TAIWANESE JOURNAL OF MATHEMATICS
Vol. 23, No. 2, pp. 473–501, April 2019
DOI: 10.11650/tjm/181113

Compressed Hierarchical Schur Algorithm for Frequency-domain Analysis of Photonic Structures

Cheng-Han Du, Yih-Peng Chiou and Weichung Wang*

In memory of Professor Hwai-Chiuan Wang

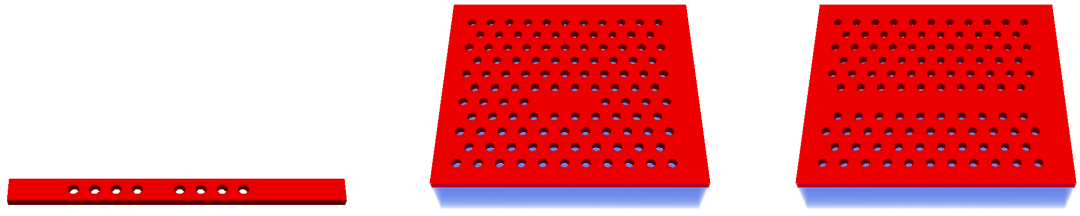
Abstract. Three-dimensional finite-difference frequency-domain analyses of partially periodic photonic structures result in large-scale ill-conditioned linear systems. Due to the lack of efficient preconditioner and reordering scheme, existed general-purpose iterative and direct solvers are inadequate to solve these linear systems in time or memory. We propose an efficient direct solver to tackle this problem. By exploring the physical properties, the coefficient matrix structure, and hardware computing efficiency, we extend the concepts of grid geometry manipulation and multi-level Schur method to propose the Compressed Hierarchical Schur algorithm (CHiS). The proposed CHiS algorithm can use less memory and remove redundant computational workloads due to the homogeneity and periodicity of photonic structures. Moreover, CHiS relies on dense BLAS3 operations of sub-matrices that can be computed efficiently with strong scalability by the latest multicore processors or accelerators. The implementation and benchmarks of CHiS demonstrate promising memory usage, timing, and scalability results. The feasibility of future hardware acceleration for CHiS is also addressed using computational data. This high-performance analysis tool can improve the design and modeling capability for various photonic structures.

1. Introduction

Photonics has been an active research topic for decades. It has been widely applied in scientific and industrial applications, such as high-speed signal transmission, processing, and sensing, just to name a few. Among the various photonic devices, *partially periodic* photonic structures have played an important role in photonics-related developments and studies. These structures are the building blocks of complete photonic circuits and networks. Partially periodic photonic structures are frequently utilized in wavelength filtering,

Received May 7, 2018; Accepted November 25, 2018.
Communicated by Suh-Yuh Yang.
2010 *Mathematics Subject Classification.* 65F05, 65Y05, 65Z05.
Key words and phrases. partially periodic photonic structures, finite-difference frequency-domain method, direct solver for ill-conditioned linear systems, Schur complements, BLAS3 operations, multithreading parallelism.
*Corresponding author.

wave confinement, and various manipulations of propagation behaviors. For example, finite structure periodicity, such as the frequency filters shown in Figure 1.1a, can lead to different wave behaviors, and a rigorous numerical analysis is required. Moreover, photonic crystal-based devices with partially periodic structures are important components in photonic designs. For example, point defects in photonic crystals, as shown in Figure 1.1b, can work as a resonance cavity [1, 31]. Linear defects, as shown in Figure 1.1c, can be used as photonic crystal waveguides [11, 30]. It is even possible to combine multiple linear and point defects for wavelength filtering [17].



(a) (b) (c)

Figure 1.1: Photonic structures with partial (finite or imperfect) periodicity are illustrated by (a) a frequency filter, (b) an L_3 photonic crystal resonance cavity, and (c) a photonic crystal waveguide.

Efficient simulation algorithms for partially periodic photonic structures are in high demand for rigorous simulations and practical designs for designing future photonic systems. Several numerical methods and algorithms have been developed to analyze field behaviors in perfectly periodic structures. Examples include the planewave expansion method [9, 10, 25] and finite difference time domain (FDTD) methods [28]. Time-domain-based formulations, such as FDTD, are a popular choice for modeling partially periodic photonic structures because of their feasible computation costs in most cases. However, in certain analyses, such as high-Q resonator and extremely narrow-band filters, the FDTD analysis requires prolonged simulation times to achieve a finer frequency resolution. A three-dimensional (3D) FDTD simulation can take a considerable amount of time to complete if the simulation requires many time steps.

Alternatively, frequency-domain simulations such as finite difference frequency domain (FDFD) methods [8] yield wave behavior analysis at specified frequencies. Designers can simply specify the target frequency range and perform corresponding frequency-domain analyses. Furthermore, FDFD can be used to rigorously analyze spectral (frequency) characteristics. A simple implementation of arbitrarily shaped total-field/scattered-field simulations has also been reported [18], which allows easy customization of different incident waves and enables great flexibility of the FDFD method for a wide array of photonic

simulations. In short, frequency-domain simulation is a vital design tool in photonic device analysis and component characterization, and it is complementary to FDTD simulations. An efficient FDFD simulation can be beneficial to the photonic research community and industry.

However, in general FDFD simulation, multiple lattices must be modeled, and their full-vectorial 3D rigorous analyses require tremendous computational resources while modeling partially periodic photonic structures. In particular, the key computational component in the FDFD method is a linear system solver and there are three challenges for solving the linear systems. First, the dimension of the linear systems in the 3D FDFD simulations can be very large, and tremendous computational resources are required. Second, the matrices corresponding to these problems are mostly ill-conditioned when the computational domain is considerably larger than the wavelength scale and a perfectly matched layer (PML) is applied [29]. Third, novel algorithms are required to efficiently take advantage of the latest computing capacities, such as multicore parallelism, to reduce the computation time. To overcome these three challenges, we propose an efficient direct solver described in the next section.

1.1. CHiS: an efficient direct solver for FDFD simulations

We develop an FDFD simulation tool for modeling partially periodic photonic structures. The kernel of the simulator is a fast direct linear system solver based on the proposed compressed hierarchical Schur algorithm (CHiS). The CHiS algorithm has been designed by exploring the physical properties of partially periodic photonic structures and matrix structures such that we can reduce workloads, use less memory, and take advantage of the computing capacities of recent hardware accelerators.

The main concepts of the proposed CHiS are derived from geometry-based nested dissection method and recursive multi-level Schur method. On top of these dissection and Schur methods, we further propose two compression schemes to use less memory space and remove redundant computational workloads. These compressions greatly improve overall computation efficiency due to a significant reduction of small BLAS3 operations. As a matrix-free implementation, CHiS avoids complete graph-based partitioning and global sparse pattern analysis. In the matrix-free setting, CHiS can also quickly identify the portion to be compressed in less than one second even for large matrices. Last but not least, CHiS can be accelerated by various modern high-performance processing units that dense BLAS3 operations such as ZGEMM and ZGETRS are efficiently executed.

This efficient FDFD analysis tool can facilitate rapid developments of future photonic circuit designs and networks [2], particularly in component design and optimization. In addition, the proposed framework is applicable to other numerical simulations with

Cartesian or staggered Cartesian grids. The effectiveness of this framework depends on the duplication property of the physical domain after the computational domain decomposition process. The simulation parameters should be tuned based on physical structures and grid alignment to expose as many duplicate sub-structures as possible. If the portion of identical sub-structures is large, then the proposed framework can provide significant memory and workload savings.

The remainder of this paper is organized as follows. We discuss the governing equations, the discretization scheme, perfectly matched layer boundary conditions, and the resulting linear system in Section 2. We propose the hierarchical Schur algorithm in Section 3 by discussing how the hierarchical Schur matrix is formed and factorized. In Section 4, we propose the compressed hierarchical Schur algorithm that contains two compression schemes, allowing us to solve the linear system by using less memory storage and performing fewer computational tasks. We present the numerical simulation results of several photonic structures and analyze the computational performance in Section 5. Finally, we conclude this paper with a discussion of future works in Section 6.

2. Problem formulation

We derive the target linear system from the following time-harmonic Maxwell equations. Assuming the time-varying term $e^{-\imath\omega t}$, we have the equations

$$\begin{aligned} \nabla \times \vec{E} = \imath\omega\mu\vec{H}, \quad \nabla \times \vec{H} = \vec{J} - \imath\omega\varepsilon\vec{E}, \\ \nabla \cdot (\varepsilon\vec{E}) = \rho, \quad \nabla \cdot (\mu\vec{H}) = 0. \end{aligned} \tag{2.1}$$

Here, $\vec{E}$ is the electric field, $\vec{H}$ is the magnetic field, $\vec{J}$ is the electric current, ρ is the charge source, and ω, μ, and ε are the angular frequency, permeability, and permittivity, respectively. If we further assume nonmagnetic materials in our simulations with $\mu = \mu_0$ and $\varepsilon = \varepsilon_0\varepsilon_r$, the vector wave equation

$$-\nabla \times \nabla \times \vec{E} + k_0^2\varepsilon_r\vec{E} = \vec{f}_{\text{src}} \tag{2.2}$$

can be derived from (2.1). The constants μ_0 and ε_0 are vacuum permeability and vacuum permittivity, respectively, ε_r is relative permittivity, $\vec{f}_{\text{src}} = -\imath\omega\mu_0\vec{J}$, $k_0 = \omega\sqrt{\mu_0\varepsilon_0} = 2\pi/\lambda_0$, λ_0 is the vacuum wavelength, and $\vec{J}$ is the current source [26, 27].

The double-curl operator in (2.2) can be discretized using Yee's scheme with central differences [32]. By using the uniform grid sizes Δx, Δy, and Δz, (2.2) can be discretized as shown in (7.1)–(7.3) of the appendix. By letting N_x, N_y, and N_z be the grid numbers and packing the x-, y-, and z-field components together, we can rewrite (7.1)–(7.3) as a linear system $Ax = b$, where x is defined in (7.4) and b is the corresponding vector of discretized f_{src}.

A perfectly matched layer (PML) based on stretched coordinates has been proposed for the frequency-domain formulation [4] to handle the truncation near domain boundaries such that the artificial reflection from the domain boundary can be effectively suppressed. When using stretched-coordinate-based PML to absorb outward wave propagation, the grid sizes in the PML region are chosen to be nonuniform to achieve optimal absorption. The vector wave equation equipped with stretched-coordinate-based PML is expressed as

$$-\nabla_{\mathrm{PML}} \times \nabla_{\mathrm{PML}} \times \vec{E} + k_0^2 \varepsilon_r \vec{E} = \vec{f}_{\mathrm{src}}, \tag{2.3}$$

where $\nabla_{\mathrm{PML}} = \widehat{x}\frac{1}{s_x}\frac{\partial}{\partial x} + \widehat{y}\frac{1}{s_y}\frac{\partial}{\partial y} + \widehat{z}\frac{1}{s_z}\frac{\partial}{\partial z}$, $\widehat{x}$, $\widehat{y}$, and $\widehat{z}$ are the unit vectors in the Cartesian coordinate system, the stretching factors $s_n = 1 + \imath\frac{\sigma_n}{\omega\varepsilon_0}$ for $n = x, y, z$, and σ_ns are position- and problem-dependent variables. Because of the scaling distribution of σ_ns, the computational domain has nonuniform grids.

In short, by using Yee's scheme and PML, we can derive the linear system

$$A_{\mathrm{PML}} x = b. \tag{2.4}$$

The matrix A_{PML} is derived from (2.3), and it includes stretched-coordinate-based PML. The matrix A_{PML} is complex and non-Hermitian as shown in [3]. The unknown vector x is defined in (7.4), and b is the discrete right-hand-side vector corresponding to $\vec{f}_{\mathrm{src}}$. In the next sections, we focus on how the linear system (2.4) can be solved efficiently.

To the best of our knowledge, little efficient iterative or direct methods have been proposed to solve the linear system (2.4). First, iterative methods can handle large sparse linear systems in general. However, finding an efficient preconditioner for the ill-conditioned target problems is non-trivial. For example, iterative methods are used to solve some specific problem settings with PML in [26], but no preconditioner is used. The condition numbers of several test linear systems are improved by a modification of the formulations [27]. However, the convergence performance varies, and the performance depends on the simulation parameters, such as physical parameters, excitation current, and other numerical settings. Several iterative linear system solvers such as STRUMPACK [6] with hierarchical semi-separables method and algebraic recursive multi-level solver [13, 19] are also reported, while their convergence properties for the general FDFD photonic simulation are not fully investigated yet. In other words, these existing methods are designed to solve the specific problems, and they are not efficient for the target problem. Second, direct methods do not involve preconditioning, and they are robust to solve ill-conditioned problems. Consequently, a direct approach has been used to perform 3D geophysical analysis by FDFD [15]. However, significant fill-ins typically occur in the eliminations that may greatly slow down the solver. Third, computer architectures evolve rapidly, and the latest processors are generally equipped with multiple or many (lightweight) computing

cores. Algorithms must be redesigned to fully use these computational capacities. Hardware accelerated sparse linear system solvers have been reported [20–22,33]. However, how we can use the computing capacities along the trend of the latest and coming processors to solve the target problems remains an open problem.

3. Hierarchical Schur algorithm (HiS)

The goal of this section is to derive the hierarchical Schur method (HiS) to solve the linear system (3.1). The concept is a variation of recursive multi-level solver [13,19], while we perform exact factorization instead of an approximated one. We first discuss how we decompose the computational domain, and then we form the hierarchical Schur coefficient matrix in Section 3.1. This hierarchical Schur coefficient matrix is denoted as A_{HS} and defined in (3.2). The matrix A_{HS} is a reorder of A_{PML}; however, A_{HS} has a particular form that leads to a balanced binary elimination tree. By taking advantage of these properties of the coefficient matrix, we discuss how we can factorize the matrix in Section 3.2. The algorithm and its main computational workloads are presented in Section 3.3.

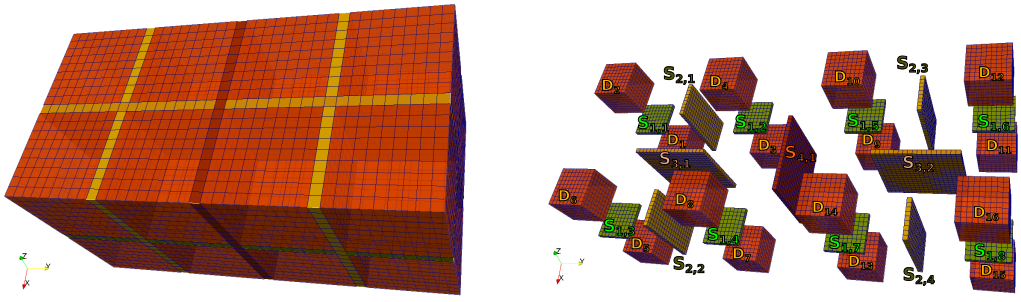
3.1. Hierarchical Schur coefficient matrix and the elimination tree

Because Yee's mesh is geometrically and regularly structured in a cuboid computational domain, we can decompose the domain and order the grid points to obtain the hierarchical Schur coefficient matrix by exploring the geometric and Yee's discretization properties on the computational domain directly [14, 24]. This approach effectively reduces the workloads of global sparse pattern analysis and fill-reduction reordering. Furthermore, A_{HS} can be implicitly defined. We can quickly compute the entries of A_{HS} on-the-fly when we need them during the computations without explicitly storing the whole A_{HS} beforehand. In other words, because the grid geometry and the sparse pattern are regular, we can directly order the matrix elements in a way that we prefer rather than analyzing the complete sparse pattern of the matrix efficiently.

In contrast, the matrix A_{HS} can be (approximately) obtained by applying the nested dissection approach [12,16] to the matrix A_{PML}. However, the nested dissection approach needs to analyze the sparse pattern of the matrix and perform graph partitioning to reorder the matrix. This procedure may be time consuming, particularly when a high-quality partitioning is required. Moreover, some partitioning parameters and schemes generally need careful tuning for efficient computing.

For the sake of simplicity and clarity, we use an example to describe how we can derive the hierarchical Schur coefficient matrix. We consider the cuboid computational domain shown in Figure 3.1a. The grid numbers of this cuboid computational domain example are

$(N_x, N_y, N_z) = (19, 39, 19)$. As shown in Figure 3.1b, the entire domain is partitioned into 16 *subdomains* evenly by the *separators* recursively. Each subdomain contains $9 \times 9 \times 9$ grids. The subdomains are denoted as D_1 to D_{16}. The separators are denoted as $S_{i,j}$, and the indices i and j stand for the separator level and separator index in the elimination tree (to be discussed later), respectively.

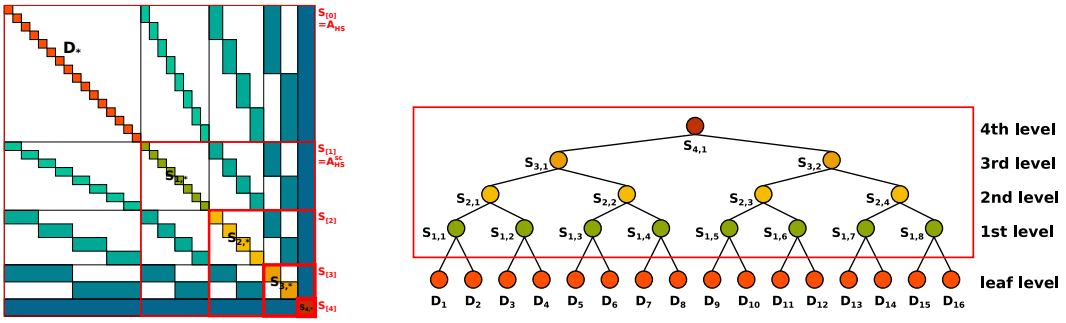


(a) (b)

Figure 3.1: A schema of customized domain decomposition. (a) The sample computational domain with $(N_x, N_y, N_z) = (19, 39, 19)$ grid points. (b) The subdomains D_* and separators $S_{*,*}$.

Next, based on the decomposition, we group the grid points in the subdomains and separators together to form the subdomain sub-matrices and separator sub-matrices. These sub-matrices contain the grid points that are ordered in the same manner locally. Furthermore, these sub-matrices are ordered globally such that the corresponding subdomain sub-matrices D_* and separator sub-matrices $S_{*,*}$ are located in the diagonal blocks of the hierarchical Schur coefficient matrix A_{HS} as shown in Figure 3.2a. The off-diagonal rectangular sub-matrices of A_{HS} are the *interface* sub-matrices, which involve the discretization of the grid points in the subdomain and separators. We will define these interface sub-matrices in detail after introducing the elimination tree.

The hierarchical Schur coefficient matrix A_{HS} and its corresponding elimination tree of this particular example are illustrated in Figures 3.2a and 3.2b, respectively. The leaf-level nodes of the elimination tree consist of the subdomain sub-matrices D_1 to D_{16}. Their parent nodes are the first-level separator sub-matrices $S_{1,*}$. The elimination tree also shows the second-, third-, and fourth-level separators $S_{2,*}$, $S_{3,*}$, and $S_{4,*}$, respectively. Let $\mathcal{A}$ be a subdomain or a separator sub-matrix, and we assume that it is a child of a sub-matrix $\mathcal{B}$ in the elimination tree. We use $\mathcal{I}_L(\mathcal{A}, \mathcal{B})$ (or $\mathcal{I}_U(\mathcal{A}, \mathcal{B})$) to denote interface sub-matrices between $\mathcal{A}$ and $\mathcal{B}$ that belong to the lower (or upper) off-diagonal part of the matrix A_{HS}. The Schur complement of a matrix M is denoted as M^{sc}. For example, the Schur complement of A_{HS} is $A_{\text{HS}}^{\text{sc}}$.



(a) (b)

Figure 3.2: (a) The hierarchy of recursive Schur complement matrices $S_{[0]} = A_{\mathrm{HS}}$, $S_{[1]} = A^{\mathrm{sc}}_{\mathrm{HS}}$, $S_{[2]}$, $S_{[3]}$, and $S_{[4]}$. (b) The corresponding elimination tree of the computational domain shown in Figure 3.1. The nodes within the red box are associated with the sub-matrices of $S_{[1]}$.

The aforementioned customized domain decomposition and grid point ordering scheme suggest that we should rewrite the target linear system (2.4) as

$$A_{\mathrm{HS}} x = b, \tag{3.1}$$

where $x = [x_{D_1}, x_{D_2}, \ldots, x_{D_{16}}, x_{S_{[1]}}]^T$ and $b = [b_{D_1}, b_{D_2}, \ldots, b_{D_{16}}, b_{S_{[1]}}]^T$. Note that the unknown vectors x's and the right-hand-side vectors b's in (2.4) and (3.1) are equivalent up to a permutation. In (3.1), the hierarchical Schur coefficient matrix is

$$A_{\mathrm{HS}} = \left[\begin{array}{cccc|c} D_1 & & & & I_U(D_1, S_{[0]}) \\ & D_2 & & & I_U(D_2, S_{[0]}) \\ & & \ddots & & \vdots \\ & & & D_{16} & I_U(D_{16}, S_{[0]}) \\ \hline I_L(D_1, S_{[0]}) & I_L(D_2, S_{[0]}) & \cdots & I_L(D_{16}, S_{[0]}) & A^{\mathrm{sc}}_{\mathrm{HS}} \end{array}\right]. \tag{3.2}$$

The lower-right block of A_{HS} is the Schur complement $A^{\mathrm{sc}}_{\mathrm{HS}}$. It consists of the separator sub-matrices in the diagonal and the corresponding interface sub-matrices in the off-diagonal, as shown in Figure 3.2a. This figure also suggests that the hierarchical Schur coefficient matrix has a recursive block arrow matrix structure. As shown in Figure 3.2b, the corresponding binary elimination tree is perfectly balanced in the sense that all of the subdomain sub-matrices and the separator sub-matrices in the same level of the elimination tree have identical dimensions and matrix entry orderings.

3.2. Factorization of the Hierarchical Schur matrix

The LU factorization of the matrix A_{HS} can be computed by the three-stage operations recursively. These operations include factorization, solve, and update. Take A_{HS} shown in Figure 3.2a as an example; the matrix has four levels of Schur complement hierarchy, and we need to perform such operations four times plus a final factorization of the matrix $S_{4,1}$.

The first-level factorization involves the sub-matrices D_*, $S_{1,*}$, $S_{2,*}$, $S_{3,*}$, $S_{4,*}$, and the mutually interacting interface sub-matrices. In particular, for $i = 1, \ldots, 16$, we (i) factorize the subdomain sub-matrices $D_i = L_i U_i$; (ii) solve the linear system with multiple right-hand-side vectors to compute $\mathcal{I}_L(D_i, A^{\mathrm{sc}}_{\mathrm{HS}})U_i^{-1}$ and $L_i^{-1}\mathcal{I}_U(D_i, A^{\mathrm{sc}}_{\mathrm{HS}})$, or only $D_i^{-1}\mathcal{I}_U(D_i, A^{\mathrm{sc}}_{\mathrm{HS}})$ with an equivalent result; and (iii) update the first-level Schur complement

$$S_{[1]} = S^{\mathrm{sc}}_{[0]} - \sum_{i=1}^{16} \mathcal{I}_L(D_1, S^{\mathrm{sc}}_{[0]})D_i^{-1}\mathcal{I}_U(D_i, S^{\mathrm{sc}}_{[0]}), \tag{3.3}$$

where $S_{[0]} = A_{\mathrm{HS}}$ and $S^{\mathrm{sc}}_{[0]} = A^{\mathrm{sc}}_{\mathrm{HS}}$. In other words, by performing these three-stage operations, we complete the first-level factorization and factor $A_{\mathrm{HS}} = L_{[1]}U_{[1]}$, where

$$L_{[1]} = \begin{bmatrix} & L_1 & & & & \\ & & L_2 & & & \\ & & & \ddots & & \\ & & & & L_{16} & \\ \mathcal{I}_L(D_1, S_{[0]})U_1^{-1} & \mathcal{I}_L(D_2, S_{[0]})U_2^{-1} & \cdots & \mathcal{I}_L(D_{16}, S_{[0]})U_{16}^{-1} & I \end{bmatrix} \tag{3.4}$$

and

$$U_{[1]} = \begin{bmatrix} U_1 & & & & L_1^{-1}\mathcal{I}_U(D_1, S_{[0]}) \\ & U_2 & & & L_2^{-1}\mathcal{I}_U(D_2, S_{[0]}) \\ & & \ddots & & \vdots \\ & & & U_{16} & L_{16}^{-1}\mathcal{I}_U(D_{16}, S_{[0]}) \\ & & & & S_{[1]} \end{bmatrix}. \tag{3.5}$$

Note that $S^{\mathrm{sc}}_{[0]}$ $(= A^{\mathrm{sc}}_{\mathrm{HS}})$ and $S_{[1]}$ have the same block sparsity, and both of them are arrowhead block matrices, as shown in Figure 3.2a.

The second-level factorization involves the *updated* $S_{1,*}$, $S_{2,*}$, $S_{3,*}$, $S_{4,*}$, and the mutual interface sub-matrices. In the second-level factorization, we apply the three-stage operations on Schur complement $S_{[1]}$ to complete the second-level factorization

and obtain the second-level Schur complement matrix $S_{[2]}$. In particular, we (i) factorize the updated $S_{1,i} = L_i U_i$, (ii) solve the multiple right-hand-side systems to obtain $I_L(S_{1,i}, S^{\text{sc}}_{[1]})U_i^{-1}$ and $L_i^{-1} I_U(S_{1,i}, S^{\text{sc}}_{[1]})$, and (iii) update the second-level Schur complement $S_{[2]} = S^{\text{sc}}_{[1]} - \sum_{i=1}^{8} \mathcal{I}_L(S_{1,i}, S^{\text{sc}}_{[1]}) S_{1,i}^{-1} \mathcal{I}_U(S_{1,i}, S^{\text{sc}}_{[1]})$. These operations are similar to the operations shown in (3.3)–(3.5) by replacing $S_{[0]}$ $(= A_{\text{HS}})$ with $S_{[1]}$.

We continue such three-stage operations recursively to complete the third- and fourth-level factorizations. The final task is to perform LU-factorization of $S_{[4]}$, which is the updated $S_{4,1}$. See Figure 3.2a for the matrices $S_{[1]}$, $S_{[2]}$, and $S_{[3]}$ and Figure 3.2b for the matrices involved in the first-, second-, third-, fourth-, and final-level factorizations. The block sparsity of the matrix A_{HS} and the factored lower and upper triangular matrices remains unchanged. That is, there is no fill-in in terms of block sub-matrices during the entire LU factorization. Moreover, $S_{[0]}$ $(= A_{\text{HS}})$, $S_{[1]}$, $S_{[2]}$, $S_{[3]}$, and $S_{[4]}$ are all block arrowhead matrices, and they are computed recursively.

3.3. The HiS algorithm and its computational tasks

We have discussed how the matrix A_{HS} can be factorized. The proposed factorization scheme is summarized conceptually in Algorithm 3.1. The operations that consume the majority of the computation time in the algorithm are Steps 3 and 4.

Algorithm 3.1 HiS: Hierarchical Schur Factorization of A_{HS} defined in (3.1)

1: For each subdomain sub-matrix D_j, perform sparse factorization and solve multiple right-hand-side linear systems $D_j^{-1}\mathcal{I}_U(D_j, *)$ corresponding to its adjacent interfaces.
2: Perform sparse matrix multiplications $\mathcal{I}_L(D_j, *)D_j^{-1} I_U(D_j, *)$ for all j and update $S^{\text{sc}}_{[0]}$ to $S_{[1]}$.
3: For the level-i separators, factorize the dense sub-matrices $S_{i,j} = L_{i,j}U_{i,j}$ and solve the multiple right-hand-side linear systems $S_{i,j}^{-1} I_U(S_{i,j}, *)$ related to interface sub-matrices.
4: Perform dense matrix multiplications of $\mathcal{I}_L(S_{i,j}, *)S_{i,j}^{-1}\mathcal{I}_U(S_{i,j}, *)$ and update $S^{\text{sc}}_{[i]}$ to $S_{[i+1]}$.
5: Move to level-$(i+1)$ and perform Steps 3–5 until the final level of the elimination tree.

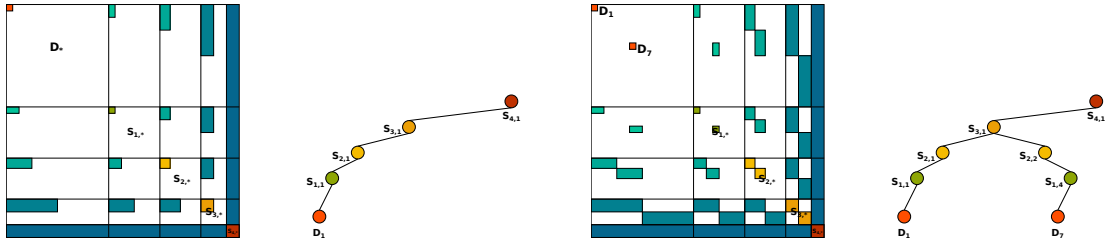
Fortunately, these dense operations are included in high-performance linear algebra libraries that are available in almost all of the advanced computing environments. For example, matrix multiplication can be executed by the ZGEMM routine from BLAS, while multiple-RHS solves and LU factorization can be computed by ZGETRS and ZGETRF from the LAPACK library, respectively. Moreover, these operations greatly rely on BLAS3 operations, which are also optimized for many hardware architectures, such as Intel MKL for multicore and manycore CPUs and CUBLAS and MAGMA for GPU accelerators. Consequently, these operations can be executed efficiently on the latest computer architectures.

4. Compressed Hierarchical Schur algorithm (CHiS)

We have proposed HiS in Section 3. Motivated by the homogeneity and periodicity of photonic structures, we further develop two approaches to "compress" the hierarchical Schur coefficient matrix A_{HS} such that we can consume less storage and perform less computations to solve the linear system (3.1). In the first scheme, as shown in Section 4.1, we consider the identical subdomains and identical separators. In the second scheme, as shown in Section 4.2, we focus on the identification of the leaf-level interface sub-matrices. The overall procedure of CHiS is summarized in Section 4.3.

4.1. De-duplication of identical subdomains and separators

The first step of the elimination tree de-duplication (ETD) is to identify identical subdomains by determining whether there is any difference in the corresponding physical parameters (e.g., material permittivity and structure locations) and the grid properties (e.g., grid size uniformity and PML settings). We then apply the same rules to check the identity of the separator sub-matrices in all levels. Clearly, the children of any two identical separator sub-matrices must also be the same. For example, if the permittivity and grid sizes in separators $S_{1,1}$ and $S_{1,2}$ are identical, then we also need $D_1 = D_3$ and $D_2 = D_4$ to confirm that $S_{1,2}$ is a duplication of $S_{1,1}$. After identifying these duplicated sub-matrices, we can store one representative sub-matrix and perform the corresponding computations once and simply refer those duplicated sub-matrices and the necessary intermediate numerical results to the representative sub-matrix.



(a) (b) (c) (d)

Figure 4.1: The coefficient matrices A_{HS} and their elimination trees with "perfect" compressions (a,b) and with compression containing defects (c,d).

We use two examples to illustrate the effects of the ETD. (i) We assume an ideal case in which the physical parameters and the grid properties of the subdomain and separator sub-matrices of the same levels are identical. In this case, there are only one subdomain type and one separator type in each level. Without loss of generality, we assume that the

representative subdomain is D_1 and that the representative separators are $S_{1,1}$, $S_{2,1}$, $S_{3,1}$, and $S_{4,1}$. The de-duplicated matrix and the corresponding elimination tree are shown in Figures 4.1a and 4.1b, respectively. (ii) We assume that a defect is located in subdomain D_7. According to the Schur complement update procedure, D_7 alters its updating contents for $S_{1,4}$, $S_{2,2}$, $S_{3,1}$, and $S_{4,1}$. See Figures 4.1c and 4.1d for an illustration of the second example. The different components of these Schur complements must be stored, and corresponding operations must be computed.

In short, structure homogeneity and periodicity are common in photonic device simulations. We can take advantage of this particular property to de-duplicate the elimination tree thanks to the Yee's discretization and the proposed grid alignment and ordering discussed in Section 3.1. Our ETD scheme explores such structures directly and efficiently without performing matrix structure analysis. This is a major advantage of the ETD scheme. For example, the identification and de-duplication takes less than one second to complete in all of the numerical tests in this paper. Furthermore, because the defects in many photonic structures and imperfect periodicity are limited, the ETD remains beneficial to many other photonic device simulations.

4.2. Compression of leaf-level interface sub-matrices

The matrix storage and computations can be further compressed by the leaf-level interface sub-matrix compression (LIC), as shown below. To achieve this goal, we first examine the structure of the interface sub-matrices. As illustrated in Figure 4.2a, each subdomain interacts with separators in the following 12 directions: $\pm x$, $\pm y$, $\pm z$, $\pm x \mp y$, $\pm y \mp z$, and $\pm x \mp z$. A separator adjoins a subdomain via face components (FCs) or edge components (ECs). We take subdomain D_{13} and separator $S_{4,1}$ as an example to illustrate these notations. As shown in Figure 4.2b, $S_{4,1}$ contains EC_{-y+z} and FC_{-y} of D_{13}. The corresponding D_{13}-$S_{4,1}$ related interface sub-matrices are denoted as $I_U(D_{13}, S_{4,1})$ and $I_L(D_{13}, S_{4,1})$, which are indicated in the top-right and bottom-left parts of Figure 4.2c, respectively. Moreover, $I_U(D_{13}, S_{4,1})$ is assembled by the sub-matrices $\mathcal{U}_{D_{13},\mathrm{EC}_{-y+z}}$ and $\mathcal{U}_{D_{13},\mathrm{FC}_{-y}}$ with a particular column arrangement based on the ordering of $S_{4,1}$. Similarly, $I_L(D_{13}, S_{4,1})$ is assembled by the sub-matrices $\mathcal{L}_{D_{13},\mathrm{EC}_{-y+z}}$ and $\mathcal{L}_{D_{13},\mathrm{FC}_{-y}}$ with a particular row arrangement based on the ordering of $S_{4,1}$. Next, we illustrate the concept of compressing the interface sub-matrices by considering D_8, D_{12}, D_{13}, and their corresponding separators. As shown in Figure 4.2d, D_8, D_{12}, and D_{13} contain the face components FC_{-y}s that are also parts of the separators $S_{2,2}$, $S_{2,3}$, and $S_{4,1}$, respectively.

The key concept of the LIC scheme is to recognize that $\mathcal{U}_{D_1,\mathrm{FC}_{-y}}$, $\mathcal{U}_{D_8,\mathrm{FC}_{-y}}$, $\mathcal{U}_{D_{12},\mathrm{FC}_{-y}}$, and $\mathcal{U}_{D_{13},\mathrm{FC}_{-y}}$ are equivalent up to suitable column permutations when the grid sizes are uniform. Moreover, D_8, D_{12}, and D_{13} are identical to D_1, and only D_1 is stored, as shown

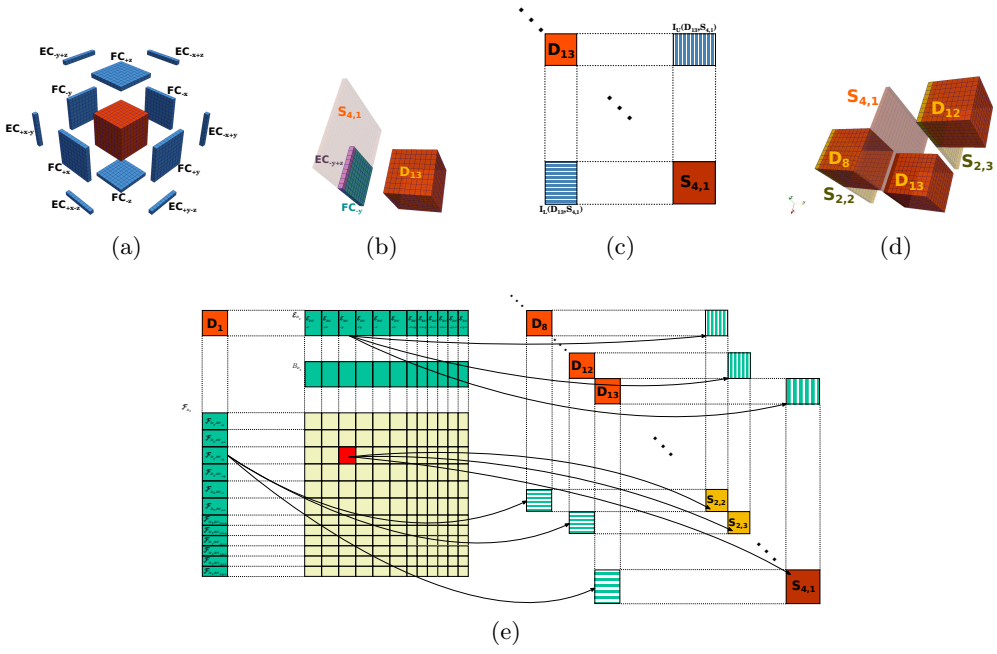


Figure 4.2: Conceptual schema of leaf-level interface sub-matrix compression. (a) Separator components adjacent to a subdomain. (b) Subdomain D_{13} and its adjacent components in $S_{4,1}$. (c) Sub-matrices related to D_{13} and $S_{4,1}$. (d) D_8, D_{12}, and D_{13} and their FC$_{-y}$s. (e) Updating multiple sub-matrices with a single restructured matrix from FC$_{-y}$.

in Figure 4.1b. Consequently, there is no need to compute $D_8^{-1}\mathcal{U}_{D_8,\mathrm{FC}_{-y}}$, $D_{12}^{-1}\mathcal{U}_{D_{12},\mathrm{FC}_{-y}}$, and $D_{13}^{-1}\mathcal{U}_{D_{13},\mathrm{FC}_{-y}}$ because the results can be extracted from $D_1^{-1}\mathcal{U}_{D_1,\mathrm{FC}_{-y}}$. Similar techniques can be applied to the sub-matrices associated with $\mathcal{L}_{D_8,\mathrm{FC}_{-y}}$, $\mathcal{L}_{D_{12},\mathrm{FC}_{-y}}$, and $\mathcal{L}_{D_{13},\mathrm{FC}_{-y}}$ because $\mathcal{L}_{D_1,\mathrm{FC}_{-y}}$, $\mathcal{L}_{D_8,\mathrm{FC}_{-y}}$, $\mathcal{L}_{D_{12},\mathrm{FC}_{-y}}$, and $\mathcal{L}_{D_{13},\mathrm{FC}_{-y}}$ are equivalent up to suitable row permutations. This compression scheme is conceptually illustrated in Figure 4.2e.

Note that although there is no FC$_{-y}$ for subdomain D_1 in the original example, we can construct a virtual FC$_{-y}$ based on Yee's mesh. Furthermore, under ideal periodicity, we can simply use interfaces between D_1 and the twelve possible FCs to describe all interface sub-matrices between subdomains and any other separators, as shown in Figures 4.1a and 4.1b.

We conclude this section by emphasizing that with a higher compression rate in the subdomains, the interface component arrangement greatly reduces the number of right-hand-side vectors in the sparse triangular solves. Consequently, LIC can significantly save overall computation time in the leaf-level computations.

4.3. The CHiS algorithm

We modify Algorithm 3.1 (HiS) to include the compression schemes ETD and LIC. The overall procedure for CHiS to factorize the hierarchical Schur coefficient matrix A_{HS} defined in (3.2) is summarized in Algorithm 7.1. CHiS greatly reduces the computational workloads in ZGEMM, ZGETRF, and ZGETRS calls thanks to the co-design of physical properties, algorithm design, and efficient architectures.

As proposed in Section 4.1, we use the elimination tree de-duplication scheme to remove the identical sub-structures. Here, we elaborate the compression scheme with more details. For the triangular solves in the leaf-level, the ETD is first implemented to remove the duplicated sparse factorization of subdomains. As illustrated in Figure 4.2e (and lines 7 and 8 of Algorithm 7.1), the following step is performed to aggregate EC- and FC-related interface sub-matrices corresponding to each subdomain D_i into $\mathcal{E}_{D_i}$ (for upper blocks) and $\mathcal{F}_{D_i}$ (for lower blocks). The aggregation eliminates multiple tiny matrix computations to increase computational efficiency and provides preparation for the LIC scheme. The aggregated $\mathcal{E}_{D_i}$ can be handled as multiple-RHS (right-hand-side vectors) for the sparse linear system $B_{D_i} = D_i^{-1}\mathcal{E}_{D_i}$. After we obtain the solution B_{D_i}, sparse matrix multiplication of $\mathcal{F}_{D_i}$ and B_{D_i} is performed in line 12 of Algorithm 7.1, which is utilized for the first-level Schur complement update based on the LIC and elimination tree. When there are many duplicates of subdomain D_i, from the LIC scheme, we can apply multiple updates at different sub-matrices in the Schur complement with the single computation of S_{D_i}, as illustrated in Figure 4.2e.

Other details of the algorithm are highlighted below.

- The coefficient matrix A_{HS} is defined implicitly based on ETD and LIC. The matrix entries in the de-duplicated sub-matrices are accessed by remapping on-the-fly.

- The matrix blocks of non-duplicate subdomains in D and their related interface blocks in off-diagonal positions are assembled in sparse form. In the hierarchical Schur complement $A_{\mathrm{HS}}^{\mathrm{sc}}$, we store the sub-matrix corresponding to each separator in dense format. Mutual interface sub-matrices between separators are also packed in a dense format.

- Information of non-zero columns and rows are recorded. The non-zero column and row information are referenced when updating next-level Schur complement sub-matrices. We use OpenMP to accelerate the permutation of the updating operation. This technique maintains the performance benefits of BLAS3 operations with higher arithmetic intensity in the CHiS without significantly increasing the computational workloads.

- In the hierarchical Schur complement $A_{\mathrm{HS}}^{\mathrm{sc}}$, the diagonal block of each non-duplicate separator is factorized by dense LU decomposition. When performing updates to the successive-level Schur complement, upper interface sub-matrix duplicates are searched after each multiple-RHS solving of $B_{S_i,j_d} = S_{i,j}^{-1} I_{U_c}$. If an upper duplicate is found, the update process is performed by reusing the solving results B_{S_i,j_d}. This ensures that the multiple-RHS solves of $B_{S_i,j_d} = S_{i,j}^{-1} I_{U_c}$ are not repetitive and that computation of de-duplication is achieved. Moreover, the workload of matrix multiplication can be reduced if the update target is de-duplicated.

5. Numerical experiments

We implement the HiS (see Algorithm 3.1) and CHiS (Algorithm 7.1) by using the C programming language and the BLAS, LAPACK, and sparse linear algebra libraries included in the Intel Math Kernel Library of the Parallel Studio XE 2016 Update 1 Suite. We use the sparse direct solver PARDISO [23] to factorize the sparse subdomain sub-matrices D_* and to solve the corresponding linear systems with multiple RHS vectors. The compiler is the Intel C compiler with the flags of "-O2-openmp". For comparison, the PARDISO released in the Intel MKL 11.3 Update 1 is used to solve the entire linear system (3.1). We perform all of the numerical experiments on a workstation equipped with dual Intel E5-2670 v3 CPUs and 256 GB of main memory. Each CPU has 12 threads.

We use the HiS and CHiS to solve the linear systems arising in the simulations of photonic structures. In Section 5.1, we introduce the dielectric waveguides and wavelength filters to be simulated and show that our simulation results are consistent with the results reported in the literature. In Section 5.2, we present the performances of HiS and CHiS in terms of memory usage, timing, multithreading parallelism, and arithmetic intensity. In Sections 5.3 and 5.4, we illustrate how the proposed algorithms can take advantage of the homogeneous and periodic structures of the photonic structures, respectively. The features shown in Sections 5.3 and 5.4 can be highly beneficial to simulations of massively homogeneous or periodic photonic structures.

5.1. Test problems for benchmark and simulation

Two sets of test problems are considered here. First, as a set of benchmark problems, we focus on the linear systems arising from various settings of straight dielectric waveguides. Figure 5.1 illustrates the schema of the dielectric waveguide. The refraction indices of the waveguide core, substrate, and upper cladding are $\sqrt{11}$, 1.5, and 1.0, respectively. The height and width of the waveguide core are 0.5 and 0.22 μm, respectively. The wavelength of this simulation case is 1.5 μm. The figure also shows the computed major

field component $\Re[E_x]$. In the computational domain, the uniform grid size $\Delta x = \Delta y = \Delta z = 0.02$ μm is used in the non-PML regions. We use a PML thickness of 8 grids on each side to absorb outward-propagating waves in all computational domain boundaries. In particular, we consider the following settings of grid numbers corresponding to the dielectric waveguides with different lengths.

(W1) Dielectric waveguide with $(N_x, N_y, N_z) = (79, 319, 39)$. The waveguide length in the y-axis is 6.4 μm, and and the resulting matrix dimension of A_{HS} is $2,948,517$.

(W2) Dielectric waveguide with $(N_x, N_y, N_z) = (79, 639, 39)$. We extend the computational domain along the y-direction by a factor of 2. The waveguide length in the y-axis becomes 12.8 μm, and the dimension of A_{HS} is $5,906,277$.

(W3) Dielectric waveguide with $(N_x, N_y, N_z) = (79, 1279, 39)$. We consider an even longer waveguide. The waveguide length in the y-axis is 25.6 μm, and the dimension of A_{HS} is $11,821,797$.

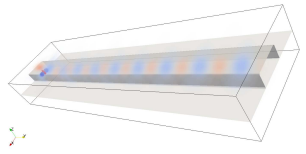


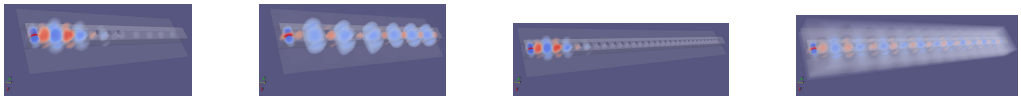
Figure 5.1: Schematics and $\Re[E_x]$ field distribution in modeled dielectric straight waveguide. Source current is a point dipole in waveguide core.

Second, as a set of simulation problems, we model a wavelength filter based on a one-dimensional periodic airhole array on a CdTe ridge waveguide [5] and solve the problems by CHiS. The schema of such a ridge waveguide is illustrated in Figure 5.2. The CdTe waveguide is 0.55 μm in width and 0.35 μm in height. The airhole radii are 0.1 μm, and the lattice of periodic airholes is 0.45 μm. The refraction indices of CdTe, SiO_2 substrate, and upper air cladding are 2.74, 1.45, and 1.0, respectively. Discretization of the non-PML domain is $(\Delta x = \Delta y = \Delta z = 0.025)$ μm. The grid number of each subdomain is $19 \times 17 \times 11$. We assume that two dipoles with wavelengths $\lambda_0 = 1500$ and 1800 nm are launched in the simulated periodic airhole structures to demonstrate frequency filtering of the device. Two examples of such periodic airhole structures are simulated, and the numerical findings are briefly summarized below to demonstrate that our simulation results are consistent with those in the literature.

(F1) A shorter filter device with 5 air holes and $(N_x, N_y, N_z) = (79, 287, 47)$. The resulting matrix dimension of A_{HS} is $3,196,893$. The computed field distributions (i.e., E_x) associated with wavelengths $\lambda_0 = 1500$ and 1800 nm are shown in Figures 5.2a

and 5.2b, respectively. Figure 5.2a suggests that such a structure provides limited filtering, and wave leakage through the hole array is visible. Conversely, Figure 5.2b suggests that the guided wave with $\lambda_0 = 1800$ nm successfully propagates through the hole array, as shown in Figure 5.2b. The results are consistent with the band analysis reported in [5]. As suggested by [5], the wavelength $\lambda_0 = 1500$ nm is in a band gap of the periodic structure, and the guided waves cannot propagate through the airhole array.

(F2) A longer filter device with 28 air holes and $(N_x, N_y, N_z) = (79, 575, 47)$. The resulting matrix dimension of A_{HS} is $6,404,925$. Figure 5.2c suggests that there is no visible guided wave at the end of the hole arrays, and the filtering effect is very effective. Meanwhile, wave propagation with $\lambda_0 = 1800$ nm is also correctly simulated.



(a) (b) (c) (d)

Figure 5.2: Illustration and calculated $\Re[E_x]$ field distribution of periodic airhole array on CdTe ridge waveguides. Parts (a) and (b) correspond to a short waveguide with 5 air holes and wavelengths $\lambda_0 = 1500$ and 1800 nm, respectively. Parts (c) and (d) correspond to a doubled-length waveguide with 28 air holes and wavelengths $\lambda_0 = 1500$ and 1800 nm, respectively.

In short, these observations imply that the proposed algorithm can be a useful simulation tool for partially periodic structure designs.

5.2. Algorithmic performance evaluations

We solve the linear system (W1), i.e., simulation of a dielectric waveguide with $(N_x, N_y, N_z) = (79, 319, 39)$, to evaluate the performances of HiS and CHiS. In HiS and CHiS, the size of the subdomain sub-matrices is one tunable parameter that significantly affects the performances of HiS and CHiS. In particular, we consider the settings in which each subdomain contains $4 \times 4 \times 4$, $9 \times 9 \times 9$, and $19 \times 19 \times 19$ grids, and the corresponding elimination trees include 13, 10, and 7 levels, respectively. The main reasons for why the subdomain size affects the computational tasks are discussed below. With more grid points in each subdomain, the dimension of the corresponding sparse linear system and the number of right-hand-side vectors in the interface sub-matrices are increased. However, the hierarchy levels and corresponding dense matrix computations are reduced. Consequently,

we use different subdomain sizes (and thus different Schur complement hierarchy levels) for performance analysis regarding the trade-off between sparse and dense linear algebra. With different subdomain sizes, we discuss the performance evaluation results in terms of memory usage, timing, and multithreading parallelism.

Subdomain size	$4 \times 4 \times 4$		$9 \times 9 \times 9$		$19 \times 19 \times 19$	
Algorithm	HiS	CHiS	HiS	CHiS	HiS	CHiS
Memory in GB	172.2 (1)	103.7 (0.60)	171.5 (1)	99.9 (0.58)	141.5 (1)	90.9 (0.64)
No. of D_*	8192 (1)	205 (.025)	1024 (1)	54 (.053)	128 (1)	24 (.19)
No. of $S_{1,*}$	4096 (1)	130 (.032)	512 (1)	36 (.070)	64 (1)	12 (.19)
No. of $S_{2,*}$	2048 (1)	78 (.038)	256 (1)	36 (.14)	32 (1)	12 (.38)
No. of $S_{3,*}$	1024 (1)	54 (.053)	128 (1)	24 (.19)	16 (1)	6 (.38)
No. of $S_{4,*}$	512 (1)	36 (.070)	64 (1)	12 (.19)	8 (1)	6 (.75)
No. of $S_{5,*}$	256 (1)	36 (.14)	32 (1)	12 (.38)	4 (1)	3 (.75)
No. of $S_{6,*}$	128 (1)	24 (.19)	16 (1)	6 (.38)	2 (1)	2 (1)
No. of $S_{7,*}$	64 (1)	12 (.19)	8 (1)	6 (.75)	1 (1)	1 (1)
No. of $S_{8,*}$	32 (1)	12 (.38)	4 (1)	3 (.75)		
No. of $S_{9,*}$	16 (1)	6 (.38)	2 (1)	2 (1)		
No. of $S_{10,*}$	8 (1)	6 (.75)	1 (1)	1 (1)		
No. of $S_{11,*}$	4 (1)	3 (.75)				
No. of $S_{12,*}$	2 (1)	2 (1)				
No. of $S_{13,*}$	1 (1)	1 (1)				

Table 5.1: Memory usage compression for the dielectric straight waveguide simulations described in Section 5.2. The table shows memory usage in GB and the compression ratios (within the parentheses) for different settings. In addition, the number of sub-matrices and the corresponding compression ratios in each level of the elimination tree are listed. Note that PARDISO requests 126.4 GB of main memory to factorize the problem.

5.2.1. Memory usage

Table 5.1 presents an overall comparison of memory usage. The memory usage is counted as the highest memory usage size in the entire factorization procedure. In the three cases, the sizes of memory usage in the CHiS are approximately 60% of the HiS. Note that the full factorization for A_{HS} performed by PARDISO requires 126.4 GB of main memory.

Table 5.1 also lists the number of sub-matrices and the compression ratios in each level of the elimination tree. These results suggest that significantly high compression ratios occur in the lower levels of the elimination tree (e.g., D_*, $S_{1,*}$, and $S_{2,*}$), particularly

for the sub-matrices due to smaller subdomains (e.g., $4 \times 4 \times 4$). For example, in the $4 \times 4 \times 4$ case, the compression ratio of the subdomain sub-matrices D_* due to CHiS is $205/8192 \approx 0.025$.

5.2.2. Total time

Figure 5.3 presents the timing results of the main computational tasks performed by HiS and CHiS. We use two CPUs and all of the 24 threads to achieve multi-thread parallelism. To factor A_{HS} in (3.1) numerically, HiS takes 741, 678, and 918 seconds in total for the $4\times4\times4$, $9\times9\times9$, and $19\times19\times19$ cases, respectively. Conversely, CHiS takes approximately 500 seconds in all three subdomain settings. In other words, CHiS is 1.47X, 1.37X, and 1.83X faster than HiS in each of the cases. For comparison, PARDISO takes 898.3 seconds to perform the numerical factorization.

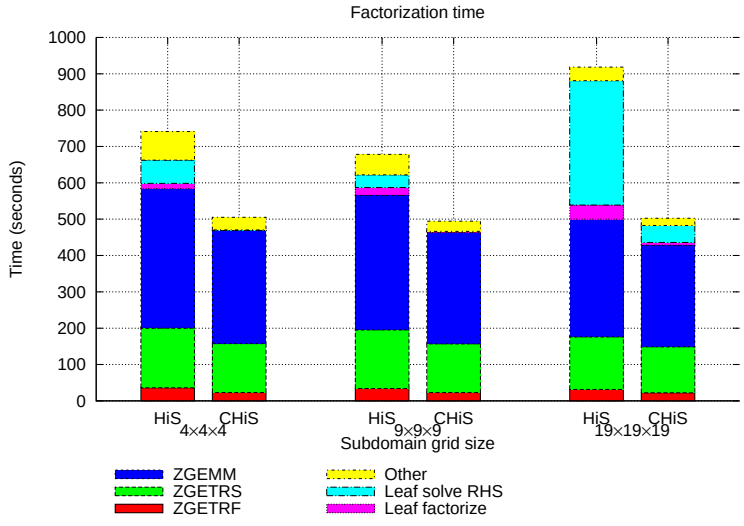


Figure 5.3: Total factorization time of HiS and CHiS with different subdomain grid sizes. In comparison, direct factorization of the same problem by PARDISO from Intel MKL is completed in 898.3 seconds.

Note that we count only numerical factorization timing here. Unlike graph-based reordering schemes, our CHiS implementation can efficiently construct the matrix elements and the corresponding elimination tree directly from the computational domain. Take Problem (W1) as an example; PARDISO calls METIS and takes 16.2 seconds to perform symbolic factorization of A_{PML} (2.4) by using 24 threads. In contrast, the construction of the matrix elimination dependency, all memory allocation, and restructuring can be completed in less than 4.7 seconds by CHiS.

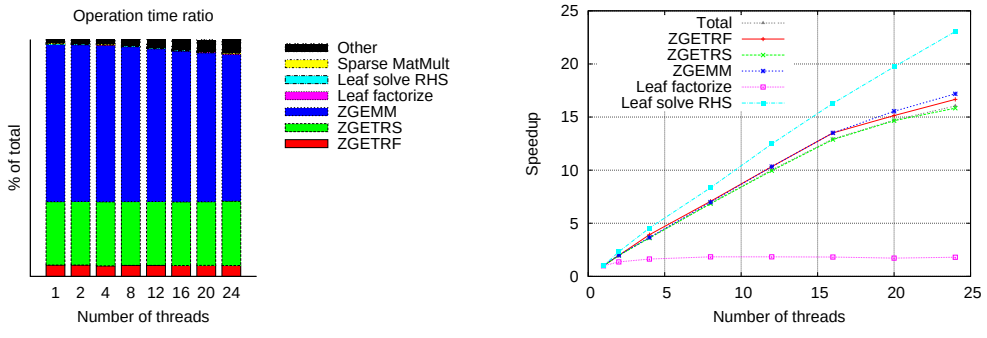
CHiS is faster mainly because of (i) elimination tree de-duplication of subdomain and

separators (Section 4.1) and (ii) leaf-level interface sub-matrices compression (Section 4.2). At the leaf-level, approximately 98%, 95%, and 81% of subdomain sub-matrices D_* are repeated and thus removed from the computations in CHiS, as shown in Table 5.1. Therefore, CHiS achieves significant savings in the leaf-level factorizations and multiple RHS solves.

For lower level separators, e.g., $S_{1,*}$ and $S_{2,*}$, compressions are more significant, and CHiS can save many computational tasks. In contrast, in the higher level separators (e.g., $S_{10,*}$ to $S_{13,*}$ in the case of the $4 \times 4 \times 4$ subdomain), little or none of the sub-matrices are identical and de-duplicated. In this case, there is little difference between the HiS and CHiS timing results.

5.2.3. Multi-thread parallelism of dense matrix operations

The efficiency of the dense BLAS3 operation can be improved by increasing the arithmetic intensity (AI) with larger matrices. A higher AI ensures that the overall computation is computing-bound, which indicates that these operations are not bottlenecked by bandwidth and that computational efficiency is likely maintained in future computer hardware. Analysis of AI and computational performance will be discussed below.



(a) (b)

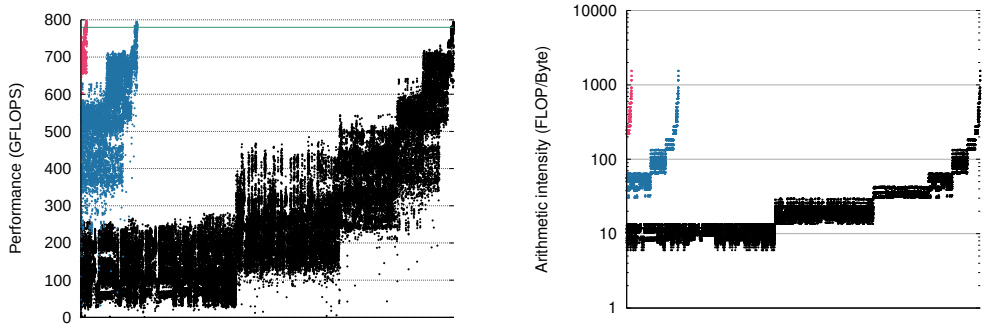
Figure 5.4: (a) Workload ratio comparisons with respect to the number of threads and (b) strong scalability of the workloads in CHiS for solving a linear system with $9 \times 9 \times 9$ subdomains.

Figure 5.3 suggests that the dense operations ZGEMM, ZGETRS, and ZGETRF in HiS and CHiS require the majority of the computational time. These operations are associated with the larger separators and sub-matrices in higher levels. Fortunately, computations of these large dense matrix operations can be efficiently performed on a multicore CPU with SIMD arithmetic units. Even better, these operations can be accelerated by, for example, NVIDIA GPU and Intel Xeon Phi. Next, we investigate internal computational

performance of CHiS in terms of strong scalability of multi-threading and arithmetic intensity.

Figures. 5.4a and 5.4b present the breakdown cost analysis of CHiS in normalized timing and the results of strong scalability, respectively. In these figures, different numbers of threads are used to solve the linear system with $9 \times 9 \times 9$ subdomains. Figure 5.4a shows that ZGEMM and ZGETRS dominate the performance of CHiS, and these two operations are highly scalable for multi-threading, as shown in Figure 5.4b.

We further study the performance of ZGEMM and ZGETRS in terms of GFLOP per second (GFLOPS) by using 24 threads of the testing workstation. Figure 5.5 shows the performance of all ZGEMM and ZGETRS calls in CHiS by using 24 threads of the testing workstation, and the arithmetic intensities of ZGEMM and ZGETRS are also shown. The FLOP of ZGEMM $C = C + A_{(M\times K)}B_{(K\times N)}$ is counted as $8MKN$. For ZGETRS, the FLOP count for solving $A_{(M\times M)}X_{(M\times N)} = B_{(M\times N)}$ is $8M^2N$. The figures suggest that the GFLOPS performance of ZGEMM is highly correlated to the subdomain size. Larger sub-matrices corresponding to larger subdomains yield better GFLOPS performance. For example, the sizes of the sub-matrices corresponding to the $4 \times 4 \times 4$ subdomain are typically too small to achieve high GFLOPS performance. In contrast, the sub-matrices corresponding to the $19 \times 19 \times 19$ subdomain are larger, and nearly all dense matrix operations achieve more than 650 GFLOPS, which is approximately 83% of the peak performance from Intel Optimized LINPACK LU factorization benchmark (780 GFLOPS).



(a) (b)

Figure 5.5: (a) GFLOPS performance of ZGEMM and ZGETRS in CHiS for solving the linear systems with $4\times4\times4$, $9\times9\times9$, and $19\times19\times19$ subdomains. The best Intel Optimized LINPACK benchmark result (LU factorization) of the testing workstation is approximately 780 GFLOPS. (b) Arithmetic intensity statistics of the ZGEMM with $4 \times 4 \times 4$, $9 \times 9 \times 9$, and $19 \times 19 \times 19$ subdomains. Each dot indicates a dense linear algebra function call. The black, blue, and red dots are associated with the $4 \times 4 \times 4$, $9 \times 9 \times 9$, and $19 \times 19 \times 19$ subdomains, respectively.

In short, the aforementioned results suggest that CHiS can be easily accelerated by modern hardwares such as NVIDIA GPU or Intel Xeon Phi because these hardwares have higher GFLOPS performance for dense matrix computations.

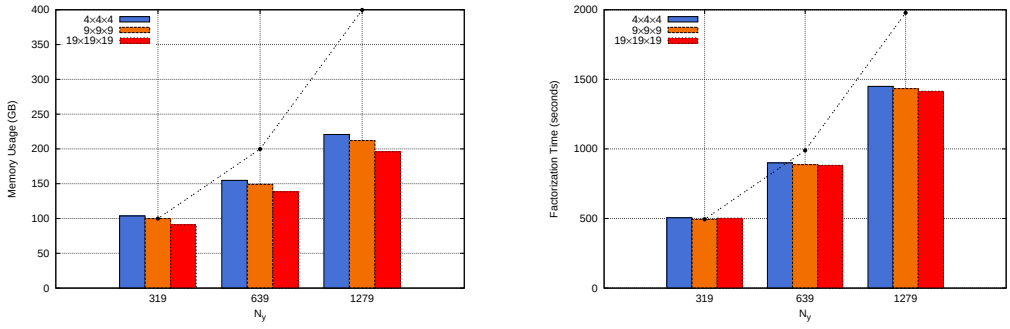
5.2.4. A short summary

These numerical experiments suggest that the overall timing performance of CHiS is an interplay of (i) the size of the subdomain and thus the size of the separator and interface sub-matrices and the number of elimination tree levels, (ii) the performance of sparse linear system solvers for multiple right-hand sides in the leaf level, and (iii) the performance of the dense operations such as ZGEMM, ZGETRS, and ZGETRF in the separator levels of the elimination tree. Furthermore, when subdomains are highly compressed, the interface component arrangement greatly reduces the number of right-hand-side vectors in the sparse linear system solvers and the overall computation time.

5.3. Memory and time savings due to homogeneous structures

We analyze the performance of CHiS for solving Problems (W1), (W2), and (W3) defined in Section 5.1 to demonstrate how CHiS can achieve memory and time savings due to homogeneous structures. In these three problems, the domains along the y-direction are doubled such that $N_y = 319$, 639, and $1,279$ in (W1), (W2), and (W3), respectively. If no compression scheme is used, we expect that the memory usage and the execution time will increase by a factor of larger than 2 because of the double-sized elimination tree and a new root node in the highest level of the elimination tree. However, Figure 5.6 shows that CHiS uses less memory and time than we expect. This performance improvement is primarily due to the elimination tree de-duplication (see Section 4.1) of Schur complement sub-matrices related to several large separators. We illustrate the effect of elimination tree de-duplication through the following example.

For the cases with $4 \times 4 \times 4$, $9 \times 9 \times 9$, or $19 \times 19 \times 19$ subdomains, the largest separator is the y-normal cross-section with dimensions $3N_xN_z = 9,243$. The most time-consuming computations are related to these largest separators. If no compression scheme is used, then the number of such separators is expected to be doubled. However, CHiS finds only 6, 9, and 12 non-identical largest separators in (W1), (W2), and (W3). These separator number counts show that the workloads do not increase proportionally with respect to the problem sizes. This improved scaling is primarily due to the compression schemes proposed in Section 4 in which the homogeneous physical structure of the waveguide among y-direction is the critical factor.



(a) (b)

Figure 5.6: (a) Memory usage and (b) timing results for solving Problems (W1), (W2), and (W3) defined in Section 5.1 by CHiS with different subdomain sizes. The black dashed lines are reference of memory usage and timing, and they are proportional to N_y in (W1), (W2), and (W3).

5.4. Performance gains due to periodicity

Periodicities are common in photonic structures. We consider Problems (F1) and (F2) introduced in Section 5.1 to show how CHiS can achieve memory and time savings in periodic structures. By using $19 \times 17 \times 11$ subdomains, CHiS factorizes the coefficient matrix arising in the shorter filter device with 5 periodic air-holes (F1) by using approximately 780 seconds and 145.8 GB of main memory. For the longer filter with 28 periodic air-holes (F2), CHiS consumes $1,299$ seconds and 187.3 GB of main memory. That is, for the double-sized problem with more periodicity, CHiS takes only 1.28X in memory usage and 1.67X in timing.

We examine the reasons responsible for this performance improvement below. From the non-duplicate elimination tree statistics, (F1) and (F2) have 4 and 3 non-duplicated large separators $S_{6,*}$. In other words, the (F2) can be processed with significantly higher compression in the $S_{6,*}$ level due to more periodicities. This fact contributes to less than 2X scale-up of memory and time consumption. The compression effect of large separators in these two wavelength filter examples indicates significant scaling advantages of CHiS for highly periodic devices.

6. Conclusion

We have proposed and implemented the compressed hierarchical Schur algorithm as a framework for efficient FDFD simulations of photonic structures by presenting the following items. (i) We discretize the governing Maxwell equations by Yee's scheme and then

partition the computational domain and reorder the grid points by using physical and geometrical information. (ii) The resulting coefficient matrix has a particular hierarchical Schur structure that can be efficiently factorized using multicore-processor-accelerated BLAS3 operations. (iii) The computational performance is further improved by removing redundant matrix storages and factorization workloads due to the homogeneity and periodicity of the photonic structures. (iv) The construction of the hierarchical Schur coefficient matrix and recolonization of the duplicate sub-matrices are achieved in a very short time (less than one second in our experience) by exploring the physical and geometrical structures without going through graphical and symbolic analysis. (v) Numerical experiments regarding straight waveguides and periodic air-holes show that CHiS is a fast solver for FDFD. CHiS can significantly reduce the memory usages and redundant computational workloads. Overall, CHiS factorizes the coefficient quickly with satisfactory linear scale-up in multicore parallelism.

The proposed CHiS can act as a kernel to enable many other simulations of various photonic devices. For example, CHiS can be applied to solve eigenvalue problems for photonic band analysis [7]. That is, if the shift-and-invert technique is applied to solve the eigenvalue problems, then the proposed linear system solver can be used to solve the embedded linear systems therein.

Several future works can enhance the performance of CHiS. Although we only implement a CPU-only CHiS algorithm in this paper, performance analysis of matrix operations shows great acceleration potentials for modern HPC hardwares such as GPU-accelerated parallel computers. To remain efficient for even larger scale simulations, CHiS must be modified to deal with large root-level sub-matrices and take advantage of the multiple computational nodes. Parameter tuning that balances the effects among subdomain size, memory use, and computational workload will be a subject for improving the performance. A satisfactory tuning result is determined by considering the subdomain size, physical homogeneous and periodic properties, computer hardware performance, and numerical library features.

7. Appendix

Part 1. Discretization of (2.2):

$$
\begin{aligned}
f_{\mathrm{src},x} = {} & \frac{E_x^{i,j,k-1}}{(\Delta z)^2} + \frac{E_x^{i,j-1,k}}{(\Delta y)^2} + \frac{E_x^{i,j+1,k}}{(\Delta y)^2} + \frac{E_x^{i,j,k+1}}{(\Delta z)^2} - \left(\frac{2}{(\Delta y)^2} + \frac{2}{(\Delta z)^2} - k_0^2 \varepsilon_r \right) E_x^{i,j,k} \\
& - \frac{1}{\Delta x \Delta y} \left(E_y^{i+1,j,k} - E_y^{i,j,k} - E_y^{i+1,j-1,k} + E_y^{i,j-1,k} \right) \\
& - \frac{1}{\Delta x \Delta z} \left(E_z^{i+1,j,k} - E_z^{i,j,k} - E_z^{i+1,j,k-1} + E_z^{i,j,k-1} \right),
\end{aligned} \tag{7.1}
$$

$$
\begin{aligned}
f_{\mathrm{src},y} = {} & \frac{E_y^{i,j,k-1}}{(\Delta z)^2} + \frac{E_y^{i-1,j,k}}{(\Delta x)^2} + \frac{E_y^{i+1,j,k}}{(\Delta x)^2} + \frac{E_y^{i,j,k+1}}{(\Delta z)^2} - \left(\frac{2}{(\Delta x)^2} + \frac{2}{(\Delta z)^2} - k_0^2 \varepsilon_r \right) E_y^{i,j,k} \\
& - \frac{1}{\Delta x \Delta y} \left(E_x^{i,j+1,k} - E_x^{i,j,k} - E_x^{i-1,j+1,k} + E_x^{i-1,j,k} \right) \\
& - \frac{1}{\Delta y \Delta z} \left(E_z^{i,j+1,k} - E_z^{i,j,k} - E_z^{i,j+1,k-1} + E_z^{i,j,k-1} \right),
\end{aligned} \tag{7.2}
$$

$$
\begin{aligned}
f_{\mathrm{src},z} = {} & \frac{E_z^{i,j-1,k}}{(\Delta y)^2} + \frac{E_z^{i-1,j,k}}{(\Delta x)^2} + \frac{E_z^{i+1,j,k}}{(\Delta x)^2} + \frac{E_z^{i,j+1,k}}{(\Delta y)^2} - \left(\frac{2}{(\Delta x)^2} + \frac{2}{(\Delta y)^2} - k_0^2 \varepsilon_r \right) E_z^{i,j,k} \\
& - \frac{1}{\Delta x \Delta z} \left(E_x^{i,j,k+1} - E_x^{i,j,k} - E_x^{i-1,j,k+1} + E_x^{i-1,j,k} \right) \\
& - \frac{1}{\Delta y \Delta z} \left(E_y^{i,j,k+1} - E_y^{i,j,k} - E_y^{i,j-1,k+1} + E_y^{i,j-1,k} \right),
\end{aligned} \tag{7.3}
$$

where $E_x^{i,j,k}$, $E_y^{i,j,k}$, and $E_z^{i,j,k}$ are the unknown discretized electric fields along the x-, y-, and z-axes, respectively. The column vector x has $3N_xN_yN_z$ entries and

$$
\begin{aligned}
x = \Big[& E_x^{1,1,1}, E_y^{1,1,1}, E_z^{1,1,1}, E_x^{2,1,1}, E_y^{2,1,1}, E_z^{2,1,1}, \ldots, E_x^{N_x,1,1}, E_y^{N_x,1,1}, E_z^{N_x,1,1}, \\
& E_x^{1,2,1}, E_y^{1,2,1}, E_z^{1,2,1}, E_x^{2,2,1}, E_y^{2,2,1}, E_z^{2,2,1}, \ldots, E_x^{N_x,N_y,1}, E_y^{N_x,N_y,1}, E_z^{N_x,N_y,1}, \\
& E_x^{1,1,2}, E_y^{1,1,2}, E_z^{1,1,2}, \ldots, E_x^{N_x,N_y,N_z}, E_y^{N_x,N_y,N_z}, E_z^{N_x,N_y,N_z} \Big]^T.
\end{aligned} \tag{7.4}
$$

Part 2. The CHiS Algorithm:

Algorithm 7.1 CHiS: Compressed Hierarchical Schur Factorization (Part 1)

1: Define physical parameters, $\vec{f}_{\mathrm{src}}$, subdomain grid number (p_x, p_y, p_z), and directional hierarchy levels (l_x, l_y, l_z)
2: **for** $i = 1$ to $2^{l_x+l_y+l_z}$ **do**
3: **if** D_i is a non-duplicate subdomain **then**
4: ▷ *Sparse factorization of non-duplicate subdomains D_i as shown in Figure 4.1*
5: Factorize D_i
6: ▷ *$\mathcal{E}_{D_i}$ and $\mathcal{F}_{D_i}$ aggregation as shown in Figure 4.2e*
7: Collect all interfaces $\mathcal{E}_{D_i} = \begin{bmatrix} \mathcal{E}_{D_i,\mathrm{FC}_{-x}} & \mathcal{E}_{D_i,\mathrm{FC}_{+x}} & \cdots \end{bmatrix}$
8: Collect all interfaces $\mathcal{F}_{D_i} = \begin{bmatrix} \mathcal{F}_{D_i,\mathrm{FC}_{-x}}^T & \mathcal{F}_{D_i,\mathrm{FC}_{+x}}^T & \cdots \end{bmatrix}^T$
9: ▷ *Solve multiple-RHS sparse linear system with factorized D_i*
10: Solve $B_{D_i} = D_i^{-1} \mathcal{E}_{D_i}$
11: ▷ *Prepared for updating the first-level Schur complement as shown in (3.3)*
12: Perform sparse matrix multiplication $S_{D_i} = \mathcal{F}_{D_i} B_{D_i}$
13: **for all** D_i-duplicated subdomains D_j **do**
14: Update all non-duplicate separators and their mutual interfaces related to D_j using S_{D_i}

Algorithm 7.1 CHiS: Compressed Hierarchical Schur Factorization (Part 2)

for $i = 1$ to $l_x + l_y + l_z$ **do**
  for $j = 1$ to $j = 2^{l_x+l_y+l_z-i}$ **do**
    if $S_{i,j}$ is a non-duplicate separator **then**
      ▷ *Factorize of non-duplicate $S_{i,j}$ as shown in Figure 4.1*
      Factorize $S_{i,j}$
      for $u_i = i + 1$ to $l_x + l_y + l_z$ **do**
        for all $S_{i,j}$-duplicated separators S_{i,j_d} with parent $S_{u_i,\lceil j_d/2^{u_i-i}\rceil}$ **do**
          Identify current $I_{U_c} = I_U(S_{i,j_d}, S_{u_i,\lceil j_d/2^{u_i-i}\rceil})$
          if $S_{u_i,\lceil j_d/2^{u_i-i}\rceil}$ is non-duplicate **then**
            ▷ *Solve multiple-RHS linear system with factorized $S_{i,j}$*
            Solve $B_{S_{i,j_d}} = S_{i,j}^{-1} I_{U_c}$
            for all subsequent $S_{i,j}$-duplicated separators $S_{i,j_{d'}}$ **do**
              ▷ *Find other interface sub-matrices identical to I_{U_c} for updating the Schur complement in different sub-matrices. Avoid repetitive solving of multiple-RHS.*
              if $j_{d'} \bmod 2^{u_i-i} \equiv j_d \bmod 2^{u_i-i}$ and $S_{u_i,\lceil j_{d'}/2^{u_i-i}\rceil} = S_{u_i,\lceil j_d/2^{u_i-i}\rceil}$ **then**
                for $l_i = i + 1$ to $l_x + l_y + l_z$ **do**
                  if $u_i < l_i$ and $S_{l_i,\lceil j_{d'}/2^{l_i-i}\rceil}$ is not duplicate **then**
                    ▷ *Update target is a lower interface sub-matrix without de-duplication in successive-level Schur complement.*
                    Identify current $I_{L_c} = I_L(S_{i,j_{d'}}, S_{l_i,\lceil j_{d'}/2^{l_i-i}\rceil})$
                    Perform dense matrix multiplication $I_{L_c} B_{S_{i,j}}$
                    Update $I_L(u_i, \lceil j_{d'}/2^{u_i-i}\rceil)$ with $I_{L_c} B_{S_{i,j}}$
                  else if $u_i > l_i$ and $S_{u_i,\lceil j_{d'}/2^{u_i-i}\rceil}$ is not duplicate **then**
                    ▷ *Update target is a upper interface sub-matrix without de-duplication in successive-level Schur complement.*
                    if $S_{l_i,\lceil j_d/2^{l_i-i}\rceil}$ is not duplicate **then**
                      Identify current $I_{L_c} = I_L(S_{i,j_{d'}}, S_{l_i,\lceil j_{d'}/2^{l_i-i}\rceil})$
                    else
                      ▷ *The corresponding lower sub-matrix may be de-duplicated. If so, use the non-duplicate one to perform the Schur complement update.*
                      Look for non-duplicate separator $S_{l_i,j_{\text{nd}}}$ of $S_{l_i,\lceil j_{d'}/2^{l_i-i}\rceil}$
                      Identify current $I_{L_c} = I_L(S_{i,2^{l_i-i}j_{\text{nd}}-(j_{d'} \bmod 2^{l_i-i})}, S_{l_i,j_{\text{nd}}})$
                    Perform dense matrix multiplication $I_{L_c} B_{S_{i,j}}$
                    Update $I_U(l_i, \lceil j_{d'}/2^{l_i-i}\rceil)$ with $I_{L_c} B_{S_{i,j}}$
                  else if $u_i = l_i$ and $S_{u_i,\lceil j_{d'}/2^{u_i-i}\rceil}$ is not duplicate **then**
                    ▷ *Update target is a diagonal sub-matrix without de-duplication in successive-level Schur complement.*
                    Identify current $I_{L_c} = I_L(S_{i,j_{d'}}, S_{l_i,\lceil j_{d'}/2^{l_i-i}\rceil})$
                    Perform dense matrix multiplication $I_{L_c} B_{S_{i,j}}$
                    Update $S_{u_i,\lceil j_{d'}/2^{u_i-i}\rceil}$ with $I_{L_c} B_{S_{i,j}}$

References

[1] Y. Akahane, T. Asano, B.-S. Song and S. Noda, *Erratum: High-Q photonic nanocavity in a two-dimensional photonic crystal*, Nature **425** (2003), no. 6961, 944–947.

[2] W. Bogaerts, M. Fiers and P. Dumon, *Design challenges in silicon photonics*, IEEE J. Sel. Top. Quantum Electron. **20** (2014), no. 4, 1–8.

[3] J. H. Bramble and J. E. Pasciak, *Analysis of a Cartesian PML approximation to the three dimensional electromagnetic wave scattering problem*, Int. J. Numer. Anal. Model. **9** (2012), no. 3, 543–561.

[4] W. C. Chew, J. M. Jin and E. Michielssen, *Complex coordinate stretching as a generalized absorbing boundary condition*, Microw. Opt. Techn. Lett. **15** (1997), no. 6, 363–369.

[5] J. García, P. Sanchis, A. Martínez and J. Martí, *1D periodic structures for slow-wave induced non-linearity enhancement*, Opt. Express **16** (2008), no. 5, 3146–3160.

[6] P. Ghysels, X. S. Li, F.-H. Rouet, S. Williams and A. Napov, *An efficient multicore implementation of a novel HSS-structured multifrontal solver using randomized sampling*, SIAM J. Sci. Comput. **38** (2016), no. 5, S358–S384.

[7] T.-M. Huang, H.-E. Hsieh, W.-W. Lin and W. Wang, *Eigendecomposition of the discrete double-curl operator with application to fast eigensolver for three-dimensional photonic crystals*, SIAM J. Matrix Anal. Appl. **34** (2013), no. 2, 369-391.

[8] ———, *Eigenvalue solvers for three dimensional photonic crystals with face-centered cubic lattice*, J. Comput. Appl. Math. **272** (2014), 350–361.

[9] J. D. Joannopoulos, S. G. Johnson, J. N. Winn and R. D. Meade, *Photonic Crystals: Molding the Flow of Light*, Princeton University Press, Princeton, NJ, 2008.

[10] S. Johnson and J. Joannopoulos, *Block-iterative frequency-domain methods for Maxwell's equations in a planewave basis*, Opt. Express **8** (2001), no. 3, 173–190.

[11] S. G. Johnson, P. R. Villeneuve, S. Fan and J. D. Joannopoulos, *Linear waveguides in photonic-crystal slabs*, Phys. Rev. B **62** (2000), no. 12, 8212–8222.

[12] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM J. Sci. Comput. **20** (1999), no. 1, 359–392.

[13] Z. Li, Y. Saad, M. Sosonkina, *pARMS: A parallel version of the algebraic recursive multilevel solver*, Numer. Linear Algebra Appl. **10** (2003), no. 5-6, 485–509.

[14] K. Nakajima, *Parallel iterative solvers for ill-conditioned problems with heterogeneous material properties*, Procedia Comput. Sci. **80** (2016), 1635–1645.

[15] S. Operto, J. Virieux, P. Amestoy, J.-Y. L'Excellent, L. Giraud and H. Ben Hadj Ali, *3D finite-difference frequency-domain modeling of visco-acoustic wave propagation using a massively parallel direct solver: A feasibility study*, Geophysics **72** (2007), no. 5, SM195–SM211.

[16] F. Pellegrini and J. Roman, *Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs*, in: *Lecture Notes in Computer Science*, 493–498, High-Performance Computing and Networking, 1996.

[17] M. Qiu, M. Mulot, M. Swillo, S. Anand, B. Jaskorzynska, A. Karlsson, M. Kamp and A. Forchel, *Photonic crystal optical filter based on contra-directional waveguide coupling*, Appl. Phys. Lett. **83** (2003), no. 25, 5121–5123.

[18] R. C. Rumpf, *Simple implementation of arbitrarily shaped total-field/scattered-field regions in finite-difference frequency-domain*, Progr. Electromagn. Res. B **36** (2012), 221–248.

[19] Y. Saad and B. Suchomel, *ARMS: an algebraic recursive multilevel solver for general sparse linear systems*, Numer. Linear Algebra Appl. **9** (2002), no. 5, 359–378.

[20] P. Sao, X. Liu, R. Vuduc and X. Li, *A sparse direct solver for distributed memory Xeon Phi-accelerated systems*, IEEE Int. Parallel Distrib. Proc. Symp., (2015), 71–81.

[21] P. Sao, R. Vuduc and X. S. Li, *A distributed CPU-GPU sparse direct solver*, in: *Lecture Notes in Computer Science*, 487–498, Euro-Par 2014 Parallel Processing, 2014.

[22] O. Schenk, M. Christen and H. Burkhart, *Algorithmic performance studies on graphics processing units*, J. Parallel Distrib. Comput. **68** (2008), no. 10, 1360–1369.

[23] O. Schenk and K. Gärtner, *Solving unsymmetric sparse systems of linear equations with PARDISO*, Future Gener. Comput. Syst. **20** (2004), no. 3, 475–487.

[24] P. G. Schmitz and L. Ying, *A fast nested dissection solver for Cartesian 3D elliptic problems using hierarchical matrices*, J. Comput. Phys. **258** (2014), 227–245.

[25] S. Shi, C. Chen and D. W. Prather, *Plane-wave expansion method for calculating band structure of photonic crystal slabs with perfectly matched layers*, J. Opt. Soc. Amer. A **21** (2004), no. 9, 1769–1775.

[26] W. Shin and S. Fan, *Choice of the perfectly matched layer boundary condition for frequency-domain Maxwell's equations solvers*, J. Comput. Phys. **231** (2012), no. 8, 3406–3431.

[27] ———, *Accelerated solution of the frequency-domain Maxwell's equations by engineering the eigenvalue distribution of the operator*, Opt. Express **21** (2013), no. 19, 22578–22595.

[28] A. Taflove and S. C. Hagness, *Computational Electrodynamics: the Finite-difference Time-domain Method*, Artech House, Boston, MA, 2005.

[29] P. Tsuji and L. Ying, *A sweeping preconditioner for Yees finite difference approximation of time-harmonic Maxwell's equations*, Front. Math. China **7** (2012), no. 2, 347–363.

[30] Y. A. Vlasov, M. O'Boyle, H. F. Hamann and S. J. McNab, *Active control of slow light on a chip with photonic crystal waveguides*, Nature **438** (2005), no. 7064, 65–69.

[31] J. Vučković and Y. Yamamoto, *Photonic crystal microcavities for cavity quantum electrodynamics with a single quantum dot*, Appl. Phys. Lett. **82** (2003), no. 15, 2374–2376.

[32] K. Yee, *Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media*, IEEE Trans. Antenn. Propag. **14** (1966), no. 3, 302–307.

[33] C. D. Yu, W. Wang and D. Pierce, *A CPU-GPU hybrid approach for the unsymmetric multifrontal method*, Parallel Comput. **37** (2011), no. 12, 759–770.

Cheng-Han Du
Institute of Applied Mathematical Sciences, National Taiwan University, Taiwan
E-mail address: b92006@csie.ntu.edu.tw

Yih-Peng Chiou
Graduate Institute of Photonics and Optoelectrionics, Graduate Institute of Communication Engineering, and Department of Electrical Engineering, National Taiwan University, Taiwan
E-mail address: ypchiou@ntu.edu.tw

Weichung Wang
Institute of Applied Mathematical Sciences, National Taiwan University, Taiwan
E-mail address: wwang@ntu.edu.tw