Chapter 1

Fast Hierarchical Boundary Element Method for Large Scale 3D Elastic Problems

I. Benedetti, A. Milazzo

Dipartimento di Ingegneria Strutturale Aerospaziale e Geotecnica,
Università degli Studi di Palermo,
Edificio 8, Viale delle Scienze, 90128, Palermo, Italy.
e-mails: i.benedetti@unipa.it, alberto.milazzo@unipa.it

M.H. Aliabadi

Department of Aeronautics, Imperial College London,
South Kensington Campus, London, SW7 2AZ, United Kingdom.
e-mail: m.h.aliabadi@imperial.ac.uk

This chapter reviews recent developments in the strategies for the fast solution of boundary element systems of equations for large scale 3D elastic problems. Both isotropic and anisotropic materials as well as cracked and uncracked solids are considered. The focus is on the combined use the hierarchical representation of the boundary element collocation matrix and iterative solution procedures. The hierarchical representation of the collocation matrix is built starting from the generation of the cluster and block trees that take into account the nature of the considered problem, i.e. the possible presence of a crack. Low rank blocks are generated through adaptive cross approximation (ACA) algorithms and the final hierarchical matrix is further coarsened through suitable procedures also used for the generation of a *coarse* preconditioner, which is built taking full advantage of the hierarchical format. The final system is solved using a GMRES iterative solver. Applications show that the technique allows considerable savings in terms of storage memory, assembly time and solution time without accuracy penalties. Such features make the method appealing for large scale applications.

1.1. 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.[1,2] 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 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[3] 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[4] 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,[5] 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[3] and Greengard and Rokhlin[5] 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.*[6] used FMMs in connection with a generalized minimum residual method (GMRES)[7] to solve 3D crack problems for the Laplace equation. Fast algorithms have also been used for the treatment of elastic problems. Fu *et al.*[8] 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[9] 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.[10]

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.[11–13] Of particular interest was the observation, due to Tyrtyshnikov,[11] that low rank approximations could be built from only few entries of the original block. Successively Bebendorf[14] 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[15] and was referred to as adaptive cross approximation (ACA). Such an algorithm allows a rela-

tively 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.[16,17] 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.[18,19] 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.[20,21]

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[22] had not shown good results, following works[23,24] 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.*[25] tested the performance of the GMRES algorithm, developed by Saad and Schultz,[7] 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. Guru Prasad *et al.*[26] 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 Urekev and Rencis,[27] while Merkel *et*

al.[28] 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,[29] 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.*[30] proposed a strategy for the construction of preconditioners for GMRES solved BEM problems, while Wang *et al.*[31] 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,[32] 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,[33] 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 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,[15] interesting applications to various electromagnetic problems have been proposed by Kurz *et al.*,[34] Zhao *et al.*[35] and Ostrowski *et al.*[36] Bebendorf and Grzhibovskis[37] have recently extended the use of ACA to the analysis of elastic problems through Galerkin BEM while Benedetti *et al.*[38] extended the use of hierarchical matrices to the treatment of 3D elastic *crack* problems through Dual Boundary Element Method. Some preliminary results about anisotropic elastic problems without crack have been presented by Benedetti *et al.*[39]

In this chapter the development and the use of the fast Hierarchical Dual Boundary Element Method for the analysis of three-dimensional elasticity crack problems is described. First the basic formulation of the DBEM for 3D fracture mechanics problems is briefly reviewed and the features of DBEM matrices are discussed. The main features of anisotropic kernels

and some strategies for their convenient use are illustrated. Next the main steps for building the hierarchical representation of the solution matrix are discussed and some algorithms are reported. Some considerations about the application of the hierarchical format to boundary element formulations of 3D crack problems are pointed out. Some applications to both isotropic and anisotropic problems complete the work and demonstrate the capability of the method.

1.2. The 3D Dual Boundary Element Method

The Dual Boundary Element Method is a general and efficient technique for modeling both two-dimensional[40,41] and three-dimensional[42,43] isotropic and anisotropic[44–48] crack problems in the framework of the BEMs.[49,50] The Dual Boundary Element 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

$$c_{ij}(\boldsymbol{x}_0)u_j(\boldsymbol{x}_0) + \fint_\Gamma T_{ij}(\boldsymbol{x}_0,\boldsymbol{x})u_j(\boldsymbol{x})\,d\Gamma = \int_\Gamma U_{ij}(\boldsymbol{x}_0,\boldsymbol{x})t_j(\boldsymbol{x})\,d\Gamma \qquad (1.1)$$

where U_{ij} and T_{ij} represent the 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 $\fint$ stands for Cauchy principal value integral, whose presence is consequence of the $O(r^{-2})$ strength of the T_{ij} integrands.

The displacement equation (1.1) 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

$$c_{ij}(\boldsymbol{x}_0^-)u_j(\boldsymbol{x}_0^-) + c_{ij}(\boldsymbol{x}_0^+)u_j(\boldsymbol{x}_0^+) + \fint_\Gamma T_{ij}(\boldsymbol{x}_0^-,\boldsymbol{x})u_j(\boldsymbol{x})\,d\Gamma = \int_\Gamma U_{ij}(\boldsymbol{x}_0^-,\boldsymbol{x})t_j(\boldsymbol{x})\,d\Gamma \qquad (1.2)$$

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

$$c_{ij}(\boldsymbol{x}_0^+)t_j(\boldsymbol{x}_0^+) - c_{ij}(\boldsymbol{x}_0^-)t_j(\boldsymbol{x}_0^-) + n_j(\boldsymbol{x}_0^+) \oint\!\!\!\!\!\!= \int_\Gamma T_{ijk}(\boldsymbol{x}_0^+, \boldsymbol{x})u_k(\boldsymbol{x})\, d\Gamma = n_j(\boldsymbol{x}_0^+) \oint\!\!\!\!\!\!= \int_\Gamma U_{ijk}(\boldsymbol{x}_0^+, \boldsymbol{x})t_k(\boldsymbol{x})\, d\Gamma \quad (1.3)$$

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 $\oint\!\!\!\!\!\!=$ stands for Hadamard principal value integral, originating from the presence of the $O(r^-3)$ kernels T_{ijk}.
Equations (1.1), (1.2) and (1.3) provide the boundary integral model for the analysis of general crack problems.

1.2.1. *Fundamental solutions*

For isotropic problems the fundamental solution kernels U_{ij}, T_{ij}, U_{ijk} and T_{ijk} are given by the Kelvin displacements and tractions and a combination of their derivatives as follows

$$U_{ij} = \frac{1}{16\,\pi(1-\nu)\mu r}[(3-4\nu)\delta_{ij} + \frac{\partial r}{\partial x_i}\frac{\partial r}{\partial x_j}] \quad (1.4)$$

$$T_{ij} = -\frac{1}{8\,\pi(1-\nu)r^2}\{[(1-2\nu)\delta_{ij} + 3\frac{\partial r}{\partial x_i}\frac{\partial r}{\partial x_j}]\frac{\partial r}{\partial n} + - (1-2\nu)(\frac{\partial r}{\partial x_i}n_j - \frac{\partial r}{\partial x_j}n_i)\} \quad (1.5)$$

$$U_{ijk} = \frac{1}{8\,\pi(1-\nu)r^2}[(1-2\nu)(\delta_{ij}\frac{\partial r}{\partial x_k} + \delta_{ik}\frac{\partial r}{\partial x_j} - \delta_{jk}\frac{\partial r}{\partial x_i}) + + 3\frac{\partial r}{\partial x_i}\frac{\partial r}{\partial x_j}\frac{\partial r}{\partial x_k}] \quad (1.6)$$

$$T_{ijk} = \frac{\mu}{4\,\pi(1-\nu)r^3}\{3\frac{\partial r}{\partial n}[(1-2\nu)\delta_{ij}\frac{\partial r}{\partial x_k} + \nu(\delta_{ik}\frac{\partial r}{\partial x_j} + \delta_{jk}\frac{\partial r}{\partial x_i}) + -5\frac{\partial r}{\partial x_i}\frac{\partial r}{\partial x_j}\frac{\partial r}{\partial x_k}] + 3\nu(\frac{\partial r}{\partial n_i}\frac{\partial r}{\partial x_j}\frac{\partial r}{\partial x_k} + \frac{\partial r}{\partial n_j}\frac{\partial r}{\partial x_i}\frac{\partial r}{\partial x_k}) + +(1-2\nu)(3\frac{\partial r}{\partial n_k}\frac{\partial r}{\partial x_i}\frac{\partial r}{\partial x_j} + n_j\delta_{ik} + n_i\delta_{jk}) - (1-4\nu)n_k\delta_{ij}\} \quad (1.7)$$

For the general 3D anisotropic elasticity problem no closed form fundamental solutions is available and fundamental solution kernels U_{ij}, T_{ij}, U_{ijk}

and T_{ijk} are numerically obtained from the Green's function for general anisotropic domains.[51] For the special case of transversely isotropic materials the fundamental solution kernels can be expressed in analytic form as reported by Pan and Chou[52] and Ariza and Dominguez.[47]

Different techniques for such numerical evaluation have been presented by various authors. Vogel and Rizzo[53] derived and applied the integral representation of the static fundamental solution for a general anisotropic three-dimensional continuum. Successively Wilson and Cruse[54] proposed an efficient numerical implementation to speed up the BEM solution of 3D anisotropic problems. Since these pioneering works much research has been focused on the derivation of fundamental solutions for three-dimensional anisotropic elasticity. Indeed, the lack of closed form fundamental solutions has hindered the use of the method in anisotropic applications. BEM approaches for general anisotropic problems have been developed using either the integral expression of the fundamental solution[51,53–55] or its explicit approximate expressions.[56–60] While the former are computationally expensive, the latter involve tedious calculations of the kernels' derivatives. For such reasons the BEM for the analysis of three-dimensional anisotropic structures has resulted quite slow and difficult to implement with respect to isotropic formulations.

The technique used in this chapter is based on the scheme proposed by Wilson and Cruse[54] and is briefly reviewed in the following.[51] Defined a unit vector $\mathbf{b}$ in the direction connecting the collocation point $\boldsymbol{x}_0$ with the integration point $\boldsymbol{x}$, the anisotropic Green's function can be written

$$G_{ij} = \frac{1}{8\pi^2 r} \int_0^{2\pi} M_{ij}^{-1}(\varphi) d\varphi \tag{1.8}$$

where r is the distance between the observed and source points and the integration is performed on the plane perpendicular to $\mathbf{b}$. The integrand function M_{ij}^{-1} are the components of the inverse of the matrix defined by

$$M_{ij} = C_{ipjq} z_p z_q \tag{1.9}$$

where C_{ipjq} are the components of the stiffness tensor and z_i are the components of a unit vector representative of the vectors that lye on the integration plane (perpendicular to $\mathbf{b}$). Such components are expressed in terms of the anomaly φ. It is worth noting that the Green function comprises a singular part depending only on r and a finite part which depends on the material properties and the direction $\boldsymbol{x}_0\boldsymbol{x}$.

To evaluate the BIE kernels of DBEM also the derivatives of the Green's

function are needed. They are given by

$$\frac{\partial G_{ij}}{\partial x_p} = -\frac{1}{4\pi^2 r^2}\int_0^\pi (-b_p M_{ij}^{-1} + z_p F_{ij}) d\varphi \tag{1.10}$$

$$\frac{\partial^2 G_{ij}}{\partial x_p \partial x_q} = \frac{1}{4\pi^2 r^2}\int_0^\pi [2b_p b_q M_{ij}^{-1} - 2(z_p b_q + z_q b_q) F_{ij} + z_p z_q A_{ij}] d\varphi \tag{1.11}$$

where once again the integration is performed on a plane perpendicular to **b** and

$$F_{ij} = C_{mnpq} M_{im}^{-1} M_{pj}^{-1} (z_n b_q + z_q b_n) \tag{1.12}$$

$$A_{ij} = C_{mnpq}[(z_n b_q + z_q b_n)(F_{im} M_{pj}^{-1} + M_{im}^{-1} F_{pj}) - 2M_{im}^{-1} M_{pj}^{-1} b_n b_q] \tag{1.13}$$

Also for the Green's function derivatives the same considerations as those for the Green function hold. In particular such functions are comprised of a singular part and a finite part depending on material properties.
Once the general anisotropic Green function and its derivatives have been introduced, the DBEM kernels are given by

$$U_{ij} = G_{ij} \tag{1.14}$$

$$T_{ij} = C_{kjmn} \frac{\partial G_{mi}}{\partial x_n} n_k \tag{1.15}$$

$$U_{ijk} = -C_{ijpq}\left(\frac{\partial G_{pk}}{\partial x_q} + \frac{\partial G_{qk}}{\partial x_p}\right) \tag{1.16}$$

$$T_{ijk} = -C_{ijpq} C_{lkmn}\left(\frac{\partial^2 G_{mp}}{\partial x_q \partial x_n} + \frac{\partial^2 G_{mq}}{\partial x_p \partial x_n}\right) n_l \tag{1.17}$$

1.2.2. *Numerical scheme*

The discrete boundary element model is built starting from the eqs. (1.1-1.3) and sub-dividing 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.[50]
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[42,43] 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. *Semi-discontinuous* eight-noded elements are used to model those portions of the external boundary intersecting the crack surfaces, when the crack reaches the external boundary. Further information on slightly different discretization procedures can be found in the works of Cisilino and Aliabadi.[61,62]

The isotropic kernels are integrated following the procedures used in the aforementioned references. The anisotropic kernels are treated using the following scheme. On each isoparametric element the integrals involved in the boundary integral equation can be written

$$\int_{-1}^{1}\int_{-1}^{1} F[\boldsymbol{x}_0(\xi,\eta),\boldsymbol{x}(\xi,\eta)]\phi_k(\xi,\eta)J(\xi,\eta)d\xi d\eta \tag{1.18}$$

where ξ and η are the local coordinates of the element, $J(\xi,\eta)$ is the Jacobian, $\phi_k(\xi,\eta)$ is the $k-th$ shape function and F is representative of the fundamental solution kernel. The integral is computed by using standard Gauss quadrature for regular elements and accounting for suitable integration schemes to improve the accuracy of nearly singular integrals.[50] Strongly and hyper singular kernels must be computed as finite part integrals (Cauchy and Hadamard principal values).

A polar coordinate transformation with the origin at the source point is introduced by writing

$$\begin{aligned}\xi &= \xi_c + \varrho\, cos(\vartheta)\\ \eta &= \eta_c + \varrho\, sin(\vartheta)\end{aligned} \tag{1.19}$$

and the integral (1.18) becomes

$$\int_{-1}^{1}\int_{-1}^{1} F[\boldsymbol{x}_0(\xi_c,\eta_c),\boldsymbol{x}(\varrho,\vartheta)]\phi_k(\varrho,\vartheta)J(\varrho,\vartheta)\varrho\, d\varrho d\vartheta \tag{1.20}$$

Recalling the form of the fundamental solution kernel singularity, the Kutt's numerical quadrature[63] can be employed to compute the inner finite part integral with respect to ϱ. For a given value of ϑ, such inner part can be approximated by the Kutt's n-point equispace quadrature giving

$$\int_0^R \frac{f(\varrho)}{\varrho^m}d\varrho \approx \frac{1}{R}\sum_{i=1}^{N}(w_i + c_i\, log\, R)f(\frac{i-1}{N}R) \tag{1.21}$$

where $m=1$ for strongly singular integrals and $m=2$ for hypersingular integrals. w_i and c_i are the approximate weights given by Kutt.[64,65] Once

the finite part has been computed the outer integral in eq. (1.20) is regular and is computed through standard Gaussian quadrature.
The computation of the integrals involved in the BIE requires the evaluation of the fundamental solution kernels at the integration points. The form of the general anisotropic Green's function and its derivatives suggests an efficient method to speed up their computation.[51,54] The modulation functions pertaining to the Green's function and its derivatives are regular and can be easily computed through standard schemes. They can be calculated in advance for different directions of the unit vector $\mathbf{b}$ and the results can be stored in a database. When a particular couple of source and observed points are considered, the direction of the corresponding unit vector $\mathbf{b}$ is determined and the related value of the modulation functions can be computed by interpolation from the stored database. The Green function and its derivatives are then given by the product of the modulation functions by the inverse powers of the source/observed point distance.

1.2.3. *DBEM systems of equations*

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

$$\begin{bmatrix} \mathbf{H}_{bb} & \mathbf{H}_{bd} & \mathbf{H}_{bt} \\ \mathbf{H}_{db} & \mathbf{H}_{dd} & \mathbf{H}_{dt} \\ \mathbf{S}_{tb} & \mathbf{S}_{td} & \mathbf{S}_{tt} \end{bmatrix} \begin{bmatrix} \boldsymbol{u}_b \\ \boldsymbol{u}_d \\ \boldsymbol{u}_t \end{bmatrix} = \begin{bmatrix} \mathbf{G}_{bb} & \mathbf{G}_{bd} & \mathbf{G}_{bt} \\ \mathbf{G}_{db} & \mathbf{G}_{dd} & \mathbf{G}_{dt} \\ \mathbf{D}_{tb} & \mathbf{D}_{td} & \mathbf{D}_{tt} \end{bmatrix} \begin{bmatrix} \boldsymbol{t}_b \\ \boldsymbol{t}_d \\ \boldsymbol{t}_t \end{bmatrix} \tag{1.22}$$

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

$$\begin{bmatrix} \mathbf{A}_{bb} & \mathbf{H}_{bd} & \mathbf{H}_{bt} \\ \mathbf{A}_{db} & \mathbf{H}_{dd} & \mathbf{H}_{dt} \\ \mathbf{B}_{tb} & \mathbf{S}_{td} & \mathbf{S}_{tt} \end{bmatrix} \begin{bmatrix} \boldsymbol{X}_b \\ \boldsymbol{u}_d \\ \boldsymbol{u}_t \end{bmatrix} = \begin{bmatrix} \boldsymbol{Y}_b \\ \boldsymbol{Y}_d \\ \boldsymbol{Y}_t \end{bmatrix} \tag{1.23}$$

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. (1.23) 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.[32]

The application of hierarchial matrices in conjunction with iterative solvers to DBEM matrices of the form given in eq.(1.23) is discussed in the next section.

1.3. Hierarchical Dual Boundary Element Method

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.[11,12,14,15] 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,[14] Bebendorf and Rjasanow[15] and especially to the paper of Bebendorf and Grzhibovskis,[37] 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. 1.1.

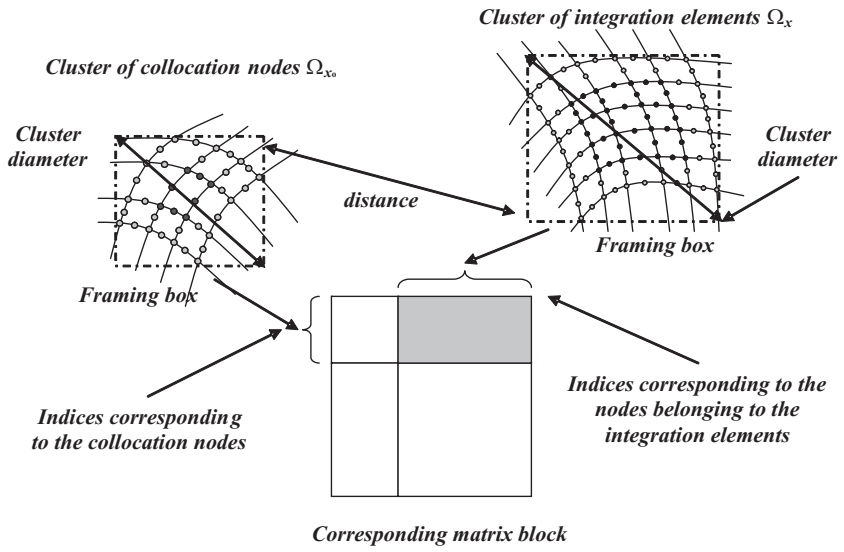

Fig. 1.1. Schematic of the boundary subdivision process.

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.
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),[66] that allows a further storage reduction without accuracy penalties. Moreover, since the initial matrix partition is generally not optimal,[67] once the blocks have been generated and recompressed, the entire structure of the hierarchical block tree can be modified by a suitable *coarsening* procedure.[18] 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 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.[20] For a direct solution, the computation of the hierarchical LU decomposition of the collocation matrix is needed to carry out an effective hierarchical inversion.[19] 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.[19]
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.

1.3.1. *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 value.
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. 1.2.

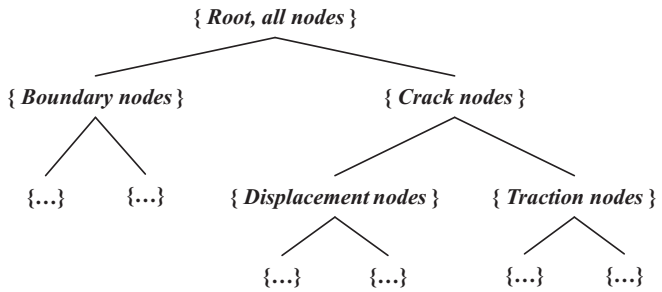

Fig. 1.2. Index sets induced by presence of cracks.

The algorithm used for the construction of the cluster tree is also reported (Algorithm 1.1). It is similar to the algorithm introduced by Giebermann[68] and used in Grasedyck[18] 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 $i-th$ coordinate of the $j-th$ collocation point, while the operator $\#(\cdot)$ gives the number of elements in a set.

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

Algorithm 1.1 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 $\#s \leq n_{min}$ **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 $s_1 = \{i \in s : (\mathbf{x}_i)_j \leq (M_j + m_j)/2\}$
  define $s_2 = s - s_1$
end if
set $sons(s) = \{s_1, s_2\}$
for i=1,2 **do**
  call SplitCluster$(s_i, \mathbf{x}, n_{min})$
end for

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

$$min(diam\ \Omega_{x_0}, diam\ \Omega_x) \leq \eta \cdot dist(\Omega_{x_0}, \Omega_x) \tag{1.24}$$

where $\eta > 0$ is a parameter influencing the number of admissible blocks on one hand and the convergence speed of the adaptive approximation of low rank blocks on the other hand.[21]
Since the actual diameters and the distance between two clusters are gen-

erally time consuming to be exactly computed, the condition is usually assessed with respect to bounding boxes parallel to the reference axes,[18,68] as already mentioned commenting on Fig. 1.1. In this case Ω_{x_0} and Ω_x in eq. (1.24) 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 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*[69] should significantly improve the quality of the initial partition, but such scheme is not tested in this work.
The algorithm is graphically illustrated in Fig. 1.3. Starting from the root (the entire matrix), each block is subdivided into four sub-blocks until either the admissibility condition is satisfied 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.

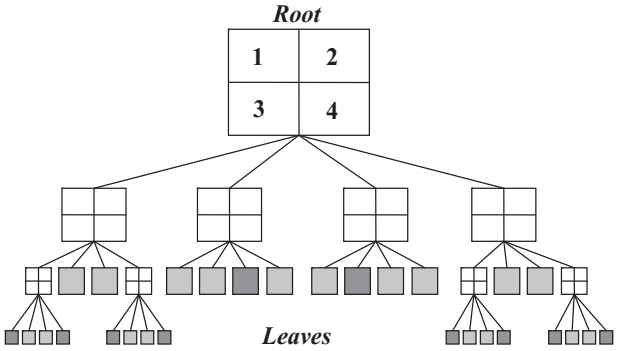

Fig. 1.3. Graphical illustration of the block splitting procedure.

The presence of cracks requires some extra considerations. As illustrated in Fig. 1.2, 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 (1.24), 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.(1.23), if the condition (1.24) 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 (1.24) 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. 1.4. 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.

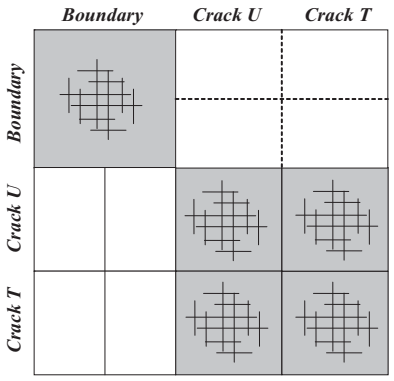

Fig. 1.4. Structure of the hierarchical matrix including crack surfaces.

Finally, it is important to consider specifically the assessment of the condition (1.24) 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 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 here reported (Algorithm 1.3.2). It takes into account the possible presence of cracks, following the considerations developed above. 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 1.2 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 $sons(B_{s\times t}) = \{\emptyset\}$
else
  set $sons(B_{s\times t}) = \{B_{\sigma\times\tau} : \sigma \in sons(s), \tau \in sons(t)\}$
end if
if $sons(B_{s\times t}) \neq \{\emptyset\}$ **then**
  for all $B_{\sigma\times\tau} \in sons(B_{s\times t})$ **do**
    call SplitBlock($B_{\sigma\times\tau}$)
  end for
end if

1.3.3. *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

$$M \simeq M_k = A \cdot B^T = \sum_{i=1}^{k} a_i \cdot b_i^T \tag{1.25}$$

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 $\|M - M_k\|_F \leq \varepsilon \|M\|_F$, 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 $i-th$ 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.[20]

Different ACA algorithms can be used to generate the approximate blocks. The original algorithm was proposed by Bebendorf[14] and was further developed by Bebendorf and Rjasanow.[15] Grasedyck[18] proposed the so called ACA+ algorithm and compared its performances to those of the standard scheme. Bebendorf and Grzhibovskis[37] 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.*[34] and Bebendorf and Kriemann,[70] while the reader is referred to the work of Grasedyck[18] for ACA+. Algorithm 1.3 describes the partially pivoted ACA scheme. In this 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. (1.25). Different schemes often differ for the choice of the pivots, that can have a noticeable effect on the convergence and quality of the approximation.[70] 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

Algorithm 1.3 Partially pivoted ACA for $B_{\sigma\times\tau}$

$k = 1$; $Z = \{\emptyset\}$
repeat
  if k=1 **then**
    i_k =min$(\sigma - Z)$
  else
    i_k =argmax$_{i\in\sigma-Z}(|(a_{k-1})_i|)$
  end if
  $Z = Z \cup \{i_k\}$
  $\overline{\mathbf{b}}_k = \mathbf{M}(i_k, :) - \sum_{r=1}^{k-1}(\mathbf{a}_r)_{i_k}\mathbf{b}_r^T$
  if $\overline{\mathbf{b}}_k \neq \mathbf{0}$ **then**
    j_k =argmax$_{j\in\sigma}(|(\overline{\mathbf{b}}_k)_j|)$; $\mathbf{b}_k = (\overline{\mathbf{b}}_k)_{j_k}^{-1}\overline{\mathbf{b}}_k$
    $\mathbf{a}_k = \mathbf{M}(:, j_k) - \sum_{r=1}^{k-1}(\mathbf{b}_r)_{j_k}\mathbf{a}_r$
    $k = k + 1$
  end if
until *The stop criterion is satisfied or* $Z = \sigma$

approximating block in terms of the Frobenius norm.[14,34] 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

$$\|M_k\|_F^2 = \|M_{k-1}\|_F^2 + 2\sum_{i=1}^{k-1}(a_k^T a_i)(b_i^T b_k) + \|a_k\|_F^2\|b_k\|_F^2 \tag{1.26}$$

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

$$\|a_k\|_F\|b_k\|_F \leq \varepsilon\|M_k\|_F \tag{1.27}$$

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 equations (1.23), 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 $\mathbf{H}$ and $\mathbf{G}$ 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 $\mathbf{H}$ columns and few $\mathbf{G}$ columns and that few $\mathbf{G}$ columns contribute to the right hand side, see Fig. 1.5.

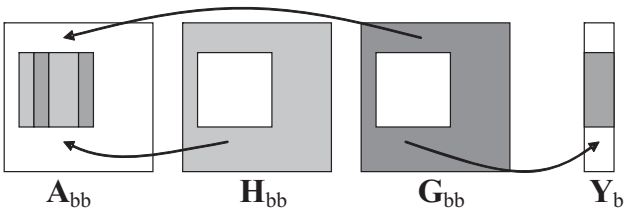

Fig. 1.5. Construction of an A block.

In this case, it may be not convenient to generate the approximation of the entire $\mathbf{G}$ block (the white block belonging to $\mathbf{G}_{bb}$ in Fig. 1.5); on the contrary it is more effective to generate the approximation of the entire $\mathbf{H}$ block through ACA, then to annihilate the terms in the rows b_i^T corresponding to the columns to be replaced with the $\mathbf{G}$ columns, and eventually to compute exactly the few needed $\mathbf{G}$ columns and add them to the representation (1.25), using row vectors filled with zeros and ones placed in the positions corresponding to the columns to be replaced; the others $\mathbf{G}$ 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 $\mathbf{H}$ and $\mathbf{G}$ that are actually needed for the construction of a specific $\mathbf{A}$ 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 can fail, as shown by Grasedyck[18] 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*[13,70] 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.

1.3.4. *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,[18,66] 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.
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*[20,21] 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.[66]
Let $M_k = A \cdot B^T$ be a low rank block. The reduced decomposition can be computed by

$$M_k = A \cdot B^T = Q_A R_A R_B^T Q_B^T = Q_A U \Sigma V^T Q_B^T \tag{1.28}$$

where $Q_X R_X$ is the QR decomposition of the matrix X, while $U\Sigma V^T$ represents the SVD of $R_A R_B^T$. If $A \in \mathbb{R}_{m\times k}$ then $Q_A \in \mathbb{R}_{m\times k}$ while $R_A \in \mathbb{R}_{k\times k}$. The SVD of the reduced matrix $R_A R_B^T \in \mathbb{R}_{k\times k}$ can then be computed by using `dgesvd.f`.
In eq. (1.28) there is still no approximation. If the singular values $\sigma_i < \varepsilon\sigma_1$ contained in the diagonal of Σ are discarded then the following approximation is obtained

$$M_k \simeq Q_A \widetilde{U}\widetilde{\Sigma}\widetilde{V}^T Q_B^T = \widetilde{A}\widetilde{B}^T \tag{1.29}$$

with $\widetilde{A} = Q_A\widetilde{U}\widetilde{\Sigma}$ and $\widetilde{B} = \widetilde{V}^T Q_B^T$. In eq. (1.29) $\widetilde{U}, \widetilde{V} \in \mathbb{R}_{k\times\widetilde{k}}$ are obtained from the original matrices discarding the last $k - \widetilde{k}$ columns, while $\widetilde{\Sigma} \in \mathbb{R}_{\widetilde{k}\times\widetilde{k}}$ is the diagonal matrix $diag\{\sigma_1, \sigma_2, \dots, \sigma_{\widetilde{k}}\}$. An analogous procedure is followed for the full rank leaves that, after discarding the smallest singular values, are stored in low rank format if the total size in low rank is less than the size requested in full rank representation. Note that the scheme can be applied immediately after a block has been generated.
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[18]and is reported in the Algorithm 1.4. Note that the unification of the four low rank sons of a block is performed by expanding their columns and rows to the parent's block dimensions by suitably placed zeros and then adding such expanded vectors to provide the parent's low rank representation.
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,[19] based on the computation of a *coarse* approximation of the collocation matrix, as described in the following.

Algorithm 1.4 Recursive CoarsenTree($B_{s\times t}$)

```
call CoarsenTree(B_{σ×τ}) for all B_{σ×τ} ∈ Sons(B_{s×t})
if (Sons(B_{s×t}) are all low rank leaves) then
  Unify the four leaves;
  Compute the SVD of the unification;
  Check the storage requirements of the SVD;
  if (storage of the SVD < storage of the four sub-blocks) then
    Store the SVD and set Sons(s × t) = {∅}
  end if
end if
```

1.3.5. *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[16] and Grasedyck and Hackbusch,[20] 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.*[21] The hierarchical LU decomposition, based on the previous arithmetic, is discussed by Bebendorf.[19] The interested reader is referred to the mentioned works for analytic details. In the following the basic algorithms are illustrated for the sake of completeness.

1.3.5.1. *Addition*

The addition between two trees with the same structure is performed by adding the leaves of the two trees in full or low rank format. Such an operation preserves the structure of the two added matrices and the result is then a hierarchical matrix with the same structure. The addition between the two trees (or branches) H_1 and H_2 is denoted sometimes by $H = H_1 \oplus H_2$.
The addition of two full rank blocks is performed by simply adding the two blocks in full matrix format. On the other hand, the addition of two low rank leaves, is given by

$$M_1 + M_2 = A_1 \cdot B_1^T + A_2 \cdot B_2^T = [A_1, A_2] \cdot [B_1, B_2]^T \tag{1.30}$$

As it is evident, the block resulting from the exact addition in low rank format has a higher rank with respect to the two added blocks. To maintain the advantages given by the low rank representation a SVD truncation to the set accuracy is eventually performed on the resulting block. The importance of this truncation will appear more clearly when the multiplication is discussed.

1.3.5.2. *Multiplication*

The hierarchical multiplication is performed between dimensionally compatible hierarchical matrices or blocks with the same or different structure. According to the literature on the subject,[20,21] the multiplication between the two trees (or sub-trees) H_1 and H_2 is denoted sometimes by $H = H_1 \odot H_2$. It is worth saying that the hierarchical multiplication and its implementation are noticeably more involved than the hierarchical addition. Such complexity is a consequence of the many different cases that must be taken into account. Let A and B be the two dimensionally compatible blocks to be multiplied. Besides the two factors, also the so called *target* matrix T and its structure must be taken into account. Three main cases are possible:

(i) : Both the two factors and the target matrix are subdivided. In this case the following recursion is invoked

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} A_{11}B_{11}+A_{12}B_{21} & A_{11}B_{12}+A_{12}B_{22} \\ A_{21}B_{11}+A_{22}B_{21} & A_{21}B_{12}+A_{22}B_{22} \end{pmatrix} \tag{1.31}$$

where also the addition in low rank format is called (note that all the additions and multiplications are intended in hierarchical format, although we used the standard operation symbols instead of $\oplus$ and $\odot$ for brevity). From these equations it is clear the importance of defining a target matrix. The entire operation is performed by multiplying blocks and adding subsequently the result to the right target block. It is to be noted that the compatibility between the dimensions of the two factors to be multiplied and those of the target matrix plays a crucial role and must be then taken carefully into account when implementing the relative algorithms. Moreover, it becomes evident as also the structure of the target block itself, i.e. if the target block is a leaf or is subdivided, is of primary importance in controlling the flow of the operation, as emerges in the following two cases;

(ii) : At least one of the two factors is a leaf, i.e. is not subdivided. This case has several sub-cases. It could either happen that only one of the factors or both of them are leaves. Moreover the leaves can be low or full rank. Each of these circumstances deserves a separate implementation, but basically all these cases are dealt with by considering that if a leaf is full rank, its dimensions are bounded by n_{min}, while if it is low rank k itself is low. It is then convenient to compute the product block by taking into account that

$$M_1 M_2 = A_1 B_1^T M_2 = A_1 \widetilde{B}_1{}^T \tag{1.32}$$

$$M_1 M_2 = M_1 A_2 B_2^T = \widetilde{A}_2 B_2{}^T \tag{1.33}$$

$$M_1 M_2 = A_1 B_1^T A_2 B_2^T = A_1 \widetilde{B}_1{}^T = \widetilde{A}_2 B_2{}^T \tag{1.34}$$

which represent the multiplication between a low rank and a full rank leaf and between two low rank leaves; the result block must be then split suitably and added to the target matrix. It is worthwhile remembering that every addition in low rank format must be followed by a truncation needed to maintain low the rank of the resulting block. Skipping this passage could lead to an explosion of the average rank of the product matrix that would destroy the advantages of the low rank representation.

(iii) : The target block is a leaf and the two factors are subdivided. In this case the multiplication is performed in the sub-blocks, the result blocks

are then expanded to the dimensions of the target, truncated and added to it. After every addition the truncation is newly performed.

The classification of the many different cases and the respective action is graphically illustrated in Fig. 1.6. It is to be noted that when a block is not subdivisible, it can be low or full rank.

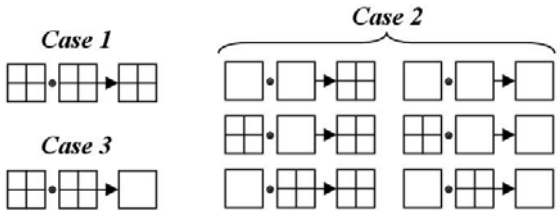

Fig. 1.6. Different cases occurring in hierarchical multiplication.

1.3.5.3. *LU decomposition*

The hierarchical LU decomposition is based on the previous developed arithmetics.[19] If a block H is divisible then its decomposition can be written as

$$\begin{pmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix} \tag{1.35}$$

that turns into the following subproblems

$$L_{11}U_{11} = H_{11} \tag{1.36}$$
$$L_{11}U_{12} = H_{12} \tag{1.37}$$
$$L_{21}U_{11} = H_{21} \tag{1.38}$$
$$L_{22}U_{22} = H_{22} - L_{21}U_{12} \tag{1.39}$$

The first equation, that represents the LU decomposition of the first subblock, can be tackled by the `LAPACK` subroutine `dgetrf.f` if the block is not further divisible. Otherwise the scheme reapplies recursively. Once L_{11} and U_{11} have been computed, the second equation gives U_{12} while the third allows the calculation of L_{21}. In both cases it is needed to distinguish between different cases and apply different suitable schemes. The fourth equation can be tackled once the previous equations have been solved.

1.3.5.4. *Inversion*

Once the basic hierarchical arithmetics, i.e. the addition and the multiplication in hierarchical format, has been implemented the algorithm for the hierarchical inversion is relatively straightforward.[16] If H is the matrix to be inverted then

$$H^{-1} = \begin{pmatrix} H_{11}^{-1}+H_{11}^{-1}H_{12}S^{-1}H_{21}H_{11}^{-1} & -H_{11}^{-1}H_{12}S^{-1} \\ -S^{-1}H_{21}H_{11}^{-1} & S^{-1} \end{pmatrix} \tag{1.40}$$

where $S = H_{22} - H_{21}H_{11}^{-1}H_{12}$. It is to be noted that the order of the computations is important. Once obtained H_{11}^{-1} it is possible to compute S and then all the other factors. Moreover the original matrix can be overwritten.

1.3.6. *System solution*

The solution of the system can be obtained either directly, through hierarchical matrix inversion,[20] or indirectly, through iterative solvers that exploit the efficient matrix-vector product in low rank format.[19,69]
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.[18,19] 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 hierarchical representation. This coarse approximation, with reduced memory storage, can then be decomposed through the hierarchical LU decomposition to give the preconditioner P. The resulting system

$$(LU)^{-1}Ax = (LU)^{-1}b \tag{1.41}$$

has a lower condition number and the convergence rate of iterative solvers is noticeably improved. It should be noted that in eq. (1.41) 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[7] with a coarse hierarchical left preconditioner has been used as solver.

1.3.7. *Some details about code implementation*

In this 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. 1.7) 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 respec-

```
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. 1.7. Basic data structure for hierarchical matrices

tively the columns and rows of the low rank representation.

1.4. Numerical experiments

In this section results obtained by applying the developed computational scheme are discussed. Both isotropic and anisotropic problems are analyzed and the performance of the developed scheme is assessed. The computations involving isotropic material structures have been performed using an Intel® Core™ 2 Duo Processor T5500 (1.66GHz) and 2 GB of RAM. The tests involving anisotropic materials have been performed on an Intel® Core™ 2 Duo Processor T9300 (2.5GHz) and 2 GB of RAM.

1.4.1. *Uncracked isotropic elastic bracket*

A mechanical element, Fig. 1.8, 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 seconds for the generation of the collocation matrix and 1484 seconds 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

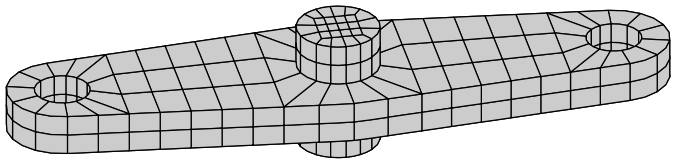

Fig. 1.8. Analyzed 3D configuration.

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 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 seconds, i.e. 57% of the standard assembly time, and it is recompressed and coarsened in 216 seconds. It may be of interest to mention that, in this specific application, 60% of the collocation matrix has been generated through ACA in 55 seconds (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 $\|x-\tilde{x}\|_{L^2}/\|x\|_{L^2} = 2.9\times10^{-4}$. 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 $\tilde{A}$ (the collocation matrix) and $\tilde{b}$ (right hand side).
Table 1.1 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

Table 1.1. Storage, time and speed up ratio 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]
$5 \cdot 10^{-1}$	1.45%	23.9	12.7	3906.0[c]	5000[c]	2.80[c]
$1 \cdot 10^{-1}$	4.95%	28.4	45.5	10.0	29	0.19
$5 \cdot 10^{-2}$	6.54%	31.5	63.2	8.7	25	0.21
$1 \cdot 10^{-2}$	10.59%	47.3	117.7	4.0	10	0.25
$5 \cdot 10^{-3}$	12.62%	58.5	153.2	3.9	9	0.28
$1 \cdot 10^{-3}$	17.31%	82.1	252.6	3.0	6	0.36
$5 \cdot 10^{-4}$	19.38%	90.8	303.4	2.6	5	0.40
$1 \cdot 10^{-4}$	25.20%	104.2	476.7	2.5	4	0.53
$5 \cdot 10^{-5}$	27.89%	97.9	586.2	2.1	3	0.60
$1 \cdot 10^{-5}$	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: $3.0 \cdot 10^{-2}$.
[c] GMRES relative accuracy: $6.1 \cdot 10^{-8}$.
[d] Only the time for copying the collocation matrix.

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. 1.9 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

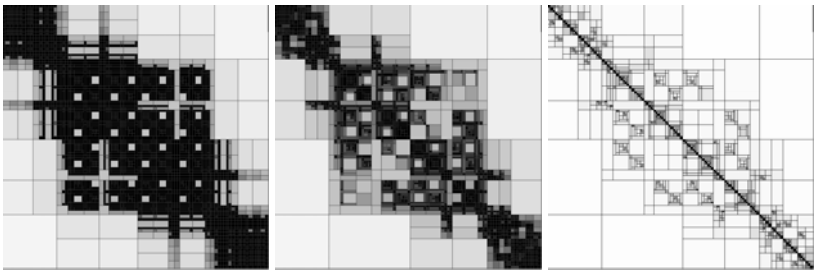

Fig. 1.9. Block-wise representation of the ACA generated matrix, the coarsened matrix and the preconditioner.

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 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 1.2 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 $\varepsilon_p = 1.0 \cdot 10^{-1}$ and $\eta = \sqrt{2}$. Memory requirements,

Table 1.2. Influence of the collocation matrix accuracy.

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

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 1.2 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 $0.1 \div 1.0\%$. 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 1.3 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 more slowly to the preset accuracy. This

Table 1.3. Influence of the admissibility parameter.

η	*Stor. A*	*Stor. B*	*N. of blocks*	*Assembly*	*Speed up*	*Coars.*	*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

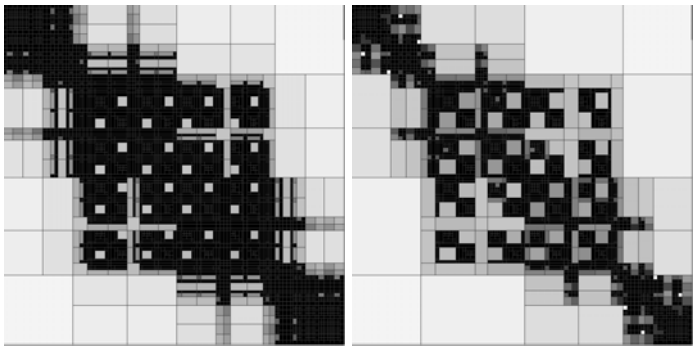

Fig. 1.10. Block-wise structure of the ACA matrix for $\eta = \sqrt{2}$ and $\eta = 4$..

aspect may have a negative effect on the assembly time, that is however balanced by more relevant reduction in the solution time. Fig. 1.10 shows the structure of the ACA matrix for two different values of η. It is evident that for $\eta = 4$ value more blocks become admissible than for $\eta = \sqrt{2}$.

1.4.2. *Embedded crack in isotropic bar*

As second configuration a cylinder with an embedded crack is considered, Fig. 1.11. The cylinder is subjected to uniaxial stress acting on the two bases, so to produce a mode I crack load.

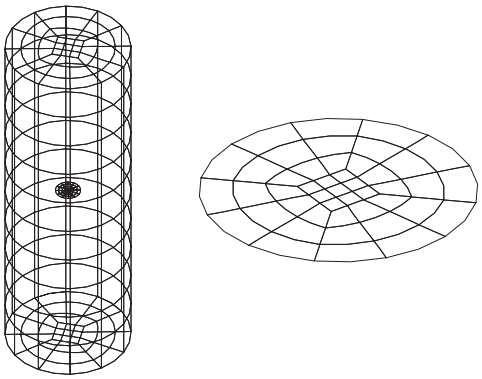

Fig. 1.11. Cylinder with embedded crack and crack basic mesh.

A mesh with 800 elements and 3652 nodes is considered. The stan-

dard technique requires 805 seconds to assemble the collocation matrix and 2394 seconds 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 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 seconds, i.e. 85% of the standard assembly time, it is compressed and coarsened in 168 seconds 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 1.4 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.

Table 1.4. 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]
$1 \cdot 10^{-0}$	0.86%	15.2	0.4	92.4	306	0.12
$5 \cdot 10^{-1}$	2.42%	15.7	115.2	60.1	197	0.15
$1 \cdot 10^{-1}$	5.00%	17.9	163.9	12.9	40	0.15
$1 \cdot 10^{-2}$	9.06%	26.5	273.2	5.9	16	0.20
$1 \cdot 10^{-3}$	13.39%	47.4	448.6	3.8	8	0.28
$1 \cdot 10^{-4}$	18.32%	68.6	802.8	2.5	4	0.43
$1 \cdot 10^{-5}$	23.69%	2.7[c]	1371.6	2.3	3	0.64

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

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 1.5. The analysis is performed setting $\varepsilon_c = 10^{-5}$ and $\varepsilon_p = 1.0$. The same considerations as in the previous case hold.

Table 1.5. Influence of the admissibility parameter.

η	*Stor. A*	*Stor. B*	*Blocks*	*A. speed up*	*Coarsening*	*S. 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

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 1.6 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.

Table 1.6. Memory savings and speed up ratios.

Elem.	*Nodes*	*Storage*	*St. Ass.*	*Speed up*	*St. Sol.*	*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

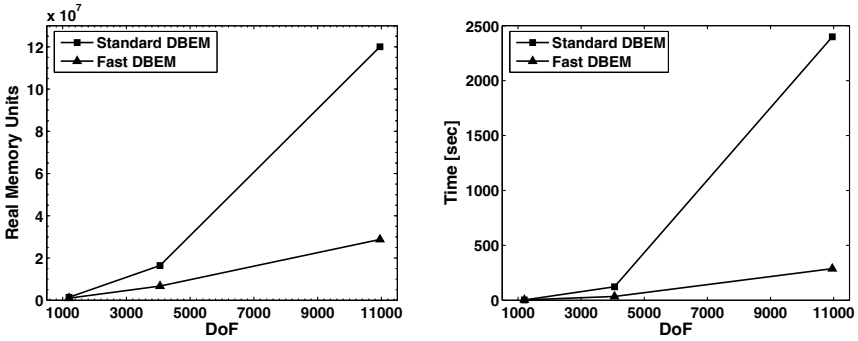

Fig. 1.12. Comparison between standard and fast DBEM.

Fig. 1.12 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.

Fig. 1.13 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.

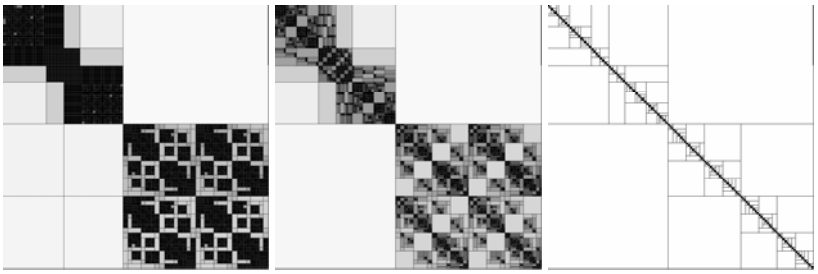

Fig. 1.13. Block-wise representation of the ACA generated matrix, the coarsened matrix and the preconditioner.

1.4.3. *Embedded crack in anisotropic bar*

The performance of the fast Hierarchical DBEM has also been tested for anisotropic crack problems, to assess the applicability of the proposed scheme to such class of problems and to evaluate the sensitivity of the method to different materials. The configuration shown in Fig. 1.14 has been analyzed (three sample meshes are depicted).

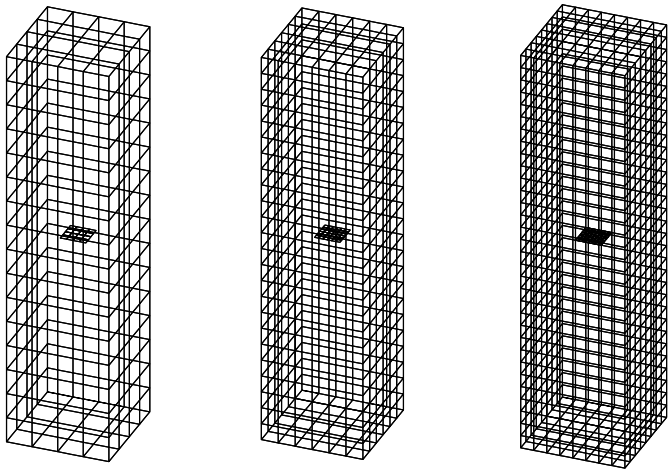

Fig. 1.14. Square crack embedded in anisotropic bar. Three uniformly refined meshes.

The prismatic bar is subjected to an opening load acting on the bases and different uniformly refined meshes have been considered for analyzing the performances of the solver at varying degrees of freedom.
Three different materials have been analyzed. The considered material properties are listed in Table 1.7. It is worth noting that, in the following reported results, the isotropic material configurations have been analyzed for uniformity by using the numerical scheme for anisotropy described in Section 1.2.2. The studies on the influence of the admissibility parameter, the collocation matrix accuracy and the preconditioner accuracy have been performed on a mesh with 720 elements and 2522 nodes.
Table 1.8 reports the influence of the admissibility parameter for the three materials. The analysis is performed setting $\varepsilon_c = 10^{-5}$ and $\varepsilon_p = 10^{-1}$. The minimal blocksize has been set to $n_{min} = 36$ and the GMRES relative

Table 1.7. Material properties [GPa].

C_{11}	C_{12}	C_{13}	C_{14}	C_{22}	C_{23}	C_{24}	C_{33}	C_{44}	C_{55}	C_{56}	C_{66}
Aluminum Polycrystal (isotropic)											
111	61	61	0	111	61	0	111	25	25	0	25
Barium sodium niobate (orthorhombic)											
239	104	50	0	247	52	0	135	65	66	0	76
Sapphire (trigonal)											
494	158	114	-23	494	114	23	496	145	145	-23	168

All the non reported constants are intended as zero.

accuracy is 10^{-8}. The same considerations as those reported for isotropic materials apply and no noticeable sensitivity with respect to the degree of anisotropy is shown. In particular it is apparent as some values of the admissibility parameter originate almost optimal block trees, thus reducing the coarsening time.

Table 1.9 reports storage and times for different preconditioner accuracies. For this purpose, the accuracy of the collocation matrix has been set to $\varepsilon_c = 10^{-5}$, the admissibility parameter has been chosen as $\eta = 4$ and

Table 1.8. Influence of the admissibility parameter.

η	*Stor. A*	*Stor. B*	*Blocks*	*A. speed up*	*Coarsening*	*S. speed up*
Aluminum Polycrystal (isotropic)						
1	53.0 %	31.7%	3757 - 1951	0.81	92.4	0.19
2	45.8 %	31.7%	3013 - 1951	0.81	66.8	0.15
4	41.6 %	31.6%	2347 - 1906	0.89	47.6	0.12
6	41.6 %	31.5%	2305 - 1867	0.90	48.0	0.12
8	41.5 %	31.5%	2287 - 1849	0.90	48.0	0.12
Barium sodium niobate (orthorhombic)						
1	52.9 %	31.9 %	3757 - 1978	0.79	90.8	0.18
2	45.5 %	31.9 %	3013 - 1978	0.79	66.0	0.15
4	41.2 %	31.8 %	2347 - 1936	0.87	47.3	0.12
6	41.2 %	31.8 %	2305 - 1900	0.87	47.1	0.12
8	41.1 %	31.8 %	2287 - 1882	0.88	47.2	0.12
Sapphire (trigonal)						
1	53.5 %	33.8%	3757 - 2152	0.82	99.5	0.19
2	46.4 %	33.8%	3013 - 2146	0.83	65.8	0.15
4	42.5 %	33.5%	2347 - 2035	0.91	47.8	0.12
6	42.5 %	33.5%	2305 - 1999	0.92	47.8	0.12
8	42.5 %	33.4%	2287 - 1981	0.93	47.6	0.12

Table 1.9. Storage and times for different preconditioner accuracies.

ε_p	*Storage*	*Setup (s)*	*LU (s)*	*GMRES (s)*	*Iterations*	*Speed up*
Aluminum Polycrystal (isotropic)						
No Prec.	0.0%	0.0	0.0	828.6[a]	3000[a]	1.27[a]
Jacobi	0.0%	0.0	0.0	750.3[b]	3000[a]	1.23[a]
$1 \cdot 10^{-0}$	1.2%	6.0	0.1	23.8	188	0.12
$1 \cdot 10^{-1}$	5.3%	6.9	17.5	3.4	23	0.12
$1 \cdot 10^{-2}$	10.3%	11.0	35.6	2.2	13	0.15
$1 \cdot 10^{-3}$	15.8%	22.8	63.9	1.2	6	0.21
$1 \cdot 10^{-4}$	22.4%	32.2	123.4	1.0	4	0.31
Barium sodium niobate (orthorhombic)						
No Prec.	0.00%	0.0	0.0	803.2[a]	3000[a]	1.31[a]
Jacobi	0.00%	0.0	0.0	764.3[a]	3000[a]	1.25[a]
$1 \cdot 10^{-0}$	1.2 %	6.9	0.1	37.2	260	0.14
$1 \cdot 10^{-1}$	5.6 %	7.5	17.8	3.6	24	0.12
$1 \cdot 10^{-2}$	10.4 %	12.0	35.4	2.9	17	0.15
$1 \cdot 10^{-3}$	16.0 %	25.1	67.9	1.1	5	0.22
$1 \cdot 10^{-4}$	22.7 %	33.3	125.7	0.8	3	0.32
Sapphire (trigonal)						
No Prec.	0.00%	0.0	0.0	825.1[a]	3000[a]	1.37[a]
Jacobi	0.00%	0.0	0.0	800.0[a]	3000[a]	1.33[a]
$1 \cdot 10^{-0}$	1.2 %	7.2	0.1	35.3	236	0.14
$1 \cdot 10^{-1}$	5.6 %	8.2	16.3	4.2	26	0.12
$1 \cdot 10^{-2}$	10.6%	13.0	40.8	2.0	11	0.16
$1 \cdot 10^{-3}$	16.6%	27.8	72.5	1.1	5	0.23
$1 \cdot 10^{-4}$	24.2%	24.8	120.3	1.1	4	0.30

[a] No convergence reached.

the minimal blocksize has been set to $n_{min} = 36$. The GMRES relative accuracy is 10^{-8}. As can be observed the coarsest preconditioners provide faster solution. Also for the anisotropic case it is shown that preconditioning is needed and both no preconditioned systems and Jacobi preconditioned systems fail to converge within the maximum number of iterations chosen. The preconditioner setup time increases at increasing requested accuracies while the number of GMRES iterations decreases. It is worth highlighting once again that more accurate preconditioners are closer to the inverse of the collocation matrix and their use virtually close the iterative technique to the direct solution.

Table 1.10 reports memory savings and speed up ratios versus the number of degrees of freedom. The settings are $\varepsilon_c = 10^{-5}$, $\varepsilon_p = 10^{-1}$, $\eta = 4$, $n_{min} = 36$ and the GMRES relative accuracy is 10^{-8}. It is apparent that storage memory, assembly time and solution time vary almost linearly with respect

Table 1.10. Memory savings and speed up ratios.

Elem.	*Nodes*	*Storage*	*St. Ass.*	*Speed up*	*St. Sol.*	*Speed Up*	*Iterations*
Aluminum Polycrystal (isotropic)							
320	1122	49.5%	39.9	1.09	56.0	0.30	19
500	1752	41.2%	79.1	1.01	213.1	0.17	18
720	2522	31.6%	149.0	0.89	638.3	0.12	23
980	3432	25.4%	283.7	0.71	1606.4	0.12	27
1280	4482	20.0%	515.0	0.61	3582.6	0.09	24
Barium sodium niobate (orthorhombic)							
320	1122	49.5%	39.9	1.08	56.2	0.30	25
500	1752	41.3%	79.0	1.01	213.3	0.18	18
720	2522	31.9%	152.1	0.87	647.3	0.12	24
980	3432	26.0%	284.7	0.70	1674.1	0.11	28
1280	4482	20.5%	518.2	0.59	3633.2	0.08	26
Sapphire (trigonal)							
320	1122	50.8%	39.8	1.11	55.9	0.31	20
500	1752	43.1%	79.3	1.02	214.0	0.17	16
720	2522	33.5%	149.0	0.92	638.3	0.12	26
980	3432	27.6%	287.6	0.72	1601.7	0.11	26
1280	4482	21.8%	515.0	0.64	3582.9	0.09	25

to the number of degrees of freedom, thus providing considerable savings for large systems in comparison to direct solvers. Such considerations apply to all the considered materials highlighting the versatility of the technique and the positive independence from the degree of anisotropy. This aspect is particularly appealing as anisotropic problems are usually quite time consuming. The presented results compare well also with those computed using the pure isotropic scheme and reported in the previous sections, thus confirming the robustness of the approach.

Fig. 1.15 depicts the assembly times and solution times at varying degrees of freedom for the three different materials. Only the assembly time shows a slight sensitivity to the material, while the solution time appears to be quite insensitive to it.

1.5. Conclusions

In this chapter recent developments in the framework of fast solutions of BEM and DBEM system of equations have been presented. The development of a fast Hierarchical DBEM has been described and applications to both isotropic and anisotropic 3D crack problems have been presented.

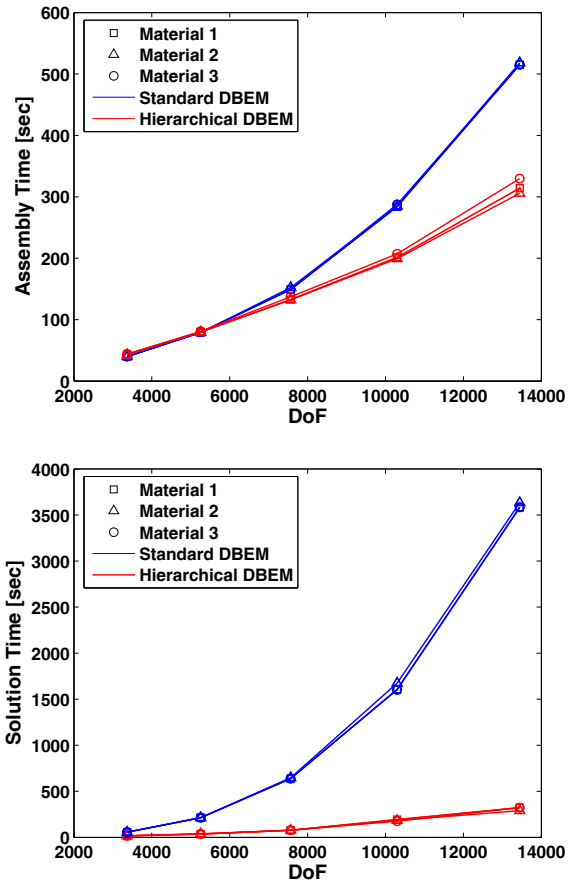

Fig. 1.15. Comparison between standard and Hierarchical DBEM.

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. Such considerations apply to both isotropic and anisotropic problems having the study demonstrated negligible sensitivity with respect to the degree of anisotropy. The results show

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.

References

1. L. C. Wrobel, *"The boundary element method: applications in thermo-fluids and acoustics"*. vol. 1, (John Wiley & Sons, Ltd, 2002).
2. M. H. Aliabadi, *"The boundary element method: applications in solids and structures"*. vol. 2, (John Wiley & Sons, Ltd, 2002).
3. H. Rokhlin, "Rapid solution of integral equations of classical potential theory", *Journal of Computational Physics*. **60**, 187–207, (1985).
4. J. Barnes and P. Hut, "A hierarchical $O(N \ln N)$ force-calculation algorithm", *Nature*. **324**, 446–449, (1986).
5. L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations", *Journal of Computational Physics*. **73**(2), 325–348, (1987).
6. N. Nishimura, K. I. Yoshida, and S. Kobayashi, "A fast multipole boundary integral equation method for crack problems in 3D", *Engineering Analysis with Boundary Elements*. **23**, 97–105, (1999).
7. Y. Saad and M. H. Schultz, "GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems", *SIAM J. Sci. Stat. Comput.* **7**(3), 856–869, (1986).
8. Y. Fu, K. J. Klimkowski, G. J. Rodin, E. Berger, J. C. Browne, J. K. Singer, R. A. Van de Geijn, and K. S. Vemaganti, "A fast solution method for three-dimensional many-particle problems of linear elasticity", *International Journal for Numerical Methods in Engineering*. **42**, 1215–1229, (1998).
9. V. Popov and H. Power, "A $O(N)$ Taylor series multipole boundary element method for three-dimensional elasticity problems", *Engineering Analysis with Boundary Elements*. **25**, 7–18, (2001).
10. W. Hackbusch and Z. P. Nowak, "On the fast matrix multiplication in the boundary element method by panel clustering", *Numerische Mathematik*. **73**, 207–243, (1989).
11. E. E. Tyrtyshnikov, "Mosaic-skeleton approximations", *Calcolo*. **33**, 47–57, (1996).
12. S. A. Goreinov, E. E. Tyrtyshnikov, and N. L. Zamarashkin, "A theory of pseudoskeleton approximations", *Linear algebra and its applications*. **261**, 1–21, (1997).
13. E. E. Tyrtyshnikov, "Incomplete cross approximation in the Mosaic-Skeleton Method", *Computing*. **64**, 367–380, (2000).

14. M. Bebendorf, "Approximation of boundary element matrices", *Numerische Mathematik*. **86**, 565–589, (2000).
15. M. Bebendorf and S. Rjasanow, "Adaptive low-rank approximation of collocation matrices", *Computing*. **70**, 1–24, (2003).
16. W. Hackbusch, "A sparse matrix arithmetic based on H-matrices. Part I: introduction to H-matrices", *Computing*. **62**, 89–108, (1999).
17. W. Hackbusch and B. N. Khoromskij, "A sparse H-matrix arithmetic. Part II: application to multidimensional problems", *Computing*. **64**, 21–47, (2000).
18. L. Grasedyck, "Adaptive recompression of H-matrices for BEM", *Computing*. **74**, 205–223, (2005).
19. M. Bebendorf, "Hierarchical LU decomposition-based preconditioners for BEM", *Computing*. **74**, 225–247, (2005).
20. L. Grasedyck and W. Hackbusch, "Construction and arithmetics of H-matrices", *Computing*. **70**, 295–334, (2003).
21. S. Börm, L. Grasedyck, and W. Hackbusch, "Introduction to hierarchical matrices with applications", *Engineering Analysis with Boundary Elements*. **27**, 405–422, (2003).
22. R. L. Mullen and J. J. Rencis, "Iterative methods for solving boundary element equations", *Computers & Structures*. **25**(5), 713–723, (1987).
23. W. J. Mansur, F. C. Araujo, and E. B. Malaghini, "Solution of BEM systems of equations via iterative techniques", *International Journal for Numerical Methods in Engineering*. **33**, 1823–1841, (1992).
24. F. P. Valente and H. L. G. Pina, "Iterative solvers for BEM algebraic systems of equations", *Engineering Analysis with Boundary Elements*. **22**, 117–124, (1998).
25. L. P. S. Barra, A. L. G. A. Coutinho, W. J. Mansur, and J. F. C. Telles, "Iterative solution of BEM equations by GMRES algorithm", *Computers & Structures*. **44**(6), 1249–1253, (1992).
26. K. Guru Prasad, J. H. Kane, D. E. Keyes, and C. Balakrishna, "Preconditioned Krylov solvers for BEA", *International Journal for Numerical Methods in Engineering*. **37**, 1651–1672, (1994).
27. T. J. Urekew and J. J. Rencis, "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, (1993).
28. M. Merkel, V. Bulgakov, R. Bialecki, and G. Kuhn, "Iterative solution of large-scale 3D-BEM industrial problems", *Engineering Analysis with Boundary Elements*. **22**, 183–197, (1998).
29. C. Y. Leung and S. P. Walker, "Iterative solution of large three-dimensional BEM elastostatic analyses using the GMRES technique", *International Journal for Numerical Methods in Engineering*. **40**, 2227–2236, (1997).
30. L. P. S. Barra, A. L. G. A. Coutinho, J. F. C. Telles, and W. J. Mansur, "Multi-level hierarchical preconditioners for boundary element systems", *Engineering Analysis with Boundary Elements*. **12**, 103–109, (1993).
31. H. Wang, Z. Yao, and P. Wang, "On the preconditioners for fast multipole boundary element methods for 2D multi-domain alastostatics", *Engineering*

Analysis with Boundary Elements. **29**, 673–688, (2005).
32. M. Benzi, "Preconditioning techniques for large linear systems: a survey", *Journal of Computational Physics*. **182**, 418–477, (2002).
33. Y. Saad and H. A. van der Vorst, "Iterative solution of linear systems in the 20th century", *Journal of Computational and Applied Mathematics*. **123**, 1–33, (2000).
34. S. Kurz, O. Rain, and S. Rjasanow, "The adaptive cross approximation technique for the 3-D boundary element method", *IEEE Transactions on Magnetics*. **38**(2), 421–424, (2002).
35. K. Zhao, M. N. Vouvakis, and J. F. Lee, "The adaptive cross approximation algorithm for accelerated method of moments computation of EMC problems", *IEEE Transaction on Electromagnetic Compatibility*. **47**, 763–773, (2005).
36. J. Ostrowski, K. I. Andjelíc, M. Bebendorf, B. Crangânu-Creţu, and J. Smajić, "Fast BEM-solution of Laplace problems with H-matrices and ACA", *IEEE Transaction on Magnetics*. **42**(4), 627–630, (2006).
37. M. Bebendorf and R. Grzhibovskis, "Accelerating Galerkin BEM for linear elasticity using adaptive cross approximation", *Mathematical Methods in the Applied Sciences*. **29**, 1721–1747, (2006).
38. I. Benedetti, M. H. Aliabadi, and G. Daví, "A fast 3D dual boundary element method based on hierarchical matrices", *International Journal of solids and Structures*. **45**(7–8), 2355–2376, (2008).
39. I. Benedetti, A. Milazzo, and M. H. Aliabadi. "A fast 3D BEM for anisotropic elasticity based on hierarchical matricess". In eds. M. H. Aliabadi and R. Abascal, *Advances in Boundary Element Techniques IX, Proceedings of the International Conference on Boundary Element Techniques*, pp. 433–438, Sevilla, Spain (July, 2008). EC, Ltd. UK.
40. A. Portela, M. H. Aliabadi, and D. P. Rooke, "The Dual Boundary Element Method: effective implementation for crack problems", *International Journal for Numerical Methods in Engineering*. **33**, 1269–1287, (1992).
41. A. Portela, M. H. Aliabadi, and D. P. Rooke, "Dual boundary element incremental analysis of crack propagation", *Computers & Structures*. **46**(2), 237–247, (1993).
42. Y. Mi and M. H. Aliabadi, "Dual Boundary Element Method for three-dimensional fracture mechanics analysis", *Engineering Analysis with Boundary Elements*. **10**, 161–171, (1992).
43. Y. Mi and M. H. Aliabadi, "Three-dimensional crack growth simulation using BEM", *Computers & Structures*. **52**(5), 871–878, (1994).
44. V. Sladek and J. Sladek, "Three dimensional crack analysis for anisotropic body", *Applied Mathematical Modelling*. **6**, 374–382, (1982).
45. A. Le Van and J. Royer, "Boundary formulation for three-dimensional anisotropic crack problems", *Applied Mathematical Modelling*. **20**, 662–674, (1996).
46. E. Pan and F. G. Yuan, "Boundary element analysis of three-dimensional cracks in anisotropic solids", *International Journal for Numerical Methods in Engineering*. **48**, 211–237, (2000).

47. M. P. Ariza and J. Dominguez, "Boundary element formulation for 3D transversly isotropic cracked bodies", *International Journal for Numerical Methods in Engineering*. **60**, 719–753, (2004).
48. Z. Q. Yue, H. T. Xiao, and E. Pan, "Stress intensity factors of square crack inclined to interface of transversly isotropic bimaterial", *Engineering Analysis with Boundary Elements*. **31**, 50–65, (2007).
49. M. H. Aliabadi, "Boundary element formulations in fracture mechanics", *Applied Mechanics Reviews*. **50**(2), 83–96, (1997).
50. M. H. Aliabadi, "A new generation of boundary element methods in fracture mechanics", *International Journal of Fracture*. **86**, 91–125, (1997).
51. N. A. Schlar, *"Anisotropic analysis using boundary elements"*. Computational Mechanics Publications, 1994).
52. Y. C. Pan and T. W. Chou, "Point force solution for an infinite transversely isotropic solid", *Journal of Applied Mechanics*. **43**, 608–612, (1976).
53. S. M. Vogel and F. J. Rizzo, "An integral equation formulation of three-dimensional anisotropic elastostatic boundary value problems", *Journal of Elasticity*. **3**, 203–216, (1973).
54. R. B. Wilson and T. A. Cruse, "Efficient implementation of anisotropic three dimensional boundary-integral equation stress analysis", *International Journal for Numerical Methods in Engineering*. **12**, 1383–1397, (1978).
55. C. Y. Wang and M. Denda, "3D BEM for general anisotropic elasticity", *International Journal of Solids & Structures*. **44**, 7073–7091, (2007).
56. T. Chen and F. Z. Lin, "Boundary integral formulations for three-dimensional anisotropic solids", *Computational Mechanics*. **15**, 485–496, (1995).
57. T. C. T. Ting and V. G. Lee, "The three-dimensional elastostatic Green's function for general anisotropic linear elastic solids", *Quarterly Journal of Mechanics and Applied Mathematics*. **50**, 407–426, (1997).
58. G. Nakamura and K. Tanuma, "A formula for the fundamental solution of anisotropic elasticity", *Quarterly Journal of Mechanics and Applied Mathematics*. **50**, 179–194, (1997).
59. M. A. Sales and L. J. Gray, "Evaluation of the anisotropic Green's function and its derivatives", *Computers & Structures*. **69**, 247–254, (1998).
60. F. Tonon, E. Pan, and B. Amadei, "Green's functions and boundary element method formulation for 3D anisotropic media", *Computers & Structures*. **79**, 469–482, (2001).
61. A. P. Cisilino and M. H. Aliabadi, "Three-dimensional BEM analysis for fatigue crack growth in welded components", *International Journal of Pressure Vessels & Piping*. **70**, 135–144, (1997).
62. A. M. H. Cisilino, A P., "Dual Boundary Element assessment of three-dimensional fatigue crack growth", *Engineering Analysis with Boundary Elements*. **28**, 1157–1173, (2004).
63. H. R. Kutt, "The numerical evaluation of principal value integrals by finite-part integration", *Numerische Mathematik*. **24**, 205–210, (1975).
64. H. R. Kutt. Quadrature formulae for finite-part integrals. Technical Report WSK 178, National Research Institute for Mathematical Sciences, CSIR, Pretoria, (1975).

65. H. R. Kutt. On the numerical evaluation of finite-part integrals involving an algebraic singularity. Technical Report WSK 179, National Research Institute for Mathematical Sciences, CSIR, Pretoria, (1975).
66. M. Bebendorf. *Effiziente numerische Losung von Randintegralgleichungen unter Verwendung von Niedrigrang-Matrizen*. PhD thesis, Universitat Saarbrucken, (2001).
67. W. Hackbusch, B. N. Khoromskij, and R. Kriemann, "Hierarchical matrices based on weak admissibility criterion", *Computing*. **73**, 207–243, (2004).
68. K. Giebermann, "Multilevel approximation of boundary integral operators", *Computing*. **67**, 183–207, (2001).
69. M. Bebendorf, "Approximate inverse preconditioning of FE systems for elliptic operators with non-smooth coefficients", *SIAM Journal on Matrix Analysis and Applications*. **27**(4), 909–929, (2006).
70. M. Bebendorf and R. Kriemann, "Fast parallel solution of boundary integral equations and related problems", *Computing and Visualization in Science*. **8**, 121–135, (2005).