A Parallel Geometric Multifrontal Solver Using Hierarchically Semiseparable Structure

SHEN WANG, Department of Mathematics, Purdue University
XIAOYE S. LI, Lawrence Berkeley National Laboratory
FRANÇOIS-HENRY ROUET, Lawrence Berkeley National Laboratory
JIANLIN XIA, Department of Mathematics, Purdue University
MAARTEN V. DE HOOP, Department of Mathematics, Purdue University

We present a structured parallel geometry-based multifrontal sparse solver using hierarchically semiseparable (HSS) representations and exploiting the inherent low-rank structures. Parallel strategies for nested dissection ordering (taking low-rankness into account), symbolic factorization, and structured numerical factorization are shown. In particular, we demonstrate how to manage two layers of tree parallelism to integrate parallel HSS operations within the parallel multifrontal sparse factorization. Such a structured multifrontal factorization algorithm can be shown to have asymptotically lower complexities in both operation counts and memory than the conventional factorization algorithms for certain partial differential equations. We present numerical results from the solution of the anisotropic Helmholtz equations for seismic imaging, and demonstrate that our new solver was able to solve 3D problems up to 600^3 mesh size, with 216M degrees of freedom in the linear system. For this specific model problem, our solver is both faster and more memory efficient than a geometry-based multifrontal solver (which is further faster than general-purpose algebraic solvers such as MUMPS and SuperLU_DIST). For the 600^3 mesh size, the structured factors from our solver need about 5.9 times less memory.

Categories and Subject Descriptors: Mathematics of computing [**Mathematical software**]: Solvers

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Sparse Gaussian elimination, multifrontal method, HSS matrices, parallel algorithm

1. INTRODUCTION

In many computational and engineering problems it is critical to solve large sparse linear systems of equations. Direct methods are attractive due to their reliability and generality, and are especially suitable for systems with different right-hand sides. However, it is prohibitively expensive to use direct methods for large-scale 3D problems, due to their superlinear complexity of memory requirement and operation count. A potential avenue to develop fast and memory-efficient direct solvers is to exploit certain *structures* in the prob-

lems. Different structures come from different natures of the underlying problems or different discretization and computation techniques. In recent years, rank structured matrices have been investigated extensively due to their potential in accelerating the solutions of various partial differential equations and integral equations. Several useful rank structured matrix representations have been developed, such as $\mathcal{H}$-matrices [Hackbusch 1999] [Hackbusch and Khoromskij 2000] [Hackbusch et al. 2002], $\mathcal{H}^2$-matrices [Börm et al. 2003] [Börm and Hackbusch 2001] [Hackbusch et al. 2000], quasiseparable matrices [Bella et al. 2008] [Eidelman and Gohberg 1999], semiseparable matrices [Chandrasekaran et al. 2005] [Vandebril et al. 2005], and multilevel low-rank structures [Li and Saad 2013].

In this paper, we develop a new class of parallel structured sparse factorization method exploiting numerically low-rank structures using hierarchically semi-separable (HSS) matrices [Chandrasekaran et al. 2006] [Xia et al. 2010]. The novelty of the HSS-structured solver is to apply the parallel HSS techniques in [Wang et al. 2013] to the intermediate dense submatrices that appear in the parallel sparse factorization methods. Specifically, we consider the multifrontal factorization method [Duff and Reid 1983] in this paper. We investigate several important aspects, such as parallel nested dissection that preserves the geometry and benefits the low-rankness, the symbolic factorization, the integration of the HSS tree parallelism within the outer multifrontal tree parallelism, and how the rank properties behave and benefit the complexities.

The resulting HSS-structured factorization can be used as a direct solver or preconditioner depending on the application's accuracy requirement and the characteristics of the PDEs. The implementation that we present in the experimental section is restricted to the solution of problems on regular grids. For some 3D model problems and broader classes of PDEs, it was shown that the HSS-structured multifrontal method costs $O(n^{4/3} \log n)$ flops [Xia 2013]. This complexity is much lower than the $O(n^2)$ cost of the traditional exact multifrontal method. The theoretical memory count is $O(n \log n)$. Numerical tests indicate that our structured parallel solver is faster and needs less memory than a standard geometric multifrontal solver (which is further faster than general-purpose algebraic solvers such as MUMPS and SuperLU_DIST). This new class of HSS-structured factorizations can be applied to much broader classes of discretized PDEs (including non-self-adjoint and indefinite ones) aiming towards optimal complexity preconditioners.

The rest of the paper is organized as follows. In Section 2 we review the multifrontal factorization algorithm and the HSS structure. Section 4 presents our new parallel HSS-structured multifrontal algorithm in detail. Analysis of the rank properties and the complexities are presented in Section 3. In Section 5, we demonstrate the parallel performance of our geometric, sparse multifrontal solver. Section 6 is devoted to the conclusions.

2. REVIEW OF THE MULTIFRONTAL AND HSS-STRUCTURED MULTIFRONTAL METHODS

In this section, we briefly review the multifrontal method, HSS representations, and HSS-structured multifrontal methods.

2.1. Multifrontal method with nested dissection ordering

The central idea of the multifrontal method is to reorganize the factorization of a large sparse matrix into the factorizations of many smaller dense matrices and partial updates to the Schur complements [Duff and Reid 1983; Liu 1992]. We now briefly recall the main ideas. We are to compute a factorization of a given matrix A, $A = LU$ if the matrix is unsymmetric, or $A = LDL^T$ if the matrix is symmetric. Without loss of generality, we assume that A is nonreducible. For a matrix A with an unsymmetric pattern, we assume that the factorization takes place with the nonzero structure of $A + A^T$, where the summation is structural. In this case, the multifrontal method relies on a structure called the elimination tree. A few

equivalent definitions are possible (we recommend the survey by Liu [Liu 1990]), and we use the following.

Definition 2.1. Assume $A = LU$, where A is a sparse, structurally symmetric, and $N \times N$ matrix. Then, the elimination tree of A is a tree of N nodes, with the ith node corresponding to the ith column of L and with the parent relations defined by:

$$parent(j) = \min\{i : i > j \text{ and } \ell_{ij} \neq 0\}, \text{ for } j = 1, \dots, N-1.$$

In practice, nodes are *amalgamated*: nodes that represent columns and rows of the factors with similar structures are grouped together in a single node. In the end, each node corresponds to a square dense matrix (referred to as a *frontal matrix*) with the following 2×2 block structure:

$$\mathcal{F}_j = \begin{bmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{bmatrix}. \tag{1}$$

Factoring the matrix consists in a bottom-up traversal of the tree, following a topological order (a node is processed before its parent). Processing a node consists in:

— forming (or *assembling*) the frontal matrix: this is achieved by summing the rows and columns of A corresponding to the variables in the $(1,1)$, $(2,1)$ and $(1,2)$ blocks, with the temporary data (update matrices) that have been produced by the child nodes; that is,

$$\mathcal{F}_j = A_j + \sum_{k:\text{ child of } j} \mathcal{U}_k; \tag{2}$$

— eliminating the fully-summed variables in the $(1,1)$ block F_{11}: this is done through a partial factorization of the frontal matrix which produces the corresponding rows and columns of the factors stored in F_{11}, F_{21} and F_{12}. At this step, the so-called Schur complement or *contribution block* is computed as $\mathcal{U}_j = F_{22} - F_{21}F_{11}^{-1}F_{12}$ and stored in a temporary memory; it will be used to form the frontal matrix associated with the parent node. Therefore, when a node is activated, it "consumes" the contribution blocks of its children.

In this process, the elimination step involves straightforward dense matrix operations, but the assembling step involves index manipulation and indirect addressing while summing up $\mathcal{U}_k$. For example, if two children's update matrices $\mathcal{U}_{c_i} = \begin{pmatrix} a_i & b_i \\ c_i & d_i \end{pmatrix}, i = 1, 2$ have subscript sets $\{1, 2\}$ and $\{1, 3\}$, respectively, then

$$\sum_{i=1,2} \mathcal{U}_{c_i} = \begin{pmatrix} a_1 & b_1 & 0 \\ c_1 & d_1 & 0 \\ 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} a_2 & 0 & b_2 \\ 0 & 0 & 0 \\ c_2 & 0 & d_2 \end{pmatrix} = \begin{pmatrix} a_1 + a_2 & b_1 & b_2 \\ c_1 & d_1 & 0 \\ c_2 & 0 & d_2 \end{pmatrix}. \tag{3}$$

This summation operation is called *extend-add*, denoted by $\oplus$, which aligns the subscript sets of two matrices by padding zero entries, and then adds matrices. The relationship between frontal matrices and update matrices can be revealed by $\mathcal{F}_j = A_j \oplus \mathcal{U}_{c_1} \oplus \mathcal{U}_{c_2} \oplus \cdots \oplus \mathcal{U}_{c_q}$, where nodes $c_1, c_2, \dots, c_q$ are the children of j.

In the multifrontal factorization, the *active memory* (at a given step in the factorization) consists of the frontal matrix being processed and a set of contribution blocks that are temporarily stored and will be consumed at a later step. The multifrontal method lends itself very naturally to parallelism since multiple processes can be employed to treat one, large enough, frontal matrix or to process concurrently frontal matrices belonging to separate subtrees. These two sources of parallelism are commonly referred to as *node* and *tree parallelism*, respectively, and their correct exploitation is the key to achieving high performance on parallel supercomputers.

In the following section, we exploit additional rank structures in the frontal matrices to achieve better cost and memory efficiency.

2.2. Low-rank property and HSS structures

We briefly summarize the key concepts of HSS structures following the definitions and notation in [Xia 2012; Xia et al. 2010]. Let F be a general $n \times n$ real or complex matrix and $\mathcal{I} = \{1, 2, \ldots, N\}$ be the set of all row and column indices. Suppose $\mathcal{T}$ is a full binary tree with $2k - 1$ nodes labeled as $i = 1, 2, \ldots, 2k - 1$, such that the root node is $2k - 1$ and the number of leaf nodes is k. Let $\mathcal{T}$ also be a *postordered* tree. That is, for each non-leaf node i of $\mathcal{T}$, its left child c_1 and right child c_2 satisfy $c_1 < c_2 < i$. Let $t_i \subset \mathcal{I}$ be an index subset associated with each node i of $\mathcal{T}$. We use $F|_{t_i \times t_j}$ to denote the submatrix of F with row index subset t_i and column index subset t_j.

HSS matrices are designed to take advantage of the low-rank property. In particular, when the off-diagonal blocks of a matrix (with hierarchical partitioning) have small (numerical) ranks, they are represented or approximated hierarchically by compact forms. These compact forms at different hierarchical levels are also related through nested basis forms. This can be seen from the definition of an *HSS form*:

Definition 2.2. We say that F is in a postordered HSS form and $\mathcal{T}$ is the corresponding HSS tree if the following conditions are satisfied:

— $t_{c_1} \cap t_{c_2} = \emptyset$, $t_{c_1} \cup t_{c_2} = t_i$ for each non-leaf node i of $\mathcal{T}$, where i has child nodes c_1 and c_2 and $t_{2k-1} = \mathcal{I}$.
— There exist matrices $D_i, U_i, R_i, B_i, W_i, V_i$ (or HSS generators) associated with each node i, such that, if i is a non-leaf node,

$$D_{2k-1} = F,$$

$$D_i = F|_{t_i \times t_i} = \begin{pmatrix} D_{c_1} & U_{c_1} B_{c_1} V_{c_2}^H \\ U_{c_2} B_{c_2} V_{c_1}^H & D_{c_2} \end{pmatrix}, \tag{4}$$

$$U_i = \begin{pmatrix} U_{c_1} R_{c_1} \\ U_{c_2} R_{c_2} \end{pmatrix}, \quad V_i = \begin{pmatrix} V_{c_1} W_{c_1} \\ V_{c_2} W_{c_2} \end{pmatrix},$$

where the superscript H means the Hermitian transpose.

The HSS generators define the HSS form of F. The use of a postordered HSS tree enables us to use a single subscript (corresponding to the label of a node of $\mathcal{T}$) for each HSS generator [Xia et al. 2010] instead of up to three subscripts as in [Chandrasekaran et al. 2006]. Figure 1 illustrates a block 8×8 HSS representation F. As a special example, its leading block 4×4 part looks like:

$$F|_{t_7 \times t_7} = \begin{pmatrix} \begin{pmatrix} D_1 & U_1 B_1 V_2^H \\ U_2 B_2 V_1^H & D_2 \end{pmatrix} & \begin{pmatrix} U_1 R_1 \\ U_2 R_2 \end{pmatrix} B_3 \begin{pmatrix} W_4^H V_4^H & W_5^H V_5^H \end{pmatrix} \\ \begin{pmatrix} U_4 R_4 \\ U_5 R_5 \end{pmatrix} B_6 \begin{pmatrix} W_1^H V_1^H & W_2^H V_2^H \end{pmatrix} & \begin{pmatrix} D_4 & U_4 B_4 V_5^H \\ U_5 B_5 V_4^H & D_5 \end{pmatrix} \end{pmatrix}.$$

For each diagonal block $D_i = F|_{t_i \times t_i}$ associated with each node i of $\mathcal{T}$, we define $F_i^- = F|_{t_i \times (\mathcal{I} \setminus t_i)}$ to be the *HSS block row*, and $F_i^{|} = F|_{(\mathcal{I} \setminus t_i) \times t_i}$ to be the *HSS block column*. They are both called *HSS blocks*. The maximum rank r (or numerical rank r for a given tolerance) of all the HSS blocks is called the *HSS rank* of F. If r is small as compared with the matrix size, we say that F has a *low-rank property*.

Once the matrix is put into the HSS format, the transformed linear system can be efficiently solved with the ULV-type factorization and solution algorithms [Chandrasekaran
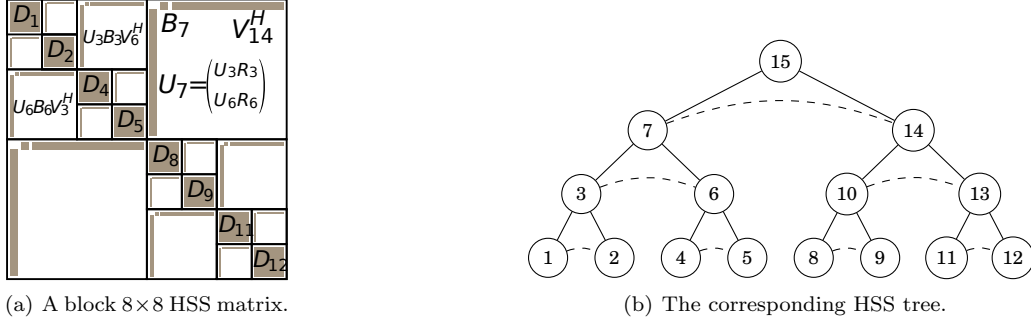
(a) A block 8×8 HSS matrix.

(b) The corresponding HSS tree.

Fig. 1.   Pictorial illustrations of a block 8×8 HSS form and the corresponding HSS tree $\mathcal{T}$.

et al. 2006; Xia et al. 2010]. The floating point operations associated with the HSS construction, ULV factorization and solution are $O(rN^2)$, $O(r^2N)$ and $O(rN)$, respectively. All these are much smaller than the $O(N^3)$ operations required by the traditional dense LU factorization algorithms for solving such linear systems. We described efficient parallel algorithms for these HSS operations on dense matrices in a previous work [Wang et al. 2013].

2.3. HSS-structured multifrontal methods

Recall that with the nested dissection ordering, there is a one-to-one correspondence between a separator and a frontal matrix $\mathcal{F}_j$. The frontal matrices are dense and their computations contribute to the dominant terms in the costs for computing the factors and the storage for the factors. Employing data-sparse compression for these frontal matrices can drastically lower the overall factorization cost and memory. (Section 3 contains the theoretical justification.)

Thus, we follow the algorithms in [Xia 2013; Xia et al. 2009] to use HSS matrices to approximate the frontal matrices $\mathcal{F}_j$. This generally involves the four steps shown in Algorithm 1. Steps (b) and (c) can be performed conveniently via the HSS algorithms in [Xia 2013; Xia et al. 2010]. For step (d), ideally, we would like to have a fast procedure which assembles the HSS forms of the contribution block directly into the HSS form of the parent frontal matrix in step (a), as shown in Figure 2(a). Unfortunately, this involves complicated HSS structured operations, because the matrix indices (hence the HSS trees) of the contribution blocks are different between the siblings, and also different from the parent. Directly performing HSS-based extend-add requires tree rotations, splitting and merging operations. This was implemented in a 2D code [Xia et al. 2009], but it may not be easily generalized to a general algebraic solver or a parallel one. Therefore, we resort to a simpler alternative method in [Xia 2013], in which we keep the contribution block as a normal dense matrix and the extend-add operation is done as in the classical (full-rank) multifrontal factorization. This is depicted in Figure 2(b), and we call this algorithm *partially structured*. Compared to a fully structured algorithm, the added cost is the decompression (HSS-to-dense) needed to compute the contribution block and the recompression (dense-to-HSS) for the parent frontal matrix.

The extend-add procedure with HSS and HSS tree manipulations (merge, rotation, etc.) is very sophisticated. The parallel implementation is even more difficult. This current partially structured version, sequential [Xia 2013] or parallel (this paper), makes the implementation much more convenient. Moreover, we would like to emphasize that this simplified version only incurs extra memory for the intermediate frontal/update matrices during the factorization. Such memory is temporary. The final factors are still structured, just like when we use structured extend-add in a fully HSS-structured algorithm. See Figure 2. The overall

---

ALGORITHM 1: The HSS-structured multifrontal algorithm.

(a) Construct an HSS approximation to $\mathcal{F}_j$.
(b) Perform a partial ULV factorization for F_{11}.
(c) Compute the contribution block $\mathcal{U}_j = F_{22} - F_{21}F_{11}^{-1}F_{12}$ via a low-rank update.
(d) Assemble $\mathcal{U}_j$ into the parent frontal matrix. (a.k.a. *extend-add*)

(a) Fully structured frontal factorization and extend-add.

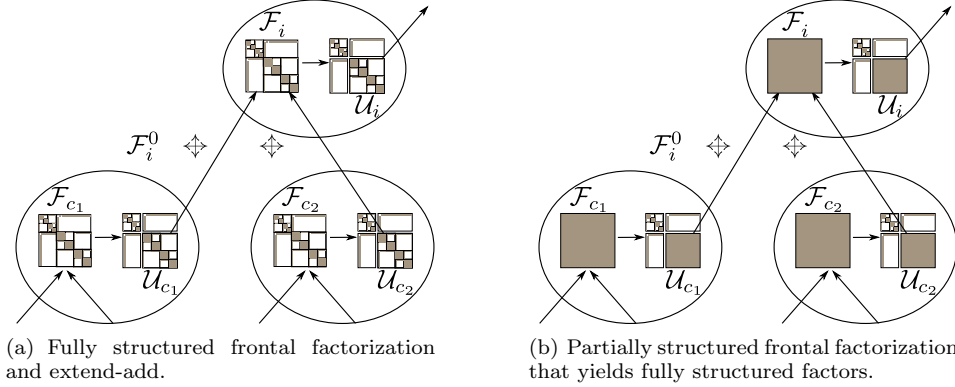(b) Partially structured frontal factorization that yields fully structured factors.

Fig. 2. Partially structured multifrontal method in [Xia 2013] that still yields fully structured factors, since the frontal and update matrices are only used temporarily.

factorization complexity is roughly the same as that of the the fully structured version (increased by just a factor of $O(\log n)$) [Xia 2013]. In addition, the complicated structured extend-add operation may not necessarily be faster than the dense one in practical parallel computations. In practice, this compromised design can lead to an efficient parallel algorithm and implementation.

2.4. Other rank-structured multifrontal approaches

Other forms of low-rank structures have been embedded in multifrontal approaches. One of them is the Hierarchically Off-Diagonal Low-Rank (HODLR) format [Aminfar et al. 2014]. In this approach, the partitioning of the matrix and the underlying tree is the same as what we use in the HSS format. As in the HSS format, the off-diagonal blocks of the input matrix are compressed, and, recursively, the diagonal blocks are partitioned and their off-diagonal blocks are compressed, i.e.,

$$D_{2k-1} = F, \text{ and } D_i = F|_{t_i \times t_i} = \begin{pmatrix} D_{c_1} & U_{c_1}B_{c_1}V_{c_2}^H \\ U_{c_2}B_{c_2}V_{c_1}^H & D_{c_2} \end{pmatrix}$$

However, contrary to the HSS format, the HODLR format does not rely on a nested basis property, i.e., it computes U_i and V_i from scratch at each node i, without using the following relation:

$$U_i = \begin{pmatrix} U_{c_1}R_{c_1} \\ U_{c_2}R_{c_2} \end{pmatrix}, \quad V_i = \begin{pmatrix} V_{c_1}W_{c_1} \\ V_{c_2}W_{c_2} \end{pmatrix}$$

The HODLR approach has been embedded in a multifrontal solver for solving elliptic PDES [Aminfar and Darve 2014]. However, at the time of writing, no parallel implementation is publicly available.

Another low-rank representation that has been implemented in a multifrontal solver is the Block Low-Rank (BLR) format [Weisbecker 2013; Amestoy et al. 2015]. In this approach, a

dense matrix is partitioned using a simple 2D blocking, and every block is compressed independently, as shown in Figure 3. The BLR format has been implemented in the MUMPS multifrontal solve [Amestoy et al. 2015]. We have been communicating with the group working on Block Low-Rank (BLR) techniques to compare their approach with ours. Experiments carried out for the 3D Helmholtz problem described in this paper show that the HSS-based approach exhibits a better asymptotic complexity but also has a larger prefactor and a slightly lower flop-rate than BLR [Weisbecker 2013]. Therefore, it is advantageous to use the HSS technique presented here for very large problems and larger machines, while using BLR might pay off for small or medium-size grids. Extending this comparison to a larger set of problems and applications is work in progress.
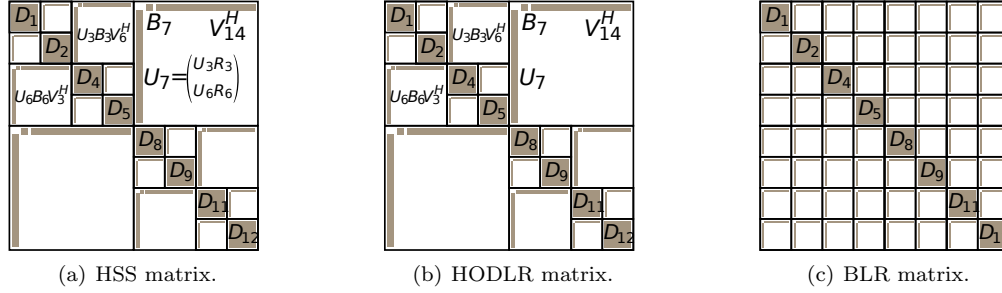

| (a) HSS matrix. | (b) HODLR matrix. | (c) BLR matrix. |

Fig. 3. Pictorial illustrations of an HSS matrix, HODLR matrix, and BLR matrix.

3. RANK PROPERTIES AND OPERATION AND MEMORY COMPLEXITIES

The complexity of HSS-structured multifrontal algorithms relies on the off-diagonal numerical ranks of the frontal matrices. The theoretical analysis of the off-diagonal rank bounds is still limited. Two recent studies on the intermediate Schur complements in the factorizations of certain discretized PDEs are closely related to our method. In the first study [Chandrasekaran et al. 2010], it is shown that, for the finite-difference matrices discretized from constant coefficient elliptic PDEs, the off-diagonal numerical ranks of the intermediate Schur complements is bounded by a constant for 2D problems, but grows with the number of mesh points along one side of the domain for 3D. In another study [Engquist and Ying 2011], Engquist and Ying study the Helmholtz equation with constant velocity field $c(x) = 1$. They prove that the rank grows with $\log k$ for 2D, where k is the size of one side of the mesh. They indicate that for 3D Helmholtz, the rank grows with k, although they do not provide a rigorous proof. Additional results can be found in [Bebendorf and Hackbusch 2003].

Note that both the studies in [Chandrasekaran et al. 2010] and [Engquist and Ying 2011] use the lexicographical ordering of the mesh points (i.e., layer-by-layer, or sweeping order). That is, the global matrix is block banded. Engquist and Ying note that both the Sommerfeld boundary condition and the layer-by-layer order are essential for this low-rank property. With a nested dissection ordering, the ranks may be higher [Engquist and Ying 2011; Martinsson 2011]. However, the ranks are generally not much higher, and may be of the same orders for some situations.

In fact, assume the matrix after one level of nested dissection of a 2D regular mesh looks like

$$A = \begin{pmatrix} A_{11} & & A_{13} \\ & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix}, \tag{5}$$

where the layer-by-layer ordering is applied to both A_{11} and A_{22}. The Schur complement of the two leading blocks is

$$S = A_{33} - A_{31}A_{11}^{-1}A_{13} - A_{32}A_{22}^{-1}A_{23}. \qquad (6)$$

Since $-A_{31}A_{11}^{-1}A_{13}$ and $-A_{32}A_{22}^{-1}A_{23}$ are the contributions to the Schur complement due to the elimination of A_{11} and A_{22}, respectively, they satisfy the same off-diagonal rank properties as in [Engquist and Ying 2011]. Thus, the off-diagonal numerical rank bound of S is at most twice that with the layer-by-layer ordering for the entire A in [Chandrasekaran et al. 2010] and [Engquist and Ying 2011]. Note that this holds regardless of the ordering of A_{11} and A_{22}. Since our goal is to factorize the entire matrix for direct solutions, a nested dissection ordering is better at preserving the sparsity and more suitable for parallel computing.

When there are multiple levels in nested dissection, the above claim similarly holds for some problems. For example, for elliptic problems with a Dirichlet boundary condition, the procedure in (5)–(6) applies to any node with two children in the assembly tree. For our numerical tests below, we fix the frequency in the Helmholtz equation, which makes the rank behaviors similar to the elliptic case for large matrix sizes.

Therefore, with nested-dissection ordering, we make some trade-off between low-rankness with better sparsity. In practice, we observed that, with our partially structured algorithm, the ranks grow as $O(k)$ in 3D (see Tables III), which corroborate Engquist's theory very well.

In [Xia 2013], Xia further considers certain rank patterns of the off-diagonal blocks on top of the rank bounds. That is, when the numerical ranks of the off-diagonal blocks at the hierarchical levels of the frontal matrices satisfy certain patterns, some more optimistic complexity estimates can be obtained. See Table I. As can be seen, in all these cases, the HSS-structured multifrontal factorization is provably faster and uses less memory than the classical algorithm.

Table I. Theoretical off-diagonal rank bounds based on [Chandrasekaran et al. 2010; Engquist and Ying 2011] and the complexities of the standard multifrontal method (MF) based on [Duff and Reid 1983; George 1973] and the HSS-structured multifrontal method (HSSMF) based on [Xia 2013], where k is the mesh size in one dimension, and $n = k^2$ in 2D and $n = k^3$ in 3D.

Problem		r	MF		HSSMF	
			Factorization flops	Memory	Factorization flops	Memory
2D $(k \times k)$	Elliptic	$O(1)$	$O(n^{3/2})$	$O(n \log n)$	$O(n \log n)$	$O(n \log \log n)$
	Helmholtz	$O(\log k)$				
3D $(k \times k \times k)$	Elliptic	$O(k)$	$O(n^2)$	$O(n^{4/3})$	$O(n^{4/3} \log n)$	$O(n \log n)$
	Helmholtz	$O(k)$				

Notice that all the above complexity analysis is based on the off-diagonal rank analysis of the Schur complements which are assumed to be computed exactly. In practice, this is usually not the case, since all the intermediate Schur complements are approximate. Thus, the actual costs of the HSS-structured factorization may be higher than the results in Table I. When appropriate accuracy is enforced in the HSS construction, we expect the off-diagonal numerical ranks of the approximate Schur complements to be close to those of the exact ones.

4. PARALLEL ALGORITHMS: NESTED DISSECTION, SYMBOLIC FACTORIZATION, AND HSS-STRUCTURED MULTIFRONTAL METHOD

In this work, we target our parallel algorithms to the discretized PDEs from the regular 2D and 3D mesh. This is mainly motivated from the need of a fast and memory-efficient solver in one of the applications we are working with—seismic imaging problems involving the

Helmholtz equations. Therefore, our parallelization strategies exploit the spatial geometry and we will demonstrate that this geometry-tailored code is faster than general sparse solvers. It is our work in progress to extend this to a general algebraic approach for arbitrary sparse matrices. In the following subsections, we present our parallel algorithms for all the phases of the solution process. We assume that the total number of processes is P, and is a power of two for ease of exposition.

4.1. Parallel nested dissection and symbolic factorization for regular meshes

For any sparse factorization methods, the order of elimination affects the amount of fill produced in the LU factors and the amount of parallelism exposed. For the regular 2D and 3D meshes considered in this paper, the best ordering method is nested dissection [George 1973], which recursively partitions the (sub)mesh via *separators*. A separator is a small set of nodes whose removal divides the rest of the (sub)mesh into two disjoint pieces. At the end of recursion, the final ordering is performed such that the lower level separator nodes are ordered before the upper level ones, see Figure 4(a) for an illustration. The recursive bisection procedure results in a *complete binary* (amalgamated) elimination tree, a.k.a. *separator tree*, see Figure 4(b). The total number of levels in nested dissection is $l = O(\log_2 n)$. Following this ordering, the LU factorization eliminates the mesh nodes (unknowns) from the lower levels to the upper levels. Nodes may get connected during elimination, see Figure 4(c).

(a) Partition/order with separators.
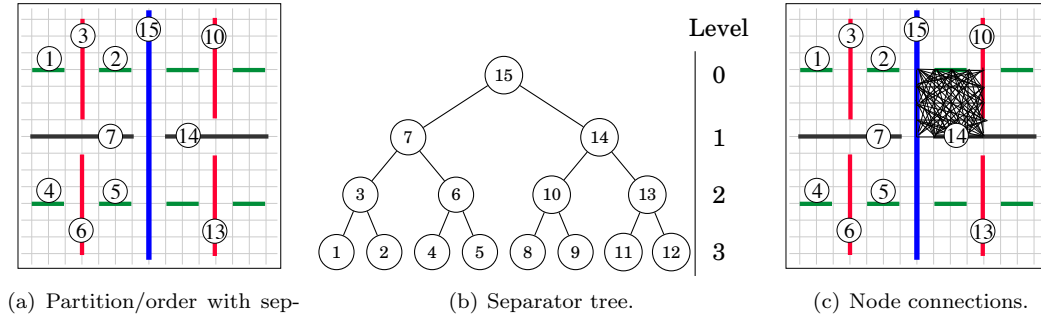
(b) Separator tree.

(c) Node connections.

Fig. 4. Separators in nested dissection of a regular mesh, the corresponding separator tree, and the node connections after the elimination of some lower level mesh points.

Algorithm 2 sketches our parallel nested dissection ordering for a cuboid of dimension $N_x \times N_y \times N_z$. The 2D case is similar.

In this algorithm, the sets of variables associated with the separators are ordered in a postorder of the separator tree. But the algorithm does not specify the variables' ordering within a separator, see line [*]. For an exact sparse factorization, the internal separator ordering has little effect on the fill-in, because the submatrix F_{11} associated with the separator is dense. A common practice is to use a lexicographical order for the mesh points within each 2D separator. However, when HSS is employed, the internal separator ordering affects the amount of compression that can be done for $\mathcal{F}_j$ and the rank distribution of the HSS blocks. Therefore, we apply another nested dissection ordering strategy for the variables within a separator. Figure 4.1 illustrate this for the 3D case, where a separator is a submesh of 2D plane (left figure). We use a partitioning/ordering that respects the geometry and retains locality of the variables within the HSS blocks. The simple example in Figure 4.1 can be extended to a larger number of parts, using an edge-based nested dissection or a Morton ordering [Morton 1966]. In an algebraic context (i.e., for general sparse

---

ALGORITHM 2: Parallel nested dissection ordering for 3D mesh with P processes.

Let $L \leq \log n$ be the number of bisection levels. (Root is at level 0.)

    $L_{par} = \log P \leq L$ be the upper parallel levels.

    $FS(j) =$ list of fully-summed variables (F_{11} in Eqn. (1)) at node j of the tree.

    $CB(j) =$ list of variables in the contribution block (F_{22} in Eqn. (1)) at node j of the tree.

    $w(j) =$ number of vertices to be ordered before node j.

    $order(x, y, z) = i$ means the mesh point (x, y, z) is the ith variable to be eliminated.

(a) Tree generation:

    (a.1) All P processes generate a binary tree `globTree` with L_{par} levels and P leaves
        (`globTree` is duplicated).

    (a.2) Each process p is assigned one leaf and generates its local subtree `locTree` with
        $L - L_{par}$ levels. There are P different `locTree`s, corresponding to different subdomains,
        and the root of each `locTree` is a leaf of the `globTree`.

(b) Compute weight w in postorder of both trees:

    Each node j of `globTree` and `locTree` is associated with a cuboid $[x_1 : x_2, y_1 : y_2, z_1 : z_2]$
    of 6 coordinates, corresponding to frontal matrix $\mathcal{F}_j$.

    $w(j + 1) = w(j) + (x_2 - x_1 + 1)(y_2 - y_1 + 1)(z_2 - z_1 + 1)$.

(c) Nested dissection ordering:

    (c.1) **Redundant work:** all P processes perform top-down L_{par} levels of recursive bisection
        down `globTree`:

        Root node $FS(0) \leftarrow [1 : N_x, 1 : N_y, 1 : N_z]$.

        For each interior node j with cuboid $[x_1 : x_2, y_1 : y_2, z_1 : z_2]$, perform cut along Z, Y and X
        axes in sequence (the following shows only Z-cut):

            $FS(\text{left child of } j) \leftarrow [x_1 : x_2, y_1 : y_2, z_1 : \frac{z_1+z_2}{2} - 1]$;

            $FS(\text{right child of } j) \leftarrow [x_1 : x_2, y_1 : y_2, \frac{z_1+z_2}{2} + 1 : z_2]$;

            $FS(j) \leftarrow [x_1 : x_2, y_1 : y_2, \frac{z_1+z_2}{2}]$; (2D plane separator)

        End For;

    [*] **2D plane separator ordering:** for each vertex (x, y, z) in the separator node j, assign:

        $1 \leq order(x, y, z) \leq (x_2 - x_1 + 1)(y_2 - y_1 + 1)(z_2 - z_1 + 1)$;

        $order(x, y, z) = order(x, y, z) + w(j)$

    (c.2) **Independent work:** each process p independently performs recursive bisection for its
        local subtree `locTree` (similar to c.1)

---

matrix), Amestoy et al. use graph partitioning tools to reorder a halo graph associated with each separator [Amestoy et al. 2015].
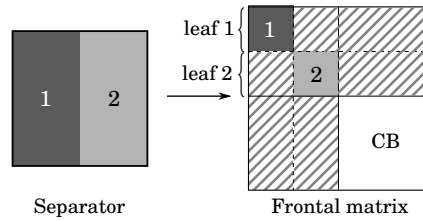

Fig. 5. Correspondence between the ordering of the variables of a plane separator (left) and the variables within the associated frontal matrix (right).

The purpose of the symbolic factorization is to identify the variables associated with each frontal matrix $\mathcal{F}_j$. Since the variables of F_{11} corresponding to the separator already determined from nested dissection, the remaining task is to compute variables in the F_{22} block. The parallel symbolic algorithm proceeds in the same postorder of the separator tree. Each process p first traverses its own `locTree`, then traverses `globTree`. For each node j of

the tree, i.e., a separator, the process finds the "span" of j, i.e., the smallest cuboid which contains j and whose faces touch neighboring separators or boundaries of the domain. There are at most 6 neighboring separators that touches the span of j, and they are ancestors of j in the elimination tree. The variables in the F_{22} block for j are the intersection of each touching separator with the span of j. This is illustrated in Figure 6 where the span of the separator we consider touches 4 other separators of the domain.
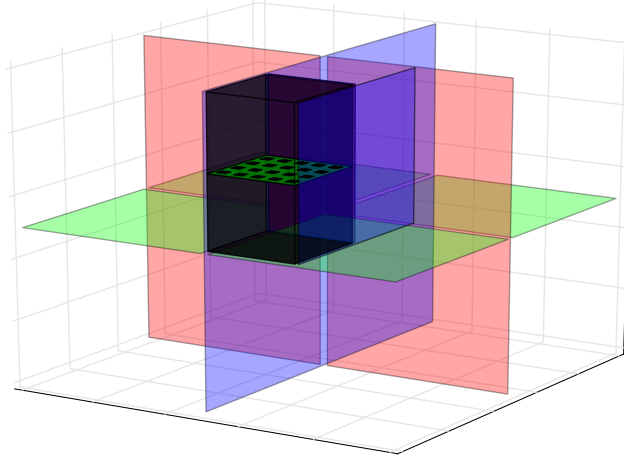

Fig. 6. Span of a separator. We consider the smallest horizontal (dotted green) separator. The (grey) cuboid is its span and touches 4 other separators that are its ancestors.

4.2. Parallel multifrontal method

After the nested dissection ordering, we can design the distributed data distribution and the parallel algorithm fully exploiting the well-structured binary separator/elimination tree. For the regular mesh geometry and the stencil defined on it, the two subparts resulting from each bisection would have similar number of mesh points (equations and variables), and hence the similar amount of computation associated with the elimination process. Therefore, we can assign half of the processes to each subpart of the bisection, which is expected to achieve nearly perfect load balance for the pure multifrontal method. However, for the HSS-structured multifrontal method, this cannot easily accommodate the load imbalance due to rank imbalance after HSS compression, which we comment more in Section 5. Figure 7(a) illustrates this matrix distribution scheme for the example with three levels of bisections. As is shown, the processes are divided into groups, which are organized as a binary tree structure with each tree node being associated with a group of processes. The process numbers of each group are displayed in each tree node. [1] There is a one-to-one correspondence between a group of processes, a separator and the associated frontal matrix $\mathcal{F}_j$.

Once we decide the above matrix distribution scheme, our parallel algorithm can be stated as in Algorithm 3. Algorithm 3 maps readily onto the standard concepts of MPI, `BLACS` [Dongarra and Whaley 1997] and `ScaLAPACK` [Blackford et al. 1997] that are used to implement the algorithm, since the majority of the operations with the frontal and update

---

[1]In parallel computing, 0-based convention is used in the naming of processes, processors, and processes, etc.
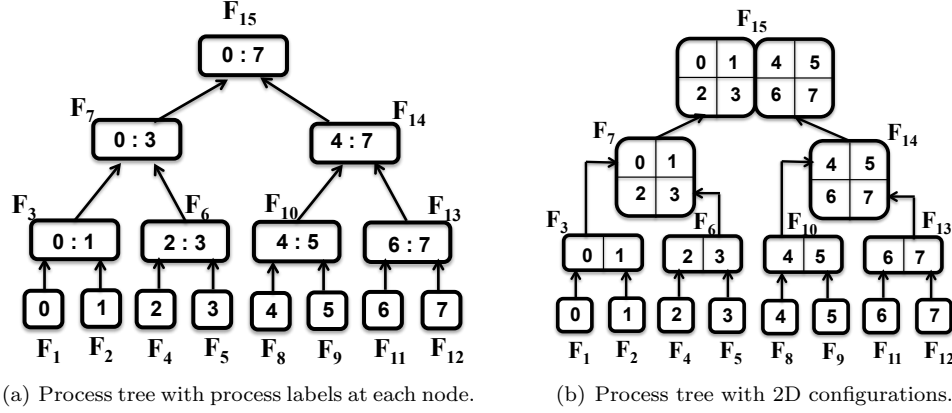
(a) Process tree with process labels at each node.

(b) Process tree with 2D configurations.

Fig. 7. Illustration of the eight processes assigned to the frontal matrices $\mathcal{F}_1$ to $\mathcal{F}_{15}$ along the separator/elimination tree in Figure 4.

ALGORITHM 3: Parallel multifrontal factorization with P processes.

Let $L = \log P$ be the number of bisection levels. Proceeds bottom-up by levels:
(a) At bottom level, L, with P nodes,
    (a.1) In parallel, each process factorizes its local subtree sequentially;
    (a.2) For $i = 0 \ldots P/2 - 1$, a pair of processes $\{2i, 2i+1\}$ performs extend-add in Eqn. (3),
    and distribute the parent frontal matrix to the pair $\{2i, 2i+1\}$.
(b) For levels $l = (L-1) \ldots 0$ in separator tree,
    (b.1) In parallel, a group of 2^{L-l} processes factorizes a frontal matrix $\mathcal{F}_j$ in Eqn. (1);
    (b.2) In parallel (except at root), a group of 2^{L-l+1} processes performs extend-add in Eqn. (3),
    and redistribute the parent frontal matrix to this process group.

---

matrices at each node are dense matrix operations. The key is to use the MPI's process grouping capability. Each process group/communicator corresponds to one node of the process tree, and the groups have the nested structure. In the example of Figure 7, we need to form seven communication groups: {0,1}, {2,3}, {4,5}, {6,7}, {0:3}, {4:7} and {0:7}.

The governing distribution scheme in `BLACS` is a 2D block cyclic matrix layout, in which the user specifies the block size of a submatrix and the shape of the 2D process grid. The blocks of the matrices are then cyclically mapped to the process grid in both row and column dimensions. Furthermore, the processes can be divided into groups (contexts) to work on independent parts of the calculations. Figure 7(b) shows the mapping of the separator tree nodes to 8 processes arranged as 2D grids. We always arrange the process grid as square as possible, i.e., $P \approx \sqrt{P} \times \sqrt{P}$. Since the context of any non-leaf node doubles each child's context, the former can be arranged to combine the two child contexts either side by side or one on top of the other.

During the extend-add operation, we need to assemble the update matrices from two children's nodes to the parent node. Since the parent's context is the union of the two child contexts, the two update matrices should be redistributed before extend-add is performed. We use the `ScaLAPACK` routine `PxGEMR2D` to carry out such a redistribution.

In Table II, we compare the performance of our geometric multifrontal code (denoted MF) to the two widely used parallel sparse direct solvers MUMPS [Amestoy et al. 2006] and SuperLU_DIST [Li and Demmel 2003]. The former uses the multifrontal method while the latter uses a supernodal fan-out (right-looking) approach. For both solvers, we use a

geometric-based nested dissection fill-reducing ordering and we disable MC64. This way, all the solvers factor the same matrix. As shown in the table, the size of the *LU* factors and the number of operations are very close for the three solvers; the slight variations are due to different ways of amalgamating the elimination tree (i.e., setting the size of the frontal matrices or the supernodes). In MUMPS, we disable delayed pivoting; pivoting is done only within fronts, as in our MF code. These settings enable us to have a fair comparison between the three codes. The problem that we use for benchmarking corresponds to a $100 \times 100 \times 100$ 3D grid. From the table, we observe that MF is faster than both MUMPS and SuperLU_DIST. This is because this code is tailored for this particular problem, while MUMPS and SuperLU_DIST are algebraic codes that can solve problems with very different sparsity pattern and numerical properties. MF also shows better strong scaling than MUMPS and similar scaling to SuperLU_DIST. These results demonstrate that our MF code has good performance, and can serve as a nice framework to include HSS-structured techniques, as well as a fair candidate to be compared with our HSS-structured version (so as to demonstrate the benefit of including HSS structures). In the following, we assess the performance of HSS-structured MF as compared with the non-structured MF code.

Table II. Comparison of the parallel performance of the geometric multifrontal code to MUMPS and SuperLU_DIST for a $100 \times 100 \times 100$ grid with different number of MPI processes. "Peak Mem" is the maximum local memory highmark among all the processes.

# procs	Solver	Factor size (GB)	Flop count ($\times 10^{12}$)	Peak Mem. (GB)	Analysis (s)	Facto. (s)	Solve (s)
32	MUMPS	16.9	54.0	0.96	2.9	69.9	0.5
	SuperLU_DIST	16.5	53.3	1.00	5.4	57.4	0.3
	MF	16.6	53.1	1.00	0.2	53.5	0.4
64	MUMPS	16.9	54.0	0.52	3.1	40.5	0.4
	SuperLU_DIST	16.5	53.3	0.67	5.8	35.0	0.3
	MF	16.6	53.1	0.50	0.2	31.6	0.2
128	MUMPS	16.9	58.1	0.36	3.2	28.7	0.4
	SuperLU_DIST	16.5	53.3	0.50	6.3	19.4	0.2
	MF	16.6	53.1	0.26	0.2	18.9	0.2
256	MUMPS	16.9	58.1	0.23	3.4	23.2	0.6
	SuperLU_DIST	16.5	53.3	0.41	6.2	14.0	0.2
	MF	16.6	53.1	0.13	0.1	10.7	0.1

4.3. Parallel HSS algorithms

Parallelizing the HSS algorithms is a challenging task in its own right. In previous work, we have developed efficient parallel algorithms for HSS matrix operations, including construction, ULV factorization and solution; see [Wang et al. 2013] for details. Recall that the HSS computations can also be modeled by a binary tree — HSS tree (see Figure 1(b)). Our parallel HSS algorithms were designed around this tree structure. In particular, we used the same node-to-process mapping principle as is used for the multifrontal separator tree in Figure 7(b). For a dense structured submatrix of size 250,000 from a 3D problem, our parallel HSS solver is over 2x faster than the LU factorization in ScaLAPACK. The factor of the compressed matrix is 70x smaller than that of the standard, full-rank LU factorization. The asymptotic communication complexity (volume and number of messages) is also reduced [Wang et al. 2013].

4.4. Parallel HSS-structured multifrontal method

Now, employing the HSS technique to each separator amounts to applying the parallel HSS algorithms to the frontal matrix that is residing in the process context associated with that separator. Thus, we can organize the parallel algorithm following an outer-inner tree

structure. The outer tree is the multifrontal elimination tree, for which each tree node is embedded with an inner tree, the HSS tree. Figure 8 illustrates such a nested tree structure and the parallel HSS-structured multifrontal method. At several lower levels of the outer tree, the frontal matrices are small and not much compression can be done, therefore, a *switching level* [Xia et al. 2009] in the outer tree is used such that dense frontal matrix operations are employed below the switching level whereas HSS techniques are applied above this level.

The implementation of the outer-inner tree computations can make use of the MPI sub-communicator or the `BLACS` context mechanism. In the following discussion, we will use the example in Figure 9 to illustrate the parallel operations. For example, in Figure 9(a), the four processes {12,13,14,15} form a process group/context for parallel factorization of their assigned separator node. Using this group of four processes, we apply the parallel HSS construction, ULV factorization and solution algorithms, given as input the logical view of the process numbers {0,1,2,3}. The matrix structure of the corresponding separator node is illustrated in Figure 9(b). There is no need for any algorithm changes internal to the parallel HSS algorithm. Similarly, at the parent node, eight processes {8,...,15} are employed for the parallel HSS calculations, with the logical view of the process numbers {0,...,7}.
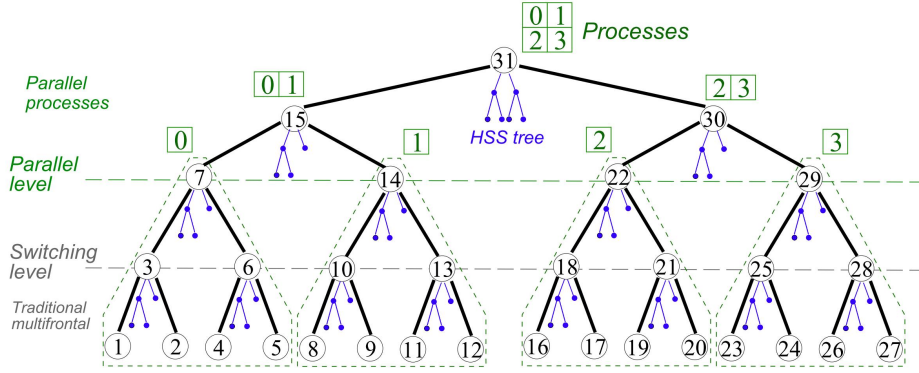

Fig. 8.   Outer tree: elimination tree. Inner tree: HSS tree.

(a) 16 processes forming various groups/contexts.

(b) An HSS-structured frontal matrix $\mathcal{F}_j$ mapped to process context {12,13,14,15}.
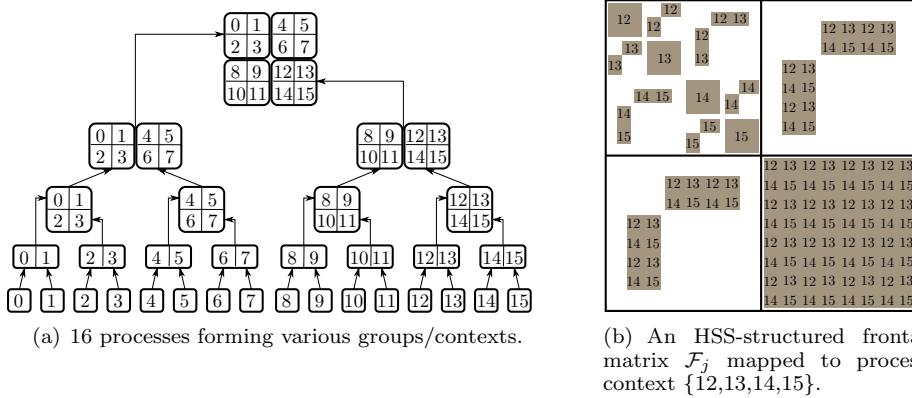
Fig. 9.   Illustration of the parallel HSS-embedded partial factorization and Schur update of a frontal matrix.

The outer tree multifrontal parallelism is the same as depicted in Algorithm 3. With HSS embedding, a major new design of the parallel algorithm is the computation of the contribution block $\mathcal{U}_|$ for the inner HSS tree (Algorithm 1, step (c)) and its interaction with the outer tree parallelism. Recall that in our partially structured multifrontal method, we apply the HSS compression technique only to the blocks with fully summed variables in the frontal matrix, i.e., F_{11}, F_{12} and F_{21}, see Figure 9(b). After HSS compression, the frontal matrix is approximated as:

$$\mathcal{F}_j \approx \begin{bmatrix} H & U_1 B_1 V_2^T \\ U_2 B_2 V_1^T & F_{22} \end{bmatrix} , \tag{7}$$

where H is an HSS representation for F_{11}, and U_1, B_1, V_2, U_2, B_2 and V_1 are the HSS generators at the root of the HSS tree associated with H. Next, we perform parallel ULV factorization to H, yielding a reduced matrix for the HSS form of $\mathcal{F}_j$:

$$\begin{bmatrix} \tilde{D}_k & \tilde{U}_1 B_1 V_2^T \\ U_2 B_2 \tilde{V}_1^T & F_{22} \end{bmatrix} , \tag{8}$$

where, the $\tilde{D}_k, \tilde{U}_1$ and $\tilde{V}_1$ are all of order r, the HSS rank. Here, we employ our parallel HSS construction and partial ULV factorization algorithms developed in [Wang et al. 2013], using four processes in this example, which accomplishes steps (a) and (b) in Algorithm 1.

The next step is decompression to form the dense update matrix $\mathcal{U}_j$ in parallel (Algorithm 1, step (c)): $\mathcal{U}_j = F_{22} - (U_2 B_2 V_1^T) H^{-1} (U_1 B_1 V_2^T)$. As was shown in [Xia 2013], this can be computed cheaply by a low-rank update as follows:

$$\mathcal{U}_j = F_{22} - (U_2 B_2 \tilde{V}_1^T U_k^{-1})(L_k^{-1} \tilde{U}_1 B_1 V_2^T) , \tag{9}$$

with the help of a (small) LU factorization of $\tilde{D}_k = L_k U_k$. In our implementation, we use `ScaLAPACK` and `PBLAS` routines to perform parallel LU factorization, parallel triangular solve (i.e., $L_k^{-1} \tilde{U}_1$) and parallel matrix-matrix multiplication.

The last step is to assemble the contribution block $\tilde{U}_j$ into the parent frontal matrix as part of extend-add. This is the same as what is needed in the pure parallel multifrontal algorithm described in Section 4.2, that is, we can use `BLACS` routine `PxGEMR2D` to first redistribute $\tilde{U}_j$ from four processes $\{12,13,14,15\}$ onto eight processes $\{8, \dots, 15\}$, before the addition operation.

Our code includes pivoting within the dense frontal matrices. No pivoting is needed to factorize the HSS frontal matrices due to the ULV factorization procedure which computes a sequence of orthogonal local factorizations. The current code does not use delayed pivoting across the children/parent fronts. In our applications, factorizations without delayed pivoting preserve the practical background of the problems and are enough to observe stability in practice [Wang et al. 2010; Wang et al. 2011; Wang et al. 2012]. In our future parallel algebraic structured solver for more general sparse matrices, we plan to include static pivoting as in SuperLU_DIST [Li and Demmel 2003] or randomized pivoting as in [Xin et al. 2013].

5. RESULTS

In this section, we present the performance results of our parallel geometric HSS-structured multifrontal solver when it is used to solve the Helmholtz equation of the following form:

$$\left(-\Delta - \frac{\omega^2}{v(x)^2}\right) u(x,\omega) = s(x,\omega), \tag{10}$$

where Δ is the Laplacian, ω is the angular frequency, $v(x)$ is the seismic velocity field, and $u(x,\omega)$ is called the time-harmonic wavefield solution to the forcing term $s(x,\omega)$. Helmholtz

equations arise frequently in real applications such as seismic imaging, where the simplest case of the acoustic wave equation is of the form (10). Here, we focus on the 3D geometry and a 27-point discretization of the Helmholtz operator on 3D regular domains. This discretization often leads to very large sparse matrices $\mathcal{A}$ which are highly indefinite and ill-conditioned. We use a fixed frequency $f = 10Hz$ ($\omega = 2\pi f$), a sampling rate of about 15 points per wavelength, wavelength $L = 150$ meters, step size $h = 10$ meters, and the PML (Perfectly Matched Layer) boundary condition. In order to study the scaling of the parallel algorithm, we increase the the number of discretization point k in each dimension, so that the domain size in each dimension is $k \times h$ meters.

It has been observed that, in the direct factorization of $\mathcal{A}$, the dense intermediate matrices may be compressible [Engquist and Ying 2011; Wang et al. 2010]. In this application, the discretized matrix has complex values, but we need to work only in single precision arithmetic as it is what is used most of the time in the full wave inversion.

We carry out the experiments on the Cray XC30 (`edison.nersc.gov`) at the National Energy Research Scientific Computing Center (NERSC). Each node has two sockets, each of which is populated with a 12-core Intel "Ivy Bridge" processor at 2.4GHz. There are 24 cores per node. Each node has 64GB memory. The peak performance of each core is 19.2 Gflops/core.

We report the detailed parallel performance results in Table III. We also show in Figure 10 how the size of the factors and the number of operations of the HSS-structured factorization grow as a function of the problem size. The grid sizes range from $100 \times 100 \times 100$ to $600 \times 600 \times 600$ (i.e, $n = 216$ millions). The number of cores ranges from 16 to 16,384. For each problem, we compare the runs of the same code using a regular multifrontal mode (by setting the switching level to 0) and the HSS-structured mode (where the switching level is chosen according to the rules in [Xia 2013]). In each case, we perform one factorization and one solution step with a single right-hand side, followed by at most 5 steps of iterative refinement to the solution. The convergence criterion is that the componentwise backward error $\max_i \frac{|Ax-b|_i}{(|A||x|+|b|)_i}$ [Oettli and Prager 1964] is less than or equal to 5×10^{-7}. In case the error does not go down to the desired accuracy, we keep the best residual.

We report and compare the following performance metrics:

— Time: the runtime and the flop rate of the factorization, the solution, and the iterative refinement phases. For the HSS-structured factorization, we also report the time spent in the rank-revealing QR, which is expected to be a dominant operation.
— Memory: the total size of the factors and the total peak of memory including the factors and the active memory. The latter is the dominant part of the memory footprint of the solver.
— Communication characteristics: the volume, the number of messages, and the time spent in communication. These are collected using the IPM performance profiling tool [Fuerlinger et al. 2010]. We did not collect data for the biggest problem as the log files become too large for very long runs with many cores.
— Accuracy: we measure the normwise relative residual $\frac{||Ax-b||}{||b||}$ and the componentwise backward error $\max_i \frac{|Ax-b|_i}{(|A||x|+|b|)_i}$ before and after iterative refinement.

One can notice that, except for the smallest 3D problem, the HSS-structured factorization is always faster than the regular, uncompressed multifrontal factorization. The new code is up to 2x faster than the pure multifrontal code (e.g., with the mesh size 300^3). The structured factor size is always smaller than that of the multifrontal code and is up to about almost 6x smaller (e.g. the mesh size 600^3). The flop rates of the HSS-structured code are lower because the HSS kernels operate on the smaller blocks, while the multifrontal kernels mostly perform large BLAS-3 operations. The size of the factors is significantly reduced but

Table III. Experimental results on 3D regular domains. We compare a regular (full-rank) multifrontal solution process ("MF" rows in the table) with our HSS-structured solver ("HSSMF"). Communication statistics are collected with IPM except for the last problem (too large).

k (mesh: $k \times k \times k; n = k^3$)		100	200	300	400	600
Number of processors P		64	256	1,024	4,096	16,384
Levels of Nested Dissection		15	18	20	21	23
Switching level		5	8	10	11	14
MF	Factorization (s)	53.2	842.0	2436.9	4217.6	6564.3
	Gflops/s	998.7	4138.0	16454.8	53587.2	393694.1
	Solution (s)	0.1	0.6	1.0	1.6	6.7
	Gflops/s	17.8	300.6	1556.9	4977.7	25541.1
	Iterative refinement (s)	0.2	0.6	1.1	2.5	44.8
	Steps	1	1	1	1	2
	Factors size (GB)	16.6	280.0	1450.1	4636.1	23788.9
	Total peak (GB)	262.0	434.7	2234.9	7119.5	36373.4
	Communication volume (GB)	68.0	2149.3	22790.5	135888.0	-
	Number of messages (millions)	4.1	71.9	576.2	2883.5	-
	Communication time(s)	5.1	42.1	136.0	518.8	-
	After solution $\frac{\|Ax-b\|}{\|b\|}$	1.0×10^{-4}	3.6×10^{-4}	1.2×10^{-2}	1.7×10^{-3}	2.8×10^{-2}
	$\max_i \frac{|Ax-b|_i}{(|A||x|+|b|)_i}$	2.9×10^{-3}	2.1×10^{-3}	3.5×10^{-3}	1.5×10^{-3}	6.5×10^{-3}
	After refinement $\frac{\|Ax-b\|}{\|b\|}$	8.3×10^{-6}	1.2×10^{-5}	1.5×10^{-5}	1.8×10^{-5}	2.2×10^{-5}
	$\max_i \frac{|Ax-b|_i}{(|A||x|+|b|)_i}$	1.5×10^{-7}	1.5×10^{-7}	3.6×10^{-7}	1.8×10^{-7}	1.6×10^{-7}
HSSMF	Compression and factorization (s)	56.9	598.6	1214.2	2477.1	5136.8
	Gflops/s	467.2	1408.5	5116.9	11084.0	43399.2
	Min RRQR time(s)	21.7	227.0	469.1	742.8	1603.3
	Max RRQR time(s)	32.6	415.8	860.3	1375.7	3430.7
	Solution (s)	0.2	0.7	2.8	10.9	63.2
	Gflops/s	58.0	178.0	166.4	115.7	68.3
	Iterative refinement(s)	1.4	13.2	13.2	68.6	350.0
	Steps	5	5	5	5	5
	Factors size (GB)	10.8	115.8	433.4	1189.0	4053.5
	Total peak (GB)	250.3	366.7	1738.4	5352.5	25287.2
	Communication volume (GB)	81.8	1794.9	14129.2	71261.2	-
	Number of messages (millions)	13.2	160.6	1175.0	6147.7	-
	Communication time(s)	7.8	49.1	157.8	583.8	-
	Largest rank found in compression	518	1036	1822	2502	5214
	After solution $\frac{\|Ax-b\|}{\|b\|}$	1.5×10^{-2}	3.0×10^{-2}	6.3×10^{-2}	6.5×10^{-2}	1.6×10^{-1}
	$\max_i \frac{|Ax-b|_i}{(|A||x|+|b|)_i}$	8.4×10^{-1}	8.9×10^{-1}	9.2×10^{-1}	8.3×10^{-1}	8.9×10^{-1}
	After refinement $\frac{\|Ax-b\|}{\|b\|}$	8.3×10^{-6}	1.2×10^{-5}	1.5×10^{-5}	1.8×10^{-5}	2.2×10^{-5}
	$\max_i \frac{|Ax-b|_i}{(|A||x|+|b|)_i}$	3.8×10^{-5}	8.5×10^{-5}	6.0×10^{-4}	1.3×10^{-3}	1.3×10^{-3}

the gains on the maximum peak of memory are not as large because the current code is not fully structured yet: we do not apply the HSS compression on the contribution blocks, as mentioned in Section 2.3. Studying and implementing a parallel fully-structured solver is work in progress.

We observe that the flop rates are disappointing for the largest problem, with both the multifrontal and the HSS-structured modes. We are currently investigating this issue with Cray. On an older system (Cray XE6) the HSS-structured code using 16,384 processes we achieved 23.2 TFlops/s, instead of 6.7 TFlops/s here.

The communication volume is also reduced when the HSS kernels are used, almost 2x reduction in the case of 400^3. The number of messages is larger for the HSS-structured factorization; this is because the HSS kernels perform many redistributions of the inter-
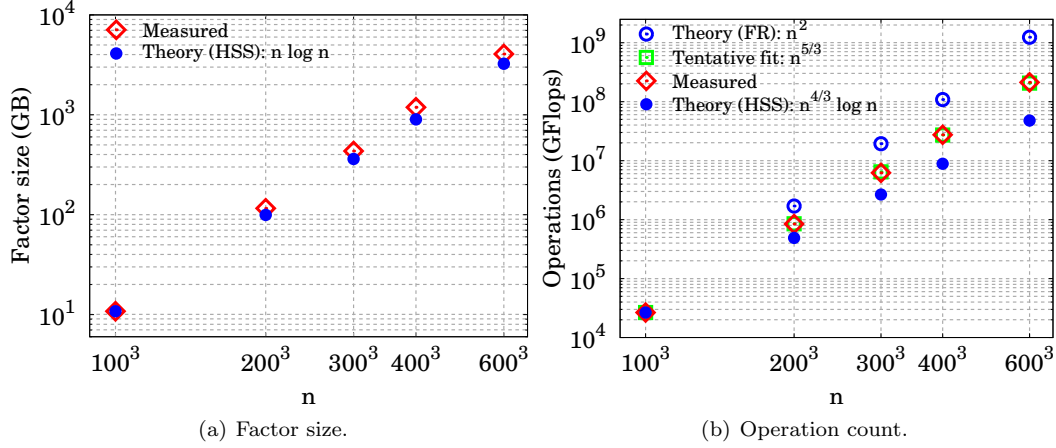
(a) Factor size.   (b) Operation count.

Fig. 10. Size of the ULV factors and operation count of the HSS-structured factorization for 3D problems.

mediate HSS blocks during the compression phase. It is important to bear in mind that although there are more messages, these messages are not on the critical path in the parallel execution, meaning that most of them are transferred *simultaneously* by different pairs of processes. Therefore, the actual communication time is comparable between the two solvers, and the time-to-solution is still much reduced for the new solver.

In terms of accuracy, the HSS-structured code is as good as the pure multifrontal code in the normwise relative residual measure. With iterative refinement, the componentwise backward error is further reduced. One can notice that the pure multifrontal factorization does not yield residuals close to the machine precision without iterative refinement. This may be because we do not perform any scaling (equilibration) of the system before the factorization. Also, our code does not use delayed pivots across the children/parent fronts. But again, only modest accuracy is desired for this class of problems and delayed pivoting is generally not necessary.

According to Figure 10, the size of the factors and the number of operations of the HSS-structured factorization are rank-dependent, as discussed in the previous section. In practice, the rank patterns may not exactly follow the prediction due to the approximation of the frontal and update matrices. We observed from our partially-structured code that the maximum rank found in compression is proportional to the mesh size k, see the corresponding row in Tables III. From this rank pattern, the factor size follows the predicted complexity $o(n \log n)$. The number of operations seems to follow a $O(n^{5/3})$ behavior; this is larger than the $O(n^{4/3} \log n)$ asymptotic behavior of a fully-structured factorization but smaller that the $O(n^2)$ complexity of a classical non-structured factorization.

We carefully analyzed the performance of the HSS-structured factorization, and found that the rank-reveling QR time exhibits serious load imbalance, which dominates the HSS construction time. For example, we observed up to 3x imbalance of $RRQR$ time for some problems. This is due to the imbalance in the ranks identified during the HSS compression of a frontal matrix. We illustrate this in Figure 11, where we show the ranks of the top two levels of the HSS tree corresponding to the root node (topmost separator) of the 200^3 problem. There are four processes working at each leaf node. The minimum rank is 413 and the maximum is 607. Table IV shows the HSS construction time and the $RRQR$ time imbalance. Our future work is to explore the other separator ordering methods and data distribution schemes to mitigate the load imbalance problem.

In Table V, we report experimental results obtained when solving different variants of the Helmholtz equation: Laplacian (Helmholtz equation with zero frequency) and Optical Diffu-
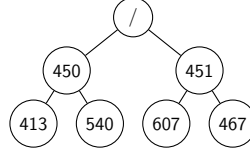
Fig. 11. Ranks after $RRQR$ compression of the root node of the $200 \times 200 \times 200$ problem.

Table IV. Parallel HSS construction time in seconds for the topmost separator of the 200^3 problem.

	HSS time	HSS rank	Min time in $RRQR$	Max time in $RRQR$
edge-based ND	32.3	646	11.0	30.7

sion (Helmholtz equation with imaginary frequency). One can observe that these equations have different compressibility properties. The Laplace equation and the Optical Diffusion problem yield lower HSS rank, smaller factors and fewer operations. For these two problems, our HSS-based factorization yields a larger speedup over the non-structured factorization than what we obtain for the general Helmholtz problem.

Table V. Solution of three variants of the Helmholtz equation, using a $300 \times 300 \times 300$ grid and 1,024 MPI tasks.

		Laplace	Optical Diffusion	Helmholtz
		$\omega = 0$	$\omega = 10 \cdot \pi \cdot i$	$\omega = 10 \cdot 2\pi$
MF	Operations	$4.0 \cdot 10^{16}$		
	Factor size (GB)	1,450.1		
HSS	Max rank	1317	1310	1822
	Operations	$5.3 \cdot 10^{15}$	$4.9 \cdot 10^{15}$	$7.0 \cdot 10^{15}$
	Factor size (GB)	431.1	423.5	433.4
Speedup HSS vs MF		2.3	2.4	2.0

6. CONCLUSIONS

We presented a parallel structured, geometry-based multifrontal sparse solver using HSS representations for low-rank approximation. The novelty of our method is to use two layers of tree parallelism to exploit the HSS structure in each dense frontal matrix, and to employ the HSS factorization and solve algorithms in place of the normal LU factorization for the front. We developed a hierarchically parallel algorithm to weave together both parallel multifrontal elimination (outer-tree parallelism) and parallel HSS factorization (inner-tree parallelism). Parallel nested dissection and symbolic factorization are also studied to preserve the geometry and to benefit the later low-rank compression. We tested our new parallel HSS-structured multifrontal code using up to 16,384 cores, and demonstrated great advantages over the standard parallel multifrontal method. The advantage is especially significant in terms of the memory for large 3D problems.

This is the first parallel algorithm design and implementation that incorporates the HSS structure in the sparse multifrontal solver, with the actual performance results on a real parallel machine.

ACKNOWLEDGMENTS

REFERENCES

Patrick Amestoy, Cleve Ashcraft, Olivier Boiteau, Alfredo Buttari, Jean-Yves L'Excellent, and Clément Weisbecker. 2015. Improving Multifrontal Methods by Means of Block Low-Rank Representations. *SIAM J. Scientific Computing* 37, 3 (2015). http://dx.doi.org/10.1137/120903476

P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet. 2006. Hybrid scheduling for the parallel solution of linear systems. *Parallel Comput.* 32, 2 (2006), 136–156.

Amirhossein Aminfar, Sivaram Ambikasaran, and Eric Darve. 2014. A Fast Block Low-Rank Dense Solver with Applications to Finite-Element Matrices. *CoRR* abs/1403.5337 (2014).

Amirhossein Aminfar and Eric Darve. 2014. A Fast Sparse Solver for Finite-Element Matrices. *CoRR* abs/1410.2697 (2014). http://arxiv.org/abs/1410.2697

M. Bebendorf and W. Hackbusch. 2003. Existence of $\mathcal{H}$-matrix approximants to the inverse FE-matrix of elliptic operators with L^∞ coefficients. *Numer. Math.* 95 (2003), 1–28.

T. Bella, Y. Eidelman, I. Gohberg, and V. Olshevsky. 2008. Computations with quasiseparable polynomials and matrices. *Theoret. Comput. Sci.* 409 (2008), 158–179.

L. S. Blackford, J. Choi, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. 1997. *ScaLAPACK Users' Guide.* SIAM, Philadelphia. 325 pages. http://www.netlib.org/scalapack.

S. Börm, L. Grasedyck, and W. Hackbusch. 2003. Introduction to hierarchical matrices with applications. *Eng. Anal. Bound. Elem* 27 (2003), 405–422.

S. Börm and W. Hackbusch. 2001. Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices. *Technical report, Leipzig, Germany: Max Planck Institute for Mathematics* 86 (2001).

S. Chandrasekaran, P. Dewilde, M. Gu, T. Pals, X. Sun, A.-J. van der Veen, and D. White. 2005. Some fast algorithms for sequentially semiseparable representations. *SIAM J. Matrix Anal. Appl.* 27 (2005), 341–364.

S. Chandrasekaran, P. Dewilde, M. Gu, and N. Somasunderam. 2010. On the numerical rank of the off-diagonal blocks of Schur complements of discretized elliptic PDEs. *SIAM J. Matrix Anal. Appl.* 31 (2010), 2261–2290.

S. Chandrasekaran, M. Gu, and T. Pals. 2006. A fast ULV decomposition solver for hierarchically semiseparable representations. *SIAM J. Matrix Anal. Appl.* 28 (2006), 603–622.

J. Dongarra and R. C. Whaley. 1997. *A User's Guide to the BLACS v1.1.* LAPACK Working Note #94. http://www.netlib.org/blacs.

I. S. Duff and J. K. Reid. 1983. The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Trans. Math. Software* 9 (1983), 302–325.

Y. Eidelman and I. Gohberg. 1999. On a new class of structured matrices. *Integr. Equat. Operat. Theor.* 34 (1999), 293–324.

B. Engquist and L. Ying. 2011. Sweeping preconditioner for the Helmholtz equation: hierarchical matrix representation. *Commun. Pure Appl. Math.* 64 (2011), 697–735.

Karl Fuerlinger, Nicholas J Wright, and David Skinner. 2010. Effective performance measurement at petascale using IPM. In *Parallel and Distributed Systems (ICPADS), 2010 IEEE 16th International Conference on.* IEEE, 373–380.

J. A. George. 1973. Nested dissection of a regular finite element mesh. *SIAM J. Numer. Anal.* 10 (1973), 345–363.

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

W. Hackbusch, L. Grasedyck, and S. Börm. 2002. An introduction to hierarchical matrices. *Math. Bohem.* 127 (2002), 229–241.

W. Hackbusch, B. Khoromskij, and S. A. Sauter. 2000. On $\mathcal{H}^2$-matrices. *Lectures on applied mathematics (Munich, 1999), Springer, Berlin* (2000), 9–29.

W. Hackbusch and B. N. Khoromskij. 2000. A sparse $\mathcal{H}$-matrix arithmetic. Part-II: Application to multidimensional problems. *Computing* 64 (2000), 21–47.

R. Li and Y. Saad. 2013. Divide and conquer low-rank preconditioners for symmetric matrices. *SIAM J. Scientific Computing* 35, 4 (2013), A2069–A2095.

X. S. Li and J. W. Demmel. 2003. SuperLU_DIST: A Scalable Distributed-Memory Sparse Direct Solver for Unsymmetric Linear Systems. *ACM Trans. Math. Software* 29, 2 (June 2003), 110–140.

J. W. H. Liu. 1990. The role of elimination trees in sparse factorization. *SIAM J. Matrix Anal. Appl.* 18 (1990), 134–172.

J. W. H. Liu. 1992. The multifrontal method for sparse matrix solution: Theory and practice. *SIAM Rev.* 34 (1992), 82–109.

P. G. Martinsson. 2011. A Fast Direct Solver for a Class of Elliptic Partial Differential Equations. *J. Scientific Computing* 38, 3 (2011), 316–330.

G. M. Morton. 1966. A computer Oriented Geodetic Data Base; and a New Technique in File Sequencing. *Technical Report, Ottawa, Canada: IBM Ltd* (1966).

W. Oettli and W. Prager. 1964. Compatibility of approximate solution of linear equations with given error bounds for coefficients and right hand sides. *Num. Math.* 6 (1964), 405–409.

R. Vandebril, M. Van Barel, G. Golub, and N. Mastronardi. 2005. A bibliography on semiseparable matrices. *Calcolo* 42 (2005), 249–270.

S. Wang, M. V. De Hoop, and J. Xia. 2010. Seismic inverse scattering via Helmholtz operator factorization and optimization. *J. Computat. Phys.* 229 (2010), 8445–8462.

S. Wang, M. V. de Hoop, and J. Xia. 2011. On 3D modeling of seismic wave propagation via a structured massively parallel multifrontal direct Helmholtz solver. *Geophys. Prospect.* 59 (2011), 857–873.

S. Wang, M. V. de Hoop, J. Xia, and X. S. Li. 2012. Massively parallel structured multifrontal solver for time-harmonic elastic waves in 3D anisotropic media. *Geophys. J. Int.* 91 (2012), 346–366.

S. Wang, X.S. Li, J. Xia, Y. Situ, and M.V. de Hoop. 2013. Efficient Scalable Algorithms for Solving Linear Systems with Hierarchically Semiseparable Structures. *SIAM J. Scientific Computing* 35, 6 (2013), C519–C544.

C. Weisbecker. 2013. *Improving multifrontal methods by means of algebraic block low-rank representations.* Ph.D. Dissertation. Institut National Polytechnique de Toulouse.

J. Xia. 2012. On the complexity of some hierarchical structured matrix algorithms. *SIAM J. Matrix Anal. Appl.* 33 (2012), 388–410.

J. Xia. 2013. Efficient structured multifrontal factorization for general large sparse matrices. *SIAM J. Sci. Comput.* 35, 2 (2013), A832–A860.

J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li. 2009. Superfast multifrontal method for large structured linear systems of equations. *SIAM J. Matrix Anal. Appl.* 31 (2009), 1382–1411.

J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li. 2010. Fast algorithms for hierarchically semiseparable matrices. *Numer. Linear Algebra Appl.* 2010 (2010), 953–976.

Z. Xin, J. Xia, M. V. de Hoop, S. Cauley, and V. Balakrishnan. 2013. A structured multifrontal method for nonsymmetric sparse matrices and its applications. *Purdue GMIG Report* (April 2013).