Electronic Preprint for Journal of Information Processing Vol.22 No.4

Regular Paper

Parallel Hierarchical Matrices with Adaptive Cross Approximation on Symmetric Multiprocessing Clusters

Akihiro Ida[1,2,a] Takeshi Iwashita[2,3] Takeshi Mifune[2,4] Yasuhito Takahashi[2,5]

Received: December 20, 2013, Accepted: June 17, 2014

Abstract: We discuss a scheme for hierarchical matrices with adaptive cross approximation on symmetric multiprocessing clusters. We propose a set of parallel algorithms that are applicable to hierarchical matrices. The proposed algorithms are implemented using the flat-MPI and hybrid MPI+OpenMP programming models. The performance of these implementations is evaluated using an electric field analysis computed on two symmetric multiprocessing cluster systems. Although the flat-MPI version gives better parallel scalability when constructing hierarchical matrices, the speed-up reaches a limit in the hierarchical matrix-vector multiplication. We succeeded in developing a hybrid MPI+OpenMP version to improve the parallel scalability. In numerical experiments, the hybrid version exhibits a better parallel speed-up for the hierarchical matrix-vector multiplication up to 256 cores.

Keywords: boundary element method, matrix approximation, hierarchical matrices, adaptive cross approximation, parallel scalability, symmetric multiprocessing clusters

1. Introduction

The boundary element method (BEM) is an important numerical method for solving partial differential equations. Other techniques include the finite element and finite difference methods. Although it is important to select the appropriate method according to analyses, use of BEM tends to be avoided despite having advantages over the other methods. A major reason is that a naïve application of BEM yields a system of linear equations with a dense coefficient matrix. For N unknowns, this dense matrix requires a memory footprint proportional to N^2 and computational effort of $O(N^2)$. This prevents large scale BEM analyses. Parallel computing offers a solution to this problem. It is also possible to use approximation techniques for dense matrices that reduce the computational cost from $O(N^2)$ to $O(N) \sim O(N \log N)$. We can use both techniques at the same time.

The dense matrices arise from the convolution integral term in the fundamental equation of BEM. Recently, a number of approximation techniques have been proposed such as the hierarchical matrix (H-matrix) [1], [2], [3], fast multipole (FMM) [4], [5] and tree [6] methods. All of these methods are based on a similar idea: for remote interactions, the kernel functions of convolution integrals are approximated by a degenerate kernel expressed by several terms of a series expansion. In this paper, we consider H-matrices combined with adaptive cross approximation (ACA) [7], [8]. The ACA is used for sub-matrices that H-matrices detect as potential low-rank sub-matrices. The use of H-matrices with ACA has several advantages over the above techniques. First, only the original entries of the coefficient matrix are used for its approximation, whereas the other techniques require a concrete form of the degenerate kernel. Second, H-matrices with ACA can easily control the accuracy of approximation. Third, H-matrices explicitly calculate all of the entries of the approximated matrix in the computer memory, whereas the FMM prepares only the coefficients of the power series expansion for the degenerate kernel. This means that we can directly apply the operations with respect to an H-matrix. Therefore, as reported in Ref. [9], H-matrices are faster than FMM in terms of the execution time on the matrix-vector product. H-matrices with ACA have been applied to practical BEM applications, and have been very effective [8], [10], [11], [12], [13], [14].

In this paper, we also consider the use of parallel computer systems. In Ref. [15], two parallel algorithms that use H-matrices with ACA were proposed for shared memory systems. These algorithms describe how to assign tasks to processing cores when constructing an H-matrix and performing an H-matrix-vector multiplication (HMVM). In both algorithms, the arithmetic is performed in units of sub-matrix appearing on an H-matrix. One of the algorithms assigns sub-matrices to processing cores such that assignments for a processing core are scattered over the whole matrix (**Fig. 1**, left). This algorithm is not suitable for cluster systems, because of the large amount of communication that is necessary for gathering and scattering the sections. In the second algorithm, the whole matrix is divided into collections of sub-matrices of an H-matrix, which are as close to square as possible (Fig. 1, right). These square shape collections are then assigned to each processing core, reducing the amount of communication

[1] ACCMS, Kyoto University, Kyoto 606–8501, Japan
[2] JST CREST, Kawaguchi, Saitama 332–0012, Japan
[3] Information Initiative Center, Hokkaido University, Sapporo, Hokkaido 060–0811, Japan
[4] Department of Electrical Engineering, Kyoto University, Kyoto 615–8530, Japan
[5] Department of Electrical Engineering, Doshisha University, Kyotanabe, Kyoto 610–0394, Japan
[a] ida@media.kyoto-u.ac.jp

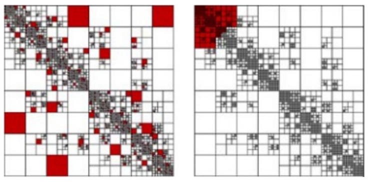

Fig. 1 Red squares represent the assignments of a worker by the algorithms in Ref. [15]. This figure is transcribed from Ref. [15].

between cores. The square-like collections are found by using the nature of quad-tree inducing H-matrices, so the algorithm does not always work for an arbitrary H-matrix. The performance of the algorithm was evaluated on a relatively small cluster system, built by connecting 16 single-CPU machines [16]. For H^2-matrices, which are special variants of H-matrices, a parallel algorithm on distributed memory clusters was proposed in Ref. [17]. The algorithm was implemented using flat-MPI approach, and the parallel scalability was also evaluated on a small cluster system with 16 single-CPU machines. In both papers above, saturation of speed-up in HMVM was observed for problems containing several tens of thousands of unknowns, though a better parallel speed-up was obtained for larger problems. Because our interest is in BEM analyses by using many more cores, we need to be able to efficiently exploit cores even in large-scale problems. Moreover, considering the trend in high performance computing on supercomputer systems, we should implement algorithms that will work on symmetric multiprocessing (SMP) cluster systems.

We propose a new parallel algorithm that uses H-matrices with ACA. We horizontally slice the H-matrix (see Fig. 3 in Section 3). The proposed parallel algorithm does not enforce any restrictions on the construction of H-matrices, because it does not depend on a specific hierarchical structure. We have developed a numerical library called Hacapk, which includes two types of implementations of the proposed parallel algorithm. One is based on the flat-MPI approach. In the other implementation, the hybrid MPI+OpenMP programming model is used. All the programs in the Hacapk library are written in the Fortran 95 programming language. The performance of the Hacapk library was evaluated using a static electric field analysis, conducted on SMP cluster systems. In Section 2, we introduce the hierarchical matrices using typical examples derived from integral equations. In Section 3, we explain our proposed parallel H-matrix method. We discuss the numerical experiments for the electric field analysis in Section 4. The last section contains conclusions and future work.

2. Hierarchical Matrices

In this section, we give an overview of H-matrices to aid the description of our parallel algorithm in the next section. The conceptual diagram of an H-matrix appears as a collection of blocks created by partitioning a square (see Fig. 1 in Section 1 and Fig. 2 in Section 3). The blocks do not overlap each other, and fill up the square without gaps. We introduce a set M, whose elements correspond to one of the blocks. Obviously, M represents the partition structure itself. Furthermore, we write the set of H-matrices with the structure M as $\mathcal{H}(M)$. An H-matrix $\tilde{A} \in \mathcal{H}(M)$ can be an efficient approximation of the specific dense matrix $A \in \mathbb{R}^{N\times N}$, where $N \in \mathbb{N}$. The typical formulation of the specific dense matrix is described in Section 2.1. If we apply the partition structure M to the matrix A, an element $m \in M$ corresponds to a sub-matrix $A|^m_{s_m\times t_m}$ on the matrix A, where the subscripts s_m and t_m denote the subsets of the row's and column's index set $\beth := \{1, \cdots, N\}$ of the matrix A, respectively. Thus, $A|^m_{s_m\times t_m}$ is defined by gathering the columns and rows of the coefficient matrix A whose ordinal numbers belong to sets s and t. For all $m \in M$, $\tilde{A}|^m$, corresponding to a conceptual block on $\tilde{A}$, gives the approximation of $A|^m_{s_m\times t_m}$, while $\tilde{A}|^m$ is defined in the form of the multiplication of two low-rank matrices as

$$\begin{aligned} &\tilde{A}|^m := V_m W_m, \\ &V_m \in \mathbb{R}^{s_m\times r_m}, \quad W_m \in \mathbb{R}^{r_m\times t_m}, \quad r_m \le \min(\#s_m, \#t_m) \end{aligned} \tag{1}$$

where $r_m \in \mathbb{N}$ indicates the rank of the matrices. If $r_m = \min(\#s_m, \#t_m)$ for all m, the H-matrix $\tilde{A}$ coincides with the dense matrix A. Because most of the ranks in an H-matrix are generally much smaller than the sub-matrix sizes $\#s_m$ and $\#t_m$, an H-matrix is considered a data-sparse matrix. Furthermore, the H-matrix becomes sparser (in terms of the amount of data) as the number of large blocks increases. A procedure for constructing H-matrices is described in Section 2.2.

2.1 Systems of Linear Equations Derived from Integral Equations with a Singular Kernel

We now describe the formulation of a system of linear equations, with the dense matrix approximated by H-matrices. Let H be a Hilbert space of functions on a $(d-1)$-dimensional domain $\Omega \subset \mathbb{R}^d$, and H' be the dual space of H. For $u \in H$, $f \in H'$ and a kernel function of a convolution operator g: $\mathbb{R}^d \times \Omega \to \mathbb{R}$, we consider an integral equation

$$\int_\Omega g(x,y)\, u(y)\, \mathrm{d}y = f. \tag{2}$$

To numerically calculate Eq. (2), we divide the domain, Ω, into elements $\Omega^h = \{\omega_j : j \in J\}$, where J is an index set. When we use the weighted residual methods such as the Ritz-Galerkin method and the collocation method, the function u is approximated from a n-dimensional subspace $H^h \subset H$. Given a basis $(\varphi_i)_{i\in\beth}$ of H^h for an index set $\beth := \{1, \cdots, N\}$, the approximant $u^h \in H^h$ to u can be expressed using a coefficient vector $\phi = (\phi_i)_{i\in\beth}$ that satisfies $u^h = \sum_{i\in\beth} \phi_i\varphi_i$. Note that the supports of the basis $\Omega^h_{\varphi_i} := \operatorname{supp}\varphi_i$ are assembled from the sets ω_j. Equation (2) is reduced to the following system of linear equations.

$$A\phi = B \tag{3}$$

In the case of the Ritz-Galerkin method, the entries of A and B are given by

$$A_{ij} = \int_\Omega \varphi_i(x) \int_\Omega g(x,y)\, \varphi_j(y)\, \mathrm{d}y\, \mathrm{d}x \quad \text{for all } i,\ j \in \beth \tag{4}$$

$$B_i = \int_\Omega \varphi_i(x)\, f\, \mathrm{d}x \quad \text{for all } i \in \beth \tag{5}$$

Suppose that the kernel functions are written in the form

$$g(x,y) \in \operatorname{span}(\{|x-y|^{-p},\ p > 0\}). \tag{6}$$

Moreover, permute the index set $\beth := \{1, \cdots, N\}$. Then there exist H-matrices that efficiently approximate the coefficient matrix $A \in \mathbb{R}^{\beth\times\beth}$. Such kernel functions appear in a number of scientific applications, for example, electric field analyses, mechanical analyses, and earthquake cycle simulations. It is well known that the N-body problem, whose formulation is different from the one above, results in a similar matrix.

We now suppose that we have two subsets (clusters) $s,\ t \in \beth$ and the corresponding domains defined by

$$\Omega^h_s := \bigcup_{i\in s} \operatorname{supp} \varphi_i, \quad \Omega^h_t := \bigcup_{i\in t} \operatorname{supp} \varphi_i. \tag{7}$$

If the Euclid distance between Ω^h_s and Ω^h_t is sufficiently large compared with their diameters, we say that the cluster pair (s, t) is 'admissible'. Moreover, we also call the corresponding submatrix $A|_{s\times t}$ is admissible. On the domain corresponding to the admissible cluster pairs, the kernel function can be approximated to a certain accuracy by a degenerate kernel such as $g(x, y) \cong \sum_{\nu=1}^{k} g^\nu_1(x) g^\nu_2(y)$, where k is a positive number. Then the corresponding sub-matrix $A|_{s\times t}$ has low-rank. Under the assumption in Eq. (6), it is possible to find a permutation and a partition of the index set $\beth$ so that there exist many large, low-rank sub-matrices.

2.2 Procedure of Constructing H-matrices with ACA

Although an H-matrix can be an approximation of a dense matrix $A \in \mathbb{R}^{\beth\times\beth}$ with entries described as in Eq. (4), we do not need to calculate all of the entries of A to construct the H-matrix. To construct A, we require the following information:

(i) The entries of A that ACA needs to calculate the matrices V_m and W_m in Eq. (1).
(ii) The coordinates of the point associated with each index $i \in \beth$.

In (ii), we typically select the center of the supports of the basis $\Omega^h_{\varphi_i}$. The coordinates in (ii) are used to find the permutation and partition, discussed in the previous subsection. These manipulations, the permutation and partition, are called 'clustering'. The clustering is based on the distance between clusters and the size of the corresponding supports, so we do not need any information about the entries of A. It is known that we can efficiently cluster by constructing the cluster tree $T_\beth$ from the index set $\beth$. Some types of tree structures (such as binary- and quad-trees) are used for the same purpose in other approximation techniques, such as in the FMM and tree methods. A cluster tree creates a block cluster tree, $T_{\beth\times\beth}$, whose node represents a block in $\beth\times\beth$. Note that a binary (quad) cluster tree creates a quad (hexadeca) block cluster tree. When we truncate branches of $T_{\beth\times\beth}$ according to the condition that $(s, t) \in T_{\beth\times\beth}$ is an admissible cluster pair, the leaves $\mathcal{L}(T_{\beth\times\beth})$ give a partition structure on $\beth \times \beth$. We adopt the partition structure $\mathcal{L}(T_{\beth\times\beth})$ as the M that defines the structure of an H-matrix.

An H-matrix is constructed using ACA in the following three steps.

1. Construct a cluster tree $T_\beth$.
2. Use the admissible condition to construct a partition structure, M from the block cluster tree, $T_{\beth\times\beth}$, with truncations.
3. Fill in the blocks of M using ACA.

In the third step, we can independently apply ACA to each block. In this step, we can control the approximation accuracy of the whole H-matrix by giving the necessary condition for ACA. The ACA tries to create the matrices V_m and W_m in Eq. (1) for all $m \in M$ such that

$$\frac{\|A - \tilde{A}\|_F}{\|A\|_F} \le \varepsilon, \tag{8}$$

where $\varepsilon \in \mathbb{R}_{>0}$ is the given error tolerance, and $\|\cdot\|_F$ denotes the Frobenius norm. The rank, r_m, of the approximated submatrix $\tilde{A}|^m_{s_m\times t_m}$ should be sufficiently small if the corresponding sub-matrix $A|^m_{s_m\times t_m}$ is admissible, otherwise $r_m = \min(\#s_m, \#t_m)$, i.e., $\tilde{A}|^m := V_m W_m$ coincides with $A|^m_{s_m\times t_m}$.

3. Our Proposal Parallel Algorithm of H-matrices

In this section, we discuss the parallel computation of H-matrices on SMP cluster systems. We assume that there are MPI processes, and, in the case of the hybrid MPI+OpenMP programming model, that a MPI process contains multiple threads. We now focus on the parallelization of HMVM and the third step of the H-matrix construction procedure, in which ACA fills in the entries of blocks on M. The first and second steps are redundantly performed by all MPI processes, i.e., all MPI processes have same information about the set M that defines the frame of sub-matrices by a permutation and partition on A. Furthermore, all MPI processes share the full multiplicand vector for HMVM using MPI communication.

3.1 Assignment of Tasks to MPI Processes

We here consider dividing the set M, among N_p MPI processes. In our algorithm, the arithmetic for filling the blocks on M, and for the HMVM, is performed in units of a block $\tilde{A}|^m$ approximating $A|^m_{s_m\times t_m}$, where $m \in M$. We discuss the process of assigning each block to an MPI processor in this subsection. The computational complexity is proportional to the total number of entries in the approximated sub-matrices $N_M := \sum_{m\in M} N_m(s_m, t_m, r_m)$ where $N_m(s_m, t_m, r_m)$ denotes the number of entries of matrices V_m and W_m approximating sub-matrix $A|^m_{s_m\times t_m}$. For all $m \in M$, the number $N_m(s_m, t_m, r_m)$ is written using the rank r_m,

$$N_m(s_m, t_m, r_m) = r_m \cdot (\#s_m + \#t_m) \tag{9}$$

Now, let the index set, $I := \{1, \cdots, N\}$, be related to the ordinal number of rows of the whole matrix A, and divide I into N_p subsets, $I_k := \{i : l_k \le i < l_{k+1}\}$, where $1 = l_1 < \cdots < l_k < \cdots < l_{N_p+1} = N + 1$. We temporarily define the separators $l_k = \mathrm{Ceil}(N/N_p)(k - 1) + 1$. Next, we define the function R_u: $M \to I$ as

$$R_u(m) := \min\{i : i \in s_m\} \text{ for } m \in M. \tag{10}$$

The function R_u associates each sub-matrix $A|^m_{s_m\times t_m}$ with a subset I_k (see **Fig. 2**). By using the function R_u, the set, M, can be divided into subsets $M_k := \{m : R_u(m) \in I_k\}$. Then the amount of jobs associated with a subset M_k is proportional to $N_{M_k} := \sum_{m\in M_k} N_m(s_m, t_m, r_m)$. If we assign a set M_k to the kth MPI processor, the load is not generally balanced among MPI

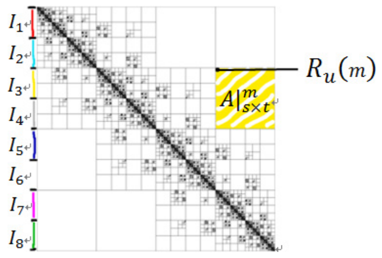

Fig. 2 Index set I is divided into subsets, I_k, and the block corresponding to a sub-matrix $A|^m_{s_m\times t_m}$ is associated with a subset I_k using $R_u(m)$.

```
Procedure $DivideM(M, N_p, N)$
  Do k=1,$N_p$
    $I_k := \{i : l_k \le i < l_{k+1}\}$
    where $l_k = \mathrm{Ceil}(N/N_p)(k-1)+1$
  End do
  For all $m \in M$, $M_k := \{m : R_u(m) \in I_k\}$
  Set $r_{\mathrm{est}}$, e.g. $r_{\mathrm{est}} = 10$
  $N_M := \sum_{m\in M} N_m(s_m, t_m, r_{\mathrm{est}})$; $N_{Mave} := N_M/N_p$
  Set $tol$ e.g. $tol = 0.1$
  Do $k$ =1,$N_p$ −1
     If $(N_{M_k} < N_{Mave})$ then $\sigma_k = 1$ else $\sigma_k = -1$
     Do While ($|N_{M_k} - N_{Mave}| > tol * N_{Mave}$
                    and $\sigma_k(N_{M_k} - N_{Mave}) < 0$)
         $l_{k+1} := l_{k+1} + \sigma_k$
       Reset $I_k := \{i : l_k \le i < l_{k+1}\}$
       For all $m \in M$, $M_k := \{m : R(m) \in I_k\}$
     End do
  End do
  Return $M_k$
End
```

Algorithm 3.1: Divide M into M_k

processors. Our algorithm attempts to balance the load by adjusting the position of the separators, l_k. To determine the best load balancing, we need knowledge of N_{M_k}. Unfortunately, it is quite difficult to determine the size of N_{M_k} before ACA fills up all entries of approximated sub-matrices. This is because the rank of an approximated sub-matrix, r_m, depends on the required accuracy, and the rank of each sub-matrix can be different. Therefore, we calculate N_{M_k} using an estimation of r_m. We calculate this value by counting the entries under the assumption that $r_m \equiv$ constant. In the calculations shown in Section 4, we have used this simple estimation method, where $r_m = 10$. The Algorithm 3.1 outlines the method for dividing M into M_k using load balancing. Finally, the subset M_k is assigned to the kth MPI processor.

3.2 Filling in the Blocks on M

When the kth MPI processor has the assignment $M_k \subset M$, we can fill in the blocks on M without any MPI communication. In the case of the flat MPI model, each MPI processor independently performs Algorithm 3.2, which is the same as the serial algorithm. When using the hybrid MPI+OpenMP programming model, we redivide M_k and assign jobs to the OpenMP threads. We propose that Algorithm 3.1 is also used for this redivision for multiple threads. In the hybrid parallel processing case, each OpenMP thread independently executes Algorithm 3.2.

```
For all $m \in M_k$ do
  If $A|^m_{s_m\times t_m}$ is dense matrix then
    Calculate all entries of $A|^m_{s_m\times t_m}$
  Else
    Create low-rank approximation of $A|^m_{s_m\times t_m}$
    by ACA
  End If
End do
```

Algorithm 3.2: Filling in blocks on M

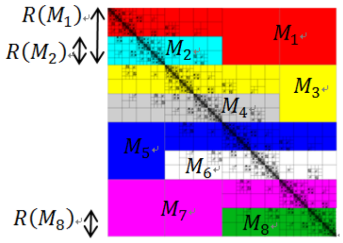

Fig. 3 Set M, which defines the frame of blocks or sub-matrices, is divided into subsets, M_k using Algorithm 3.1. The different colors represent the blocks assigned to different MPI-processors. The shape of the block of sub-matrices, M_k is like a bar or a 'key'.

3.3 H-matrix-vector Multiplication (HMVM)

We here consider the algorithm for calculating a vector $y \in \mathbb{R}^N$ as the result of an HMVM

$$y := \tilde{A}x \tag{11}$$

for a given vector $x \in \mathbb{R}^N$, and the H-matrix $\tilde{A} \in \mathcal{H}(M)$ whose sub-matrices set M is divided into subsets M_k, as shown in Section 3.1. In the actual applications, HMVM is carried out multiple times, more than ten thousand times in some cases. Under our assumption, the kth MPI processor has part of the H-matrix $\tilde{A}_{M_k} := \{\tilde{A}|^m_{s_m\times t_m} : m \in M_k\}$, and the entire vector x. Because the result of an HMVM, y_k, is generally a part of the entire vector y, MPI communication is required before the next HMVM. The range of y_k on the entire vector y is determined using R: $M \to I$, where

$$R(M_k) := \{i:\ i \in s_m \text{ for all } m \in M_k\},\quad \text{for } M_k \subset M. \tag{12}$$

as in **Fig. 3**. The HMVM algorithm for the kth MPI process is described in Algorithm 3.3. Each MPI process only communicates with its neighbor. When using the hybrid MPI+OpenMP programming model, the first step in Algorithm 3.3 is carried out by OpenMP threads in each MPI process. Although we have to redivide M_k for multithreading, the situation is slightly different from that in the division of M for MPI processes. The redivision of M_k can be performed after all entries of blocks on the H-matrix have been calculated. Therefore, we can use exact information about the rank of the approximated sub-matrices. Finally, we propose that the set M_k is redivided into subsets M_{kl} using Algorithm 3.1 with the estimation value r_{est} changed to the correct value of the rank for each approximated sub-matrix, where the subscript l denotes the number of the OpenMP thread. The first step of Algorithm 3.3 is modified to Algorithm 3.4 for the OpenMP paradigm.

```
//{first step: calculation of $y_k$}
$y_k := 0$
For all $m \in M_k$ do
    $y_k|_{s_m} := y_k|_{s_m} + A|^{m}_{s_m \times t_m} \cdot x|_{t_m}$
End do
//{second step: calculation of $y$ with MPI comm.}
$y := 0$
Sync
Do $l$=1, $N_p - 1$
    $P_s = \mathbf{Mod}(k - l, N_p)+1$
    $P_a = \mathbf{Mod}(k, N_p)+1$
    Send $y_{P_s}$ to $P_a$-th MPI processor
    $P_r = \mathbf{Mod}(k - l - 1, N_p)+1$
    $P_b = \mathbf{Mod}(k - 2, N_p)+1$
    Receive $y_{P_r}$ from $P_b$-th MPI processor
    Sync
    $y|_{R(M_{P_r})} := y|_{R(M_{P_r})} + y_{P_r}$
End do
```

Algorithm 3.3: HMVM on the k^{th} MPI processor

```
//{first step: calculation of $y_k$}
!$omp parallel private( $y_l, m$)
//{ $y_l$ and $m$ are thread private.}
  $y_l := 0$
  For all $m \in M_{kl}$ do
      $y_l|_{s_m} := y_l|_{s_m} + A|^{m}_{s_m \times t_m} \cdot x|_{t_m}$
  End do
  $y_k := 0$
  $y_k|_{R(M_k)} := y_k|_{R(M_k)} + y_l$ using the atomic
                                        option of OpenMP
```

Algorithm 3.4: The first step of Algorithm 3.3 modified for OpenMP paradigm (execution in l^{th} thread).

4. Evaluation of Performance

We will now discuss the performance of the proposed parallel algorithm for H-matrices. To test the methods, we have selected an electrostatic field problem. We have assumed that a perfect conductor, which has the shape of a sphere, is set in a space such that the distance between the center of the sphere and the surface of the ground with zero electric potential is 0.5 m. The radius of the sphere conductor is 0.25 m, and the conductor is charged such that its electric potential becomes 1 V. We used the surface charge method to calculate the electrical charge on the surface of the conductor. The surface charge was distributed as shown in **Fig. 4**. When applying BEM to the above electrostatic field analyses, we have used the formulation described in Section 2.2, with the kernel function given by

$$g(x, y) = \frac{1}{4\pi\epsilon} |x - y|^{-1}. \tag{13}$$

Here, ϵ denotes the electric permittivity. We selected the collocation method from the various types of weighted residual methods. We divided the surface of the conductor into triangular elements (see Fig. 4), and used step functions as the base function of BEM. Four types of meshes were calculated for the numerical experiments, shown in **Table 1**. We applied H-matrices to the coefficient matrices of the systems of linear equations derived from the above formulation.

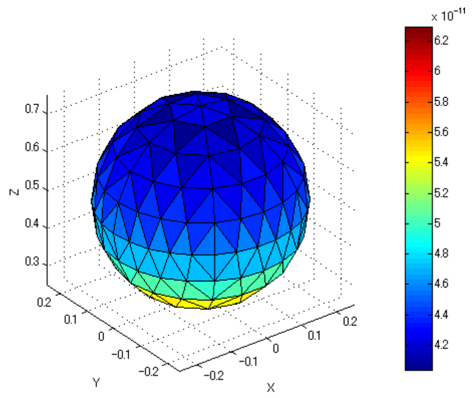

Fig. 4 The calculated surface charge density.

Table 1 Number of elements included in the meshes used for numerical experiments.

Mesh No.	Number of elements
1	1,000
2	10,400
3	101,250
4	1,004,400

Table 2 Computer systems used for the numerical experiments.

Name	Proccesor	Memory	Network
Fujitsu FX10 at Tokyo Univ.	SPARC64™ IXfx (16cores/node)	32GB/node	5 GB/s, Tofu
Appro GreenBlade 8000 at Kyoto Univ.	Xeon E5 ™ (8core ×2socket /node)	64GB/node	InfiniBand, Fat tree
Desktop PC	Xeon X5680 (6core×2socket)	24GB	Nothing (Standalone PC)

We evaluated the performance of our Hacapk library and the proposed algorithm on two SMP cluster systems and a personal computer (**Table 2**).

4.1 Serial Computing

Before we evaluated the parallel scalability of the Hacapk library, its basic performance was examined using serial computing. We compared the execution time of the Hacapk library with HLib. HLib is a well-known library for H-matrices developed by the Max Planck Institute [18], and has been shown to be very effective when applied to practical applications [10], [13], [14]. We used only one core of the Desktop PC, and meshes 1 and 2 for the comparison. We used an Intel C++ Composer XE 2011 with the -O3 optimization option to compile Hlib, and an Intel

Table 3 Execution times of Hacapk and HLib when constructing H-matrices.

ε	N=1,000		N=10,400	
	Hacapk(s)	Hlib(s)	Hacapk(s)	Hlib(s)
1.0e - 3	0.411	0.478	8.44	9.96
1.0e - 4	0.488	0.565	10.55	11.96
1.0e - 5	0.560	0.637	12.50	13.89

Table 4 Execution time of Hacapk and Hlib when performing an HMVM.

ε	N=1,000		N=10,400	
	Hacapk(s)	Hlib(s)	Hacapk(s)	Hlib(s)
1.0e - 3	8.05e-4	1.74e-3	1.66e-2	3.48e-2
1.0e - 4	9.40e-4	2.01e-3	1.90e-2	3.99e-2
1.0e - 5	1.06e-3	2.26e-3	2.27e-2	5.55e-2

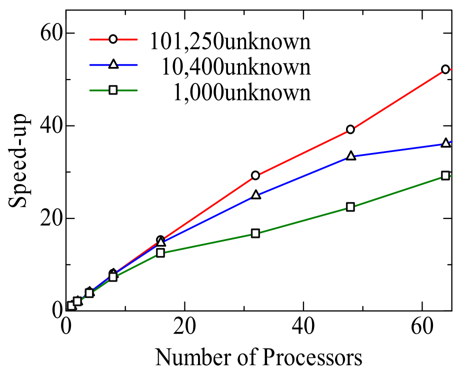

Fig. 5 Parallel scalability when constructing H-matrices.

Fortran Composer XE 2011 with -O3 optimization option to compile Hacapk. First, we investigated the execution times of Hacapk and HLib when constructing H-matrices. Then, we varied the error tolerance that controls the approximation accuracy of the H-matrices by letting $\varepsilon = 10^{-3}, 10^{-4}, 10^{-5}$. In all cases shown in **Table 3**, Hacapk is slightly faster than HLib. Next, we examined the execution time of one HMVM step. The results are shown in **Table 4**, where the values are averages over 50 executions of HMVM. In all cases, Hacapk was executed in approximately half the time of HLib. We have confirmed that our Hacapk library outperforms the existing H-matrices software using serial computing. This is because of the differences in the programming languages and data structures of Hacapk and HLib. HLib is written in C++ and expresses H-matrices using the tree structure of pointers, whereas Hacapk is written in Fortran 95 and expresses H-matrices using a simple structure of arrays.

4.2 Flat-MPI Programming Model

We investigated the parallel scalability of the version implemented using a flat-MPI parallel programming model in the Hacapk library. Strong scaling experiments were carried out for meshes 1, 2, and 3. We used a Fujitsu FX10 for the test analysis. The error tolerance of the remainder of our experiments was ε = 1.0e-4. **Figure 5** shows the speed-up versus the execution time of the serial computing in the H-matrix construction step using Hacapk, while **Fig. 6** shows the result for one HMVM step. They have been plotted as a function of the number of processes. The larger the data size becomes, the better parallel scalability

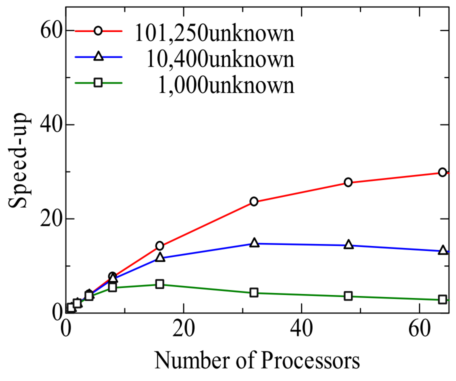

Fig. 6 Parallel scalability when performing an HMVM.

Hacapk attains in both the H-matrix construction and HMVM steps.

In the H-matrix construction, a more than 50-fold speed-up is obtained by using 64 cores for the largest model (N = 101,250). This is because MPI communication is not necessary in the H-matrix construction algorithm. Moreover, because the calculation of the kernel function for the construction step is computationally intensive, the parallel speed-up ratio is not overly affected by the memory bandwidth. We consider the difference between the ideal and obtained speed-ups to be due to the load-imbalance among the MPI processes.

Conversely, we observed limits in the speed-up ratios for HMVM. Although the attained maximum speed-up ratio becomes higher as the data size becomes larger, the speed-up ratio seems to have reached its limit when 64 MPI-processors are used, even for the largest data. For HMVM, we pay the cost of parallelization through the MPI communication and waiting costs. We believe that the ceiling is caused by the MPI communication costs because the load-imbalance does not significantly deteriorate the parallel speed-up ratio when constructing H-matrices.

4.3 Hybrid MPI+OpenMP Programming Model

One way of improving the parallel scalability of the flat-MPI version in HMVM is to reduce the cost of MPI communication. The cost for one MPI collective communication generally depends on the number of MPI processes involved. Although only one-to-one communication is used in our implementation, we have observed a similar property. This is because each process calls SEND and RECV *Np-1* times. When we conducted our analysis on an SMP cluster, we reduced the number of MPI processes to introduce a hybrid MPI+OpenMP parallel programming model. In this case, OpenMP threads were used inside each SMP node, instead of MPI processes. We then have to consider how many MPI processors were replaced with OpenMP threads. Although increasing the number of OpenMP threads reduces the MPI communication, the conflict in writing data between OpenMP threads also increases (in Algorithm 3.4). Therefore, it is not obvious how many OpenMP threads should be used. Furthermore, this depends on the architecture of an SMP cluster system.

We investigated the effect of using a hybrid MPI+OpenMP programming model for Hacapk performing HMVM. Numerical experiments were carried out on two SMP cluster systems:

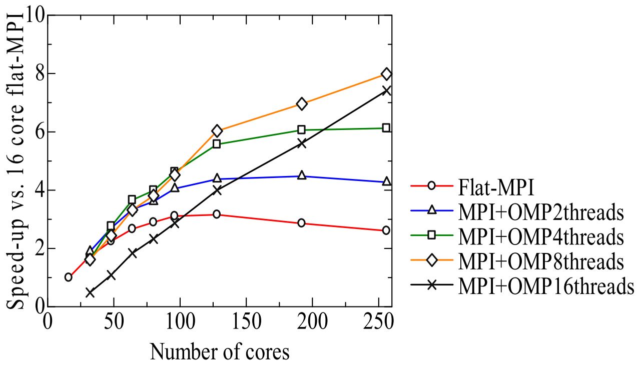

Fig. 7 Parallel scalability when performing an HMVM on FX10.

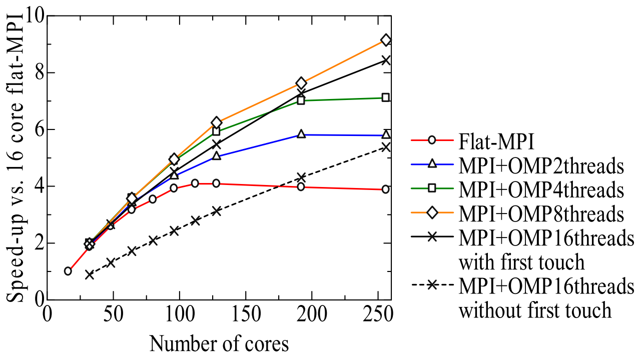

Fig. 8 Parallel scalability when performing an HMVM on GreenBlade 8000.

Fujitsu FX10 and Appro GreenBlade 8000. We examined the speed-up versus the execution time when the Flat-MPI version performed a HMVM on a single computational node, i.e., 16 cores. We used data set 4, which involves more than 1 million DOFs. The results of using the Flat-MPI version are plotted as circles in **Fig. 7** and **Fig. 8**. For both systems, the speed-up reached its ceiling at around 100 cores. We regarded the results of the Flat-MPI version as the standard. We then executed the hybrid MPI+OpenMP version for the same situations, varying the number of OpenMP threads to be 2, 4, 8, and 16. In both systems, the parallel scalability improved as the number of threads increased from 1 (Flat-MPI) to 8. It is expected that the speed-up ratio will continue to grow, even when using over 256 cores, when 8 threads are working on each SMP node. In these numerical experiments, we could not get the best results when replacing all MPI-processes in a node by OpenMP threads, although we did attain nearly linear scalability in FX10. In GreenBlade8000, which has two processors (sockets) in each node, it is not typically beneficial to use more threads than cores in a processor. This is because an overhead is required for the data cache coherency between sockets. In FX10, which consists of single socket nodes, it is not easy to give a simple explanation for the advantage of 8 threads over 16. The result might be due to the trade-off between the MPI communication cost and the effect of the atomic operation in multithreading. Moreover, we expect from Fig. 7 that the use of 16 threads might be advantageous, when more than 256 cores are used. In the future, we will examine this by running the numerical experiments using more cores.

Table 5 Execution times and percentages in electric field simulations by using serial computing on FX10.

	N=10,400		*N*=101,250		*N*=1,004,400	
	Time(s)	%	Time(s)	%	Time(s)	%
Construction of the system of linear equations	28.791	95	417.54	96.8	7056.8	96.5
BiCGSTAB	0.4501	1.5	5.2435	1.2	120.35	1.6
Others	1.0466	3.5	13.234	3.0	136.23	1.9
Total simulation	30.288	100	436.02	100	7313.4	100

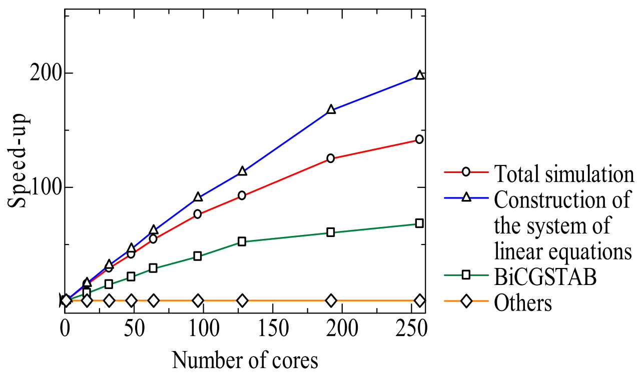

Fig. 9 Parallel scalability of the electric field simulation when applying the proposed parallel algorithms on FX10.

4.4 Parallel Scalability of Total Simulation

We here discuss how much of total simulation time is accelerated by the proposed parallel algorithms for the construction of H-matrices and HMVM.

The simulation for electric field problems with BEM, described at the top of this section, can be divided into three components. One is the construction of the system of linear equations, another is the solution of the system, and the other is the remained processing needed for performing the simulation. In terms of calculation time, the construction of an H-matrix for the coefficient matrix occupies almost of the construction of the system of linear equations. In order to solve the system, we adopt the BiCGSTAB method, which is an iterative method. The algorithm of BiCGSTAB consists of scalar operations, inner products for vectors and the matrix-vector multiplications which appear twice an iteration. In our simulation, the matrix-vector multiplication is performed as HMVM.

For mesh data 2, 3, and 4, we investigated the execution times of components mentioned above and the percentages against the total simulation time. The results are shown in **Table 5**. All the calculations are carried out by using serial computing on FX10. The convergence criterion of BiCGSTAB is that the relative residual with 2-norm becomes less than 1.0e-6. Then, the numbers of iterations of BiCGSTAB were around 10, which means that HMVM was carried out only around 20 times. The execution time for the construction of the system of linear equations occupied more than 95 percent of total simulation time.

Figure 9 shows the speed-up of total simulation and each component versus the execution times on the serial computing when using the proposed parallel algorithms. For the calculations, we used mesh data 4 and the hybrid MPI+OpenMP version with 8 threads on FX10, which showed the best result of experiments in previous subsections. We gained a fairly good parallel scalability for total simulation. About 150-fold speed-up was obtained by using 256 cores. If we could also improve the parallel scalability of arithmetic in BiCGSTAB except for HMVM and the other processing, we can attain still better parallel scalability.

In the case of the simulation, the parallel scalability strongly depends on the performance of the H-matrix construction. The performance of HMVM becomes significant when we deal with the simulation such that the HMVM is performed hundreds of thousands of times, e.g. the earthquake cycle simulation [10].

5. Conclusions and Future Work

In this paper, we discuss schemes for H-matrices with ACA on SMP cluster systems. We propose a set of parallel algorithms for H-matrices with ACA, which consist of algorithms that construct H-matrices and perform HMVM. The proposed algorithms can be applied to arbitrary H-matrices without any restriction on the structures of H-matrices.

The proposed algorithms were implemented by using MPI and OpenMP. The Hacapk library includes implementations that use the flat-MPI and the hybrid MPI+OpenMP programming models. We examine the performance of Hacapk using an electric static field analysis, based on the surface charge method. In serial computing, the Hacapk library performed better than the existing H-matrices software, HLib. We developed the Hacapk library from scratch and so far have not utilized any existing libraries for linear algebra computations. In order to attain further speed-up, we have a plan to utilize other libraries, for example, BLAS for arithmetic in HMVM.

On an SMP cluster system, the flat-MPI version suffered from a saturation of speed-up in HMVM, though it showed a rather good parallel scalability when constructing the H-matrix. Because the saturation was believed to be caused by the MPI communication cost, we decided to develop an H-matrix library based on the hybrid MPI+OpenMP parallel programming model. The distributed parallel HMVM requires collective communication among processes, and the reduced number of processes results in a decrease in the MPI communication cost. In the hybrid parallel version, the number of processes is reduced by introducing multi-thread parallelization in each SMP node. We investigated the effect of using the hybrid MPI+OpenMP programming model for HMVM on two SMP cluster systems: Fujitsu FX10 at The University of Tokyo, and Appro GreenBlade 8000 at Kyoto University. On both systems, the parallel scalability is significantly improved from the flat-MPI version by using a hybrid MPI+OpenMP version. The best result is observed when eight OpenMP threads are used in each SMP node. In these cases, the parallel performance increased up to 256 cores, which was the maximum used in our experiments.

If we used many more cores in the analyses, even Hacapk based on the hybrid MPI+OpenMP programming model would reach a speed-up ceiling. To overcome this problem, we believe it would be necessary to introduce other parallel processing techniques to further reduce the communication cost. The overlap of the computation and communication can be considered to hide the communication latency. Moreover, it may be possible to improve the algorithm for HMVM in an SMP node by using information about the cluster tree that constructs the H-matrices. Another key issue is avoiding conflicts in writing the result of HMVM to OpenMP threads. In the future, we will enhance Hacapk to tackle these problems.

Acknowledgments The authors wish to thank Dr. T. Hiraishi, the Kyoto University, for his useful discussions.

References

[1] Hackbusch, W.: A Sparse Matrix Arithmetic Based on H-matrices. I. Introduction to H-matrices, *Computing*, Vol.62, No.2, pp.89–108 (1999).

[2] Hackbusch, W. and Khoromskij, B.N.: A Sparse H-matrix Arithmetic. II. Application to Multi-dimensional Problems, *Computing*, Vol.64, No.1, pp.21–47 (2000).

[3] Börm, S., Grasedyck, L. and Hackbusch, W.: Hierarchical Matrices, Lecture Note, Max-Planck-Institut fur Mathematik (2006).

[4] Greengard, L. and Rokhlin, V.: A New Version of the Fast Multipole Method for the Laplace Equation in Three Dimensions, *Acta Numerica*, Vol.6, pp.229–269 (1997).

[5] Cheng, H., Greengard, L. and Rokhlin, V.: A Fast Adaptive Multipole Algorithm in Three Dimensions, *J. Comput. Phys.*, Vol.155, pp.468–498 (1999).

[6] Barnes, J. and Hut, P.: A Hierarchical O($N \log N$) Force-calculation Algorithm, *Nature*, Vol.324, No.4, pp.446–449 (1986).

[7] Kurtz, S., Rain, O. and Rjasanow, S.: The Adaptive Cross-Approximation Technique for the 3-D Boundary-Element Method, *IEEE Trans. Magn.*, Vol.38, No.2, pp.421–424 (2002).

[8] Bebendorf, M. and Rjasanow, S.: Adaptive Low-Rank Approximation of Collocation Matrices, *Computing*, Vol.70, No.1, pp.1–24 (2003).

[9] Takahashi, Y.: Large-Scale Electromagnetic Field Analysis by Using Integral Equation Method for Practical Electric Machine Design, PhD thesis, Waseda University (2008).

[10] Ohtani, M. et al.: Fast Computation of Quasi-Dynamic Earthquake Cycle Simulation with Hierarchical Matrices, *Procedia Computer Science*, Vol.4, pp.1456–1465 (2011).

[11] Ostrowski, J., Andjelic, Z. and Bebendorf, M.: Fast BEM-Solution of Laplace Problems with H-Matrices and ACA, *IEEE Trans. Magn.*, Vol.42, No.4, pp.627–630 (2006).

[12] Popović, N. and Praetorius, D.: H-Matrix Techniques for Stray-field Computations in Computational Micromagnetics, *Large-Scale Scientific Computing Lecture Notes in Computer Science*, Vol.3743, pp.102–110 (2006).

[13] Busch, I., Ernst, O. and Ullmann, E.: Expansion of Random Field Gradients Using Hierarchical Matrices, *PAMM Proc. Appl. Math. Mech.*, Vol.11, pp.91–914 (2011).

[14] Benner, P. and Mach, T.: Locally Optimal Block Preconditioned Conjugate Gradient Method for Hierarchical Matrices, *PAMM Proc. Appl. Math. Mech.*, Vol.11, pp.741–742 (2011).

[15] Kriemann, R.: Parallel H-Matrix Arithmetics on Shared Memory Systems, *Computing*, Vol.74, pp.273–297 (2005).

[16] Bebendorf, M. and Kriemann, R.: Fast Parallel Solution of Boundary Integral Equations and Related Problems, *Comput. Visual Sci.*, Vol.8, pp.121–135 (2005).

[17] Börm, S. and Bendoraityte, J.: Distributed H^2-matrices for Non-local Operators, *Comput. Visual Sci.*, Vol.11, pp.237–249 (2008).

[18] Börm, S. and Grasedyck, L.: Hierarchical Matrices, available from ⟨http://www.hlib.org⟩.

Akihiro Ida was born in Japan in 1971. He received his B.Math and M.E. degrees from Nagoya University in 1994 and 1996, respectively. In 2008, Chuo University awarded him a Ph.D. degree in mathematics. He researched and developed linear solvers at VINAS Co., Ltd. He currently engages in a Japanese national project of JST CREST as an assistant professor in ACCMS, Kyoto University. His research interests include discretization methods for integro-differential equations, magnetohydrodynamics and high performance computing.

Takeshi Iwashita was born in 1971. He received a B.E., an M.E., and a Ph.D. from Kyoto University in 1992, 1995, and 1998, respectively. In 1998–1999, he worked as a post-doctoral fellow of the JSPS project in the Graduate School of Engineering, Kyoto University. He moved to the Data Processing Center of the same university in 2000. In 2003–2014, he worked as an associate professor in the Academic Center for Computing and Media Studies, Kyoto University. He currently works as a professor in the Information Initiative Center, Hokkaido University. His research interests include high performance computing, linear iterative solver, and electromagnetic field analysis. He is a member of IEEE, SIAM, IPSJ, IEEJ, JSIAM, JSCES, and JSAEM.

Takeshi Mifune received his Ph.D. degree in engineering from Kyoto University, Japan in 2003. Since 2003, he has been an assistant professor at the Department of Electrical Engineering, Kyoto University. His research interests are in computational electromagnetics, electromagnetic finite element analysis, and parallel computing.

Yasuhito Takahashi was born in Chiba, Japan, on September 20, 1980. He received his B.Eng., M.Eng. and Ph.D. degrees in 2003, 2005 and 2008, respectively, from Waseda University, Tokyo, Japan. From 2006 to 2008, he worked as a research associate in Faculty of Science and Engineering, Waseda University. From 2008 to 2010, he was a Global COE assistant professor in Department of Systems Science, Graduate School of Informatics, Kyoto University. Since 2010, he has been an assistant professor at the Department of Electrical Engineering, Doshisha University. His major fields of interest are large-scale electromagnetic field computation, modeling of magnetic properties and photovoltaic power generation system.