & U^{h,b}_{i,2j+1} \end{pmatrix}$ are numerically low-rank matrices in $\mathbb{C}^{(N/2m)\times 2r}$. Compute their rank-$r$ approximations by the standard truncated SVD, transform it into the form of (4.5) and denote them as

$$\begin{pmatrix} U^{h,t}_{i,2j} & U^{h,t}_{i,2j+1} \end{pmatrix} \approx U^{h+1}_{2i,j} G^h_{2i,j} \text{ and } \begin{pmatrix} U^{h,b}_{i,2j} & U^{h,b}_{i,2j+1} \end{pmatrix} \approx U^{h+1}_{2i+1,j} G^h_{2i+1,j} \tag{4.15}$$

for $i = 0, 1, \ldots, m-1$ and $j = 0, 1, \ldots, m/2-1$. The matrices in (4.15) can be assembled into two new sparse matrices, such that

$$U^h \approx U^{h+1} G^h = \begin{pmatrix} U^{h+1}_0 & & & \\ & U^{h+1}_1 & & \\ & & \ddots & \\ & & & U^{h+1}_{2m-1} \end{pmatrix} \begin{pmatrix} G^h_0 & & & \\ & G^h_1 & & \\ & & \ddots & \\ & & & G^h_{m-1} \end{pmatrix},$$

where

$$U^{h+1}_i = \begin{pmatrix} U^{h+1}_{i,0} & U^{h+1}_{i,1} & \cdots & U^{h+1}_{i,m/2-1} \end{pmatrix}$$

for $i = 0, 1, \ldots, 2m-1$, and

$$G_i^h = \left(\begin{array}{cccc} G_{2i,0}^h & & & \\ & G_{2i,1}^h & & \\ & & \ddots & \\ & & & G_{2i,m/2-1}^h \\ \hline G_{2i+1,0}^h & & & \\ & G_{2i+1,1}^h & & \\ & & \ddots & \\ & & & G_{2i+1,m/2-1}^h \end{array}\right)$$

for $i = 0, 1, \ldots, m-1$.

Since there are $O(1)$ nonzero entries in each $G_{i,j}^h$ and there are $O(N)$ such submatrices, there are only $O(N)$ nonzero entries in G^h. See Figure 4.2 top for an example of the factorization $U^h \approx U^{h+1}G^h$ for the left factor U^h with $L = 6$, $h = 3$ and $r = 1$ in Figure 4.1.

Similarly, for any ℓ between h and $L-1$, we can factorize $U^\ell \approx U^{\ell+1}G^\ell$, because the columns in $\begin{pmatrix} U_{i,2j}^{\ell,t} & U_{i,2j+1}^{\ell,t} \end{pmatrix}$ and $\begin{pmatrix} U_{i,2j}^{\ell,b} & U_{i,2j+1}^{\ell,b} \end{pmatrix}$ are in the column space of the numerically low-rank matrices $K_{2i,j}^{\ell+1}$ and $K_{2i+1,j}^{\ell+1}$, respectively. Computing the rank-r approximations via the standard truncated SVD and transforming them into the form of (4.5) give

$$\begin{pmatrix} U_{i,2j}^{\ell,t} & U_{i,2j+1}^{\ell,t} \end{pmatrix} \approx U_{2i,j}^{\ell+1} G_{2i,j}^\ell \text{ and } \begin{pmatrix} U_{i,2j}^{\ell,b} & U_{i,2j+1}^{\ell,b} \end{pmatrix} \approx U_{2i+1,j}^{\ell+1} G_{2i+1,j}^\ell \tag{4.16}$$

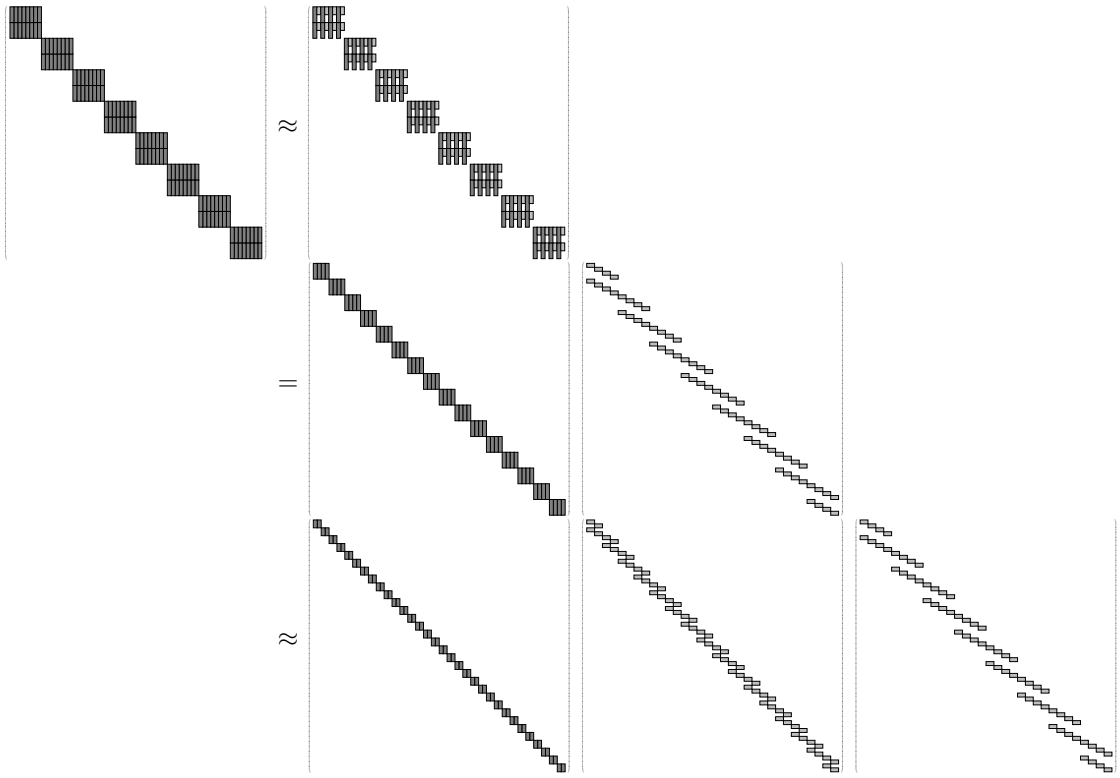

Figure 4.2: The recursive factorization of U^3 in Figure 4.1. Gray factors are matrices inheriting the complementary low-rank property. Top: left matrix: U^3 with each diagonal block partitioned into smaller blocks according to Equation (4.13) as indicated by black rectangles; middle-left matrix: low-rank approximations of submatrices in U^3 given by Equation (4.15); middle right matrix: U^4; right matrix: G^3. Bottom: U^4 in the first row is further factorized into $U^4 \approx U^5 G^4$, giving $U^3 \approx U^5 G^4 G^3$.

for $i = 0, 1, \ldots, 2^\ell - 1$ and $j = 0, 1, \ldots, 2^{L-\ell-1} - 1$. After assembling these factorizations together, we obtain

$$U^\ell \approx U^{\ell+1} G^\ell = \begin{pmatrix} U_0^{\ell+1} & & & \\ & U_1^{\ell+1} & & \\ & & \ddots & \\ & & & U_{2^{\ell+1}-1}^{\ell+1} \end{pmatrix} \begin{pmatrix} G_0^\ell & & & \\ & G_1^\ell & & \\ & & \ddots & \\ & & & G_{2^\ell-1}^\ell \end{pmatrix},$$

where

$$U_i^{\ell+1} = \begin{pmatrix} U_{i,0}^{\ell+1} & U_{i,1}^{\ell+1} & \cdots & U_{i,2^{L-\ell-1}-1}^{\ell+1} \end{pmatrix}$$

for $i = 0, 1, \ldots, 2^{\ell+1} - 1$, and

$$G_i^\ell = \left(\begin{array}{cccc} G_{2i,0}^\ell & & & \\ & G_{2i,1}^\ell & & \\ & & \ddots & \\ & & & G_{2i,2^{L-\ell-1}-1}^\ell \\ \hline G_{2i+1,0}^\ell & & & \\ & G_{2i+1,1}^\ell & & \\ & & \ddots & \\ & & & G_{2i+1,2^{L-\ell-1}-1}^\ell \end{array} \right)$$

for $i = 0, 1, \ldots, 2^\ell - 1$.

After $L - h$ steps of recursive factorizations

$$U^\ell \approx U^{\ell+1} G^\ell$$

for $\ell = h, h+1, \ldots, L-1$, we obtain the recursive factorization of U^h as

$$U^h \approx U^L G^{L-1} \cdots G^h. \tag{4.17}$$

See Figure 4.2 bottom for an example of a recursive factorization for the left factor U^h with $L = 6$, $h = 3$ and $r = 1$ in Figure 4.1.

Similar to the analysis of G^h, it is also easy to check that there are only $O(N)$ nonzero entries in each G^ℓ in (4.46). Since there are $O(N)$ diagonal blocks in U^L and each block contains $O(1)$ entries, there is $O(N)$ nonzero entries in U^L.

4.3.2.2 Recursive factorization of V^h

The recursive factorization of V^h is similar to the one of U^h. In each step of the factorization

$$(V^\ell)^* \approx (H^\ell)^* (V^{\ell+1})^*,$$

we take advantage of the low-rank property of the row space of $K_{i,2j}^{L-\ell-1}$ and $K_{i,2j+1}^{L-\ell-1}$ to obtain rank-r approximations. Applying the exact same procedure of Section 4.3.2.1 now to V^ℓ leads to the recursive factorization $V^h \approx V^L H^{L-1} \cdots H^h$, or equivalently

$$(V^h)^* \approx (H^h)^* \cdots (H^{L-1})^* (V^L)^*, \tag{4.18}$$

with all factors containing only $O(N)$ nonzero entries. See Figure 4.3 for an example of a recursive factorization $(V^h)^* \approx (H^h)^* \cdots (H^{L-2})^* (V^{L-1})^*$ for the left factor V^h with $L = 6$, $h = 3$ and $r = 1$ in Figure 4.1.

Given the recursive factorization of U^h and $(V^h)^*$ in (4.46) and (4.18), we reach

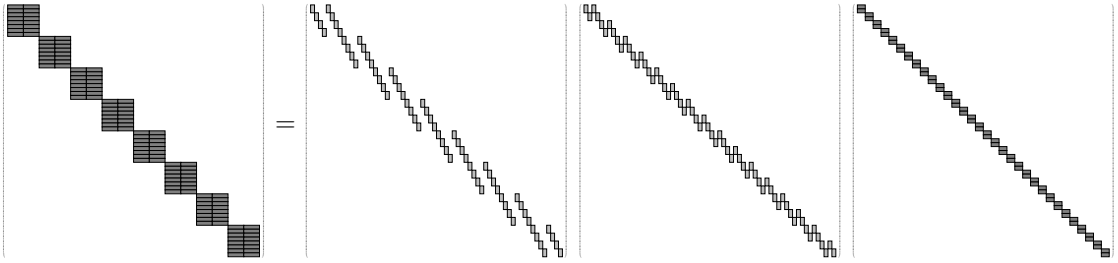

Figure 4.3: The recursive factorization $(V^3)^* \approx (H^3)^*(H^4)^*(V^5)^*$ of $(V^3)^*$ in Figure 4.1

the butterfly factorization

$$K \approx U^L G^{L-1} \cdots G^h M^h (H^h)^* \cdots (H^{L-1})^* (V^L)^*, \tag{4.19}$$

where all factors are sparse matrices with $O(N)$ nonzero entries. For a given input vector $g \in \mathbb{C}^N$, the $O(N^2)$ matrix-vector multiplication $u = Kg$ can be approximated by a sequence of $O(\log N)$ sparse matrix-vector multiplications given by the butterfly factorization.

4.3.3 Complexity analysis

Following the construction algorithm of a butterfly factorization, the complexity analysis naturally consists of two parts: the middle level factorization and the recursive factorization.

The complexity of the middle level factorization depends on which case of Assumption 4.3.1 is adopted.

- For the first case, the approximate SVDs are determined by the application of K and K^* to Gaussian random matrices in $\mathbb{C}^{N \times N^{1/2}(r+p)}$ and the rank-r approximations of K_{ij}^h for each (i, j) pair. Assume that each matrix-vector

multiplication by K or K^* via the given black-box routines requires $O(C_K(N))$ operations (which is at least $O(N)$). Then the dominant cost is due to applying K and K^* $O(N^{1/2})$ times, which yields an overall computational complexity of $O(C_K(N)N^{1/2})$.

- In the second case, the approximate SVDs are computed via random sampling for each K_{ij}^h of the $O(N)$ pairs (i, j). The complexity of performing random sampling for each such block is $O(N^{1/2})$. Hence, the overall computational complexity is $O(N^{3/2})$.

In the recursive factorization, U^ℓ at level ℓ consists of $O(2^\ell)$ diagonal blocks of size $O(N/2^\ell) \times O(N/2^\ell)$. In each diagonal block, there are $O(N/2^\ell)$ factorizations in (4.45). Since the operation complexity of performing one factorization in (4.45) is $O(N/2^\ell)$, it takes $O(N^2/2^\ell)$ operations to factorize U^ℓ. Summing up the operations at all levels gives the total complexity for recursively factorizing U^h:

$$\sum_{\ell=h}^{L-1} O(N^2/2^\ell) = O(N^{3/2}). \tag{4.20}$$

Similarly, the operation complexity for recursively compressing V^h is also $O(N^{3/2})$.

The memory peak of the butterfly factorization occurs in the middle level factorization since we have to store the initial factorization in (4.8). There are $O(N^{3/2})$ nonzero entries in U^h and V^h, and $O(N)$ in M^h. Hence, the total memory complexity is $O(N^{3/2})$. The total operation complexity for constructing the butterfly factorization is summarized in Table 4.1.

It is worth pointing out that the memory complexity can be reduced to $O(N \log N)$, when we apply the random sampling method to construct each block in the initial factorization in (4.8) separately. Instead of factorizing U^h and V^h at the end of the

		Randomized SVD	Randomized sampling
Factorization Complexity	Middle level factorization	$O(C_K(N)N^{1/2})$	$O(N^{3/2})$
	Recursive factorization	$O(N^{3/2})$	
	Total	$O(C_K(N)N^{1/2})$	$O(N^{3/2})$
Memory Complexity		$O(N^{3/2})$	$O(N \log N)$
Application Complexity		$O(N \log N)$	

Table 4.1: Computational complexity and memory complexity of the butterfly factorization. $C_K(N)$ is the operation complexity of one application of K or K^*. In most of the cases encountered, $C_K(N) = O(N \log N)$.

middle level factorization, we can factorize the left and right factors U_i^h and V_i^h in (4.8) on the fly to avoid storing all factors in (4.8). For a fixed i, we generate U_i^h from K_{ij}^h for all j, and recursively factorize U_i^h. The memory cost is $O(N)$ for storing U_i^h and $O(N^{1/2} \log N)$ for storing the sparse matrices after its recursive factorization. Repeating this process for $i = 1, \ldots, N^{1/2}$ gives the complete factorization of U^h. The factorization of V^h is conducted similarly. The total memory complexity is $O(N \log N)$.

The operation and memory complexity for the application of the butterfly factorization are governed by the number of nonzero entries in the factorization: $O(N \log N)$.

4.3.4 Numerical results

This section presents three numerical examples to demonstrate the effectiveness of the algorithms proposed above. The first example is an FIO in [17] and the second example is a special function transform in [75]. Both examples provide an explicit

kernel function that becomes a one-dimensional complementary low-rank matrix after discretization. This allows us to apply the butterfly factorization construction algorithm with random sampling. The computational complexity and the memory cost are $O(N^{3/2})$ and $O(N \log N)$ in this case.

The third example is a composition of two FIOs for which an explicit kernel function of their composition is not available. Since we can apply either the butterfly algorithm in [17] or the butterfly factorization to evaluate these FIOs one by one, a fast algorithm for computing the composition is available. We apply the butterfly factorization construction algorithm with random matrix-vector multiplication to this example which requires $O(N^{3/2} \log N)$ operations and $O(N^{3/2})$ memory cost.

Our implementation is in MATLAB. The numerical results were obtained on a server computer with a 2.0 GHz CPU. The additive oversampling parameter is $p = 5$ and the multiplicative oversampling parameter is $q = 3$.

Let $\{u^d(x), x \in X\}$ and $\{u^a(x), x \in X\}$ denote the results given by the direct matrix-vector multiplication and the butterfly factorization. The accuracy of applying the butterfly factorization algorithm is estimated by the following relative error

$$\epsilon^a = \sqrt{\frac{\sum_{x \in S} |u^a(x) - u^d(x)|^2}{\sum_{x \in S} |u^d(x)|^2}}, \tag{4.21}$$

where S is a point set of size 256 randomly sampled from X.

Example 1. This example is to evaluate a one-dimensional FIO of the following form:

$$u(x) = \int_{\mathbb{R}} e^{2\pi\imath\Phi(x,\xi)} \widehat{f}(\xi) d\xi, \tag{4.22}$$

where $\widehat{f}$ is the Fourier transform of f, and $\Phi(x,\xi)$ is a phase function given by

$$\Phi(x,\xi) = x\cdot\xi + c(x)|\xi|, \quad c(x) = (2+\sin(2\pi x))/8. \tag{4.23}$$

The discretization of (4.22) is

$$u(x_i) = \sum_{\xi_j} e^{2\pi\imath\Phi(x_i,\xi_j)}\widehat{f}(\xi_j), \quad i,j = 1,2,\ldots,N, \tag{4.24}$$

where $\{x_i\}$ and $\{\xi_j\}$ are uniformly distributed points in $[0,1)$ and $[-N/2,N/2)$ following

$$x_i = (i-1)/N \text{ and } \xi_j = j-1-N/2. \tag{4.25}$$

(4.24) can be represented in a matrix form as $u = Kg$, where $u_i = u(x_i)$, $K_{ij} = e^{2\pi\imath\Phi(x_i,\xi_j)}$ and $g_j = \widehat{f}(\xi_j)$. The matrix K satisfies the complementary low-rank property as proved in [17, 60]. The explicit kernel function of K allows us to use the construction algorithm with random sampling. Table 4.2 summarizes the results of this example for different grid sizes N and truncation ranks r.

Example 2. Next, we provide an example of a special function transform. This example can be further applied to accelerate the Fourier-Bessel transform that is important in many real applications. Following the standard notation, we denote the Hankel function of the first kind of order m by $H_m^{(1)}$. When m is an integer, $H_m^{(1)}$ has a singularity at the origin and a branch cut along the negative real axis. We are interested in evaluating the sum of Hankel functions over different orders,

$$u(x_i) = \sum_{j=1}^{N} H_{j-1}^{(1)}(x_i)g_j, \quad i = 1,2,\ldots,N, \tag{4.26}$$

N, r	ϵ^a	$T_{Factor}(min)$	$T_d(sec)$	$T_a(sec)$	T_d/T_a
1024,4	2.49e-05	2.92e-01	2.30e-01	3.01e-02	7.65e+00
4096,4	4.69e-05	1.62e+00	2.64e+00	4.16e-02	6.35e+01
16384,4	5.77e-05	1.22e+01	2.28e+01	1.84e-01	1.24e+02
65536,4	6.46e-05	8.10e+01	2.16e+02	1.02e+00	2.12e+02
262144,4	7.13e-05	4.24e+02	3.34e+03	4.75e+00	7.04e+02
1024,6	1.57e-08	1.81e-01	1.84e-01	1.20e-02	1.54e+01
4096,6	3.64e-08	1.55e+00	2.56e+00	6.42e-02	3.98e+01
16384,6	6.40e-08	1.25e+01	2.43e+01	3.01e-01	8.08e+01
65536,6	6.53e-08	9.04e+01	2.04e+02	1.77e+00	1.15e+02
262144,6	6.85e-08	5.45e+02	3.68e+03	8.62e+00	4.27e+02
1024,8	5.48e-12	1.83e-01	1.78e-01	1.63e-02	1.09e+01
4096,8	1.05e-11	1.98e+00	2.71e+00	8.72e-02	3.11e+01
16384,8	2.09e-11	1.41e+01	3.34e+01	5.28e-01	6.33e+01
65536,8	2.62e-11	1.17e+02	2.10e+02	2.71e+00	7.75e+01
262144,8	4.13e-11	6.50e+02	3.67e+03	1.52e+01	2.42e+02

Table 4.2: Numerical results for the FIO given in (4.24). N is the size of the matrix; r is the fixed rank in the low-rank approximations; T_{Factor} is the factorization time of the butterfly factorization; T_d is the running time of the direct evaluation; T_a is the application time of the butterfly factorization; T_d/T_a is the speedup factor.

which is analogous to expansion in orthogonal polynomials. The points x_i are defined via the formula,

$$x_i = N + \frac{2\pi}{3}(i-1) \tag{4.27}$$

which are bounded away from zero. It is demonstrated in [75] that (4.26) can be represented via $u = Kg$ where K satisfies the complementary low-rank property, $u_i = u(x_i)$ and $K_{ij} = H^{(1)}_{j-1}(x_i)$. The entries of matrix K can be calculated efficiently and the construction algorithm with random sampling is applied to accelerate the evaluation of the sum (4.26). Table 4.3 summarizes the results of this example for different grid sizes N and truncation ranks r.

N, r	ϵ^a	$T_{Factor}(min)$	$T_d(sec)$	$T_a(sec)$	T_d/T_a
1024,4	2.35e-06	8.78e-01	8.30e-01	1.06e-02	7.86e+01
4096,4	5.66e-06	5.02e+00	5.30e+00	2.83e-02	1.87e+02
16384,4	6.86e-06	3.04e+01	5.51e+01	1.16e-01	4.76e+02
65536,4	7.04e-06	2.01e+02	7.59e+02	6.38e-01	1.19e+03
1024,6	2.02e-08	4.31e-01	7.99e-01	9.69e-03	8.25e+01
4096,6	4.47e-08	6.61e+00	5.41e+00	4.52e-02	1.20e+02
16384,6	5.95e-08	4.19e+01	5.62e+01	1.61e-01	3.48e+02
65536,6	7.86e-08	2.76e+02	7.60e+02	1.01e+00	7.49e+02

Table 4.3: Numerical results with the matrix given by (4.26).

From Table 4.2 and 4.3, we note that the accuracy of the butterfly factorization is well controlled by the max rank r. For a fixed rank r, the accuracy is almost independent of N. In practical applications, one can set the desired ϵ ahead and increase the truncation rank r until the relative error reaches ϵ.

The tables for Example 1 and Example 2 also provide numerical evidence for the asymptotic complexity of the proposed algorithms. The construction algorithm based on random sampling is of computational complexity $O(N^{3/2})$. When we quadruple

the problem size, the running time of the construction sextuples and is better than we expect. The reason is that in the random sampling method, the computation of a middle matrix requires pseudo-inverses of $r \times r$ matrices whose complexity is $O(r^3)$ with a large prefactor. Hence, when N is not large, the running time will be dominated by the $O(r^3 N)$ computation of middle matrices. The numbers also show that the application complexity of the butterfly factorization is $O(N \log N)$ with a prefactor much smaller than the butterfly algorithm with Chebyshev interpolation [17]. In example 1, when the relative error is $\epsilon \approx 10^{-5}$, the butterfly factorization truncates the low-rank submatrices with rank 4 whereas the butterfly algorithm with Chebyshev interpolation uses 9 Chebyshev grid points. The speedup factors are 200 on average.

Example 3. In this last example, we consider a composition of two FIOs, which is the discretization of the following operator

$$u(x) = \int_{\mathbb{R}} e^{2\pi\imath\Phi_2(x,\eta)} \int_{\mathbb{R}} e^{-2\pi\imath y\eta} \int_{\mathbb{R}} e^{2\pi\imath\Phi_1(y,\xi)} \widehat{f}(\xi) d\xi dy d\eta. \tag{4.28}$$

For simplicity, we consider the same phase function $\Phi_1 = \Phi_2 = \Phi$ as given by (4.23). By the discussion of Example 1 for one FIO, we know the discrete analog of the composition (4.28) can be represented as

$$u = KFKFf =: KFKg, \quad \text{with } g = Ff,$$

where F is the standard Fourier transform in matrix form, K is the same matrix as in Example 1, $u_i = u(x_i)$, and $g_j = \widehat{f}(\xi_j)$. Under mild assumptions as discussed in [52], the composition of two FIOs is an FIO. Hence, the new kernel matrix $\tilde{K} = KFK$

again satisfies the complementary low-rank property, though typically with slightly increased ranks.

Notice that it is not reasonable to compute the matrix $\tilde{K}$ directly. However, we have the fast Fourier transform (FFT) to apply F and the butterfly factorization that we have built for K in Example 1 to apply K. Therefore, the construction algorithm with random matrix-vector multiplication is applied to factorize $\tilde{K}$.

Since the direct evaluation of each u_i takes $O(N^2)$ operations, the exact solution $\{u_i^d\}_{i\in S}$ for a selected set S is infeasible for large N. We apply the butterfly factorization of K and the FFT to evaluate $\{u_i\}_{i\in S}$ as an approximation to the exact solution $\{u_i^d\}_{i\in S}$. These approximations are compared to the results $\{u_i^a\}_{i\in S}$ that are given by applying the butterfly factorization of $\tilde{K}$. Table 4.4 summarizes the results of this example for different grid sizes N and truncation ranks r.

N, r	ϵ^a	$T_{Factor}(min)$	$T_d(sec)$	$T_a(sec)$	T_d/T_a
1024,4	1.40e-02	3.26e-01	3.64e-01	4.74e-03	7.69e+01
4096,4	1.96e-02	4.20e+00	6.59e+00	2.52e-02	2.62e+02
16384,4	2.34e-02	4.65e+01	3.75e+01	1.15e-01	3.25e+02
65536,4	2.18e-02	4.33e+02	3.73e+02	6.79e-01	5.49e+02
1024,8	6.62e-05	3.65e-01	3.64e-01	8.25e-03	4.42e+01
4096,8	8.67e-05	4.94e+00	6.59e+00	5.99e-02	1.10e+02
16384,8	1.43e-04	6.23e+01	3.75e+01	3.47e-01	1.08e+02
65536,8	1.51e-04	6.91e+02	3.73e+02	1.76e+00	2.12e+02
1024,12	1.64e-08	4.79e-01	3.64e-01	1.48e-02	2.46e+01
4096,12	1.05e-07	6.35e+00	6.59e+00	1.12e-01	5.88e+01
16384,12	2.55e-07	7.58e+01	3.75e+01	7.64e-01	4.91e+01
65536,12	2.69e-07	7.63e+02	3.73e+02	4.39e+00	8.49e+01

Table 4.4: Numerical results for the composition of two FIOs.

Table 4.4 shows the numerical results of the butterfly factorization of $\tilde{K}$. The accuracy improves as we increase the truncation rank r. Comparing Table 4.4 with

Table 4.2, we notice that, for a fixed accuracy, the rank used in the butterfly factorization of the composition of FIOs should be larger than the rank used in a single FIO butterfly factorization. This is expected since the composition is in general more complicated than the individual FIOs. T_{Factor} grows on average by a factor of ten when we quadruple the problem size. This agrees with the estimated $O(N^{3/2} \log N)$ computational complexity for constructing the butterfly factorization. The column T_a shows that the empirical application time of our factorization is close to the estimated complexity $O(N \log N)$.

4.4 Multidimensional butterfly factorization

4.4.1 Two-dimensional butterfly factorization

This section presents the two-dimensional butterfly factorization for a kernel matrix $K = (K(x, \xi))_{x \in X, \xi \in \Omega}$ that satisfies the complementary low-rank property in $X \times \Omega$ with X and Ω given in (3.2) and (3.3). One particular example to keep in mind is the Fourier transform. The Fourier operators, as shown in [75], satisfy the complementary low-rank property throughout the domain in any dimension. Therefore, the two-dimensional butterfly factorization can be applied to Fourier transform. Once the factorization is constructed, the application of the Fourier transform is almost linear scaling and fully scalable in any dimension.

4.4.1.1 Notations and overall structure

We adopt the notation of the one-dimensional butterfly factorization introduced in Section 4.3 and adjust them to the two-dimensional case of this chapter.

Recall that n is the number of grid points on each dimension and $N = n^2$ is

the total number of points. Suppose that T_X and T_Ω are complete quadtrees with $L = \log n$ levels and, without loss of generality, L is an even integer. For a fixed level ℓ between 0 and L, the quadtree T_X has 4^ℓ nodes at level ℓ. By defining $\mathcal{I}^\ell = \{0, 1, \ldots, 4^\ell - 1\}$, we denote these nodes by A_i^ℓ with $i \in \mathcal{I}^\ell$. These 4^ℓ nodes at level ℓ are further ordered according to a Z-order curve (or Morton order) as illustrated in Figure 4.4. Based on this Z-ordering, the node A_i^ℓ at level ℓ has four child nodes denoted by $A_{4i+t}^{\ell+1}$ with $t = 0, \ldots, 3$. The nodes plotted in Figure 4.4 for $\ell = 1$ (middle) and $\ell = 2$ (right) illustrate the relationship between the parent node and its child nodes. Similarly, in the quadtree T_Ω, the nodes at the $L - \ell$ the are denoted as $B_j^{L-\ell}$ for $j \in \mathcal{I}^{L-\ell}$.

For any level ℓ between 0 and L, the kernel matrix K can be partitioned into $O(N)$ submatrices $K_{A_i^\ell, B_j^{L-\ell}} := (K(x,\xi))_{x \in A_i^\ell, \xi \in B_j^{L-\ell}}$ for $i \in \mathcal{I}^\ell$ and $j \in \mathcal{I}^{L-\ell}$. For simplicity, we shall denote $K_{A_i^\ell, B_j^{L-\ell}}$ as $K_{i,j}^\ell$, where the superscript ℓ denotes the level in the quadtree T_X. Because of the complementary low-rank property, every submatrix $K_{i,j}^\ell$ is numerically low-rank with the rank bounded by a uniform constant r independent of N.

The two-dimensional butterfly factorization consists of two stages. The first stage computes the factorizations

$$K_{i,j}^h \approx U_{i,j}^h S_{i,j}^h \left(V_{j,i}^h\right)^*$$

for all $i, j \in \mathcal{I}^h$ at the middle level $h = L/2$, following the form (4.3). These factorizations can then be assembled into three sparse matrices U^h, M^h, and V^h to give rise to a factorization for K:

$$K \approx U^h M^h \left(V^h\right)^*. \tag{4.29}$$

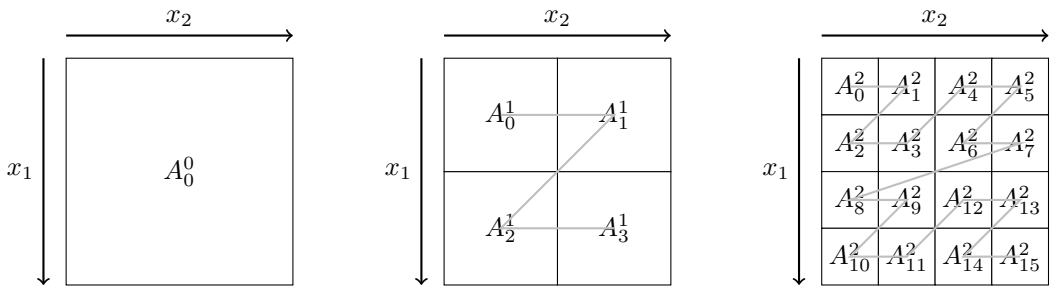

Figure 4.4: An illustration of Z-order curve cross levels. The superscripts indicate the different levels while the subscripts indicate the index in the Z-ordering. The light gray lines show the ordering among the subdomains on the same level. Left: The root at level 0. Middle: At level 1, the domain A_0^0 is divided into 2×2 subdomains A_i^1 with $i \in \mathcal{I}^1 = \{0, 1, 2, 3\}$. These 4 subdomains are ordered according to the Z-ordering. Right: At level 2, the domain A_0^0 is divided into 4×4 subdomains A_i^2 with $i \in \mathcal{I}^2 = \{0, 1, \ldots, 15\}$. These 16 subdomains are ordered similarly.

This stage is referred to as the *middle level factorization* and is described in Section 4.4.1.2. In the second stage, we recursively factorize the left and right factors U^h and V^h to obtain

$$U^h \approx U^L G^{L-1} \cdots G^h \quad \text{and} \quad \left(V^h\right)^* \approx \left(H^h\right)^* \cdots \left(H^{L-1}\right)^* \left(V^L\right)^*,$$

where the matrices on the right hand side in each formula are sparse matrices with $O(N)$ nonzero entries. Once they are ready, we assemble all factors together to produce a data-sparse approximate factorization for K:

$$K \approx U^L G^{L-1} \cdots G^h M^h \left(H^h\right)^* \cdots \left(H^{L-1}\right)^* \left(V^L\right)^*, \tag{4.30}$$

This stage is referred to as the *recursive factorization* and is discussed in Section 4.4.1.3.

4.4.1.2 Middle level factorization

We consider the construction of the multidimensional butterfly factorization for two cases in Assumption 4.3.1, where K is now the kernel matrix for two-dimensional problems.

In Case (i), we construct an approximate rank-r SVD of each $K^h_{i,j} \in \mathbb{R}^{n\times n}$ with $i, j \in \mathcal{I}^h$ using the SVD via random matrix-vector multiplication (Section 4.2.1). This requires applying each $K^h_{i,j}$ to a Gaussian random matrix $C_j \in \mathbb{C}^{n\times(r+k)}$ and its adjoint to a Gaussian random matrix $R_i \in \mathbb{C}^{(r+k)\times n}$. Here r is the desired numerical rank and k is the oversampling parameter. If a black box routine for applying the matrix K and its adjoint is available, this can be done in an efficient way as follows. For each $j \in \mathcal{I}^h$, one constructs a zero-padded random matrix $C^P_j \in \mathbb{C}^{N\times(r+k)}$ by padding zero to C_j. From the relationship

$$KC^P_j = K\begin{pmatrix} 0 \\ C_j \\ 0 \end{pmatrix} = \begin{pmatrix} K^h_{0,j}C_j \\ \vdots \\ K^h_{4^h-1,j}C_j \end{pmatrix}, \tag{4.31}$$

it is clear that applying K to the matrix C^P_j produces $K^h_{i,j}C_j$ for all $i \in \mathcal{I}^h$. Similarly, we construct zero-padded random matrices $R^P_i \in \mathbb{C}^{N\times(r+k)}$ by padding zero to R_i and compute

$$K^*R^P_i = K^*\begin{pmatrix} 0 \\ R_i \\ 0 \end{pmatrix} = \begin{pmatrix} \left(K^h_{i,0}\right)^* R_i \\ \vdots \\ \left(K^h_{i,4^h-1}\right)^* R_i \end{pmatrix} \tag{4.32}$$

by using the black-box routine for applying the adjoint of K. Finally, the approximated rank-r SVD of $K^h_{i,j}$ for each pair of $i \in \mathcal{I}^h$ and $j \in \mathcal{I}^h$ is computed from $K^h_{i,j}C_j$

and $\left(K^h_{i,j}\right)^* R_i$.

In Case (ii), since an arbitrary entry of K can be evaluated in $O(1)$ operations, the approximate rank-r SVD of $K^h_{i,j}$ is computed using the SVD via random sampling (Section 4.2.2).

In both cases, once the approximate rank-r SVD is ready, we transform it into the form of (4.3):

$$K^h_{i,j} \approx U^h_{i,j} S^h_{i,j} \left(V^h_{j,i}\right)^*. \tag{4.33}$$

Here the columns of the left and right factors $U^h_{i,j}$ and $V^h_{j,i}$ are scaled by the singular values of $K^h_{i,j}$ such that $U^h_{i,j}$ and $V^h_{j,i}$ keep track of the importance of the column and row bases for further factorizations.

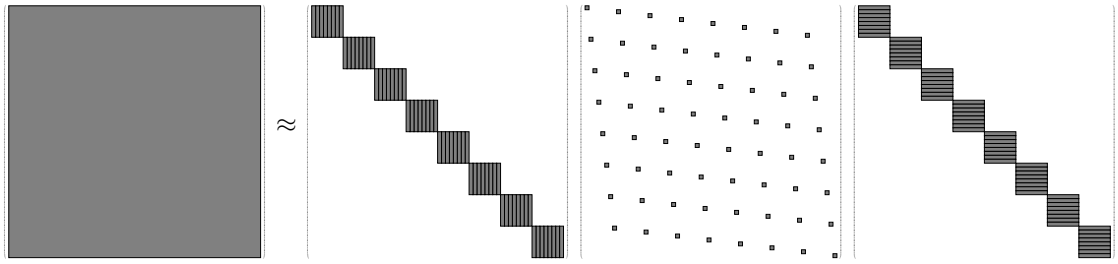

Figure 4.5: The middle level factorization of a complementary low-rank matrix $K \approx U^2 M^2 (V^2)^*$ where $N = n^2 = 4^2$ and $r = 1$. Grey blocks indicate nonzero blocks. U^2 and V^2 are block-diagonal matrices with 4 blocks. The diagonal blocks of U^2 and V^2 are assembled according to Equation (4.35) and (4.36) as indicated by gray rectangles. M^2 is a 4×4 block matrix with each block $M^2_{i,j}$ itself being an 4×4 block matrix containing diagonal weight matrix on the (j, i) block.

After computing the rank-r factorization in (4.33) for all i and j in $\mathcal{I}^h$, we assemble all left factors $U^h_{i,j}$ into a matrix U^h, all middle factors into a matrix M^h, and all right factors into a matrix V^h so that

$$K \approx U^h M^h (V^h)^*. \tag{4.34}$$

Here U^h is a block diagonal matrix of size $N \times rN$ with n diagonal blocks U_i^h of size $n \times rn$:

$$U^h = \begin{pmatrix} U_0^h & & & \\ & U_1^h & & \\ & & \ddots & \\ & & & U_{4^h-1}^h \end{pmatrix},$$

where each diagonal block U_i^h consists of the left factors $U_{i,j}^h$ for all j as follows:

$$U_i^h = \begin{pmatrix} U_{i,0}^h & U_{i,1}^h & \cdots & U_{i,4^h-1}^h \end{pmatrix} \in \mathbb{C}^{n \times rn}. \tag{4.35}$$

Similarly, V^h is a block diagonal matrix of size $N \times rN$ with n diagonal blocks V_j^h of size $n \times rn$, where each diagonal block V_j^h consists of the right factors $V_{j,i}^h$ for all i as follows:

$$V_j^h = \begin{pmatrix} V_{j,0}^h & V_{j,1}^h & \cdots & V_{j,4^h-1}^h \end{pmatrix} \in \mathbb{C}^{n \times rn}. \tag{4.36}$$

The middle matrix $M^h \in \mathbb{C}^{rN \times rN}$ is an $n \times n$ block matrix. The (i,j)-th block $M_{i,j}^h \in \mathbb{C}^{rn \times rn}$ is itself an $n \times n$ block matrix. The only nonzero block of $M_{i,j}^h$ is the (j,i)-th block, which is equal to the $r \times r$ matrix $S_{i,j}^h$, and the other blocks of $M_{i,j}^h$ are zero. We refer to Figure 4.5 for a simple example of the middle level factorization when $N = 4^2$.

4.4.1.3 Recursive factorization

In this section, we shall discuss how to recursively factorize

$$U^\ell \approx U^{\ell+1} G^\ell \tag{4.37}$$

and

$$(V^\ell)^* \approx (H^\ell)^* (V^{\ell+1})^* \tag{4.38}$$

for $\ell = h, h+1, \ldots, L-1$. After these recursive factorizations, we can construct the two-dimensional butterfly factorization

$$K \approx U^L G^{L-1} \cdots G^h M^h \left(H^h\right)^* \cdots \left(H^{L-1}\right)^* \left(V^L\right)^* \tag{4.39}$$

by substituting these recursive factorizations into (4.34).

Let us first consider the recursive factorization of U^h.

In the middle level factorization, we utilized the low-rank property of $K^h_{i,j}$, the kernel matrix restricted in the domain $A^h_i \times B^h_j \in T_X \times T_\Omega$, to obtain $U^h_{i,j}$ for $i, j \in \mathcal{I}^h$. We shall now use the complementary low-rank property at level $\ell = h+1$, i.e., the matrix $K^{h+1}_{i,j}$ restricted in $A^{h+1}_i \times B^{h-1}_j \in T_X \times T_\Omega$ is numerical low-rank for $i \in \mathcal{I}^{h+1}$ and $j \in \mathcal{I}^{h-1}$. These factorizations of the column bases from level h generate the column bases at level $h+1$ through the following four steps: splitting, merging, truncating, and assembling.

Splitting. In the middle level factorization, we have constructed

$$U^h = \begin{pmatrix} U^h_0 & & & \\ & U^h_1 & & \\ & & \ddots & \\ & & & U^h_{4^h-1} \end{pmatrix} \quad \text{with} \quad U^h_i = \begin{pmatrix} U^h_{i,0} & U^h_{i,1} & \cdots & U^h_{i,4^h-1} \end{pmatrix} \in \mathbb{C}^{n \times rn},$$

where each $U^h_{i,j} \in \mathbb{C}^{n \times r}$. Each node A^h_i in the quadtree T_X on the level h has four child nodes on the level $h+1$, denoted by $\{A^{h+1}_{4i+t}\}_{t=0,1,2,3}$. According to this structure,

one can split $U_{i,j}^h$ into four parts in the row space,

$$U_{i,j}^h = \begin{pmatrix} U_{i,j}^{h,0} \\ \hline U_{i,j}^{h,1} \\ \hline U_{i,j}^{h,2} \\ \hline U_{i,j}^{h,3} \end{pmatrix}, \tag{4.40}$$

where $U_{i,j}^{h,t}$ approximately spans the column space of the submatrix of K restricted to $A_{4i+t}^{h+1} \times B_j^h$ for each $t = 0, \ldots, 3$. Combining this with the definition of U_i^h gives rise to

$$U_i^h = \begin{pmatrix} U_{i,0}^h & U_{i,1}^h & \cdots & U_{i,4^h-1}^h \end{pmatrix} = \begin{pmatrix} U_{i,0}^{h,0} & U_{i,1}^{h,0} & \cdots & U_{i,4^h-1}^{h,0} \\ \hline U_{i,0}^{h,1} & U_{i,1}^{h,1} & \cdots & U_{i,4^h-1}^{h,1} \\ \hline U_{i,0}^{h,2} & U_{i,1}^{h,2} & \cdots & U_{i,4^h-1}^{h,2} \\ \hline U_{i,0}^{h,3} & U_{i,1}^{h,3} & \cdots & U_{i,4^h-1}^{h,3} \end{pmatrix} =: \begin{pmatrix} U_i^{h,0} \\ \hline U_i^{h,1} \\ \hline U_i^{h,2} \\ \hline U_i^{h,3} \end{pmatrix}, \tag{4.41}$$

where $U_i^{h,t}$ approximately spans the column space of the matrix K restricted to $A_{4i+t}^{h+1} \times \Omega$.

Merging. The merging step merges adjacent matrices $U_{i,j}^{h,t}$ in the column space to obtain low-rank matrices. For any $i \in \mathcal{I}^h$ and $j \in \mathcal{I}^{h-1}$, the merged matrix

$$\begin{pmatrix} U_{i,4j+0}^{h,t} & U_{i,4j+1}^{h,t} & U_{i,4j+2}^{h,t} & U_{i,4j+3}^{h,t} \end{pmatrix} \in \mathbb{C}^{n/4 \times 4r} \tag{4.42}$$

approximately spans the column space of $K_{4i+t,j}^{h+1}$ corresponding to the domain $A_{4i+t}^{h+1} \times B_j^{h-1}$. By the complementary low-rank property of the matrix K, we know $K_{4i+t,j}^{h+1}$

is numerically low-rank. Hence, the matrix in (4.42) is also a numerically low-rank matrix. This is the merging step equivalent to moving from level h to level $h-1$ in T_Ω.

Truncating. The third step computes its rank-r approximation using the standard truncated SVD and putting it to the form of (4.4). For each $i \in \mathcal{I}^h$ and $j \in \mathcal{I}^{h-1}$, the factorization

$$\begin{pmatrix} U^{h,t}_{i,4j+0} & U^{h,t}_{i,4j+1} & U^{h,t}_{i,4j+2} & U^{h,t}_{i,4j+3} \end{pmatrix} \approx U^{h+1}_{4i+t,j} G^h_{4i+t,j}, \tag{4.43}$$

defines $U^{h+1}_{4i+t,j} \in \mathbb{C}^{n/4 \times r}$ and $G^h_{4i+t,j} \in \mathbb{C}^{r \times 4r}$.

Assembling In the final step, we construct the factorization $U^h \approx U^{h+1} G^h$ using (4.43). Since $\mathcal{I}^{h+1}$ is the same as $\{4i+t\}_{i \in \mathcal{I}^h, t=0,1,2,3}$, one can arrange (4.43) for all i

and j into a single formula as follows:

$$U^h \approx U^{h+1} G^h =$$

$$\begin{pmatrix} U_0^{h+1} & & & & & & & & & & \\ & \ddots & & & & & & & & & \\ & & U_3^{h+1} & & & & & & & & \\ & & & U_4^{h+1} & & & & & & & \\ & & & & \ddots & & & & & & \\ & & & & & U_7^{h+1} & & & & & \\ & & & & & & \ddots & & & & \\ & & & & & & & U_{4^{h+1}-4}^{h+1} & & & \\ & & & & & & & & \ddots & & \\ & & & & & & & & & U_{4^{h+1}-1}^{h+1} & \end{pmatrix} \begin{pmatrix} G_0^h & & & \\ \vdots & & & \\ G_3^h & & & \\ & G_4^h & & \\ & \vdots & & \\ & G_7^h & & \\ & & \ddots & \\ & & & G_{4^{h+1}-4}^h \\ & & & \vdots \\ & & & G_{4^{h+1}-1}^h \end{pmatrix},$$

where the blocks are given by

$$U_i^{h+1} = \begin{pmatrix} U_{i,0}^{h+1} & U_{i,1}^{h+1} & \cdots & U_{i,4^{h-1}-1}^{h+1} \end{pmatrix}$$

and

$$G_i^h = \begin{pmatrix} G_{i,0}^h & & & \\ & G_{i,1}^h & & \\ & & \ddots & \\ & & & G_{i,4^{h-1}-1}^h \end{pmatrix}$$

for $i \in \mathcal{I}^{h+1}$. Figure 4.6 shows a toy example of the recursive factorization of U^h when $N = 4^2$, $h = 2$ and $r = 1$. Since there are $O(1)$ nonzero entries in each $G_{i,j}^h$ and $O(4^{h+1} \cdot 4^{h-1}) = O(N)$ such matrices, there are only $O(N)$ nonzero entries in G^h.

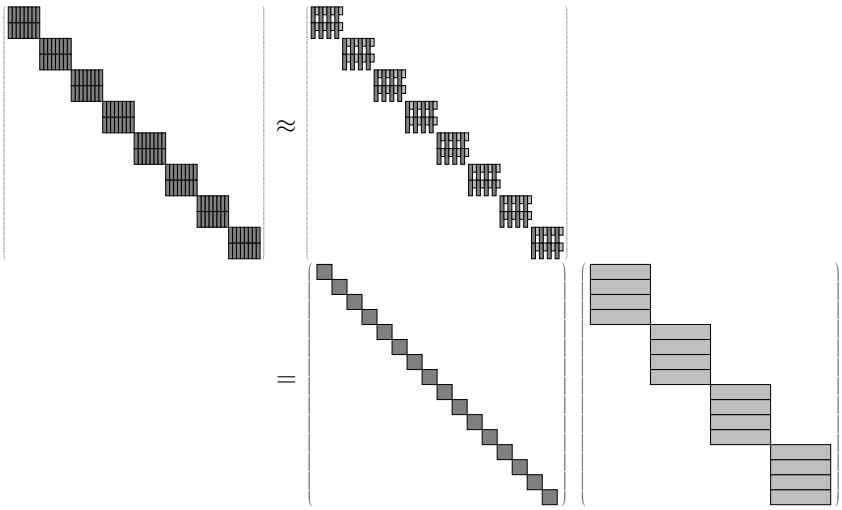

Figure 4.6: The recursive factorization of U^2 in Figure 4.5. Left matrix: U^2 with each diagonal block partitioned into smaller blocks according to Equation (4.40) as indicated by black rectangles; Middle-left matrix: low-rank approximations of submatrices in U^2 given by Equation (4.43); Middle right matrix: U^3; Right matrix: G^2.

In a similar way, we can now factorize $U^\ell \approx U^{\ell+1} G^\ell$ for $h < \ell \leqslant L-1$. As before, the key point is that the columns of

$$\begin{pmatrix} U^{\ell,t}_{i,4j+0} & U^{\ell,t}_{i,4j+1} & U^{\ell,t}_{i,4j+2} & U^{\ell,t}_{i,4j+3} \end{pmatrix} \tag{4.44}$$

approximately span the column space of $K^{\ell+1}_{4i+t,j}$, which is of rank r numerically due to the complementary low-rank property. Computing its rank-r approximation via the standard truncated SVD results in a form of (4.4)

$$\begin{pmatrix} U^{\ell,t}_{i,4j+0} & U^{\ell,t}_{i,4j+1} & U^{\ell,t}_{i,4j+2} & U^{\ell,t}_{i,4j+3} \end{pmatrix} \approx U^{\ell+1}_{4i+t,j} G^\ell_{4i+t,j} \tag{4.45}$$

for $i \in \mathcal{I}^\ell$ and $j \in \mathcal{I}^{L-\ell-1}$. After assembling these factorizations together, we obtain

$$U^\ell \approx U^{\ell+1} G^\ell = \begin{pmatrix} U_0^{\ell+1} & & & & & & & & \\ & \ddots & & & & & & & \\ & & U_3^{\ell+1} & & & & & & \\ & & & U_4^{\ell+1} & & & & & \\ & & & & \ddots & & & & \\ & & & & & U_7^{\ell+1} & & & \\ & & & & & & \ddots & & \\ & & & & & & & U_{4^{\ell+1}-4}^{\ell+1} & \\ & & & & & & & & \ddots \\ & & & & & & & & U_{4^{\ell+1}-1}^{\ell+1} \end{pmatrix} \begin{pmatrix} G_0^\ell & & & \\ \vdots & & & \\ G_3^\ell & & & \\ & G_4^\ell & & \\ & \vdots & & \\ & G_7^\ell & & \\ & & \ddots & \\ & & & G_{4^{\ell+1}-4}^\ell \\ & & & \vdots \\ & & & G_{4^{\ell+1}-1}^\ell \end{pmatrix},$$

where

$$U_i^{\ell+1} = \begin{pmatrix} U_{i,0}^{\ell+1} & U_{i,1}^{\ell+1} & \cdots & U_{i,4^{L-\ell-1}-1}^{\ell+1} \end{pmatrix}$$

and

$$G_i^\ell = \begin{pmatrix} G_{i,0}^\ell & & & \\ & G_{i,1}^\ell & & \\ & & \ddots & \\ & & & G_{i,4^{L-\ell-1}-1}^\ell \end{pmatrix}$$

for $i \in \mathcal{I}^{\ell+1}$.

After the $L-h$ step of recursive factorizations $U^\ell \approx U^{\ell+1} G^\ell$ for $\ell = h, h+1, \ldots, L-1$, the recursive factorization of U^h takes the following form:

$$U^h \approx U^L G^{L-1} \cdots G^h. \tag{4.46}$$

Similarly to the analysis of G^h, it is also easy to check that there are only $O(N)$ nonzero entries in each G^ℓ in (4.46). As to the first factor U^L, it has $O(N)$ nonzero entries since there are $O(N)$ diagonal blocks in U^L and each block contains $O(1)$ entries.

Now we analogically conduct the recursive factorization of V^h.

The recursive factorization of V^ℓ is similar to that of U^ℓ for $\ell = h, h+1, \ldots, L-1$. At each level ℓ, we benefit from the fact that

$$\begin{pmatrix} V^{\ell,t}_{j,4i+0} & V^{\ell,t}_{j,4i+1} & V^{\ell,t}_{j,4i+2} & V^{\ell,t}_{j,4i+3} \end{pmatrix}$$

approximately spans the row space of $K^{L-\ell-1}_{i,4j+t}$ and hence is numerically low-rank for $j \in \mathcal{I}^{L-\ell}$ and $i \in \mathcal{I}^{\ell-1}$. Applying the same procedure to V^h leads to

$$V^h \approx V^L H^{L-1} \cdots H^h. \tag{4.47}$$

4.4.1.4 Complexity analysis

By combining the results of the middle level factorization in (4.34) and the recursive factorizations in (4.46) and (4.47), we obtain the final butterfly factorization

$$K \approx U^L G^{L-1} \cdots G^h M^h \left(H^h\right)^* \cdots \left(H^{L-1}\right)^* \left(V^L\right)^*, \tag{4.48}$$

each factor of which contains $O(N)$ nonzero entries. We refer to Figure 4.7 for an illustration of the butterfly factorization of K when $N = 16^2$.

The complexity of constructing the butterfly factorization comes from two parts: the middle level factorization and the recursive factorization. For the middle level factorization, the construction cost is different depending on which of the two cases

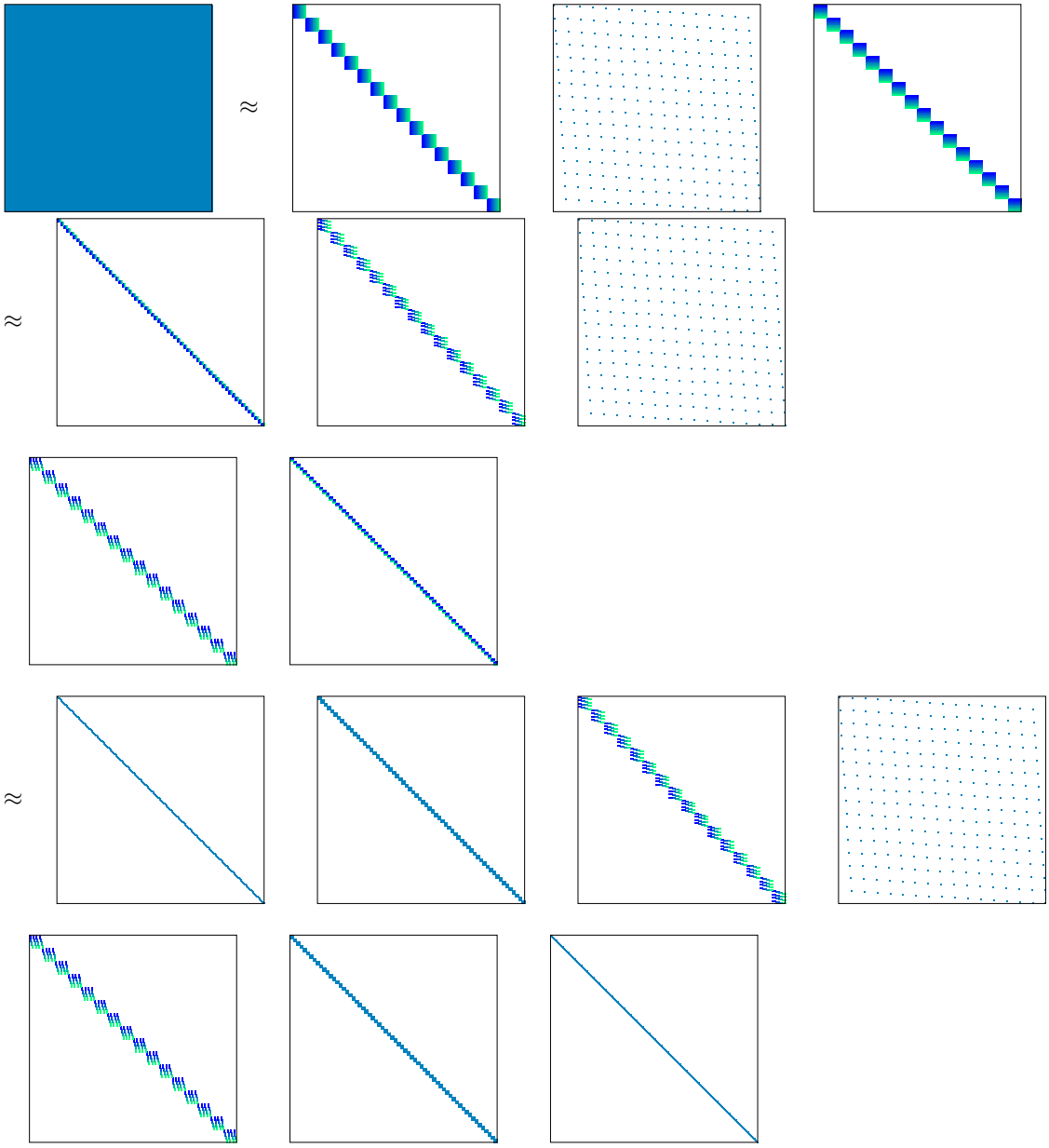

Figure 4.7: A full butterfly factorization for a two-dimensional problem of size 16^2 and fixed rank $r = 1$. The above figure visualizes the matrices in $K \approx U^3 M^3 (V^3)^* \approx U^4 G^3 M^3 (H^3)^* (V^4)^* \approx U^5 G^4 G^3 M^3 (H^3)^* (H^4)^* (V^5)^*$.

mentioned in Assumption 4.3.1 is under consideration, since they use different approaches in constructing rank-r SVDs at the middle level.

- In Case (i), the dominant cost is to apply K and K^* to $N^{1/2}$ Gaussian random matrices of size $N \times O(1)$. Assuming that the given black-box routine for applying K and K^* to a vector takes $O(C_K(N))$ operations, the total operation complexity is $O(C_K(N)N^{1/2})$.

- In Case (ii), we apply the SVD procedure with random sampling to N submatrices of size $N^{1/2} \times N^{1/2}$. Since the operation complexity for each submatrix is $O(N^{1/2})$, the overall complexity is $O(N^{3/2})$.

In the recursive factorization stage, most of the work comes from factorizing U^h and V^h. There are $O(\log N)$ stages appeared in the factorization of U^h. At the ℓ stage, the matrix U^ℓ to be factorized consists of 4^ℓ diagonal blocks. There are $O(N)$ factorizations and each factorization takes $O(N/4^\ell)$ operations. Hence, the operation complexity to factorize U^ℓ is $O(N^2/4^\ell)$. Summing up all the operations in each step yields the overall operation complexity for recursively factorizing U^h:

$$\sum_{\ell=h}^{L-1} O(N^2/4^\ell) = O(N^{3/2}). \tag{4.49}$$

The peak of the memory usage of the butterfly factorization is due to the middle level factorization where we need to store the results of $O(N)$ factorizations of size $O(N^{1/2})$. Hence, the memory complexity for the two-dimensional butterfly factorization is $O(N^{3/2})$. For Case (ii), one can actually do better by following the same argument in [59]. One can interleave the order of generation and recursive factorization of $U^h_{i,j}$ and $V^h_{j,i}$. By factorizing $U^h_{i,j}$ and $V^h_{j,i}$ individually instead of formulating (4.34), the memory complexity in Case (ii) can be reduced to $O(N \log N)$.

The cost of applying the butterfly factorization is equal to the number of nonzero entries in the final factorization, which is $O(N \log N)$. Table 4.5 summarizes the complexity analysis for the two-dimensional butterfly factorization. Comparing Table 4.1 and Table 4.5, we find that the complexity for multidimensional butterfly factorization is actually the same as the one-dimensional butterfly factorization.

		SVD via rand. matvec	SVD via rand. sampling
Factorization Complexity	Middle level factorization	$O(C_K(N)N^{1/2})$	$O(N^{3/2})$
	Recursive factorization	$O(N^{3/2})$	
	Total	$O(C_K(N)N^{1/2})$	$O(N^{3/2})$
Memory Complexity		$O(N^{3/2})$	$O(N \log N)$
Application Complexity		$O(N \log N)$	

Table 4.5: The time and memory complexity of the two-dimensional butterfly factorization. Here $C_K(N)$ is the complexity of applying the matrices K and K^* to a vector. For most butterfly algorithms, $C_K(N) = O(N \log N)$.

4.4.1.5 Extensions

We have introduced the two-dimensional butterfly factorization for a complementary low-rank kernel matrix K in the entire domain $X \times \Omega$. Although we have assumed the uniform grid in (3.2) and (3.3), the butterfly factorization extends naturally to more general settings.

In the case with non-uniform point sets X or Ω, one can still construct a butterfly factorization for K following the same procedure. More specifically, we still construct two trees T_X and T_Ω *adaptively* via hierarchically partitioning the square domains covering X and Ω. For non-uniform point sets X and Ω, the numbers of points in

A_i^ℓ and $B_j^{L-\ell}$ are different. If a node does not contain any point inside it, it is simply discarded from the quadtree.

The complexity analysis summarized in Table 4.5 remains valid in the case of non-uniform point sets X and Ω. On each level $\ell = h, \ldots, L$ of the butterfly factorization, although the sizes of low-rank submatrices are different, the total number of submatrices and the numerical rank remain the same. Hence, the total operation and memory complexity remains the same as summarized in Table 4.5.

4.4.2 Polar butterfly factorization

In the previous section, we have introduced a two-dimensional butterfly factorization for a complementary low-rank kernel matrix K in the joint domain $X \times \Omega$. In this section, we will introduce a polar butterfly factorization to deal with the kernel function $K(x, \xi) = e^{2\pi\imath\Phi(x,\xi)}$. Such a kernel matrix has a singularity at $\xi = 0$ and the approach taken here follows the polar butterfly algorithm proposed in [17] and reviewed in Section 3.2. Although we have shown in Section 3.5 that the multiscale butterfly algorithm has smaller pre-factor than the polar butterfly algorithm, the polar butterfly algorithm is still widely used when the points in Ω is on the polar grid [53] and its application should be accelerated as well.

4.4.2.1 Factorization algorithm

Combining the polar butterfly algorithm with the butterfly factorization outlined in Section 4.4.1 gives rise to the following polar butterfly factorization (PBF).

1. *Preliminary.* Take the polar transformation of each point in Ω and reformulate

the problem

$$u(x) = \sum_{\xi \in \Omega} e^{2\pi \imath \Phi(x,\xi)} g(\xi), \quad x \in X, \tag{4.50}$$

into

$$u(x) = \sum_{p \in P} e^{2\pi \imath n \Psi(x,p)} g(p), \quad x \in X. \tag{4.51}$$

2. *Factorization.* Apply the two-dimensional butterfly factorization to the kernel $e^{2\pi \imath n \Psi(x,p)}$ defined on a non-uniform point set in $X \times P$. The corresponding kernel matrix is approximated as

$$K \approx U^L G^{L-1} \cdots G^h M^h \left(H^h\right)^* \cdots \left(H^{L-1}\right)^* \left(V^L\right)^*. \tag{4.52}$$

Since the polar butterfly factorization essentially applies the original butterfly factorization to non-uniform point sets X and P, it has the same complexity as summarized in Table 4.5. Depending on the SVD procedure employed in the middle level factorization, we refer to it either as PBF-m (when SVD via random matrix-vector multiplication is used) or as PBF-s (when SVD via random sampling is used).

4.4.2.2 Numerical results

This section presents two numerical examples to demonstrate the efficiency of the polar butterfly factorization. The numerical results were obtained in MATLAB on a server with 2.40 GHz CPU and 1.5 TB of memory.

In this section, we denote by $\{u^p(x)\}_{x \in X}$ the results obtained via the PBF. The relative error of the PBF is estimated as follows, by comparing $u^p(x)$ with the exact

values $u(x)$.

$$e^p = \sqrt{\frac{\sum_{x\in S} |u^p(x) - u(x)|^2}{\sum_{x\in S} |u(x)|^2}}, \tag{4.53}$$

where S is a set of 256 randomly sampled points from X.

Example 1. This example is a two-dimensional generalized Radon transform that is an FIO defined as follows:

$$u(x) = \sum_{\xi\in\Omega} e^{2\pi\imath\Phi(x,\xi)} g(\xi), \quad x \in X, \tag{4.54}$$

with the phase function given by

$$\begin{aligned} \Phi(x,\xi) =& x\cdot\xi + \sqrt{c_1^2(x)\xi_1^2 + c_2^2(x)\xi_2^2}, \\ c_1(x) =& (2+\sin(2\pi x_1)\sin(2\pi x_2))/16, \\ c_2(x) =& (2+\cos(2\pi x_1)\cos(2\pi x_2))/16, \end{aligned} \tag{4.55}$$

where X and Ω are defined in (3.2) and (3.3). The computation in (4.54) approximately integrates over spatially varying ellipses, for which $c_1(x)$ and $c_2(x)$ are the axis lengths of the ellipse centered at the point $x \in X$. The corresponding matrix form of (4.54) is simply

$$u = Kg, \quad K = (e^{2\pi\imath\Phi(x,\xi)})_{x\in X,\xi\in\Omega}. \tag{4.56}$$

As $e^{2\pi\imath\Phi(x,\xi)}$ is known explicitly, we are able to use the PBF-s (i.e., the one with random sampling in the middle level factorization) to approximate the kernel matrix K given by $e^{2\pi\imath\Phi(x,\xi)}$. After the construction of the butterfly factorization, the summation in (4.54) can be evaluated efficiently by applying these sparse factors to $g(\xi)$.

Table 4.6 summarizes the results of this example.

n, r	ϵ^p	$T_{f,p}(min)$	$T_p(sec)$	Speedup
64,6	2.46e-02	6.51e-01	2.37e-02	1.54e+02
128,6	7.55e-03	9.84e+00	2.30e-01	1.67e+02
256,6	5.10e-02	2.73e+01	6.23e-01	7.55e+02
512,6	1.46e-02	4.00e+02	7.88e+00	4.15e+02
64,14	7.93e-04	7.34e-01	5.98e-02	8.72e+01
128,14	7.28e-04	1.17e+01	7.15e-01	4.28e+01
256,14	2.15e-03	3.93e+01	1.46e+00	2.86e+02
512,14	1.25e-03	5.63e+02	1.05e+01	3.35e+02
64,22	6.96e-05	7.40e-01	8.24e-02	4.51e+01
128,22	7.23e-05	1.16e+01	1.04e+00	3.69e+01
256,22	2.44e-04	5.14e+01	5.94e+00	7.74e+01

Table 4.6: Numerical results provided by the PBF with random sampling algorithm for the FIO in (4.54). n is the number of grid points in each dimension; $N = n^2$ is the size of the kernel matrix; r is the max rank used in the low-rank approximation; $T_{f,p}$ is the factorization time of the PBF; T_d is the running time of the direct evaluation; T_p is the application time of the PBF. The last column shows the speedup factor compared to the direct evaluation.

Example 2. This example evaluates the composition of two FIOs with the same phase function $\Phi(x, \xi)$. This is given explicitly by

$$u(x) = \sum_{\eta \in \Omega} e^{2\pi \imath \Phi(x,\eta)} \sum_{y \in X} e^{-2\pi \imath y \cdot \eta} \sum_{\xi \in \Omega} e^{2\pi \imath \Phi(y,\xi)} g(\xi), \quad x \in X, \tag{4.57}$$

where the phase function is given in (4.55). The corresponding matrix representation is

$$u = KFKg, \tag{4.58}$$

where K is the matrix given in (4.56) and F is the matrix representation of the discrete Fourier transform. Under relatively mild assumptions (see [52] for details),

the composition of two FIOs is again an FIO. Hence, the kernel matrix

$$\widetilde{K} := KFK \tag{4.59}$$

of the product can be approximated by the butterfly factorization. Notice that the kernel function of $\widetilde{K}$ defined by (4.59) is not given explicitly. However, (4.59) provides fast algorithms for applying $\widetilde{K}$ and its adjoint through the fast algorithms for K and F. For example, the butterfly factorization of Example 1 enables the efficient application of K and K^* in $O(N \log N)$ operations. Applying of F and F^* can be done by the fast Fourier transform in $O(N \log N)$ operations. Therefore, we can apply the PBF-m (i.e., the one with random matrix-vector multiplication) to factorize the kernel $\widetilde{K} = KFK$. Table 4.7 summarizes the numerical results of this example, the composing of two FIOs.

n, r	ϵ^p	$T_{f,p}(min)$	$T_p(sec)$	Speedup
64,12	3.84e-02	6.22e+00	2.18e-02	3.34e+02
128,12	1.31e-02	3.86e+02	1.80e-01	4.25e+02
64,20	2.24e-03	8.58e+00	3.04e-02	2.39e+02
128,20	2.23e-03	3.68e+02	3.60e-01	2.13e+02

Table 4.7: Numerical results provided by the PBF with random SVD algorithm for the composition of FIOs given in (4.58).

Discussion. The numerical results in Tables 4.6 and 4.7 support the asymptotic complexity analysis. When we fix r and let n grow, the actually running time fluctuates around the asymptotic scaling since the implementation of the algorithms differ slightly depending on whether L is odd or even. However, the overall trend matches well with the $O(N^{3/2})$ construction cost and the $O(N \log N)$ application cost. For a fixed n, one can improve the accuracy by increasing the truncation rank r. From

the tables, one observes that the relative error decreases by a factor of 10 when we increase the rank r by 8 every time. In the second example, since the composition of two FIOs typically has higher ranks compared to a single FIO, the numerical rank r used for the composition is larger than that for a single FIO in order to maintain comparable accuracy.

4.4.3 Multiscale butterfly factorization

In this section, we discuss yet another approach for constructing butterfly factorization for the kernel $K(x,\xi) = e^{2\pi\imath\Phi(x,\xi)}$ with singularity at $\xi = 0$. This is based on the multiscale butterfly algorithm introduced Section 3.4.

4.4.3.1 Factorization algorithm

Combining the multiscale butterfly algorithm with the butterfly factorization outlined in Section 4.4.1 gives rise to the following multiscale butterfly factorization (MBF).

1. *Preliminary.* Decompose domain Ω into subdomains as in (**??**). Reformulate the problem into a multiscale summation according to (**??**):

$$K = K_C R_C + \sum_{t=0}^{\log_2 n - s} K_t R_t. \tag{4.60}$$

   Here K_C and K_t are kernel matrices corresponding to $X \times \Omega_C$ and $X \times \Omega_t$. R_C and R_t are the restriction operators to the domains Ω_C and Ω_t respectively.

2. *Factorization.* Recall that $L = \log_2 n$. For each $t = 0, 1, \ldots, L-s$, apply the two-dimensional butterfly factorization on $K(x,\xi) = e^{2\pi\imath\Phi(x,\xi)}$ restricted in $X \times \Omega_t$. Let $\widetilde{\Omega}_t$ be the smallest square that contains Ω_t. Define $L_t = 2\lfloor (L-t)/2 \rfloor$, where

$\lfloor \cdot \rfloor$ is the largest integer less than or equal to a given number. We construct two quadtrees T_X and $T_{\widetilde{\Omega}_t}$ of depth L_j with X and $\widetilde{\Omega}_t$ being the roots, respectively. Applying the two-dimensional butterfly factorization using the quadtrees T_X and $T_{\widetilde{\Omega}_t}$ gives the t-th butterfly factorization:

$$K_t \approx U_t^{L_t} G_t^{L_t-1} \cdots G_t^{\frac{L_t}{2}} M_t^{\frac{L_t}{2}} \left(H_t^{\frac{L_t}{2}} \right)^* \cdots \left(H_t^{L_t-1} \right)^* \left(V_t^{L_t} \right)^* .$$

Note that 1/4 of the tree $T_{\widetilde{\Omega}_t}$ is empty and we can simply ignore the computation for these parts. This is a special case of non-uniform point sets. Once we have computed all butterfly factorizations, the multiscale summation in (4.60) is approximated by

$$K \approx K_C R_C + \sum_{t=0}^{L-s} U_t^{L_t} G_t^{L_t-1} \cdots M_t^{\frac{L_t}{2}} \cdots \left(H_t^{L_t-1} \right)^* \left(V_t^{L_t} \right)^* R_t. \tag{4.61}$$

The idea of the hierarchical decomposition of Ω not only avoids the singularity of $K(x, \xi)$ at $\xi = 0$, but also maintains the efficiency of the butterfly factorization. The butterfly factorization for the kernel matrix restricted in $X \times \Omega_t$ is a special case of non-uniform butterfly factorization in which the center of Ω_t contains no point. Since the number of points in Ω_t is decreasing exponentially in t, the operation and memory complexity of the multiscale butterfly factorization is dominated by the butterfly factorization of K_t for $t = 0$, which is bounded by the complexity summarized in Table 4.5. Depending on the SVD procedure in the middle level factorization, we refer this factorization either as MBF-m (when SVD via random matrix-vector multiplication is used) or as MBF-s (when SVD via random sampling is used).

4.4.3.2 Numerical results

This section presents two numerical examples to demonstrate the efficiency of the MBF as well. The numerical results are obtained in the same environment as the one used in Section 4.4.2.2. Here we denote by $\{u^m(x), x \in X\}$ the results obtained via the MBF. The relative error is estimated by

$$e^m = \sqrt{\frac{\sum_{x\in S} |u^m(x) - u(x)|^2}{\sum_{x\in S} |u(x)|^2}}, \tag{4.62}$$

where S is a set of 256 randomly sampled from X. In the multiscale decomposition of Ω, we recursively divide Ω until the center part is of size 16 by 16.

Example 1. We revisit the first example in Section 4.4.2.2 to illustrate the performance of the MBF,

$$u(x) = \sum_{\xi\in\Omega} e^{2\pi\imath\Phi(x,\xi)} g(\xi), \quad x \in X, \tag{4.63}$$

with a kernel $\Phi(x, \xi)$ given by

$$\begin{aligned} \Phi(x,\xi) =& x\cdot\xi + \sqrt{c_1^2(x)\xi_1^2 + c_2^2(x)\xi_2^2}, \\ c_1(x) =& (2 + \sin(2\pi x_1)\sin(2\pi x_2))/16, \\ c_2(x) =& (2 + \cos(2\pi x_1)\cos(2\pi x_2))/16, \end{aligned} \tag{4.64}$$

where X and Ω are defined in (3.2) and (3.3). Table 4.8 summarizes the results of this example obtained by applying the MBF-s.

Example 2. Here we revisit the second example in Section 4.4.2.2 to illustrate the performance of the MBF. Recall that the matrix representation of a composition of

n, r	ϵ^m	$T_{f,m}(min)$	$T_m(sec)$	Speedup
64,12	1.58e-02	4.48e-01	4.09e-02	1.13e+02
128,12	1.47e-02	5.64e+00	1.93e-01	2.02e+02
256,12	2.13e-02	2.16e+01	5.51e-01	9.26e+02
512,12	1.97e-02	2.97e+02	5.07e+00	6.45e+02
64,20	5.51e-03	4.74e-01	6.11e-02	6.17e+01
128,20	4.27e-03	5.95e+00	5.01e-01	7.63e+01
256,20	1.68e-03	3.03e+01	2.51e+00	1.79e+02
512,20	2.02e-03	4.57e+02	1.14e+01	2.98e+02
64,28	7.42e-05	7.18e-01	3.92e-02	6.23e+01
128,28	8.46e-05	1.23e+01	5.42e-01	7.43e+01
256,28	5.63e-04	6.73e+01	3.23e+00	1.43e+02
512,28	4.18e-04	7.20e+02	1.66e+01	2.14e+02

Table 4.8: Numerical results provided by the MBF with the random sampling algorithm for the FIO given in (4.63). n is the number of grid points in each dimension; $N = n^2$ is the size of the kernel matrix; r is the max rank used in low-rank approximation; $T_{f,m}$ is the factorization time of the MBF; T_d is the running time of the direct evaluation; T_m is the application time of the MBF; T_d/T_m is the speedup factor.

two FIOs is

$$u = \widetilde{K}g = KFKg, \tag{4.65}$$

and that there are fast algorithms to apply K, F and their adjoints. Hence, we can apply the MBF-m (i.e., with the random matrix-vector multiplication) to factorize $\widetilde{K}$ into the form of (4.61). Table 4.9 summarizes the results.

n, r	ϵ^m	$T_{f,m}(min)$	$T_m(sec)$	Speedup
64,16	1.86e-02	4.05e+00	1.95e-02	4.23e+02
128,16	1.76e-02	1.27e+02	1.86e-01	4.17e+02
64,24	4.43e-03	5.37e+00	2.52e-02	3.27e+02
128,24	3.02e-03	1.79e+02	2.29e-01	3.40e+02

Table 4.9: MBF numerical results for the composition of FIOs given in (4.65).

Discussion. The results in Tables 4.8 and 4.9 agree with the $O(N^{3/2}\log N)$ complexity analysis of the construction algorithm. As we double the problem size n, the factorization time increases by a factor 9 on average. The actual application time in these numerical examples matches the theoretical operation complexity of $O(N\log N)$. In Table 4.8, the relative error decreases by a factor of 10 when the increment of the rank r is 6. In Table 4.9, the relative error decreases by a factor of 6 when the increment of the rank r is 8.

4.5 Remarks on parallelization

Similar to the parallelization of the multiscale butterfly algorithm in Section 3.6, the butterfly factorizations can also be parallelized efficiently. Assume that the problem is of size N and P processes are given, where $P \leqslant N$. In one dimension, the middle level factorization, recursive factorization, and application of the butterfly factorization can all be fully parallelized.

Assume two cases in the middle level factorization are parallelized.

(i) Only black-box routines for computing Kg and K^*g on P processes in $O(\frac{N}{P}\log N)$ operations are given;

(ii) Only a black-box routine for evaluating any entry of the matrix K on any process in $O(1)$ operations is given.

In the first case, the middle level factorization consists two steps: applying K and K^* to random vectors and constructing SVD within each block. The parallelization of the first step is done through the parallel black-box routines with an average cost $O(N^{1/2}\log N)$ per-block. Since $P \leqslant N$, each block on the middle level is assigned to

a single process. The following rank-r approximate SVD can be constructed locally, which cost $O(N^{1/2})$ operations and no communication in each block. The total cost for the middle level factorization in this case is $O(N^{1/2}\log N)$ per-block. The second case is even simpler. Every single SVD via random sampling is inside the block and calculated on a single process without communication. The complexity is $O(N^{1/2})$ per-block.

The parallelization of the recursive factorization of U^h can be done as follows. At the beginning of the recursive factorization, all N blocks $U^\ell_{i,2j}, U^\ell_{i,2j+1}$ for $i = 0, 1, \ldots, 2^\ell - 1$ and $j = 0, 1, \ldots, 2^{L-\ell-1} - 1$ are individually owned by single processes. Splitting them into top and bottom half, and merging two contiguous halves require an intra-block communication. The process that owns the block sends half of its matrix to the owner of a neighbor block. At the same time, it receives half of another matrix from the owner of that neighbor block. These one-to-one communications cost $O(\beta 2^\ell + \alpha)$ per-block, where α is the latency and β is the inverse bandwidth. Any other computation is calculated within the block. The recursive factorization of V^h can be parallelized in the same manner.

Summing up the operations and communications for middle level factorization and recursive factorization at all levels gives the total complexity per-process,

$$\begin{aligned} &O(\frac{N^{3/2}}{P}\log N) + \sum_{\ell=h}^{L-1}\left(O(\frac{N^2}{2^\ell P} + \beta\frac{N2^\ell}{P} + \alpha\frac{N}{P})\right) \\ =&O(\frac{N^{3/2}}{P}\log N + \beta\frac{N^{3/2}}{P} + \alpha\frac{N}{P}\log N), \end{aligned} \tag{4.66}$$

where N/P is the number of blocks owned by a process.

Since the butterfly factorization can be viewed as a compressed algebraic representation of the butterfly algorithm, the parallelization of the application of the butterfly

factorization exactly follows the parallel butterfly algorithm [76].

In a similar way, the multidimensional butterfly factorization, the polar butterfly factorization, and the multiscale butterfly factorization can be parallelized following the discussions in Section 3.6 and [76].

4.6 Conclusion

This chapter first introduces a butterfly factorization in one dimension as a data-sparse approximation of complementary low-rank matrices. More precisely, it represents such an $N \times N$ dense matrix as a product of $O(\log N)$ sparse matrices. The factorization can be built efficiently if either a fast algorithm for applying the matrix and its adjoint is available or an explicit expression for the entries of the matrix is given. The butterfly factorization gives rise to highly efficient matrix-vector multiplications with $O(N \log N)$ operation and memory complexity. The butterfly factorization is also useful when an existing butterfly algorithm is repeatedly applied, because the application of the butterfly factorization is significantly faster than pre-existing butterfly algorithms.

We also have introduced three multidimensional butterfly factorization as data-sparse representations of a class of kernel matrices coming from multidimensional integral transforms. When the integral kernel $K(x, \xi)$ satisfies the complementary low-rank property in the entire domain, the butterfly factorization introduced in Section 4.4.1 represents an $N \times N$ kernel matrix as a product of $O(\log N)$ sparse matrices. In the FIO case for which the kernel $K(x, \xi)$ is singular at $\xi = 0$, we propose two extensions: (1) the polar butterfly factorization that incorporates a polar coordinate transformation to remove the singularity and (2) the multiscale butterfly

factorization that relies on a hierarchical partitioning in the Ω domain. For both extensions, the resulting butterfly factorization takes $O(N \log N)$ storage space and $O(N \log N)$ steps for computing a single matrix-vector multiplication as before.

The butterfly factorization for higher dimensions ($d > 2$) can be constructed in a similar way. For the polar butterfly factorization, one simply applies a d-dimensional spherical transformation to the frequency domain Ω. For the multiscale butterfly factorization, one can again decompose the frequency domain as a union of dyadic shells centered round the singularity at $\xi = 0$.

Bibliography

[1] S. Ambikasaran and E. Darve. An $\mathcal{O}(N \log N)$ fast direct solver for partial hierarchically semi-separable matrices: with application to radial basis function interpolation. *SIAM J. Sci. Comput.*, 57(3):477–501, 2013.

[2] P. R. Amestoy, C. Ashcraft, O. Boiteau, A. Buttari, J.-Y. L'Excellent, and C. Weisbecker. Improving multifrontal methods by means of block low-rank representations. *SIAM J. Sci. Comput.*, 37(3):A1451–A1474, 2015.

[3] P. R. Amestoy, I. S. Duff, and J.-Y. L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. *Comput. Methods Appl. Mech. Eng.*, 184(24):501–520, 2000.

[4] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM J. Matrix Anal. Appl.*, 23(1):15–41, jan 2001.

[5] C. Anderson and M. D. Dahleh. Rapid computation of the discrete Fourier transform. *SIAM J. Sci. Comput.*, 17(4):913–919, jul 1996.

[6] F. Andersson, M. V. de Hoop, and H. Wendt. Multiscale discrete approximation of Fourier integral operators. *Multiscale Model. Simul.*, 10(1):111–145, jan 2012.

[7] A. Averbuch, E. Braverman, R. Coifman, M. Israeli, and A. Sidi. Efficient computation of oscillatory integrals via adaptive multiscale local Fourier bases. *Appl. Comput. Harmon. Anal.*, 9(1):19–53, jul 2000.

[8] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Minimizing communication in numerical linear algebra. *SIAM J. Matrix Anal. Appl.*, 32(3):866–901, 2011.

[9] G. Bao and W. W. Symes. Computation of pseudo-differential operators. *SIAM J. Sci. Comput.*, 17(2):416–429, mar 1996.

[10] M. Bebendorf. Efficient inversion of the Galerkin matrix of general second-order elliptic operators with nonsmooth coefficients. *Math. Comput.*, 74(251):1179–1199, 2005.

[11] M. Bebendorf and W. Hackbusch. Existence of $\mathcal{H}$-matrix approximants to the inverse FE-matrix of elliptic operators with L^∞-coefficients. *Numer. Math.*, 95(1):1–28, 2003.

[12] S. Börm. Approximation of solution operators of elliptic partial differential equations by $\mathcal{H}$- and $\mathcal{H}^2$-matrices. *Numer. Math.*, 115(2):165–193, 2010.

[13] B. Bradie, R. Coifman, and A. Grossmann. Fast numerical computations of oscillatory integrals related to acoustic scattering, I. *Appl. Comput. Harmon. Anal.*, 1(1):94–99, dec 1993.

[14] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A multigrid tutorial*. Society for Industrial and Applied Mathematics, Philadelphia, 2nd edition, 2000.

[15] E. J. Candès, L. Demanet, D. L. Donoho, and L. Ying. Fast discrete curvelet transforms. *Multiscale Model. Simul.*, 5(3):861–899, jan 2006.

[16] E. J. Candès, L. Demanet, and L. Ying. Fast computation of Fourier integral operators. *SIAM J. Sci. Comput.*, 29(6):2464–2493, jan 2007.

[17] E. J. Candès, L. Demanet, and L. Ying. A fast butterfly algorithm for the computation of Fourier integral operators. *Multiscale Model. Simul.*, 7(4):1727–1750, jan 2009.

[18] E. J. Candès and D. L. Donoho. New tight frames of curvelets and optimal representations of objects with piecewise C^2 singularities. *Commun. Pure Appl. Math.*, 57(2):219–266, feb 2004.

[19] E. J. Candès and D. L. Donoho. Continuous curvelet transform I. resolution of the wavefront set. *Appl. Comput. Harmon. Anal.*, 19(2):162–197, sep 2005.

[20] E. J. Candès and D. L. Donoho. Continuous curvelet transform II. discretization and frames. *Appl. Comput. Harmon. Anal.*, 19(2):198–222, sep 2005.

[21] S. Chandrasekaran, P. Dewilde, M. Gu, T. Pals, X. Sun, A.-J. van der Veen, and D. White. Some fast algorithms for sequentially semiseparable representations. *SIAM J. Matrix Anal. Appl.*, 27(2):341–364, 2005.

[22] S. Chandrasekaran, P. Dewilde, M. Gu, and N. Somasunderam. On the numerical rank of the off-diagonal blocks of Schur complements of discretized elliptic PDEs. *SIAM J. Matrix Anal. Appl.*, 31(5):2261–2290, 2010.

[23] H. Cheng, Z. Gimbutas, P.-G. Martinsson, and V. Rokhlin. On the compression of low rank matrices. *SIAM J. Sci. Comput.*, 26(4):1389–1404, 2005.

[24] E. Chow, R. D. Falgout, J. J. Hu, R. S. Tuminaro, and U. M. Yang. A survey of parallelization techniques for multigrid solvers. In *Parallel Process. Sci. Comput.*,

chapter 10, pages 179–201. Society for Industrial and Applied Mathematics, jan 2006.

[25] E. Cordero, F. Nicola, and L. Rodino. Sparsity of Gabor representation of Schrödinger propagators. *Appl. Comput. Harmon. Anal.*, 26(3):357–370, may 2009.

[26] T. A. Davis. *Direct methods for sparse linear systems*, volume 2 of *Fundamentals of Algorithms*. Society for Industrial and Applied Mathematics, 2006.

[27] M. V. de Hoop, G. Uhlmann, A. Vasy, and H. Wendt. Multiscale discrete approximations of Fourier integral operators associated with canonical transformations and caustics. *Multiscale Model. Simul.*, 11(2):566–585, jan 2013.

[28] L. Demanet and L. Ying. Wave atoms and sparsity of oscillatory patterns. *Appl. Comput. Harmon. Anal.*, 23(3):368–387, nov 2007.

[29] L. Demanet and L. Ying. Scattering in flatland: efficient representations via wave atoms. *Found. Comput. Math.*, 10(5):569–613, oct 2010.

[30] L. Demanet and L. Ying. Discrete symbol calculus. *SIAM Rev.*, 53(1):71–104, jan 2011.

[31] L. Demanet and L. Ying. Fast wave computation via Fourier integral operators. *Math. Comput.*, 81(279):1455–1486, sep 2012.

[32] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. H. Liu. A supernodal approach to sparse partial pivoting. *SIAM J. Matrix Anal. Appl.*, 20(3):720–755, 1999.

[33] I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Trans. Math. Softw.*, 9(3):302–325, sep 1983.

[34] A. Dutt and V. Rokhlin. Fast Fourier transforms for nonequispaced data. *SIAM J. Sci. Comput.*, 14(6):1368–1393, nov 1993.

[35] M. S. Elias. *Harmonic analysis: real-variable methods, orthogonality, and oscillatory integrals*. 43. Princeton University Press, Princeton, 1993.

[36] B. Engquist and L. Ying. A fast directional algorithm for high frequency acoustic scattering in two dimensions. *Commun. Math. Sci.*, 7(2):327–345, 2009.

[37] R. D. Falgout and J. E. Jones. Multigrid on massively parallel architectures. In *Multigrid Methods VI*, pages 101–107. Springer, Berlin, 2000.

[38] A. George. Nested dissection of a regular finite element mesh. *SIAM J. Numer. Anal.*, 10(2):345–363, 1973.

[39] A. Gillman and P.-G. Martinsson. A direct solver with $O(N)$ complexity for variable coefficient elliptic PDEs discretized via a high-order composite spectral collocation method. *SIAM J. Sci. Comput.*, 36(4):A2023–A2046, jan 2014.

[40] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, 4th edition, 2013.

[41] L. Greengard and J.-Y. Lee. Accelerating the nonuniform fast Fourier transform. *SIAM Rev.*, 46(3):443–454, jan 2004.

[42] L. Grigori, J. W. Demmel, and X. S. Li. Parallel symbolic factorization for sparse LU with static pivoting. *SIAM J. Sci. Comput.*, 29(3):1289–1314, jan 2007.

[43] W. Hackbusch. A sparse matrix arithmetic based on $\mathcal{H}$-matrices. I. Introduction to $\mathcal{H}$-matrices. *Computing*, 62(2):89–108, 1999.

[44] W. Hackbusch and S. Börm. Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices. *Computing*, 69(1):1–35, 2002.

[45] W. Hackbusch and B. N. Khoromskij. A sparse $\mathcal{H}$-matrix arithmetic. Part II. Application to multi-dimensional problems. *Computing*, 64(1):21–47, 2000.

[46] N. Halko, P.-G. Martinsson, and J. A. Tropp. Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Rev.*, 53(2):217–288, jan 2011.

[47] S. Hao and P.-G. Martinsson. A direct solver for elliptic PDEs in three dimensions based on hierarchical merging of Poincaré-Steklov operators. *J. Comput. Appl. Math.*, 308:419–434, 2016.

[48] M. T. Heath. Parallel direct methods for sparse linear systems. In *Parallel Numer. algorithms*, volume 4 of *ICASE/LaRC Interdiscip. Ser. Sci. Eng.*, pages 55–90. Kluwer Academic Publishers, 1997.

[49] K. L. Ho and L. Greengard. A fast semidirect least squares algorithm for hierarchically block separable matrices. *SIAM J. Matrix Anal. Appl.*, 35(2):725–748, jan 2014.

[50] K. L. Ho and L. Ying. Hierarchical interpolative factorization for elliptic operators: differential equations. *Commun. Pure Appl. Math.*, 69(8):1415–1451, 2016.

[51] K. L. Ho and L. Ying. Hierarchical interpolative factorization for elliptic operators: integral equations. *Commun. Pure Appl. Math.*, 69(7):1314–1353, jul 2016.

[52] L. Hörmander. Fourier integral operators. I. *Acta Math.*, 127(0):79–183, 1971.

[53] J. Hu, S. Fomel, L. Demanet, and L. Ying. A fast butterfly algorithm for generalized Radon transforms. *Geophysics*, 78(4):U41–U51, 2013.

[54] D. Huybrechs and S. Vandewalle. A two-dimensional wavelet-packet transform for matrix compression of integral equations with highly oscillatory kernel. *J. Comput. Appl. Math.*, 197(1):218–232, 2006.

[55] M. Izadi. Parallel $\mathcal{H}$-matrix arithmetic on distributed-memory systems. *Comput. Vis. Sci.*, 15(2):87–97, apr 2012.

[56] R. E. Kleinman and G. F. Roach. Boundary integral equations for the three-dimensional Helmholtz equation. *SIAM Rev.*, 16(2):214–236, apr 1974.

[57] R. Kriemann. $\mathcal{H}$-LU factorization on many-core systems. *Comput. Vis. Sci.*, 16(3):105–117, jun 2013.

[58] Y. Li and H. Yang. Interpolative butterfly factorization. *SIAM J. Sci. Comput.*, 39(2):A503–A531, 2017.

[59] Y. Li, H. Yang, E. R. Martin, K. L. Ho, and L. Ying. Butterfly factorization. *Multiscale Model. Simul.*, 13(2):714–732, jan 2015.

[60] Y. Li, H. Yang, and L. Ying. A multiscale butterfly algorithm for multidimensional Fourier integral operators. *Multiscale Model. Simul.*, 13(2):1–18, jan 2015.

[61] Y. Li, H. Yang, and L. Ying. Multidimensional butterfly factorization. *Applied and Computational Harmonic Analysis*, 2017.

[62] Y. Li and L. Ying. Distributed-memory hierarchical interpolative factorization. Technical report, 2016.

[63] E. Liberty, F. Woolfe, P.-G. Martinsson, V. Rokhlin, and M. Tygert. Randomized algorithms for the low-rank approximation of matrices. *Proc. Natl. Acad. Sci. U. S. A.*, 104(51):20167–72, dec 2007.

[64] L. Lin, J. Lu, and L. Ying. Fast construction of hierarchical matrix representation from matrix-vector multiplication. *J. Comput. Phys.*, 230(10):4071–4087, 2011.

[65] J. W. H. Liu. The role of elimination trees in sparse factorization. *SIAM J. Matrix Anal. Appl.*, 11(1):134–172, 1990.

[66] J. W. H. Liu. The multifrontal method for sparse matrix solution: theory and practice. *SIAM Rev.*, 34(1):82–109, mar 1992.

[67] X. Liu, J. Xia, and M. V. de Hoop. Parallel randomized and matrix-free direct solvers for large structured dense linear systems. *SIAM J. Sci. Comput.*, 38(5):1–32, jan 2016.

[68] M. W. Mahoney and P. Drineas. CUR matrix decompositions for improved data analysis. *Proc. Natl. Acad. Sci. U. S. A.*, 106(3):697–702, jan 2009.

[69] P.-G. Martinsson. A fast direct solver for a class of elliptic partial differential equations. *SIAM J. Sci. Comput.*, 38(3):316–330, 2009.

[70] P.-G. Martinsson. A fast randomized algorithm for computing a hierarchically semiseparable representation of a matrix. *SIAM J. Matrix Anal. Appl.*, 32(4):1251–1274, oct 2011.

[71] P.-G. Martinsson. Blocked rank-revealing QR factorizations: How randomized sampling can be used to avoid single-vector pivoting. Technical report, may 2015.

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

[73] V. Minden, A. Damle, K. L. Ho, and L. Ying. A technique for updating hierarchical skeletonization-based factorizations of integral operators. *Multiscale Model. Simul.*, 14(1):42–64, jan 2016.

[74] M. O'Neil. *A new class of analysis-based fast transforms.* PhD thesis, Yale University, New Haven, 2008.

[75] M. O'Neil, F. Woolfe, and V. Rokhlin. An algorithm for the rapid evaluation of special function transforms. *Appl. Comput. Harmon. Anal.*, 28(2):203–226, 2010.

[76] J. Poulson, L. Demanet, N. Maxwell, and L. Ying. A parallel butterfly algorithm. *SIAM J. Sci. Comput.*, 36(1):C49–C65, feb 2014.

[77] J. Poulson, B. Engquist, S. Li, and L. Ying. A parallel sweeping preconditioner for heterogeneous 3D Helmholtz equations. *SIAM J. Sci. Comput.*, 35(3):C194–C212, 2013.

[78] J. Poulson, B. Marker, R. A. van de Geijn, J. R. Hammond, and N. A. Romero. Elemental: A new framework for distributed memory dense matrix computations. *ACM Trans. Math. Softw.*, 39(2):13:1–13:24, feb 2013.

[79] Y. Saad. Parallel iterative methods for sparse linear systems. In *Stud. Comput. Math.*, volume 8, pages 423–440. 2001.

[80] Y. Saad. *Iterative methods for sparse linear systems*, volume 8 of *Stud. Comput. Math.* Society for Industrial and Applied Mathematics, second edition, 2003.

[81] P. G. Schmitz and L. Ying. A fast direct solver for elliptic problems on general meshes in 2D. *J. Comput. Phys.*, 231(4):1314–1338, 2012.

[82] P. G. Schmitz and L. Ying. A fast nested dissection solver for Cartesian 3D elliptic problems using hierarchical matrices. *J. Comput. Phys.*, 258:227–245, 2014.

[83] D. S. Scott. Efficient all-to-all communication patterns in hypercube and mesh topologies. In *Sixth Distrib. Mem. Comput. Conf.*, pages 398–403. IEEE, 1991.

[84] D. S. Seljebotn. Wavemoth-fast spherical harmonic transforms by butterfly matrix compression. *Astrophys. J. Suppl. Ser.*, 199(1):5, mar 2012.

[85] D. O. Trad, T. J. Ulrych, and M. D. Sacchi. Accurate interpolation with high-resolution time-variant Radon transforms. *GEOPHYSICS*, 67(2):644–656, mar 2002.

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

[87] R. Wang, Y. Li, M. W. Mahoney, and E. Darve. Structured block basis factorization for scalable kernel matrix evaluation. Technical report, 2015.

[88] S. Wang, X. S. Li, F.-H. Rouet, J. Xia, and M. V. de Hoop. A parallel geometric multifrontal solver using hierarchically semiseparable structure. *ACM Trans. Math. Softw.*, 42(3):21:1–21:21, may 2016.

[89] F. Woolfe, E. Liberty, V. Rokhlin, and M. Tygert. A fast randomized algorithm for the approximation of matrices. *Appl. Comput. Harmon. Anal.*, 25(3):335–366, nov 2008.

[90] J. Xia. Efficient structured multifrontal factorization for general large sparse matrices. *SIAM J. Sci. Comput.*, 35(2):A832–A860, 2013.

[91] J. Xia. Randomized sparse direct solvers. *SIAM J. Matrix Anal. Appl.*, 34(1):197–227, 2013.

[92] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li. Superfast multifrontal method for large structured linear systems of equations. *SIAM J. Matrix Anal. Appl.*, 31(3):1382–1411, 2009.

[93] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li. Fast algorithms for hierarchically semiseparable matrices. *Numer. Linear Algebr. with Appl.*, 17(6):953–976, dec 2010.

[94] Z. Xin, J. Xia, M. V. de Hoop, S. Cauley, and V. Balakrishnan. A distributed-memory randomized structured multifrontal method for sparse direct solutions. Technical Report 17, 2014.

[95] H. Yang and L. Ying. A fast algorithm for multilinear operators. *Appl. Comput. Harmon. Anal.*, 33(1):148–158, jul 2012.

[96] B. Yazici, L. Wangy, and K. Duman. Synthetic aperture inversion with sparsity constraints. In *2011 Int. Conf. Electromagn. Adv. Appl.*, pages 1404–1407. IEEE, sep 2011.

[97] L. Ying. Sparse Fourier transform via butterfly algorithm. *SIAM J. Sci. Comput.*, 31(3):1678–1694, jan 2009.

[98] L. Ying, G. Biros, and D. Zorin. A kernel-independent adaptive fast multipole algorithm in two and three dimensions. *J. Comput. Phys.*, 196(2):591–626, 2004.