A TECHNIQUE FOR UPDATING HIERARCHICAL FACTORIZATIONS OF INTEGRAL OPERATORS

VICTOR MINDEN*, ANIL DAMLE†, KENNETH L. HO‡, AND LEXING YING§

Abstract. We present a method for updating certain hierarchical factorizations for solving linear integral equations with elliptic kernels. In particular, given a factorization corresponding to some initial geometry or material parameters, we can locally perturb the geometry or coefficients and update the initial factorization to reflect this change with asymptotic complexity that is poly-logarithmic in the total number of unknowns and linear in the number of perturbed unknowns. We apply our method to the recursive skeletonization factorization and hierarchical interpolative factorization and demonstrate scaling results for a number of different 2D problem setups.

Key words. factorization updating, local perturbations, hierarchical factorizations, integral equations

AMS subject classifications. 65R20, 15A23, 65F30

1. Introduction. In engineering and the physical sciences, many fundamental problems of interest can be expressed as an integral equation (IE) of the form

\[ a(x)u(x) + b(x) \int_{\Omega} K(x,y)c(y)u(y) \, d\Omega(y) = f(x), \quad x \in \Omega \subset \mathbb{R}^d, \]

where \( a(x), b(x), \) and \( c(x) \) are given functions typically representing material parameters, \( u(x) \) is the unknown function to be determined, \( K(x,y) \) is some integral kernel, \( f(x) \) is some known right-hand side, and the dimension \( d = 2 \) or \( 3 \). Typically, \( K(x,y) \) is associated with some underlying elliptic PDE (i.e., it is the Green’s function or its derivative) and it thus tends to be singular at \( x = y \).

Discretizing the integral operator in (1.1) with \( N \) degrees of freedom (DOFs) via the Nyström, collocation, or Galerkin method (see [18]) reduces our problem to solving a linear system,

\[ Au = f, \]

where the matrix \( A \in \mathbb{C}^{N \times N} \) is dense. We make the assumption that subblocks of \( A \) corresponding to physically separated sets of unknowns are numerically low-rank. For example, this is well-known to be the case for elliptic kernels where \( K(x,y) \) for \( \|x-y\| \) large is sufficiently smooth. This observation is the cornerstone of a number of fast (linear or quasi-linear time complexity) direct algorithms for solving (1.2), which typically employ some form of hierarchical subdivision of space to expose and take advantage of the inherent physical structure of the underlying problem.

1.1. Background. Fast direct solvers for solving the linear systems arising from discretized integral equations via the compression of low-rank blocks exist in a number

---

*Institute for Computational and Mathematical Engineering, Stanford University, Stanford, CA 94305. Email: vminden@stanford.edu
†Institute for Computational and Mathematical Engineering, Stanford University, Stanford, CA 94305. Email: damle@stanford.edu
‡Department of Mathematics, Stanford University, Stanford, CA 94305. Email: klho@stanford.edu
§Department of Mathematics and Institute for Computational and Mathematical Engineering, Stanford University, Stanford, CA 94305. Email: lexing@math.stanford.edu
of different incarnations. The seminal work in such compressed representations is the \( \mathcal{H} \)- and \( \mathcal{H}^2 \)-matrices of Hackbusch et al., which provide an important theoretical framework but in practice exhibit large constant factors in the asymptotic scaling.

A framework designed more explicitly for solving elliptic integral equations by compressing low-rank blocks in a hierarchical fashion dates back at least to [20] based on observations in [24] and [12], and has since been utilized and refined by a number of different authors (see, e.g., [11] [15] [17]). We refer to these methods as "skeletonization-based" since at their core they employ the interpolative decomposition for compression using the skeletonization process described in [6]. Conceptually, these methods are closely related to methods for systems involving so-called hierarchically semi-separable (HSS) matrices (see, e.g., [3] [4] [25]), and recent work has explicitly combined the HSS and skeletonization frameworks [7]. Notable related schemes employing similar ideas include [5] [1] [2].

1.2. Problem statement. In this paper, we consider solving a sequence of problems of the form (1.1) that are related via local perturbations. For our purposes, a perturbation can consist of the modification, addition, or removal of unknowns. For example, two broad categories of problems that fit this description are

(i) localized geometric perturbations (see Figure 1.1) wherein the domain of integration is modified, and
(ii) localized coefficient perturbations, wherein the material parameters \( a(x), b(x), \) or \( c(x) \) are modified in a local region.

Sequences of problems fitting these categories can arise, for example, in the case of design problems where the physical system described by the linear operator is a device that we want to design to optimize some objective function.

In the case where the overall updates performed are sufficiently small and the number of DOFs does not change, it is in principle possible to use a factorization of the base system as a preconditioner for an iterative method such as CG [14] or GMRES [23]. However, the quality of this preconditioner will in practice naturally vary depending on the nature of the update.

If the unknowns affected by the modifications are restricted to a fixed set of relatively small size, then an efficient way to solve the sequence of problems is to keep track of the updates and use the Sherman-Morrison-Woodbury formula, taking advantage of the initial factorization of the base system [11]. If the collection of unknowns to be modified across time comprises any substantial fraction of the total
unknowns, even if each individual update is only comprised of a small number of unknowns, this is not a viable strategy. Furthermore, such a procedure does not yield a factorization of the perturbed system.

1.3. Contribution. In this work we present a methodology for efficiently updating certain forms of the aforementioned hierarchical factorizations. More specifically, if a factorization has been computed for a fixed problem and the physical problem is perturbed in some way we may update the existing factorization to construct a valid factorization for the perturbed problem. Consequently, at each step in a sequence of perturbations we maintain a valid factorization for the matrix representing the current form of the problem. The explicit construction of a new factorization directly allows for the efficient solution of linear systems.

The idea of updating matrix factorizations to solve sequences of related systems is not a new one. For example, in the linear programming community it is common practice to maintain an LU factorization of a sparse matrix $A$ that permits the addition or deletion of rows/columns of $A$, or a general rank-one update [8]. Further, it is well-known how to update the QR factorization of a matrix after any of those same operations, see section 12.5 of [10], or [13] for a more-detailed description of a high-performance implementation for the case of addition of rows.

In the case where the unknowns that are to be modified are fixed and known a priori, it is possible to order the unknowns in an LU decomposition such that those that will be modified are eliminated last [22], which can be used for IE design problems where only one small portion of the geometry is to be changed across all updates.

The updating techniques described above, however, do not apply to fast hierarchical factorizations such as those in which we are interested, which necessitates the development of new methodologies for updating factorizations of such structure.

2. Preliminaries. The updating ideas presented in this paper apply, in principle, to many of the existing fast algorithms for IEs. For concreteness, we present them in the context of quadtree-based generalized triangular factorizations as presented in [17], in contrast to the telescoping decompositions previously discussed in, e.g., [20, 15, 9]. For brevity, we restrict our discussion to solving quasi-1D problems (i.e., curves in the plane) and true 2D problems. We begin by reviewing these factorizations and the necessary linear algebra preliminaries.

2.1. Hierarchical decomposition of space to expose redundancies. Given a set of DOFs discretizing (1.1), fast algorithms for solving the linear system require a way to expose compressible interactions between sets of DOFs. Figure 2.1 shows spatial and graphical representations of a quadtree used to subdivide the bounding box containing the unknowns discretizing our domain $\Omega$.

We follow the notation that the root node corresponds to level $\ell = 0$ and each successive level of the tree denotes the set of children of any node at the preceding level, e.g., the nodes on level $\ell = 1$ are the four children of the node at level $\ell = 0$. Given this spatial decomposition of the DOFs, we have explicitly exposed the spatial relationship between sets of points and can construct a factorization using low-rank approximations to describe interactions between DOFs in non-overlapping boxes.

When constructing a quadtree on a domain $\Omega$, we will hereinafter assume that the tree is constructed such that leaf-level boxes each contain a number of unknowns bounded by occupancy parameter $n_{occ}$ independent of $N$. In other words, the tree is adaptive and no single node gets “too big”. This is essential for the complexity estimates of most fast algorithms employing this structure and, indeed, for the updating...
Fig. 2.1: Spatial partitioning of a domain (left) and the corresponding full quadtree (right). Here, box $b_0$ contains the entire domain and corresponds to the root node, and each subsequent level is given by uniform subdivision of boxes on the previous level.

technique of this paper. We note that this assumption implies that construction of the hierarchical decomposition is a super-linear process with complexity $O(N \log N)$, but in practice constructing the quadtree does not significantly contribute to runtime.

For the remainder of this section, we will use $p$ and $q$ to denote sets of indices associated with distinct boxes in our tree. Then, for the system matrix $A$, $A_{p,q}$ and $A_{q,p}$ are subblocks of $A$ corresponding to interactions between the DOFs associated with the index sets $p$ and $q$. We use the notation $A_{p,q}$ to refer to the submatrix of $A$ given by subselecting only the columns of $A$ but keeping all rows, and similarly the notation $A_{q,p}$ refers to the submatrix of $A$ given by subselecting the rows of $A$ but keeping all columns. We will further define the set

\begin{equation}
B_\ell = \{ b \mid b \text{ is a box on level } \ell \text{ of the quadtree} \},
\end{equation}

which will be convenient later.

### 2.2. The interpolative decomposition.

Due to the fact that kernel interactions between the DOFs in a box and the DOFs outside that box are numerically low-rank, fast algorithms for solving (1.2) use some form of compression to approximate blocks of $A$ corresponding to interactions between DOFs in non-overlapping boxes. One such method is the **interpolative decomposition** (ID), which we define below in a non-standard fashion.

**Definition 2.1.** Consider a general matrix $A$ with rows indexed by the index set $p$ and columns indexed by the index set $q$. An interpolative decomposition with tolerance $\epsilon > 0$ is a partitioning of $q$ into index sets associated with so-called ‘skeleton’ and ‘redundant’ columns denoted $\hat{q}$ and $\tilde{q}$, respectively, and a corresponding interpolation matrix $T_q$ such that

\[ A_{p,\tilde{q}} = A_{p,q}T_q + E, \]

where $\|E\|_2 \leq \epsilon \|A_{p,q}\|_2$. In other words, the redundant columns are approximated as a linear combination of the skeleton columns to within the prescribed relative accuracy.

Note that the ID of Definition 2.1 trivially always exists by taking the skeleton set to be all of $q$. In general, however, we aim to use the ID to compress $q$ such that
\( \hat{q} \) is a strict subset of \( q \). To see when this is possible, let \( A_{p,q} \) have a pivoted QR decomposition given by

\[
A_{p,q} \Pi \equiv \begin{bmatrix} A_{p,\hat{q}} & A_{p,\hat{q}} \end{bmatrix} = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix},
\]

where \( \Pi \) is a permutation matrix chosen such that \( R_{11} \) is non-singular. Then, we have that

\[
A_{p,\hat{q}} - A_{p,\hat{q}} R_{11}^{-1} R_{12} = (Q_1 R_{12} + Q_2 R_{22}) - (Q_1 R_{11}^{-1} R_{11} R_{12}) = Q_2 R_{22},
\]

with \( \|Q_2 R_{22}\|_2 = \|R_{22}\|_2 \). Thus, if \( \|R_{22}\|_2 \) is sufficiently small the desired bound holds. In practice, we compute an ID with accuracy parameter \( \epsilon \) by performing a pivoted QR factorization and terminating when the remaining \( R_{22} \) block is estimated to be sufficiently small using the first diagonal entry as a surrogate for \( \|R_{22}\|_2 \). The amount of compression that is attained by the fixed-tolerance ID therefore depends on the numerical rank of the matrix \( A_{p,q} \), and thus we find that the number of skeletons \( |\hat{q}| \) is small only if \( A_{p,q} \) is numerically low-rank such as is the case when \( A \) is the matrix of (1.2) and \( p \) and \( q \) are index sets corresponding to non-overlapping sets of boxes.

We refer the reader to [6, 19] and references therein for a more detailed presentation of IDs and algorithms to compute them.

If \( A \) is symmetric, we note that the index partitioning used to approximate \( A_{p,q} \) can be used to form an approximation of \( A_{q,p} \) to the same relative accuracy by interpolating the rows indexed by \( \hat{q} \) with those indexed by \( \hat{q} \). If not, one can compute an ID of the stacked matrix

\[
\tilde{A}_{p,q} = \begin{bmatrix} A_{p,q} \\ A_{q,p} \end{bmatrix},
\]

resulting in a set of skeleton points \( \hat{q} \) that is appropriate both for interpolating the columns of \( A_{p,q} \) and the rows of \( A_{q,p} \), and thus provides a data-sparse representation of the interactions between \( p \) and \( q \).

### 2.2.1. Acceleration using equivalent interactions.

The typical method for computing the ID in Definition 2.1 uses a pivoted QR factorization, which in principle requires choosing the skeletons \( \hat{q} \) in a computation explicitly involving all rows of the submatrix \( A_{p,q} \). In practice, however, our matrix \( A \) comes from the discretization of some elliptic kernel, which frequently satisfies some form of Green’s theorem wherein the values of the kernel inside a domain \( \Omega \) can be recovered from those on the boundary \( \partial \Omega \). Therefore, a key trick for reducing algorithmic complexity and increasing locality that is common in the literature is the use of an equivalent proxy surface [6, 7, 9, 11, 15, 20, 21, 26, 17].

Suppose that our DOFs \( q \) correspond to unknowns in the box \( b_1 \) in Figure 2.2. We draw a smooth proxy surface \( \Gamma_{\text{prox}} \) around \( b_1 \) and note that, in the continuous limit, all interactions between DOFs in \( b_1 \) and DOFs in boxes outside the proxy surface, e.g., \( b_2 \), can be equivalently represented as interactions between \( b_1 \) and \( \Gamma_{\text{prox}} \) followed by an operator with small norm that recovers the values at \( b_2 \) from those on \( \Gamma_{\text{prox}} \). As such, in the compression of DOFs in \( b_1 \), we can drop all rows of \( A \) corresponding to boxes outside \( \Gamma_{\text{prox}} \) and append a smaller number of rows corresponding to a discretization of the interactions with \( \Gamma_{\text{prox}} \). In a linear algebraic sense, this corresponds to making
Fig. 2.2: Using the black circle as a proxy surface $\Gamma_{\text{prox}}$, we represent all interactions between $b_1$ and the boxes outside $\Gamma_{\text{prox}}$ using equivalent interactions with the proxy surface. In particular, the DOFs in box $b_2$ are in a sense “shielded” from those in $b_1$, and vice versa.

the approximation

$$A_{p,q} = \begin{bmatrix} A_{\text{near}} \\ A_{\text{far}} \end{bmatrix} \approx \begin{bmatrix} A_{\text{near}} \\ WA_{\text{prox}} \end{bmatrix} = \begin{bmatrix} I \\ W \end{bmatrix} \begin{bmatrix} A_{\text{near}} \\ A_{\text{prox}} \end{bmatrix},$$

where $A_{\text{near}}$ is the sub-block of $A$ corresponding to interactions between $q$ and unknowns in the shaded boxes adjacent to $b_1$, $A_{\text{far}}$ is the sub-block of $A$ corresponding to interactions between $q$ and DOFs in boxes entirely outside the proxy surface, and $A_{\text{prox}}$ is the discretization of interactions between $q$ and the DOFs chosen to discretize the proxy surface itself (“proxy points”). With this representation, we can now perform an ID of simply the right-most matrix above and use this to obtain an ID of the original matrix. Since the interpolation matrix $W$ potentially has many more rows than columns, computing an ID of this surrogate matrix is much less computationally expensive. For a more thorough treatment of the proxy surface we direct the reader to [17].

2.3. Skeletonization. Once we use the ID to expose redundancy in $A$, we proceed using block Gaussian elimination to eliminate the redundant points $\hat{q}$ to approximately sparsify $A$ via a multiplicative procedure known as skeletonization factorization [17].

To begin, suppose that $p$ and $q = \hat{q} \cup \tilde{q}$ are index sets such that without loss of generality (i.e., up to a permutation) $A$ is given in block form as

$$A = \begin{bmatrix} A_{p,p} & A_{p,\hat{q}} & A_{p,\tilde{q}} \\ A_{\hat{q},p} & A_{\hat{q},\hat{q}} & A_{\hat{q},\tilde{q}} \\ A_{\tilde{q},p} & A_{\tilde{q},\hat{q}} & A_{\tilde{q},\tilde{q}} \end{bmatrix} \approx \begin{bmatrix} A_{p,p} & A_{p,\hat{q}} & A_{p,\tilde{q}} \\ A_{\hat{q},p} & A_{\hat{q},\hat{q}} & A_{\hat{q},\tilde{q}} \\ T^*_{\hat{q}}A_{\hat{q},p} & T^*_{\hat{q}}A_{\hat{q},\hat{q}} & T^*_{\hat{q}}A_{\hat{q},\tilde{q}} \end{bmatrix},$$

where by an ID of the form (2.2) we have that $A_{p,q} \approx A_{p,\hat{q}}T_{\hat{q}}$ and $A_{\hat{q},p} \approx T^*_{\hat{q}}A_{\hat{q},p}$ up to relative error $O(\epsilon)$. Then by a sequence of block row and column operations, we can eliminate the top-right and bottom-left blocks to obtain

$$Q_{\hat{q}}^*AQ_{\hat{q}} \approx \begin{bmatrix} A_{p,p} & A_{p,\hat{q}} \\ A_{\hat{q},p} & A_{\tilde{q},\hat{q}} \\ D_{\tilde{q},\hat{q}} \\ D_{\hat{q},\tilde{q}} \end{bmatrix},$$

(2.4)
where \( Q_q \) is given by

\[
Q_q = \begin{bmatrix} I & -T_q \\ I & I \end{bmatrix}
\]

and the \( D \) subblocks are linear combinations of the \( A \) subblocks. We use these matrices to define the skeletonization factorization of \( A \) via a final step of elimination of the DOFs \( \bar{q} \).

**Definition 2.2.** Assume that \( D_{\bar{q}, \bar{q}} \) in (2.4) is nonsingular. We define the skeletonization factorization of \( A \) with respect to the DOFs \( q \) as

\[
Z_q(A) \equiv \begin{bmatrix} A_{p,p} & A_{p,\bar{q}} \\ A_{\bar{q},p} & D_{\bar{q},\bar{q}} \end{bmatrix} \approx R_q^*Q_q^*AQ_qS_q \equiv U_q^*AV_q,
\]

where \( D_{\bar{q},\bar{q}} = A_{\bar{q},\bar{q}} - D_{\bar{q},q}D_{q,q}^{-1}D_{\bar{q},q} \), the matrices \( R_q^* \) and \( S_q \) are given by

\[
R_q^* = \begin{bmatrix} I & I \\ I & D_{\bar{q},\bar{q}}D_{q,q}^{-1}I \end{bmatrix}, \quad S_q = \begin{bmatrix} I & I \\ -D_{\bar{q},\bar{q}}D_{q,q}^{-1}I & I \end{bmatrix},
\]

and the remaining matrices are as defined in (2.4) and (2.5).

We note that in \( Z_q(A) \) the redundant DOFs have been completely decoupled from the rest while leaving the off-diagonal interactions unchanged. Henceforth, we refer to this skeletonization factorization as “skeletonization” for short, though this is not to be confused with the sense in which the term is used in [6].

We see that the matrices \( U_q \) and \( V_q \) in Definition 2.2 give an approximate sparsification of \( A \), since \( Z_q(A) \) is block-sparse and \( A \approx U_q^{-1}Z_q(A)V_q^{-1} \) with approximation error of order \( \epsilon \). It is worth noting that, while \( U_q \) and \( V_q \) are matrices of the same size as \( A \), they themselves are highly-structured – each is the product of block unit-triangular matrices – and thus they and their inverses can be applied cheaply by simply storing and using the non-trivial blocks.

**2.3.1. Group skeletonization.** As in [17], we will find it useful to work with collections of disjoint index sets and to skeletonize with respect to all of these index sets simultaneously. In particular, for a set of boxes \( S \), we will define the collection of DOF sets associated with boxes in \( S \) as

\[
C(S) \equiv \{ c \mid c \text{ is the set of active DOFs interior to a box } b \in S \},
\]

where we use the term “active” to refer to DOFs that have not been eliminated at any level \( \ell' > \ell \). With the above definition in mind, we define skeletonization of the matrix \( A \) with respect to the set of boxes \( S \) as

\[
Z_S(A) \approx U_S^*AV_S,
\]

with \( U_S \equiv \prod_{c \in C(S)} U_c \) and \( V_S \equiv \prod_{c \in C(S)} V_c \) where each \( U_c \) and \( V_c \) is as defined in Definition 2.2. There is a subtlety to this definition that we must stress: \( Z_S(A) \) is not the result of sequentially skeletonizing with respect to each of the index sets in \( C \). In other words, we do not first compress blocks of \( A \) relative to some index set \( c_1 \), then blocks of \( U_{c_1}AV_{c_1} \) relative to \( c_2 \) and so on. Rather, the skeletonization in \( Z_S(A) \) is...
completely parallelizable over the disjoint index sets corresponding to different boxes, and each ID in $Z_A(S)$ is performed using the original subblocks of $A$. By defining things in this fashion, $Z_A(S)$ is completely independent of the order used to iterate over index sets $c \in C(S)$.

2.4. Recursive skeletonization factorization. Now we use the tools above to construct the recursive skeletonization factorization \[17\] for solving (1.2) that is described in Algorithm 2.1, which is an alternative approach to that of \[15\] for expressing the hierarchical compression scheme of \[20\].

Consider the rectangular bounding box containing the unknowns discretizing $\Omega$ in (1.1), hierarchically decomposed with a quadtree containing levels $\ell = 0, 1, \ldots, L$. Recalling that $B_L$ is the set of boxes on level $L$, we follow the definition in (2.6) and skeletonize with respect to $B_L$ to obtain $Z_{B_L}(A)$. Once we have completed this procedure for level $L$, we move to level $L - 1$. Here, the DOFs associated with a box on level $L - 1$ are defined to be the collection of skeleton DOFs belonging to the children of that box, since the other DOFs have already been eliminated (i.e., are inactive). Furthermore, the off-diagonal blocks of this matrix corresponding to interactions between DOFs in distinct boxes in $B_{L-1}$ are just aggregated off-diagonal subblocks from the previous level, and thus are unmodified kernel interactions. Therefore, we can now skeletonize $Z_{B_L}(A)$ to further compress the active DOFs that remain at level $L - 1$ obtaining $Z_{B_{L-1}}(Z_{B_L}(A))$.

Repeating the recursive skeletonization process for each level up to (but not including) the root, we at each step eliminate a subset of the unknowns that were skeletons at the previous level. This results in a factorization of $A$ into a (long) product of $U$ and $V$ matrices enacting the elimination, as seen in Algorithm 2.1.

Algorithm 2.1 Recursive skeletonization factorization (rskel) \[17\]

\[
A_L \leftarrow A \quad \triangleright \text{initialize}
\]
\[
\text{for } \ell = L, L - 1, \ldots, 1 \text{ do} \quad \triangleright \text{loop from finest to coarsest level}
\]
\[
A_{\ell-1} \leftarrow Z_{B_\ell}(A_\ell) \approx U_{B_\ell} A_\ell V_{B_\ell} \quad \triangleright \text{skeletonize level}
\]
\[
\text{end for}
\]
\[
A \approx F \equiv U_{B_L}^{-1} \cdots U_{B_1}^{-1} A_0 V_{B_1}^{-1} \cdots V_{B_L}^{-1}
\]

As previously mentioned, in Algorithm 2.1 each matrix $U_{B_\ell}$ or $V_{B_\ell}$ is the product of many $U_c$ and $V_c$ matrices each corresponding to a single box on level $\ell$. These matrices $U_c$ and $V_c$ represent a sequence of simple block elimination steps that

1. zero all interactions between the redundant DOFs in that box and the DOFs in other boxes, and,

2. otherwise act only locally to that box.

As such, the matrices $U_{B_\ell}$ and $V_{B_\ell}$ corresponding to each level are used purely for notational convenience, and each of the $U_c$ and $V_c$ factors should be stored and applied independently in block form.

For elliptic kernels, the computational complexity of rskel in 2D depends on the intrinsic dimension of the underlying domain of integration $\Omega$ and not on the ambient dimension, as noted in \[15, 17\]. For example, for quasi-1D problems we have the result that construction of the factorization $F$ or application of $F$ or $F^{-1}$ costs $O(N)$ operations with a small constant on the leading term. We see in Figure

\[^{1}\text{Note that, at this point, all DOFs are active.}\]
that the skeleton unknowns tend to cluster near the boundaries of the quadtree boxes, in this case leading to roughly constant-size skeleton sets at the top level. However, as seen in Figure 2.4 in true 2D problems the clustering of DOFs near box boundaries results in the number of remaining skeleton DOFs at the top level scaling as $O(N^{1/2})$, making factorization with $\text{rskel}$ asymptotically more expensive and thus necessitating modifications as described in Section 2.5.

2.5. The hierarchical interpolative factorization. For true 2D problems, complexity estimates give that the construction of $F$ in Algorithm 2.1 costs roughly $O(N^{3/2})$. To recover linear complexity for this case, we use the hierarchical interpolative factorization [17], which is based on the same fundamental operations as $\text{rskel}$ but adds an extra step of skeletonization between quadtree levels.

In particular, after the first level of compression in Figure 2.4 we associate each active DOF with the edge to which it is closest. This leads to what we refer to as an edge level between box levels of the quadtree, where now each edge has an associated set of DOFs $c$. We denote these edge levels with half-integer labels, such that analogously to $B_\ell$ we can define $B_{\ell-1/2}$ to be the collection of edges on level $\ell - 1/2$ with the corresponding collection of DOF sets $C(B_{\ell-1/2})$. Thus, we can now skeletonize the DOFs belonging to each edge before proceeding to the next box level of the quadtree. This process of alternating between skeletonizing boxes and skeletonizing edges is summarized in Algorithm 2.2 and is illustrated in Figure 2.5.
Fig. 2.5: Active DOFs before skeletonizing each level $\ell$ of $\text{hif}$ in 2D. The growth of the number of skeleton DOFs that was observed in Figure 2.4 appears to have been reduced dramatically.

Algorithm 2.2 Hierarchical interpolative factorization in 2D ($\text{hif}$)

$A_L \leftarrow A$  \hspace{1cm} \triangleright \text{initialize}

for $\ell = L, L - 1, \ldots, 1$ do

$A_{\ell-1/2} \leftarrow Z_{B_1}(A_\ell) \approx U_{B_1}^* A_\ell V_{B_1}$  \hspace{1cm} \triangleright \text{skeletonize box level}

$A_{\ell-1} \leftarrow Z_{B_{\ell-1/2}}(A_{\ell-1/2}) \approx U_{B_{\ell-1/2}}^* A_{\ell-1/2} V_{B_{\ell-1/2}}$  \hspace{1cm} \triangleright \text{skeletonize edge level}

end for

$A \approx F \equiv U_{B_L}^{-1} U_{B_{L-1/2}}^{-1} \cdots U_{B_1}^{-1} U_{B_{1/2}}^{-1} A_0 V_{B_1}^{-1} \cdots V_{B_{L-1/2}}^{-1} V_{B_L}^{-1}$

There are a few idiosyncrasies to Algorithm 2.2 that we do not address here (e.g., compression of Schur complement interactions). We direct the reader to [17] for a thorough treatment.

Assuming that the edge levels admit sufficient compression (observed in practice), the computational complexity of $\text{hif}$ for elliptic kernels is once again $O(N)$ for computing the factorization $F$, as well as for applying $F$ or $F^{-1}$ (see [17] for details).

3. Updating algorithm. With Algorithm 2.1 and Algorithm 2.2 in mind, we consider the use of the proxy surface for accelerated compression as in Figure 2.2. At level $L$, skeletonization of box $b_1$ depends only on DOFs that belong to (are contained in) box $b_1$ or other boxes whose interiors intersect the proxy surface. In particular, to motivate updating, we make the key observation that skeletonization of $b_1$ is completely independent of DOFs in $b_2$ due to the fact that $b_2$ is outside the proxy surface around $b_1$. In fact, the only DOFs on which skeletonization of $b_1$ depends are those contained in the shaded boxes, i.e., those boxes with DOFs inside the proxy surface.
surface around $b_1$. With this in mind, we make the following definition.

**Definition 3.1.** For a box $b$ at quadtree level $\ell$, the neighbor set of $b$, $\nu(b)$, is defined to be the set of all boxes that contain DOFs against which $b$ must skeletonize, i.e.,

$$\nu(b) \equiv \{ b' \in B_\ell \mid b' \text{ is adjacent to } b \text{ and } \ell' \leq \ell \}.$$  

We refer to this set as the neighbors of $b$. Note that in general $b' \in \nu(b)$ does not imply $b \in \nu(b')$.

The requirement in Definition 3.1 that the neighbor boxes of $b$ be at most as refined as $b$ itself is necessary for the case of heterogeneous refinement. For example, in Figure 2.1 we see that $b_1 \in \nu(b_5)$, and indeed skeletonizing $b_5$ requires knowledge of the DOFs in $b_1$. In contrast, we see that $b_5 \notin \nu(b_1)$, and this is because skeletonizing $b_1$ will require knowledge not of the DOFs in $b_5$ but rather the DOFs in its parent, $b_2$.

For clarity of exposition we will first describe the updating technique of this paper in terms of the rskel algorithm, and later mention briefly the necessary modifications for hif. To begin, suppose that we have on hand an initial factorization $F$ corresponding to some base discretized problem of the form of (1.2). The factorization $F$ can be represented as a collection of small matrices comprising the sub-blocks associated with skeletonization of each box in the spatial discretization, i.e.,

$$F \sim \{ (T,D_{q,q},D_{\bar{q},q},D_{q,\bar{q}},D_{\bar{q},\bar{q}}) \mid q \text{ is the set of active DOFs for a box } b \},$$

(3.1) where here each block is as defined in Definition 2.2. In fact, this is essentially how one must actually store $F$ to obtain optimal asymptotic runtime.

We consider making a local perturbation to the problem geometry or coefficients such that a small number of leaf boxes contain modified DOFs, and hereafter use $A$ to denote the new perturbed matrix. For simplicity, we assume that this perturbation is such that the quadtree for the initial problem is still valid for the new problem, in that we have not exceeded the constant bound $n_{occ}$ on the number of DOFs per leaf box. We note, however, that the technique and theory extend directly to more complicated situations where, e.g., the tree must split over-occupied nodes or delete empty nodes.

Our basic updating strategy is to track potential changes through the quadtree based on which unknowns have been directly perturbed. While this could be done as a preprocessing step prior to any refactorization, we will describe the method here as a gradual process of propagating the changes at the time of refactorization.

**3.1. Propagation rules.** Based on earlier observations, we assert that, at any level of the spatial hierarchy, the only nodes that require re-skeletonization are those that either contain active modified DOFs or have neighbors that contain active modified DOFs. This is simple to understand by looking at the contrapositive – if a box does not contain modified DOFs and neither do its neighbors, skeletonization of that box will not differ between the old and the new factorization. This leads to two rules for tracking the propagation of nodes that must be re-skeletonized (marked nodes):

1. if box $b$ is re-skeletonized, its parent now contains modified DOFs and thus we must mark its parent for re-skeletonization, and,
2. if box $b$ directly contains modified DOFs we must mark all boxes $b'$ such that $b \in \nu(b')$ for re-skeletonization, since modifications have occurred to DOFs within the proxy circles of such boxes.
Fig. 3.1: Top-left: Suppose we have locally perturbed a problem such that only DOFs in the dark gray leaf box are modified, and thus only that box and its neighbors (light gray) need to be reskeletonized at the leaf level. In particular, re-skeletonization of the white boxes is not required. Top-right: At the next level, the dark gray box contains DOFs that have been modified from the level below, and thus that box and its neighbors must be reskeletonized. Bottom: The corresponding quadtree nodes for the boxes that must be refactored.

The first rule above dictates how markings propagate up the tree, whereas the second describes the propagation of markings within a single level of the tree, at least in the case of uniform spatial discretization (i.e., all leaf nodes occupy the same level of the hierarchy). For example, Figure 3.1 shows the propagation of markings if initially only the bottom left leaf box contains modified DOFs.

3.2. Re-skeletonization. At the bottom level, $\ell = L$, of the tree, it is evident that re-skeletonization of a box simply requires re-skeletonization of the appropriate DOF set with respect to the original entries of the matrix $A$ that is being skeletonized. To re-skeletonize at any higher level $\ell < L$ of the tree, however, we must skeletonize with respect to the entries of the matrix $A_\ell$ in Algorithm 2.1. Put another way, we see in Algorithm 2.1 that there is a sense of the state of $A$ at level $\ell$ of the tree, and it is the relevant entries of blocks of that matrix that must be used to calculate the various $D$ blocks in the skeletonization procedure at that level.

Re-skeletonizing a box $b \in B_\ell$ in \texttt{rskel} requires knowledge of the diagonal block at level $\ell$ corresponding to DOFs in $b$, and the off-diagonal blocks at level $\ell$ corresponding to interactions between DOFs in $b$ and DOFs in any $b' \in \nu(b)$. In \texttt{rskel} the diagonal blocks will contain Schur complement updates from previous levels, but the off-diagonal blocks are always unmodified kernel interactions. This is a consequence of two facts:

1. skeletonization only zeros off-diagonal blocks and modifies diagonal blocks, and,
2. diagonal blocks satisfy a nesting property: a diagonal block corresponding to a box at level \( \ell \) becomes a sub-block of the diagonal block corresponding to its parent box. Thus, to reconstruct the state of the blocks needed to re-skeletonize \( b \), we simply use the blocks \( D_{\hat{q},\hat{q}} \) corresponding to skeleton DOFs for each child of \( b \), which can easily be extracted from a representation of the form (3.1).

Notationally, for a set of boxes \( S \) where only a subset \( \mu \) of marked boxes needs to be refactored, we will use \( Z^\mu_S(A) \) to denote the re-skeletonization of \( A \) with respect to the DOF sets associated with boxes in \( \mu \) while the factors associated with DOF sets in \( \sigma \equiv S \setminus \mu \) are implicitly understood to be the same as in the initial factorization, i.e.,

\[
Z^\mu_S(A) \approx \tilde{U}_\mu^* U^*_\sigma V^*_\sigma \tilde{V}_\mu^* \equiv \tilde{U}^* U^*_\sigma V^*_\sigma \tilde{V}^*,
\]

where \( U_\sigma \equiv \prod_{c \in C(\sigma)} U_c \) is a factor of the previously defined matrix \( U_S \), \( \tilde{U}_\mu \equiv \prod_{c \in C(\mu)} \tilde{U}_c \) is a new matrix corresponding to the marked boxes that must be updated, and the \( V \) matrices are defined in an analogous fashion. Since \( U_\sigma \) and \( V_\sigma \) have already been computed in the initial factorization, only the \( U_\mu \) and \( V_\mu \) matrices need to be computed in the updating scheme, thus avoiding redundant calculations.

### 3.3. Updating \( \text{rskel} \)

With the two propagation rules for updates, we define the marked set at each level as follows.

**Definition 3.2.** The set \( \mu_\ell \) is defined to be the set of boxes that require re-skeletonization at level \( \ell \), i.e.,

\[
\mu_\ell \equiv \{ b \in B_\ell \mid b \text{ has modified DOFs or } b' \in \nu(b) \text{ has modified DOFs} \}.
\]

With this definition in mind, updating becomes a simple two-step process for each level \( \ell = L, \ldots, 1 \). First, the propagation rules must be applied to determine the marked set \( \mu_\ell \) for the current level, which determines the set of boxes that must be re-skeletonized. Then, the group re-skeletonization is performed according to the process outlined in Section 3.2. Repeating this level-by-level, we obtain Algorithm 3.1 which is intentionally written analogously to Algorithm 2.1. It is important to note that the updating process here is not an approximate one: the updated factorization \( F \) that is obtained is identical to that which would have been obtained via the computation of a completely new \( \text{rskel} \) factorization.

Thus far, we have assumed for pedagogical purposes that the structure of the hierarchical decomposition does not change between the initial factorization and the refactorization. In the case of tree refinement where quadtree nodes are created or deleted the core updating algorithm does not change, but the set \( B_\ell \) will now itself have changed, potentially changing size, and it is necessary to perform some minor bookkeeping to ensure that added nodes are always in the marked set and that factors corresponding to deleted nodes are removed from the factorization.

### 3.4. Modifications for \( \text{hif} \)

To adapt the updating process for \( \text{rskel} \) to an updating process for \( \text{hif} \), we use the same basic building blocks of identifying and re-skeletonizing marked boxes and marked edges. Incorporating the half-integer edge levels, however, complicates the process.

In particular, in Section 3.2 we used the nesting property of \( \text{rskel} \) to assert that reconstructing the blocks of \( A_\ell \) at any level for re-skeletonization of a node requires only looking at the diagonal blocks corresponding to its child nodes. For \( \text{hif} \), this is
Algorithm 3.1 Updating $\text{rskel}$

\[
A_L \leftarrow A
\]
\[
\text{for } \ell = L, L-1, \ldots, 1 \text{ do} \quad \triangleright \text{loop from finest to coarsest level}
\]
\[
\mu_\ell \leftarrow \{b \in B_\ell \mid b \text{ has modified DOFs or } b' \in \nu(b) \text{ has modified DOFs}\}
\]
\[
A_{\ell-1} \leftarrow Z_{k_\ell}(A_\ell) \approx \tilde{U}_{B_\ell}A_\ell\tilde{V}_{B_\ell} \quad \triangleright \text{re-skeletonize those boxes that need it}
\]
\end{for}
\[
A \approx F \equiv \tilde{U}_{B_L} \cdots \tilde{U}_{B_1}A_0\tilde{V}_{B_1} \cdots \tilde{V}_{B_L}
\]

no longer the case, as the addition of the edge levels means that there is a different notion of “diagonal block” for the edge levels than for the box levels, and thus keeping track of the state of interactions between DOFs becomes more complicated, see Figure 3.2. The process of reconstructing the diagonal interactions for the DOF set $p$ of a box on level $\ell$ of the tree is outlined in Algorithm 3.2, where we first loop over the children on the box level below, then make the necessary modifications from the edge level. Reconstructing the diagonal interactions for the DOF set of an edge on a half-integer level is analogous; it is necessary to loop over the edges one level below and then incorporate the changes from the box level immediately below. We note that for off-diagonal interactions, the process is the same except that the first loop can be skipped, since the only possible Schur complement updates in this case come from one half-level below.

Algorithm 3.2 Reconstructing box-level diagonal interactions in $\text{hif}$

Given: box $b \in B_\ell$ with corresponding active DOF set $p$

\[
(A_\ell)_{p,p} \leftarrow A_{p,p} \quad \triangleright \text{initialize}
\]
\[
\text{for each child box of } b \text{ with active DOFs } q \text{ do} \quad \triangleright \text{loop over boxes below}
\]
\[
(A_\ell)_{q,q} \leftarrow D_{q,q}
\]
\end{for}
\[
\text{for each edge of a child box of } b \text{ with active DOFs } q \text{ do} \quad \triangleright \text{loop over edges below}
\]
\[
(A_\ell)_{q,q} \leftarrow D_{q,q} \quad \triangleright \text{note: overwrites child box interactions as needed}
\]
\end{for}

3.5. Computational complexity. Intuitively, if the number of leaf-level boxes containing modified DOFs is small, the number of marked boxes (or shaded nodes in Figure 3.1) remains asymptotically smaller than the total number of boxes in a way that will be made rigorous. Thus, if each box has roughly the same skeletonization cost, we see that updating will be asymptotically less expensive than complete refactorization.

The assumptions that ensure $\text{rskel}$ is computationally efficient (asymptotic complexity $O(N)$ to factor and apply) serve to control the number of active DOFs at each level. Let $k_\ell$ denote a bound on the number of skeleton DOFs for a single box at level $\ell$. For $\text{rskel}$, standard multipole estimates show that factorization of an elliptic system with a quasi-1D boundary leads to $k_\ell$ growing linearly as we progress up the tree, i.e., $k_\ell = \mathcal{O}(L - \ell)$ (cf. 17). The cost of skeletonizing a box is dominated by the cost of the ID, which is cubic in the number of skeleton DOFs. Therefore the cost of skeletonizing a box $b \in B_\ell$ is $\mathcal{O}((L - \ell)^3)$.

After a single leaf-level perturbation, Lemma 3.3 shows that the total number of boxes that need to be re-skeletonized is bounded by a small constant $C$. With
Fig. 3.2: Suppose the two boxes in (a) have distinct parents. After box-level elimination, DOFs from the two boxes have distinct Schur complement updates to their self-interactions. In (b), the circled DOFs now belong to the same edge and thus share a diagonal block update. Finally, the remaining active DOFs in (c) will need to be skeletonized against each other at the next level while taking the previous updates into account.

**Lemma 3.3.** Suppose that a single leaf-level node contains directly-modified DOFs at the beginning of the `skel` updating technique of Algorithm 3.1. Then, the size of the marked set, $|\mu_\ell|$, is bounded by a small dimension-dependent constant $C$ independent of $N$ and $\ell$.

**Proof.** In any dimension, $d$, we can associate each box $b$ on a given level with a $d$-tuple of integer coordinates or indices, $(b_1, \ldots, b_d)$. It is natural to consider the $\ell_\infty$-distance associated with this representation,

$$\|b - b'\|_\infty \equiv \|(b_1, \ldots, b_d) - (b'_1, \ldots, b'_d)\|_\infty,$$

which codifies the idea of “$b$ is $\|b - b'\|_\infty$ boxes away from $b'”$. With this distance in mind, we begin by defining the concept of reach at a level, $r_\ell$. With $b_L \in B_L$ as the single box containing directly modified DOFs, we note that $b_L$ has exactly one ancestor on level $\ell < L$, which we denote as $b_\ell$. Thus, we define the reach at level $\ell$ as

$$r_\ell \equiv \max_{b' \in \mu_\ell} \|b_\ell - b'\|_\infty,$$

$i.e.$, it is the number of boxes away the marked set has extended from its center, $b_\ell$.

The key observation with a single directly modified leaf box is that the bottom level reach is $r_L = 1$ and the reach at subsequent levels does not much exceed this size. In particular, the reach satisfies the recurrence relation

$$r_\ell = \left\lceil \frac{r_{\ell+1}}{2} \right\rceil + 1,$$

where division by two corresponds to the fact that marked boxes on a level are contiguous and $r_{\ell+1}$ contiguous boxes have at most $\left\lceil \frac{r_{\ell+1}}{2} \right\rceil$ parents, and adding one corresponds to marking all neighbors of these parents. This relation has a fixed point
at \( r_\ell = 2 \). Therefore, in \( d \) dimensions, the size of the marked set is bounded as
\[
|\mu_\ell| \leq (2r_\ell + 1)^d \leq C.
\]

**Theorem 3.4 (Complexity of updating \texttt{rskel}).** Suppose we use the updating technique of Section 3.3 to make direct modifications to DOFs in a number \( m \) of leaf-level boxes for an integral equation with elliptic kernel on a quasi-1D domain. Then, the complexity of updating the \texttt{rskel} factorization is \( \mathcal{O}(m \log^4 N) \).

**Proof.** On level \( \ell \) we need to refactor \( |\mu_\ell| \) boxes each of which has at most \( k_\ell \) skeleton DOFs. This costs \( \mathcal{O}(k_\ell^3) \) per box. This means the total re-factorization time, \( t \), grows as
\[
t = \sum_{\ell=0}^{L} |\mu_\ell| \mathcal{O}(k_\ell^3) = \mathcal{O} \left( mC \sum_{\ell=0}^{L} (L - \ell)^3 \right) = \mathcal{O}(mC \log^4 N),
\]
where we have used the fact that the marked set resulting from \( m \) leaf-level modifications is no bigger than \( m \) times the maximum marked set size of a single box, \( C \) from Lemma 3.3. Note that this bound is clearly weak, e.g., the number of marked boxes on a level is of course limited by the total number of boxes on that level.

The asymptotic scaling of updating \texttt{hif} using the same technique as for \texttt{rskel} follows essentially the same path of reasoning. For \texttt{hif}, we assume that the number of remaining skeletons for a box at level \( \ell \) grows like \( \mathcal{O}(L - \ell) \). This rate of growth is strongly supported by numerical experiments (see [17]) though remains a conjecture at this time. Then, while the constant is larger, the number of marked box nodes or edge nodes on each level remains bounded as can be shown in a procedure similar to the box-counting in Lemma 3.3. Finally, summing over both box- and edge-levels, it is not difficult to show that the cost of updating \texttt{hif} after \( m \) leaf-level perturbations has asymptotic complexity \( \mathcal{O}(m \log^4 N) \). For both \texttt{hif} and \texttt{rskel}, it is interesting to observe that when a constant number of leaf boxes are modified, the cost of updating a factorization is asymptotically less expensive than application of the factorization.

**4. Numerical results.** We now present two examples showing the asymptotic scaling of our updating routine, one for \texttt{rskel} and one for \texttt{hif}. For each example, the following, if applicable, are given:

- \( \epsilon \): base relative precision of the ID;
- \( N \): total number of DOFs in the problem;
- \( t_f \): wall clock time for constructing the factorization \( F \) in seconds;
- \( t_{u,c} \): wall clock time for updating a constant number of points in the factorization \( F \);
- \( t_{u,p} \): wall clock time for updating a constant proportion of points in the factorization \( F \).

**4.1. Example 1: Laplace double-layer potential on a rounded square.**

We first present an example of modifying the boundary geometry for a boundary integral equation formulation of the Laplace equation. Consider the interior Dirichlet Laplace problem,

\begin{align}
\Delta u(x) &= 0, & x \in \Omega \subset \mathbb{R}^2, \\
u(x) &= f(x), & x \in \Gamma = \partial \Omega,
\end{align}
Fig. 4.1: For our first example, we start with $\Gamma_1$, the square with rounded corners following the dashed curve. Then, for updating we adjust the rounding parameter to obtain $\Gamma_2$, the square with the sharper, solid corners.

which can be written as a second-kind integral equation with unknown surface density $\sigma(x)$ as

$$
-\frac{1}{2} \sigma(x) + \int_{\Gamma} \frac{\partial G}{\partial \nu_y} (\|x - y\|) \sigma(y) d\Gamma(y) = f(x), \quad x \in \Gamma,
$$

where $G(r) = -\log r/(2\pi)$ is the fundamental solution of the free-space PDE and $\nu_y$ is the outward-facing unit normal at $y \in \Gamma$.

We use the trapezoid rule to discretize (4.3) on $\Gamma = \Gamma_1$, a square with rounded corners corresponding to a certain radius of curvature as in Figure 4.1 and then factor the resulting system using \texttt{rskel}. With this base factorization, we move the quadrature points where necessary such that they discretize $\Gamma_2$, a rounded square with a different radius of curvature, and change the necessary quadrature weights to reflect the new arc lengths. We then use the old factorization as input to our updating algorithm to construct the factorization for the new geometry.

To investigate asymptotic scaling of the updating algorithm as we increase the number of discretization points $N$, there are two primary ways to increase the problem size. The first is to choose $\Gamma_2$ to have a radius of curvature at the corners that is independent of $N$, which implies that a fixed proportion of the discretization points will be modified. The second is to use a variable radius of curvature for $\Gamma_2$ such that the number of moved discretization points is constant. In the first case, we expect to see linear scaling with $N$, since the number of modified leaf-level boxes is $O(N)$, and in the second case theory dictates poly-log scaling.

The data for this example can be seen in Table 4.1 with Figure 4.2 showing corresponding scaling results for both the case of updating constant proportion and a constant number of DOFs.

4.2. Example 2: the Lippmann-Schwinger equation. To demonstrate updating of $\sigma$ for the true 2D case, we consider the Lippmann-Schwinger equation for Helmholtz scattering of an incoming wave with frequency $k$,

$$
\sigma(x) + k^2 \int_{\Omega} K(\|x - y\|) w(y) \sigma(y) d\Omega(y) = f(x), \quad x \in \Omega = (0, 1)^2.
$$

Here, $K(r) = (i/4) H_0^{(1)}(kr)$ is the fundamental solution of the Helmholtz equation written in terms of the zeroth order Hankel function of the first kind, $H_0^{(1)}(x)$, and
Table 4.1: Timing results for Example 1: rounded square with \texttt{rskel}. We see that doubling the number of points doubles the time to update a fixed proportion of points, but that the time to update a constant number of points grows slowly with $N$.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|}
\hline
$\epsilon$ & $N$ & $t_f$ & $t_{u,p}$ & $t_{u,c}$ \\
\hline
$10^{-3}$ & 524288 & 1.1e+1 & 2.8e+0 & 4.6e-2 \\
& 1048576 & 2.2e+1 & 5.5e+0 & 4.7e-2 \\
& 2097152 & 4.7e+1 & 1.2e+1 & 5.2e-2 \\
\hline
$10^{-6}$ & 524288 & 1.4e+1 & 3.6e+0 & 8.8e-2 \\
& 1048576 & 2.8e+1 & 7.1e+0 & 9.6e-2 \\
& 2097152 & 5.7e+1 & 1.4e+1 & 1.0e-1 \\
\hline
$10^{-9}$ & 2097152 & 1.7e+1 & 4.6e+0 & 1.5e-1 \\
& 1048576 & 3.5e+1 & 8.9e+0 & 1.7e-1 \\
& 2097152 & 7.0e+1 & 1.8e+1 & 1.8e-1 \\
\hline
\end{tabular}
\caption{Timing results for Example 1: rounded square with \texttt{rskel}.}
\end{table}

Fig. 4.2: Timing results for Example 1 with \texttt{rskel} on the rounded square. Circular markers denote factor times and square markers denote update times for tolerances $\epsilon$ of $10^{-3}$ (black), $10^{-6}$ (gray), and $10^{-9}$ (white). Left: updating a fixed proportion of points with guide curve $O(N)$. Right: updating a constant number of points with guide curves $O(N)$, and $O(\log^4 N)$, from top to bottom.

$w(x)$ is a function representing the scatter. This kernel is not elliptic, but has qualitatively similar behavior for small $k$. Assuming that $w(x)$ is non-negative, we can make the change of variables $u(x) = \sqrt{w(x)} \sigma(x)$ to obtain the symmetric form

\begin{equation}
(4.5) \quad u(x) + k \sqrt{w(x)} \int_{\Omega} K(||x-y||) \left[ k \sqrt{w(y)} \right] u(y) d\Omega(y) = \sqrt{w(x)} f(x), \quad x \in \Omega.
\end{equation}

Given $w(x)$, we discretize (4.5) using a uniform $\sqrt{N} \times \sqrt{N}$ grid, where the diagonal entries $A_{ii}$ of the matrix $A$ are computed adaptively and the off-diagonal entries $A_{ij}$ are approximated using one-point quadratures. For this example, we will consider starting with the function

\begin{equation}
(4.6) \quad w_0(x) = \exp(-16 ||x-c||^2),
\end{equation}
a Gaussian centered at \( c = [0.5, 0.5]^T \), and then modifying the scatterer by adding a perturbation that is essentially localized to construct

\[
w_1(x) = w_0(x) + \exp(-s\|x - d\|^2),
\]

where the perturbation is a Gaussian centered at \( d = [0.8, 0.8]^T \) truncated to machine precision and \( s = s(N) \) is an adaptive scale parameter. In particular, we choose \( s(N) \) such that roughly 340 points lay within the region where the perturbation is greater than machine precision, which isolates the perturbation to a number of leaf-level boxes of the quadtree that is independent of \( N \). An example perturbed scatterer can be seen in Figure 4.3. We choose the frequency \( k = (2\pi)/10 \) for our tests, whereas the box has sides of unit length.

The data for this example can be seen in Table 4.2 with the corresponding scaling plot in Figure 4.3. For clarification, we note that the small dip in \( t_{u,c} \) seen between the last two rows of Table 4.2 is not an error and appears to be due to the interplay between the fixed occupancy parameter for leaves of the quadtree and the distribution of discretization points in the domain.

5. Conclusions. Our examples indicate that the updating algorithm behaves as expected given our theoretical results, with linear scaling in the total number of leaf-boxes containing DOFs that have been directly modified and poly-log scaling in the total number of DOFs. This is a result that is perhaps not surprising theoretically, but should prove to be of great utility for real-world implementations of these algorithms.

In contrast to the Sherman-Morrison-Woodbury strategy for solving perturbed systems [11], the end result of our algorithm is the factorization corresponding to the new system. Furthermore, this process is exact, i.e., it results in the same factorization as if a new one had been computed from scratch. One advantage of this strategy is that it allows for subsequent updates to a new set of localized DOFs, possibly located in a different region of the domain.
Table 4.2: Timing results for Example 2: Lippmann-Schwinger with \texttt{hif}. Note the slow, roughly polylog growth of $\tau_{u,c}$ with respect to $N$ in contrast to the approximately linear growth of $t_f$.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|}
\hline
$\epsilon$ & $N$ & $t_f$ & $\tau_{u,c}$ \\
\hline
$10^{-3}$ & $512^2$ & 4.6e+2 & 6.2e+1 \\
& $1024^2$ & 1.9e+3 & 1.0e+2 \\
& $2048^2$ & 7.8e+3 & 1.4e+2 \\
& $4096^2$ & 3.2e+4 & 2.0e+2 \\
\hline
$10^{-6}$ & $512^2$ & 2.4e+3 & 1.0e+3 \\
& $1024^2$ & 1.0e+4 & 3.0e+3 \\
& $2048^2$ & 3.8e+4 & 4.8e+3 \\
& $4096^2$ & 1.4e+5 & 4.6e+3 \\
\hline
\end{tabular}
\end{table}

For the recursive skeletonization factorization, our updating process is not difficult to implement. Our examples show that in cases where we update even large portions of the domain it is possible to recover the constant factor complexity difference between updating and complete refactorization. While this is not as dramatic a performance gain as in the case of truly localized updates where only a small portion of the domain is updated, it is still arguably “morally correct” in the sense that it is demonstrably more efficient and requires only marginally more work to implement.

For the hierarchical interpolative factorization, we saw that updating requires more bookkeeping than for \texttt{rskel} due to the diagonal updates at the edge level. However, the updating algorithm still shows the same asymptotic scaling.

While here we have only discussed updating for 2D integral equations, all the ideas presented in this paper extend directly to the 3D case. In fact, it is for 3D problems that we expect to see the biggest performance gain from using an updating procedure instead of completely refactoring the system. The reason for this stems again from simple box counting – the number of white nodes in the 3D analogue of Figure 3.1 grows more quickly with respect to the depth of the tree.

Additionally, just as the \texttt{hif} can be carried out in the partial differential equation case \cite{16}, so too can the updating procedure described here. The ideas of box-marking and keeping track of diagonal interactions extend directly, but now the DOFs in the linear system come from, e.g., a finite difference discretization. This is current work that we will present in a future publication.

\textbf{Acknowledgments.} V.M. is supported by a U.S. Department of Energy Computational Science Graduate Fellowship under grant number DE-FG02-97ER25308. A.D. is partially supported by a National Science Foundation Graduate Research Fellowship under grant number DGE-1147470 and a Simons Graduate Research Assistantship. K.H. is supported by a National Science Foundation Mathematical Sciences Postdoctoral Research Fellowship under grant number DMS-1203554. L.Y. is partially supported by the National Science Foundation under award DMS-1328230 and the U.S. Department of Energy’s Advanced Scientific Computing Research program under award DE-FC02-13ER26134/DE-SC0009409. The authors thank Lenya Ryzhik for computing resources, as well as Austin Benson and Bradley Nelson for useful comments on early drafts of this manuscript.
REFERENCES

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

[2] J. Bremer, A fast direct solver for the integral equations of scattering theory on planar curves with corners, Journal of Computational Physics, 231 (2012), pp. 1879 – 1899.

[3] S. Chandrasekaran, P. Dewilde, M. Gu, W. Lyons, and T. Pals, A fast solver for HSS representations via sparse matrices, SIAM Journal on Matrix Analysis and Applications, 29 (2007), pp. 67–81.

[4] S. Chandrasekaran, M. Gu, and T. Pals, A fast ULV decomposition solver for hierarchically semiseparable matrices, Journal of Scientific Computing, 57 (2013), pp. 477–501.

[5] Y. Chen, A fast, direct algorithm for the Lippmann-Schwinger integral equation in two dimensions, Advances in Computational Mathematics, 16 (2002), pp. 175–190.

[6] H. Cheng, Z. Gimbutas, P.G. Martinsson, and V. Rokhlin, On the compression of low rank matrices, SIAM Journal on Scientific Computing, 26 (2005), pp. 1389–1404.

[7] E. Corona, P.G. Martinsson, and D. Zorin, An $O(N)$ Direct Solver for Integral Equations on the Plane, ArXiv e-prints, (2013).

[8] P.E. Gill, W. Murray, M.A. Saunders, and M.H. Wright, Maintaining LU factors of a general sparse matrix, Linear Algebra and its Applications, 8889 (1987), pp. 239 – 270.

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

[10] G.H. Golub and C.F. Van Loan, Matrix Computations (3rd Ed.), Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[11] L. Greengard, D. Gueyffier, P.G. Martinsson, and V. Rokhlin, Fast direct solvers for integral equations in complex three-dimensional domains, Acta Numerica, 18 (2009), pp. 243–275.

[12] Leslie Greengard and V Rokhlin, On the numerical solution of two-point boundary value problems, Communications on Pure and Applied Mathematics, 44 (1991), pp. 419–452.

[13] B.C. Gunter and R.A. Van De Geijn, Parallel out-of-core computation and updating of the QR factorization, ACM Trans. Math. Softw., 31 (2005), pp. 60–78.

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

[15] K.L. Ho and L. Greengard, A fast direct solver for structured linear systems by recursive skeletonization, ArXiv e-prints, (2011).

[16] K.L. Ho and L. Ying, Hierarchical interpolative factorization for elliptic operators: differential equations, ArXiv e-prints, (2013).

[17] K.L. Ho and L. Ying, Hierarchical interpolative factorization for elliptic operators: integral equations, ArXiv e-prints, (2013).

[18] R. Kress, Linear Integral Equations, no. v. 82 in Applied Mathematical Sciences, Springer New York, 1999.

[19] E. Liberty, F. Woolfe, P.G. Martinsson, V. Rokhlin, and M. Tygert, Randomized algorithms for the low-rank approximation of matrices, Proceedings of the National Academy of Sciences, 104 (2007), pp. 20167–20172.

[20] P.G. Martinsson and V. Rokhlin, A fast direct solver for boundary integral equations in two dimensions, J. Comput. Phys., 205 (2005), pp. 1–23.

[21] X.M. Pan, J.G. Wei, Z. Peng, and X.Q. Sheng, A fast algorithm for multiscale electromagnetic problems using interpolative decomposition and multilevel fast multipole algorithm, Radio Science, 47 (2012), pp. n/a–n/a.

[22] E.S. Quintana-Ortí and R.A. Van De Geijn, Updating an LU factorization with pivoting, ACM Trans. Math. Softw., 35 (2008), pp. 11:1–11:16.

[23] Y. Saad and M. H.Kwatz, GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 856–869.

[24] P. Starr and V. Rokhlin, On the numerical solution of two-point boundary value problems II, Communications on Pure and Applied Mathematics, 47 (1994), pp. 1117–1159.

[25] J.L. Xia, S. Chandrasekaran, M. Gu, and X.S. Li, Fast algorithms for hierarchically semiseparable matrices, Numerical Linear Algebra With Applications, 17 (2010), p. 953–976.

[26] L. Ying, G. Biros, and D. Zorin, A kernel-independent adaptive fast multipole algorithm in two and three dimensions, J. Comput. Phys., 196 (2004), pp. 591–626.