Parametric analysis of ACA-based solver for BEM 3D

T. Grytsenko & A. Peratta
Wessex Institute of Technology, UK

Abstract

The standard Boundary Element Method (BEM) with single domain yields the dense linear system of equation (LSE). This LSE cannot be solved efficiently by existing iterative solvers such as CG, GMRES, TFQMR, etc. Moreover, such LSE cannot be stored in advanced formats such as Compressed Sparse Row or Modified Sparse Row and therefore cannot be compressed in order to economise memory consumption. Methods such as the fast multipole method and panel clustering gain their efficiency from approximating the kernel function. This paper implements and studies the parameters and performance of the Adaptive Cross Approximation (ACA) algorithm for reducing the rank of off-diagonal blocks in the three dimensional BEM. The ACA algorithm is purely algebraic and does not need to deal with the kernel. The algorithm uses a hierarchical matrix storage approach where the matrix is split into many blocks. The off-diagonal blocks of this matrix represent remote interactions between source points and are therefore approximated by low-rank matrices with the ACA approach. Those blocks, which describe close interactions between source points, are stored without any changes. These reorganisations in conjunction with novel algorithms for manipulation with H-matrices reduce the calculation complexity of matrix-vector multiplication (MVM) to approximately $O(N)$. This simplification of MVM paves the road for speeding up of the solution of LSE coming from BEM, as MVM is a key-operation for many existing solvers. This paper analyses the ACA-based solver and describes how to choose parameters for this solver when used in conjunction with BEM. Finally, a 3D numerical example solved with this technique is presented.
Keywords: BEM 3D, hierarchical matrix, adaptive cross approximation, iterative solver.

1 Introduction

The classical BEM method yields a dense non-symmetric linear system of equations (LSE). This LSE cannot be solved efficiently by existing iterative solvers such as CG, GMRES, TFQMR, etc, which exploit the sparse pattern of the matrix. Moreover, it is inefficient to store such dense LSE in sparse formats such as Compressed Sparse Row (CSR) or Modified Sparse Row (MSR). In contrast, other methods such as the fast multipole method and panel clustering gain their efficiency from approximating the kernel function [1,2], yielding a sparse LSE. This paper implements and studies the parameters and performance of the Adaptive Cross Approximation (ACA) algorithm [5] for reducing the rank of off-diagonal blocks in the three dimensional BEM matrix. The great advantage of the ACA algorithm is that it is purely algebraic and does not need to deal with the kernel. The algorithm uses a hierarchical matrix storage approach where the matrix is split into many blocks classified into two categories, weakly and strongly coupled. The former are off-diagonal blocks, which represent remote interactions between source points and field elements, and therefore can be approximated by low-rank matrices. For this purpose Adaptive Cross Approximation (ACA) algorithm is used [5]. These blocks are stored in a special Rk-format. The latter blocks describe close interactions between source points, and field elements are stored without any changes in a full-matrix format. This reorganisation of the LSE, implemented in conjunction with novel algorithms for manipulation with H-matrices, reduces the calculation complexity of matrix-vector multiplication (MVM) to approximately $O(N)$ [2]. The simplification of MVM considerably reduces the computational burden of the solving stage.

This paper is organised as follows: in Section 2, key-points of H-matrices technique are given, particularly their structure, cluster tree, domain partitioning and admissibility conditions; in Section 3, two versions of ACA-algorithm are described; Section 4 analyses ACA-algorithm and describes how to choose parameters for this solver; Section 5 represents numerical results; Section 6 makes the conclusions; our acknowledgements are expressed in the final section.

2 Hierarchical matrices

This section provides a brief introduction to hierarchical matrices and explains their structure and creation. The concepts of mesh partitioning, cluster tree structure and admissibility condition [2–4] are also described.

2.1 Mesh partitioning and its storage scheme

The main task of mesh partitioning scheme is to build a hierarchy of source points (indices) according to geometrical criterion. The indices I of nodes are stored in the so-called cluster tree [2] (see Figure 1). The root of the tree T_I is the index set $T_I = \{0,...,N-1\}$ where N is the number of degrees of freedom.

The subsequent node t with more than N_{ind} indices, where N_{ind} is a parameter, has exactly two successors: the first contains the first half of its indices and the second one the second half. Nodes with no more than N_{ind} indices are leaves. There are many ways to build such a partition [6]. In this work geometric a bisection algorithm has been used. Cluster tree for a circle with 10 degrees of freedom (DOF) and $N_{ind} = 2$ is represented in Figure 1.

In Figure 1 notation $I_{5(2)}^{(3)}$ means that node number 5 is situated on third level of cluster tree and consists of two indices or degrees of freedom. Cluster tree is a basis for construction of H-matrix.

2.2 Structure of hierarchical matrices

In general case H-matrix represents hierarchy of Rk-matrices and Full-matrices [2,4]. An $n \times m$ matrix M of rank at most k is said to be stored in Rk-matrix representation if it is stored in factorised form $M = AB^T$ where matrices $A \in R^{n \times k}$ and $B \in R^{m \times k}$ are both stored as an array (column-wise). In C programming language it looks as in Figure 2 (i).

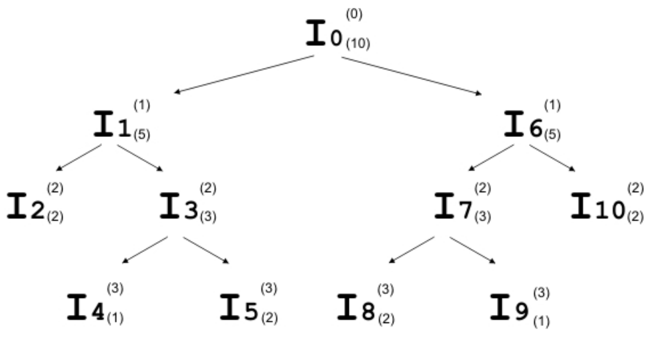

Figure 1: Cluster tree (N=*10*, N_{ind} = 2).

`struct rkmatrix {`
`Int k;`
`Int rows;`
`Int cols;`
`Double * a;`
`Double * b;`
`};`
	`struct   fullmatrix`
	`{`
	`Int rows;`
	`Int cols;`
	`Double * e;`
	`};`
		`struct`
		`supermatrix {`
		`Int rows;`
		`Int cols;`
		`Int block_rows;`
		`Int block_cols;`
		`Prkmatrix r;`
		`Pfullmatrix f;`
		`Psupermatrix * s;`
		`};`

(i) – Rk- matrix	(ii) – Full-matrix	(iii) – Super matrix

Figure 2: Basic structures in H-matrix theory.

An $n \times m$ matrix M is said to be stored in Full matrix representation if the entries M_{ij} are stored as real numbers in an array of length $n*m$ in the order $M_{11},...,M_{n1},M_{12},...,M_{n2},\dots,M_{1m},...,M_{nm}$. In C programming language it looks as in Figure 2 (ii).

The matrix M is said to be stored in H-matrix representation if the submatrices or blocks are stored either in Full-matrix or Rk-matrix representation. Each block in the H-matrix tree can be: (i) a leaf: then the corresponding matrix block is represented by Full matrix or Rk-matrix; (ii) a node: then the block is decomposed into sons. This means that the matrix corresponding to this block is a supermatrix that consists of submatrices corresponding to its sons. In C programming language it looks as in Figure 2 (iii).

2.3 Admissibility condition

Each block in H-matrix represents interactions of nodes in a cluster tree, particularly interactions of their DOFs. Those nodes of cluster tree which are geometrically situated far one from the other form the blocks in Rk-format and those that situated closely yields the blocks in Full-format. There is a special criterion in order to find how far one from the others nodes are:

$$dist(C_{N1}, C_{N2}) \geq \gamma * Max\_diam(C_{N1}, C_{N2}), \tag{1}$$

where C_{N1}, C_{N2} are clusters in a cluster tree; γ is a coefficient and $Max\_diam(C_{N1}, C_{N2})$ is a function that finds maximal diameter among clusters C_{N1}, C_{N2}. This criterion calls admissibility condition [4]. Those clusters that are admissible according to (1) form blocks in Rk-format and those which are not generate blocks in Full-format.

Adaptive cross approximation (ACA)-algorithm described in the next section is developed for low-rank approximation of large matrices. This algorithm works with whole H-matrix. If a block is not admissible then its entries are stored without approximation, otherwise ACA can be applied. So, ACA approximates only admissible block, i.e. blocks stored in Rk-format.

3 Adaptive cross approximation

The off-diagonal blocks of H-matrix describe remote interactions between source points and can be approximated by low rank matrices [5]. ACA algorithm is developed to generate low rank approximants for such blocks. In contrast to other methods such as fast multipole, panel clustering, etc., the low-rank approximant is not generated by replacing the kernel function of the integral operator. This algorithm is fully algebraic and uses the original matrix entries to compute the low-rank approximant. ACA algorithm does not change kernel function and can be applied only to the discrete integral operators with asymptotically smooth kernels.

There are two versions of ACA algorithm: fully and partially pivoted. Fully pivoted realisation is feasible in a case when coefficient matrix $\mathbf{A}$ of the equation

$$\mathbf{A}\,\mathbf{x} = \mathbf{b} \tag{2}$$

is already computed. In this case, ACA in conjunction with H-matrix technique are used to speedup solving of LSE. However, if matrix A is not yet computed and there is a possibility to generate its entries individually there is no need to compute whole matrix beforehand. Instead, ACA approximation can be integrated into computation stage in order to approximate this matrix on the fly. In this case not all entries of whole matrix must be calculated. For this purpose partially pivoted ACA algorithm can be used. Detailed description can be found in [5].

There is an error of ACA-approximation ε_{ACA}, which defines how deep the approximation is. $\varepsilon_{ACA} = 0$ means that there is no approximation, any $\varepsilon_{ACA} > 0$ leads to changes in the original matrix A (2). By using ACA algorithm any $\varepsilon_{ACA} \geq 0$ may be reached. The memory capacity needed for storing an approximant depends on ε_{ACA}: higher error will need less memory. However, high ε_{ACA} may damage LSE and as a result will lead to a very coarse result, especially if the system is ill-conditioned. So, selection of ε_{ACA} is a matter of balance between memory capacity, CPU time and precision of the result x of the equation (2).

4 Selection of ACA parameters

It is desirable that the auxiliary parameters controlling the behaviour of a solver are adjusted in order to optimise: CPU time T_{CPU}, memory consumption M and the solution error ε_{sol}. ACA-based solver involves some auxiliary parameters that play a key role in the optimisation of those main characteristics: γ from the equation (1), size of cluster N_{ind} in cluster tree, ACA-error ε_{ACA} and number of DOFs N.

In order to find optimal values for each parameter it is necessary to define dependencies between them. The most important parameter is the solution error, which depends on three auxiliary parameters $\gamma, N_{ind}, \varepsilon_{ACA}$. Different values of γ and N_{ind} yield different cluster tree and as a result different H-matrix:

$$K_{BL}(N, N_{ind}) = a + b\,N + c\,N^2 + d\,/\,N_{ind} \tag{3}$$

where $K_{BL} = N_{RK} / N_{FULL}$ is the ratio between the number of Rk-blocks and Full-blocks in H-matrix, and *a*, *b*, *c*, *d* are fitting coefficients. This relationship and others in this paper have been established by means of non-linear least-squares method for the model described in Section 5. Depending on parameter γ the fitting coefficients change as follows:

$$K_{BL} = 0.0257 + 1e-04\,N + 8.65e-09\,N^2 + \\ +33.37/N_{ind} \quad for\ \gamma = 0.5 \tag{4}$$

$$K_{BL} = 0.725 + 9.83e-05 * N + 1.34e-09 * N^2 + \\ +36.98/N_{ind} \quad for\ \gamma = 1.0 \tag{5}$$

$$K_{BL} = 0.492 + 2.2e-04 * N - 3.13e-08 * N^2 + \\ +68.98/N_{ind} \quad for\ \gamma = 2.5 \tag{6}$$

The higher γ yields higher K_{BL} and as a result higher solution error:

$$\ln \varepsilon_{sol}(K_{BL}, \varepsilon_{ACA}) = -4.21 - 1.67 * e^{-K_{BL}} + 38.35 * \varepsilon_{ACA} \tag{7}$$

where $\varepsilon_{sol} = \| \mathbf{b} - \mathbf{A}\,\mathbf{x} \|_2$ (see eq. 2)

Coefficient K_{BL} depends not only on γ and N_{ind} but also on model geometry and mesh partitioner. The equations shown above are only applicable to the theoretical example (see Section 5) and geometric bisection partitioning. Different geometries may lead to different forms of dependencies but the tendencies are the same:

(i) The less N_{ind} yields higher K_{BL};
(ii) The higher γ yields higher K_{BL};
(iii) The higher K_{BL} yields higher solution error;
(iv) The higher ε_{ACA} yields higher solution error.

In most cases ε_{sol} and ε_{ACA} are of the same order of magnitude. So, in order to guarantee the solution error it is feasible to choose ε_{ACA} so that it is one order less the solution error, i.e. if the solution error $\varepsilon_{sol} = 10^{-5}$ then $\varepsilon_{ACA} = 10^{-6}$.

Let us consider how K_{BL} and ε_{ACA} affect memory consumption for different values of N. In order to simplify equations two dependencies have been measured: for floating K_{BL} keeping the same $\varepsilon_{ACA} = 0.001$ (8) and for different ε_{ACA} keeping the same $K_{BL} \approx 2.0$ (9).

$$\ln M(N, K_{BL}) = 15.1 + 0.056N^2 - 1.18K_{BL} + 0.57K_{BL}^{1.5} - 0.7\,K_{BL}^2 \tag{8}$$

$$\ln M(N, \varepsilon_{ACA}) = 5.7 + 1.34 \ln N - 0.093 \ln \varepsilon_{ACA} + 1.12 / \ln \varepsilon_{ACA} \quad (9)$$

The evaluation of functions $T_{CPU}(N, K_{BL})$ and $T_{CPU}(N, \varepsilon_{ACA})$ gives very similar dependencies as for memory consumption M (8), (9) and shows that the higher K_{BL} as well as higher ε_{ACA} reduce T_{CPU} and M. There is a contradiction: from one hand, higher K_{BL} as well as higher ε_{ACA} reduce memory consumption but from the other hand they yield higher solution error. The solution of this contradiction is a question of balancing between involved parameters. There are three possible cases:

1. The solution has to be obtained as fast as possible and its error is not critical. In this case CPU time has priority. ε_{ACA} has to be chosen as high as possible and K_{BL} has to be optimal. An optimal value of K_{BL} depends on N_{ind}. Figure 3 shows that optimal value of N_{ind} is not the smallest one and therefore function $T_{CPU}(N_{ind})$ has a minimum in N_{ind}^{opt} - optimal value of N_{ind} for current cluster tree. This happens because very small values of N_{ind} lead to huge K_{BL}. As a result H-matrix represents a very rough approximation of the source matrix A (2) and iterative techniques for the solution of LSE need more iterations to converge. This fact obviously has a connection with the condition number of LSE, which goes up as ε_{ACA} grows. There is no special rule on how to find optimal value of N_{ind} for any cluster tree but in most cases this value is less than $0.05\,N$, where N is a total number of degrees of freedom.

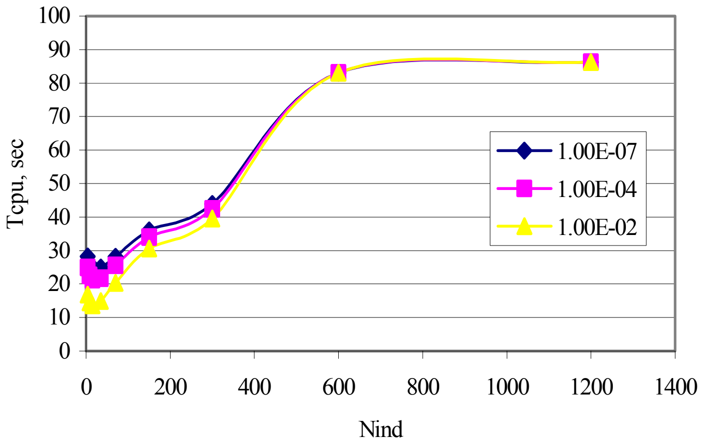

Figure 3: T_{CPU} vs. N_{ind} for different ACA-errors.

2. CPU time is not very critical but solution error has to be as small as possible. In this case K_{BL} and ε_{ACA} also have to be as small as possible. ε_{ACA} in this case has to be chosen one order less the needed solution error.

3. Memory capacity is limited and therefore memory consumption must be minimal. This case is very similar to the case number 1 because optimisation of CPU time is directly connected to minimisation of memory consumption. However, memory consumption does not have any tricks with N_{ind} and smaller N_{ind} leads to smaller matrix A (2) as lower N_{ind} generates more Rk-blocks which can be compressed up to chosen error ε_{ACA}.

5 Numerical examples

This section demonstrates efficiency of H-matrix technique in conjunction with ACA algorithm. The aim of these computations is to examine the performance of the ACA algorithm and to apply the results of the analysis made in Section 4. The ACA-based solver deals with Laplace equation with mixed boundary conditions: Dirichlet and Neumann. The boundary is meshed by linear triangular elements. The H-matrix storage approach and the ACA algorithm have been used as a black-box solver. As an input ACA-based solver takes matrix A (2) which has to be generated in advance by a BEM software, vector of coordinates for each DOF corresponding to each column and row in matrix A as well as the right-hand side vector of eq. (2). After ACA-approximation generalised minimal residual method (GMRES) is applied with the accuracy 10^{-6}. The solver is based on modified HLib library [7]. Geometrically, model is represented by a unitary cube (height, width and breadth = 1) with 5248 collocation nodes. A comparison of CPU time, memory capacity and error $\|\mathbf{b}-\mathbf{Ax}\|_2$ between the ACA-based solver with $N_{ind}=16$ and $\gamma = 2.5$ and a Direct LU solver is presented in Table 1.

Table 1: Performance of ACA-based solver compared with Direct (LU) solver.

Solver	ε_{ACA}	T_{CPU}, sec	M, Mb	$\|\mathbf{b}-\mathbf{Ax}\|_2$
Direct (LU)	-	3870	220	0.0
ACA-based	10^{-6}	9.28	76	0.000006
	10^{-5}	7.58	60	0.000023
	10^{-4}	6.1	46	0.000174
	10^{-3}	4.76	34	0.002076
	10^{-2}	3.84	23	0.023374
	10^{-1}	3.43	14	0.698741

As a conclusion, ACA-solver gives reasonable speedup and saves memory needed for storing LSE. The ACA-solution with $\varepsilon_{ACA} = 10^{-3}$ still has good precision and compared to the direct solution it needs 6 times less memory and can be obtained much faster. Figure 4 shows the comparison between the result x_i obtained with ACA-based and Direct LU solvers, in the horizontal and vertical axes, respectively.

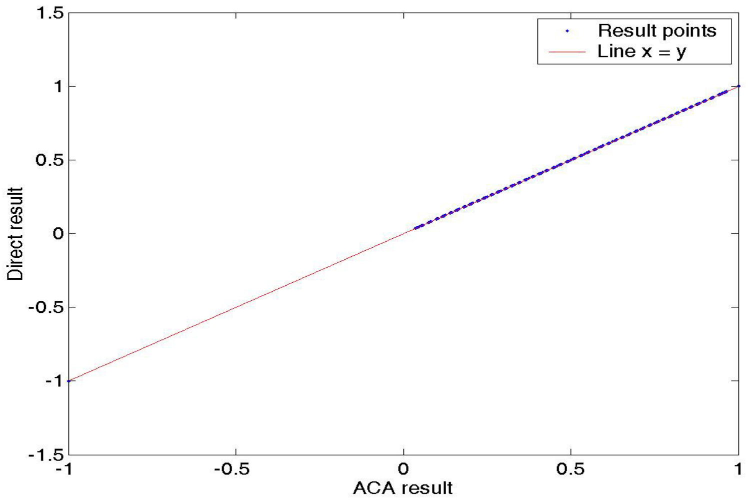

Figure 4: ACA- vs. Direct-solution comparison.

6 Conclusions

As a result of this work the main parameters that control the performance of an ACA-based solver for BEM have been explained and experimental dependencies have been established and cast into simple analytical expressions. A set of three possible cases of restrictions in computational resources have been established and on its basis appropriate recommendations regarding values of the parameters have been proposed. At the end, a practical toy example demonstrates the numerical properties of ACA-based solver for LSE and proves its efficiency.

Acknowledgements

The first author would like to thank the Wessex Institute of Technology for the support by scholarship and to express his gratitude to the Max-Planck Institute for their HLib library, which has been used in this work.

References

[1] Stefan Kurz, Oliver Rain and Sergej Rjasanow. *The Adaptive Cross-Approximation Technique for the 3-D Boundary-Element Method*. IEEE Transaction on Magnetics, Vol. 38, No. 2, March 2002

[2] Stefen Borm, Lars Grasedyck and Wolfgang Hackbusch. *Hierarchical Matrices*, April 2005

[3] M. Bebendorf and R. Grzibovski. *Accelerating Galerkin BEM for Linear Elasticity using Adaptive Cross Approximation*, May 2006

[4] Wolfgang Hackbusch, Lars Grasedyck and Steffen Borm. *An Introduction to Hierarchical matrices*. Proceedings of EQUADIFF 10, Prague, August 2001

[5] M. Bebendorf and S. Rjasanov. *Adaptive Low-Rank Approximation of Collocation Matrices*. Computing 70, 1-24, Springer-Verlag 2003

[6] Teresco James D, Karen D Devine, Joseph E Flaherty. *Partitioning and Dynamic Load Balancing for the Numerical Solution of Partial Differential Equations*. Numerical Solution of Partial Differential Equations on Parallel Computers, August 2005

[7] *Hlib library v.1.3*. Max Planck Institute for Mathematics in the Sciences. Inselstrasse 22-26, 04103 Leipzig, Germany