FAST FACTORIZATION UPDATE FOR GENERAL ELLIPTIC EQUATIONS UNDER MULTIPLE COEFFICIENT UPDATES

XIAO LIU*, JIANLIN XIA†, AND MAARTEN V. DE HOOP‡

Abstract. For discretized elliptic equations, we develop a new factorization update algorithm that is suitable for incorporating coefficient updates with large support and large magnitude in subdomains. When a large number of local updates are involved, in addition to the standard factors in various (interior) subdomains, we precompute some factors in the corresponding exterior subdomains. Exterior boundary maps are constructed hierarchically. The data dependencies among tree-based interior and exterior factors are exploited to enable extensive information reuse. For coefficient updates in a subdomain, *only the interior problem in that subdomain needs to be re-factorized and there is no need to propagate updates to other tree nodes.* The combination of the new interior factors with a chain of existing factors quickly provides the new global factor and thus an effective solution algorithm. The introduction of exterior factors avoids updating higher-level subdomains with large system sizes, and makes the idea suitable for handling multiple occurrences of updates. The method can also accommodate the case when the support of updates changes to different subdomains. Numerical tests demonstrate the efficiency and especially the advantage in complexity over a standard factorization update algorithm.

Key words. elliptic equations, coefficient update, fast factorization update, exterior boundary map, exterior factor, Schur complement domain decomposition

AMS subject classifications. 15A23, 65F05, 65N22, 65Y20

1. Introduction.

In the solution of elliptic partial differential equations (PDEs) in practical fields such as inverse problems and computational biology, one often needs to update the coefficients associated with subdomains. For example, one key application in inverse problems is the iterative reconstruction of the wavespeed governed by the Helmholtz equation [21], which needs to incorporate modified coefficients into the following *reference problem*:

$$Lu = f \text{ in } D, \quad L = -\nabla \cdot p_2(x)\nabla + p_1(x)\cdot\nabla + p_0(x), \tag{1.1}$$

where D is the domain of interest, L is the partial differential operator, and $p_0(x)$, $p_1(x)$, and $p_2(x)$ are coefficient functions of L with the variable x representing a point in D. Standard boundary conditions can be imposed on ∂D (the boundary of D), including:

- Dirichlet boundary conditions such as $u = 0$ on ∂D.
- Neumann boundary conditions such as $\nu \cdot p_2(x)\nabla u = 0$ on ∂D, where ν denotes the outward unit normal vector. This boundary condition corresponds to the leading-order term of L, as can be seen from integration by parts

$$-\int_D (\nabla \cdot p_2(x)\nabla u) v \mathrm{d}x = \int_D (p_2(x)\nabla u)\cdot\nabla v \mathrm{d}x - \int_{\partial D} (\nu \cdot p_2(x)\nabla u) v \mathrm{d}\sigma, \tag{1.2}$$

*Department of Computational and Applied Mathematics, Rice University, Houston, TX 77005 (xiao.liu@rice.edu)

†Department of Mathematics and Department of Computer Science, Purdue University, West Lafayette, IN 47907 (xiaj@purdue.edu). The research of Jianlin Xia was supported in part by NSF CAREER Award DMS-1255416 and NSF Grant DMS-1819166.

‡Department of Computational and Applied mathematics, Rice University, Houston, TX 77005 (mdehoop@rice.edu). Maarten V. de Hoop gratefully acknowledges support from the Simons Foundation under the MATH + X program, NSF under grant DMS-1559587, and the corporate members of the Geo-Mathematical Group at Rice University and Total.

where v is a test function used for deriving the corresponding weak formulation and σ is the surface measure on ∂D. Clearly, p_2 shows up in this boundary condition due to integration by parts, but lower-order terms of L are not involved in this boundary condition.

- Robin boundary conditions such as $\alpha u + \nu \cdot p_2(x)\nabla u = 0$ on ∂D, where α is some scalar constant.

If inhomogeneous boundary conditions are involved, then we assume that the nonzero functions are absorbed into the right-hand side function f. After discretizations with continuous Galerkin [7] or finite difference approaches, we get a system of linear equations with a sparse coefficient matrix. The right-hand side may also be sparse when the function f has local support, but we do not rely on this type of sparsity here.

1.1. Coefficient update problem. Given the reference problem (1.1), the *coefficient update problem* is written as

$$\tilde{L}\tilde{u} = f \text{ in } D, \quad \tilde{L} = -\nabla \cdot \tilde{p}_2(x)\nabla + \tilde{p}_1(x) \cdot \nabla + \tilde{p}_0(x), \tag{1.3}$$

where $\tilde{p}_0(x)$, $\tilde{p}_1(x)$, and $\tilde{p}_2(x)$ are the modified coefficients and $\tilde{u}$ is the new solution.

The modification is localized if the coefficient update $(\tilde{L} - L)$ has small support. Assume that the function f is the same for both (1.1) and (1.3) and that we know the reference solution u of (1.1). Then (1.3) is equivalent to

$$\tilde{L}(\tilde{u} - u) = f - \tilde{L}u = (L - \tilde{L})u. \tag{1.4}$$

In order to update the solution from u to $\tilde{u}$, one can either solve (1.3) for $\tilde{u}$ directly or solve (1.4) for the difference $\tilde{u} - u$. Regarding the support of the right-hand side, f in (1.3) is not guaranteed to be locally supported, but the support of $(L - \tilde{L})u$ used in (1.4) is always contained in the support of the coefficient update $\tilde{L} - L$. The reason is that the right-hand side $(L - \tilde{L})u$ is zero at locations where L equals $\tilde{L}$. Hence, we choose to solve (1.4) for the update term $\tilde{u} - u$.

There are several strategies for solving either (1.3) or (1.4). For iterative solution, one can either reuse the preconditioner for L or perform additional changes for better convergence. For direct solution, if there is only a small amount of local updates, then the Sherman-Morrison-Woodbury (SMW) formula may be used [39]. However, if there are many local updates (or a sequence of local updates), then a *factorization update* from L to $\tilde{L}$ is preferred. Standard factorization update methods follow the data dependencies in the factorization processes, and recompute those factors that are changed. Here, we propose a different approach that significantly reduces the cost by changing the data dependencies according to the locations of the updates.

1.2. Existing work. Sparse direct solvers provide robust solutions to the fixed reference problem (1.1). After nested dissection reordering [12], the factorization of an $n \times n$ sparse discretized matrix generally costs $O(n^{3/2})$ in 2D, and $O(n^2)$ in 3D. Recent software packages provide the option of solving sparse right-hand sides, for example, MUMPS [28, 31] and PARDISO [32, 29]. A similar factorization process can be derived from Schur-complement domain decomposition strategies [5, 15, 18, 26, 30].

In recent years, rank-structured representations were developed to effectively compress fill-in and obtain fast factorizations of elliptic problems. Several such representations are $\mathcal{H}$ matrices [16], $\mathcal{H}^2$ matrices[17], and hierarchically semiseparable (HSS) matrices [3, 37]. Sparse factorization with HSS operations is proposed in [14, 34, 35, 36].

Updating LU factorizations of general matrices has been studied in [2, 4, 8, 13]. For sparse factorizations, these methods propagate updates from child nodes to ancestors in elimination trees. For dense discretized integral operators, updates to local geometries and kernels are studied in [9, 27, 40]. In [9], the update of the structures and the values of hierarchical matrices under adaptive refinement is discussed. In [27], the changes are propagated bottom-up in a quadtree. The SMW formula is used in [40] to compute the action of the inverse. For all of these methods, the updates are typically restricted to a few entries or low-rank updates. If the updates have large support or move locations, these methods may become inefficient.

For updating the coefficients in the PDE problem (1.3), the amount of modifications can be large due to the volumetric change in the support of $(\tilde{L} - L)$. For such a situation, it is beneficial to decompose the problem into a modified interior problem and a fixed exterior problem. This idea traces back to [21, 22], where boundary integral equations are formulated for piecewise constant media. For inhomogeneous reference problems, related formulations are developed in [20, 33], where the fundamental solution is replaced by the inverse matrix of some finite difference stencil. In order to efficiently precompute selected parts of the inverse, the location of the updates usually needs to be fixed.

1.3. Overview of the proposed method. We propose a new direct update method to solve (1.4) that does not need to propagate computational information globally like in standard factorization update approaches. The method is suitable for coefficient updates with different locations and volumes. The method has a precomputation step that factorizes the reference problem in various interior and exterior subdomains. When the problem changes in some subdomains, re-factorizations in those subdomains are not avoidable for direct methods. In our proposed method, the factorization update is only restricted to those subdomains with updates and is thus highly efficient. The solution is updated by solving (1.4) using the locality of the new right-hand side.

The method starts from a domain partitioning governed by a binary tree (denoted by $\mathcal{T}$), similarly to related direct solvers [18, 26, 15], and the binary tree is an analogue of the assembly tree [11]. In the factorization of the reference problem, *interior boundary value problems* for adjacent subdomains are combined by eliminating their shared interface. The work flow is *bottom-up* in $\mathcal{T}$. That is, child nodes pass data to parents.

For solving coefficient update problems with a relatively large amount of updates, we precompute additional factors following a *top-down* traversal of $\mathcal{T}$ before knowing the specific region or value of perturbations. As a major novelty of this work, the top-down process constructs factors for *exterior boundary value problems*, which helps to bypass existing data dependencies. Then for the solution of (1.4), we only re-factorize the smallest subdomain containing the updates, and select existing factors of exterior problems which remain unchanged. For each subtree $\tilde{\mathcal{T}} \subset \mathcal{T}$ corresponding to the updates, the solution update algorithm treats the nodes inside and outside $\tilde{\mathcal{T}}$ separately. Inside $\tilde{\mathcal{T}}$, the solution algorithm is similar to the traditional one, but requires the factors of the updated system. Outside $\tilde{\mathcal{T}}$, a boundary value problem is solved using the factorization of the exterior problems.

The advantages of our method include:

- For the factorization update, the use of tree-based interior and exterior factors enables us to change only the factors inside the region of coefficient updates, namely, only the nodes in $\tilde{\mathcal{T}}$. There is *no propagation of updates to other* *nodes*. Thus, *the factorization update cost only depends on the size of the updates* instead of the total number of unknowns.

- The method is suitable for incorporating coefficient updates with *large support and large magnitude in subdomains.*
- Because the precomputation prepares for coefficient updates in *any subtree* of $\mathcal{T}$, *the supports of updates are allowed to change to different subdomains.*

The method is tested on the transmission problem for the Helmholtz equation [22]. The precomputation has the same scaling as related direct factorizations. The method is especially suitable for large number of changes (e.g., 10^5 points), because the re-factorization cost is *independent of the total number of unknowns.*

The remaining sections are organized as follows. We formulate the interior and exterior problems in Section 2. Hierarchical factorization algorithms are developed in Section 3 for the coefficient update problems. The algorithm complexity is estimated in Section 4 and is supported by the performance tests in Section 5. In Section 6, some conclusions are drawn and future work is discussed.

2. Interior and exterior problems and basic solution update methods.

Factorization update problems can be complicated in general because there are many different scenarios regarding the locations and sizes of the updates. We first present the method for the simplest case and then generalize it to more advanced forms. In Section 2.1, updates in fixed locations are solved by a one-level relation between an interior and an exterior problem. In Section 2.2, a two-level method gives additional flexibility to change the locations and sizes of the updates.

The problem of changing the coefficient in the interior of a subdomain is originally formulated and solved using potential theory, see for example [22, Theorem 4.1]. Note that the fundamental solution is challenging to compute or to store in inhomogeneous media. We choose instead a Schur-complement domain decomposition formulation, which focuses on solving sub-problems on the boundaries of subdomains.

Let Ω_i be an open subdomain of D indexed by an integer i, we start by introducing unknowns in the interior of Ω_i and on the non-physical boundary $\partial\Omega_i - \partial D$, where the minus sign denotes the set theoretical difference. (Later, all subdomains like Ω_i are assumed to be open.) If we want to restrict the PDE (1.1) in Ω_i, a boundary condition is needed on the non-physical boundary to obtain a uniquely solvable problem. We choose to impose a Robin boundary condition, which has historically been used in domain decomposition formulations (see, e.g., [10]). For direct methods, it is shown in [15, 30] that the set of linear equations derived from the Robin boundary condition has some unique block structures like in [30, Equation (2.9)] and (2.9) to be derived in Section 2.2. Those block structures are convenient for solving factorization update problems in Section 2.2. Therefore, we consider an auxiliary local PDE problem in the following form:

$$\begin{cases} Lu^{(i)} = f^{(i)} & \text{in } \Omega_i, \\ \alpha u^{(i)} + \nu \cdot \left(p_2 \nabla u^{(i)}\right) = g^{(i)} & \text{on } \partial\Omega_i - \partial D, \end{cases} \tag{2.1}$$

where L is defined in (1.1) with leading-order coefficient function $p_2(x)$, $f^{(i)}$ is called the *interior source*, $g^{(i)}$ is called the *boundary source*, ν is the outward unit normal vector, and α is a nonzero scalar coefficient in the Robin-type boundary condition. The problem (2.1) focuses on the part of L restricted to the subdomain Ω_i. Again, p_2 appears in the boundary condition because of integration by parts (1.2). The free parameter α is chosen for well-posedness in the sense of Hadamard. A positive α is suitable for the Poisson problem ($L = -\Delta$) as in [10], and an imaginary α is often used for the Helmholtz problem as in [15, 30].

Suppose there is a way to solve the problem (2.1) for given $f^{(i)}$ and $g^{(i)}$. In order for the solution of (2.1) to be the same as that of (1.1) in Ω_i, $f^{(i)}$ in Ω_i needs to be the same as f in (1.1), and an *interface problem* needs to be formulated and solved to get the correct $g^{(i)}$. To prepare for the interface problem, we introduce the *boundary data* $\hat{g}^{(i)}$ on $\partial\Omega_i - \partial D$ defined as

$$\hat{g}^{(i)} = -\alpha u^{(i)} + \nu \cdot \left(p_2 \nabla u^{(i)}\right), \quad \text{on } \partial\Omega_i - \partial D. \tag{2.2}$$

$\hat{g}^{(i)}$ differs from $g^{(i)}$ by a minus sign in the term $-\alpha u^{(i)}$. Observe that $\hat{g}^{(i)}$ has a linear relation with $f^{(i)}$ and $g^{(i)}$, which can be written formally as

$$\hat{g}^{(i)} = T^{(i)} g^{(i)} + S^{(i)} f^{(i)} = \begin{pmatrix} T^{(i)} & S^{(i)} \end{pmatrix} \begin{pmatrix} g^{(i)} \\ f^{(i)} \end{pmatrix}, \tag{2.3}$$

where $T^{(i)}$ is the *boundary map* from the boundary source $g^{(i)}$ to the boundary data $\hat{g}^{(i)}$, and $S^{(i)}$ is the *interior-to-boundary map* from the interior source $f^{(i)}$ to the boundary data. After discretizations, the problem (2.1) can be solved using a direct factorization. The goal of introducing $T^{(i)}$ and $S^{(i)}$ is to reduce the PDE problem (1.1) to a subproblem on the artificial interface $\partial\Omega_i - \partial D$, which is important for reducing the cost of the factorization update. Note that if there is no minus sign in $-\alpha u^{(i)}$ in (2.2) (i.e., $\hat{g}^{(i)} = g^{(i)}$), then $T^{(i)}$ is an identity operator, $S^{(i)} = 0$, and they lose all the information about the PDE.

$T^{(i)}$ is a square dense matrix, and the size equals the number of unknowns on the artificial $\partial\Omega_i - \partial D$ which is usually much smaller than the number of unknowns in the subdomain Ω_i. $S^{(i)}$ has the same number of rows as $T^{(i)}$, but the number of columns is the number of unknowns in Ω_i. Explicit construction of $S^{(i)}$ should be avoided because the column size can be large. Although $T^{(i)}$ and $S^{(i)}$ are dense matrices and may not have explicit expressions for the entries, the matrix-vector product in (2.3) can be conveniently computed as follows:

$$\hat{g}^{(i)} = g^{(i)} - 2\alpha u^{(i)}, \quad \text{on } \partial\Omega_i - \partial D,$$

which is directly from (2.2) and the second equation of (2.1). This matrix-vector product is described in Algorithm 2.1 (TSMV) that will be frequently referenced later. One direct solution of (2.1) is needed to compute the product. For the rest of the paper, we use matrix notation for ease of exposition.

Algorithm 2.1 Matrix-vector product of $\begin{pmatrix} T^{(i)} & S^{(i)} \end{pmatrix}$ for the ith subdomain Ω_i

```
1: procedure TSMV($i, g^{(i)}, f^{(i)}$)          ▷ Compute $T^{(i)} g^{(i)} + S^{(i)} f^{(i)}$
2:     Solve (2.1) to get the solution $u^{(i)}$ in $\Omega_i$
3:     Compute $\hat{g}^{(i)} = g^{(i)} - 2\alpha u^{(i)}$ on $\partial\Omega_i - \partial D$ based on (2.1)–(2.2)
4:     return $\hat{g}^{(i)}$
5: end procedure
```

2.1. One-level method and interior and exterior problems. For solving the problem (1.4) with coefficient updates in Ω_i, we consider a one-level partitioning of D into the *interior subdomain* Ω_i and the *exterior subdomain* Ω_{-i} defined as the relative complement of Ω_i's closure in D. That is, $\Omega_{-i} = D - \overline{\Omega}_i$. Ω_i and Ω_{-i} share the artificial boundary

$$\partial\Omega_i - \partial D = \partial\Omega_{-i} - \partial D.$$

See the left panel of Figure 2.1 for an example. The index of the exterior subdomain is set as the negative of the index of the corresponding interior subdomain, and we assume that all interior subdomains have positive indices to avoid confusion. We call Ω_i and Ω_{-i} *level-one subdomains* of (the level-zero subdomain) D.

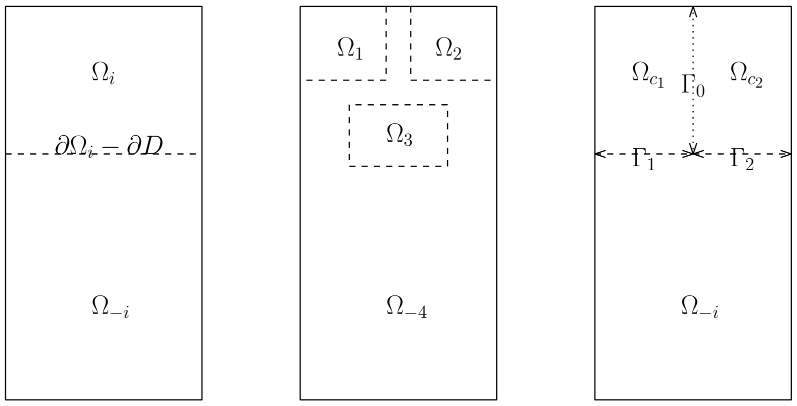


FIG. 2.1. *Illustrations of different types of domain partitioning in Section 2. Left panel: partitioning of D into Ω_i and Ω_{-i}; Middle panel: partitioning of D into $\Omega_1, \Omega_2, \Omega_3$ and Ω_{-4}; Right panel: partitioning of D into Ω_i and Ω_{-i} where Ω_i is further partitioned into Ω_{c_1} and Ω_{c_2}.*

Similar to (2.3), for the exterior subdomain Ω_{-i}, we have

$$\hat{g}^{(-i)} = T^{(-i)} g^{(-i)} + S^{(-i)} f^{(-i)}, \tag{2.4}$$

where $T^{(-i)}$ ($S^{(-i)}$) is the boundary map (interior-to-boundary map) for Ω_{-i}. Following [10, Equation (1.3)], the *transmission condition* on $\partial\Omega_i - \partial D$ is

$$g^{(i)} = -\hat{g}^{(-i)}, \quad \hat{g}^{(i)} = -g^{(-i)}, \tag{2.5}$$

since the outward normal changes sign across the interface. By eliminating the boundary data $\hat{g}^{(\pm i)}$ in (2.3)–(2.4), we get the following interface problem:

$$\begin{pmatrix} T^{(i)} & I \\ I & T^{(-i)} \end{pmatrix} \begin{pmatrix} g^{(i)} \\ g^{(-i)} \end{pmatrix} = \begin{pmatrix} -S^{(i)} f^{(i)} \\ -S^{(-i)} f^{(-i)} \end{pmatrix}. \tag{2.6}$$

We can define and factorize the coupled matrix as

$$M^{(i,-i)} := \begin{pmatrix} T^{(i)} & I \\ I & T^{(-i)} \end{pmatrix} = \begin{pmatrix} I & T^{(i)} \\ & I \end{pmatrix} \begin{pmatrix} & I - T^{(i)} T^{(-i)} \\ I & T^{(-i)} \end{pmatrix}. \tag{2.7}$$

Based on the current formulation, we propose an algorithm for directly solving the simplest coefficient update problem in which the region of modifications Ω_i is known and fixed. Here, we assume that L is discretized in $\Omega_{\pm i}$ using, say, a finite element method. The factorization operations related to the reference operator L include the following steps.

1. Factorize the discretized operator L in $\Omega_{\pm i}$ by a sparse LU factorization.

2. Construct $T^{(\pm i)}$. The jth column of $T^{(\pm i)}$ can be computed by calling $\mathsf{TSMV}(\pm i, e_j, 0)$ in Algorithm 2.1, where e_j is the jth column of the identity matrix. To improve the efficiency, this multiplication is computed with multiple right-hand sides.

Then for each coefficient update problem (1.3), the solution process has three steps.

1. Solve the reference problem $Lu = f$:
   (a) Set $f^{(\pm i)}$ as f restricted to $\Omega_{\pm i}$.
   (b) Solve the interface problem (2.6) for $g^{(\pm i)}$, where $S^{(\pm i)} f^{(\pm i)}$ is computed by calling $\mathsf{TSMV}(\pm i, 0, f^{(\pm i)})$.
   (c) Solve the local PDE (2.1) in $\Omega_{\pm i}$ to get $u^{(\pm i)}$, which is the solution u in $\Omega_{\pm i}$.
2. Factorize the discretized operator $\tilde{L}$ in Ω_i, and construct the new boundary map $\tilde{T}^{(i)}$ using TSMV.
3. Solve the coefficient update problem $\tilde{L}(\tilde{u} - u) = (L - \tilde{L})u$ by applying Step 1(a)–(c) to the new right-hand side $(L - \tilde{L})u$, where the new factors are used in Ω_i.

The factorization cost of the reference operator depends on the size and shape of $\Omega_{\pm i}$, and the factorization update cost depends on the size of Ω_i. If the interior subdomain Ω_i is much smaller than the exterior one Ω_{-i}, the method is very effective because the factorization in Ω_i is much cheaper than that in Ω_{-i}. Solving the coefficient update problem does not involve $S^{(-i)}$ because $L = \tilde{L}$ in Ω_{-i}.

Remark 2.1. Before describing more sophisticated generalizations, we show that this method can already be beneficial for *coefficient updates in disjoint locations*. If the problem can be modified in at most $i-1$ subdomains denoted by $\{\Omega_j : j = 1, 2, \ldots, i-1\}$ with disjoint closure, then we choose $\Omega_i = \bigcup_{j=1}^{i-1} \Omega_j$ as their union. The middle panel of Figure 2.1 gives an example for $i = 4$. The factorization and solution update method is the same as before, and we only highlight one additional property. For the factorization (solution) in Ω_i, we factorize (solve) the problems in the subdomains $\Omega_1, \Omega_2, \ldots, \Omega_{i-1}$ independently. Each operator for Ω_i is decoupled here, that is

$$\begin{aligned} T^{(i)} &= \operatorname{diag}(T^{(1)}, T^{(2)}, \ldots, T^{(i-1)}), \\ S^{(i)} &= \operatorname{diag}(S^{(1)}, S^{(2)}, \ldots, S^{(i-1)}), \end{aligned}$$

where diag() is used to denote a block diagonal matrix. Because of the decoupled forms, the method is essentially still a one-level method and the level-one subdomains are $\Omega_1, \Omega_2, \ldots, \Omega_{i-1}$, and Ω_{-i}. The factorization update cost contains the sum of the re-factorization costs in $\Omega_1, \Omega_2, \ldots, \Omega_{i-1}$, and the re-factorization cost of $M^{(i,-i)}$ which depends cubically on the total number of points on those boundaries $\partial\Omega_1 - \partial D$, $\partial\Omega_2 - \partial D$, $\ldots$, $\partial\Omega_{i-1} - \partial D$. This is better than a complete re-factorization when the subdomains Ω_j's have small sizes.

2.2. Two-level method. If a level-one subdomain Ω_i is partitioned further into two non-overlapping subdomains $\Omega_{c_1}, \Omega_{c_2}$ as in the right panel of Figure 2.1, and coefficient updates may be restricted to one of the subdomains, then the domain decomposition framework (2.1) and (2.6) applies to Ω_{c_1} and Ω_{c_2} as well by changing the interior subdomain. The method in Section 2.1 is not optimal here because it either recomputes everything when the interior subdomain changes, or updates the factorization in the large subdomain Ω_i for all the cases. Here, we discuss a two-level direct method that improves the effectiveness by exploiting shared information for different cases.

The method is based on the inherent dependencies among different subdomains. The set of subdomains has a partial order governed by the subset relation "$\subseteq$". The graph in Figure 2.2 visualizes the partial order, each edge of which starts from a subset and points to a superset. Three tree structures can be extracted from the graph in Figure 2.2, which are illustrated separately in Figure 2.3. According to the support of coefficient modifications, one of the three tree structures can be selected to solve the problem:

- For modifications in the large subdomain Ω_i, the interior subdomain is Ω_i which contains Ω_{c_1} and Ω_{c_2}, and the exterior subdomain is Ω_{-i};
- For modifications in Ω_{c_1}, the interior subdomain is Ω_{c_1}, and the exterior subdomain is Ω_{-c_1} which contains Ω_{c_2} and Ω_{-i};
- For modifications in Ω_{c_2}, the interior subdomain is Ω_{c_2}, and the exterior subdomain is Ω_{-c_2} which contains Ω_{c_1} and Ω_{-i}.

For Ω_i, Ω_{-c_1}, and Ω_{-c_2}, each one contains two subdomains. Here, it is important to effectively combine the results from smaller subdomains.

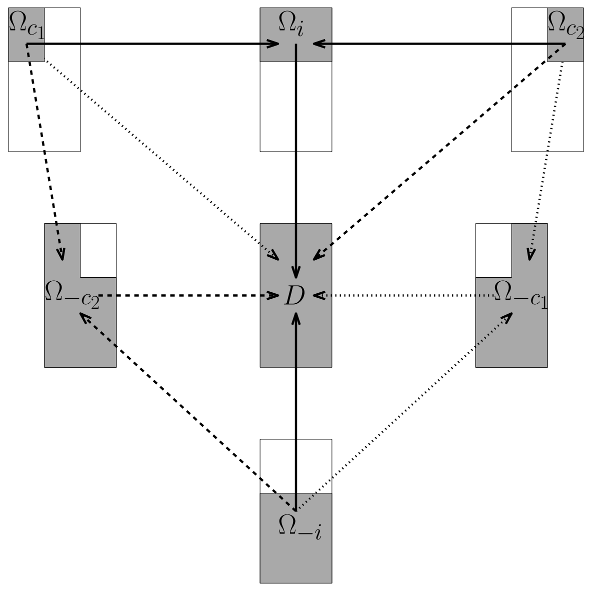


FIG. 2.2. *Graph structures of the two-level method in Section 2.2. The solid, dashed, and dotted edges give the three trees in Figure 2.3. Each arrow points from a subset to a superset. The geometric relations are based on the right panel of Figure 2.1. Each shaded area represents a subdomain.*

The factorization of the related interior and exterior problems have some similarities with the simplest case (2.6), but the formulas become more sophisticated because now Ω_{c_1}, Ω_{c_2}, and Ω_{-i} have different shared boundaries. We define them as

$$\Gamma_0 = (\partial\Omega_{c_1} \cap \partial\Omega_{c_2}) - \partial D, \quad \Gamma_1 = (\partial\Omega_{c_1} \cap \partial\Omega_{-i}) - \partial D, \quad \Gamma_2 = (\partial\Omega_{c_2} \cap \partial\Omega_{-i}) - \partial D. \tag{2.8}$$

The right panel of Figure 2.1 illustrates their locations.

Similar to the derivation from (2.5) to (2.6), solution operators for Ω_i can be

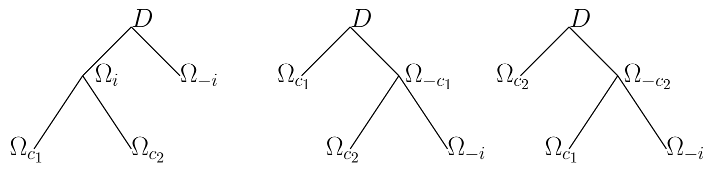


FIG. 2.3. *Tree structures extracted from Figure 2.2. The three trees have the same set of leaves:* $\Omega_{c_1}, \Omega_{c_2}, \Omega_{-i}$.

obtained from merging Ω_{c_1} and Ω_{c_2}. The same transmission condition (2.5) is imposed on Γ_0, and we have

$$\begin{pmatrix} T_{0,0}^{(c_1)} & I & T_{0,1}^{(c_1)} & 0 \\ I & T_{0,0}^{(c_2)} & 0 & T_{0,2}^{(c_2)} \\ T_{1,0}^{(c_1)} & 0 & T_{1,1}^{(c_1)} & 0 \\ 0 & T_{2,0}^{(c_2)} & 0 & T_{2,2}^{(c_2)} \end{pmatrix} \begin{pmatrix} g_0^{(c_1)} \\ g_0^{(c_2)} \\ g_1^{(c_1)} \\ g_2^{(c_2)} \end{pmatrix} = \begin{pmatrix} -h_0^{(c_1)} \\ -h_0^{(c_2)} \\ \hat{g}_1^{(c_1)} - h_1^{(c_1)} \\ \hat{g}_2^{(c_2)} - h_2^{(c_2)} \end{pmatrix}, \tag{2.9}$$

where $g_k^{(j)}$ denotes the restriction of $g^{(j)}$ on Γ_k, $T_{0,1}^{(j)}$ denotes the restriction of $T^{(j)}$ on $\Gamma_0 \times \Gamma_1$, $h_0^{(c_1)}$ denotes the restriction of $h^{(c_1)} := S^{(c_1)} f^{(c_1)}$ on Γ_0, $h_0^{(c_2)}$ denotes the restriction of $h^{(c_2)} := S^{(c_2)} f^{(c_2)}$ on Γ_0, and the other notation can be similarly understood. The equation is rewritten from (2.3) for Ω_{c_1} and Ω_{c_2}, and the transmission condition is substituted in the first two block rows to eliminate $\hat{g}_0^{(c_1)}$ and $\hat{g}_0^{(c_2)}$. The coupling between subdomains lies in the leading 2×2 block

$$M^{(c_1,c_2)} = \begin{pmatrix} T_{0,0}^{(c_1)} & I \\ I & T_{0,0}^{(c_2)} \end{pmatrix}. \tag{2.10}$$

Choose the boundary and interior sources for Ω_i as $g^{(i)} = \begin{pmatrix} g_1^{(c_1)} \\ g_2^{(c_2)} \end{pmatrix}$ and $f^{(i)} = \begin{pmatrix} f^{(c_1)} \\ f^{(c_2)} \end{pmatrix}$, respectively. Similar to derivation in [30, Equations (2.9)–(2.14)], the Schur complement system of $M^{(c_1,c_2)}$ in (2.9) is essentially

$$T^{(i)} g^{(i)} = \begin{pmatrix} \hat{g}_1^{(c_1)} \\ \hat{g}_2^{(c_2)} \end{pmatrix} - S^{(i)} f^{(i)},$$

where

$$T^{(i)} = \begin{pmatrix} T_{1,1}^{(c_1)} & \\ & T_{2,2}^{(c_2)} \end{pmatrix} - \begin{pmatrix} T_{1,0}^{(c_1)} & \\ & T_{2,0}^{(c_2)} \end{pmatrix} (M^{(c_1,c_2)})^{-1} \begin{pmatrix} T_{0,1}^{(c_1)} & \\ & T_{0,2}^{(c_2)} \end{pmatrix}, \tag{2.11}$$

$$S^{(i)} f^{(i)} = \begin{pmatrix} h_1^{(c_1)} \\ h_2^{(c_2)} \end{pmatrix} - \begin{pmatrix} T_{1,0}^{(c_1)} & \\ & T_{2,0}^{(c_2)} \end{pmatrix} (M^{(c_1,c_2)})^{-1} \begin{pmatrix} h_0^{(c_1)} \\ h_0^{(c_2)} \end{pmatrix}. \tag{2.12}$$

We do not form $S^{(i)}$ explicitly because it can be much larger than the boundary map $T^{(i)}$. (2.12) can be used to compute fast matrix-vector products instead.

For the exterior subdomain Ω_{-c_1}, we merge Ω_{c_2} and Ω_{-i} with similar procedures. Using the transmission condition (2.5) on Γ_2 and ignoring the interior sources for simplicity, we have

$$\begin{pmatrix} T_{2,2}^{(c_2)} & I & T_{2,0}^{(c_2)} & 0 \\ I & T_{2,2}^{(-i)} & 0 & T_{2,1}^{(-i)} \\ T_{0,2}^{(c_2)} & 0 & T_{0,0}^{(c_2)} & 0 \\ 0 & T_{1,2}^{(-i)} & 0 & T_{1,1}^{(-i)} \end{pmatrix} \begin{pmatrix} g_2^{(c_2)} \\ g_2^{(-i)} \\ g_0^{(c_2)} \\ g_1^{(-i)} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \hat{g}_0^{(c_2)} \\ \hat{g}_1^{(-i)} \end{pmatrix}. \tag{2.13}$$

(2.13) is derived in the same way as (2.9), but is not equivalent to (2.9). Let the leading 2×2 block be

$$M^{(c_2,-i)} = \begin{pmatrix} T_{2,2}^{(c_2)} & I \\ I & T_{2,2}^{(-i)} \end{pmatrix}. \tag{2.14}$$

By computing the Schur complement of $M^{(c_2,-i)}$, we get

$$T^{(-c_1)} = \begin{pmatrix} T_{0,0}^{(c_2)} & \\ & T_{1,1}^{(-i)} \end{pmatrix} - \begin{pmatrix} T_{0,2}^{(c_2)} & \\ & T_{1,2}^{(-i)} \end{pmatrix} (M^{(c_2,-i)})^{-1} \begin{pmatrix} T_{2,0}^{(c_2)} & \\ & T_{2,1}^{(-i)} \end{pmatrix}. \tag{2.15}$$

Clearly, we can also merge Ω_{c_1} and Ω_{-i} by exchanging the role of c_1 and c_2 in (2.14)–(2.15).

After the technical derivations, we would like to point out the key relationships among boundary maps that govern the factorization algorithm. According to (2.11) and previous derivations in [15, 30], the interior boundary maps have the following structure:

$$\begin{pmatrix} T_{0,0}^{(c_1)} & T_{0,1}^{(c_1)} \\ T_{1,0}^{(c_1)} & T_{1,1}^{(c_1)} \end{pmatrix}, \begin{pmatrix} T_{0,0}^{(c_2)} & T_{0,2}^{(c_2)} \\ T_{2,0}^{(c_2)} & T_{2,2}^{(c_2)} \end{pmatrix} \xrightarrow[\text{eliminate } \Gamma_0]{\text{factorize } M^{(c_1,c_2)}} \begin{pmatrix} T_{1,1}^{(i)} & T_{1,2}^{(i)} \\ T_{2,1}^{(i)} & T_{2,2}^{(i)} \end{pmatrix},$$

where points on Γ_0 need to be eliminated because they are inside Ω_i. For the exterior ones, we similarly have

$$\begin{pmatrix} T_{0,0}^{(c_2)} & T_{0,2}^{(c_2)} \\ T_{2,0}^{(c_2)} & T_{2,2}^{(c_2)} \end{pmatrix}, \begin{pmatrix} T_{1,1}^{(-i)} & T_{1,2}^{(-i)} \\ T_{2,1}^{(-i)} & T_{2,2}^{(-i)} \end{pmatrix} \xrightarrow[\text{eliminate } \Gamma_2]{\text{factorize } M^{(c_2,-i)}} \begin{pmatrix} T_{0,0}^{(-c_1)} & T_{0,1}^{(-c_1)} \\ T_{1,0}^{(-c_1)} & T_{1,1}^{(-c_1)} \end{pmatrix},$$

$$\begin{pmatrix} T_{0,0}^{(c_1)} & T_{0,1}^{(c_1)} \\ T_{1,0}^{(c_1)} & T_{1,1}^{(c_1)} \end{pmatrix}, \begin{pmatrix} T_{1,1}^{(-i)} & T_{1,2}^{(-i)} \\ T_{2,1}^{(-i)} & T_{2,2}^{(-i)} \end{pmatrix} \xrightarrow[\text{eliminate } \Gamma_1]{\text{factorize } M^{(c_1,-i)}} \begin{pmatrix} T_{0,0}^{(-c_2)} & T_{0,2}^{(-c_2)} \\ T_{2,0}^{(-c_2)} & T_{2,2}^{(-c_2)} \end{pmatrix}.$$

Notice the following important points.

- Instead of factorizing the exterior problems in Ω_{-c_1} and Ω_{-c_2} independently, we have *reused the factorization results* from the existing interior subdomains Ω_{c_2} and Ω_{c_1}, and also another exterior subdomain Ω_{-i} which has a smaller size than Ω_{-c_1} and Ω_{-c_2}.
- Assuming that one has the appropriate data structures for storing interior boundary maps [15, 30], then it is easy to see that each exterior boundary map $T^{(-i)}$ has the same format as the corresponding interior one $T^{(i)}$. The major difference is in the pivot blocks: $M^{(c_1,c_2)}$, $M^{(c_2,-i)}$, and $M^{(c_1,-i)}$ are not related to one another because they are for different parts of the boundaries.

Finally, for computing the solution update, we develop tree-based algorithms built upon the leaf subdomains Ω_{c_1}, Ω_{c_2}, and Ω_{-i} by using (2.10)–(2.15). For example, if the coefficient updates and the right-hand sides are supported in Ω_{c_1}, the solution process is as follows.

1. Factorize the updated operator $\tilde{L}$ in Ω_{c_1} and form $\tilde{T}^{(c_1)}$.
2. Solve the coupled system (2.6) for $\partial\Omega_{c_1}$:

$$\begin{pmatrix} \tilde{T}^{(c_1)} & I \\ I & T^{(-c_1)} \end{pmatrix} \begin{pmatrix} g^{(c_1)} \\ g^{(-c_1)} \end{pmatrix} = \begin{pmatrix} -\tilde{S}^{(c_1)} f^{(c_1)} \\ 0 \end{pmatrix}.$$

3. Compute the solution in Ω_{c_1} by solving (2.1) with the factors of $\tilde{L}$ and sources $f^{(c_1)}$ and $g^{(c_1)}$.
4. Choose $g_0^{(c_2)} = g_0^{(-c_1)}$ on Γ_0 and $g_1^{(-i)} = g_1^{(-c_1)}$ on Γ_1, and then solve the first two block rows of (2.13) rewritten as

$$M^{(c_2,-i)} \begin{pmatrix} g_2^{(c_2)} \\ g_2^{(-i)} \end{pmatrix} = \begin{pmatrix} -T_{2,0}^{(c_2)} g_0^{(c_2)} \\ -T_{2,1}^{(-i)} g_1^{(-i)} \end{pmatrix}. \tag{2.16}$$

5. Compute the solution in Ω_{c_2} and Ω_{-i} by solving (2.1) with the factors of L and boundary sources $g^{(c_2)}$ and $g^{(-i)}$, respectively.

For steps 1 to 3, we follow the existing strategy in Section 2.1 by finding the correct boundary sources between the interior subdomain Ω_{c_1} and the exterior subdomain Ω_{-c_1}. For steps 4 to 5, we compute the solution update in Ω_{-c_1} by finding the boundary sources between the two subdomains Ω_{c_2} and Ω_{-i}. If we are only interested in having the solution near the coefficient updates, we can terminate the solution process at step 3 to save the solution cost.

This two-level method does not need to fix the locations of coefficient updates. Updates in Ω_i, Ω_{c_1}, and Ω_{c_2} are highly efficient since $\tilde{L}$ only needs to be factorized at the locations where it differs from L. This two-level process illustrates the capability of dealing with coefficient updates of different volumes. The results of this section provide key components of the hierarchical algorithms in Section 3.

3. Hierarchical algorithms.

In this section, we write the complete hierarchical algorithm for solving coefficient update problems. In particular, we focus on generalizing the two-level method in Section 2.2 to a constructive multi-level method. The multi-level method involves the tree-based domain partitioning. Comparing with simpler alternatives in Section 2, the multi-level method is more flexible because it supports updates in any subdomain used in the domain partitioning, and is more efficient because the computational cost is minimized by isolating the smallest subdomains containing the coefficient updates. Besides a factorization update in subdomains, the major steps include: introduction of exterior subdomains in the domain partitioning, factorization of interior and exterior problems, and solution update with localized right-hand sides.

The computational domain D is partitioned hierarchically following a tree denoted by $\mathcal{T}$. For notational simplicity, we restrict the discussion to binary trees. Each parent subdomain is the union of two child subdomains. Intuitive examples of the domain partitioning can be found in [15, Figure 2]. Here, we let every node in $\mathcal{T}$ have a positive index in order to introduce the indexing of exterior subdomains. As a tree-based solver, the basic design is as follows:

- For each leaf node i, Section 2.1 has described the way to solve the local problem (2.1) in the leaf subdomain Ω_i based on boundary and interior sources. We keep all the relevant information about (2.1) at leaf nodes, such as local mesh and coefficient information used to generate and update the local linear system.

- For each non-leaf node i, according to (2.8)–(2.9) in Section 2.2, we only need to keep track of the shared artificial boundaries with i's children.

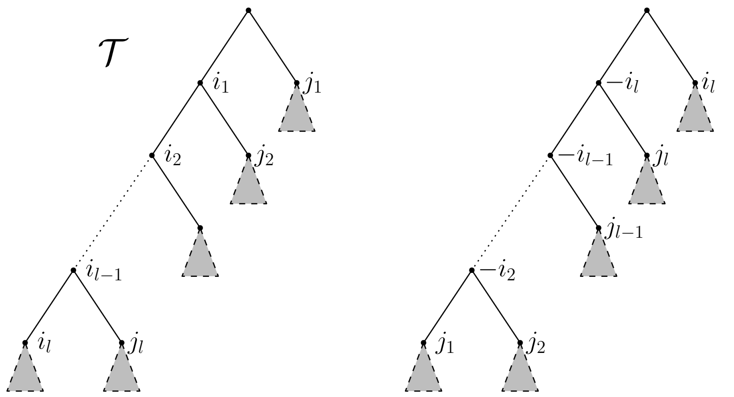


FIG. 3.1. *Transformation between trees of subdomains. Left panel: the original tree $\mathcal{T}$ with the associated subdomains; Right panel: the new tree for localized solution in Ω_{i_l}. Each shaded triangle associated with a node represents all the descendants of the node.*

3.1. Transformation of binary domain partitioning.

3.1. Transformation of binary domain partitioning. The domain partitioning needs to be updated when the coefficient changes. Suppose the problem is modified in Ω_p for a level-l node p. Write the path from the root i_0 to p as $i_0 \to i_1 \to \cdots \to i_l = p$, so $\Omega_{i_0} \supset \Omega_{i_1} \supset \cdots \supset \Omega_{i_l} = \Omega_p$. Therefore, modifications in Ω_p not only lead to changes in the subtree generated by p, but also propagate along the path to the root. The goal here is to reorganize the domain partitioning such that p is a child of the root, then changes in Ω_p do not propagate to multiple larger subdomains.

Denote i_k's sibling by j_k for $1 \le k \le l$. See the left panel of Figure 3.1 for the illustration of i_k, j_k in $\mathcal{T}$. In short, the related subdomains have the following relation in $\mathcal{T}$:

$$D = \Omega_{i_0} \xLeftarrow{\cup\Omega_{j_1}} \Omega_{i_1} \xLeftarrow{\cup\Omega_{j_2}} \Omega_{i_2} \xLeftarrow{\cup\Omega_{j_3}} \cdots \xLeftarrow{\cup\Omega_{j_l}} \Omega_{i_l}.$$

For the exterior subdomains on the path from i_1 to i_l, we have the following relation:

$$\Omega_{j_1} = \Omega_{-i_1} \xRightarrow{\cup\Omega_{j_2}} \Omega_{-i_2} \xRightarrow{\cup\Omega_{j_3}} \Omega_{-i_3} \xRightarrow{\cup\Omega_{j_4}} \cdots \xRightarrow{\cup\Omega_{j_l}} \Omega_{-i_l}.$$

Motivated by this relation, we construct the new binary domain partitioning step by step as follows.

1. For the root node i_0, let i_l, $-i_l$ be its children. The entire domain $D = \Omega_{i_0}$ can be partitioned into the interior subdomain Ω_{i_l} and the exterior subdomain Ω_{-i_l}. We preserve the partitioning in Ω_{i_l}, and continue with the new node $-i_l$.
2. For the node $-i_k$ with $k \in \{l, l-1, \ldots, 2\}$, let j_k, $-i_{k-1}$ be $-i_k$'s children. Since $\Omega_{i_{k-1}}$ contains Ω_{i_k} and Ω_{j_k} in $\mathcal{T}$, we can partition Ω_{-i_k} into Ω_{j_k} and $\Omega_{-i_{k-1}}$. We preserve the partitioning in Ω_{j_k} and continue with the new node $-i_{k-1}$. Notice that $\Omega_{j_1} = \Omega_{-i_1}$, so we can use j_1 to replace the new node $-i_{k-1}$ for $k = 2$.

The new binary tree is visualized in the right panel of Figure 3.1. The new tree can be constructed in $O(l)$ operations, because $l - 1$ nodes are removed and $l - 1$

nodes are introduced. From the construction process, we see that the new elements $\{-i_k\}$ are not leaf nodes. That is to say, every exterior subdomain introduced here is a union of existing interior subdomains. The key results are summarized into the following theorem.

THEOREM 3.1. *Given a binary tree $\mathcal{T}$, let $\{\Omega_i : i \in \mathcal{T}\}$ be a binary domain partitioning of D. For a level-l node $p \in \mathcal{T}$ with $l > 1$, there exists a well-defined binary domain partitioning such that*
1. *Ω_p is a child subdomain of D,*
2. *the elements of $\{\Omega_i : i$ is an ancestor of p in $\mathcal{T}, 1 \leq \mathrm{level}(i) < l\}$ are removed,*
3. *the elements of $\{\Omega_{-i} : i$ is an ancestor of p in $\mathcal{T}, 1 < \mathrm{level}(i) \leq l\}$ are inserted,*
4. *every new element cannot be a leaf in the new binary partitioning.*

The new domain partitioning is used to isolate the perturbations in Ω_p, because the level-one subdomains are precisely Ω_p and Ω_{-p}. The interior problem in Ω_p needs to be re-factorized, but the exterior problem in Ω_{-p} remains the same.

3.2. Hierarchical factorization and solution update. Inspired by the two-level example in Section 2.2, we describe the family of hierarchical algorithms needed for solving coefficient update problems, including the factorization and solution of interior and exterior problems. The major novelties are the hierarchical algorithms of exterior problems.

The factorization of interior problems follows a bottom-up (postordered) traversal of the tree $\mathcal{T}$. If the node i is a leaf, we factorize the discretized PDE (2.1) in Ω_i and store the boundary map matrix $T^{(i)}$. If i has children, then the boundary map $T^{(i)}$ can be constructed from those at its children using (2.11). The construction of interior boundary maps has been developed in [15]. Since the process is the foundation of exterior problems and factorization update, we review this result in Algorithm 3.1, FACINT, using the notation in this paper. This algorithm can be understood as applying a sparse LU factorization method to a sparse matrix with special structures. Using the terminologies of the multifrontal method [11], (2.9) can be thought of as the frontal matrix at a non-leaf node i which is assembled using update matrices at child nodes c_1, c_2. At least for non-leaf nodes, the factorization of (2.9) has the same numerical stability as LU. The corresponding solution algorithm contains forward and backward substitutions, which are described in Algorithm 3.3. Notice that the factorization, factorization update, and solution algorithms are specialized for elliptic PDE problems and the methods rely heavily on the derivations in Section 2 due to the special discretization and domain decomposition setup. Thus, they do not work for general sparse matrices. In addition, no approximation is involved in our algorithms.

The construction of exterior boundary maps follows a top-down (reverse postordered) traversal of $\mathcal{T}$. The major difference from computing interior boundary maps is that the *data dependency is reversed*. For the node i with children c_1, c_2, we have $\Omega_{c_1}, \Omega_{c_2} \subset \Omega_i$ for the interior problems, but $\Omega_{-c_1}, \Omega_{-c_2} \supset \Omega_{-i}$ for the exterior ones. Based on (2.15), we construct $T^{(-c_1)}$ from $T^{(-i)}, T^{(c_2)}$ and construct $T^{(-c_2)}$ from $T^{(-i)}, T^{(c_1)}$. This process is described in Algorithm 3.2, FACEXT. Each new $T^{(-i)}$ corresponds to the Schur complement from eliminating the points outside Ω_i. The ordering of LU is changed repeatedly in Algorithm 3.2. Like in other sparse direct solvers, it becomes nontrivial to keep track of the numerical stability. For simplicity, we assume there is no stability issue in the algorithms.

For the coefficient update problem (1.4), recall that the coefficient update and the right-hand side are supported in the same subdomain Ω_p for some node p in $\mathcal{T}$. According to the solution process at the end of Section 2.2, the major steps include: refactorization in Ω_p, computing boundary sources on the boundary $\partial\Omega_p$, and extracting the solution inside and outside Ω_p. This is Algorithm 3.4, NEWUPD–SOLEXT.

In NEWUPD, the modified operator $\tilde{L}$ in Ω_p is factorized and the solution in Ω_p is computed using Algorithm 3.3. Let $\tilde{\mathcal{T}}$ be the subtree of $\mathcal{T}$ corresponding to p. The part of $\tilde{L}$ corresponding to $\tilde{\mathcal{T}}$ is re-factorized. Inside Ω_p, each subdomain is visited twice by a postordered traversal and a reverse postordered traversal of $\tilde{\mathcal{T}}$. SOLEXT extends the solution to the exterior subdomain Ω_{-p} by solving a boundary value problem. It has a top-down traversal of the new domain partitioning inside Ω_{-p} defined in Theorem 3.1. Note that the new domain partitioning is not stored explicitly. The while loop in SOLEXT deduces the new parent-child relation on the fly. At each step, we get the solution of a subdomain along the path from p to the root of $\mathcal{T}$, and the cost increases for high-level problems. As mentioned near the end of Section 2.2, the algorithm can be terminated in the middle once the desired part of the solution is computed.

In general, *one does not need to know which subdomain is going to be changed* in FACEXT, and its output can handle coefficient updates in any subdomain of the domain partitioning. If we have additional information about p, the cost and storage can be further reduced by only calculating the exterior factors related to p. As can be seen in Theorem 3.1 and SOLEXT, the related nodes correspond to the ancestors of p.

To illustrate the benefits of our method, we compare it with a standard way of updating the factorization in FACINT, which is to recompute all those factors that are changed as in standard sparse factorizations. It not only recomputes the factorization in $\tilde{\mathcal{T}}$, but also propagates the changes to all the ancestors in $\mathcal{T}$. The following set of nodes are visited in a postordered traversal.

$$\tilde{\mathcal{F}} = \{i \in \mathcal{T} | i \in \tilde{\mathcal{T}} \text{ or is an ancestor of some node of } \tilde{\mathcal{T}}\}.$$

We have implemented this type of factorization update and name the routine STDUPD to compare with our method. STDUPD changes the outermost loop of FACINT by replacing $\mathcal{T}$ with $\tilde{\mathcal{F}}$.

TABLE 3.1
Major properties of the hierarchical factorization and solution algorithms. Let Ω_p be the modified subdomain. The costs are estimated in Section 4 for two-dimensional PDEs, where n is the matrix size, and $n_l \ll n$ is the update size.

Name	Output	Tree traversal	Cost
FACINT	all interior factors	postorder of $\mathcal{T}$	$O(n^{3/2})$
FACEXT	all exterior factors	reverse postorder of $\mathcal{T}$	$O(n^{3/2})$
NEWUPD	solution in Ω_p	postorder and reverse postorder of the $\tilde{\mathcal{T}}$	$O(n_l^{3/2})$
SOLEXT	solution in Ω_{-p}	reverse postorder of other subtrees of $\mathcal{T}$	$O(n \log n)$
STDUPD	new interior factors	postorder of a larger subtree $\tilde{\mathcal{F}} \supset \tilde{\mathcal{T}}$	$O(n^{3/2})$

In summary, Table 3.1 lists the roles and properties of the major routines, and for convenience, the complexity estimates in Section 4 are listed as well. We suggest

Algorithm 3.1 Factorization of interior problems

```
 1: procedure FACINT($\mathcal{T}, L$)
 2:     for each $i \in \mathcal{T}$ following the postordered traversal do
 3:         if $i$ is a leaf then
 4:             Factorize the discretized $L$ in $\Omega_i$ by a sparse LU factorization
 5:             Construct $T^{(i)}$ in (2.3), the $j$th column of which is TSMV($i, e_j, 0$)
 6:         else
 7:             $(c_1, c_2) \leftarrow i$'s children
 8:             Factorize $M^{(c_1,c_2)}$ defined in (2.10)
 9:             Compute $T^{(i)}$ from $T^{(c_1)}$ and $T^{(c_2)}$ using (2.11)
10:         end if
11:     end for
12:     return $T^{(*)}$, factors of $M^{(*,*)}$, and factors of $L$ restricted in leaf subdomains
13: end procedure
```

Algorithm 3.2 Factorization of exterior problems

```
 1: procedure FACEXT($\mathcal{T}, T^{(*)}$)
 2:     for each $i \in \mathcal{T}$ following a reverse postordered traversal do
 3:         if $i$ is not a leaf then
 4:             $(c_1, c_2) \leftarrow i$'s children
 5:             Factorize $M^{(c_1,-i)} = \begin{pmatrix} T^{(c_1)}_{1,1} & I \\ I & T^{(-i)}_{1,1} \end{pmatrix}, M^{(c_2,-i)} = \begin{pmatrix} T^{(c_2)}_{2,2} & I \\ I & T^{(-i)}_{2,2} \end{pmatrix}$
 6:             Based on (2.15), compute $T^{(-c_1)}$ via
                $$\begin{pmatrix} T^{(c_2)}_{0,0} & \\ & T^{(-i)}_{1,1} \end{pmatrix} - \begin{pmatrix} T^{(c_2)}_{0,2} & \\ & T^{(-i)}_{1,2} \end{pmatrix} (M^{(c_2,-i)})^{-1} \begin{pmatrix} T^{(c_2)}_{2,0} & \\ & T^{(-i)}_{2,1} \end{pmatrix}$$
 7:             Compute $T^{(-c_2)}$ via
                $$\begin{pmatrix} T^{(c_1)}_{0,0} & \\ & T^{(-i)}_{2,2} \end{pmatrix} - \begin{pmatrix} T^{(c_1)}_{0,1} & \\ & T^{(-i)}_{2,1} \end{pmatrix} (M^{(c_1,-i)})^{-1} \begin{pmatrix} T^{(c_1)}_{1,0} & \\ & T^{(-i)}_{1,2} \end{pmatrix}$$
 8:         end if
 9:     end for
10:     return $T^{(*)}$ and factors of $M^{(*,*)}$
11: end procedure
```

the following calling sequence for solving coefficient update problems:

1. NEWUPD($\mathcal{T}, i_0, L, f, \dots$) for factorizing L and solving $Lu = f$, where i_0 is the root of $\mathcal{T}$;
2. FACEXT($\mathcal{T}, \dots$) for factorizing exterior problems;
3. NEWUPD($\mathcal{T}, p, \tilde{L}, (L - \tilde{L})u, \dots$) for the solution update $\tilde{u} - u$ in Ω_p and the exterior boundary source $g^{(-p)}$;
4. SOLEXT($\mathcal{T}, p, g^{(-p)}, \dots$) for the solution update $\tilde{u} - u$ in Ω_{-p}.

Note that the solution steps (1, 3, and 4) can be trivially extended for solving multiple right-hand sides. There are several qualitative arguments about the cost effectiveness of this family of algorithms. The factorization of exterior problems does

Algorithm 3.3 Forward and backward substitutions for the solution algorithms

```
1: procedure SOLF($\mathcal{T}, f, T^{(*)}, M^{(*,*)}$)    ▷ Compute $s^{(i)} = S^{(i)} f^{(i)}$ for $i \in \mathcal{T}$
2:     for each $i \in \mathcal{T}$ following the postordered traversal do
3:         if $i$ is a leaf then
4:             $s^{(i)} \leftarrow$TSMV($i, 0, f|_{\Omega_i}$)    ▷ Compute $S^{(i)} f|_{\Omega_i}$
5:         else
6:             $(c_1, c_2) \leftarrow i$'s children
7:             Based on (2.12), compute
                   $$s^{(i)} \leftarrow \begin{pmatrix} s_1^{(c_1)} \\ s_2^{(c_2)} \end{pmatrix} - \begin{pmatrix} T_{1,0}^{(c_1)} & \\ & T_{2,0}^{(c_2)} \end{pmatrix} (M^{(c_1,c_2)})^{-1} \begin{pmatrix} s_0^{(c_1)} \\ s_0^{(c_2)} \end{pmatrix}$$
8:         end if
9:     end for
10:    return $s^{(*)}$
11: end procedure

1: procedure SOLB($\mathcal{T}, f, s^{(*)}, g^{(i_0)}, T^{(*)}, M^{(*,*)}$)
                   ▷ Compute $g^{(i)}$ for $i \in \mathcal{T}$ and the true solution $u$, $i_0$ is the root of $\mathcal{T}$
2:     for each $i \in \mathcal{T}$ following a reverse postordered traversal do
3:         if $i$ is a leaf then
4:             Compute $u|_{\Omega_i}$ by solving (2.1) with $f^{(i)} = f|_{\Omega_i}$ and newly obtained $g^{(i)}$
5:         else
6:             $(c_1, c_2) \leftarrow i$'s children
7:             $g_1^{(c_1)} \leftarrow g_1^{(i)}, \quad g_2^{(c_2)} \leftarrow g_2^{(i)}$
8:             Solve the first two block rows of (2.9) as
                   $$\begin{pmatrix} g_0^{(c_1)} \\ g_0^{(c_2)} \end{pmatrix} \leftarrow -(M^{(c_1,c_2)})^{-1} \begin{pmatrix} s_0^{(c_1)} + T_{0,1}^{(c_1)} g_1^{(i)} \\ s_0^{(c_2)} + T_{0,2}^{(c_2)} g_2^{(i)} \end{pmatrix}$$
9:         end if
10:    end for
11:    return $u$
12: end procedure
```

not increase the order of factorization complexity, because the cost depends on the sizes of boundaries $\{\partial\Omega_i\}$ in the same way as existing factorization of interior problems. The cost of the re-factorization step is low because it only depends on the local problem size in Ω_p. The cost of solution is low if terminated early because Algorithm 3.4 visits smaller subdomains first. Similar to existing sparse direct solvers, Algorithm 3.1–3.4 have two levels of parallelism: parallel traversals of tree structures and parallel dense matrix operations. In addition, $T^{(-c_1)}$ and $T^{(-c_2)}$ in Algorithm 3.2 can be computed in parallel.

4. Algorithm complexity.

In this section, we estimate the complexity of the algorithms presented in Section 3. The major components of our method include a precomputation step that constructs interior and exterior boundary maps of the reference problem, a factorization update step that modifies the factors of an interior problem, and a solution update step to get the final solution.

Algorithm 3.4 Factorization and solution update with modified coefficients in Ω_p

```
1: procedure NEWUPD($\mathcal{T}, p, \tilde{L}, f, T^{(-p)}$)          ▷ Factorization and Solution in $\Omega_p$
2:     $\tilde{\mathcal{T}} \leftarrow$ subtree($p$)                     ▷ Subtree of $\mathcal{T}$ with root $p$
3:     FACINT($\tilde{\mathcal{T}}, \tilde{L}$) for $\tilde{T}^{(*)}, \tilde{M}^{(*,*)}$ in $\Omega_p$
4:     $s^{(*)} \leftarrow$ SOLF($\tilde{\mathcal{T}}, f, \tilde{T}^{(*)}, \tilde{M}^{(*,*)}$)   ▷ Forward sweep in $\tilde{\mathcal{T}}$ via Algorithm 3.3
5:     Based on (2.6), solve
           $$\begin{pmatrix} \tilde{T}^{(p)} & I \\ I & T^{(-p)} \end{pmatrix} \begin{pmatrix} g^{(p)} \\ g^{(-p)} \end{pmatrix} = \begin{pmatrix} -s^{(p)} \\ 0 \end{pmatrix}$$
6:     $u^{(p)} \leftarrow$ SOLB($\tilde{\mathcal{T}}, f, s^{(*)}, g^{(p)}, \tilde{T}^{(*)}, \tilde{M}^{(*,*)}$)   ▷ Backward sweep in $\tilde{\mathcal{T}}$
7:     return $u^{(p)}, g^{(-p)}$
8: end procedure

1: procedure SOLEXT($\mathcal{T}, p, g^{(-p)}, T^{(*)}, M^{(*,*)}$)         ▷ Solution in $\Omega_{-p}$
2:     $c_1 \leftarrow p$
3:     while $c_1$ is not the root do
4:         $c_2 \leftarrow c_1$'s sibling, $i \leftarrow c_1$'s parent
5:         $g_0^{(c_2)} \leftarrow g_0^{(-c_1)}, \quad g_1^{(-i)} \leftarrow g_1^{(-c_1)}$
6:         Based on the first two rows of (2.13) or (2.16), compute
               $$\begin{pmatrix} g_2^{(c_2)} \\ g_2^{(-i)} \end{pmatrix} \leftarrow -(M^{(c_2,-i)})^{-1} \begin{pmatrix} T_{2,0}^{(c_2)} g_0^{(-c_1)} \\ T_{2,1}^{(-i)} g_1^{(-c_1)} \end{pmatrix}$$
7:         $u^{(-p)}|_{\Omega_{c_2}} \leftarrow$ SOLB(subtree($c_2$), 0, 0, $g^{(c_2)}, T^{(*)}, M^{(*,*)}$)   ▷ Solution in $\Omega_{c_2}$
8:         $c_1 \leftarrow i$                                        ▷ Continue with $\Omega_{-i}$
9:     end while
10:    return $u^{(-p)}$
11: end procedure
```

For an $n \times n$ discretized linear system from a d-dimensional elliptic problem ($d = 2$ or 3), for convenience, the following assumption is used to estimate the complexity.

Assumption 4.1. Let $\mathcal{T}$ be a complete binary tree containing $\mathbf{l}$ levels. Each level-k subdomain of the domain partitioning $\{\Omega_i : i \in \mathcal{T}\}$ contains $O(n_k)$ interior unknowns and $O(m_k)$ boundary unknowns, where

$$n_k = 2^{-k} n, \quad m_k = n_k^{(d-1)/d}.$$

Furthermore, let $n_{\mathbf{l}} = O(1)$. Here, the constants in the big O notation are assumed to be uniformly bounded.

Remark 4.1. The condition on n_k and m_k requires that the domain partitioning is balanced. The fractional power in m_k comes from the dimension reduction from a d-dimensional domain to a $(d-1)$-dimensional boundary.

If boundary maps are stored as dense matrices, then according to (2.11) and (2.15), the precomputation of interior and exterior boundary maps has dense factorizations and multiplications at every node. The complexity $\mathcal{C}_{\text{pre}}$ and the storage $\mathcal{S}_{\text{pre}}$

are respectively

$$
\begin{aligned}
\mathcal{C}_{\text{pre}} &= \sum_{k=0}^{\mathbf{l}} 2^k O\left(m_k^3\right) = \begin{cases} O(n^{3/2}) & \text{in 2D,} \\ O(n^2) & \text{in 3D,} \end{cases} \\
\mathcal{S}_{\text{pre}} &= \sum_{k=0}^{\mathbf{l}} 2^k O\left(m_k^2\right) = \begin{cases} O(n \log n) & \text{in 2D,} \\ O(n^{4/3}) & \text{in 3D.} \end{cases}
\end{aligned} \tag{4.1}
$$

This is the cost of both FACINT in Algorithm 3.1 and FACEXT in Algorithm 3.2. The results are in the same orders as those in the direct factorization of sparse matrices with nested dissection reordering.

Consider modifying the problem in some level-l subdomain Ω_p containing $O(n_l)$ interior unknowns. The subtree corresponding to Ω_p has $(\mathbf{l}-l)$ levels. The complexity $\mathcal{C}_{\text{upd}}$ and storage $\mathcal{S}_{\text{upd}}$ of local factorization update are respectively

$$
\begin{aligned}
\mathcal{C}_{\text{upd}} &= \sum_{k=0}^{\mathbf{l}-l} 2^k O\left(m_{k+l}^3\right) = \begin{cases} O(n_l^{3/2}) & \text{in 2D,} \\ O(n_l^2) & \text{in 3D,} \end{cases} \\
\mathcal{S}_{\text{upd}} &= \sum_{k=0}^{\mathbf{l}-l} 2^k O\left(m_{k+l}^2\right) = \begin{cases} O(n_l \log n_l) & \text{in 2D,} \\ O(n_l^{4/3}) & \text{in 3D.} \end{cases}
\end{aligned} \tag{4.2}
$$

Observe that $\mathcal{C}_{\text{upd}}$ and $\mathcal{S}_{\text{upd}}$ only depend on the number of interior unknowns in Ω_p. This is the cost of the factorization update, which is the call of FACINT at Line 3 of Algorithm 3.4.

In comparison, we consider the naive factorization update method which changes the factors following the original data dependencies in $\mathcal{T}$. In addition to the re-factorization in Ω_p that has complexity $\mathcal{C}_{\text{upd}}$ in (4.2), the naive method has an additional step which updates every ancestor of p. This additional step costs

$$
\begin{aligned}
\mathcal{C}_{\text{anc}} &= \sum_{k=0}^{l-1} O\left(m_k^3\right) = \begin{cases} O(n^{3/2}) & \text{in 2D,} \\ O(n^2) & \text{in 3D,} \end{cases} \\
\mathcal{S}_{\text{anc}} &= \sum_{k=0}^{l-1} O\left(m_k^2\right) = \begin{cases} O(n) & \text{in 2D,} \\ O(n^{4/3}) & \text{in 3D.} \end{cases}
\end{aligned} \tag{4.3}
$$

This additional cost, on the contrary, is primarily determined by n because the ancestors of p have larger and larger matrix sizes. The factorization update cost is reduced from $\mathcal{C}_{\text{anc}} + \mathcal{C}_{\text{upd}}$ in STDUPD to $\mathcal{C}_{\text{upd}}$ in the proposed method. *If $n_l \ll n$, then the new method avoided the dominant cost (4.3) that is comparable to the cost (4.1) for re-factorizing the entire problem.*

The solution update in Algorithm 3.4 has the solution in Ω_p and Ω_{-p}, and the computational cost is proportional to the memory access. The solution complexity is $\mathcal{S}_{\text{upd}}$ in Ω_p, and is $\mathcal{S}_{\text{pre}}$ in Ω_{-p}. This is the cost of Algorithm 3.4, excluding the factorization update step. If the exterior solution is terminated early, then the total cost can be as low as $\mathcal{S}_{\text{upd}}$.

The following theorem summarizes the complexity of the proposed algorithms.

Theorem 4.1. *Let the domain partitioning satisfy Assumption 4.1. The cost of precomputation in Algorithm 3.1 (*FACINT*) and Algorithm 3.2 (*FACEXT*) is governed by the matrix size via (4.1). For the proposed method, the cost of factorization update is (4.2), which only depends on the size of the updated subdomain.*

FACINT and FACEXT have the same order of complexity as in (4.1). To get an idea of when the proposed factorization update algorithm has advantages over STDUPD, we compare the constant factors in the complexities of FACINT and FACEXT. We start by comparing the cost of (2.11) in FACINT and that of (2.15) in FACEXT.

LEMMA 4.2. *Let $A_1, C_1^T \in \mathbb{C}^{r_1 s\times s}$, $B_1, B_2 \in \mathbb{C}^{s\times s}$, and $A_2, C_2^T \in \mathbb{C}^{r_2 s\times s}$. The following matrix can be computed in $2[(r_1+r_2)^2+r_1r_2+(r_1+r_2)+\frac{4}{3}]s^3$ floating-point operations (plus some lower-order terms):*

$$U = \begin{pmatrix} A_1 & \\ & A_2 \end{pmatrix} \begin{pmatrix} B_1 & I \\ I & B_2 \end{pmatrix}^{-1} \begin{pmatrix} C_1 & \\ & C_2 \end{pmatrix}.$$

Proof. Note

$$\begin{pmatrix} B_1 & I \\ I & B_2 \end{pmatrix} = \begin{pmatrix} I & B_1 \\ & I \end{pmatrix} \begin{pmatrix} I - B_1B_2 & \\ & I \end{pmatrix} \begin{pmatrix} & I \\ I & B_2 \end{pmatrix}.$$

The cost of the multiplication B_1B_2 is approximately $2s^3$, and the LU factorization of $I - B_1B_2$ costs approximately $\frac{2}{3}s^3$. (Some lower-order terms are dropped in the estimates.) Also,

$$\begin{pmatrix} A_1 & \\ & A_2 \end{pmatrix} \begin{pmatrix} & I \\ I & B_2 \end{pmatrix}^{-1} = \begin{pmatrix} -A_1B_2 & A_1 \\ A_2 & \end{pmatrix},$$

$$\begin{pmatrix} I & B_1 \\ & I \end{pmatrix}^{-1} \begin{pmatrix} C_1 & \\ & C_2 \end{pmatrix} = \begin{pmatrix} C_1 & -B_1C_2 \\ & C_2 \end{pmatrix}.$$

The multiplications A_1B_2 and B_1C_2 take approximately $2(r_1+r_2)s^3$ flops. Then

$$U = \begin{pmatrix} -A_1B_2 & A_1 \\ A_2 & \end{pmatrix} \begin{pmatrix} (I - B_1B_2)^{-1} & \\ & I \end{pmatrix} \begin{pmatrix} C_1 & -B_1C_2 \\ & C_2 \end{pmatrix},$$

where the LU solution with $(r_1+r_2)s$ right-hand sides takes approximately $2(r_1 + r_2)s^3$ operations, and the five matrix multiplications afterwards take approximately $2((r_1+r_2)^2+r_1r_2)s^3$ operations. Summing up the costs of all the steps gives the final answer. ∎

The formula of U in Lemma 4.2 clearly gives the shared pattern of (2.11) and (2.15). Recall the definition of $\Gamma_0, \Gamma_1, \Gamma_2$ in (2.8). For computing (2.11), s is the size of Γ_0, and r_1 (r_2) is the ratio between the size of Γ_1 (Γ_2) and s. For computing (2.15), s is the size of Γ_2, r_1 is the ratio between the size of Γ_0 and s, and r_2 is the ratio between the size of Γ_1 and s. The precise cost depends on the shapes of subdomains, and we give some 2D examples as follows.

Take an example of merging two square subdomains into a rectangle. Assume that each side length has m sampling points. Γ_1 (Γ_2) is three times as long as Γ_0. Let $r_1 = r_2 = 3, s = m$ in Lemma 4.2, and we get the cost of computing (2.11) as $2(52 + \frac{1}{3})m^3$. Let $r_1 = \frac{1}{3}, r_2 = 1, s = 3m$ in Lemma 4.2, and then the cost of computing (2.15) is $2 \cdot 129m^3$. Since (2.15) is used twice, FACEXT is approximately 4.93 times as expensive as FACINT for this case.

Take another example of partitioning a square subdomain into two rectangles that are equal in size. Assume that each side length of the square has $2m$ sampling points. Γ_1 (Γ_2) is twice as long as Γ_0. Let $r_1 = r_2 = 2, s = 2m$ in Lemma 4.2, and then the cost of computing (2.11) is $32(12 + \frac{2}{3})m^3$. Let $r_1 = \frac{1}{2}, r_2 = 1, s = 4m$ in Lemma 4.2, and then the cost of computing (2.15) is $32(22+\frac{1}{3})m^3$. Since (2.15) is used twice, FACEXT is approximately 3.53 times as expensive as FACINT for this case.

The two examples are essential for generalizing the comparison to a recursive partitioning of a square domain. The second example is applied to partition each square into two rectangles, and the first example is useful at the next level during the partitioning of each rectangle into two squares. Due to the recursive structure, we only need to compare the constant factors in two adjacent levels, and the same ratio holds for any even number of levels. Consider partitioning a square with $2m$ points on each side length into four squares with m points on each side length, by combining the results of the two examples, the ratio between the cost of FACEXT and that of FACINT is

$$2\frac{32(22+\frac{1}{3})+4\cdot 129}{32(12+\frac{2}{3})+4(52+\frac{1}{3})}\approx 4.00,$$

where the factor of 2 in the front comes from using (2.15) twice, and the numbers in the first example are doubled because there are two rectangles involved. Since FACEXT is done only once to the reference problem, this approach becomes suitable for multiple updated problems. In this case, comparing with a naive factorization update like STDUPD, the new method has advantages with more than four local updates for sufficiently large problem sizes. When there are many updates, the benefit of the factorization update is significant.

The cost of FACEXT can be reduced by excluding some subtrees of $\mathcal{T}$, which requires some knowledge on where the problem is never updated. As mentioned near the end of Section 3, FACEXT has an additional parallelism comparing with FACINT. Line 6–7 of Algorithm 3.2 can be computed in parallel, which could ideally reduce the run time of FACEXT by two.

5. Numerical tests.

In this section, we check how the cost of our direct method scales with respect to the size of the computational domain and the support of the coefficient update. The method is able to solve general elliptic problems with coefficient updates. A particular problem of interest is the variable-coefficient Helmholtz equation

$$-\Delta u(x)-k^2(x)u(x)=f(x),$$

where $k(x)$ is the wavenumber that may be updated in various applications.

The domain of interest is chosen as $D=(0,1)\times(0,1)$. We discretize the Helmholtz equation by a continuous Galerkin method with fourth-order nodal Lagrange bases in a regular triangular mesh. We refer to [19] for the method and code for determining nodal points and computing partial derivatives. The performance of the direct method is mostly determined by the matrix size and sparsity pattern. The matrix size equals the number of nodal points in the domain, and high-order schemes usually lead to more nonzeros. The reference wavenumber function is plotted in Figure 5.1, but similar performance can be reproduced for other choices of wavenumber functions. The performance is not sensitive to the choice of boundary conditions either, and we use the impedance boundary condition $\partial_n+\mathrm{i}ku=0$ on ∂D, where the wavenumber is location independent on the boundary. For the coefficient updates, the wavenumber is reduced by 1/2 in different subdomains.

The algorithms are implemented in MATLAB (available at https://github.com/xiaoliurice/FACUPD) and are run in serial on a Linux workstation with 3.5GHz CPU and 64GB RAM. We check the complexity of the proposed method (Algorithms 3.1–3.4), and compare with the standard factorization update approach (STDUPD)

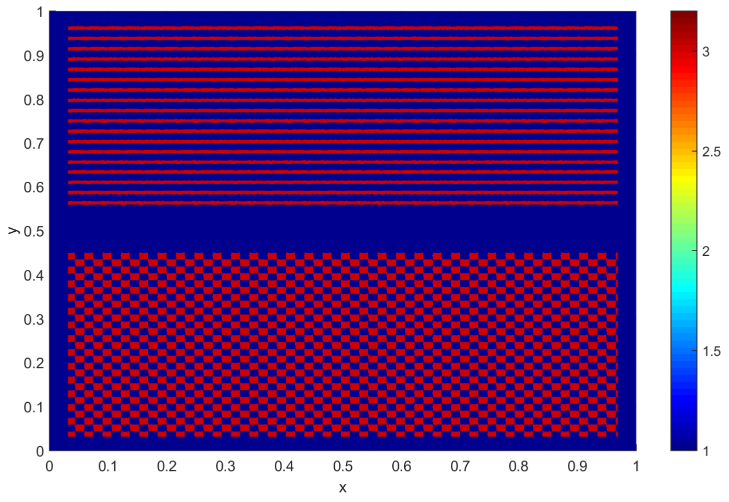


FIG. 5.1. *Wavenumber function of the Helmholtz equation. The wavenumber is normalized by its smallest value.*

described in Section 3.2. We report the runtime, the number of floating-point operations (flops), and the storage in terms of the number of nonzeros in the factors. For counting the flops, we sum up the number of addition, subtraction, multiplication, and division operations of all the actual linear algebra operations.

First, we check the dependence of the factorization and solution costs on the matrix size n. We increase n by refining the mesh and doubling the wavenumber simultaneously. This choice fixes the sampling rate of the discrete Helmholtz problem. The test results are listed in Table 5.1. As estimated by (4.1) and visualized in Figure 5.2(a), the factorizations of the interior problems (Algorithm 3.1) and the exterior problems (Algorithm 3.2) have the total complexity $O(n^{3/2})$.

Then for the same setup as in Table 5.1, Table 5.2 lists the costs of solving coefficient update problems when the number of points in the modified subdomain is kept fixed as $n_l = 160^2$. Similar results are obtained for three types of locations: near a corner, near the center of an edge, and near the center of D. Algorithm 3.4 (NEWUPD) contains the re-factorization and solution in the modified subdomain, and the cost (mainly for the factorization) does not depend on the matrix size n. In comparison, the factorization update cost of STDUPD is $O(n^{3/2})$, The test results are consistent with the complexity estimates. The significant advantage of the new factorization update NEWUPD over the standard one STDUPD is apparent from Figure 5.2(b). For the matrix size $n = 2561^2$, the cost of NEWUPD is about 78 times lower than STDUPD.

The solution update costs for both methods are $O(n \log n)$. The new method uses Algorithm 3.4 (SOLEXT) for the solution in the exterior subdomain. The standard method solves (1.3). Table 5.2(a) shows that SOLEXT in the new method needs only about half of the cost of the standard solution update. SOLEXT is faster because it does not need to visit every subdomain twice, although the standard update method can solve (1.3) directly and does not need the solution of the reference problem (1.1). For both methods, the solution updates have reasonable costs.

For the largest computational domain with n fixed, we also vary the size n_l of the modified subdomain. The results are listed in Table 5.3 and plotted in Figure 5.3. The cost of NEWUPD is dominated by the direct factorization in the modified subdomain. The dependence on n_l as illustrated in Figure 5.3 is a little better than the estimate in (4.2). The cost of SOLEXT does not increase because n is fixed. As expected, if n_l gets closer to n, the cost of NEWUPD becomes closer to that of STDUPD. (Note that the benefit of our method is when there are multiple sets of local updates.)

TABLE 5.1
Test of direct factorization and solution costs for the reference problem (1.1).

(a) *Problem setup*

Matrix size	321^2	641^2	1281^2	2561^2
#nonzeros	2,437,184	9,748,736	38,994,944	155,979,776

(b) *Factorization of interior problems*

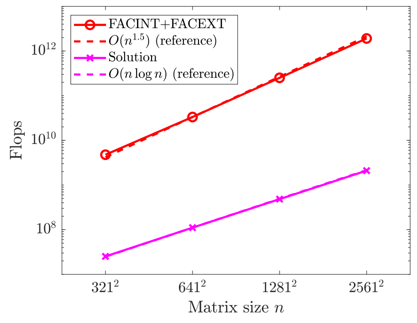
Time	1.77s	7.70s	33.10s	156.30s
Flops	3.11×10^9	1.58×10^{10}	8.93×10^{10}	5.62×10^{11}
Factor storage	9.03×10^6	4.65×10^7	2.31×10^8	1.11×10^9

(c) *Factorization of exterior problems*

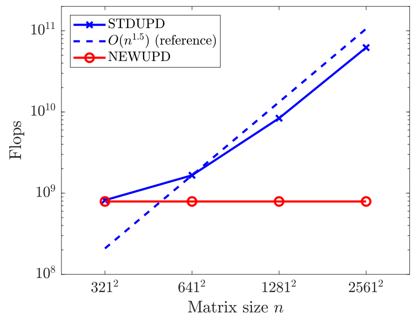
Time	0.52s	3.75s	25.02s	170.29s
Flops	1.66×10^9	1.75×10^{10}	1.62×10^{11}	1.35×10^{12}
Factor storage	3.87×10^6	2.56×10^7	1.46×10^8	7.66×10^8

(d) *Solution of the reference problem*

Time	0.08s	0.32s	1.39s	7.08s
Flops	2.52×10^7	1.11×10^8	4.83×10^8	2.10×10^9



(a) *Flops in Table 5.1*



(b) *Flops of* STDUPD *and* NEWUPD *in Table 5.2(a)*

FIG. 5.2. *Scaling plots for fixed update size.*

These test results demonstrate that the proposed algorithms are capable of solving the challenging cases where the coefficient updates have large magnitude and support. The algorithms can accommodate large amounts of modifications fairly easily. In addition, the solutions of the new update method are as accurate as results from more expensive standard update methods. We have not encountered a test case where the accuracy has a noticeable loss. We anticipate that accuracy losses may occur when some interior or exterior subproblems become nearly singular. We plan to study the

Table 5.2
Solution update for modifying $k(x)$ at 160^2 points. $\mathbf{l}$ is the total number of levels in the domain partitioning (see Assumption 4.1). l is the level of the modified subdomain. The accuracy of the updated solution u is measured as $\|u-v\|_\infty/\|v\|_\infty$, where v is computed via the standard factorization update method STDUPD *described in Section 3.*

Matrix size	321^2	641^2	1281^2	2561^2
$(\mathbf{l}, l)$	$(6,2)$	$(8,4)$	$(10,6)$	$(12,8)$

(a) *Updates near the corner $x_1 = 0, x_2 = 0$*

Subdomain	$(0,\frac{1}{2})^2$	$(0,\frac{1}{4})^2$	$(0,\frac{1}{8})^2$	$(0,\frac{1}{16})^2$
NEWUPD time	0.45s	0.44s	0.44s	0.46s
NEWUPD flops	7.90×10^8	7.90×10^8	7.90×10^8	7.90×10^8
SOLEXT time	0.03s	0.14s	0.61s	2.62s
SOLEXT flops	9.34×10^6	5.31×10^7	2.49×10^8	1.12×10^9
Accuracy	8.38×10^{-16}	1.56×10^{-16}	1.38×10^{-16}	9.08×10^{-17}
STDUPD time	0.43s	0.51s	1.10s	5.70s
STDUPD flops	8.19×10^8	1.66×10^9	8.38×10^9	6.21×10^{10}
Solution time	0.07s	0.28s	1.16s	7.30s
Solution flops	2.52×10^7	1.11×10^8	4.83×10^8	2.10×10^9

(b) *Updates near the center of an edge $x_1 = 0, x_2 = \frac{1}{2}$*

Subdomain	$(0,\frac{1}{2})\times(\frac{1}{2},1)$	$(0,\frac{1}{4})\times(\frac{1}{2},\frac{3}{4})$	$(0,\frac{1}{8})\times(\frac{1}{2},\frac{5}{8})$	$(0,\frac{1}{16})\times(\frac{1}{2},\frac{9}{16})$
NEWUPD time	0.44s	0.48s	0.50s	0.58s
NEWUPD flops	7.91×10^8	9.43×10^8	9.43×10^8	9.43×10^8
SOLEXT time	0.03s	0.14s	0.71s	2.95s
SOLEXT flops	9.32×10^6	5.28×10^7	2.49×10^8	1.13×10^9
Accuracy	7.60×10^{-16}	4.69×10^{-16}	4.02×10^{-16}	4.97×10^{-16}
STDUPD time	0.43s	0.59s	1.16s	5.83s
STDUPD flops	8.20×10^8	1.73×10^9	8.71×10^9	6.46×10^{10}

(c) *Updates near the center $x_1 = \frac{1}{2}, x_2 = \frac{1}{2}$*

Subdomain	$(\frac{1}{2},1)^2$	$(\frac{1}{2},\frac{3}{4})^2$	$(\frac{1}{2},\frac{5}{8})^2$	$(\frac{1}{2},\frac{9}{16})^2$
NEWUPD time	0.44s	0.54s	0.56s	0.63s
NEWUPD flops	7.95×10^8	1.19×10^9	1.19×10^9	1.19×10^9
SOLEXT time	0.03s	0.14s	0.71s	2.89s
SOLEXT flops	9.25×10^6	5.20×10^7	2.47×10^8	1.11×10^9
Accuracy	9.34×10^{-16}	1.47×10^{-15}	1.46×10^{-15}	2.07×10^{-15}
STDUPD time	0.43s	0.59s	1.31s	6.72s
STDUPD flops	8.25×10^8	1.90×10^9	9.95×10^9	7.43×10^{10}

accuracy in detail in future work.

We would also like to mention that, the large magnitude and support of the updates make the modified problems no longer close to the reference problem. This

TABLE 5.3
Test for a fixed matrix size (2561^2) *and increasing modified subdomain sizes. l is the level of the modified subdomain.*

Update size	160^2	320^2	640^2	1280^2
l	8	6	4	2
Subdomain	$(\frac{1}{2}, \frac{9}{16})^2$	$(\frac{1}{2}, \frac{5}{8})^2$	$(\frac{1}{2}, \frac{3}{4})^2$	$(\frac{1}{2}, 1)^2$
NEWUPD time	0.65s	2.71s	12.21s	44.56s
NEWUPD flops	1.19×10^9	7.06×10^9	4.66×10^{10}	1.47×10^{11}
SOLEXT time	4.17s	4.73s	4.16s	1.95s
SOLEXT flops	1.12×10^9	1.10×10^9	1.03×10^9	8.09×10^8
Accuracy	2.07×10^{-15}	2.22×10^{-15}	5.27×10^{-15}	2.69×10^{-15}
STDUPD time	6.82s	8.18s	14.57s	40.11s
STDUPD flops	7.43×10^{10}	7.72×10^{10}	9.26×10^{10}	1.65×10^{11}

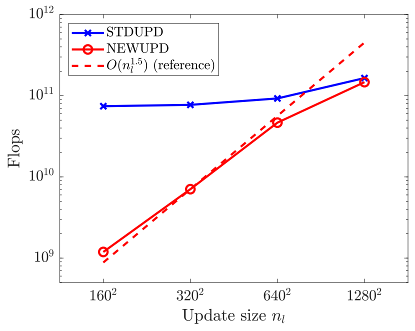


FIG. 5.3. *Scaling plot for Table 5.3.*

situation is handled efficiently with our algorithms, but causes troubles to methods such as iterative solvers using the factorization of the reference problem as a preconditioner. To verify this, we reuse the factorization of the reference problem as a preconditioner. For the problems considered in Table 5.2, Table 5.4 shows the results of the preconditioned iterative method. Limited by the large runtime, we can only check the first two cases in Table 5.3 using the setup in Table 5.4, which need 54 and 1048 iterations that take 746.98s and 9261.66s, respectively. The computation time is much longer than in our factorization update algorithm due to the large number of iterations. This is because that the reference problem and the modified problem are not close to each other in the tests. In addition, our direct update algorithm can handle large amounts of modifications fairly easily.

6. Conclusions and future work.

We developed a new framework for updating the factorization of discretized elliptic operators. A major significance is the hierarchical construction of exterior boundary maps. For each modified operator, we only need to update the factorization for locations where the coefficients are updated, and the locations of coefficient update are allowed to change to different subdomains. Tree-based algorithms were given for solving the interior and exterior problems. The complexity estimates show that the cost of factorization update only depends on the size of the modified subdomain. Numerical tests show that the new method is consid-

TABLE 5.4

Preconditioned iterative solution of the problems in Table 5.2. The preconditioner is the factorization of the reference problem. GMRES restarts every 60 iterations and stops when the relative residual error is below 10^{-4}.

Matrix size	321^2	641^2	1281^2	2561^2

(a) *Updates near the corner* $x_1 = 0, x_2 = 0$

Subdomain	$(0, \frac{1}{2})^2$	$(0, \frac{1}{4})^2$	$(0, \frac{1}{8})^2$	$(0, \frac{1}{16})^2$
#iterations	51	47	43	43
Iteration time	9.05s	29.25s	101.11s	541.53s

(b) *Updates near the center of an edge* $x_1 = 0, x_2 = \frac{1}{2}$

Subdomain	$(0, \frac{1}{2}) \times (\frac{1}{2}, 1)$	$(0, \frac{1}{4}) \times (\frac{1}{2}, \frac{3}{4})$	$(0, \frac{1}{8}) \times (\frac{1}{2}, \frac{5}{8})$	$(0, \frac{1}{16}) \times (\frac{1}{2}, \frac{9}{16})$
#iterations	51	158	154	133
Iteration time	8.89s	80.84s	303.72s	1225.82s

(c) *Updates near the center* $x_1 = \frac{1}{2}, x_2 = \frac{1}{2}$

Subdomain	$(\frac{1}{2}, 1)^2$	$(\frac{1}{2}, \frac{3}{4})^2$	$(\frac{1}{2}, \frac{5}{8})^2$	$(\frac{1}{2}, \frac{9}{16})^2$
#iterations	51	56	55	54
Iteration time	8.93s	37.91s	148.49s	645.18s

erably less expensive than the standard factorization update method. The solution update algorithms produce high accuracies as in standard factorization update algorithms. The method is suitable for solving the challenging cases where there are multiple updates with large magnitude.

The current method has expensive factorization steps as with standard sparse direct solvers. It is feasible to introduce rank-structured matrices so that the precomputation step can have nearly linear complexity and storage for elliptic problems. Rank-structured methods can accelerate both the factorization of exterior problems and the factorization update. Recent work on interconnected hierarchical structures [25] may be used for the acceleration of our algorithms. It is also interesting to study whether this fast factorization update approach can be extended to general sparse matrices. There seems to be some resemblance between the factorization of exterior problems and the method in selected inversion [23]. Technical challenges such as changes in the symbolic factorization need to be studied in depth in order to get a general algebraic method.

Acknowledgement. We would like to thank Yuanzhe Xi and Christopher Wong for some discussions and comments. We are also very grateful for the valuable suggestions from the editor and the three anonymous referees.

REFERENCES

[1] P. Amestoy, I. Duff, J. L'Excellent, Y. Robert, F. Rouet and B. Uçar, *On computing inverse entries of a sparse matrix in an out-of-core environment*, SIAM J. Sci. Comput., 34 (2012), pp. 1975–1999.

[2] J. M. Bennett, *Triangular factors of modified matrices*, Numer. Math., 7 (1965), pp. 217–221.

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

[4] S. M. Chan, and V. Brandwajn, *Partial matrix refactorization*, IEEE trans. Power Systems, PWRS-1 (1986), pp. 193–200.
[5] T. F. Chan, and D. Goovaerts, *Schur complement domain decomposition algorithms for spectral methods*, Appl. Numer. Math., 6 (1989), pp. 53–64.
[6] Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam, *Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate*, ACM Trans. Math. Software, 35 (2008), p. 22.
[7] ,B. Cockburn, J. Gopalakrishnan, and R. Lazarov *Unified hybridization of discontinuous Galerkin, mixed, and continuous Galerkin methods for second order elliptic problems*, SIAM J. Numer. Anal., 47 (2007), pp. 1319–1365.
[8] T. A. Davis, and W. W. Hager, *Modifying a sparse Cholesky factorization*, SIAM J. Matrix Anal. Appl., 20 (1999), pp. 606–627.
[9] J. Djokić, *Efficient update of hierarchical matrices in the case of adaptive discretization schemes*, Ph.D. thesis, Leipzig University, Leipzig, Germany, 2006.
[10] J. Douglas, and C. Huang, *An accelerated domain decomposition procedure based on Robin transmission conditions*, BIT Numer. Math, 37 (1997), pp. 678–686.
[11] I. S. Duff, and J. K. Reid, *The multifrontal solution of indefinite sparse symmetric linear equations*, ACM Trans. Math. Software, 9 (1983), pp. 302–325.
[12] A. George, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345–363.
[13] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright, *Maintaining LU factors of a general sparse matrix*, Linear Algebra Appl., 88 (1987), pp. 239–270.
[14] A. Gillman and P. G. Martinsson, *A direct solver with O(n) complexity for variable coefficient elliptic PDEs discretized via a high-order composite spectral collocation method*, SIAM J. Sci. Comput., 36 (2014), pp. A2023–A2046.
[15] A. Gillman, A. H. Barnett, and P. G. Martinsson, *A spectrally accurate direct solution technique for frequency-domain scattering problems with variable media*, BIT Numer. Math., 55 (2015), pp. 141–170.
[16] W. Hackbusch, L. Grasedyck, and S. Börm, *An introduction to hierarchical matrices*, Math. Bohem., 127 (2002), pp. 229–241.
[17] W. Hackbusch, and S. Börm, *Data-sparse approximation by adaptive $\mathcal{H}^2$-matrics*, Computing, 69 (2002), pp. 1–35.
[18] W. Hackbusch, B. N. Khoromskij, and R. Kriemann *Direct Schur complement method by domain decomposition based on $\mathcal{H}$-matrix approximation*, Comput. Vis. Sci., 8 (2005), pp. 179–188.
[19] J. S. Hesthaven and T. Warburton, *Nodal discontinuous Galerkin methods: algorithms, analysis, and applications*, Springer Science & Business Media, 2007.
[20] M. Jakobsen and B. Ursin, *Full waveform inversion in the frequency domain using direct iterative T-matrix methods*, J. Geophys. Eng., 12 (2015), p. 400.
[21] R. Kittappa, and R. E. Kleinman, *Acoustic scattering by penetrable homogeneous objects*, J. Math. Phys., 16 (1975), pp. 421–432.
[22] R. Kress, and G. F. Roach, *Transmission problems for the Helmholtz equation*, J. Math. Phys., 19 (1977), pp. 1433–1437.
[23] L. Lin, C. Yang, J. C. Meza, J. Lu, and L. Ying, *SelInv–An algorithm for selected inversion of a sparse symmetric matrix*, ACM Trans. Math. Software, 37 (2011), p. 40.
[24] X. Liu, J. Xia, and M. V. de Hoop, *Parallel randomized and matrix-free direct solvers for large structured dense linear systems*, SIAM J. Sci. Comput., 38 (2016), pp. S508–S538.
[25] X. Liu, J. Xia, and M. V. de Hoop, *A fast direct elliptic solver via interconnected hierarchical structures*, Purdue CCAM Report CCAM-2019-2.
[26] P. G. Martinsson, *A direct solver for variable coefficient elliptic PDEs discretized via a composite spectral collocation method*, J. Comput. Phys., 242 (2013), pp. 460–479.
[27] V. Minden, A. Damle, K. L. Ho, and L. Ying, *A technique for updating hierarchical skeletonization-based factorizations of integral operators*, Multiscale Model. Simul., 14 (2016), pp. 42–64.
[28] MUMPS, *A multifrontal massively parallel sparse direct solver*, http://mumps.enseeiht.fr.
[29] PARDISO, *Parallel sparse direct solver PARDISO*, http://www.pardiso-project.org.
[30] M. Pedneault, C. Turc, and Y. Boubendir, *Schur complement domain decomposition methods for the solution of multiple scattering problems*, IMA J. Appl. Math., 82 (2017), pp. 1104–1134.
[31] F. H. Rouet, *Memory and performance issues in parallel multifrontal factorizations and triangular solutions with sparse right-hand sides*, Ph.D. thesis, University of Toulouse, Toulouse, France, 2012.

[32] O. Schenk, and K. Gärtner *Solving unsymmetric sparse systems of linear equations with PARDISO*, Future Gener. Comput. Syst., 20 (2004), pp. 475–487.
[33] B. Willemsen, A. Malcolm, and W. Lewis, *A numerically exact local solver applied to salt boundary inversion in seismic full-waveform inversion.*, Geophys. J. Int, 204 (2016), pp. 1703–1720.
[34] J. Xia, *Randomized sparse direct solvers*, SIAM J. Matrix Anal. Appl., 34 (2013), pp. 197–227.
[35] J. Xia, *Efficient structured multifrontal factorization for general large sparse matrices*, SIAM J. Sci. Comput., 35 (2013), pp. A832-A860.
[36] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, *Superfast multifrontal method for large structured linear systems of equations*, SIAM J. Matrix Anal. Appl., 31 (2009), pp. 1382–1411.
[37] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, *Fast algorithms for hierarchically semiseparable matrices*, Numer. Linear Algebra Appl., 17 (2010), pp. 953–976.
[38] Z. Xin, J. Xia, M. V. de Hoop, S. Cauley, and V. Balakrishnan, *A distributed-memory randomized structured multifrontal method for sparse direct solutions*, SIAM J. Sci. Comput., 39 (2017), pp. C292–C318.
[39] E. L. Yip, *A note on the stability of solving a rank-p modification of a linear system by the Sherman-Morrison-Woodbury formula*, SIAM J. Sci. and Stat. Comput., 7 (1984), pp. 507–513.
[40] Y. Zhang and A. Gillman, *A fast direct solver for boundary value problems on locally perturbed geometries*, J. Comput. Phys., 356 (2018), pp. 356–371.