ACCURACY EXPLICITLY CONTROLLED $\mathcal{H}^2$-MATRIX ARITHMETIC IN LINEAR COMPLEXITY AND FAST DIRECT SOLUTIONS FOR LARGE-SCALE ELECTROMAGNETIC ANALYSIS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Miaomiao Ma

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2019

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Dr. Dan Jiao, Chair

School of Electrical and Computer Engineering

Dr. Weng Cho Chew

School of Electrical and Computer Engineering

Dr. Steven D. Pekarek

School of Electrical and Computer Engineering

Dr. Jianlin Xia

Department of Mathematics

Approved by:

Dr. Dimitrios Peroulis

Head of the School Graduate Program

To my father

ACKNOWLEDGMENTS

Foremost, I would like to express my deepest gratitude to my advisor Professor Dan Jiao for giving me this opportunity to work with her at Purdue University. This work can never be completed without her sharp insights, constant inspiration and kind encouragement. Professor Jiao is always there and willing to help when I feel confused or get stuck with research. And she is able to see the whole picture and guide me through the research mist with her solid numerical and physical visions. As she knows almost every detail about my research project, she could provide very useful suggestions to further push the research limits. When the numerical experiments could not agree with physical understanding, she will not give up until we figure out the reasons. Knowing that she is always there for me is really a strong dose of courage to help me keep calm and carry on in my research career. I feel so lucky and gratitude to have Professor Jiao as my advisor!

I would like to thank other members of my PhD advisory committee: Professor Weng Cho Chew, Professor Steven D. Pekarek and Professor Jianlin Xia for their precious time, support and suggestions regarding my research work.

Thanks are also extended to all the fellow labmates who worked with me in On-Chip Electromagnetics Lab here at Purdue: Dr. Bangda Zhou, Dr. Woochan Lee, Dr. Md. Gaffar, Dr. Jin Yan, Dr. Ping Li, Dr. Yanpu Zhao, Dr. Kaiyuan Zeng, Dr. Yu Zhao, Li Xue, Chang Yang, Shuzhan Sun, Yifan Wang, Michael Hayashi and Zhangchao Wei for their constant support and the friendly yet professional environment in the lab.

Last but not the least, I am thoroughly indebted to my family for their everlasting love and support during all these years. I dedicate this work to all of you!

TABLE OF CONTENTS

Page

Page

Page

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

ABSTRACT

Ma, Miaomiao Ph.D., Purdue University, December 2019. Accuracy Explicitly Controlled $\mathcal{H}^2$-Matrix Arithmetic in Linear Complexity and Fast Direct Solutions for Large-Scale Electromagnetic Analysis. Major Professor: Dan Jiao.

The design of advanced engineering systems generally results in large-scale numerical problems, which require efficient computational electromagnetic (CEM) solutions. Among existing CEM methods, iterative methods have been a popular choice since conventional direct solutions are computationally expensive. The optimal complexity of an iterative solver is $O(NN_{it}N_{rhs})$ with N being matrix size, N_{it} the number of iterations and N_{rhs} the number of right hand sides. How to invert or factorize a dense matrix or a sparse matrix of size N in $O(N)$ (optimal) complexity with explicitly controlled accuracy has been a challenging research problem. For solving a dense matrix of size N, the computational complexity of a conventional direct solution is $O(N^3)$; for solving a general sparse matrix arising from a 3-D EM analysis, the best computational complexity of a conventional direct solution is $O(N^2)$. Recently, an $\mathcal{H}^2$-matrix based mathematical framework has been developed to obtain fast dense matrix algebra. However, existing linear-complexity $\mathcal{H}^2$-based matrix-matrix multiplication and matrix inversion lack an explicit accuracy control. If the accuracy is to be controlled, the inverse as well as the matrix-matrix multiplication algorithm must be completely changed, as the original formatted framework does not offer a mechanism to control the accuracy without increasing complexity.

In this work, we develop a series of new accuracy controlled fast $\mathcal{H}^2$ arithmetic, including matrix-matrix multiplication (MMP) without formatted multiplications, minimal-rank MMP, new accuracy controlled $\mathcal{H}^2$ factorization and inversion, new accuracy controlled $\mathcal{H}^2$ factorization and inversion with concurrent change of cluster

bases, $\mathcal{H}^2$-based direct sparse solver and new HSS recursive inverse with directly controlled accuracy. For constant-rank $\mathcal{H}^2$-matrices, the proposed accuracy directly controlled $\mathcal{H}^2$ arithmetic has a strict $O(N)$ complexity in both time and memory. For rank that linearly grows with the electrical size, the complexity of the proposed $\mathcal{H}^2$ arithmetic is $O(NlogN)$ in factorization and inversion time, and $O(N)$ in solution time and memory for solving volume IEs. Applications to large-scale interconnect extraction as well as large-scale scattering analysis, and comparisons with state-of-the-art solvers have demonstrated the clear advantages of the proposed new $\mathcal{H}^2$ arithmetic and resulting fast direct solutions with explicitly controlled accuracy. In addition to electromagnetic analysis, the new $\mathcal{H}^2$ arithmetic developed in this work can also be applied to other disciplines, where fast and large-scale numerical solutions are being pursued.

1. INTRODUCTION

1.1 Challenges

The design of advanced engineering systems generally results in large-scale numerical problems, which require efficient computational electromagnetic (CEM) solutions. The CEM solvers generate either dense or sparse matrix systems. How to invert or factorize a dense matrix or a sparse matrix of size N in $O(N)$ (optimal) complexity with explicitly controlled accuracy has been a challenging research problem. The $\mathcal{H}^2$ matrix is a general mathematical framework [1–4] for compact representation and efficient computation of dense matrices. It can be utilized to develop fast solvers for electromagnetic analysis. Many existing fast solvers can be interpreted in this framework, although their developments may predate the $\mathcal{H}^2$-framework. For example, the matrix structure resulting from the well-known Fast Multipole Method (FMM)-based methods [5, 6] is an $\mathcal{H}^2$ matrix. In other words, the $\mathcal{H}^2$ matrix can be viewed as an algebraic generalization of the FMM method. Multiplying an $\mathcal{H}^2$ matrix by a vector has a complexity of $O(N\log N)$ for solving electrically large surface integral equations (SIEs). In mathematical literature like [7], it is shown that the storage, matrix-vector multiplication (MVM), and matrix-matrix multiplication of an $\mathcal{H}^2$ matrix can be performed in optimal $O(N)$ complexity for constant-rank cases. However, no such complexity is shown for either matrix factorization or inversion, regardless of whether the rank is a bounded constant or an increasing variable. Later in [8, 9], fast matrix inversion and LU factorization algorithms are developed for $\mathcal{H}^2$ matrices. They have shown a complexity of $O(N)$ for solving constant-rank cases like electrically small and moderate problems for both surface and volume IEs (VIEs) [10–13]; and a complexity of $O(N\log N)$ for solving electrically large VIEs [14–16]. There exist other significant contributions in fast direct solvers such as [17–27]. Recently, an HSS-

matrix structure [28] has also been explored for electromagnetic analysis [29,30]. The HSS matrix, as a special class of $\mathcal{H}^2$ matrix, requires only one admissible block be formed for each node in the $\mathcal{H}^2$ tree.

Despite a significantly reduced complexity, in existing direct solutions of $\mathcal{H}^2$ matrices like [8,9], formatted multiplications and additions are performed instead of actual ones, where the cluster bases used to represent a matrix are also used to represent the matrix's inverse as well as LU factors. During the inversion and factorization procedure, only the coupling matrix of each admissible block is computed, while the cluster bases are kept to be the same as those in the original matrix. Physically speaking, such a choice of the cluster bases for the inverse matrix often constitutes an accurate choice. However, mathematically speaking, the accuracy of the inversion and factorization cannot be directly controlled. The lack of an accuracy control is also observed in the $\mathcal{H}^2$-based formatted matrix-matrix multiplication in mathematical literature such as [7]. If the accuracy is to be controlled, the inverse as well as the matrix-matrix multiplication algorithm must be completely changed, as the original formatted framework does not offer a mechanism to control the accuracy without increasing complexity. Algorithm wise, the collect operation involves approximations, whose accuracy is not directly controlled. This operation is performed when multiplying a non-leaf block with a non-leaf block, or multiplying a nonleaf block with an admissible block, with the target block being admissible. In this operation, the four blocks, either admissible or inadmissible, are collected to a single admissible block based on the cluster bases of the original matrix. If the target block cannot be accurately represented by the original cluster bases, then an error would occur. When the accuracy of the direct solution is not satisfactory, one can only change the original $\mathcal{H}^2$-representation, i.e., change the cluster bases and/or the rank for representing the original matrix (based on a prescribed accuracy), with the hope that they can better represent the inverse and LU factors. Therefore, the accuracy of the resulting direct solution is not directly controlled. A direct solution with an explicit accuracy control should perform every multiplication and addition as it is without assuming a format

of the matrix involved in the computation. Meanwhile, every operation in the direct solution should be either exact or performed based on a prescribed accuracy. This is what is pursued and achieved in this research work. The computer used for numerical results sections in this thesis has an Intel(R) Xeon(R) CPU E5-2690 v2 running at 3.00GHz, and only a *single core* is employed for carrying out all the computations.

1.2 Computational Electromagnetic Methods

Computational electromagnetics is the science of solving Maxwell's equations to analyze the interaction between electromagnetic fields and physical objects and the environment. The CEM methods can be categorized into two classes, the partial differencial equation (PDE) based methods and integral equation (IE) based ones. The IE-based methods have been a popular choice since they avoid the use of any artificial absorbing boundary condition (ABC) and they can handle open-region problems efficiently. There are two major classes of IE-based solvers, surface IE (SIE)-based solvers and Volume IE (VIE)-based ones. SIE-based solvers are very efficient when calculating homogeneous problems since they only need to discretize the surface. Compared to SIE-based solvers, VIE-based solvers have a great flexibility in handling complicated geometry and inhomogeneous materials. However, both SIE and VIE generally lead to dense linear systems of equations. When an iterative solver is used, the operation count is proportional to $O(N_{it}N_{rhs}N^2)$, where N_{rhs} is the number of right-hand sides, N_{it} is the number of iterations and N is the matrix size. When a direct solver is used, the computational cost would be $O(N^3)$. It is computationally expensive for solving electrically large problems. Therefore, there is a continued need to reduced the complexity of IE solvers.

1.2.1 Integral Equations

Capacitance Extraction Formulation

For capacitance extraction of interconnects in a single material, we consider charges on the conducting surfaces producing electric potential, which results in the following integral equation

$$\phi(\vec{r}) = \int_S \frac{\rho_s(\vec{r'})}{\epsilon} g(\vec{r}, \vec{r'}) ds', \tag{1.1}$$

where ρ_s is surface charge density, ϵ is permittivity, and $g(\vec{r}, \vec{r'})$ is free-space static Green's function. A method-of-moments based solution of (1.1) results in the following dense system of equations

$$\mathbf{Z}q = v, \tag{1.2}$$

where q vector consists of charges on each discretization patch, and right hand side vector v is composed of the potential assigned to each conductor. In non-uniform dielectrics, (1.1) is augmented by the electric potential due to equivalent surface charges at the dielectric discontinuity.

Volume Integral Equation Formulation

For a full-wave analysis of interconnects, the most general IE formulation would be a volume integral equation based formulation, which accounts for arbitrary dielectric and conductive inhomogeneity. Consider an interconnect exposed to an external field $\vec{E}^i(\vec{r})$, based on the volume equivalence principle, the equivalent volume current $\vec{J} = j\omega(\bar{\epsilon} - \epsilon_0)\vec{E}$ radiating in the background material produces the scattered field. Thus, the total field $\vec{E}$ at any point $\vec{r}$ is equal to the sum of the incident field and the scattered field

$$\vec{E}(\vec{r}) + \nabla\phi^s(\vec{r}) + j\omega A^s(\vec{r}) = \vec{E}^i(\vec{r}), \tag{1.3}$$

which is expressed in the following form of the volume integral equation [31],

$$\frac{\vec{D}(\vec{r})}{\bar{\epsilon}(\vec{r})} - \mu\omega^2 \int_V \kappa(\vec{r'})\vec{D}(\vec{r'})g(\vec{r},\vec{r'})dv' - \nabla \int_V \nabla' \cdot \left(\frac{\kappa(\vec{r'})\vec{D}(\vec{r'})}{\epsilon_0} \right) g(\vec{r},\vec{r'})dv' = \vec{E}^i(\vec{r}), \tag{1.4}$$

where Green's function $g(\vec{r},\vec{r'}) = e^{-jk_0|\vec{r}-\vec{r'}|}/4\pi|\vec{r}-\vec{r'}|$, ω being the angular frequency, k_0 is the free space wave number, κ is the contrast ratio defined as

$$\kappa(\vec{r}) = \frac{\bar{\epsilon}(\vec{r}) - \epsilon_0}{\bar{\epsilon}(\vec{r})}, \tag{1.5}$$

and $\vec{D}(\vec{r})$ is

$$\vec{D}(\vec{r}) = \bar{\epsilon}(r)\vec{E}, \tag{1.6}$$

which is related to the equivalent volume current by $\vec{J}(\vec{r}) = j\omega\kappa(\vec{r})\vec{D}(\vec{r})$. From the third term in (1.4), it can be seen that

$$\phi^s(\vec{r}) = -\int_V \nabla' \cdot \left(\frac{\kappa(\vec{r'})\vec{D}(\vec{r'})}{\epsilon_0} \right) g(\vec{r},\vec{r'})dv', \tag{1.7}$$

which can further be written as

$$\phi^s(\vec{r}) = \int_V \frac{\rho_v(\vec{r'})}{\epsilon_0} g(\vec{r},\vec{r'})dv' + \int_S \frac{\rho_s(\vec{r'})}{\epsilon_0} g(\vec{r},\vec{r'})ds', \tag{1.8}$$

where ρ_v is the density of equivalent volume charges, and ρ_s is the density of the equivalent surface charges at the material discontinuity where $\kappa(\vec{r})$ is discontinuous. In this paper, the vector basis functions employed to expand $\vec{D}$ are the Schaubert-Wilton-Glisson (SWG) bases [31].

1.2.2 Partial Difference Equations

Finite Element Methods

Considering a general physical layout of a package or integrated circuit involving inhomogeneous materials and arbitrarily shaped lossy conductors, the electric field $\mathbf{E}$ satisfies the following second-order vector wave equation

$$\nabla \times (\frac{1}{\mu_r} \times \mathbf{E}) + jk_0\eta_0\sigma\mathbf{E} - k_0^2\varepsilon_r\mathbf{E} = -jk_0Z_0\mathbf{j}, \tag{1.9}$$

where μ_r is relative permeability, ε_r is relative permittivity, σ is conductivity, k_0 is free-space wave number, Z_0 is free-space wave impedance, and $\mathbf{j}$ is current density. A finite element based solution of (1.9) subject to pertinent boundary conditions results in the following linear system of equations

$$YX = B, \tag{1.10}$$

where $Y \in \mathbb{C}^{N\times N}$ is a sparse matrix, and matrix B is composed of one or multiple right hand side vectors. Sparse matrix Y can be expended as,

$$Y = -k_0^2 T + jk_0 G + S \tag{1.11}$$

where T, G, and S are assembled from their elemental contributions as the following,

$$\begin{aligned} T_{ij} &= \iiint_v \varepsilon_r \mathbf{n}_i \cdot \mathbf{n}_j \, \mathrm{d}v \\ S_{ij} &= \iiint_v \frac{1}{\mu_r} (\nabla \times \mathbf{n}_i) \cdot (\nabla \times \mathbf{n}_j) \, \mathrm{d}v \\ G_{ij} &= \iiint_v \eta_0 \sigma \mathbf{n}_i \cdot \mathbf{n}_j \, \mathrm{d}v + \iint_s (\hat{n} \times \mathbf{n}_i) \cdot (\hat{n} \times \mathbf{n}_j) \, \mathrm{d}s, \end{aligned} \tag{1.12}$$

where v denotes the volume of each element, s is the boundary, $\hat{n}$ is the unit normal vector and $\mathbf{n}_i$ is the vector basis function used to expand unknown $\mathbf{e}$ in each element. When the size of (1.10) is large, its efficient solution relies on fast and large-scale matrix solutions.

1.3 Mathematical Background

Rank-structured matrices, such as $\mathcal{H}$- (hierarchical), $\mathcal{H}^2$-, and HSS (Hierarchically Semiseparable) matrices [1-2], are one important class of matrices whose structures can be utilized to develop fast solvers for analyzing electromagnetic problems [3-4].

1.3.1 $\mathcal{H}$ Matrix

An $\mathcal{H}$ matrix [3, 4] is generally stored in a tree structure, with each node in the tree called a cluster. The number of unknowns in each cluster at the leaf level is no greater than *leafsize*, a predefined constant. An $\mathcal{H}$ matrix is partitioned into multilevel admissible and inadmissible blocks based on an admissibility condition.

$$\max\{\text{diam}(\Omega_t), \text{diam}(\Omega_s)\} \leq \eta \times \text{dist}(\Omega_t, \Omega_s), \tag{1.13}$$

where Ω_t (Ω_s) denotes the geometrical support of cluster t (s), $diam\{\cdot\}$ is the Euclidean diameter of a cluster, $dist\{\cdot,\cdot\}$ denotes the Euclidean distance between two clusters, and η is a positive parameter that can be used to control the admissibility condition. As an example, a four-level block cluster $\mathcal{H}$-tree is illustrated in Fig. 1.1,

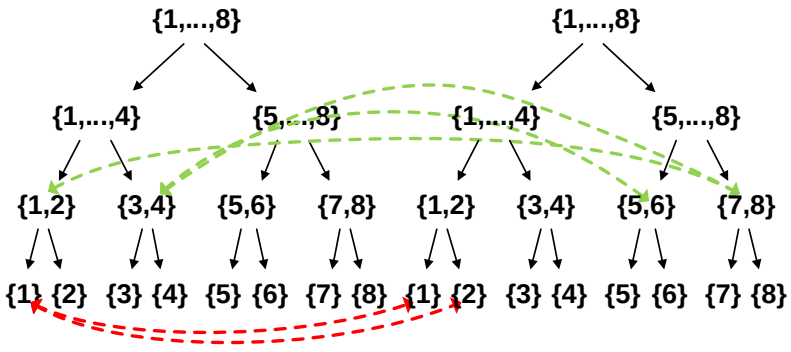


Fig. 1.1.: Illustration of a block cluster tree.

where a green link denotes an admissible block, formed between a row cluster and a column cluster that satisfy the admissibility condition; and a red link denotes an inadmissible block. The block cluster tree structure combined with data-sparse storage of the matrix blocks in admissible blocks complete the $\mathcal{H}$-matrix representation of a matrix. The storage requirement of such $\mathcal{H}$-matrix representation is shown to be $O(N \log N)$ through theoretical complexity analysis. After the $\mathcal{H}$-matrix representation is established, basic $\mathcal{H}$-matrix arithmetics are defined on it. In an $\mathcal{H}$ matrix,

an inadmissible block keeps its original full matrix form, while an admissible block is represented by a low-rank matrix shown as the following:

$$\tilde{\mathbf{Y}}^{t\times s} = \mathbf{A}_{\#t\times k} \cdot \mathbf{B}^{T}_{\#s\times k}, \tag{1.14}$$

where k is the rank, and # denotes the cardinality of a set. The error of a rank-k approximation can be evaluated as

$$\|\mathbf{Y}^{t\times s} - \tilde{\mathbf{Y}}^{t\times s}\|_2 = \sigma_{k+1}, \tag{1.15}$$

in which σ_{k+1} is the maximum singular value among truncated singular values. By applying (1.15), the error of an $\mathcal{H}$-matrix representation can be quantitatively controlled. With a rank-k representation, an $\mathcal{H}$-matrix significantly accelerates matrix-related operations and reduces storage cost for a prescribed accuracy as compared to a full-matrix based representation. The resultant $\mathcal{H}$ matrix is as shown in Fig. 1.2.

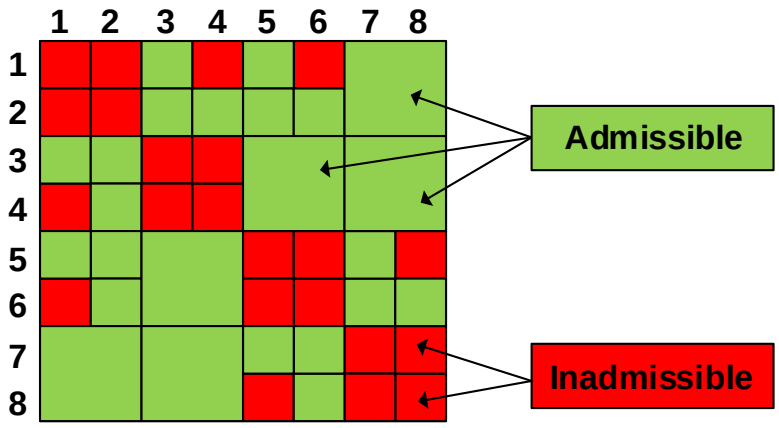


Fig. 1.2.: Matrix partition where admissible blocks are stored in the format of (1.14).

1.3.2 $\mathcal{H}^2$ Matrix

The $\mathcal{H}^2$ matrix [1–4] is a special class of $\mathcal{H}$ matrix. The only difference is that the admissible blocks are stored using nested representation. An admissible block in an $\mathcal{H}^2$ matrix can be represented as

$$\mathbf{Z}_{t,s} = (\mathbf{V}_t)_{\#t\times k}(\mathbf{S}_{t,s})_{k\times k}(\mathbf{V}_s)^T_{\#s\times k}, \tag{1.16}$$

where $\mathbf{V}_t$ ($\mathbf{V}_s$) is called cluster basis of cluster t (s), $\mathbf{S}_{t,s}$ is called coupling matrix, # denotes the cardinality of a set, and k is rank. The cluster basis in an $\mathcal{H}^2$ matrix has a nested property. This means the cluster basis for a non-leaf cluster t, $\mathbf{V}_t$, can be expressed by its two children's cluster bases, $\mathbf{V}_{t_1}$ and $\mathbf{V}_{t_2}$, as the following

$$(\mathbf{V}_t)_{\#t\times k} = \begin{bmatrix} (\mathbf{V}_{t_1})_{\#t_1\times k_1} & 0 \\ 0 & (\mathbf{V}_{t_2})_{\#t_2\times k_2} \end{bmatrix} \begin{bmatrix} (\mathbf{T}_{t_1})_{k_1\times k} \\ (\mathbf{T}_{t_2})_{k_2\times k} \end{bmatrix} \tag{1.17}$$

where $\mathbf{T}_{t_1}$ and $\mathbf{T}_{t_2}$ are called transfer matrices. In an $\mathcal{H}^2$ matrix, the number of blocks formed by a single cluster at each tree level is bounded by a constant, C_{sp}. The size of an inadmissible block is *leafsize*, and inadmissible blocks appear only at the leaf level. Because of the nested relationship of cluster bases, the cluster bases only need to be stored for leaf clusters. For non-leaf clusters, only transfer matrices need to be stored. So an $\mathcal{H}^2$ matrix is more compact and computational efficient due to its nested property as compared to an $\mathcal{H}$ matrix.

In mathematical literature [7], for constant-rank cases, it is shown that the computational complexity of an $\mathcal{H}^2$ matrix is optimal (linear) in storage, matrix-vector multiplication, and matrix-matrix multiplication. In [8–13], it is shown that $O(N)$ direct solutions can also be developed for $\mathcal{H}^2$ matrices for constant-rank cases as well as electrically moderate cases whose rank can be controlled by a rank function. For electrically large analysis, the rank of an $\mathcal{H}^2$-representation of IE kernels is not constant any more for achieving a prescribed accuracy. Different $\mathcal{H}^2$-representations also result in different rank and their growth rates with electrical size, and hence different complexities. For example, if an FMM-based $\mathcal{H}^2$-representation is used to model an

IE operator, the asymptotic rank would scale *quadratically* with the electrical size. If an interpolation based $\mathcal{H}^2$-representation is used to model an IE operator, the asymptotic rank would scale *quadratically* with the electrical size in surface IEs (SIEs), and *cubically* with the electrical size in volume IEs. Despite its full-rank model in SIEs, the FMM complexity is as low as $O(N\log N)$ for one matrix-vector multiplication, because its coupling matrix is diagonal and transfer matrix is sparse. On the other hand, if one uses an SVD-based minimal-rank representation, the resultant coupling matrix is not diagonal. However, the asymptotic rank of such a minimal-rank representation only scales linearly with electrical size for general 3-D problems as shown by the analysis given in [32]. In this analysis, there are additional findings that are not utilized in the FMM-based rank analysis or a source-observer separated rank analysis, such as SVD turns to a Fourier analysis in a linear-shift invariant system, and the Fourier transform of Green's function reveals that not all the plane waves are important, and only those whose wave number is closest to the given wave number k_0 make the most important contribution. These plane waves are counted for prescribed accuracy for 1-, 2-, and 3-D problems in [32]. It is found that the growth rate with electrical size is no greater than linear.

1.3.3 HSS Matrix

The HSS matrix can be viewed as a special class of $\mathcal{H}^2$ matrix. An HSS matrix only permits one admissible block formed for each node in the $\mathcal{H}^2$-tree. Instead of using admissibility condition in (1.13) to note admissible blocks in $\mathcal{H}$ and $\mathcal{H}^2$ matrices, all the off-diagonal blocks are admissible in HSS matrix. As shown in Fig. 1.3, all the off-diagonal blocks have nested low-rank representation $(\mathbf{V}_t)_{\#t\times k}(\mathbf{S}_{t,s})_{k\times k}(\mathbf{V}_s)^T_{\#s\times k}$.

1.4 Contributions of This Work

In this work, we develop a series of new accuracy controlled fast $\mathcal{H}^2$ arithmetic, including matrix-matrix multiplication (MMP) without formatted multiplications,

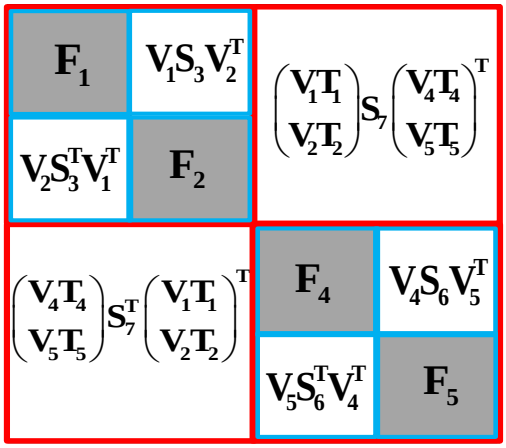


Fig. 1.3.: Illustration of a two-level HSS matrix.

minimal-rank MMP, new accuracy controlled $\mathcal{H}^2$ factorization and inversion, new accuracy controlled $\mathcal{H}^2$ factorization and inversion with concurrent change of cluster bases, $\mathcal{H}^2$-based direct sparse solver and new HSS recursive inverse with directly controlled accuracy. For constant-rank $\mathcal{H}^2$ matrices, the proposed accuracy directly controlled $\mathcal{H}^2$-arithmetic has a strict $O(N)$ complexity in both time and memory. For rank that linearly grows with the electrical size, the complexity of the proposed $\mathcal{H}^2$ arithmetic is $O(NlogN)$ in factorization and inversion time, and $O(N)$ in solution time and memory for solving volume IEs. Applications to large-scale interconnect extraction as well as large-scale scattering analysis, and comparisons with state-of-the-art solvers have demonstrated the clear advantages of the proposed new $\mathcal{H}^2$ arithmetic and resulting fast direct solutions with explicitly controlled accuracy. In addition to electromagnetic analysis, the new $\mathcal{H}^2$ arithmetic developed in this work can also be applied to other disciplines, where fast and large-scale numerical solutions are being pursued.

2. ACCURACY CONTROLLED $\mathcal{H}^2$-MATRIX-MATRIX PRODUCT WITH APPENDED CLUSTER BASES IN LINEAR COMPLEXITY

2.1 Introduction

The $\mathcal{H}^2$ matrix [7] serves as a general mathematical framework for compact representation and efficient computation of large dense systems. Both partial differential equation (PDE) and integral equation (IE) operators in electromagnetics can be represented as $\mathcal{H}^2$ matrices with controlled accuracy. The development of $\mathcal{H}^2$-matrix arithmetic such as addition, multiplication, and inverse are of critical importance to the development of fast solvers in electromagnetics [9]. Under the $\mathcal{H}^2$-matrix framework, it has been shown that an $\mathcal{H}^2$-matrix-based addition, matrix-vector product (MVP), and matrix-matrix product (MMP) all can be performed in linear complexity for constant-rank $\mathcal{H}^2$ [7]. However, the accuracy of existing $\mathcal{H}^2$-MMP algorithm like [7] is not directly controlled. This is because given two $\mathcal{H}^2$ matrices $\mathbf{A}_{\mathcal{H}^2}$ and $\mathbf{B}_{\mathcal{H}^2}$, the matrix structure and cluster bases of their product $\mathbf{C} = \mathbf{A}_{\mathcal{H}^2} \times \mathbf{B}_{\mathcal{H}^2}$ are assumed, and a formatted multiplication is performed, whose accuracy is not controlled. For example, the row cluster bases of $\mathbf{A}_{\mathcal{H}^2}$ and the column cluster bases of $\mathbf{B}_{\mathcal{H}^2}$ are assumed to be those of $\mathbf{C}$. This treatment lacks accuracy control since the original cluster basis may not be able to represent the new content generated during the MMP. The algorithm in [7] can be accurate if the cluster bases of the original matrices can also be used to accurately represent the matrix product. However, this is unknown in general applications, and hence the accuracy of existing MMP algorithm is not controlled. One can find many cases where a formatted multiplication can fail. For example, the row cluster bases of $\mathbf{A}_{\mathcal{H}^2}$ and the column cluster bases of $\mathbf{B}_{\mathcal{H}^2}$ are assumed to be those of $\mathbf{C}$. This treatment lacks accuracy control since the

original cluster basis may not be able to represent the new content generated during the MMP. For example, when multiplying a full-matrix block $\mathbf{F}$ by a low-rank block $\mathbf{V}_t\mathbf{S}\mathbf{V}_s{}^T$, treating the result as a low-rank block is correct. However, it is inaccurate to use the original row cluster basis as the product's row cluster basis, since the latter has been changed to $\mathbf{F}\mathbf{V}_t$. Even though a subblock of full matrix in $\mathbf{A}_{\mathcal{H}^2}$ is multiplied by a full-matrix block of $\mathbf{B}_{\mathcal{H}^2}$, the result of which should be a full-matrix block, if the targeted block in $\mathbf{C}$ is determined to be an admissible block based on the admissibility criterion, the matrix product is projected onto the row cluster basis of $\mathbf{A}_{\mathcal{H}^2}$ and the column cluster basis of $\mathbf{B}_{\mathcal{H}^2}$ to compute. The posteriori multiplication in [4] is more accurate than the formatted multiplication in [7]. But it is only suitable for special $\mathcal{H}^2$ matrices. Besides, this posteriori multiplication need much more computational time and memory during the computation. It needs to first present the product into an $\mathcal{H}$ matrix and then converts it into an $\mathcal{H}^2$ matrix, the complexity of which is not linear.

In this chapter, we develop two accuracy controlled $\mathcal{H}^2$-based MMP algorithms with appended cluster bases. The original $\mathcal{H}^2$-matrix structure can be preserved in the matrix product, or can be changed based on the structures of the two multipliers. The additional cluster bases are appended to old cluster bases based on the prescribed accuracy during the computation of the matrix-matrix product. Meanwhile, we are able to keep the computational complexity to be linear for constant-rank $\mathcal{H}^2$. For variable-rank cases such as those for electrically large analysis, the proposed MMP is also efficient.

2.2 Structure Preserved $\mathcal{H}^2$-Matrix-Matrix Product

2.2.1 Proposed Alorithm

To compute $\mathbf{A}_{\mathcal{H}^2} \times \mathbf{B}_{\mathcal{H}^2} = \mathbf{C}$, unlike the existing $\mathcal{H}^2$ formatted MMP [7], which is recursive, we propose to perform a one-way tree traversal from leaf level to root level. While doing the multiplications at each level, we update the row and column

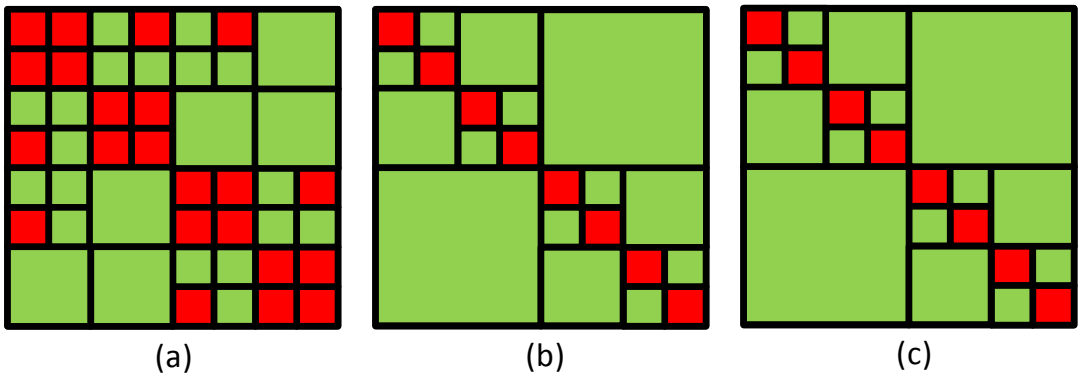


Fig. 2.1.: An $\mathcal{H}^2$-matrix structure. (a) $\mathbf{A}_{\mathcal{H}^2}$. (b) $\mathbf{B}_{\mathcal{H}^2}$. (c) $\mathbf{C}_{\mathcal{H}^2}$.

cluster bases based on prescribed accuracy to represent the product accurately. The proposed algorithm also facilitates parallelization. We will use the $\mathcal{H}^2$ matrices shown in Fig. 2.1 to illustrate the algorithm, but the algorithm is valid for any $\mathcal{H}^2$ matrix. In this figure, green blocks are admissible, and red ones are inadmissible. Here the structures of $\mathbf{A}_{\mathcal{H}^2}$, $\mathbf{B}_{\mathcal{H}^2}$ and $\mathbf{C}_{\mathcal{H}^2}$ matrices are determined based on the admissibility condition [4].

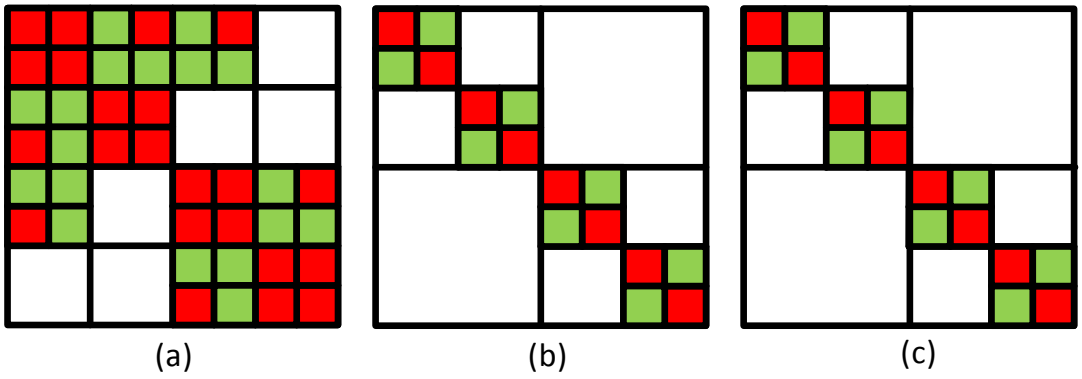


Fig. 2.2.: $\mathcal{H}^2$ matrix at leaf level. (a) $\mathbf{A}^L_{\mathcal{H}^2}$. (b) $\mathbf{B}^L_{\mathcal{H}^2}$. (c) $\mathbf{C}^L_{\mathcal{H}^2}$.

Leaf level

We start from leaf level ($l = L$). At leaf level, there are in total four matrix-matrix multiplication cases, i.e., 1) $\mathbf{F_A} \times \mathbf{F_B}$; 2) $\mathbf{F_A} \times \mathbf{R_B}$; 3) $\mathbf{R_A} \times \mathbf{F_B}$; 4) $\mathbf{R_A} \times \mathbf{R_B}$. We will use the leaf level blocks in Fig. 2.1 to illustrate the algorithm at this level, which is shown in Fig. 2.2. The resulting matrix block has two cases: 1) leaf level full block $\mathbf{F_C}$, noted as red in Fig. 2.2(c) and 2) leaf level admissible block $\mathbf{R}^L_{\mathbf{C}}$, noted as green in Fig. 2.2(c). The white blocks in Fig. 2.2 are zeros, meaning they are upper level blocks and not used for this level multiplications. Here we will show how to perform matrix-matrix multiplication based on different target matrix blocks.

1) Full block as target

If the product matrix block is leaf level full block $\mathbf{F_C}$, we can perform these four cases multiplications exactly by full matrix multiplications. For the admissible leaf blocks in four cases, we convert them into full matrix representation as $\mathbf{R}_{t,s} = \mathbf{V}_t\mathbf{S}_{t,s}(\mathbf{V}_s)^T$ in advance and then calculate products. Since the matrix is of $leafsize$, the computational cost for matrix-matrix multiplication is low as $O(leafsize^3)$.

2) Leaf level admissible block as target

If the target block is leaf level admissible block $\mathbf{R}^L_{\mathbf{C}}$, we need to update row and column cluster bases using multiplications from case-1 to case-3 to represent the product with controlled accuracy. The fourth case is exact because

$$\mathbf{R}^{\mathbf{A}}_{i,j} \times \mathbf{R}^{\mathbf{B}}_{j,k} = \mathbf{V}^{\mathbf{A}}_{i,r}\mathbf{S}^{\mathbf{A}}_{i,j}(\mathbf{V}^{\mathbf{A}}_{j,c})^T \times \mathbf{V}^{\mathbf{B}}_{j,r}\mathbf{S}^{\mathbf{B}}_{j,k}(\mathbf{V}^{\mathbf{B}}_{k,c})^T. \tag{2.1}$$

The product matrix block is obviously admissible and it preserves the original row cluster basis $\mathbf{V}^{\mathbf{A}}_{i,r}$, and column cluster basis $\mathbf{V}^{\mathbf{B}}_{k,c}$. Here the $\mathbf{V}^{\mathbf{A}}_{i,r}$ means the ith row cluster bases in matrix $\mathbf{A}$. We only need to calculate the coupling matrix, which is the center part, $\mathbf{S}^{\mathbf{A}}_{i,j}\widetilde{\mathbf{B}}_j\mathbf{S}^{\mathbf{B}}_{j,k}$, where $\widetilde{\mathbf{B}}_j$ is cluster bases product $(\mathbf{V}^{\mathbf{A}}_{j,c})^T \times \mathbf{V}^{\mathbf{B}}_{j,r}$, which can be computed in advance in linear complexity. For the other three cases, we need

to update cluster bases to preserve the low rank property of the target block. For the first case, $\mathbf{F}^{\mathbf{A}}_{i,j} \times \mathbf{F}^{\mathbf{B}}_{j,k}$, both the row and the column cluster bases need to be updated. For the second case, the products are clearly low rank due to the low-rank property of admissible blocks as:

$$\mathbf{F}^{\mathbf{A}}_{i,j} \times \mathbf{R}^{\mathbf{B}}_{j,k} = (\mathbf{F}^{\mathbf{A}}_{i,j}\mathbf{V}^{\mathbf{B}}_{j,r}) \times \mathbf{S}^{\mathbf{B}}_{j,k} \times (\mathbf{V}^{\mathbf{B}}_{k,c})^T. \tag{2.2}$$

So we only need to use $\mathbf{F}^{\mathbf{A}}_{i,j}\mathbf{V}^{\mathbf{B}}_{j,r}$ to update the row cluster basis $\mathbf{V}^{\mathbf{A}}_{i,r}$. This is because the row cluster basis of this product block is no longer $\mathbf{V}^{\mathbf{A}}_{i,r}$, if $\mathbf{F}^{\mathbf{A}}_{i,j}\mathbf{V}^{\mathbf{B}}_{j,r}$ contains new information not included in $\mathbf{V}^{\mathbf{A}}_{i,r}$. Similarly, the third case also has clearly a low rank product as:

$$\mathbf{R}^{\mathbf{A}}_{i,j} \times \mathbf{F}^{\mathbf{B}}_{j,k} = \mathbf{V}^{\mathbf{A}}_{i,r} \times \mathbf{S}^{\mathbf{A}}_{i,j} \times [(\mathbf{V}^{\mathbf{A}}_{j,c})^T\mathbf{F}^{\mathbf{B}}_{j,k}]. \tag{2.3}$$

And we need to use $(\mathbf{V}^{\mathbf{A}}_{j,c})^T\mathbf{F}^{\mathbf{B}}_{j,k}$ to update column cluster basis $\mathbf{V}^{\mathbf{B}}_{k,c}$ to get additional information for representing the product accurately. Since there are many case-1, case-2 and case-3 products encountered at the leaf level for the same row or column cluster, to systematically update the i's row cluster basis $\mathbf{V}^{\mathbf{A}}_{i,r}$ and k's column cluster basis $\mathbf{V}^{\mathbf{B}}_{k,c}$, we consider these products at the same time.

To update i's row cluster basis $\mathbf{V}^{\mathbf{A}}_{i,r}$, we need to consider all the products in case-1 and case-2. So we first find all the products $\mathbf{F}^{\mathbf{A}}_{i,j} \times \mathbf{F}^{\mathbf{B}}_{j,k}$ with target $\mathbf{C}_{i,k}$ to be an admissible block at i's row of $\mathbf{C}_{\mathcal{H}^2}$, whose number is bounded by constant C^2_{sp}. We calculate the Gram matrix sum as:

$$\mathbf{V}^{\mathbf{A}}_{i,new1} = \sum_{j=1}^{O(C_{sp})} \sum_{k=1}^{O(C_{sp})} (\mathbf{F}^{\mathbf{A}}_{i,j}\mathbf{F}^{\mathbf{B}}_{j,k})(\mathbf{F}^{\mathbf{A}}_{i,j}\mathbf{F}^{\mathbf{B}}_{j,k})^H \tag{2.4}$$

Then we find all the dense matrices in i's row of $\mathbf{A}_{\mathcal{H}^2}$, whose number is also bounded by constant C_{sp}, and multiply them with corresponding row cluster basis $\mathbf{V}^{\mathbf{B}}_{j,r}$ in $\mathbf{B}_{\mathcal{H}^2}$, obtaining Gram matrix sum as:

$$\mathbf{V}^{\mathbf{A}}_{i,new2} = \sum_{j=1}^{O(C_{sp})} (\mathbf{F}^{\mathbf{A}}_{i,j}\mathbf{V}^{\mathbf{B}}_{j,r})(\mathbf{F}^{\mathbf{A}}_{i,j}\mathbf{V}^{\mathbf{B}}_{j,r})^H. \tag{2.5}$$

Since the magnitude of Gram matrix sum $\mathbf{V}^{\mathbf{A}}_{i,new1}$ and $\mathbf{V}^{\mathbf{A}}_{i,new2}$ differs greatly, we normalize them separately and sum them up to get

$$\mathbf{V}^{\mathbf{A}}_{i,new} = \widehat{\mathbf{V}^{\mathbf{A}}_{i,new1}} + \widehat{\mathbf{V}^{\mathbf{A}}_{i,new2}}. \tag{2.6}$$

The $\widehat{\ }$ above $\mathbf{V}^{\mathbf{A}}_{i,new1}$ and $\mathbf{V}^{\mathbf{A}}_{i,new2}$ means matrix normalization, and we could use matrix norm such as F-norm to normalize them. We then use this $\mathbf{V}^{\mathbf{A}}_{i,new}$ to augment i's row cluster basis in $\mathbf{A}_{\mathcal{H}^2}$ by deducting the original cluster bases $\mathbf{V}^{\mathbf{A}}_{i,r}$ as $\mathbf{G}^{\mathbf{A}}_i = [\mathbf{I} - \mathbf{V}^{\mathbf{A}}_{i,r}(\mathbf{V}^{\mathbf{A}}_{i,r})^H] \times \mathbf{V}^{\mathbf{A}}_{i,new} \times [\mathbf{I} - \mathbf{V}^{\mathbf{A}}_{i,r}(\mathbf{V}^{\mathbf{A}}_{i,r})^H]^H$. After this, we perform an SVD on $\mathbf{G}^{\mathbf{A}}_i$ to obtain the additional cluster bases $\mathbf{V}^{\mathbf{A},add}_{i,r}$ for cluster i based on prescribed accuracy ϵ_{add}. Now the updated i's row cluster bases $\widetilde{\mathbf{V}}^{\mathbf{A}}_{i,r} = [\mathbf{V}^{\mathbf{A}}_{i,r}\ \mathbf{V}^{\mathbf{A},add}_{i,r}]$ can be used to accurately represent the admissible blocks in i's row of $\mathbf{C}_{\mathcal{H}^2}$.

For updating the column cluster basis in k's column in $\mathbf{B}_{\mathcal{H}^2}$, $\mathbf{V}^{\mathbf{B}}_{k,c}$, the steps are similar. The case-1 and case-3 products are used to update column cluster basis. For case-1, we consider all the products $\mathbf{F}^{\mathbf{A}}_{i,j} \times \mathbf{F}^{\mathbf{B}}_{j,k}$ with target $\mathbf{C}_{i,k}$ to be an admissible block in k's column of $\mathbf{C}_{\mathcal{H}^2}$ and calculate the Gram matrix sum as:

$$\mathbf{V}^{\mathbf{B}}_{k,new1} = \sum_{i=1}^{O(C_{sp})} \sum_{j=1}^{O(C_{sp})} (\mathbf{F}^{\mathbf{A}}_{i,j}\mathbf{F}^{\mathbf{B}}_{j,k})^T (\mathbf{F}^{\mathbf{A}}_{i,j}\mathbf{F}^{\mathbf{B}}_{j,k})^*. \tag{2.7}$$

Here the superscript $*$ means taking the complex conjugate of each entry in the matrix. We then find all the dense matrices in k's column of $\mathbf{B}_{\mathcal{H}^2}$ and multiply them with corresponding column cluster basis $\mathbf{V}^{\mathbf{A}}_{j,c}$ in $\mathbf{A}_{\mathcal{H}^2}$, obtaining Gram matrix sum:

$$\mathbf{V}^{\mathbf{B}}_{k,new2} = \sum_{j=1}^{O(C_{sp})} [(\mathbf{V}^{\mathbf{A}}_{j,c})^T \mathbf{F}^{\mathbf{B}}_{j,k}]^T [(\mathbf{V}^{\mathbf{A}}_{j,c})^T \mathbf{F}^{\mathbf{B}}_{j,k}]^*. \tag{2.8}$$

We also normalize $\mathbf{V}^{\mathbf{B}}_{k,new1}$ and $\mathbf{V}^{\mathbf{B}}_{k,new2}$ and sum them up as:

$$\mathbf{V}^{\mathbf{B}}_{k,new} = \widehat{\mathbf{V}^{\mathbf{B}}_{k,new1}} + \widehat{\mathbf{V}^{\mathbf{B}}_{k,new2}}. \tag{2.9}$$

We then use this $\mathbf{V}^{\mathbf{B}}_{k,new}$ to augment the cluster basis by computing $\mathbf{G}^{\mathbf{B}}_k = [\mathbf{I} - \mathbf{V}^{\mathbf{B}}_{k,c}(\mathbf{V}^{\mathbf{B}}_{k,c})^H] \times \mathbf{V}^{\mathbf{B}}_{k,new} \times [\mathbf{I} - \mathbf{V}^{\mathbf{B}}_{k,c}(\mathbf{V}^{\mathbf{B}}_{k,c})^H]^H$. After this, we perform an SVD on $\mathbf{G}^{\mathbf{B}}_k$ to obtain the additional cluster bases $\mathbf{V}^{\mathbf{B},add}_{k,c}$ for cluster i based on prescribed accuracy

ϵ_{add}. Now the updated k's column cluster basis $\widetilde{\mathbf{V}}^{\mathbf{B}}_{k,c} = [\mathbf{V}^{\mathbf{B}}_{k,c}\ \mathbf{V}^{\mathbf{B},add}_{k,c}]$ can be used to accurately represent the admissible blocks in k's column of $\mathbf{C}_{\mathcal{H}^2}$.

After updating row and column cluster bases of the product matrix, we can now calculate the coupling matrices of case-1 , case-2 and case-3 multiplications. The case-4 products are known. So we show the coupling matrices for these four cases as:

$$\mathbf{S}^{\mathbf{C}}_{i,k} = \begin{cases} (\widetilde{\mathbf{V}}^{\mathbf{A}}_{i,r})^H \mathbf{F}^{\mathbf{A}}_{i,j} \mathbf{F}^{\mathbf{B}}_{j,k} (\widetilde{\mathbf{V}}^{\mathbf{B}}_{k,c})^* & \text{case-1} \\ (\widetilde{\mathbf{V}}^{\mathbf{A}}_{i,r})^H \mathbf{F}^{\mathbf{A}}_{i,j} \mathbf{V}^{\mathbf{B}}_{j,r} \mathbf{S}^{\mathbf{B}}_{j,k} & \text{case-2} \\ \mathbf{S}^{\mathbf{A}}_{i,j} (\mathbf{V}^{\mathbf{A}}_{j,c})^T \mathbf{F}^{\mathbf{B}}_{j,k} (\widetilde{\mathbf{V}}^{\mathbf{B}}_{k,c})^* & \text{case-3} \\ \mathbf{S}^{\mathbf{A}}_{i,j} \widetilde{\mathbf{B}}_j \mathbf{S}^{\mathbf{B}}_{j,k} & \text{case-4.} \end{cases} \tag{2.10}$$

Then the resulting admissible blocks in $\mathbf{C}_{\mathcal{H}^2}$ all can be represented as $\mathbf{R}^{\mathbf{C}}_{i,k} = \widetilde{\mathbf{V}}^{\mathbf{A}}_{i,r} \times \mathbf{S}^{\mathbf{C}}_{i,k} \times (\widetilde{\mathbf{V}}^{\mathbf{B}}_{k,c})^T$.

In order to speed up leaf level multiplications, we can collect the $\mathbf{F}$ blocks in $\mathbf{A}_{\mathcal{H}^2}$ by $(\widetilde{\mathbf{V}}^{\mathbf{A}}_{i,r})^H \mathbf{F}^{\mathbf{A}}_{i,j} \mathbf{V}^{\mathbf{B}}_{j,r}$ and $\mathbf{F}$ blocks in $\mathbf{B}_{\mathcal{H}^2}$ by $(\mathbf{V}^{\mathbf{A}}_{j,c})^T \mathbf{F}^{\mathbf{B}}_{j,k} (\widetilde{\mathbf{V}}^{\mathbf{B}}_{j,c})^*$ in advance to facilitate case-2 products $\mathbf{F}^{\mathbf{A}}_{i,j} \times \mathbf{R}^{\mathbf{B}}_{j,k}$ for different k and case-3 products $\mathbf{R}^{\mathbf{A}}_{i,j} \times \mathbf{F}^{\mathbf{B}}_{j,k}$ for different i by computations of rank size. Note that this collect is accurate now because it is for products between $\mathbf{F}$ and $\mathbf{R}$, and the products' row and column cluster bases have been updated to account for $\mathbf{F} \times \mathbf{R}$ and $\mathbf{R} \times \mathbf{F}$ cases. Here, we conclude all the operations at the leaf level:

1. Prepare cluster bases product $\widetilde{\mathbf{B}}$;
2. Updating row and column cluster bases to $\widetilde{\mathbf{V}}$;
3. Collect the $\mathbf{F}$ blocks in $\mathbf{A}_{\mathcal{H}^2}$ and $\mathbf{B}_{\mathcal{H}^2}$;
4. Perform four cases of multiplications.

Non-leaf level

After finishing leaf level multiplication, we need to update the original row and column transfer matrices with zeros due to the leaf level cluster bases updates to keep the nested property of cluster bases, which is as shown below

$$\begin{bmatrix} \mathbf{V}_{t_1}\mathbf{T}_{t_1} \\ \mathbf{V}_{t_2}\mathbf{T}_{t_2} \end{bmatrix} = \begin{bmatrix} [\mathbf{V}_{t_1}\mathbf{V}_{t_1}^{add}] & \mathbf{0} \\ \mathbf{0} & [\mathbf{V}_{t_2}\mathbf{V}_{t_2}^{add}] \end{bmatrix} \begin{bmatrix} \mathbf{T}_{t_1} \\ \mathbf{0} \\ \mathbf{T}_{t_2} \\ \mathbf{0} \end{bmatrix}. \tag{2.11}$$

In this way, we can keep the nested property of original cluster bases and all the products can use these updated cluster bases. Since multiplications at different nonleaf levels are the same, we will use level $(L-1)$ as an example to show the computations. And this level multiplication is shown in Fig. 2.3 as $\mathbf{A}_{\mathcal{H}^2}^{L-1}$, $\mathbf{B}_{\mathcal{H}^2}^{L-1}$ and $\mathbf{C}_{\mathcal{H}^2}^{L-1}$. At level $(L-1)$, we perform four types of matrix products: 1) $\mathbf{NL} \times \mathbf{R}$; 2) $\mathbf{R} \times \mathbf{NL}$; 3) $\mathbf{R} \times \mathbf{R}$; and 4) $\mathbf{NL} \times \mathbf{NL}$. And the resulting matrix block also have two cases: 1) non-leaf block at this level, noted as red in Fig. 2.3(c) and 2) admissible block at this level, noted as green in Fig. 2.3 (c). For these two different resulting matrix blocks, the operations are the same, i.e. calculating the coupling matrices. The only difference is the post process operations after getting products, which will be discussed later. Here

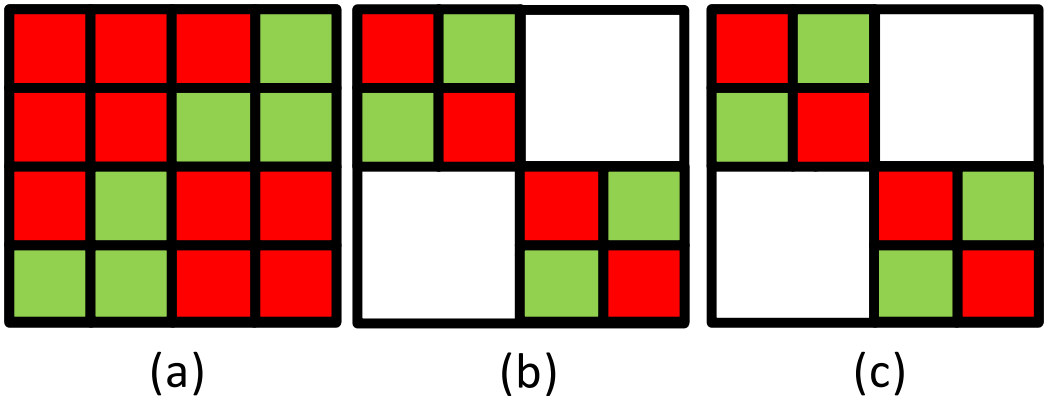


Fig. 2.3.: $\mathcal{H}^2$-matrix block at non-leaf level $(L-1)$. (a) $\mathbf{A}_{\mathcal{H}^2}^{L-1}$. (b) $\mathbf{B}_{\mathcal{H}^2}^{L-1}$. (c) $\mathbf{C}_{\mathcal{H}^2}^{L-1}$.

for the case-4 multiplication, which is $\mathbf{NL} \times \mathbf{NL}$, the target block is only admissible block at this level. Because the $\mathbf{NL} \times \mathbf{NL} \to \mathbf{NL}$ has been taken care by lower levels' multiplications. For case-4 multiplication, we need to update both row and column transfer matrices.

The **NL** block used in case-1 and case-2 multiplications at level $(L-1)$ is a merged matrix with four blocks from the collected **F** matrices and coupling matrices at leaf level L so that the block is of rank size. Such a merged **NL** block of rank size is accurate because we've already updated leaf level cluster bases to count multiplication between **F** and **R** and this **NL** block is used in multiplications with **R**. It resembles **F** at the leaf level. For example, a merged block at level $(L-1)$ in $\mathbf{A}_{\mathcal{H}^2}$ is like

$$(\mathbf{NL}^{\mathbf{A}}_{i,j})_{merge} = \begin{bmatrix} (\widetilde{\mathbf{V}}^{\mathbf{A}}_{i_1,r})^H \mathbf{F}^{\mathbf{A}}_{i_1,j_1} \mathbf{V}^{\mathbf{B}}_{j_1,r} & \begin{bmatrix} \mathbf{S}^{\mathbf{A}}_{i_1,j_2} \widetilde{\mathbf{B}}_{j_2} \\ \mathbf{0} \end{bmatrix} \\ \begin{bmatrix} \mathbf{S}^{\mathbf{A}}_{i_2,j_1} \widetilde{\mathbf{B}}_{j_1} \\ \mathbf{0} \end{bmatrix} & (\widetilde{\mathbf{V}}^{\mathbf{A}}_{i_2,r})^H \mathbf{F}^{\mathbf{A}}_{i_2,j_2} \mathbf{V}^{\mathbf{B}}_{j_2,r} \end{bmatrix}. \tag{2.12}$$

The collect operation on the left uses updated row cluster bases, since the updated cluster bases are accurate for these multiplications. And the collect operation on the right uses row cluster basis in $\mathbf{B}_{\mathcal{H}^2}$. Because the **NL** block in $\mathbf{A}_{\mathcal{H}^2}$ will be used to multiply admissible blocks in $\mathbf{B}_{\mathcal{H}^2}$. And a merged **NL** block in $\mathbf{B}_{\mathcal{H}^2}$ is like

$$(\mathbf{NL}^{\mathbf{B}}_{i,j})_{merge} = \begin{bmatrix} (\mathbf{V}^{\mathbf{A}}_{i_1,c})^T \mathbf{F}^{\mathbf{B}}_{i_1,j_1} (\widetilde{\mathbf{V}}^{\mathbf{B}}_{j_1,c})^* & \begin{bmatrix} \widetilde{\mathbf{B}}_{j_2} \mathbf{S}^{\mathbf{B}}_{i_1,j_2} \\ \mathbf{0} \end{bmatrix} \\ \begin{bmatrix} \widetilde{\mathbf{B}}_{j_1} \mathbf{S}^{\mathbf{B}}_{i_2,j_1} \\ \mathbf{0} \end{bmatrix} & (\mathbf{V}^{\mathbf{A}}_{i_2,c})^T \mathbf{F}^{\mathbf{B}}_{i_2,j_2} (\widetilde{\mathbf{V}}^{\mathbf{B}}_{j_2,c})^* \end{bmatrix}. \tag{2.13}$$

Similarly, the left collect use column cluster basis in $\mathbf{A}_{\mathcal{H}^2}$ and right collect use updated column cluster bases. Case-3 again does not require any updates on transfer matrices, since the transfer matrices in equation (2.11) keeps the original cluster bases at a nonleaf level. And the coupling matrix is $\mathbf{S}^{\mathbf{A}}_{i,j} \widetilde{\mathbf{B}}_j \mathbf{S}^{\mathbf{B}}_{j,k}$. For case-1, the row transfer matrices of the product need to be updated. Meanwhile, the column transfer matrices need updates for case-2.

In order to update row transfer matrix $\mathbf{T}^{\mathbf{A}}_{i,r}$, we need two steps. The first step is from case-4 multiplication. We first calculate the Gram matrix sum as:

$$\mathbf{T}^{\mathbf{A}}_{i,new1} = \sum_{j=1}^{O(C_{sp})} \sum_{k=1}^{O(C_{sp})} (\mathbf{NL}^{\mathbf{A}}_{i,j}\mathbf{NL}^{\mathbf{B}}_{j,k})(\mathbf{NL}^{\mathbf{A}}_{i,j}\mathbf{NL}^{\mathbf{B}}_{j,k})^H. \tag{2.14}$$

The $\mathbf{NL}^{\mathbf{A}}_{i,j}\mathbf{NL}^{\mathbf{B}}_{j,k}$ product is calculated at lower level multiplication and merge the resulting coupling matrices. The second step update is due to case-1 multiplication at non-leaf level. We find all the $\mathbf{NL}^{\mathbf{A}}_{i,j}$ blocks for i's nonleaf cluster and multiply them with corresponding transfer matrices $\mathbf{T}^{\mathbf{B}}_{j,r}$. And we calculate the Gram matrix sum as

$$\mathbf{T}^{\mathbf{A}}_{i,new2} = \sum_{j=1}^{O(C_{sp})} ((\mathbf{NL}^{\mathbf{A}}_{i,j})_{merge}\mathbf{T}^{\mathbf{B}}_{j,r})((\mathbf{NL}^{\mathbf{A}}_{i,j})_{merge}\mathbf{T}^{\mathbf{B}}_{j,r})^H. \tag{2.15}$$

Again, we normalize the two Gram matrices and get

$$\mathbf{T}^{\mathbf{A}}_{i,new} = \widehat{\mathbf{T}^{\mathbf{A}}_{i,new1}} + \widehat{\mathbf{T}^{\mathbf{A}}_{i,new2}}. \tag{2.16}$$

We then use this $\mathbf{T}^{\mathbf{A}}_{i,new}$ to augment the transfer matrix of $\mathbf{T}^{\mathbf{A}}_{i,r}$ by calculating $\mathbf{G}^{\mathbf{A}}_i = [\mathbf{I} - \mathbf{T}^{\mathbf{A}}_{i,r}(\mathbf{T}^{\mathbf{A}}_{i,r})^H] \times \mathbf{T}^{\mathbf{A}}_{i,new} \times [\mathbf{I} - \mathbf{T}^{\mathbf{A}}_{i,r}(\mathbf{T}^{\mathbf{A}}_{i,r})^H]^H$. After this, we perform an SVD on $\mathbf{G}^{\mathbf{A}}_i$ to obtain the additional transfer matrices $\mathbf{T}^{\mathbf{A},add}_{i,r}$ for non-leaf cluster i based on prescribed accuracy ϵ_{add}. Then the updated row transfer matrices for i's row cluster in $\mathbf{C}_{\mathcal{H}^2}$ is $\widetilde{\mathbf{T}}^{\mathbf{A}}_{i,r} = [\mathbf{T}^{\mathbf{A}}_{i,r}\ \mathbf{T}^{\mathbf{A},add}_{i,r}]$. Similarly, we can update the column transfer matrices for non-leaf cluster k, which is $\mathbf{T}^{\mathbf{B}}_{k,c}$. The first part is

$$\mathbf{T}^{\mathbf{B}}_{k,new1} = \sum_{i=1}^{O(C_{sp})} \sum_{j=1}^{O(C_{sp})} (\mathbf{NL}^{\mathbf{A}}_{i,j}\mathbf{NL}^{\mathbf{B}}_{j,k})^T(\mathbf{NL}^{\mathbf{A}}_{i,j}\mathbf{NL}^{\mathbf{B}}_{j,k})^*. \tag{2.17}$$

And the second part is

$$\mathbf{T}^{\mathbf{B}}_{i,new2} = \sum_{j=1}^{O(C_{sp})} [(\mathbf{T}^{\mathbf{A}}_{j,r})^T(\mathbf{NL}^{\mathbf{B}}_{i,j})_{merge}]^T[(\mathbf{T}^{\mathbf{A}}_{j,r})^T(\mathbf{NL}^{\mathbf{B}}_{i,j})_{merge}]^*. \tag{2.18}$$

Then we normalize the two Gram matrices and get

$$\mathbf{T}^{\mathbf{B}}_{k,new} = \widehat{\mathbf{T}^{\mathbf{B}}_{k,new1}} + \widehat{\mathbf{T}^{\mathbf{B}}_{k,new2}}. \tag{2.19}$$

We then use this $\mathbf{T}^{\mathbf{B}}_{k,new}$ to augment the transfer matrix of $\mathbf{T}^{\mathbf{B}}_{k,c}$ by calculating $\mathbf{G}^{\mathbf{B}}_{k} = [\mathbf{I} - \mathbf{T}^{\mathbf{B}}_{k,c}(\mathbf{T}^{\mathbf{B}}_{k,c})^{H}] \times \mathbf{T}^{\mathbf{B}}_{k,new} \times [\mathbf{I} - \mathbf{T}^{\mathbf{B}}_{k,c}(\mathbf{T}^{\mathbf{B}}_{k,c})^{H}]^{H}$. After this, we perform an SVD on $\mathbf{G}^{\mathbf{B}}_{k}$ to obtain the additional transfer matrices $\mathbf{T}^{\mathbf{B},add}_{k,c}$ for nonleaf cluster k based on prescribed accuracy ϵ_{add}. Then the updated column transfer matrices for k's column cluster in $\mathbf{C}_{\mathcal{H}^2}$ is $\widetilde{\mathbf{T}}^{\mathbf{B}}_{k,c} = [\mathbf{T}^{\mathbf{B}}_{k,c} \ \mathbf{T}^{\mathbf{B},add}_{k,c}]$. After updating row and column transfer matrices, we now can calculate the coupling matrices of these four cases products at non-leaf level as:

$$\mathbf{S}^{\mathbf{C}}_{i,k} = \begin{cases} (\widetilde{\mathbf{T}}^{\mathbf{A}}_{i,r})^{H}(\mathbf{NL}^{\mathbf{A}}_{i,j})_{merge}\mathbf{T}^{\mathbf{B}}_{j,r}\mathbf{S}^{\mathbf{B}}_{j,k} & \text{case-1} \\ \mathbf{S}^{\mathbf{A}}_{i,j}(\mathbf{T}^{\mathbf{A}}_{j,c})^{T}(\mathbf{NL}^{\mathbf{B}}_{j,k})_{merge}(\widetilde{\mathbf{T}}^{\mathbf{B}}_{k,c})^{*} & \text{case-2} \\ \mathbf{S}^{\mathbf{A}}_{i,j}\widetilde{\mathbf{B}}_{j}\mathbf{S}^{\mathbf{B}}_{j,k} & \text{case-3.} \\ (\widetilde{\mathbf{T}}^{\mathbf{A}}_{i,r})^{H}\mathbf{NL}^{\mathbf{A}}_{i,j}\mathbf{NL}^{\mathbf{B}}_{j,k}(\widetilde{\mathbf{T}}^{\mathbf{B}}_{k,c})^{*} & \text{case-4} \end{cases} \tag{2.20}$$

Since there are many case-1 and case-2 products encountered at the non-leaf level, we again collect **NL** matrix blocks in $\mathbf{A}_{\mathcal{H}^2}$ by $(\widetilde{\mathbf{T}}^{\mathbf{A}}_{i,r})^{H}\mathbf{NL}^{\mathbf{A}}_{i,j}\mathbf{T}^{\mathbf{B}}_{j,r}$ and **NL** blocks in $\mathbf{B}_{\mathcal{H}^2}$ by $(\mathbf{T}^{\mathbf{A}}_{j,c})^{T}\mathbf{NL}^{\mathbf{B}}_{j,k}(\widetilde{\mathbf{T}}^{\mathbf{B}}_{j,c})^{*}$ in advance.

In conclusion, at a non-leaf level, we do:

1. Merge four small collected blocks to **NL** for $\mathbf{A}_{\mathcal{H}^2}$ and $\mathbf{B}_{\mathcal{H}^2}$;
2. Updating row and column transfer matrices to $\widetilde{\mathbf{T}}$;
3. Collect the **NL** blocks in $\mathbf{A}_{\mathcal{H}^2}$ and $\mathbf{B}_{\mathcal{H}^2}$ using updated $\widetilde{\mathbf{T}}$;
4. Perform four cases of multiplications.

After we finish one-way bottom-up tree traversal to calculate block matrix products at all the levels, i.e. from leaf level to minimal admissible level. We need to perform post-processing for the coupling matrices at a nonleaf level if the resulting matrix block in $\mathbf{C}_{\mathcal{H}^2}$ is **NL**. This could be efficiently done by performing one-way top-down split process, as shown in [7]. The post processing stage is just split the coupling matrices at **NL** to lower level admissible blocks or inadmissible blocks.

2.2.2 Accuracy and Complexity Analysis

In this section, we analyze the accuracy and computational complexity of the proposed algorithm to calculate $\mathcal{H}^2$-matrix-matrix products.

Accuracy

Different from existing formatted $\mathcal{H}^2$-matrix-matrix products [7], in the proposed new algorithm, the accuracy of the product is directly controlled. When generating an $\mathcal{H}^2$ matrix to represent the original dense matrix, the accuracy is controlled by $\epsilon_{\mathcal{H}^2}$, which is the same as in [15]. The accuracy of the proposed $\mathcal{H}^2$-matrix-matrix products is controlled by ϵ_{add}.

Time and Memory Complexity

The proposed structure-preserving MMP involves $O(L)$ levels of computation. At each level, there are 2^l clusters. For each cluster, we have $O(C_{sp})$ admissible and inadmissible blocks. For each block, we need to perform $O(C_{sp})$ block matrix multiplications. Each of them costs $O(k_l^3)$. The time complexity of the proposed MMP can be found as

$$\textbf{Time Complexity} = \sum_{l=0}^{L} C_{sp}^2 2^l k_l^3 = {C_{sp}}^2 \sum_{l=0}^{L} 2^l k_l^3. \tag{2.21}$$

And the storage for each block is $O(k_l^2)$, with each cluster having C_{sp} blocks. So the memory complexity is

$$\textbf{Memory Complexity} = \sum_{l=0}^{L} C_{sp} 2^l k_l^2 = C_{sp} \sum_{l=0}^{L} 2^l k_l^2. \tag{2.22}$$

Recall k_l is the rank at tree level l. Hence, (2.21) and (2.22) show that the overall complexity is a function of rank k_l. Taking into account the rank's growth with electrical size as shown in [32], we can get the time and memory complexity of proposed MMP for different rank scaling. For constant-rank $\mathcal{H}^2$ matrices, since k_l is a constant

irrespective of matrix size, the complexity of the proposed direct solution is strictly $O(N)$ in both CPU time and memory consumption. The analysis is as below:

For constant k_l:

$$\textbf{Time Complexity} = C_{sp}^2 k_l^3 \sum_{l=0}^{L} 2^l = O(N), \tag{2.23}$$

$$\textbf{Memory Complexity} = C_{sp} k_l^2 \sum_{l=0}^{L} 2^l = O(N). \tag{2.24}$$

For electrodynamic analysis, to ensure a prescribed accuracy, the rank becomes a function of electrical size, and thereby tree level. Different $\mathcal{H}^2$-matrix representations can result in different complexities, because their rank's behavior is different. Using a minimal-rank $\mathcal{H}^2$-representation, as shown by [32], the rank grows linearly with electrical size for general 3-D problems. In a VIE, k_l is proportional to the cubic root of matrix size at level l, because this is the electrical size at level l. Hence for a VIE, (2.21) and (2.22) become

For k_l linearly growing with electrical size:

$$\textbf{Time Complexity} = {C_{sp}}^2 \sum_{l=0}^{L} 2^l \left[\left(\frac{N}{2^l} \right)^{\frac{1}{3}} \right]^3 = O(NlogN), \tag{2.25}$$

$$\textbf{Memory Complexity} = C_{sp} \sum_{l=0}^{L} 2^l \left[\left(\frac{N}{2^l} \right)^{\frac{1}{3}} \right]^2 = O(N). \tag{2.26}$$

So the time complexity of proposed MMP algorithm for 3-D electrodynamic analysis is $O(NlogN)$, and the memory complexity is $O(N)$.

2.2.3 Numerical Results

In order to demonstrate the accuracy and low computational complexity of the proposed fast $\mathcal{H}^2$-matrix-matrix multiplication for general $\mathcal{H}^2$ matrices, we use $\mathcal{H}^2$ matrices resulting from large-scale capacitance extraction and volume integral equations (VIE) for electromagnetic analysis. The capacitance extraction matrix is shown

in [9]. The VIE formulation we use is based on [31] with SWG vector bases for expanding electric flux density in each tetrahedral element. A variety of large-scale examples involving over one million unknowns are simulated to examine the accuracy and complexity of the proposed MMP algorithm. The $\mathcal{H}^2$ matrix for each example is constructed based on the method described in [14, 15]. The capacitance matrix is used to demonstrate the proposed MMP algorithm performance for constant-rank $\mathcal{H}^2$ matrices. And we also simulate large scale 2-D and 3-D examples to show the time and memory complexity for the proposed MMP with other rank scalings.

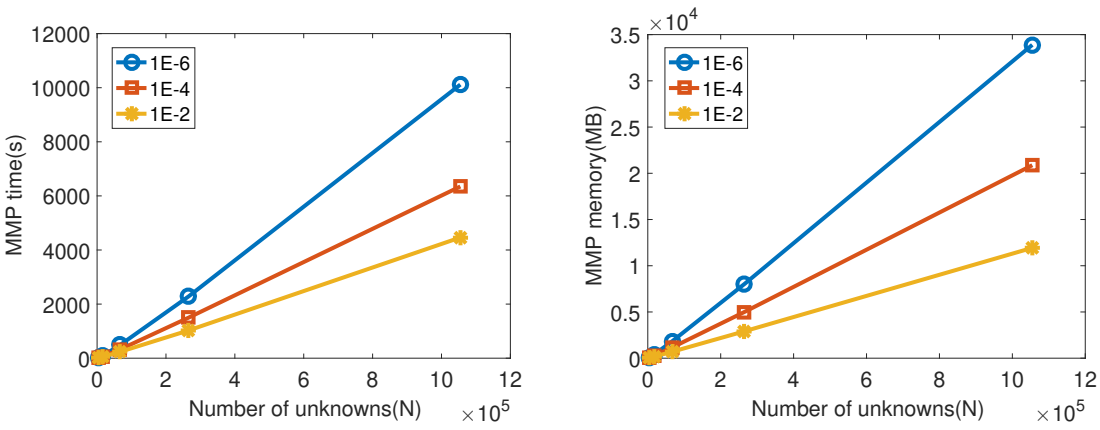
Two-layer Cross Bus

The first example is the capacitance extraction of a 2-layer cross bus structure. In each layer, there are m conductors, and each conductors has a dimension of $1 \times 1 \times (2m+1)$ m^3. We simulate a suite of such structures with 16, 32, 64, 128 and 256 buses respectively. The parameters used in the $\mathcal{H}^2$-matrix construction are $leafsize = 30$, admissibility condition [7] $\eta = 1.0$, and $\epsilon_{\mathcal{H}^2} = 10^{-4}$. For the proposed $\mathcal{H}^2$ MMP, the updated cluster bases are truncated at 10^{-2}, 10^{-4} and 10^{-6}. As shown in Fig. 2.4(a) and Fig. 2.4(b), the proposed MMP has linear time and memory performance for different truncation thresholds. We also check the product error by matrix-vector product as

$$\epsilon_{rel} = \frac{||\mathbf{C}_{\mathcal{H}^2} x - \mathbf{A}_{\mathcal{H}^2}(\mathbf{B}_{\mathcal{H}^2} x)||_F}{||\mathbf{A}_{\mathcal{H}^2}(\mathbf{B}_{\mathcal{H}^2} x)||_F}. \tag{2.27}$$

The reference vector is $\mathbf{A}_{\mathcal{H}^2} \times (\mathbf{B}_{\mathcal{H}^2} \times x)$, in which x is a random right hand side. We first calculate $y = \mathbf{B}_{\mathcal{H}^2} \times x$, and then $\mathbf{A}_{\mathcal{H}^2} \times y$. The $\mathcal{H}^2$ matrix vector product is carried out as shown in [7], which is exact. From Table 2.1, we can see the accuracy of proposed MMP is directly controlled by ϵ_{add} used in appending cluster bases.

Since no executable of [7] is available in public domain, We have implemented our own version of the formatted MMP, and termed it existing $\mathcal{H}^2$ MMP. In this way, we are able to compare our proposed MMP with the existing one using exactly the same $\mathcal{H}^2$ matrix. This is more objective comparison since the different $\mathcal{H}^2$ matrices



(a) (b)

Fig. 2.4.: Structure-preserving MMP performance for $\mathbf{A}_{\mathcal{H}^2} \times \mathbf{B}_{\mathcal{H}^2}$ of large-scale capacitance extraction. (a) Time scaling v.s. N. (b) Memory scaling v.s. N.

Table 2.1.: Structure-preserving MMP error at different ϵ_{add} for large-scale capacitance extraction.

N	4,480	17,152	67,072	265,216	1,054,720
1E-2	8.05E-4	8.45E-4	6.84E-4	6.25E-4	1.45E-3
1E-4	9.50E-5	1.31E-4	9.11E-5	7.39E-5	5.98E-5
1E-6	5.63E-6	6.12E-6	5.59E-6	6.16E-6	1.04E-5

would have different detailed structure and content, resulting different MMP time and memory. In Fig. 2.5, we compare the CPU run time and memory usage of the two MMP algorithms as a function of N. Clear linear scaling can be observed from these two MMP, but the proposed on takes only about half time as well as achieve better accuracy. For the last case having 1,054,720 unknowns, the proposed one only takes 36 minutes to do MMP, while the existing formatted one takes 75 minutes. This can be shown in Table 2.2. Since the accuracy of existing $\mathcal{H}^2$-MMP are uncontrollable, we can only achieve similar random RHS error by not updating cluster bases for

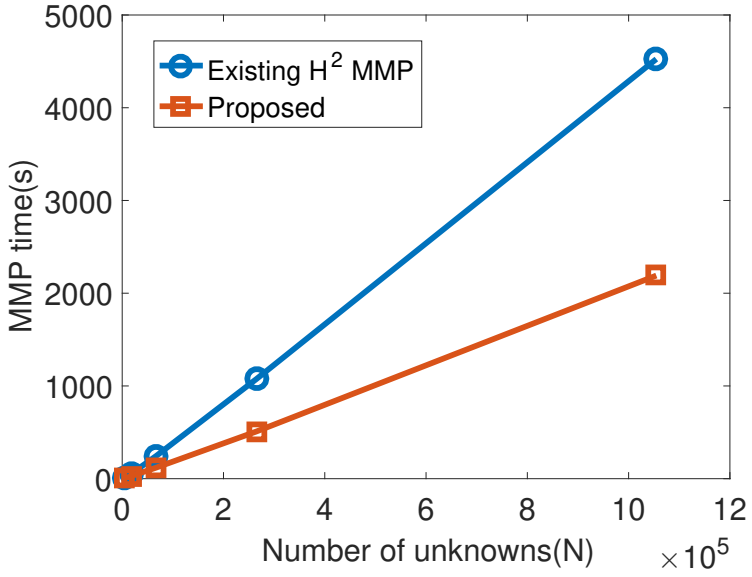


Fig. 2.5.: Structure-preserving MMP time comparison with existing MMP for capacitance matrix.

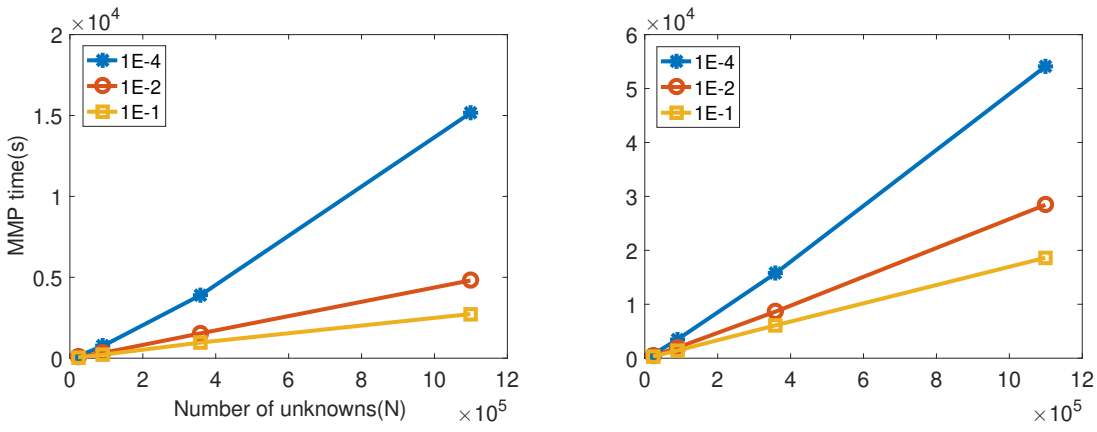
proposed one. So the memory usage is the same for these two methods, which is the original $\mathcal{H}^2$-matrix memory.

Table 2.2.: Structure-preserving MMP error comparison for the large-scale capacitance extraction.

N	4,480	17,152	67,072	265,216	1,054,720
Existing [7]	5.88E-3	1.11E-2	1.04E-2	2.01E-2	5.02E-2
Proposed	5.76E-3	2.75E-3	2.33E-3	4.38E-3	9.22E-3

Large-scale Dielectric Slab

We then simulate a dielectric slab with $\epsilon_r = 2.54$ at 300 MHz. The thickness of the slab is fixed to be $0.1\lambda_0$. The width and length are simultaneously increased from $4\lambda_0$, $8\lambda_0$, $16\lambda_0$, to $28\lambda_0$. With a mesh size of $0.1\lambda_0$, the resultant N ranges



(a) (b)

Fig. 2.6.: Structure-preserving MMP performance for $\mathbf{A}_{\mathcal{H}^2} \times \mathbf{A}_{\mathcal{H}^2}$ of 2-D slab from 4λ to 28λ. (a) Time scaling v.s. N. (b) Memory scaling v.s. N.

from 22,560 to 1,098,720 for this suite of slab structures.The parameters used in the $\mathcal{H}^2$-matrix construction are $leafsize = 40$, admissibility condition [7] $\eta = 2.0$, and $\epsilon_{\mathcal{H}^2} = 10^{-4}$. For the proposed $\mathcal{H}^2$ MMP, the updated cluster bases truncation ϵ_{add} are 10^{-1}, 10^{-2} and 10^{-4} respectively, to examine the computational complexity and error controllability of the proposed MMP. Based on [32], the rank's growth rate with electrical size for 2-D slab is lower than linear, and being a square root of he log-linear of the electric size. Substituting such a rank's growth into the complexity analysis in (2.23) and (2.24), we will obtain linear complexity in both memory and time.

In Fig. 2.6(a), we plot the structure-preserving MMP time with respect to N, for all three different choices of cluster bases truncation. It is clear that the smaller ϵ_{add} value, the larger the MMP time. However, the complexity remains the same as linear regardless of the choice of ϵ_{add}. The memory cost is plotted in Fig. 2.6(b). Obviously, it scales linearly with the number of unknowns. The error of the proposed structure-preserving MMP is measured the same by random right hand side error as shown in equation (2.27). In Table 2.3, we list the error ad a function of ϵ_{add}.

Excellent accuracy can be observed in the entire unknown range. Furthermore, the accuracy can be controlled by ϵ_{add}, and overall smaller ϵ_{add} results in better accuracy.

Table 2.3.: Structure-preserving MMP error at different ϵ_{add} for 2-D large-scale dielectric slab.

N	22,560	89,920	359,040	1,098,720
1E-1	5.66E-3	7.47E-3	1.14E-2	8.20E-3
1E-2	1.40E-3	1.67E-3	2.70E-3	2.81E-3
1E-4	1.54E-4	1.99E-4	2.69E-4	2.76E-4

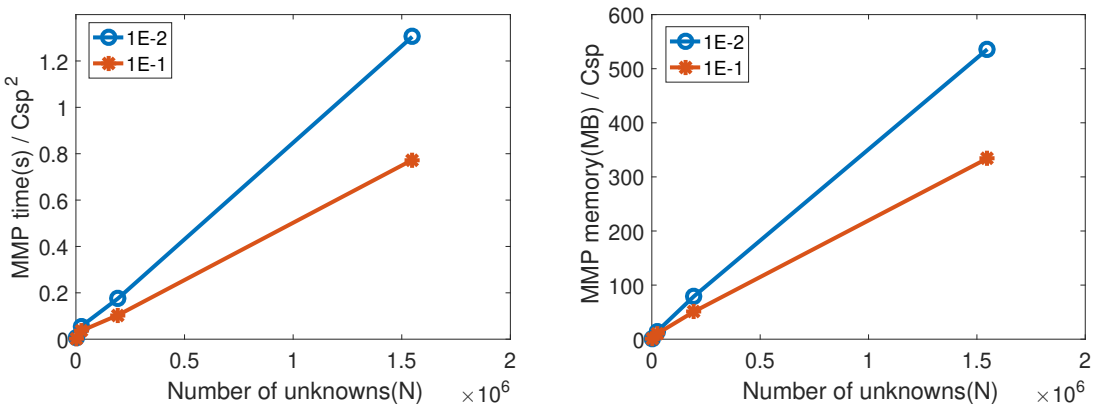
Large-scale Array of Dielectric Cubes

Next, we simulate a large-scale array of dielectric cubes at 300 MHz. The relative permittivity of the cube is $\epsilon_r = 4.0$. Each cube is of size $0.3\lambda_0 \times 0.3\lambda_0 \times 0.3\lambda_0$. The distance between adjacent cubes is kept to be $0.3\lambda_0$. The number of the cubes is increased along the $x-$, $y-$, and $z-$ directions simultaneously from 2 to 16, thus producing a 3-D cube array from $2 \times 2 \times 2$ to $16 \times 16 \times 16$ elements. The number of unknowns N is respectively 3,024, 24,192, 193,536, and 1,548,288 for these arrays. During the construction of $\mathcal{H}^2$ matrix, we set $leafsize = 20$, $\eta = 1$ and $\epsilon_{\mathcal{H}^2} = 10^{-2}$. For the proposed $\mathcal{H}^2$ MMP, the updated cluster bases are truncated at 10^{-1}, 10^{-2} due to limit of server memory.

Table 2.4.: C_{sp} as a function of N for the dielectric cube array.

N	3,024	24,192	193,536	653,184	1,548,288
C_{sp}	16	42	95	595	126

For cubic growth of unknowns for 3-D problems, we observe that constant C_{sp} is not saturated until in the order of millions of unknowns, as can be seen from Table



(a) (b)

Fig. 2.7.: Structure-preserving MMP performance for $\mathbf{A}_{\mathcal{H}^2} \times \mathbf{A}_{\mathcal{H}^2}$ of 3-D cube array. (a) Time scaling v.s. N. (b) Memory scaling v.s. N.

2.4. It is thus, important to analyze the performances of the proposed structure-preserving MMP as $Memory/C_{sp}$ and $Multiplicationtime/C_{sp}^2$ respectively. In Fig. 2.7 (a) and Fig. 2.7 (b), we plot the $\mathcal{H}^2$-matrix-matrix multiplication time divided by C_{sp}^2, and the storage cost normalized with C_{sp} with respect to N. As can be seen, their scaling rate with N agrees very well with our theoretical complexity analysis. For the over one-million unknown case which is a $16 \times 16 \times 16$ cube array having thousands of cube elements, the error is still controlled to be as small as 0.82% at $\epsilon_{add} = 10^{-1}$. The random RHS error of the proposed structure-preserving MMP is listed in Table 2.5 for this example, which again reveals excellent accuracy and error controllability of the proposed MMP. We also compare the time usage for proposed

Table 2.5.: Structure-preserving MMP error at different ϵ_{add} for 3-D dielectric cube array.

N	3,024	24,192	193,536	1,548,288
1E-1	5.66E-3	7.47E-3	1.14E-2	8.20E-3
1E-2	1.40E-3	1.67E-3	2.70E-3	2.81E-3

Table 2.6.: Structure-preserving MMP error comparison with existing MMP for 3-D cube array.

N	3,024	24,192	193,536	1,548,288
Existing [7]	9.02E-2	1.01E-1	1.77E-1	2.74E-1
Proposed	8.03E-2	7.39E-2	1.03E-1	1.25E-1

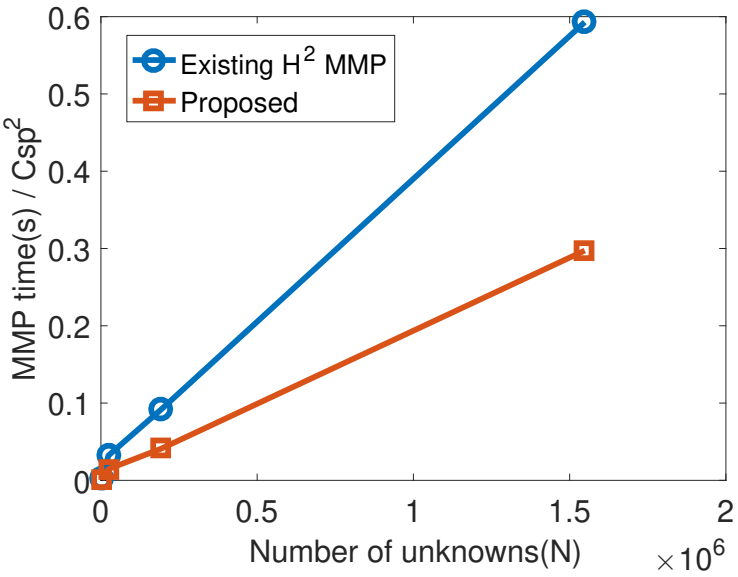


Fig. 2.8.: Structure-preserving MMP time comparison with existing MMP for 3-D cube array VIE matrix.

MMP with existing MMP [7] using this 3-D example. As shown in Fig. 2.8, the proposed structure-preserving MMP take about half time as compared to the existing one. But the proposed one has better accuracy even not updating cluster bases, i.e. setting $\epsilon_{add} = 0$. This is listed in Table 2.6.

2.3 $\mathcal{H}^2$-Matrix-Matrix Product without Formatted Multiplications

2.3.1 Proposed Alorithm

In an $\mathcal{H}^2$ matrix shown in chapter 1.3.2, the entire matrix is partitioned into inadmissible blocks at leaf level, noted as $\mathbf{F}$, and admissible blocks at multiple nonleaf levels, noted as $\mathbf{R}$. A large matrix block consisting of $\mathbf{F}$ and $\mathbf{R}$ is called a nonleaf block $\mathbf{NL}$. An admissible block has the following form $\mathbf{Z}_{t,s} = (\mathbf{V}_t)_{\#t\times r}(\mathbf{S}_{t,s})_{r\times r}(\mathbf{V}_s)^T_{\#s\times r}$. The cluster basis $\mathbf{V}$ in an $\mathcal{H}^2$ matrix is nested [7].

To compute $\mathbf{A}_{\mathcal{H}^2} \times \mathbf{B}_{\mathcal{H}^2} = \mathbf{C}$, unlike the existing $\mathcal{H}^2$ formatted MMP [7] that is recursive, we propose to perform a one-way tree traversal from leaf level to root level. While doing the multiplications at each level, we update the row and column cluster bases based on prescribed accuracy to represent the product accurately. The proposed algorithm also facilitates parallelization. We will use the $\mathcal{H}^2$ matrices shown in Fig. 2.9 to illustrate the algorithm, but the algorithm is valid for any $\mathcal{H}^2$ matrix. In this figure, green blocks are admissible, and red ones are inadmissible.

At leaf level ($l = L$), there are in total four matrix-matrix product cases, i.e., 1) $\mathbf{F} \times \mathbf{F} \to \mathbf{F}$; 2) $\mathbf{F} \times \mathbf{R} \to \mathbf{R}$; 3) $\mathbf{R} \times \mathbf{F} \to \mathbf{R}$; 4) $\mathbf{R} \times \mathbf{R} \to \mathbf{R}$. The computation of

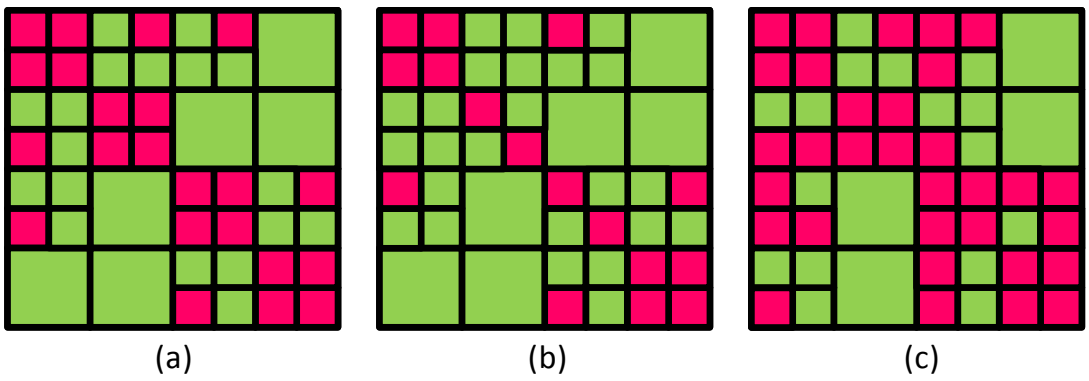


Fig. 2.9.: An $\mathcal{H}^2$-matrix structure. (a) $\mathbf{A}_{\mathcal{H}^2}$. (b) $\mathbf{B}_{\mathcal{H}^2}$. (c) $\mathbf{C}_{\mathcal{H}^2}$.

the first case is exact, like $\mathbf{F}^{\mathbf{A}}_{1,2} \times \mathbf{F}^{\mathbf{B}}_{2,1} = \mathbf{F}^{\mathbf{C}}_{1,1}$ in the example of Fig. 2.9. The fourth case is also exact because

$$\mathbf{R}^{\mathbf{A}}_{i,j} \times \mathbf{R}^{\mathbf{B}}_{j,k} = \mathbf{V}^{\mathbf{A}}_{i,r}\mathbf{S}^{\mathbf{A}}_{i,j}(\mathbf{V}^{\mathbf{A}}_{j,c})^T \times \mathbf{V}^{\mathbf{B}}_{j,r}\mathbf{S}^{\mathbf{B}}_{j,k}(\mathbf{V}^{\mathbf{B}}_{k,c})^T. \tag{2.28}$$

The target block is obviously admissible and it preserves the original row cluster basis $\mathbf{V}^{\mathbf{A}}_{i,r}$, and column cluster basis $\mathbf{V}^{\mathbf{B}}_{k,c}$. One only needs to calculate the coupling matrix, which is the center part, $\mathbf{S}^{\mathbf{A}}_{i,j}\widetilde{\mathbf{B}}_j\mathbf{S}^{\mathbf{B}}_{j,k}$, where $\widetilde{\mathbf{B}}_j$ is cluster bases product $(\mathbf{V}^{\mathbf{A}}_{j,c})^T \times \mathbf{V}^{\mathbf{B}}_{j,r}$, which can be computed in advance in linear complexity. For the second and third cases, the products are clearly low rank due to the low-rank property of admissible blocks. For the second case, we compute it as

$$\mathbf{F}^{\mathbf{A}}_{i,j} \times \mathbf{R}^{\mathbf{B}}_{j,k} = (\mathbf{F}^{\mathbf{A}}_{i,j}\mathbf{V}^{\mathbf{B}}_{j,r}) \times \mathbf{S}^{\mathbf{B}}_{j,k} \times (\mathbf{V}^{\mathbf{B}}_{k,c})^T. \tag{2.29}$$

Meanwhile, we also use $\mathbf{F}^{\mathbf{A}}_{i,j}\mathbf{V}^{\mathbf{B}}_{j,r}$ to update the row cluster basis $\mathbf{V}^{\mathbf{A}}_{i,r}$. This is because the row cluster basis of the product block is no longer $\mathbf{V}^{\mathbf{A}}_{i,r}$, if $\mathbf{F}^{\mathbf{A}}_{i,j}\mathbf{V}^{\mathbf{B}}_{j,r}$ contains new information not included in $\mathbf{V}^{\mathbf{A}}_{i,r}$. Since there are many case-2 products encountered at the leaf level, to systematically update the i's row cluster basis $\mathbf{V}^{\mathbf{A}}_{i,r}$, we first find all the dense matrices in i's row of $\mathbf{A}_{\mathcal{H}^2}$, whose number is bounded by constantC_{sp}. We then multiply them with corresponding row cluster basis $\mathbf{V}^{\mathbf{B}}_{j,r}$ in $\mathbf{B}_{\mathcal{H}^2}$, obtaining Gram matrix sum $\mathbf{V}^{\mathbf{A}}_{i,new} = \sum_{j=1}^{O(C_{sp})}(\mathbf{F}^{\mathbf{A}}_{i,j}\mathbf{V}^{\mathbf{B}}_{j,r})(\mathbf{F}^{\mathbf{A}}_{i,j}\mathbf{V}^{\mathbf{B}}_{j,r})^H$. We then use this to augment the cluster basis by computing $\mathbf{G}^{\mathbf{A}}_i = [\mathbf{I} - \mathbf{V}^{\mathbf{A}}_{i,r}(\mathbf{V}^{\mathbf{A}}_{i,r})^H] \times \mathbf{V}^{\mathbf{A}}_{i,new} \times [\mathbf{I} - \mathbf{V}^{\mathbf{A}}_{i,r}(\mathbf{V}^{\mathbf{A}}_{i,r})^H]^H$. After this, we perform an SVD on $\mathbf{G}^{\mathbf{A}}_i$ to obtain the additional cluster bases $\mathbf{V}^{\mathbf{A},add}_{i.r}$ for cluster i based on prescribed accuracy ϵ_{add}. Now the updated i's row cluster bases $\widetilde{\mathbf{V}}^{\mathbf{A}}_{i,r} = [\mathbf{V}^{\mathbf{A}}_{i,r}\ \mathbf{V}^{A,add}_{i,r}]$ can be used to accurately represent the admissible blocks in i's row of $\mathbf{C}_{\mathcal{H}^2}$. For the case-3 MMP at the leaf level, we compute it as $\mathbf{R}^{\mathbf{A}}_{i,j} \times \mathbf{F}^{\mathbf{B}}_{j,k} = \mathbf{V}^{\mathbf{A}}_{i,r} \times \mathbf{S}^{\mathbf{A}}_{i,j} \times [(\mathbf{V}^{\mathbf{A}}_{j,c})^T\mathbf{F}^{\mathbf{B}}_{j,k}]$. Meanwhile, we update j's column cluster basis in $\mathbf{B}_{\mathcal{H}^2}$ using $(\mathbf{V}^{\mathbf{A}}_{j,c})^T\mathbf{F}^{\mathbf{B}}_{j,k}$ in a similar way as updating row cluster bases to accurately represent the admissible blocks in j's column of $\mathbf{C}_{\mathcal{H}^2}$. In order to speed up leaf level multiplications, we can collect the $\mathbf{F}$ blocks in $\mathbf{A}_{\mathcal{H}^2}$ by $(\widetilde{\mathbf{V}}^{\mathbf{A}}_{i,r})^H\mathbf{F}^{\mathbf{A}}_{i,j}\mathbf{V}^{\mathbf{B}}_{j,r}$ and $\mathbf{F}$ blocks in $\mathbf{B}_{\mathcal{H}^2}$ by $(\mathbf{V}^{\mathbf{A}}_{j,c})^T\mathbf{F}^{\mathbf{B}}_{j,k}(\widetilde{\mathbf{V}}^{\mathbf{B}}_{j,c})^*$ in advance to facilitate case-2 products $\mathbf{F}^{\mathbf{A}}_{i,j} \times \mathbf{R}^{\mathbf{B}}_{j,k}$

for different k and case-3 products $\mathbf{R}^{\mathbf{A}}_{i,j} \times \mathbf{F}^{\mathbf{B}}_{j,k}$ for different i by computations of rank size. Note that this collect is accurate now because it is for products between $\mathbf{F}$ and $\mathbf{R}$, and the products' row and column cluster bases have been updated to account for $\mathbf{F} \times \mathbf{R}$ and $\mathbf{R} \times \mathbf{F}$ cases. Here, we conclude all the operations at the leaf level:

1. Prepare cluster bases product $\widetilde{\mathbf{B}}$;
2. Updating row and column cluster bases to $\widetilde{\mathbf{V}}$;
3. Collect the $\mathbf{F}$ blocks in $\mathbf{A}_{\mathcal{H}^2}$ and $\mathbf{B}_{\mathcal{H}^2}$ using $\widetilde{\mathbf{V}}$;
4. Perform four cases of multiplications.

At each non-leaf level, we perform three types of matrix products: 1) $\mathbf{NL} \times \mathbf{R} \rightarrow \mathbf{R}$; 2) $\mathbf{R} \times \mathbf{NL} \rightarrow \mathbf{R}$; and 3) $\mathbf{R} \times \mathbf{R} \rightarrow \mathbf{R}$. These products are performed as they are without assuming their target structure. Case-3 again does not require any update on the cluster bases. The $\mathbf{NL}$ block at level l is a merged matrix of the collected $\mathbf{F}$ matrices and coupling matrices at level $(l+1)$ so that the block is of rank size, whose expressions are as shown in equation (2.12) and (2.13). Such a merged $\mathbf{NL}$ block of rank size, denoted by $\widetilde{\mathbf{NL}}$, resembles $\mathbf{F}$ at the leaf level. Again, for case-1, the row cluster basis of the product needs to be updated. This is performed by using $\sum_{j=1}^{O(C_{sp})} (\widetilde{\mathbf{NL}}_{ij}\mathbf{T}_j)(\widetilde{\mathbf{NL}}_{ij}\mathbf{T}_j)^H$ to augment the transfer matrix of $\mathbf{T}_i$. So at a non-leaf level, we do:

1. Merge four small blocks to $\widetilde{\mathbf{NL}}_{i,j}$ for $\mathbf{A}_{\mathcal{H}^2}$ and $\mathbf{B}_{\mathcal{H}^2}$;
2. Updating row and column transfer matrices to $\widetilde{\mathbf{T}}$;
3. Collect the $\widetilde{\mathbf{NL}}$ blocks in $\mathbf{A}_{\mathcal{H}^2}$ and $\mathbf{B}_{\mathcal{H}^2}$ using updated $\widetilde{\mathbf{T}}$;
4. Perform three cases of multiplications.

After we finish one-way bottom-up tree traversal to calculate block matrix products at all the levels, we split the coupling matrices in the admissible blocks in the original $\mathbf{C}$ structure to the newly introduced admissible blocks at different levels to

get the final $\mathbf{C}_{\mathcal{H}^2}$, as shown in Fig. 2.9(c). At each tree level l, there are $2^l C_{sp}^2$ products to be computed, each of which costs $O(r_l^3)$, where r_l is the rank at level l. The resulting complexity is clearly $O(N)$ for constant rank or rank that changes logarithmically with electrical size. For electrodynamic analysis, the rank grows linearly with electrical size for general 3-D problems. In a VIE, r_l is proportional to the cubic root of matrix size at level l, because this is the electrical size at level l. Hence for a VIE, the time complexity is

$$\textbf{Time Complexity} = {C_{sp}}^2 \sum_{l=0}^{L} 2^l \left[\left(\frac{N}{2^l} \right)^{\frac{1}{3}} \right]^3 = O(NlogN). \tag{2.30}$$

And the memory complexity is

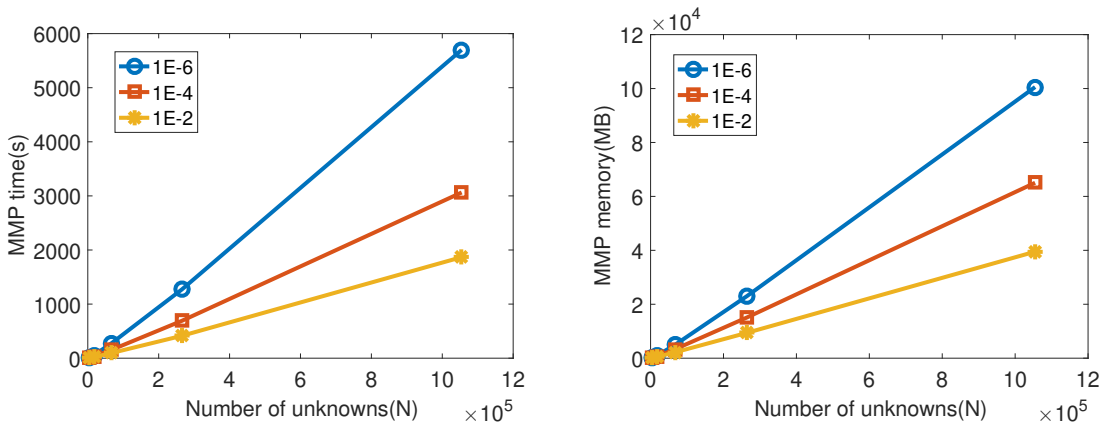
$$\textbf{Memory Complexity} = C_{sp} \sum_{l=0}^{L} 2^l \left[\left(\frac{N}{2^l} \right)^{\frac{1}{3}} \right]^2 = O(N). \tag{2.31}$$

2.3.2 Numerical Results

In order to demonstrate the accuracy and low computational complexity of the proposed fast $\mathcal{H}^2$-matrix-matrix multiplication for general $\mathcal{H}^2$ matrices, we use $\mathcal{H}^2$ matrices resulting from large-scale capacitance extraction and volume integral equations (VIE) for electromagnetic analysis. The capacitance extraction matrix is shown in [9]. The VIE formulation we use is based on [31] with SWG vector bases for expanding electric flux density in each tetrahedral element.

Two-layer Cross Bus

The first example is the capacitance extraction of a 2-layer cross bus structure. In each layer, there are m conductors, and each conductors has a dimension of $1 \times 1 \times (2m+1)$ m^3. We simulate a suite of such structures with 8, 16, 32, 64, and 128 buses respectively. The parameters used in the $\mathcal{H}^2$-matrix construction are $leafsize = 30$, admissibility condition [7] $\eta = 1$, and cluster bases truncation at 10^{-4}. For the proposed $\mathcal{H}^2$ MMP, the updated cluster bases are truncated at 10^{-2},10^{-4} and 10^{-6}.



(a) (b)

Fig. 2.10.: MMP performance for $\mathbf{A}_{\mathcal{H}^2} \times \mathbf{B}_{\mathcal{H}^2}$ of large-scale capacitance extraction. (a) Time scaling v.s. N. (b) Memory scaling v.s. N.

As shown in Fig. 2.10, the proposed MMP has linear time and memory performance for different truncation thresholds. The time and memory of the proposed MMP in comparison with existing $\mathcal{H}^2$ MMP [7] are also shown in Fig. 2.11, and the accuracy comparison is in Table 2.7.

Table 2.7.: $\mathcal{H}^2$ MMP error performance at different ϵ_{add} and comparison with existing MMP for large-scale capacitance extraction.

N	4,480	17,152	67,072	265,216	1,054,720
Existing $\mathcal{H}^2$ [7]	5.88E-3	1.11E-2	1.04E-2	2.01E-2	5.02E-2
Proposed at 10^{-2}	6.81E-4	6.16E-4	5.40E-4	3.95E-4	3.80E-4
Proposed at 10^{-4}	7.08E-5	8.89E-5	6.31E-5	1.09E-4	2.37E-4
Proposed at 10^{-6}	5.15E-6	5.41E-6	5.41E-6	5.14E-6	1.24E-5

Large-scale Dielectric Slab

We also simulate a suite of dielectric slabs at 300 MHz using VIE formulation, and $\lambda/10$ meshing rule. The simulation parameters are $leafsize = 40$, $\eta = 2$, 10^{-3}

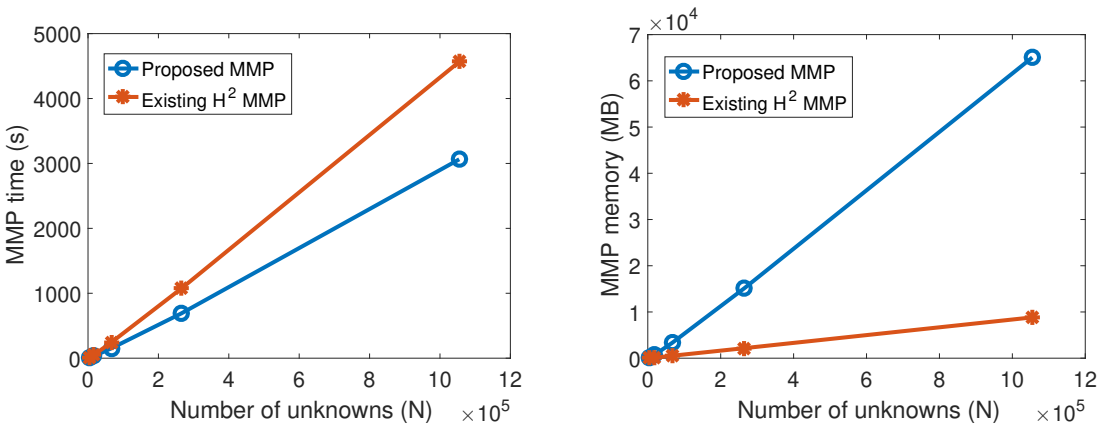


Fig. 2.11.: MMP performance for $\mathbf{A}_{\mathcal{H}^2} \times \mathbf{B}_{\mathcal{H}^2}$ of large-scale capacitance extraction and comparison with existing MMP. (a) Time scaling v.s. N. (b) Memory scaling v.s. N.

accuracy for generating the $\mathcal{H}^2$ matrix, 10^{-1} and 10^{-2} for updating cluster bases in MMP. The time and memory complexity is shown in Fig. 2.12, which is clearly linear. The MMP error is shown at Table 2.8 for the four unknown cases simulated.

Table 2.8.: $\mathcal{H}^2$ MMP error at different ϵ_{add} for 2-D dielectric slab.

N	22,560	89,920	359,040	1,098,720
1E-1	7.97E-3	9.30E-3	1.16E-2	1.03E-2
1E-2	2.81E-3	3.34E-3	5.97E-3	3.52E-3

Large-scale Array of Dielectric Cubes

We also simulate a large-scale array of dielectric cubes at 300 MHz. The relative permittivity of the cube is $\epsilon_r = 4.0$. Each cube is of size $0.3\lambda_0 \times 0.3\lambda_0 \times 0.3\lambda_0$. The distance between adjacent cubes is kept to be $0.3\lambda_0$. The number of the cubes is increased along the $x-$, $y-$, and $z-$ directions simultaneously from 2 to 16, thus producing a 3-D cube array from $2 \times 2 \times 2$ to $16 \times 16 \times 16$ elements. The number

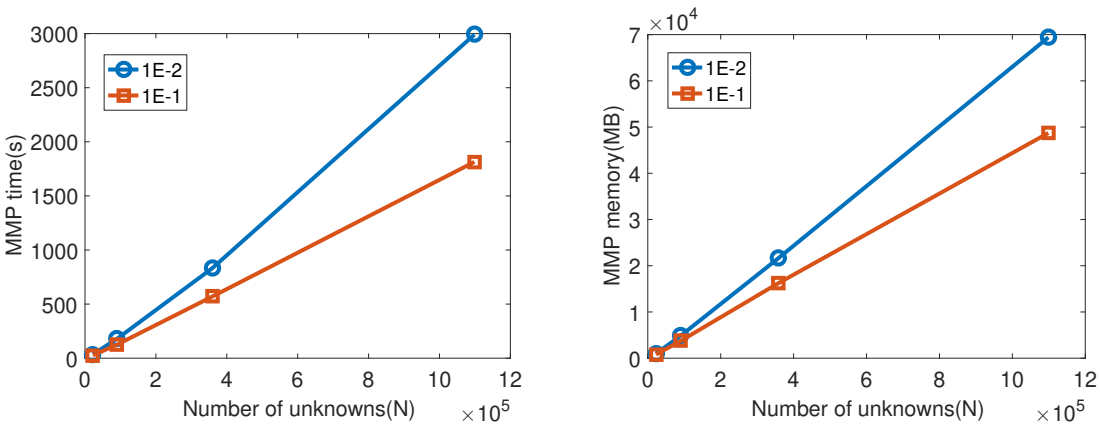


Fig. 2.12.: MMP performance for $\mathbf{A}_{\mathcal{H}^2} \times \mathbf{A}_{\mathcal{H}^2}$ of electrically large dielectric slab scattering from 8λ to 28λ. (a) Time scaling v.s. N. (b) Memory scaling v.s. N.

of unknowns N is respectively 3,024, 24,192, 193,536, and 1,548,288 for these arrays. During the construction of $\mathcal{H}^2$ matrix, we set $leafsize = 40$, $\eta = 2$ and $\epsilon_{\mathcal{H}^2} = 10^{-1}$. For the proposed $\mathcal{H}^2$ MMP, the updated cluster bases are truncated at 10^{-1}, 10^{-2} due to limit of server memory. The time and memory performance is shown in Fig. 2.13 and the errors are shown in Table 2.9.

Table 2.9.: $\mathcal{H}^2$ MMP error at different ϵ_{add} for 3-D dielectric cube array.

N	3,024	24,192	193,536	1,548,288
1E-1	2.25E-2	2.28E-2	3.23E-2	4.19E-2
1E-2	5.67E-3	5.30E-3	6.68E-2	1.42E-2

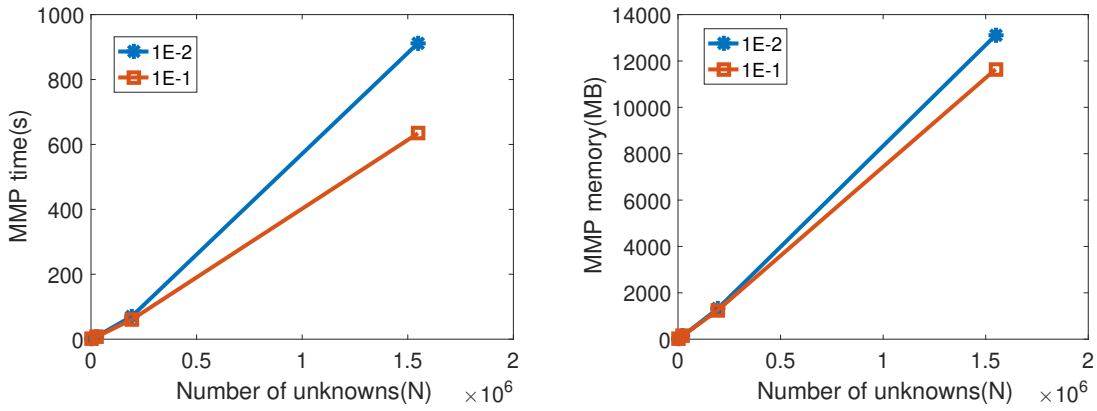


Fig. 2.13.: MMP performance for $\mathbf{A}_{\mathcal{H}^2} \times \mathbf{A}_{\mathcal{H}^2}$ of a suite of cube array from $2 \times 2 \times 2$ to $16 \times 16 \times 16$. (a) Time scaling v.s. N. (b) Memory scaling v.s. N.

2.4 Conclusion

In this chapter, we first develop a fast algorithm to calculate $\mathcal{H}^2$-matrix-matrix products for general $\mathcal{H}^2$ matrices. The key feature of this algorithm is that it preserves the $\mathcal{H}^2$-matrix structure and the accuracy of the product is *explicitly* controlled. We then introduce a fast algorithm to calculate $\mathcal{H}^2$-matrix-matrix products for general $\mathcal{H}^2$ matrices without assuming a target format. The cluster bases are updated based on the prescribed accuracy during the computation of the matrix-matrix product. The proposed algorithms have been applied to calculate $\mathcal{H}^2$-matrix-matrix products for large-scale capacitance extraction whose kernel is static and real-valued and electrically large VIEs whose kernel is oscillatory and complex-valued. For constant-rank $\mathcal{H}^2$ matrices, the proposed MMP algorithms have an $O(N)$ complexity in both time and memory. For rank growing with the electrical size linearly, the proposed MMPs have an $O(NlogN)$ complexity time and $O(N)$ complexity in memory. $\mathcal{H}^2$-matrix products with millions of unknowns are simulated on a single core CPU in fast CPU run time. Comparisons with existing $\mathcal{H}^2$-matrix-matrix products have demonstrated excellent accuracy and efficiency of the proposed new MMP algorithms.

3. RANK-MINIMIZED AND STRUCTURE-KEPT $\mathcal{H}^2$-MATRIX-MATRIX PRODUCT IN LINEAR COMPLEXITY WITH CONTROLLED ACCURACY

3.1 Introduction

Under the $\mathcal{H}^2$-matrix framework, it has been shown that an $\mathcal{H}^2$-matrix-based addition, matrix-vector product (MVP), and matrix-matrix product (MMP) all can be performed in linear complexity for constant-rank $\mathcal{H}^2$ [7]. However, the accuracy of existing $\mathcal{H}^2$-MMP algorithm like [7] is not directly controlled and the efficiency can be further improved for accuracy controlled MMP. The *exact* MMP introduced in chapter 7 of [4] overcomes the limitations of the formatted multiplications in [7] as we discussed in chapter 2. It could calculate the *exact* product of $\mathbf{A}_{\mathcal{H}^2}$ and $\mathbf{B}_{\mathcal{H}^2}$ without any approximation error. However, it can never encounter the situation that the target block $\mathbf{C}_{i,k}$ is admissible, but the $\mathbf{A}_{i,j}$ and $\mathbf{B}_{j,k}$ are inadmissible. Besides, the *exact* MMP count all the potential spaces together as product cluster bases $\mathbf{V}_t^{\mathbf{C}}$, which will lead to a very high computational complexity and further affect higher-level operators like LU factorization and inversion. The posteriori multiplication in chapter 8 of [4] combines the advantages of the obove two MMP algorithms, which is more accurate than the formatted multiplication in [7] and more efficient than the *exact* MMP in [4]. However, it needs to calculate intermediate product in $\mathcal{H}$-matrix format and then convert the $\mathcal{H}$-matrix to an $\mathcal{H}^2$-matrix, which needs much more computational time and memory during the process. The complexity is not linear for constant-rank $\mathcal{H}^2$. And it still need to avoid the situation that the target block $\mathbf{C}_{i,k}$ is admissible, but the $\mathbf{A}_{i,j}$ and $\mathbf{B}_{j,k}$ are inadmissible, which is not for general $\mathcal{H}^2$-matries.

Recently, in [33, 34], a new linear complexity algorithm is developed to compute the product of two $\mathcal{H}^2$ matrices in controlled accuracy. The original cluster bases are changed by appending new cluster bases to account for the updates to the original matrix generated during MMP. The product structure can be preserved as in [34], or can be changed based on the structure of the two multipliers as in [33]. However, the rank of the changed cluster bases by appending new ones can only be higher than the original rank.

In this work, we propose a new algorithm to do the $\mathcal{H}^2$ matrix-matrix multiplication with controlled accuracy. The cluster bases are calculated based on the prescribed accuracy during the computation of the matrix-matrix product. Meanwhile, we are able to keep the computational complexity to be linear for constant-rank $\mathcal{H}^2$. For variable-rank cases such as those for electrically large analysis, the proposed MMP is also efficient since it only involves $O(2^l)$ computations at level l, each of which costs $O(r_l^3)$ only, where r_l is the rank at tree level l. This algorithm serves as a fundamental arithmetic in the error-controlled fast inverse, LU factorization, solution for many right hand sides, etc. Numerical experiments have demonstrated its accuracy and low complexity. Compared to the MMP algorithm developed in chapter 2, the algorithm in this chapter is more efficient in memory. Meanwhile, it provides more flexibility in developing error-controlled fast inverse, LU factorization, solution for many right hand sides, etc. Numerical experiments have demonstrated its accuracy and low complexity.

3.2 Proposed $\mathcal{H}^2$ Matrix-Matrix Product Algorithm—Leaf Level

To compute $\mathbf{A}_{\mathcal{H}^2} \times \mathbf{B}_{\mathcal{H}^2} = \mathbf{C}_{\mathcal{H}^2}$, unlike the existing $\mathcal{H}^2$ formatted MMP [7], which is recursive, we propose to perform a one-way tree traversal from leaf level all the way up to the level that has largest admissible block. Here we call it minimal admissible level. While doing the multiplications at each level, we calculate the row and column cluster bases based on prescribed accuracy to represent the products accurately. We will use

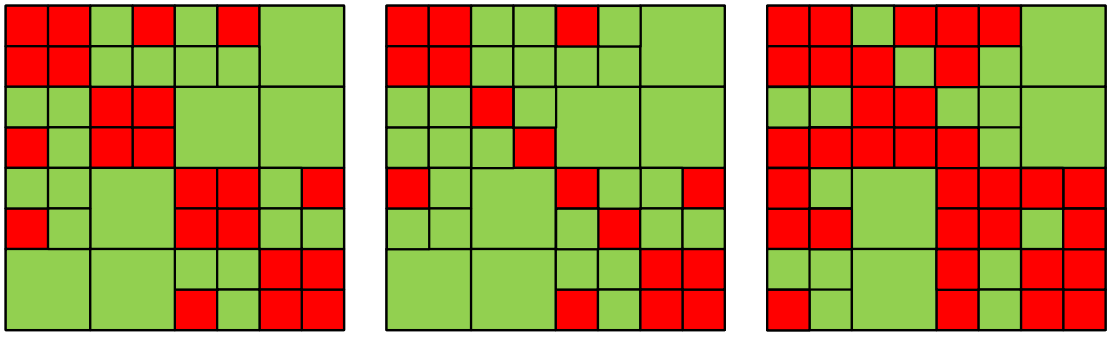


Fig. 3.1.: An $\mathcal{H}^2$-matrix structure. (a) $\mathbf{A}_{\mathcal{H}^2}$. (b) $\mathbf{B}_{\mathcal{H}^2}$. (c) $\mathbf{C}_{\mathcal{H}^2}$.

the $\mathcal{H}^2$ matrices shown in Fig. 3.1 to illustrate the algorithm, but the algorithm is valid for any $\mathcal{H}^2$ matrix. In this figure, green blocks are admissible, and red ones are inadmissible. Here the structures of $\mathbf{A}_{\mathcal{H}^2}$, $\mathbf{B}_{\mathcal{H}^2}$ and $\mathbf{C}_{\mathcal{H}^2}$ matrices are based on the admissibility condition in equation (1.13). During the product calculation, the following algorithm that we introduced here will keep the structure of product $\mathbf{C}_{\mathcal{H}^2}$ matrix. However, the algorithm can easily modified to another version where the the product structure will be changed due to the two multipliers as shown in [33].

In this section, we detail proposed algorithm for leaf-level multiplications.

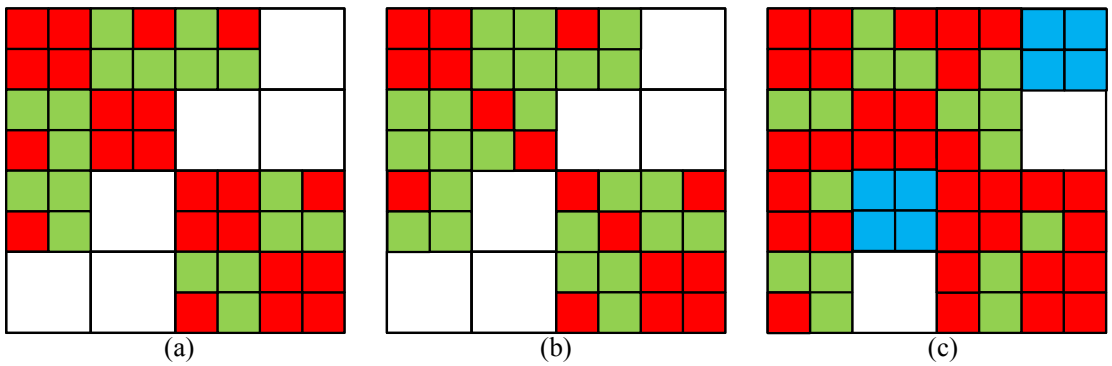


Fig. 3.2.: $\mathcal{H}^2$ matrix at leaf level. (a) $\mathbf{A}^L_{\mathcal{H}^2}$. (b) $\mathbf{B}^L_{\mathcal{H}^2}$. (c) $\mathbf{C}^L_{\mathcal{H}^2}$.

We start from leaf level ($l = L$). Let $\mathbf{F}$ denote an inadmissible block, which is stored as a full matrix, and $\mathbf{R}$ be an admissible block. At leaf level, there are in total four matrix-matrix multiplication cases, i.e.,

- Case-1: $\mathbf{F^A} \times \mathbf{F^B}$
- Case-2: $\mathbf{F^A} \times \mathbf{R^B}$
- Case-3: $\mathbf{R^A} \times \mathbf{F^B}$
- Case-4: $\mathbf{R^A} \times \mathbf{R^B}$

The resulting matrix block in $\mathbf{C}$ is of two kinds: First, full matrix block, denoted by $\mathbf{F^C}$, marked in red in Fig. 3.2(c); Second, admissible block of leaf size, which could be located at leaf level, denoted by $\mathbf{R}^{\mathbf{C},L}$ as marked in green in Fig. 3.2(c); which could also appear as a subblock in the non-leaf level l as marked in blue in Fig. 3.2(c). The blue blocks in Fig. 3.2(c) are only for temporary storage, which will be changed to green admissible blocks during the upper level multiplication to preserve the structure of $\mathbf{C}_{\mathcal{H}^2}$ matrix. The white blocks in Fig. 3.2 denote those blocks that are not involved in the leaf level multiplication. Next we show how to perform each matrix-matrix multiplication based on the two kinds of target blocks.

3.2.1 Product is an inadmissible block (full matrix) in C

If the product matrix is a full block $\mathbf{F^C}$, we can perform the four cases of multiplications exactly as they are by full matrix multiplications. For the admissible leaf blocks in four cases, we convert them into full matrices and then compute products. Since the size of these matrices is of $leafsize$, a user-defined constant, the computational cost is constant for each of such computations.

3.2.2 Product is an admissible block in C

If the product is admissible in $\mathbf{C}$ whether it is a leaf-level block or a subblock of a non-leaf admissible block, case-4 can be performed as it is since the product matrix is obviously admissible, which also preserves the original row and column cluster bases. In other words, the row cluster basis of $\mathbf{A}$ is that of $\mathbf{C}$; and the column cluster basis of $\mathbf{B}$ is kept in $\mathbf{C}$. To see this point clearly, we can write

$$\textbf{case-4: } \mathbf{R}^{\mathbf{A}}_{i,j} \times \mathbf{R}^{\mathbf{B}}_{j,k} = \mathbf{V}^{\mathbf{A}}_{i_r} \mathbf{S}^{\mathbf{A}}_{i,j} (\mathbf{V}^{\mathbf{A}}_{j_c})^T \times \mathbf{V}^{\mathbf{B}}_{j_r} \mathbf{S}^{\mathbf{B}}_{j,k} (\mathbf{V}^{\mathbf{B}}_{k_c})^T, \tag{3.1}$$

where subscripts i, j, and k denote cluster index, subscript r denotes the corresponding cluster is a row cluster, whereas c denotes the cluster is a column cluster. For example, $\mathbf{V}^{\mathbf{A}}_{i_r}$ denotes the cluster basis of row cluster i in $\mathbf{A}$, and $\mathbf{V}^{\mathbf{B}}_{k_c}$ denotes the cluster basis of column cluster k in $\mathbf{B}$. Equation (3.1) can be written in short as

$$\mathbf{R}^{\mathbf{A}}_{i,j} \times \mathbf{R}^{\mathbf{B}}_{j,k} = \mathbf{V}^{\mathbf{A}}_{i_r} \mathbf{S}^{\mathbf{C}}_{i,k} (\mathbf{V}^{\mathbf{B}}_{k_c})^T, \tag{3.2}$$

in which $\mathbf{S}^{\mathbf{C}}_{i,k}$ is the part in between the two cluster bases, which denotes the coupling matrix of the product admissible block in $\mathbf{C}$. Clearly, this case of multiplication does not change the original row and column cluster bases.

For the other three cases, in existing MMP algorithms, a formatted multiplication is performed, which is done in the same way as case-4, i.e., using the original cluster bases of $\mathbf{A}$ and $\mathbf{B}$ or pre-assumed bases as the cluster bases of the product block. This obviously can be inaccurate since cases-1, 2, and 3, if performed as they are, would result in different cluster bases in the product matrix, which cannot be assumed. Specifically, case-1 results in a different row as well as column cluster bases in the product admissible block because

$$\textbf{case-1: } \mathbf{F}^{\mathbf{A}}_{i,j} \times \mathbf{F}^{\mathbf{B}}_{j,k}; \tag{3.3}$$

case-2 yields a different row cluster basis since

$$\textbf{case-2: } \mathbf{F}^{\mathbf{A}}_{i,j} \times \mathbf{R}^{\mathbf{B}}_{j,k} = (\mathbf{F}^{\mathbf{A}}_{i,j} \mathbf{V}^{\mathbf{B}}_{j_r}) \times \mathbf{S}^{\mathbf{B}}_{j,k} \times (\mathbf{V}^{\mathbf{B}}_{k_c})^T; \tag{3.4}$$

whereas case-3 results in a different column cluster basis in the product admissible block, because

$$\textbf{case-3: } \mathbf{R}^{\mathbf{A}}_{i,j} \times \mathbf{F}^{\mathbf{B}}_{j,k} = \mathbf{V}^{\mathbf{A}}_{i_r} \times \mathbf{S}^{\mathbf{A}}_{i,j} \times \left((\mathbf{V}^{\mathbf{A}}_{j_c})^T \mathbf{F}^{\mathbf{B}}_{j,k} \right). \tag{3.5}$$

If we do not update the cluster bases in the product matrix, the accuracy of the multiplication is not controllable. Therefore, in the proposed algorithm, we update row and column cluster bases for multiplication cases 1, 2, and 3 based on prescribed accuracy. We also have to do so with the nested property taken into consideration so that the computation at nonleaf levels can be performed efficiently.

For case-1, both row and column cluster bases of the product block need to be updated. For case-2, we need to use $\mathbf{F}^{\mathbf{A}}_{i,j}\mathbf{V}^{\mathbf{B}}_{j_r}$ to update the original row cluster basis $\mathbf{V}^{\mathbf{A}}_{i_r}$. For case-3, we need to use $(\mathbf{V}^{\mathbf{A}}_{j_c})^T\mathbf{F}^{\mathbf{B}}_{j,k}$ to update column cluster basis $\mathbf{V}^{\mathbf{B}}_{k_c}$. Since there are many case-1, 2 and 3 products encountered at the leaf level for the same row or column cluster, we develop the following algorithm to systematically update the cluster bases. In this procedure, **we also have to take the computation at all nonleaf levels into consideration so that the changed cluster bases at the leaf level can be reused at the nonleaf levels.** To achieve this goal, when we update the cluster basis due to the case-1, 2, and 3 multiplications associated with this cluster, not only we consider the product admissible block in the leaf level, but also the admissible blocks at all nonleaf levels. In other words, when computing $\mathbf{A}_{i,j}$ multiplied by $\mathbf{B}_{j,k}$, if the $\mathbf{C}_{i,k}$ block is part of a non-leaf admissible block, we will take the corresponding multiplication into account to update the cluster bases. The detailed algorithms are as follows.

3.2.3 Computation of new cluster bases in matrix product $\mathbf{C}_{\mathcal{H}^2}$

First, we show how to calculate the new row cluster bases of $\mathbf{C}_{\mathcal{H}^2}$. Take an arbitrary row cluster i as an example, let its cluster basis in $\mathbf{C}$ be denoted by $\mathbf{V}^{\mathbf{C}}_{i_r}$. This cluster basis is affected by both case-1 and case-2 multiplications, as analyzed in the above. We first find all the case-1 multiplications associated with cluster i,

i.e., all $\mathbf{F}^{\mathbf{A}}_{i,j} \times \mathbf{F}^{\mathbf{B}}_{j,k}$ whose product block $\mathbf{C}_{i,k}$ is admissible. Again, notice that the $\mathbf{C}_{i,k}$ can be either admissible at leaf level or be part of a non-leaf admissible block. For any cluster i, the number of $\mathbf{F}^{\mathbf{A}}_{i,j}$ is bounded by constant C_{sp}, since the number of inadmissible blocks that can be formed by a cluster is bounded by C_{sp}. For the same reason, the number of $\mathbf{F}^{\mathbf{B}}_{j,k}$ for cluster j is also bounded by constant C_{sp}. Hence, the total number of $\mathbf{F}^{\mathbf{A}}_{i,j} \times \mathbf{F}^{\mathbf{B}}_{j,k}$ multiplications is bounded by C^2_{sp}, thus also a constant. Then we calculate the Gram matrix sum of these products as:

$$\mathbf{G}^{\mathbf{C},L}_{i_{r1}} = \sum_{j=1}^{O(C_{sp})} \sum_{k=1}^{O(C_{sp})} (\mathbf{F}^{\mathbf{A}}_{i,j}\mathbf{F}^{\mathbf{B}}_{j,k})(\mathbf{F}^{\mathbf{A}}_{i,j}\mathbf{F}^{\mathbf{B}}_{j,k})^H, \tag{3.6}$$

in which superscript H denotes a Hermitian matrix. We also find all case-2 products associated with cluster i, which is the number of $\mathbf{F}^{\mathbf{A}}_{i,j}$ formed by cluster i at leaf level in $\mathbf{A}_{\mathcal{H}^2}$. This is also bounded by C_{sp}. Since in case-2 products, $\mathbf{F}^{\mathbf{A}}_{i,j}$ is multiplied by an admissible block in $\mathbf{B}$, and hence $\mathbf{V}^{\mathbf{B}}_{j_r}$, we compute

$$\mathbf{G}^{\mathbf{C},L}_{i_{r2}} = \sum_{j=1}^{O(C_{sp})} (\mathbf{F}^{\mathbf{A}}_{i,j}\mathbf{V}^{\mathbf{B}}_{j_r})(\mathbf{F}^{\mathbf{A}}_{i,j}\mathbf{V}^{\mathbf{B}}_{j_r})^H, \tag{3.7}$$

which incorporates all of the new cluster bases information due to case-2 products.

For case-3 and case-4 multiplications, the row cluster bases of $\mathbf{A}_{\mathcal{H}^2}$ matrix are kept to be those of $\mathbf{C}$. So we account for the contribution of $\mathbf{V}^{\mathbf{A}}_{i_r}$ as

$$\mathbf{G}^{\mathbf{C},L}_{i_{r3}} = \mathbf{V}^{\mathbf{A}}_{i_r}(\mathbf{V}^{\mathbf{A}}_{i_r})^H. \tag{3.8}$$

The column space spanning $\mathbf{G}^{\mathbf{C},L}_{i_{r1}}$, $\mathbf{G}^{\mathbf{C},L}_{i_{r2}}$ and $\mathbf{G}^{\mathbf{C},L}_{i_{r3}}$ would be the new cluster basis of i, since it takes both the original cluster basis and the change to the cluster basis due to matrix products into consideration. Since the magnitude of the three matrices may differ greatly, we normalize them before summing them up so that each component is captured. We thus obtain

$$\mathbf{G}^{\mathbf{C},L}_{i_{r3}} = \widehat{\mathbf{G}^{\mathbf{C},L}_{i_{r1}}} + \widehat{\mathbf{G}^{\mathbf{C},L}_{i_{r2}}} + \widehat{\mathbf{G}^{\mathbf{C},L}_{i_{r3}}}. \tag{3.9}$$

The $\widehat{\ }$ above $\mathbf{G}^{\mathbf{C},L}_{i_{r1}}$, $\mathbf{G}^{\mathbf{C},L}_{i_{r2}}$ and $\mathbf{G}^{\mathbf{C},L}_{i_{r3}}$ denotes a normalized matrix. We then perform an SVD on $\mathbf{G}^{\mathbf{C},L}_{i_{r3}}$ to obtain the row cluster bases for cluster i of $\mathbf{C}_{\mathcal{H}^2}$ based on prescribed

accuracy ϵ_{trunc}. The singular vectors whose normalized singular values are greater than ϵ_{trunc} make the new row cluster basis $\mathbf{V}^{\mathbf{C}}_{i_r}$. It can be used to accurately represent the admissible blocks related to cluster i in $\mathbf{C}_{\mathcal{H}^2}$. Here, notice that the proposed algorithm for computing matrix-product cluster bases keeps nested property of $\mathbf{V}^{\mathbf{C}}_{i_r}$. This is because the Gram matrix sums in (3.6), (3.7) and (3.8) take the upper level admissible products into account.

To compute the column cluster bases in $\mathbf{C}_{\mathcal{H}^2}$, the steps are similar to the row cluster basis computation. We account for the contributions from all the four cases of products to compute column cluster bases. As can be seen from (3.3) and (3.5), in case-1 and case-3 products, the column cluster bases are changed from the original ones; whereas in case-2 and case-4 products, the column cluster bases are kept the same as those in $\mathbf{B}$.

Consider an arbitrary column cluster k in $\mathbf{C}_{\mathcal{H}^2}$. We find all of the case-1 products associated with k, which is $\mathbf{F}^{\mathbf{A}}_{i,j} \times \mathbf{F}^{\mathbf{B}}_{j,k}$ with target $\mathbf{C}_{i,k}$ being admissible either at the leaf or non-leaf level. The number of such multiplications is bounded by C_{sp}^2. We then compute the sum of their Gram matrices as:

$$\mathbf{G}^{\mathbf{C},L}_{k_{c1}} = \sum_{i=1}^{O(C_{sp})} \sum_{j=1}^{O(C_{sp})} (\mathbf{F}^{\mathbf{A}}_{i,j}\mathbf{F}^{\mathbf{B}}_{j,k})^T (\mathbf{F}^{\mathbf{A}}_{i,j}\mathbf{F}^{\mathbf{B}}_{j,k})^*. \tag{3.10}$$

Here, the superscript * denotes a complex conjugate. We also find all of the case-3 products associated with k, which is $\mathbf{R}^{\mathbf{A}}_{i,j} \times \mathbf{F}^{\mathbf{B}}_{j,k}$ with target $\mathbf{C}_{i,k}$ being admissible either at the leaf or non-leaf level. Hence, the new column cluster basis takes a form of $(\mathbf{V}^{\mathbf{A}}_{j_c})^T \times \mathbf{F}^{\mathbf{B}}_{j,k}$. The number of such multiplications is also bounded by C_{sp}. The sum of their Gram matrices can be computed as:

$$\mathbf{G}^{\mathbf{C},L}_{k_{c2}} = \sum_{j=1}^{O(C_{sp})} \left((\mathbf{V}^{\mathbf{A}}_{j_c})^T \mathbf{F}^{\mathbf{B}}_{j,k}\right)^T \left((\mathbf{V}^{\mathbf{A}}_{j_c})^T \mathbf{F}^{\mathbf{B}}_{j,k}\right)^*. \tag{3.11}$$

For case-2 and case-4 products, the original column cluster bases of $\mathbf{B}_{\mathcal{H}^2}$ are kept in $\mathbf{C}_{\mathcal{H}^2}$, hence, we compute

$$\mathbf{G}^{\mathbf{C},L}_{k_{c3}} = \mathbf{V}^{\mathbf{B}}_{k_c} (\mathbf{V}^{\mathbf{B}}_{k_c})^H. \tag{3.12}$$

We also normalize these three Gram matrices $\mathbf{G}^{\mathbf{C},L}_{k_{c1}}$, $\mathbf{G}^{\mathbf{C},L}_{k_{c2}}$ and $\mathbf{G}^{\mathbf{C},L}_{k_{c3}}$ and sum them up as:

$$\mathbf{G}^{\mathbf{C},L}_{k_c} = \widehat{\mathbf{G}^{\mathbf{C},L}_{k_{c1}}} + \widehat{\mathbf{G}^{\mathbf{C},L}_{k_{c2}}} + \widehat{\mathbf{G}^{\mathbf{C},L}_{k_{c3}}}. \tag{3.13}$$

We then perform an SVD on this $\mathbf{G}^{\mathbf{C},L}_{k_c}$ and truncate the singular values based on prescribed accuracy ϵ_{trunc} to obtain the column cluster bases $\mathbf{V}^{\mathbf{C}}_{k_c}$ for cluster k. Now this new column cluster basis $\mathbf{V}^{\mathbf{C}}_{k_c}$ can be used to accurately represent the admissible blocks formed by column cluster k in $\mathbf{C}_{\mathcal{H}^2}$.

3.2.4 Computation of the four cases of multiplications with the product block being admissible

After computing the new row and column cluster bases of the product matrix, for the multiplication whose target is an admissible block described in Section 3.2.2, the computation becomes the coupling matrix computation since the cluster bases have been generated. For the four cases of multiplications, their coupling matrices have the following expressions:

$$\mathbf{S}^{\mathbf{C}}_{i,k} = \begin{cases} (\mathbf{V}^{\mathbf{C}}_{i_r})^H \mathbf{F}^{\mathbf{A}}_{i,j} \mathbf{F}^{\mathbf{B}}_{j,k} (\mathbf{V}^{\mathbf{C}}_{k_c})^* & \text{case-1} \\ (\mathbf{V}^{\mathbf{C}}_{i_r})^H \mathbf{F}^{\mathbf{A}}_{i,j} \mathbf{V}^{\mathbf{B}}_{j_r} \mathbf{S}^{\mathbf{B}}_{j,k} (\mathbf{V}^{\mathbf{B}}_{k_c})^T (\mathbf{V}^{\mathbf{C}}_{k_c})^* & \text{case-2} \\ (\mathbf{V}^{\mathbf{C}}_{i_r})^H \mathbf{V}^{\mathbf{A}}_{i_r} \mathbf{S}^{\mathbf{A}}_{i,j} (\mathbf{V}^{\mathbf{A}}_{j_c})^T \mathbf{F}^{\mathbf{B}}_{j,k} (\mathbf{V}^{\mathbf{C}}_{k_c})^* & \text{case-3} \\ (\mathbf{V}^{\mathbf{C}}_{i_r})^H \mathbf{V}^{\mathbf{A}}_{i_r} \mathbf{S}^{\mathbf{A}}_{i,j} \mathbf{B}_j \mathbf{S}^{\mathbf{B}}_{j,k} (\mathbf{V}^{\mathbf{B}}_{k_c})^T (\mathbf{V}^{\mathbf{C}}_{k_c})^* & \text{case-4.} \end{cases} \tag{3.14}$$

The resulting admissible blocks in $\mathbf{C}_{\mathcal{H}^2}$ are nothing but $\mathbf{R}^{\mathbf{C}}_{i,k} = \mathbf{V}^{\mathbf{C}}_{i_r} \times \mathbf{S}^{\mathbf{C}}_{i,k} \times (\mathbf{V}^{\mathbf{C}}_{k_c})^T$.

In (3.14), the $\mathbf{B}_j$ is the cluster bases product, which is as shown below:

$$\mathbf{B}_j = (\mathbf{V}^{\mathbf{A}}_{j_c})^T \times \mathbf{V}^{\mathbf{B}}_{j_r}. \tag{3.15}$$

Since it is only related to the original cluster bases, it can be prepared in advance before the MMP computation. Using the nested property of the cluster bases, $\mathbf{B}_j$ can be computed in linear time for all clusters j, be j a leaf or a non-leaf cluster.

In (3.14), the $(\mathbf{V}^{\mathbf{C}}_{i_r})^H \mathbf{V}^{\mathbf{A}}_{i_r}$ is simply the projection of the original row cluster basis of $\mathbf{A}$ onto the new cluster basis of the product matrix $\mathbf{C}$. Similarly, $(\mathbf{V}^{\mathbf{B}}_{k_c})^T (\mathbf{V}^{\mathbf{C}}_{k_c})^*$ denotes

the projection of the original column cluster basis of $\mathbf{B}$ onto the newly generated column cluster basis in $\mathbf{C}$. The two cluster basis projections can also be computed for every leaf cluster after the new cluster bases have been generated. Hence, we compute

$$\begin{aligned} \mathbf{P}_i^{\mathbf{A}} &= (\mathbf{V}_{i_r}^{\mathbf{C}})^H \mathbf{V}_{i_r}^{\mathbf{A}}, \\ \mathbf{P}_k^{\mathbf{B}} &= (\mathbf{V}_{k_c}^{\mathbf{B}})^T (\mathbf{V}_{k_c}^{\mathbf{C}})^* \end{aligned} \tag{3.16}$$

for each leaf row cluster i, and each column leaf cluster k. In this way, it can be reused without recomputation for each admissible block formed by i or k.

In (3.14), we can also see that the $\mathbf{F}$ block is front and back multiplied by cluster bases. It can be viewed as an $\mathbf{F}$ block collected based on the front (row) and back (column) cluster bases, which becomes a matrix of rank size. Specifically, in (3.14), there are three kinds of collected blocks

$$\begin{aligned} (\mathbf{F}_{i,j}^{\mathbf{A}} \mathbf{F}_{j,k}^{\mathbf{B}})_{coll.} &= (\mathbf{V}_{i_r}^{\mathbf{C}})^H (\mathbf{F}_{i,j}^{\mathbf{A}} \mathbf{F}_{j,k}^{\mathbf{B}}) \left(\mathbf{V}_{k_c}^{\mathbf{C}}\right)^* \\ (\mathbf{F}_{i,j}^{\mathbf{A}})_{coll.} &= (\mathbf{V}_{i_r}^{\mathbf{C}})^H \mathbf{F}_{i,j}^{\mathbf{A}} \mathbf{V}_{j_r}^{\mathbf{B}} \\ (\mathbf{F}_{j,k}^{\mathbf{B}})_{coll.} &= (\mathbf{V}_{j_c}^{\mathbf{A}})^T \mathbf{F}_{j,k}^{\mathbf{B}} \left(\mathbf{V}_{k_c}^{\mathbf{C}}\right)^*, \end{aligned} \tag{3.17}$$

which is used in case-1, 2, and 3 multiplication respectively.

As can be seen from (3.14), the case-1 multiplication with an admissible block being the target can be performed by first computing the full-matrix product, and then collecting the product onto the new row and column cluster bases of the product matrix. This collect operation is accurate because the newly generated row and column cluster bases have taken such a case-1 multiplication into consideration when being generated. As for the case-2 multiplication, as can be seen from (3.14), we can use the $\mathbf{F}_{i,j}$ collected based on the new row cluster basis and the original column cluster basis, the size of which is rank, to multiply the coupling matrix of $\mathbf{S}_{j,k}$, and then multiply the column basis projection matrix since the column bases have been changed. Similarly, for case-3, we use the collected block $(\mathbf{F}_{j,k}^{\mathbf{B}})_{coll.}$, and front multiply it by the coupling matrix of $\mathbf{S}_{i,j}$, and then front multiply a row cluster basis transformation matrix. As for case-4, we multiply the coupling matrix of $\mathbf{A}$'s admissible block by the cluster basis product, and then by the coupling matrix of $\mathbf{B}$'s admissible block.

Since the row and column cluster bases have been changed to account for the other cases of multiplications, at the end, we need to front and back multiply the cluster basis transformation matrices to complete the computation of case-4. Summarizing the aforementioned, the coupling matrix in (3.14) can be efficiently computed as

$$\mathbf{S}^{\mathbf{C}}_{i,k} = \begin{cases} (\mathbf{V}^{\mathbf{C}}_{i_r})^H \mathbf{F}^{\mathbf{A}}_{i,j} \mathbf{F}^{\mathbf{B}}_{j,k} (\mathbf{V}^{\mathbf{C}}_{k_c})^* & \text{case-1} \\ (\mathbf{F}^{\mathbf{A}}_{i,j})_{coll.} \mathbf{S}^{\mathbf{B}}_{j,k} \mathbf{P}^{\mathbf{B}}_{k} & \text{case-2} \\ \mathbf{P}^{\mathbf{A}}_{i} \mathbf{S}^{\mathbf{A}}_{i,j} (\mathbf{F}^{\mathbf{B}}_{j,k})_{coll.} & \text{case-3} \\ \mathbf{P}^{\mathbf{A}}_{i} \mathbf{S}^{\mathbf{A}}_{i,j} \mathbf{B}_j \mathbf{S}^{\mathbf{B}}_{j,k} \mathbf{P}^{\mathbf{B}}_{k} & \text{case-4.} \end{cases} \tag{3.18}$$

3.2.5 Summary of overall algorithm at leaf level

Here, we conclude all the operations related to leaf level computation when the target is an admissible block:

1. Prepare cluster bases product $\mathbf{B}$;
2. Compute all the leaf-level row and column cluster bases of product matrix $\mathbf{C}_{\mathcal{H}^2}$;
3. Collect the $\mathbf{F}$ blocks in $\mathbf{A}_{\mathcal{H}^2}$ and $\mathbf{B}_{\mathcal{H}^2}$ based on the new row and/or column cluster bases, also prepare cluster bases transformation matrix $\mathbf{P}$;
4. Perform four cases of multiplications.

After leaf level multiplications, we need to merge four coupling matrices at a non-leaf level admissible block, as shown by the blue blocks in Fig. 3.2 (c). These matrices correspond to the multiplication case of a nonleaf block **NL** multiplied by a nonleaf block **NL** generating an admissible block at next level. The merged block is the coupling matrix of this next-level admissible block. It will be used to update next level transfer matrices. The details will be given in next section.

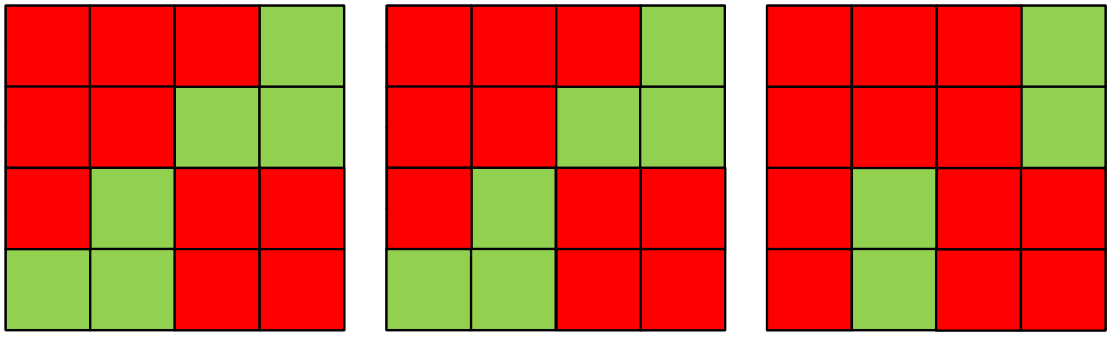


Fig. 3.3.: $\mathcal{H}^2$-matrix block at non-leaf level $(L-1)$. (a) $\mathbf{A}_{\mathcal{H}^2}^{L-1}$. (b) $\mathbf{B}_{\mathcal{H}^2}^{L-1}$. (c) $\mathbf{C}_{\mathcal{H}^2}^{L-1}$.

3.3 Proposed $\mathcal{H}^2$ Matrix-Matrix Product Algorithm—Non-Leaf Level

After finishing the leaf level multiplication, we proceed to non-leaf level multiplications. In Fig. 3.3, we use level $L-1$ as an example to illustrate $\mathbf{A}_{\mathcal{H}^2}^{L-1}$, $\mathbf{B}_{\mathcal{H}^2}^{L-1}$, and $\mathbf{C}_{\mathcal{H}^2}^{L-1}$.

At a nonleaf level l, there are also in total four matrix-matrix multiplication cases, i.e.,

- Case-1: $\mathbf{NL^A} \times \mathbf{NL^B}$
- Case-2: $\mathbf{NL^A} \times \mathbf{R^B}$
- Case-3: $\mathbf{R^A} \times \mathbf{NL^B}$
- Case-4: $\mathbf{R^A} \times \mathbf{R^B}$,

where **NL** denotes a non-leaf block. The resulting matrix block in **C** is also of two kinds: 1) non-leaf block **NL** at this level, marked in red in Fig. 3.3 (c), and 2) admissible block **R**, marked in green in Fig. 3.3 (c). Next we show how to perform each case of multiplications based on the two kinds of target blocks.

3.3.1 Product is an NL block in C

The **NL** target block would not exist for a case-1 multiplication, since if a case-1 multiplication results in an **NL** block, that computation should have been performed at previous level. As for the other three cases of multiplications, since at least one admissible block is present in the multipliers, the product must be an admissible block. Hence, we compute them as having an admissible block as the product, using the algorithm described in the following subsection, and associate the resulting admissible block with the **NL** block. After the computation is done at all levels, we perform a backward split operation to split the admissible block associated with each **NL** block to each leaf block of **C** based on its structure.

3.3.2 Product is an admissible block in C

Similar to the leaf level, if the product is an admissible block in **C** whether at the same non-leaf level or at an upper level, case-4 can be performed as it is since the product matrix is obviously admissible, which also preserves the original row and column cluster bases. We can write

$$\textbf{case-4: } \mathbf{R}^{\mathbf{A}}_{i,j} \times \mathbf{R}^{\mathbf{B}}_{j,k} = \mathbf{V}^{\mathbf{A}}_{i_r^{ch}} \mathbf{T}^{\mathbf{A}}_{i_r} \mathbf{S}^{\mathbf{A}}_{i,j} (\mathbf{T}^{\mathbf{A}}_{j_c})^T (\mathbf{V}^{\mathbf{A}}_{j_c^{ch}})^T \times \mathbf{V}^{\mathbf{B}}_{j_r^{ch}} \mathbf{T}^{\mathbf{B}}_{j_r} \mathbf{S}^{\mathbf{B}}_{j,k} (\mathbf{T}^{\mathbf{B}}_{k_c})^T (\mathbf{V}^{\mathbf{B}}_{k_c^{ch}})^T, \tag{3.19}$$

where $\mathbf{T}$ denotes a transfer matrix, and superscript ch denotes the two children clusters of the non-leaf cluster i. If the cluster bases at leaf level and the transfer matrices at non-leaf levels are kept the same as before, then the computation of (3.19) is to calculate the coupling matrix at level l, which is

$$\mathbf{S}^{\mathbf{C}}_{i,k} = \mathbf{S}^{\mathbf{A}}_{i,j}(\mathbf{B}_j)\mathbf{S}^{\mathbf{B}}_{j,k}. \tag{3.20}$$

It is a product of three small matrices whose size is the rank at this tree level. Rewriting (3.19) as

$$\textbf{case-4: } \mathbf{R}^{\mathbf{A}}_{i,j} \times \mathbf{R}^{\mathbf{B}}_{j,k} = \mathbf{V}^{\mathbf{A}}_{i_r^{ch}} \mathbf{T}^{\mathbf{A}}_{i_r} \mathbf{S}^{\mathbf{C}}_{i,k} (\mathbf{T}^{\mathbf{B}}_{k_c})^T (\mathbf{V}^{\mathbf{B}}_{k_c^{ch}})^T. \tag{3.21}$$

If we exclude the children cluster bases in the front and at the back, we can see that $\mathbf{T}$ serves as the new cluster basis at this level. In other words, at a non-leaf level l, if we treat this level as the bottom level of the remaining tree, then the transfer matrix of the non-leaf cluster is nothing but the leaf cluster basis of the shortened tree.

Similar to the leaf-level computation, the other three cases of multiplications will result in a change of cluster basis in the matrix product. Specifically, case-1 results in a different row as well as column cluster bases in the product admissible block because

$$\textbf{case-1: } \mathbf{NL}^{\mathbf{A}}_{i,j} \times \mathbf{NL}^{\mathbf{B}}_{j,k}; \tag{3.22}$$

case-2 yields a different row cluster basis since

$$\textbf{case-2: } \mathbf{NL}^{\mathbf{A}}_{i,j} \times \mathbf{R}^{\mathbf{B}}_{j,k} = (\mathbf{NL}^{\mathbf{A}}_{i,j} \mathbf{V}^{\mathbf{B}}_{j_r}) \times \mathbf{S}^{\mathbf{B}}_{j,k} \times (\mathbf{V}^{\mathbf{B}}_{k_c})^T; \tag{3.23}$$

whereas case-3 results in a different column cluster basis in the product admissible block, because

$$\textbf{case-3: } \mathbf{R}^{\mathbf{A}}_{i,j} \times \mathbf{NL}^{\mathbf{B}}_{j,k} = \mathbf{V}^{\mathbf{A}}_{i_r} \times \mathbf{S}^{\mathbf{A}}_{i,j} \times \left((\mathbf{V}^{\mathbf{A}}_{j_c})^T \mathbf{NL}^{\mathbf{B}}_{j,k} \right). \tag{3.24}$$

If we do not update the cluster bases in the product matrix, the accuracy of the multiplication is not controllable. However, if we update the cluster basis as they are, it is computationally very expensive since the matrix block size keeps increasing when we proceed from leaf level towards the root level. In addition to the cost of changing cluster bases, if we have to carry out the multiplications at each non-leaf level using the actual matrix block size, then the computation is also prohibitive. Therefore, the fast algorithm we develop here is to perform all computations using the rank size at each tree level, and meanwhile control the accuracy.

In the proposed algorithm, to account for the updates to the original matrix during the MMP procedure, the cluster bases of $\mathbf{C}$ are computed level by level, which are manifested by the changed leaf cluster bases and the transfer matrices at nonleaf levels. At a non-leaf level, its children-level cluster bases have already been computed, and they are different from the original ones in $\mathbf{A}$ and $\mathbf{B}$. However,

the new cluster bases have taken the upper-level multiplications into consideration. Hence, we can accurately represent the multiplication at the current non-leaf level using newly generated children cluster bases.

Take case-1 product as an example, where we perform $\mathbf{NL}^{\mathbf{A}}_{i,j} \times \mathbf{NL}^{\mathbf{B}}_{j,k}$ obtaining an admissible $\mathbf{R}^{\mathbf{C}}_{i,k}$.We can accurately represent this product using the children cluster bases of i and k as follows:

$$\textbf{case-1: } \mathbf{NL}^{\mathbf{A}}_{i,j} \times \mathbf{NL}^{\mathbf{B}}_{j,k} = \begin{bmatrix} \mathbf{V}^{\mathbf{C}}_{i1_r} & \\ & \mathbf{V}^{\mathbf{C}}_{i2_r} \end{bmatrix} (\mathbf{NL}^{\mathbf{A}}_{i,j}\mathbf{NL}^{\mathbf{B}}_{j,k})_{coll.} \begin{bmatrix} (\mathbf{V}^{\mathbf{C}}_{k1_c})^T & \\ & (\mathbf{V}^{\mathbf{C}}_{k2_c})^T \end{bmatrix}, \tag{3.25}$$

in which

$$(\mathbf{NL}^{\mathbf{A}}_{i,j}\mathbf{NL}^{\mathbf{B}}_{j,k})_{coll.} = \begin{bmatrix} (\mathbf{V}^{\mathbf{C}}_{i1_r})^H & \\ & (\mathbf{V}^{\mathbf{C}}_{i2_r})^H \end{bmatrix} \left(\mathbf{NL}^{\mathbf{A}}_{i,j}\mathbf{NL}^{\mathbf{B}}_{j,k}\right) \begin{bmatrix} (\mathbf{V}^{\mathbf{C}}_{k1_c})^* & \\ & (\mathbf{V}^{\mathbf{C}}_{k2_c})^* \end{bmatrix}. \tag{3.26}$$

This collected block, $(\mathbf{NL}^{\mathbf{A}}_{i,j}\mathbf{NL}^{\mathbf{B}}_{j,k})_{coll.}$, is actually the coupling matrix merged from the four small coupling matrices computed at previous level, when dealing with the multiplication case of having a target block as a subblock in the upper-level admissible block. It can be written as

$$(\mathbf{NL}^{\mathbf{A}}_{i,j}\mathbf{NL}^{\mathbf{B}}_{j,k})_{coll.} = \begin{bmatrix} \mathbf{S}^{\mathbf{C}}_{i_1,k_1} & \mathbf{S}^{\mathbf{C}}_{i_1,k_2} \\ \mathbf{S}^{\mathbf{C}}_{i_2,k_1} & \mathbf{S}^{\mathbf{C}}_{i_2,k_2} \end{bmatrix}. \tag{3.27}$$

Each of the four coupling matrices has been obtained at previous level. From (3.26), it is clear that using the nested property of the cluster bases, the collect operation does not need to start from leaf level, but using the four blocks obtained at previous one level.

For case-2 product, it can also be accurately expanded in the space of the children row cluster bases, and hence

$$\textbf{case-2: } \mathbf{NL}^{\mathbf{A}}_{i,j} \times \mathbf{R}^{\mathbf{B}}_{j,k} = \begin{bmatrix} \mathbf{V}^{\mathbf{C}}_{i1_r} & \\ & \mathbf{V}^{\mathbf{C}}_{i2_r} \end{bmatrix} \mathbf{NL}^{\mathbf{A}}_{i,j\,coll.} \mathbf{T}^{\mathbf{B}}_{j_r} \mathbf{S}^{\mathbf{B}}_{j,k} (\mathbf{V}^{\mathbf{B}}_{k_c})^T, \tag{3.28}$$

where

$$\mathbf{NL}^{\mathbf{A}}_{i,j_{coll.}} = \begin{bmatrix} (\mathbf{V}^{\mathbf{C}}_{i1_r})^H & \\ & (\mathbf{V}^{\mathbf{C}}_{i2_r})^H \end{bmatrix} \mathbf{NL}^{\mathbf{A}}_{i,j} \begin{bmatrix} \mathbf{V}^{\mathbf{B}}_{j1_r} & \\ & \mathbf{V}^{\mathbf{B}}_{j2_r} \end{bmatrix}, \tag{3.29}$$

which is $\mathbf{NL}^{\mathbf{A}}_{i,j}$ collected based on the children's new row cluster bases in $\mathbf{C}$ and the original column cluster bases in $\mathbf{B}$. From (3.28), it can be seen that if excluding the children cluster bases, then $\mathbf{NL}^{\mathbf{A}}_{i,j_{coll.}}\mathbf{T}^{\mathbf{B}}_{j_r}$ resembles the $\mathbf{F}^{\mathbf{A}}_{i,j}\mathbf{V}^{\mathbf{B}}_{j_r}$ in the leaf level case-2 product. In other words, if we treat the current non-leaf level as the leaf level, then $\mathbf{NL}^{\mathbf{A}}_{i,j_{coll.}}$ is equivalent to a full matrix block, whereas $\mathbf{T}$ is the leaf cluster basis. An example of $\mathbf{NL}^{\mathbf{A}}_{i,j_{coll.}}$ block at level $(L-1)$ in $\mathbf{A}_{\mathcal{H}^2}$ can be seen below:

$$(\mathbf{NL}^{\mathbf{A}}_{i,j})_{coll.} = \begin{bmatrix} (\mathbf{F}^{\mathbf{A}}_{i_1,j_1})_{coll.} & \mathbf{P}^{\mathbf{A}}_{i_1}\mathbf{S}^{\mathbf{A}}_{i_1,j_2}\mathbf{B}_{j_2} \\ \mathbf{P}^{\mathbf{A}}_{i_2}\mathbf{S}^{\mathbf{A}}_{i_2,j_1}\mathbf{B}_{j_1} & (\mathbf{F}^{\mathbf{A}}_{i_2,j_2})_{coll.} \end{bmatrix}, \tag{3.30}$$

which consists of collected full matrices whose expressions are shown in (3.17), and projected coupling matrices of admissible blocks. Again, using the nested property of both new and original cluster bases, the collect operation does not need to start from leaf level, but using the four blocks obtained at previous one level. Each collect operation only costs $O(k_l)^3$, where k_l is the rank at level l.

Since the cluster bases at the previous level have been computed, for case-1 and case-2 products at a non-leaf level, we only need to compute the center block associated with the current non-leaf level, and this computation can be carried out in the same way as how we carry out leaf-level computation, if we treat the current non-leaf level as the leaf level of the remaining tree. The same is true to case-3 product, where we have

$$\textbf{case-3: } \mathbf{R}^{\mathbf{A}}_{i,j} \times \mathbf{NL}^{\mathbf{B}}_{j,k} = \mathbf{V}^{\mathbf{A}}_{i_r}\mathbf{S}^{\mathbf{A}}_{i,j}(\mathbf{T}^{\mathbf{A}}_{j_c})^T\mathbf{NL}^{\mathbf{B}}_{j,k_{coll.}} \begin{bmatrix} (\mathbf{V}^{\mathbf{C}}_{k1_c})^T & \\ & (\mathbf{V}^{\mathbf{C}}_{k2_c})^T \end{bmatrix}, \tag{3.31}$$

in which

$$\mathbf{NL}^{\mathbf{B}}_{j,k_{coll.}} = \begin{bmatrix} (\mathbf{V}^{\mathbf{A}}_{j1_r})^T & \\ & (\mathbf{V}^{\mathbf{A}}_{j2_r})^T \end{bmatrix} \mathbf{NL}^{\mathbf{B}}_{j,k} \begin{bmatrix} (\mathbf{V}^{\mathbf{C}}_{k1_c})^* & \\ & (\mathbf{V}^{\mathbf{C}}_{k2_c})^* \end{bmatrix}. \tag{3.32}$$

We can see that $(\mathbf{T}^{\mathbf{A}}_{j_c})^T \mathbf{NL}^{\mathbf{B}}_{j,k_{coll.}}$ resembles the $(\mathbf{V}^{\mathbf{A}}_{j_c})^T \mathbf{F}^{\mathbf{B}}_{j,k}$ in the leaf level case-3 product. An example of collected **NL** block in $\mathbf{B}_{\mathcal{H}^2}$ is given as follows

$$(\mathbf{NL}^{\mathbf{B}}_{i,j})_{coll.} = \begin{bmatrix} (\mathbf{F}^{\mathbf{B}}_{i_1,j_1})_{coll.} & \mathbf{B}_{i_1}\mathbf{S}^{\mathbf{B}}_{i_1,j_2}\mathbf{P}^{\mathbf{B}}_{j_2} \\ \mathbf{B}_{i_2}\mathbf{S}^{\mathbf{B}}_{i_2,j_1}\mathbf{P}^{\mathbf{B}}_{j_1} & (\mathbf{F}^{\mathbf{B}}_{i_2,j_2})_{coll.} \end{bmatrix}, \tag{3.33}$$

which consists of collected full matrices whose expressions are shown in (3.17), and projected coupling matrices of admissible blocks.

Since the cluster bases have been changed at previous level, we also represent the case-4 product using the new children cluster bases of i and k, thus

$$\textbf{case-4: } \mathbf{R}^{\mathbf{A}}_{i,j} \times \mathbf{R}^{\mathbf{B}}_{j,k} = \begin{bmatrix} (\mathbf{V}^{\mathbf{C}}_{i1_r}) & \\ & (\mathbf{V}^{\mathbf{C}}_{i2_r}) \end{bmatrix} \mathbf{R}^{\mathbf{C}}_{i,k,proj} \begin{bmatrix} (\mathbf{V}^{\mathbf{C}}_{k1_c})^T & \\ & (\mathbf{V}^{\mathbf{C}}_{k2_c})^T \end{bmatrix}, \tag{3.34}$$

and

$$\mathbf{R}^{\mathbf{C}}_{i,k,proj} = \begin{bmatrix} (\mathbf{P}^{\mathbf{A}}_{i1}) & \\ & (\mathbf{P}^{\mathbf{A}}_{i2}) \end{bmatrix} (\mathbf{T}^{\mathbf{A}}_{i_r}\mathbf{S}^{\mathbf{C}}_{i,k}(\mathbf{T}^{\mathbf{B}}_{k_c})^T) \begin{bmatrix} (\mathbf{P}^{\mathbf{B}}_{k1}) & \\ & (\mathbf{P}^{\mathbf{B}}_{k2}) \end{bmatrix}, \tag{3.35}$$

which can be written in short as

$$\mathbf{R}^{\mathbf{C}}_{i,k,proj} = \mathbf{P}^{\mathbf{A}}_{i^{ch}}(\mathbf{T}^{\mathbf{A}}_{i_r}\mathbf{S}^{\mathbf{C}}_{i,k}(\mathbf{T}^{\mathbf{B}}_{k_c})^T)\mathbf{P}^{\mathbf{B}}_{k^{ch}}, \tag{3.36}$$

where *ch* denotes children. Here, there is a cluster basis transformation matrix in the front and at the back.

3.3.3 Computation of the new non-leaf level transfer matrices in C

If the target block is an admissible block at a nonleaf level, we need to represent it as $\mathbf{R}_{t,s} = \mathbf{T}_t\mathbf{S}_{t,s}(\mathbf{T}_s)^T$ in controlled accuracy. Hence, we need to calculate new row and column transfer matrices $\mathbf{T}$ of product matrix $\mathbf{C}_{\mathcal{H}^2}$. First, we introduce how to calculate the row transfer matrices. Similar to leaf level, case-1 and 2 products result in a change in the row cluster basis and hence row transfer matrix. Case-3 and 4 products do not require a change of transfer matrix if the cluster bases have not been changed at previous level. However, since the cluster bases have been changed at previous level, the transfer matrix requires an update as well.

For an arbitrary non-leaf cluster i, we first find all of the case-1 products associated with i. Each of such a product leads to a coupling matrix merged from the four coupling matrices obtained at previous level computation, denoted by $(\mathbf{NL}^{\mathbf{A}}_{i,j}\mathbf{NL}^{\mathbf{B}}_{j,k})_{coll.}$. Using them, we calculate the Gram matrix sum as:

$$\mathbf{G}^{\mathbf{C},l}_{i_{r1}} = \sum_{\#(i,k)=1}^{O(C_{sp}^2)} (\mathbf{NL}^{\mathbf{A}}_{i,j}\mathbf{NL}^{\mathbf{B}}_{j,k})_{coll.}((\mathbf{NL}^{\mathbf{A}}_{i,j}\mathbf{NL}^{\mathbf{B}}_{j,k})_{coll.})^H. \tag{3.37}$$

The second step is to take case-2 multiplications at a non-leaf level into consideration for row transfer matrix calculation of product matrix $\mathbf{C}_{\mathcal{H}^2}$. We find all the collected nonleaf blocks $\mathbf{NL}^{\mathbf{A}}_{i,j_{coll.}}$ of cluster i at level l in $\mathbf{A}_{\mathcal{H}^2}$ matrix and multiply them with corresponding transfer matrices $\mathbf{T}^{\mathbf{B}}_{j_r}$ from $\mathbf{B}_{\mathcal{H}^2}$ matrix. And we calculate the Gram matrix sum as

$$\mathbf{G}^{\mathbf{C},l}_{i_{r2}} = \sum_{j=1}^{O(C_{sp})} ((\mathbf{NL}^{\mathbf{A}}_{i,j})_{coll.}\mathbf{T}^{\mathbf{B}}_{j_r})((\mathbf{NL}^{\mathbf{A}}_{i,j})_{coll.}\mathbf{T}^{\mathbf{B}}_{j_r})^H. \tag{3.38}$$

Finally, we count the contributions from case-3 and case-4 products by computing

$$\mathbf{G}^{\mathbf{C},l}_{i_{r3}} = \mathbf{P}^{\mathbf{A}}_{i^{ch}}\mathbf{T}^{\mathbf{A}}_{i_r}(\mathbf{T}^{\mathbf{A}}_{i_r})^H(\mathbf{P}^{\mathbf{A}}_{i^{ch}})^H. \tag{3.39}$$

Again, we normalize these three Gram matrices and obtain

$$\mathbf{G}^{\mathbf{C},l}_{i_r} = \widehat{\mathbf{G}^{\mathbf{C},l}_{i_{r1}}} + \widehat{\mathbf{G}^{\mathbf{C},l}_{i_{r2}}} + \widehat{\mathbf{G}^{\mathbf{C},l}_{i_{r3}}}. \tag{3.40}$$

We then calculate an SVD of this $\mathbf{G}^{\mathbf{C},l}_{i_r}$ and truncate the singular values based on prescribed accuracy ϵ_{trunc} to obtain row transfer matrix $\mathbf{T}^{\mathbf{C}}_{i_r}$ for cluster i at nonleaf level.

Similarly, we can compute the new column transfer matrices for non-leaf cluster k, which is $\mathbf{T}^{\mathbf{C}}_{k_c}$. The first part is

$$\mathbf{G}^{\mathbf{C},l}_{k_{c1}} = \sum_{i=1}^{O(C_{sp})} ((\mathbf{NL}^{\mathbf{A}}_{i,j}\mathbf{NL}^{\mathbf{B}}_{j,k})_{coll.})^T((\mathbf{NL}^{\mathbf{A}}_{i,j}\mathbf{NL}^{\mathbf{B}}_{j,k})_{coll.})^*. \tag{3.41}$$

The second part is

$$\mathbf{G}^{\mathbf{C},l}_{k_{c2}} = \sum_{j=1}^{O(C_{sp})} ((\mathbf{T}^{\mathbf{A}}_{j_c})^T(\mathbf{NL}^{\mathbf{B}}_{j,k})_{coll.})^T((\mathbf{T}^{\mathbf{A}}_{j_c})^T(\mathbf{NL}^{\mathbf{B}}_{j,k})_{coll.})^*. \tag{3.42}$$

The third part is

$$\mathbf{G}_{k_{c3}}^{\mathbf{C},l} = (\mathbf{P}_{k^{ch}}^{\mathbf{B}})^T(\mathbf{T}_{k_c}^{\mathbf{B}})^*(\mathbf{T}_{k_c}^{\mathbf{B}})^T(\mathbf{P}_{k^{ch}}^{\mathbf{B}})^*. \tag{3.43}$$

Then we normalize the three Gram matrices and sum them up as

$$\mathbf{G}_{k_c}^{\mathbf{C},l} = \widehat{\mathbf{G}_{k_{c1}}^{\mathbf{C},l}} + \widehat{\mathbf{G}_{k_{c2}}^{\mathbf{C},l}} + \widehat{\mathbf{G}_{k_{c3}}^{\mathbf{C},l}}. \tag{3.44}$$

After we perform an SVD on $\mathbf{G}_{k_c}^{\mathbf{C},l}$ matrix and truncate the singular values based on prescribed accuracy ϵ_{trunc}, we get new column transfer matrix $\mathbf{T}_{k_c}^{\mathbf{C}}$.

3.3.4 Computation of the four cases of multiplications with the product block being admissible

Now we obtain both row and column transfer matrices for product matrix $\mathbf{C}_{\mathcal{H}^2}$, hence, the four multiplications become the computation of the coupling matrices, so that the admissible block at the current level has a form of $\mathbf{R}_{t,s} = \mathbf{T}_t\mathbf{S}_{t,s}(\mathbf{T}_s)^T$. The coupling matrix $\mathbf{S}$'s calculation is similar to that of leaf level in (3.14), which has the following expressions:

$$\mathbf{S}_{i,k}^{\mathbf{C}} = \begin{cases} (\mathbf{T}_{i_r}^{\mathbf{C}})^H(\mathbf{NL}_{i,j}^{\mathbf{A}}\mathbf{NL}_{j,k}^{\mathbf{B}})_{coll.}(\mathbf{T}_{k_c}^{\mathbf{C}})^* & \text{case-1} \\ (\mathbf{T}_{i_r}^{\mathbf{C}})^H(\mathbf{NL}_{i,j}^{\mathbf{A}})_{coll.}\mathbf{T}_{j_r}^{\mathbf{B}}\mathbf{S}_{j,k}^{\mathbf{B}}(\mathbf{V}_{k_c}^{\mathbf{B}})^T(\mathbf{V}_{k_c}^{\mathbf{C}})^* & \text{case-2} \\ (\mathbf{V}_{i_r}^{\mathbf{C}})^H\mathbf{V}_{i_r}^{\mathbf{A}}\mathbf{S}_{i,j}^{\mathbf{A}}(\mathbf{T}_{j_c}^{\mathbf{A}})^T(\mathbf{NL}_{j,k}^{\mathbf{B}})_{coll.}(\mathbf{T}_{k_c}^{\mathbf{C}})^* & \text{case-3} \\ (\mathbf{V}_{i_r}^{\mathbf{C}})^H\mathbf{V}_{i_r}^{\mathbf{A}}\mathbf{S}_{i,j}^{\mathbf{A}}\mathbf{B}_j\mathbf{S}_{j,k}^{\mathbf{B}}(\mathbf{V}_{k_c}^{\mathbf{B}})^T(\mathbf{V}_{k_c}^{\mathbf{C}})^* & \text{case-4.} \end{cases} \tag{3.45}$$

Again, we should prepare some matrix products in advance so that we can achieve linear complexity MMP for constant rank $\mathcal{H}^2$-matrix. For nonleaf levels, the cluster bases product $\mathbf{B}_j$ can be readily calculated using children's cluster bases based on the nested property. For example, given a nonleaf cluster j, we can generate $\mathbf{B}_j$ by using the cluster bases product of its children clusters j_1 and j_2, which is shown as:

$$\mathbf{B}_j = (\mathbf{T}_{j_{1_c}}^{\mathbf{A}})^T\mathbf{B}_{j_1}\mathbf{T}_{j_{1_r}}^{\mathbf{B}} + (\mathbf{T}_{j_{2_c}}^{\mathbf{A}})^T\mathbf{B}_{j_2}\mathbf{T}_{j_{2_r}}^{\mathbf{B}}. \tag{3.46}$$

Besides, since the cluster bases product $\mathbf{B}_j$ only involve original cluster bases in $\mathbf{A}_{\mathcal{H}^2}$ and $\mathbf{B}_{\mathcal{H}^2}$ matrices, we can prepare the above $\mathbf{B}_j$ for all leaf and nonleaf clusters before

MMP algorithm. In addition, the nonleaf level cluster bases projection (transformation) can also be calculated using children's ones as shown in (3.16). The formulas are given below:

$$\begin{aligned}
\mathbf{P}_i^{\mathbf{A}} &= (\mathbf{V}_{i_r}^{\mathbf{C}})^H \mathbf{V}_{i_r}^{\mathbf{A}} = (\mathbf{T}_{i_{1r}}^{\mathbf{C}})^H \mathbf{P}_{i_1}^{\mathbf{A}} \mathbf{T}_{i_{1r}}^{\mathbf{A}} + (\mathbf{T}_{i_{2r}}^{\mathbf{C}})^H \mathbf{P}_{i_2}^{\mathbf{A}} \mathbf{T}_{i_{2r}}^{\mathbf{A}}; \\
\mathbf{P}_k^{\mathbf{B}} &= (\mathbf{V}_{k_c}^{\mathbf{B}})^T (\mathbf{V}_{k_c}^{\mathbf{C}})^* = (\mathbf{T}_{k_{1c}}^{\mathbf{B}})^T \mathbf{P}_{k_1}^{\mathbf{B}} (\mathbf{T}_{k_{1c}}^{\mathbf{C}})^* + (\mathbf{T}_{k_{2c}}^{\mathbf{B}})^T \mathbf{P}_{k_2}^{\mathbf{B}} (\mathbf{T}_{k_{2c}}^{\mathbf{C}})^*.
\end{aligned} \tag{3.47}$$

We also compute the collected **NL** matrix block in $\mathbf{A}_{\mathcal{H}^2}$ and $\mathbf{B}_{\mathcal{H}^2}$ at current level l by the following equation,

$$\begin{aligned}
(\mathbf{NL}_{i,j}^{\mathbf{A}})_{coll.}^{(l)} &= (\mathbf{T}_{i_r}^{\mathbf{C}})^H (\mathbf{NL}_{i,j}^{\mathbf{A}})_{coll.}^{(l+1)} \mathbf{T}_{j_r}^{\mathbf{B}} \\
(\mathbf{NL}_{j,k}^{\mathbf{B}})_{coll.}^{(l)} &= (\mathbf{T}_{j_c}^{\mathbf{A}})^T (\mathbf{NL}_{j,k}^{\mathbf{B}})_{coll.}^{(l+1)} (\mathbf{T}_{k_c}^{\mathbf{C}})^*
\end{aligned} \tag{3.48}$$

where superscript l denotes tree level. After we prepare the matrix products in (3.46), (3.47), and (3.48), we can proceed to calculate the coupling matrices in (3.45) efficiently as:

$$\mathbf{S}_{i,k}^{\mathbf{C}} = \begin{cases}
(\mathbf{T}_{i_r}^{\mathbf{C}})^H (\mathbf{NL}_{i,j}^{\mathbf{A}} \mathbf{NL}_{j,k}^{\mathbf{B}})_{coll.}^{(l)} (\mathbf{T}_{k_c}^{\mathbf{C}})^* & \text{case-1} \\
(\mathbf{NL}_{i,j}^{\mathbf{A}})_{coll.}^{(l)} \mathbf{S}_{j,k}^{\mathbf{B}} \mathbf{P}_k^{\mathbf{B}} & \text{case-2} \\
\mathbf{P}_i^{\mathbf{A}} \mathbf{S}_{i,j}^{\mathbf{A}} (\mathbf{NL}_{j,k}^{\mathbf{B}})_{coll.}^{(l)} & \text{case-3} \\
\mathbf{P}_i^{\mathbf{A}} \mathbf{S}_{i,j}^{\mathbf{A}} \mathbf{B}_j \mathbf{S}_{j,k}^{\mathbf{B}} \mathbf{P}_k^{\mathbf{B}} & \text{case-4.}
\end{cases} \tag{3.49}$$

All the coupling matrices calculation are performed in rank size k_l. So the computational cost is $O(k_l^3)$. After coupling matrices calculation in (3.49), all the admissible products at this nonleaf level multiplication can be represented as $\mathbf{R}_{i,j}^{\mathbf{C}} = \mathbf{T}_{i_r}^{\mathbf{C}} \mathbf{S}_{i,j}^{\mathbf{C}} \mathbf{T}_{j_c}^{\mathbf{C}}$.

3.3.5 Summary of overall algorithm at each non-leaf level

The cluster bases products $\mathbf{B}_j$ have been computed for all clusters j before the MMP starts, since they are only related to the original cluster bases.

At each non-leaf level, we do the following:

1. Collect four blocks in an **NL** block in $\mathbf{A}_{\mathcal{H}^2}$ to a block of $O(k_{l+1})$ size, using the newly generated children row cluster bases of **C** (transfer matrices if children

are not at the leaf level) and the original column cluster bases of $\mathbf{B}$ (or transfer matrices). This is to generate the $(\mathbf{NL}^{\mathbf{A}}_{i,j})_{coll.}$, shown in (3.29).

2. Collect four blocks in an $\mathbf{NL}$ block in $\mathbf{B}_{\mathcal{H}^2}$ to a block of $O(k_{l+1})$ size, using the original row cluster bases of $\mathbf{A}$ (or transfer matrices) and the new children column cluster bases of $\mathbf{C}$ (transfer matrices if children are not at the leaf level). This is to generate $(\mathbf{NL}^{\mathbf{B}}_{j,k})_{coll.}$, shown in (3.32).

3. Merge four blocks in an $\mathbf{R}$ block in $\mathbf{C}_{\mathcal{H}^2}$. This corresponds to the $(\mathbf{NL}^{\mathbf{A}}_{i,j}\mathbf{NL}^{\mathbf{B}}_{j,k})^{(l)}_{coll.}$ in (3.49).

4. Calculate new row and column transfer matrices of product matrix $\mathbf{C}_{\mathcal{H}^2}$ at this level.

5. Prepare cluster bases projections $\mathbf{P}^{\mathbf{A}}_{i}$, $\mathbf{P}^{\mathbf{B}}_{k}$, and perform an $\mathbf{NL}$ block collect shown in (3.48);

6. Perform four cases of multiplications shown in (3.49).

After we finish one-way bottom-up tree traversal to calculate block matrix products at all the levels, i.e. from leaf level all the way up to minimal admissible level, we need to perform a post-processing for the coupling matrices associated with the $\mathbf{NL}$ blocks in $\mathbf{C}_{\mathcal{H}^2}$. They exist because of the multiplications cases described in Section 3.3.1. This could be efficiently done by performing one-way top-down split process, the same as the matrix backward transformation shown in [7]. This post processing stage is to split the coupling matrices in $\mathbf{NL}$ to lower level admissible or inadmissible blocks.

3.4 Accuracy and Complexity Analysis

In this section, we analyze the accuracy and computational complexity of the proposed algorithm to compute $\mathcal{H}^2$-matrix-matrix products.

3.4.1 Accuracy

Different from existing formatted $\mathcal{H}^2$-matrix-matrix products [7], in the proposed new algorithm, the accuracy of the product is directly controlled by ϵ_{trunc}. No formatted multiplications are performed, and the cluster bases are changed to represent the updates to the original matrix accurately. This makes each operation performed in the proposed MMP controlled by accuracy or exact. When generating an $\mathcal{H}^2$-matrix to represent the original dense matrix, the accuracy is controlled by $\epsilon_{\mathcal{H}^2}$, which is the same as in [15].

3.4.2 Time and Memory Complexity

The proposed MMP involves $O(L)$ levels of computation. At each level, there are 2^l clusters. For each cluster, the cost of changing the cluster bases at the leaf level due to four cases of multiplications is to perform $O(C_{sp})^2$ multiplications, and each of which has a constant cost, as can be seen from (3.10), (3.11), and (3.12). The cost of changing the cluster bases at the non-leaf level due to the four cases of multiplications is also to perform $O(C_{sp})^2$ multiplications for each cluster, and each of which has a cost of $O(k_l)^3$, as can be seen from (3.37), (3.38), (3.39). Notice that the **NL** blocks in **A** and **B** are collected level by level, at each level, there are $2^l O(C_{sp})$ **NL** block, and each collect operation also costs $O(k_l)^3$ only. Other auxiliary matrices are generated using a similar computational cost.

As for the computation of the four cases of multiplications at each level, each case involves $O(C_{sp})^2$ multiplications for each cluster, and each of which costs $O(k_l)^3$ at the non-leaf level and $O(leafsize)^3$ at the leaf level as can be seen from (3.18), and (3.49).

Hence, the time complexity of the proposed MMP can be found as

$$\textbf{Time Complexity} = \sum_{l=0}^{L} C_{sp}^2 2^l O(k_l)^3 = C_{sp}^2 \sum_{l=0}^{L} 2^l O(k_l)^3. \tag{3.50}$$

And the storage for each block is $O(k_l^2)$, with each cluster having C_{sp} blocks. So the memory complexity is

$$\textbf{Memory Complexity} = \sum_{l=0}^{L} C_{sp} 2^l O(k_l)^2 = C_{sp} \sum_{l=0}^{L} 2^l O(k_l)^2. \tag{3.51}$$

Recall k_l is the rank at tree level l. Hence, (3.50) and (3.51) show that the overall complexity is a function of rank k_l. Taking into account the rank's growth with electrical size as shown in [32], we can get the time and memory complexity of proposed MMP for different rank scaling. For constant-rank $\mathcal{H}^2$-matrices, since k_l is a constant irrespective of matrix size, the complexity of the proposed direct solution is strictly $O(N)$ in both CPU time and memory consumption, as shown below.

For constant k_l:

$$\textbf{Time Complexity} = C_{sp}^2 k_l^3 \sum_{l=0}^{L} 2^l = O(N), \tag{3.52}$$

$$\textbf{Memory Complexity} = C_{sp} k_l^2 \sum_{l=0}^{L} 2^l = O(N). \tag{3.53}$$

For electrodynamic analysis, to ensure a prescribed accuracy, the rank becomes a function of electrical size, and thereby tree level. Different $\mathcal{H}^2$-matrix representations can result in different complexities, because their rank's behavior is different. Using a minimal-rank $\mathcal{H}^2$-representation, as shown by [32], the rank grows linearly with electrical size for general 3-D problems. In a VIE, k_l is proportional to the cubic root of matrix size at level l, because this is the electrical size at level l. Hence for a VIE, (3.50) and (3.51) become

For k_l linearly growing with electrical size:

$$\textbf{Time Complexity} = {C_{sp}}^2 \sum_{l=0}^{L} 2^l \left[\left(\frac{N}{2^l} \right)^{\frac{1}{3}} \right]^3 = O(NlogN), \tag{3.54}$$

$$\textbf{Memory Complexity} = C_{sp} \sum_{l=0}^{L} 2^l \left[\left(\frac{N}{2^l} \right)^{\frac{1}{3}} \right]^2 = O(N). \tag{3.55}$$

So the time complexity of the proposed MMP algorithm for 3D electrodynamic analysis is $O(NlogN)$, and the memory complexity is $O(N)$.

3.5 Numerical Results

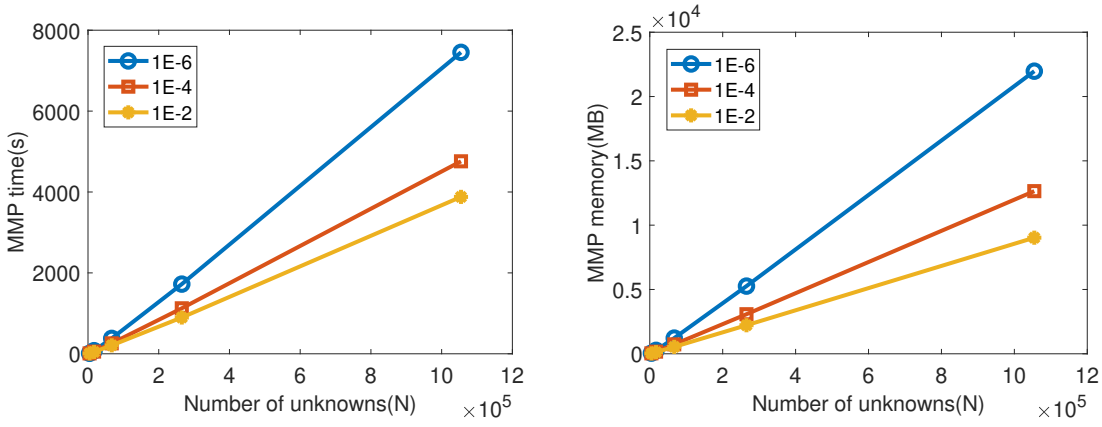
In order to demonstrate the accuracy and low computational complexity of the proposed fast $\mathcal{H}^2$-matrix-matrix multiplication for general $\mathcal{H}^2$ matrices, we again use $\mathcal{H}^2$ matrices resulting from large-scale capacitance extraction and volume integral equations (VIE) for electromagnetic analysis.

3.5.1 Two-layer Cross Bus

The first example is the capacitance extraction of a 2-layer cross bus structure. In each layer, there are m conductors, and each conductors has a dimension of $1 \times 1 \times (2m+1)\ m^3$. We simulate a suite of such structures with 16, 32, 64, 128 and 256 buses respectively. The parameters used in the $\mathcal{H}^2$ matrix construction are $leafsize = 30$, admissibility condition [7] $\eta = 1.0$, and $\epsilon_{\mathcal{H}^2} = 10^{-4}$. For the proposed $\mathcal{H}^2$ MMP, the calculated cluster bases are truncated at 10^{-2}, 10^{-4} and 10^{-6}. As shown in Fig. 3.4, the proposed MMP has linear time and memory performance for different truncation thresholds. We also check the product error by matrix-vector product as equation (2.27). From Table 3.1, we can see the accuracy of proposed MMP is directly controlled by ϵ_{trunc} used in appending cluster bases.

Table 3.1.: $\mathcal{H}^2$ MMP error at different ϵ_{trunc} for large-scale capacitance extraction matrices.

N	4,480	17,152	67,072	265,216	1,054,720
1E-2	4.35E-2	5.72E-2	5.73E-2	5.80E-2	5.97E-2
1E-4	3.71E-3	3.72E-3	3.86E-3	3.80E-3	3.67E-3
1E-6	2.82E-4	3.28E-4	3.86E-4	4.50E-4	5.66E-4



(a) (b)

Fig. 3.4.: MMP performance for $\mathbf{A}_{\mathcal{H}^2} \times \mathbf{B}_{\mathcal{H}^2}$ of cross buses. (a) Time scaling v.s. N. (b) Memory scaling v.s. N.

3.5.2 Large-scale Dielectric Slab

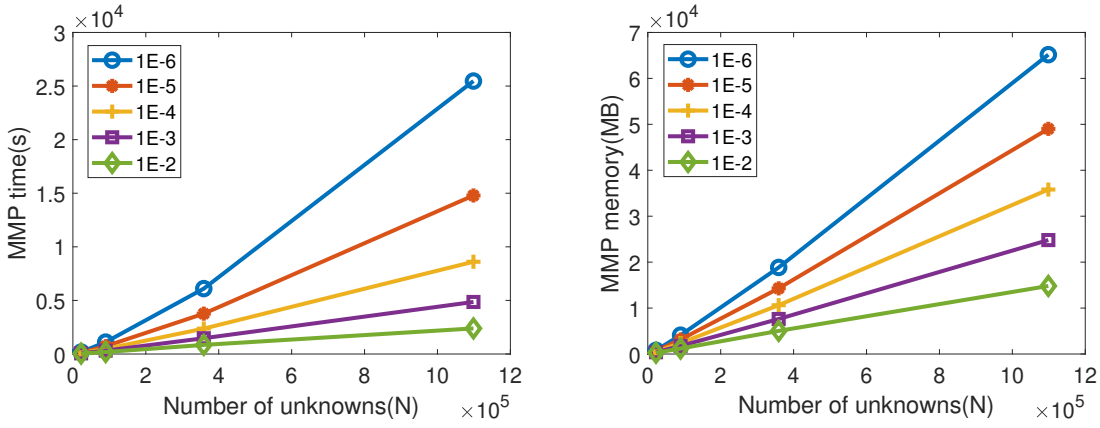
We then simulate a dielectric slab with $\epsilon_r = 2.54$ at 300 MHz. The thickness of the slab is fixed to be $0.1\lambda_0$. The width and length are simultaneously increased from $4\lambda_0$, $8\lambda_0$, $16\lambda_0$, to $28\lambda_0$. With a mesh size of $0.1\lambda_0$, the resultant N ranges from 22,560 to 1,098,720 for this suite of slab structures.The parameters used in the $\mathcal{H}^2$-matrix construction are $leafsize = 40$, admissibility condition [7] $\eta = 2.0$, and $\epsilon_{\mathcal{H}^2} = 10^{-3}$. For the proposed $\mathcal{H}^2$ MMP, the calculated cluster bases truncation ϵ_{trunc} are 10^{-2}, 10^{-3}, 10^{-4}, $10^{-5}$and 10^{-6} respectively, to examine the computational complexity and error controllability of the proposed MMP. Based on [32], the rank's growth rate with electrical size for 2-D slab is lower than linear, and being a square root of the log-linear of the electric size. Substituting such a rank's growth into the complexity analysis in (2.21) and (2.22), we will obtain linear complexity in both memory and time.

In Fig. 3.5 (a), we plot the MMP time with respect to N, for all different choices of cluster bases truncation. It is clear that the smaller ϵ_{trunc} value, the larger the MMP time. However, the complexity remains the same as linear regardless of the

choice of ϵ_{trunc}. The memory cost is plotted in Fig. 3.5 (b). Obviously, it scales linearly with the number of unknowns. The error of the proposed MMP is measured the same by random right hand side error as shown in equation (2.27). In table 3.2, we list the error as a function of ϵ_{trunc}. Excellent accuracy can be observed in the entire unknown range. Furthermore, the accuracy can be controlled by ϵ_{trunc}, and overall smaller ϵ_{trunc} results in better accuracy.

Table 3.2.: $\mathcal{H}^2$ MMP error for 2-D slab.

N	22560	89920	359040	1098720
1E-2	8.54E-3	1.06E-2	1.49E-2	1.07E-2
1E-3	2.52E-3	3.17E-3	4.23E-3	3.79E-3
1E-4	7.86E-4	9.76E-4	1.38E-3	1.23E-3
1E-5	2.91E-4	3.37E-4	4.22E-4	4.11E-4
1E-6	8.04E-5	9.85E-5	1.27E-4	1.36E-4



(a) (b)

Fig. 3.5.: MMP performance for $\mathbf{A}_{\mathcal{H}^2} \times \mathbf{A}_{\mathcal{H}^2}$ of 2-D slab from 4λ to 28λ. (a) Time scaling v.s. N. (b) Memory scaling v.s. N.

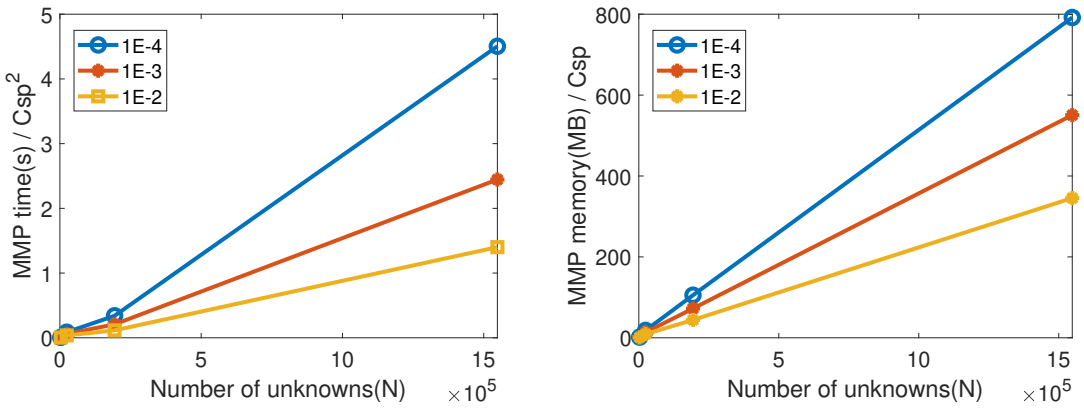
3.5.3 Large-scale Array of Dielectric Cubes

Next, we simulate a large-scale array of dielectric cubes at 300 MHz. The relative permittivity of the cube is $\epsilon_r = 4.0$. Each cube is of size $0.3\lambda_0 \times 0.3\lambda_0 \times 0.3\lambda_0$. The distance between adjacent cubes is kept to be $0.3\lambda_0$. The number of the cubes is increased along the $x-$, $y-$, and $z-$ directions simultaneously from 2 to 16, thus producing a 3-D cube array from $2 \times 2 \times 2$ to $16 \times 16 \times 16$ elements. The number of unknowns N is respectively 3,024, 24,192, 193,536, and 1,548,288 for these arrays. During the construction of $\mathcal{H}^2$ matrix, we set $leafsize = 20$, $\eta = 1$ and $\epsilon_{\mathcal{H}^2} = 10^{-2}$. For the proposed $\mathcal{H}^2$ MMP, the calculated cluster bases are truncated at 10^{-2}, 10^{-3} and 10^{-4}. In Fig. 3.6 (a) and Fig. 3.6 (b), we plot the $\mathcal{H}^2$-matrix-matrix multiplication time divided by C_{sp}^2, and the storage cost normalized with C_{sp} with respect to N. As can be seen, their scaling rate with N agrees very well with our theoretical complexity analysis. For the over one-million unknown case which is a $16 \times 16 \times 16$ cube array having thousands of cube elements, the error is still controlled to be as small as 0.809% at $\epsilon_{trunc} = 10^{-4}$. The random RHS error of the proposed MMP is listed in Table 3.3 for this example, which again reveals excellent accuracy and error controllability of the proposed MMP. We also compare the accuarcy for

Table 3.3.: $\mathcal{H}^2$ MMP error at different ϵ_{trunc} for 3-D cube array.

N	3024	24192	193536	1548288
Existing [7]	9.02E-2	1.01E-1	1.77E-1	2.74E-1
1E-2	1.91E-2	2.38E-2	3.82E-2	6.58E-2
1E-3	5.51E-3	7.23E-3	1.06E-2	2.16E-2
1E-4	1.48E-3	2.46E-3	3.69E-3	8.09E-3

proposed MMP with existing MMP [7] using this 3-D example. As shown in Table 3.3, the proposed MMP has much better accuracy.



(a) (b)

Fig. 3.6.: MMP performance for $\mathbf{A}_{\mathcal{H}^2} \times \mathbf{A}_{\mathcal{H}^2}$ of 3-D cube array. (a) Time scaling v.s. N. (b) Memory scaling v.s. N.

3.6 Conclusion

In this chapter, we develop a fast algorithm to calculate $\mathcal{H}^2$-matrix-matrix products for general $\mathcal{H}^2$ matrices. This proposed algorithm not only has *explicitly* controlled accuracy of the product, but also generate the *minimal-rank* representation of the product. As compared with the MMP algorithm in chapter 2, whose old cluster bases are kept, this new MMP algorithm proposed here is more memory efficient. Besies, the proposed algorithm has been applied to calculate $\mathcal{H}^2$-matrix-matrix products for large-scale capacitance extraction matrices whose kernel is static and real-valued and electrically large VIEs whose kernel is oscillatory and complex-valued. For constant-rank $\mathcal{H}^2$ matrices, the proposed MMP has an $O(N)$ complexity in both time and memory. For rank growing with the electrical size linearly, the proposed MMP has an $O(NlogN)$ complexity time and $O(N)$ complexity in memory. $\mathcal{H}^2$-matrices products with millions of unknowns are simulated on a single core CPU in fast CPU run time. Comparisons with existing $\mathcal{H}^2$-matrix-matrix products have demonstrated the excellent accuracy and efficiency of this new MMP algorithm.

4. ACCURACY DIRECTLY CONTROLLED FAST DIRECT SOLUTION OF GENERAL $\mathcal{H}^2$ MATRICES

4.1 Introduction

The $\mathcal{H}^2$ matrix has be utilized to develop fast solvers for electromagnetic analysis. In [4, 14], fast $\mathcal{H}^2$-matrix inversion algorithms are developed. They have shown a complexity of $O(N)$ for solving electrically small and moderate problems for both surface and volume IEs (VIEs); and a complexity of $O(NlogN)$ for solving electrically large VIEs. Despite a significantly reduced complexity, in existing direct solutions of $\mathcal{H}^2$ matrices, the cluster bases used to represent a dense matrix are also used to represent the matrix's inverse as well as LU factors. Physically speaking, such a choice of the cluster bases often constitutes an accurate choice. However, mathematically speaking, the accuracy of the inversion and factorization cannot be directly controlled. The lack of a direct accuracy control is also observed in the $\mathcal{H}^2$-based matrix-matrix multiplication in mathematical literature. Algorithm wise, the collect operation in [3] involves approximations. This operation is performed when multiplying a non-leaf block with a non-leaf block, or multiplying a nonleaf block with an admissible block, with the target block being admissible. In this operation, the four admissible or inadmissible blocks at the children level are collected to a single admissible block based on the cluster bases used to represent the target block in the original matrix. During the inversion and factorization procedure, only the coupling matrix of each admissible block is computed, while the cluster bases are not changed. When the accuracy of the direct solution is not satisfactory, one can only change the original $\mathcal{H}^2$-representation, i.e., change the cluster bases or the rank of the original matrix, with the hope that they can better represent the inverse and LU factors.

Recently, an HSS matrix has also been introduced to develop fast direct solutions of exact arithmetic. However, the structure of the HSS matrix is not efficient for representing general electromagnetic problems. The resultant rank of HSS matrix can be too high to be computed efficiently, especially for electrically large analysis. From this perspective, an $\mathcal{H}^2$ matrix is a more efficient structure for general applications, since the distance between the sources and observers can be used to represent a dense matrix with a much reduced rank. However, its accuracy-controlled direct solution is still lacking in the open literature. The contribution of this work is such an accuracy-controlled direct solution of general $\mathcal{H}^2$ matrices, which is also applied to solve electrically large VIEs. As a demonstration of its performance, we show how it is used to solve electrically large VIEs, with an $O(NlogN)$ complexity in factorization and $O(N)$ complexity in solution and memory, and with a strictly controlled accuracy.

4.2 Proposed Factorization and Inversion Algorithms

Given a general $\mathcal{H}^2$ matrix $\mathbf{Z}$, the proposed direct solution is a one-way tree traversal from the leaf level all the way up to the root level of the $\mathcal{H}^2$-tree. At each level, the factorization proceeds from the left to the right across all the clusters. This is very different from the algorithm of [9, 11, 14, 15], where a recursive procedure is performed. To help better understand the proposed algorithm, while we present a generic algorithm, we will use the $\mathcal{H}^2$ matrix shown in Fig. 4.1(a), as an example to illustrate the overall procedure. In this figure, green blocks are admissible blocks, and red ones represent inadmissible blocks.

4.2.1 Computation at the Leaf Level

For the first leaf cluster

We start from the leaf level ($l = L$). For the first cluster, whose index is denoted by $i = 1$, we perform three steps as follows.

Step 1. We compute the complementary cluster basis $\mathbf{V}_i^{\perp}$, which is orthogonal to $\mathbf{V}_i$ (cluster basis of cluster i). This can be easily done by carrying out an SVD or QL factorization on $\mathbf{V}_i$. The cost is not high, instead, it is constant at the leaf level. We then combine $\mathbf{V}_i$ with $\mathbf{V}_i^{\perp}$ to form $\widetilde{\mathbf{Q}}_i$ matrix as the following

$$\widetilde{\mathbf{Q}}_i = [(\mathbf{V}_i^{\perp})_{\#i\times(\#i-k_l)} \ (\mathbf{V}_i)_{\#i\times k_l}], \tag{4.1}$$

in which k_l is the rank at level l (now $l = L$), and # denotes the number of unknowns contained in a set.

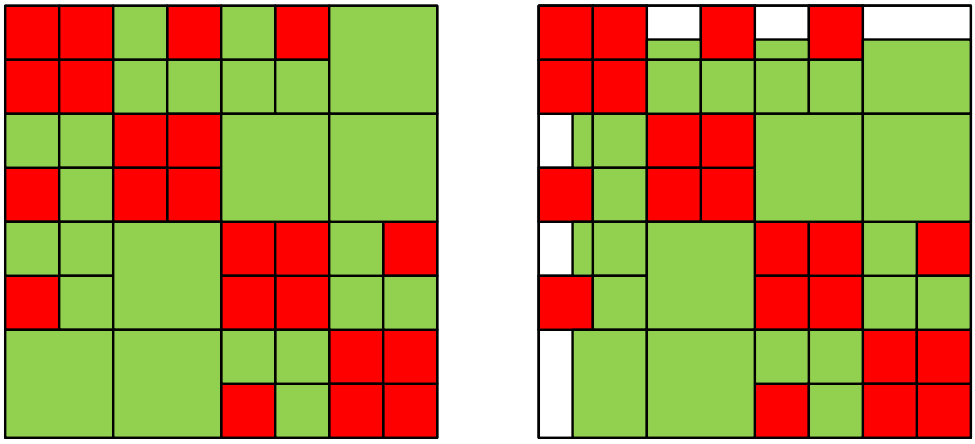
Step 2. Let $\mathbf{Z}_{cur}$ be the current matrix that remains to be factorized. For leaf cluster $i = 1$, $\mathbf{Z}_{cur} = \mathbf{Z}$. We compute a transformed matrix $\mathbf{Q}_i^H \mathbf{Z}_{cur} \overline{\mathbf{Q}}_i$, and use it to overwrite $\mathbf{Z}_{cur}$, thus

$$\mathbf{Z}_{cur} = \mathbf{Q}_i^H \mathbf{Z}_{cur} \overline{\mathbf{Q}}_i, \tag{4.2}$$

where $\mathbf{Q}_i^H$ denotes $\mathbf{Q}_i$'s complex conjugate transpose, and

$$\mathbf{Q}_i = diag\{\mathbf{I}_1, \mathbf{I}_2, ..., \widetilde{\mathbf{Q}}_i, ..., \mathbf{I}_{\#\{\text{leaf clusters}\}}\}, \tag{4.3}$$

is a block diagonal matrix. The i-th diagonal block in $\mathbf{Q}_i$ is $\widetilde{\mathbf{Q}}_i$, and other blocks are identity matrices $\mathbf{I}$, the size of each of which is the corresponding leaf-cluster size. The subscripts in the right hand side of (4.3) denote the indexes of diagonal blocks, which are also the indexes of leaf clusters, and $\#\{\text{leaf clusters}\}$ stands for the number of leaf clusters. If leaf cluster $i = 1$ is being computed, $\widetilde{\mathbf{Q}}_i$ appears in the first block of (4.3). In addition, the $\overline{\mathbf{Q}}_i$ in (4.2) denotes the complex conjugate of $\mathbf{Q}_i$, as the cluster bases we construct are complex-valued to account for general $\mathcal{H}^2$-matrices. In (4.3), $\mathbf{Q}_i$ only consists of one $\widetilde{\mathbf{Q}}_i$, where i is the current cluster being computed. This is because $\widetilde{\mathbf{Q}}_i$ for other clusters are not ready for use yet, since in the proposed



(a) (b)

Fig. 4.1.: (a) Original $\mathcal{H}^2$ matrix. (b) Illustration of $\mathbf{Q}_1^H \mathbf{Z} \overline{\mathbf{Q}}_1$.

algorithm, we will update cluster bases to take into account the contribution from fill-ins. This will become clear in the sequel.

The purpose of doing this step, i.e., obtaining (4.2), is to introduce zeros in the matrix being factorized, since

$$\widetilde{\mathbf{Q}}_i^H \times \mathbf{V}_i = \begin{bmatrix} \mathbf{0}_{(\#i-k_l)\times k_l} \\ \mathbf{I}_{k_l \times k_l} \end{bmatrix} \tag{4.4}$$

$$\mathbf{V}_i^T \times \overline{\widetilde{\mathbf{Q}}}_i = [\mathbf{0}_{k_l \times (\#i-k_l)} \ \mathbf{I}_{k_l \times k_l}]. \tag{4.5}$$

The operation of (4.2) thus zeros out the first $(\#i - k_l)$ rows of the admissible blocks formed by row cluster i, as well as the first $(\#i - k_l)$ columns in $\mathbf{Z}_{cur}$, as shown by the white blocks in Fig. 4.1(b). The real computation of (4.2) is, hence, in the inadmissible blocks of cluster i, as shown by the four red blocks associated with cluster $i = 1$ in Fig. 4.1(b).

Step 3. After Step 2, the first $(\#i - k_l)$ rows and columns of $\mathbf{Z}_{cur}$ in the admissible blocks of cluster i become zero. Hence, if we eliminate the first $(\#i - k_l)$ unknowns of cluster i via a partial LU factorization, the admissible blocks of cluster i at these rows and columns would stay as zero, and thus not affected. As a result, the resulting

L and U factors, as well as the fill-in blocks generated, would only involve a constant number of blocks, which is bounded by $O(C_{sp}^2)$.

In view of the above property, in Step 3, we perform a partial LU factorization to eliminate the first $(\#i - k_l)$ unknowns of cluster i in $\mathbf{Z}_{cur}$. Let this set of unknowns be denoted by i'. $\mathbf{Z}_{cur}$ can be cast into the following form accordingly

$$\mathbf{Z}_{cur} = \begin{bmatrix} \mathbf{F}_{i'i'} & \mathbf{Z}_{i'c} \\ \mathbf{Z}_{ci'} & \mathbf{Z}_{cc} \end{bmatrix}, \tag{4.6}$$

in which $\mathbf{F}_{i'i'}$ denotes the diagonal block of the $(\#i-k_l)$ unknowns, $\mathbf{Z}_{i'c}$ is the remaining rectangular block in the row of set i'. Here, c contains the rest of the unknowns which do not belong to i'. The $\mathbf{Z}_{ci'}$ is the transpose block of $\mathbf{Z}_{i'c}$, and $\mathbf{Z}_{cc}$ is the diagonal block formed by unknowns in set c. It is evident that $\mathbf{Z}_{i'c}$ ($\mathbf{Z}_{ci'}$) only consists of $O(C_{sp})$ inadmissible blocks, as the rest of the blocks are zero. Take leaf cluster $i = 1$ as an example. It forms inadmissible blocks with clusters $i = 2, 4, 6$, as shown by the red blocks in Fig. 4.1(b). Hence, $\mathbf{Z}_{i'c} = [\mathbf{F}_{12}, \mathbf{F}_{14}, \mathbf{F}_{16}]$.

After performing a partial LU factorization to eliminate the unknowns contained in i', we obtain

$$\mathbf{Z}_{cur} = \begin{bmatrix} \mathbf{L}_{i'i'} & \mathbf{0} \\ \mathbf{Z}_{ci'}\mathbf{U}_{i'i'}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \widetilde{\mathbf{Z}}_{cc} \end{bmatrix} \begin{bmatrix} \mathbf{U}_{i'i'} & \mathbf{L}_{i'i'}^{-1}\mathbf{Z}_{i'c} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}, \tag{4.7}$$

where

$$\mathbf{F}_{i'i'} = \mathbf{L}_{i'i'}\mathbf{U}_{i'i'}, \tag{4.8}$$

and the Schur complement is

$$\widetilde{\mathbf{Z}}_{cc} = \mathbf{Z}_{cc} - \mathbf{Z}_{ci'}\mathbf{U}_{i'i'}^{-1}\mathbf{L}_{i'i'}^{-1}\mathbf{Z}_{i'c}, \tag{4.9}$$

in which $\mathbf{Z}_{ci'}\mathbf{U}_{i'i'}^{-1}\mathbf{L}_{i'i'}^{-1}\mathbf{Z}_{i'c}$ represents the fill-in blocks introduced due to the elimination of i'-unknowns. Because of the zeros introduced in the admissible blocks, the $\mathbf{Z}_{i'c}$ and $\mathbf{Z}_{ci'}$ only consist of inadmissible blocks formed by cluster i, the number of which is bounded by constant C_{sp}. Hence, the number of fill-in blocks introduced by this step is also bounded by a constant, which is $O(C_{sp}^2)$.

Equation (4.7) can further be written in short as

$$\mathbf{Z}_{cur} = \mathbf{L}_i^l \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \widetilde{\mathbf{Z}}_{cc} \end{bmatrix} \mathbf{U}_i^l, \tag{4.10}$$

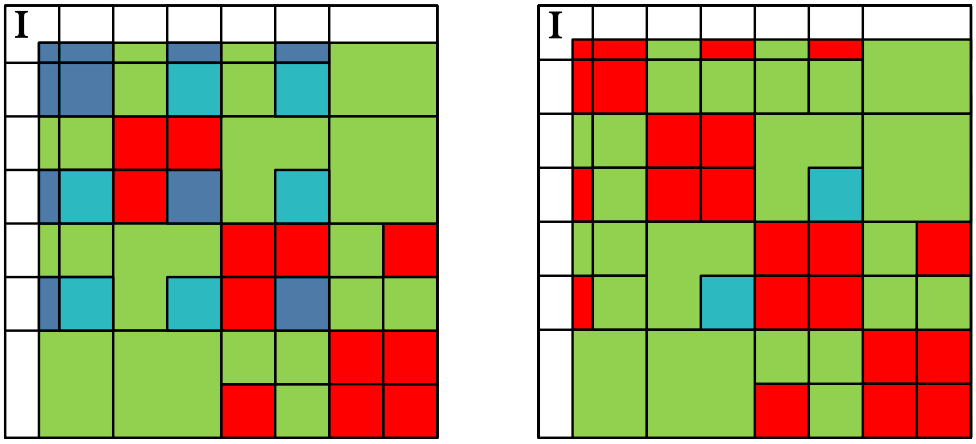
in which the first matrix of (4.7), which is the $\mathbf{L}$ factor generated after a partial LU of cluster i at level l, is denoted by $\mathbf{L}_i^l$. Similarly, the rightmost matrix of (4.7), which is the $\mathbf{U}$ factor generated after a partial LU of cluster i at level l, is denoted by $\mathbf{U}_i^l$. The center matrix is the one that remains to be factorized. Thus, we let

$$\mathbf{Z}_{cur} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \widetilde{\mathbf{Z}}_{cc} \end{bmatrix}. \tag{4.11}$$

This matrix is illustrated in Fig. 4.2(a), where the fill-in blocks are added upon previous $\mathbf{Z}_{cur}$. If the fill-in block is added upon an inadmissible block, we add the fill-in directly to the inadmissible block. For the example shown in Fig. 4.1(a), these blocks are the blocks marked in dark blue in Fig. 4.2(a) after the i' set of unknowns in cluster $i = 1$ is eliminated. If the fill-in appears in an admissible block, we store the fill-in block temporarily in this admissible block for future computation. These blocks are the blocks marked in light blue in Fig. 4.2(a). To explain in more detail, from (4.9), it can be seen that the fill-in block formed between cluster j and cluster k has the following form

$$\mathbf{F}_{j,k} = \mathbf{Z}_{ji'} \mathbf{U}_{i'i'}^{-1} \mathbf{L}_{i'i'}^{-1} \mathbf{Z}_{i'k}, \tag{4.12}$$

which is also low rank in nature. The row cluster j and column cluster k belong to the set of clusters that form inadmissible blocks with cluster i, whose number is bounded by C_{sp}. If the original block formed between j and k, $\mathbf{Z}_{cur,jk}$, is admissible, no computation is needed for $\mathbf{F}_{j,k}$, i.e., we do not add it directly to the admissible block. Instead, we store it with this admissible block temporarily until cluster j is reached during the factorization. At that time, $\mathbf{F}_{j,k}$ will be used to compute an updated cluster basis for j. If the elimination of different clusters results in more than one fill-in block in admissible $\mathbf{Z}_{cur,jk}$ block, the later ones are added upon existing



(a) (b)

Fig. 4.2.: (a) $\mathbf{Z}_{cur}$ after partial LU of cluster 1 with fill-in blocks marked in blue. (b) Fill-in blocks of cluster 2 turned green after cluster basis update.

$\mathbf{F}_{j,k}$. In other words, the fill-ins at the admissible $\mathbf{Z}_{cur,jk}$ block are summed up and stored in a single $\mathbf{F}_{j,k}$.

In summary, Step 3 is to perform a partial LU factorization as shown in (4.7) to eliminate the first $\#i - k_l$ unknowns in cluster i. The storage and computation cost of this step can be analyzed as follows. In (4.7), the LU factorization of $\mathbf{F}_{i'i'}$ into $\mathbf{L}_{i'i'}$ and $\mathbf{U}_{i'i'}$ costs $O(leafsize^3)$ in computation, and $O(leafsize^2)$ in memory, which is hence constant. $\mathbf{Z}_{ci'}$ ($\mathbf{Z}_{i'c}$) only involves $O(C_{sp})$ inadmissible blocks. Hence, the computation of $\mathbf{Z}_{ci'}\mathbf{U}_{i'i'}^{-1}$ as well as $\mathbf{L}_{i'i'}^{-1}\mathbf{Z}_{i'c}$ is simply $O(C_{sp})$ operations, each of which scales as $O(leafsize^3)$, and hence the cost is constant also. The results of the partial factorization, i.e., $\mathbf{L}_i^l$ and $\mathbf{U}_i^l$, are stored in $O(C_{sp})$ blocks of *leafsize*. Thus, the memory cost is also constant. As for the fill-in blocks, their number is bounded by C_{sp}^2, and each of which is of *leafsize*. Hence, the computation and storage associated with fill-in blocks are constant too, at the leaf level for each cluster.

For other leaf clusters

Next, we proceed to the second leaf cluster $i = 2$ and other leaf clusters. For these leaf clusters, all the steps are the same as those performed for the first leaf cluster, except for adding another step before Step 1. We term this step as Step 0.

Step 0. This step is to update cluster basis $\mathbf{V}_i$ to account for the contributions of the fill-ins due to the elimination of previous clusters. If we do not update cluster bases $\mathbf{V}_i$, they are not accurate any more to represent the admissible blocks formed by cluster i, since the contents of these blocks have been changed due to the addition of the fill-in blocks. However, if we let the admissible block updated by fill-ins become a full-matrix block, i.e., an inadmissible block, the number of fill-ins would keep increasing after each step of partial factorization, and hence the resultant complexity would be too high to tolerate. To overcome this problem, we propose to update cluster bases as follows.

For cluster i, we take all the fill-in blocks associated with its admissible blocks, and use them to update cluster basis $\mathbf{V}_i$. These fill-in blocks are shown in (4.12), which have been generated and stored in the corresponding admissible blocks during the elimination of previous clusters. Let them be $\mathbf{F}_{i,j_k}(k = 1, 2, ..., O(C_{sp}))$, where the first subscript denotes the row cluster i, and the second being the column cluster index. The number of such blocks is bounded by $O(C_{sp})$ because the number of admissible blocks formed by a single cluster, like i, is no greater than C_{sp}. Next, we compute

$$\mathbf{G}_i = (\mathbf{I} - \mathbf{V}_i\mathbf{V}_i^H)(\sum\nolimits_k \mathbf{F}_{i,j_k}\mathbf{F}_{i,j_k}^H)(\mathbf{I} - \mathbf{V}_i\mathbf{V}_i^H)^H. \tag{4.13}$$

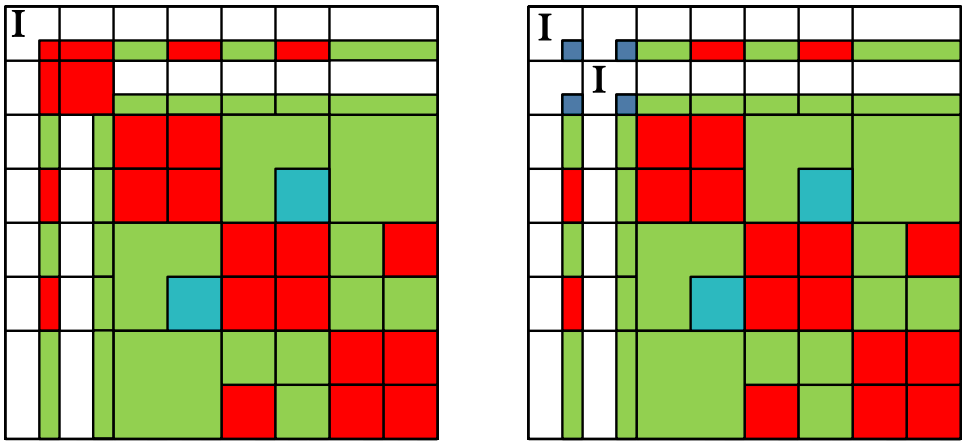
We then perform an SVD on $\mathbf{G}_i$, and truncate the singular vector matrix based on prescribed accuracy $\epsilon_{fill-in}$, obtaining

$$\mathbf{G}_i \overset{\epsilon_{fill-in}}{=} (\mathbf{V}_i^{add})\Sigma(\mathbf{V}_i^{add})^H. \tag{4.14}$$

The $\mathbf{V}_i^{add}$ is the additional cluster basis we supplement, since it captures the part of the column space of the fill-in blocks, which is not present in the original $\mathbf{V}_i$. The cluster basis can then be updated for cluster i as

$$\widetilde{\mathbf{V}}_i = [\mathbf{V}_i \ \mathbf{V}_i^{add}]. \tag{4.15}$$

By doing so, we are able to keep the original admissible blocks to be admissible while accurately taking into account the fill-in's contributions. Hence, the number of fill-ins introduced due to the elimination of each cluster can be kept to be a constant, instead of growing. In addition, by keeping the original cluster basis $\mathbf{V}_i$ in the new bases instead of generating a completely new one, we can keep the nested relationship in the original $\mathcal{H}^2$-matrix for efficient computations at upper levels. The computational cost of such a supplement of cluster bases is constant at leaf level, and scales as $O(k_l^3)$ at other levels (this will become clear soon) for each cluster.



(a) (b)

Fig. 4.3.: (a) $\mathbf{Q}_2^H \mathbf{Z}_{cur} \overline{\mathbf{Q}}_2$. (b) $\mathbf{Z}_{cur}$ after partial LU of cluster 2.

In Fig. 4.2(b) , the Step 0 is reflected by turning the light blue blocks associated with cluster $i = 2$ in Fig. 4.2(a) back to green blocks, symbolizing that they become

admissible again. After $\mathbf{V}_i$ is updated to $\widetilde{\mathbf{V}}_i$, we find its complementary bases, i.e., perform Step 1, to obtain

$$\widetilde{\mathbf{Q}}_i = [(\widetilde{\mathbf{V}}_i^{\perp})_{\#i\times(\#i-k_l)} \ (\widetilde{\mathbf{V}}_i)_{\#i\times k_l}]. \tag{4.16}$$

We then use $\widetilde{\mathbf{Q}}_i$ to generate (4.3), and project the current matrix to be factorized, shown in Fig. 4.2(b), to the matrix shown in (4.2). This again will zero out the first $(\#i - k_l)$ rows (columns) of the admissible blocks formed by cluster i, as illustrated in Fig. 4.3(a). We then proceed to Step 3 to perform a partial LU factorization. After this step, the matrix $\mathbf{Z}_{cur}$, which remains to be factorized, is illustrated in Fig. 4.3(b). Steps 0 to 3 are then repeated for the rest of the leaf clusters.

Updating the coupling matrix at the leaf level and the transfer matrix at one level higher

After the computation at the leaf level is finished, before proceeding to the next level, we need to update the coupling matrices at the leaf level, as well as the transfer matrices at one level higher. This is because the cluster bases have been updated at the leaf level.

For admissible blocks without fill-ins, their coupling matrices can be readily updated by appending zeros. For example, considering an admissible block formed between clusters i and k . Its coupling matrix $\mathbf{S}_{i,k}$ is updated to

$$\widetilde{\mathbf{S}}_{i,k} = \begin{bmatrix} \mathbf{S}_{i,k} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}, \tag{4.17}$$

because in this way

$$\widetilde{\mathbf{V}}_i \widetilde{\mathbf{S}}_{i,k} \widetilde{\mathbf{V}}_k^T = \mathbf{V}_i \mathbf{S}_{i,k} \mathbf{V}_k^T. \tag{4.18}$$

For admissible blocks with fill-ins, their coupling matrices are updated by adding the fill-in part onto the augmented coupling matrix, shown in (4.17), as

$$\widetilde{\mathbf{S}}_{i,k} = \widetilde{\mathbf{S}}_{i,k} + \widetilde{\mathbf{V}}_i^H \mathbf{F}_{i,k} \overline{\widetilde{\mathbf{V}}}_k, \tag{4.19}$$

in which $\mathbf{F}_{i,k}$ represents the fill-in block associated with the admissible block.

We also need to update the transfer matrices at one level higher, i.e., at the $l = L - 1$ level. This can be readily done by adding zeros to the original transfer matrices. To be clearer, consider a non-leaf cluster t, whose two children clusters are t_1 and t_2, its cluster basis can be written as

$$\begin{bmatrix} \mathbf{V}_{t_1}\mathbf{T}_{t_1} \\ \mathbf{V}_{t_2}\mathbf{T}_{t_2} \end{bmatrix} = \begin{bmatrix} [\mathbf{V}_{t_1}\mathbf{V}_{t_1}^{add}] & \mathbf{0} \\ \mathbf{0} & [\mathbf{V}_{t_2}\mathbf{V}_{t_2}^{add}] \end{bmatrix} \begin{bmatrix} \mathbf{T}_{t_1} \\ \mathbf{0} \\ \mathbf{T}_{t_2} \\ \mathbf{0} \end{bmatrix}, \tag{4.20}$$

where $\mathbf{T}_{t_1}$ and $\mathbf{T}_{t_2}$ are original transfer matrices associated with non-leaf cluster t.

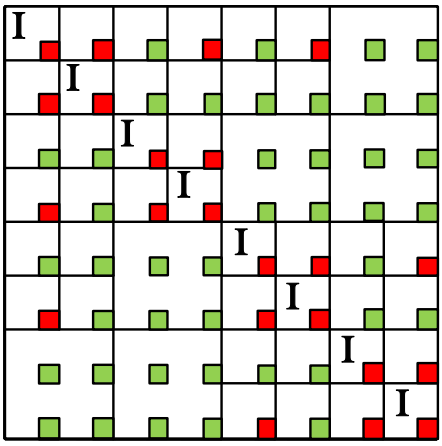


Fig. 4.4.: $\mathbf{Z}_{cur}$ left to be factorized after leaf-level computation.

The transfer matrices and coupling matrices at other levels all remain the same as before. They are not updated until corresponding levels are reached.

4.2.2 Computation at the Non-leaf Level

After all the leaf-level computation, we obtain a matrix shown in Fig. 4.4, which is current $\mathbf{Z}_{cur}$ left to be factorized. All the empty blocks are zero blocks. The diagonal

blocks corresponding to the unknowns that have been eliminated are identity matrices shown as $\mathbf{I}$. The nonzero blocks, shown as shaded blocks, each is of size $(k_{l+1} \times k_{l+1})$, where l is current tree level, $l+1$ is the previous tree level whose computation is finished, and k_{l+1} denotes the rank at the $(l+1)$-th level. The structure shown in Fig. 4.4 is not only true for $l = L-1$ level, i.e., one level above the leaf level, but also true for every non-leaf level l, except that the size of each block is different at different tree levels, since k_{l+1} is different especially for electrically large analysis. The computation at a non-leaf level is performed as follows.

Merging and permuting

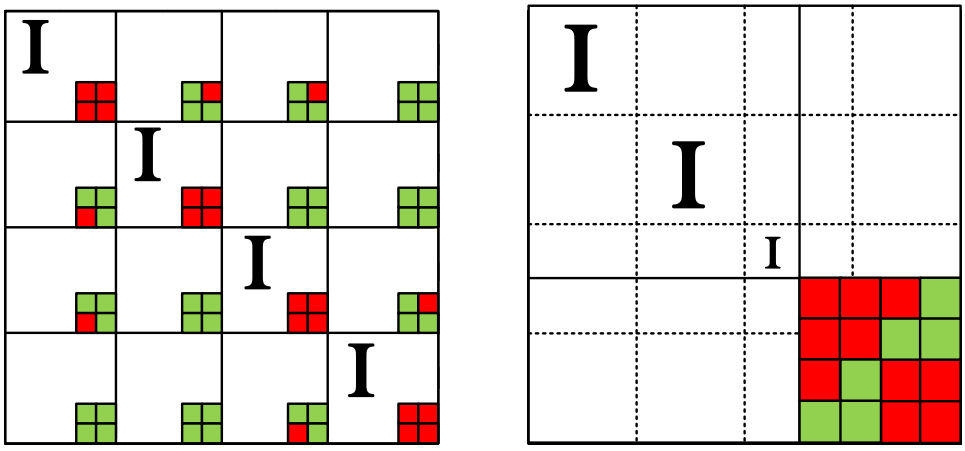
In $\mathbf{Z}_{cur}$, whose structure is shown in Fig. 4.4, we permute the matrix, and merge the four blocks of a non-leaf cluster, formed between its two children clusters and themselves, to a single block, as shown by the transformation from Fig. 4.4 to Fig. 4.5(a). After this, we further permute the unknowns that have not been eliminated to the end. This results in a matrix shown in Fig. 4.5(b). As a result, the bottom right block becomes the matrix that remains to be factorized. This matrix is of size $2^l(2k_{l+1}) \times 2^l(2k_{l+1})$, because it is formed between 2^l clusters at the non-leaf level l, and each cluster is of size $2k_{l+1}$. Obviously, this matrix is again an $\mathcal{H}^2$-matrix, as shown by green admissible blocks and red inadmissible blocks in Fig. 4.5(b). The difference with the leaf level is that each block, be its inadmissible or admissible, now is of size $2k_{l+1} \times 2k_{l+1}$. Hence, in the proposed direct solution algorithm, at each non-leaf level, the computation is performed on the matrix blocks whose size is the *rank* at that level, and hence efficient.

Now, each admissible block is of size only $2k_{l+1} \times 2k_{l+1}$. Its expression is different from that at the leaf level. Consider an admissible block formed between clusters t and s at a non-leaf level l, as shown by the green blocks in Fig. 4.5(a). It has the following form:

$$(\text{Admissible block})_{t,s}^{(l)} = \mathbf{T}_t \mathbf{S}_{t,s} \mathbf{T}_s^T, \tag{4.21}$$

where $\mathbf{T}_t$ is the transfer matrix for non-leaf cluster t, whose size is $O(2k_{l+1} \times k_l)$, $\mathbf{T}_s$ is the transfer matrix for cluster s whose size is also $O(2k_{l+1} \times k_l)$, and $\mathbf{S}_{t,s}$ is the coupling matrix whose size is $O(k_l \times k_l)$.

Since the matrix block left to be factorized after merging and permutation is again an $\mathcal{H}^2$-matrix, we can apply the same procedure performed at the leaf level, i.e., from Step 0 to Step 3, to the non-leaf level. In addition, from (4.21), it can also be seen clearly that now, transfer matrix $\mathbf{T}$ at a non-leaf level plays the same role as cluster basis $\mathbf{V}$ at the leaf level. Therefore, wherever $\mathbf{V}$ is used in the leaf-level computation, we replace it by $\mathbf{T}$ at a non-leaf level. In the following, we will briefly go through the Steps 0–3 performed at a non-leaf level, as the essential procedure is the same as that in the leaf level.



(a) (b)

Fig. 4.5.: (a) Merging to next level. (b) Permuting to obtain $\mathbf{Z}_{cur}$ to be factorized at next level.

Steps 0 to 3

Step 0 at the non-leaf level. This step is to update transfer matrix $\mathbf{T}_i$ to account for the contributions of the fill-ins due to the elimination of previous clusters.

For cluster i, we take all the fill-in blocks associated with its admissible blocks, and use them to update transfer matrix $\mathbf{T}_i$. These fill-in blocks are shown in (4.12), which have been generated and stored during the factorization of previous clusters. Let them be $\mathbf{F}_{i,j_k}(k = 1, 2, ..., O(C_{sp}))$, where the first subscript denotes the row cluster i. Different from the fill-ins at leaf level, the $\mathbf{F}_{i,j_k}$ now is of $(2k_{l+1} \times 2k_{l+1})$ size at a non-leaf level l. We then compute

$$\mathbf{G}_i = (\mathbf{I} - \mathbf{T}_i\mathbf{T}_i^H)(\sum\nolimits_k \mathbf{F}_{i,j_k}\mathbf{F}_{i,j_k}^H)(\mathbf{I} - \mathbf{T}_i\mathbf{T}_i^H)^H, \tag{4.22}$$

where all the k admissible blocks formed by cluster i are considered. Performing an SVD on $\mathbf{G}_i$, and truncating the singular vector matrix based on prescribed accuracy $\epsilon_{fill-in}$, we obtain

$$\mathbf{G}_i \overset{\epsilon_{fill-in}}{=} (\mathbf{T}_i^{add})\Sigma(\mathbf{T}_i^{add})^H. \tag{4.23}$$

The $\mathbf{T}_i^{add}$ is hence the additional column space we should supplement in the transfer matrix. The new transfer matrix $\widetilde{\mathbf{T}}_i$ can then be obtained as

$$\widetilde{\mathbf{T}}_i = [\mathbf{T}_i \ \mathbf{T}_i^{add}]. \tag{4.24}$$

The cost of this step for each cluster at level l is $O({k_l}^3)$.

Step 1 at a non-leaf level. We compute the complementary cluster basis $\widetilde{\mathbf{T}_i}^{\perp}$, which is orthogonal to $\widetilde{\mathbf{T}_i}$. This computation is not expensive, as it costs only $O({k_l}^3)$ at level l. We then combine $\widetilde{\mathbf{T}_i}$ with $\widetilde{\mathbf{T}_i}^{\perp}$ to form $\widetilde{\mathbf{Q}}_i$ matrix as the following

$$\widetilde{\mathbf{Q}}_i = [(\widetilde{\mathbf{T}_i}^{\perp})_{2k_{l+1}\times(2k_{l+1}-k_l)} \ (\widetilde{\mathbf{T}_i})_{2k_{l+1}\times k_l}]. \tag{4.25}$$

Step 2 at a non-leaf level. With $\widetilde{\mathbf{Q}}_i$ computed, we build a block diagonal matrix

$$\mathbf{Q}_i = diag\{\mathbf{I}_1, \mathbf{I}_2, ..., \widetilde{\mathbf{Q}}_i, ..., \mathbf{I}_{\#\{\text{clusters at level } l\}}\}, \tag{4.26}$$

using which we construct

$$\mathbf{Z}_{cur} = \mathbf{Q}_i^H\mathbf{Z}_{cur}\overline{\mathbf{Q}}_i, \tag{4.27}$$

where $\mathbf{Z}_{cur}$ only consists of the block that remains to be factorized, as shown by the bottom right matrix block in Fig. 4.5(b), which is not an identity. Similar to that

in the leaf level, by doing so, we zero out first $(2k_{l+1} - k_l)$ rows of the admissible blocks formed by row cluster i, as well as their transpose entries in $\mathbf{Z}_{cur}$. The real computation of (4.27) is hence in the inadmissible blocks of non-leaf cluster i, the cost of which is $O(k_l)^3$.

Step 3 at a non-leaf level. We perform a partial LU factorization to eliminate the first $(2k_{l+1} - k_l)$ unknowns of cluster i in $\mathbf{Z}_{cur}$. Let this set of unknowns be denoted by i'. We obtain

$$\mathbf{Z}_{cur} = \begin{bmatrix} \mathbf{L}_{i'i'} & \mathbf{0} \\ \mathbf{Z}_{ci'}\mathbf{U}_{i'i'}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \widetilde{\mathbf{Z}}_{cc} \end{bmatrix} \begin{bmatrix} \mathbf{U}_{i'i'} & \mathbf{L}_{i'i'}^{-1}\mathbf{Z}_{i'c} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \tag{4.28}$$

where $\mathbf{F}_{i'i'} = \mathbf{L}_{i'i'}\mathbf{U}_{i'i'}$, and $\widetilde{\mathbf{Z}}_{cc}$ is a Schur complement. Each fill-in block is again either directly added, if the to-be added block is inadmissible, or stored temporarily in the corresponding admissible block. Equation (4.28) can further be written in short as

$$\mathbf{Z}_{cur} = \mathbf{L}_i^l \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \widetilde{\mathbf{Z}}_{cc} \end{bmatrix} \mathbf{U}_i^l. \tag{4.29}$$

At a non-leaf level l, each of $\mathbf{L}_{i'i'}$ and $\mathbf{U}_{i'i'}$ is of size at most $(2k_{l+1} - k_l) \times 2k_{l+1}$. The $\mathbf{Z}_{i'c}$ ($\mathbf{Z}_{ci'}$) has only C_{sp} blocks which are also inadmissible, each of which is of size at most $2k_{l+1} \times 2k_{l+1}$. The Schur complement involves C_{sp}^2 fill-in blocks, each of which is of size at most $2k_{l+1}$ by $2k_{l+1}$. Therefore, generating the L and U factors, as well as computing the fill-in blocks, only involves a constant number of computations, each of which scales as $O({k_l}^3)$ at a non-leaf level. The storage of the $\mathbf{L}$ and $\mathbf{U}$ factors, as well as the fill-in blocks, only takes $O({k_l}^2)$ multiplied by constant $O(C_{sp}^2)$.

Updating the coupling matrix at current level and the transfer matrix at one level higher

After the computation at the current non-leaf level l is finished, before proceeding to the next level, $l - 1$, we need to update the coupling matrices at current level, as well as the transfer matrices at one level higher. This is because the transfer matrices have been updated at the current level.

Similar to the procedure at the leaf level, for admissible blocks without fill-ins, their coupling matrices can be readily updated by appending zeros, as shown in (4.17). For admissible blocks with fill-ins, their coupling matrices are updated by adding the fill-in part onto the augmented coupling matrix $\widetilde{\mathbf{S}}_{i,k}$ as

$$\widetilde{\mathbf{S}}_{i,k} = \widetilde{\mathbf{S}}_{i,k} + \widetilde{\mathbf{T}}_i^H \mathbf{F}_{i,k} \overline{\widetilde{\mathbf{T}}}_k, \tag{4.30}$$

in which $\mathbf{F}_{i,k}$ represents the fill-in block associated with the admissible block formed between clusters i and k.

We also need to update the transfer matrices at one level higher, i.e., at the $l-1$ level. This can be readily done by adding zeros to the original transfer matrices, corresponding to the matrix block of $\mathbf{T}_i^{add}$.

4.2.3 Algorithm Summary

The aforementioned computation at the non-leaf level is continued level by level until we reach the minimal level that has admissible blocks, denoted by l_0. The overall procedure can be realized by the pseudo-code shown in **Algorithm 1**, where the detail of each step can be found from previous section.

The aforementioned direct solution procedure results in the following factorization of $\mathbf{Z}$:

$$\begin{aligned} \mathbf{Z} &= \mathcal{L}\mathcal{U}, \\ \mathcal{L} &= \left\{ \prod\nolimits_{l=L}^{l_0} \left[\left(\prod\nolimits_{i=1}^{2^l} \mathbf{Q}_i^l \mathbf{L}_i^l \right) \mathbf{P}_l^T \right] \right\} \mathbf{L}, \\ \mathcal{U} &= \mathbf{U} \left\{ \prod\nolimits_{l=l_0}^{L} \left[\mathbf{P}_l \left(\prod\nolimits_{i=2^l}^{1} \mathbf{U}_i^l \mathbf{Q}_i^{l^T} \right) \right] \right\}. \end{aligned} \tag{4.31}$$

In the above, $\mathbf{Q}_i^l$ denotes $\mathbf{Q}_i$ at level l. For leaf level, it is shown in (4.3), and for a non-leaf level, it is shown in (4.26). Each $\mathbf{Q}_i^l$ has only one block that is not identity, whose size is $2k_{l+1} \times 2k_{l+1}$ at the non-leaf level, and $leafsize \times leafsize$ at the leaf level. Each $\mathbf{L}_i^l$ is shown in (4.10) or (4.29), which has $O(C_{sp})$ blocks of *leafsize* at the leaf level, and $O(C_{sp})$ blocks of $O(k_{l+1})$ size at a non-leaf level. The same is true to $\mathbf{U}_i^l$. The $\mathbf{L}$ and $\mathbf{U}$ without sub- and super-scripts are the $\mathbf{L}$ and $\mathbf{U}$ factors of the final

Algorithm 1 Proposed direct solution of general $\mathcal{H}^2$-matrices

1: Let $\mathbf{Z}_{cur} = \mathbf{Z}$
2: **for** $l = L$ **to** l_0 **do**
3: **for** *cluster* : i =1 **to** 2^l **do**
4: Update cluster basis (Step 0)
5: Obtain projection matrix $\mathbf{Q}_i$ (Step 1)
6: Do $\mathbf{Z}_{cur} = \mathbf{Q}_i^H \mathbf{Z}_{cur} \overline{\mathbf{Q}}_i$ (Step 2)
7: Perform partial LU $\mathbf{Z}_{cur} = \mathbf{L}_i^l \mathbf{Z}_{cnt} \mathbf{U}_i^l$ (Step 3)
8: Let $\mathbf{Z}_{cur} = \mathbf{Z}_{cnt}$
9: **end for**
10: Update coupling matrices at level l
11: Update transfer matrices at level $l-1$
12: Permute and merge small $k_l \times k_l$ matrices to next level
13: **end for**
14: Do LU on the final block, $\mathbf{Z}_{cur} = \mathbf{LU}$.

matrix that remains to be factorized in level l_0. The non-identity blocks in $\mathbf{L}$ and $\mathbf{U}$ are of size $O(2^{l_0} k_{l_0})$. The $\mathbf{P}_l$ matrix is a permutation matrix that merges small $k_l \times k_l$ matrices at level l to one level higher.

The factorization shown in (4.31) also results in an explicit form of $\mathbf{Z}$'s inverse, which can be written as

$$\begin{aligned} &\mathbf{Z}^{-1} = \\ &\left\{ \prod_{l=L}^{l_0} \left[\left(\prod_{i=1}^{2^l} \overline{\mathbf{Q}}_i^l (\mathbf{U}_i^l)^{-1} \right) \mathbf{P}_l^T \right] \right\} \mathbf{U}^{-1} \times \\ &\mathbf{L}^{-1} \left\{ \prod_{l=l_0}^{L} \left[\mathbf{P}_l \left(\prod_{i=2^l}^{1} (\mathbf{L}_i^l)^{-1} \mathbf{Q}_i^{lH} \right) \right] \right\}. \end{aligned} \tag{4.32}$$

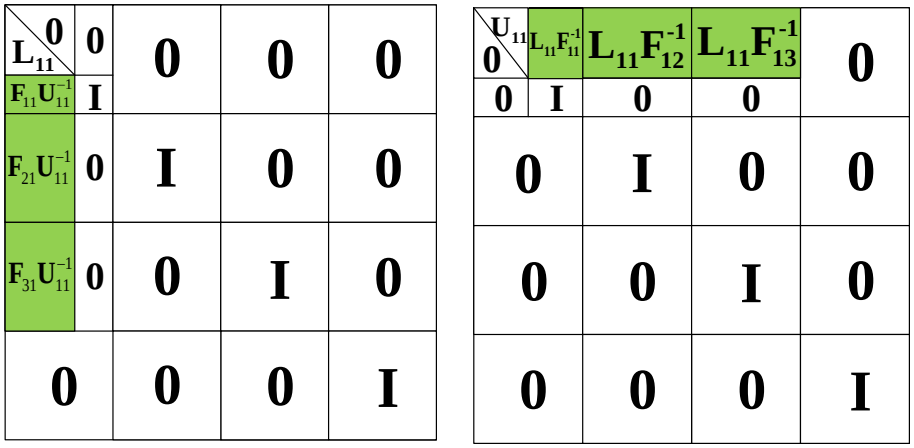
In addition, in the proposed direct solution, one can also flexibly stop at a level before l_0 is reached. This may be desired if at a tree level, the number of zeros introduced is small as compared to the block size at that level. Although one can still proceed to the next level, the efficiency gain may be smaller than that gained if one

just stops at the current level, and finishes the factorization. This typically happens at a few levels lower than the l_0 level.

4.3 Proposed Matrix Solution (Backward and Forward Substitution) Algorithm

After factorization, the solution of $\mathbf{Z}x = b$ can be obtained in two steps: $\mathcal{L}y = b$ which can be called as a backward substitution, and $\mathcal{U}x = y$ called as a forward substitution. Rigorously speaking, since projection matrix $\mathbf{Q}_i^l$ and permutation matrix $\mathbf{P}_l$ are involved in $\mathcal{L}$ and $\mathcal{U}$ factors, the matrix solution here is not a conventional forward/backward substitution.

4.3.1 Forward Substitution



(a) (b)

Fig. 4.6.: (a) $\mathbf{L}_1$ matrix. (b) $\mathbf{U}_1$ matrix.

In this step, we solve

$$\mathcal{L}y = b. \tag{4.33}$$

From (4.32), it can be seen that the above can be solved as

$$y = \mathcal{L}^{-1}b = \mathbf{L}^{-1}\left\{\prod_{l=l_0}^{L}\left[\mathbf{P}_l\left(\prod_{i=2^l}^{1}(\mathbf{L}_i^l)^{-1}\mathbf{Q}_i^{lH}\right)\right]\right\}b. \tag{4.34}$$

Take a 3-level $\mathcal{H}^2$-matrix as an example, the above is nothing but

$$y = \mathbf{L}^{-1}\mathbf{P}_1\mathbf{L}_4^{-1}\mathbf{Q}_4^H\mathbf{L}_3^{-1}\mathbf{Q}_3^H\mathbf{L}_2^{-1}\mathbf{Q}_2^H\mathbf{L}_1^{-1}\mathbf{Q}_1^Hb. \tag{4.35}$$

Obviously, three basic operations are involved in the forward substitution: multiplying $(\mathbf{Q}_i^l)^H$ by a vector, multiplying $(\mathbf{L}_i^l)^{-1}$ by a vector, and $\mathbf{P}_l$-based permutation.

Algorithm 2 Forward Substitution $b \leftarrow \mathcal{L}^{-1}b$

1: **for** $l = L$ **to** l_0 **do**
2: **for** *cluster* : $i = 1$ **to** 2^l **do**
3: Update b's b_i part by $b_i \leftarrow (\widetilde{\mathbf{Q}_i}^l)^H b_i$.
4: Do $b \leftarrow (\mathbf{L}_i^l)^{-1}b$
5: **end for**
6: Permute b by multiplying with $\mathbf{P}_l$
7: **end for**
8: Do $b \leftarrow \mathbf{L}^{-1}b$

The algorithm of the forward substitution is shown in **Algorithm 2**. The $(\mathbf{L}_i^l)^{-1}b$ in step 4 of this algorithm only involves $O(C_{sp})$ matrix-vector multiplications of *leaf-size* or rank size associated with cluster i at level l. To see this point more clearly, we can use the $\mathbf{L}_1$ matrix shown in Fig. 4.6(a) as an example, where $\mathbf{L}_1$ is resultant from the partial factorization of cluster 1 in an $\mathcal{H}^2$-matrix. It can be seen that $\mathbf{L}_1$ has only $O(C_{sp})$ blocks that need to be computed and stored, each of which is *leafsize* at the leaf level, or rank size at a non-leaf level l. These blocks are the inadmissible blocks

formed with cluster i. For such type of $\mathbf{L}_i$, the $\mathbf{L}_i^{-1}b$ can be readily computed. For $\mathbf{L}_1$ shown in Fig. 4.6(a), the $\mathbf{L}_1^{-1}b$ is

$$\begin{aligned}\mathbf{L}_1^{-1}b &= \begin{bmatrix} \begin{bmatrix} \mathbf{L}_{11}^{-1} & \mathbf{0} \\ -\mathbf{T}_{11}\mathbf{L}_{11}^{-1} & \mathbf{I} \end{bmatrix} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \begin{bmatrix} -\mathbf{T}_{21}\mathbf{L}_{11}^{-1} & \mathbf{0} \end{bmatrix} & \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \begin{bmatrix} -\mathbf{T}_{31}\mathbf{L}_{11}^{-1} & \mathbf{0} \end{bmatrix} & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \begin{bmatrix} b_{11} \\ b_{12} \end{bmatrix} \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \\ &= \begin{bmatrix} \begin{bmatrix} \widetilde{b}_{1,1} \\ b_{1,2} - \mathbf{T}_{11}\widetilde{b}_{1,1} \end{bmatrix} \\ b_2 - \mathbf{T}_{21}\widetilde{b}_{1,1} \\ b_3 - \mathbf{T}_{31}\widetilde{b}_{1,1} \\ b_4 \end{bmatrix} \end{aligned} \tag{4.36}$$

where $\mathbf{T}_{i1} = \mathbf{F}_{i1}\mathbf{U}_{11}^{-1}$ $(i = 1, 2, 3)$, and $\widetilde{b}_{1,1} = \mathbf{L}_{11}^{-1}b_{1,1}$. Therefore, this operation actually only involves $O(C_{sp})$ matrix-vector multiplications related to clusters 1, 2 and 3, which are those clusters that form inadmissible blocks with cluster 1, the number of which is bounded by C_{sp}. After forward substitution, the right hand side b vector is overwritten by solution y.

4.3.2 Backward Substitution

After finishing forward substitution, we solve

$$\mathcal{U}x = y \tag{4.37}$$

From (4.32), it can be seen that the above can be solved as

$$x = \mathcal{U}^{-1}y = \left\{ \prod_{l=L}^{l_0} \left[\left(\prod_{i=1}^{2^l} \overline{\mathbf{Q}}_i^l \mathbf{U}_i^{l^{-1}} \right) \mathbf{P}_l^T \right] \right\} \mathbf{U}^{-1}y. \tag{4.38}$$

For the same 3-level $\mathcal{H}^2$-matrix example, the above is nothing but

$$x = \overline{\mathbf{Q}}_1\mathbf{U}_1^{-1}\overline{\mathbf{Q}}_2\mathbf{U}_2^{-1}\overline{\mathbf{Q}}_3\mathbf{U}_3^{-1}\overline{\mathbf{Q}}_4\mathbf{U}_4^{-1}\mathbf{P}_1^T\mathbf{U}^{-1}y. \tag{4.39}$$

The backward substitution starts from minimal admissible block level l_0, and finishes at the leaf level, as shown in **Algorithm 3**. The $(\mathbf{U}_i^l)^{-1}y$ in step 5 is performed efficiently similar to how (4.36) is used to compute $(\mathbf{L}_i^l)^{-1}b$, which also only involve $O(C_{sp})$ dense blocks as the example shown in Fig. 4.6(b). There are also three basic operations in **Algorithm 3**: $(\mathbf{U}_i^l)^{-1}$-based matrix-vector multiplication, $\overline{\mathbf{Q}}_i^l$-based matrix-vector multiplication, and $\mathbf{P}_l$-based permutation. After the backward substitution is done, y is overwritten by solution x.

Algorithm 3 Backward Substitution $y \leftarrow \mathcal{U}^{-1}y$

1: Do $y \leftarrow \mathbf{U}^{-1}y$
2: **for** $l = l_0$ **to** L **do**
3: Permute y by front multiplying $\mathbf{P}_l^T$
4: **for** $clusteri = 2^l$ **to** 1 **do**
5: Do $y \leftarrow (\mathbf{U}_i^l)^{-1}y$
6: Update $y_i \leftarrow (\overline{\mathbf{Q}}_i^l)y_i$.
7: **end for**
8: **end for**

4.3.3 Accuracy and Complexity Analysis

In this section, we analyze the accuracy and computational complexity of the proposed direct solver.

Accuracy

When generating an $\mathcal{H}^2$-matrix to represent the original dense matrix, the accuracy is controlled by $\epsilon_{\mathcal{H}^2}$, which is the same as the accuracy parameter used in equations (6), (7), (11), and (15) of [16]. The accuracy of the proposed direct solution is controlled by $\epsilon_{fill-in}$. This is different from [9, 11, 15], since it is directly controlled.

As can be seen from chapter 4.2, there are no approximations involved in the proposed solution procedure, except that when we update the cluster bases to account for the contribution from the fill-ins, we use (4.14) or (4.23) to truncate the singular vectors. The accuracy of this step is controlled by parameter $\epsilon_{fill-in}$, which can be set to any desired value.

Time and Memory Complexity

From (4.31), it is evident that the whole computation involves $\mathbf{Q}_i^l$, $\mathbf{L}_i^l$, and $\mathbf{U}_i^l$ for each cluster i at every level l, the storage and computation of each of which are respectively $O(k_l^2)$, and $O(k_l^3)$ at a non-leaf level, and constant at the leaf level. Recall that $\mathbf{Q}_i^l$ is obtained from Step 0 and Step 1, $\mathbf{L}_i^l$, and $\mathbf{U}_i^l$ are obtained from Step 3, whose underlying blocks are generated in Step 2. As for the permutation matrix $\mathbf{P}_l$, it is sparse involving only $O(2^l)$ integers to store at level l, and hence having an $O(N)$ cost in both memory and time.

From another perspective, the overall procedure of the proposed direct solution is shown in **Algorithm 1**. It involves $O(L)$ levels of computation. At each level, there are 2^l clusters. For each cluster, we have performed four steps of computation from Step 0, 1, 2, to 3, as shown by the tasks listed at the end of lines 4–7 in **Algorithm 1**. Each of these steps costs $O(k_l)^3$ for each cluster, as analyzed in chapter 4.2. After each level is done, we update coupling matrices, and transfer matrices at one-level higher, the cost of which is also $O(k_l)^3$ for every admissible block, or cluster. So the factorization and inversion complexity are similar as shown in chapter 2.2.2.

4.4 Numerical Results

In order to demonstrate the accuracy and low computational complexity of the proposed direct solution of general $\mathcal{H}^2$-matrices, we apply it to solve VIEs for electromagnetic analysis. Both circuit and large-scale scattering problems are simulated. The VIE formulation we use is based on [31] with SWG vector bases for expanding

electric flux density in each tetrahedral element. First, the scattering from a dielectric sphere, whose analytical solution is known, is simulated to validate the proposed direct solver. Then, a variety of large-scale examples involving over one million unknowns are simulated on a single CPU core to examine the accuracy and complexity of the proposed direct solver. The $\mathcal{H}^2$-matrix for each example is constructed based on the method described in [14–16]. The accuracy parameter $\epsilon_{\mathcal{H}^2}$ used is 10^{-3} for the $\mathcal{H}^2$-matrix construction. The accuracy parameter $\epsilon_{fill-in}$ used in the direct solution is varied to examine the error controllability of the proposed direct solution. Frobenius norm is used whenever norm is calculated.

4.4.1 Scattering from Dielectric Sphere

We simulate a dielectric sphere of radius a with different electric sizes k_0a and dielectric constant ϵ_r. The incident electric field is a normalized $-z$ propagating plane wave polarized along the x-direction. This incident field is also used in the following examples. During the $\mathcal{H}^2$-matrix construction, $leafsize = 25$ and $\eta = 1$ are used. The accuracy parameter in the direct solution is set to be $\epsilon_{fill-in} = 10^{-6}$. In Fig. 4.7, the radar cross section (RCS in dBsm) of the sphere is plotted as a function of θ for zero azimuth for $k_0a = 0.408, \epsilon_r = 36$; $k_0a = 0.408, \epsilon_r = 4$; $k_0a = 0.816, \epsilon_r = 4$; and $k_0a = 1.632, \epsilon_r = 4$ respectively. They all show good agreement with the analytical Mie Series solution, which validates the proposed direct solution.

We have also investigated the accuracy of the proposed direct solution as a function of the accuracy-control parameter $\epsilon_{fill-in}$. In Table 4.1, we list the relative residual error for all the four sphere examples simulated using $\epsilon_{fill-in} = 10^{-4}, 10^{-5}$, and 10^{-6} respectively. Examples 1, 2, 3, and 4 in Table 4.1 correspond to Fig. 4.7(a), (b), (c), and (d) respectively. As can be seen from Table 4.1, the smaller the $\epsilon_{fill-in}$, the better the solution accuracy, verifying the error controllability of the proposed direct solution. We have also compared RCS obtained from this direct solver against the

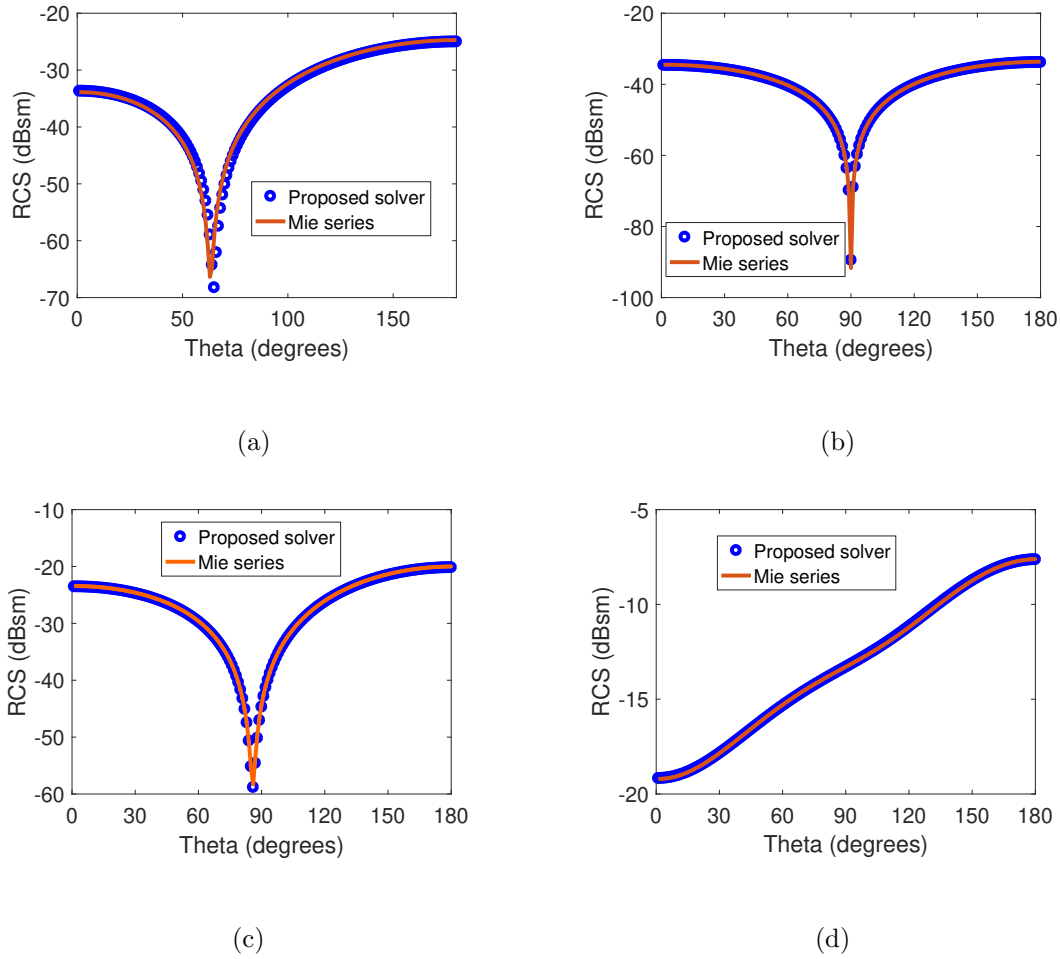


Fig. 4.7.: Simulated RCS of a dielectric sphere for different electric sizes and dielectric constants. (a) $k_0a = 0.408$, $\epsilon_r = 36.0$. (b) $k_0a = 0.408$, $\epsilon_r = 4.0$. (c) $k_0a = 0.816$, $\epsilon_r = 4.0$. (d) $k_0a = 1.632$, $\epsilon_r = 4.0$.

analytical Mie Series solution for $\epsilon_{fill-in} = 10^{-4}, 10^{-5}$, and 10^{-6}. The difference only appears after the fourth- or fifth-digit.

Table 4.1.: Direct solution error measured by relative residual ϵ_{rel} of the dielectric sphere as a function of $\epsilon_{fill-in}$.

$\epsilon_{fill-in}$	Ex. 1	Ex. 2	Ex. 3	Ex. 4
10^{-6}	0.01%	0.002%	0.003%	0.011%
10^{-5}	0.04%	0.006%	0.01%	0.04%
10^{-4}	0.11%	0.015%	0.024%	0.1%

4.4.2 Large-scale Dielectric Slab

Simulation using proposed direct solution

We then simulate a dielectric slab with $\epsilon_r = 2.54$ at 300 MHz, which is also simulated in [14, 15]. The thickness of the slab is fixed to be $0.1\lambda_0$. The width and length are simultaneously increased from $4\lambda_0$, $8\lambda_0$, $16\lambda_0$, to $32\lambda_0$. With a mesh size of $0.1\lambda_0$, the resultant N ranges from 22,560 to 1,434,880 for this suite of slab structures. The *leafsize* is chosen to be 25 and $\eta = 1$. The $\epsilon_{fill-in}$ is set to be 10^{-2}, 10^{-4}, and 10^{-6} respectively to examine the solution accuracy, computational complexity, and error controllability of the proposed direct solution.

Notice that this example represents a 2-D problem characteristics. Based on [32], the rank's growth rate with electric size is lower than linear, and being a square-root of the log-linear of the electric size. Substituting such a rank's growth rate into the complexity analysis shown in (2.23) and (2.24), we will obtain linear complexity in both memory and time. The rank numerically found is 12, 17, 28, and 49 respectively for the four slab examples, whose growth rate with electrical size is clearly no greater than linear.

In Fig. 4.8(a), we plot the factorization time with respect to N, for all three different choices of $\epsilon_{fill-in}$. It is clear that the smaller the $\epsilon_{fill-in}$, the larger the factorization time. However, the complexity remains the same as linear regardless of the choice of $\epsilon_{fill-in}$. In addition, the factorization time is not increased much by

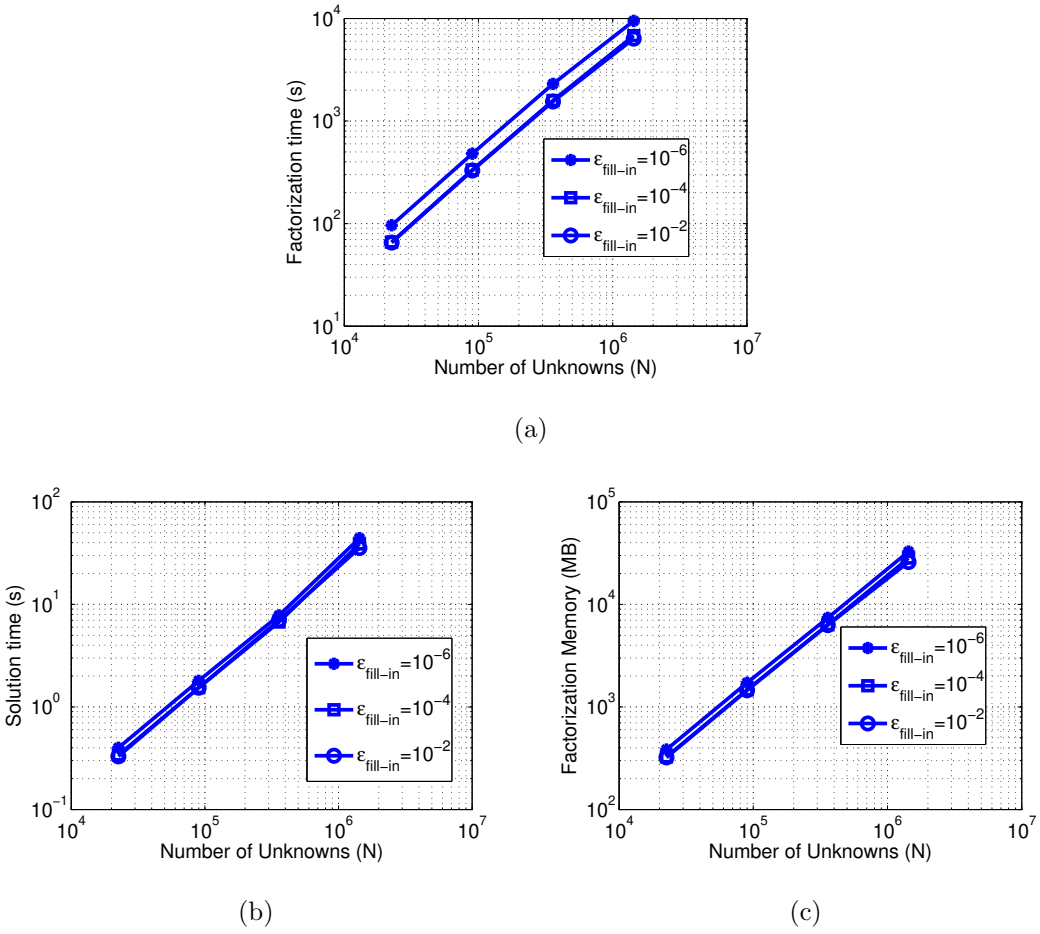


Fig. 4.8.: Solver performance of a large-scale dielectric slab for different choices of $\epsilon_{fill-in}$. (a) Factorization time v.s. N. (b) Solution time v.s. N. (c) Memory v.s. N.

decreasing $\epsilon_{fill-in}$. This is because the $\epsilon_{fill-in}$ is used to append the original cluster bases with additional ones to capture the fill-ins' contribution. When the number of additional cluster bases added is small, the cost is still dominated by the original number of cluster bases. The solution time and memory cost are plotted in Fig. 4.8(b), and Fig. 4.8(c) respectively. Obviously, both scale linearly with the number of unknowns. The error of the proposed direct solution is measured by computing the relative residual $\epsilon_{rel} = ||\mathbf{Z}_{\mathcal{H}^2}x - b||/||b||$, where $\mathbf{Z}_{\mathcal{H}^2}$ is the input $\mathcal{H}^2$-matrix to be solved. The relative residual ϵ_{rel} of the proposed direct solution is listed in Table 4.2 as a function of $\epsilon_{fill-in}$. Excellent accuracy can be observed in the entire unknown

range. Furthermore, the accuracy can be controlled by $\epsilon_{fill-in}$, and overall smaller $\epsilon_{fill-in}$ results in better accuracy.

Table 4.2.: Direct solution error measured by relative residual, ϵ_{rel}, for the dielectric slab example.

$\epsilon_{fill-in}$	10^{-2}	10^{-4}	10^{-6}
ϵ_{rel} $(N = 22,560)$	0.86%	0.19%	0.03%
ϵ_{rel} $(N = 89,920)$	1.32%	0.25%	0.057%
ϵ_{rel} $(N = 359,040)$	3.59%	1.88%	0.66%
ϵ_{rel} $(N = 1,434,880)$	2.68%	1.05%	0.56%

Table 4.3.: Performance comparison between this solver with $\epsilon_{fill-in} = 10^{-5}$ and [14–16] for the dielectric slab example.

N	89,920	359,040
Factorization (s) [This]	353	1,662
Solution (s) [This]	1.68	7.07
Inversion (s) [14–16]	2.75e+03	1.65e+04
Relative Residual [This]	0.18%	0.40%
Inverse Error [14–16]	3.9%	7.49%

Comparison with direct solvers in [14–16]

Since this example is also simulated in our previous work [14–16], we have compared the two direct solvers in CPU run time and accuracy. The $\epsilon_{fill-in} = 10^{-5}$ is used. For a fair comparison, we set up this solver to solve the same $\mathcal{H}^2$-matrix solved in [16], and also use the same computer. As can be seen from Table 4.3, the proposed new direct solution takes much less time than that of [14–16]. The speedup is more

than one order of magnitude in large examples. Even though the factorization time is shown, as can be seen from (4.32), the inversion time in the proposed algorithm is similar to factorization time. In addition, because of a direct error control, the error of the proposed solution is also much less than that of [14–16].

4.4.3 Large-scale Array of Dielectric Cubes

Simulation with proposed direct solution

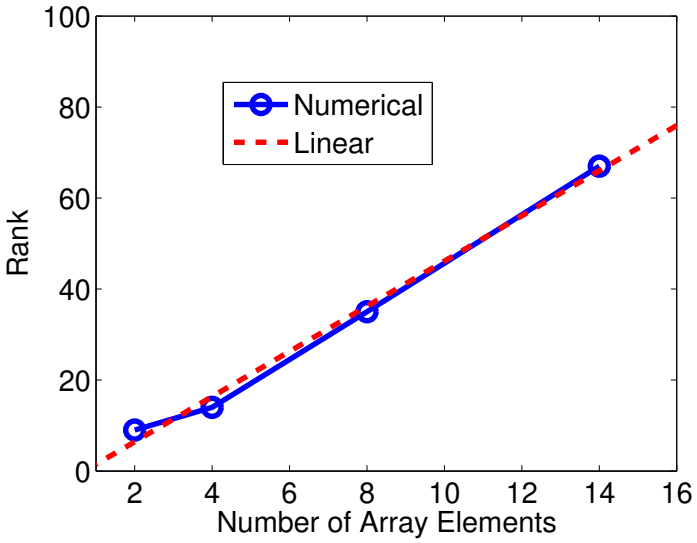


Fig. 4.9.: Rank of the dielectric cube array example v.s. number of array elements.

Next, we simulate a large-scale array of dielectric cubes at 300 MHz. The relative permittivity of the cube is $\epsilon_r = 4.0$. Each cube is of size $0.3\lambda_0 \times 0.3\lambda_0 \times 0.3\lambda_0$. The distance between adjacent cubes is kept to be $0.3\lambda_0$. The number of the cubes is increased along the x-, y-, and z-direction simultaneously from 2 to 14, thus producing a 3-D cube array from $2 \times 2 \times 2$ to $14 \times 14 \times 14$ elements. The number of unknowns N is respectively 3,024, 24,192, 193,536, and 1,037,232 for these arrays. The *leafsize* is 25 and $\eta = 1$. The $\epsilon_{fill-in}$ is chosen as 10^{-3}, 10^{-4}, and 10^{-5} respectively.

For the cubic growth of unknowns for 3-D problems, we observe that constant C_{sp} is not saturated until in the order of millions of unknowns, as can be seen from Table 4.4. It is thus important to analyze the performances of the proposed direct solver as (Memory or Solution time)/C_{sp} and (Factorization time)/C_{sp}^2 respectively.

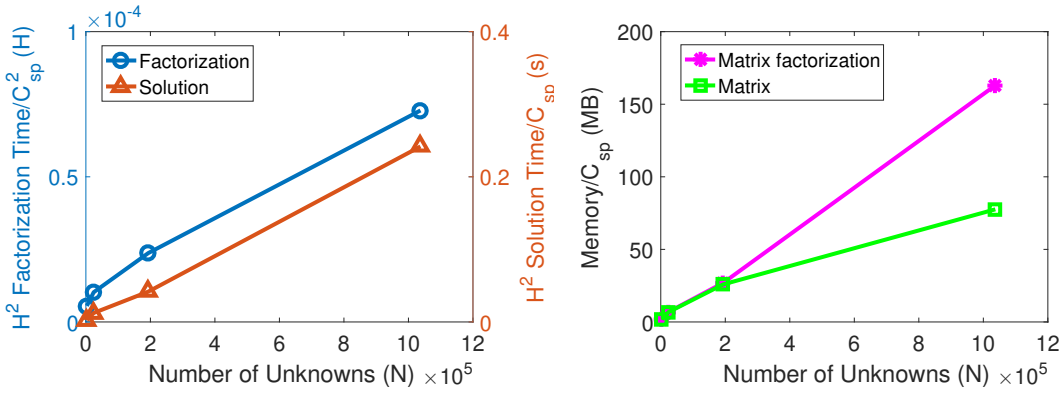
Table 4.4.: C_{sp} as a function of N for the dielectric cube array.

N	3,024	24,192	193,536	378,000	1,037,232
C_{sp}	16	42	95	327	270

As for the rank of this example, we plot the maximal rank of each unknown case in Fig. 4.9. It can be clearly seen that the rank's growth rate with the number of array elements, thereby electrical size, is no greater than linear. For such a growth rate, the resultant complexities are given in (2.25) and (2.26). In Fig. 4.10(a) and Fig. 4.10(b), we plot the direct factorization time normalized by C_{sp}^2, and the storage cost normalized with C_{sp} with respect to N. As can be seen, their scaling rate with N agrees very well with our theoretical complexity analysis regardless of the choice of $\epsilon_{fill-in}$. The relative residual ϵ_{rel} of the proposed direct solution is listed in Table 4.5 for this example, which again reveals excellent accuracy and error controllability of the proposed direct solution.

Table 4.5.: Direct solution error measured by relative residual for the cube array.

$\epsilon_{fill-in}$	10^{-3}	10^{-4}	10^{-5}
ϵ_{rel} $(N=3,024)$	0.27%	0.14%	0.09%
ϵ_{rel} $(N=24,192)$	0.74%	0.31%	0.15%
ϵ_{rel} $(N=193,536)$	2.22%	0.88%	0.36%
ϵ_{rel} $(N=1,037,232)$	5.32%	4.02%	1.3%



(a) (b)

Fig. 4.10.: Solver performance of a suite of cube arrays for different choices of $\epsilon_{fill-in}$. (a) Factorization time v.s. N. (b) Memory v.s. N.

Comparison with direct solvers in [14–16]

Since this example with dielectric constant $\epsilon_r = 2.54$ is also simulated in our previous work [16], we have compared the two direct solvers in CPU run time and accuracy. As can be seen from Table 4.6, the proposed new direct solution again takes much less time, while achieving better accuracy.

Table 4.6.: Performance comparison with [14–16] for the dielectric cube array example.

N	3024	24,192	193,536
Factorization (s) [This]	4.24	52.6	503
Solution (s) [This]	0.03	0.3	3.07
Inversion (s) [14–16]	13.95	444.01	1.15e+04
Relative Residual [This]	0.04%	0.13%	0.13e%
Inverse Error [14–16]	0.9%	1.73%	3.03%

4.5 Conclusion

In this chapter, we develop fast factorization and inversion algorithms for general $\mathcal{H}^2$ matrices. The key features of this new direct solution are its directly controlled accuracy and low computational complexity. The entire direct solution does not involve any approximation, except that the fill-in block is compressed to update the cluster basis. However, this compression is well controlled by accuracy parameter $\epsilon_{fill-in}$. For constant-rank $\mathcal{H}^2$ matrices, the proposed direct solution has an $O(N)$ complexity in both time and memory. For rank growing with the electrical size linearly, the proposed direct solution has an $O(N\log N)$ complexity in factorization and inversion, and $O(N)$ in solution time and memory when solving VIEs. The proposed direct solution has been applied to solve electrically large VIEs whose kernel is oscillatory and complex-valued, which has been difficult to solve. Millions of unknowns are directly solved on a single core CPU in fast CPU run time. Comparisons with state-of-the-art $\mathcal{H}^2$-based direct VIE solvers have demonstrated the accuracy of this new direct solution as well as significantly improved computational efficiency. Being a generic direct solution to an $\mathcal{H}^2$ matrix, this algorithm can also be applied to solve other integral equations and partial differential equations.

5. DIRECT SOLUTION OF GENERAL $\mathcal{H}^2$-MATRICES WITH CONTROLLED ACCURACY AND CONCURRENT CHANGE OF CLUSTER BASES FOR ELECTROMAGNETIC ANALYSIS

5.1 Introduction

Many fast solvers have been developed to solve large-scale electromagnetic problems [6, 18, 19, 21–25, 30, 35–37]. To meet real-world modeling and simulation challenges, there is a continued need to accelerate the computation as well as increase the accuracy and modeling capabilities of existing computational electromagnetic methods.

Recently, in [38], a new fast direct solution algorithm with controlled accuracy is developed for solving general $\mathcal{H}^2$ matrices. In this algorithm, the additions and multiplications are performed as they are without using formatted operations. Each operation is either exact or controlled by a prescribed accuracy. However, the cluster bases of the original $\mathcal{H}^2$ matrix are appended with new ones to account for the fill-in's contributions. The resulting rank, i.e., the number of cluster bases can only be higher than the original one instead of lower. If the cluster bases can be changed based on the updated matrix obtained during the direct solution procedure, and with prescribed accuracy, the resulting solver can be potentially more efficient. Despite such an advantage, it is difficult to completely change the cluster bases during the direct solution procedure, since the original nested structure can be ruined, and the computational complexity would become very high.

In this chapter, we develop an efficient direct solution algorithm to overcome the aforementioned challenge. The cluster bases of the original $\mathcal{H}^2$ matrix are completely changed to a new set during the factorization procedure based on accuracy. Mean-

while, the complexity of the overall algorithm is not increased. As a result, the updates to the original matrix during the direct solution procedure are efficiently and accurately represented by the new cluster bases. The proposed new direct solver has been applied to both static capacitance extraction and full-wave scattering analysis. A clear $O(N)$ complexity is observed in factorization, solution time, memory, and with a strictly controlled accuracy for constant-rank $\mathcal{H}^2$ matrices. For variable-rank $\mathcal{H}^2$ matrices like those commonly encountered in electrically large problems, the complexity is also low, and can be analytically derived based on the rank's behavior for a given problem. The solver is also shown to outperform state-of-the-art direct solvers in both accuracy and efficiency. The details of this algorithm are given in chapter 5.3. In addition, we have demonstrated the performance of the proposed new direct solution by a large number of numerical experiments. Detailed comparisons with state-of-the-art direct solvers in both accuracy and efficiency are also performed.

5.2 Proposed Direct Solution

The proposed direct solution is a one-way bottom-up tree traversal of the $\mathcal{H}^2$-tree. At each tree level, the factorization proceeds across all clusters, and for each cluster a partial factorization is performed. We also transform the matrix that remains to be factorized in such a way that the computation is performed only on small matrices of rank size at each tree level. The accuracy of every operation is either exact or strictly controlled by desired accuracy. The whole algorithm is very different from the recursive inverse and formatted multiplications performed in [8, 9]. To make the proposed algorithm easier for understanding, we use the $\mathcal{H}^2$ matrix shown in Fig. 5.1(a) as an example to illustrate the overall procedure. However, the algorithm presented here is generic applicable to any $\mathcal{H}^2$ matrix.

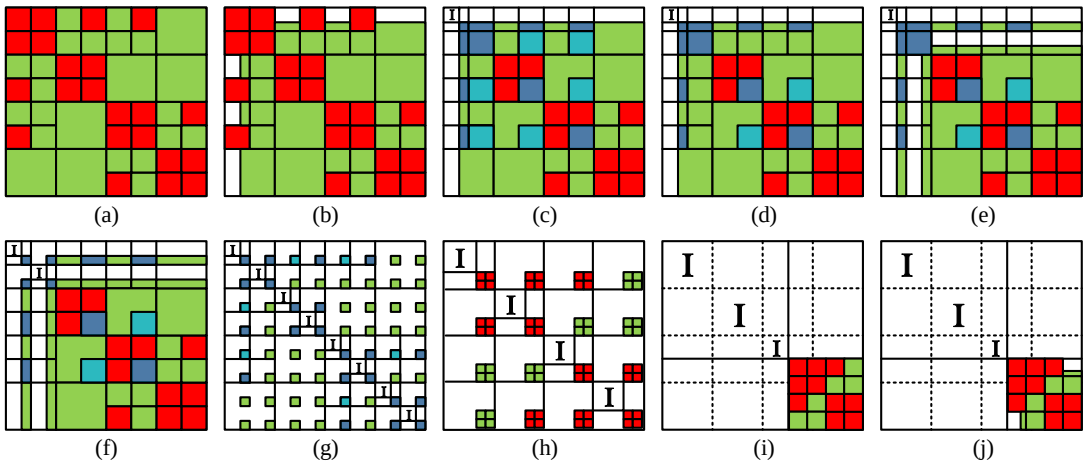


Fig. 5.1.: (a) An $\mathcal{H}^2$-matrix structure. (b) $\mathbf{Q}_1^H \mathbf{Z} \overline{\mathbf{Q}}_1$. (c) After partial LU of cluster 1. (d) After cluster basis 2 update. (e) $\mathbf{Q}_2^H \mathbf{Z} \overline{\mathbf{Q}}_2$. (f) After partial LU of cluster 2. (g) $\mathcal{H}^2$ matrix after leaf-level elimination. (h) Merging to next level. (i) Ancestor level structure. (j) Ancestor level $\mathbf{Q}_t^H \mathbf{Z} \overline{\mathbf{Q}}_t$.

5.2.1 Computation at Leaf Level

Denote the tree level of the leaf level to be L, the computations at level $l = L$ include the following steps:

- Let $\mathbf{Z}_{cur} = \mathbf{Z}$. For each leaf cluster i, do:

  1. Step 1. Change the original cluster basis $\mathbf{V}_i$ to a new cluster basis $\widetilde{\mathbf{V}}_i$ to account for fill-ins, for $i > 1$.
  2. Step 2. Compute the complementary basis $\widetilde{\mathbf{V}}_i^{\perp}$ to form $\widetilde{\mathbf{Q}}_i = [(\widetilde{\mathbf{V}}_i^{\perp})\ (\widetilde{\mathbf{V}}_i)]$.
  3. Step 3. Compute projected matrix $\mathbf{Q}_i^H \mathbf{Z}_{cur} \overline{\mathbf{Q}}_i$.
  4. Step 4. Perform a partial LU factorization to eliminate the first $(\#i - k)$ unknowns of cluster i in $\mathbf{Z}_{cur} = \mathbf{Q}_i^H \mathbf{Z} \overline{\mathbf{Q}}_i$.

$$\mathbf{Q}_i^H \mathbf{Z} \overline{\mathbf{Q}}_i = \begin{bmatrix} \mathbf{L}_{i'i'} & \mathbf{0} \\ \mathbf{F}_{ci'} \mathbf{U}_{i'i'}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{I} & \widetilde{\mathbf{F}}_{cc} \end{bmatrix} \begin{bmatrix} \mathbf{U}_{i'i'} & \mathbf{L}_{i'i'}^{-1} \mathbf{F}_{i'c} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}, \tag{5.1}$$

- Update the coupling matrix of each admissible block at the leaf level
- Merge and permute the matrix to prepare for $\mathbf{Z}_{cur}$ for next-level computation

The details of the aforementioned steps can be found in [38], and the overall procedure is illustrated in Fig. 5.1. In [38], the cluster bases are updated by appending new ones to the original set. Here, we propose an efficient algorithm to completely change cluster bases as follows. To change cluster basis $\mathbf{V}_i$ to accommodate the matrix updates, we first put all the i-related admissible matrix blocks into a single matrix denoted by $\mathbf{Z}_i$. Since the cluster bases need to be nested, the i-related matrix blocks include not only those admissible blocks formed at the same tree level as that of cluster i, but also those admissible blocks formed by the parents of i at all upper levels. Thus, we have

$$\mathbf{Z}_i = \left[\mathbf{Z}_{i,j_1}\ \mathbf{Z}_{i,j_2}\ ...\ \mathbf{Z}^{L-1}_{ip,jp_1}\ \mathbf{Z}^{L-1}_{ip,jp_2}\ ...\ \mathbf{Z}^{l_0}_{ip,jp_1}\ \mathbf{Z}^{l_0}_{ip,jp_2}\ ...\right]. \tag{5.2}$$

where $\mathbf{Z}_{i,j_1}, \mathbf{Z}_{i,j_2}$... are admissible blocks at the tree level of cluster i, and the rest are from upper levels; the superscript represents tree level, and L is the leaf level, while l_0 is the minimum level that has admissible blocks; the ip denotes cluster i's parent clusters at level l, and jp_1, jp_2...stand for the column cluster indices of the admissible blocks formed with row cluster ip. To construct a new cluster basis for cluster i, we compute the Gram matrix of $\mathbf{Z}_i$, which is

$$\mathbf{Z}_{i,2} = \mathbf{Z}_i\mathbf{Z}_i^H, \tag{5.3}$$

The computation of $\mathbf{Z}_{i,2}$ appears to be expensive. In this work, we develop a fast linear-time tree traversal algorithm to obtain $\mathbf{Z}_{i,2}$ efficiently. The detailed procedure will be elaborated in chapter 5.3. After computing $\mathbf{Z}_{i,2}$, we then add the fill-in's contributions to update the cluster basis of cluster i by computing

$$\widetilde{\mathbf{Z}}_{i,2} = \mathbf{Z}_{i,2} + \sum\nolimits_m \mathbf{F}_{i,j_m}\mathbf{F}^H_{i,j_m}, \tag{5.4}$$

where $\mathbf{F}_{i,j_m}$ are fill-in blocks associated with the admissible blocks of cluster i. The $\widetilde{\mathbf{Z}}_{i,2}$ is a small matrix whose size is $leafsize$. We then perform an SVD on $\widetilde{\mathbf{Z}}_{i,2}$, the

resultant singular vector matrix truncated based on accuracy ϵ_{acc} is the new cluster basis for cluster i, denoted by $\widetilde{\mathbf{V}}_i$. Thus,

$$\widetilde{\mathbf{Z}}_{i,2} \overset{\epsilon_{acc}}{\approx} \widetilde{\mathbf{V}}_i \Sigma \widetilde{\mathbf{V}}_i^H . \tag{5.5}$$

Hence, 5.5 can be obtained in constant complexity for each cluster using SVD. This step is reflected in Fig. 5.1(d) where the light blue blocks are turned into green admissible blocks for cluster 2.

5.2.2 Computation at Non-leaf Levels

At a non-leaf level l, the matrix to be factorized is of size $2^l(2k_{l+1})$ by $2^l(2k_{l+1})$, as shown by the bottom right block in Fig. 5.1 (i). This is because it is formed between 2^l clusters at the non-leaf level l, and each block is of size $2k_{l+1} \times 2k_{l+1}$. Obviously, this matrix is again an $\mathcal{H}^2$ matrix, as shown by green admissible blocks and red inadmissible blocks in Fig. 5.1 (j). The difference with the leaf level is that each block, be its inadmissible or admissible, now is of size $2k_{l+1} \times 2k_{l+1}$. Hence, in the proposed direct solution algorithm, at each non-leaf level, the computation is performed on the matrix blocks whose size is the *rank* at that level, and hence efficient.

Take an admissible block formed between clusters t and s at non-leaf level $l = L-1$ an an example. These blocks are shown by the green blocks in Fig. 5.1 (j). Originally, this block has a form of $\mathbf{V}_t \mathbf{S}_{t,s} \mathbf{V}_s^T$. Due to the left multiplication with $\mathbf{Q}_{t_1}^H$ and $\mathbf{Q}_{t_2}^H$, and the right multiplication with $\overline{\mathbf{Q}}_{s_1}$ and $\overline{\mathbf{Q}}_{s_2}$, where $t_{1,2}$ and $s_{1,2}$ are the two children's of t, and s respectively, this admissible block has been zeroed out in some of its rows and columns, leaving the following part only:

$$(\text{Admissible block})_{t,s}^{(L-1)} = \mathbf{T}_t^{new} \mathbf{S}_{t,s} \left(\mathbf{T}_s^{new}\right)^T , \tag{5.6}$$

where

$$\mathbf{T}_t^{new} = \begin{bmatrix} \widetilde{\mathbf{V}}_{t_1}^H \mathbf{V}_{t_1} \mathbf{T}_{t_1} \\ \widetilde{\mathbf{V}}_{t_2}^H \mathbf{V}_{t_2} \mathbf{T}_{t_2} \end{bmatrix} = \begin{bmatrix} \mathbf{B}_{t_1}^T \mathbf{T}_{t_1} \\ \mathbf{B}_{t_2}^T \mathbf{T}_{t_2} \end{bmatrix} \tag{5.7}$$

in which

$$\mathbf{B}_i = \mathbf{V}_i^T \overline{\widetilde{\mathbf{V}}}_i \tag{5.8}$$

is the projection of the new leaf cluster basis of cluster i generated at the leaf level onto the original cluster basis of cluster i. As can be seen from (5.7), $\mathbf{B}_i^T$ can be viewed as a modified cluster basis at the leaf level.

For an admissible block at a level l higher than $L-1$, it has the following form

$$(\text{Admissible block})_{t,s}^{l} = \mathbf{T}_t^{new}\mathbf{T}_t^{L-2}...\mathbf{T}_t^{l}\mathbf{S}_{t,s}\left(\mathbf{T}_s^{l}\right)^T...\left(\mathbf{T}_s^{L-2}\right)^T\left(\mathbf{T}_s^{new}\right)^T, \tag{5.9}$$

where $\mathbf{T}$ without superscript *new* is the original transfer matrix, and the integer index in the superscript denotes the tree level of the transfer matrix.

If we view the current non-leaf level $L-1$ being computed as the leaf level of the tree that remains to be factorized, then from (5.6) and (5.9), it can be seen that $\mathbf{T}_t^{new}$ is the new leaf-level cluster basis, while the transfer matrices at upper levels remain the same as before. We also notice that the new leaf-level cluster basis $\mathbf{T}_t^{new}$ is of dimension $O(2k_{l+1} \times k_l)$, whereas the $\mathbf{S}_{t,s}$ is of size $O(k_l \times k_l)$.

The above is for non-leaf level $L-1$. For an arbitrary non-leaf level l_c, we start the computation from the following bottom-level (we can view it as current leaf level) admissible blocks:

$$(\text{Admissible block})_{t,s}^{(l_c)} = \mathbf{T}_t^{new}\mathbf{S}_{t,s}\left(\mathbf{T}_s^{new}\right)^T, \tag{5.10}$$

where

$$\mathbf{T}_t^{new} = \begin{bmatrix} \widetilde{\mathbf{T}}_{t_1}^{new,H}\mathbf{T}_{t_1}^{new}\mathbf{T}_{t_1} \\ \widetilde{\mathbf{T}}_{t_2}^{new,H}\mathbf{T}_{t_2}^{new}\mathbf{T}_{t_2} \end{bmatrix} = \begin{bmatrix} \mathbf{B}_{t_1}^T\mathbf{T}_{t_1} \\ \mathbf{B}_{t_2}^T\mathbf{T}_{t_2} \end{bmatrix}, \tag{5.11}$$

in which

$$\mathbf{B}_i = \left(\mathbf{T}_i^{new}\right)^T \overline{\widetilde{\mathbf{T}}}_i^{new} \tag{5.12}$$

is the projection of the new cluster basis of cluster i generated at previous level (l_c+1 level) onto the original cluster basis of cluster i at previous level (l_c+1 level). Similar to (5.9), for an admissible block at a level l higher than l_c, it has the following form

$$(\text{Admissible block})_{t,s}^{l} = \mathbf{T}_t^{new}\mathbf{T}_t^{l_c-1}...\mathbf{T}_t^{l}\mathbf{S}_{t,s}\left(\mathbf{T}_s^{l}\right)^T...\left(\mathbf{T}_s^{l_c-1}\right)^T\left(\mathbf{T}_s^{new}\right)^T, \tag{5.13}$$

where $\mathbf{T}$ without superscript *new* is the original transfer matrix.

Since at every level, we are factorizing an $\mathcal{H}^2$ matrix, the computation at a non-leaf level l follows what is done at the leaf level, which is summarized below:

- For each cluster i, do
  1. Step 1. Change transfer matrix $\mathbf{T}_i^{new}$ to a new $\widetilde{\mathbf{T}}_i$ to account for fill-ins (algorithm described in Section 5.3)
  2. Step 2. Compute the complementary basis $\widetilde{\mathbf{T}}_i^{\perp}$ and build $\widetilde{\mathbf{Q}}_i$
  3. Step 3. Compute $\mathbf{Q}_i^H \mathbf{Z}_{cur} \overline{\mathbf{Q}}_i$ to introduce zeros into the admissible blocks of i, which only involves the computation of $O(C_{sp})$ blocks of rank size
  4. Step 4. Let $\mathbf{Z}_{cur} = \mathbf{Q}_i^H \mathbf{Z}_{cur} \overline{\mathbf{Q}}_i$. Perform a partial LU factorization of $\mathbf{Z}_{cur}$ to eliminate the first $(2k_{l+1} - k_l)$ unknowns in cluster i, where k_l is the rank of $\widetilde{\mathbf{T}}_i$
- Update the coupling matrix of each admissible block at current level
- Merge and permute the matrix to prepare for $\mathbf{Z}_{cur}$, the matrix that remains to be factorized

The aforementioned computation is continued level by level until we reach the minimal level $l = l_0$ that has admissible matrix blocks.

5.2.3 Overall Factorization

The overall procedure, results in the following factorization of $\mathbf{Z}$:

$$\begin{aligned} \mathbf{Z} &= \mathcal{L}\mathcal{U}, \\ \mathcal{L} &= \left\{ \prod_{l=L}^{l_0} \left[\left(\prod_{i=1}^{2^l} \mathbf{Q}_i^l \mathbf{L}_i^l \right) \mathbf{P}_l^T \right] \right\} \mathbf{L}, \\ \mathcal{U} &= \mathbf{U} \left\{ \prod_{l=l_0}^{L} \left[\mathbf{P}_l \left(\prod_{i=2^l}^{1} \mathbf{U}_i^l \mathbf{Q}_i^{l^T} \right) \right] \right\}. \end{aligned} \tag{5.14}$$

In the above, $\mathbf{Q}_i^l$ denotes $\mathbf{Q}_i$ at level l. Each $\mathbf{Q}_i^l$ has only one block that is not identity, whose size is $2k_{l+1} \times 2k_{l+1}$ at the non-leaf level, and $leafsize \times leafsize$ at the leaf

level. Each $\mathbf{L}_i^l$ has $O(C_{sp})$ blocks of $leafsize$ at the leaf level, and $O(C_{sp})$ blocks of $O(k_{l+1})$ size at a non-leaf level. The same is true to $\mathbf{U}_i^l$. The $\mathbf{L}$ and $\mathbf{U}$ without sub- and super-scripts are the $\mathbf{L}$ and $\mathbf{U}$ factors of the final matrix that remains to be factorized at level l_0. The non-identity blocks in $\mathbf{L}$ and $\mathbf{U}$ are of size $O(2^{l_0}k_{l_0})$ by $O(2^{l_0}k_{l_0})$. For $l_0 = 1$, it is simply $2k_{l_0}$ by $2k_{l_0}$. The $\mathbf{P}_l$ matrix is a permutation matrix that merges small $k_l \times k_l$ matrices at level l to one level higher. The factorization shown in (5.14) also results in an explicit form of $\mathbf{Z}$'s inverse. We can solve the linear equation $\mathbf{Z}x = B$ in two ways. One is to use the inverse to multiply right hand side B. The other is to perform a forward and backward substitution based on (5.14), as shown in [38].

5.3 Algorithm for Concurrent Change of Cluster Bases During Matrix Factorization

In this section, we detail the proposed algorithm on how to change the cluster basis concurrently with the matrix factorization, so that the update to the original matrix during factorization can be accurately and efficiently represented.

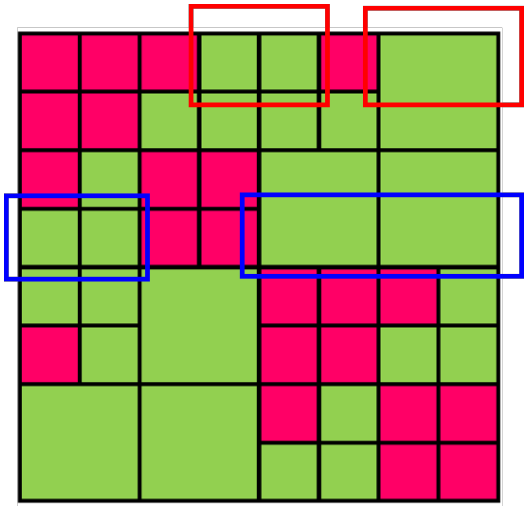


Fig. 5.2.: Illustration of $\mathbf{Z}_i$ for $i = 1$ and $i = 4$.

5.3.1 Concurrent Change of Cluster Bases at Leaf Level

At the leaf level, for each cluster, we assemble a $\mathbf{Z}_i$ matrix as shown in (5.2), which is repeated below for convenience of explanation:

$$\mathbf{Z}_i = \left[\mathbf{Z}_{i,j_1}\ \mathbf{Z}_{i,j_2}\ ...\ \mathbf{Z}^{L-1}_{ip,jp_1}\ \mathbf{Z}^{L-1}_{ip,jp_2}\ ...\ \mathbf{Z}^{l_0}_{ip,jp_1}\ \mathbf{Z}^{l_0}_{ip,jp_2}\ ...\right]. \tag{5.15}$$

This is nothing but all the admissible blocks formed by cluster i at its tree level as well as those in i's parent levels. In Fig. 5.2, we give two examples: $\mathbf{Z}_i$ for cluster $i = 1$ highlighted in red rectangular box, and that for cluster $i = 4$ highlighted in blue box. In [39–41], we used the admissible blocks in the original matrix to construct $\mathbf{Z}_i$. In this case, $\mathbf{Z}_{i,j_m}$ and $\mathbf{Z}^l_{ip,jp_m}$ in (5.15) have the following expressions:

$$\mathbf{Z}_{i,j_m} = \mathbf{V}_i \mathbf{S}_{i,j_m} \mathbf{V}^T_{j_m} \quad (m = 1, 2, ..O(C_{sp})) \tag{5.16}$$

$$\mathbf{Z}^l_{ip,jp_m} = \mathbf{V}_i \mathbf{T}_i^{L-1} ... \mathbf{T}_i^l \mathbf{S}^l_{ip,jp_m} \mathbf{V}^T_{jp_m} \quad (l = L-1, L-2, ..., l_0), \tag{5.17}$$

where $\mathbf{T}^l_i$ denotes the transfer matrix associated with cluster i or its parent cluster at level l.

We then compute Gram matrix, $\mathbf{Z}_{i,2}$, which is

$$\mathbf{Z}_{i,2} = \mathbf{Z}_i \mathbf{Z}_i^H. \tag{5.18}$$

Due to the nested property of admissible blocks as can be seen from (5.17), $\mathbf{Z}_{i,2}$ can be efficiently obtained through a top-down tree traversal procedure. To explain, substituting (5.16) and (5.17) into (5.15), and then (5.18), we obtain

$$\mathbf{Z}_{i,2} = \mathbf{V}_i \mathbf{S}^i_{sum} \mathbf{V}_i^H, \tag{5.19}$$

where

$$\mathbf{S}^i_{sum} = \begin{cases} \widetilde{\mathbf{S}}^i_{sum} & l(i) = l_0 \\ \widetilde{\mathbf{S}}^i_{sum} + \mathbf{T}_i \mathbf{S}^{ip}_{sum} \mathbf{T}_i{}^H & l_0 < l(i) \leq L \end{cases} \tag{5.20}$$

in which $l(i)$ denotes the tree level of cluster i, ip is the immediate parent cluster of i, $\mathbf{T}_i$ denotes the transfer matrix between i and ip, l_0 is the minimum level that has admissible blocks, and

$$\widetilde{\mathbf{S}}^i_{sum} = \sum_{j=1}^{O(C_{sp})} \mathbf{S}_{i,j} \mathbf{B}_j \mathbf{B}_j^H \mathbf{S}^H_{i,j}, \tag{5.21}$$

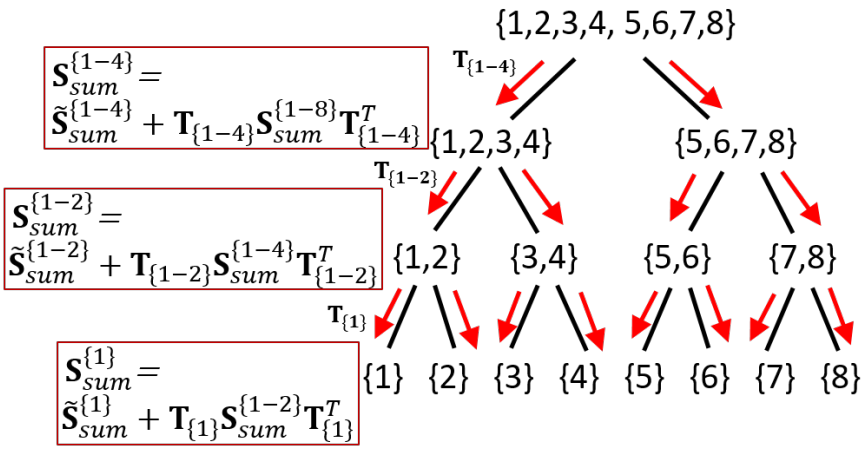


Fig. 5.3.: Illustration of top-down tree traversal for computing $\mathbf{S}_{sum}^{i}$.

with

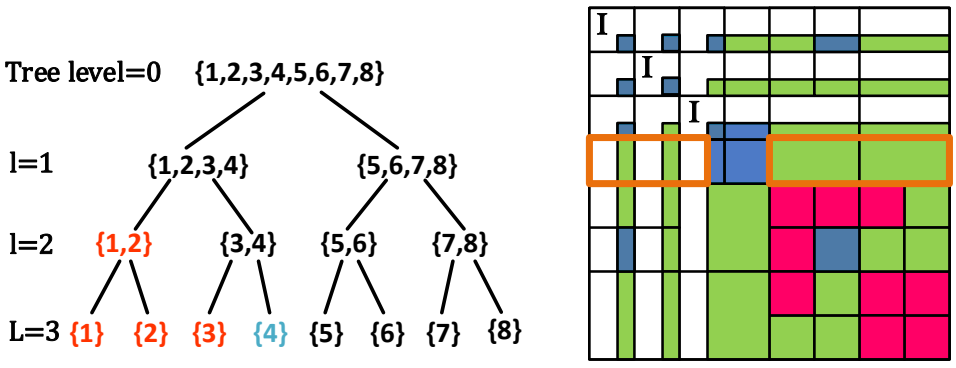
$$\mathbf{B}_j\mathbf{B}_j^H = \mathbf{V}_j^T\overline{\mathbf{V}}_j = \mathbf{I} \tag{5.22}$$

for unitary cluster basis $\mathbf{V}_j$. Obviously, $\widetilde{\mathbf{S}}_{sum}^{i}$ includes all of the admissible blocks at i's own level, and $\mathbf{S}_{i,j}$ are coupling matrices of these blocks. In contrast, the non-tilted $\mathbf{S}_{sum}^{i}$ includes both $\widetilde{\mathbf{S}}_{sum}^{i}$ and the aggregated contribution from i's parent admissible blocks, as can be seen from (5.20).

To compute (5.19) efficiently, we need to obtain $\mathbf{S}_{sum}^{i}$ for each cluster i, be it nonleaf or leaf. To do so, as can be seen from (5.20), we need to first compute $\widetilde{\mathbf{S}}_{sum}^{i}$ for each cluster i. The $\widetilde{\mathbf{S}}_{sum}^{i}$ can be obtained for all clusters i in linear time for constant-rank $\mathcal{H}^2$ matrices, as the computational cost of (5.21) is $O(C_{sp})k^3$ for each cluster, and there are in total $O(N)$ clusters. After obtaining $\widetilde{\mathbf{S}}_{sum}^{i}$, based on (5.20), the non-tilted $\mathbf{S}_{sum}^{i}$ for all non-leaf and leaf clusters can be obtained by performing a top-down traversal of the cluster tree from level l_0 down to the leaf level. This procedure is illustrated in Fig. 5.3. First, starting from level l_0 (which is the level of cluster $i = \{1-8\}$ in Fig. 5.3), for a cluster i at this level, $\mathbf{S}_{sum}^{i} = \widetilde{\mathbf{S}}_{sum}^{i}$, since there are no parent-level admissible blocks. In Fig. 5.3, $\mathbf{S}_{sum}^{\{1-8\}} = \widetilde{\mathbf{S}}_{sum}^{\{1-8\}}$. Such an $\mathbf{S}_{sum}^{i}$ is then split to its two children clusters, which is to perform $\mathbf{T}_i\mathbf{S}_{sum}^{ip}\mathbf{T}_i{}^H$ for its left, and right child cluster i respectively. As a result, the $\mathbf{S}_{sum}^{i}$ for the two children clusters

can be obtained. As can be seen from Fig. 5.3, $\mathbf{S}_{sum}^{\{1-4\}}$ is obtained by adding $\widetilde{\mathbf{S}}_{sum}^{\{1-4\}}$ with $\mathbf{T}_{\{1-4\}}\mathbf{S}_{sum}^{\{1-8\}}\mathbf{T}_{\{1-4\}}{}^{H}$. Such a procedure continues until we reach the leaf level. Since each split and addition operation of $\left(+\mathbf{T}_i\mathbf{S}_{sum}^{ip}\mathbf{T}_i{}^{H}\right)$ costs $O(k^3)$, and there are $O(N)$ clusters in the tree, the whole process of obtaining $\mathbf{S}_{sum}^{i}$ after $\widetilde{\mathbf{S}}_{sum}^{i}$ is computed also costs $O(N)$ operations.

After computing $\mathbf{S}_{sum}^{i}$, and thereby the Gram matrix shown in (5.19) for each leaf cluster, we add the fill-in's contributions as shown in (5.4). We then perform an SVD, from which the new cluster basis of each leaf cluster is obtained. Since the matrix on which SVD is performed is of $leafsize$, and there are $O(N)$ such matrices, the total cost of this step is also linear.



(a) (b)

Fig. 5.4.: (a) $\mathcal{H}^2$-tree with eliminated and uneliminated nodes. (b) $\mathbf{Z}_i$ matrix (orange box) for $i = 4$.

However, during the factorization procedure, when eliminating a cluster i, among all of its admissible blocks shown in (5.15), some have already been changed due to the computation performed on previous clusters. To see this point clearly, take cluster $i = 4$ shown in Fig. 5.4 (a) as an example. When this cluster is being eliminated, clusters $i = 1, 2, 3$, colored in red, are already eliminated, while clusters $i = 5, 6, 7, 8$, colored in black, are not eliminated yet. The admissible blocks formed between cluster $i = 4$ and eliminated clusters, which are $\mathbf{Z}_{4,1}$ and $\mathbf{Z}_{4,2}$ shown in orange box, are different from those in the original matrix, because cluster bases of $i = 1, 2$

have been changed, and the $\mathbf{Z}_{4,1}$ and $\mathbf{Z}_{4,2}$ blocks have been multiplied from the right by $\overline{\mathbf{Q}}_1$, and $\overline{\mathbf{Q}}_2$ respectively.

If we number the clusters based on their elimination order in sequence, then, for $j_m > i$ and $jp_m > i_p$, the admissible blocks $\mathbf{Z}_{i,j_m}$ and $\mathbf{Z}^l_{ip,jp_m}$ are formed with uneliminated clusters. They have the same form as those shown in (5.16) and (5.17). However, the admissible blocks $\mathbf{Z}_{i,j_m}$ and $\mathbf{Z}^l_{ip,jp_m}$ for $j_m < i$ and $jp_m < i_p$, i.e., those formed with eliminated clusters are not the same as those in the original $\mathcal{H}^2$-matrix $\mathbf{Z}$ anymore. Instead, they have the following form due to $\mathbf{Q}$ matrix multiplication performed in *Step 1*:

$$\widetilde{\mathbf{Z}}_{i,j_m} = \mathbf{V}_i \mathbf{S}_{i,j_m} \mathbf{V}_{j_m}^T \overline{\overline{\mathbf{V}}}_{j_m} \quad (j_m < i) \tag{5.23}$$

$$\widetilde{\mathbf{Z}}^l_{ip,jp_m} = \mathbf{V}_i \mathbf{T}_i^{L-1} ... \mathbf{T}_i^l \mathbf{S}^l_{ip,jp_m} \mathbf{V}_{jp_m}^T \overline{\overline{\mathbf{V}}}_{jp_m} \quad (jp_m < ip). \tag{5.24}$$

To highlight the fact that the aforementioned blocks are different from those in the original matrix, in (5.23) and (5.24), we use $\widetilde{\mathbf{Z}}$ instead of $\mathbf{Z}$ to represent these blocks. Similarly, we update (5.15) as follows to reflect the fact that some of the underlying blocks have been changed.

$$\widetilde{\mathbf{Z}}_i = [\widetilde{\mathbf{Z}}_{i,j_1}\ \widetilde{\mathbf{Z}}_{i,j_2}\ ..., \mathbf{Z}_{i,j_m}, ..., \widetilde{\mathbf{Z}}^{L-1}_{ip,jp_1}\ \widetilde{\mathbf{Z}}^{L-1}_{ip,jp_2}\ ..., \mathbf{Z}^{L-1}_{ip,jm}, ...,\ \widetilde{\mathbf{Z}}^{l_0}_{ip,jp_1}\ \mathbf{Z}^{l_0}_{ip,jp_2}\ ...]. \tag{5.25}$$

We then use (5.25) to compute the Gram matrix of $\mathbf{Z}_i$, which becomes

$$\mathbf{Z}_{i,2} = \widetilde{\mathbf{Z}}_i \widetilde{\mathbf{Z}}_i^H = \mathbf{V}_i \mathbf{S}^{i,new}_{sum} \mathbf{V}_i^H, \tag{5.26}$$

where

$$\mathbf{S}^{i,new}_{sum} = \begin{cases} \widetilde{\mathbf{S}}^{i,new}_{sum} & l(i) = l_0 \\ \widetilde{\mathbf{S}}^{i,new}_{sum} + \mathbf{T}_i \mathbf{S}^{ip,new}_{sum} {\mathbf{T}_i}^H & l_0 < l(i) \leq L \end{cases} \tag{5.27}$$

Using nested property, the above can still be computed via a top-down tree traversal procedure similar to Fig. 5.3. However, the $\widetilde{\mathbf{S}}^{i,new}_{sum}$ now is different from (5.21). We can divide the $\widetilde{\mathbf{S}}^{i,new}_{sum}$ into two parts as

$$\widetilde{\mathbf{S}}^{i,new}_{sum} = \widetilde{\mathbf{S}}^{i}_{sum,1} + \widetilde{\mathbf{S}}^{i}_{sum,2}. \tag{5.28}$$

The first part, denoted by $\widetilde{\mathbf{S}}^i_{sum,1}$, is from the admissible blocks formed between cluster i and eliminated clusters at i's tree level, as shown in (5.23) and (5.24). The second part, denoted by $\widetilde{\mathbf{S}}^i_{sum,2}$, is from the admissible blocks formed between cluster i and uneliminated clusters, whose expressions are given in (5.16) and (5.17). For the first part, since the admissible block has a form of (5.23), we compute it as

$$\widetilde{\mathbf{S}}^i_{sum,1} = \sum_{j(j<i)} \mathbf{S}_{i,j}\mathbf{B}^{new}_j \left(\mathbf{B}^{new}_j\right)^H \mathbf{S}^H_{i,j}, \tag{5.29}$$

where $\mathbf{B}^{new}_j$ is the projection of the new cluster basis onto the original cluster basis as shown below,

$$\mathbf{B}^{new}_j = \mathbf{V}^T_j \overline{\widetilde{\mathbf{V}}}_j. \tag{5.30}$$

For a non-leaf cluster j, using the nested property of cluster basis $\mathbf{V}_j$, $\mathbf{B}^{new}_j$ can be computed from $\mathbf{B}$ of its two children clusters, $\mathbf{B}^{new}_{j1}$ and $\mathbf{B}^{new}_{j2}$, which can be written as $\mathbf{B}^{new}_j = \left[\mathbf{T}^T_1\mathbf{B}^{new}_{j1}\ \mathbf{T}^T_2\mathbf{B}^{new}_{j2}\right]$. Here, $\mathbf{T}_1$ and $\mathbf{T}_2$ are the left and right transfer matrices of a non-leaf cluster j. The $\mathbf{B}^{new}_j$ is nothing but the rightmost part of (5.23) and (5.24). Hence, $(\mathbf{B}^{new}_j)^T$ can be viewed as the new cluster basis for j. For the second part, since the admissible block has its original form shown in (5.16), we have

$$\widetilde{\mathbf{S}}^i_{sum,2} = \sum_{j(j>i)} \mathbf{S}_{i,j}\mathbf{B}_j\mathbf{B}^H_j\mathbf{S}^H_{i,j} = \sum_{j(j>i)} \mathbf{S}_{i,j}\mathbf{S}^H_{i,j}. \tag{5.31}$$

Like $\widetilde{\mathbf{S}}^i_{sum}$, the $\widetilde{\mathbf{S}}^{i,new}_{sum}$ can be obtained for all clusters in linear time. However, **unlike $\widetilde{\mathbf{S}}^i_{sum}$, the $\widetilde{\mathbf{S}}^{i,new}_{sum}$ cannot be generated in advance before the factorization, since the updated admissible blocks formed by i are not known until we reach the step of eliminating cluster i.** Hence, $\widetilde{\mathbf{S}}^{i,new}_{sum}$ should be generated instantaneously during the factorization procedure, when it is time to eliminate cluster i. The same holds true for the top-down process of splitting the parent cluster's $\mathbf{S}^{i,new}_{sum}$ to its children clusters to obtain $\mathbf{S}^{i,new}_{sum}$ for each cluster i. In what follows, we use the cluster tree shown in Fig. 5.5 to illustrate the entire process of obtaining $\mathbf{S}^{i,new}_{sum}$ for each leaf cluster.

We number leaf clusters from left to right as $1, 2, 3, ...$, which is also the order of elimination. For the first cluster $i = 1$, we first compute its $\widetilde{\mathbf{S}}^{i,new}_{sum}$, denoted by a circle

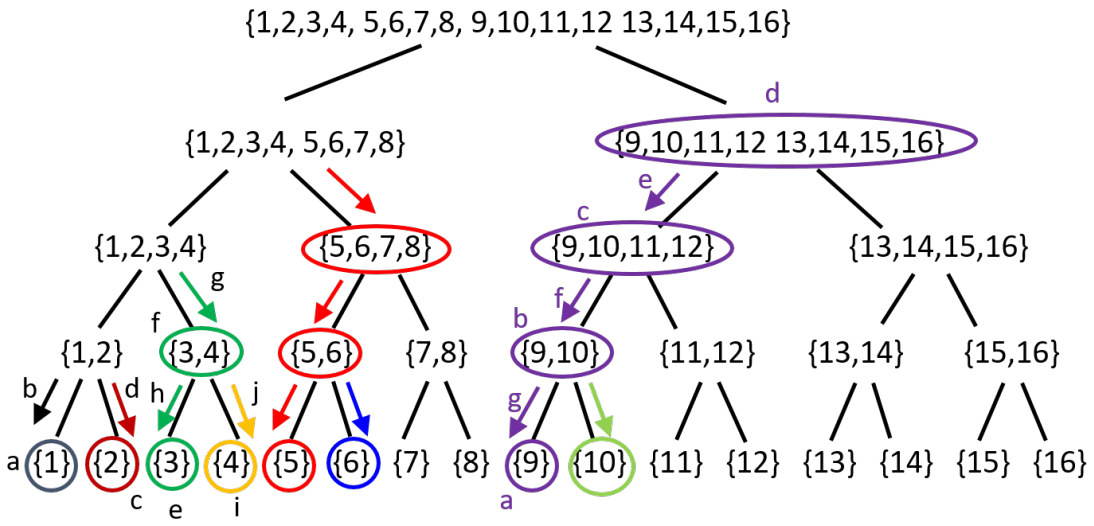


Fig. 5.5.: Illustration of the procedure for instantaneously computing $\mathbf{S}_{sum}^{i,new}$.

around cluster $i = 1$ in Fig. 5.5. Since no clusters have been eliminated, $\widetilde{\mathbf{S}}_{sum}^{i,new}$ is the same as that in the original matrix, thus $\widetilde{\mathbf{S}}_{sum}^{i}$. We then check whether $\mathbf{S}_{sum}^{ip,new}$ is known or not. Since the immediate parent cluster of $i = 1$ has not been updated either in its admissible blocks, $\mathbf{S}_{sum}^{ip,new} = \mathbf{S}_{sum}^{ip}$. As a result, (5.27) can be readily computed for cluster $i = 1$. The arrow from $i = 1$'s parent cluster to $i = 1$ represents the $(+\mathbf{T}_i \mathbf{S}_{sum}^{ip,new} \mathbf{T}_i{}^H)$ operation in (5.27), which is to split $\mathbf{S}_{sum}^{ip,new}$ to its child, and then add it with the child's $\widetilde{\mathbf{S}}_{sum}^{new}$.

For cluster $i = 2$, since when it is factorized, cluster $i = 1$ has been factorized, we compute its $\widetilde{\mathbf{S}}_{sum,1}^{i}$ using (5.29), where $\mathbf{B}_1^{new}$ is employed. Meanwhile, we compute its $\widetilde{\mathbf{S}}_{sum,2}^{i}$ using (5.31), which captures the admissible blocks formed between $i = 2$ and uneliminated clusters. The sum of the two makes $\widetilde{\mathbf{S}}_{sum}^{i,new}$, which can be different from that in the original matrix, i.e., $\widetilde{\mathbf{S}}_{sum}^{i}$. Notice that considering all eliminated and uneliminated clusters, there are at most $O(C_{sp})$ clusters that can form admissible blocks with a cluster. The step of computing $\widetilde{\mathbf{S}}_{sum}^{i,new}$ for $i = 2$ is denoted by a circle surrounding cluster $i = 2$ in Fig. 5.5. Since i's immediate parent cluster ip has a known $\mathbf{S}_{sum}^{ip,new}$, which is equal to $\mathbf{S}_{sum}^{ip}$, summing up the $\widetilde{\mathbf{S}}_{sum}^{i,new}$ and the second term

in (5.27) (denoted by the red arrow pointing from $i = 2$'s parent to $i = 2$), we obtain $\mathbf{S}_{sum}^{i,new}$ for cluster $i = 2$.

We then proceed to cluster $i = 3$. When this cluster is factorized, both clusters $i = 1$ and $i = 2$ have been factorized, we hence compute its $\widetilde{\mathbf{S}}_{sum,1}^{i}$ using (5.29), and $\widetilde{\mathbf{S}}_{sum,2}^{i}$ using (5.31), the sum of which makes $\widetilde{\mathbf{S}}_{sum}^{i,new}$. As for the parent cluster's contribution to $i = 3$, since now the parent cluster of clusters 1 and 2, cluster $i = \{1, 2\}$, has an updated $\mathbf{B}_{i}^{new}$, $\widetilde{\mathbf{S}}_{sum}^{\{3,4\},new}$ would have to be computed as it may be different from the original one. Thus, we compute $\widetilde{\mathbf{S}}_{sum}^{i,new}$ for $i = 3$'s parent cluster $\{3, 4\}$, the step of which is denoted by the circle around $\{3, 4\}$, and then $\mathbf{S}_{sum}^{\{3,4\},new}$ can be found from (5.27) by realizing that $\mathbf{S}_{sum}^{ip,new} = \mathbf{S}_{sum}^{\{1,2,3,4\}}$, which is cluster $\{1, 2, 3, 4\}$'s $\mathbf{S}_{sum}$ in the original matrix. After $\mathbf{S}_{sum}^{\{3,4\},new}$ is found, the $\mathbf{S}_{sum}^{i=3,new}$ can be readily computed by splitting $\mathbf{S}_{sum}^{\{3,4\},new}$'s contribution to it, denoted by the arrow labeled as h.

For cluster $i = 4$, we first compute its $\widetilde{\mathbf{S}}_{sum}^{4,new}$. We then check whether its parent cluster's $\mathbf{S}_{sum}^{new}$ is known or not, which turns out to be known since $\mathbf{S}_{sum}^{\{3,4\},new}$ has been found when cluster $i = 3$ is computed. Thus, a single splitting of $\mathbf{S}_{sum}^{\{3,4\},new}$ to $i = 4$, and adding it with $\widetilde{\mathbf{S}}_{sum}^{4,new}$ yields $\mathbf{S}_{sum}^{\{4\},new}$. This step is represented by the yellow arrow in Fig. 5.5.

The algorithm shown in **Algorithm 4** summarizes the overall procedure of instantaneously computing $\mathbf{S}_{sum}^{i,new}$ for all leaf clusters. Basically, for an arbitary leaf cluster i, we first compute its $\widetilde{\mathbf{S}}_{sum}^{i,new}$. We then check whether $\mathbf{S}_{sum}^{ip,new}$ is known or not. If known, then using (5.27), the $\mathbf{S}_{sum}^{i,new}$ is obtained for leaf cluster i. If not, that means ip's admissible blocks have been changed, thus we compute ip's $\widetilde{\mathbf{S}}_{sum}^{new}$. After that, we move one level up and check whether the $\mathbf{S}_{sum}^{new}$ is known for ip's parent cluster. We continue to do so until we reach the level where i's parent cluster has a known $\mathbf{S}_{sum}^{new}$. This cluster's $\mathbf{S}_{sum}^{new}$ is known either because it is the first cluster at that level, thus $\mathbf{S}_{sum}^{i,new}$ is equal to the original $\mathbf{S}_{sum}^{i}$, or because it is at the minimal level where admissible block exists, and thus $\mathbf{S}_{sum}^{i,new} = \widetilde{\mathbf{S}}_{sum}^{i,new}$, or because it has been obtained during previous cluster computation. For example, for cluster $i = 4$ shown in Fig. 5.5, its $\mathbf{S}_{sum}^{ip,new}$ is known from the computation done for cluster $i = 3$. As another example,

Algorithm 4 Proposed Algorithm for Computing $\mathbf{S}_{sum}^{i,new}$ for All Leaf Clusters

for *cluster* : i =1 **to** 2^L **do**
  Compute $\widetilde{\mathbf{S}}_{sum}^{i,new}$ (using (5.28)).
  $l = l(i)$: Let l to be the tree level of cluster i
  $j = i$
  while $\mathbf{S}_{sum}^{parent(j),new}$ is not known **do**
    $j = parent(j)$: Let next j be j's parent
    Compute $\widetilde{\mathbf{S}}_{sum}^{j,new}$.
  end while
  for $l = l(j)$ **to** $l(i) - 1$ **do**
    $k = parent(i, l)$: Let k be i's parent at level l
    Compute $\mathbf{S}_{sum}^{k,new} = \widetilde{\mathbf{S}}_{sum}^{k,new} + \mathbf{T}_k \mathbf{S}_{sum}^{parent(k),new} \mathbf{T}_k{}^H$.
  end for
  Compute $\mathbf{S}_{sum}^{i,new} = \widetilde{\mathbf{S}}_{sum}^{i,new} + \mathbf{T}_i \mathbf{S}_{sum}^{parent(i),new} \mathbf{T}_i{}^H$.
end for

for cluster $i = 9$, we have to reach its parent cluster at level $L - 4$, where $\mathbf{S}_{sum}^{i,new}$ is known. After that, we perform a top down tree-traversal to split $\mathbf{S}_{sum}^{i,new}$ to children clusters and compute the $\mathbf{S}_{sum}^{i,new}$ of children clusters level by level down until we reach leaf level. The sequence of computation is marked in purple for cluster $i = 9$ using letters from a to g.

In Fig. 5.5, a circle around a cluster denotes the computation of its $\widetilde{\mathbf{S}}_{sum}^{i,new}$, and an arrow denotes a *split and add* operation from its parent $\mathbf{S}_{sum}^{i,new}$. The sequence of computation is also marked in letters from a to i for the computation of $\mathbf{S}_{sum}^{i,new}$ for clusters from $i = 1$ to $i = 4$. Obviously it is different from that shown in Fig. 5.3, which is performed in advance before the matrix factorization. The procedure shown in Fig. 5.5 is performed concurrently with the factorization. Thus, the $\widetilde{\mathbf{S}}_{sum}^{i,new}$ is computed based on the updated admissible blocks at the time when cluster i is factorized. In Fig. 5.5, we also use different color to denote the computation associated with the generation of $\mathbf{S}_{sum}^{i,new}$ for a leaf cluster i. For example, the circles and arrows in red are the computations performed for cluster $i = 5$; and those in purple are the computations performed for cluster $i = 9$. Since each arrow in Fig. 5.5 is associated with a cost of $O(k^3)$, each circle is associated with a cost of $O(k^3)C_{sp}$, and there are in total $O(N)$ clusters in the tree, the overall cost is $O(N)$.

Before we move to next level, we update the $\widetilde{\mathbf{S}}_{sum}^{i,new}$ for each cluster which is now $\widetilde{\mathbf{S}}_{sum,1}^{i,new}$ only as all clusters at the current level have been eliminated. After that, we obtain $\mathbf{S}_{sum}^{i,new}$ for each cluster, which now can be performed based on the procedure shown in Fig. 5.3 since every block is known. This $\mathbf{S}_{sum}^{i,new}$ will be used as the initial $\mathbf{S}_{sum}^{i}$ to start the process of new cluster basis computation at the next non-leaf level.

5.3.2 Concurrent Change of Cluster Bases at a Non-Leaf Level

At a non-leaf level l, the computation of changing cluster bases is similar to that at the leaf level, because we can view the current non-leaf level l as the new leaf level, and the $\mathbf{T}_t^{new}$ shown in (5.6) and (5.10) as new leaf level cluster bases. Meanwhile,

the coupling matrices at this level and up, and the transfer matrices at upper levels all stay the same as those in the original matrix.

Similar to the leaf-level computation, for every cluster at level l (viewed as current leaf level), we assemble all of the i-related blocks into $\mathbf{Z}_i$ like (5.15), which includes the i-related admissible blocks at not only i's tree level but also i's parent levels. Similar to leaf level's situation, some of the admissible blocks are formed between cluster i and eliminated clusters, while the others are from uneliminated clusters. Notice that now the row dimension of these blocks is only $2k_{l+1}$ as can be seen from (5.6). The Gram matrix for non-leaf $\mathbf{Z}_i$ can then be computed as

$$\mathbf{Z}_{i,2} = \mathbf{Z}_i \mathbf{Z}_i^H = \mathbf{T}_i^{new} \mathbf{S}_{sum}^{i,new} \mathbf{T}_i^{new,H} \tag{5.32}$$

where the $\mathbf{S}_{sum}^{i,new}$ is obtained using a process similar to that at the leaf level. The procedure is illustrated in Fig. 5.6. Again, we instantaneously compute $\widetilde{\mathbf{S}}_{sum}^{i,new}$ and $\mathbf{S}_{sum}^{i,new}$. Now, we have

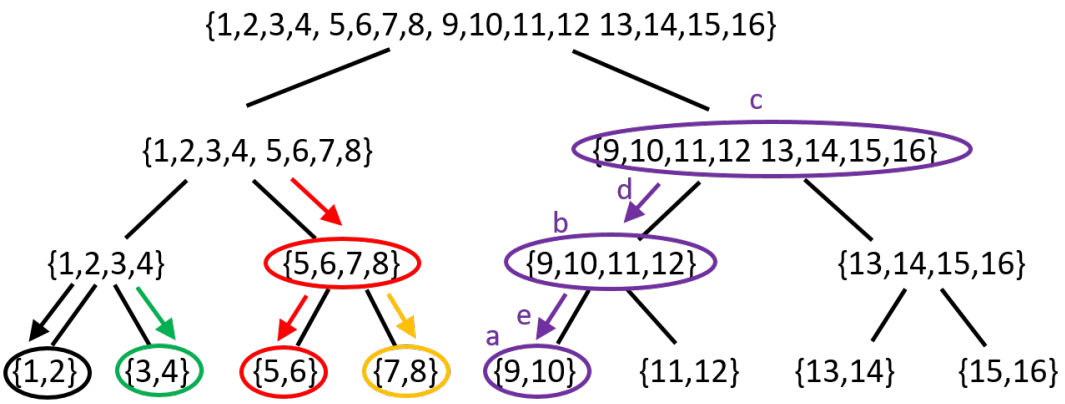


Fig. 5.6.: Illustration of instantaneously computing $\mathbf{S}_{sum}^{i,new}$ during a non-leaf level factorization.

$$\widetilde{\mathbf{S}}_{sum,1}^{i} = \sum_{j(j<i)} \mathbf{S}_{i,j} \mathbf{B}_j^{new} \left(\mathbf{B}_j^{new}\right)^H \mathbf{S}_{i,j}^H, \tag{5.33}$$

where $\mathbf{B}_j^{new}$ is

$$\mathbf{B}_j^{new} = \left(\mathbf{T}_j^{new}\right)^T \widetilde{\overline{\mathbf{T}}}_j^{new}, \tag{5.34}$$

and

$$\widetilde{\mathbf{S}}^{i}_{sum,2} = \sum_{j(j>i)} \mathbf{S}_{i,j}\mathbf{B}_j\mathbf{B}_j^H\mathbf{S}_{i,j}^H, \tag{5.35}$$

with

$$\mathbf{B}_j = \left(\mathbf{T}_j^{new}\right)^T \overline{\mathbf{T}}_j^{new}. \tag{5.36}$$

Here, different from (5.31), at a non-leaf level, $\mathbf{B}_j\mathbf{B}_j^H$ may not be identity any more.

We then add fill-in's contributions to update the cluster basis of cluster i by computing

$$\widetilde{\mathbf{Z}}_{i,2} = \mathbf{Z}_{i,2} + \sum\nolimits_m \mathbf{F}_{i,j_m}\mathbf{F}_{i,j_m}^H, \tag{5.37}$$

where $\mathbf{F}_{i,j_m}$ are fill-in blocks associated with the admissible blocks of cluster i. The $\widetilde{\mathbf{Z}}_{i,2}$ is a small matrix whose size is rank k. Its singular value decomposition (SVD) can be readily performed. The resultant singular vector matrix truncated based on prescribed accuracy ϵ_{acc} is the new transfer matrix $\widetilde{\mathbf{T}}_i^{new}$, thus

$$\widetilde{\mathbf{Z}}_{i,2,O(k_l)\times O(k_l)} \overset{\epsilon_{acc}}{\approx} \widetilde{\mathbf{T}}_i^{new}\Sigma'\widetilde{\mathbf{T}}_i^{new,H}. \tag{5.38}$$

The overall computational cost is again linearly proportional to the number of clusters in the $\mathcal{H}^2$-tree being handled at level l. As a result, the overall computational cost of changing cluster basis is $O(2^{L-1})$ at tree level $L-1$, $O(2^{L-2})$ at tree level $L-2$, etc. Hence, the total computational cost is $O(N)$.

5.4 Accuracy and Complexity Analysis

The accuracy of the proposed direct solution is controlled by ϵ_{acc}. This is different from [8, 9], since it is directly controlled. As can be seen from previous sections, there are no approximations involved in the proposed solution, except that when we change the cluster bases to account for the contribution from the fill-ins, we use (5.5) to truncate the singular vectors at the leaf level, and (5.38) to truncate the singular vectors at a non-leaf level. The accuracy of this step is controlled by parameter ϵ_{acc}, which can be set to any desired value.

From (5.14), it is evident that the whole computation involves $\mathbf{Q}_i^l$, $\mathbf{L}_i^l$, and $\mathbf{U}_i^l$ for cluster i at level l, the storage and computation of each of which are respectively $O(k_l^2)$, and $O(k_l^3)$ at a non-leaf level, and constant at the leaf level. Since there are $O(N)$ clusters in a tree, the total cost of generating these factors is linear for constant-rank $\mathcal{H}^2$ matrices. As for the permutation matrix $\mathbf{P}_l$, it is sparse involving only $O(2^l)$ integers to store at level l, and hence having an $O(N)$ cost in both memory and time. The change of the cluster basis costs $O(N)$ in time and memory also as analyzed in Section 5.3. As a result, the time and memory complexity of the proposed direct solution is linear for constant-rank $\mathcal{H}^2$ matrices. For variable-rank $\mathcal{H}^2$, the complexity of the proposed direct solution can also be analytically derived based on rank's behavior.

5.5 Numerical Results

The accuracy and complexity of the proposed direct solver are examined by performing both capacitance extraction and full-wave electromagnetic analysis of large-scale structures.

5.5.1 Two-Layer Cross Bus

The first example is a 2-layer cross bus structure in a uniform dielectric, as shown in [9]. In each layer, there are m conductors, and each conductor has a dimension of $1\times1\times(2m+1)$ m^3. We simulate a suite of such structures with $m = 16, 32, 64$, 128 and 512 respectively, whose number of unknowns N is respectively 17,152, 67,072, 265,216, 1,055,232 and 4,206,592. The parameters used in the proposed direct solver are $leafsize = 20$, $\eta = 1$, and $\epsilon_{\mathcal{H}^2} = 1e-4$ for $\mathcal{H}^2$-matrix construction, and $\epsilon_{acc} = 1e-8$ for direct solution. The resultant rank in the $\mathcal{H}^2$ matrix as well as the proposed direct solution is found to be a constant no greater than 5 at each tree level. FastCap [35] and the $O(N)$ inverse solver in [9] are also used to simulate the same example. The expansion order used in FastCap is 3. In Fig. 5.7(a), we compare the total CPU run

time of the three solvers as a function of N. Clear linear scaling can be observed from the proposed direct solver and that of [9], and also the proposed direct solver is shown to take less CPU time. We also compare the memory usage of the three solvers shown in Fig. 5.7(b). The fast $\mathcal{H}^2$-inverse solver keeps original $\mathcal{H}^2$-matrix memory, which need least memory. The proposed solver costs only slightly more memory than fast $\mathcal{H}^2$-inverse method due to cluster basis change during factorization, while both of them are much more memory efficient than FastCap. We have also used the capacitance matrix extracted from FastCap with expansion order 3 as the reference to compare the accuracy of the proposed direct solver with that of the fast $\mathcal{H}^2$-inverse. The error is assessed by $||\mathbf{C} - \mathbf{C}_{ref}||/||\mathbf{C}_{ref}||$, where $\mathbf{C}_{ref}$ is from FastCap, and $\mathbf{C}$ is from either this direct solver or fast $\mathcal{H}^2$-inverse. As can be seen from Fig. 5.7(c), the proposed direct solver exhibits better accuracy while using less CPU time.

5.5.2 On-Chip Interconnects

We have also performed a full-wave VIE simulation of on-chip interconnects. A suite of large-scale on-chip bus structures from a 4×4 to 64×64 are simulated at 30 GHz, with an x-polarized incident electric field. The conductivity of the metal is 5.8×10^7 S/m. The dimensions of each bus are $1\mu m \times 1\mu m \times 20\mu m$. The horizontal distance between the centers of two neighboring buses is $20\mu m$. And the vertical distance is $40\mu m$. Each bus is discretized into 322 unknowns. The total number of unknowns ranges from 5,152 to 1,318,912. For the $\mathcal{H}^2$ tree construction, we set *leafsize* to be 25 and $\eta = 1$, with $\epsilon_{\mathcal{H}^2} = 10^{-4}$. The accuracy parameter for controlling the direct solution is set to be $\epsilon_{acc} = 10^{-4}$. In Fig. 5.8(b), we compare the memory cost versus N of [38] with this new solver. In Fig. 5.8(a), we compare factorization time as a function of N. Clear linear complexities can be observed in CPU time and memory consumption for both solvers. However, in [38], the cluster bases are appended instead of completely changed during matrix factorization. The same accuracy parameter 10^{-4} is used to append the original cluster bases to take into account the fill-ins

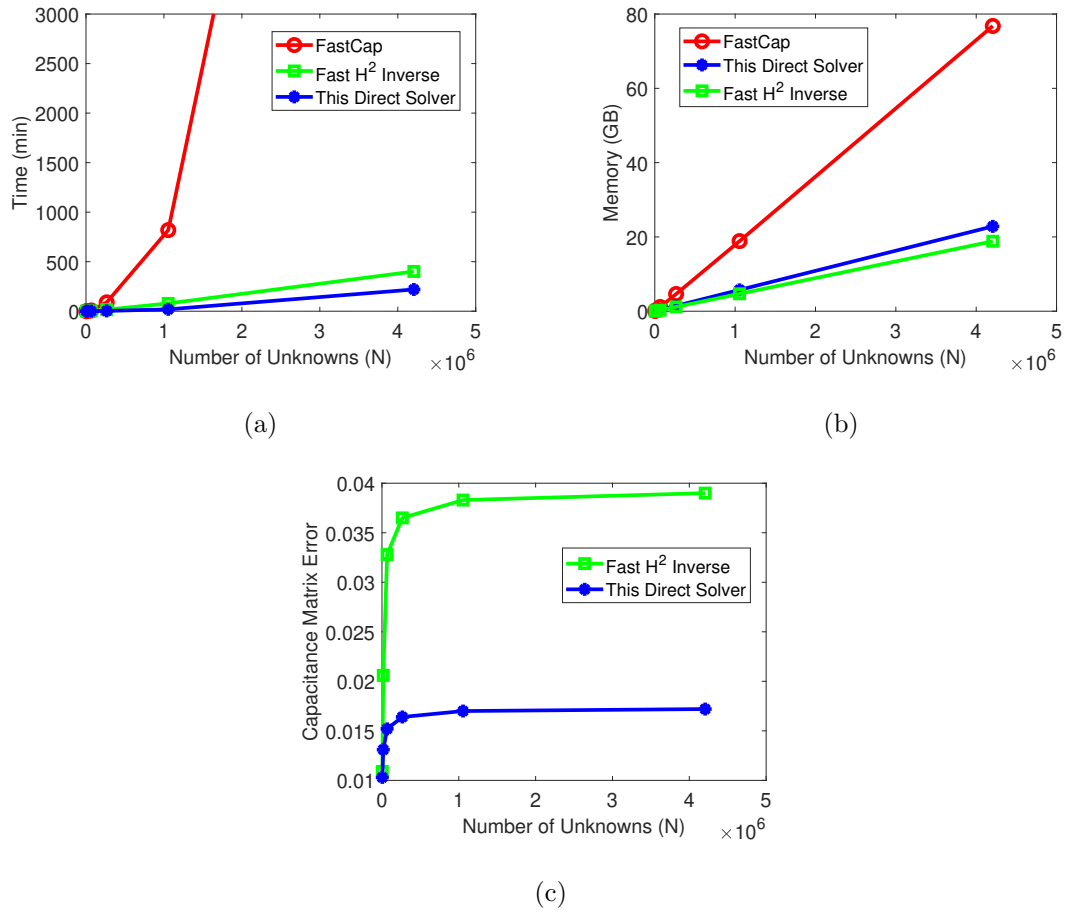
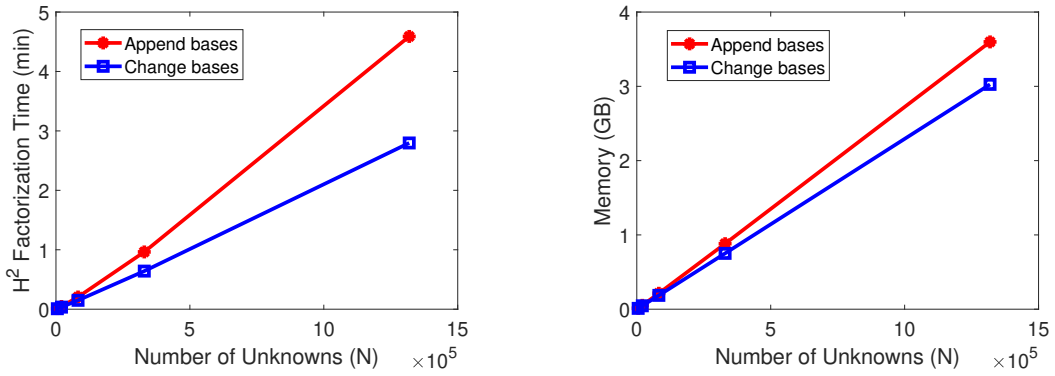


Fig. 5.7.: Capacitance extraction of a two-layer cross bus interconnect. (a) Time complexity. (b) Memory complexity. (c) Capacitance matrix error.

contribution during the factorization. To be specific, after deducting the fill-in blocks component in the original space of cluster bases from the fill-in block, we truncate the remainder using 10^{-4} tolerance to determine additional cluster bases to append. As can be seen from Fig. 5.8(b) and Fig. 5.8(a), this new direct solver with a concurrent change of the cluster basis is computationally more efficient. The accuracy of the proposed direct solver is assessed by relative residual and listed in Table 5.1. Excellent accuracy can be observed in the entire unknown range.

Table 5.1.: Direct solution error measured by relative residual ϵ_{rel} for the full-wave VIE interconnect simulation example.

N	5152	20,608	82,432	329,728	1,318,912
This solver	1.008e-5	1.005e-5	1.004e-5	1.003e-5	1.002e-5



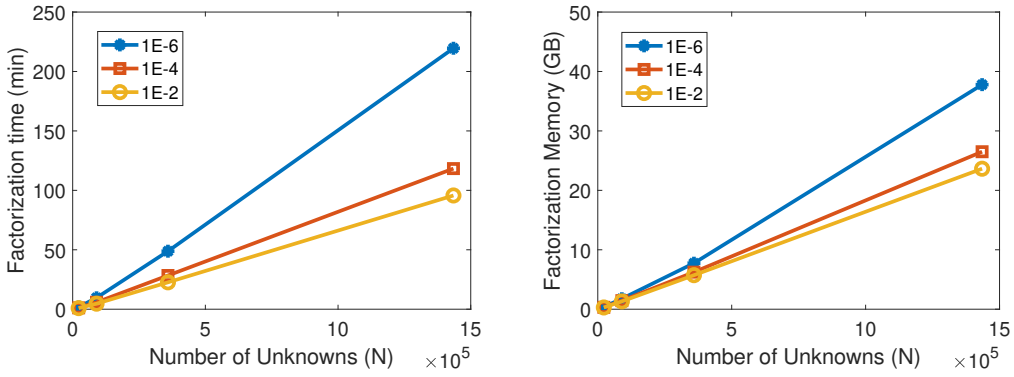
(a) (b)

Fig. 5.8.: Solver performance of lossy on-chip buses. (a) Factorization time v.s. N. (b) Memory v.s. N.

5.5.3 Large-scale Dielectric Slab

We then simulate a dielectric slab with $\epsilon_r = 2.54$ at 300 MHz, which is also simulated in [15]. The thickness of the slab is fixed to be $0.1\lambda_0$. The width and length are simultaneously increased from $4\lambda_0$, $8\lambda_0$, $16\lambda_0$, to $32\lambda_0$. With a mesh size of $0.1\lambda_0$, the resultant N ranges from 22,560 to 1,434,880 for this suite of slab structures. The *leafsize* is chosen to be 25 and $\eta = 1$. The ϵ_{acc} is set to be 10^{-2}, 10^{-4}, and 10^{-6} respectively to examine the solution accuracy, computational complexity, and error controllability of the proposed direct solution.

In Fig. 5.9(a), we plot the factorization time with respect to N, for all three different choices of ϵ_{acc}. It is clear that the smaller the ϵ_{acc}, the larger the factorization time. However, the complexity remains the same as linear regardless of the choice of



(a) (b)

Fig. 5.9.: Solver performance of a suite of dielectric slab for different choices of ϵ_{acc}. (a) Factorization time v.s. N. (b) Memory v.s. N.

ϵ_{acc}. The memory cost are plotted in Fig. 5.9(b) respectively. Obviously, both scale linearly with the number of unknowns. The error of the proposed direct solution is measured by computing the relative residual $\epsilon_{rel} = ||\mathbf{Z}_{\mathcal{H}^2}x - b||/||b||$, where $\mathbf{Z}_{\mathcal{H}^2}$ is the input $\mathcal{H}^2$ matrix in the equation to be solved. The relative residual ϵ_{rel} of the proposed direct solution is listed in Table 5.2 as a function of ϵ_{acc}. Excellent accuracy can be observed in the entire unknown range. Furthermore, the accuracy can be controlled by ϵ_{acc}, and overall smaller ϵ_{acc} results in better accuracy.

Table 5.2.: Direct solution error measured by relative residual, ϵ_{rel}, for the dielectric slab example.

ϵ_{acc}	10^{-2}	10^{-4}	10^{-6}
ϵ_{rel} $(N = 22,560)$	0.0290	0.0014	0.0002
ϵ_{rel} $(N = 89,920)$	0.0311	0.0021	0.0010
ϵ_{rel} $(N = 359,040)$	0.0368	0.0056	0.0033
ϵ_{rel} $(N = 1,434,880)$	0.0471	0.0113	0.0020

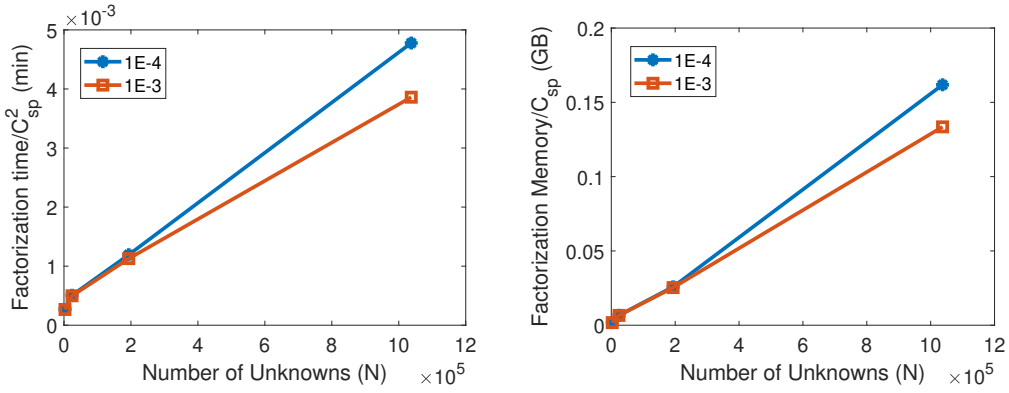
Since this example is also simulated in our previous work [15, 16], we have compared the two direct solvers in CPU run time and accuracy. The $\epsilon_{acc} = 10^{-4}$ is used. For a fair comparison, we set up this solver to solve the same $\mathcal{H}^2$ matrix solved in [16], and also use the same computer. As can be seen from Table 5.3, the proposed new direct solution takes much less time than that of [15, 16], which is direct $\mathcal{H}^2$-matrix inversion. The speedup is about one order of magnitude in large examples. In addition, because of a direct error control, the error of the proposed solution is also much less than that of [15, 16].

Table 5.3.: Performance comparison between this solver with $\epsilon_{acc} = 10^{-4}$ and [15, 16] for the dielectric slab example.

N	89,920	359,040
Factorization (s) [This]	358	1676
Solution (s) [This]	1.59	6.76
Inversion (s) [15, 16]	2.75e+03	1.65e+04
Relative Residual [This]	0.21%	0.56%
Inverse Error [15, 16]	3.9%	7.49%

5.5.4 Large-scale Array of Dielectric Cubes

We also simulate a large-scale array of dielectric cubes at 300 MHz. The relative permittivity of the cube is $\epsilon_r = 4.0$. Each cube is of size $0.3\lambda_0 \times 0.3\lambda_0 \times 0.3\lambda_0$. The distance between adjacent cubes is kept to be $0.3\lambda_0$. The number of the cubes is increased along the $x-$, $y-$, and $z-$ directions simultaneously from 2 to 14, thus producing a 3-D cube array from $2 \times 2 \times 2$ to $14 \times 14 \times 14$ elements. The number of unknowns N is respectively 3,024, 24,192, 193,536, and 1,037,232 for these arrays. During the construction of $\mathcal{H}^2$ matrix, we set $leafsize = 25$, $\eta = 1$ and $\epsilon_{\mathcal{H}^2} = 10^{-4}$. The ϵ_{acc} is chosen as 10^{-3}, 10^{-4}, and 10^{-5} respectively. The time and memory



(a) (b)

Fig. 5.10.: Solver performance of a suite of cube array from $2 \times 2 \times 2$ to $14 \times 14 \times 14$. (a) Time scaling v.s. N. (b) Memory scaling v.s. N.

performance is shown in Fig. 5.10(a) and Fig. 5.10(b) and the errors are shown in Table 5.4. In Fig. 5.10(a) and Fig. 5.10(b), we plot the direct factorization time

Table 5.4.: Direct solution error measured by relative residual for the cube array example.

N	3024	24192	193536	1037232
1E-3	0.0110	0.0258	0.0378	0.1641
1E-4	0.0012	0.0033	0.0048	0.0282

normalized by C_{sp}^2, and the storage cost normalized with C_{sp} with respect to N. As can be seen, their scaling rate with N agrees very well with our theoretical complexity analysis regardless of the choice of ϵ_{acc}.

5.6 Conclusion

In this chapter, we develop a new direct solution for general $\mathcal{H}^2$ matrices. This new direct solution not only features a directly controlled accuracy, but also a complete

change of cluster basis that is done concurrently with the factorization to efficiently represent the update to the original $\mathcal{H}^2$ matrix. The complexity of the overall direct solution, however, is still kept linear for constant-rank $\mathcal{H}^2$ matrices by developing fast algorithms. Millions of unknowns are directly solved on a single core CPU in fast CPU run time. Comparisons with state-of-the-art $\mathcal{H}^2$-based direct solvers have demonstrated the accuracy of this new direct solution as well as significantly improved computational efficiency. Being a generic direct solution to an $\mathcal{H}^2$ matrix, this work can also be applied to solve other integral equations and partial differential equations.

6. FAST $\mathcal{H}^2$-BASED ALGORITHMS FOR DIRECTLY SOLVING SPARSE MATRIX IN LINEAR COMPLEXITY

6.1 Introduction

Prevailing sparse matrix solvers for solving large-scale electromagnetic problems are iterative solvers. Although the original finite-element matrix is sparse, thus one matrix-vector multiplication only costs $O(N)$, the computational complexity of iterative finite-element solvers depends on the number of iterations, the number of right hand sides, and the cost of preconditioners. Since the L and U factors as well as the inverse of a sparse matrix are generally dense, the optimal complexity of a direct FEM solution in exact arithmetic has been found to be $O(N^2)$, where N is matrix size. In [42], a linear-complexity direct finite-element solver is developed for large-scale electromagnetic analysis with controlled accuracy. This solver takes advantage of the sparse linear algebra, and meanwhile accelerates all the dense matrix computations incurred during the direct solution procedure by $\mathcal{H}$-matrix-based fast techniques. An $\mathcal{H}$ matrix, unlike $\mathcal{H}^2$ matrices [7], does not have a nested representation. The resultant computational complexity is, in general, higher than that of an $\mathcal{H}^2$-based nested algorithm. However, it is challenging to develop an $\mathcal{H}^2$-based direct solution for sparse matrices since to take advantage of the zeros in the original matrix, we cannot treat the entire sparse matrix as an $\mathcal{H}^2$ matrix and directly invert or factorize it. If we do so, many more fill-ins will be introduced in the L and U factors, which makes the resultant direct solver slower than prevailing direct sparse solvers such as multifrontal methods. However, leveraging the framework of the multifrontal solver and accelerating all the dense frontal matrix computations is challenging, since every node in the elimination tree has its own structure, and it is difficult to communicate between different $\mathcal{H}^2$ matrices while keeping the computa-

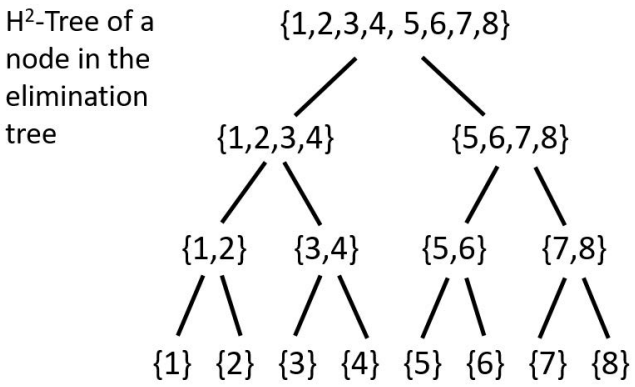


Fig. 6.1.: $\mathcal{H}^2$-tree of a node (separator or domain) in the elimination tree.

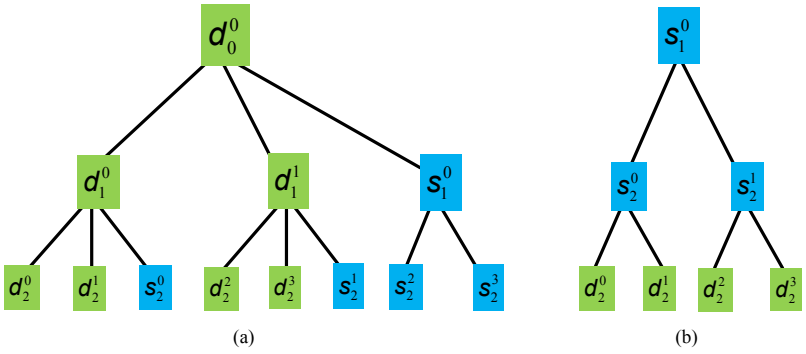


Fig. 6.2.: (a) Nested dissection. (b) Elimination tree.

tion nested across the elimination tree. In this work, we successfully overcome this challenge and develop a linear complexity $\mathcal{H}^2$-based direction solution to reduce the complexity of directly solving an FEM matrix to linear. Numerical experiments have demonstrated its performance.

6.2 Proposed Direct Solution

We partition the unknowns using nested dissection since such a scheme minimizes fill-ins in the factorization. This partition is shown in Fig. 6.2(a), which yields

an elimination tree shown in Fig. 6.2(b), where domains are at the leaf level, and separators are at the non-leaf levels. The size of the separators increases from the leaf level to the root level of the elimination tree. For large separators close to root level, we perform a binary bisection for fast computation. The proposed direct sparse matrix solution is a bottom-up traversal of the elimination tree. In the levels close to leaf level, where the separator size is small, we perform direct LU factorization on those separators (nodes in the elimination tree). For each node n in the elimination tree, we first find its boundary nodes, the set of which is denoted by b. This set includes all unknowns that have a crosstalk with the unknowns in set n. We then form the frontal matrix of node n and LU factorize this frontal matrix as

$$\begin{bmatrix} \mathbf{Y}_{nn} & \mathbf{Y}_{nb} \\ \mathbf{Y}_{bn} & \mathbf{Y}_{bb} \end{bmatrix} = \begin{bmatrix} \mathbf{L}_{nn} & \mathbf{0} \\ \mathbf{H}_{nb}\mathbf{U}_{nn}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \widetilde{\mathbf{Y}}_{bb} \end{bmatrix} \begin{bmatrix} \mathbf{U}_{nn} & \mathbf{L}_{nn}^{-1}\mathbf{Y}_{nb} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}. \tag{6.1}$$

It is known that the frontal matrix becomes denser and denser during the elimination procedure because of fill-ins generated from the factorization of previous nodes. In order to speed up the factorization, we first construct an $\mathcal{H}^2$ matrix representation of $\mathbf{Y}_{nn}$, $\mathbf{Y}_{nb}$, and $\mathbf{Y}_{bn}$. This starts from a level l_b, where the node size is small, and hence the computational cost is a constant. Then at levels above l_b, there is no need to construct an $\mathcal{H}^2$ matrix from full matrices, as the computations at previous levels have already been carried out in $\mathcal{H}^2$-format. The $\mathcal{H}^2$-tree of a node n in the elimination tree is illustrated in Fig. 6.1. An example of the $\mathcal{H}^2$-based frontal matrix is shown in Fig. 6.3(a), where all green blocks are admissible, and red ones are inadmissible. The conventional factorization of 6.1 costs cube complexity. We develop a fast $\mathcal{H}^2$-factorization algorithm to compute 6.1 in linear complexity, and also to make the resultant Schur complement, i.e., fill-ins in the $\mathbf{Y}_{bb}$, be represented in $\mathcal{H}^2$-format and conformal to the $\mathcal{H}^2$-structure of the b nodes. Notice that these b nodes are located in the levels above the level of n in the elimination tree, and each of which has its own tree like that shown in Fig. 6.1. In addition, often the fill-ins are only added to a portion of the blocks in the node contained in b, rather than the entire block of the node. Different from the dense matrix scenario, here, we cannot completely factorize

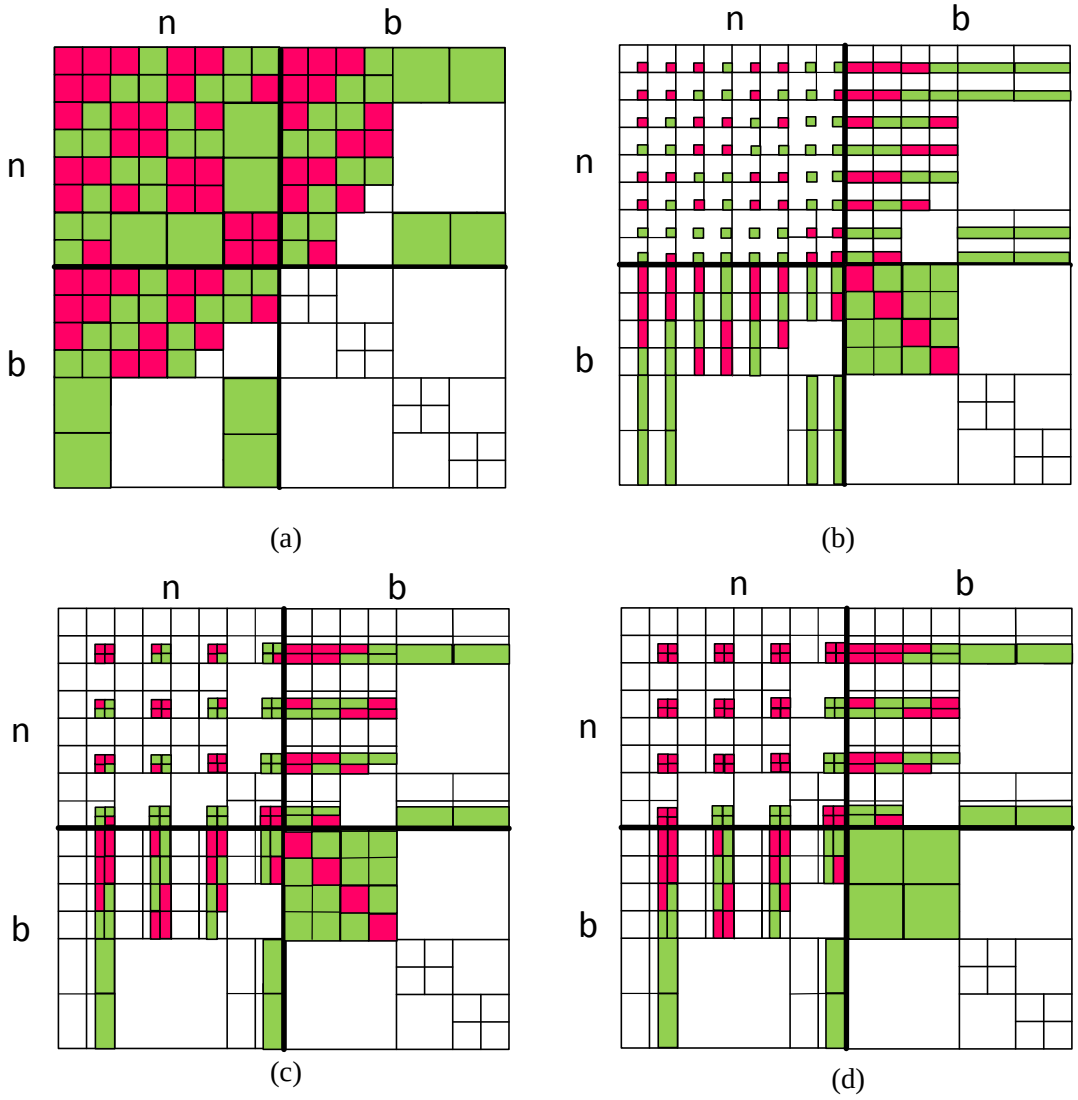


Fig. 6.3.: (a) $\mathcal{H}^2$-based frontal matrix of a node n. (b) Frontal matrix after leaf-level factorization of $\mathbf{Y}_{nn}$. (c) Frontal matrix merged after leaf-level factorization of $\mathbf{Y}_{nn}$. (d) Illustration of fill-ins in $\mathbf{Y}_{bb}$.

the entire frontal matrix since only the $\mathbf{Y}_{nn}$ part is ready for factorization. Hence, we go through the $\mathcal{H}^2$-tree of node n only, and factorize $\mathbf{Y}_{nn}$ level by level with the fast $\mathcal{H}^2$-algorithm developed in this work. Take the leaf level of node n as an example, to zero out the rows in the admissible blocks of leaf cluster t of node n, we also need to consider the admissible blocks in $\mathbf{Y}_{nb}$. The computations at the leaf level of node

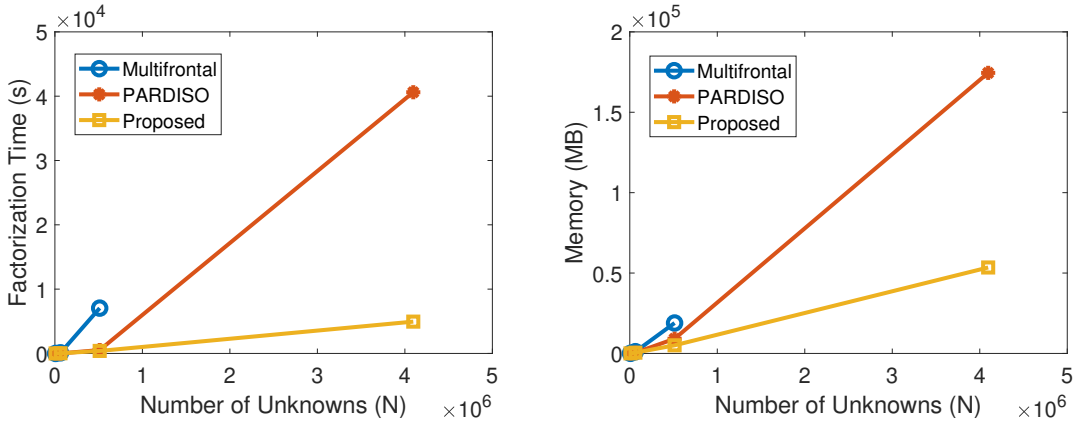
n hence include the following steps. Let $\mathbf{Y}_{cur} = \mathbf{Y}_{nn}$. For each leaf cluster i, do the following steps: 1) Change the original cluster basis of $\mathbf{V}_i$ to a new cluster basis $\widetilde{\mathbf{V}}_i$ to account for fill-ins in both $\mathbf{Y}_{nn}$ and $\mathbf{Y}_{nb}$. (2) Compute the complementary basis $\widetilde{\mathbf{V}}_i^{\perp}$ and build $\widetilde{\mathbf{Q}}_i = [(\mathbf{V}_i^{\perp})_{\#i\times(\#i-k_l)}\ (\mathbf{V}_i)_{\#i\times k_l}]$ (3) Compute $\mathbf{Q}_i\mathbf{Y}_{cur}\overline{\mathbf{Q}}_i$ to introduce zeros into admissible blocks, which only involves the computation of $O(Csp)$ inadmissible blocks. (4) Let $\mathbf{Y}_{cur} = \mathbf{Q}_i\mathbf{Y}_{cur}\overline{\mathbf{Q}}_i$. Perform partial LU factorization of $\mathbf{Y}_{cur}$ to eliminate the first unknowns in cluster i, where k_i is the rank of $\widetilde{\mathbf{V}}_i$. Finally, update the coupling matrix of each admissible block at the leaf level; Merge and permute the matrix to next level. At the end of the leaf-level computation, we obtain a matrix shown in Fig. 6.3(b), which is then merged to become a matrix shown in Fig. 6.3(c). Before we factorize this matrix, we have to prepare for cluster bases for clusters in b. Otherwise, the fill-ins, i.e.,$\mathbf{Y}_{cur,bn}\mathbf{Y}_{cur,nb}$ in Fig. 6.3(c) would have to be performed on the size of b, which is large. For every cluster in b, we consider all the fill-in blocks in b's admissible as well as inadmissible blocks (which are red blocks of low rank now) as shown in Fig. 6.3(c). We use them to generate the cluster basis for the clusters in b. After this operation, we are able to accurately collect the partially factorized $\mathbf{Y}_{cur,bn}$ and $\mathbf{Y}_{cur,nb}$ to rank size and then compute fill-ins. We also append the newly generated cluster basis to the existing ones in b so that the $\mathcal{H}^2$ matrix for the nodes to be eliminated after node n can be readily assembled. When we reach the root level of elimination tree, we do a root level $\mathcal{H}^2$-LU factorization. After we finish the factorization, we perform forward and backward substitution using the factors obtained to solve matrix equation. Since each step is computed either in rank size or $leafsize$, the overall complexity is smaller than that of [42]. This is because the non-nested $\mathcal{H}$-based computation in [42] involves the row and column dimensions of underlying matrix blocks, which are larger than rank.

6.3 Numerical Results

The proposed sparse direct solver is then tested on both Laplacian matrix and FEM matrix.

6.3.1 Laplacian Matrix

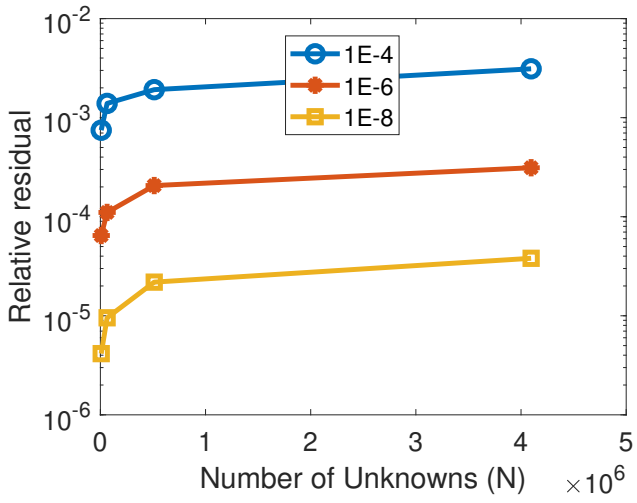
The first example is Laplacian matrix $\nabla^2\Phi = 0$ on a 3-D cube grid with side grid length from 20 to 160. The resulting unknown size is from $8,000$ to $4,096,000$. As shown in figure 6.4, the proposed solver outperforms state of the art direct sparse solver in both factorization time and memory. From the test results in our group [42], the Pardiso solver from Intel is the most efficient one among state-of-the-art sparse direct solvers. So the comparison with Pardiso shows that our proposed solver is much more efficient and also obtains linear complexity.



(a) (b)

Fig. 6.4.: Solver performance of a 3-D cube grid as comparison of state of the art direct sparse solver. (a) Factorization time v.s. N. (b) Memory v.s. N.

We also checked the accuracy of proposed solver at different $\epsilon_{fill-in} = 1E-4, 1E-6, 1E-8$. The relative residual of proposed direct sparse solver is plotted in Fig. 6.5.



(a)

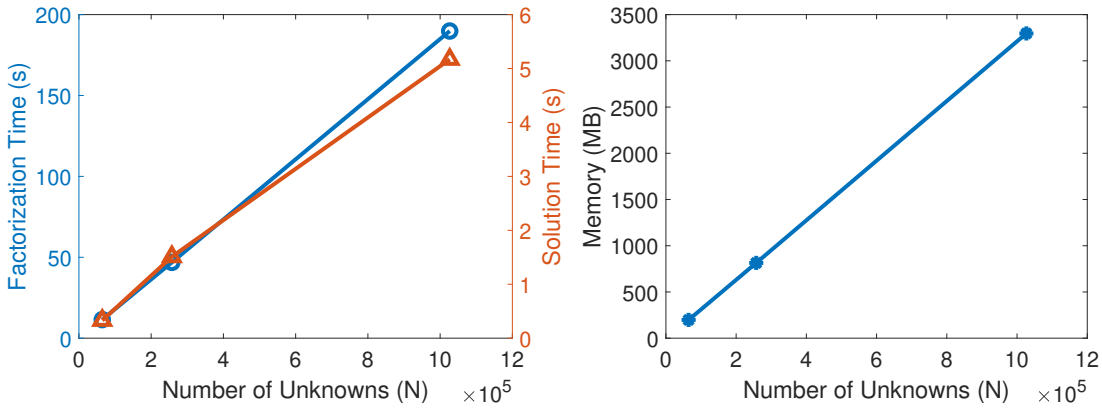
Fig. 6.5.: Solver relative residual of a 3-D cube grid.

6.3.2 FEM Matrix of Dielectric Slab

A dielectric object is discretized into a tetrahedron mesh with a mesh size of wavelength over 10 at frequency of $300MHz$. The dielectric constant is 2.54. We then increase the size of the object, resulting in N from $64,641$ to $1,026,561$. The $leafsize$ is 10, and the factorization accuracy is set as $1e-6$ in the proposed direct sparse solver. The FEM formulation is as shown in chapter 1.2.2. In Fig. 6.6, we draw the factorization, solution time and memory as a function of N. Clear linear scaling can be observed. We also list the relative residual error in Table 6.1.

Table 6.1.: Direct sparse solver error measured by relative residual.

N	64,641	257,281	1,026,561
Relative residual	3.494E-2	5.568E-2	8.537E-3



(a) (b)

Fig. 6.6.: Solver performance of FEM matrix on a dielectric slab. (a) Factorization and solution time v.s. N. (b) Memory v.s. N.

6.4 Conclusion

A new nested direct sparse matrix solution is developed for directly solving the sparse system matrix in linear complexity. In this solver, we accelerate all dense matrix computations underlying the state-of-the-art multifrontal based direct solver by developing fast $\mathcal{H}^2$-based algorithms. Each computation is performed on small matrices of rank size, which is also the rank of the separator instead of the rank of the original 3-D problem. Direct solutions of millions of unknowns in a few minutes on a single CPU core have demonstrated the superior performance of the proposed direct finite element solver.

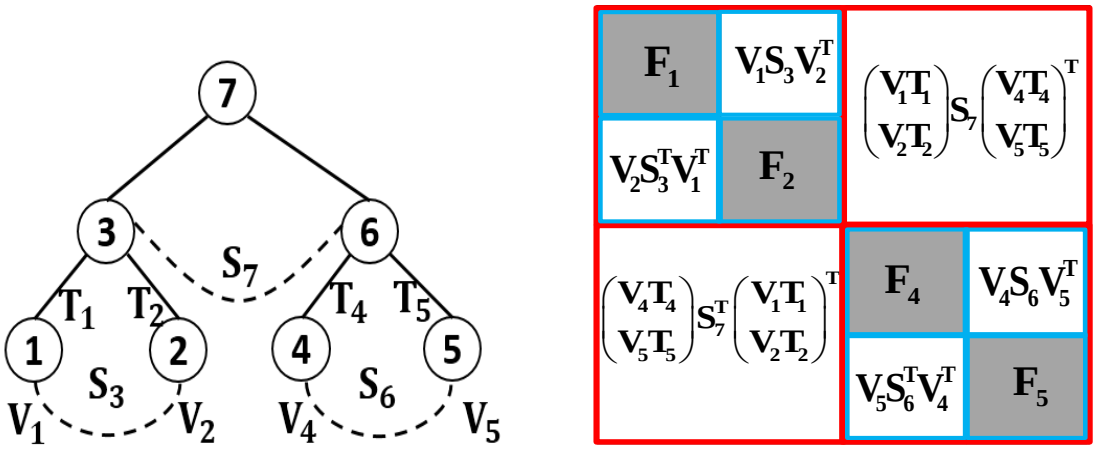
7. NEW HSS MATRIX INVERSION WITH DIRECTLY CONTROLLED ACCURACY

7.1 Introduction

State-of-the-art fast IE solvers have significantly reduced the computational complexity of dense matrix-vector multiplications from $O(N^2)$ to $O(NlogN)$ for solving electrically large problems [15]. Due to the fundamental difference between wave physics and static physics, it is challenging to further reduce the complexity. Recently, a VIE solver with $O(N)$ matrix-vector multiplication and $O(NlogN)$ matrix inversion has been successfully developed for electrically large analyses [16], and a subsequent fast $\mathcal{H}^2$-inversion algorithm. The inversion is observed to involve approximate arithmetic, although its accuracy can be controlled.

In this chapter, we develop a new HSS recursive inverse with directly controlled accuracy. In this new direct solver, we first represent the dense VIE system matrix by an HSS matrix with controlled accuracy. The HSS matrix is then inverted with directly controlled accuracy. An HSS matrix [28] has only one admissible block for each node in the cluster tree, which can be viewed as a special class of $\mathcal{H}^2$ matrix. This representation is feasible because as long as the sources are separated from observers, there exists a low-rank representation irrespective of electrical size. In mathematical literature, a fast HSS factorization algorithm has been developed for symmetric positive definite (SPD) matrices [28]. In this work, we develop new HSS inversion algorithms for solving general HSS matrices. The proposed algorithm also preserves the HSS matrix structure during the direct solution procedure. And by updating the cluster bases, we are able to control the accuracy of proposed direct solution and achieve machine precision via keeping the original rank. The resultant direct VIE solver is shown to have $O(NlogN)$ complexity in inversion, $O(N)$ complexity in so-

lution and memory for solving electrically large problems, which are also numerically demonstrated.



(a) (b)

Fig. 7.1.: Illustration of an HSS matrix (a)tree. (b) A two-level matrix.

7.2 Proposed Algorithm

The HSS system matrix $\mathbf{G}$ can be rewritten as

$$\mathbf{G} = \begin{bmatrix} \mathbf{G}_{11} & \mathbf{G}_{12} \\ \mathbf{G}_{21} & \mathbf{G}_{22} \end{bmatrix}. \tag{7.1}$$

In the above equation, we know $\mathbf{G}_{11}$ and $\mathbf{G}_{22}$ are full matrices or nonleaf matrices, while $\mathbf{G}_{12}$ and $\mathbf{G}_{21}$ are low-rank admissible matrices in the HSS matrix structure. For the matrix shown in equation 7.1, we can recursively obtain its inverse by using the Matrix Inversion Lemma [43]

$$\mathbf{G}^{-1} = \begin{bmatrix} \mathbf{G}_{11}^{-1} + \mathbf{G}_{11}^{-1} \times \mathbf{G}_{12} \times \mathbf{S}^{-1} \times \mathbf{G}_{21} \times \mathbf{G}_{11}^{-1} & -\mathbf{G}_{11}^{-1} \times \mathbf{G}_{12} \times \mathbf{S}^{-1} \\ -\mathbf{S}^{-1} \times \mathbf{G}_{21} \times \mathbf{G}_{11}^{-1} & \mathbf{S}^{-1} \end{bmatrix} \tag{7.2}$$

where $\mathbf{S}^{-1} = \mathbf{G}_{22} - \mathbf{G}_{21} \times \mathbf{G}_{11}^{-1} \times \mathbf{G}_{12}$. The above recursive inverse can be realized level by level by the follow steps,

1. HSS-inverse($\underline{\mathbf{G}}_{11}$, $\mathbf{X}_{11}$);

2. $\mathbf{G}_{21} \times \underline{\mathbf{G}}_{11} \rightarrow \mathbf{X}_{21}$, $\underline{\mathbf{G}}_{11} \times \mathbf{G}_{12} \rightarrow \mathbf{X}_{12}$;

3. $\mathbf{G}_{22} - \mathbf{X}_{21} \times \mathbf{G}_{12} \rightarrow \mathbf{G}_{22}$;

4. HSS-inverse($\underline{\mathbf{G}}_{22}$, $\mathbf{X}_{22}$);

5. $-\underline{\mathbf{G}}_{22} \times \mathbf{X}_{21} \rightarrow \mathbf{G}_{21}, -\mathbf{X}_{12} \times \underline{\mathbf{G}}_{22} \rightarrow \mathbf{G}_{12}$;

6. $\underline{\mathbf{G}}_{11} - \underline{\mathbf{G}}_{12} \times \mathbf{X}_{21} \rightarrow \mathbf{G}_{11}$.

in which the $\mathbf{G}$ that is different from the original $\mathbf{G}$ is underlined. The underlined $\mathbf{G}$ is overwritten by $\mathbf{G}^{-1}$ in the recursive computation.

As can be seen from the six steps in the above, we compute the inverse level by level. We start from the root level. We descend the block cluster tree of $\mathbf{G}$ all the way to the leaf level. At each level, we perform a number of matrix-matrix multiplications. For leaf level blocks, we perform a normal full matrix inverse.

Here, we show how to accurately invert HSS matrix by updating cluster bases and transfer matrices during the inverse procedure. We will use the example HSS matrix in Fig. 7.1(b) to illustrate the computations at leaf level and nonleaf level.

7.2.1 Leaf Level

At the leaf level, we first compute the dense matrix inverse of $\mathbf{G}_{11}^L$, which is $\mathbf{F}_1$ matrix in Fig. 7.1(b). Then we compute two matrix-matrix multiplications as $\mathbf{G}_{21}^L \times \underline{\mathbf{G}}_{11}^L \rightarrow \mathbf{X}_{21}^L$, $\underline{\mathbf{G}}_{11}^L \times \mathbf{G}_{12}^L \rightarrow \mathbf{X}_{12}^L$, where the $\underline{\mathbf{G}}_{11}^L$ is the inverse of the original $\mathbf{G}_{11}^L$. Because the $\mathbf{G}_{21}^L$ and $\mathbf{G}_{12}^L$ are low-rank admissible blocks, we have.

$$\begin{aligned} \mathbf{X}_{21}^L &= \mathbf{G}_{21}^L \times \underline{\mathbf{G}}_{11}^L = \mathbf{V}_2 \times \mathbf{S}_{21} \times (\mathbf{V}_1^T \times \underline{\mathbf{G}}_{11}^L) \\ \mathbf{X}_{12}^L &= \underline{\mathbf{G}}_{11}^L \times \mathbf{G}_{12}^L = (\underline{\mathbf{G}}_{11}^L \times \mathbf{V}_1) \times \mathbf{S}_3 \times \mathbf{V}_2^T. \end{aligned} \tag{7.3}$$

As can be seen from the above equation, the $\mathbf{X}_{21}^L$ and $\mathbf{X}_{12}^L$ are also low-rank admissible blocks. In order to calculate the two products accurately, we need to update the cluster bases of cluster 1 by

$$\tilde{\mathbf{V}}_1 = orth(\underline{\mathbf{G}}_{11}^L \times \mathbf{V}_1). \tag{7.4}$$

The original cluster bases $\mathbf{V}_1$ is then updated to $\tilde{\mathbf{V}}_1$ and the updated rank is bounded by the original rank. The products hence can be accurately represented using the updated cluster basis as

$$\begin{aligned} \mathbf{X}_{21}^L &= \mathbf{V}_2 \times (\mathbf{S}_3^T \times \mathbf{V}_1^T \times \underline{\mathbf{G}}_{11}^L \times \tilde{\mathbf{V}}_1) \times \tilde{\mathbf{V}}_1^T \\ \mathbf{X}_{12}^L &= \tilde{\mathbf{V}}_1 \times (\tilde{\mathbf{V}}_1^T \times \underline{\mathbf{G}}_{11}^L \times \mathbf{V}_1 \times \mathbf{S}_3) \times \mathbf{V}_2^T. \end{aligned} \tag{7.5}$$

Since the row and column cluster bases are known or generated, the computation of the above is to compute the matrices in the parentheses, which are the two coupling matrices: $\mathbf{S}_{21} \times \mathbf{V}_1^T \times \underline{\mathbf{G}}_{11}^L \times \tilde{\mathbf{V}}_1$ as the coupling matrix of $\mathbf{X}_{21}^L$; and $\tilde{\mathbf{V}}_1^T \times \underline{\mathbf{G}}_{11}^L \times \mathbf{V}_1 \times \mathbf{S}_{12}$ as the coupling matrix of $\mathbf{X}_{12}^L$. Since the two are transpose of each other, only one needs to be computed. This is done by computing $(\underline{\mathbf{G}}_{11}^L)_{collect} = \tilde{\mathbf{V}}_1^T \times \underline{\mathbf{G}}_{11}^L \times \mathbf{V}_1$, which can be viewed as collecting $\underline{\mathbf{G}}_{11}^L$ block using the original cluster bases as the column bases, and the updated ones as the new row bases. The computational cost is just $O(k^3)$. Then we multiply the collected block with the original coupling matrix, the cost of which is also $O(k^3)$.

Next, we proceed to compute updated $\mathbf{G}_{22}^L$ block as $\mathbf{G}_{22}^L - \mathbf{X}_{21}^L \times \mathbf{G}_{12}^L$. At leaf level, this can be readily done by converting the two admissible matrices $\mathbf{X}_{21}^L$ and $\mathbf{G}_{12}^L$ to full matrices and perform full matrix multiplication. The computational cost is $O(leafsize^3)$, a constant. We then calculate the inverse matrix of the updated $\mathbf{G}_{22}^L$ by normal full matrix inverse.

Now we come to computer the following two products

$$\begin{aligned} \mathbf{G}_{21}^L &= -\underline{\mathbf{G}}_{22}^L \times \mathbf{X}_{21}^L = (-\underline{\mathbf{G}}_{22}^L \times \mathbf{V}_2) \times \mathbf{S}_{21} \times \tilde{\mathbf{V}}_1^T \\ \mathbf{G}_{12}^L &= -\mathbf{X}_{12}^L \times \underline{\mathbf{G}}_{22}^L = -\tilde{\mathbf{V}}_1 \times \mathbf{S}_{12} \times (\mathbf{V}_2^T \times \underline{\mathbf{G}}_{22}^L), \end{aligned} \tag{7.6}$$

which are shown as Step 5 in the procedure given at the beginning of this section. The two make the $\mathbf{G}_{21}^L$ and $\mathbf{G}_{12}^L$ in the inverse matrix. As can be seen, they remain

to be admissible as their counterparts in the original $\mathbf{G}$. However, the cluster bases 2 should be updated as $\tilde{\mathbf{V}}_2 = orth(\underline{\mathbf{G}}_{22}^L \times \mathbf{V}_2)$. Using the new cluster bases, we can rewrite (7.6) as the following

$$\begin{aligned} \mathbf{G}_{21}^L &= \tilde{\mathbf{V}}_2 \times (-\tilde{\mathbf{V}}_2^T \times \underline{\mathbf{G}}_{22}^L \times \mathbf{V}_2 \times \mathbf{S}_{21}) \times \tilde{\mathbf{V}}_1^T \\ \mathbf{G}_{12}^L &= \tilde{\mathbf{V}}_1 \times (-\mathbf{S}_{12} \times \mathbf{V}_2^T \times \underline{\mathbf{G}}_{22}^L \times \tilde{\mathbf{V}}_2) \times \tilde{\mathbf{V}}_2^T, \end{aligned} \tag{7.7}$$

where we can first compute $(\underline{\mathbf{G}}_{22}^L)_{collect} = \tilde{\mathbf{V}}_2^T \times \underline{\mathbf{G}}_{22}^L \times \mathbf{V}_2$, then multiply it with coupling matrix $\mathbf{S}_{21}$, the computation cost of which is $O(k^3)$. The computation of (7.7) only involves the new coupling matrix computation, which is the matrix shown in the parentheses of (7.7).

In Step 6, we calculate $\mathbf{G}_{11}^L = \underline{\mathbf{G}}_{11}^L - \underline{\mathbf{G}}_{12}^L \times \mathbf{X}_{21}^L$. At leaf level, this can be readily done by converting admissible blocks $\underline{\mathbf{G}}_{12}^L$ and $\mathbf{X}_{21}^L$ to full matrices. Now we obtain the inverse of $\mathbf{G}_{11}$ at the $(L-1)$-th level, which is

$$\mathbf{G}_{11}^{L-1} = \begin{bmatrix} \tilde{\mathbf{F}}_1 & \tilde{\mathbf{V}}_1 \times \tilde{\mathbf{S}}_{12} \times \tilde{\mathbf{V}}_2^T \\ \tilde{\mathbf{V}}_2 \times \tilde{\mathbf{S}}_{21}^T \times \tilde{\mathbf{V}}_1^T & \tilde{\mathbf{F}}_2 \end{bmatrix} \tag{7.8}$$

and continue the computation to the non-leaf level.

7.2.2 Nonleaf Level

The computation at nonleaf levels l is similar to that at the leaf level. We need to collect the nonleaf blocks correctly to perform multiplications. For nonleaf level, we only need to perform six matrix-matrix multiplications, which can be reduced to the computation of four matrix-matrix multiplications using symmetry. The step-by-step multiplications are given as follows.

Since Step 1 has been performed at previous level, the computation starts from Step 2 of the procedure, where $\mathbf{G}_{21}^l \times \underline{\mathbf{G}}_{11}^l \rightarrow \mathbf{X}_{21}^l$, and $\underline{\mathbf{G}}_{11}^l \times \mathbf{G}_{12}^l \rightarrow \mathbf{X}_{12}^l$ are computed. Here, again the $\mathbf{G}_{21}^l$ and $\mathbf{G}_{12}^l$ are the low-rank admissible blocks in the original $\mathbf{G}$.

But the $\underline{\mathbf{G}}_{11}^{l}$ block is a nonleaf matrix block, as shown in (7.8). Rewriting it for a non-leaf level l, we have

$$\underline{\mathbf{G}}_{11}^{l} = \begin{bmatrix} \underline{\mathbf{F}}_1 & \tilde{\mathbf{V}}_1 \times \underline{\mathbf{S}}_3 \times \tilde{\mathbf{V}}_2^T \\ \tilde{\mathbf{V}}_2 \times \underline{\mathbf{S}}_3^T \times \tilde{\mathbf{V}}_1^T & \underline{\mathbf{F}}_2. \end{bmatrix}. \tag{7.9}$$

Since this block is used for computing $\underline{\mathbf{G}}_{11}^{l} \times \mathbf{G}_{12}^{l}$, i.e., it will be back multiplied by $\mathbf{V}_1$, we can first project (collect) it to a small matrix of $2k \times 2k$ size using the row cluster bases of $\tilde{\mathbf{V}}_1$ and the column cluster bases of $\mathbf{V}_1$. The collected $\underline{\mathbf{G}}_{11}^{l}$ has the following form

$$(\underline{\mathbf{G}}_{11}^{l})_{collect} = \begin{bmatrix} \tilde{\mathbf{V}}_1^T \times \underline{\mathbf{F}}_1 \times \mathbf{V}_1 & \underline{\mathbf{S}}_3 \times \tilde{\mathbf{V}}_2^T \times \mathbf{V}_2 \\ \underline{\mathbf{S}}_3^T \times \tilde{\mathbf{V}}_1^T \times \mathbf{V}_1 & \tilde{\mathbf{V}}_2^T \times \underline{\mathbf{F}}_2 \times \mathbf{V}_2 \end{bmatrix} = \begin{bmatrix} (\underline{\mathbf{G}}_{11}^{l+1})_{collect} & \underline{\mathbf{S}}_3 \times \mathbf{B}_2 \\ \underline{\mathbf{S}}_3^T \times \mathbf{B}_1 & (\underline{\mathbf{G}}_{22}^{l+1})_{collect} \end{bmatrix}. \tag{7.10}$$

Here we use $\mathbf{B}$ to denote cluster bases product, which are $\mathbf{B}_2 = \tilde{\mathbf{V}}_2^T \times \mathbf{V}_2$ and $\mathbf{B}_1 = \tilde{\mathbf{V}}_1^T \times \mathbf{V}_1$. Since the $(\underline{\mathbf{G}}_{11}^{L})_{collect}$ and $(\underline{\mathbf{G}}_{22}^{L})_{collect}$ blocks has already been collected at the leaf level, we only need to merge the four small $k \times k$ blocks to obtain $(\underline{\mathbf{G}}_{11}^{l})_{collect}$.

The $\mathbf{G}_{12}$ block in Fig. 7.1(b) is

$$\mathbf{G}_{12}^{l} = \begin{bmatrix} \mathbf{V}_1\mathbf{T}_1 \\ \mathbf{V}_2\mathbf{T}_2 \end{bmatrix} \times \mathbf{S}_7 \times \begin{bmatrix} \mathbf{V}_4\mathbf{T}_4 \\ \mathbf{V}_5\mathbf{T}_5 \end{bmatrix}^T. \tag{7.11}$$

Since the product $\underline{\mathbf{G}}_{11}^{l} \times \mathbf{G}_{12}^{l} \rightarrow \mathbf{X}_{12}^{l}$ will result in a different transfer matrix, for accuracy we need to update the transfer matrix as follows

$$\begin{bmatrix} \tilde{\mathbf{T}}_1 \\ \tilde{\mathbf{T}}_2 \end{bmatrix} = orth\left((\underline{\mathbf{G}}_{11}^{l})_{collect} \times \begin{bmatrix} \mathbf{T}_1 \\ \mathbf{T}_2 \end{bmatrix} \right). \tag{7.12}$$

With the transfer matrix $\mathbf{T}$ updated to $\tilde{\mathbf{T}}$, we compute

$$\mathbf{X}_{12}^{l} = \begin{bmatrix} \tilde{\mathbf{V}}_1\tilde{\mathbf{T}}_1 \\ \tilde{\mathbf{V}}_2\tilde{\mathbf{T}}_2 \end{bmatrix} \times \left(\begin{bmatrix} \tilde{\mathbf{T}}_1 \\ \tilde{\mathbf{T}}_2 \end{bmatrix}^T \times (\underline{\mathbf{G}}_{11}^{l})_{collect} \times \begin{bmatrix} \mathbf{T}_1 \\ \mathbf{T}_2 \end{bmatrix} \times \mathbf{S}_7 \right) \times \begin{bmatrix} \mathbf{V}_4\mathbf{T}_4 \\ \mathbf{V}_5\mathbf{T}_5 \end{bmatrix}^T, \tag{7.13}$$

which is also a nonleaf admissible block with nested cluster bases. The computation in the above equation can be performed efficiently by using collected $(\underline{\mathbf{G}}_{11}^{l})_{collect}$ as

$$(\underline{\mathbf{G}}_{11}^{l})_{collect} = \begin{bmatrix} \tilde{\mathbf{T}}_1 \\ \tilde{\mathbf{T}}_2 \end{bmatrix}^T \times (\underline{\mathbf{G}}_{11}^{l})_{collect} \times \begin{bmatrix} \mathbf{T}_1 \\ \mathbf{T}_2 \end{bmatrix}. \tag{7.14}$$

Then the computation of (7.13) only involves collected block matrix and coupling matrix, whose size is $k \times k$. The resulting computational cost is $O(k^3)$. Similarly, we use the symmetry property to get $\mathbf{X}_{21}^l$, whose coupling matrix is the transpose matrix of coupling matrix in $\mathbf{X}_{12}^l$.

Then we need to calculate $\mathbf{G}_{22}^l - \mathbf{X}_{21}^l \times \mathbf{G}_{12}^l$ and store it in original $\mathbf{G}_{22}^l$ block. Here $\mathbf{X}_{21}^l$ and $\mathbf{G}_{12}^l$ are admissible blocks, for which we only need to calculate the coupling matrices product. We call this product of $\mathbf{X}_{21}^l$ and $\mathbf{G}_{12}^l$ an auxiliary block $\mathbf{R}$. However, at a nonleaf level l, $\mathbf{G}_{22}^l$ is a non-leaf block, apparently, we have to split the $\mathbf{R}$ block representing $\mathbf{X}_{21}^l \times \mathbf{G}_{12}^l$ all the way down to the leaf blocks of $\mathbf{G}_{22}^l$ to obtain its Schur complement, and thereby the inverse. However, in this way, the complexity cannot be linear. We address this problem by developing a procedure of simultaneous splitting and inverse of the $\mathbf{G}_{22}^l$, where the splitting is performed based on the sequence of the inverse computation of $\mathbf{G}_{22}$ such that only one splitting is needed between a parent block and its direct children block. Since the inverse is recursive, in order to compute the inverse of the nonleaf block $\mathbf{G}_{22}^l$, we need to first compute the inverse of $\mathbf{G}_{22}^l$'s 11 child block. In general, 11 is also a nonleaf block, in order to compute its inverse, we again need to split the auxiliary block $\mathbf{R}$ in the 11 block, respectively, to their children. This process continues until 11 becomes full matrix, the inverse of which can be directly computed. In this procedure, we only need to do one split operation from the current level to the children that is one level down, and the auxiliary blocks can be added together due to different level's computation to process only once.

The computation of the inverse of $\mathbf{G}_{22}^l$ is similar to that of $\mathbf{G}_{11}^l$ after the contribution of the Schur-complement is added. We need to update the cluster bases of $\mathbf{V}_4$ and $\mathbf{V}_5$ as shown in Fig. 7.1(b) in a similar way as we update cluster bases $\mathbf{V}_1$ and $\mathbf{V}_2$. After we get the inverse of $\mathbf{G}_{22}^l$, we need to compute two products as $-\underline{\mathbf{G}}_{22}^l \times \mathbf{X}_{21}^l \rightarrow \mathbf{G}_{21}^l, -\mathbf{X}_{12}^l \times \underline{\mathbf{G}}_{22}^l \rightarrow \mathbf{G}_{12}^l$. Similarly, we only need to calculate

$-\underline{\mathbf{G}}_{22}^{l} \times \mathbf{X}_{21}^{l}$ due to symmetry property. The product is also an admissible block, the computation of which is done by first generating

$$(\underline{\mathbf{G}}_{22}^{l})_{collect} = \begin{bmatrix} \tilde{\mathbf{V}}_4^T \times \underline{\mathbf{F}}_4 \times \mathbf{V}_4 & \underline{\mathbf{S}}_6 \times \tilde{\mathbf{V}}_5^T \times \mathbf{V}_5 \\ \underline{\mathbf{S}}_6^T \times \tilde{\mathbf{V}}_4^T \times \mathbf{V}_4 & \tilde{\mathbf{V}}_5^T \times \underline{\mathbf{F}}_5 \times \mathbf{V}_5 \end{bmatrix} = \begin{bmatrix} (\underline{\mathbf{G}}_{11}^{l+1})_{collect} & \underline{\mathbf{S}}_6 \times \mathbf{B}_5 \\ \underline{\mathbf{S}}_6^T \times \mathbf{B}_4 & (\underline{\mathbf{G}}_{22}^{l+1})_{collect} \end{bmatrix}. \tag{7.15}$$

The transfer matrix is then updated as

$$\begin{bmatrix} \tilde{\mathbf{T}}_4 \\ \tilde{\mathbf{T}}_5 \end{bmatrix} = orth\left((\underline{\mathbf{G}}_{22}^{l})_{collect} \times \begin{bmatrix} \mathbf{T}_4 \\ \mathbf{T}_5 \end{bmatrix} \right). \tag{7.16}$$

Then the output $\mathbf{G}_{21}^{L}$ block is

$$\mathbf{G}_{21}^{l} = \begin{bmatrix} \tilde{\mathbf{V}}_4\tilde{\mathbf{T}}_4 \\ \tilde{\mathbf{V}}_5\tilde{\mathbf{T}}_5 \end{bmatrix} \times \left(\begin{bmatrix} \tilde{\mathbf{T}}_4 \\ \tilde{\mathbf{T}}_5 \end{bmatrix}^T \times (\underline{\mathbf{G}}_{22}^{l})_{collect} \times \begin{bmatrix} \mathbf{T}_4 \\ \mathbf{T}_5 \end{bmatrix} \times \underline{\mathbf{S}}_7^T \right) \times \begin{bmatrix} \tilde{\mathbf{V}}_1\tilde{\mathbf{T}}_1 \\ \tilde{\mathbf{V}}_2\tilde{\mathbf{T}}_2 \end{bmatrix}^T. \tag{7.17}$$

The computation in the above equation only involves collected matrix and coupling matrix multiplication, the cost of which is $O(k^3)$. And the collected matrix block is

$$(\underline{\mathbf{G}}_{22}^{l})_{collect} = \begin{bmatrix} \tilde{\mathbf{T}}_4 \\ \tilde{\mathbf{T}}_5 \end{bmatrix}^T \times (\underline{\mathbf{G}}_{22}^{l})_{collect} \times \begin{bmatrix} \mathbf{T}_4 \\ \mathbf{T}_5 \end{bmatrix}. \tag{7.18}$$

The final step at a nonleaf level is to calculate $\mathbf{G}_{11}^{l} = \underline{\mathbf{G}}_{11}^{l} - \underline{\mathbf{G}}_{12}^{l} \times \mathbf{X}_{21}^{l}$, which is similar to $\mathbf{G}_{22}^{l}$ computation. Since $\underline{\mathbf{G}}_{12}^{l}$ and $\mathbf{X}_{21}^{l}$ are low-rank admissible blocks, we first compute their auxiliary product $\mathbf{R}$. We store the auxiliary block $\mathbf{R}$ in $\mathbf{G}_{11}^{l}$, without splitting it all the way down to the leaf blocks. Because if we do so, the complexity cannot be linear. At the end of the inverse, we perform a one-way top-down tree traversal to split all the $\mathbf{R}$ blocks associated with $\mathbf{G}_{11}$ level by level until we reach the leaf level. Since $\mathbf{G}_{11}$'s inverse is involved in the subsequent computation, this R block is added upon the collected $\mathbf{G}_{11}$ at level l to do the block matrix multiplication with $\mathbf{G}_{12}$ at level l. We conclude the algorithm in the above discussion in the pseudocode shown in **Algorithm** 5.

The overall computation involves updating cluster bases and collecting dense blocks at leaf level and nonleaf blocks at nonleaf level to perform multiplications.

Algorithm 5 Proposed new HSS-inverse, ($\mathbf{G}$, $\mathbf{X}$) ($\mathbf{G}$ is input matrix, output $\mathbf{G}$ is its inverse, $\mathbf{X}$ is for temporarily storage)

if matrix $\mathbf{G}$ is a nonleaf matrix block **then**
  Split auxiliary block $\mathbf{R}$ if it's in $\mathbf{G}_{11}$
  HSS-inverse($\underline{\mathbf{G}}_{11}$, $\mathbf{X}_{11}$)
  Update cluster bases or transfer matrices using $\underline{\mathbf{G}}_{11}$
  $\mathbf{G}_{21} \times \underline{\mathbf{G}}_{11} \rightarrow \mathbf{X}_{21}$, $\underline{\mathbf{G}}_{11} \times \mathbf{G}_{12} \rightarrow \mathbf{X}_{12}$,
  $\mathbf{G}_{22} - \mathbf{X}_{21} \times \mathbf{G}_{12} \rightarrow \mathbf{G}_{22}$,
  Split auxiliary block $\mathbf{R}$ if it's in $\mathbf{G}_{22}$
  HSS-inverse($\underline{\mathbf{G}}_{22}$, $\mathbf{X}_{22}$)
  Update cluster bases or transfer matrices using $\underline{\mathbf{G}}_{22}$
  $-\underline{\mathbf{G}}_{22} \times \mathbf{X}_{21} \rightarrow \mathbf{G}_{21}, -\mathbf{X}_{12} \times \underline{\mathbf{G}}_{22} \rightarrow \mathbf{G}_{12}$,
  $\underline{\mathbf{G}}_{11} - \underline{\mathbf{G}}_{12} \times \mathbf{X}_{21} \rightarrow \mathbf{G}_{11}$,
else
  DirectInverse($\mathbf{G}$)(normal full matrix inverse)
end if
Backward transformation after inverse procedure

At each tree level l, there are 2^l clusters to be computed, each of which costs $O(k^3)$, where k is the rank at level l. The resulting complexity is clearly $O(N)$ for constant rank or rank that changes logarithmically with electrical size. For electrodynamic analysis, the rank grows linearly with electrical size for general 3-D problems. In a VIE, k is proportional to the cubic root of matrix size at level l, because this is the electrical size at level l. Hence for a VIE, the time complexity is

$$\textbf{Time Complexity} = \sum_{l=0}^{L} 2^l \left[\left(\frac{N}{2^l} \right)^{\frac{1}{3}} \right]^3 = O(NlogN), \tag{7.19}$$

and the memory complexity is

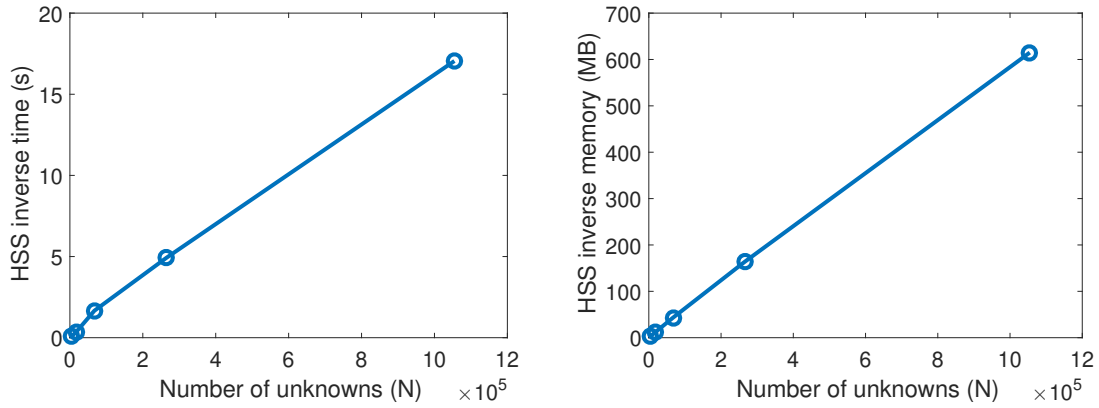
$$\textbf{Memory Complexity} = \sum_{l=0}^{L} 2^l \left[\left(\frac{N}{2^l} \right)^{\frac{1}{3}} \right]^2 = O(N). \tag{7.20}$$

7.3 Numerical Results

In order to demonstrate the accuracy and low computational complexity of the proposed new HSS recursive inverse for general HSS matrices, we use HSS matrices resulting from large-scale capacitance extraction and volume integral equations (VIE) for electromagnetic analysis for both circuit and scattering applications.

7.3.1 Two-layer Cross Bus

The first example is the capacitance extraction of a 2-layer cross bus structure. In each layer, there are m conductors, and each conductors has a dimension of $1 \times 1 \times (2m+1)$ m^3. We simulate a suite of such structures with 8, 16, 32, 64, and 128 buses respectively. The parameters used in the $\mathcal{H}^2$-matrix construction are $leafsize = 30$, admissibility condition [7] $\eta = 1$, and cluster bases truncation at 10^{-4}. As shown in Fig. 7.2, the proposed new HSS inverse has linear time and memory performance with different unknown size N. We also check the inverse residual error as shown in Table 7.1. The proposed new HSS inverse has machine precision with the rank being the same as the original HSS matrix.



(a) (b)

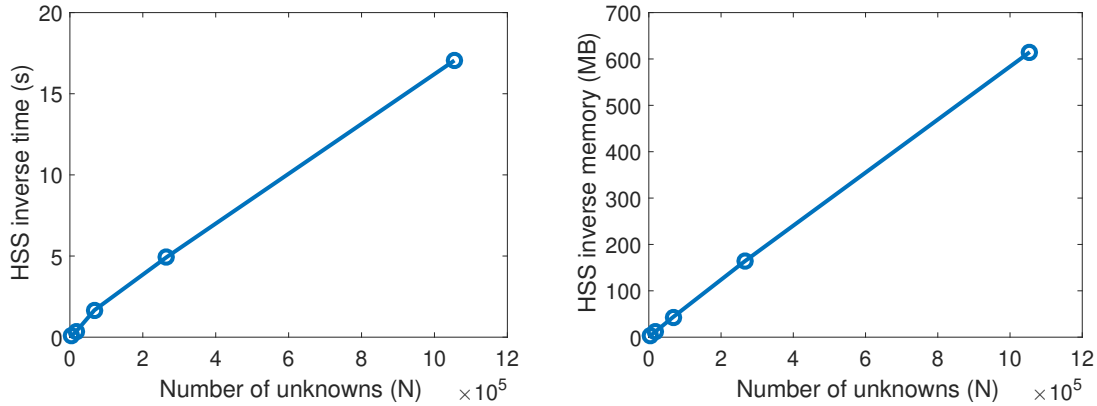
Fig. 7.2.: New HSS inverse performance for 2-layer bus array example. (a) Time performance. (b) Memory performance.

Table 7.1.: New HSS inversion relative residual for 2-layer bus array.

N	4,480	17,152	67,072	265,216	1,054,720
Relative residual	4.41E-15	6.27E-15	7.28E-15	1.52E-14	1.52E-14

7.3.2 Dielectric Cube Array

We then simulate a large-scale dielectric cube array incident by a plane wave at 3e+5 Hz. The number of array elements simulated is from $2 \times 2 \times 2$ to $16 \times 16 \times 16$, with 1 m spacing between adjacent cubes. The N ranges from 3,024 to 1,548,288. In Fig. 7.3, we plot the time and memory complexity with unknown N. The complexities are shown to agree very well with theoretical prediction. And the relative residual errors remains almost constant at 10^{-15} order as shown in Table 7.2.



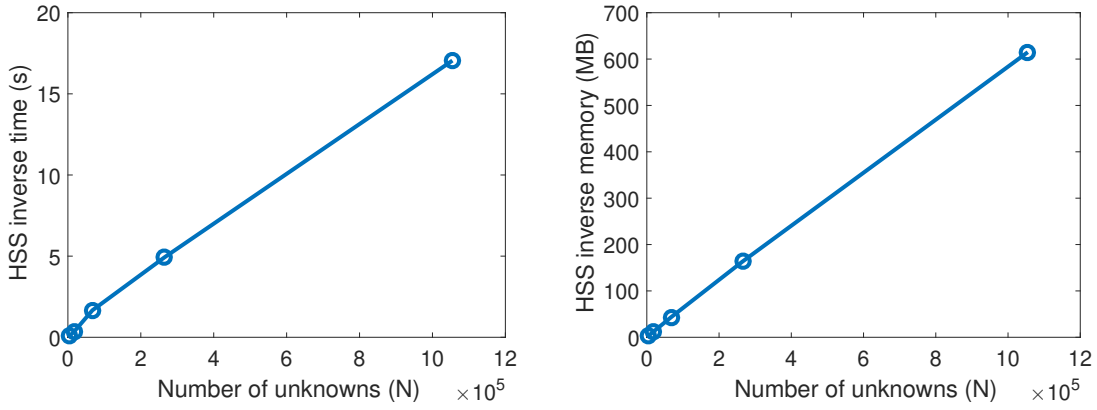
(a) (b)

Fig. 7.3.: New HSS inverse performance for cube array at 3e+5 Hz. (a) Time performance. (b) Memory performance.

Table 7.2.: New HSS inversion relative residual for cube array at 3e+5 Hz.

N	3,024	24,192	193,536	1,548,288
Relative residual	1.51E-15	1.49E-15	1.50E-15	1.50E-15

We also simulate a large-scale dielectric cube array from $2 \times 2 \times 2$ to $8 \times 16 \times 16$ elements at 3e+8 Hz, whose unknown number N is from 3,024 to 774,144. Each cube is of size $0.3 \times 0.3 \times 0.3$ m^3. The dielectric constant is 2.54 and the $leafsize$ used is 378. The center-to-center distance between cube elements is set to keep the admissibility parameter η constant. In Fig. 7.4, we plot the inverse time and memory as a function of N. A clear linear scaling with N is observed. And the relative residual errors are very small as shown in Table 7.3.



(a) (b)

Fig. 7.4.: New HSS inverse performance for cube array at 3e+8 Hz. (a) Time performance. (b) Memory performance.

Table 7.3.: New HSS inversion relative residual for cube array at 3e+8 Hz.

N	3,024	24,192	193,536	774,144
Relative residual	1.73E-15	2.37E-15	3.84E-15	4.17E-15

7.4 Conclusion

In this chapter, we propose a new HSS recursive inverse with directly controlled accuracy. The inverse matrix has the same structure as original HSS matrix, but the content is completely changed. Instead of performing formatted multiplications, we update the cluster bases of the inverse HSS matrix to accurately represent the inversion. Numerical results involving capacitance extraction and scattering analysis show that the proposed new algorithm is error controllable, and it can reach machine precision.

8. SUMMARY AND FUTURE WORK

8.1 Summary

The following is a summary of the contributions of this work:

8.1.1 Accuracy Controlled $\mathcal{H}^2$-Matrix-Matrix Product With Appended Cluster Bases in Linear Complexity

In this work, we develop a method to calculate the $\mathcal{H}^2$-matrix-matrix product with controlled accuracy in linear complexity. Instead of projecting the product onto existing cluster bases, we update the cluster bases by appending to the original cluster bases while calculating the matrix product. This product $\mathcal{H}^2$-matrix structure can be preserved or changed based on two multipliers.

8.1.2 Rank-Minimized and Structure-Kept $\mathcal{H}^2$-Matrix-Matrix Product in Linear Complexity with Controlled Accuracy

In this work, we propose a new algorithm to do the $\mathcal{H}^2$ matrix-matrix multiplication with controlled accuracy and also preserve the original $\mathcal{H}^2$-matrix structure, which is determined by admissibility condition. The cluster bases are completely changed instead of being appended to existing ones based on the prescribed accuracy during the computation of the matrix-matrix product. Meanwhile, we are able to keep the computational complexity to be linear.

8.1.3 Accuracy Directly Controlled Fast Direct Solution of General $\mathcal{H}^2$-Matrices

In this work, we develop a new fast direct solution for general $\mathcal{H}^2$ matrices, including both factorization and inversion. During the factorization of $\mathcal{H}^2$-matrices, we append new cluster bases to the original ones to account for the contributions of the fill-ins introduced due to elimination of previous clusters. This solution has a directly controlled accuracy, while still achieving the linear computational complexity.

8.1.4 Direct Solution of General $\mathcal{H}^2$-Matrices with Controlled Accuracy and Concurrent Change of Cluster Bases for Electromagnetic Analysis

In this work, we develop a new fast direct solution for general $\mathcal{H}^2$ matrices with completely new cluster bases. During the factorization of $\mathcal{H}^2$-matrices, we compute the cluster bases for the factorized matrices and inversion based on prescribed accuracy. So the resulting matrix's rank is minimized. The cluster bases are completely different from the cluster bases of the original $\mathcal{H}^2$-matrix.

8.1.5 Fast $\mathcal{H}^2$-Based Algorithms for Directly Solving Sparse Matrix in Linear Complexity

In this work, a new nested direct sparse matrix solution is developed for directly solving sparse system matrix in linear complexity. In this solver, we accelerate all dense matrix computations underlying the state-of-the-art multifrontal based direct solver by developing fast $\mathcal{H}^2$-based algorithms. Each computation is performed on small matrices of rank size, which is also the rank of the separator instead of the rank of the original 3-D problem. Direct solutions of millions of unknowns in a few minutes on a single CPU core have demonstrated the superior performance of the proposed direct sparse solver.

8.1.6 New HSS Recursive Inverse with Directly Controlled Accuracy

In this work, we propose a new way to calculate the inverse matrix of HSS-matrices. Instead of doing formatted multiplications and additions, whose accuracy is not controllable, we propose to calculate the new cluster cluster bases for the inversion. Furthermore, the inversion keeps original HSS-matrix structure and we are able to control the inversion accuracy.

8.2 Future Work

The presented dissertation also leads to possible research directions as listed below:

1. The new accuracy controlled and structure kept $\mathcal{H}^2$-matrix-matrix product in chapter 2 and 3 can be used to develop new error-controlled $\mathcal{H}^2$-matrix recursive inverse.

2. Further reduction on computational complexity will be pursued for solving electrically large surface and volume IEs in order to achieve a strict $O(N)$ complexity for large-scale electrodynamic analysis.

3. Employing hardware acceleration techniques, like parallelization on the $\mathcal{H}^2$-matrix based direct solvers and $\mathcal{H}^2$-matrix-matrix product so as to further reduce the computational time.

4. Various applications of this work will be conducted, such as the analysis of larger-scale realistic integrated circuits and scattering problems.

REFERENCES

REFERENCES

[1] W. Hackbusch and K. B., "A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: Introduction to $\mathcal{H}$-matrices," *Computing*, vol. 62, pp. 89–108, 1999.

[2] ——, "A sparse matrix arithmetic. Part II: Application to multi-dimensional problems," *Computing*, vol. 64, pp. 21–47, 2000.

[3] B. S., G. L., and H. W., *Hierarchical Matrices*. Lecture Note 21 of the Max Planck Institute for Mathematics in the Sciences, 2003.

[4] S. Borm, *Efficient Numerical Methods for Non-local Operators: $\mathcal{H}^2$-Matrix Compression Algorithms, Analysis*. Zurich, Switzerland: European Math. Soc., 2010.

[5] V. Rokhlin, "Diagonal forms of translation operators for the helmholtz equation in three dimensions," *Appl. Comput. Harmon. Anal.*, vol. 1, pp. 82–93, Dec 1993.

[6] W. C. Chew, J. M. Jin, E. Michielssen, and J. M. Song, "Fast and efficient algorithms in computational electromagnetics," *Norwood, MA: Artech House*, 2001.

[7] S. Börm, "$\mathcal{H}^2$-matrix arithmetics in linear complexity," *Computing*, vol. 77, pp. 1–28, 2006.

[8] W. Chai, D. Jiao, and C. C. Koh, "A direct integral-equation solver of linear complexity for large-scale 3D capacitance and impedance extraction," in *Proc. 46th ACM/EDAC/IEEE Design Automat. Conf.*, July 2009, pp. 752–757.

[9] W. Chai and D. Jiao, "Dense matrix inversion of linear complexity for integral-equation based large-scale 3-D capacitance extraction," *IEEE Trans. MTT*, vol. 59, no. 10, pp. 2404–2421, October 2011.

[10] ——, "An LU decomposition based direct integral equation solver of linear complexity and higher-order accuracy for large-scale interconnect extraction," *IEEE Trans. Adv. Packag.*, vol. 33, no. 4, pp. 794–803, Nov 2010.

[11] ——, "Direct matrix solution of linear complexity for surface integral-equation based impedance extraction of complicated 3-D structures," *Proceedings of the IEEE, special issue on Large Scale Electromagnetic Computation for Modeling and Applications*, vol. 101, pp. 372–388, Feb 2013.

[12] ——, "Linear-complexity direct and iterative integral equation solvers accelerated by a new rank-minimized-representation for large-scale 3-D interconnect extraction," *IEEE Trans. Microw. Theory Techn.*, vol. 61, no. 8, pp. 2792–2805, Aug 2013.

[13] S. Omar and D. Jiao, "A linear complexity direct volume integral equation solver for full-wave 3-D circuit extraction in inhomogeneous materials," *IEEE Trans. Microw. Theory Techn.*, vol. 63, no. 3, pp. 897–912, Mar 2015.

[14] ——, "An $O(N)$ iterative and $O(NlogN)$ direct volume integral equation solvers for large-scale electrodynamic analysis," *the 2014 International Conference on Electromagnetics in Advanced Applications (ICEAA)*, Aug 2014.

[15] ——, "Minimal-rank $\mathcal{H}^2$-matrix based iterative and direct volume integral equation solvers for large-scale scattering analysis," *Proc. IEEE Int. Symp. Antennas Propag.*, Jul 2015.

[16] ——, "$O(N)$ iterative and $O(NlogN)$ fast direct volume integral equation solvers with a minimal-rank $\mathcal{H}^2$-representation for large-scale 3-D electrodynamic analysis," *http://arxiv.org/abs/1703.01175*, Mar 2017.

[17] P. G. Martinsson and V. Rokhlin, "A fast direct solver for scattering problems involving elongated structures," *J. Comput. Phys.*, vol. 221, pp. 288–302, 2007.

[18] J. Shaeffer, "Direct solve of electrically large integral equations for problem sizes to 1 m unknowns," *IEEE Trans. Antennas Propag.*, vol. 56, no. 8, pp. 2306–2313, Aug 2008.

[19] R. J. Adams, Y. Xu, X. Xu, J. Choi, S. D. Gedney, and F. X. Canning, "Modular fast direct electromagnetic analysis using local-global solution modes," *IEEE Trans. Antennas Propag.*, vol. 56, no. 8, pp. 2427–2441, Aug 2008.

[20] E. Winebrand and A. Boag, "A multilevel fast direct solver for em scattering from quasi-planar objects," *Proc. Int. Conf. Electromagn. Adv. Appl.*, pp. 640–643, Sept 2009.

[21] L. Greengard, D. Gueyffier, P. G. Martinnson, and V. Rokhlin, "Fast direct solvers for integral equations in complex three-dimensional domains," *Acta Numerica*, vol. 18, pp. 243–275, May 2009.

[22] A. Heldring, J. M. Rius, J. M. Tamayo, J. Parron, and E. Ubeda, "Multiscale compressed block decomposition for fast direct solution of method of moments linear system," *IEEE Trans. Antennas Propag.*, vol. 59, no. 2, pp. 526–536, Feb 2011.

[23] A. Freni, P. D. Vita, P. Pirinoli, L. Matekovits, and G. Vecchi, "Fast-factorization acceleration of MoM compressive domain-decomposition," *IEEE Trans. Antennas Propag.*, vol. 59, no. 12, pp. 4588–4599, Dec 2011.

[24] ——, "Fast direct solver for essentially convex scatterers using multilevel non-uniform grids," *IEEE Trans. Antennas Propag.*, vol. 62, pp. 4314–4324, 2014.

[25] H. Guo, Y. Liu, J. Hu, and E. Michielssen, "A butterfly-based direct integral equation solver using hierarchical LU factorization for analyzing scattering from electrically large conducting objects," *arXiv preprint arXiv:1610.00042*, 2016.

[26] S. Ambikasaran and E. Darve, "The inverse fast multipole method," *arXiv: 1407.1572*, 2014.

[27] P. Coulier, H. Pouransari, and E. Darve, "The inverse fast multipole method: using a fast approximate direct solver as a preconditioner for dense linear systems," *arXiv: 1508.01835v2*, 2016.

[28] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, "Fast algorithms for hierarchically semiseparable matrices," *Numerical Linear Algebra with Applications*, vol. 17, pp. 953–976, 2010.

[29] A. B. Manić, B. M. Notaroš, F. Rouet, and X. S. Li, "Efficient EM scattering analysis based on MoM, HSS direct solver, and RRQR decomposition," *Proc. IEEE Int. Symp. Antennas Propag.*, Jul 2015.

[30] A. B. Manić, A. Smull, B. M. Notaroš, F. Rouet, and X. S. Li, "Efficient scalable parallel higher order direct MoM-SIE method with hierarchically semiseparable structures for 3d scattering," *IEEE Trans. AP*, Feb 2017.

[31] D. H. Schaubert, D. R. Wilton, and A. W. Glisson, "A tetrahedral modeling method for electromagnetic scattering by arbitrarily shaped inhomogeneous dielectric bodies," *IEEE Trans. Antennas Propag.*, vol. 32, no. 1, pp. 77–85, 1984.

[32] W. Chai and D. Jiao, "Theoretical study on the rank of integral operators for broadband electromagnetic modeling from static to electrodynamic frequencies," *IEEE Trans. Compon. Packag. Manuf. Technol.*, vol. 3, no. 12, pp. 2113–2126, Dec 2013.

[33] M. Ma and D. Jiao, "Accuracy controlled $\langle^2$-matrix-matrix product in linear complexity and its applications," *2018 IEEE International Symposium on Antennas and Propagation & USNC URSI National Radio Science Meeting*, Jul 2018.

[34] ——, "Accuracy-controlled and structure-preserved $\langle^2$-matrix-matrix product in linear complexity," *2018 International Conference on Electromagnetics in Advanced Applications (ICEAA)*, Sep 2018.

[35] K. Nabors and J. White, "Fastcap: A multipole accelerated 3-D capacitance extraction program," *IEEE Trans. on CAD*, vol. 10, no. 11, pp. 1447–1459, Nov 1991.

[36] Z. Zhu, B.Song, and J.White, "Algorithms in fastimp: A fast and wide- band impedance extraction program for complicated 3-D geometries," in *Proc. 40th Design Automat. Conf.*, 2003, pp. 712–717.

[37] W. Shi, J. Liu, N. Kakani, and T. Yu, "A fast hierarchical algorithm for 3-D capacitance extraction," *IEEE Trans. on CAD*, pp. 330–336, 2002.

[38] M. Ma and D. Jiao, "Accuracy directly controlled fast direct solution of general $\mathcal{H}^2$-matrices and its application to solving electrodynamic volume integral equations," *IEEE Trans. MTT*, vol. 66, no. 1, pp. 35–48, Aug 2017.

[39] ——, "Accuracy controlled direct integral equation solver of linear complexity with change of basis for large-scale interconnect extraction," *IEEE MTT-S International Microwave Symposium (IMS2018)*, Jun 2018.

[40] ——, "Direct solution of general $\mathcal{H}^2$-matrix with controlled accuracy and change of cluster bases for large-scale electromagnetic analysis," *IEEE MTT-S International Conference on Numerical Electromagnetic and Multiphysics Modeling and Optimization (NEMO2018)*, Aug 2018.

[41] ——, "Linear-complexity direct integral equation solver with explicit accuracy control for large-scale interconnect extraction," *Appl. Comput. Electromag. (ACES)*, Mar 2018.

[42] B. Zhou and D. Jiao, "Direct finite element solver of linear complexity for large-scale 3-d electromagnetic analysis and circuit extraction," *IEEE Trans. MTT*, vol. 63, no. 10, pp. 3066–3080, Oct 2015.

[43] H. Boltz, "Matrix inversion lemma," *Wikipedia 2011*, p. [Online], Available:https://en.wikipedia.org/wiki/Invertible_matrix.

VITA

VITA

Miaomiao Ma received the B.S. degree in electronic engineering and information science from the University of Science and Technology of China, Hefei, China, in 2014. Since 2014, she has been working toward the Ph.D. degree at Purdue University, West Lafayette, IN, USA. She is with the School of Electrical and Computer Engineering, Purdue University, as a member of the On-Chip Electromagnetics Group. Her current research interests include computational electromagnetics, fast and high-capacity numerical methods and scattering analysis.

Ms. Ma was the recipient of an Honorable Mention Award of the IEEE International Symposium on Antennas and Propagation in 2016 and 2018. She also received the Best Student Paper Award from the IEEE International Conference on Wireless Information Technology and Systems (ICWITS) and Applied Computational Electromagnetics (ACES) in 2016.