A Fast Block Low-Rank Dense Solver with Applications to Finite-Element Matrices

AmirHossein Aminfar[a,1,*], Sivaram Ambikasaran[b,2], Eric Darve[c,1]

[a] *496 Lomita Mall, Room 104, Stanford, CA, 94305*
[b] *Warren Weaver Hall, Room-1105A, 251, Mercer Street, New York, NY 10012*
[c] *496 Lomita Mall, Room 209, Stanford, CA, 94305*

Abstract

This article presents a fast solver for the dense "frontal" matrices that arise from the multifrontal sparse elimination process of 3D elliptic PDEs. The solver relies on the fact that these matrices can be efficiently represented as a hierarchically off-diagonal low-rank (HODLR) matrix. To construct the low-rank approximation of the off-diagonal blocks, we propose a new pseudo-skeleton scheme, the boundary distance low-rank approximation, that picks rows and columns based on the location of their corresponding vertices in the sparse matrix graph. We compare this new low-rank approximation method to the adaptive cross approximation (ACA) algorithm and show that it achieves betters speedup specially for unstructured meshes. Using the HODLR direct solver as a preconditioner (with a low tolerance) to the GMRES iterative scheme, we can reach machine accuracy much faster than a conventional LU solver. Numerical benchmarks are provided for frontal matrices arising from 3D finite element problems corresponding to a wide range of applications.

Keywords: Fast direct solvers, Iterative solvers, Numerical linear algebra, Hierarchically off-diagonal low-rank matrices, multifrontal elimination, Adaptive cross approximation.

1. Introduction

In many engineering applications, solving large finite element systems is of great significance. Consider the system

$$Ax = b$$

arising from the finite element discretization of an elliptic PDE, where $A \in \mathbb{R}^{N \times N}$ is a sparse matrix with a symmetric pattern. In many practical applications, the matrix A might be ill-conditioned and thus, challenging for iterative methods. On the other hand, conventional direct solver algorithms, while being robust in handling ill-conditioned matrices, are computationally expensive ($\mathcal{O}(N^{1.5})$ for 2D meshes and $\mathcal{O}(N^2)$ for 3D meshes). Since one of the main bottlenecks in the direct multifrontal solve process is the high computational cost of solving dense frontal matrices, we mainly focus on solving these matrices in this article. Our goal is to build an iterative solver, which utilizes a fast direct solver as a preconditioner for the dense frontal matrices. The direct solver in this scheme acts as a highly accurate

*Corresponding author. +1 650-644-7624
Email address: aminfar@stanford.edu (AmirHossein Aminfar)
[1]Mechanical Engineering Department, Stanford University
[2]Courant Institute of Mathematical Sciences, New York University

pre-conditioner. This approach combines the advantages of the iterative and direct solve algorithms, i.e., it is fast, accurate and robust in handling ill-conditioned matrices.

To be consistent with our previous work, we adopt the notations used in [3]. We should also mention that 'n' refers to the size of dense matrices and 'N' refers to the size of sparse matrices (e.g., number of degrees of freedom in a finite-element mesh).

In the next section, we review the previous literature on both dense structured solvers and sparse multifrontal solvers. We then introduce a hierarchical off-diagonal low-rank (from now on abbreviated as HODLR) direct solver in Section 4. In Section 5, we introduce the boundary distance low-rank (BDLR) algorithm as a robust low-rank approximation scheme for representing the off-diagonal blocks of the frontal matrices. Section 6 discusses the application of the iterative solver with a fast HODLR direct solver preconditioner to the sparse multifrontal solve process and demonstrates the solver for a variety of 3D meshes. We also show an application in combination with the FETI-DP method [21], which is a family of domain decomposition algorithms to accelerate finite-element analysis on parallel computers.

2. Previous Work

2.1. Fast Direct Solvers for Dense Hierarchical Matrices

Hierarchical matrices are data sparse representation of a certain class of dense matrices. This representation relies on the fact that these matrices can be sub-divided into a hierarchy of smaller block matrices, and certain sub-blocks (based on the admissibility criterion) can be efficiently represented as a low-rank matrix. We refer the readers to [29, 33, 27, 30, 12, 15, 13] for more details. These matrices were introduced in the context of integral equations [29, 33, 62, 43] arising out of elliptic partial differential equations and potential theory. Subsequently, it has also been observed that dense fill-ins in finite element matrices [60], radial basis function interpolation [3], kernel density estimation in machine learning, covariance structure in statistic models [16], Bayesian inversion [3, 5, 6], Kalman filtering [45], and Gaussian processes [4], can also be efficiently represented as data-sparse hierarchical matrices. Broadly speaking, these matrices can be grouped into two general categories based on the admissibility criterion: (i) Strong admissibility: sub-blocks that correspond to the interaction between well-separated clusters are low-rank; (ii) Weak admissibility: sub-blocks corresponding to non-overlapping interactions are low-rank. Ambikasaran [1] provides a detailed description of these different hierarchical structures.

We review some of the previously developed structured dense solvers for hierarchical matrices and discuss them in relation to our work. Hackbusch [29, 27] introduced the concept of $\mathcal{H}$-matrices, which are the most general class of hierarchical matrices with the strong admissibility criterion [29, 27, 30, 32, 31, 33, 34, 9, 10, 12]. Contrary to the HODLR matrix structure, in which the off-diagonal blocks are low-rank, in $\mathcal{H}$-matrices, the off-diagonal blocks are further decomposed into low-rank and full-rank blocks. Thus, the rank can be kept small. In HODLR, we make a single low-rank approximation for the off-diagonal blocks and the rank is larger as a result. Hence, the HODLR structure makes for a much simpler representation and is often used because of its simplicity compared to the $\mathcal{H}$-matrix structure. Hackbusch [27] suggests a recursive block low-rank factorization scheme for $\mathcal{H}$-matrices. This method is based on the idea that all the dense matrix algebra (matrix multiplication and matrix addition) can be replaced by $\mathcal{H}$-matrix algebra. As a result, the inverse of an $\mathcal{H}$-matrix can also be approximated as an $\mathcal{H}$-matrix itself. This results in a computational complexity of $\mathcal{O}(n \log^2(n))$ for an $\mathcal{H}$-matrix factorization.

We note that the approach in this paper is based on the Woodbury matrix identity. It is therefore different from the algorithm in Hackbusch [27] for example. The latter is based on a block LU factorization, while the Woodbury identity reduces the global solve to block diagonal solves followed by a correction update.

The HODLR matrix structure is the most general off-diagonal low-rank structure with weak admissibility. Solvers for this matrix class have a computational cost of $\mathcal{O}(n \log^2 n)$. In an HODLR matrix, the off-diagonal low-rank bases do not have a nested structure across different levels [3]. The HSS matrix is an HODLR matrix but, in addition, has a nested off-diagonal low-rank structure. Solvers for the HSS matrices have an $\mathcal{O}(n)$ complexity [61, 14].

Martinsson and Rokhlin [51] discuss an $\mathcal{O}(n)$ direct solver for boundary integral equations based on the HSS structure. Their method is based on the fact that for a matrix of rank r, there exists a well-conditioned column operation, which leaves r columns unchanged and sets the remaining columns to zero. Using this idea, they derive a two-sided compressed factorization of the inverse of the HSS matrix. Their generic algorithm requires $\mathcal{O}(n^2)$ operations to construct the inverse. However, they accelerate their algorithm to $\mathcal{O}(n \log^\kappa(n))$ when applied to two-dimensional contour integral equations.

Chandrasekaran et al. [15] present a fast $\mathcal{O}(n)$ direct solver for HSS matrices. In their article, they construct an implicit ULV^H factorization of an HSS matrix, where U and V are unitary matrices, L is a lower triangular matrix and ^{H} is the transpose conjugate operator. Their method is based on the Woodbury matrix identity and the fact that for a low-rank representation of the form UBV^H, where U and V are thin matrices with r columns, there exists a unitary transformation Q, in which only the last r rows of QU are nonzero. They use this observation to recursively solve the linear system of equations. Since this method requires constructing an HSS tree, the authors suggest an algorithm that uses the SVD or the rank revealing QR decomposition, recursively, to construct the HSS tree in $\mathcal{O}(n^2)$ time.

Gillman et al. [25] discuss an $\mathcal{O}(n)$ algorithm for directly solving integral equations in one-dimensional domains. The algorithm relies on applying the Sherman-Morrison-Woodbury formula (see for example [3]) recursively to an HSS tree structure to achieve $\mathcal{O}(r^2 n)$ solve complexity, where r is the rank of the off diagonal blocks in the HSS matrix. They also describe an $\mathcal{O}(r^2 n)$ algorithm for constructing an HSS representation of the matrix resulting from a Nyström discretization of a boundary integral equation.

Ho and Greengard [38] present a fast direct solver for HSS matrices. They use the interpolative decomposition (ID) algorithm which is based on matrix skeletonization (see for example [17]) with random sampling to obtain the low-rank representations of the off-diagonal blocks. The idea of using skeletonization in matrix factorization was first introduced by Gu and Eisenstat [28] in constructing a strong rank-revealing QR factorization. Other than [38], many fast direct solvers use interpolative decomposition (ID) for low-rank approximation (see for example [51] and [25]). The computational complexity of the low-rank approximation algorithm used in [38] is $\mathcal{O}(mn \log r + r^2 n)$ for a matrix $K \in \mathbb{R}^{m \times n}$. After obtaining the hierarchical matrix representation of the original dense matrix, new variables and equations are introduced into the system of equations. Finally, all equations are assembled into an extended sparse matrix and a conventional sparse solver is used to factorize the sparse matrix. This method has a computational complexity of $\mathcal{O}(n)$ for both the pre-computation and solution phases for boundary integral equations in 2D, while in 3D, these phases cost $\mathcal{O}(n^{1.5})$ and $\mathcal{O}(n \log(n))$ respectively.

Kong et al. [42] have developed an $\mathcal{O}(n^2)$ dense solver for HODLR matrices. Similar to [51], they accelerate their algorithm to $\mathcal{O}(n \log^2(n))$, when applied to boundary integral equations. Their method uses the Sherman-Morrison-Woodbury formula to construct a

one-sided hierarchical factorization of the inverse of these matrices, in which each factor is a block diagonal matrix. The low-rank approximation scheme in their paper is based on the rank revealing QR algorithm. The authors use the pivoted Gram-Schmidt algorithm to obtain r orthogonal basis vectors for the low-rank matrix in question. For a matrix $K \in \mathbb{R}^{m \times n}$ with rank r, this low-rank approximation scheme requires $\mathcal{O}(mnr)$ operations. Then, they use a randomized algorithm from [57] to accelerate their low-rank approximation scheme. This accelerated low-rank approximation algorithm costs $\mathcal{O}(mn \log(l+lnr))$ in the general case where $r < l < \min(m, n)$.

Ambikasaran and Darve [3] present an $\mathcal{O}(n \log^2(n))$ solver for HODLR matrices and an $\mathcal{O}(n \log(n))$ solver for p-HSS matrices. This approach differs from the approach mentioned in [42] in the fact that, while [42] constructs the inverse, [3] constructs a factorization of the matrix. Each factor in this factorization scheme is a block diagonal matrix with each block being a low-rank perturbation of the identity matrix. The authors then use the Sherman-Morrison-Woodbury formula to invert each block in the block diagonal factors. The article uses the Chebyshev low-rank approximation scheme to factorize the off-diagonal blocks.

As mentioned above, solvers for the HSS matrix structure have the lowest computational complexity — $\mathcal{O}(r^2 n)$, r being the rank of approximation — among other hierarchically off-diagonal low-rank matrix structures. While the HSS structure is attractive, the nested structure makes it more complicated and more difficult to work with, compared to the simpler HODLR structure. Furthermore, the off-diagonal rank increases from root to leaves in the HSS tree, whereas the off-diagonal ranks at each level are independent from each other in the HODLR structure. This often leads to lower average off-diagonal rank in the HODLR structure compared to HSS.

A point worth mentioning is that the solver discussed in the current article relies on purely algebraic technique (instead of analytic or geometry based techniques) to construct the low-rank approximation of the off-diagonal blocks. Analytic low-rank approximation techniques like the Chebyshev low-rank approximation, multipole expansions, etc., are only applicable when the matrix definition involves an analytical kernel function.

In this article, we propose a boundary distance low-rank approximation (from now on abbreviated as BDLR), which relies on the underlying sparse matrix graph to choose the desired rows and columns in constructing a low-rank representation. We also compare with the adaptive cross approximation algorithm [53] (from now on abbreviated as ACA), which is also a purely algebraic scheme to construct low-rank approximations of the off-diagonal blocks.

Due to its black-box nature, the solver can handle a wide range of dense matrices arising from boundary integral equations, covariance matrices in statistics, frontal matrices arising in the context of finite-element matrices, etc. Table 1 summarizes the dense solver algorithms mentioned above.

2.2. Fast Direct Solvers for Sparse Matrices

As mentioned in Section 2.1, we are interested in accelerating the direct solve process for finite-element matrices. In this article, we focus on the finite-element discretization of elliptic PDEs. One common way of factorizing such matrices is using the sparse Cholesky factorization. The efficiency of this algorithm strongly depends on the ordering of mesh nodes [56]. Sparse Cholesky factorization takes $\mathcal{O}(N^2)$ flops in 2D with a typical row-wise or column-wise mesh ordering, where N is the number of degrees of freedom [60]. The most efficient method for solving such matrices is the multifrontal method with nested dissec-

Article	Matrix Class	Factorization	Application
Hackbusch [29, 27]	$\mathcal{H}$	Recursive block factorization of the matrix	BEM integral operators
Martinsson and Rokhlin [51]	HSS	Two sided compressed factorization of the inverse	2D boundary integral equations
Chandrasekaran et al. [15]	HSS	ULV^H factorization of the matrix	Radial basis function matrices
Gillman et al. [25]	HSS	Data sparse factorization of the inverse	1D integral equations with Nyström discretization
Ho and Greengard [38]	HSS	Factorization of the extended sparse system	2D and 3D boundary integral equations
Kong et al. [42]	HODLR	One sided hierarchical factorization of the inverse	Boundary integral equations
Ambikasaran and Darve [3]	HODLR, p-HSS	Block-diagonal factorization of the matrix	Interpolation using radial basis functions
This article	HODLR	Recursive block LU factorization of the matrix	Finite-element matrices

Table 1: Summary of fast dense structured solvers.

tion [23], which takes $\mathcal{O}(N^{1.5})$ flops for two-dimensional and $\mathcal{O}(N^2)$ for three dimensional meshes [56].

The multifrontal method was originally introduced by Duff & Reid [20], George [23] and Liu [47], as an extension to the frontal method of Irons [40]. In this algorithm, the overall factorization is done by factorizing smaller dense frontal matrices [46]. For each node or super-node in the elimination tree, the frontal matrix is obtained using an update process called the "extend-add" process, which involves updates from the previously eliminated nodes.

Martinsson [49] uses a spiral elimination approach along with HSS compression of Schur complements to achieve $\mathcal{O}(N \log^2 N)$ time complexity. This approach is not based on the multifrontal method and requires a mesh that can be partitioned into concentric annuli.

Gillman and Martinsson [24] proposed an accelerated nested dissection algorithm for obtaining the Dirichlet-to-Neumann operator associated with a 2D elliptic boundary value problem. The authors approximate the Schur complements that appear in the elimination process as hierarchically block separable (HBS) matrices, a structure similar to HSS matrices. Using this matrix structure, they are able to obtain the Dirichlet-to-Neumann operator with a cost of $\mathcal{O}(N)$ compared to $\mathcal{O}(N^{1.5})$ of the conventional multifrontal method with nested dissection.

There have been some recent efforts to reduce the computational cost of the multifrontal method with nested dissection. Xia et al. [60] observed that frontal and update matrices in the multifrontal elimination process can be approximated with hierarchically semi-separable (HSS) matrices. The authors develop a structured extend-add process to facilitate the formation of the frontal matrices using the HSS structure. Next, they perform a structured dense Cholesky factorization on the obtained frontal matrix. The authors use the algorithm in [14] to compute the explicit factorizations of HSS matrices. Using this procedure, they are able to achieve nearly linear time complexity for 2D meshes. However, only regular well shaped meshes in 2D are considered in the article. Schmitz and Ying. [56] extend the

approach of [60] to a more general setting of unstructured and adaptive grids in 2D.

Xia [58] introduced an efficient multifrontal factorization for general sparse matrices. The author approximates the frontal matrices using the HSS structure and introduces the concept of reduced HSS matrices that reduce the computational cost of operation on HSS matrices. For simplicity, this approach keeps the update matrices as dense matrices which leads to high memory consumption for large sparse matrices. Xia [59] introduces a new algorithm that overcomes this deficiency by randomization. That is, instead of passing dense update matrices along the elimination tree, this approach passes a skinny randomized matrix vector product. In addition to saving memory, this approach only requires skinny extend adds (extend adds on all rows and only a subset of columns) which leads to improvements in efficiency. This method is based on the work of Martinsson [50] which provides an algorithm for constructing HSS matrices using randomized matrix vector products.

Amestoy et al. [7] introduce a new low-rank matrix format called the Block Low-Rank (BLR) structure, a flat, non-hierarchical block matrix structure, for representing frontal matrices obtained in the multifrontal elimination process. The authors show that BLR is a good alternative to hierarchical structures like $\mathcal{H}$ and HSS matrices in terms of storage costs, flop count and parallelization for representing frontal matrices. The article demonstrates that the BLR format reduces the flop count and storage requirements for factorizing frontal matrices arising from a variety of large matrices coming from different physics applications. The solver uses conventional extend-add operations as implemented in MUMPS [8].

The approach presented here is based on the multifrontal method [46]. It does not require constructing and maintaining HSS trees and can be applied to any mesh structure. Our method is based on the observation that the frontal matrices obtained during the multifrontal elimination process have an HODLR structure. This observation was also made by [60].

In order to factorize (eliminate) these frontal matrices, we present a dense HODLR structured solver. If the rank r is $\mathcal{O}(1)$ (that is function of ϵ only), the algorithm has a computational cost of $\mathcal{O}(r^2 n \log^2 n)$ for an $n \times n$ frontal matrix. When solving 3D PDEs, we typically have that $r \in \mathcal{O}(n^{1/2})$. In that case, the computational cost is $\mathcal{O}(r^2 n)$, where r is the largest rank found, at the top of the tree. For 3D PDEs, the final cost is therefore $\mathcal{O}(n^2)$. It is the same as what is reported for HSS in [59] (see Table 4.1, p. 219).

We will benchmark the structured elimination (solve) process for frontal matrices corresponding to separators at various levels of the sparse elimination tree, for many different types of sparse matrices. It is worth mentioning that contrary to previous works which have mainly benchmarked matrices in the University of Florida Sparse Matrix Collections [18], we focus on frontal matrices arising from large and complicated mesh structures. These matrices are often very ill-conditioned and cannot be solved using traditional iterative techniques like GMRES [54] with diagonal preconditioning. Our benchmarks show that obtaining a good preconditioner for unstructured meshes is significantly harder compared to structured meshes. Furthermore, solving 3D problems is an order of magnitude more difficult than 2D problems as the off-diagonal rank is significantly higher in 3D. Hence, this article mainly focuses on 3D meshes.

Table 2 shows a summary of various fast sparse matrix solvers in the literature.

3. An Iterative Solver with Direct Solver Preconditioning

In this paper, we investigate using a fast HODLR direct solver as a preconditioner to the GMRES [54] iterative scheme. In this case, we use a relatively low accuracy for the

Article	Methodology	Test Cases & Application
Martinsson [49]	HSS compression and spiral elimination	Meshes that can be partitioned into concentric annuli
Gillman and Martinsson [24]	Approximating Schur complements as HBS matrices and using HBS algebra.	2D elliptic boundary value problems discretized using a 5 point stencil on a regular square grid.
Xia et al. [60]	HSS approximation of frontal matrices and structured extend-add	2D structured meshes
Schmitz and Ying [56]	Modified [60] to accommodate adaptive and unstructured grids	2D adaptive and unstructured meshes that roughly follow the pattern of a regular mesh
Xia [58]	Introduction of reduced HSS matrices that reduce the operation cost on HSS matrices. For simplifications, the update matrices are kept as dense matrices.	Helmholtz Equation in 2D and University of Florida Sparse Matrix Collections [18]
Xia [59]	HSS compression using randomization techniques in [50]. Passing randomized matrix vector products instead of dense update matrices and performing skinny extend-add operations.	Helmholtz Equation in 2D and University of Florida Sparse Matrix Collections [18]
Amestoy et al. [7]	BLR format for representing frontal matrices. No discussion of BLR extend-add process.	Large matrices coming from different physics applications

Table 2: Summary of fast sparse direct solvers.

direct solver.

We will show that this approach is much faster than both a conventional LU solver and a high accuracy direct HODLR solver. We should also mention that this preconditioning method can be applied to any iterative solver (conjugate gradient (CG) [37], etc..).

4. A Fast Direct Solver for HODLR Matrices

One bottleneck of sparse solvers is the factorization of the dense frontal matrices that appear during the multifrontal elimination process. To accelerate the factorization of dense frontal matrices, we approximate them as HODLR matrices. As mentioned in Section 2.1, HODLR matrices can be factorized in $\mathcal{O}(n \log^2 n)$ which is a significant improvement over conventional dense factorizations which typically scale as $\mathcal{O}(n^3)$.

4.1. HODLR Matrices

An HODLR matrix has low-rank off-diagonal blocks at multiple levels. As described in [3], a 2-level HODLR matrix, $K \in \mathbb{R}^{n \times n}$, can be written as shown in Equation (2):

$$K = \begin{bmatrix} K_1^{(1)} & U_1^{(1)}(V_{1,2}^{(1)})^T \\ U_2^{(1)}(V_{2,1}^{(1)})^T & K_2^{(1)} \end{bmatrix} \tag{1}$$

$$= \begin{bmatrix} \begin{bmatrix} K_1^{(2)} & U_1^{(2)}(V_{1,2}^{(2)})^T \\ U_2^{(2)}(V_{2,1}^{(2)})^T & K_2^{(2)} \end{bmatrix} & U_1^{(1)}(V_{1,2}^{(1)})^T \\ U_2^{(1)}(V_{2,1}^{(1)})^T & \begin{bmatrix} K_3^{(2)} & (U_3^{(2)})^T(V_{3,4}^{(2)})^T \\ U_4^{(2)}(V_{4,3}^{(1)})^T & K_4^{(2)} \end{bmatrix} \end{bmatrix} \tag{2}$$

where for a p-level HODLR matrix, $K_i^{(p)} \in \mathbb{R}^{n/2^p \times n/2^p}$, $U_{2i-1}^{(p)}, U_{2i}^{(p)}, V_{2i-1,2i}^{(p)}, V_{2i,2i-1}^{(p)} \in \mathbb{R}^{n/2^p \times r}$ and $r \ll n$. Further nested compression of the off-diagonal blocks will lead to HSS structures [3].

4.2. Solver Derivation and Algorithm

Contrary to the method introduced by Hackbusch [29] which utilizes sequential block LU factorization, the HODLR direct solve algorithm presented in this section is based on the Woodbury matrix identity (see for example [35, 3]). Although we do not use the formula explicitly, we perform the exact same operations. Looking at Equation (4), our method assumes that both diagonal blocks are nonsingular and factorizes them independently. However, Hackbusch [29] only assumes that top diagonal block is invertible and factorizes the top diagonal block first. He then constructs the remaining Schur complement and continues on with the factorization. In comparing the two methods, one can see that because of the independent factorization of the diagonal blocks, the method presented in this section is better suited to parallel implementations.

Consider the following linear equation:

$$Kx = F \tag{3}$$

where $K \in \mathbb{R}^{n \times n}$ is an HODLR matrix and x, $F \in \mathbb{R}^{n \times s}$. Now let's write K as a one-level HODLR matrix and rewrite Equation (3) :

$$K = \begin{bmatrix} K_1^{(1)} & U_1^{(1)} V_{1,2}^{(1)T} \\ U_2^{(1)} V_{2,1}^{(1)T} & K_2^{(1)} \end{bmatrix} \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \end{bmatrix} \tag{4}$$

where $x_i^{(1)}, F_i^{(1)} \in \mathbb{R}^{(\frac{n}{2} \times s)}$. We now introduce two new variables $y_1^{(1)}$ and $y_2^{(1)}$:

$$y_1^{(1)} = V_{2,1}^{(1)T} x_1^{(1)} \tag{5}$$

$$y_2^{(1)} = V_{1,2}^{(1)T} x_2^{(1)} \tag{6}$$

Rearranging (4), we have:

$$\underbrace{\begin{bmatrix} K_1^{(1)} & 0 & 0 & U_1^{(1)} \\ 0 & K_2^{(1)} & U_2^{(1)} & 0 \\ -V_{2,1}^{(1)T} & 0 & I & 0 \\ 0 & -V_{1,2}^{(1)T} & 0 & I \end{bmatrix}}_{\widehat{K}} \underbrace{\begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ y_1^{(1)} \\ y_2^{(1)} \end{bmatrix}}_{\widehat{x}} = \underbrace{\begin{bmatrix} F_1 \\ F_2 \\ 0 \\ 0 \end{bmatrix}}_{\widehat{F}} \tag{7}$$

We now factorize the top diagonal block of $\widehat{K}$ which consists of $K_1^{(1)}$ and $K_2^{(1)}$. Since this subblock of $\widehat{K}$ is a block diagonal matrix, this means that we only need to factorize $K_1^{(1)}$ and $K_2^{(1)}$. After eliminating the top off diagonal block, we are left with the Schur complement:

$$S^{(1)} = \begin{bmatrix} I & V_{2,1}^{(1)T} (K_1^{(1)})^{-1} U_1^{(1)} \\ V_{1,2}^{(1)T} (K_2^{(1)})^{-1} U_2^{(1)} & I \end{bmatrix} \tag{8}$$

All we have to do now, is to solve the Schur complement:

$$S^{(1)} \begin{bmatrix} y_1^{(1)} \\ y_2^{(1)} \end{bmatrix} = \begin{bmatrix} V_{2,1}^{(1)^T} (K_1^{(1)})^{-1} F_1 \\ V_{1,2}^{(1)^T} (K_2^{(1)})^{-1} F_2 \end{bmatrix} \tag{9}$$

At this point, we can write $x_1^{(1)}$ and $x_2^{(1)}$ in terms of $(K_1^{(1)})^{-1}$ and $(K_2^{(1)})^{-1}$:

$$\begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \end{bmatrix} = \begin{bmatrix} (K_1^{(1)})^{-1} & 0 \\ 0 & (K_2^{(1)})^{-1} \end{bmatrix} \begin{bmatrix} F_1 - U_1^{(1)} y_2^{(1)} \\ F_2 - U_2^{(1)} y_1^{(1)} \end{bmatrix} \tag{10}$$

Since, both $K_1^{(1)}$ and $K_2^{(1)}$ are HODLR matrices, we can apply the same procedure for factorizing them. Thus, we have arrived at a recursive algorithm for solving (7). The factorization step corresponds to the computation and storage of all the terms that are independent of the right hand side (i.e., the Schur complements at all levels).

4.3. Algorithm Summary

We now summarize the recursive HODLR direct solver algorithm. For a matrix such as $K \in \mathbb{R}^{n \times n}$, we have to carry out the following procedure at each recursion level (p) for all $1 \leq i \leq 2^p$:

4.3.1. Factorize

1. Find the low-rank approximation of the off-diagonal blocks ($U_{2i-1}^{(p)}$, $U_{2i}^{(p)}$, $V_{2i-1,2i}^{(p)}$, $V_{2i,2i-1}^{(p)}$).

2. Define $Z_1^0 = 0$. For each level p, starting at the top level ($p = 0$), let:

$$\begin{bmatrix} Z_{2i-1}^{(p+1)} \\ Z_{2i}^{(p+1)} \end{bmatrix} = \begin{bmatrix} U_{2i-1}^{(p+1)} & Z_i^{(p)} \\ U_{2i}^{(p+1)} & \end{bmatrix} \tag{11}$$

   In the equation above, on the right-hand side, we are vertically concatenating two matrices to form a matrix at level $p + 1$.

3. Recursively solve the following equations:

$$\begin{bmatrix} d_{2i-1}^{(p+1)} & c_{2i-1}^{(p+1)} \end{bmatrix} = (K_{2i-1}^{(p+1)})^{-1} Z_{2i-1}^{(p+1)} \tag{12}$$

$$\begin{bmatrix} d_{2i}^{(p+1)} & c_{2i}^{(p+1)} \end{bmatrix} = (K_{2i}^{(p+1)})^{-1} Z_{2i}^{(p+1)} \tag{13}$$

   where $d^{(p+1)}$ and $c^{(p+1)}$ correspond to the $U^{(p+1)}$ and $Z^{(p)}$ portion of the right hand sides respectively.

4. Obtain $S_i^{(p)}$, using Equations (8) and (9):

$$S_i^{(p)} = \begin{bmatrix} I & (V_{2i,2i-1}^{(p+1)})^T d_{2i-1}^{(p+1)} \\ (V_{2i-1,2i}^{(p+1)})^T d_{2i}^{(p+1)} & I \end{bmatrix} \tag{14}$$

5. Obtain $d_i^{(p)}$, $c_i^{(p)}$ for $p \geq 1$ using:

$$\begin{bmatrix} d_i^{(p)} & c_i^{(p)} \end{bmatrix} = \left(I - \begin{bmatrix} 0 & d_{2i-1}^{(p+1)} \\ d_{2i}^{(p+1)} & 0 \end{bmatrix} (S_i^{(p)})^{-1} \begin{bmatrix} V_{2i,2i-1}^{(p+1)\ T} & 0 \\ 0 & V_{2i-1,2i}^{(p+1)\ T} \end{bmatrix} \right) \begin{bmatrix} c_{2i-1}^{(p+1)} \\ c_{2i}^{(p+1)} \end{bmatrix} \tag{15}$$

4.3.2. Solve

1. Define $z_1^0 = F$. For each level p, starting at the top level ($p = 0$), let:

$$\begin{bmatrix} z_{2i-1}^{(p+1)} \\ z_{2i}^{(p+1)} \end{bmatrix} = z_i^p \tag{16}$$

2. Recursively solve the following equations:

$$x_{2i-1}^{(p+1)} = (K_{2i-1}^{(p+1)})^{-1} z_{2i-1}^{(p+1)} \tag{17}$$

$$x_{2i}^{(p+1)} = (K_{2i}^{(p+1)})^{-1} z_{2i}^{(p+1)} \tag{18}$$

3. Obtain $x_i^{(p)}$ for $p \geq 0$ using:

$$x_i^{(p)} = \left(I - \begin{bmatrix} 0 & d_{2i-1}^{(p+1)} \\ d_{2i}^{(p+1)} & 0 \end{bmatrix} (S_i^{(p)})^{-1} \begin{bmatrix} V_{2i,2i-1}^{(p+1)\ T} & 0 \\ 0 & V_{2i-1,2i}^{(p+1)\ T} \end{bmatrix} \right) \begin{bmatrix} x_{2i-1}^{(p+1)} \\ x_{2i}^{(p+1)} \end{bmatrix} \tag{19}$$

Note that $(S_i^{(p)})^{-1}$ was previously computed and this step is therefore only a series of matrix-matrix products. Hence, the computational cost is small compared to the previous factorization.

4.4. Solver Computational Cost

Assuming we use a fast ($\mathcal{O}(n)$) low-rank approximation scheme, the cost of constructing and storing an HODLR matrix is $\mathcal{O}(nr\log(n))$ [3], where r is the rank of approximation. Looking at the procedure described in Section 4.3, we can write the following:

$$C^{(p)}(r, s, n) = 2C^{(p+1)}\left(r, s + r, \frac{n}{2}\right) + \mathcal{O}(nr^2) + \mathcal{O}(nsr) \tag{20}$$

where $C^{(p)}(r, s, n)$ is the computational cost associated with solving an $n \times n$ HODLR matrix at level p with s right hand sides and off-diagonal blocks of rank r. Equation (20) suggests that the cost of solving a HODLR matrix at level p with s right hand sides is made up of three contributions. The first contribution is associated with solving the two diagonal blocks at the lower level $(p+1)$ with $s+r$ right hand sides. The second contribution comes from constructing the Schur complement $S^{(p)}$ (Equation (8)) and the third contribution is the cost of constructing the right hand side of Equation (9). Writing Equation (20) as a sum, we have:

$$C^{(0)}(r, s, n) = \sum_{p=1}^{\log\left(\frac{n}{r}\right)} \mathcal{O}(pnr^2 + nsr) \tag{21}$$

If the off-diagonal rank is constant throughout various levels in the HODLR tree, the computational cost of the algorithm is $\mathcal{O}(r^2 n \log^2(n))$ according to Equation (21).

However, in many practical cases, the rank decays from root to leaves in the HODLR tree. Assume we can approximate r as $O(n_p^{1/2})$ where n_p is the size of a block at level p. Then, we have: $r_p = O(\frac{r_1}{2^{p/2}})$, where r_1 is the rank at the top level. According to

Equation (20), the total computational cost involves two sums:

$$\sum_{p=1}^{\log\left(\frac{n}{r}\right)} r_p^2 = O(r_1^2)$$

$$\sum_{p=1}^{\log\left(\frac{n}{r}\right)} r_p \sum_{q=2}^{p}(s+r_q) = O(\sum_{p=1}^{\log\left(\frac{n}{r}\right)} r_p(s+r_1)) = O(r_1(s+r_1))$$

Note in particular that the second sum is $O(r_1^2)$ instead of $O(r^2 \log^2 n)$. Finally:

$$C^{(0)}(r,s,n) = \mathcal{O}(nr^2) \tag{22}$$

This result shows that in cases where the off-diagonal rank is decreasing, HODLR solvers can become very efficient and can compete with HSS solvers.

5. Low-Rank Approximation Schemes

In this section, we discuss the various low-rank approximations schemes used for obtaining a low-rank representation of the off-diagonal blocks of the HODLR matrices in consideration. Although a variety of low-rank approximation algorithms (SVD, rank revealing LU, rank revealing QR, randomized algorithms, etc) are available, we require a scheme that has a computational cost of $\mathcal{O}(rn)$ where r is the rank of approximation and n is the size of the matrix. In the context of this work, we prefer not to use randomized SVD methods since matrix vector products are not as efficient for HODLR matrices as they are for HSS structures. This limits our choices to methods like Chebyshev, partial pivoting ACA (Section 5.1) and the pseudo-skeleton low-rank approximation algorithm (Section 5.3). Each of these methods has certain drawbacks:

- The Chebyshev low-rank approximation algorithm is only suited to cases dealing with interaction of points via smooth kernels.
- The partial pivoting ACA algorithm works well when the leverage score of the matrix [48] is uniform. That is, all rows and columns have fairly the same importance when constructing the low-rank approximation. However, in cases where certain rows or columns play a special role and are critical to include in the low-rank approximation, ACA might fail to properly identify them, resulting in an inaccurate low-rank approximation.
- The accuracy of the pseudo-skeleton low-rank approximation scheme strongly depends on the method used for selecting rows and columns.

In order to construct a fast and robust low-rank approximation scheme, we introduce a method for selecting rows and columns in the pseudo-skeleton low-rank approximation algorithm. We call this new method the boundary distance low-rank approximation scheme (BDLR).

5.1. ACA Low-Rank Approximation

We use the ACA algorithm with partial pivoting as described by Rjasanow [53]. This algorithm is an algebraic low-rank approximation scheme and works on any dense matrix without any prior knowledge of the matrix. Both full pivoting and partial pivoting ACA search the matrix or the remaining Schur complement for the largest entry and use this entry as the pivot. The full pivoting algorithm, similar to rank revealing LU, scans all the matrix entries. Partial pivoting ACA avoids this expensive search by looking at the largest entry in a single row/column at each step.

The partial pivoting ACA algorithm has a cost of $\mathcal{O}(r(m+n))$, for a matrix $A \in \mathbb{R}^{m\times n}$ [53], where r is the rank of approximation.

5.2. Randomized Algorithms

Randomized algorithms as described by [55, 36, 19, 22] arrive at a low-rank approximation of matrix A by forming a lower dimensional matrix Y obtained from sampling rows and/or columns of the original matrix or by applying random projections to matrix A. They then obtain the orthonormal basis Q for the range of Y and approximate A as:

$$A \approx QQ^T A \tag{23}$$

For a matrix of size $n \times n$, and without a fast matrix-vector product, these methods have a computational cost of $\mathcal{O}(n^2)$. Otherwise, the cost can be brought down to $\mathcal{O}(n)$ or $\mathcal{O}(n \log n)$.

5.3. Pseudo-Skeleton and Boundary Distance Low-Rank Approximation

In order to construct a fast and accurate solver, we need an accurate and robust method to construct low-rank approximations.

As we will show, BDLR is very robust and leads to accurate low-rank approximations. It works well in problems where the matrix can be related to a Green's function. (This is true for all linear PDE problems. Note that the Green's function needs to be smooth, with a singularity at the origin). In that case, large entries correspond to points close in space, which we associate as a simplification to nodes in the graph that are connected by few edges. Although this is a simple heuristic, it worked very well in our examples and allowed us to efficiently form accurate low-rank approximations.

The BDLR algorithm is a row and column selection algorithm in the pseudo-skeleton low-rank approximation scheme. The pseudo-skeleton algorithm allows us to construct a low-rank approximation of a matrix by choosing a subset of rows and columns of that matrix. As mentioned in [26], for a low-rank matrix A, if we pick a set of row indices ($i \in I = \{i_1, ..., i_r\}$) and a set of column indices ($j \in J = \{j_1, ..., j_r\}$) and define matrices C and R such that :

$$R = A(I, :) \tag{24}$$

$$C = A(:, J) \tag{25}$$

Then, we can approximate A to be :

$$A \approx C\widehat{A}^{-1}R \tag{26}$$

where $\widehat{A} = A(I, J)$. If $\widehat{A}$ is not a square matrix or rank deficient, the Moore-Penrose pseudoinverse is needed for $\widehat{A}^{-1}$. In order to achieve a certain accuracy, one can increase

the number of chosen rows and columns until the desired accuracy is reached. To monitor the error in the scheme, we pick rows and columns that are not in the set of rows and columns already chosen for low-rank approximation. We then monitor the relative Frobenius norm error on these rows and columns and increase the rank of the approximation until the relative Frobenius norm error falls below a certain tolerance.

For a rank r pseudo-skeleton low-rank approximation, the inversion of $\widehat{A}$ has a computational cost of $\mathcal{O}(r^3)$. Monitoring the error has a computational cost of $\mathcal{O}(mr + nr - r^2)$ for $A \in \mathbb{R}^{m \times n}$. Thus, this method has an asymptotic complexity of $\mathcal{O}(nr)$.

As mentioned in Section 1, we are predominantly interested in solving dense frontal matrices arising from the multifrontal elimination process of sparse finite-element matrices. In this case, every frontal matrix has a corresponding sparse matrix, which is a diagonal subblock of the original finite-element matrix. This sparse matrix describes a graph that has rows and columns of the dense matrix as its vertices and the edges in this graph correspond to nonzero entries in the sparse matrix and describe the connection between these points. We use this graph in constructing the low-rank approximation of the off-diagonal blocks.

Entries in dense matrix blocks that correspond to FEM or BEM applications can be related to the inverse of a Green's function. The Green's function is large at short distances. We have a similar behavior for our dense blocks. Hence, we want to identify row/column pairs corresponding to large entries. These correspond to nodes in the graph that are close, that is connected by few edges. Therefore we use the distance between a row vertex in the graph and the column vertex set (e.g., if the vertex corresponds to a row, we consider the distance to the set of vertices associated with the columns, and vice versa) as a good criterion to determine whether to pick a row/column or not.

For a set of row (column) vertices, we define the boundary vertices as the subset of vertices for which there exists an edge in the interaction graph connecting them to a vertex in the column (row) set. Figure 1(a) shows an example of a matrix which corresponds to the interactions of a set of row points with a set of column points. In this particular example, the blue vertices are the boundary vertices. That is, they are the vertices closest to the boundary between the row and column set of points.

Now that we have defined the boundary nodes, we can designate an index d for every vertex in the row (column) set. This index is defined as the distance of a vertex to the vertices in the boundary set. In order to construct the low-rank approximation, we choose rows and columns based on their d index value. That is, we first choose rows (columns) that are in the boundary set ($d = 0$). We then add rows (columns) with a distance of one to the boundary ($d = 1$). For example, in Figure 1(a), the green nodes are labeled ($d = 1$) as they are separated from the blue boundary nodes ($d = 0$) with only one edge. We continue adding points based on the d index, until we reach the desired accuracy. Figure 1(b) shows that the BDLR algorithm approximates the interaction of a set of row and column nodes with the interaction of the ones that are closest to the boundary (interaction of blue nodes).

As mentioned above, calculating the pseudo skeleton low-rank approximation requires us to calculate the pseudoinverse of $\widehat{A}$. For the BDLR algorithm, instead of using the SVD for calculating the pseudoinverse ($\widehat{A}^{-1}$), we use a full pivoting LU factorization, which is slightly cheaper:

$$\widehat{A} = P^{-1}LUQ^{-1} \tag{27}$$

where P and Q are permutation matrices. Let r be the rank of $\widehat{A}$. Define $\widetilde{R}$ and $\widetilde{C}$ as:

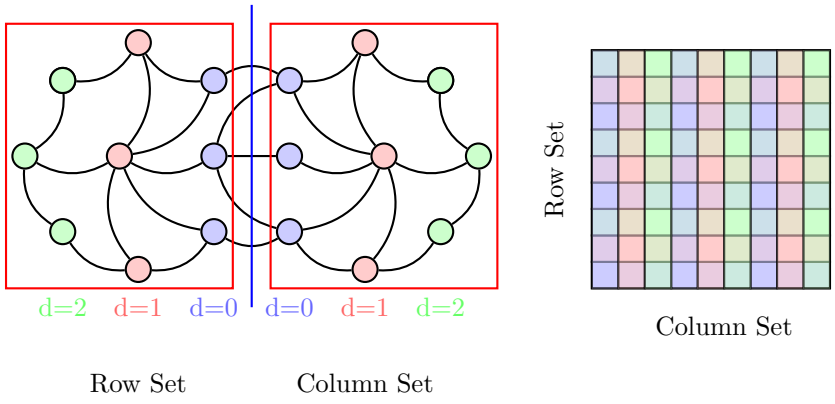
$$\widetilde{C} = (CQ)(:, 1:r)(U(1:r, 1:r))^{-1} \tag{28}$$

$$\widetilde{R} = (L(1:r, 1:r))^{-1}(PR)(1:r, :) \tag{29}$$

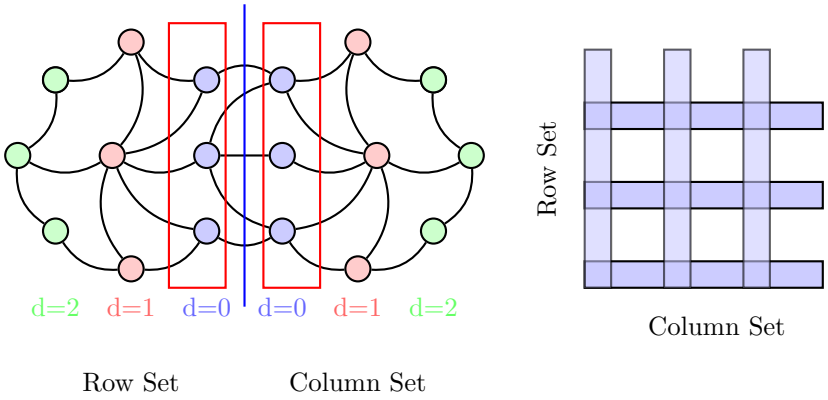
where C and R are the subset of columns and rows we have picked using the BDLR scheme. We then have:

$$A \approx \widetilde{C}\ \widetilde{R} \tag{30}$$

$(U(1:r, 1:r))^{-1}$ and $(L(1:r, 1:r))^{-1}$ correspond to lower-triangular solves. The inverse matrices are not explicitly computed.



(a) Full Matrix Representation



(b) Low-Rank Matrix Representation

Figure 1: Classification of vertices based on distance from the other set.

6. Application for Multifrontal Solve Process and Numerical Benchmarks

In this section, we demonstrate how our fast dense solver algorithm can be applied to a sparse multifrontal solve process. We will not explain the multifrontal algorithm in detail. For a detailed review of the multifrontal method see [46]. We applied our fast solver as described in Section 3 to a variety of 3D finite-element problems.

We investigate frontal matrices at various levels of the sparse matrix elimination tree corresponding to the elasticity equation.

We use SCOTCH [52] to do the reordering in the sparse multifrontal solver. Our goal is to apply our fast dense solver to the dense frontal matrices obtained in the multifrontal elimination process of a sparse finite-element matrix, and speed up the multifrontal algorithm to approximately $\mathcal{O}(N^{4/3})$. The results shown in this paper can be viewed as a proof of concept of this idea. We should also mention that the approach presented in this article is fully general. We use SCOTCH [52], (which can partition any graph) to obtain the separators and the resulting separators can always be handled by our algorithm, without any change.

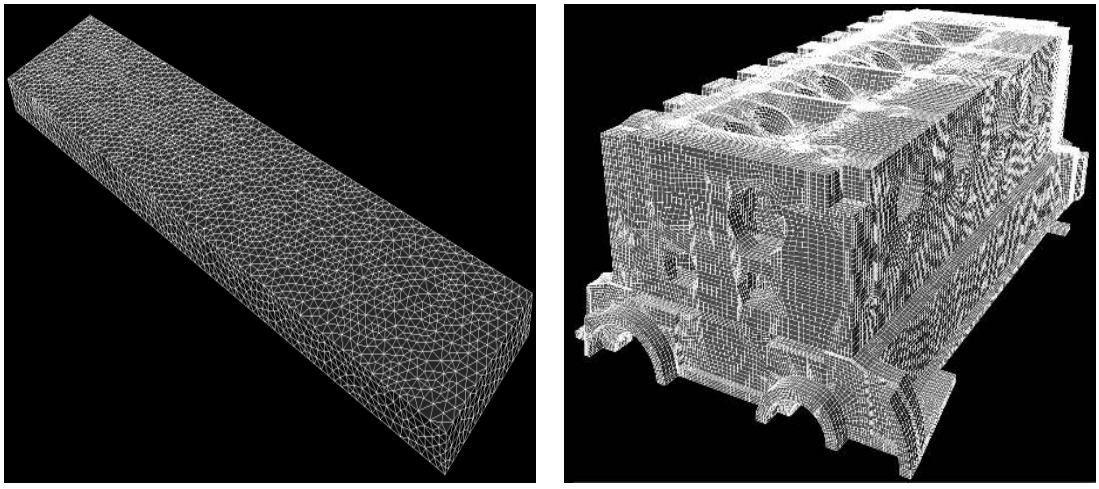
As our code uses the Eigen C++ library for matrix manipulations, we use the Eigen direct solvers as benchmark references.

6.1. Elasticity Equation for a 3D Beam and a Cylinder Head Geometry

We consider the 3D Navier-Cauchy elastostatics equations with a beam geometry (figure 2(a)):

$$(\lambda + \mu)\nabla(\nabla \cdot \boldsymbol{u}) + \mu\nabla^2\boldsymbol{u} + \boldsymbol{F} = 0 \tag{31}$$

where $\boldsymbol{u}$ is the displacement vector and λ and μ are Lamé parameters. For the beam geometry, we use 10-node tetrahedral elements (see for example Section 10.2 of this document[3]) to discretize the above equation. For the cylinder head geometry, the mesh is composed of 8-node hexahedral, 6-node pentahedral and 4-node tetrahedral solid elements, and also 3-node shell elements. Figures 2(a) and 2(b) show a sample beam and cylinder head geometry respectively. As can be seen, the meshes are unstructured for both geometries.



(a) Beam (b) Cylinder Head

Figure 2: 3D unstructured mesh for the beam and cylinder head geometries. .

We compare the fast BDLR direct solver and the ACA direct solver as preconditioners to the GMRES iterative scheme. Because of the particular geometry of the beam mesh, all frontal matrices are relatively small ($\leq$ 2K) for this particular case.

As can be seen in Figure 4, the singular values of a sample frontal matrix off-diagonal block decay rapidly and the block is in fact low-rank. Figures 6(a) and 6(b) show the

[3] `http://www.colorado.edu/engineering/CAS/courses.d/AFEM.d/AFEM.Ch10.d/AFEM.Ch10.pdf`

distance of row (column) index of each pivot obtained in the full pivoting LU factorization from the boundary between the row and column sets of vertices in the interaction graph for the beam problem. As we expected, larger pivots correspond to rows and columns that are closer to the boundary. Figures 7(a) and 7(b) compare the relative error in approximating the top off-diagonal block using SVD versus the BDLR approximation for the beam and cylinder head geometry respectively. That is, each point (x,y) in this plot represents the relative error in approximation (y) if we wanted a rank (x) approximation using one of the low-rank approximation algorithms. Needless to say, this corresponds to choosing the top singular values in the SVD decomposition and choosing rows and columns that are closest to the boundary in the BDLR approximation. As can be seen in the plot, the curves associated with the BDLR scheme have a tolerance (ϵ). This means that after the LU factorization of $\widehat{A}$ (see Section 5.3), we only keep rows and columns corresponding to pivots that are larger than ϵ times the magnitude of the largest pivot. We use this convention for all BDLR approximations in this paper. We can observe that as we decrease ϵ, we obtain a more accurate low-rank representation via the BDLR algorithm for the beam geometry. For the more complicated cylinder head geometry, we see that in order to obtain a good approximation for low values of ϵ, more rows and columns need to be included in the low-rank approximation which corresponds to a higher depth parameter (d) in the BDLR scheme.

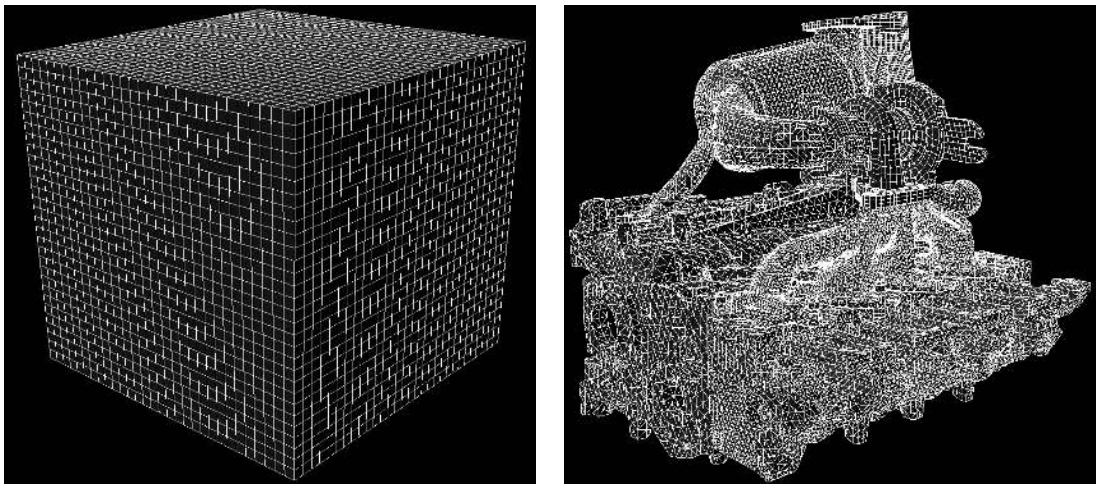
Figures 8(a) and 8(b) show a level by level timing of the factorization, solve and low-rank approximation of the BDLR solver applied to sample frontal matrices corresponding to the beam and cylinder head geometries respectively. As can be seen, the off-diagonal rank decays from root to leaf which confirms our assumptions in Section 4.4. Figures 9(a) and 9(b) show a detailed convergence analysis and comparison between the BDLR and ACA solvers as preconditioners to the GMRES iterative scheme.

6.2. FETI-DP Solver for a 3D Elasticity Problem

Domain decomposition (DD) methods solve a problem by splitting it into several subdomains. Local problems are solved on each subdomain and a global linear system is used to couple these local solutions into a global solution for the entire problem [11]. FETI methods [21, 44] are a family of domain decomposition algorithms with Lagrange multipliers that have been developed for the fast sequential and parallel iterative solution of large-scale systems of equations arising from the finite-element discretization of partial differential equations [21].

In this article, we consider two sparse local FETI-DP matrices arising from the finite-element discretization of an elasticity problem in three dimensions. The first matrix corresponds to solving the elasticity equation with a structured mesh in three dimensions (figure 3(a)) while the second matrix corresponds to solving the same problem using the geometry of an engine in an unstructured mesh (figure 3(b)). Both matrices correspond to the stiffness matrix of one subdomain of a linear elastic 3D solid finite element model (Equation (31)) of their respective geometry. The discretization for the cube geometry uses 8-node (trilinear) hexahedral elements (see for example Section 11.3 of this online document[4]) while the discretization for the engine geometry uses 10-node tetrahedral elements (see for example Section 10.2 of this document[5]).

[4] `http://www.colorado.edu/engineering/CAS/courses.d/AFEM.d/AFEM.Ch11.d/AFEM.Ch11.pdf`
[5] `http://www.colorado.edu/engineering/CAS/courses.d/AFEM.d/AFEM.Ch10.d/AFEM.Ch10.pdf`



(a) Structured Mesh (b) Unstructured Mesh

Figure 3: FETI-DP benchmark meshes. Figure (a) shows a structured and figure (b) shows an unstructured 3D FETI-DP mesh.

We apply the BDLR and ACA direct solver preconditioner to frontal matrices arising from the multifrontal elimination of local matrices in a FETI-DP solver. Figures 6(c) and 6(d) show that the largest pivot values of a sample off-diagonal block of a frontal matrix arising from the cube geometry correspond to rows and columns that are closer to the boundary. Figures 6(e) and 6(f) show that for the unstructured engine mesh, although most large pivots correspond to rows and columns near the boundary, there are some important rows and columns that are not included in the points closest to the boundary.

Figures 7(c) show that the error in the BDLR method is comparable to the SVD (optimal) algorithm for the structured cube problem. Figure 7(d) shows that similar to Figure 7(b), we need to include more points (rows and columns), in order to achieve an accurate low-rank approximation for $\epsilon = 10^{-10}$.

In other words, if there are insufficient rows and columns in the BDLR approximation, the matrix $\widehat{A}$ (see Section 5.3) becomes low-rank and results in a LU factorization with vey small pivots. These small pivots are the cause of the large relative error as they become very large when inverted.

Figures 9(c) and 9(d) show the convergence rate of various BDLR and ACA direct solver preconditioners for a sample frontal matrix arising from the cube and engine mesh respectively.

6.3. Scaling

We conduct a scaling experiment to examine the scaling of the fast HODLR solver with BDLR low-rank approximation. We use the top level frontal matrix arising from the structured cube in Section 6.2 as the model problem.

For a front of size $n \times n$ arising from a sparse matrix with N degrees of freedom, we expect the off diagonal rank to scale as $\mathcal{O}(n^{1/2})$. As the fast HODLR solver scales as $\mathcal{O}(r^2 n)$, we expect the direct solve time (HODLR factorization time plus low-rank approximation time) to scale as $\mathcal{O}(n^2)$.

Figures 5(a) and 5(b) show the direct solve time versus dense front size (n) and sparse matrix size (N) for various BDLR input paramters. As can be seen, the direct solve time

roughly scales as $\mathcal{O}(n^2)$ and $\mathcal{O}(N^{4/3})$ for a variety of BDLR input parameters as expected. As we will demonstrate in an upcoming article, this enables us to construct a fast sparse solver which scales as $\mathcal{O}(N^{4/3})$ which is much better compared to the $\mathcal{O}(N^2)$ scaling of the conventional multifrontal method for 3D meshes. Table 3 summarizes the timings obtained in this experiment.

6.4. Summary

Table 4 summarizes the solver timings for various frontal matrices that we benchmarked. As can be seen, the iterative solve scheme with both a fast BDLR and ACA direct solver preconditioner can reach near machine accuracy much faster than a conventional LU solver in almost all cases. Furthermore, both BDLR and ACA achieve a relatively good speedup for all cases. However, for very large cases (1.5M structured cube and 2.3M unstructured cylinder head), one can observe that BDLR achieves higher speedup compared to ACA. One important point to note, is that convergence of both BDLR and ACA depends on the chosen parameters. For ACA, one can get better results by decreasing the tolerance. For BDLR, in order to achieve a given tolerance, one has to increase the depth parameter (d). It is possible for BDLR not to converge for a certain tolerance and a depth parameter. This is because the depth and accuracy are related. In particular, the efficiency of the method is sometimes found to degrade if we reduce ϵ too much without increasing d sufficiently. This corresponds to the fact that we are trying to get a more accurate low-rank approximation but the pool of sample points is not sufficiently large to provide the desired accuracy. In that case, reducing ϵ may, in fact, lead to a degradation in the preconditioner, rather than an improvement.

An important advantage of the BDLR algorithm is that the rows and columns required for constructing the low-rank approximation are known a priori based on the structure of the separator graph. As we will demonstrate in a future article, this will allow us to significantly accelerate the extend-add process and allows us to avoid constructing large dense frontal and update matrices as we will only keep track of rows and columns required by the BDLR algorithm.

7. Conclusion and Future Work

To reach our final goal of constructing a fast multifrontal solver, we need to improve the slow dense solves for the frontal matrices, which we demonstrate through various benchmarks using the HODLR solver. Using block low-rank structures like the HODLR structure significantly reduces the memory consumption of a multifrontal sparse solver. In practice, this is currently one of the main bottlenecks on existing hardware.

One of the major challenges in constructing a fast direct sparse solver is the need for a low-rank approximation scheme that works for algebraic Schur complements, as found in multifrontal solvers. We have addressed this issue by introducing the BDLR low-rank approximation scheme which, as we have shown, is a very robust algorithm when applied to such matrices. The major advantage of BDLR over ACA is the fact that it allows us to only keep track of certain rows an columns in the multifrontal extend-add process. As we will show in a follow-up paper, this significantly improves the runtime and memory consumption of the solve process.

One of the drawbacks of this method is that it relies on off-diagonal blocks being low-rank, which may not be true asymptotically as $N \to \infty$ for points distributed on a 2D or 3D manifold. More precisely, in 2D, the rank stays fairly constant but in 3D, there is a slow

growth like $n^{1/2}$ where n is the size of the dense matrix or front. However, the HODLR scheme has a computational cost of $\mathcal{O}(r^2 n \log^2 n)$ with a relatively small constant in front of $r^2 n \log^2 n$. As a result, even if the rank r increases, the method stays competitive.

A major advantage of the mentioned direct solve algorithm is its parallel scalability. Since we make two independent recursive calls on each of the diagonal blocks, this method will scale very well. As a result, despite its somewhat higher number of flops (compared to an optimal $\mathcal{O}(n)$ method), the algorithm may run faster on large-scale parallel computers where communication and concurrency are key.

We estimate that the direct solver presented here starts being faster as soon as the rank r is $r \sim 0.4n$ compared to an $(2/3)n^3$ LU factorization algorithm. Recent work by Ambikasaran and Darve [2] has overcome the growth of rank in all dimensions by requiring a compression of well-separated clusters only. It is also worth mentioning that Ho and Ying [39] attempt to reduce the rank, when using interpolative decompositions to build low-rank approximations, with a scheme that, in essence, is able to reduce the dimensionality of the problem in a recursive manner.

8. Acknowledgments

The authors would like to acknowledge Prof. Charbel Farhat and Dr. Philip Avery for providing us with the FETI test matrices. We also want to thank Prof. Pierre Ramet and Dr. Mathieu Faverge for their collaboration on this work.

Part of this research was done at Stanford University, and was supported in part by the U.S. Army Research Laboratory, through the Army High Performance Computing Research Center, cooperative agreement W911NF-07-0027. This material is also based upon work supported by the Department of Energy under award number DE-NA0002373-1.

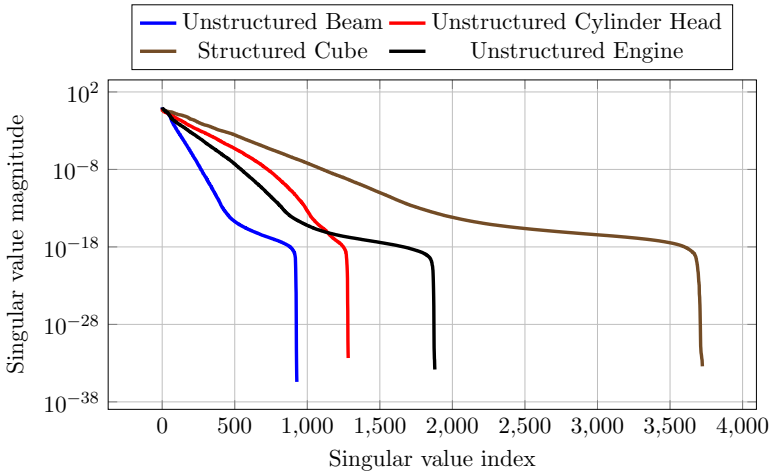
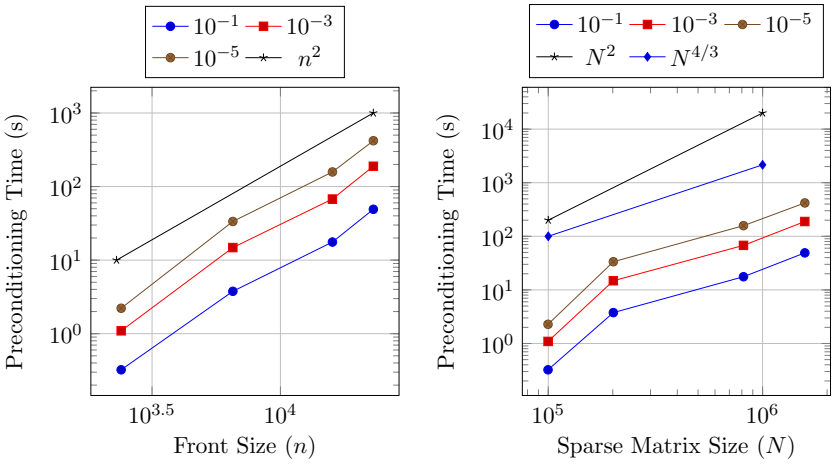


Figure 4: Singular value decay for a variety of sample off-diagonal blocks of frontal matrices. The beam, cylinder head, cube and engine geometries correspond to blocks of size 0.95K, 1.3K, 3.75K and 1.9K respectively.

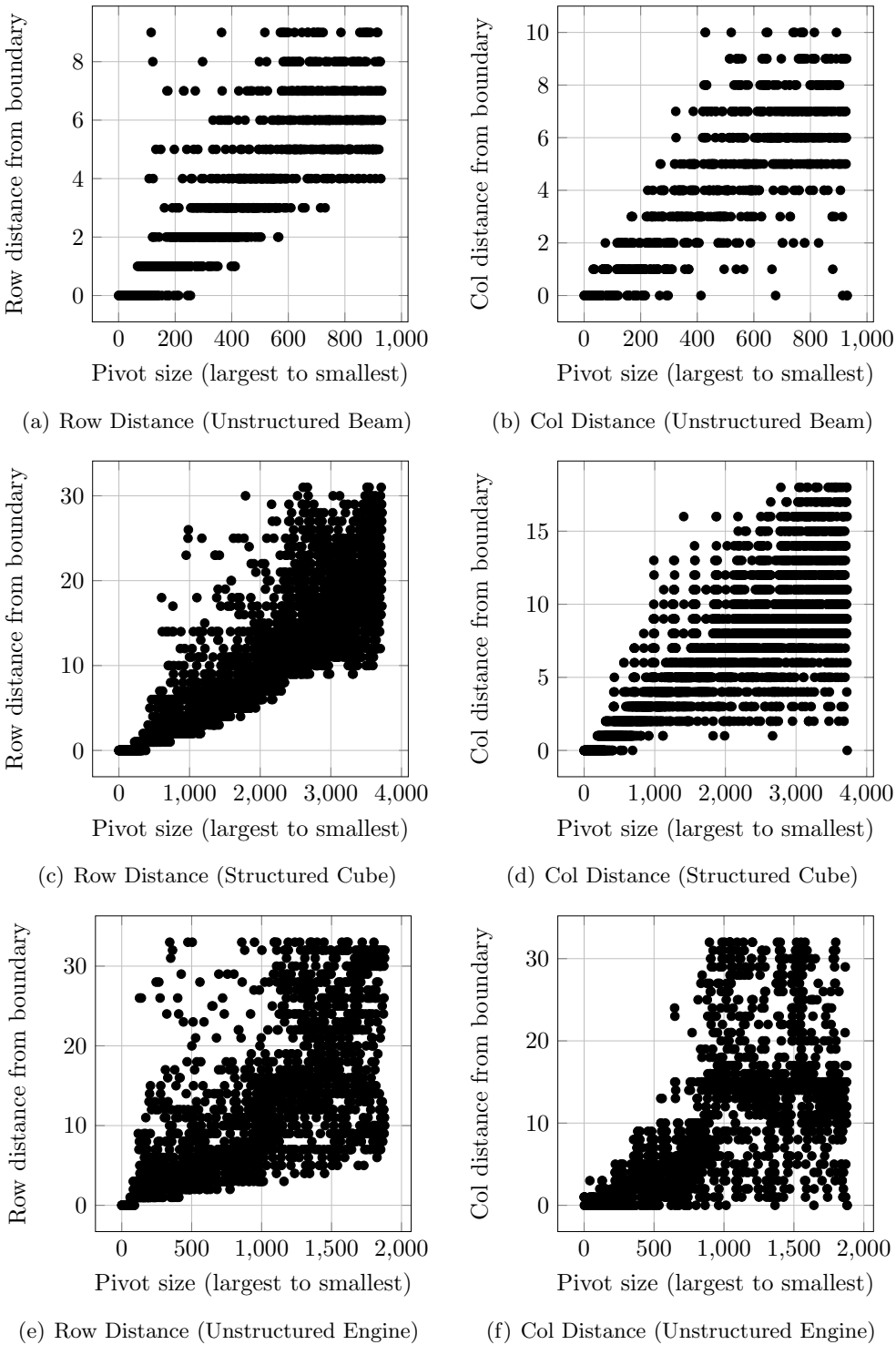


(a) Fact Time vs. Front Size (b) Fact Time vs. Sparse Matrix Size

Figure 5: Scaling of the dense front preconditioning times of the HODLR solver with BDLR low-rank approximations with front size and sparse matrix size. As a model problem, we use the top level frontal matrices arising from the sparse elimination of a structured cube as described in Section 6.2. The BDLR depth parameter (d) was set to 0,2,4 for tolerances of 10^{-1},10^{-3},10^{-5} respectively.

N	n	ϵ	d	D
99.7K	2.4K	1e-1	0	3.23e-1
		1e-3	2	1.09e0
		1e-5	4	2.22e0
201.2K	6.5K	1e-1	0	3.76e0
		1e-3	2	1.48e1
		1e-5	4	3.35e1
814.6K	15.9K	1e-1	0	1.76e1
		1e-3	2	6.77e1
		1e-5	4	1.58e2
1.6M	23.0K	1e-1	0	4.91e1
		1e-3	2	1.89e2
		1e-5	4	4.19e2

Table 3: Scaling of preconditioning times of the HODLR solver with BDLR low-rank approximations with front size and sparse matrix size. See caption of Figures 5(a) and 5(b) for experiment details. N and n refer to the sparse matrix size and the dense front sizes respectively. ϵ and d denote the tolerance and depth parameter used in the BDLR low-rank approximation and D is the direct solve time (which consists of low-rank approximation time and factorization time).



(a) Row Distance (Unstructured Beam)

(b) Col Distance (Unstructured Beam)

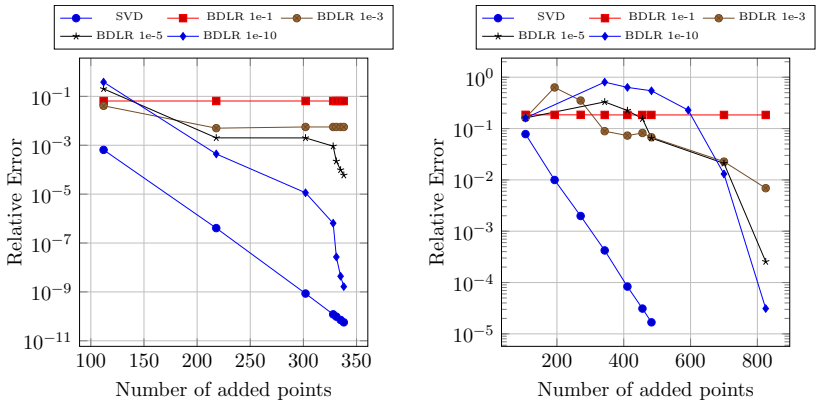
(c) Row Distance (Structured Cube)

(d) Col Distance (Structured Cube)

(e) Row Distance (Unstructured Engine)

(f) Col Distance (Unstructured Engine)

Figure 6: Row (column) distance versus pivot size for a variety of off-diagonal blocks of sample frontal matrices. Row (column) distance is the distance corresponding to the row (column) index of a pivot from the boundary as defined in Figure 1(a). This graph shows that large pivots are near the boundary interface, whereas the pivot size decays as we move away. This justifies heuristically our approach with BDLR. a,b) An off diagonal block of an unstructured beam geometry frontal matrix of size 0.95K. c,d) An off diagonal block of an structured cube geometry frontal matrix of size 3.75K. e,f) An off diagonal block of an unstructured engine geometry frontal matrix of size 1.9K.



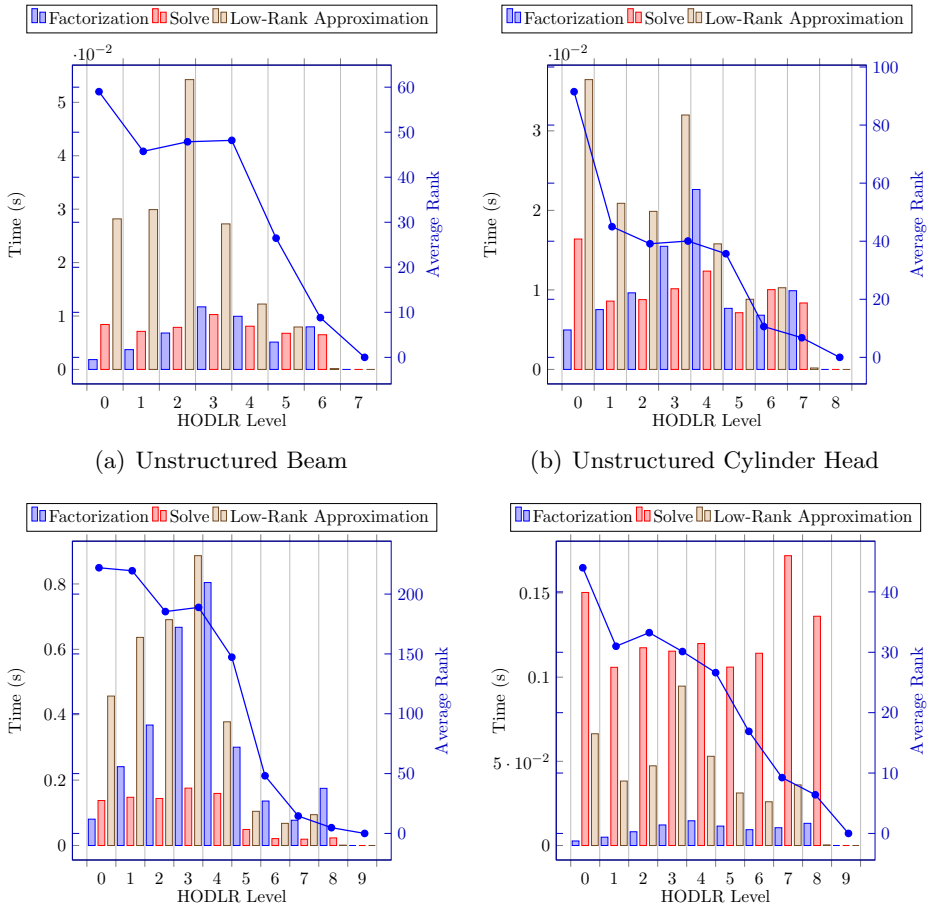
(a) Unstructured Beam

(b) Unstructured Cylinder Head

(c) Structured Cube

(d) Unstructured Engine

Figure 7: BDLR error vs SVD error for a variety of sample frontal matrix off-diagonal blocks. BDLR accuracy is used to truncate the pivots and number of points is the size of the block for full pivoting in the pseudo-skeleton approach (size of $\hat{A}$ in Equation (27)). a) An off diagonal block of an unstructured beam geometry frontal matrix of size 0.95K. b) An off diagonal block of an unstructured cylinder head geometry frontal matrix of size 1.3K. c) An off diagonal block of an structured cube geometry frontal matrix of size 3.75K. d) An off diagonal block of an unstructured engine geometry frontal matrix of size 1.9K.



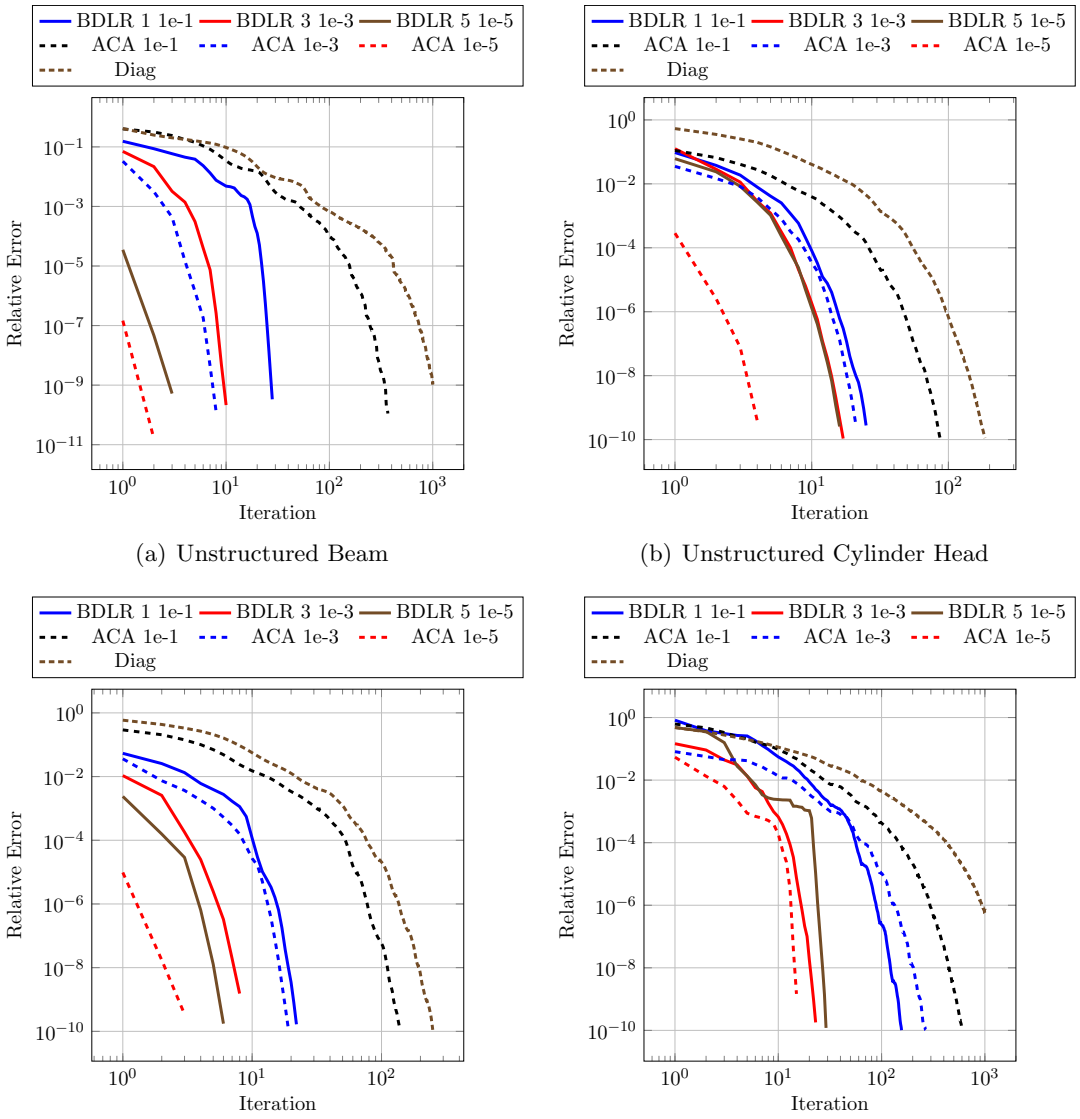
(a) Unstructured Beam

(b) Unstructured Cylinder Head

(c) Structured Cube

(d) Unstructured Engine

Figure 8: Off-diagonal rank and level by level timings for various frontal matrices. The off-diagonal ranks correspond to a BDLR low-rank approximation scheme with depth of 1 and tolerance of 10^{-1}. The left axis corresponds to runtimes for various stages in the solve process as a function of HODLR level. The right axis shows the off-diagonal rank versus HODLR level. Note that factorization time does not include low-rank approximation time. a) A frontal matrix corresponding to an unstructured beam geometry of size 1.9K. b) A frontal matrix corresponding to an unstructured cylinder head mesh of size 2.6K. c) A frontal matrix corresponding to a structured cube mesh of size 7.5K. d) A frontal matrix corresponding to an unstructured engine mesh of size 3.8K.



(a) Unstructured Beam

(b) Unstructured Cylinder Head

(c) Structured Cube

(d) Unstructured Engine

Figure 9: Convergence analysis for BDLR and ACA preconditioners with the GMRES iterative scheme for a variety of frontal matrices. The curve labeled 'Diag' corresponds to GMRES with diagonal preconditioning. a) A frontal matrix corresponding to an unstructured beam mesh of size 1.9K. b) A frontal matrix corresponding to an unstructured cylinder head mesh of size 2.6K. c) A frontal matrix corresponding to a structured cube mesh of size 7.5K. d) A frontal matrix corresponding to an unstructured engine mesh of size 3.8K.

Matrix Type	Mesh Type	Level	Matrix Size		LR	1e-1				1e-3				1e-5				LU	Speed up
			Sparse	Dense	Method	LR	P	T	I	LR	P	T	I	LR	P	T	I		
FETI Local	Cube	1st	1.5M	23K	ACA	4.59e-1	1.26e0	8.13e1	223	6.72e1	1.46e2	1.97e2	72	2.29e2	5.15e2	5.38e2	15	6.21e2	**7.64**
					BDLR	1.31e1	4.70e1	6.84e1	37	7.01e1	1.81e2	1.95e2	14	1.75e2	4.07e2	4.21e2	7		**9.08**
		1st	400K	7.5K	ACA	1.93e-1	4.65e-1	6.50e0	141	5.30e0	1.41e1	1.62e1	20	1.45e1	3.80e1	3.86e1	3	2.17e1	**3.34**
					BDLR	1.31e0	4.32e0	6.22e0	27	6.68e0	1.64e1	1.75e1	9	1.58e1	3.50e1	3.64e1	9		**3.49**
		2nd		5.2K	ACA	1.34e-1	3.65e-1	2.16e0	77	1.89e0	5.34e0	6.39e0	19	4.90e0	1.37e1	1.40e1	3	7.75e0	**3.59**
					BDLR	5.30e-1	1.73e0	2.34e0	17	2.50e0	6.14e0	6.58e0	7	6.42e0	1.35e1	1.41e1	6		**3.31**
		2nd		5.0K	ACA	1.56e-1	4.17e-1	2.09e0	74	2.15e0	6.14e0	7.17e0	19	5.18e0	1.48e1	1.51e1	2	6.80e0	**3.25**
					BDLR	5.49e-1	1.69e0	2.31e0	18	2.90e0	6.64e0	7.07e0	7	7.81e0	1.55e1	1.59e1	5		**2.94**
		3rd		2.0K	ACA	2.69e-2	8.41e-2	2.67e-1	45	1.52e-1	5.01e-1	5.75e-1	9	3.25e-1	1.06e0	1.11e0	3	5.00e-1	**1.87**
					BDLR	6.44e-2	1.83e-1	2.49e-1	12	2.18e-1	5.65e-1	6.11e-1	5	4.80e-1	1.15e0	1.20e0	4		**2.00**
		3rd		2.8K	ACA	2.59e-2	7.89e-2	4.08e-1	61	3.28e-1	1.13e0	1.30e0	13	7.38e-1	2.46e0	2.56e0	3	1.19e0	**2.92**
					BDLR	1.40e-1	3.81e-1	5.07e-1	15	5.45e-1	1.24e0	1.35e0	7	1.24e0	2.62e0	2.87e0	13		**2.35**
		3rd		2.2K	ACA	2.06e-2	6.53e-2	1.89e-1	29	1.24e-1	4.40e-1	4.99e-1	7	2.52e-1	9.10e-1	9.49e-1	2	6.29e-1	**3.33**
					BDLR	5.59e-2	1.46e-1	2.01e-1	10	1.79e-1	4.47e-1	4.91e-1	5	3.49e-1	8.71e-1	9.20e-1	4		**3.13**
		3rd		2.5K	ACA	3.95e-2	1.18e-1	3.35e-1	41	2.74e-1	9.01e-1	9.90e-1	7	6.66e-1	2.21e0	2.29e0	2	8.90e-1	**2.65**
					BDLR	9.92e-2	2.87e-1	3.82e-1	13	3.88e-1	9.72e-1	1.04e0	5	7.94e-1	2.04e0	2.15e0	6		**2.33**
		4th		2.5K	ACA	4.89e-2	1.34e-1	4.38e-1	57	5.13e-1	1.52e0	1.70e0	13	1.11e0	3.33e0	3.42e0	2	8.90e-1	**2.03**
					BDLR	1.72e-1	4.58e-1	5.79e-1	15	8.03e-1	1.72e0	1.85e0	7	1.75e0	3.59e0	3.73e0	5		**1.54**
		4th		2.2K	ACA	4.97e-2	1.28e-1	2.96e-1	35	3.42e-1	1.02e0	1.11e0	7	6.74e-1	2.14e0	2.20e0	2	5.77e-1	**1.95**
					BDLR	1.24e-1	3.23e-1	4.00e-1	12	4.72e-1	1.06e0	1.12e0	5	1.32e0	2.50e0	2.59e0	4		1.44
	Engine	6th	400K	3.8K	ACA	1.46e-2	5.28e-2	4.84e0	601	3.58e-1	9.47e-1	5.33e0	268	1.25e0	3.32e0	3.83e0	16	2.90e0	0.60
					BDLR	1.36e-1	2.29e-1	1.71e0	154	1.53e0	1.99e0	2.36e0	24	1.20e0	2.28e0	2.87e0	25		**1.70**
		9th		2.8K	ACA	8.85e-3	2.91e-2	1.23e0	248	5.61e-2	2.03e-1	3.90e-1	26	1.94e-1	6.29e-1	6.76e-1	3	1.24e0	**3.18**
					BDLR	4.32e-2	6.70e-2	4.66e-1	81	1.05e-1	1.80e-1	3.21e-1	22	1.95e-1	3.82e-1	5.31e-1	19		**3.86**
		13th		2.5K	ACA	8.86e-3	2.97e-2	1.22e0	248	5.81e-2	2.06e-1	3.98e-1	26	1.94e-1	6.33e-1	6.79e-1	3	8.17e-1	**2.05**
					BDLR	4.78e-2	7.99e-2	3.26e-1	56	1.36e-1	2.49e-1	3.60e-1	19	2.95e-1	5.65e-1	6.98e-1	17		**2.51**
Stiffness	Beam	1st	300K	1.9K	ACA	1.80e-2	4.68e-2	2.86e0	x	1.32e-1	4.29e-1	5.11e-1	13	2.38e-1	8.11e-1	8.60e-1	4	3.80e-1	0.74
					BDLR	7.56e-2	1.39e-1	3.65e-1	67	3.27e-1	5.50e-1	6.36e-1	16	6.69e-1	1.14e0	1.20e0	7		1.04
		2nd		1.9K	ACA	2.49e-2	6.67e-2	1.14e0	358	1.29e-1	4.28e-1	4.78e-1	7	2.47e-1	8.32e-1	8.64e-1	2	3.87e-1	0.81
					BDLR	7.79e-2	1.41e-1	2.52e-1	32	3.13e-1	5.26e-1	5.92e-1	12	5.72e-1	1.04e0	1.08e0	4		**1.54**
		2nd		1.9K	ACA	1.96e-2	5.38e-2	2.86e0	x	1.15e-1	3.93e-1	4.57e-1	10	2.37e-1	8.08e-1	8.54e-1	4	3.90e-1	0.86
					BDLR	6.04e-2	1.12e-1	3.15e-1	62	2.75e-1	4.76e-1	5.57e-1	15	5.93e-1	1.05e0	1.11e0	7		1.24
		3rd		1.9K	ACA	2.17e-2	6.19e-2	6.03e-1	185	1.09e-1	3.68e-1	4.09e-1	6	2.33e-1	7.86e-1	8.17e-1	2	3.70e-1	0.90
					BDLR	5.79e-2	1.09e-1	2.01e-1	28	2.88e-1	4.87e-1	5.55e-1	12	6.24e-1	1.08e0	1.13e0	5		1.84
		3rd		1.9K	ACA	2.00e-2	5.52e-2	1.09e0	369	1.17e-1	3.93e-1	4.45e-1	8	2.26e-1	7.71e-1	8.00e-1	2	3.70e-1	0.94
					BDLR	5.60e-2	1.09e-1	2.11e-1	31	2.62e-1	4.60e-1	5.21e-1	11	6.04e-1	1.05e0	1.10e0	5		**1.75**
	CHead	5th	2.3M	24K	ACA	4.14e-1	1.02e0	3.23e2	829	1.50e1	3.21e1	5.30e2	x	5.39e1	1.15e2	1.30e2	22	6.98e2	**5.37**
					BDLR	2.00e0	4.05e0	3.96e2	x	4.83e0	1.15e1	7.80e1	149	2.14e1	3.39e1	8.57e1	110		**8.95**
		2nd	330K	4.8K	ACA	2.80e-2	8.40e-2	4.39e0	265	9.76e-1	2.62e0	3.45e0	24	3.90e0	9.46e0	9.71e0	3	5.86e0	**1.7**
					BDLR	1.10e-1	2.21e-1	2.37e0	114	2.35e-1	6.15e-1	3.33e0	117	9.15e-1	1.67e0	4.27e0	94		**2.47**
		4th		2.6K	ACA	1.20e-2	3.74e-2	4.45e-1	88	2.40e-1	7.03e-1	9.21e-1	22	1.13e0	2.72e0	2.86e0	5	9.30e-1	**2.09**
					BDLR	6.99e-2	1.69e-1	3.26e-1	28	2.35e-1	5.70e-1	7.46e-1	21	4.94e-1	1.22e0	1.43e0	18		**2.85**

Table 4: Summary of solver speed for various benchmark cases. All timings are measured in seconds. The GMRES accuracy and maximum number of iterations was set to 10^{-10} and 1000 respectively for all cases. The letter 'x' indicates that the solver did not converge within 1000 iterations. For BDLR, 'x' means that the solver had a slow rate of convergence. However, in case of ACA, 'x' means that the solver might be diverging. All LU timings are obtained using Eigen's [41] partial pivoting LU solver. Level indicates the level of the dense frontal matrix in the sparse elimination tree. 'LR' denotes low-rank approximation time, 'P' denotes the total preconditioning (direct solve) time (which is equal to LR + factorization time), 'T' denotes the total solve time (which is equal to P + iteration time) and 'I' refers to the number of iterations in the iterative solver. For BDLR, we used a depth of 0, 2 and 4 for tolerances 10^{-1}, 10^{-3} and 10^{-5} respectively. For the 4.8K and 23K cylinder head matrices, the results on the last BDLR column were obtained using a tolerance of 10^{-4} and a depth of 10. We have calculated the speedups by comparing the runtime of the conventional LU solver to the fastest runtime (lowest 'T') for each case.

[1] Ambikasaran, S., 2013. Fast algorithms for dense numerical linear algebra and applications. Ph.D. thesis, Stanford University.

[2] Ambikasaran, S., Darve, E., 2014. The inverse fast multipole method. arXiv preprint arXiv:1407.1572.

[3] Ambikasaran, S., Darve, E. F., 2013. An $\mathcal{O}(N \log N)$ fast direct solver for partial hierarchically semi-separable matrices. Journal of Scientific Computing 57 (3), 477–501.

[4] Ambikasaran, S., Foreman-Mackey, D., Greengard, L. F., Hogg, D. W., O'Neil, M., 2014. Fast direct methods for Gaussian processes and the analysis of NASA Kepler mission data. arXiv preprint arXiv:1403.6015.

[5] Ambikasaran, S., Li, J. Y., Kitanidis, P. K., Darve, E. F., 2013. Large-scale stochastic linear inversion using hierarchical matrices. Computational Geosciences 17 (6), 913–927.

[6] Ambikasaran, S., Saibaba, A. K., Darve, E. F., Kitanidis, P. K., 2013. Fast algorithms for Bayesian inversion. In: Computational Challenges in the Geosciences. Springer, pp. 101–142.

[7] Amestoy, P., Ashcraft, C., Boiteau, O., Buttari, A., L'Excellent, J. Y., Weisbecker, C., 2012. Improving multifrontal methods by means of block low-rank representations. SIAM Journal on Scientific Computing (Under Review).

[8] Amestoy, P., Duff, I. S., Koster, J., L'Excellent, J.-Y., 2001. A fully asynchronous multifrontal solver using distributed dynamic scheduling. SIAM Journal on Matrix Analysis and Applications 23 (1), 15–41.

[9] Bebendorf, M., 2000. Approximation of boundary element matrices. Numerische Mathematik 86 (4), 565–589.

[10] Bebendorf, M., 2008. Hierarchical Matrices: A Means to Efficiently Solve Elliptic Boundary Value Problems. Vol. 63 of Lecture Notes in Computational Science and Engineering (LNCSE). Springer-Verlag, iSBN 978-3-540-77146-3.

[11] Benamou, J., Despres, B., 1997. A domain decomposition method for the Helmholtz equation and related optimal control problems. Journal of Computational Physics 136, 68–82.

[12] Börm, S., Grasedyck, L., Hackbusch, W., 2003. Hierarchical matrices. Lecture notes 21.

[13] Chandrasekaran, S., Dewilde, P., Gu, M., Lyons, W., Pals, T. P., 2006. A fast solver for HSS representations via sparse matrices. SIAM Journal on Matrix Analysis and Applications 29 (1), 67–81.

[14] Chandrasekaran, S., Gu, M., Li, X., Xia, J., 2008. Some fast algorithms for hierarchially semiseparable matrices. Tech. rep., UCLA.

[15] Chandrasekaran, S., Gu, M., Pals, T., 2006. A fast ULV decomposition solver for hierarchially semiseparable representations. SIAM Journal on Matrix Analysis and Applications 28 (3), 603–622.

[16] Chen, J., 2014. Data structure and algorithms for recursively low-rank compressed matrices. Argonne National Laboratory.

[17] Cheng, H., Gimbutas, Z., Martinsson, P., Rokhlin, V., 2005. On the compression of low rank matrices. SIAM Journal on Scientific Computing.

[18] Davis, T. A., HU, Y., 2011. University of Florida Sparse Matrix Collection. URL `http://www.cise.ufl.edu/research/sparse/matrices/index.html`

[19] Deshpande, A., Vempala, S., 2006. Adaptive sampling and fast low-rank matrix approximation. Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques.

[20] Duff, I., Reid, J., 1983. The multifrontal solution of indefinite sparse symmetric linear equations. ACM Transactions on Mathematical Software 9 (3), 302–325.

[21] Farhat, C., Lesoinne, M., LeTallec, P., Pierson, K., Rixen, D., 2001. FETI-DP: a dual-primal unified FETI method – Part 1: A faster alternative to the two-level FETI method. International Journal for Numerical Methods in Engineering 50, 1523–1544.

[22] Frieze, A., Kannan, R., Vempala, S., 2004. Fast monte-carlo algorithms for finding low-rank approximations. Journal of the ACM.

[23] George, A., 1973. Nested dissection of a regular finite element mesh. SIAM Journal on Matrix Analysis and Applications.

[24] Gillman, A., Martinsson, P., 2014. An O(n) algorithm for constructing the solution operator to 2d elliptic boundary value problems in the absence of body loads. Advances in Computational Mathematics.

[25] Gillman, A., Young, P., Martinsson, P., 2012. A direct solver with $O(N)$ complexity for integral equations on one-dimensional domains. Frontiers of Mathematics in China 7 (2), 217–247.

[26] Goreinov, S., Tyrtyshnikov, E., Zamarashkin, N., 1997. A theory of pseudoskeleton approximations. Linear Algebra and Its Applications 261, 1–21.

[27] Grasedyck, L., Hackbusch, W., 2003. Construction and arithmetics of $\mathcal{H}$-matrices. Computing 70 (4), 295–334.

[28] Gu, M., Eisenstat, S., 1996. Efficient algorithms for computing a strong rank-revealing qr factorization. SIAM Journal on Scientific Computing.

[29] Hackbusch, W., 1999. A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: Introduction to $\mathcal{H}$-matrices. Computing 62 (2), 89–108.

[30] Hackbusch, W., Börm, S., 2002. Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices. Computing 69 (1), 1–35.

[31] Hackbusch, W., Börm, S., 2002. $\mathcal{H}^2$-matrix approximation of integral operators by interpolation. Applied Numerical Mathematics 43 (1), 129–143.

[32] Hackbusch, W., Khoromskij, B., Sauter, S. A., 2000. On $\mathcal{H}^2$-matrices. In: Bungartz, H.-J., Hoppe, R. H., Zenger, C. (Eds.), Lectures on Applied Mathematics. Springer Berlin Heidelberg, pp. 9–29.

[33] Hackbusch, W., Khoromskij, B. N., 2000. A sparse $\mathcal{H}$-matrix arithmetic. Computing 64 (1), 21–47.

[34] Hackbusch, W., Khoromskij, B. N., 2000. A sparse $\mathcal{H}$-matrix arithmetic: general complexity estimates. Journal of Computational and Applied Mathematics 125 (1), 479–501.

[35] Hager, W. W., 1989. Updating the inverse of a matrix. SIAM Review 31 (2), 221–239.

[36] Halko, N., Martinsson, P., Tropp., J., 2009. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. SIAM Review.

[37] Hestenes, M. R., Stiefel, E., 1952. Methods of conjugate gradients for solving linear systems. Journal of Research of the National Bureau of Standards 49 (6), 409–436.

[38] Ho, K., Greengard, L., 2012. A fast direct solver for structured linear systems by recursive skeletonization. SIAM Journal on Scientific Computing 34 (5), A2507–2532.

[39] Ho, K., Ying, L., 2013. Hierarchical interpolative factorization for elliptic operators: integral equations. arXiv preprint arXiv:1307.2666.

[40] Irons, B., 1970. A frontal solution program for finite element analysis. International Journal for Numerical Methods in Engineering 2, 5–32.

[41] Jacob, B., 2010. Eigen v3. http://eigen.tuxfamily.org.

[42] Kong, W., Bremer, J., Rokhlin, V., 2011. An adaptive fast direct solver for boundary integral equations in two dimensions. Applied and Computational Harmonic Analysis 31, 346–369.

[43] Lai, J., Ambikasaran, S., Greengard, L. F., 2014. A fast direct solver for high frequency scattering from a large cavity in two dimensions. SIAM Journal on Scientific and Statistical Computing.

[44] Li, J., O.Widlund, 2005. FETI-DP, BDDC, and block Cholesky methods. International Journal for Numerical Methods in Engineering.

[45] Li, J. Y., Ambikasaran, S., Darve, E. F., Kitanidis, P. K., 2014. A Kalman filter powered by $\mathcal{H}^2$-matrices for quasi-continuous data assimilation problems. Water Resources Research.

[46] Liu, J., March 1992. The multifrontal method for sparse matrix solution theory and practice. SIAM Review 34 (1), 82–109.

[47] Liu, J. W. H., Mar. 1992. The multifrontal method for sparse matrix solution: Theory and practice. SIAM Review 34 (1), 82–109.

[48] Mahoney, M. W., Drineas, P., 2009. Cur matrix decompositions for improved data analysis. Proceedings of the National Academy of Sciences.

[49] Martinsson, P., 2009. A fast direct solver for a class of elliptic partial differential equations. Journal of Scientific Computing 38, 316–330.

[50] Martinsson, P., 2011. A fast randomized algorithm for computing a hierarchially semiseperable representation of a matrix. SIAM Journal on Matrix Analysis and Applications.

[51] Martinsson, P., Rokhlin, V., 2005. A fast direct solver for boundary integral equations in two dimensions. Journal of Computational Physics, 1–23.

[52] Pellegrini, F., Roman, J., 1996. *SCOTCH*: *A* software package for static mapping by dual recursive bipartitioning of process and architecture graphs. High-Performance Computing and Networking 1067, 493–498.

[53] Rjasanow, S., 2002. Adaptive cross approximation of dense matrices. In: International Association for Boundary Element Methods Conference, IABEM.

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

[55] Sapp, B. J., 2011. Randomized algorithms for low-rank matrix decomposition. Computer and Information Science, University of Pennsylvania.

[56] Schmitz, P., Ying, L., 2012. A fast direct solver for elliptic problems on general meshes in 2D. Journal of Computational Physics 231, 1314–1338.

[57] Woolfe, F., Liberty, E., Rokhlin, V., Tygert, M., 2008. A fast randomized algorithm for the approximation of matrices. Applied and Computational Harmonic Analysis 25 (335-366).

[58] Xia, J., 2013. Effcient strcutured multifrontal factorization for general large sparse matrices. SIAM Journal on Scientific Computing.

[59] Xia, J., 2013. Randomized sparse direct solvers. SIAM Journal on Matrix Analysis and Applications.

[60] Xia, J., Chandrasekaran, S., Gu, M., Li, X. S., 2009. Superfast multifrontal method for large structured linear systems of equations. SIAM Journal on Matrix Analysis and Applications 31 (3), 1382–1411.

[61] Xia, J., Chandrasekaran, S., Gu, M., Li, X. S., 2010. Fast algorithms for hierarchially semiseparable matrices. Numerical Linear Algebra with Applications 17, 953–976.

[62] Ying, L., 2009. Fast algorithms for boundary integral equations. In: Multiscale Modeling and Simulation in Science. Springer, pp. 139–193.