Lattice H-Matrices on Distributed-Memory Systems

Akihiro Ida
Information Technology Center,
The University of Tokyo,
Tokyo, Japan
e-mail: ida@cc.u-tokyo.ac.jp

Abstract*— Low-rank approximation methods, such as hierarchical (H) matrices and block low-rank (BLR) matrices, can approximate dense matrices that come from scientific integral equations to reduce computational costs and memory usage. When considering the efficient use of a massive number of MPI processes on distributed-memory systems, we must balance the computational load and construct an efficient communication pattern among MPI processors. Unfortunately, the complicated structure of H-matrices prevents us from meeting these requirements. Simplifying the matrix structure is one possible approach to solve this problem, and the lattice structures found in BLR-matrices are one of the most convenient structures for this approach. However, as a trade-off, the memory usage increases from H-matrices, which use O(N log N), to BLR-matrices, which use O(N^1.5). In this study, we propose a new method called "lattice H-matrices." In short, the lattice H-matrices are constructed by utilising H-matrices as submatrices in blocks of lattice structures observed in BLR-matrices. By assigning the lattice blocks to MPI processes, we can utilise sophisticated existing parallel algorithms for dense matrices. We demonstrate how the lattice block size should be defined and confirm that the memory complexity of the lattice H-matrices remains O(N log N) when using appropriate block sizes depending on the number of MPI processors. Accordingly, the lattice H-matrices maintain the advantages of both H-matrices and BLR-matrices. We examine the efficiency of lattice H-matrices in arithmetic functions, such as H-matrix generation and H-matrix-vector multiplication, in large-scale problems on distributed-memory systems. In numerical experiments of electric field analyses, we confirmed that a relatively good load balance is maintained in the case of lattice H-matrices even if we use a large number of processes. The implementation of the lattice H-matrices exhibits a parallel speed-up that reaches as high as about 4,000 MPI processes. It is confirmed that the implementation of lattice H-matrix version is significantly faster than of normal H-matrix version for a large number of processes.

Keywords—parallel computing; low-rank approximation; hierarchical matrices; block low rank matrices

I. Introduction

The boundary element method (BEM) is widely utilised in scientific analyses. The naïve application of BEM yields linear equation systems with a dense coefficient matrix, which requires memory usage of O(N^2), where N is the number of unknowns. Although this memory usage law prevents large-scale BEM analyses, parallel computing using a distributed-memory computing system offers a solution to this problem.

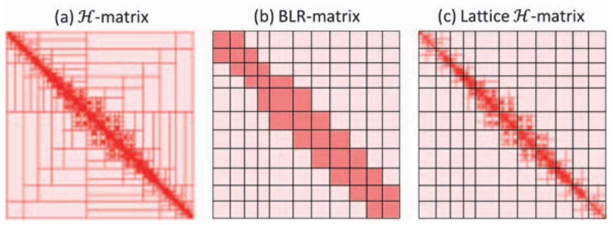

Figure 1: An $\mathcal{H}$-matrix structure constructed by normal $\mathcal{H}$-matrices (a) and its conversion to a BLR (b) and our proposed lattice $\mathcal{H}$-matrix (c). Blocks painted in deep red show dense sub-matrices and blocks in light red indicate low-rank sub-matrices.

The author's group have developed an open-source framework, named ppOpen-APPL/BEM, for use in parallel BEM analyses [1, 2]. Another technique for large-scale analyses is the use of the approximation techniques, such as Hierarchical ($\mathcal{H}$) matrices [3, 4], which reduce memory usage of the matrices from $O(N^2)$ to $O(N \log N)$. In the project developing ppOpen-APPL/BEM, we also developed an open-source, parallel $\mathcal{H}$-matrices library named $\mathcal{H}$ACApK [1, 2, 5]. The $\mathcal{H}$ACApK library provides the functions for the generation of $\mathcal{H}$ matrices and for $\mathcal{H}$-matrix-vector multiplication (HMVM) that work on distributed-memory systems. Only a few investigations of parallel $\mathcal{H}$-matrices on distributed-memory systems have been reported in the literature, even including those related to our $\mathcal{H}$ACApK library. Furthermore, most of these papers only have results up to a few dozen MPI processes. In the case of the $\mathcal{H}$ACApK library, the parallel speed-up reaches a limit near several hundreds of processes. To efficiently use a massive number of processes, we must balance the computational load and construct an efficient communication pattern. Unfortunately, the complicated structure of normal $\mathcal{H}$-matrices prevents the satisfaction of these requirements. Although simplifying the structure could solve this problem, as a trade-off, the memory usage of the $\mathcal{H}$-matrices would increase.

In addition to the approximation of dense matrices that come from integral equations in BEM analyses, $\mathcal{H}$-matrices can also be applied to the approximation of Schur complements in LU factorisation and the inversion of matrices in elliptic partial differential equations [6]. For these applications, functions related to the summation and multiplication of submatrices are required. However, realising these arithmetic functions is complicated because $\mathcal{H}$-matrices often have complex structures, as shown in Fig. 1(a). Recently, block low-rank (BLR) matrices [7] have been proposed to realise these complex arithmetic functions instead of $\mathcal{H}$-matrices. BLR-matrices are characterised by a simple, nonhierarchical, low-rank format based on lattice structures, as shown in Fig. 1(b). Their lattice structures are similar to the format of a block-divided dense matrix and allow the use of existing algorithms for dense matrices. In [8], we showed that the lattice structure is also beneficial for constructing efficient communication patterns between processes on distributed-memory systems and reducing communication costs. However, several thousands of processes were needed for problems with one million unknowns to achieve significant speed-up of matrix-vector multiplication using BLR compared to HMVM with normal $\mathcal{H}$-matrices. The reason for the large number of required processes is because the memory complexity of BLR $O(N^{1.5})$ is higher than that of $O(N \log N)$, so the BLR required much more memory than if it were solved with normal $\mathcal{H}$-matrices.

In this study, we propose a new variant of the low-rank structured matrices, called "lattice $\mathcal{H}$-matrices." As shown in Fig. 1(c), lattice $\mathcal{H}$-matrices appear to be a hybrid of $\mathcal{H}$-matrices and BLR-matrices. In fact, lattice $\mathcal{H}$-matrices are defined by introducing the lattice structure into normal $\mathcal{H}$-matrices. In other words, the lattice $\mathcal{H}$-matrices are constructed by utilising $\mathcal{H}$-matrices as submatrices in blocks of the lattice structures observed in BLR-matrices. When assigning the lattice blocks to MPI processes, we expect the lattice $\mathcal{H}$-matrices to maintain the advantages of both $\mathcal{H}$-matrices and BLR-matrices. The advantages of the proposed matrices combine the high compressibility of $\mathcal{H}$-matrices, which reduces the memory usage, and the convenience of matrix arithmetic with BLR-matrices (e.g., matrix generation, matrix-vector multiplication, matrix-matrix multiplication, matrix inversion and LU factorisation). We can then restrict ourselves to performing complex arithmetic originating from the $\mathcal{H}$-matrices structure only in serial computing or thread processing on a CPU node. In this manner, we can achieve adequate load balancing and efficient communication patterns between MPI processes on distributed-memory systems. In this article, we focus on the matrix generation and the HMVM operations because we can accomplish large-scale BEM simulations if these two arithmetic functions are provided on distributed-memory systems. In Section II, we define target problems derived from integral equations to apply the low-rank structured matrices. Furthermore, we explain our proposed lattice $\mathcal{H}$-matrices and estimate their memory complexity. In Section III, we propose parallel algorithms for the lattice $\mathcal{H}$-matrices and discuss the load balancing. We discuss the numerical experiments for the electric field analysis in Section IV. We explain the related works of this paper in Section V. The last section contains conclusions and future work.

II. Introduction to Lattice $\mathcal{H}$-matrices

A. Formulation for BEM Analyses

In this section, we describe the formulation of a system of linear equations using BEM in which the dense matrices are approximated by low-rank structured matrices. Let H be a Hilbert space of functions on a two-dimensional domain $\Omega \subset \mathbb{R}^3$ and H' the dual space of H. For $u \in H, f \in H,'$ and with a kernel function $g: \mathbb{R}^3 \times \Omega \to \mathbb{R}$, we consider the integral equation

$$\int_\Omega g(x,y)\, u(y) \mathrm{d}y = f. \tag{1}$$

To numerically calculate (1), the domain Ω is divided into elements $\Omega^h = \{\omega_j : j \in J\}$, where J is an index set. By using a weighted residual method, such as the Galerkin method, the function u is approximated from an n-dimensional subspace $H^h \subset H$. For a given basis $(\varphi_i)_{i \in \daleth}$ of H^h for an index set $\daleth := \{1, \cdots, N\}$, the approximant $u^h \in H^h$ to u can be expressed using a coefficient vector $\phi = (\phi_i)_{i \in \daleth}$ that satisfies $u^h = \sum_{i \in \daleth} \phi_i \varphi_i$. Note that the supports of the basis $\Omega^h_{\varphi_i} := \operatorname{supp} \varphi_i$ are assembled from the sets ω_j. Equation (1) is therefore reduced to the following system of linear equations.

$$A\phi = B. \tag{2}$$

In the case of the Galerkin method, entries of A are given by

$$A_{ij} = \int_\Omega \int_\Omega \varphi_i(x) g(x,y)\, \varphi_j(y) \mathrm{d}y \, \mathrm{d}x \quad \text{for all } i,j \in \daleth. \tag{3}$$

Although the coefficient matrix A is a dense matrix without explicit structure, we can find implicit structures under the following assumption. We now focus on a submatrix $A|_{s \times t}$, where the subscripts s and t denote the subsets of the row's and column's index set $\daleth$. The corresponding domains for two subsets (clusters) $s, t \in \daleth$ are defined by

$$\Omega^h_s := \bigcup_{i \in s} \operatorname{supp} \varphi_i \,, \ \Omega^h_t := \bigcup_{j \in t} \operatorname{supp} \varphi_j. \tag{4}$$

We classify a cluster pair (s,t) as 'admissible', if the Euclid distance between Ω^h_s and Ω^h_t is sufficiently large compared with their diameters:

$$\min\{\operatorname{diam}(\Omega^h_s),\ \operatorname{diam}(\Omega^h_t)\} \leq \eta \operatorname{dist}(\Omega^h_s,\ \Omega^h_t), \tag{5}$$

where η is a positive constant number depending on the kernel function g and the division Ω^h. On the domain corresponding to the admissible pairs, $x \in \Omega^h_s, y \in \Omega^h_t$, we assume that the kernel function can be approximated with a certain accuracy by a degenerate kernel such as

$$g(x,y) \cong \sum_{r=1}^{k} g^r_1(x) g^r_2(y), \tag{6}$$

where k is a positive number. Under the assumption, the double integral in (3) is separated into two single integrals with respect to x and y. For the dense submatrix $A|_{s \times t}$ corresponding to the far-remote interaction between Ω^h_s and Ω^h_t, its approximation $\tilde{A}|^k_{s \times t}$ can be represented in the form of the following low-rank approximation (LRA):

$$\begin{gathered} A|_{s\times t} \cong \widetilde{A}|^k_{s\times t} = VW, V \in \mathbb{R}^{s \times k}, W \in \mathbb{R}^{k \times t}, \\ V_{ir} = \int_\Omega \varphi_i(x) g^r_1(x)\, \mathrm{d}x,\ W_{rj} = \int_\Omega g^r_2(y) \varphi_j(y)\, \mathrm{d}y. \end{gathered} \tag{7}$$

If we have knowledge of the concrete form of the kernel function, we could carry out the mathematical computation of the power series expansion and integrals by hand. It is sometimes difficult or very complicated to obtain the form of such a degenerate kernel. For these problems, algebraic LRA methods, such as adaptive cross approximation (ACA) [9] are available.

B. Partition of Low-rank Structured Matrices and Accuracy

As we confirmed in II.*A*, the matrix A in (2) includes submatrices $A|_{s\times t}$ that could be approximated by low-rank matrices as $\tilde{A}|^{k}_{s\times t}$ in the form (7), if the block $s\times t$ was appropriately chosen in $\beth\times\beth$. As the set of subsets of $\beth\times\beth$, we here introduce a partition structure M such that the blocks $m(:=s_m\times t_m)\in M$ do not overlap one another and they fill up the square $\beth\times\beth$ without gaps. The partition structure M can represent conceptual diagrams of low-rank structured matrices (Fig.1). Then, we find that M consists of M_R, which represents the set of blocks corresponding to low-rank submatrices, and M_D, whcih represents the set of blocks corresponding to dense submatrices, i.e. $M_R\cup M_D=M$. For a block $m\in M_D$, the corresponding submatrix $\tilde{A}|^{D}_{s_m\times t_m}$ of the approximated matrix $\tilde{A}$ is taken to coincide with the original submatrix $A|_{s_m\times t_m}$, i.e. $\tilde{A}|^{D}_{s_m\times t_m}$ is a dense submatrix and no approximation error occurs. For all $m\in M_R$ and a given error tolerance $\varepsilon\in\mathbb{R}_{>0}$, we assume the corresponding submatrices $A|_{s_m\times t_m}$ are approximated by $\tilde{A}|^{k_m}_{s_m\times t_m}$ satisfying the following expression:

$$\left\|A|_{s_m\times t_m}-\tilde{A}|^{k_m}_{s_m\times t_m}\right\|_F \Big/ \left\|\tilde{A}|^{k_m}_{s_m\times t_m}\right\|_F \leqq \varepsilon, \qquad (8)$$

where $\|\cdot\|_F$ denotes the Frobenius norm. Then, for the approximated matrix $\tilde{A}$ with M, we can prove $\left\|A-\tilde{A}\right\|_F/\left\|\tilde{A}\right\|_F\leqq\varepsilon$ [9]. Although the approximated matrices with different partition structures would have different number of block and entries, it is guaranteed that they have the required accuracy. We have many alternatives of partition structures.

C. Construction of Lattice $\mathcal{H}$-Matrices

Among possible partition structures in $\beth\times\beth$, a structure $M_{\mathcal{H}}$ constructed by using $\mathcal{H}$-matrices is one of the most efficient structures such that there exist many large, low-rank submatrices. By adding conditions for the contraction procedure of $M_{\mathcal{H}}$ by$\mathcal{H}$-matrices, we can construct most of low-rank structured matrices including BLR-matrices and lattice $\mathcal{H}$-matrices, i.e. they can be regarded as special cases of $\mathcal{H}$-matrices. Because we define lattice $\mathcal{H}$-matrices by introducing the lattice structure of BLR-matrices into the $\mathcal{H}$-matrices, we first describe the construction procedures of normal $\mathcal{H}$-matrices and BLR-matrices before introducing the lattice $\mathcal{H}$-matrices.

In the $\mathcal{H}$-matrices, a cluster tree $T_\beth$ is first generated by using the so-called clustering method based on geometrical information associated with the index set $\beth$ (Fig. 2). The index set is divided into subsets(sons) $\beth^d_i$ at each depth d such that $\beth=\sum_i\beth^d_i$. The clustering process continues until each cluster size $|\beth^d_i|$ becomes sufficiently small. Typically, the cluster size at the deepest level D, $|\beth^D_i|$, is equal to 15. A block cluster tree $T_{\beth\times\beth}$ is created from $T_\beth$ by defining that the node of $T_{\beth\times\beth}$ at each depth d is a block $\beth^d_i\times\beth^d_j$, where $\beth\times\beth=\bigcup_{i,j}\beth^d_i\times\beth^d_j$. Considering the truncation of branches of $T_{\beth\times\beth}$ according to the condition that $\beth^d_i\times\beth^d_j\in T_{\beth\times\beth}$ is an admissible cluster pair, the leaves $\mathcal{L}(T_{\beth\times\beth})$ then represent a partition structure on $\beth\times\beth$. This truncation is performed in the direction from the root to the leaves while checking the admissible condition (5) at each node of $T_{\beth\times\beth}$ (Fig. 3). We adopt the partition structure $\mathcal{L}(T_{\beth\times\beth})$ as $M_{\mathcal{H}}$ which defines a structure in a normal $\mathcal{H}$-matrix.

A partition structure $M_{\mathrm{B}}$for BLR-matrices is regarded as a simplified partition structure of normal $\mathcal{H}$-matrices. To construct a BLR structure, we do not need the block cluster tree $T_{\beth\times\beth}$, unlike the case with $\mathcal{H}$-matrices. We simply truncate all branches of the cluster tree $T_\beth$ at a depth level B and take the induced blocks in $\beth\times\beth$ as $M_{\mathrm{B}}:=\left\{\beth^B_{\mathrm{i}}\times\beth^B_{\mathrm{j}}\;\middle|\;\beth^B_{\mathrm{i}},\beth^B_{\mathrm{j}}\in\mathcal{L}(T_\beth)\right\}$. The verification of the admissible condition is performed on each block $\beth^B_i\times\beth^B_j$ after the structure is defined (Fig. 4).

For lattice $\mathcal{H}$-matrices, we utilise construction procedures for both normal $\mathcal{H}$-matrices and BLR-matrices. The construction procedure for the partition structure of lattice $\mathcal{H}$-matrices consists of the following steps (Fig. 5):

1. Construct a cluster tree $T_\beth$ for the index set $\beth$.
2. Set a lattice depth L, and truncate all branches of $T_\beth$ at L.
3. Construct lattice structure $M_L=\left\{\beth^L_{\mathrm{i}}\times\beth^L_{\mathrm{j}}\;\middle|\;\beth^L_{\mathrm{i}},\beth^L_{\mathrm{j}}\in\mathcal{L}(T_\beth)\right\}$.
4. For each lattice block $\beth^L_i\times\beth^L_j\in M_L$, check the admissibility.
5. For non-admissible lattice blocks $\beth^L_i\times\beth^L_j\in M_L$, make a block cluster tree $T_{\beth^L_i\times\beth^L_j}$.
6. Truncate $T_{\beth^L_i\times\beth^L_j}$ using the admissible condition, and obtain an $\mathcal{H}$-matrix structure $m^{L_{ij}}_{\mathcal{H}}$ for a block $\beth^L_i\times\beth^L_j$.

We obtain a lattice $\mathcal{H}$-matrix by calculating low-rank submatrices for admissible blocks detected in step 4 and step 6, and dense submatrices for the remaining blocks. The depth L, which defines the lattice block size, has great influence on the memory usage, which is discussed in the next subsection.

A lattice $\mathcal{H}$-matrix with the same structure can be constructed by just splitting the low-rank submatrices of the corresponding normal $\mathcal{H}$-matrix. This means that all the information of the normal $\mathcal{H}$-matrix would be expected to be possessed in the corresponding lattice $\mathcal{H}$-matrices. Therefore, if normal $\mathcal{H}$-matrices are applicable for a problem, lattice $\mathcal{H}$-matrices would be also applicable for the problem.

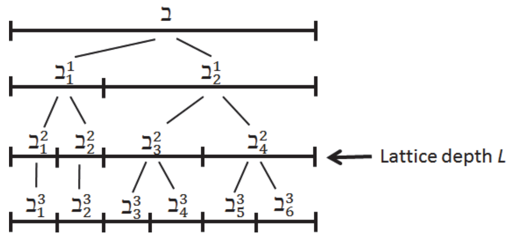

Figure 2: Conceptual diagram of the cluster tree $T_\beth$ made of the index set $\beth$.

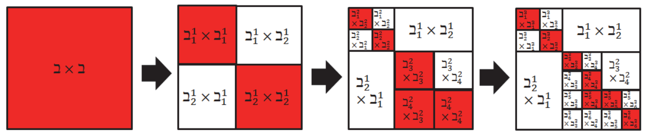
$\beth\times\beth$
$\beth^1_1\times\beth^1_1$
$\beth^1_1\times\beth^1_2$
$\beth^1_2\times\beth^1_1$
$\beth^1_2\times\beth^1_2$

Figure 3: Conceptual diagram of the block cluster tree $T_{\beth\times\beth}$ with truncation of admissible branches, which creates a normal $\mathcal{H}$-matrices structure. White painted blocks show submatrices judged as low-rank, and red painted blocks indicate submatrices judged as full rank.

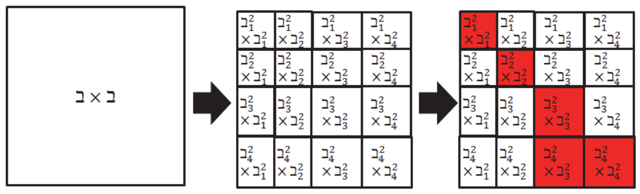
Figure 4: Conceptual diagram of the construction of BLR-matrices.

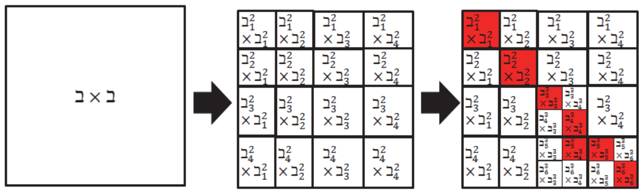
Figure 5: Conceptual diagram of the construction of lattice $\mathcal{H}$-matrices.

D. *Memory Complexity of Lattice $\mathcal{H}$-Matrices*

For normal $\mathcal{H}$-matrices, the theoretical complexity has been previously well investigated in [3, 4], and their memory complexity is O($N \log N$). On the other hand, for BLR-matrices, the memory complexity is O($N^{1.5}$) [8]. The memory usage of lattice $\mathcal{H}$-matrices is expected to fall somewhere between normal $\mathcal{H}$-matrices and BLR-matrices. In this section, we estimate the memory complexity of lattice $\mathcal{H}$-matrices.

We here consider the checking process for the admissible condition in step 4 of the construction procedure of lattice $\mathcal{H}$-matrices described in II.*C*. The diagonal lattice blocks must be non-admissible because they include the interaction between a cluster and itself. The blocks next to (and near) diagonals would also be non-admissible if we consider a high dimensional problem. In contrast, we could suppose that far off-diagonal lattice blocks are admissible because they correspond to far-remote interactions. Even if this is not the case, the corresponding $\mathcal{H}$-submatrix would have a shallow depth compared with a diagonal one, and contain only low-rank submatrices. Therefore, to simplify the estimation, we assume that far off-diagonal blocks on the lattice are low-rank submatrices. Furthermore, it is assumed that all lattice blocks have square shapes, all lattice blocks are the same size and all low-rank submatrices are rank-1 matrices, that is, $|\beth_i^L|=|\beth_j^L|\equiv l$ and $k_m \equiv 1$ (see Fig. 6). Moreover, we assume that the number of non-admissible lattice blocks is $\gamma N (1 \le \gamma \le N/l)$, and memory usage of sub-matrices on non-admissible lattice blocks are the same size $g(l)$. In the case of Fig.1(b), (c) and Fig.6, $\gamma \cong 3$. Each rank-1 matrix of the off-diagonal blocks on the lattice has $2l$ elements. The number of lattice blocks along a side must be N/l. Accordingly, the dependency of memory usage on the block size l is given by the following function:

$$f(l) = g(l)\gamma N/l + 2(N^2/l - \gamma N). \tag{9}$$

In the case of BLR-matrices, $f(l) = \gamma l N + 2(N^2/l - \gamma N)$ because all non-admissible blocks are represented as dense matrices; $g(l)=l^2$. At point $l_{\min} = \sqrt{2N/\gamma}$, the function $f(l)$ assumes the minimum value $f(l_{\min}) = 2\sqrt{2\gamma N^3} - 2\gamma N$. Thus, it is confirmed that the memory complexity of BLR is O($N^{1.5}$).

According to [10], in the case of $\mathcal{H}$-submatrices with assumptions above, $g(l) = 2l \log l + l$, the function $f(l)$ is given by

$$f(l) = 2\gamma N \log l + 2N^2/l - \gamma N. \tag{10}$$

For the range of lattice block sizes $1 \le l \le N$, the function $f(l)$ is monotonically increasing. Additionally, the function takes a minimum value at the point $l = N$ that coincides with a normal $\mathcal{H}$-matrix and a maximum value at point $l = 1$ that corresponds to a dense matrix. For $0 < \alpha \le 1$, by substituting $l = \alpha N$ into (10), we obtain

$$f(\alpha N) = 2\gamma N \log N + N(2\gamma \log \alpha + 2/\alpha - \gamma). \tag{11}$$

The lattice block size l (and α) should depend on the number of MPI processors N_p used, but not on the matrix size N. For a given N_p, we can assume that α is constant. Then, we determine that the memory complexity for lattice $\mathcal{H}$-matrices is O($N \log N$).

We assume that the lattice structure is utilised for communications among MPI processors. If serial computing is expected, the lattice structure is of no relevance; here, we should set $\alpha = 1$, which corresponds to a normal $\mathcal{H}$-matrix. In Section IV. *C*., we will discuss the proper block size of the lattice after the assignment of tasks to MPI processes, as shown in Section III.

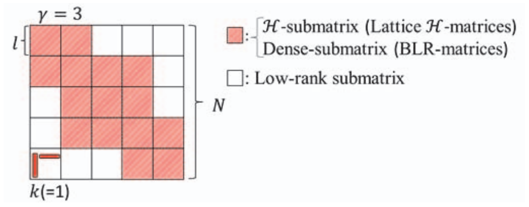
$\gamma = 3$
l
N
$k(=1)$
🟥: $\mathcal{H}$-submatrix (Lattice $\mathcal{H}$-matrices)
Dense-submatrix (BLR-matrices)
□: Low-rank submatrix

Figure 6: Conceptual image of a lattice $\mathcal{H}$-matrix simplified for the estimation of memory complexity. Blocks with red stripe show non-addmissible blocks and white painted blocks indicate addmissible blocks.

III. Parallelisation of Lattice $\mathcal{H}$-Matrices Arithmetic

A. *Assignment of Tasks to MPI Processes*

We now consider dividing the sets $M_{\mathcal{H}}$ for normal $\mathcal{H}$-matrices and M_L for the lattice structure of lattice $\mathcal{H}$-matrices introduced in Sections II.*B* and II.*C* among N_pMPI processes. In our algorithms for normal $\mathcal{H}$-matrices, arithmetic is performed block by block in units of block $m \in M_{\mathcal{H}}$, which corresponds to a submatrix. In the case of lattice $\mathcal{H}$-matrices, an element $m_{\mathcal{H}}^{L_{ij}} \in M_L$ indicates a lattice block including blocks or a block containing a low-rank submatrix. We assign tasks to an MPI processor in groups according to the lattice block $m_{\mathcal{H}}^{L_{ij}}$ for the parallelisation of lattice $\mathcal{H}$-matrices arithmetic. Hereafter, for notational simplicity, we will denote $M_{\mathcal{H}}$ and M_L by M, a block $m \in M_{\mathcal{H}}$ and a lattice block $m_{\mathcal{H}}^{L_{ij}} \in M_L$ by m if there is no reason to distinguish them.

The assignment for normal $\mathcal{H}$-matrices is discussed in [5], which is utilised in the $\mathcal{H}$ACApK open-source library [1, 2]. As shown in Fig. 7, the set of blocks is sliced along the row direction, but blocks are not separated. As a result, a portion

M_p is assigned to an MPI process, where p is the MPI process number. This assignment strategy minimises both the maximum height of a portion $R(M_p)$ and the load imbalance among the MPI processes as much as possible.

For BLR-matrices, we discussed the use of a so-called 2D cyclic assignment strategy in [8]. We adopt the same strategy here for lattice $\mathcal{H}$-matrices. As shown in Figs. 1(c) and 6, the lattice $\mathcal{H}$-matrices remain a structure that is conceptually similar to a block-divided dense matrix. To utilise the existing algorithms for dense matrices, we here introduce the concept of a "process grid" used in ScaLAPACK [11], which is a library of high-performance linear algebra routines for distributed-memory systems. In this technique, the process grid has N_{pr} processes in a row and N_{pl} processes in a column, such that $N_{pr} \times N_{pl} = N_p$. An MPI process can be referenced by its row and column coordinates $P(P_r, P_l)$ within the grid. By arraying the process grid on the lattice, cyclically in the row and column direction, the set M_L of lattice blocks is divided into N_p subsets. Figure 8 demonstrates a case using nine MPI processes with a process grid of $N_{pr} = 3$ and $N_{pl} = 4$. The different colours represent the lattice blocks assigned to different MPI processors. In contrast to the assignment for normal $\mathcal{H}$-matrices shown in Fig. 7, blocks of a given colour are distributed in different locations in the entire matrix. Therefore, a subset M_p is composed of discontinuous blocks.

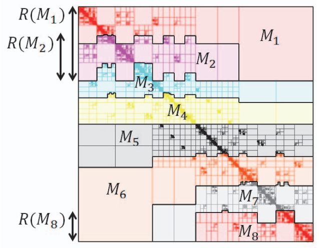

Figure 7: Assignment of blocks to MPI processes for normal $\mathcal{H}$-matrices [4]. The set M, which defines the frame of blocks or submatrices, is divided into portions M_k comprising continuous blocks. The different colours represent the blocks assigned to different MPI processors. The shape of the block of submatrices, M_p, is like a bar or a "key".

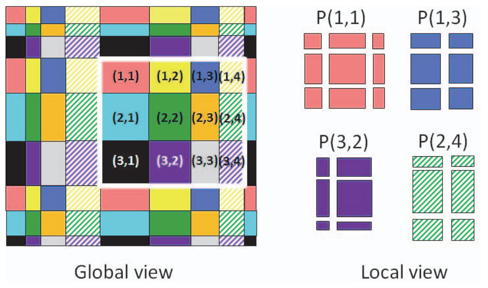

Figure 8: Assignment of blocks to MPI processes for the lattice $\mathcal{H}$-matrices. The 2D cyclic assignment strategy based on the process grid is adopted. Each MPI process $P(P_r, P_l)$ performs arithmetic functions on blocks painted by the same colour.

B. Construction of Lattice $\mathcal{H}$-Matrices in Parallel

The construction procedure of lattice $\mathcal{H}$-matrices is conceptually the same steps as that of normal $\mathcal{H}$-matrices:

1. Construct a cluster tree $T_\beth$ for index set $\beth$.
2. Construct a partition M using $T_\beth$.
3. Fill in the all submatrices using a LRA.

In the proposed method, we only parallelise the third step and redundantly perform the first and second steps for the structure construction in all MPI processes. Each MPI processor independently performs LRA on the assigned blocks without any MPI communication.

C. Lattice $\mathcal{H}$-Matrix-Vector Multiplication

Matrix-vector multiplication is the most frequently used arithmetic in BEM simulations. We here consider the algorithm for calculating a vector $y \in \mathbb{R}^\beth$ that is the result of an HMVM

$$y := \tilde{A}x \tag{12}$$

for a given vector $x \in \mathbb{R}^\beth$ and (lattice) $\mathcal{H}$-matrix $\tilde{A}$ with a structure M divided into subsets M_p. In actual applications, the HMVM is carried out multiple times, sometimes more than ten thousand times in some applications. Under our assumptions, the p-th MPI processor has sections of the (lattice) $\mathcal{H}$-matrix $\tilde{A}_{M_p} := \{\tilde{A}|_{s_m \times t_m} : m \in M_p\}$. For an HMVM in an MPI process p, $\tilde{A}_{M_p}$is operated with x_p, which is a part of the entire vector x, and the result y_p is generally a part of the entire vector y. To perform the next HMVM using the result of the previous HMVM, certain MPI communications are required to obtain x_p. For normal $\mathcal{H}$-matrices, the parallel algorithms to obtain the required x_p are described in our previous paper [4]. In the algorithms, we define $x_p \equiv x$ because each MPI processor has a 1D-sliced portion of the entire matrix, as shown in Fig. 7. To realise the situation in which all MPI processors have the entire result vector y, which is x_p in the next step, MPI communications are performed via the MPI_iSEND and MPI_iRECV functions between MPI processors that have adjacent assigned portions of submatrices.

We introduce two parallel algorithms for the lattice $\mathcal{H}$-matrices with the 2D cyclic assignment introduced in Section III. *A*. These algorithms are adaptation of the PUMMA [12] which is an algorithm of matrix multiplication for dense matrices on distributed-memory systems. An overview of the first algorithm is shown in Fig. 9. After arithmetic of multiplication $\tilde{A}|_{s_m \times t_m} \cdot x_p$ in each MPI process, a process obtains sections of the result vector by using the MPI_ALLREDUCE function in each row of the process grid. To obtain the entire vector y, MPI processes belonging to the same column in the process grid exchange their partial vectors, using the MPI_iSEND and MPI_iRECV functions. We here call this algorithm "LatticeHMVM_fullVector", because all MPI processes have the entire vector y as the next x_p. We introduce the second algorithm, which we call "LatticeHMVM_partVector." An overview of the algorithm is shown in Fig. 10. We here restrict ourselves to a square-shaped process grid, which means that $N_{pr} = N_{pl} = \sqrt{N_p}$. In this situation, each MPI process does not need to possess the entire vector y as the next x_p. The required x_p is common in each column of the process grid. It is convenient to gather information into MPI processes located along the diagonal of the process grid because diagonal submatrices on the entire matrix must be assigned to the diagonal MPI processors and diagonal submatrices have a square shape as $\beth_i^L \times \beth_i^L$. Therefore, diagonal MPI processes obtain the required x_p by using the MPI_REDUCE function along each row in the process grid and then send it to other MPI processes in each column in the process grid using the MPI_BROADCAST function. If the user's code permits the HMVM function in the $\mathcal{H}$-matrices library to return only a part of the result vector, LatticeHMVM_partVector is potentially a very beneficial algorithm because the communication traffic is significantly reduced from the LatticeHMVM_fullVector.

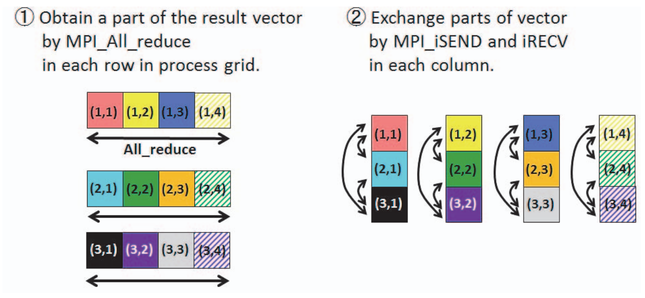
① Obtain a part of the result vector by MPI_All_reduce in each row in process grid.
② Exchange parts of vector by MPI_iSEND and iRECV in each column.
(1,1) (1,2) (1,3) (1,4)
All_reduce
(2,1) (2,2) (2,3) (2,4)
(3,1) (3,2) (3,3) (3,4)

Figure 9: Overview of the algorithm LatticeHMVM_fullVector, which is a parallel HMVM for lattice $\mathcal{H}$-matrices.

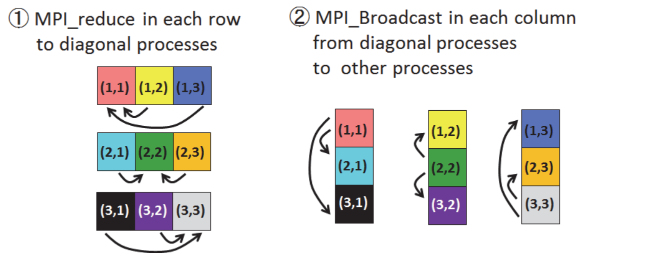
① MPI_reduce in each row to diagonal processes
② MPI_Broadcast in each column from diagonal processes to other processes
(1,1) (1,2) (1,3)
(2,1) (2,2) (2,3)
(3,1) (3,2) (3,3)

Figure 10: Overview of the algorithm LatticeHMVM_partVector, which is available only if the shape of process grid is square.

D. *Relation between the Load Balance and the Shape of the Process Grid*

The 2D cyclic assignment introduced in Section III. *A*. is known for efficient strategy for dense matrices as it is utilised in ScaLAPACK. However, when applying the method to lattice $\mathcal{H}$-matrices, it is concerning that the load balance may worsen depending on the shape of the process grid. This load imbalance may be caused by a maldistribution of dense and $\mathcal{H}$-submatrices across the lattice locations and the difference in rank between low-rank submatrices. According to the nature of $\mathcal{H}$-matrices, the dense and $\mathcal{H}$-submatrices concentrate around the diagonal part of the lattice. In terms of the ranks of the low-rank submatrices, submatrices near the diagonal are likely to have larger ranks than far off-diagonal submatrices. If the shape of the process grid is square, an MPI process located on the diagonal part of the process grid must continue to be assigned diagonal lattice blocks, and an MPI process located at an off-diagonal part of the process grid would be never assigned diagonal lattice blocks. Figure 11 shows an example using a 4×4 process grid. In this case, the MPI process $P(1,1)$ is assigned many dense and $\mathcal{H}$-submatrices, whereas the MPI process $P(1,4)$ is only assigned low-rank submatrices.

To prevent the above-mentioned load imbalance, we can use a process grid with a rectangular shape. Considering the amount of communication in the LatticeHMVM_fullVector, the rectangle should be as close to square as possible, such as $N_{pr} = N_{pl} + 1$ (which we will call "Rectangle1") or $N_{pr} = N_{pl} + 2$ (which we will call "Rectangle2"). When we use the rectangular process grid, the LatticeHMVM_partVector is not available even if the application code permits the vector distribution. For practical examples, the load balances in the cases of process grids with shapes square, Rectangle1 and Rectangle2 are subsequently examined in Section IV. *D*.

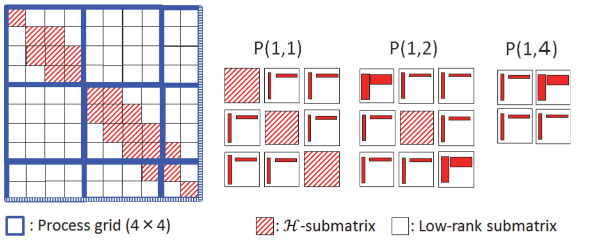

Figure 11: Conceptual image of distribution of $\mathcal{H}$- submatrices on the lattice. The square-shaped process grid results in load imbalance among processes.

IV. Numerical Experiments

A. *Implementation and Computation Environment*

We will now discuss the performance of parallel algorithms for normal and lattice $\mathcal{H}$-matrices. For normal $\mathcal{H}$-matrices, we used the flat-MPI version of the $\mathcal{H}$ACApK library. For lattice $\mathcal{H}$-matrices, we reimplemented functions of the $\mathcal{H}$ACApK. For the common parts of these $\mathcal{H}$-matrices, such as LRA or submatrix-vector multiplication, the functions in the $\mathcal{H}$ACApK library were used in the new implementation where possible. Throughout the numerical experiments, we use the ACA as an LRA by setting the error tolerance ε=1.0e-4 in (8).

As a test problem, we have selected the static electric field analyses described in the next subsection. We evaluated the parallel scalability when constructing the (lattice) $\mathcal{H}$-matrices and performing the HMVM. All calculations were carried out using the Fujitsu FX10 at Tokyo University, which is equipped with SPARC64™ IXfx and 32 GB memory. The FX10 system utilises Tofu for the interconnect network, which has a link throughput of 5 GB/s. We used a FUJITU Fortran compiler with the -Kfast optimisation option and a FUJITU MPI library. We adopted a flat-MPI programming model and used the same number of MPI processes as cores on a node where possible.

B. *Calculation Models and Constructed $\mathcal{H}$-Matrices*

In this subsection, we introduce an electrostatic field problem concerning perfect conductors with examples of target matrices for (lattice) $\mathcal{H}$ -matrices approximation. When applying BEM to the problem, we use the formulation described in Section II.*a* with the kernel function $g(x,y)$ in (1) given by $g(x,y) = (4\pi\epsilon|x-y|)^{-1}$. Here ϵ denotes the electric permittivity. We divide the surface of the conductors into triangular elements and use step functions for the base function φ_i of BEM. The conductors are placed in a uniform electric field in the *z*-direction in 3D space with 0[V] at the ground; then the induced surface charge on the conductors is calculated. To solve the linear system (2), we adopt the BiCGSTAB method which is an iterative method using HMVM. The convergence criterion of BiCGSTAB is that the relative residual with 2-norm becomes less than 1.0e-4.

To investigate the dependency of N on the memory usage of lattice $\mathcal{H}$-matrices, we present a series of examples in which the conductor has the shape of a humanoid, as shown in Fig. 12 (left). The humanoids are placed standing on the ground, and they are arrayed on a $q \times q$ grid with $q = 1, 2, \cdots 10$. Then, the numbers of equations are N = 19664, 78656, 176976, 314624, 491600, 707904, 963536, 1258496, 1592784 and 1966400 for each q, respectively. Figure 12 (left) shows the case with $q = 5$, and Fig. 12 (right) shows the corresponding $\mathcal{H}$-matrix structure constructed from normal $\mathcal{H}$-matrices. As can be seen in the figure on the right, there are many small blocks spread over the entire matrix. Furthermore, to investigate the dependency of the partition structure on the parallel scalability, we present three more examples which have different aspects of the partition structure as shown in Fig. 13. They are named "Humanoid line 30", "Dipole" and "Spheres 50", respectively. For these examples, the numbers of equations are N = 589920, 676000 and 1188000.

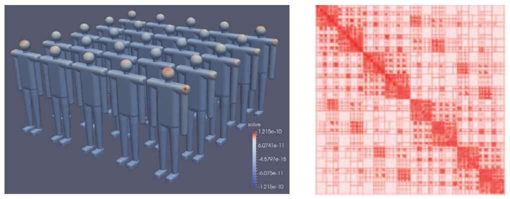

Figure 12: Analytical objects in the shape of humanoids (left) and corresponding $\mathcal{H}$-matrix structure (right) constructed from normal $\mathcal{H}$-matrices for "humanoids 5×5". Deep-red blocks show dense submatrices and light-red blocks indicate low-rank submatrices.

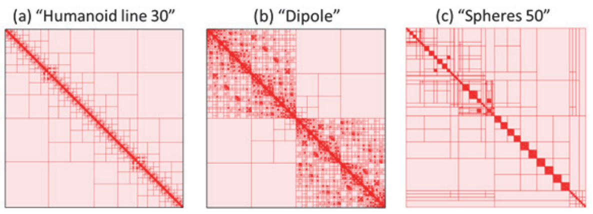

Figure 13: $\mathcal{H}$-matrix structures constructed from normal $\mathcal{H}$-matrices.

C. Appropriate Number of Blocks on Lattice

In this subsection, we discuss the proper block size of the lattice. A block size that is as large as possible is desirable to reduce the memory usage of the lattice $\mathcal{H}$-matrices. In contrast, a small block size, which results in a large number of lattice blocks, is better for the load balancing when we adopt the assignment strategy introduced in Section III.*A*. For the problem of "humanoids 5×5" explained in the previous subsection, we observed the memory usage of the lattice $\mathcal{H}$-matrices and their construction time using 64, 144, 256 and 576 MPI processors while varying $\sqrt{N_L}$, which is the number of lattice blocks in the row (and column) of the lattice, from 1 to 1506. The results are shown in Table I. Although the memory usage monotonously increases as the number of blocks increases, the construction times take minimum values near the middle of the range due to the load imbalance and the changing computational amount. From the results, we can confirm that the proper block size depends on the number of processors to minimise the calculation time. To obtain a good load balance, if we assume that a colour corresponding to an MPI process must appear c times in a row of the lattice, the parameter α in (11) should be set to $\alpha = 1/(c\sqrt{N_p})$, where N_p indicates the number of MPI processes. Hereafter, all the experiments on the lattice $\mathcal{H}$-matrices are performed under the condition where a lattice block size is set to $c = 15$.

TABLE I. Memory usage of lattice $\mathcal{H}$-matrices and construction time when varying $\sqrt{N_L}$, the number of lattice blocks in a row, for the "humanoids 5×5" model with N = 491,600. The case $\sqrt{N_L}$ = 1 corresponds to the normal $\mathcal{H}$-matrix.

$\sqrt{N_L}$	Memory [MB]	Construction time of lattice $\mathcal{H}$-matrices[s]			
		576MPI	256MPI	144MPI	64MPI
1	13177	—	—	—	—
32	13277	106.6	147.0	173.4	222.2
96	14748	53.8	79.2	103.2	174.2
196	17393	49.6	84.4	113.5	185.8
396	21028	46.1	76.3	114.6	209.4
752	29318	49.8	85.0	132.9	248.1
1506	49392	67.2	119.2	193.6	386.0

D. Load Balance

To quantify the load balance between the MPI processes, we introduce the load-balancing efficiency E when using N_p MPI processes:

$$E := \frac{\mu}{\mu_{\text{pmax}} N_p} \tag{13}$$

where μ denotes the memory usage of the entire matrix and μ_{pmax} is the largest memory assigned to an MPI process. If all MPI processes have the same size of assignment, $\mu_{\text{pmax}} = \mu/N_p$, and $E = 1$, which indicates a perfect load balance. Larger μ_{pmax} values will cause load imbalance, and the efficiency E will approach zero.

We expect the 2D cyclic assignment for the lattice $\mathcal{H}$-matrices to improve the load balance from the quasi-1D-sliced assignment for normal $\mathcal{H}$-matrices. As discussed in Section III. *D.*, the shape of the process grid influences the load balance. We evaluate the load-balancing efficiency E for normal $\mathcal{H}$-matrices and the lattice $\mathcal{H}$-matrices with the three types of process grid introduced previously (i.e., square, Rectangle1 and Rectangle2). In Fig. 14, the observed results for the problems introduced in IV.*B* are shown as a function of the number of processors N_p. Although normal $\mathcal{H}$-matrices show high efficiencies for small numbers of processes, the performance rapidly decays as the number of processes increases. In contrast, the efficiencies for the cases with lattice $\mathcal{H}$-matrices do not rapidly change regardless of the number of processes. It is notable that the efficiencies with the square-shape process grids are lower than those of the rectangular process grids. Because there is no significant difference between the Rectangle1 and Rectangle2 cases, we use the process grid from Rectangle1 when constructing the lattice $\mathcal{H}$-matrices and performing the algorithm LatticeHMVM_fullVector in experiments in the following subsections.

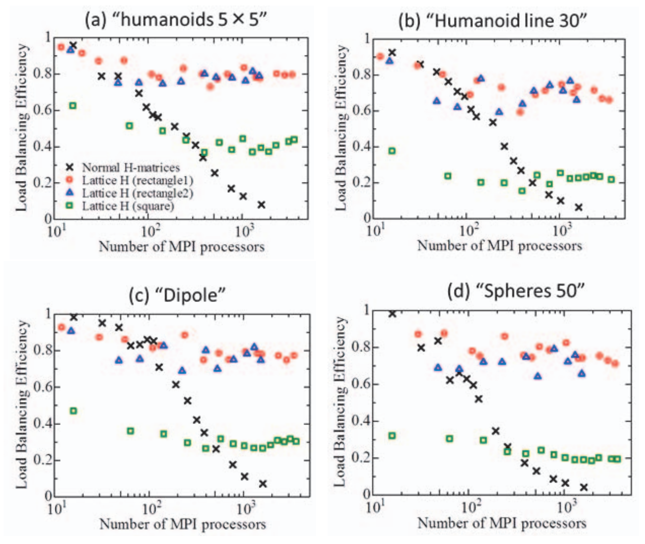

Figure 14: Load-balancing efficiency E

E. Memory Usage

For the example problems introduced in IV.B, we observe the memory usage of the lattice $\mathcal{H}$-matrices when varying the number of MPI processes from 1 to 3,422, and the results are given in Fig. 15. The case with a single process corresponds to the use of normal $\mathcal{H}$-matrices. We can see from the figure that too many MPI processors cause a rapid increase in memory usage in all problems. The acceptable number of MPI processes depends on the number of equations N.

To investigate the dependency of N on the memory usage of lattice $\mathcal{H}$-matrices, we fix the number of MPI processors to 36 and 576. The models of the humanoid series are used for this investigation. Figure 16 shows the results with the case of normal $\mathcal{H}$-matrices. We can see from the figure that the difference in memory usage between normal $\mathcal{H}$-matrices and the lattice $\mathcal{H}$-matrices with 36 and 576 processors becomes small as N increases. Moreover, we can confirm in Fig. 16 that the memory complexity of the lattice $\mathcal{H}$-matrices is $O(N \log N)$ if we fix the number of the used MPI processors.

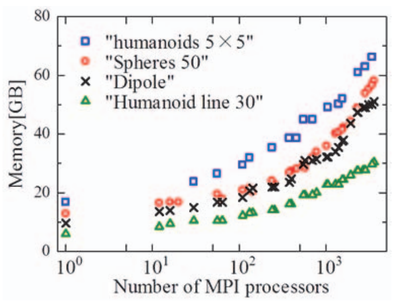

Figure 15: Dependency of N_p on memory usage of lattice $\mathcal{H}$-matrices.

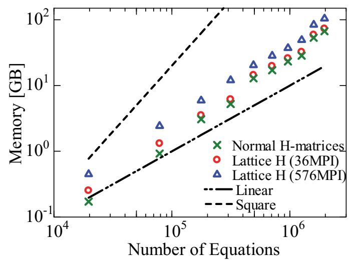

Figure 16: Memory usage of normal and lattice $\mathcal{H}$-matrices.

F. Performance Evaluation when Generating Matrices

We investigate the execution times when performing the third step in the $\mathcal{H}$-matrices construction procedure explained in Section III. B. Figure 17 shows the observed results as a function of the number of used MPI processors. The blue lines indicate the case with normal $\mathcal{H}$-matrices, and the red lines represent the lattice $\mathcal{H}$-matrices in Fig. 17.

The speed-up of the normal $\mathcal{H}$-matrices reaches a limit at approximately 300 processors for "humanoids 5×5," 100 processors for "Spheres 50," and 200 processors for "humanoid line 30" and "Dipole." The reason for this limitation is the load imbalance because MPI communications are not needed. In contrast, we observe a speed-up that is as high as 3,400 cores for the case with lattice $\mathcal{H}$-matrices. Because the calculation overhead of lattice $\mathcal{H}$-matrices is two to three times larger than that of normal $\mathcal{H}$-matrices, the execution time of lattice $\mathcal{H}$-matrices is also slightly larger when normal $\mathcal{H}$-matrices continue to demonstrate a speed-up. The execution times of lattice $\mathcal{H}$-matrices are about one-tenth to one-third the best execution times of normal $\mathcal{H}$-matrices when more than 3,400 cores are used.

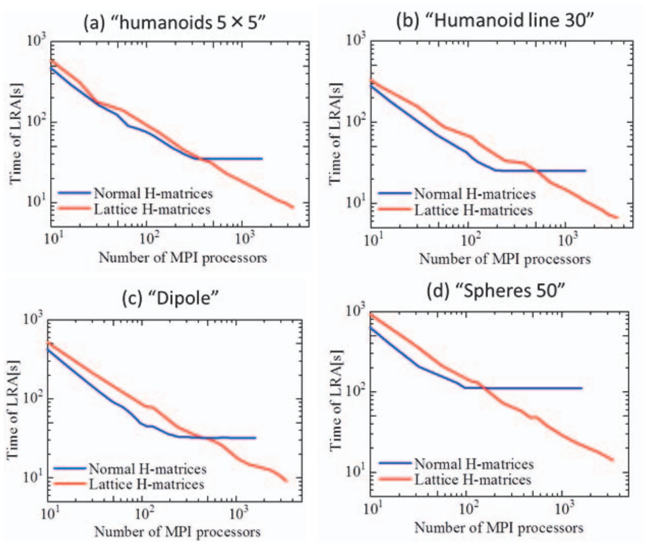

Figure 17: Parallel scalability when performing LRA with normal and lattice $\mathcal{H}$-matrices.

G. Performance Evaluation When Performing HMVM

For the HMVM, we evaluate three types of parallel algorithms: one for normal $\mathcal{H}$-matrices and two for lattice $\mathcal{H}$-

matrices, as described in Section III.*C*. The observed execution times for a single HMVM are shown in Fig. 18. We consider the fastest times of 100 HMVM trials for each algorithm.

For normal $\mathcal{H}$-matrices, which correspond to the blue lines in Fig. 18, we observe limits in the speed-up at approximately 30 to 100 processors, respectively, which is significantly smaller than the 100 to 300 processors when performing LRA shown in the previous subsection. This limitation is caused by the MPI communication costs because the load imbalance does not significantly deteriorate the parallel speed-up when performing LRA. Although we observed the breakdown of the times for the body calculation and for the MPI communications, we will not show the results here for brevity. According to the observation, the times for the body calculation monotonously decrease, but the times for MPI communications increase as the number of processors increases. When the number of the MPI processors is approximately equal to that of the speed-up limitations, the increased time of MPI communications surpasses the decreased time for the calculation body.

The curve in the case with the LatticeHMVM_fullVector algorithm of lattice $\mathcal{H}$-matrices is indicated by red lines in Fig. 18. In contrast to the blue lines for the case with normal $\mathcal{H}$-matrices, the shapes of the red lines are similar. We observe the parallel speed-up up to approximately 500 MPI processors in all cases. As mentioned before, the computational amount of the calculation body when using lattice $\mathcal{H}$ -matrices is approximately two to three times larger than that for normal $\mathcal{H}$-matrices. However, the MPI communication cost of the LatticeHMVM_fullVector is much smaller than that of the algorithm for normal $\mathcal{H}$-matrices. As a result, the red curves fall primarily along the blue curves for normal $\mathcal{H}$-matrices when a small number of MPI processors are used. When the best times for the execution are compared, the LatticeHMVM_fullVector is 1.5-fold to 3-fold faster than the algorithm for normal $\mathcal{H}$-matrices.

We can achieve a more significant improvement of the HMVM if the user's program permits the use of LatticeHMVM_partVector with more than 700 MPI processors. The observed execution times when using the algorithm LatticeHMVM_partVector are shown as the green line in Fig. 18. It is expected that the speed-up will continue even when using over 4,000 MPI processors. However, for the case in which other algorithms continue to have a speed-up, the execution time of LatticeHMVM_partVector is much slower than of other algorithms. This is because the square-shaped process grid needed for LatticeHMVM_partVector results in a low load-balance efficiency, as discussed in Section III.*D*. The fastest execution time observed when using 3,600 cores is approximately two times faster than LatticeHMVM_fullVector, which indicates that it is 3-fold to 6-fold faster than the algorithm for normal $\mathcal{H}$-matrices. Although the times for the calculation body using LatticeHMVM_partVector are much larger than those with LatticeHMVM_fullVector, the MPI communication costs of LatticeHMVM_partVector are significantly smaller than those of LatticeHMVM_fullVector. For LatticeHMVM_fullVector, the MPI communication costs increase proportionally with the number of used processors, whereas they do not increase in the case of LatticeHMVM_partVector even if a large number of MPI processors are used.

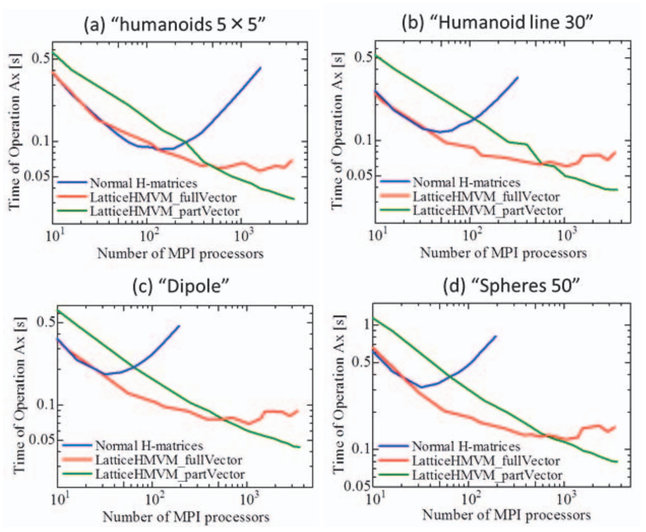

Figure 18: Parallel scalability when performing HMVM.

V. Related Work

Although $\mathcal{H}$-matrices have been applied to BEM analyses and have been demonstrated to be effective [3, 5, 13], there are few investigations into their parallelisation. Parallel algorithms for $\mathcal{H}$ -matrices on shared-memory computing systems are proposed in [14], and the algorithms assign tasks based on the space-filling curves in the structure of the $\mathcal{H}$-matrices. In [15], the enhancement of the algorithms for distributed-memory systems is discussed, and the performance is evaluated on a small-cluster system, built by connecting 16 single-CPU machines. The parallel algorithms for $\mathcal{H}$ -matrices on symmetric multiprocessing clusters are proposed in [5] and adopted in the open-source $\mathcal{H}$-matrices library, $\mathcal{H}$ACApK [1]. As mentioned in this paper, the assignment strategy in $\mathcal{H}$ACApK is based on the row-wise, 1D-sliced decomposition of the matrix. The cyclic assignment of the 1D-sliced decomposition is proposed in [13].

To improve the parallel scalability and the convenience of matrix arithmetic from $\mathcal{H}$ -matrices, some formats with a simpler structure have been proposed in the literature. In Hierarchically Semi-Separable (HSS) matrices [16, 17] and hierarchically off-diagonal low-rank (HODLR) matrices [18], all off-diagonal blocks are assumed to be low-rank; this assumption is called a weak admissibility condition. By exploiting the simplified structure, the HSS software STRUMPACK exhibits a better parallel scalability—up to several thousands of cores [19]. However, the increase in the ranks of off-diagonal matrices would result in the worse asymptotic complexity, if the weak admissibility condition is applied to high dimensional problem [17]. The idea of BLR [7] was proposed in 2015 to simplify the complex arithmetic of hierarchical methods and to approximate the Schur complements appearing in multifrontal solvers [20]. In [8], we proposed parallelisation algorithms for BLR and discussed the applicability of BLR instead of normal $\mathcal{H}$-matrices in large-scale BEM analyses. The lattice $\mathcal{H}$-matrices dealt with in this paper are a drastic improvement of our previous work because they combine the advantages of $\mathcal{H}$-matrices with BLR. The lattice $\mathcal{H}$-matrices are expected to be possessed all information of the corresponding $\mathcal{H}$-matrices in contrast to most of the simplified formats, which lose a part of information.

VI. Conclusion

In this study, we proposed a new variant of low-rank structured matrices, called "lattice $\mathcal{H}$ -matrices," and introduced their parallelisation algorithms. The concept of the lattice $\mathcal{H}$-matrices is to utilise the advantages of both normal $\mathcal{H}$-matrices and BLR-matrices. In particular, these advantages are the high memory compressibility of normal $\mathcal{H}$-matrices and the simple structure of BLR-matrices. In the lattice $\mathcal{H}$-matrices, the simple structure of the lattice coming from BLR is utilised to exploit distributed-memory systems using MPI processes. Each MPI process has assigned tasks in block units on the lattice. The lattice structure permits the use of 2D-cyclic assignment for good load balance and efficient communication patterns. Although we must perform arithmetic due to the complex structure coming from $\mathcal{H}$-matrices in each MPI process, it would be acceptable in serial or thread computing on a CPU node. Thanks to the complex structure in each block of the lattice, lattice $\mathcal{H}$-matrices reduce the memory complexity from BLR O($N^{1.5}$) to $O(N \log N)$. We confirmed the memory complexity both in theory and in practical experiments.

To realise 2D-cyclic assignment for arithmetic with lattice $\mathcal{H}$-matrices, we introduced a "process grid." We discussed the relation between the shape of the process grid and load balancing and demonstrated that we can obtain good load balance for a rectangularly shaped process grid compared with a square one. In numerical experiments, 2D-cyclic assignment based on the rectangular process grid attained high-efficiency load balances. With the lattice $\mathcal{H}$-matrices constructed with the assignment, the speed-up limit increases to as high as 4,000 processes, and this speed-up is expected to increase, which is in contrast to the speed-up limit near several hundreds of processors for normal $\mathcal{H}$-matrices. The lattice $\mathcal{H}$-matrices achieved 3- to 10-fold acceleration when compared with normal $\mathcal{H}$-matrices constructed with quasi-1D-sliced assignment. For the lattice $\mathcal{H}$-matrix-vector multiplication on MPI processes, we introduced algorithms: LatticeHMVM_fullVector and LatticeHMVM_partVector, which differ in communication pattern. In numerical experiments, we observed speed-ups up to about 500 processes for LatticeHMVM_fullVector and about 4,000 processes for LatticeHMVM_partVector, in contrast to the speed-up saturation with only a few dozen processes for normal $\mathcal{H}$-matrices. The fastest execution time of LatticeHMVM_partVector was 3 to 6 times faster than the algorithm for normal $\mathcal{H}$-matrices.

In this article, we only considered simple arithmetic, matrix generation and matrix-vector multiplication of lattice $\mathcal{H}$-matrices. However, we believe that our proposed lattice $\mathcal{H}$-matrices are promising for more complex arithmetic, such as matrix-matrix multiplication, LU factorisation and matrix inversion, on distributed-memory systems. In the near future, we will investigate these applications. If the complex structure of $\mathcal{H}$-submatrices prevents high performance even in computing on a CPU node, we could alternatively consider the hierarchical use of BLR-submatrices instead of the $\mathcal{H}$-submatrices.

Acknowledgment

This work was partially supported by JSPS KAKENHI Grant Number 17H01749 and 17K19962.

References

[1] *ppOpen-HPC Project*. Accessed: Feb. 26, 2018. [Online]. Available: http://ppopenhpc.cc.u-tokyo.ac.jp/ppopenhpc/.

[2] Iwashita T. *et.al.*, "Software Framework for Parallel BEM Analyses with H-matrices Using MPI and OpenMP," *Proc. Comput. Sci.*, vol. 108, pp.2200–2209, 2017.

[3] Hackbusch W., "A Sparse Matrix Arithmetic Based on H-matrices. I. Introduction to H-matrices," *Computing*, Vol. 62(2), pp. 89-108,1999.

[4] Börm S. *et.al.*, *Hierarchical Matrices* (Lecture Note). Bonn, Germany: Max-Planck-Institut fur Mathematik*,* 2006.

[5] Ida A. *et.al.*, "Parallel Hierarchical Matrices with Adaptive Cross Approxima-tion on Symmetric Multiprocessing Clusters," *J. Inf. Process.*, vol. 22, no. 4, pp.642-650, 2014.

[6] Bebendorf M., "Efficient inversion of the galerkin matrix of general second-order elliptic operators with non-smooth coefficients," *Math. Comp.* , vol. 74(251), pp. 1179-1199, 2004.

[7] Amestoy P. *et.al.*, "Improving multifrontal methods by means of block low-rank representations," *SIAM J. Sci. Comput.,* vol. 37(3), pp.1451-1474, 2015.

[8] Ida A. *et.al.*, "Parallel Hierarchical Matrices with Block Low-rank Representation on Distributed Memory Computer Systems," *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region. ACM,* pp.232-240, 2018

[9] Bebendorf M., "Approximation of boundary element matrices," *Numer. Math.* vol. 86, 4, pp.565–589(2000).

[10] Bebendorf M.: *Hierarchical Matrices*, Springer, Section 1.3.1 (2008).

[11] *ScaLAPACK Home Page*. Accessed: Feb. 26, 2018. [Online]. Available: http://www.netlib.org/scalapack/scalapack_home.html

[12] Choi, J. *et.al.*, "PUMMA: Parallel Universal Matrix Multiplication Algorithms on distributed memory concurrent computers," *Concurrency: Practice and Experience*, Vol 6(7), pp.543-570, 1994.

[13] Ando K. *et.al.*, "Parallel-algorithm Extension for Tsunami and Earth quake-cycle Simulators for Massively Parallel Execution on the K Computer," *Int. J. HPC Appl.*, vol. 30(4) pp.454–468, 2016.

[14] Kriemann R., "Parallel H-Matrix Arithmetics on Shared Memory Systems," *Computing*, vol. 74, pp.273-297, 2005.

[15] Bebendorf M. and Kriemann R., "Fast Parallel Solution of Boundary Integral Equations and Related Problems," *Comput. Visual. Sci.*, vol. 8, pp.121-135, 2005.

[16] Chandrasekaran S. *et.al.*, "A fast solver for HSS representations via sparse matrices," *SIAM J.Matrix Anal. Appl.*, vol. 29(1), pp.67–81, 2006.

[17] Xia J. *et.al.*, "Superfast multifrontal method for large structured linear systems of equations," *SIAM J.Matrix Anal. Appl.* vol. 31(3), pp.1382–1411, 2009.

[18] Ambikasaran S. and Darve E., "An O(NlogN) fast direct solver for partial hierarchically semi-separable matrices," *J. Sci. Comput.*, vol. 57, pp.477–501, 2013.

[19] Rouet F.H., *et.al.*, "A Distributed-Memory Package for Dense Hierarchically Semi-Separable Matrix Computations Using Randomization," *ACM Trans. Math. Softw.*, vol. 42(4), Article 27, 2016.

[20] Patrick R. *et.al.*, "Fast 3D frequency-domain full waveform inversion with a parallel Block Low-Rank multifrontal direct solver: application to OBC data from the North Sea," *Geophysics,* vol. 81(6), pp.363 –383, 2016.