CSIAM Trans. Appl. Math.
doi: 10.4208/csiam-am.2020-0206

Vol. **2**, No. 3, pp. 431-459
September 2021

Distributed-Memory $\mathcal{H}$-Matrix Algebra I: Data Distribution and Matrix-Vector Multiplication

Yingzhou Li[1,*], Jack Poulson[2] and Lexing Ying[3]

[1] *School of Mathematical Science, Fudan University, Shanghai, China.*
[2] *Hodge Star, Toronto, Canada.*
[3] *Department of Mathematics and ICME, Stanford University, Stanford, CA 94305, USA.*

Received 29 August 2020; Accepted 13 February 2021

Abstract. We introduce a data distribution scheme for $\mathcal{H}$-matrices and a distributed-memory algorithm for $\mathcal{H}$-matrix-vector multiplication. Our data distribution scheme avoids an expensive $\Omega(P^2)$ scheduling procedure used in previous work, where P is the number of processes, while data balancing is well-preserved. Based on the data distribution, our distributed-memory algorithm evenly distributes all computations among P processes and adopts a novel tree-communication algorithm to reduce the latency cost. The overall complexity of our algorithm is $\mathcal{O}\left(\frac{N\log N}{P}+\alpha\log P+\beta\log^2 P\right)$ for $\mathcal{H}$-matrices under weak admissibility condition, where N is the matrix size, α denotes the latency, and β denotes the inverse bandwidth. Numerically, our algorithm is applied to address both two- and three-dimensional problems of various sizes among various numbers of processes. On thousands of processes, good parallel efficiency is still observed.

AMS subject classifications: 65F99, 65Y05

Key words: Parallel fast algorithm, $\mathcal{H}$-matrix, distributed-memory, parallel computing.

1 Introduction

For linear elliptic partial differential equations, the blocks of both forward and backward operators, when restricted to non-overlapping domains, are numerically low-rank [7]. Hence both operators can be represented in a data sparse form. Many fast algorithms benefit from this low-rank property and apply these operators in quasi-linear scaling. Such fast algorithms include but not limit to tree-code [4, 41], fast multipole method

*Corresponding author. *Email addresses:* yingzhouli@fudan.edu.cn (Y. Li), jack@hodgestar.com (J. Poulson), lexing@stanford.edu (L. Ying)

(FMM) [3, 11, 18–20, 39, 42, 50], panel clustering method [21], etc. The low-rank structures in these fast algorithms are revealed via various interpolation techniques such as: pole expansion, Chebyshev interpolation, equivalent interaction, etc [15, 19, 39, 50].

In contrast to approximating the application of operators, another group of research focuses on approximating operators directly in compressed matrix forms. As one of the earliest members in this group, $\mathcal{H}$-matrix [5–7, 17, 21, 22, 26–28] hierarchically compresses operators restricted to far-range interactions by low-rank matrices. The memory cost and matrix-vector multiplication complexity are quasi-linear with respect to the degrees of freedom (DOFs) in the problem. Shortly after introducing $\mathcal{H}$-matrix, Hackbusch et al. [23] again introduced $\mathcal{H}^2$-matrix, which uses nested low-rank bases to further reduce the memory cost and multiplication complexity down to linear. Related to the fast algorithms above, $\mathcal{H}$-matrix and $\mathcal{H}^2$-matrix can be viewed as algebraic versions of tree code and FMM respectively. But they are more flexible in choosing different admissibility conditions and low-rank compression techniques, which are related to general advantages of algebraic representations.

Developments in the $\mathcal{H}$-matrix group and extensions beyond the group are explored in the past decade. Hierarchical off-diagonal low-rank matrix (HOLDER) [2] and hierarchical semi-separable matrix (HSS) [48] are two popular hierarchical matrices with the simplest admissibility condition, i.e., weak admissibility condition. Different from hierarchical matrices, recursive skeletonization factorization (RS) [37] and hierarchical interpolative factorization (HIF) [24, 25] introduce separators in the domain partition and compress the operator as products of sparse matrices. The partition and factorization in RS and HIF are in the similar spirit as that in multifrontal method [1, 12] and superLU method [30], while extra low-rank approximations are introduced to compress the interactions within frontals. Other algebraic representations include block low-rank approximation [49], block basis factorization [45], etc. The benefits of algebraic representations over analytical fast algorithms come in two folds: 1) numerical low-rank approximation is more effective than interpolation; 2) matrix factorization and inversion become feasible. We emphases that these algebraic representations are not only valid for linear elliptic operators, but also valid for operators associated with low-to-medium frequency Helmholtz equations and radial basis function kernel matrices. When operators admit high-frequency property, the low-rank structure appears in a very different way comparing to that in all aforementioned fast algorithms, and are also well-studied by the community [8, 9, 13, 14, 31–34, 38].

Many of these fast algorithms and algebraic representations have been parallelized on either shared-memory or distributed-memory setting to be applicable to practical problems of interest [10, 16, 18, 35, 40–44, 46, 47, 50]. Here we focus on the parallelization of $\mathcal{H}$-matrix. Kriemann [28, 29] implemented a shared-memory parallel $\mathcal{H}$-matrix using a block-wise distribution, i.e., each block is assigned to a single process. Processes assigned to blocks near root level are responsible for computations of complexity linear in N, where N is the total DOFs. Hence the speedup of such a parallelization scheme is theoretically upper bounded by $\mathcal{O}(\log N)$ and limited in practice up to 16 processes.

Izadi [26, 27] published detailed algorithms for $\mathcal{H}$-matrix addition, matrix-vector multiplication, matrix-matrix multiplication and matrix inversion under distributed-memory setting. In [26, 27], the data of $\mathcal{H}$-matrix are evenly distributed among all processes according to their global matrix indices, which is similar to our data distribution for one-dimensional problems with uniform discretization but different from ours for other setups. The computations in [26,27] are distributed under task-based parallelization, whose the scheduling part costs $\Omega(P^2)$ operations on P processes. According to numerical results therein, good parallel efficiency is limited up to 16 processes.

1.1 Contribution

In this paper, we first propose a balanced data distribution scheme for $\mathcal{H}$-matrices based on the underlying domain geometry[†]. In $\mathcal{H}$-matrix, the domain is usually hierarchically partitioned and then organized in a domain tree structure. In order to avoid any expensive scheduling procedure, our processes are also organized in a tree structure in correspondence to that of the hierarchical domain partition. Each process then owns a unique piece of the domain and also own the associated data in $\mathcal{H}$-matrix. Following such a data distribution, all data in $\mathcal{H}$-matrix are evenly distributed among all processes. For a $\mathcal{H}$-matrix of size N distributed on P processes, the memory cost is $\mathcal{O}\big(\frac{N\log N}{P}\big)$ on each process. Our data distribution scheme is scalable up to $P=\mathcal{O}(N)$ processes.

Building on top of our data distribution, a distributed-memory parallel algorithm is proposed to conduct the $\mathcal{H}$-matrix-vector multiplication. Our parallel algorithm consists of several parts: a computation part, three consecutive communication parts, and another computation part. When the input and output vectors are distributed according to the tree structure of processes, both computation parts are communication-free. Then a novel data communication scheme, known as the tree-communication, is introduced to significantly reduce costs in two of the communication parts. The remaining communication part consists of a constant number of point-to-point communication on each process. Mainly due to the process organization and the tree-communication scheme, the expensive scheduling procedure is totally avoided throughout our algorithm. The overall computational and communication complexities, then, are $\mathcal{O}\big(\frac{N\log N}{P}\big)$ and $\mathcal{O}\big(\alpha\log P+\beta\big(\log^2 P+\log\frac{N}{P}+(\frac{N}{P})^{\frac{d-1}{d}}\big)\big)$[‡] respectively, where d is the dimension of the problem, α denotes the message latency, and β denotes the inverse bandwidth.

Finally, the parallel algorithm is applied to two-dimensional and three-dimensional problems of sizes varying from a few thousands to a quarter billion on massive number of processes. The parallel scaling is still found to be near-ideal on computational resources available to us, up to a few thousands processes. In all cases, our $\mathcal{H}$-matrix-vector multiplications are completed within a few seconds.

[†]When the domain geometry of the problem is not available and only the graph connectivity of the problem is known, our data distribution scheme can be extended to use the hierarchical partition of the graph instead.
[‡]This is complexity for $\mathcal{H}$-matrices under standard admissibility conditions and an upper for $\mathcal{H}$-matrices under weak admissibility condition.

1.2 Organization

The rest of the paper is organized as follows. In Section 2, we revisit $\mathcal{H}$-matrix together with admissibility conditions. Section 3 introduces our balanced data distribution scheme. The distributed-memory $\mathcal{H}$-matrix-vector multiplication algorithm is detailed in Section 4. Section 5 presents numerical results for two-dimensional and three-dimensional problems of various sizes. Finally, we conclude the paper in Section 6 together with some discussion on future work.

2 Preliminary

In this section, we first review the definition and the structure of $\mathcal{H}$-matrix. Then the $\mathcal{H}$-matrix-vector multiplication follows in a straightforward way.

Let us assume that $\mathcal{K}(t,k)$ is a kernel satisfying the hierarchical low-rank property as in tree code or $\mathcal{H}$-matrix. Then applying Nyström discretization to the integral equation,

$$u(t) = \int_{\Omega} \mathcal{K}(t,s) f(s)\,\mathrm{d}s, \quad \text{for } t \in \Omega, \tag{2.1}$$

results a matrix-vector multiplication, and the matrix therein can be approximated by an $\mathcal{H}$-matrix. Throughout the rest paper, we use the concepts of a domain and the Nyström discretization points in the domain interchangeably. For example, a matrix restricted to $\Omega_1 \times \Omega_2$ means that the matrix restricted to the row and column indices corresponding to the discretization points in Ω_1 and Ω_2 respectively. In (2.1), the operator maps from the domain Ω to itself. In practice, $\mathcal{H}$-matrix can also be used to approximate operators mapping from one domain to another and the rest of the paper can be extended to such a setting with a minor update on domain notations. To simplify our presentation, we limit ourselves to the self mapping case.

In the above setting, the structure of the $\mathcal{H}$-matrix fundamentally relies on the hierarchical partition of the domain Ω, which is defined as follows:

Definition 2.1 (Domain tree)**.** *A tree $\mathbb{T}_\Omega = (\mathcal{V}_\Omega, \mathcal{E}_\Omega)$ with the vertex set $\mathcal{V}_\Omega$ and the edge set $\mathcal{E}_\Omega$ is called a domain tree of Ω if the following conditions hold:*

1. *All nodes in $\mathbb{T}_\Omega$ are subdomains of Ω;*
2. *The set of children of a domain $\omega \in \mathcal{V}_\Omega$, denoted as $\mathcal{C}(\omega) = \{\nu \in \mathcal{V}_\Omega \,|\, \exists (\omega,\nu) \in \mathcal{E}_\Omega\}$, is either empty or a partition of ω;*
3. *$\Omega \in \mathcal{V}_\Omega$ is the root of $\mathbb{T}_\Omega$.*

When a (quasi-)uniform discretization of a regular d-dimensional domain $\Omega = [0,1]^d$ is considered, the domain tree is constructed via applying a 2^d uniform partition recursively. Such domains are later referred as ideal d-dimensional domains. Fig. 1 illustrates

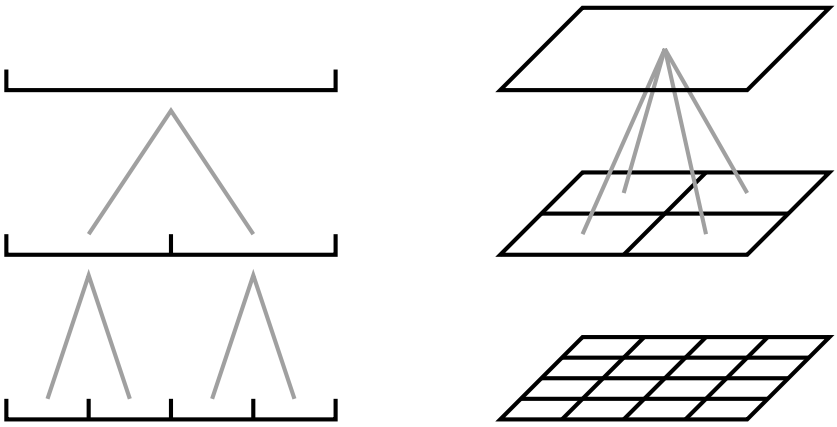


Figure 1: Hierarchical partition of ideal one-dimensional domain (left) and two-dimensional domain (right). Gray lines indicate connections between parent domains and their child subdomains.

two domain tree associated with an ideal one-dimensional domain and an ideal two-dimensional domain.

The low-rank submatrices in an $\mathcal{H}$-matrix are determined by admissibility conditions. There are many different admissibility conditions leading to different $\mathcal{H}$-matrix structures. Here we introduce two of them: weak admissibility condition and standard admissibility condition.

Definition 2.2 (Weak admissibility condition)**.** *Two domains, ω and ν, are weakly admissible if $\omega\cap\nu=\emptyset$.*

Definition 2.3 (Standard admissibility condition)**.** *Two domains, ω and ν, are standard admissible if*

$$\min\big(\mathrm{diam}(\omega),\mathrm{diam}(\nu)\big)\leq\rho\,\mathrm{dist}(\omega,\nu), \tag{2.2}$$

where $\mathrm{diam}(\omega)$ is the diameter of ω, $\mathrm{dist}(\omega,\nu)$ is the distance between two domains, and ρ is a constant adjusting the size of buffer zone.

Weak admissibility condition is the simplest admissibility condition used in practice and leads to the simplest $\mathcal{H}$-matrix structure. While standard admissibility condition is more complicated, but widely used in many fast algorithms [4,19]. Importantly, for linear elliptic differential operators with L_∞ coefficients discretized by a local basis set, both the forward differential operator and its inverse can be well-approximated by $\mathcal{H}$-matrix under standard admissibility condition. Throughout this paper, we adopt $\rho\equiv\sqrt{d}$. Fig. 2 and Fig. 3 shows the weak admissibility condition and the standard admissibility condition respectively for both ideal one-dimensional and two-dimensional domains. One more popular admissibility condition, known as strong admissibility condition [6, 37], simply replaces the "min" in (2.2) by "max". Strong admissibility condition and standard admissibility condition are the same on ideal domains.

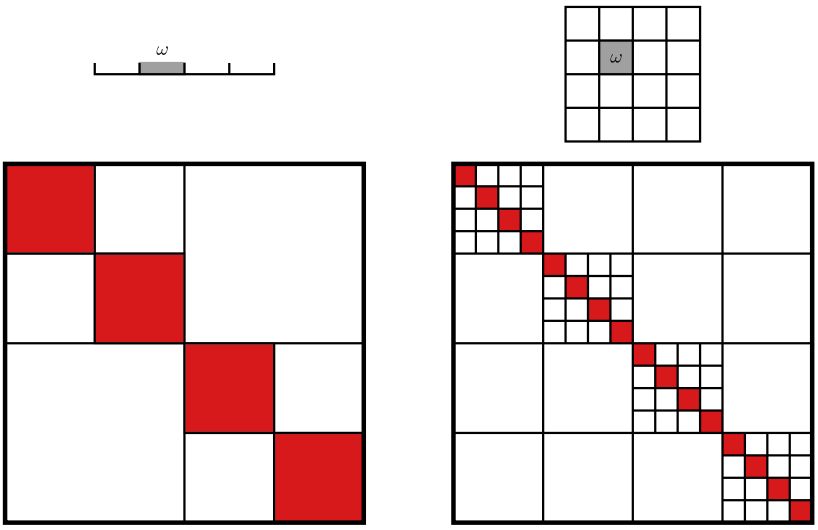


Figure 2: Weak admissibility condition and the corresponding $\mathcal{H}$-matrices for ideal one-dimensional (left) and two-dimensional (right) domains. First row shows domain partitions and gray blocks are non-admissible domains to ω. The second row shows the corresponding $\mathcal{H}$-matrices with red submatrices being dense and white ones being low-rank.

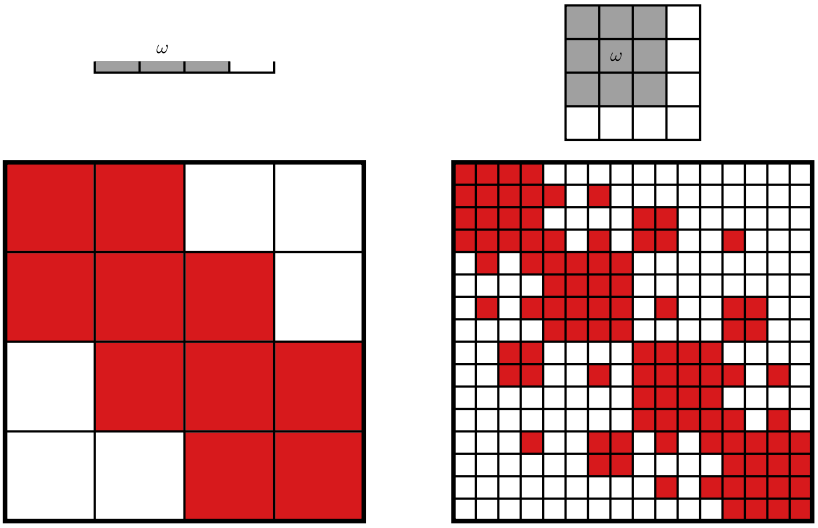


Figure 3: Standard admissibility condition and the corresponding $\mathcal{H}$-matrices for ideal one-dimensional (left) and two-dimensional (right) domains.

Based on the domain tree and admissibility conditions, we are ready to precisely define $\mathcal{H}$-matrix as an approximation of $\mathcal{K}$ mapping from $\Omega_s = \Omega$ to $\Omega_t = \Omega$, i.e., an approximation of $\mathcal{K}_{\Omega_t \times \Omega_s} = \mathcal{K}|_{\Omega_t \times \Omega_s}$. Domains Ω_t and Ω_s are called ***the target domain*** and ***the source domain*** respectively, where the target domain is associated with row indices and the source domain is associated with column indices.

Definition 2.4 ($\mathcal{H}$-matrix)**.** *Assume that $\mathcal{K}$ maps vectors defined on source domain Ω_s to vectors defined on target domain Ω_t. $\mathcal{K}$ is an $\mathcal{H}$-matrix with rank r and domain trees $\mathbb{T}_{\Omega_t}$ and $\mathbb{T}_{\Omega_s}$ if the following conditions hold in order: for each child subdomain pairs of $\Omega_t \times \Omega_s$, i.e., $\omega_t \times \omega_s \in \mathcal{C}(\Omega_t) \times \mathcal{C}(\Omega_s)$,*

1. *if $\mathcal{C}(\omega_t) = \emptyset$ or $\mathcal{C}(\omega_s) = \emptyset$, then $\mathcal{K}_{\omega_t \times \omega_s}$ is* **a dense matrix** *$D_{\omega_t \times \omega_s}$; else*
2. *if ω_t and ω_s are admissible, then $\mathcal{K}_{\omega_t \times \omega_s}$ is* **a low-rank matrix** *with rank r, i.e., $\mathcal{K}_{\omega_t \times \omega_s} = U_{\omega_t \times \omega_s} V_{\omega_t \times \omega_s}^{\top}$ for $U_{\omega_t \times \omega_s} \in \mathbb{R}^{|\omega_t| \times r}$, $V_{\omega_t \times \omega_s} \in \mathbb{R}^{|\omega_s| \times r}$, and $|\cdot|$ denotes the DOFs in the domain; otherwise*
3. *$\mathcal{K}_{\omega_t \times \omega_s}$ is* **an $\mathcal{H}$-matrix** *with rank r and domain trees $\mathbb{T}_{\omega_t}$ and $\mathbb{T}_{\omega_s}$.*

In the above definition, three conditions must be checked in the given order. The third condition defines the hierarchical structure of $\mathcal{H}$-matrix.

In order to further clarify the definition, we walk readers through the ideal two-dimensional domain case under weak admissibility condition. We begin with $\mathcal{K}$ mapping from $\Omega_s = \Omega = [0,1]^2$ to $\Omega_t = \Omega = [0,1]^2$. There are 16 child subdomain pairs in $\mathcal{C}(\Omega_t) \times \mathcal{C}(\Omega_s)$. Among all 16 pairs, all child domains have their child domains. Hence the first condition in Definition 2.4 fails for all pairs. We then check the weak admissibility condition. There are 12 out of 16 pairs are admissible, i.e., all non-overlapping domain pairs on the first level as in Fig. 1. Therefore, there are 12 off-diagonal submatrices are low-rank, which are denoted by the big white blocks in Fig. 2(right). For the rest 4 child subdomain pairs, they are $\mathcal{H}$-matrices of them own. We can continue this process until the leaf level of the domain tree and resolve the entire $\mathcal{H}$-matrix as in Fig. 2(right). The $\mathcal{H}$-matrix under standard admissibility condition is much more complicated. Fig. 3 depicts the $\mathcal{H}$-matrices with the same domain and domain tree as that in Fig. 2 but under the standard admissibility condition instead.

The $\mathcal{H}$-matrix-vector multiplication can be processed efficiently as long as we can read from the input vector and write to the output vector restricting to subdomains. We denote the $\mathcal{H}$-matrix as $\mathcal{K}$ mapping from Ω to Ω and the $\mathcal{H}$-matrix-vector multiplication as,

$$y = \mathcal{K}x,$$

where both x and y are vectors defined on Ω. We first initialize the output vector y as a zero vector. Then, we traverse all submatrices in $\mathcal{K}$ that contains data, i.e., dense submatrices and low-rank submatrices. For any such submatrix, denoted as $\mathcal{K}_{\omega_t \times \omega_s}$, we conduct the matrix-vector multiplication and add the results to the output vector,

$$y_{\omega_t} = y_{\omega_t} + \mathcal{K}_{\omega_t \times \omega_s} x_{\omega_s} = \begin{cases} y_{\omega_t} + D_{\omega_t \times \omega_s} x_{\omega_s}, \\ y_{\omega_t} + U_{\omega_t \times \omega_s} (V_{\omega_t \times \omega_s}^{\top} x_{\omega_s}), \end{cases} \tag{2.3}$$

where y_{ω_t} and x_{ω_s} denote the vector restricted to domain ω_t and ω_s respectively. When (2.3) is completed for all submatrices in $\mathcal{K}$, the vector y is already the final $\mathcal{H}$-matrix-vector multiplication result.

As shown in many previous work [6, 21], for the regular domain with almost uniformly distributed discretization points, the memory cost for the $\mathcal{H}$-matrix is $\mathcal{O}(rN\log N)$, where N is the total DOFs and r is the numerical rank. The $\mathcal{H}$-matrix-vector multiplication can be achieved in $\mathcal{O}(rN\log N)$ operations. Here we mainly reviewed the structure and matrix-vector multiplication of $\mathcal{H}$-matrix. The construction algorithms of $\mathcal{H}$-matrix [6, 36] as well as other algebraic operations such as: matrix-matrix multiplication, matrix factorization, etc., have been extensively studied in the literature, which are beyond the scope of this paper and we omit the detailed discussion.

3 $\mathcal{H}$-matrix data distribution

The organization of processes and the associated data distribution avoid expensive parallel scheduling procedure as in [26, 27]. In Section 3.1, we first explain our hierarchical organization of processes. Then in Section 3.2, the data distribution together with the load balancing are discussed.

3.1 Hierarchical process organization

Processes are organized in correspondence with the domain tree $\mathbb{T}_\Omega$. The main idea is to assign subdomains to processes as balanced as possible while preserving the hierarchical structure.

Let the P processes be indexed from 0 to $P-1$, and P be upper bounded by the number of leaf nodes in $\mathbb{T}_\Omega$ [§]. The set of all processes, denoted as $\mathcal{P}=\{0,1,\cdots,P-1\}$, is called ***the process group***. We then traverse the domain tree to assign the process group, subgroups, or individual processes to nodes in $\mathbb{T}_\Omega$. Regarding the root node in $\mathbb{T}_\Omega$, i.e., domain Ω, we assign the entire process group $\mathcal{P}$ to it. From now on, we consider a general domain Ω^ℓ in $\mathbb{T}_\Omega$ at level ℓ with a general process group $\mathcal{P}^\ell$ assigned. The assignment of subgroups of $\mathcal{P}^\ell$ to child subdomains of Ω^ℓ obeys the following conditions:

1) If the number of child subdomains of Ω^ℓ is smaller than or equal to the number of processes in $\mathcal{P}^\ell$, i.e., $|\mathcal{C}(\Omega^\ell)|\leq|\mathcal{P}^\ell|$, then $\mathcal{P}^\ell$ is partitioned into $|\mathcal{C}(\Omega^\ell)|$ subgroups such that the number of processes in each subgroup is proportional to the DOFs in the corresponding child subdomain. Each subgroup is then assigned to the corresponding child subdomain.

2) If the number of child subdomains of Ω^ℓ is bigger than the number of processes in $\mathcal{P}^\ell$, i.e., $|\mathcal{C}(\Omega^\ell)|>|\mathcal{P}^\ell|$, then $\mathcal{C}(\Omega^\ell)$ are organized into $|\mathcal{P}^\ell|$ parts such that the total DOFs in each part are balanced. Each process is then assigned to subdomains in one part.

[§]Having more processes than the number of leaf nodes ($\mathcal{O}(N)$) is feasible if the later algorithm description is slightly modified. While, such a setup is not of practical usage. Hence we omit the detail.

According to the process organization strategy, a process participates and only participates one process group at each level. When a single process is assigned to a subdomain ω in $\mathbb{T}_\Omega$, it is assigned to all descendants of ω in $\mathbb{T}_\Omega$. We can then combine all subdomains that are singly owned by a process p and denote the union as ω_p. All such unions $\{\omega_p\}_{p=0}^{P-1}$ form a balanced partition of Ω, where the balancing factor is upper bounded by twice the balancing factor of $\mathbb{T}_\Omega$. The balancing factor here is referring to the ratio of the heaviest workload and the lightest workload among all processes. Further, in each process group or subgroup, the process with smallest index is called ***the group leader***, e.g., 0 is the group leader of $\mathcal{P}$. When a process is the group leader at a level ℓ, then it is the group leader in all descendant groups it participates. For example, process 0 is group leaders of all process groups it participates, whose workload is the heaviest among all processes.

Fig. 4 top show a four level domain tree together with its process assignment of $\mathcal{P} = \{0,1,\cdots,7\}$ for an ideal one-dimensional domain. Each process is assigned to a unique subdomain at level 3. In addition, Fig. 5 top show a three level domain tree together with its process assignment of $\mathcal{P} = \{0,1,\cdots,7\}$ for an ideal two-dimensional domain. We find that process owns two subdomains at level 2 and eight processes form a perfect partition of the domain.

3.2 Data distribution and load balancing

Definition 2.4 explicitly shows that all data (matrix entries) are in two types of submatrices, either dense submatrices or low-rank submatrices. The hierarchical submatrices defined by the third conditions in Definition 2.4 exist virtually for recursion purpose. Hence, we just need to distribute dense submatrices and low-rank submatrices among processes.

Low-rank submatrix. Consider a low-rank submatrix associated with domain pair $\Omega_t \times \Omega_s$, where Ω_t and Ω_s are the target and source domains respectively. According to the process organization defined in Section 3.1, there are two process groups assigned to Ω_t and Ω_s, denoted as $\mathcal{P}_t$ and $\mathcal{P}_s$ respectively. We distribute two factors in the low-rank submatrices, $U_{\Omega_t \times \Omega_s}$ and $V_{\Omega_t \times \Omega_s}$, to two process groups, i.e., $U_{\Omega_t \times \Omega_s}$ is stored among $\mathcal{P}_t$ and $V_{\Omega_t \times \Omega_s}$ is stored among $\mathcal{P}_s$. If either $\mathcal{P}_t$ or $\mathcal{P}_s$ has only one process, then the process owns the entire matrix. Now, assume there are more than one process in $\mathcal{P}_t$. Since $U_{\Omega_t \times \Omega_s}$ is a tall and skinny matrix, it is distributed in a block row fashion. For each process $p \in \mathcal{P}_t$, the rows corresponding to ω_p is owned by process p, where ω_p is the singly owned subdomain of p. If there are more than one process in $\mathcal{P}_s$, then $V_{\Omega_t \times \Omega_s}$ is distributed in the same way among processes in $\mathcal{P}_s$.

Dense submatrix. Consider a dense submatrix associated with domain pair $\Omega_t \times \Omega_s$ and the corresponding process group pair $\mathcal{P}_t \times \mathcal{P}_s$. There are three scenarios of the sizes

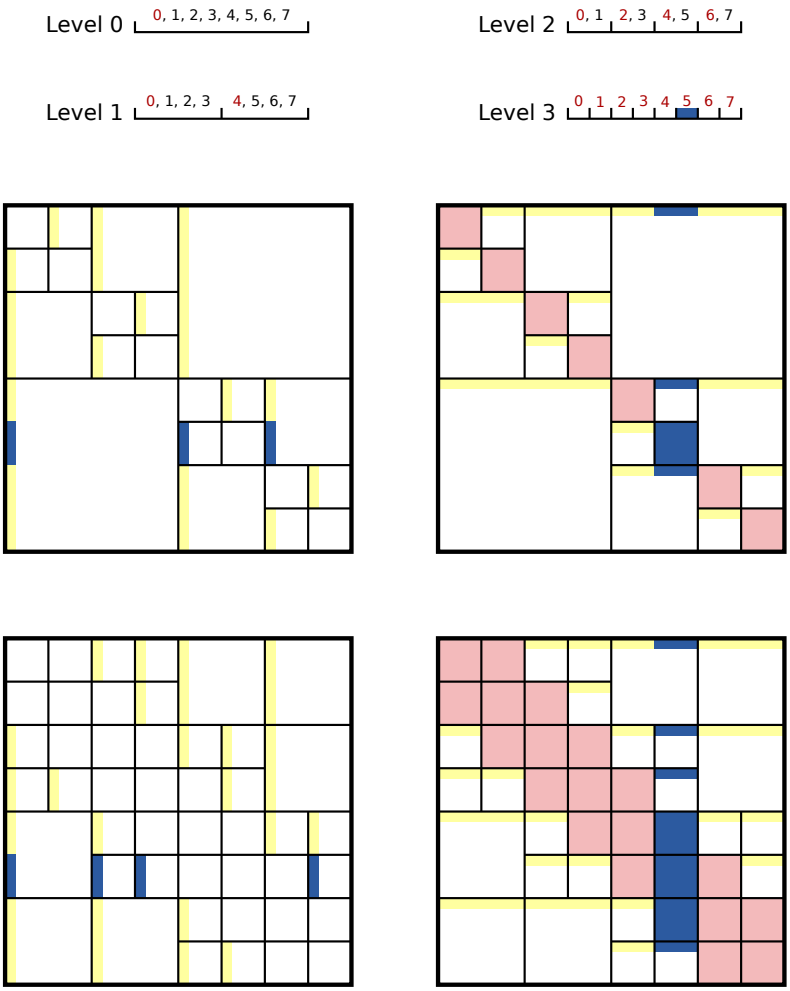


Figure 4: $\mathcal{H}$-matrix data distribution of an ideal one-dimensional domain with weak and standard admissibility condition on 8 processes. Top part is the four-level domain tree together with its process assignment. Red processes are group leaders. Middle parts and bottom parts are the distributed $\mathcal{H}$-matrix with weak and standard admissibility condition respectively. Left columns are $\mathcal{H}$-matrix owned by target process groups and right columns are owned by source process groups. Blue blocks indicate data owned by process 5 whereas light red and yellow blocks indicate data owned by other processes.

of process groups: (i) $|\mathcal{P}_t| = |\mathcal{P}_s| = 1$; (ii) $|\mathcal{P}_t| = 1$ and $|\mathcal{P}_s| > 1$; (iii) $|\mathcal{P}_t| > 1$ and $|\mathcal{P}_s| = 1$. In the first scenario, the dense matrix $D_{\Omega_t \times \Omega_s}$ is owned by $\mathcal{P}_s$. In the second scenario, the transpose of dense matrix, $D^{\top}_{\Omega_t \times \Omega_s}$, is distributed among $\mathcal{P}_s$ in the same way as the distribution of $V_{\Omega_t \times \Omega_s}$ above. In the last scenario, the dense matrix $D_{\Omega_t \times \Omega_s}$ is distributed among $\mathcal{P}_t$ in the same way as the distribution of $U_{\Omega_t \times \Omega_s}$ above.

Once the data distribution strategies are applied to all submatrices, the $\mathcal{H}$-matrix is then fully distributed among $\mathcal{P}$. To further facilitate the understanding of the overall data distribution, Fig. 4 and Fig. 5 show the data in $\mathcal{H}$-matrices for an ideal one-dimensional

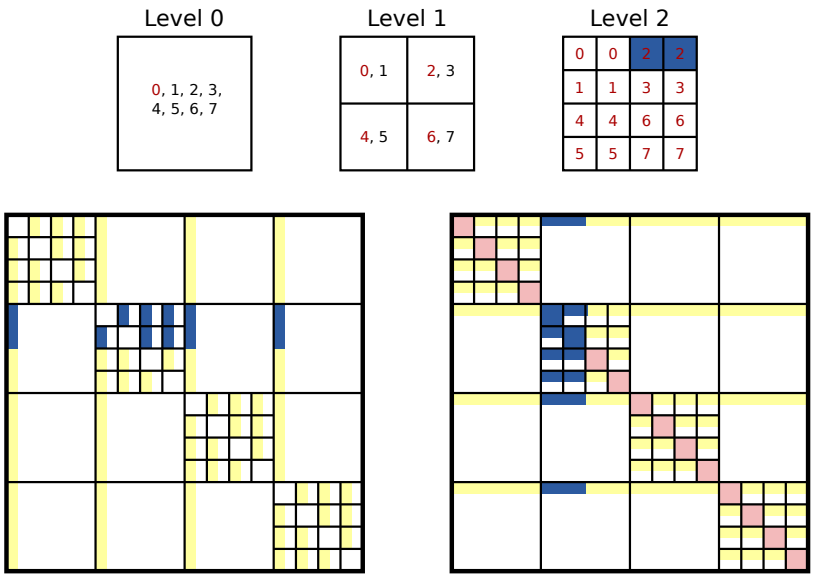


Figure 5: $\mathcal{H}$-matrix data distribution of an ideal two-dimensional domain with weak admissibility condition on 8 processes. Top part is the three-level domain tree together with its process assignment. Red processes are group leaders. Bottom parts are data in $\mathcal{H}$-matrix owned by target process groups (left) and source process groups (right). Blue blocks indicate data owned by process 2 whereas light red and yellow blocks indicate data owned by other processes.

domain with weak and standard admissibility condition and an ideal two-dimensional domain with weak admissibility condition respectively. Both $\mathcal{H}$-matrices are distributed among process group $\mathcal{P}$ of size eight. In Fig. 4 and Fig. 5, blue blocks highlight the data owned by process 5 and process 2 respectively.

Remark 3.1. The data distribution strategies we introduced here are suitable and efficient for a sequence of parallel-friendly $\mathcal{H}$-matrix algebraic operations, e.g., matrix-vector multiplication, matrix-matrix multiplication, matrix compression, matrix addition, etc. While some other $\mathcal{H}$-matrix algebraic operations, like $\mathcal{H}$-matrix-LU factorization and $\mathcal{H}$-matrix-inversion, are not parallel-friendly since the operations therein depends sequentially on each other. Our data distribution strategies work for these operations as well, while the efficiency is left to be further explored.

Remark 3.2. These data distribution strategies can also be easily extended to $\mathcal{H}^2$-matrix. The nested basis in $\mathcal{H}^2$-matrix can be distributed among all processes in the similar way as we distribute low-rank factors. While the tiny middle matrix in each low-rank block in $\mathcal{H}^2$-matrix could be singly owned by either its source or target group leader. Given such distribution strategies for $\mathcal{H}^2$-matrix, all its algebraic operations can be parallelized in an analog way as that for $\mathcal{H}$-matrix.

We now discuss the load balancing of the distributed $\mathcal{H}$-matrix. As shown in Figs. 4 and 5, the load balancing is different for different admissibility conditions. Fig. 5 un-

der weak admissibility condition shows an ideal load balancing whereas Fig. 4 under standard admissibility condition shows slightly unbalanced data distribution. In the following, we assume the domain is an ideal d-dimensional domain, $[0,1]^d$ with n uniform discretization points on each dimension and $N=n^d$ discretization points in total. In such an ideal case, each process own the same size of subdomain.

Assume that the weak admissibility condition is applied. At each level on $\mathbb{T}_\Omega$, any domain Ω^ℓ has the same number of admissible domains. Each process participate one domain on the target side and another on the source side. Hence all processes own exactly the same amount of data in low-rank submatrices at each level. For all low-rank submatrices throughout levels, data are evenly distributed among all processes. Regarding the dense submatrices, they are all of the same size and owned by their source processes. Since all processes own the same size domains on the source side, and these domains have the same amount of dense submatrices, all processes own the same amount of dense submatrix data. Overall, the data of dense submatrices and low-rank submatrices are evenly distributed among all processes and the load balancing in this case is ideal.

While, when the standard admissibility condition is applied, the load balancing depends on the boundary condition of the problem. If the periodic boundary condition is adopted, the load balancing is still ideal. While, if a non-periodic boundary condition is adopted, the data loads are different for processes owning domains near the center and processes owning domains near corners. Since all low-rank submatrices are evenly owned by processes in its process groups, the load balancing factor is simply the ratio of the numbers of low-rank submatrices for different processes, i.e., the numbers of admissible domains. Consider level ℓ, which is neither the first two levels nor the last one. A center subdomain $\Omega^\ell_{\text{center}}$'s parent domain has 3^d non-admissible neighbor domains, each of which is partitioned into 2^d subdomains at level ℓ. Excluding non-admissible subdomains of $\Omega^\ell_{\text{center}}$, there are $3^d\cdot 2^d-3^d$ admissible subdomains of $\Omega^\ell_{\text{center}}$. However, a corner subdomain $\Omega^\ell_{\text{corner}}$'s parent domain is also a corner domain and has 2^d non-admissible neighbor domains. Through the similar calculation, $\Omega^\ell_{\text{corner}}$ has $2^d\cdot 2^d-2^d$ admissible subdomains at level ℓ. Hence the load balancing factor is $\frac{3^d}{2^d}$. Such a factor also holds to the load balancing of dense submatrices. Overall, asymptotically as N goes to infinity, the load balancing factor for distributed $\mathcal{H}$-matrix under standard admissibility condition and non-periodic boundary condition is upper bounded by $\left(\frac{3}{2}\right)^d$. Since this factor is independent of both N and P, we still regard our data distribution in this case as a balanced one.

4 Distributed-memory $\mathcal{H}$-matrix-vector multiplication

$\mathcal{H}$-matrix-vector multiplication is the fundamental operation in $\mathcal{H}$-matrix algebra and reveals the value of $\mathcal{H}$-matrix as a fast algorithm. Further, it is also one of basic operations involved in other $\mathcal{H}$-matrix algebraic operations, including, matrix-matrix multiplication,

matrix compression, matrix factorization, and matrix inversion. As briefly reviewed in Section 2, the sequential $\mathcal{H}$-matrix-vector multiplication is as simple as looping over all low-rank and dense submatrices, multiplying the submatrix to the input vector restricted to the source domain, and adding the result to the output vector restricted to the target domain. However, the distributed-memory version is much more complicated. Based on the data distribution as in Section 3, we present the distributed-memory $\mathcal{H}$-matrix-vector multiplication algorithm in this section followed by its complexity analysis.

4.1 Algorithm

Distributed-memory $\mathcal{H}$-matrix-vector multiplication algorithm mainly consists of the following five steps:

Step 1. Source side local computation;

Step 2. Tree-reduction on source process tree;

Step 3. Data transfer from source to target;

Step 4. Tree-broadcast on target process tree;

Step 5. Target side local computation.

Among these five steps, Steps 1 and 5 only involve computations and are communication-free whereas Steps 2, 3, and 4 focus on efficient communication under our data distribution and process organization. We will elaborate five steps in detail one-by-one. Throughout the following description, we assume the input vector x is already distributed in the block row fashion among process group $\mathcal{P}$. More precisely, for any process $p\in\mathcal{P}$, it owns $x_{\omega_p}=x|_{\omega_p}$ for ω_p being p's singly owned domain. The output vector y will be distributed exactly in the same way as x.

4.1.1 Source side local computation

The source side local computation goes through all submatrices containing data, i.e., low-rank submatrices and dense submatrices, and conducts all communication-free calculations. We now describe specific operations for submatrices of different types.

Low-rank submatrix. Consider a low-rank submatrix associated with $\Omega_t\times\Omega_s$ and process groups $\mathcal{P}_t\times\mathcal{P}_s$. The explicit block form of $V_{\Omega_t\times\Omega_s}$ and x_{Ω_s} admit,

$$V_{\Omega_t\times\Omega_s}=\begin{pmatrix} v_{p_0} \\ \vdots \\ v_{p_{|\mathcal{P}_s|-1}} \end{pmatrix}, \quad x_{\Omega_s}=\begin{pmatrix} x_{p_0} \\ \vdots \\ x_{p_{|\mathcal{P}_s|-1}} \end{pmatrix}, \tag{4.1}$$

where $p_i \in \mathcal{P}_s$, v_{p_i} and x_{p_i} are stored on process p_i. We aim to compute the product of $V_{\Omega_t\times\Omega_s}$ and x_{Ω_s} as,

$$V_{\Omega_t\times\Omega_s}^{\top} x_{\Omega_s} = \sum_{i=0}^{|\mathcal{P}_s|-1} v_{p_i}^{\top} x_{p_i}, \tag{4.2}$$

where the summation over i requires communication since $v_{p_i}^{\top} x_{p_i}$ are owned by different processes for different i. Hence, in this step, we only compute

$$z_{\text{local}} = v_{p_i}^{\top} x_{p_i} \tag{4.3}$$

on process p_i without conducting any communication. The communication for the summation over i in (4.2) is postponed until the next step.

Dense submatrix. Consider a dense submatrix associated with $\Omega_t\times\Omega_s$ and $\mathcal{P}_t\times\mathcal{P}_s$. When there are more than one process in the target process group, i.e., $|\mathcal{P}_t|>1$, the data in this submatrix are owned by the target process group. No local computation is needed and we assign $z_{\text{local}} = x_{\Omega_s}$ for later communications. When there is only one process in the target process group, i.e., $|\mathcal{P}_t|=1$, the data are distributed among the source process group as,

$$D_{\Omega_t\times\Omega_s} = \begin{pmatrix} d_{p_0} & \cdots & d_{p_{|\mathcal{P}_s|-1}} \end{pmatrix}, \quad x_{\Omega_s} = \begin{pmatrix} x_{p_0} \\ \vdots \\ x_{p_{|\mathcal{P}_s|-1}} \end{pmatrix}, \tag{4.4}$$

for $p_i\in\mathcal{P}_s$ and $|\mathcal{P}_s|\geq 1$. Similar to the low-rank submatrix case, we aim to compute

$$D_{\Omega_t\times\Omega_s} x_{\Omega_s} = \sum_{i=0}^{|\mathcal{P}_s|-1} d_{p_i} x_{p_i}. \tag{4.5}$$

Instead, we only conduct local computation in this step, $z_{\text{local}} = d_{p_i} x_{p_i}$, on each process $p_i\in\mathcal{P}_s$ without communication.

4.1.2 Tree-reduction on source process tree

This step implements the communication required summations in (4.2) and (4.5). Naïvely, we can perform many MPI reductions[¶], one for each submatrices and reduce the summation results to their group leaders. However, such a naïve reduction strategy requires many more messages than the *tree-reduction* to be introduced below, which benefits most from the hierarchical organization of both the $\mathcal{H}$-matrix and processes.

The preliminary step in tree-reduction is to collect and pack local results that require communication in (4.2) and (4.5). For each process, we visit the $\mathcal{H}$-matrix level by level from root to leaf. At each level, each process participates and only participates in one process group. Hence, local results are about to be reduced to the same group leader

[¶]We refer to "MPI_Reduce" with addition operation as the reduction throughout this paper.

and are packed together in an array in the same ordering. Across levels, we concatenate packed local results together until one level before the level where process group has only one process. We denote the maximum number of such levels as L_P.

Then a sequence of reductions are conducted from level L_P backward to the root level. At level L_P, all processes reduce the entire concatenated array to their own group leaders at this level. Group leaders at level L_P then have already collected their group members' contributions to summations from root level to level L_P-1. Hence, those non-leader group members at level L_P no longer participate the rest communications in this step. At a following level $\ell=L_P-1, L_P-2, \cdots, 1$, the participating processes are those group leaders at level $\ell+1$. They reduce their concatenated array from level 1 to level ℓ (with contributions from their own group members) to their own group leaders at level ℓ. When all reductions are completed, all group leaders own the summations (4.2) and (4.5) of their groups. Slightly abuse of notation, we still denote these summation results as z_{local}.

Remark 4.1. When the domain and discretization are far from balanced ones, the process tree is also not balanced. Hence, it is possible that at some level $\ell < L_P$, a process is the process group of its own. In this case, such a process do not need to participate the reduction at level ℓ or lower. We do not exclude such cases from our description above, but do exclude them from our implementation.

Fig. 6 depicts the flow of a tree-reduction for an ideal one-dimensional domain distributed evenly on 8 processes. Although there is no communication-required data on the root level in $\mathcal{H}$-matrix-vector multiplication, we still include data cubics on level 0 in the figure to demonstrate the idea and show the extendability of the tree-reduction to more than two levels.

4.1.3 Data transfer from source to target

After the previous step, all local data, z_{local}, are stored on their own group leaders on the source side. In order to finish the computation, local data should be sent to the processes in the target group. To better benefit from the hierarchical structure, we accomplish the communication in this and next steps. In this step, local data will be sent from the source group leaders to the corresponding target group leaders. Then the next step is responsible for broadcasting local data to the processes in target groups.

Given a pair of target and source group leaders, p_t and p_s, they could be the group leaders of many submatrices. Hence process p_s first packs local data in all those submatrices and then send them in one message to process p_t. After process p_t received the packed local data, it then unpacks the data to submatrices.

Remark 4.2. We emphasize that a process only participates at most $\mathcal{O}(\log P)$ number of group leader pairs. Let us consider process 0 as the source group leader, which acts most frequently as the source group leader among all processes. As we mentioned before, each process only participates one process group on each level of the process tree. Process 0 is then the group leaders of one process group on each level, which adds to $\mathcal{O}(\log P)$

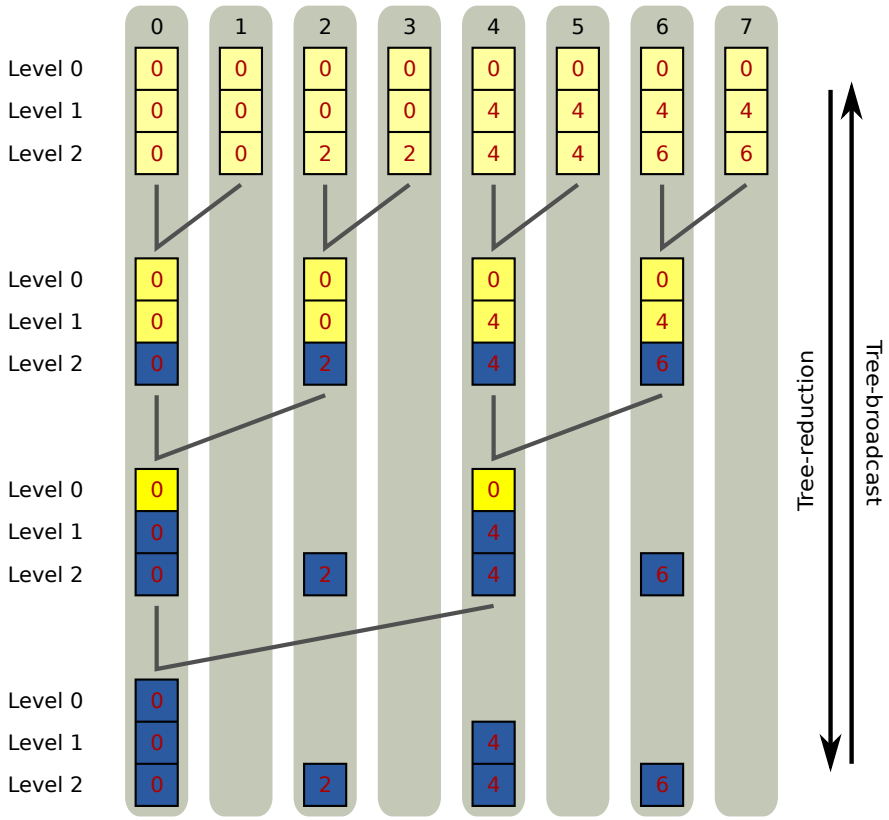


Figure 6: Tree-communication flowchart. Tree-reduction and tree-broadcast flow from top to bottom and from bottom to top respectively. Different columns with gray background are the concatenated arrays owned by different processes. Each cubic is the packed data on the corresponding level and the number in the cubic indicates its group leader. For tree-reduction, yellow cubics are local data to be reduced to their group leaders and summed together whereas blue cubics are the final summation results owned only by group leaders. As shown in the figure, only yellow cubics and their owner processes participate the reduction communications. For tree-broadcast, blue cubics are original packed data to be sent to group members. Yellow cubics are packed data been broadcasted. Light yellow cubics are final broadcasted data.

groups. A source process group on each level only interacts with a constant number of target process groups, where the constant depends on the admissibility condition. Hence process 0 is paired with a constant number of target group leaders at each level. Summing all levels together, process 0 is paired with $\mathcal{O}(\log P)$ target group leaders.

4.1.4 Tree-broadcast on target process tree

Consider a low-rank submatrices associated with $\Omega_t \times \Omega_s$ with process groups $\mathcal{P}_t \times \mathcal{P}_s$ as an example. The matrix vector multiplication admits,

$$U_{\Omega_t\times\Omega_s} V^{\top}_{\Omega_t\times\Omega_s} x_{\Omega_s} = \begin{pmatrix} u_{p_0}\left(V^{\top}_{\Omega_t\times\Omega_s} x_{\Omega_s}\right) \\ \vdots \\ u_{p_{|\mathcal{P}_t|-1}}\left(V^{\top}_{\Omega_t\times\Omega_s} x_{\Omega_s}\right) \end{pmatrix} = \begin{pmatrix} u_{p_0} z_{\text{local}} \\ \vdots \\ u_{p_{|\mathcal{P}_t|-1}} z_{\text{local}} \end{pmatrix}, \tag{4.6}$$

where $p_i \in \mathcal{P}_t$ and z_{local} is the summation in (4.2). After the previous step, in each submatrices, z_{local} is owned by the target group leaders. Hence, in order to conduct the product of $u_{p_i} z_{\text{local}}$ as in (4.6), z_{local} needs to be shared with all target group members. A similar equation can be written down for dense submatrices with target process groups of size greater than one. In this step, we hierarchically broadcast the local data z_{local} from the group leaders to the group members together and name it as *tree-broadcast*, which is the reverse procedure of tree-reduction.

Similar to tree-reduction, we first collect and pack local results that require communication. For each group leader, we visit the $\mathcal{H}$-matrix level by level from root to leaf. At each level, local results that are about to be broadcasted to the same group are packed together in an array. Across levels, we concatenate packed local results together until level L_P.

Then a sequence of broadcasts are executed from the first level forward to level L_P. At a level $\ell = 1, \cdots, L_P - 1$, the group leaders broadcast their array from level 1 to level ℓ to those subgroup leaders at level $\ell+1$. Subgroup leaders then concatenate the received array together with their own packed array. Once the concatenating procedure is accomplished, we move on to the next level. Finally, at level L_P, group leaders broadcast their entire array to all their group members. All processes in target process group, in the end, received all needed local data for each submatrices they participated.

Similar level skipping for the unbalanced target process tree can be done for tree-broadcast as that for tree-reduction in Remark 4.1. Fig. 6 illustrates a tree-broadcast procedure for an ideal one-dimensional domain distributed on 8 processes.

4.1.5 Target side local computation

The target side local computation goes through all low-rank and dense submatrices and conducts aggregation of the product results onto output vector y. Here we assume the output vector y is initialized to be all zero. We describe operations for different types of submatrices.

Low-rank submatrix. Consider a low-rank submatrix associated with $\Omega_t \times \Omega_s$ and process groups $\mathcal{P}_t \times \mathcal{P}_s$. As shown in (4.6), for a process $p_i \in \mathcal{P}_t$, the product result is $u_{p_i} z_{\text{local}}$. After previous step, z_{local} is owned by p_i. Hence we only need to process the following communication-free computation,

$$y_{p_i} = y_{p_i} + u_{p_i} z_{\text{local}}, \tag{4.7}$$

where y_{p_i} is the output vector y restricted to the subdomain in Ω_t owned by p_i.

Dense submatrix. Consider a dense submatrix associated with $\Omega_t \times \Omega_s$ and process groups $\mathcal{P}_t \times \mathcal{P}_s$. If there is only one process in $\mathcal{P}_t$, then the matrix-vector multiplication as in (4.5) has already been conducted in the first step and the result z_{local} is also owned by

$\mathcal{P}_t$ after previous communication steps. Hence we simply add it to the output vector,

$$y_{\Omega_t} = y_{\Omega_t} + z_{\text{local}}. \tag{4.8}$$

If there are more than one process in $\mathcal{P}_t$, then the dense matrix is owned by $\mathcal{P}_t$ in a block row fashion and the matrix vector multiplication admits,

$$D_{\Omega_t \times \Omega_s} x_{\Omega_s} = \begin{pmatrix} d_{p_0} x_{\Omega_s} \\ \vdots \\ d_{p_{|\mathcal{P}_t|-1}} x_{\Omega_s} \end{pmatrix} = \begin{pmatrix} d_{p_0} z_{\text{local}} \\ \vdots \\ d_{p_{|\mathcal{P}_t|-1}} z_{\text{local}} \end{pmatrix}, \tag{4.9}$$

where $p_i \in \mathcal{P}_t$ and each p_i has a copy of z_{local}. In this step, process p_i is responsible for the following computation,

$$y_{p_i} = y_{p_i} + d_{p_i} z_{\text{local}}, \tag{4.10}$$

where y_{p_i} is the same as that in (4.7).

Remark 4.3. Here we described the algorithm computing $y = \mathcal{K}x$ for a distributed-memory $\mathcal{H}$-matrix $\mathcal{K}$. A more standard matrix-vector multiplication operator in linear algebra would be $y = \alpha\mathcal{K}x + \beta y$, which is the "GEMV" operation in level 2 BLAS. Such an operation can be easily adopted here if we do not initialize y as a zero vector and modify (4.7), (4.8), and (4.10) accordingly. All the rest steps remain unchanged.

4.2 Complexity analysis

In this section, we analyze the computational and the communication complexities of the distributed-memory $\mathcal{H}$-matrix-vector multiplication algorithm. To simplify the notation, we denote $L_p = \mathcal{O}(\log P)$ and $L_N = \mathcal{O}(\log N)$ as the number of levels in process trees[‖] and domain trees respectively.

The computational complexity is easy to conclude given our previous analysis on the data balancing in Section 3.2. Notice that our total number of floating-point operations stay identical to that of sequential $\mathcal{H}$-matrix-vector multiplication if the extra computation in tree-reduction is excluded. While, the computation in tree-reduction is of lower order comparing to that of dense matrix-vector multiplication conducted on each processes. Hence, the extra computation in communication steps can be ignored in our complexity analysis. Further, processes conduct float operations proportional to amounts of data they owned. Thanks to the balanced data distribution, we conclude that the computational operations are also balanced across all processes and each process conduct $\mathcal{O}(\frac{N \log N}{P})$ operations.

The communication complexity consists of two parts: the latency (α) and the per-process inverse bandwidth (β). The complexity analysis for the latency is relatively

[‖]Here we count the number of levels in a process tree until the first level such that all process groups contain one process.

simpler and stay the same for different admissibility conditions. The latency is essentially counting the number of send/receive communications. Each process in the tree-reduction and tree-broadcast steps conducts a reduction and broadcast among constant number of processes. Hence each process conduct $\mathcal{O}(1)$ send/receive communications on each level. Summing all L_P levels together, the latencies for both tree-reduction and tree-broadcast are $\mathcal{O}(\alpha \log P)$. Regarding the Step 3 in our algorithm, as discussed in Remark 4.2, each process only communicates with $\mathcal{O}(\log P)$ other processes. Hence the latency for Step 4 and the overall latency are $\mathcal{O}(\alpha \log P)$. The complexities of inverse bandwidth, however, are different for different admissibility conditions and are discussed separately.

Weak admissibility condition. Consider the tree-reduction and tree-broadcast steps. At a given level $\ell \leq L_P$, each process only participates one process group and owns a constant number of submatrices. Hence the final concatenated array is of length $\mathcal{O}(L_P)$. Process 0 is the most communication intensive process. For level $\ell=1,\cdots,L_P$, it communicates an array of size $\mathcal{O}(\ell)$ in both tree-reduction and tree-broadcast. Therefore, process 0 in total send and receive $\mathcal{O}(L_P^2)$ data, which is an upper bound for other processes. The inverse bandwidth complexities for the tree-reduction and tree-broadcast steps are then $\mathcal{O}(\beta \log^2 P)$.

The inverse bandwidth complexity for the third step is very much simplified for $\mathcal{H}$-matrices under weak admissibility condition due to one crucial difference between weak admissibility condition and other admissibility conditions. $\mathcal{H}$-matrices under weak admissibility condition only have $\mathcal{H}$-submatrices along their diagonal blocks, whereas $\mathcal{H}$-matrices under other admissibility conditions have $\mathcal{H}$-submatrices on off-diagonal blocks. Under the distributed-memory setting, such a property means that the source and target process groups remain the same for all $\mathcal{H}$-submatrices when weak admissibility condition is adopted. Hence only low-rank submatrices are distributed among different source and target process groups. Now we again consider process 0, who are group leaders across all levels. For levels below L_P, process 0 does not participate any submatrices with different source and target process groups. For level L_P and above, process 0 is responsible to send the entire reduced array of length $\mathcal{O}(L_P)$ to other processes. Hence the inverse bandwidth complexities for process 0 is $\mathcal{O}(\beta \log P)$, which is the upper bound for other processes.

Overall, the complexity, including both computational complexity and communication complexity, for distributed-memory $\mathcal{H}$-matrices under weak admissibility condition on P processes is

$$\mathcal{O}\left(\frac{N \log N}{P} + \alpha \log P + \beta \log^2 P\right). \tag{4.11}$$

Standard admissibility condition. All communication complexity analyses under the weak admissibility condition carry over to that under the standard admissibility condi-

tion with a different prefactor, which is determined by the number of admissible neighbors. Some extra communication costs come from those $\mathcal{H}$-submatrices singly owned by different target process and source process. In this case, no tree-communication is needed. But the source process need to pack all local data in this $\mathcal{H}$-submatrices and send them to the target process. The amount of local data in the $\mathcal{H}$-submatrices is a constant times the number of low-rank and dense submatrices. Such $\mathcal{H}$-submatrices are mostly corresponding to neighboring subdomains and are of sizes $\frac{N}{P}$. With a complicated calculation, which is omitted here, such $\mathcal{H}$-submatrices have $\mathcal{O}\big(\log\frac{N}{P}\big)$ low-rank submatrices and $\mathcal{O}\big((\frac{N}{P})^{\frac{d-1}{d}}\big)$ dense submatrices, where d is the dimension of the problem. The number of low-rank submatrices essentially calculates the number of levels whereas the number of dense submatrices calculates the number of the subdomains of finest scale on the interface of the two neighboring subdomains. Hence the extra communication cost under standard admissibility condition is $\mathcal{O}\big(\beta\big(\log\frac{N}{P}+(\frac{N}{P})^{\frac{d-1}{d}}\big)\big)$.

Overall, the complexity for distributed-memory $\mathcal{H}$-matrices under standard admissibility condition on P processes is

$$\mathcal{O}\left(\frac{N\log N}{P}+\alpha\log P+\beta\left(\log^2 P+\log\frac{N}{P}+\left(\frac{N}{P}\right)^{\frac{d-1}{d}}\right)\right). \tag{4.12}$$

Remark 4.4. According to (4.11) and (4.12), we notice the trade-off between the computational complexity and the communication complexity. When P is much smaller than N, the dominate cost comes from the computational part. While as P approaches N, the computational cost is then $\mathcal{O}(\log N)$ whereas the communication complexity is $\mathcal{O}(\log^2 P)$ dominating the cost.

5 Numerical results

All numerical experiments were performed on the Texas Advanced Computing Center (TACC) cluster, Stampede2. This cluster has 4,200 Intel Knights Landing nodes, each with 68 cores, 96 GB of DDR memory. Nodes are interconnected via Intel Omni-Path network with a fat tree topology. We allocate various number of nodes for our tests and each node runs 32 MPI processes. The memory limit per process is 3 GB.

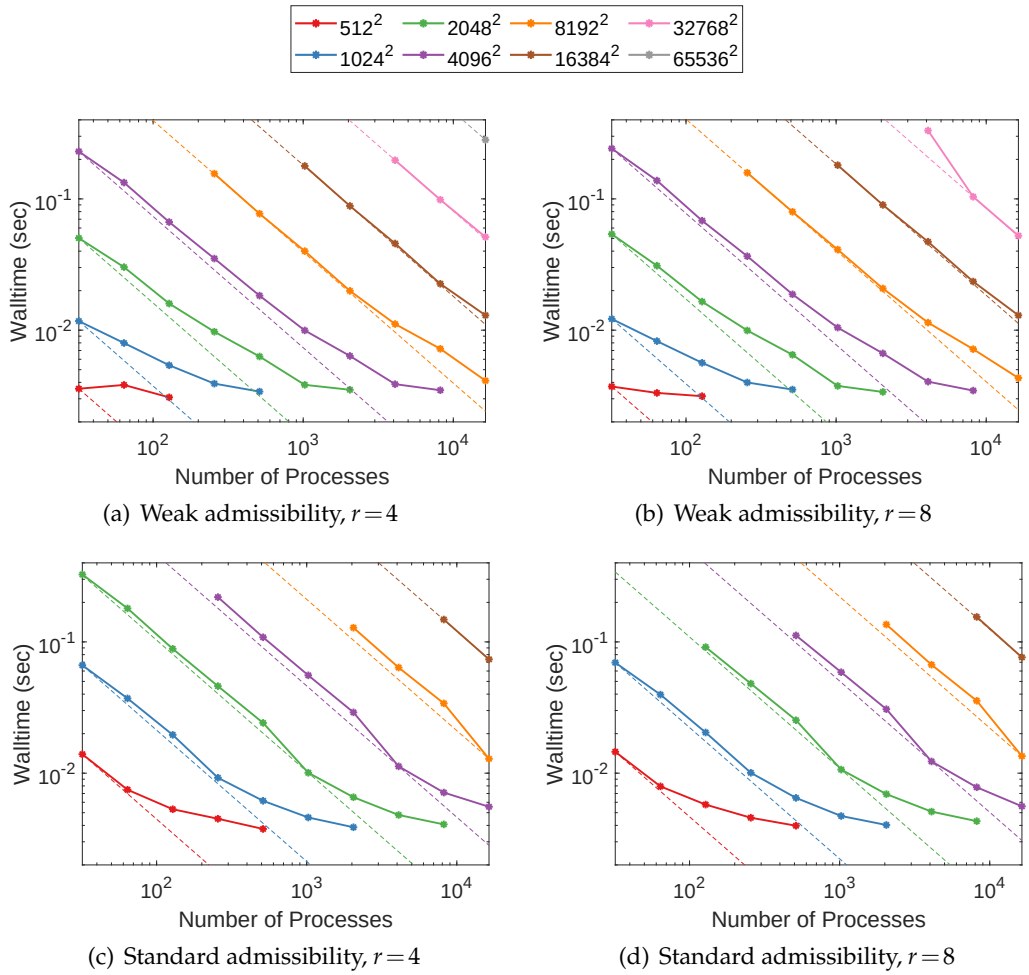
In the following numerical results, we adopt a few measurements to demonstrate the parallel efficiency of our algorithm. In addition to the regular wall-clock time (walltime), we also calculate the speedup as well as the efficiency factor. Given a problem, we denote P_0 as the smallest number of processes that are able to solve the problem and solve it in t_0 seconds. Meanwhile, solving the problem among P_1 processes for $P_1\geq P_0$ takes t_1 seconds. The speedup and the efficiency factor (percentage) in this case are,

$$\text{Speedup}=\frac{P_0t_0}{t_1} \quad \text{and} \quad \text{Eff}=\frac{P_0t_0}{P_1t_1}\cdot 100, \tag{5.1}$$

respectively.

5.1 $\mathcal{H}$-matrices for two-dimensional problems

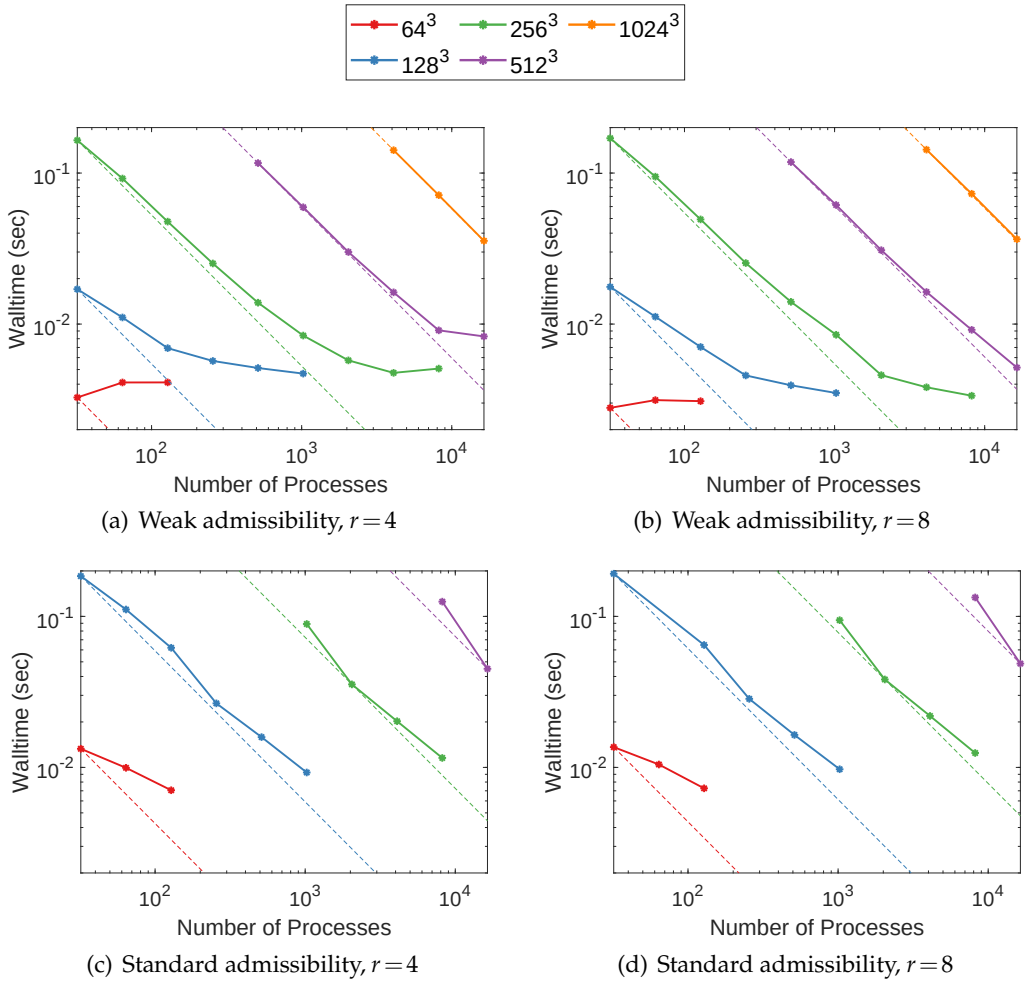
Let $\Omega=[0,1]^2$ be the domain of interest. We discretize the problem with n points on each dimension for $n=512,1024,\cdots,65536$. Hence the corresponding matrices are of size varying from $512^2\times512^2$ up to $65536^2\times65536^2$. The structure of an $\mathcal{H}$-matrix is then determined by a hierarchical partition of Ω. Since the construction of $\mathcal{H}$-matrix is beyond the scope of this paper and $\mathcal{H}$-matrix-vector multiplication does not rely on the properties of the underlying problems, we fill dense submatrices and low-rank submatrices in $\mathcal{H}$-matrices by random numbers and use these random $\mathcal{H}$-matrices to explore the parallel



(a) Weak admissibility, $r=4$ (b) Weak admissibility, $r=8$

(c) Standard admissibility, $r=4$ (d) Standard admissibility, $r=8$

Figure 7: Strong scaling of $\mathcal{H}$-matrix-vector multiplication for various two-dimensional problems on various number of processes (up to 16384 processes). Figures (a) and (b) are $\mathcal{H}$-matrices under weak admissibility condition with rank being 4 and 8 respectively. Figures (c) and (d) are $\mathcal{H}$-matrices under standard admissibility condition with rank being 4 and 8 respectively. Solid lines are strong scaling curves and dash lines are their corresponding theoretical references. Different colors are problems of different sizes as indicated in the legend.



(a) Weak admissibility, $r=4$ (b) Weak admissibility, $r=8$

(c) Standard admissibility, $r=4$ (d) Standard admissibility, $r=8$

Figure 8: Strong scaling of $\mathcal{H}$-matrix-vector multiplication for various three-dimensional problems on various number of processes (up to 16384 processes). Figures (a) and (b) are $\mathcal{H}$-matrices under weak admissibility condition with rank being 4 and 8 respectively. Figures (c) and (d) are $\mathcal{H}$-matrices under standard admissibility condition with rank being 4 and 8 respectively. Solid lines are strong scaling curves and dash lines are their corresponding theoretical references. Different colors are problems of different sizes as indicated in the legend.

scaling of our algorithm. Also random input vectors are used in our tests. Both weak admissibility condition and standard admissibility condition are explored. In addition, we use two choices of r, r=4 and r=8, where the later makes problems more computation intensive. Each $\mathcal{H}$-matrix is distributed among various number of processes, from 32 up to 16384. The reported runtime is averaged over 128 random input vectors.

Fig. 8 depicts strong scaling plots for different $\mathcal{H}$-matrices and Table 1 further details walltimes, speedups and efficiency factors. In both weak admissibility condition cases, Figs. 7(a) and 7(b), strong scaling is well-preserved as we keep doubling the number of processes. Towards the end of each curve, when the communication cost dominates the

Table 1: Numerical results of distributed-memory $\mathcal{H}$-matrix-vector multiplication for two-dimensional problems.

N	r	P	Weak			Standard		
			Time (s)	Speedup	Eff (%)	Time (s)	Speedup	Eff (%)
512^2	4	32	3.58e-03	32.0x	100.0	1.39e-02	32.0x	100.0
		64	3.83e-03	30.0x	46.8	7.48e-03	59.6x	93.2
		128	3.08e-03	37.2x	29.1	5.31e-03	84.0x	65.6
1024^2	4	32	1.17e-02	32.0x	100.0	6.64e-02	32.0x	100.0
		64	7.98e-03	47.0x	73.5	3.71e-02	57.2x	89.4
		128	5.40e-03	69.5x	54.3	1.96e-02	108.5x	84.8
		256	3.91e-03	95.9x	37.4	9.23e-03	230.3x	90.0
		512	3.40e-03	110.2x	21.5	6.16e-03	345.0x	67.4
2048^2	4	32	5.03e-02	32.0x	100.0	3.26e-01	32.0x	100.0
		64	3.02e-02	53.3x	83.2	1.80e-01	58.0x	90.6
		128	1.59e-02	100.9x	78.8	8.86e-02	117.9x	92.1
		256	9.72e-03	165.5x	64.6	4.60e-02	226.9x	88.6
		512	6.30e-03	255.4x	49.9	2.42e-02	432.4x	84.4
		1024	3.83e-03	419.9x	41.0	1.01e-02	1037.1x	101.3
		2048	3.51e-03	457.8x	22.4	6.56e-03	1592.4x	77.8
4096^2	4	32	2.30e-01	32.0x	100.0	-	-	-
		64	1.33e-01	55.2x	86.2	-	-	-
		128	6.66e-02	110.4x	86.2	-	-	-
		256	3.51e-02	209.4x	81.8	2.19e-01	256.0x	100.0
		512	1.83e-02	401.3x	78.4	1.08e-01	517.7x	101.1
		1024	9.95e-03	738.6x	72.1	5.56e-02	1009.2x	98.6
		2048	6.36e-03	1155.7x	56.4	2.91e-02	1932.2x	94.3
8192^2	4	256	1.56e-01	256.0x	100.0	-	-	-
		512	7.71e-02	516.5x	100.9	-	-	-
		1024	4.00e-02	995.6x	97.2	-	-	-
		2048	1.99e-02	1999.3x	97.6	1.28e-01	2048.0x	100.0
		4096	1.11e-02	3577.2x	87.3	6.38e-02	4118.2x	100.5
		8192	7.21e-03	5523.2x	67.4	3.40e-02	7723.1x	94.3
		16384	4.12e-03	9666.5x	59.0	1.29e-02	20411.5x	124.6
16384^2	4	1024	1.78e-01	1024.0x	100.0	-	-	-
		2048	8.83e-02	2066.0x	100.9	-	-	-
		4096	4.57e-02	3991.3x	97.4	-	-	-
		8192	2.26e-02	8091.8x	98.8	1.48e-01	8192.0x	100.0
		16384	1.30e-02	14063.9x	85.8	7.36e-02	16471.7x	100.5
32768^2	4	4096	1.97e-01	4096.0x	100.0	-	-	-
		8192	9.86e-02	8188.2x	100.0	-	-	-
		16384	5.13e-02	15744.5x	96.1	-	-	-
65536^2	4	16384	2.82e-01	16384.0x	100.0	-	-	-

walltime, the walltime remain flat for a long time, which means that the communication cost grows very mildly as the number of processes increases. In standard admissibility condition cases, Figs. 7(c) and 7(d), good strong scaling is also observed in most cases. Comparing to the weak admissibility condition cases, especially towards the end of each curve, the communication cost kicks in earlier as the number of processes increase, which is due to the different prefactors in the complexity analysis in Section 4.2. Table 1 provides more evidences supporting our comments. We emphasize that the parallel efficiencies are impressive especially for larger problems. For example, in both $N=4096^2$ and $N=8192^2$ cases, parallel efficiencies are above 72 percent in weak admissibility condition cases and above 90 percent in standard admissibility condition cases, even when thousands of processes are used. Finally, we would like to comment on the weak scaling. Although not been plotted in figures, weak scaling** can be read from connecting dots vertically in figures. Clearly, on the top half of each figure, the weak scaling is near ideal (flat). Hence we claim that our algorithm and implementation give numerical results of both good strong scaling and weak scaling.

5.2 $\mathcal{H}$-matrices for three-dimensional problems

In this section, we perform numerical results for domain $\Omega=[0,1]^3$. We discretize the problem with n being $64,128,\cdots,1024$ and the corresponding matrices are of size varying from $64^3\times 64^3$ up to $1024^3\times 1024^3$. Similar as in the two-dimensional cases, we adopt random $\mathcal{H}$-matrices and random input vectors to explore the parallel scaling of our algorithm. Both weak admissibility condition and standard admissibility condition are explored as well as two choices of r. Each $\mathcal{H}$-matrix is distributed among various number of processes, from 32 up to 16384. Reported runtime is averaged over 128 random input vectors.

Comments for two-dimensional problems as in Section 5.1 apply seamless to three-dimensional problems. Both under weak and standard admissibility condition cases, strong scaling and weak scaling are well-preserved as the number of processes increases. $\mathcal{H}$-matrices under weak admissibility condition show better parallel efficiencies comparing to that under standard admissibility condition. Now we focus on the comparison of two-dimensional problems and three-dimensional problems. Comparing Fig. 7(a) and Fig. 7(b) to Fig. 8(a) and Fig. 8(b) respectively, we find that all four figures show similar strong scaling as well as weak scaling. This behavior has already been predicted by (4.11), where the complexity under weak admissibility condition is independent of the dimensionality of the problem. While, comparing Fig. 7(c) and Fig. 7(d) to Fig. 8(c) and Fig. 8(d) respectively, two-dimensional problems show better strong scaling than their three-dimensional counterparts. Under standard admissibility condition, the number of neighboring subdomains increases as the dimension increases, which also implies that the required communication cost will increase. As detailed in (4.12), the communication

**The computational cost grows quasi-linearly whereas the number of processes grows linearly. Here our weak scaling definition ignores the extra logarithmic factor.

Table 2: Numerical results of distributed-memory $\mathcal{H}$-matrix-vector multiplication for three-dimensional problems.

N	r	P	Weak			Standard		
			Time (s)	Speedup	Eff (%)	Time (s)	Speedup	Eff (%)
64^3	4	32	3.27e-03	32.0x	100.0	1.33e-02	32.0x	100.0
		64	4.11e-03	25.5x	39.8	9.94e-03	42.8x	66.8
		128	4.11e-03	25.4x	19.9	7.06e-03	60.2x	47.0
128^3	4	32	1.70e-02	32.0x	100.0	1.85e-01	32.0x	100.0
		64	1.11e-02	49.2x	76.9	1.11e-01	53.2x	83.2
		128	6.93e-03	78.6x	61.4	6.20e-02	95.4x	74.6
		256	5.70e-03	95.6x	37.3	2.66e-02	222.7x	87.0
		512	5.12e-03	106.4x	20.8	1.59e-02	372.5x	72.8
		1024	4.70e-03	115.9x	11.3	9.26e-03	638.9x	62.4
256^3	4	32	1.65e-01	32.0x	100.0	-	-	-
		64	9.20e-02	57.3x	89.6	-	-	-
		128	4.77e-02	110.6x	86.4	-	-	-
		256	2.52e-02	209.4x	81.8	-	-	-
		512	1.39e-02	380.7x	74.4	-	-	-
		1024	8.40e-03	628.4x	61.4	8.91e-02	1024.0x	100.0
		2048	5.75e-03	917.6x	44.8	3.55e-02	2572.3x	125.6
		4096	4.75e-03	1109.8x	27.1	2.02e-02	4510.0x	110.1
		8192	5.07e-03	1039.8x	12.7	1.16e-02	7894.9x	96.4
512^3	4	512	1.16e-01	512.0x	100.0	-	-	-
		1024	5.94e-02	1004.2x	98.1	-	-	-
		2048	3.00e-02	1987.9x	97.1	-	-	-
		4096	1.62e-02	3674.9x	89.7	-	-	-
		8192	9.09e-03	6561.1x	80.1	1.25e-01	8192.0x	100.0
		16384	8.29e-03	7196.6x	43.9	4.50e-02	22811.5x	139.2
1024^3	4	4096	1.42e-01	4096.0x	100.0	-	-	-
		8192	7.14e-02	8131.5x	99.3	-	-	-
		16384	3.56e-02	16320.9x	99.6	-	-	-

complexity depends monotonically on the dimension d. Hence, as proved by numerical results, the communication cost dominate the walltime earlier for bigger d.

6 Conclusion

In this paper, we introduce the data distribution of distributed $\mathcal{H}$-matrices and a distributed-memory $\mathcal{H}$-matrices-vector multiplication algorithm.

Given the tree structure of the domain organization in $\mathcal{H}$-matrix, we also organize our processes in a process tree. Two process trees are adopted for the target and source

domains. Under our data distribution scheme, the load balancing factors are constants for both weak admissibility condition (the constant is independent of dimension d) and standard admissibility condition (the constant depends on d). For problems of extremely large size N, our data distribution scheme allows the number of processes to grow as big as $\mathcal{O}(N)$. In this case, each process owns a part of the $\mathcal{H}$-matrix, whose size depends only logarithmically on N. Therefore, our data distribution is feasible for problems of extremely large sizes on massive number of processes.

The proposed distributed-memory $\mathcal{H}$-matrix-vector multiplication algorithm is parallel efficient. Specifically under our tree organizations of both processes and data, we introduce a tree communication scheme, i.e., "tree-reduce" and "tree-broadcast", to significantly reduce the latency complexity. All required computations in sequential $\mathcal{H}$-matrix-vector multiplication are evenly distributed among all processes. Importantly, our algorithm totally avoids the expensive scheduling step, which is as expensive as $\Omega(P^2)$ on P processes. Overall, our algorithm complexities for a d-dimensional problem of size N distributed among P processes are $\mathcal{O}\big(\frac{N\log N}{P}+\alpha\log P+\beta\log^2 P\big)$ and $\mathcal{O}\big(\frac{N\log N}{P}+\alpha\log P+\beta\big(\log^2 P+\log\frac{N}{P}+(\frac{N}{P})^{\frac{d-1}{d}}\big)\big)$ for weakly admissibility condition and standard admissibility condition respectively, where α denotes the latency and β denotes the per-process inverse bandwidth.

There are several future directions for improvement, both in algorithm and in implementation. Instead of pure "MPI" parallelization, one can combine "OpenMP" and "MPI" to further reduce the local communications within a node. This could improve the communication complexity, especially for $\mathcal{H}$-matrices under standard admissibility condition, by a big factor. Other $\mathcal{H}$-matrix algebraic operations can also be efficiently parallelized given our data distribution and process organization. In a companion paper, we will introduce distributed-memory $\mathcal{H}$-matrix compression, $\mathcal{H}$-matrix addition, as well as $\mathcal{H}$-matrix-$\mathcal{H}$-matrix multiplication.

Availability. The distributed-memory $\mathcal{H}$-matrix code, DMHM, is available under the GPLv3 license at `https://github.com/YingzhouLi/dmhm`. The code support both two-dimensional and three-dimensional problems.

Acknowledgments

The authors acknowledge the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing HPC resources that have contributed to the research results reported within this paper. The work of YL is supported in part by the US National Science Foundation under awards DMS-1454939 and DMS-2012286, and by the US Department of Energy via grant DE-SC0019449. The work of LY is partially supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) program and the National Science Foundation under award DMS-1818449.

References

[1] Amestoy, P., A. Buttari, I. Duff, A. Guermouche, J.-Y. L'Excellent, and B. Uçar (2011). Multifrontal method. In D. Padua (Ed.), *Encyclopedia of Parallel Computing*, pp. 1209–1216. Boston, MA: Springer US.

[2] Aminfar, A. H., S. Ambikasaran, and E. Darve (2016, Jan). A fast block low-rank dense solver with applications to finite-element matrices. *J. Comput. Phys. 304*, 170–188.

[3] Anderson, C. R. (1992, Jul). An implementation of the fast multipole method without multipoles. *SIAM J. Sci. Stat. Comput. 13*(4), 923–947.

[4] Barnes, J. and P. Hut (1986). A hierarchical $O(N\log N)$ force-calculation algorithm. *Nature 324*(6096), 446–449.

[5] Bebendorf, M. (2007, Jul). Why finite element discretizations can be factored by triangular hierarchical matrices. *SIAM J. Numer. Anal. 45*(4), 1472–1494.

[6] Bebendorf, M. (2008). *Hierarchical matrices* (1st ed.), Volume 63. Springer Publishing Company, Incorporated.

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

[8] Benson, A. R., J. Poulson, K. Tran, B. Engquist, and L. Ying (2014, Aug). A parallel directional fast multipole method. *SIAM J. Sci. Comput. 36*(4), C335–C352.

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

[10] Chen, C., H. Pouransari, S. Rajamanickam, E. G. Boman, and E. Darve (2018, May). A distributed-memory hierarchical solver for general sparse linear systems. *Parallel Comput. 74*, 49–64.

[11] Cheng, H., L. Greengard, and V. Rokhlin (1999, Nov). A fast adaptive multipole algorithm in three dimensions. *J. Comput. Phys. 155*(2), 468–498.

[12] Duff, I. S., A. M. Erisman, and J. K. Reid (1986). *Direct Methods for Sparse Matrices*. USA: Oxford University Press, Inc.

[13] Engquist, B. and L. Ying (2007, Aug). Fast directional multilevel algorithms for oscillatory kernels. *SIAM J. Sci. Comput. 29*(4), 1710–1737.

[14] Engquist, B. and L. Ying (2009). A fast directional algorithm for high frequency acoustic scattering in two dimensions. *Commun. Math. Sci. 7*(2), 327–345.

[15] Fong, W. and E. F. Darve (2009, Dec). The black-box fast multipole method. *J. Comput. Phys. 228*(23), 8712–8725.

[16] Ghysels, P., X. S. Li, F. H. Rouet, S. Williams, and A. Napov (2016, Oct). An efficient multicore implementation of a novel HSS-structured multifrontal solver using randomized sampling. *SIAM J. Sci. Comput. 38*(5), S358–S384.

[17] Grasedyck, L. and W. Hackbusch (2003, Jul). Construction and arithmetics of H-matrices. *Computing 70*(4), 295–334.

[18] Greengard, L. and W. D. Gropp (1990, Jan). A parallel version of the fast multipole method. *Comput. Math. with Appl. 20*(7), 63–71.

[19] Greengard, L. and V. Rokhlin (1987, Dec). A fast algorithm for particle simulations. *J. Comput. Phys. 73*(2), 325–348.

[20] Greengard, L. and V. Rokhlin (1997). A new version of the fast multipole method for the Laplace equation in three dimensions. *Acta Numer. 6*, 229–269.

[21] Hackbusch, W. (1999). A sparse matrix arithmetic based on $\mathcal{H}$-matrices. I. introduction to $\mathcal{H}$-matrices. *Computing 62*(2), 89–108.

[22] Hackbusch, W. and B. N. Khoromskij (2000, Dec). Sparse H-matrix arithmetic: General complexity estimates. *J. Comput. Appl. Math. 125*(1-2), 479–501.
[23] Hackbusch, W., B. N. Khoromskij, and S. A. Sauter (2000). On $\mathcal{H}^2$-matrices. In *Lect. Appl. Math.*, pp. 9–29. Springer Berlin Heidelberg.
[24] Ho, K. L. and L. Ying (2016a). Hierarchical interpolative factorization for elliptic operators: differential equations. *Commun. Pure Appl. Math. 69*(8), 1415–1451.
[25] Ho, K. L. and L. Ying (2016b, Jul). Hierarchical interpolative factorization for elliptic operators: integral equations. *Commun. Pure Appl. Math. 69*(7), 1314–1353.
[26] Izadi, M. (2012a, Jul). *Hierarchical matrix techniques on massively parallel computers*. Ph. D. thesis, Max Planck Institute for Mathematics in the Sciences.
[27] Izadi, M. (2012b, Apr). Parallel $\mathcal{H}$-matrix arithmetic on distributed-memory systems. *Comput. Vis. Sci. 15*(2), 87–97.
[28] Kriemann, R. (2005, May). Parallel $\mathcal{H}$-matrix arithmetics on shared memory systems. *Computing 74*(3), 273–297.
[29] Kriemann, R. (2013, Jun). $\mathcal{H}$-LU factorization on many-core systems. *Comput. Vis. Sci. 16*(3), 105–117.
[30] Li, X. S., J. Demmel, J. Gilbert, L. Grigori, and M. Shao (2011). Superlu. In D. Padua (Ed.), *Encyclopedia of Parallel Computing*, pp. 1955–1962. Boston, MA: Springer US.
[31] Li, Y. and H. Yang (2017). Interpolative butterfly factorization. *SIAM J. Sci. Comput. 39*(2), A503–A531.
[32] Li, Y., H. Yang, E. R. Martin, K. L. Ho, and L. Ying (2015, Jan). Butterfly factorization. *Multiscale Model. Simul. 13*(2), 714–732.
[33] Li, Y., H. Yang, and L. Ying (2015, Jan). A multiscale butterfly algorithm for multidimensional Fourier integral operators. *Multiscale Model. Simul. 13*(2), 1–18.
[34] Li, Y., H. Yang, and L. Ying (2018, May). Multidimensional butterfly factorization. *Appl. Comput. Harmon. Anal. 44*(3), 737–758.
[35] Li, Y. and L. Ying (2017). Distributed-memory hierarchical interpolative factorization. *Res. Math. Sci. 4*(12), 23.
[36] Lin, L., J. Lu, and L. Ying (2011). Fast construction of hierarchical matrix representation from matrix-vector multiplication. *J. Comput. Phys. 230*(10), 4071–4087.
[37] Minden, V., K. L. Ho, A. Damle, and L. Ying (2017, Apr). A recursive skeletonization factorization based on strong admissibility. *Multiscale Model. Simul. 15*(2), 768–796.
[38] O'Neil, M., F. Woolfe, and V. Rokhlin (2010). An algorithm for the rapid evaluation of special function transforms. *Appl. Comput. Harmon. Anal. 28*(2), 203–226.
[39] Rokhlin, V. (1985, Sep). Rapid solution of integral equations of classical potential theory. *J. Comput. Phys. 60*(2), 187–207.
[40] Rouet, F. H., X. S. Li, P. Ghysels, and A. Napov (2016, Jun). A distributed-memory package for dense hierarchically semi-separable matrix computations using randomization. *ACM Trans. Math. Softw. 42*(4), 1–35.
[41] Salmon, J. K. and M. S. Warren (1994, Jun). Fast parallel tree codes for gravitational and fluid dynamical N-body problems. *Int. J. Supercomput. Appl. High Perform. Comput. 8*(2), 129–142.
[42] Singh, J. P., C. Holt, J. L. Hennessy, and A. Gupta (1993, Nov). A parallel adaptive fast multipole method. In *Supercomput. '93Proceedings 1993 ACM/IEEE Conf. Supercomput.*, pp. 54–65.
[43] Takahashi, T., C. Chen, and E. Darve (2020, Feb). Parallelization of the inverse fast multipole method with an application to boundary element method. *Comput. Phys. Commun. 247*, 106975.
[44] Wang, R., C. Chen, J. Lee, and E. Darve (2019, Mar). PBBFMM3D: a parallel black-box algo-

rithm for kernel matrix-vector multiplication. http://arxiv.org/abs/1903.02153.
[45] Wang, R., Y. Li, M. W. Mahoney, and E. Darve (2019, Dec). Block basis factorization for scalable kernel evaluation. *SIAM J. Matrix Anal. Appl. 40*(4), 1497–1526.
[46] Wang, S., X. S. Li, J. Xia, Y. Situ, and M. V. De Hoop (2013, Dec). Efficient scalable algorithms for solving dense linear systems with hierarchically semiseparable structures. *SIAM J. Sci. Comput. 35*(6).
[47] Warren, M. S. and J. K. Salmon (1993). A parallel hashed oct-tree N-body algorithm. In *Proc. Supercomput. Conf.*, New York, New York, USA, pp. 12–21. Published by IEEE.
[48] Xia, J., S. Chandrasekaran, M. Gu, and X. S. Li (2010, Dec). Fast algorithms for hierarchically semiseparable matrices. *Numer. Linear Algebr. with Appl. 17*(6), 953–976.
[49] Xing, X. and E. Chow (2018, Nov). An efficient method for block low-rank approximations for kernel matrix systems. http://arxiv.org/abs/1811.04134.
[50] Ying, L., G. Biros, D. Zorin, and H. Langston (2003, Nov). A new parallel kernel-independent fast multipole method. In *SC '03 Proc. 2003 ACM/IEEE Conf. Supercomput.*, pp. 14.