Applied Mathematics and Computation 359 (2019) 17–33

Contents lists available at ScienceDirect

Applied Mathematics and Computation

journal homepage: www.elsevier.com/locate/amc

Fast direct isogeometric boundary element method for 3D potential problems based on HODLR matrix

F.L. Sun, C.Y. Dong*, Y.H. Wu, Y.P. Gong

Department of Mechanics, School of Aerospace Engineering, Beijing Institute of Technology, Beijing 100081, China

ARTICLE INFO

Keywords:
Fast direct solver
Isogeometric boundary element method
Accelerated adaptive cross approximation
3D potential problems

ABSTRACT

A novel fast direct solver based on isogeometric boundary element method (IGABEM) is presented for solving 3D potential problems, which uses the hierarchical off-diagonal low-rank (HODLR) matrix structure arising from the discretization of boundary integral equations. Since the HODLR matrix can be factored into the product form of some diagonal blocks, we can use the Sherman–Morrison–Woodbury formula to solve the inverse of a HODLR matrix efficiently. For large scale problems, an accelerated adaptive cross approximation algorithm is developed to decompose the off-diagonal submatrices. In numerical implementation, bivariate NURBS basis functions are used to describe the geometry. Meanwhile, the same NURBS basis functions are also used to approximate the unknown boundary quantities. The present method is applied to some numerical examples, including an infinite space containing twenty spherical cavities. The numerical results clearly show that the fast direct solver developed in the paper can obtain accurate results with less CPU time.

1. Introduction

In recent years, the isogeometric analysis (IGA) [1] has been applied in several fields including structural mechanics, solid mechanics, fluid mechanics and contact mechanics. The IGA utilizes the NURBS which are used by CAD to describe the complex geometries exactly, instead of Lagrange polynomial, and the same basis functions are also used to approximate the unknown fields. A vast array of literature has focused on the application of IGA combined with finite element method (FEM). In recent years, the combination of IGA and boundary element method (BEM) known as IGABEM has been widely concerned and applied successfully in various fields [2–6]. The IGABEM contains the advantages of the conventional BEM, such as reducing the dimensionality of the problem by one, and can provide accurate geometric boundaries. It is well known that although the IGABEM only needs to discretize the boundary of domain, which contains N degrees of freedom, we require $O(N^2)$ operations to generate the $N \times N$ coefficient matrix which is dense and unsymmetrical like the conventional BEM. And $O(N^3)$ operations are needed to solve the linear system of equations. Therefore, storage memory and CPU time increase rapidly as the problem size increases.

Over the past thirty years, the fast multipole method (FMM) [7,8], Fourier transformation based methods [9] and hierarchical matrices (H-matrices) [10,12,13] have been developed to overcome the complexity barrier. Up to now, some researchers have applied the fast matrix-vector product techniques to analyze various problems of IGABEM. González et al. [11] first used the FMM with NURBS surface to analyze the scattering of complex bodies. One of the main problems of the

* Corresponding author.
E-mail address: cydong@bit.edu.cn (C.Y. Dong).

https://doi.org/10.1016/j.amc.2019.04.030
0096-3003/

FMM is to adopted explicitly multipole expansions of the kernel functions which are not trivial especially for anisotropic problems. Unlike FMM, the H-matrices are fully developed from the matrix level and thus do not need to consider the multipole expansions of the kernel functions. So if the dense matrix is divided into hierarchical submatrices, the adaptive cross approximation (ACA) algorithm can be used to approximate the dense submatrices which capture the admissibility condition by low rank matrices [12]. However, it is well known that both FMM and H-matrices are based on iterative techniques, and the number of iterations depends largely on the issues considered. The larger the condition number of the coefficient matrix is, the slower the convergence rate is. Consequently, we need more CPU time and more memory. Recently, tremendous effort has been devoted to develop fast direct solver for dense linear systems from boundary integral equations [14–18]. For example, Lai et al. [16] presented a method with HODLR matrix to directly solve the high frequency scattering from a large cavity in two dimensions. Huang and Liu [17] also developed a fast direct solver with HODLR matrix to solve 3D potential problem by conventional BEM. Based on HODLR matrix, we can avoid using iterative techniques to solve the linear system of equations. Since the off-diagonal submatrices of the HODLR matrix are low rank matrices, the inverse of the matrix can be provided by the Sherman–Morrison–Woodbury formula [19].

As mentioned above, some literatures have already used the FMM and H-matrices with NURBS basis functions to solve the linear systems from boundary integral equations by iterative techniques. To the authors' best knowledge, there is no report in the literature on the fast direct solver based on the combination of HODLR matrix and IGABEM. Nevertheless, if we directly utilize the ACA decomposition to obtain the low rank approximation of the off-diagonal submatrices, the rank of the low rank matrices may be large and the computational time may be long, which result in low solving efficiency, especially for large scale problems since singular values of the matrix may not decay very quickly in magnitude. In this paper, we present an efficient fast direct solver based on HODLR matrix with accelerated ACA algorithm and the IGABEM with NURBS basis functions for 3D potential problems. In numerical implementation, Bézier extraction is used to facilitate the combination of IGA and existing boundary element codes. At same time, it is more convenient to construct binary tree of surface elements by using discontinuous potentials and normal derivatives on the boundary surfaces.

The paper is organized in the following sections. In Section 2, the boundary integral equation for potential problems is presented. In Section 3, the HODLR matrix is described and the fast direct solver algorithm is introduced. In Section 4, we apply the fast direct solver to the IGABEM and give the accelerated ACA algorithm to decrease the CPU time of ACA decomposition. In Section 5, numerical examples are presented to demonstrate the accuracy and efficiency of the accelerated algorithm in the fast direct solver. Finally, our main conclusions are summarized in Section 6.

2. Boundary integral equation and discretization

In this section, we consider the boundary value problem for Laplace's equation via IGABEM. Assumed that Ω is a bounded domain, Ω^c is its open complement, and Γ is their common boundary.

2.1. Boundary integral equation

For 3D interior potential problem of the domain Ω with boundary Γ, in the absence of body source, the boundary integral equation is given as [20–23]

$$c(\boldsymbol{y})u(\boldsymbol{y}) = \int_\Gamma q(\boldsymbol{x})u^*(\boldsymbol{x},\boldsymbol{y})d\Gamma(\boldsymbol{x}) - \int_\Gamma u(\boldsymbol{x})q^*(\boldsymbol{x},\boldsymbol{y})d\Gamma(\boldsymbol{x}) \tag{1}$$

where $u(\boldsymbol{x})$ and $q(\boldsymbol{x})$ are the potential and normal derivative of $u(\boldsymbol{x})$ at the field point $\boldsymbol{x}$, respectively. $u(\boldsymbol{y})$ is the potential at the source point $\boldsymbol{y}$, and $c(\boldsymbol{y})$ is the geometric coefficient of the source point $\boldsymbol{y}$. $u^*(\boldsymbol{x},\boldsymbol{y})$ and $q^*(\boldsymbol{x},\boldsymbol{y})$ are fundamental solutions which are defined as

$$u^*(\boldsymbol{x},\boldsymbol{y}) = \frac{1}{4\pi r} \tag{2a}$$

$$q^*(\boldsymbol{x},\boldsymbol{y}) = \frac{\partial u^*(\boldsymbol{x},\boldsymbol{y})}{\partial \boldsymbol{n}_{\boldsymbol{x}}} \tag{2b}$$

where r is the distance between the field point $\boldsymbol{x}$ and the source point $\boldsymbol{y}$. $\boldsymbol{n}$ is the unit outward normal vector at point $\boldsymbol{x}$.

2.2. IGABEM

Based on the IGA concept, we use the NURBS basis functions applied to describe the problem geometry to approximate the boundary quantities. We first give a brief overview over the construction of NURBS basis functions as shown below. The interested readers can find a more detailed explanation in [1,2].

Given a non-decreasing knot vector $\Xi=\{\xi_1, \xi_2,\ldots, \xi_{n+p+1}\}$, B-spline basis functions of degree p, $N_{i,p}$ with $1 \le i \le n$ are defined recursively starting with the zeroth order basis function, namely $p=0$, as

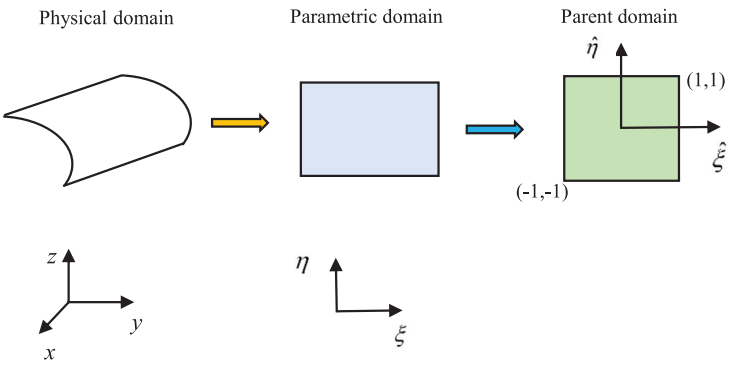


Fig. 1. Element defined in the physical domain, parametric domain and parent domain.

$$N_{i,0} = \begin{cases} 1 & \text{if } \xi_i \le \xi < \xi_{i+1}, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

and for $p = 1,2,3,\ldots$

$$N_{i,p}(\xi) = N_{i,p-1}(\xi)(\xi - \xi_i)/\left(\xi_{i+p} - \xi_i\right) + N_{i+1,p-1}(\xi)\left(\xi_{i+p+1} - \xi\right)/\left(\xi_{i+p+1} - \xi_{i+1}\right) \tag{4}$$

where n is the number of basis functions, ξ_i represents ith knot point. Based on the B-spline, the NURBS basis functions are defined as

$$R_{i,p}(\xi) = N_{i,p}(\xi)\omega_i \Big/ \left(\sum_{j=1}^{n} N_{j,p}(\xi)\omega_j\right) \tag{5}$$

where $N_{i,p}(\xi)$ is the ith B-spline basis function, and ω_i is the positive weight associated with the ith control point.

Based on the NURBS basis functions, if there are two knot vectors $\Xi = \{\xi_1, \xi_2, \ldots, \xi_{n+p+1}\}$ and $\mathscr{H} = \{\eta_1, \eta_2, \ldots, \eta_{m+q+1}\}$ as well as bidirectional net of control points $\boldsymbol{P}_{i,j}$ $(i = 1,2,\ldots,n;\ j = 1,2,\ldots,m)$, the NURBS surface is given by the tensor product of the NURBS basis functions in the parametric domain $[\xi_1, \xi_{n+p+1}] \times [\eta_1, \eta_{m+q+1}]$ as follows

$$S(\xi, \eta) = \sum_{i=1}^{n}\sum_{j=1}^{m} R_{i,j}^{p,q}(\xi, \eta)\boldsymbol{P}_{i,j} \tag{6}$$

and the bivariate basis functions $R_{i,j}^{p,q}(\xi, \eta)$ are as follows

$$R_{i,j}^{p,q}(\xi, \eta) = \frac{N_{i,p}(\xi)M_{j,q}(\eta)\omega_{i,j}}{\sum_{\hat{i}=1}^{n}\sum_{\hat{j}=1}^{m} N_{\hat{i},p}(\xi)M_{\hat{j},q}(\eta)\omega_{\hat{i},\hat{j}}} \tag{7}$$

where $N_{i,p}$ and $M_{j,q}$ are respectively the B-spline basis functions in ξ and η directions. $\omega_{i,j}$ is the weight related to the control point $\boldsymbol{P}_{i,j}$.

In the present paper, the bivariate basis functions are applied to build the boundary surface and the physical quantities. Then we split the integrals over the entire physical domain into elemental integrals. After discretizing the boundary surface, the physical elemental domain is mapped to the non-zero knot intervals $[\xi_i, \xi_{i+1}] \times [\eta_j, \eta_{j+1}]$, and the parameter element is linearly mapped to the parent element $[-1, 1] \times [-1, 1]$, as shown in Fig. 1.

The local basis function defined on the parameter element is given as

$$N_b^e(\xi, \eta) = R_a^{p,q}(\xi, \eta) \tag{8}$$

where b is the local basis function number, e is the element number. a=conn(e, b), where conn() is a connectivity function. The geometry boundary, potential and normal derivative of potential can be interpolated by Eq. (8) as

$$\boldsymbol{x}^e(\xi, \eta) = \sum_{b=1}^{(p+1)(q+1)} N_b^e(\xi, \eta)\boldsymbol{x}_b \tag{9a}$$

$$u^e(\xi, \eta) = \sum_{b=1}^{(p+1)(q+1)} N_b^e(\xi, \eta)d_b \tag{9b}$$

$$q^e(\xi, \eta) = \sum_{b=1}^{(p+1)(q+1)} N_b^e(\xi, \eta)t_b \tag{9c}$$

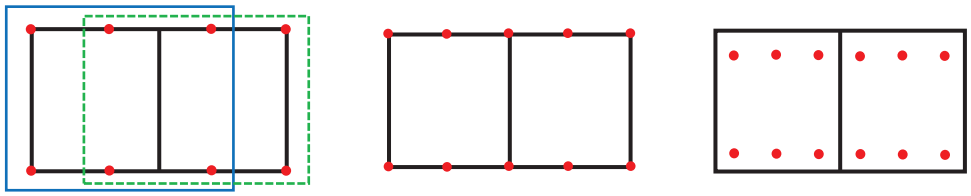


Fig. 2. (a) The distribution of the collocation points on elements with knot vectors Ξ={0,0,0,0.5,1,1,1} and $\mathscr{H}$={0,0,1,1}. (b) The collocation points on Bézier elements with knot vectors Ξ={0,0,0,0.5,0.5,1,1,1} and $\mathscr{H}$={0,0,1,1}. (c) The collocation points based on (b) are moved inside the elements.

in which $\boldsymbol{x}_b$ are the coordinates of control points. d_b and t_b are respectively the potential and normal derivative parameters at corresponding control points.

From Fig. 1, we can see that the conversion relationship between parametric element and parent element is $\xi = (1-\hat{\xi})\xi_i/2 + (1+\hat{\xi})\xi_{i+1}/2,\ \xi \in [\xi_i, \xi_{i+1}]$ and $\eta = (1-\hat{\eta})\eta_j/2 + (1+\hat{\eta})\eta_{j+1}/2,\ \eta \in [\eta_j, \eta_{j+1}]$, so the standard Gauss–Legendre rule can be used to compute the final integral. Then the discretizing forms of the boundary integral equations are given as

$$c\sum_{l=1}^{(p+1)(q+1)} N_l(\hat{\xi}_s, \hat{\eta}_s)d_l = \sum_{e=1}^{NE}\sum_{l=1}^{(p+1)(q+1)} U_l t_l - \sum_{e=1}^{NE}\sum_{l=1}^{(p+1)(q+1)} T_l d_l \tag{10}$$

where NE is the element numbers, and

$$U_l = \int_{-1}^{1}\int_{-1}^{1} u^*(\boldsymbol{s}, \boldsymbol{x}(\hat{\xi}, \hat{\eta}))N_l^e(\hat{\xi}, \hat{\eta})J(\hat{\xi}, \hat{\eta})d\hat{\xi}d\hat{\eta}$$

$$T_l = \int_{-1}^{1}\int_{-1}^{1} q^*(\boldsymbol{s}, \boldsymbol{x}(\hat{\xi}, \hat{\eta}))N_l^e(\hat{\xi}, \hat{\eta})J(\hat{\xi}, \hat{\eta})d\hat{\xi}d\hat{\eta}$$

where $J(\hat{\xi}, \hat{\eta}) = |\frac{\partial \vec{r}}{\partial \hat{\xi}} \times \frac{\partial \vec{r}}{\partial \hat{\eta}}|$ is the Jacobian in which $\vec{r}$ is the position vector of the field point. $\boldsymbol{s}$ is the collocation point. $\hat{\xi}_s$ and $\hat{\eta}_s$ are the local coordinates of the collocation point $\boldsymbol{s}$. In this paper, the weakly singularity and strongly singularity of the boundary integral equation are computed by the power series expansion method [24].

By rearranging Eq. (10) into matrix form with respect to the boundary condition, the Eq. (10) can be written as

$$\begin{pmatrix}-\boldsymbol{G}_1 & \boldsymbol{H}_2\end{pmatrix}\begin{pmatrix}\boldsymbol{t}_1 \\ \boldsymbol{d}_2\end{pmatrix} = \begin{pmatrix}-\boldsymbol{H}_1 & \boldsymbol{G}_2\end{pmatrix}\begin{pmatrix}\bar{\boldsymbol{d}}_1 \\ \bar{\boldsymbol{t}}_2\end{pmatrix} \tag{11}$$

Then the final equation is written as

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{12}$$

where $\bar{\boldsymbol{d}}_1$ and $\bar{\boldsymbol{t}}_2$ are given values at the control points. $\boldsymbol{d}_2$ and $\boldsymbol{t}_1$ are unknown values at the control points. And $\boldsymbol{A}$=$\begin{pmatrix}-\boldsymbol{G}_1 & \boldsymbol{H}_2\end{pmatrix}$, $\boldsymbol{x}$=$\begin{pmatrix}\boldsymbol{t}_1 \\ \boldsymbol{d}_2\end{pmatrix}$, and $\boldsymbol{b}$=$\begin{pmatrix}-\boldsymbol{H}_1 & \boldsymbol{G}_2\end{pmatrix}\begin{pmatrix}\bar{\boldsymbol{d}}_1 \\ \bar{\boldsymbol{t}}_2\end{pmatrix}$.

2.3. Collocation point strategy

In IGA, the collocation points are obtained by the Greville abscissae definition [25], which is defined as (see Fig. 2(a))

$$\xi_i' = \left(\xi_{i+1} + \xi_{i+2} + \cdots + \xi_{i+p}\right)/p \qquad i = 1, 2, ..., n \tag{13a}$$

$$\eta_j' = \left(\eta_{j+1} + \eta_{j+2} + \cdots + \eta_{j+q}\right)/q \qquad j = 1, 2, ..., m \tag{13b}$$

From Fig. 2(a), we can see that there are two elements and the two frames contain the control points belonging to two elements, respectively, which is different from the convention BEM. In order to facilitate the IGA to use in the existing boundary element codes, we here use the Bézier extraction technique [26,27] which has been considered as an efficient method of the IGA. The collocation points are shown in Fig. 2(b). Since normal derivative of potential is not unique at the corner points, discontinuous element is chosen to avoid this problem. In other words, the collocation points based on Fig. 2(b) are moved inside the elements, as shown in Fig. 2(c).

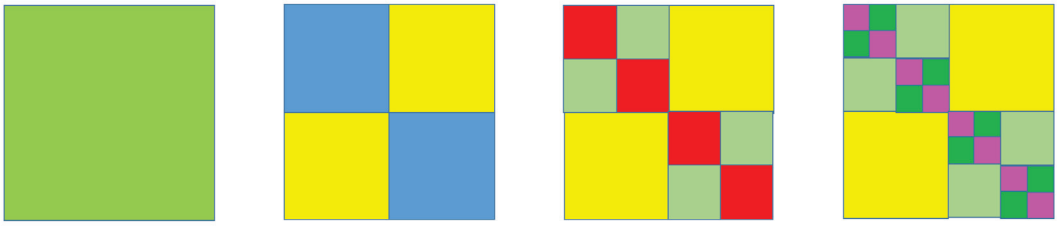


Fig. 3. HODLR matrices at different.

3. Fast algorithm of HODLR matrices

As mentioned above, H-matrix plays an important role in fast direct solver. In this section, we describe a scheme to solve Eq. (12) using the Sherman–Morrison–Woodbury formula [19]. To construct the inverse of matrix, the concept of HODLR matrix is introduced, which is one type of H-matrix. Now, we briefly outline the HODLR matrix as follows.

3.1. One-level HODLR matrix

Consider a matrix $\boldsymbol{A} \in \mathbb{R}^{(n+m)\times(n+m)}$, the one-level ($r=1$) scheme is written as (see Fig. 3)

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{D}_1 & \boldsymbol{O}_1 \\ \boldsymbol{O}_2 & \boldsymbol{D}_2 \end{bmatrix} \tag{14}$$

where $\boldsymbol{D}_1 \in \mathbb{R}^{n\times n}$ and $\boldsymbol{D}_2 \in \mathbb{R}^{m\times m}$, $\boldsymbol{O}_1 \in \mathbb{R}^{n\times m}$ and $\boldsymbol{O}_2 \in \mathbb{R}^{m\times n}$. And the two off-diagonal matrices $\boldsymbol{O}_1$ and $\boldsymbol{O}_2$ are divided into $\boldsymbol{U}_1 \in \mathbb{R}^{n\times k_1}$, $\boldsymbol{V}_1 \in \mathbb{R}^{k_1\times m}$ and $\boldsymbol{U}_2 \in \mathbb{R}^{m\times k_2}$, $\boldsymbol{V}_2 \in \mathbb{R}^{k_2\times n}$, respectively, namely $\boldsymbol{O}_1=\boldsymbol{U}_1\boldsymbol{V}_1$ and $\boldsymbol{O}_2=\boldsymbol{U}_2\boldsymbol{V}_2$. During the decomposition process, the ACA is utilized to obtain the low-rank decomposition of the off-diagonal submatrices. Thus when $k_1 \ll \min(m, n)$ and $k_2 \ll \min(m, n)$, we can apply Sherman–Morrison–Woodbury formula to construct matrix $\boldsymbol{A}^{-1}$ effectively. The detail description is shown in Algorithm 1, which is slightly different from the algorithm in Ref. [18], that is, this algorithm adopts the pivoted Gram–Schmidt algorithm to construct low-rank decomposition.

Algorithm 1.

$$\boldsymbol{A}_1 = \begin{bmatrix} \boldsymbol{D}_1 & \\ & \boldsymbol{D}_2 \end{bmatrix} \tag{15}$$

Step 1. Compute and store $\boldsymbol{D}_1^{-1}$ and $\boldsymbol{D}_2^{-1}$, then

$$\boldsymbol{A}_1^{-1} = \begin{bmatrix} \boldsymbol{D}_1^{-1} & \\ & \boldsymbol{D}_2^{-1} \end{bmatrix} \tag{16}$$

Step 2. Decompose the off-diagonal matrices using ACA.
Step 3. Calculate $\boldsymbol{D}_1^{-1}\boldsymbol{U}_1$ and $\boldsymbol{D}_2^{-1}\boldsymbol{U}_2$, then set $\tilde{\boldsymbol{U}}_1=\boldsymbol{D}_1^{-1}\boldsymbol{U}_1$ and $\tilde{\boldsymbol{U}}_2=\boldsymbol{D}_2^{-1}\boldsymbol{U}_2$, we have

$$\boldsymbol{A}_0 = \begin{bmatrix} I & \tilde{\boldsymbol{U}}_1\boldsymbol{V}_1 \\ \tilde{\boldsymbol{U}}_2\boldsymbol{V}_2 & I \end{bmatrix} \tag{17}$$

So $\boldsymbol{A}$ is factored as $\boldsymbol{A}=\boldsymbol{A}_1\boldsymbol{A}_0$. Assuming

$$\boldsymbol{U} = \begin{bmatrix} \tilde{\boldsymbol{U}}_1 & \\ & \tilde{\boldsymbol{U}}_2 \end{bmatrix}, \boldsymbol{V} = \begin{bmatrix} & \boldsymbol{V}_1 \\ \boldsymbol{V}_2 & \end{bmatrix} \tag{18}$$

in which $\boldsymbol{U} \in \mathbb{R}^{(n+m)\times(k_1+k_2)}$, $\boldsymbol{V} \in \mathbb{R}^{(k_1+k_2)\times(n+m)}$. Then Eq. (17) can be rewritten as $\boldsymbol{A}_0 = I + \boldsymbol{U}\boldsymbol{V}$, and compute $\boldsymbol{A}_0^{-1}=\boldsymbol{I}-\boldsymbol{U}(\boldsymbol{I}+\boldsymbol{V}\boldsymbol{U})^{-1}\boldsymbol{V}$ by Sherman–Morrison–Woodbury formula.
Step 4. Compute $\boldsymbol{A}^{-1}=\boldsymbol{A}_0^{-1}\boldsymbol{A}_1^{-1}$.

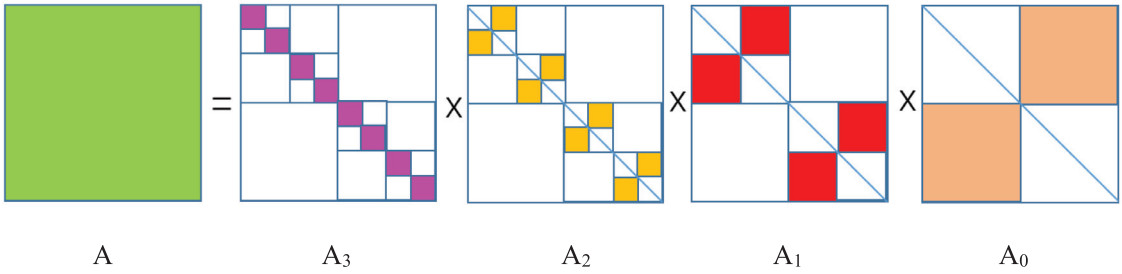


Fig. 4. The factorization process of the three levels.

3.2. Multi-level HODLR matrix

The one-level HODLR matrix is applied recursively to obtain a multi-level HODLR matrix. To illustrate this point, two level ($r=2$) scheme can be written as

$$\boldsymbol{A}=\begin{bmatrix}\begin{bmatrix}\boldsymbol{D}_{21} & \boldsymbol{U}_{21}\boldsymbol{V}_{21}\\ \boldsymbol{U}_{22}\boldsymbol{V}_{22} & \boldsymbol{D}_{22}\end{bmatrix} & \boldsymbol{U}_1\boldsymbol{V}_1\\ \boldsymbol{U}_2\boldsymbol{V}_2 & \begin{bmatrix}\boldsymbol{D}_{23} & \boldsymbol{U}_{23}\boldsymbol{V}_{23}\\ \boldsymbol{U}_{24}\boldsymbol{V}_{24} & \boldsymbol{D}_{24}\end{bmatrix}\end{bmatrix} \tag{19}$$

where $\boldsymbol{D}_{ij}$, $(i=2;\ j=1,2,\ldots,2i)$ are new square diagonal submatrices, and i denote the number of level. $\boldsymbol{U}_{ij}$ and $\boldsymbol{V}_{ij}$, $(i=2;\ j=1,2,\ldots,2i)$ are new off-diagonal submatrices, as shown in Fig. 3. Fig. 3 depicts the HODLR matrices for three levels. From these figures, we can see that for level $r=1, 2, \ldots, \boldsymbol{R}$, there are 2^r new diagonal matrices and 2^r new off-diagonal matrices in each level. We assume that the diagonal matrices on each level are invertible and all off-diagonal matrices are low-rank matrices by using ACA decomposition which is presented in detail in Section 4. Therefore, based on Algorithm 1, matrix $\boldsymbol{A}$ can be factored as

$$\boldsymbol{A}=\boldsymbol{A}_R\ldots\boldsymbol{A}_1\boldsymbol{A}_0 \tag{20}$$

in which $\boldsymbol{A}_i$, $i=0, 1, \ldots, R\text{-}1$, are the matrices with 2^i diagonal sub-matrices, respectively, and each diagonal submatrix is an identity matrix with low-rank approximation. When $i=\boldsymbol{R}$, $\boldsymbol{A}_{\boldsymbol{R}}$ is a diagonal matrix consisting of $2^{\boldsymbol{R}}$ diagonal submatrices. Fig. 4 describes the factorization process of the above three levels, namely $\boldsymbol{R}=3$. By recursive application of the Sherman–Morrison–Woodbury formula, the invert of $\boldsymbol{A}$ can be obtained. A detailed algorithm is given in Appendix A, i.e. Algorithm 2.

4. Fast algorithm of isogeometric boundary integral equations

In order to use the algorithm mentioned in Section 3 to solve Eq. (12), a HODLR matrix is needed. In references [16,18,28], we know that if the matrix is generated by BIE for potential theory, it commonly has a HODLR structure. Hence, when the matrix $\boldsymbol{A}$ is formed by using Eq. (10), we can exploit the geometric structure to obtain a hierarchical matrix. In the context of HODLR matrix, the procedure takes advantage of a binary tree. First, for the root the whole boundary elements are divided into two clusters, i.e. $r=1$, and the off-diagonal submatrices in this level are $\boldsymbol{O}_1$ and $\boldsymbol{O}_2$. The off-diagonal matrices are decomposed by using ACA. So, for any off-diagonal matrix $\boldsymbol{O}\in\mathbb{R}^{n\times m}$ which requires $n\times m$ entries storage scales for the initial matrix, only $k\times(n+m)$ entries need to be stored. Second, two new boundary element clusters are again divided into

Algorithm 2.

Step 0. $\boldsymbol{B}=\boldsymbol{A}$, set level=0.
Step 1.
(1) If the first visit to tree node $\boldsymbol{B}$, go to step 2.
(2) If the second visit to tree node $\boldsymbol{B}$, go to step 3.
Step 2.
(1) If level=$\boldsymbol{R}$-1, invert the left and right children $\boldsymbol{D}_1$ and $\boldsymbol{D}_2$ of $\boldsymbol{B}$. Store $\boldsymbol{D}_1{}^{-1}$ and $\boldsymbol{D}_2{}^{-1}$. Go to step 1.
(2) If level<$\boldsymbol{R}$-1, pass down to its left child and update $\boldsymbol{B}$ to its left child, and level=level+1. Go back to step1.
Step 3.
(1) Decompose the off-diagonal matrices using ACA. Then calculate $\boldsymbol{D}_1{}^{-1}\boldsymbol{U}_1$ and $\boldsymbol{D}_2{}^{-1}\boldsymbol{U}_2$, then set $\tilde{\boldsymbol{U}}_1=\boldsymbol{D}_1{}^{-1}\boldsymbol{U}_1$ and $\tilde{\boldsymbol{U}}_2=\boldsymbol{D}_2{}^{-1}\boldsymbol{U}_2$, and compute $\boldsymbol{C}=(\boldsymbol{I}+\boldsymbol{V}\boldsymbol{U})^{-1}$, where $\boldsymbol{U}$ is as in (4)-(4), and $\boldsymbol{V}$ in (4)-(4).
(2) If level=0, we are done. Otherwise, go to step 4.
Step 4.
(1) If $\boldsymbol{B}$ is a left child of its parent, update $\boldsymbol{B}$ to its parent's right child. Go back to step 1.
(2) If $\boldsymbol{B}$ is a left child of its parent, update $\boldsymbol{B}$ to its parent, and level=level-1. Go back to step 1.

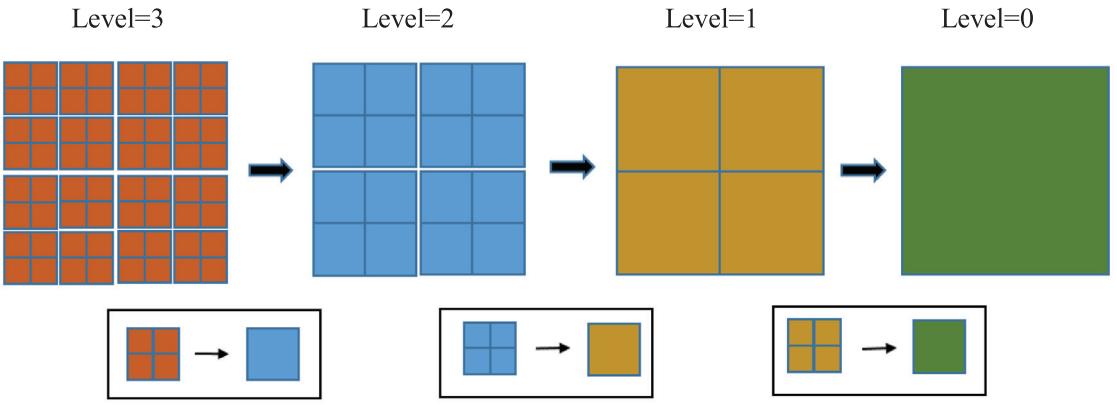


Fig. 5. The sketch of the accelerated algorithm.

two clusters, respectively, i.e. $r=2$. And two new off-diagonal low-rank submatrices are generated by each decomposition. This procedure continues until $r=\boldsymbol{R}$. As mentioned above, there is no need to build the whole matrix entries beforehand. And the off-diagonal submatrices are stored in low-rank form.

The main idea of ACA, e.g. $\boldsymbol{O} \in \mathbb{R}^{n\times m}$, is that we choose a tolerance ε to control how many evaluations are necessary, and the product of matrices $\boldsymbol{U} \in \mathbb{R}^{n\times k}$ and $\boldsymbol{V} \in \mathbb{R}^{k\times m}$ creates a matrix $\boldsymbol{S}_k \in \mathbb{R}^{n\times m}$ with the rank k. Then we have

$$\boldsymbol{O}=\boldsymbol{S}_k+\boldsymbol{R}_k \text{ and } \|\boldsymbol{R}_k\|_F \le \varepsilon\|\boldsymbol{O}\|_F \tag{21}$$

Here, $\boldsymbol{R}_k$ is the residual matrix, and $\|\cdot\|_F$ is the Forbenius norm. As mentioned earlier, the desired value k is small because it not only saves storage space, but also makes more efficient use of the Sherman–Morrison–Woodbury formula. Nevertheless, if we implement the ACA algorithm directly on the off-diagonal submatrix $\boldsymbol{O}$, the ACA decomposition may be slow, especially for large scale matrices. Since the singular values may not decay exponentially, the calculation time increases. Literature [12,29,30] presented a method to solve the elastic problems by using spatial direction decomposition of Cauchy data, which led to a blocked scheme to improve the ACA algorithm in H-matrices. Inspired by this method, the off-diagonal submatrix $\boldsymbol{O}$ can be decomposed by use of the blocked scheme to accelerate the ACA decomposition. The accelerated algorithm is shown below.

4.1. One-level accelerated algorithm

For the sake of simplicity, we first consider one-level HODLR submatrix $\boldsymbol{O}_1 \in \mathbb{R}^{n\times m}$ to discuss the accelerated algorithm. The submatrix $\boldsymbol{O}_1$ is broken down into the following form (see Fig. 5, level=1)

$$\boldsymbol{O}_1=\begin{bmatrix}\boldsymbol{B}_{11} & \boldsymbol{B}_{12}\\ \boldsymbol{B}_{13} & \boldsymbol{B}_{14}\end{bmatrix} \tag{22}$$

where $\boldsymbol{B}_{ki} \in \mathbb{R}^{n_{ki}\times m_{ki}}$, $(k=1; i=1,\ldots,4)$ are decomposed by utilizing ACA, respectively. The subscript k denotes the number of level. Thus, matrix $\boldsymbol{O}_1$ can be rewritten as

$$\boldsymbol{O}_1=\begin{bmatrix}\boldsymbol{U}_{11}\boldsymbol{V}_{11} & \boldsymbol{U}_{12}\boldsymbol{V}_{12}\\ \boldsymbol{U}_{13}\boldsymbol{V}_{13} & \boldsymbol{U}_{14}\boldsymbol{V}_{14}\end{bmatrix}=\begin{bmatrix}\boldsymbol{U}_{11} & & \boldsymbol{U}_{12} & \\ & \boldsymbol{U}_{13} & & \boldsymbol{U}_{14}\end{bmatrix}\begin{bmatrix}\boldsymbol{V}_{11} & \\ \boldsymbol{V}_{13} & \\ & \boldsymbol{V}_{12}\\ & \boldsymbol{V}_{14}\end{bmatrix} \tag{23}$$

where $\boldsymbol{U}_{ij} \in \mathbb{R}^{n_{ij}\times k_{ij}}$ and $\boldsymbol{V}_{ij} \in \mathbb{R}^{k_{ij}\times m_{ij}}$ $(i=1; j=1,\ldots,4)$. As mentioned in Ref. [17], if we directly make use of Eq. (23), the sum $\sum_{j=1}^{4} k_{1j}$ may be not small enough for us to effectively use the Sherman–Morrison–Woodbury formula. In order to improve the efficiency, $[\boldsymbol{V}_{11}; \boldsymbol{V}_{13}]$ and $[\boldsymbol{V}_{12}; \boldsymbol{V}_{14}]$ are also decomposed by ACA, respectively, and obtain the following formulas

$$\begin{bmatrix}\boldsymbol{V}_{11}\\ \boldsymbol{V}_{13}\end{bmatrix}=\begin{bmatrix}\boldsymbol{G}_{11}\\ \boldsymbol{G}_{13}\end{bmatrix}\boldsymbol{P}_{11},\quad \begin{bmatrix}\boldsymbol{V}_{12}\\ \boldsymbol{V}_{14}\end{bmatrix}=\begin{bmatrix}\boldsymbol{G}_{12}\\ \boldsymbol{G}_{14}\end{bmatrix}\boldsymbol{P}_{12} \tag{24}$$

Then

$$\boldsymbol{O}_1=\boldsymbol{U}_1\boldsymbol{V}_1 \tag{25}$$

where $\boldsymbol{U}_1=\begin{bmatrix}\boldsymbol{U}_{11}\boldsymbol{G}_{11} & \boldsymbol{U}_{12}\boldsymbol{G}_{12}\\ \boldsymbol{U}_{13}\boldsymbol{G}_{13} & \boldsymbol{U}_{14}\boldsymbol{G}_{14}\end{bmatrix}$, $\boldsymbol{V}_1=\begin{bmatrix}\boldsymbol{P}_{11} & \\ & \boldsymbol{P}_{12}\end{bmatrix}$. Finally, we obtain the low-rank approximation of off-diagonal submatrix $\boldsymbol{O}_1$. The benefit of this algorithm is that the ACA decomposition can be implemented on smaller matrices, which can improve the computational efficient, so we call it the accelerated ACA algorithm.

4.2. Multi-level accelerated algorithm

In this section, we construct a two-level scheme, i.e. level=2 (see Fig. 5). The one-level scheme in Section 4.1 can be implemented recursively. Then we have

$$\boldsymbol{O}_1=\begin{bmatrix}\begin{bmatrix}\boldsymbol{B}_{211} & \boldsymbol{B}_{212}\\ \boldsymbol{B}_{213} & \boldsymbol{B}_{214}\end{bmatrix} & \begin{bmatrix}\boldsymbol{B}_{221} & \boldsymbol{B}_{222}\\ \boldsymbol{B}_{223} & \boldsymbol{B}_{224}\end{bmatrix}\\ \begin{bmatrix}\boldsymbol{B}_{231} & \boldsymbol{B}_{232}\\ \boldsymbol{B}_{233} & \boldsymbol{B}_{234}\end{bmatrix} & \begin{bmatrix}\boldsymbol{B}_{241} & \boldsymbol{B}_{242}\\ \boldsymbol{B}_{243} & \boldsymbol{B}_{244}\end{bmatrix}\end{bmatrix} \tag{26}$$

For each matrix $\boldsymbol{B}_{kij}\in\mathbb{R}^{n_{kij}\times m_{kij}}$, $(k=2;\ i=1,\ldots,4;\ j=1,\ldots,4)$, in which the subscript k denotes the number of level, the ACA is applied to obtain the compression form as following

$$\boldsymbol{O}_1=\begin{bmatrix}\begin{bmatrix}\boldsymbol{U}_{211}\boldsymbol{V}_{211} & \boldsymbol{U}_{212}\boldsymbol{V}_{212}\\ \boldsymbol{U}_{213}\boldsymbol{V}_{213} & \boldsymbol{U}_{214}\boldsymbol{V}_{214}\end{bmatrix} & \begin{bmatrix}\boldsymbol{U}_{221}\boldsymbol{V}_{221} & \boldsymbol{U}_{222}\boldsymbol{V}_{222}\\ \boldsymbol{U}_{223}\boldsymbol{V}_{223} & \boldsymbol{U}_{224}\boldsymbol{V}_{224}\end{bmatrix}\\ \begin{bmatrix}\boldsymbol{U}_{231}\boldsymbol{V}_{231} & \boldsymbol{U}_{232}\boldsymbol{V}_{232}\\ \boldsymbol{U}_{233}\boldsymbol{V}_{233} & \boldsymbol{U}_{234}\boldsymbol{V}_{234}\end{bmatrix} & \begin{bmatrix}\boldsymbol{U}_{241}\boldsymbol{V}_{241} & \boldsymbol{U}_{242}\boldsymbol{V}_{242}\\ \boldsymbol{U}_{243}\boldsymbol{V}_{243} & \boldsymbol{U}_{244}\boldsymbol{V}_{244}\end{bmatrix}\end{bmatrix} \tag{27}$$

Then the four sub-matrices can be written as

$$\begin{bmatrix}\boldsymbol{U}_{211}\boldsymbol{V}_{211} & \boldsymbol{U}_{212}\boldsymbol{V}_{212}\\ \boldsymbol{U}_{213}\boldsymbol{V}_{213} & \boldsymbol{U}_{214}\boldsymbol{V}_{214}\end{bmatrix}=\begin{bmatrix}\boldsymbol{U}_{211}\boldsymbol{G}_{211} & \boldsymbol{U}_{212}\boldsymbol{G}_{212}\\ \boldsymbol{U}_{213}\boldsymbol{G}_{213} & \boldsymbol{U}_{214}\boldsymbol{G}_{214}\end{bmatrix}\begin{bmatrix}\boldsymbol{P}_{211} & \\ & \boldsymbol{P}_{212}\end{bmatrix}=\boldsymbol{U}_{11}\boldsymbol{V}_{11} \tag{28a}$$

$$\begin{bmatrix}\boldsymbol{U}_{221}\boldsymbol{V}_{221} & \boldsymbol{U}_{222}\boldsymbol{V}_{222}\\ \boldsymbol{U}_{223}\boldsymbol{V}_{223} & \boldsymbol{U}_{224}\boldsymbol{V}_{224}\end{bmatrix}=\begin{bmatrix}\boldsymbol{U}_{221}\boldsymbol{G}_{221} & \boldsymbol{U}_{222}\boldsymbol{G}_{222}\\ \boldsymbol{U}_{223}\boldsymbol{G}_{223} & \boldsymbol{U}_{224}\boldsymbol{G}_{224}\end{bmatrix}\begin{bmatrix}\boldsymbol{P}_{221} & \\ & \boldsymbol{P}_{222}\end{bmatrix}=\boldsymbol{U}_{12}\boldsymbol{V}_{12} \tag{28b}$$

$$\begin{bmatrix}\boldsymbol{U}_{231}\boldsymbol{V}_{231} & \boldsymbol{U}_{232}\boldsymbol{V}_{232}\\ \boldsymbol{U}_{233}\boldsymbol{V}_{233} & \boldsymbol{U}_{234}\boldsymbol{V}_{234}\end{bmatrix}=\begin{bmatrix}\boldsymbol{U}_{231}\boldsymbol{G}_{231} & \boldsymbol{U}_{232}\boldsymbol{G}_{232}\\ \boldsymbol{U}_{233}\boldsymbol{G}_{233} & \boldsymbol{U}_{234}\boldsymbol{G}_{234}\end{bmatrix}\begin{bmatrix}\boldsymbol{P}_{231} & \\ & \boldsymbol{P}_{232}\end{bmatrix}=\boldsymbol{U}_{13}\boldsymbol{V}_{13} \tag{28c}$$

$$\begin{bmatrix}\boldsymbol{U}_{241}\boldsymbol{V}_{241} & \boldsymbol{U}_{242}\boldsymbol{V}_{242}\\ \boldsymbol{U}_{243}\boldsymbol{V}_{243} & \boldsymbol{U}_{244}\boldsymbol{V}_{244}\end{bmatrix}=\begin{bmatrix}\boldsymbol{U}_{241}\boldsymbol{G}_{241} & \boldsymbol{U}_{242}\boldsymbol{G}_{242}\\ \boldsymbol{U}_{243}\boldsymbol{G}_{243} & \boldsymbol{U}_{244}\boldsymbol{G}_{244}\end{bmatrix}\begin{bmatrix}\boldsymbol{P}_{241} & \\ & \boldsymbol{P}_{242}\end{bmatrix}=\boldsymbol{U}_{14}\boldsymbol{V}_{14} \tag{28d}$$

So we obtain the formula

$$\boldsymbol{O}_1=\begin{bmatrix}\boldsymbol{U}_{11}\boldsymbol{V}_{11} & \boldsymbol{U}_{12}\boldsymbol{V}_{12}\\ \boldsymbol{U}_{13}\boldsymbol{V}_{13} & \boldsymbol{U}_{14}\boldsymbol{V}_{14}\end{bmatrix} \tag{29}$$

Refer to Section 4.1, we can get the final form

$$\boldsymbol{O}_1=\boldsymbol{U}_1\boldsymbol{V}_1 \tag{30}$$

Fig. 5 shows the implementation process of accelerated algorithm for three levels. We can see that after dividing the matrix, we start from the bottom level (level=3) to construct the low-rank matrix recursively. The detailed description of the multi-level accelerated ACA decomposition is given in Algorithm 3. For any off-diagonal submatrix $\boldsymbol{O}$, we can effectively obtain the low-rank approximation using Algorithm 3. And we set a threshold which controls the number of level. Then a quadtree structure can be constructed. In the Algorithm 3, we adopt the following notations, i.e. let 1-child, 2-child, 3-child and 4-child denote the four children of one tree node for every recursive step, as shown in Fig. 6. From Fig. 6, we can see that the tree is a complete quadtree. That is, if one of the children is equal to zero, the tree nodes at this level are all leaves. We define that in each recursive step, let $\boldsymbol{O}$ have the 2×2 block form

$$\boldsymbol{O}=\begin{bmatrix}\boldsymbol{B}_1 & \boldsymbol{B}_2\\ \boldsymbol{B}_3 & \boldsymbol{B}_4\end{bmatrix}$$

where $\boldsymbol{B}_1$, $\boldsymbol{B}_2$, $\boldsymbol{B}_3$ and $\boldsymbol{B}_4$ are the four children of $\boldsymbol{O}$. Note that the proposed algorithm can be applied to any matrix corresponding to the problem under consideration without changing the algorithm.

Algorithm 3.

Step 0. The first tree node is root node $\boldsymbol{O}=\boldsymbol{O}_1$.
Step 1.
(1) If the first visit to tree node $\boldsymbol{O}$, go to step 2.
(2) If the second visit to tree node $\boldsymbol{O}$, go to step 3.
Step 2.
(1) If the children of $\boldsymbol{O}$ are NULL, decompose the submatrices $\boldsymbol{B}_1$, $\boldsymbol{B}_2$, $\boldsymbol{B}_3$ and $\boldsymbol{B}_4$ by ACA. Then apply Eqs. (23)–(25) to obtain the low-rank approximation of matrix $\boldsymbol{O}$.
If $\boldsymbol{O}$ is the root, we are done.
Otherwise if $\boldsymbol{O}$ is the 1-child, we find the 2-child of the father of $\boldsymbol{O}$, and set it $\boldsymbol{O}$, go to step 1. Otherwise if $\boldsymbol{O}$ is the 2-child, we find the 3-child of the father of $\boldsymbol{O}$, and set it $\boldsymbol{O}$, go to step 1. Otherwise if $\boldsymbol{O}$ is the 3-child, we find the 4-child of the father of $\boldsymbol{O}$, and set it $\boldsymbol{O}$, go to step 1. Otherwise update $\boldsymbol{O}$ to its father, go to step 1.
(2) If the children of $\boldsymbol{O}$ are not NULL, pass down to its 1-child and update $\boldsymbol{O}$ to its 1-child. Go back to step1.
Step 3.
By step 2, we obtain Eq. (29). Then apply Eqs. (23)–(25) to obtain the low-rank approximation of matrix $\boldsymbol{O}$.
If $\boldsymbol{O}$ is the root, we are done.
Otherwise if $\boldsymbol{O}$ is the 1-child, we find the 2-child of the father of $\boldsymbol{O}$, and set it $\boldsymbol{O}$, go to step 1.
Otherwise if $\boldsymbol{O}$ is the 2-child, we find the 3-child of the father of $\boldsymbol{O}$, and set it $\boldsymbol{O}$, go to step 1.
Otherwise if $\boldsymbol{O}$ is the 3-child, we find the 4-child of the father of $\boldsymbol{O}$, and set it $\boldsymbol{O}$, go to step 1.
Otherwise update $\boldsymbol{O}$ to its father, go to step 1.

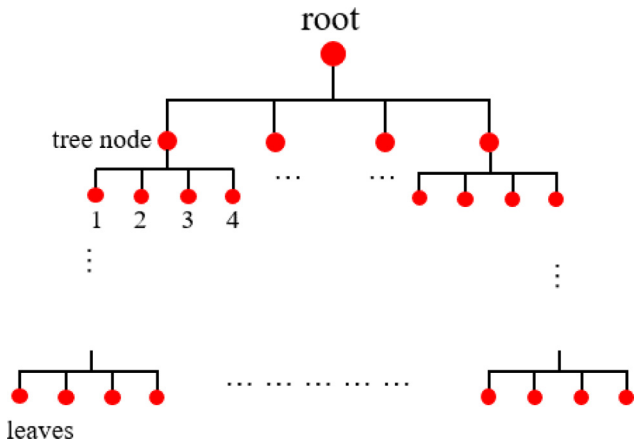


Fig. 6. Four cluster.

5. Numerical examples

In numerical implementation, to obtain accurate numerical results, the boundary meshes need to be refined. We introduce the refinement process. The knot vector is Ξ={0,0,0,1,1,1}, i.e. there is one element. To refine it once, the knot vector becomes Ξ={0,0,0,0.5,1,1,1}, i.e. there are two elements. To refine it further, the knot vector becomes Ξ={0,0,0,0.25,0.5,0.75,1,1,1}, i.e. there are four elements, and so on. In this paper, we apply the Bézier elements to investigate the algorithms, so each new knot during refinement is inserted p-times. And the default tolerance in ACA decomposition is set to 10^{-5}. Intel® Math Kernel Library (https://software.intel.com/en-us/mkl) is used to calculate the inverse of the diagonal matrices. For presenting the numerical results clearly, we compare three methods to assess the efficiency of the method proposed in this paper, i.e. Method 1: The conventional IGABEM; Method 2: The HODLR scheme without accelerated ACA algorithm for IGABEM; and Method 3: The HODLR scheme with accelerated ACA algorithm for IGABEM.

In this section, four numerical examples for different models are given. All examples have been run on a computer with Intel Xeon 2.4 GHZ CPU. The following notations are utilized to present the numerical results.

N	the degree of freedom
t_1	the total CPU time to solve Eq. (12) by Method 1.
t_2	the total CPU time to solve Eq. (12) by Method 2.
t_3	the total CPU time to solve Eq. (12) by Method 3.
t_{ACA}	the CPU time of no accelerated ACA decomposition for one-level off-diagonal submatrix $\boldsymbol{O}_1$.
t_{mACA}	the CPU time of accelerated ACA decomposition for one-level off-diagonal submatrix $\boldsymbol{O}_1$.
Re	the relative error $\sqrt{\|\boldsymbol{u}_{num}-\boldsymbol{u}_{exact}\|^2/\|\boldsymbol{u}_{exact}\|^2}$.

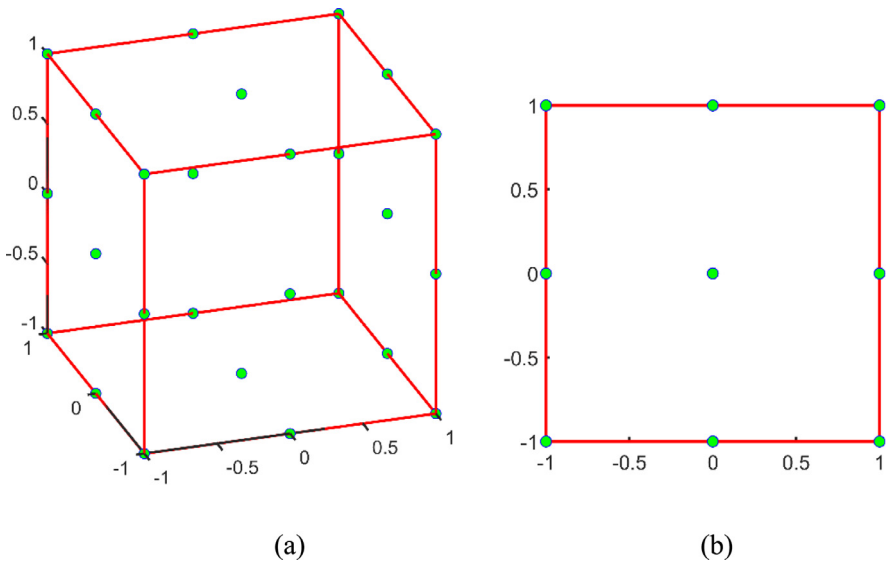


Fig. 7. (a) The sketch of the cube model. (b) One boundary surface of the cube model.

Table 1
The numerical results of potential at the interior points with Method 3.

x	$N=864$	$N=3456$	$N=13{,}824$	Exact
−0.95	0.855408	0.849372	0.850011	0.85
−0.90	0.899381	0.900008	0.899996	0.90
−0.80	1.000003	0.999993	0.999997	1.00
−0.60	1.199989	1.199995	1.199998	1.20
−0.40	1.399993	1.399997	1.399999	1.40
−0.20	1.599996	1.599998	1.599999	1.60
0.0	1.800000	1.800000	1.800000	1.80
0.20	2.000004	2.000001	2.000001	2.00
0.40	2.200007	2.200003	2.200001	2.20
0.60	2.400011	2.400005	2.400002	2.40
0.80	2.600062	2.600006	2.600003	2.60
0.90	2.697759	2.700056	2.700004	2.70
0.95	2.768662	2.747766	2.750055	2.75

Table 2
The total CPU time for three methods and ACA decomposition CPU time for Methods 2 and 3.

N	t_1(s)	t_2(s)	t_{ACA}(s)	t_3(s)	t_{mACA}(s)
864	3.58	2.71	1.25	3.56	1.58
3456	229.38	133.93	56.38	56.37	23.34
13,824	15,393.90	13,975.78	6060.97	1180.42	484.14

5.1. Cube model

A cube model is considered whose boundary is not smooth. In order to describe the non-smooth model, six surface patches for cube model are needed to stitch into a single model. In this example, the six surfaces have the same common knot vector along the common boundary lines. Fig. 7(a) displays the initial elements and control points, whereas one of boundary surfaces is given in Fig. 7(b). The knot vectors for one surface are Ξ={0,0,0,1,1,1} and $\mathscr{H}$={0,0,0,1,1,1}. The potential on the boundary is $u = x+3y/4+7z/5+9/5$ [5].

The numerical results of potential at the interior points along x axis with respect to degrees of freedom (DOFs) N are given in Table 1, and the exact results are also displayed for comparison. From Table 1, we can clearly see that with the increase of N the numerical results of method 3 are convergence and in good agreement with the exact results. When the interior points are close to the boundary, the numerical results are poor with the sparse grids. However, with the increment of N, the numerical results become good. Thus, by increasing the DOFs of problem, we can more accurately calculate the physical quantities at the points near the boundary. In Table 2, we investigate the CPU time by three methods. Table 2 shows

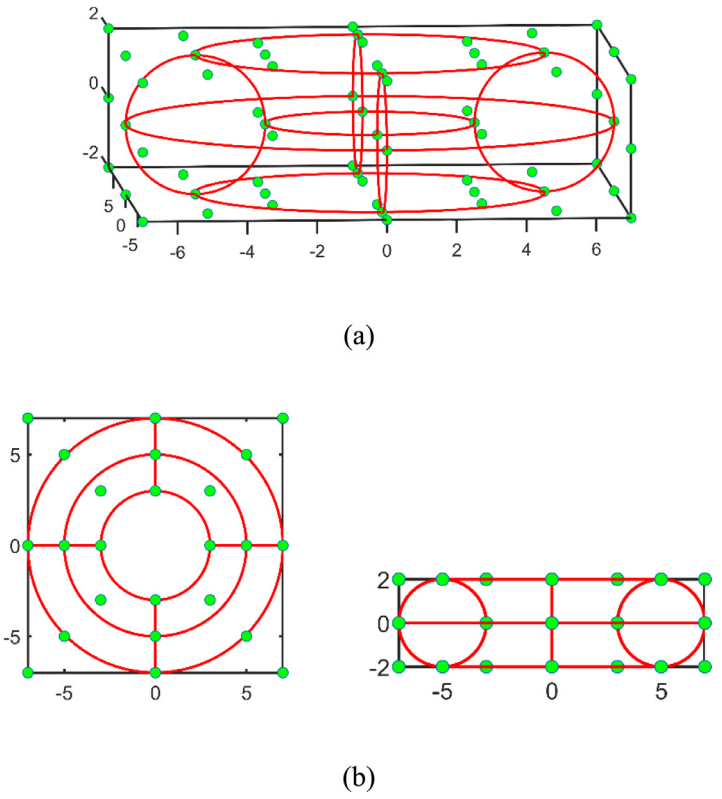


Fig. 8. (a) The initial sketch of the torus. (b) The top and side views of torus. denotes the control point and the polynomial orders $p=2$ and $q=2$. The knot vectors $\Xi=\{0,0,0,1,1,2,2,3,3,4,4,4\}$ and $\mathscr{H}=\{0,0,0,1,1,2,2,3,3,4,4,4\}$.

the total CPU time t_1, t_2 and t_3, as well as the ACA decomposition time t_{ACA} and t_{mACA}. Compared CPU time t_1, t_2 and t_3, we can see that t_3 is minimum. As the scale of matrix becomes large, we also can see that the accelerated ACA algorithm is more effective than conventional ACA decomposition. Hence, the method 3 presented in this paper can obtain accurate and effective numerical results for non-smooth boundary problem.

5.2. Torus model

In this example, we consider the torus model with major radius $R=5$ and minor radius $r=3$, and the center of the torus is at the origin. The initial elements and the control points of torus model are shown in Fig. 8(a). The top and side views are in Fig. 8(b). The potential distribution along the boundary is [17]

$$\begin{aligned} u(x,y,z) = &\sinh\left(\frac{\sqrt{2}}{4}x\right)\sin\left(\frac{y}{4}\right)\sin\left(\frac{z}{4}\right) \\ &+\sin\left(\frac{x}{4}\right)\sinh\left(\frac{\sqrt{2}}{4}y\right)\sin\left(\frac{z}{4}\right) \\ &+\sin\left(\frac{x}{4}\right)\sin\left(\frac{y}{4}\right)\sinh\left(\frac{\sqrt{2}}{4}z\right) \end{aligned}$$

Fig. 9(a) shows the exact contour plot of the normal derivative of the potential on the boundary surface. Fig. 9(b) gives the numerical results with $N=36{,}000$ solved by method 3. From Fig. 9, we can see that the results obtained from method 3 match the exact solutions very well. The convergence is investigated in Fig. 10. It can be seen that the three methods with IGABEM performs better than the conventional BEM, as we expected. The three methods with IGABEM are very close to each other, which demonstrates the method 3 also can obtain accurate numerical results. In terms of the efficiency, Fig. 11 illustrates total CUP time t_1, t_2 and t_3, respectively. With the increase of degrees of freedom, we can find that method 2 uses less CPU time than method 1. Meanwhile, as we can see from t_2 and t_3, the method 3 is much more efficient than method 2. Thus, method 3 with accelerated algorithm is more applicable when the scale of the matrix becomes larger. In Table 3, we only investigate the ACA decomposition. From Table 3, the accelerated algorithm with ACA is obviously faster. So for the large scale problem, the accelerated algorithm presented in this paper is efficient.

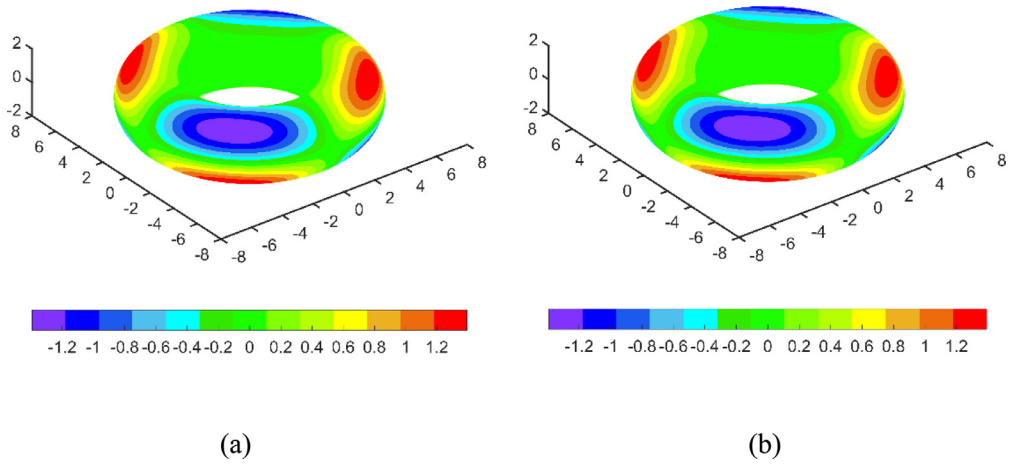


Fig. 9. The results of normal derivative of the potential along the surface.

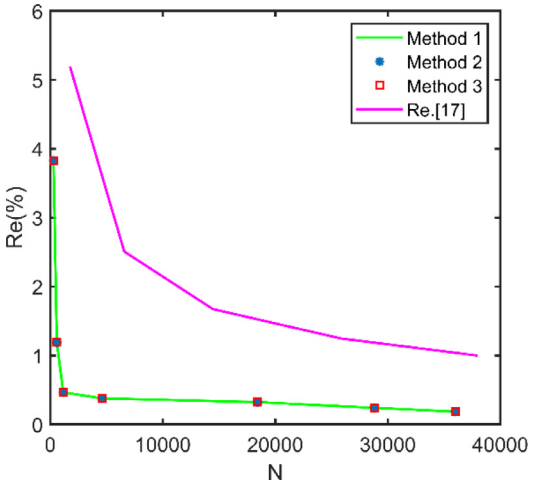


Fig. 10. Convergence of the normal derivative of potential along the boundary.

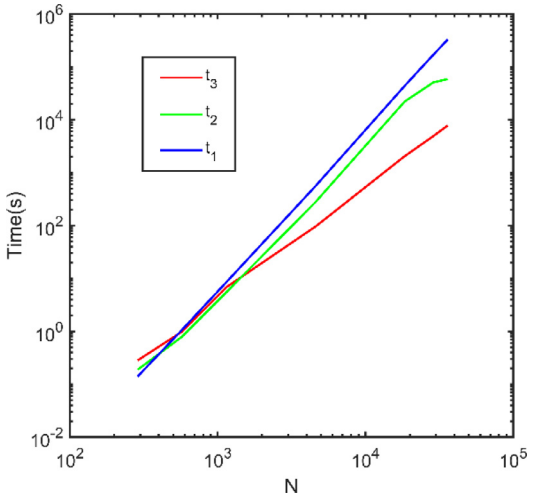
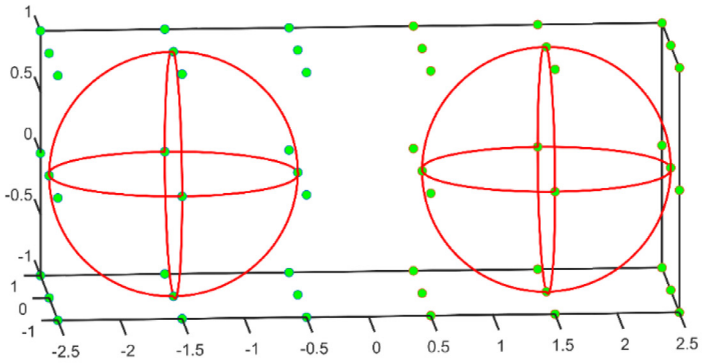


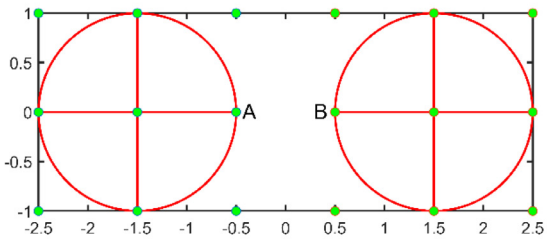
Fig. 11. CPU time for solving the system equations.

Table 3
The CPU time for ACA decomposition of Methods 2 and 3.

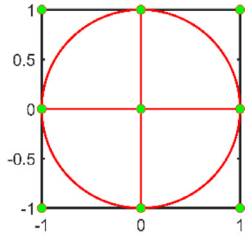
N	t_{ACA}(s)	$t_{mACA}(s)$
288	0.07	0.09
576	0.36	0.41
1152	2.57	3.21
4608	136.02	45.71
18,432	10,183.52	968.32
28,800	25,302.40	2304.16
36,000	26,444.97	3436.97



(a)



(b)



(c)

Fig. 12. (a) The initial sketch of the two spherical cavities. (b) The top view of two spherical cavities. (c) The side views of two spherical cavities denotes the control point and the polynomial orders $p=2$ and $q=2$. The knot vectors $\Xi=\{0,0,0,1,1,2,2,3,3,4,4,4\}$ and $\mathscr{H}=\{0,0,0,1,1,2,2,2\}$.

5.3. An infinite space with two spherical cavities

In the third example, we consider an exterior problem with two unit spherical cavities, as shown in Fig. 12 which displays the initial elements and control points, under the constant potential $u=1$ on the boundary surfaces of two spherical cavities. The centers of two spherical cavities are at $(-1.5,0,0)$ and $(1.5,0,0)$, respectively, and the distance between A and B of two spherical cavities is $d=1$ (see Fig. 12(b)).

The 36,864 DOFs are utilized to obtain the numerical results. Figs. 13(a), 13(b), 13(c) and 13(d) depict the potential u distribution within the matrix by method 3 with distance between two spherical cavities, respectively, and the distances d are chosen as 0.5, 1.0, 2.0 and 3.0. Through the xy cross section in Fig. 13, we can see that potential distribution around the cavities are similar to each other, and the potential u slowly decreases from the boundary of spherical cavities to infinite. However, we also can find that the interaction between cavities weakens with the increase of distance d between two cavities. Table 4 gives the total CPU time t_2, t_3 and ACA decomposition CPU time t_{ACA}, t_{mACA}. t_3 takes less time than t_2, and t_{mACA} also takes less than t_{ACA}. The reason for the decrease of t_{mACA} is the accelerated algorithm applied to ACA decomposition in method 3. Thus, the decrease of t_{mACA} further leads to the decrease of t_3. In Table 4, as the distance d increases, we can find that both the total CPU time and ACA decomposition CPU time decrease. Although the DOFs ($N=36{,}000$) of the torus

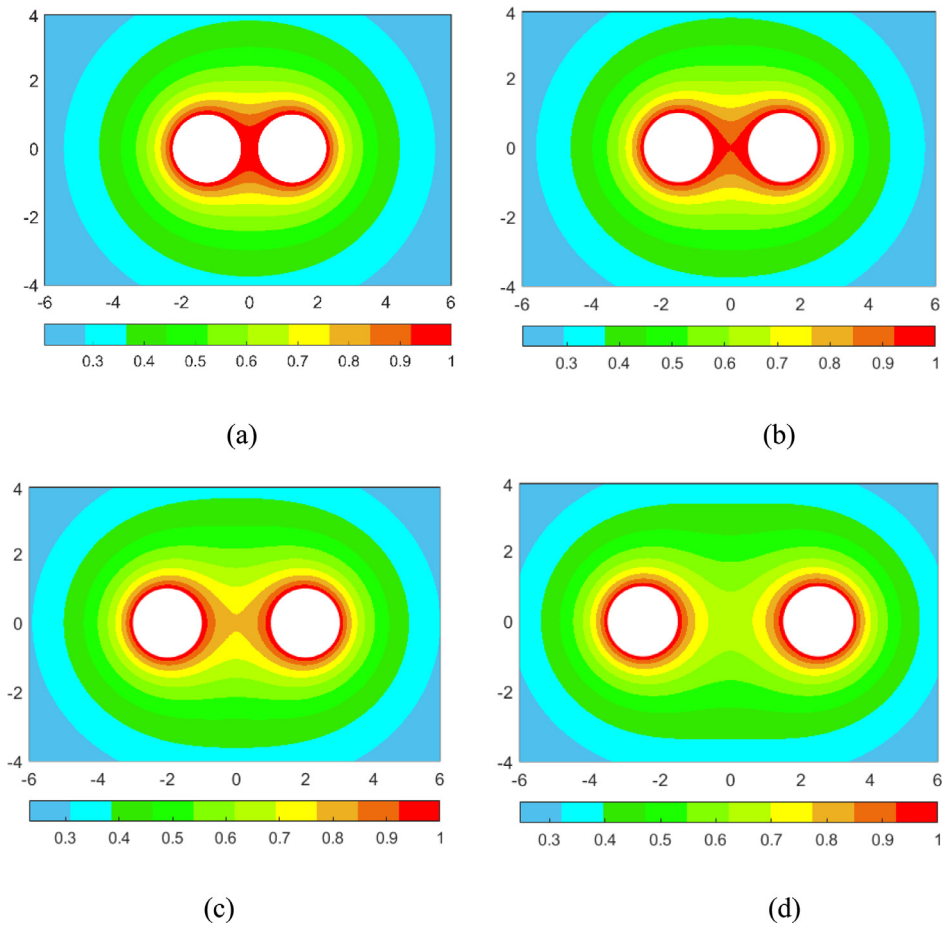


Fig. 13. The xy cross section for the potential u with the distance d between two spherical cavities. (a) Distance $d=0.5$. (b) Distance $d=1.0$. (c) Distance $d=2.0$. (d) Distance $d=3.0$.

Table 4
The total CPU time and CPU time of ACA decomposition for Methods 2 and 3.

	t_2	t_{ACA}	t_3	t_{mACA}
$d=0.5$	6752.33	3146.61	2062.05	782.87
$d=1.0$	3684.74	1591.32	1649.25	564.93
$d=2.0$	2178.88	849.65	1303.97	407.97
$d=3.0$	1698.39	613.09	1186.65	338.49
Torus ($N=36{,}000$)	56,594.14	26,444.97	7758.05	3436.97

model is smaller than the DOFs ($N=36{,}864$) of two spherical cavities model ($d=0.5$), torus model takes more time. Based on this phenomenon, we use ACA decomposition when solving Eq. (12), and the CPU time of methods 2 and 3 depends on the model.

When $d=2$ with $N=72{,}000$, Fig. 14 compares methods 2, 3 and the FEM results from Abaqus. It can be obviously observed that these three methods have good consistency. However, in Fig. 14(b), t_2=8741.08 s and t_{ACA}=4779.00 s, and in Fig. 14(c), t_3=3502.70 s and t_{mACA}=940.32 s. Therefore, the accelerated algorithm of method 3 can obtain more accurate results and is more effective for large scale problems.

5.4. An infinite space with twenty spherical cavities

Example 5.4 is the extension of example 5.3, where twenty unit spherical cavities are embedded in an infinite matrix, as shown in Fig. 15. And the twenty spherical cavities are under the constant potential $u=1$ on the boundary surfaces.

All numerical results are obtained with $N=92{,}160$ in this example. Fig. 16(a) shows the potential distribution within the matrix with the distance $d=1.0$. In order to observe the influence of the distance for distribution of potential within

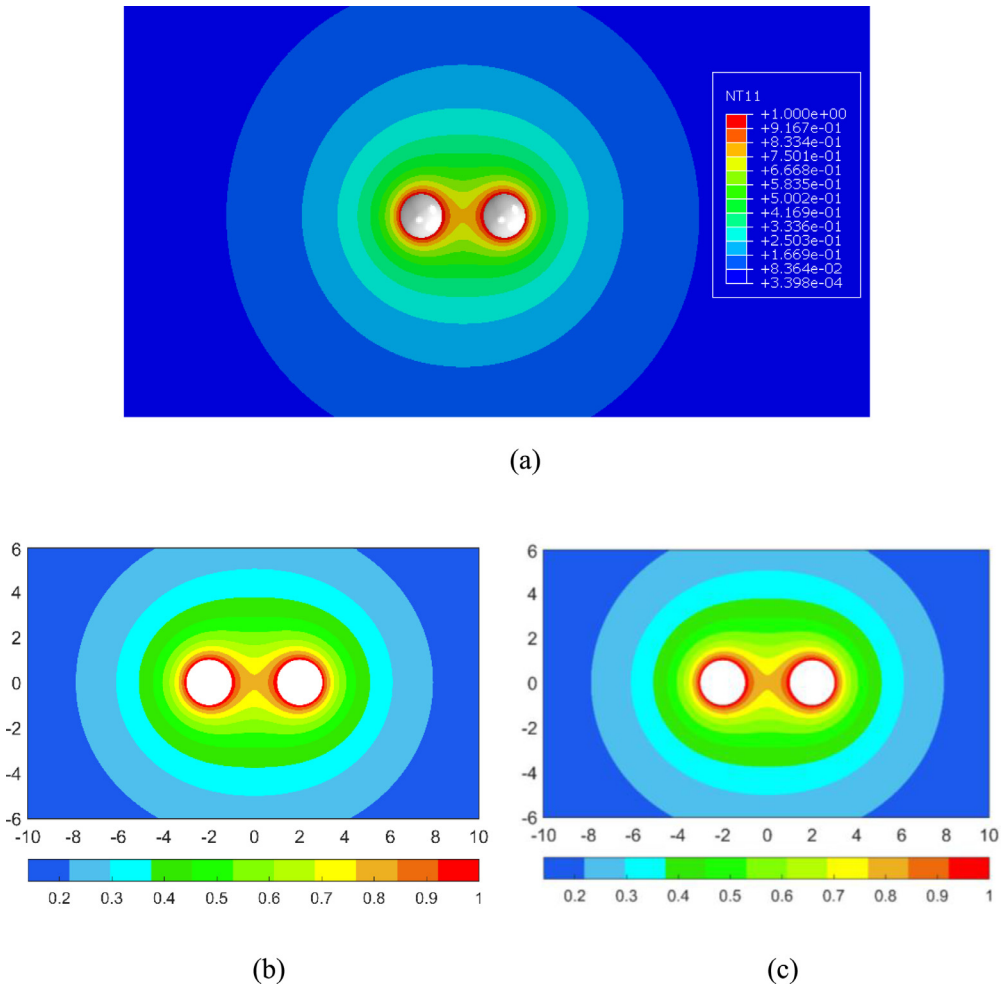


Fig. 14. The *xy* cross section for the potential u with the distance $d = 2.0$ between two spherical cavities. (a) Abaqus results; (b) Results of method 2; (c) Results of method 3.

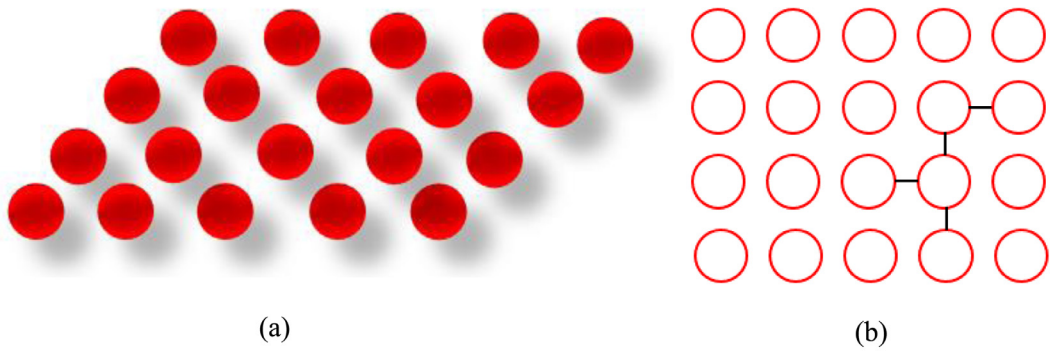


Fig. 15. (a) Sketch of the twenty spherical cavities in infinite space. (b) The *xy* cross section of the twenty spherical cavities. represents the distance d between two spherical cavities.

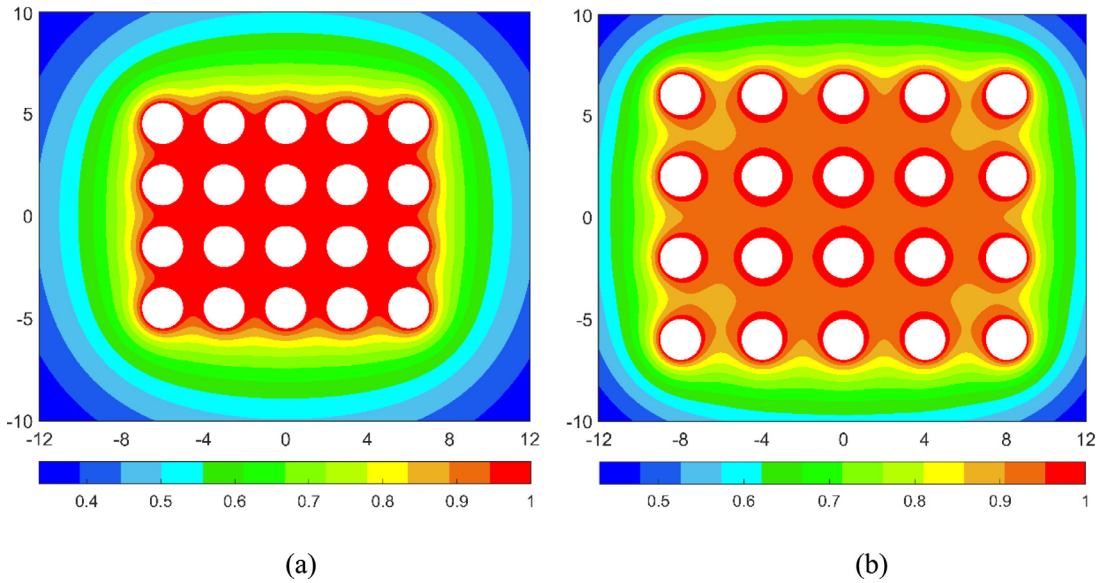


Fig. 16. The *xy* cross section for the potential *u* for the twenty spherical cavities.

Table 5
The total CPU time and CPU time of ACA decomposition for methods 2 and 3.

	t_2	t_{ACA}	t_3	t_{mACA}
$d=1$	132,966.65	60,539.04	8730.91	2918.54
$d=2$	71,236.81	33,711.49	6978.55	2123.73

matrix, distance d is added to 2.0. In Fig. 16(b), we can see that the distribution of potential around the spherical cavities in Fig. 16(b) is similar to the distribution in Fig. 16(a). The difference is that as the distance d increases, the interaction between spherical cavities decreases as shown in example 5.3. Table 5 gives the CPU time for methods 2 and 3, respectively, with respect to distance d. We can clearly see that the method 3 is more effective, as in example 5.3. The reason is that the accelerated ACA algorithm can save a lot of CPU time.

6. Conclusion

A novel fast direct method based on isogeometric boundary element method has been presented for solving 3D potential problems. The fast direct method utilizes the structure of the HODLR matrix, whose off-diagonal submatrices are low-rank matrices. The ACA algorithm is applied to obtain the low-rank matrices of off-diagonal submatrices. Then the HODLR matrix can be factored into the product form of some diagonal blocks. Therefore, we can use the Sherman–Morrison–Woodbury formula to solve the inverse of a HODLR matrix. However, for large scale problems, if we implement the ACA algorithm directly on the off-diagonal submatrix, the calculation time of ACA decomposition may increase, as described in the paper. In this paper, an accelerated ACA algorithm is presented to improve the existing algorithm, and the results are satisfactory.

In the numerical computation, bivariate NURBS basis functions are used to depict the geometry. Meanwhile, the same NURBS basis functions are used to approximate the boundary quantities. Moreover, Bézier extraction has been introduced to facilitate the coupling of IGA and BEM. At the end of the paper, some numerical examples are testified with the present method. We can see from the numerical results that the fast direct method developed in the paper can obtain accurate results with less CPU time. And the presented method can be extended to 3D elastostatic problems easily.

Acknowledgment

The research is supported by the National Natural Science Foundation of China (11672038).

Appendix A.

The multi-level algorithm

Before introducing the multi-level algorithm to obtain the matrix $\boldsymbol{A}^{-1}$, we define the notations. In each step let $\boldsymbol{B}$ have the form

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{D}_1 & \boldsymbol{U}_1\boldsymbol{V}_1 \\ \boldsymbol{U}_2\boldsymbol{V}_2 & \boldsymbol{D}_2 \end{bmatrix}$$

where $\boldsymbol{D}_1$ and $\boldsymbol{D}_2$ are the left and right children of $\boldsymbol{B}$. Hence we know for the root $\boldsymbol{B}=\boldsymbol{A}$. Then a binary tree structure can be constructed, and the description is given in Algorithm 2 [18].

References

[1] T.J.R. Hughes, J.A. Cottrell, Y. Bazilevs, Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement, Comput. Methods Appl. Mech. Eng. 194 (39–41) (2005) 4135–4195.
[2] R.N. Simpson, S.P.A. Bordas, J. Trevelyan, T. Rabczuk, A two-dimensional Isogeometric Boundary Element Method for elastostatic analysis, Comput. Methods Appl. Mech. Eng. 209–212 (324) (2012) 87–100.
[3] R.N. Simpson, M.A. Scott, M. Taus, D.C. Thomas, H. Lian, Acoustic isogeometric boundary element analysis, Comput. Methods Appl. Mech. Eng. 269 (2) (2014) 265–290.
[4] Y. Bai, C.Y. Dong, Z.Y. Liu, Effective elastic properties and stress states of doubly periodic array of inclusions with complex shapes by isogeometric boundary element method, Compos. Struct. 128 (2015) 54–69.
[5] Y.P. Gong, C.Y. Dong, X.C. Qin, An isogeometric boundary element method for three dimensional potential problems, J. Comput. Appl. Math. 313 (2017) 454–468.
[6] M.J. Peake, J. Trevelyan, G. Coates, Extended isogeometric boundary element method (XIBEM) for two-dimensional Helmholtz problems, Comput. Methods Appl. Mech. Eng. 259 (2) (2013) 93–102.
[7] V. Rokhlin, Rapid solution of integral-equations of classical potential-theory, J. Comput. Phys. 60 (2) (1985) 187–207.
[8] R. Coifman, V. Rokhlin, S. Wandzura, The fast multipole method for the wave equation: a pedestrian prescription, Antennas Propag. Mag. IEEE 35 (3) (1993) 7–12.
[9] J. Cooley, J. Tukey, An algorithm for the machine calculation of complex Fourier series, Math. Comput. 19 (90) (1965) 297–301.
[10] W. Hackbusch, A sparse matrix arithmetic based on H-matrices, Computing 62 (1999) 89–108.
[11] I. González, O. Gutiérrez, F.S. De Adana, M.F. Cátedra, Application of multilevel fast multipole method to the analysis of the scattering of complex bodies modeled by NURBS surfaces, in: Proceedings of the IEEE Antennas and Propagation Society International Symposium, 2006, pp. 1887–1890.
[12] B. Marussig., J. Zechner, G. Beer, T.P. Fries, Fast isogeometric boundary element method based on independent field approximation, Comput. Methods Appl. Mech. Eng. 284 (2015) 458–488.
[13] Z. Liu, M. Majeed, F. Cirak, R.N. Simpson, Isogeometric FEM-BEM coupled structural-acoustic analysis of shells using subdivision surfaces, Int. J. Numer. Methods Eng. 113 (9) (2018) 1507–1530.
[14] M. Bebendorf, S. Rjasanow, Adaptive low-rank approximation of collocation matrices, Computing 70 (1) (2003) 1–24.
[15] L. Greengard, D. Gueyffier, P.-G. Martinsson, V. Rokhlin, Fast direct solvers for integral equations in complex three-dimensional domains, Acta Numer. 18 (1) (2009) 243–275.
[16] J. Lai, S. Ambikasaran, L.F. Greengard, A fast direct solver for high frequency scattering from a large cavity in two dimensions, SIAM J. Sci. Comput. 36 (6) (2014) 887–903.
[17] S. Huang, Y.J. Liu, A new fast direct solver for the boundary element method, Comput. Mech. 60 (4) (2017) 1–14.
[18] W.K. Kong, J. Bremer, V. Rokhlin, An adaptive fast direct solver for boundary integral equations in two dimensions, Appl. Comput. Harmonic Anal. 31 (3) (2011) 346–369.
[19] J. Sherman, W.J. Morrison, Adjustment of an inverse matrix corresponding to a change in one element of a given matrix, Ann. Math. Stat. 21 (1) (1950) 124–127.
[20] G. Xie, F. Zhou, J. Zhang, X. Zheng, C. Huang, New variable transformations for evaluating nearly singular integrals in 3D boundary element method, Eng. Anal. Bound. Elem. 37 (9) (2013) 1169–1178.
[21] K. Hayami, H. Matsumoto, A numerical quadrature for nearly singular boundary element integrals, Eng. Anal. Bound. Elem. 13 (2) (1994) 143–154.
[22] J.H. Iv, Y. Miao, H.P. Zhu, The distance Sinh transformation for the numerical evaluation of nearly singular integrals over curved surface elements, Comput. Mech. 53 (2) (2013) 359–369.
[23] F.J. Wang, W. Chen, W.Z. Qu, Y. Gu, A BEM formulation in conjunction with parametric equation approach for three-dimensional inverse heat conduction problems, Eng. Anal. Bound. Elem. 63 (114) (2016) 1–14.
[24] X.W. Gao, An effective method for numerical evaluation of general 2D and 3D high order singular boundary integrals, Comput. Methods Appl. Mech. Eng. 199 (45) (2010) 2856–2864.
[25] F. Auricchio, L.B.D. Veiga, T.J.R. Hughes, A. Reali, G. Sangalli, Isogeometric collocation methods, Math. Models Methods Appl. Sci. 20 (11) (2010) 2075–2107.
[26] M.A. Scott, M.J. Borden, C.V. Verhoosel, T.W. Sederberg, T.J.R. Hughes, Isogeometric finite element data structures based on Bézier extraction of T-splines, Int. J. Numer. Methods Eng. 88 (2) (2011) 126–156.
[27] M.J. Borden, M.A. Scott, J.A. Evans, T.J.R. Hughes, Isogeometric finite element data structures based on Bézier extraction of NURBS, Int. J. Numer. Methods Eng. 87 (2011) 15–47.
[28] S. Ambikasaran, E. Darve, An O(N logN) fast direct solver for partial hierarchically semi-separable matrices, J. Sci. Comput. 57 (3) (2013) 477–501.
[29] M. Bebendorf, R. Grzhibovskis, Accelerating Galerkin BEM for linear elasticity using adaptive cross approximation, Math. Methods Appl. Sci. 29 (14) (2006) 1721–1747.
[30] J. Zechner, G. Beer, A fast elasto-plastic formulation with hierarchical matrices and the boundary element method, Comput. Mech. 51 (4) (2013) 443–453.