Available online at www.sciencedirect.com
ScienceDirect
International Journal of Solids and Structures 45 (2008) 2355–2376
INTERNATIONAL JOURNAL OF SOLIDS AND STRUCTURES
www.elsevier.com/locate/ijsolstr

A fast 3D dual boundary element method based on hierarchical matrices

I. Benedetti [a], M.H. Aliabadi [b,*], G. Davì [c]

[a] Dipartimento di Ingegneria Aerospaziale, Università di Pisa, Via G. Caruso, 56122 Pisa, Italy
[b] Department of Aeronautics, Imperial College London, South Kensington Campus, Roderic Hill Building, Exhibition Road, SW72AZ London, UK
[c] Dipartimento di Tecnologie ed Infrastrutture Aeronautiche, Università degli Studi di Palermo, Edificio 8, Viale delle Scienze, 90128 Palermo, Italy

Received 17 September 2007; received in revised form 19 November 2007
Available online 8 December 2007

Abstract

In this paper a fast solver for three-dimensional BEM and DBEM is developed. The technique is based on the use of hierarchical matrices for the representation of the collocation matrix and uses a preconditioned GMRES for the solution of the algebraic system of equations. The preconditioner is built exploiting the hierarchical arithmetic and taking full advantage of the hierarchical format. Special algorithms are developed to deal with crack problems within the context of DBEM. The structure of DBEM matrices has been efficiently exploited and it has been demonstrated that, since the cracks form only small parts of the whole structure, the use of hierarchical matrices can be particularly advantageous. Test examples presented show that, with the proposed technique, substantial increase in number of elements over the crack surfaces leads only to moderate increases in memory storage and solution time.

Keywords: Dual boundary element method; Hierarchical matrices; Fast solvers; Large scale computations

1. Introduction

The boundary element method (BEM) has been developed over the last three decades as a powerful numerical tool for the analysis and solution of many physical and engineering problems (Wrobel, 2002; Aliabadi, 2002). Today it represents a viable alternative to other numerical approaches, such as the finite element method (FEM).

The main advantage of boundary element techniques is the reduction in the degrees of freedom needed to model a given physical system. Such reduction is allowed by the underlying boundary integral formulation

* Corresponding author.
E-mail addresses: i.benedetti@imperial.ac.uk (I. Benedetti), m.h.aliabadi@imperial.ac.uk (M.H. Aliabadi), davi@unipa.it (G. Davì).

0020-7683/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.ijsolstr.2007.11.018

which requires, for its numerical solution, only the discretization of the boundary of the analyzed domain. This results not only in a reduction in size of the system matrix, but also in faster data preparation.

However, as the size of the problem increases, the time required to solve the final system of equations increases considerably. The system matrix obtained by the application of the boundary element method is fully populated and not symmetric. This results in increased storage memory requirements as well as increased solution time with respect to the finite element method. Since the matrix is fully populated the memory required to store its coefficients is of order $O(N^2)$, where N denotes the number of unknowns. On the other hand, the solution of the system requires $O(N^3)$ operations if direct solvers are used or $O(M \times N^2)$ if iterative solvers are employed, where M denotes the number of iterations.

Much research has been devoted to the improvement of BE solution methods. In the early 1980s, Rokhlin (1985) developed an iterative strategy for the solution of the integral equations arising in classical potential theory. This work introduced the idea of coupling iterative solvers and the harmonic expansion of the kernels on suitable clusters of far field boundary elements, in order to reduce the computational cost of the solution process. In particular, the method was aimed at reducing the number of operations required to evaluate the matrix–vector products occurring in the application of iterative solvers and resulted in an $O(N)$ algorithm for the solution of the original equations.

Similar algorithms for the reduction of the *computational complexity* of the solution process were also developed in other fields of investigation not directly related to the boundary element method. Particularly interesting is the algorithm devised by Barnes and Hut (1986) for the treatment of the gravitational N-body problem. They developed an $O(N \ln N)$ strategy based on the preliminary hierarchical subdivision of the space into cubic cells and on the following approximation of the mutual action between cells through a recursive scheme. A similar approach was presented by Greengard and Rokhlin (1987), who used multipole expansions to approximate potential and force fields of various nature generated by systems of many particles.

Although these algorithms were mainly developed for N-body problems, they can be extended to the treatment of boundary value problems, due to their similar underlying mathematical structure. The boundary element method is in fact based on calculation of influence coefficients of the solution matrix by integration of the fundamental solution collocated on some *source* point, which represents a certain mutual influence between couples of points, over some surface elements. From this point of view the approaches developed by Rokhlin (1985) and Greengard and Rokhlin (1987) are analogous, as already pointed out by the authors themselves.

Starting from these early works, fast multipole methods (FMMs) have been developed to solve efficiently boundary element formulations for different kinds of problems. Nishimura et al. (1999) used FMMs in connection with a generalized minimum residual method (GMRES) (Saad and Schultz, 1986) to solve 3D crack problems for the Laplace equation. Fast algorithms have also been used for the treatment of elastic problems. Fu et al. (1998) developed a FMM boundary element method for 3D many-particle elastic problems based on spherical harmonic expansions of the kernel functions, while Popov and Power (2001) used Taylor expansions to obtain an $O(N)$ algorithm for 3D elasticity as well.

Another interesting method intended for enhancing the matrix–vector multiplication was the panel clustering approach developed by Hackbusch and Nowak (1989).

Although above techniques are very effective and provide a valuable tool for the fast solution of boundary element problems, their main disadvantage is that the knowledge of the kernel expansion is required in order to carry out the integration; all the terms of the series needed to reach a given accuracy must be computed in advance and then integrated, which can lead to a significant modification of the integration procedures in standard BEM codes.

From an algebraic point of view however, the integration of a *degenerate kernel*, i.e. of a kernel expanded in series, over a cluster of elements corresponds to the approximation of the corresponding matrix block by a *low rank* block. This idea paved the way to the development of purely algebraic techniques for the approximation of large BEM matrices, like the *mosaic-skeleton* method (Tyrtyshnikov, 1996, 2000; Goreinov et al., 1997). Of particular interest was the observation, due to Tyrtyshnikov (1996), that low rank approximations could be built from only few entries of the original block. Successively Bebendorf (2000) proposed a method for the construction of such approximations, based on the computation of selected rows and columns from the original blocks. The technique was further developed by Bebendorf and Rjasanow (2003) and was referred to as

adaptive cross approximation (ACA). Such an algorithm allows a relatively simple generation of the approximation and enables both storage and matrix–vector multiplication in almost linear complexity.

The subdivision of the matrix into a hierarchical tree of sub-blocks and the blockwise approximation by low rank blocks is the basis for the *hierarchical representation* of the collocation matrix (Hackbusch, 1999; Hackbusch and Khoromskij, 2000). Analogously to FMMs, the use of the hierarchical format is aimed at reducing the storage requirement and the computational complexity arising in the boundary element method. Having represented the coefficient matrix in hierarchical format, the solution of the system can be obtained either directly, by inverting the matrix in hierarchical format, or indirectly, by using iterative schemes with or without preconditioners (Grasedyck, 2005; Bebendorf, 2005). Both choices rely on the use of formatted matrix operations, i.e. on a suitable arithmetic for hierarchical matrices developed to take advantage of this special format (Grasedyck and Hackbusch, 2003; Börm et al., 2003).

The use of iterative techniques, however, takes particular advantage of the employment of the hierarchical format. Different iterative solvers for algebraic systems of equations stemming from 2D and 3D BEM problems have been investigated. While early studies (Mullen and Rencis, 1987) had not shown good results, following works (Mansur et al., 1992; Valente and Pina, 1998) confirmed the applicability of iterative solution procedures to BEM systems and showed the potentiality for operations count reduction with respect to Gauss elimination especially for large systems; on the other hand they reported serious lack of convergence for the worst ill-conditioned cases, when mixed boundary conditions are present, thus pointing out the need for preconditioning the system. Barra et al. (1992) tested the performance of the GMRES algorithm, developed by Saad and Schultz (1986), for the solution of two-dimensional elasticity BEM equations, observing a more rapid convergence with respect to other iterative strategies, especially when preconditioning was used. They mentioned the possibility of developing new preconditioners based on the inherent nature of the BEM. Prasad et al. (1994) discussed the performance of several Krylov subspace methods and related such performance to the structure of the BEM matrices for some two and three-dimensional thermal and elastic problems, highlighting the effect of the relative magnitude of the coefficients of the system matrix on the convergence of the algorithms. Moreover, they pointed out that the use of suitable preconditioning improves the eigenvalues clustering and the diagonal dominance of the matrix, thus resulting in enhanced convergence. Their analysis demonstrated that preconditioned Krylov methods, especially preconditioned GMRES, could be competitive or superior to direct methods. The importance of the diagonal dominance for the iterative solution of BEM equations has been shown by Urekew and Rencis (1993), while Merkel et al. (1992) focused on eigenvalues clustering and its effect on the convergence of iterative solvers applied to the solution of some thermal and elastic industrial problems. Some issues in the analysis of larger three-dimensional BEM systems through preconditioned GMRES were evidenced by Leung and Walker (1997), who also proposed a strategy to overcome some limitations and extend the applicability of the algorithm to systems with some thousands of unknowns. Barra et al. (1993) proposed a strategy for the construction of preconditioners for GMRES solved BEM problems, while Wang et al. (2005) investigated a class of preconditioners for fast multipole BEMs.

All the aforementioned studies have demonstrated the importance of preconditioners for an effective iterative solution. A general survey on preconditioners for improving the performance and reliability of Krylov subspace methods has recently been presented by Benzi (2002), who pointed out that the intense research on preconditioners has blurred the distinction between direct and iterative solvers. The importance of the subject has also been stressed by Saad and van der Vorst (2000), in their survey of iterative solvers for linear systems.

In this context, the use of the hierarchical format for BEM matrices, in conjunction with Krylov subspace methods, constitutes a recent and interesting development. The method proves to be efficient in dealing with large BEM systems and offers a quite natural approach to the construction of effective preconditioners.

Hierarchical matrices and their arithmetic have been extensively studied and assessed and their application has proved successful for the analysis of some interesting realistic problems. Apart from some benchmark tests reported in the papers devoted to the development of the technique, see for example the work of Bebendorf and Rjasanow (2003), interesting applications to various electromagnetic problems have been proposed by Kurz et al. (2002), Zhao et al. (2005) and Ostrowski et al. (2006). Bebendorf and Grzhibovskis (2006) have recently extended the use of ACA to the analysis of elastic problems through Galerkin BEM. However, no application of hierarchical matrices to elastic *crack* problems is reported in the literature.

In this paper, the development of a fast DBEM based on hierarchical matrices for the analysis of three-dimensional elasticity cracks problems is presented. First, the basic formulation of the DBEM for 3D fracture mechanics problems is briefly reviewed and the features of DBEM matrices are discussed. Next the main steps for building the hierarchical representation of the solution matrix are illustrated. Some considerations about the application of the hierarchical format to boundary element formulations of 3D crack problems are pointed out. Some applications complete the work and demonstrate the capability of the method.

2. The 3D dual boundary element method

The dual boundary element method is a general and efficient technique for modeling both two-dimensional (Portela et al., 1992, 1993) and three-dimensional (Mi and Aliabadi, 1992, 1994) crack problems in the framework of the BEMs (Aliabadi, 1997a,b). The method is based on the use of two independent boundary integral equations, namely the displacement integral equation, collocated on the external boundary and on one of the crack surfaces, and the traction integral equations, collocated on the other crack surface and introduced to overcome the problems originating from the coincidence of the crack nodes.

Assuming continuity of displacements at the boundary nodes, the boundary integral representation for the displacements u_j is given by

$$c_{ij}(\boldsymbol{x}_0)u_j(\boldsymbol{x}_0) + \int\!\!\!\!\!\!-_{\Gamma} T_{ij}(\boldsymbol{x}_0,\boldsymbol{x})u_j(\boldsymbol{x})\,\mathrm{d}\Gamma = \int_{\Gamma} U_{ij}(\boldsymbol{x}_0,\boldsymbol{x})t_j(\boldsymbol{x})\,\mathrm{d}\Gamma \tag{1}$$

where U_{ij} and T_{ij} represent the Kelvin displacement and traction fundamental solutions at the boundary point $\boldsymbol{x}$ when collocating at the point $\boldsymbol{x}_0$, c_{ij} are coefficients depending on the boundary geometry and computed through rigid body considerations and the symbol $\int\!\!\!\!\!\!-$ stands for Cauchy principal value integral, whose presence is consequence of the $\mathrm{O}(r^{-2})$ strength of the T_{ij} integrands.

The displacement equation (1) is collocated on the boundary Γ and on one of the crack surfaces. When collocated at the crack node $\boldsymbol{x}_0^-$, it assumes the form

$$c_{ij}(\boldsymbol{x}_0^-)u_j(\boldsymbol{x}_0^-) + c_{ij}(\boldsymbol{x}_0^+)u_j(\boldsymbol{x}_0^+) + \int\!\!\!\!\!\!-_{\Gamma} T_{ij}(\boldsymbol{x}_0^-,\boldsymbol{x})u_j(\boldsymbol{x})\,\mathrm{d}\Gamma = \int_{\Gamma} U_{ij}(\boldsymbol{x}_0^-,\boldsymbol{x})t_j(\boldsymbol{x})\,\mathrm{d}\Gamma \tag{2}$$

where $\boldsymbol{x}_0^-$ and $\boldsymbol{x}_0^+$ are the two coincident crack nodes. For smooth crack surfaces at the point $\boldsymbol{x}_0^-$, it is $c_{ij}(\boldsymbol{x}_0^-) = c_{ij}(\boldsymbol{x}_0^+) = (1/2)\delta_{ij}$.

The traction integral equation collocated at the point $\boldsymbol{x}_0^+$, where continuity of strains is assumed, is given by

$$c_{ij}(\boldsymbol{x}_0^+)t_j(\boldsymbol{x}_0^+) - c_{ij}(\boldsymbol{x}_0^-)t_j(\boldsymbol{x}_0^-) + n_j(\boldsymbol{x}_0^+)\neq\!\!\!\!\!\!-_{\Gamma} T_{ijk}(\boldsymbol{x}_0^+,\boldsymbol{x})u_k(\boldsymbol{x})\,\mathrm{d}\Gamma = n_j(\boldsymbol{x}_0^+)\int\!\!\!\!\!\!-_{\Gamma} U_{ijk}(\boldsymbol{x}_0^+,\boldsymbol{x})t_k(\boldsymbol{x})\,\mathrm{d}\Gamma \tag{3}$$

where the kernels U_{ijk} and T_{ijk} contain derivatives of U_{ij} and T_{ij}, respectively, n_j are the component of the outward normal at the point $\boldsymbol{x}_0^+$ and $\neq$ stands for Hadamard principal value integral, originating from the presence of the $\mathrm{O}(r^-3)$ kernels T_{ijk}.

Eqs. (1)–(3) provide the boundary integral model for the analysis of general crack problems. The discrete model is built on them starting from a suitable discretization of the external boundary and the crack surfaces into a set of boundary elements over which the displacements and the tractions, as well as the geometry, are expressed by means of suitable shape functions and nodal values (Aliabadi, 1997b).

Care must be taken in the choice of suitable boundary elements, in order to fulfill the conditions for the existence of the singular integrals. In particular the existence of Cauchy and Hadamard principal values requires Hölder continuity of the displacements and their derivatives at the collocation points. Such restrictions can be satisfied through the use of special boundary elements. In this work, the same modeling strategy as that adopted by Mi and Aliabadi (1992, 1994) is used. The continuity of the displacement derivatives, which is the stronger constraint required for the existence of the integrals in the traction equation, is guaranteed by using *discontinuous* eight-node quadratic elements for the modeling of *both* the crack surfaces. The boundary, on which only the displacement equation is collocated, is modeled by using *continuous* eight-node quadratic elements. Further information on slightly different discretization procedures can be found in Cisilino and Aliabadi (1997, 2004).

2.1. DBEM systems of equations

The DBEM leads to a system of equations that can be written in the form

$$\begin{bmatrix} \mathbf{H}_{bb} & \mathbf{H}_{bd} & \mathbf{H}_{bt} \\ \mathbf{H}_{db} & \mathbf{H}_{dd} & \mathbf{H}_{dt} \\ \mathbf{S}_{tb} & \mathbf{S}_{td} & \mathbf{S}_{tt} \end{bmatrix} \begin{bmatrix} \boldsymbol{u}_b \\ \boldsymbol{u}_d \\ \boldsymbol{u}_t \end{bmatrix} = \begin{bmatrix} \mathbf{G}_{bb} & \mathbf{G}_{bd} & \mathbf{G}_{bt} \\ \mathbf{G}_{db} & \mathbf{G}_{dd} & \mathbf{G}_{dt} \\ \mathbf{D}_{tb} & \mathbf{D}_{td} & \mathbf{D}_{tt} \end{bmatrix} \begin{bmatrix} \boldsymbol{t}_b \\ \boldsymbol{t}_d \\ \boldsymbol{t}_t \end{bmatrix} \tag{4}$$

where the subscripts denote the boundary of the domain (b), the crack surface where the displacement integral equation is collocated (d) and the crack surface where the traction integral equation (t) is collocated. The blocks $\mathbf{H}_{bd}$ and $\mathbf{G}_{bd}$, for example, contain the coefficients computed by integrating T_{ij} and U_{ij} over the elements lying on the displacement crack surface (d) while collocating the integral equation on the boundary nodes (b). The blocks $\mathbf{S}$ and $\mathbf{D}$ are obtained from the integration of the kernels T_{ijk} and U_{ijk}, when the traction integral equation is collocated on the related crack surface.

After the application of the boundary conditions, assuming free crack surfaces, the final system is written

$$\begin{bmatrix} \mathbf{A}_{bb} & \mathbf{H}_{bd} & \mathbf{H}_{bt} \\ \mathbf{A}_{db} & \mathbf{H}_{dd} & \mathbf{H}_{dt} \\ \mathbf{B}_{tb} & \mathbf{S}_{td} & \mathbf{S}_{tt} \end{bmatrix} \begin{bmatrix} \boldsymbol{X}_b \\ \boldsymbol{u}_d \\ \boldsymbol{u}_t \end{bmatrix} = \begin{bmatrix} \boldsymbol{Y}_b \\ \boldsymbol{Y}_d \\ \boldsymbol{Y}_t \end{bmatrix} \tag{5}$$

where the vector $\boldsymbol{X}_b$ contains unknown boundary displacements or tractions and the vectors $\boldsymbol{Y}$ are obtained, after applying the boundary conditions, as a linear combination of the columns of the blocks $\mathbf{H}$, $\mathbf{G}$, $\mathbf{D}$ and $\mathbf{S}$ corresponding to the prescribed displacement and traction nodal values. The blocks $\mathbf{A}_{bb}$ and $\mathbf{A}_{db}$ contain a mix of columns from the corresponding blocks $\mathbf{H}_{bb}$ and $\mathbf{G}_{bb}$, $\mathbf{H}_{db}$ and $\mathbf{G}_{db}$, while $\mathbf{B}_{tb}$ mixes columns from $\mathbf{D}_{tb}$ and $\mathbf{S}_{tb}$.

The coefficient matrix in Eq. (5) has some important features stemming from the inherent BEM characteristics and the special computational strategy used. The matrix is in fact fully populated, non-symmetric and not definite. Such features are common to the matrices generally produced by the BEM. Moreover, the off-diagonal blocks $\mathbf{H}_{dt}$ and $\mathbf{S}_{td}$ are characterized by the presence of high-strength terms. Such terms originate from the geometrical coincidence of the two crack surfaces, that implies the presence of singular terms in the related blocks. When collocating on the displacement crack surface, for example, both the collocation point $\boldsymbol{x}_0^-$ and the geometrically coincident point $\boldsymbol{x}_0^+$ are singular, thus generating high-strength terms in both $\mathbf{H}_{dd}$ and $\mathbf{H}_{dt}$. Moreover it is to be noted that the blocks $\mathbf{D}$ and $\mathbf{S}$, originating from the collocation of the hypersingular boundary integral equation, i.e. the traction equation, contain terms whose strength is considerably higher with respect to those contained in the blocks $\mathbf{H}$ and $\mathbf{G}$.

The solution of fully populated, non-symmetric and not definite systems, especially when accuracy and reliability are of primary concern, is usually tackled by direct methods, such as Gauss elimination, as they are easy to implement, robust and tend to require predictable time and storage resources. However, when dealing with large three-dimensional systems, involving several thousands of equations, the use of direct solvers becomes too expensive, scaling poorly in terms of operations count and memory requirements. In such cases iterative solvers may represent a preferable choice, becoming mandatory for very large systems (Benzi, 2002).

The application of hierarchial matrices in conjunction with iterative solvers to DBEM matrices of the form given in Eq. (5) is discussed in the next section.

3. Hierarchical matrices for DBEM

In this section, the use of hierarchical matrices for the approximation and solution of systems of equations produced by the DBEM is discussed. Before going into the details of the method, it is useful to give a summary of the conditions that must be met, and the steps that must be carried out, to obtain the hierarchical representation.

The overall objective of this special format is to reduce the storage requirements as well as to speed up the operations involving the matrix, by representing the matrix itself as a collection of blocks, some of which admit a particular approximated representation that can be obtained by computing only few entries from

the original matrix. These special blocks are called *low rank blocks*. The blocks that cannot be represented in this way must be computed and stored entirely and are called *full rank blocks*.

Low rank blocks constitute an approximation of suitably selected blocks of the discrete integral operator based, from the analytical point of view, on a suitable expansion of the kernel of the continuous integral operator (Tyrtyshnikov, 1996; Goreinov et al., 1997; Bebendorf, 2000; Bebendorf and Rjasanow, 2003). This expansion, and consequently the existence of low rank *approximants*, is based on the *asymptotic smoothness* of the kernel functions, i.e. on the fact that the kernels $U_{ij}(\boldsymbol{x}_0, \boldsymbol{x})$ and $T_{ij}(\boldsymbol{x}_0, \boldsymbol{x})$ are singular only when $\boldsymbol{x}_0 = \boldsymbol{x}$. For more precise information about asymptotic smoothness the interested reader is referred to the works of Bebendorf (2000), Bebendorf and Rjasanow (2003) and especially to the paper of Bebendorf and Grzhibovskis (2006), where the application to elastic solids is considered, laying ground to the applicability of ACA to the class of problems considered in the present work. Here it is only mentioned that the asymptotic smoothness represents a sufficient condition for the existence of low rank approximants and that it does not exclude strongly or hyper-singular kernels, like $U_{ijk}(\boldsymbol{x}_0, \boldsymbol{x})$ and $T_{ijk}(\boldsymbol{x}_0, \boldsymbol{x})$. Moreover, the regularity of the boundary over which the approximation is carried out is not requested.

Once the conditions for the approximants existence have been assessed, the subdivision of the matrix into low and full rank blocks is based on geometrical considerations. Every block in the matrix is characterized by two subsets of indices, corresponding, respectively, to the row and column indices. In the standard collocation BEM every row index is associated to a degree of freedom of a collocation node, while every column index is associated to a degree of freedom of a discretization node, whose coefficient is computed by integrating on those elements to which the node itself belongs. Every block is then related to two sets of boundary elements, the one containing the collocation points corresponding to the row indices, here denoted by Ω_{x_0}, and the one grouping the elements over which the integration is carried out, denoted by Ω_x, that contains the nodes corresponding to the columns. If these two sets of boundary elements are *separated*, then the block will be represented and stored in low rank format, while it will be entirely generated and stored in full rank format otherwise. The blocks meeting the requirement of separation are called *admissible*. A schematic of the process leading to the boundary subdivision, and the correspondence with the suitable matrix block, is illustrated in Fig. 1. In the figure, both the cluster of collocation points, related to a set of rows of the collocation matrix, and the cluster of integration elements, which contain the nodes related to the columns of the matrix, are shown. As schematically illustrated, the admissibility condition is actually checked choosing suitable boxes framing the two clusters. This strategy is dictated by the need of reducing the computational costs of the boundary subdivision and especially the following admissibility check.

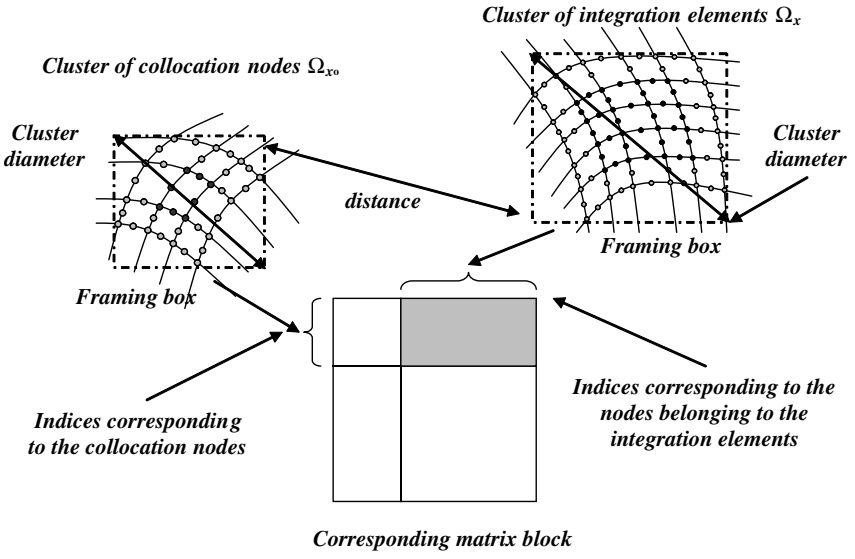

Fig. 1. Schematic of the boundary subdivision process.

The entire process leading to the subdivision in sub-blocks and to their further classification is based on a previous *hierarchical partition* of the matrix index set aimed at grouping subsets of indices corresponding to contiguous nodes and elements, on the basis of some computationally efficient geometrical criterion. This partition is stored in a binary tree of index subsets, or *cluster tree*, that constitutes the basis for the following construction of the hierarchical *block* subdivision, that will be stored in a quaternary *block tree*. Such a process of hierarchical subdivision and tree generation will be further illustrated in the following sections; here it is important to focus on the fact that the block (quaternary) tree stems from the index (binary) tree.

As the admissible blocks have been located, their approximation can be computed. While in fast multipole or panel clustering methods the knowledge of the explicit form of the kernel expansion is required in order to approximate the integral operator, low rank blocks can be generated directly by computing some entries from the original blocks, through ACA algorithms. It is important to highlight that the ACA allows to reach adaptively the *a priori* selected accuracy. These features make such a technique particularly appealing, as it is not necessary to modify or rewrite the routines for the boundary integration in previously developed codes.

Once the basic hierarchical representation has been set up, the collocation matrix can be treated in different ways to obtain the system solution. It is worth noting that the representation obtained by ACA is not yet optimal in terms of storage requirements. The low rank blocks can be in fact *recompressed*, taking advantage of the reduced singular value decomposition (SVD) (Bebendorf, 2001), that allows a further storage reduction without accuracy penalties. Moreover, since the initial matrix partition is generally not optimal (Hackbusch et al., 2004), once the blocks have been generated and recompressed, the entire structure of the hierarchical block tree can be modified by a suitable *coarsening* procedure (Grasedyck, 2005). These consecutive manipulations have the objective of further reducing the storage requirements and speeding up the solution maintaining the preset accuracy. It is important to note that such recompression schemes can be applied sequentially immediately after the blocks generation. When a block has been generated, it can be immediately recompressed. Afterwards, a collection of four contiguous recompressed blocks can be tested for coarsening. This fact implies that the needed memory is less than that required for storing the ACA generated matrix.

As an optimal coarsened representation is obtained, the solution of the system can be tackled either by direct solvers or iterative methods. In both cases, the efficiency of the solution relies on the use of a special arithmetic, i.e. a set of algorithms that implement the operations on matrices represented in hierarchical format, such as addition, multiplication, and inversion (Grasedyck and Hackbusch, 2003). For a direct solution, the computation of the hierarchical LU decomposition of the collocation matrix is needed to carry out an effective hierarchical inversion (Bebendorf, 2005). Iterative solutions, on the other hand, are mainly based on the efficiency of the matrix–vector product, but can be noticeably sped up by the use of suitable preconditioners. An effective preconditioner for BE matrices based on hierarchical LU decomposition has recently been proposed by Bebendorf (2005).

In the following the mentioned points will be further developed and the main algorithms involved will be discussed. The modifications required to take into account the presence of cracks will be pointed out.

3.1. Boundary subdivision and cluster tree

The construction of a partition of the matrix index set is the basis for the following definition of the hierarchical block tree. The objective of the partition is to subdivide the index set into subsets (or clusters) of indices corresponding to contiguous boundary element nodes. Such process leads to the identification of separate or not separate couples of boundary element groups. In the case of three-dimensional elastic problems, as three different indices are associated to each discretization point, it is preferable to partition the set of the boundary nodes indices itself. The process starts from the complete set of indices $I = \{1, 2, \ldots, n\}$, where n denotes the number of collocation points. This initial set constitutes the *root* of the tree. Each cluster in the tree, called *tree node*, not to be confused with geometrical discretization nodes, is split into two subsets, called *sons*, on the basis of some selected criterion. The common tree node from which two sets originate is called the *parent*. The tree nodes that cannot be further split are the *leaves* of the tree. Usually a node cannot be further split when it contains a number of indices equal to or less than a minimum number $n_{\min}$, called *cardinality* of the tree, previously fixed.

However the procedure must be slightly modified for the analysis of cracked configurations through the DBEM. The crack can be either embedded or emanate from a surface, but in any case it is located *inside* the boundary surface and its geometry is usually clearly distinguishable from the geometry of the boundary. This circumstance naturally induces a first distinction, dictated by the geometry, between boundary and crack nodes. Moreover, as already mentioned, crack modeling requires special considerations: the crack is discretized by using discontinuous elements, collocating displacement equations on the nodes belonging to one crack surface and traction equations on the other. As different integral operators, or kernels, correspond to displacement and traction integral equations, it is then appropriate to further distinguish between the nodes on one crack surface and the nodes on the other one; this fact results in the two sons of the *crack nodes cluster* being subdivided into the cluster containing the nodes on which displacement equations are collocated and the one corresponding to the nodes on which the traction equations are collocated. This subdivision and the sets of node indices stemming from this process are graphically represented in Fig. 2. The algorithm used for the construction of the cluster tree is reported in Appendix A, Algorithm 1.

3.2. Block tree and admissibility condition

The block tree is built recursively starting from the complete index set $I \times I$ (both rows and columns) of the collocation matrix and the previously found cluster tree. The objective of this process is to split hierarchically the matrix into sub-blocks and to classify the leaves of the tree into admissible (low rank) or non-admissible (full rank) blocks. The classification is based on a geometrical criterion that assesses the separation of the clusters of boundary elements associated to the considered block. Such a criterion takes into consideration the features of the boundary mesh. For 3D DBEM, eight-noded continuous and discontinuous quadrilateral elements are used. Let Ω_{x_0} denote the cluster of elements containing the discretization nodes corresponding to the row indices of the considered block and Ω_x the set of elements over which the integration is carried out to compute the coefficient corresponding to the column indices. The admissibility condition can be written

$$\min(\mathrm{diam}\Omega_{x_0}, \mathrm{diam}\Omega_x) \leqslant \eta \cdot \mathrm{dist}(\Omega_{x_0}, \Omega_x) \tag{6}$$

where $\eta > 0$ is a parameter influencing the number of admissible blocks on one hand and the convergence speed of the adaptive approximation of lowrank blocks on the other hand (Börm et al., 2003).

Since the actual diameters and the distance between two clusters are generally time consuming to be exactly computed, the condition is usually assessed with respect to bounding boxes parallel to the reference axes (Giebermann, 2001; Grasedyck, 2005), as already mentioned commenting on Fig. 1. In this case Ω_{x_0} and Ω_x in Eq. (6) are replaced by the boxes Q_{x_0} and Q_x. The bounding box clustering technique adopted in the present work is generally used for its simplicity, although it produces non-optimal partitions that can be improved by suitable procedures, as will be illustrated in the following. Other clustering techniques able to produce better initial partitions have been proposed in the literature. The construction using the *principal component analysis* (Bebendorf, 2006) significantly improves the quality of the initial partition.

The algorithm for the block tree generation is graphically illustrated in Fig. 3. Starting from the root (the entire matrix), each block is subdivided into four sub-blocks until either the admissibility condition is satisfied

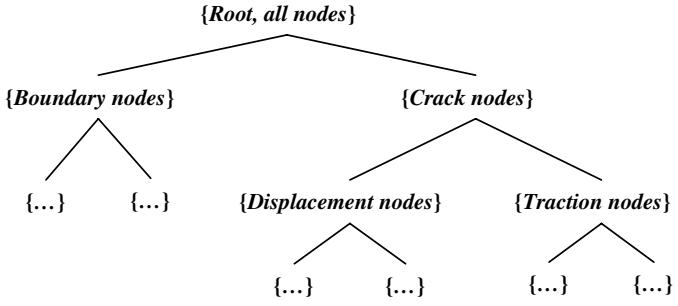

Fig. 2. Index sets induced by presence of cracks.

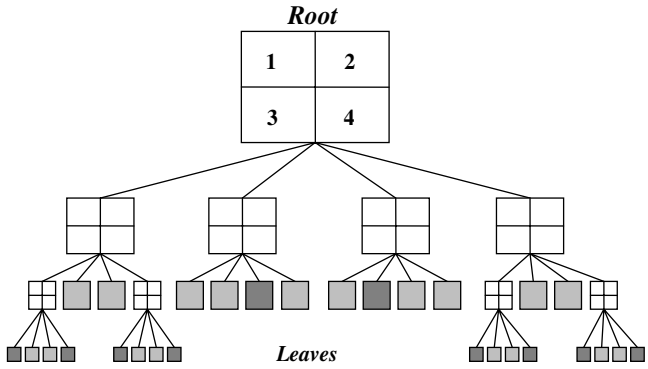

Fig. 3. Graphical illustration of the block splitting procedure.

or the block is sufficiently small that it cannot be further subdivided. The clear grey boxes represent low rank blocks while the dark grey boxes are the full rank ones. The presence of cracks requires some extra considerations. As illustrated in Fig. 2, when a crack is present, the second level of the cluster tree has two nodes, the first corresponding to the discretization nodes on the boundary and the other corresponding to those on the crack. While it is absolutely acceptable to check the admissibility condition for the matrix block corresponding to collocation on the boundary nodes and integration on the two coincident crack surfaces *considered as a whole*, the *vice versa* is not true. When generally collocating on the crack nodes, two different kinds of boundary integral equations are being evoked, namely displacement equations and traction equations. These correspond to different integral operators that should be approximated separately. Besides the admissibility condition (6), the supplementary constraint that the block corresponding to collocation on the nodes of the crack cluster *as a whole* and integration on the boundary *is inadmissible* must be considered. In other words, referring to Eq. (5), if the condition (6) is satisfied, it is admissible to approximate the two submatrices $\mathbf{H}_{bd}$ and $\mathbf{H}_{bt}$ through a single approximate low rank block, while it is not admissible to check the condition (6) for the sub-matrix comprised of $\mathbf{A}_{db}$ and $\mathbf{B}_{tb}$. This further condition may induce a characteristic asymmetric structure on the hierarchical block tree, as shown in Fig. 4. Notice that the dashed line appearing between the two white blocks does not separate it into two sub-blocks, but is drawn only to point out the lost block tree symmetry. Finally, it is important to consider specifically the assessment of the condition (6) when clusters of boundary elements and crack elements are involved at the same time. As mentioned above, the admissibility condition is generally checked considering framing boxes; however cracks are always *contained* by the external boundary

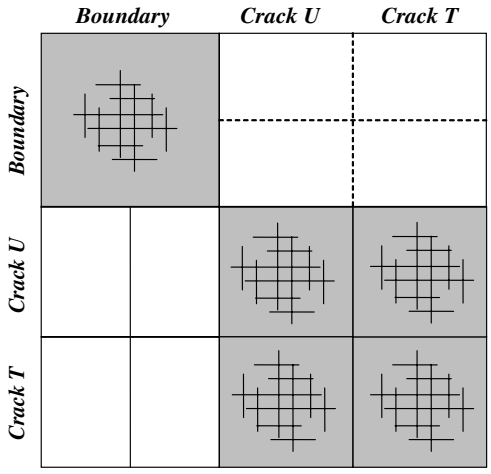

Fig. 4. Structure of the hierarchical matrix including crack surfaces.

and the common procedure, being the boxes one inside the other, could lead to the premature classification of some perfectly admissible blocks as inadmissible. To avoid such a circumstance, a special procedure has been devised. When the distance between a cluster of crack elements and a cluster of boundary elements is being computed, only the crack elements are framed by a box, and the distance between the two clusters is computed considering the boundary cluster element by element. This procedure avoids the premature skipping of admissible blocks, especially in the very favorable case of embedded cracks. The element by element distance check, if carried out only when crack elements are involved, does not result too expensive computationally.

The extended admissibility condition is used in the algorithm for the generation of the block tree reported in Appendix A, Algorithm 2.

3.3. Low rank blocks and ACA algorithm

Let M be an $m \times n$ admissible block in the collocation matrix. It admits the low rank representation

$$M \simeq M_k = A \cdot B^{\mathrm{T}} = \sum_{i=1}^{k} a_i \cdot b_i^{\mathrm{T}} \tag{7}$$

where A is of order $m \times k$ and B is of order $n \times k$, where k is the *rank* of the new representation. The approximating block M_k satisfies the relation $\|M - M_k\|_F \leqslant \varepsilon \|M\|_F$, where $\|\cdot\|_F$ represents the *Frobenius norm* and ε is the set accuracy. Sometimes it is useful to represent the matrix using the alternative sum representation, where a_i and b_i are the ith columns of A and B, respectively. The approximate representation allows storage savings with respect to the full rank representation and speeds up the matrix–vector product (Grasedyck and Hackbusch, 2003).

Different ACA algorithms can be used to generate the approximate blocks. The original algorithm was proposed by Bebendorf (2000) and was further developed by Bebendorf and Rjasanow (2003). Grasedyck (2005) proposed the so called ACA+ algorithm and compared its performances to those of the standard scheme. Bebendorf and Grzhibovskis (2006) proposed a strategy for overcoming some problems that may arise when populating some low rank blocks involving the interaction, through double layer kernels, of sets of coplanar elements. Useful illustrations of the basic ACA scheme can be found in the works of Kurz et al. (2002) and Bebendorf and Kriemann (2005), while the reader is referred to the work of Grasedyck (2005) for ACA+. In the present work, also a slightly different scheme has been used to circumvent some problems originating when computing some particular blocks, as illustrated in the following.

The above-mentioned schemes allow to reach *adaptively* the *a priori* set accuracy ε and are substantially based on the computation of selected columns and rows of the original block that, suitably manipulated, furnish exactly the columns a_i and b_i appearing in Eq. (7). Different schemes often differ for the choice of the pivots, that can have a noticeable effect on the convergence and quality of the approximation (Bebendorf and Kriemann, 2005). Once a new column a_i and row b_i^{T} have been generated and added to those previously computed, the convergence toward the required accuracy is checked against a suitable stopping criterion. If the criterion is satisfied the computation is stopped, else a new couple column–row is generated and stored.

The stopping criterion is based on the assessment of the convergence of the approximating block in terms of the Frobenius norm (Bebendorf, 2000; Kurz et al., 2002). Since the original blocks are not accessible, only the partial approximations M_k, with k running, are used to check the convergence. The Frobenius norm can be computed by the following recursive formula

$$\|M_k\|_F^2 = \|M_{k-1}\|_F^2 + 2\sum_{i=1}^{k-1}(a_k^{\mathrm{T}} a_i)(b_i^{\mathrm{T}} b_k) + \|a_k\|_F^2 \|b_k\|_F^2 \tag{8}$$

where a_k and b_k represent the column and row computed at the kth iteration. A suitable stopping criterion can be expressed as

$$\|a_k\|_F \|b_k\|_F \leqslant \varepsilon \|M_k\|_F \tag{9}$$

that prescribes to stop the iteration when the inequality is satisfied for a required preset accuracy ε.

The construction of the approximating block not only reduces the storage needed to represent an admissible block, but also reduces the assembly time for the set-up of the collocation matrix, as the integration is carried out only on those elements that allow the population of the required columns and rows.

With reference to the form of the DBEM system of Eq. (5), some additional consideration on the construction of the approximating blocks is convenient.

The blocks contained in $\mathbf{A}_{bb}$ and $\mathbf{A}_{db}$ include a mix of columns from the blocks $\mathbf{H}_{bb}$ and $\mathbf{G}_{bb}$ and $\mathbf{H}_{db}$ and $\mathbf{G}_{db}$, respectively, and require some attention. Moreover, some of the columns from the corresponding **H** and **G** blocks may give a contribution to the right hand side. Let us suppose that, following the application of the boundary conditions, the low rank block M belonging to the sub-matrix $\mathbf{A}_{bb}$ is comprised of many **H** columns and few **G** columns and that few **G** columns contribute to the right hand side, see Fig. 5. In this case, it may be not convenient to generate the approximation of the entire **G** block (the white block belonging to $\mathbf{G}_{bb}$ in Fig. 5); on the contrary it is more effective to generate the approximation of the entire **H** block through ACA, then to annihilate the terms in the rows b_i^{T} corresponding to the columns to be replaced with the **G** columns, and eventually to compute exactly the few needed **G** columns and add them to the representation (7), using row vectors filled with zeros and ones placed in the positions corresponding to the columns to be replaced; the others **G** columns, among the few exactly computed, contribute to the right hand side, through appropriate coefficients.

The choice between computing a block through ACA and computing few columns exactly relies on the number of columns from a block **H** and **G** that are actually needed for the construction of a specific **A** block. If the number of needed columns is much less than the average rank, it is convenient to compute the columns exactly, since the ACA representation of the few columns would require more columns (and rows) than their exact representation.

Analogous considerations hold for the block $\mathbf{B}_{tb}$, that mixes columns from $\mathbf{S}_{tb}$ and $\mathbf{D}_{tb}$.

The blocks contained in $\mathbf{H}_{bd}$ and $\mathbf{H}_{bt}$ do not require special considerations and can be computed through standard ACA. However, it is interesting to observe that, if the crack is sufficiently far from the boundary, both these sub-matrices belongs to a single big low rank block: this is the block for which the numerical compression works better.

The blocks contained in $\mathbf{H}_{dd}$, $\mathbf{H}_{dt}$, $\mathbf{S}_{td}$ and $\mathbf{S}_{tt}$ often stem from the integration of T_{ij} and T_{ijk} over clusters of coplanar elements (case of plane crack). In such circumstance the standard ACA may fail, as shown by Grasedyck (2005) with a counterexample for partial pivoting. To avoid this potential problem, and the consequent loss of accuracy, ACA+ can be used for the approximation of these blocks. In this work, however, a different strategy, based on the computation of more than one rows at each ACA iteration, has been used. The strategy is referred to as *big-volume search* (Tyrtyshnikov, 2000; Bebendorf and Kriemann, 2005) and, although slightly more expensive then standard ACA, has proven to be very reliable in the performed numerical experiments. Briefly, the search of the pivot is not limited to a single row per iteration, but is extended to more rows that are generated and stored at every ACA step. The efficiency of the scheme is hence based on the availability of more matrix entries.

3.4. Block recompression and tree coarsening

After the blocks have been generated, a further reduction of the required memory can be achieved by suitable recompression schemes (Bebendorf, 2001; Grasedyck, 2005), so that the amount of memory required for the final storage is lower than that needed for the ACA generated matrix. Moreover the recompression schemes further speed up the computation, maintaining the desired preset accuracy ε.

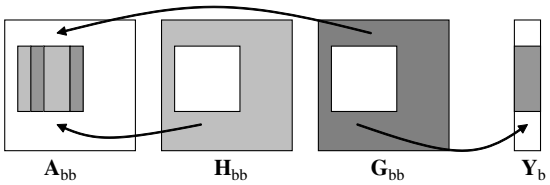

Fig. 5. Construction of an A block.

Two different recompression schemes can be applied: one acts on the single leaves while the other modifies the entire tree structure, through a process of *reabsorption* of the leaves.

The block recompression, i.e. the first scheme, is sometimes referred to as *truncation* (Grasedyck and Hackbusch, 2003; Börm et al., 2003) and is based on the SVD of low and full rank blocks. Since the size of the full rank leaves is bounded by $n_{\min}$, their SVD can be efficiently computed by suitable available efficient algorithms, like the `LAPACK dgesvd.f`; for the low rank leaves, on the other hand, it is possible to perform a reduced SVD (Bebendorf, 2001). Once the SVD has been performed, the singular values for which the condition $\sigma_i < \varepsilon\sigma_1$ holds, if existing, can be discarded together with the corresponding columns and rows without accuracy penalties. Such procedure operates on single blocks, reducing the size of low rank blocks and converting the full rank blocks satisfying the previous condition into low rank blocks. It is important to emphasize the role of the accuracy ε that appears in all the computations. The requested accuracy could also be lowered, if a less accurate representation of the matrix would be needed for special purposes. The importance of this fact will become apparent in the construction of a preconditioner for iterative solvers.

Besides the blockwise SVD it is possible to apply a further recompression, referred to as *tree coarsening*, aimed at modifying the entire tree structure. The coarsening tries to unify groups of four leaves that are sons of the same block tree node. If some conditions are met, the SVD decomposition of the unification of the four blocks is computed and, if after discarding the smaller singular values along with the corresponding columns and rows, the result requires less storage, the four blocks are unified, or reabsorbed, in the parent tree node. Note as this scheme can be applied when four adjacent sub-blocks have been generated and recompressed. The procedure that performs the coarsening has been presented by Grasedyck (2005).

Two points are stressed here: (a) the block recompression is applied immediately after the block generation and not after the generation of the entire matrix and this allows an actual reduction of the storage requirements; (b) the recompression schemes provide a valuable tool for the construction of an effective preconditioner (Bebendorf, 2005), based on the computation of a *coarse* approximation of the collocation matrix, as described in the following.

3.5. Hierarchical arithmetic

To carry out operations on hierarchical matrices, it is necessary to define a suitable hierarchical arithmetic. The operations involving the entire hierarchical matrices (operations between matrices, trees or sub-trees) are based on elementary operations between low or full rank blocks (operations between blocks or leaves). Often such operations, involving the SVD, require a truncation with respect to a previously set accuracy, which acquires thus a fundamental importance in all the considered algorithms. *All the hierarchical operations are defined with respect to a set accuracy*.

The main difficulty when implementing such algorithms is distinguishing between the full range of possible different cases. The multiplication, for example, is defined with respect to the two block factors and the so called target block. Each of these blocks could be subdivisible or not subdivisible and, if not subdivisible, i.e. leaf, could be low or full rank. Each of these cases requires separate treatment aimed at reducing the overall complexity of the operation.

Rigorous information on addition between hierarchical matrices, truncation with respect to a given accuracy, matrix–vector multiplication, matrix–matrix multiplication and hierarchical inversion can be found in the works of Hackbusch (1999) and Grasedyck and Hackbusch (2003), where also some algorithms are given and their arithmetic complexity is analyzed. A collection of useful algorithms for practical implementation is given by Börm et al. (2003). The hierarchical LU decomposition, based on the previous arithmetic, is discussed by Bebendorf (2005). The interested reader is referred to the mentioned works for details.

3.6. System solution

The solution of the system can be obtained either directly, through hierarchical matrix inversion (Grasedyck and Hackbusch, 2003), or indirectly, through iterative solvers that exploit the efficient matrix–vector product in low rank format (Bebendorf, 2005, 2006).

The iterative solvers can be used with or without preconditioners. When the condition number is high and slows down the convergence rate, as is often the case when dealing with BEM systems, a preconditioner can be computed taking full advantage of the representation in hierarchical format. If $Ax = b$ is the system to be solved, then a left preconditioner is an easily invertible matrix P such that the condition number of the system $P^{-1}Ax = P^{-1}b$ results lower than the original one, improving thus the convergence rate of the iterative solver.

The hierarchical representation offers the opportunity to build naturally an effective preconditioner (Bebendorf, 2005; Grasedyck, 2005). A *coarse* preconditioner can be obtained by first generating a coarse approximation $A(\varepsilon_p)$ of the original collocation matrix $A(\varepsilon_c)$, where the relationship $\varepsilon_p > \varepsilon_c$ holds, ε denoting the set accuracy for the hierarchical representation. This coarse approximation, with reduced memory storage, can then be decomposed through hierarchical LU decomposition to give the preconditioner P. The resulting system

$$(LU)^{-1}Ax = (LU)^{-1}b \tag{10}$$

has a lower condition number and the convergence rate of iterative solvers is noticeably improved. It should be noted that in Eq. (10) there is no need to compute the matrix product $(LU)^{-1}A$, as it is more efficient to use directly the forward and backward substitution for the inversion of the matrix LU in iterative solution schemes. Backward and forward substitutions take in fact full advantage of the hierarchical matrix–vector multiplication and this is the reason why the preconditioner is LU decomposed.

In this work, the GMRES (Saad and Schultz, 1986) with a coarse hierarchical left preconditioner has been used as solver.

3.7. Some details about code implementation

In the present work, subroutines and functions for the treatment of hierarchical matrices have been implemented in `FORTRAN 90`. Different modules have been implemented to deal with the diverse tasks involved in the hierarchical treatment of the boundary element elastostatic problems. The basic module `Hdata` implements all the needed data structures; the module `Htree` contains all the procedures that build the cluster and block trees starting from the boundary information; the module `Hsetup` implements all the subroutines that set up the hierarchical matrix computing low and full rank blocks, also forcing the boundary conditions; the module `Hblocks` implements all the procedures that work on single blocks, like the full and reduced SVD; eventually, the module `Harithmetics` implements the arithmetics on the trees (addition, multiplication, LU decomposition and inversion).

The basic data structure is called `BlockNode` (Fig. 6) and allows the natural treatment of either the full or low rank blocks of the hierarchical matrix.

The field `IndexSet` identifies the position of the block in the full matrix, identifying rows and columns on the basis of a previously defined index partition; the field `Identifier` can be assigned three different values that allow to establish if the block has sons, i.e. if it is not a leaf, or, in case it is a leaf, if it is low or full rank; the pointers `parent`, `son11`, `son12`, `son21` and `son22` are needed to build and maintain the structure of the quaternary hierarchical tree. It is worth stressing again that some operations, like the coarsening of the block tree, rely on the possibility of moving through the tree in both directions, i.e. from root to leaves and from leaf to root.

If the block is a leaf, on the basis of the value taken by `Identifier` the suitable data structure is allocated to store the information. If the block is full rank then the array `FBL` is allocated, its dimension being inferred by `IndexSet`. On the other end, if the block is low rank, then the pointers `HeadColums`, `TailColums`, `HeadRows` and `TailRows` are associated: they point to the head and tail of two different lists of vectors collecting, respectively, the columns and rows of the low rank representation.

4. Numerical experiments

In this section results obtained by applying the developed computational scheme are discussed. All the computations have been performed using an Intel® Core™ 2 Duo Processor T5500 (1.66 GHz) and 2 GB of RAM.

```
TYPE BlockNode

    INTEGER, DIMENSION(6) :: IndexSet

    INTEGER :: Identifier

    INTEGER :: Rank

    TYPE(BlockNode), POINTER :: parent

    TYPE(BlockNode), POINTER :: son11

    TYPE(BlockNode), POINTER :: son12

    TYPE(BlockNode), POINTER :: son21

    TYPE(BlockNode), POINTER :: son22

    TYPE(vector), POINTER :: HeadColumns

    TYPE(vector), POINTER :: TailColumns

    TYPE(vector), POINTER :: HeadRows

    TYPE(vector), POINTER :: TailRows

    DOUBLE PRECISION, DIMENSION(:,:), POINTER :: FBL

END TYPE BlockNode
```

Fig. 6. Basic data structure for hierarchical matrices.

4.1. Uncracked elastic bracket

A mechanical element, Fig. 7, is first analyzed, to obtain some insight into the structure of the hierarchical collocation matrix for structures without cracks. The mechanical bracket has the two central cylinders clamped and is anti-symmetrically loaded at the holes, in such a way that the resulting load is a moment lying along the central axis. A mesh with 1032 elements and 3094 nodes has been considered. The standard method required 350 s for the generation of the collocation matrix and 1484 s for the solution of the system through Gauss elimination. These times have been compared with the related times required by the fast method, to obtain both the assembly speed up ratio and the solution speed up ratio at different parameter settings. In particular, the standard assembly time has been compared with the time needed for the ACA generation of the collocation matrix, while the standard solution time has been compared with the fast solution time, which is comprised of the collocation matrix compression and coarsening time, of the preconditioner generation, coarsening and LU decomposition time and of the GMRES iteration time.

A first set of analyses has been performed to investigate the effect of the preconditioner accuracy on the convergence of the iterative solution. For this purpose, the accuracy of the collocation matrix has been set

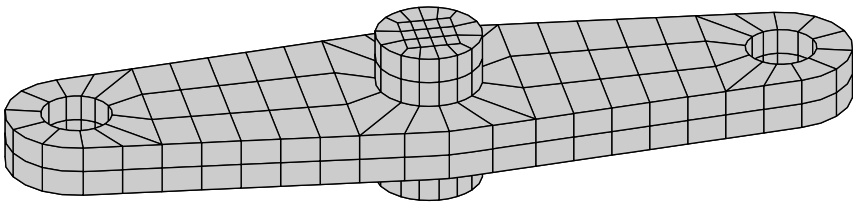

Fig. 7. Analyzed 3D configuration.

to $\varepsilon_c = 10^{-5}$, the admissibility parameter has been chosen as $\eta = \sqrt{2}$ and the minimal blocksize has been set to $n_{min} = 36$. The GMRES relative accuracy is 10^{-8}.

With these settings, the hierarchical collocation matrix is stored using only 35.61% (after coarsening) of the memory required for the allocation of the original matrix. The matrix is generated through ACA in 198 s, i.e. 57% of the standard assembly time, and it is recompressed and coarsened in 216 s. It may be of interest to mention that, in this specific application, 60% of the collocation matrix has been generated through ACA in 55 s (28% of the hierarchical assembly time). The remaining 40% of the collocation matrix is in full rank format and its evaluation requires the remaining 72% of the hierarchical assembly time, mainly due to the computation of the singular integrals whose evaluation is needed to populate such blocks. The approximate solution accuracy is $\|x - \tilde{x}\|_{L^2}/\|x\|_{L^2} = 2.9 \times 10^{-4}$. Both the collocation matrix coarsening time and the solution accuracy are independent from the preconditioner accuracy. The GMRES converges towards the same approximate solution, whose accuracy depends only on the accuracy of $\widetilde{A}$ (the collocation matrix) and $\tilde{b}$ (right hand side).

Table 1 reports the storage memory needed to store the preconditioner, expressed as percentage of the full collocation matrix, the time required to set up and decompose the preconditioner, the time and the number of iterations required by the GMRES to converge and, finally, the solution speed up ratio, defined as the ratio between the fast solution time and the standard solution time. Times are expressed in seconds. It is interesting to note as the time required to set up the preconditioner and for its LU decomposition grows when the preconditioner required accuracy grows (as ε_p becomes smaller the required accuracy increases). On the other hand, quite naturally, the number of GMRES iterations and the iterative solution time decrease when ε_p decreases. In the extreme case of the preconditioner retaining the same accuracy as that of the collocation matrix, the preconditioner LU decomposition can be used for a direct solution. It is worth noting, however, that a very coarse preconditioner ($\varepsilon_p = 10^{-1}$) provides the fastest solution, as is evident from the reported solution speed up ratios, while both the unpreconditioned GMRES and the Jacobi preconditioned GMRES fail to converge.

Fig. 8 shows the blockwise structure of the collocation matrix as generated by ACA, the coarsened collocation matrix and the structure of the coarsest effective preconditioner ($\varepsilon_p = 10^{-1}$). The number of blocks obtained for the selected minimal blocksize goes from 3547 in the ACA generated matrix to 2545 in the coarsened matrix, while the preconditioner counts 1063 blocks. Every block is filled with a tone of grey proportional to the ratio between the memory required for low rank representation and the memory in full rank format. Full rank blocks are black while almost white blocks are those for which the numerical compression works better. It is worth noting the reduction in the number of blocks obtained going from the ACA generated matrix to the coarsened matrix. The big difference in the number of blocks is due to the suboptimal choice of the admissibility parameter η, as will be discussed in the following. It is worthwhile to remember also that

Table 1
Storage, times and speed up ratios for different preconditioner accuracies

ε_p	Storage (%)	Setup (s)	LU (s)	GMRES (s)	Iterations	Speed up
No. Prec.	0.00	0.0	0.0	3470.8[a]	5000[a]	2.49[a]
Jacobi	0.00	0.0	0.0	3474.6[b]	5000[b]	2.49[b]
5×10^{-1}	1.45	23.9	12.7	3906.0[c]	5000[c]	2.80[c]
1×10^{-1}	4.95	28.4	45.5	10.0	29	0.19
5×10^{-2}	6.54	31.5	63.2	8.7	25	0.21
1×10^{-2}	10.59	47.3	117.7	4.0	10	0.25
5×10^{-3}	12.62	58.5	153.2	3.9	9	0.28
1×10^{-3}	17.31	82.1	252.6	3.0	6	0.36
5×10^{-4}	19.38	90.8	303.4	2.6	5	0.40
1×10^{-4}	25.20	104.2	476.7	2.5	4	0.53
5×10^{-5}	27.89	97.9	586.2	2.1	3	0.60
1×10^{-5}	35.61	3.1[d]	988.2	2.3	3	0.81

[a] Reached GMRES relative accuracy: 3.0×10^{-6} (no convergence).
[b] GMRES relative accuracy: 3.0×10^{-2}.
[c] GMRES relative accuracy: 6.1×10^{-8}.
[d] Only the time for copying the collocation matrix.

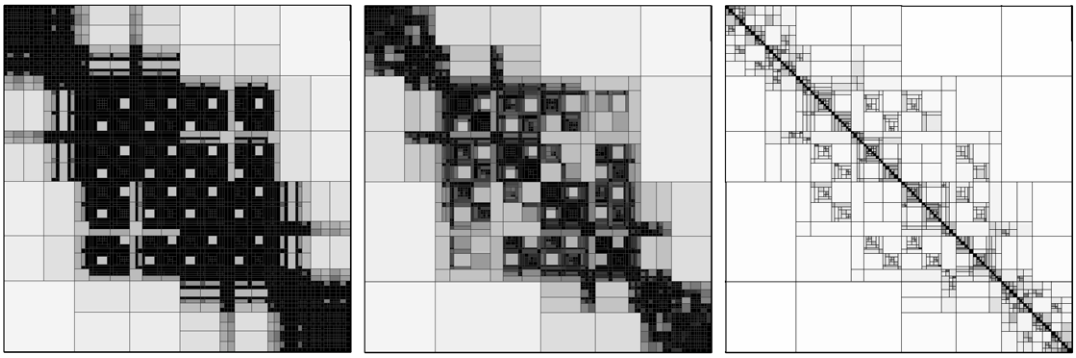

Fig. 8. Block-wise representation of the ACA generated matrix, the coarsened matrix and the preconditioner.

the required memory is not that needed for the ACA generated matrix, but directly that (lower) required by the coarsened matrix. Here the two different compressions are shown to illustrate the mechanism of coarsening, but the operation can be performed recursively while populating the blocks.

Table 2 reports memory requirements before and after coarsening, assembly time and assembly speed up ratio, solution time (compression and coarsening, preconditioner generation and LU decomposition, GMRES) and solution speed up ratio and the accuracy of the final solution at different values of the collocation matrix requested accuracy. The tests have been performed by setting $\varepsilon_p = 1.0 \times 10^{-1}$ and $\eta = \sqrt{2}$.

Memory requirements, assembly times and solution times decrease when the preset accuracy decreases, as the average rank of the approximation is reduced. However, reducing the requested accuracy obviously reduces also the approximation quality of the final solution. From an engineering point of view, the selection of a suitable criterion for selecting the collocation matrix accuracy is of fundamental importance. Note that the L^2 norm used in Table 2 does not give insight into the quality of the approximation for engineering purposes. A node by node check of the solution has confirmed however that, for a selected accuracy $\varepsilon_c = 1.0 \times 10^{-5}$, the average errors are of order 0.1–1.0%. Bigger percentage errors can occur for degrees of freedom whose standard solution values are smaller than the requested accuracy. This consideration suggests to set the accuracy at the same order of magnitude as that of the smaller quantities of interest in the analysis.

Table 3 reports the memory storage before and after coarsening, the number of blocks before and after coarsening, the assembly time and assembly speed up ratio, the coarsening time and the solution speed up ratio at different values of the admissibility parameter. The other parameters have been set to $\varepsilon_c = 10^{-5}$ and $\varepsilon_p = 10^{-1}$. The time for generating and manipulating the preconditioner is independent from η. On the contrary, the time required for coarsening the matrix strongly depends on it. The choice of η directly affects the quality of the ACA generated matrix and a good choice allows to obtain a matrix closer to the optimal matrix produced by the coarsening procedure, as can also be noted from the reduction in the number of blocks. This is the reason of the influence on the coarsening time. Note as the matrices obtained after coarsening require almost the same memory, regardless to the initial ACA generated matrix storage: the coarsening produces in fact an almost optimal hierarchical matrix, reducing the differences related to the choice of η. However, though a larger part of the matrix is generated through ACA, which should lead to a reduction in the assembly time, the average rank of the approximation increases, as the new admissible blocks converge

Table 2
Influence of the collocation matrix accuracy

ε_c	Storage A (%)	Storage B (%)	Assembly (s)	Speed up	Solution (s)	Speed up	$\frac{\|x-\bar{x}\|_{L^2}}{\|x\|_{L^2}}$
10^{-6}	57.49	45.59	218.3	0.62	319.1	0.21	6.0×10^{-6}
10^{-5}	53.62	35.61	197.6	0.57	300.2	0.20	2.9×10^{-4}
10^{-4}	50.09	25.17	180.5	0.52	269.6	0.18	5.6×10^{-3}
10^{-3}	47.15	17.28	169.0	0.48	201.9	0.14	3.4×10^{-2}
10^{-2}	44.17	10.56	159.1	0.45	150.8	0.10	1.6×10^{-1}

Table 3
Influence of the admissibility parameter

η	Storage A (%)	Storage B (%)	No. of blocks	Assembly (s)	Speed up	Coarsening (s)	Speed up
$\sqrt{2}$	53.62	35.61	3547–2545	197.6	0.57	216.2	0.20
3	45.04	34.61	2581–2284	209.2	0.60	117.5	0.13
4	43.73	34.17	2341–2152	218.3	0.62	98.9	0.12
5	43.52	33.63	2257–2044	220.5	0.63	111.2	0.13

more slowly to the preset accuracy. This aspect may have a negative effect on the assembly time, that is however balanced by more relevant reduction in the solution time. Fig. 9 shows the structure of the ACA matrix for two different values of η. It is evident that for $\eta = 4$ more blocks become admissible than for $\eta = \sqrt{2}$.

4.2. Embedded crack

As second configuration a cylinder with an embedded crack is considered, Fig. 10. The cylinder is subjected to uniaxial stress acting on the two bases, so to produce a mode I crack load. A mesh with 800 elements and 3652 nodes is considered. The standard technique requires 805 s to assemble the collocation matrix and 2394 s to solve the system. It is worth noting that the numerical integration of the singular, strongly singular and hypersingular kernels occurring during the assembly of the collocation matrix requires, in this case, 60% of the standard integration time. Since singular integrals occur near the diagonal blocks, which are full rank in the hierarchical representation, such percentage represents the lower bound for the hierarchical assembly

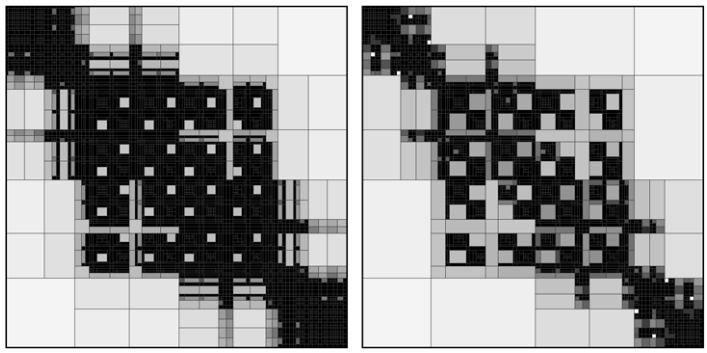

Fig. 9. Block-wise structure of the ACA matrix for $\eta = \sqrt{2}$ and $\eta = 4$.

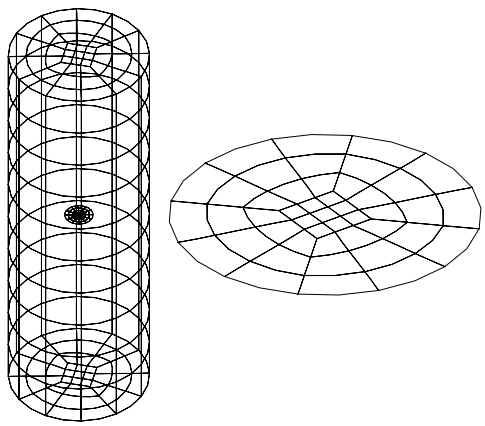

Fig. 10. Cylinder with embedded crack and crack basic mesh.

time. First, a parametric analysis on the influence of the preconditioner accuracy is performed, as in the previous case. For this first set of computations, the collocation matrix accuracy is set to $\varepsilon_c = 10^{-5}$, the admissibility parameter is $\eta = 6$ and $n_{min} = 36$. With this parameters choice the collocation matrix is generated in 689 s, i.e. 85% of the standard assembly time, it is compressed and coarsened in 168 s and is stored using 23.70% of the original space. The accuracy of the final solution is $\|x - \tilde{x}\|_{L^2}/\|x\|_{L^2} = 1.2 \times 10^{-4}$, which is very good in terms of point by point accuracy. Table 4 reports the preconditioner storage, the preconditioner setup and LU decomposition times (in seconds), the GMRES solution time, the number of GMRES iterations and the solution speed up ratio. Again, it can be observed that the required memory, the setup time and the LU decomposition time grow as the required preconditioner accuracy grows. On the contrary, the number of GMRES iterations, and the GMRES time as consequence, decreases as the accuracy grows. Moreover, it is important to note as, also in this case, the best speed up ratio is obtained with the coarsest preconditioner ($\varepsilon_p = 1.0$), while a fine preconditioner could be used as a direct solver. Finally, it should be noted that the construction of the hierarchical preconditioner is actually necessary, as the unpreconditioned GMRES as well as the Jacobi preconditioned GMRES fail to converge.

The influence of the admissibility parameter has been investigated and the results are reported in Table 5. The analysis is performed setting $\varepsilon_c = 10^{-5}$ and $\varepsilon_p = 1.0$. The same considerations as in the previous case hold. Finally, three different meshes have been analyzed to obtain some insight into the behavior of the solver at varying numbers of degrees of freedom. The first mesh has 66 elements and 400 nodes, the second 300 elements and 1352 nodes and the third uses 800 elements and 3652 nodes. The settings are $\varepsilon_c = 10^{-5}$, $\varepsilon_p = 1.0$, $\eta = 6$. Table 6 reports the results obtained for the three different meshes in terms of memory ratio, assembly speed up ratio, solution speed up ratio and number of GMRES iterations. Also the times for standard assembly and standard solution are reported, expressed in seconds. It appears evident as the advantages of the described technique become more relevant with larger meshes. While memory savings are always obtained also for coarse meshes, the assembly and solution speed up ratios are less than one only beyond certain threshold, under which the direct solver performs better. Fig. 11 shows the comparison in terms of required memory and solution time between standard and fast DBEM. It is worth noting the almost linear behavior of the fast DBEM with respect to the number of degrees of freedom.

Table 4
Storage and times for different preconditioner accuracies

ε_p	Storage (%)	Setup (s)	LU (s)	GMRES (s)	Iterations	Speed up
No. Prec.	0.00	0.0	0.0	3642.1[a]	5000[a]	1.52[a]
Jacobi	0.00	0.0	0.0	3644.3[b]	5000[b]	1.52[b]
1×10^{0}	0.86	15.2	0.4	92.4	306	0.12
5×10^{-1}	2.42	15.7	115.2	60.1	197	0.15
1×10^{-1}	5.00	17.9	163.9	12.9	40	0.15
1×10^{-2}	9.06	26.5	273.2	5.9	16	0.20
1×10^{-3}	13.39	47.4	448.6	3.8	8	0.28
1×10^{-4}	18.32	68.6	802.8	2.5	4	0.43
1×10^{-5}	23.69	2.7[c]	1371.6	2.3	3	0.64

[a] The relative GMRES accuracy was 5.9×10^{-5} (no convergence reached).
[b] The GMRES relative accuracy was 8.8×10^{-2}.
[c] Only the time for copying the collocation matrix.

Table 5
Influence of the admissibility parameter

η	Storage A (%)	Storage B (%)	No. of blocks	Assembly speed up	Coarsening (s)	Solution speed up
2	43.94	23.72	4735–2380	0.85	229.7	0.14
4	38.10	23.72	4003–2380	0.85	185.4	0.12
6	35.85	23.70	3643–2338	0.86	168.2	0.12
8	35.15	23.67	3433–2314	0.86	162.4	0.11

Table 6
Memory savings and speed up ratios

Elements	Nodes	Storage (%)	Standard assembly (s)	Speed up	Standard solution (s)	Speed up	Iterations
66	400	69.78	79.4	1.07	3.2	1.23	49
300	1352	40.74	215.5	1.04	122.8	0.28	113
800	3652	23.70	805.5	0.86	2398.8	0.12	306

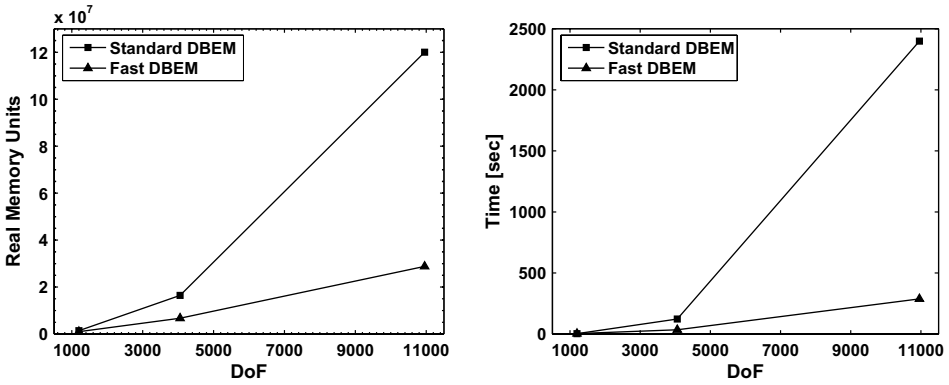

Fig. 11. Comparison between standard and fast DBEM.

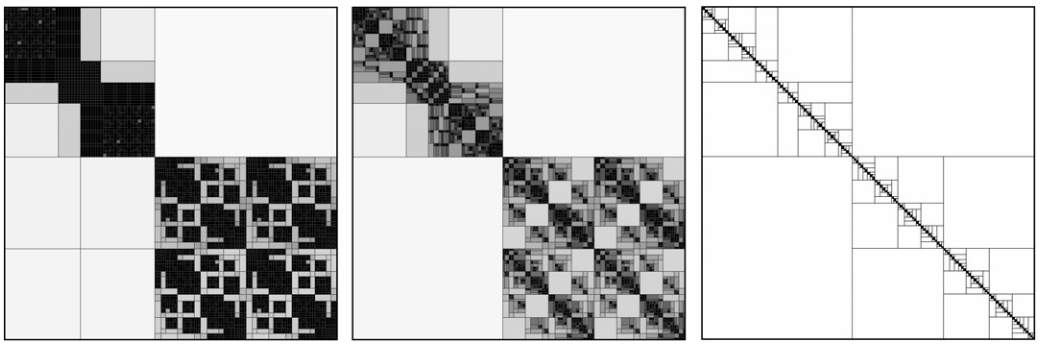

Fig. 12. Block-wise representation of the ACA generated matrix, the coarsened matrix and the preconditioner.

Fig. 12 shows the blockwise structure of the collocation matrix for the finest mesh as generated by ACA, the coarsened collocation matrix and the structure of the preconditioner. The number of blocks goes from 3643 in the ACA generated matrix to 2338 in the coarsened matrix and 430 blocks in the preconditioner. It is interesting to point out how the low rank blocks corresponding to collocation on the boundary and integration on the crack and *vice versa* are clearly distinguishable. The geometry and mesh features have a numerical counterpart in the blockwise structure of the hierarchical matrices.

5. Conclusions

In this paper, a new fast solver for three-dimensional BEM and DBEM was successfully developed and implemented. The method was shown to be very effective and reliable in terms of accuracy. Moreover, it allows a considerable reduction in both the amount of memory needed for storing the system coefficients and the time required for the system solution. In particular, it has been shown that both storage memory and solution time vary almost linearly with respect to the number of degrees of freedom, thus providing considerable savings for large systems in comparison to direct solvers. It has also been demonstrated that

hierarchical matrices are particularly suitable for crack problems. This is due to cracks normally being isolated surfaces which are far from most of the remaining boundary, hence corresponding to large low rank blocks. This allows, as demonstrated, to increase the number of elements on the crack surfaces with only modest increases in storage memory and solution time.

Appendix A. Algorithms for hierarchical DBEM

The algorithms needed for dealing with the presence of cracks in the framework of the fast DBEM based on the use of hierarchical matrices are described in this Appendix.

The first is the algorithm for the generation of the hierarchical cluster tree. It is similar to the one introduced by Giebermann (2001) and used in Grasedyck (2005) and is aimed at generating a geometrically balanced cluster tree, after the initial subdivision between boundary nodes, displacement crack nodes and traction crack nodes. It requires, as input, a set of indices associated to a set of collocation points, the set of the coordinates of such points and the minimum number of points allowed in a subset, i.e. the cardinality. The output of the procedure is the entire structure of the binary tree, from the root to the leaves. Note that $(\mathbf{x}_j)_i$ indicates the ith coordinate of the jth collocation point, while the operator $\#(\cdot)$ gives the number of elements in a set.

The algorithm for the generation of the quaternary block tree is also given. It takes into account the possible presence of cracks, following the considerations developed in Section 3.2.

The output of the procedure, when it is initially called passing the block corresponding to the entire matrix, is the block tree from the root to the leaves, that are classified in low and full rank.

Algorithm 1. Recursive SplitCluster $(s, \mathbf{x}, n_{min})$

if s is the tree root cluster **then**
  define $s_1 = \{i \in s : \mathbf{x}_i \in$ set of *boundary* nodes$\}$
  define $s_2 = \{i \in s : \mathbf{x}_i \in$ set of *crack* nodes$\}$
else if s is the cluster of *all* the crack nodes **then**
  define $s_1 = \{i \in s : \mathbf{x}_i \in$ *set of displacement* crack nodes$\}$
  define $s_2 = \{i \in s : \mathbf{x}_i \in$ set of *traction* crack nodes$\}$
else if $\#s \leqslant n_{min}$ **then**
  set $sons(s) = \{\emptyset\}$
  return
else
  for i=1,3 **do**
    $M_i = max\{(\mathbf{x}_j)_i : j \in s\}$
    $m_i = min\{(\mathbf{x}_j)_i : j \in s\}$
  end for
  find j such that $M_j - m_j$ is the largest
  define $s_1 = \{i \in s : (\mathbf{x}_i)_j \leqslant (M_j + m_j)/2\}$
  define $s_2 = s - s_1$
end if
set $sons(s) = \{s_1, s_2\}$
for i=1,2
  call SplitCluster$(s_i, \mathbf{x}, n_{min})$
end for

Algorithm 2. Recursive SplitBlock$(B_{s\times t}, n_{min})$

if s is the cluster of all the *crack* nodes **then**
  set $sons(B_{s\times t}) = \{B_{\sigma\times\tau} : \sigma \in sons(s), \tau \in sons(t)\}$
else if $B_{s\times t}$ is *admissible* **then**
  set $B_{s\times t}$ as a *low rank* block
  set $sons(B_{s\times t}) = \{\emptyset\}$
else if $\#s < n_{min}$ or $\#t < n_{min}$ **then**
  set $B_{s\times t}$ as a *full rank* block

```
  set sons(B_{s×t}) = {∅}
else
  set sons(B_{s×t}) = {B_{σ×τ} : σ ∈ sons(s), τ ∈ sons(t)}
end if
if sons(B_{s×t}) ≠ {∅} then
  for all B_{σ×τ} ∈ sons(B_{s×t}) do
    call SplitBlock(B_{σ×τ})
  end for
end if
```

References

Aliabadi, M.H., 1997a. Boundary element formulations in fracture mechanics. Applied Mechanics Reviews 50 (2), 83–96.

Aliabadi, M.H., 1997b. A new generation of boundary element methods in fracture mechanics. International Journal of Fracture 86 (2), 91–125.

Aliabadi, M.H., 2002. In: The Boundary Element Method: Applications in Solids and Structures, vol. 2. John Wiley & Sons Ltd., England.

Barnes, J., Hut, P., 1986. A hierarchical $O(N \ln N)$ force-calculation algorithm. Nature 324, 446–449.

Barra, L.P.S., Coutinho, A.L.G.A., Mansur, W.J., Telles, J.F.C., 1992. Iterative solution of BEM equations by GMRES algorithm. Computers & Structures 44 (6), 1249–1253.

Barra, L.P.S., Coutinho, A.L.G.A., Telles, J.F.C., Mansur, W.J., 1993. Multi-level hierarchical preconditioners for boundary element systems. Engineering Analysis with Boundary Elements 12, 103–109.

Bebendorf, M., 2000. Approximation of boundary element matrices. Numerische Mathematik 86, 565–589.

Bebendorf, M., 2001. Effiziente numerische Lösung von Randintegralgleichungen unter Verwendung von Niedrigrang-Matrizen. Ph.D. thesis, Universität Saarbrücken, 2000. dissertation.de, Verlag im Internet, ISBN 3-89825-183-7.

Bebendorf, M., 2005. Hierarchical LU decomposition-based preconditioners for BEM. Computing 74, 225–247.

Bebendorf, M., 2006. Approximate inverse preconditioning of FE systems for elliptic operators with non-smooth coefficients. SIAM Journal on Matrix Analysis and Applications 27 (4), 909–929.

Bebendorf, M., Grzhibovskis, R., 2006. Accelerating Galerkin BEM for linear elasticity using adaptive cross approximation. Mathematical Methods in the Applied Sciences 29, 1721–1747.

Bebendorf, M., Kriemann, R., 2005. Fast parallel solution of boundary integral equations and related problems. Computing and Visualization in Science 8, 121–135.

Bebendorf, M., Rjasanow, S., 2003. Adaptive low-rank approximation of collocation matrices. Computing 70, 1–24.

Benzi, M., 2002. Preconditioning techniques for large linear systems: a survey. Journal of Computational Physics 182, 418–477.

Börm, S., Grasedyck, L., Hackbusch, W., 2003. Introduction to hierarchical matrices with applications. Engineering Analysis with Boundary Elements 27, 405–422.

Cisilino, A.P., Aliabadi, M.H., 1997. Three-dimensional BEM analysis for fatigue crack growth in welded components. International Journal of Pressure Vessels and Piping 70, 135–144.

Cisilino, A.P., Aliabadi, M.H., 2004. Dual boundary element assessment of three-dimensional fatigue crack growth. Engineering Analysis with Boundary Elements 28, 1157–1173.

Fu, Y., Klimkowski, K.J., Rodin, G.J., Berger, E., Browne, J.C., Singer, J.K., Van de Geijn, R.A., Vemaganti, K.S., 1998. A fast solution method for three-dimensional many-particle problems of linear elasticity. International Journal for Numerical Methods in Engineering 42, 1215–1229.

Giebermann, K., 2001. Multilevel approximation of boundary integral operators. Computing 67, 183–207.

Goreinov, S.A., Tyrtyshnikov, E.E., Zamarashkin, N.L., 1997. A theory of pseudoskeleton approximations. Linear Algebra and its Applications 261, 1–21.

Grasedyck, L., 2005. Adaptive recompression of H-matrices for BEM. Computing 74, 205–223.

Grasedyck, L., Hackbusch, W., 2003. Construction and arithmetics of H-matrices. Computing 70, 295–334.

Greengard, L., Rokhlin, V., 1987. A fast algorithm for particle simulations. Journal of Computational Physics 73 (2), 325–348.

Guru Prasad, K., Kane, J.H., Keyes, D.E., Balakrishna, C., 1994. Preconditioned Krylov solvers for BEA. International Journal for Numerical Methods in Engineering 37, 1651–1672.

Hackbusch, W., 1999. A sparse matrix arithmetic based on H-matrices. Part I: introduction to H-matrices. Computing 62, 89–108.

Hackbusch, W., Khoromskij, B.N., 2000. A sparse H-matrix arithmetic. Part II: application to multidimensional problems. Computing 64, 21–47.

Hackbusch, W., Khoromskij, B.N., Kriemann, R., 2004. Hierarchical matrices based on weak admissibility criterion. Computing 73, 207–243.

Hackbusch, W., Nowak, Z.P., 1989. On the fast matrix multiplication in the boundary element method by panel clustering. Numerische Mathematik 73, 207–243.

Kurz, S., Rain, O., Rjasanow, S., 2002. The adaptive cross approximation technique for the 3-D boundary element method. IEEE Transactions on Magnetics 38 (2), 421–424.

Leung, C.Y., Walker, S.P., 1997. Iterative solution of large three-dimensional BEM elastostatic analyses using the GMRES technique. International Journal for Numerical Methods in Engineering 40, 2227–2236.

Mansur, W.J., Araujo, F.C., Malaghini, E.B., 1992. Solution of BEM systems of equations via iterative techniques. International Journal for Numerical Methods in Engineering 33, 1823–1841.

Merkel, M., Bulgakov, V., Bialecki, R., Kuhn, G., 1992. Iterative solution of large-scale 3D-BEM industrial problems. Engineering Analysis with Boundary Elements 22, 183–197.

Mi, Y., Aliabadi, M.H., 1992. Dual boundary element method for three-dimensional fracture mechanics analysis. Engineering Analysis with Boundary Elements 10, 161–171.

Mi, Y., Aliabadi, M.H., 1994. Three-dimensional crack growth simulation using BEM. Computers & Structures 52 (5), 871–878.

Mullen, R.L., Rencis, J.J., 1987. Iterative methods for solving boundary element equations. Computers & Structures 25 (5), 713–723.

Nishimura, N., Yoshida, K.I., Kobayashi, S., 1999. A fast multipole boundary integral equation method for crack problems in 3D. Engineering Analysis with Boundary Elements 23, 97–105.

Ostrowski, J., Andjelíc, K.I., Bebendorf, M., Crangânu-Creţu, B., Smajić, J., 2006. Fast BEM-solution of Laplace problems with H-matrices and ACA. IEEE Transactions on Magnetics 42 (4), 627–630.

Portela, A., Aliabadi, M.H., Rooke, D.P., 1992. The dual boundary element method: effective implementation for crack problems. International Journal for Numerical Methods in Engineering 33, 1269–1287.

Portela, A., Aliabadi, M.H., Rooke, D.P., 1993. Dual boundary element incremental analysis of crack propagation. Computers & Structures 46 (2), 237–247.

Popov, V., Power, H., 2001. A O(N) Taylor series multipole boundary element method for three-dimensional elasticity problems. Engineering Analysis with Boundary Elements 25, 7–18.

Rokhlin, H., 1985. Rapid solution of integral equations of classical potential theory. Journal of Computational Physics 60 (2), 187–207.

Saad, Y., Schultz, M.H., 1986. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. SIAM Journal on Scientific and Statistical Computing 7 (3), 856–869.

Saad, Y., van der Vorst, H.A., 2000. Iterative solution of linear systems in the 20th century. Journal of Computational and Applied Mathematics. 123, 1–33.

Tyrtyshnikov, E.E., 1996. Mosaic-skeleton approximations. Calcolo 33, 47–57.

Tyrtyshnikov, E.E., 2000. Incomplete cross approximation in the mosaic-skeleton method. Computing 64, 367–380.

Urekew, T.J., Rencis, J.J., 1993. The importance of diagonal dominance in the iterative solution of equations generated from the boundary element method. International Journal for Numerical Methods in Engineering 36, 3509–3527.

Valente, F.P., Pina, H.L.G., 1998. Iterative solvers for BEM algebraic systems of equations. Engineering Analysis with Boundary Elements 22, 117–124.

Wang, H., Yao, Z., Wang, P., 2005. On the preconditioners for fast multipole boundary element methods for 2D multi-domain alastostatics. Engineering Analysis with Boundary Elements 29, 673–688.

Wrobel, L.C., 2002. In: The Boundary Element Method: Applications in Thermo-Fluids and Acoustics, vol. 1. John Wiley & Sons Ltd., England.

Zhao, K., Vouvakis, M.N., Lee, J.F., 2005. The adaptive cross approximation algorithm for accelerated method of moments computation of EMC problems. IEEE Transaction on Electromagnetic Compatibility 47, 763–773.