Journal of Computational Physics 231 (2012) 1314–1338

Contents lists available at SciVerse ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

A fast direct solver for elliptic problems on general meshes in 2D

Phillip G. Schmitz [a,*], Lexing Ying [b]

[a] Department of Mathematics, The University of Texas at Austin, 1 University Station, C1200, Austin, Texas 78712, United States
[b] Department of Mathematics and ICES, The University of Texas at Austin, 1 University Station, C1200, Austin, Texas 78712, United States

ARTICLE INFO

Article history:
Received 7 December 2010
Received in revised form 26 September 2011
Accepted 11 October 2011
Available online 19 October 2011

Keywords:
Elliptic equations
Fast algorithms
Multifrontal methods
Hierarchical matrices
Sparse matrix

ABSTRACT

We present a fast direct algorithm for solutions to linear systems arising from 2D elliptic equations. We follow the approach in Xia et al. (2009) on combining the multifrontal method with hierarchical matrices. We present a variant of that approach with additional hierarchical structure, extend it to quasi-uniform meshes, and detail an adaptive decomposition procedure for general meshes. Linear time complexity is shown for a quasi-regular grid and demonstrated via numerical results for the adaptive algorithm.

1. Introduction

In this paper we will consider the solution of an elliptic problem such as:

$$-\mathrm{div}(a(\mathbf{x})\nabla u(\mathbf{x})) + V(\mathbf{x})u(\mathbf{x}) = f(\mathbf{x}) \quad \text{on } \Omega, \quad u = 0 \quad \text{on } \partial\Omega, \tag{1}$$

in a two dimensional domain Ω where $a(\mathbf{x}) > 0$ and $V(\mathbf{x}) > 0$. There are two main classes of solvers for sparse linear systems: direct [1] and iterative [2] methods. We will only be concerned with direct methods in this paper.

Clearly the naïve inversion of the sparse matrix should be avoided and a sparse Cholesky decomposition should be used instead. However, the efficiency of the sparse Cholesky decomposition depends on choosing a reordering to reduce fill-in of non-zeros in the factors. Various graph-theoretic approaches such as the (approximate) minimum degree algorithm [3] or nested dissection [4] can be used to determine a good reordering, but finding the optimal ordering in general is difficult.

The most efficient direct method for solving this problem is the multifrontal method with nested dissection [5,6,1,7] (referred to as multifrontal method in short in the rest of this paper). The central idea of this method is to partition the domain using a nested hierarchical structure and generate the LU (or LDL^t) factorization from the bottom up to minimize the fill-ins. The computational cost of the multifrontal method scales like $\mathcal{O}(N^{1.5})$ in two dimensions where N is the number of degrees of freedom. The multifrontal method is often formulated in a block factorization form in order to take full advantage of the existing dense linear algebra routines (BLAS3). Though quite efficient for many applications, it might still be quite costly when N is very large.

Recently Xia et al. have worked on improving the multifrontal method with nested dissection to achieve linear complexity, $\mathcal{O}(N)$ in [8]. The main observation is that the fill-in blocks of the LDL^t factorization are highly compressible using the hierarchical semiseparable matrix [9] or H-matrix [10] frameworks. By representing and manipulating these blocks efficiently within these frameworks [11], one obtains linear or almost linear complexity algorithms for the solution of the discrete

* Corresponding author.
E-mail addresses: pschmitz@math.utexas.edu (P.G. Schmitz), lexing@math.utexas.edu (L. Ying).

0021-9991/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2011.10.013

system. In [8], this program is carried out in the setting of regular Cartesian grids. A similar approach is proposed in [12] where a spiral elimination order replaces the multifrontal nested dissection method. Recently, a substantial amount of research has also been devoted to developing direct solvers for linear systems from integral equations. In [13] an essentially linear complexity algorithm is presented for the 2D non-oscillatory integral equations, while an $\mathcal{O}(N^{1.5})$ algorithm has appeared recently in [14] for the 3D non-oscillatory case. Fast direct solvers for oscillatory kernels are still out of reach both in 2D and 3D.

The main contribution of the current paper is to extend the approach of [8] to achieve linear complexity in the more general settings of unstructured and adaptive grids. The rest of this paper is structured as follows. In Section 2, we introduce the hierarchical partitioning used in this paper and review the multifrontal method. Our hierarchical structure is essentially a quadtree, but it also supports a natural hierarchical partitioning of geometric components of the algorithm so that unnecessary re-orderings are avoided for the algorithms in the later sections. Section 3 describes the algorithm that combines the nested dissection multifrontal method and the hierarchical matrix algebra. Our presentation follows the idea in [8] but is slightly different in the representation and inversion of the Schur complement matrices. For the experts, [8] inverts these matrices with a bottom-up algorithm, while the description here follows a top-down algorithm. The main advantage of our approach is that it provides the basic setup to address more general grids and extension to 3D. Section 4 describes the generalization of the algorithm to quasi-uniform unstructured meshes, while Section 5 presents the algorithm for an adaptive mesh featuring a range of element sizes and densities (which may, for example, arise from the mesh having been adaptively refined for a particular problem). Theoretical complexity analyses are complemented by numerical results demonstrating the properties of the proposed algorithm.

2. Hierarchical partitioning and multifrontal method

Our algorithm and hierarchical matrix decomposition is closely tied to our geometric decomposition while in [8] the matrix manipulations and the relationships between matrices on different levels is more abstract. The steps needed to combine matrices as one moves up a level in our decomposition flow naturally from easily being able to identify which geometric sets of vertices give rise to which blocks in our matrices.

For simplicity, we assume that the domain of interest Ω is $[0,1]^2$. We introduce a uniform $(P2^Q+1)\times(P2^Q+1)$ Cartesian grid covering $[0,1]^2$, where P is a positive integer of $\mathcal{O}(1)$ and Q will turn out to be the depth of the hierarchical decomposition to be introduced. The Cartesian grid is further triangulated to support piecewise linear basis functions (see Fig. 1). Since the Dirichlet zero boundary condition is specified in (1), we will be concerned with solving for the values of u at the $N=(P2^Q-1)\times(P2^Q-1)$ interior vertices. We often use lowercase Greek letters α and β to denote these vertices.

2.1. Hierarchical partitioning

We discretize (1) on the above triangulation with piecewise-linear continuous finite element basis functions $\{\phi_\alpha(\mathbf{x})\}$. Each $\phi_\alpha(\mathbf{x})$ is equal to 1 at vertex α and 0 on the other vertices. The stiffness matrix M is then given by

$$(M)_{\alpha\beta}=\int_{[0,1]^2}\nabla\phi_\alpha(\mathbf{x})\cdot a(\mathbf{x})\nabla\phi_\beta(\mathbf{x})+V(\mathbf{x})\phi_\alpha(\mathbf{x})\phi_\beta(\mathbf{x})\mathrm{d}\mathbf{x}.$$

Denote the whole domain $\Omega=[0,1]\times[0,1]$ by $\mathcal{D}_{0;0,0}$ and more generally define the contiguous *blocks* on a level q with

$$\mathcal{D}_{q;i,j}=\left[\frac{i}{2^q},\frac{i+1}{2^q}\right]\times\left[\frac{j}{2^q},\frac{j+1}{2^q}\right]\quad\text{for}\quad 0\leqslant i,j<2^q,$$

for $0\leqslant q\leqslant Q$. Clearly at level q, there are $2^q\times2^q$ blocks whose union is equal to Ω. Notice that $\mathcal{D}_{q;i,j}$ is defined to be a closed set so it contains vertices on its boundary. For each $\mathcal{D}_{Q;i,j}$, we can introduce a small matrix $M_{Q;i,j}$, which is the restriction of M to the vertices in $\mathcal{D}_{Q;i,j}$, and formed via:

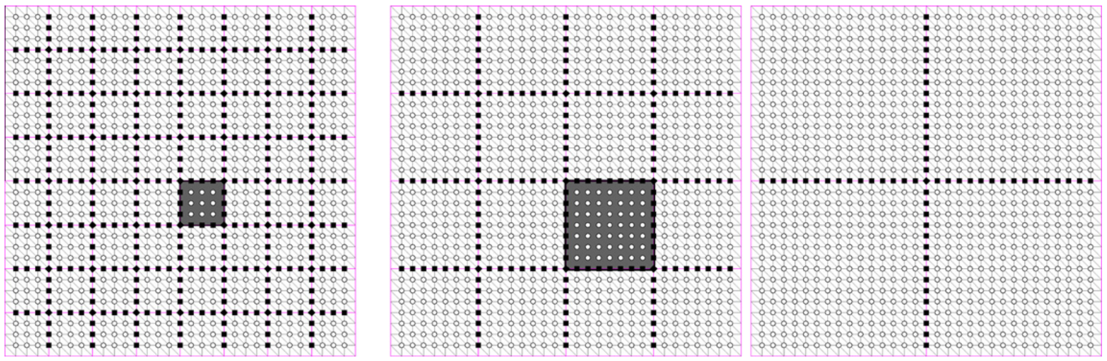


Fig. 1. Left: The whole domain is decomposed into 8 × 8 blocks on the leaf level with 5 × 5 vertices in each block (away from the domain boundary). Middle: Decomposed into 4 × 4 blocks on the next level with 9 × 9 vertices in each block. Right: Decomposed into 2 × 2 blocks on yet the next level.

$$\left(M_{Q;i,j}\right)_{\alpha\beta} = \int_{\mathcal{D}_{Q;i,j}} \nabla\phi_\alpha(\mathbf{x}) \cdot a(\mathbf{x})\nabla\phi_\beta(\mathbf{x}) + V(\mathbf{x})\phi_\alpha(\mathbf{x})\phi_\beta(\mathbf{x})\mathrm{d}\mathbf{x}, \tag{2}$$

where α and β are restricted to the vertices in $\mathcal{D}_{Q;i,j}$ since all basis functions centered on vertices outside $\mathcal{D}_{Q;i,j}$ are zero inside $\mathcal{D}_{Q;i,j}$. It is clear that these matrices $M_{Q;i,j}$ sum (after suitable injection) to the full matrix M.

Let Q be the deepest level of our hierarchical decomposition. Using the blocks introduced above, the whole domain is partitioned into $2^Q \times 2^Q$ contiguous blocks:

$$\mathcal{D}_{Q;i,j} = \left[\frac{i}{2^Q}, \frac{i+1}{2^Q}\right] \times \left[\frac{j}{2^Q}, \frac{j+1}{2^Q}\right] \quad \text{for} \quad 0 \leqslant i,j < 2^Q,$$

as illustrated in Fig. 1 (for the case $Q = 3$). There blocks are called *leaf level blocks* and the number of vertices in each of them is $(P + 1) \times (P + 1) = O(1)$.

We denote the set of vertices in $\mathcal{D}_{Q;i,j}$ by $\mathcal{V}_{Q;i,j}$. The vertices of $\mathcal{V}_{Q;i,j}$ can be decomposed into *elements* (which are vertex set themselves), depending on how many different blocks the faces containing that vertex belong to.

- *Facet element*, which includes the vertices contained in a single block. There is only one facet element for each block.
- *Segment element*, which includes the vertices shared by 2 blocks. There are 4 segment elements (top, bottom, left, and right) for each block and each segment element is shared by two blocks.
- *Corner element*, which contains only a corner where 4 blocks meet. There are 4 corner elements for each block and each corner element is shared by 4 blocks.

Notice that the boundary of a block (away from the boundary of the domain) is made up of 4 segments of length $P - 1$ and 4 corners. It is convenient to label these facet, segment, and corner elements uniformly in a Cartesian fashion as follows. All elements at leaf level Q are labelled as $\mathcal{E}_{Q;i,j}$ with $0 \leqslant i, j \leqslant 2^{Q+1}$. The vertex set $\mathcal{V}_{Q;i,j}$ of $\mathcal{D}_{Q;i,j}$ is then made up of the 9 elements:

$$\begin{array}{lll} \mathcal{E}_{Q;2i,2j+2} & \mathcal{E}_{Q;2i+1,2j+2} & \mathcal{E}_{Q;2i+2,2j+2} \\ \mathcal{E}_{Q;2i,2j+1} & \mathcal{E}_{Q;2i+1,2j+1} & \mathcal{E}_{Q;2i+2,2j+1} \\ \mathcal{E}_{Q;2i,2j} & \mathcal{E}_{Q;2i+1,2j} & \mathcal{E}_{Q;2i+2,2j} \end{array}$$

where the facet element $\mathcal{E}_{Q;2i+1,2j+1}$ is unique to the block $\mathcal{D}_{Q;i,j}$ but the surrounding elements are shared with the neighboring blocks. It is straightforward that the type of element is determined by the parity of i and j, as indicated in the following table:

Element	$i \pmod 2$	$j \pmod 2$
Corner	0	0
▬ Segment	1	0
▮ Segment	0	1
Facet	1	1

To support the multifrontal algorithm to be described, we regard the vertex set $\mathcal{V}_{Q;i,j}$ of the block $\mathcal{D}_{Q;i,j}$ as the disjoint union of the interior vertices and those on the boundary of the block (i.e. shared with other blocks). More precisely, we have:

$$\mathcal{V}_{Q;i,j} = \mathcal{I}_{Q;i,j} \uplus \mathcal{B}_{Q;i,j}, \quad \mathcal{I}_{Q;i,j} = \mathcal{E}_{Q;2i+1,2j+1}, \quad \mathcal{B}_{Q;i,j} = \biguplus_{\substack{0 \leqslant i',j' \leqslant 2 \\ (i',j') \neq (1,1)}} \mathcal{E}_{Q;2i+i',2j+j'}.$$

Here we use the symbol $\uplus$ to distinguish disjoint union from the more general union $\cup$. In Fig. 2 we show how the vertices $\mathcal{V}_{Q;i,j}$ in the block $\mathcal{D}_{Q;i,j}$ are the disjoint union of the 9 elements $\mathcal{E}_{Q;2i,2j}, \ldots, \mathcal{E}_{Q;2i+2,2j+2}$ where 1 element $\mathcal{E}_{Q;2i+1,2j+1}$ is in the "interior" $\mathcal{I}_{Q;i,j}$ of the block and the other 8 are on the "boundary" $\mathcal{B}_{Q;i,j}$ of the block.

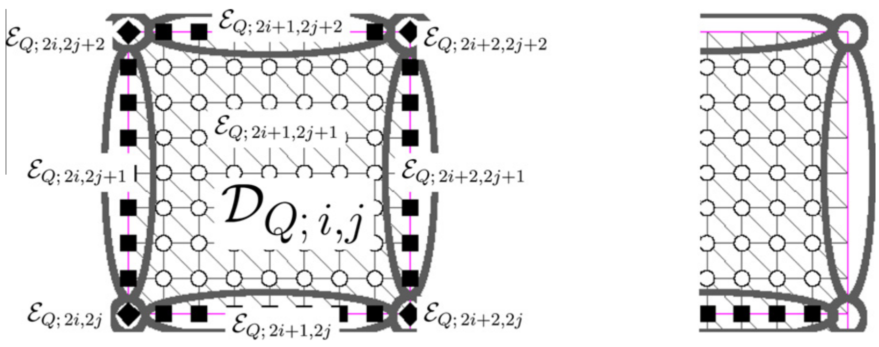
Based on what we have introduced so far, we can define the vertex sets and elements for blocks at other levels from bottom up. For a fixed level $q < Q$, suppose that $\mathcal{V}_{q+1;i,j}, \mathcal{I}_{q+1;i,j}, \mathcal{B}_{q+1;i,j}$, and $\mathcal{E}_{q+1;i,j}$ are already defined for blocks $\mathcal{D}_{q+1;i,j}$ on level $q + 1$. Then for a block $\mathcal{D}_{q;i,j}$, its vertex set $\mathcal{V}_{q;i,j}$ is defined to be the union of the boundary vertices of its child blocks, i.e.

$$\mathcal{V}_{q;i,j} = \mathcal{B}_{q+1;2i,2j} \cup \mathcal{B}_{q+1;2i+1,2j} \cup \mathcal{B}_{q+1;2i,2j+1} \cup \mathcal{B}_{q+1;2i+1,2j+1}.$$

Notice that only the boundary vertices on level $q + 1$ appear in this definition and the reason is that only these vertices "survive" to the next level (level q) in the multifrontal algorithm. The vertex set $\mathcal{V}_{q;i,j}$ is further decomposed into two parts:

- $\mathcal{I}_{q;i,j}$: interior vertices that are interior to the block $\mathcal{D}_{q;i,j}$.
- $\mathcal{B}_{q;i,j}$: boundary vertices that are shared with neighboring blocks.

More precisely, using the definition of the elements from level $q + 1$ we have:



$\mathcal{E}_{Q;\,2i,2j+2}$
$\mathcal{E}_{Q;\,2i+1,2j+2}$
$\mathcal{E}_{Q;\,2i+2,2j+2}$
$\mathcal{E}_{Q;\,2i+1,2j+1}$
$\mathcal{E}_{Q;\,2i,2j+1}$
$\mathcal{E}_{Q;\,2i+2,2j+1}$
$\mathcal{D}_{Q;\,i,j}$
$\mathcal{E}_{Q;\,2i,2j}$
$\mathcal{E}_{Q;\,2i+1,2j}$
$\mathcal{E}_{Q;\,2i+2,2j}$

Fig. 2. Left: The block $\mathcal{D}_{Q;i,j}$ whose set of vertices $\mathcal{V}_{Q;i,j}$ is the disjoint union of 9 elements $\mathcal{E}_{Q;2i,2j}, \ldots, \mathcal{E}_{Q;2i+2,2j+2}$. Right: Near the boundary of the whole domain, some of these elements may be empty.

$$\mathcal{I}_{q;i,j} = \mathcal{E}_{q+1;4i+1,4j+2} \uplus \mathcal{E}_{q+1;4i+3,4j+2} \uplus \mathcal{E}_{q+1;4i+2,4j+1} \uplus \mathcal{E}_{q+1;4i+2,4j+3} \uplus \mathcal{E}_{q+1;4i+2,4j+2},$$

and

$$\begin{aligned}\mathcal{B}_{q;i,j} = {} & \left(\mathcal{E}_{q+1;4i+1,4j} \uplus \mathcal{E}_{q+1;4i+2,4j} \uplus \mathcal{E}_{q+1;4i+3,4j}\right) \biguplus \left(\mathcal{E}_{q+1;4i+1,4j+4} \uplus \mathcal{E}_{q+1;4i+2,4j+4} \uplus \mathcal{E}_{q+1;4i+3,4j+4}\right) \\ & \biguplus \left(\mathcal{E}_{q+1;4i,4j+1} \uplus \mathcal{E}_{q+1;4i,4j+2} \uplus \mathcal{E}_{q+1;4i,4j+3}\right) \biguplus \left(\mathcal{E}_{q+1;4i+4,4j+1} \uplus \mathcal{E}_{q+1;4i+4,4j+2} \uplus \mathcal{E}_{q+1;4i+4,4j+3}\right) \\ & \biguplus \mathcal{E}_{q+1;4i,4j} \uplus \mathcal{E}_{q+1;4i+4,4j} \uplus \mathcal{E}_{q+1;4i,4j+4} \uplus \mathcal{E}_{q+1;4i+4,4j+4}.\end{aligned}$$

This decomposition of $\mathcal{V}_{q;i,j}$ into $\mathcal{I}_{q;i,j}$ and $\mathcal{B}_{q;i,j}$ is illustrated in the following diagram:

$\mathcal{E}_{q+1;\,4i,4j+4}$	$\mathcal{E}_{q+1;\,4i+1,4j+4}$	$\mathcal{E}_{q+1;\,4i+2,4j+4}$	$\mathcal{E}_{q+1;\,4i+3,4j+4}$	$\mathcal{E}_{q+1;\,4i+4,4j+4}$
$\mathcal{E}_{q+1;\,4i,4j+3}$		$\mathcal{E}_{q+1;\,4i+2,4j+3}$		$\mathcal{E}_{q+1;\,4i+4,4j+3}$
$\mathcal{E}_{q+1;\,4i,4j+2}$	$\mathcal{E}_{q+1;\,4i+1,4j+2}$	$\mathcal{E}_{q+1;\,4i+2,4j+2}$	$\mathcal{E}_{q+1;\,4i+3,4j+2}$	$\mathcal{E}_{q+1;\,4i+4,4j+2}$
$\mathcal{E}_{q+1;\,4i,4j+1}$		$\mathcal{E}_{q+1;\,4i+2,4j+1}$	$\mathcal{I}_{q;\,i,j}$	$\mathcal{E}_{q+1;\,4i+4,4j+1}$
$\mathcal{E}_{q+1;\,4i,4j}$	$\mathcal{E}_{q+1;\,4i+1,4j}$	$\mathcal{E}_{q+1;\,4i+2,4j}$	$\mathcal{E}_{q+1;\,4i+3,4j}$	$\mathcal{E}_{q+1;\,4i+4,4j}$

$\mathcal{B}_{q;\,i,j}$

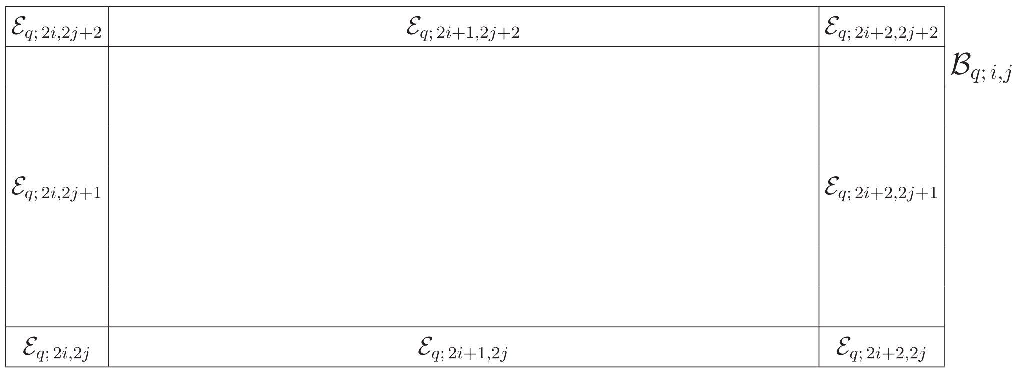
The interior $\mathcal{I}_{q;i,j}$ at level q consists of 5 elements from level $q + 1$: 4 segments and 1 corner, while the boundary $\mathcal{B}_{q;i,j}$ is made up of 16 elements from level $q + 1$: 8 segments and 8 corners. In order to support the algorithms to be described, we need introduce a decomposition of $\mathcal{B}_{q;i,j}$ into elements at level q. To do that, we create new *elements* on level q by combining elements from level $q + 1$. The rules for combining elements are as follows:

Corner	$\mathcal{E}_{q;\,2i,2j}$	$=$	$\mathcal{E}_{q+1;\,4i,4j}$
▬ Segment	$\mathcal{E}_{q;\,2i+1,2j}$	$=$	$\mathcal{E}_{q+1;\,4i+1,4j} \uplus \mathcal{E}_{q+1;\,4i+2,4j} \uplus \mathcal{E}_{q+1;\,4i+3,4j}$
▮ Segment	$\mathcal{E}_{q;\,2i,2j+1}$	$=$	$\mathcal{E}_{q+1;\,4i,4j+1} \uplus \mathcal{E}_{q+1;\,4i,4j+2} \uplus \mathcal{E}_{q+1;\,4i,4j+3}$

More specifically, each new segment at level q is the disjoint union of 3 contiguous elements (a segment element, a corner element and another segment element) from level $q + 1$. Alternatively, we can consider the segment on level q as being composed of the *segment-corner-segment* group of child elements on level $q + 1$. In this way a natural geometric hierarchy is created for the segment elements and $\mathcal{B}_{q;i,j}$ can be represented at level q as the union of 4 segments and 4 corners.

$$\mathcal{B}_{q;i,j} = \mathcal{E}_{q;2i+1,2j} \uplus \mathcal{E}_{q;2i+1,2j+2} \uplus \mathcal{E}_{q;2i,2j+1} \uplus \mathcal{E}_{q;2i+2,2j+1} \biguplus \mathcal{E}_{q;2i,2j} \uplus \mathcal{E}_{q;2i+2,2j} \uplus \mathcal{E}_{q;2i,2j+2} \uplus \mathcal{E}_{q;2i+2,2j+2}.$$

This new decomposition of $\mathcal{B}_{q;i,j}$ at level q is illustrated in the following diagram:


$\mathcal{E}_{q;\,2i,2j+2}$
$\mathcal{E}_{q;\,2i+1,2j+2}$
$\mathcal{E}_{q;\,2i+2,2j+2}$
$\mathcal{B}_{q;\,i,j}$
$\mathcal{E}_{q;\,2i,2j+1}$
$\mathcal{E}_{q;\,2i+2,2j+1}$
$\mathcal{E}_{q;\,2i,2j}$
$\mathcal{E}_{q;\,2i+1,2j}$
$\mathcal{E}_{q;\,2i+2,2j}$

This process of generating $\mathcal{V}_{q;i,j}, \mathcal{I}_{q;i,j}, \mathcal{B}_{q;i,j}$, and elements on level q from the ones on level $q + 1$ is repeated until we reach the top level $q = 0$. At level 0, due to the zero Dirichlet boundary condition specified in (1), $\mathcal{V}_{0;0,0}$ is made up the vertices on the largest cross inside the domain, $\mathcal{I}_{0;0,0} = \mathcal{V}_{0;0,0}$, and $\mathcal{B}_{0;0,0} = \emptyset$.

Let us illustrate the above discussion using a concrete example with $Q = 3$. At level $Q = 3$, we start with 8×8 blocks on the leaf level. Each block $\mathcal{D}_{2;i,j}$ on level 2 is the union of four child blocks $\mathcal{D}_{3;i',j'}$ with $\lfloor i'/2 \rfloor = i$ and $\lfloor j'/2 \rfloor = j$ for $0 \leqslant i,j < 4$. The vertices associated to $\mathcal{D}_{2;i,j}$ will be:

$$\mathcal{V}_{2;i,j} = \mathcal{B}_{3;2i,2j} \cup \mathcal{B}_{3;2i+1,2j} \cup \mathcal{B}_{3;2i,2j+1} \cup \mathcal{B}_{3;2i+1,2j+1},$$

which again decomposes into two disjoint sets, the boundary $\mathcal{B}_{2;i,j}$ which contains vertices shared with other blocks on level 2 and the interior $\mathcal{I}_{2;i,j}$ which contains vertices unique to that block. Now we can continue by combining 4 adjacent child blocks on level 2 to obtain the vertex set for $\mathcal{D}_{1;i,j}$:

$$\mathcal{V}_{1;i,j} = \mathcal{B}_{2;2i,2j} \cup \mathcal{B}_{2;2i+1,2j} \cup \mathcal{B}_{2;2i,2j+1} \cup \mathcal{B}_{2;2i+1,2j+1}$$

for $0 \leqslant i,j < 2$ and again decompose these sets on level 1 into $\mathcal{B}_{1;i,j}$ and $\mathcal{I}_{1;i,j}$. Repeating this procedure one more time we arrive at level 0 with vertex set $\mathcal{V}_{0;0,0}$, (empty) boundary $\mathcal{B}_{0;0,0}$ and interior $\mathcal{I}_{0;0,0}$ with

$$\mathcal{V}_{0;0,0} = \mathcal{I}_{0;0,0} = \mathcal{E}_{1;2,1} \uplus \mathcal{E}_{1;2,3} \uplus \mathcal{E}_{1;1,2} \uplus \mathcal{E}_{1;3,2} \uplus \mathcal{E}_{1;2,2}.$$

As we pointed out earlier, the segment elements on the higher levels are naturally endowed with a hierarchical structure, for example:

$$\mathcal{E}_{1;2,1} = \mathcal{E}_{2;4,1} \uplus \mathcal{E}_{2;4,2} \uplus \mathcal{E}_{2;4,3} = (\mathcal{E}_{3;8,1} \uplus \mathcal{E}_{3;8,2} \uplus \mathcal{E}_{3;8,3}) \biguplus \mathcal{E}_{3;8,4} \biguplus (\mathcal{E}_{3;8,5} \uplus \mathcal{E}_{3;8,6} \uplus \mathcal{E}_{3;8,7}).$$

This hierarchical decomposition leads to a tree-like structure on the vertex sets illustrated in Fig. 3. Notice that the interior vertex sets $\mathcal{I}_{q;i,j}$ on a fixed level q are disjoint. In fact all $\mathcal{I}_{q;i,j}$ are disjoint and their union over all possible choices of $0 \leqslant q \leqslant Q, 0 \leqslant i,j < 2^q$ is the set of interior vertices of the whole domain.

2.2. Multifrontal method

We now describe the multifrontal method using the hierarchical structure introduced above. Our presentation tends to emphasize the geometric aspect rather than the algebraic aspect of the method. More traditional presentations can be found

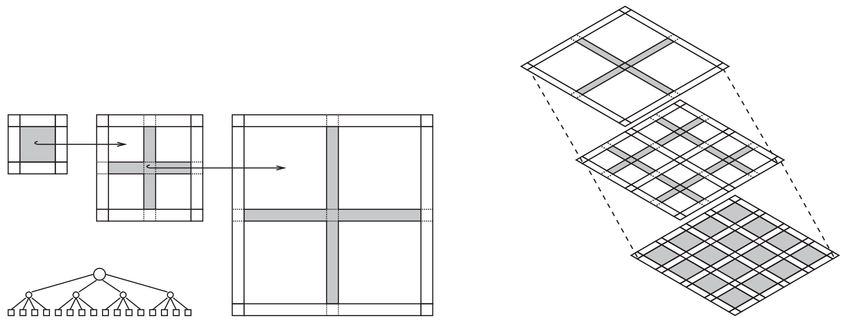


Fig. 3. Geometric tree of vertex sets resulting from a domain decomposition. Left: Blocks at different levels along a tree path from the leaf level to the top level. The gray regions denote the interior vertices $\mathcal{I}_{q;i,j}$ for each block. Right: The union of all interior vertex sets $\mathcal{I}_{q;i,j}$ is equal to the whole set of interior vertices.

in standard references [6,7]. The basic tool of multifrontal method is the block LDL^t decomposition induced by the Schur complement. For a 2×2 block matrix:

$$\begin{pmatrix} A & B^t \\ B & C \end{pmatrix},$$

the Schur complement gives rise to a factorization:

$$\begin{pmatrix} A & B^t \\ B & C \end{pmatrix} = \begin{pmatrix} I & \\ BA^{-1} & I \end{pmatrix} \begin{pmatrix} A & \\ & S \end{pmatrix} \begin{pmatrix} I & A^{-1}B^t \\ & I \end{pmatrix}$$

where $S = C - BA^{-1}B^t$.

Let us first consider the matrix $M_{Q;i,j}$ defined in (2) for the leaf block $\mathcal{D}_{Q;i,j}$. Since it is restricted to the vertices in $\mathcal{V}_{Q;i,j} = \mathcal{I}_{Q;i,j} \uplus \mathcal{B}_{Q;i,j}$, we obtain a 2×2 block matrix decomposition of $M_{Q;i,j}$:

$$M_{Q;i,j} = \begin{pmatrix} A_{Q;i,j} & B^t_{Q;i,j} \\ B_{Q;i,j} & C_{Q;i,j} \end{pmatrix} = L_{Q;i,j} \begin{pmatrix} A_{Q;i,j} & \\ & S_{Q;i,j} \end{pmatrix} L^t_{Q;i,j}, \tag{3}$$

where $A_{Q;i,j} : \mathcal{I}_{Q;i,j} \to \mathcal{I}_{Q;i,j}, B_{Q;i,j} : \mathcal{I}_{Q;i,j} \to \mathcal{B}_{Q;i,j}, C_{Q;i,j} : \mathcal{B}_{Q;i,j} \to \mathcal{B}_{Q;i,j}$, and

$$L_{Q;i,j} = \begin{pmatrix} I_{\mathcal{I}_{Q;i,j}} & \\ B_{Q;i,j}A^{-1}_{Q;i,j} & I_{\mathcal{B}_{Q;i,j}} \end{pmatrix}.$$

Here and from now on, we always order the interior vertices $\mathcal{I}_{Q;i,j}$ in front of the boundary ones $\mathcal{B}_{Q;i,j}$. Extending each $M_{Q;i,j}$ by zeros for the vertices not in $\mathcal{V}_{Q;i,j}$ and taking the sum over all of them, we get:

$$M = \sum_{i,j} M_{Q;i,j}.$$

Now extend $L_{Q;i,j}$ to the whole vertex set by setting it to be identity on the complement of $\mathcal{V}_{Q;i,j}$. Since the interior vertex sets $\mathcal{I}_{Q;i,j}$ are disjoint for different blocks $\mathcal{D}_{Q;i,j}$, each one of the $L_{Q;i,j}$ commutes with another distinct $L_{Q;i',j'}$. Therefore:

$$L_Q := \prod_{i,j} L_{Q;i,j}$$

is well defined. Note that these $L_{Q;i,j}$ and this product L_Q is useful in our presentation but is never formed explicitly in the actual algorithm, only the $B_{Q;i,j}A^{-1}_{Q;i,j}$ calculated during the Schur complement is used in our algorithms (detailed later).

We will develop a suitable ordering for the rows and columns of M as we proceed. Define:

$$\mathcal{I}_Q := \biguplus_{i,j} \mathcal{I}_{Q;i,j} \quad \text{and} \quad \mathcal{B}_Q := \bigcup_{i,j} \mathcal{B}_{Q;i,j}.$$

The union of these two sets covers the entire set of vertices for which we constructed M, and hence we can write:

$$M = \begin{pmatrix} A_Q & B^t_Q \\ B_Q & C_Q \end{pmatrix} = L_Q \begin{pmatrix} A_Q & \\ & S_Q \end{pmatrix} L^t_Q,$$

where $A_Q : \mathcal{I}_Q \to \mathcal{I}_Q, B_Q : \mathcal{I}_Q \to \mathcal{B}_Q, C_Q : \mathcal{B}_Q \to \mathcal{B}_Q$, and $S_Q = C_Q - B_Q A^{-1}_Q B^t_Q$. For each $\mathcal{D}_{Q-1;i,j}$, define a matrix $M_{Q-1;i,j} : \mathcal{V}_{Q-1;i,j} \to \mathcal{V}_{Q-1;i,j}$ to be the sum of the matrices $S_{Q;i',j'}$ of its four child blocks $\mathcal{D}_{Q;i',j'}$. From the fact that the union of $\mathcal{V}_{Q-1;i,j}$ is indeed $\mathcal{B}_Q$, it is not difficult to see that S_Q is in fact of the sum of all $M_{Q-1;i,j}$ (if we extend each $M_{Q-1;i,j}$ to be zero outside $\mathcal{V}_{Q-1;i,j}$). Now recall that each $\mathcal{V}_{Q-1;i,j}$ decomposes into $\mathcal{I}_{Q-1;i,j}$ and $\mathcal{B}_{Q-1;i,j}$. It then induces a decomposition of $\mathcal{B}_Q$ into the union of:

$$\mathcal{I}_{Q-1} := \biguplus_{i,j} \mathcal{I}_{Q-1;i,j} \quad \text{and} \quad \mathcal{B}_{Q-1} := \bigcup_{i,j} \mathcal{B}_{Q-1;i,j}.$$

and provides a 2×2 block form for $M_{Q-1;i,j}$:

$$M_{Q-1;i,j} := \begin{pmatrix} A_{Q-1;i,j} & B^t_{Q-1;i,j} \\ B_{Q-1;i,j} & C_{Q-1;i,j} \end{pmatrix}$$

where $A_{Q-1;i,j} : \mathcal{I}_{Q-1;i,j} \to \mathcal{I}_{Q-1;i,j}, B_{Q-1;i,j} : \mathcal{I}_{Q-1;i,j} \to \mathcal{B}_{Q-1;i,j}$, and $C_{Q-1;i,j} : \mathcal{B}_{Q-1;i,j} \to \mathcal{B}_{Q-1;i,j}$. We can then perform another Schur complement on this 2×2 block matrix to obtain:

$$M_{Q-1;i,j} = L_{Q-1;i,j} \begin{pmatrix} A_{Q-1;i,j} & \\ & S_{Q-1;i,j} \end{pmatrix} L^t_{Q-1;i,j}.$$

Now the combined effect of:

$$L_{Q-1} := \prod_{i,j} L_{Q-1;i,j},$$

where again we extend the $L_{Q-1;i,j}$ by the identity over the rest of $\mathcal{B}_Q$, is to factor S_Q into:

$$S_Q = L_{Q-1}\begin{pmatrix} A_{Q-1} & \\ & S_{Q-1}\end{pmatrix}L_{Q-1}^t,$$

where $A_{Q-1} : \mathcal{I}_{Q-1} \to \mathcal{I}_{Q-1}$ and $S_{Q-1} : \mathcal{B}_{Q-1} \to \mathcal{B}_{Q-1}$, and therefore:

$$M = L_Q\begin{pmatrix} A_Q & \\ & S_Q\end{pmatrix}L_Q^t = L_Q L_{Q-1}\begin{pmatrix} A_Q & & \\ & A_{Q-1} & \\ & & S_{Q-1}\end{pmatrix}L_{Q-1}^t L_Q^t,$$

where we abuse notation extending L_{Q-1} by the identity on $\mathcal{I}_Q$ as required. Recall that L_{Q-1} was the product of the $L_{Q-1;i,j}$ extended by the identity and we have continued this process to the entire vertex set. Continuing in this fashion at level q, we decompose $\mathcal{B}_{q+1}$ as $\mathcal{I}_q \uplus \mathcal{B}_q$ with:

$$\mathcal{I}_q := \biguplus_{i,j} \mathcal{I}_{q;i,j} \quad \text{and} \quad \mathcal{B}_q := \bigcup_{i,j} \mathcal{B}_{q;i,j},$$

introduce 2×2 block matrices $M_{q;i,j}$ for each $\mathcal{V}_{q;i,j}$, and apply the $L_{q;i,j}$ matrices. Finally, at level 0, we stop at $\mathcal{B}_1 = \mathcal{I}_0$ (since $\mathcal{B}_0 = \emptyset$) and obtain the following factorization for M:

$$M = L_Q L_{Q-1} \dots L_1 \begin{pmatrix} A_Q & & & & \\ & A_{Q-1} & & & \\ & & \ddots & & \\ & & & A_1 & \\ & & & & A_0 \end{pmatrix} L_1^t \dots L_{Q-1}^t L_Q^t,$$

where $A_q : \mathcal{I}_q \to \mathcal{I}_q$. Each of the A_q for $q = 0, \dots, Q$ will in fact be block diagonal if we treat:

$$\mathcal{I}_q := \biguplus_{0 \leqslant i,j < 2^q} \mathcal{I}_{q;i,j},$$

taking each of the sets $\mathcal{I}_{q;i,j}$ in turn for our ordering.

The solution to (1) can then be found by applying:

$$M^{-1} = L_Q^{-t} L_{Q-1}^{-t} \dots L_1^{-t} \begin{pmatrix} A_Q^{-1} & & & & \\ & A_{Q-1}^{-1} & & & \\ & & \ddots & & \\ & & & A_1^{-1} & \\ & & & & A_0^{-1} \end{pmatrix} L_1^{-1} \dots L_{Q-1}^{-1} L_Q^{-1}$$

to the right side of the linear system, which can be constructed in $\mathcal{O}(N^{1.5})$ steps and applied in $\mathcal{O}(N \log N)$ steps. To see this, consider Q levels with leaf blocks of size $(P+1) \times (P+1)$ so that $N \simeq (P2^Q)^2 = P^2 2^{2Q}$. For each level q, we use $s(q) \simeq P2^{Q-q}$ to denote the segment size. Then, the cost of multiplying the matrices for each block on level q will be $\mathcal{O}(s(q)^3)$ while the cost of a matrix–vector multiply will be $\mathcal{O}(s(q)^2)$. Thus the total cost, suppressing constants, for setting up M^{-1} will be:

$$\sum_{q=0}^{Q} (s(q))^3 \cdot 2^{2q} = \sum_{q=0}^{Q} P^3 2^{3(Q-q)} \cdot 2^{2q} = \mathcal{O}\left(N^{1.5}\right)$$

and that for applying it to a vector:

$$\sum_{q=0}^{Q} (s(q))^2 \cdot 2^{2q} = \sum_{q=0}^{Q} P^2 2^{2(Q-q)} \cdot 2^{2q} = \mathcal{O}(N \log N)$$

since $Q = \mathcal{O}(\log N)$.

3. Multifrontal method with hierarchical matrices

In [8], Xia et al. proposed bringing the computational cost to linear complexity $O(N)$ by combining the nested dissection multifrontal method with hierarchical matrices. Roughly speaking, hierarchical matrices are the matrices for which the degrees of freedom are grouped and ordered into hierarchical clusters using a notion of geometric closeness and the off-diagonal blocks in this ordering are numerically low-rank. Due to this low-rank property, an $N \times N$ hierarchical matrix can be stored efficiently with $O(N\log N)$ space by approximating the off-diagonal blocks at all scales with low-rank factorizations. Moreover, most of matrix operations such as matrix–vector product, matrix addition, matrix multiplication, matrix inversion, and some matrix factorizations, can be carried out in the hierarchical matrix algebra in essentially linear time, possibly with extra logarithmic factors. This topic has experienced rapid development in the past ten years and more details on hierarchical matrices can be found, for example, in [10,9].

The main observation of Xia et al. in [8] is that the matrices $M_{q;i,j}$ and its submatrices introduced in the multifrontal algorithm can be represented using hierarchical matrices. Therefore, the Schur complement calculations can be performed with hierarchical matrix algebra in almost linear time. In order to accommodate our adaptive algorithms where the nested dissection stops at different levels for different areas of a mesh we use a top-down construction and manipulation of hierarchical matrices in contrast to Xia et al.'s bottom-up approach. Table 1 lists the numerical ranks obtained in a test for a large aligned Cartesian mesh. The ranks exhibit logarithmic growth with small initial values and increase at most by 2 each time the matrix dimension doubles. This logarithmic growth of the numerical ranks is important for the complexity analysis in Section 3.3.

The algorithm and implementation proposed in [8] is rather complicated. It was not straightforward to us, at least, how to generalize their approach to unstructured and adapted meshes. We argue that the geometric decomposition introduced in Section 2 provides us with a more natural hierarchical structure on which the hierarchical matrix operations of $M_{q;i,j}$ and its submatrices can be defined more explicitly and efficiently.

Recall that the matrix $M_{q;i,j}$ is defined as a linear map from $\mathcal{V}_{q;i,j}$ to itself. Since $\mathcal{V}_{q;i,j}$ is made up of 21 elements from level $q+1$, $M_{q;i,j}$ has a 21×21 block structure. From its 2×2 block structure formed by $\mathcal{I}_{q;i,j}$ and $\mathcal{B}_{q;i,j}$, it induces:

- a5 $\times$ 5 block structure for $A_{q;i,j}$,
- a 16×5 block structure for $B_{q;i,j}$, and
- a 16×16 block structure for $C_{q;i,j}$ and $S_{q;i,j}$,

where each block in all three cases represents the interaction between two elements on level $q+1$. If the interaction is between two disjoint blocks, the block is then stored in factorized form since it is considered off-diagonal. For example, as $B_{q;i,j}$ is between $\mathcal{I}_{q;i,j}$ and $\mathcal{B}_{q;i,j}$, all its blocks are in factorized form.

If the interaction is a self-interaction, the hierarchical matrix structure is used. For example, the large diagonal blocks of $A_{q;i,j}$,$C_{q;i,j}$, and $S_{q;i,j}$ represent interaction between a segment element $\mathcal{E}_{q+1;i,j}$ on level $q+1$ and itself and they are hence in hierarchical form. The hierarchical structure of these blocks naturally appears from the geometric decomposition discussed in Section 2. Let us recall that each segment $\mathcal{E}_{q+1;i,j}$ (above the leaf level) is decomposed into the union of two segments and a corner from level $q+1$. Using this decomposition, the self-interaction of this segment can be naturally represented as a

Table 1
The maximum numerical ranks for factorized matrices in square off-diagonal blocks observed while solving $-\Delta u = f$ with $\epsilon_a = 10^{-12}$ and $\epsilon_r = 10^{-6}$. These grow like $\mathcal{O}(\log s)$.

Segment size s	31	63	127	255	511	1023	2047
A^{-1}	8	9	11	12	13	15	16
B	10	11	13	15	16	18	–
S	10	11	13	15	16	18	19

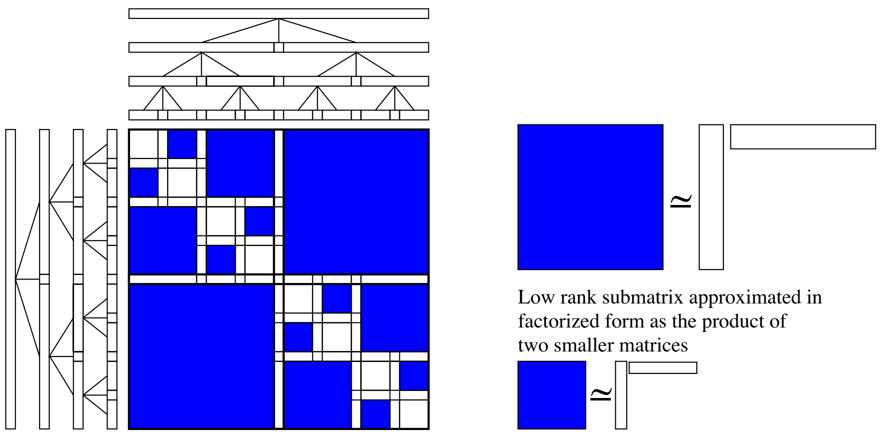

Low rank submatrix approximated in factorized form as the product of two smaller matrices

Fig. 4. Hierarchical subdivision of the sub-block of a matrix representing a segment-segment self-interaction.

3×3 block matrix, with each block representing the interaction between the constituting elements from level $q+1$. Each off-diagonal block can be represented in the low-rank factorized form, while the two large diagonal blocks associated with two segments from level $q+1$ are again represented as 3×3 block matrices hierarchically if level $q+1$ is above the leaf level. A typical example is illustrated in Fig. 4. The decomposition of the whole $M_{q;i,j}$ matrix is illustrated in Fig. 5.

One extra important structure appears in $S_{q;i,j} : \mathcal{B}_{q;i,j} \to \mathcal{B}_{q;i,j}$. Recall that the boundary $\mathcal{B}_{q;i,j}$ also has a decomposition in terms of eight elements on level q, which implies that the matrix $S_{q;i,j}$ also has an 8×8 block decomposition on level q. The transformation of the 16×16 decomposition of $S_{q;i,j}$ into its 8×8 decomposition is an important part of our algorithm and will be detailed later.

3.1. Algorithms

Under our geometric hierarchical setup, the multifrontal factorization of M with hierarchical matrices takes two stages:

1. At the leaf level we calculate $M_{Q;i,j}$, which is the restriction M to $\mathcal{D}_{Q;i,j}$, and then perform the Schur complement to obtain $S_{Q;i,j}$.
2. Move up level by level combining the 4 child $S_{q+1;i',j'}$ matrices into the matrix $M_{q;i,j}$ which again, after the Schur complement, provides the matrix $S_{q;i,j}$ of the parent block.

Algorithm 1 shows how the factorized form of M is constructed. Here we use the following convention of referring to a submatrix: if $G \in \mathbb{R}^{|\mathcal{J}|\times|\mathcal{J}|}$ is a matrix whose rows and columns are labeled by the index set $\mathcal{J}$ then for $\mathcal{X} \subset \mathcal{J}$ we write $G(\mathcal{X},\mathcal{X}) \in \mathbb{R}^{|\mathcal{X}|\times|\mathcal{X}|}$ for the submatrix of G consisting of the rows and columns in $\mathcal{X}$. Manipulating this matrix affects the underlying values in G.

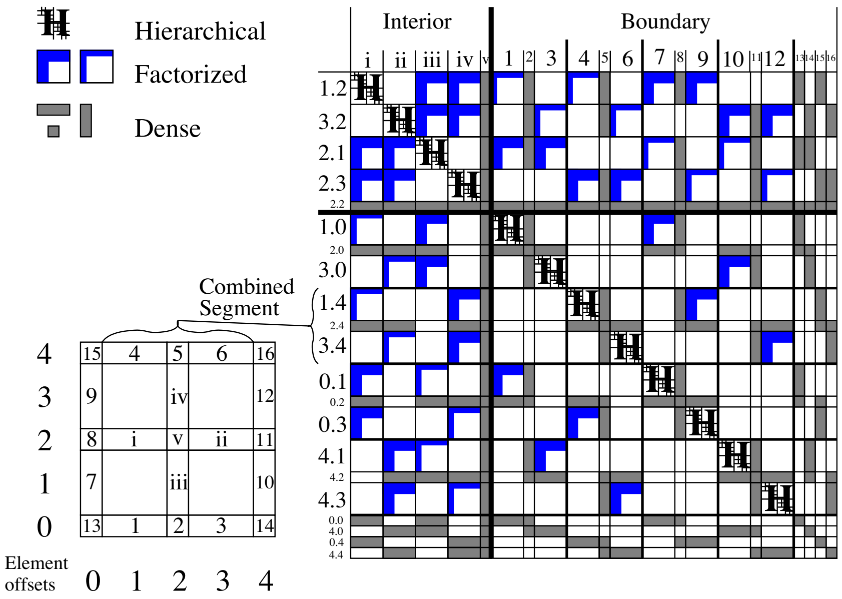


Fig. 5. Block decomposition of $M_{q;i,j}$ into 21 = 5 + 16 blocks. The relative sizes of segments and corners is typical of leaf elements and in general for higher levels the segments would be much more dominant. On the top we have labeled the blocks sequentially using i to v for the interior and 1 to 16 for the boundary with the corresponding elements numbered in the accompanying diagram. On the left we have used the element offsets 0,0–4,4.

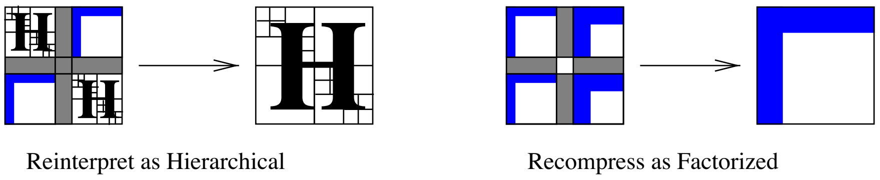


Fig. 6. Illustration of two components of the merge procedure, reinterpreting a group of matrices as hierarchical and recompressing into a new factorized form.

Algorithm 1. Setup the factorization of M

```
1:  for $i = 0$ to $2^Q - 1$ do
2:    for $j = 0$ to $2^Q - 1$ do
3:      Calculate the matrix $M_{Q;i,j}$ as in (2)
4:      Invert $A_{Q;i,j}$ using dense matrix methods.
5:      $S_{Q;i,j} \leftarrow C_{Q;i,j} - B_{Q;i,j} A^{-1}_{Q;i,j} B^t_{Q;i,j}$
6:      Store $A^{-1}_{Q;i,j}$, $B_{Q;i,j}$ and $S_{Q;i,j}$
7:    end for
8:  end for
9:  for $q = Q - 1$ to 1 do
10:   for $i = 0$ to $2^q - 1$ do
11:     for $j = 0$ to $2^q - 1$ do
12:       Start with zero $M_{q;i,j}$
13:       for $i' = 0,1$ do
14:         for $j' = 0,1$ do
15:           $M_{q;i,j}\left(\mathcal{B}_{q+1;2i+i',2j+j'}, \mathcal{B}_{q+1;2i+i',2j+j'}\right) \leftarrow M_{q;i,j}(\mathcal{B}_{q+1;2i+i',2j+j'}, \mathcal{B}_{q+1;2i+i',2j+j'}) + S_{q+1;2i+i',2j+j'}$
16:         end for
17:       end for
18:       Define $M_{q;i,j} = \begin{pmatrix} A_{q;i,j} & B^t_{q;i,j} \\ B_{q;i,j} & C_{q;i,j} \end{pmatrix}$
19:       Invert $A_{q;i,j}$
20:       $S_{q;i,j} \leftarrow C_{q;i,j} - B_{q;i,j} A^{-1}_{q;i,j} B^t_{q;i,j}$
21:       Store $A^{-1}_{q;i,j}$ and $B_{q;i,j}$
22:       Merge and Store $S_{q;i,j}$
23:     end for
24:   end for
25: end for
26: Start with zero $M_{0;0,0}$
27: for $i' = 0,1$ do
28:   for $j' = 0,1$ do
29:     $M_{0;0,0}\left(\mathcal{B}_{1;i',j'}, \mathcal{B}_{1;i',j'}\right) \leftarrow M_{0;0,0}(\mathcal{B}_{1;i',j'}, \mathcal{B}_{1;i',j'}) + S_{1;i',j'}$
30:   end for
31: end for
32: Invert $A_{0;0,0}$
33: Store $A^{-1}_{0;0,0}$
```

The step *Merge* $S_{q;i,j}$ in Algorithm 1 is required because, as we mentioned earlier, one needs to reinterpret the 16×16 block structure corresponding to the 16 boundary elements at level $q + 1$ as an 8×8 block structure corresponding to the 8 merged boundary elements on level q. While the 4 corner vertices are unaffected, the segment-corner-segment merging of child elements is reflected in combining 3×3 submatrices into a new submatrix. The vertex ordering we built up from the leaf level ensures that, in fact, these 9 submatrices form a contiguous 3×3 group. Thus no rearrangement of the rows and columns of $S_{q;i,j}$ is required. In terms of the hierarchical matrix representation, if the new submatrix is on the diagonal and should have a hierarchical representation this is achieved by simply reinterpreting the 3×3 submatrices as part of a new hierarchical decomposition. On the other hand, if the new submatrix is off-diagonal and should be represented in factorized form, this "recompression" can be performed efficiently using QR factorizations since the major parts are already in factorized form. These two cases are illustrated graphically in Fig. 6.

Note that this merge step is only required to maintain the expected complexity of the hierarchical matrix algebra. The usual permutations and "extend-add" operations of general multifrontal approaches are avoided because the node ordering and hierarchical division of our matrices is built up from the lowest level to be compatible with the nested disection. Step 15 of Algorithm 1 which adds together the child $S_{q+1;i',j'}$ matrices is the analogue of "extend-add" but is mainly injection with some dense matrix addition. The geometric separation of the child domains ensures that at most one of the child $S_{q+1;i',j'}$ matrices contributes to any of the 8×8 un-merged child segment-segment interactions in the parent $M_{q;i,j}$. While the illustration in Fig. 5 features segments that are the same size and aligned with each other we shall see later in Section 4 that our algorithm does not rely on all the segments being the same size or aligned with a grid. The resulting pattern of entries in the parent $M_{q;i,j}$ block matrix follows from the topological relationships of exactly four shared segments from the children meeting in the central corner (away from the domain boundaries) and the segment-corner-segment child elements on the parent boundary combining to form the parent segments.

To solve the original $Mu = f$, we compute $u = M^{-1}f$ using the multifrontal decomposition of the matrix M^{-1}:

$$M^{-1}f = L_Q^{-t}L_{Q-1}^{-t}\dots L_1^{-t}\begin{pmatrix} A_Q^{-1} & & & & \\ & A_{Q-1}^{-1} & & & \\ & & \ddots & & \\ & & & A_1^{-1} & \\ & & & & A_0^{-1} \end{pmatrix} L_1^{-1}\dots L_{Q-1}^{-1}L_Q^{-1}f.$$

To carry out this calculation, we first apply each factor $L_{Q;i,j}^{-1}$ in L_Q^{-1}, then those from L_{Q-1}^{-1} and so on. Once we have completed all the $L_{q;i,j}^{-1}$, we apply the diagonal blocks $A_{q;i,j}^{-1}$, and then all the $L_{q;i,j}^{-t}$ for $q = 1,\dots,Q$. If we write $u_{\mathcal{I}_{q;i,j}}$ for the (consecutive) group of components of u corresponding to the set of vertices $\mathcal{I}_{q;i,j}$, and similarly $u_{\mathcal{B}_{q;i,j}}$, then the solution can be calculated as in Algorithm 2 where we combine the action of $A_{q;i,j}^{-1}$ and $L_{q;i,j}^{-1}$ since they are the only ones which affect $u_{\mathcal{I}_{q;i,j}}$ on the first pass from the leaves to the root of the tree.

Algorithm 2. Solving $Mu = f$

```
 1: $u \leftarrow f$
 2: for $q = Q$ to 1 do
 3:   fori = 0 to $2^q - 1$ do
 4:     for $j = 0$ to $2^q - 1$ do
 5:       $u_{\mathcal{I}_{q;i,j}} \leftarrow A_{q;i,j}^{-1}u_{\mathcal{I}_{q;i,j}}$
 6:       $u_{\mathcal{B}_{q;i,j}} \leftarrow u_{\mathcal{B}_{q;i,j}} - B_{q;i,j}A_{q;i,j}^{-1}u_{\mathcal{I}_{q;i,j}}$
 7:     end for
 8:   end for
 9: end for
10: $u_{\mathcal{I}_{0;0,0}} \leftarrow A_{0;0,0}^{-1}u_{\mathcal{I}_{0;0,0}}$
11: for $q = 1$ to $Q$ do
12:   for $i = 0$ to $2^q - 1$ do
13:     for $j = 0$ to $2^q - 1$do
14:       $u_{\mathcal{I}_{q;i,j}} \leftarrow u_{\mathcal{I}_{q;i,j}} - A_{q;i,j}^{-1}B_{q;i,j}^t u_{\mathcal{B}_{q;i,j}}$
15:     end for
16:   end for
17: end for
```

3.2. Implementation details

In our implementation, the matrices at several lowest levels are in fact represented as dense matrices, instead of hierarchical matrices. This is to avoid small dense matrix computations and to achieve the best complexity at several lowest levels (see the discussion on page 300 of [11]).

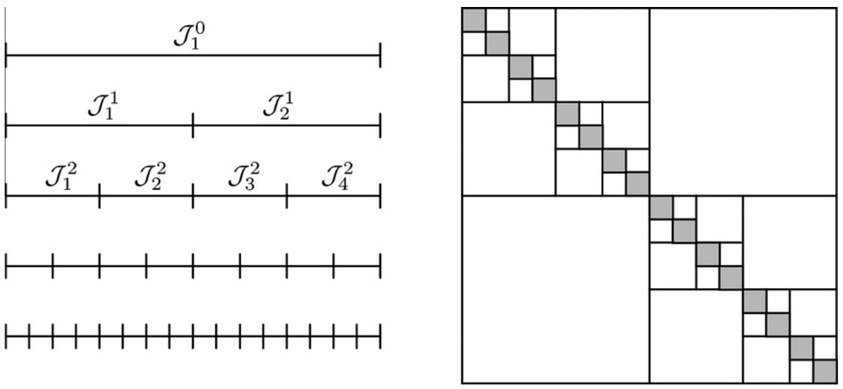


Fig. 7. 1D domain decomposition into sets $\mathcal{J}_i^\ell$ at level ℓ, with corresponding self-interaction matrix decomposition at level 4. The dark blocks are dense while the others are low rank and can be represented in factorized form.

We refer the reader to [10] for details on hierarchical matrix operations but present some aspects of our implementation. The underlying dense matrix algebra is performed using BLAS and LAPACK (in particular Intel's MKL), for example matrix inversion of dense matrices is performed via LU-factorization.

The inversion of a hierarchical matrix proceeds using row operations on the block structure. This requires the inversion of the hierarchical matrices on the diagonal and so the problem is recursive. The recursion ends when the matrices on the diagonal are dense and no longer hierarchical, and then the inversion is performed using the dense matrix techniques described above. Thus inversion requires hierarchical matrix multiplication and addition which we now discuss.

We utilize a simplified one dimensional setting with bisection to illustrate the approach (see Fig. 7).

The index set $\mathcal{J}_1^0$ is partitioned hierarchically with bisection which stops when each set $\mathcal{J}_i^\ell$ contains only a small number of indices. We denote the restriction of a matrix G to $\mathcal{J}_i^\ell$ and $\mathcal{J}_{i'}^\ell$ by $G_{i,i'}^\ell$.

3.2.1. Matrix addition and subtraction

Consider the sum of two matrices G and H with their off-diagonal factorizations denoted by $G_{i,j}^\ell \approx U_{i,j}^\ell \left(V_{i,j}^\ell\right)^t$ and $H_{i,j}^\ell \approx X_{i,j}^\ell \left(Y_{i,j}^\ell\right)^t$. Under the block matrix notation, the sum is

$$\begin{pmatrix} G_{1,1}^1 & G_{1,2}^1 \\ G_{2,1}^1 & G_{2,2}^1 \end{pmatrix} + \begin{pmatrix} H_{1,1}^1 & H_{1,2}^1 \\ H_{2,1}^1 & H_{2,2}^1 \end{pmatrix} = \begin{pmatrix} G_{1,1}^1 + H_{1,1}^1 & G_{1,2}^1 + H_{1,2}^1 \\ G_{2,1}^1 + H_{2,1}^1 & G_{2,2}^1 + H_{2,2}^1 \end{pmatrix}.$$

First,

$$G_{1,2}^1 + H_{1,2}^1 \approx U_{1,2}^1 \left(V_{1,2}^1\right)^t + X_{1,2}^1 \left(Y_{1,2}^1\right)^t = \left(U_{1,2}^1 \; X_{1,2}^1\right)\left(V_{1,2}^1 \; Y_{1,2}^1\right)^t.$$

Notice that the new factorized form for the sum will have an increased size compared to those for $G_{1,2}^1$ and $H_{1,2}^1$. One needs to recompress the last two matrices in order to prevent the size of the low rank factorization from increasing indefinitely. More precisely, if $U_{1,2}^1$ has width r_1 and $X_{1,2}^1$ has width r_2, then using $\left(U_{1,2}^1 \; X_{1,2}^1\right) = QR$ and $\left(V_{1,2}^1 \; Y_{1,2}^1\right) = \widetilde{Q}\widetilde{R}$, the sum we seek is $QR(\widetilde{Q}\widetilde{R})^t = Q(R\widetilde{R}^t)\widetilde{Q}^t$, and we need only perform the SVD, $R\widetilde{R}^t = \overline{U}\overline{\Sigma}\overline{V}^t$, on a square matrix of size $r_1 + r_2$. Finally the resulting factors for the sum will have width $r' \leqslant r_1 + r_2$ if we keep r' of the singular values (and associated columns from $\overline{U}$ and $\overline{V}$) for our truncated SVD. Thus:

$$\left(U_{1,2}^1 \; X_{1,2}^1\right)\left(V_{1,2}^1 \; Y_{1,2}^1\right)^t = \underbrace{Q\overline{U}\overline{\Sigma}}_{\text{width } r'} \left(\underbrace{\widetilde{Q}\overline{V}}_{\text{width } r'}\right)^t.$$

The same procedure is carried out for $G_{2,1}^1 + H_{2,1}^1$ to compute the necessary factorization.

Second, let us consider the diagonal blocks. $G_{1,1}^1 + H_{1,1}^1$ and $G_{2,2}^1 + H_{2,2}^1$ are done recursively since they are two sums of a similar nature to our original sum, but of smaller size. Eventually the diagonal blocks are dense and standard matrix addition is performed.

3.2.2. Matrix–vector multiplication

Assuming the vector is also decomposed according to the index sets block multiplication is performed. The two matrices from each factorized off-diagonal form are dense and the on-diagonal hierarchical matrices are treated recursively. Eventually the diagonal blocks are dense and standard matrix–vector multiplication is performed. It should be clear that a similar procedure works for vector–matrix multiplication.

3.2.3. Matrix multiplication

Let us consider the product of two matrices G and H with their off-diagonal factorizations given again by $G_{i,j}^\ell \approx U_{i,j}^\ell \left(V_{i,j}^\ell\right)^t$ and $H_{i,j}^\ell \approx X_{i,j}^\ell \left(Y_{i,j}^\ell\right)^t$. In block matrix form, the product is:

$$\begin{pmatrix} G_{1,1}^1 & G_{1,2}^1 \\ G_{2,1}^1 & G_{2,2}^1 \end{pmatrix} \cdot \begin{pmatrix} H_{1,1}^1 & H_{1,2}^1 \\ H_{2,1}^1 & H_{2,2}^1 \end{pmatrix} = \begin{pmatrix} G_{1,1}^1 H_{1,1}^1 + G_{1,2}^1 H_{2,1}^1 & G_{1,1}^1 H_{1,2}^1 + G_{1,2}^1 H_{2,2}^1 \\ G_{2,1}^1 H_{1,1}^1 + G_{2,2}^1 H_{2,1}^1 & G_{2,1}^1 H_{1,2}^1 + G_{2,2}^1 H_{2,2}^1 \end{pmatrix}.$$

First, the off-diagonal block:

$$G_{1,1}^1 H_{1,2}^1 + G_{1,2}^1 H_{2,2}^1 \approx G_{1,1}^1 X_{1,2}^1 \left(Y_{1,2}^1\right)^t + U_{1,2}^1 \left(V_{1,2}^1\right)^t H_{2,2}^1.$$

The computation $G_{1,1}^1 X_{1,2}^1$ is multiplication of a hierarchical matrix with a dense matrix with a small number of columns and proceeds in essentially the same way as matrix–vector multiplication and similarly $\left(V_{1,2}^1\right)^t H_{2,2}^1$ mimics vector–matrix multiplication. Once they are done, the remaining sum is then similar to the sum of the factorized off-diagonal parts of the matrix addition algorithm. The other off-diagonal block $G^1_{2,1}H^1_{1,1} + G^1_{2,2}H^1_{2,1}$ is done in the same way.

Next, consider the diagonal blocks. Take $G^1_{1,1}H^1_{1,1} + G^1_{1,2}H^1_{2,1}$ as an example. The first part $G^1_{1,1}H^1_{1,1}$ is done using recursion. The second part is:

$$G^1_{1,2}H^1_{2,1} \approx U^1_{1,2}\underbrace{\left(V^1_{1,2}\right)^t X^1_{2,1}}\left(Y^1_{2,1}\right)^t.$$

Performing the middle product first minimizes the computational cost. The final sum $G^1_{1,1}H^1_{1,1} + G^1_{1,2}H^1_{2,1}$ is done using a matrix addition algorithm similar to the one described above. The same procedure can be carried out for the computation of $G^1_{2,1}H^1_{1,2} + G^1_{2,2}H^1_{2,2}$.

In general the hierarchical Schur complement matrices on level q combine in groups of 4 by injection and addition to form the hierarchical matrix on level $q-1$ (see Fig. 5 for the typical structure of this new hierarchical matrix). The Schur complement calculation then involves hierarchical matrix operations and results in a new hierarchical matrix representing the Schur complement. Only for the lowest levels where we have chosen to use dense matrices instead of hierarchical ones will the Schur complement be dense.

3.3. Complexity

For the complexity analysis, recall that a leaf node at level Q contains $(P+1)\times(P+1)$ vertices and $N \simeq (P2^Q)^2 = P^2 2^{2Q} = O(2^{2Q})$. Here all logarithms are taken with base 2.

At level q, the size of a segment element is $s(q) = P2^{Q-q} = \mathcal{O}(2^{Q-q})$, therefore the matrices $M_{q;i,j}$,$A_{q;i,j}$,$B_{q;i,j}$, $C_{q;i,j}$, and $S_{q;i,j}$ are all of dimension $\mathcal{O}(s(q))$. A crucial quantity is the rank of the off-diagonal blocks of these matrices. In our case the rank varies within the hierarchical form but we have observed that the rank increases logarithmically with segment size, so that the maximum rank will be $\mathcal{O}(\log s(q))$. This agrees with the experimental observations of Börm [15] regarding the ranks of the factorized blocks for the inverse of an elliptic operator, although he found a theoretical bound of $\mathcal{O}((\log s(q))^3)$. To cover both the observed and theoretical bounds we will continue our analysis with the general ansatz that the rank will be $\mathcal{O}((\log s(q))^\rho)$ for some integer $\rho \geqslant 1$.

In Algorithm 1, the dominant computation is the formation of the Schur complement for each block $\mathcal{D}_{q;i,j}$, which involves inversion, multiplications, and addition of hierarchical matrices. The cost of these operations is given in [11] as $\mathcal{O}(r^2(\log n)^2 n)$ where r is the maximum rank of the factorized parts, $n\times n$ the full size of the matrix and log (n) the number of block subdivisions (depth of the decomposition tree) in the hierarchical form. In our case, since $r = \mathcal{O}((\log s(q))^\rho)$ and $n = \mathcal{O}(s(q))$, this is equal to:

$$\mathcal{O}\Big((\log s(q))^{2\rho}\cdot(\log s(q))^2\cdot s(q)\Big) = \mathcal{O}\Big((\log s(q))^{2\rho+2}\cdot s(q)\Big).$$

Now, since there are 2^{2q} Schur complements at each level and Q levels in total, the overall cost of Algorithm 1 is on the order of:

$$\sum_{q=0}^{Q}(Q-q)^{2\rho+2}\cdot 2^{Q-q}\cdot 2^{2q} = \mathcal{O}(2^{2Q}) = O(N).$$

In Algorithm 2, the dominant cost is the matrix vector multiplication in the hierarchical matrix form. In [11], this cost is shown to be of order $\mathcal{O}(r^2(\log n)^2 n)$ where r is again the maximum rank of the factorized parts, $n\times n$ is the size of the matrix. At level $q, r = \mathcal{O}((\log s(q))^\rho)$ and $n = \mathcal{O}(s(q))$, and the cost is:

$$\mathcal{O}\Big((\log s(q))^{2\rho}\cdot\log s(q)\cdot s(q)\Big) = \mathcal{O}\Big((\log s(q))^{2\rho+1}\cdot s(q)\Big).$$

Summing this cost over 2^{2q} Schur complements at each of Q levels gives the cost of Algorithm 2:

$$\sum_{q=0}^{Q}(Q-q)^{2\rho+1}\cdot 2^{Q-q}\cdot 2^{2q} = \mathcal{O}(2^{2Q}) = O(N).$$

To further speed up the addition of the hierarchical matrices one can use probabilistic [16] low-rank approximants instead of those calculated via SVD, but in the multiplication of two factorized low-rank matrices of size $n\times n$ and rank r we still need $\mathcal{O}(nr^2)$ multiplications.

3.4. Numerical results

All numerical tests are run on a 2.13 GHz processor. Execution times are measured in seconds for the *Setup* phase (Algorithm 1) and the *Solve* phase (Algorithm 2).

To test our algorithm we setup the factorized form of M and solve 100 random problems generated as follows: Select $x^\star \in \mathbb{R}^N$ with independent standard normal components, and calculate $f = Mx^\star$ using the sparse original M. Then solve $Mx = f$ and determine the worst relative L_2 error:

$$\frac{\|x - x^\star\|_2}{\|x^\star\|_2},$$

over the 100 samples.

Following [10] we construct the low rank approximations at the hierarchical levels using common matrix manipulations such as QR and SVD. During these procedures we keep only (the part of the decomposition corresponding to) those singular values:

1. larger than the *absolute cutoff* ϵ_a and
2. within the *relative cutoff* ϵ_r of the largest singular value.

Addition and multiplication of hierarchical matrices also involves these kinds of *truncated SVD*. These two parameters, ϵ_a and ϵ_r, can be varied depending on the specific problem and the desired accuracy of the output.

The first test is the Laplace equation $-\Delta u = f$ on $[0,1]^2$ with zero Dirichlet boundary condition. In Table 2 we show how the setup time and the error vary using fixed $\epsilon_a = 10^{-12}$ and various choices of ϵ_r. The resulting error compares well with the

Table 2
Numerical results for a uniform mesh on $[0,1]^2$ using $\epsilon_a = 10^{-12}$ and various choices of ϵ_r for $-\Delta u = f$.

N	Q	$\epsilon_r = 10^{-4}$		$\epsilon_r = 10^{-6}$		$\epsilon_r = 10^{-8}$		$\epsilon_r = 10^{-10}$	
		Setup	Error	Setup	Error	Setup	Error	Setup	Error
16129	4	0.84	1.34e−04	0.84	1.47e−06	0.85	8.22e−09	0.84	5.24e−11
65025	5	3.75	2.66e−04	3.85	2.13e−06	3.92	2.70e−08	3.94	1.70e−10
261121	6	16.14	7.60e−04	16.84	5.68e−06	17.27	5.37e−08	17.63	4.07e−10
1046529	7	67.59	1.37e−03	71.23	1.58e−05	72.76	9.60e−08	75.72	1.05e−09
4190209	8	282.24	2.38e−03	295.41	2.77e−05	306.69	2.99e−07	320.26	2.29e−09
16769025	9	1167.50	4.62e−03	1226.11	6.38e−05	1277.76	4.56e−07	1337.95	6.49e−09

Table 3
Numerical results for a uniform mesh using $\epsilon_a = 10^{-12}$ and $\epsilon_r = 10^{-6}$ for $-\Delta u = f$. In the second set of results ϵ_r was initially 10^{-6} for $N = 16129$ and then it was halved for each increase in size.

N	$\epsilon_r = 10^{-6}$			Halved each step		
	Setup	Solve	Error	Setup	Solve	Error
16129	0.84	0.02	1.47e−06	0.85	0.02	1.47e−06
65025	3.85	0.11	2.13e−06	3.86	0.11	9.41e−07
261121	16.84	0.49	5.68e−06	16.93	0.49	1.44e−06
1046529	71.23	2.08	1.58e−05	72.39	2.06	1.32e−06
4190209	295.41	8.89	2.77e−05	305.49	8.90	1.46e−06
16769025	1226.11	36.90	6.38e−05	1266.18	36.74	1.16e−06

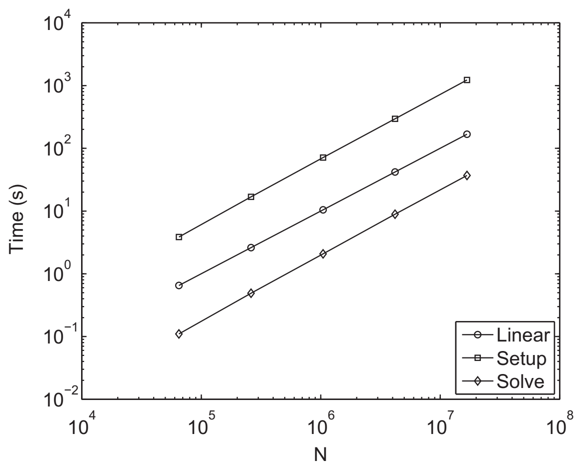


Fig. 8. A log–log plot of the time taken for the setup and solve phases of the algorithm against the number of degrees of freedom, along with $t = 10^{-5}N$ for comparison.

chosen value of ϵ_r, with each improvement of 10^{-2} on ϵ_r costing about a 5% increase in runtime. The increase in runtime from $N = 16129$ to $N = 16769025$ is reasonably close to the expected linear increase of 4 each step.

Notice from the results in Table 2 that the error increases slightly with N ([8] experienced a similar increase). To compensate for this, we can reduce ϵ_r as N increases as shown in the second test, which uses fixed $\epsilon_a = 10^{-12}$ and starts with $\epsilon_r = 10^{-6}$. The results in Table 3 show that the reduction of ϵ_r results in a minor impact on computational cost. Because a smaller ϵ_r implies higher ranks, the runtime scaling is slightly worse for the second set, but still close to the ideal factor of 4. The error remains relatively constant as desired. Alternatively, one could use our solver as a preconditioner for PCG or GMRES if better accuracy is desired. Plotting runtime against N on a log–log plot as in Fig. 8 allows us to compare the growth in runtime for the setup and solve algorithms with linear growth.

In the third test reported in Table 4 we solve $-\operatorname{div}(a(\mathbf{x})\nabla u) = f$ on $[0,1]^2$ with zero Dirichlet boundary condition for a more general $a(\mathbf{x})$ which jumps between 10^{-2} and 10^{2} with $\epsilon_a = 10^{-12}$ and $\epsilon_r = 10^{-10}$ rather than $\epsilon_r = 10^{-6}$ to accommodate the jumps of order 10^4 in $a(\mathbf{x})$. The runtime scaling is again close to the optimal value and the error is well controlled.

In the fourth test with results shown in Table 5 we study the case of positive $V(\mathbf{x})$ in $-\Delta u + V(\mathbf{x})u = f$. One experiment uses $V(\mathbf{x})$ chosen uniformly in $[0, 10^5]$, while the second experiment uses $V(\mathbf{x})$ that takes value 0 on 95% of the triangles and 10^5 on the remaining 5%. The scaling for the two scenarios is very similar but the one with the jumps is slightly slower.

In the fifth test displayed in Table 6 we show how the algorithm can be extended to slightly more general $V(\mathbf{x})$ which are chosen uniformly in $[-100, 100]$ and $[-100, 0]$. In the latter case we set $\epsilon_r = 10^{-8}$ in order to maintain an error near our target 10^{-6}. This demonstrates that, while some adjustments have to be made for a non-positive definite system, the algorithm still works well with close to optimal runtime scaling.

Table 4
Numerical results for a uniform mesh on $[0,1]^2$ using $\epsilon_a = 10^{-12}$ and $\epsilon_r = 10^{-10}$ for $-\operatorname{div}(a(\mathbf{x})\nabla u) = f$ where $a(\mathbf{x})$ jumps by 10^4 from one subset of the domain to another, more specifically $a(\mathbf{x}) \equiv 10^{-2}$ except for two regions $[0.25, 050]^2$ and $[0.50, 0.75]^2$ where $a(\mathbf{x}) \equiv 10^2$.

N	Setup	Solve	Error
16129	0.86	0.02	6.24e−06
65025	3.94	0.11	1.21e−05
261121	17.54	0.51	3.15e−05
1046529	74.95	2.25	8.21e−06
4190209	317.93	9.95	1.26e−06

Table 5
Results for $-\Delta u + V(\mathbf{x})u = f$ with a positive $V(x)$. Here we use $\epsilon_a = 10^{-12}$ and $\epsilon_r = 10^{-6}$. In the first set of results $V(\mathbf{x})$ is chosen uniformly in $[0, 10^5]$ and in second set of results $V(\mathbf{x})$ is identically 10^5 on a randomly chosen 5% of the triangles and identically 0 on the remaining 95%.

N	V uniformly in $[0, 10^5]$			V jumps between 0 and 10^5		
	Setup	Solve	Error	Setup	Solve	Error
11618	0.79	0.02	3.78e−09	0.90	0.02	4.31e−08
46865	3.96	0.11	1.44e−08	4.26	0.13	9.16e−08
188249	17.17	0.52	3.69e−08	18.26	0.48	1.42e−07
754577	75.72	2.30	8.67e−08	79.79	3.02	2.80e−07
3021473	332.44	7.94	1.75e−07	353.59	11.65	6.51e−07

Table 6
Slightly more general $V(\mathbf{x})$ is also possible. Here are the results for an aligned Cartesian mesh using $\epsilon_a = 10^{-12}$ for $-\Delta u + V(\mathbf{x})u = f$, where for small negative $V(\mathbf{x})$ we have to adjust ϵ_r to maintain the error around 10^{-6}.

N	V uniformly in $[-100, 100]$ $\epsilon_r = 10^{-6}$			V uniformly in $[-100, 0]$ $\epsilon_r = 10^{-8}$		
	Setup	Solve	Error	Setup	Solve	Error
16129	0.95	0.03	1.47e−06	1.01	0.03	8.37e−07
65025	4.09	0.13	2.12e−06	4.58	0.15	7.03e−06
261121	18.49	0.55	5.74e−06	20.33	0.66	3.07e−06
1046529	78.28	2.34	1.57e−05	86.69	2.78	6.29e−06
4190209	328.57	9.98	2.77e−05	369.58	11.78	9.55e−06

4. Quasi-uniform mesh

In this section, we discuss how to extend the approach described above to quasi-uniform meshes, which are those with:

1. the angles of every triangle uniformly bounded away from zero, and
2. a bounded ratio between the area of the largest and smallest triangles.

Common ways to extend algorithms involving nested disection to unstructured meshes [8, Section 4.6] use graph partitioning algorithms, such as in [17] or other algebraic [18] methods to determine a node ordering and hierarchical disection. However, there is little control over the topology of the resulting disection and the meeting of separators from different levels. Since our method relies strongly on the clean geometric hierarchy to provide a node ordering and permutation free matrix algebra we introduce our own, more geometric, approach that preserves the relationship between segments and corners and their parents and children in the resulting geometric hierarchy.

4.1. Algorithm

We first decompose the triangles into a hierarchical structure as follows. Cartesian grid lines are overlaid on the domain, dividing it into $2^Q \times 2^Q$ blocks as the uniform case. Now triangles may fall partly in one of these areas and partly in another. So the contiguous blocks of faces are chosen by assigning a triangle to the block in which its centroid falls. The vertices of all the triangles in the block form the vertex set for that block.

In the quasi-uniform case the vertex classification is slightly more difficult because blocks may meet at a vertex which is shared by only 3 blocks instead of the consistent 4 blocks in the uniform case. This situation is illustrated in Fig. 9. To overcome this issue we introduce the notion of a *generalized corner*, which can be a group of vertices instead of a single one. This concept allows us to recover the regular relationship between segments and corners that we observed in the uniform setting where 4 segments meet at a corner. Now similar to the Cartesian case, the vertices $\mathcal{V}_{Q;i,j}$ can be classified into three types of elements:

- Facet element, which includes the vertices contained only in 1 block.
- Segment element, which includes the vertices on the border between 2 blocks.
- Generalized corner element, which includes the vertices shared by at least 3 blocks near a corner.

This definition can lead to *generalized corners* with more than 1 vertex, but at most a small number such as 3. Once this classification is available, we can define the elements $\mathcal{E}_{Q;i,j}$, the interior set $\mathcal{I}_{Q;i,j}$, and the boundary set $\mathcal{B}_{Q;i,j}$ as before. The relative sizes of segments and corners is not affected too much and the contribution of the corners to $\mathcal{B}_{Q;i,j}$ and $\mathcal{I}_{Q;i,j}$ is still much less than that of the segments. In Fig. 10 (left) we illustrate that 4 out of the 9 generalized corners contain more than one vertex.

Note that, though the number of vertices on a particular segment between two generalized corners may vary, the topological relationship between the four segments and four corners surrounding the facet is the same as in the aligned Cartesian case.

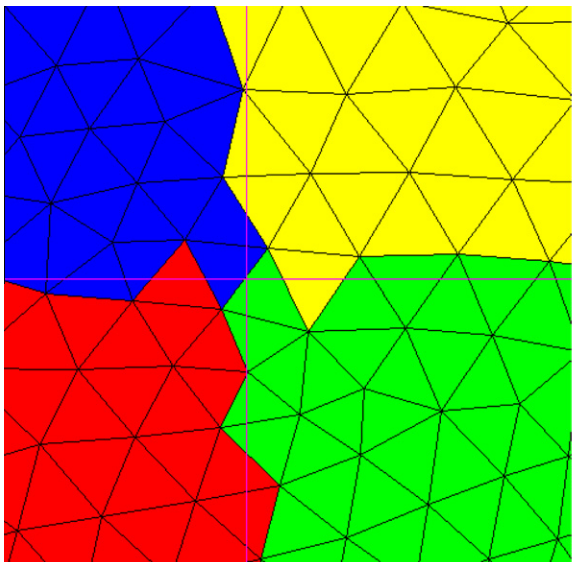


Fig. 9. Four blocks meeting in a generalized corner.

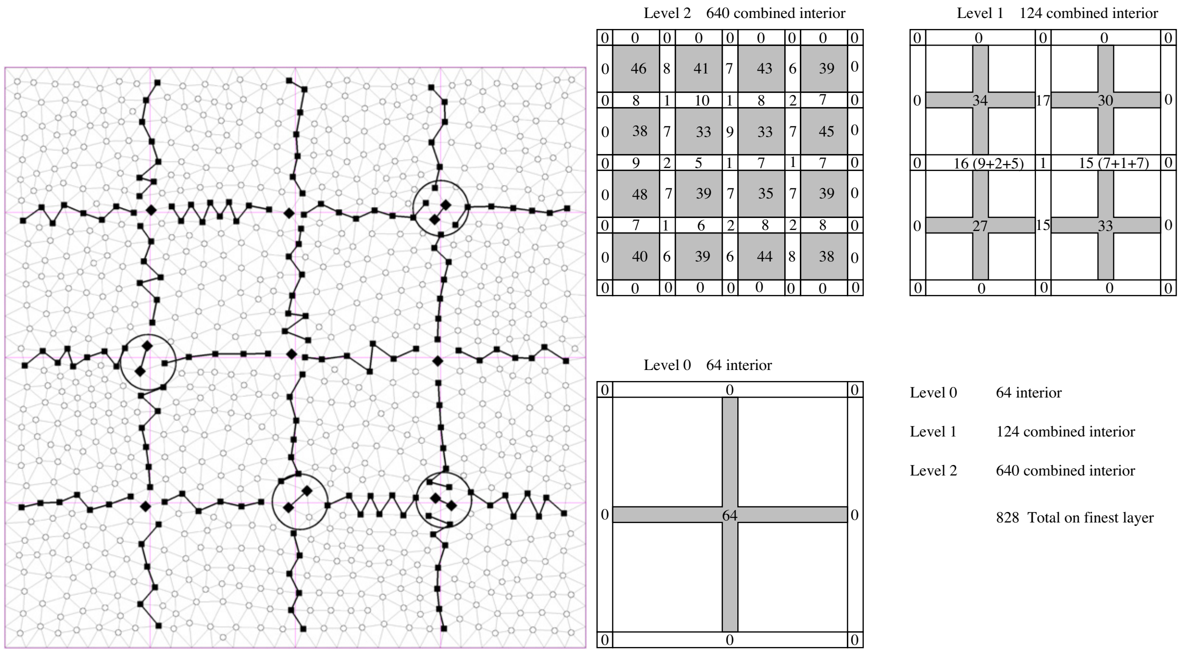


Fig. 10. Decomposition of a quasi-uniform mesh illustrating the use of generalized corners. Left: the block structure at the leaf level. Right: the sizes of $\mathcal{V}_{q;i,j}, \mathcal{I}_{q;i,j}$, and $\mathcal{B}_{q;i,j}$ at different levels.

Though there are alternative approaches to the domain decomposition that avoid the introduction of the cornering vertices, our scheme has the advantages of allowing the boundary between two blocks to be shared easily and leading to natural hierarchical groupings of segment-corner-segment. Other schemes can lead to double boundary layers and difficult groupings. We could also use a decomposition similar to the one in [8] where the domain is divided into two pieces each time, alternating between directions parallel to one axis and then the other. This leads to a tree of double the depth and the need to handle two different forms of lifting values from the matrices corresponding to one level to the level above (merging the parts in the Schur complement matrices and combining two child Schur complement matrices into the parent matrix). Once $\mathcal{E}_{Q;i,j}, \mathcal{I}_{Q;i,j}, \mathcal{B}_{Q;i,j}$ are available, we can define these sets for higher level blocks, similar to what has been done in the uniform Cartesian case.

While the sizes of the segments may vary the hierarchical relationship is still the same and all the main steps of the algorithm go through as before. Algorithms 1 and 2 depend only on the definition of the combinatorial relationship between the elements $\mathcal{E}_{q;i,j}$. Thus the block decomposition of the matrices and Algorithms 1 and 2 apply without major modification. The changes to be aware of include allowing for variable size corners (all corners in the uniform case have exactly size 1) and segments, and making the merging procedure more flexible.

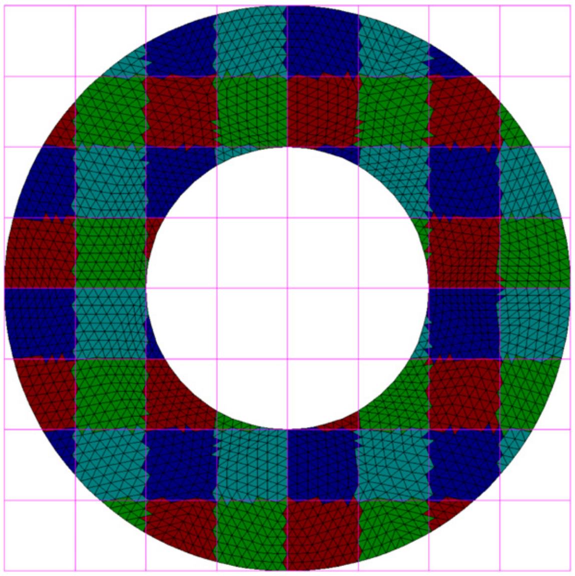
Now in the more general case, when the lengths of the segments are not equal, the analysis would be harder but to obtain the same complexity we need only have the average segment size on a level halve each time and the range of segment sizes be bounded by multiples of the average. This would ensure that our decomposition tree would remain the same sort of logarithmic depth and the ranks of the off-diagonal blocks will grow at the same sort of rate.

4.2. Numerical results

The first test on a quasi-uniform mesh on $[0,1]^2$ is for the equation $-\text{div}\,(a(\mathbf{x})\nabla u) = f$ where $a(\mathbf{x})$ is a constant on each triangle chosen independently and uniformly from $[10^{-2}, 10^2]$. The results with $\epsilon_a = 10^{-12}$ and $\epsilon_r = 10^{-6}$ are shown in Table

Table 7
Numerical results for a quasi-uniform mesh on $[0,1]^2$ using $\epsilon_a = 10^{-12}$ and $\epsilon_r = 10^{-6}$ for $-\text{div}\,(a(\mathbf{x})\nabla u) = f$ where $a(\mathbf{x})$ is chosen uniformly from $[10^{-2}, 10^2]$.

N	Setup	Solve	Error
11618	0.79	0.02	8.21e−07
46865	3.73	0.10	3.05e−06
188249	16.61	0.46	6.53e−06
754577	71.28	1.94	1.18e−05
3021473	306.40	8.43	2.17e−05



N	Setup	Solve	Error
134080	12.83	0.34	1.35e-6
538496	54.27	1.42	3.23e-6
2158336	240.80	5.92	6.12e-6
8642048	1055.56	25.24	1.29e-5

Fig. 11. Numerical results for a quasi-uniform mesh on an annulus using $\epsilon_a = 10^{-12}$ and $\epsilon_r = 10^{-6}$ for $-\Delta u = f$.

7. The runtime scaling is almost linear as expected and the error approximately doubles with each quadrupling of the problem size, as before. The increase of the error can be remedied easily by decreasing ϵ_r as N increases, as shown in the Cartesian case.

The second test is on a more general quasi-uniform mesh on an annulus for $-\Delta u = f$ using $\epsilon_a = 10^{-12}$ and $\epsilon_r = 10^{-6}$. The results are shown in Fig. 11 with similar scaling and error behavior. Notice that there are many small or empty blocks created by the uniform subdivision – we will show how to remedy this in the next section.

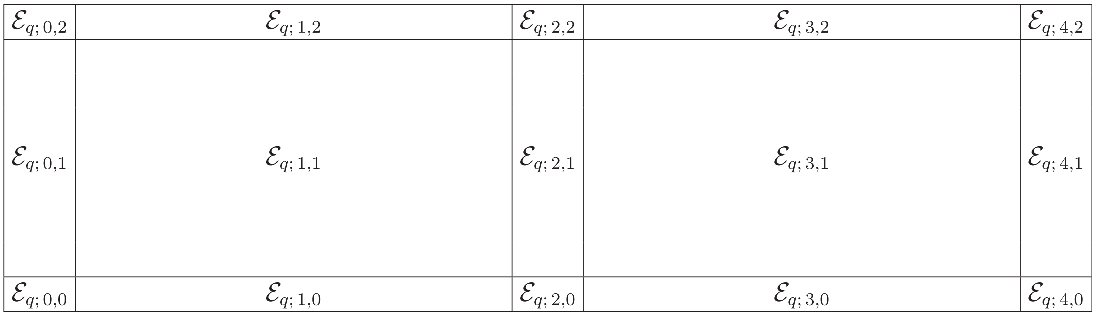
5. Adaptive decomposition

For more general domains this regular geometric subdivision method may be non-optimal, since some leaves will have fewer internal vertices than others, if the mesh is denser in some areas than others. It would increase efficiency to take these things into account when subdividing. In this section we generalize our approach to the setting of adaptive meshes.

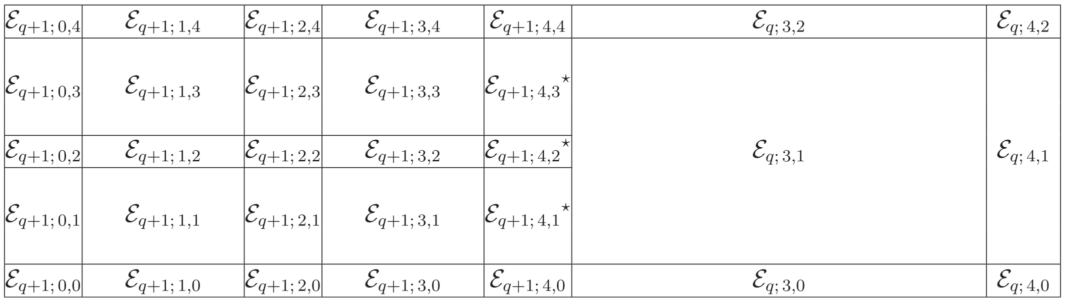
5.1. Domain decomposition procedure

In the uniform and quasi-uniform cases, the leaf level elements are determined first and the other elements are built from the bottom up. Now, since we do not know where and on what level we shall stop dividing, we have to work from the top down, dividing elements as required.

We start by specifying a constant which is the maximum number of vertices allowed in a leaf block. The square bounding box of the domain is divided into 4 equally sized pieces and every face is assigned to a different one of these blocks depending on the position of the centroid. The total number of vertices in (the faces in) each block is compared to the desired leaf size, and if greater the block is divided again. All the blocks on the same level are examined and divided as required. Once all these blocks have been visited the blocks in the next level are evaluated and divided if necessary. Eventually all the blocks will contain less than the desired amount of (non-boundary) vertices. Let us illustrate the division of leaf elements on level q into new leaf elements on level $q+1$ using the specific example with two neighboring blocks $\mathcal{D}_{q;0,0}$ and $\mathcal{D}_{q;1,0}$, which cover $\mathcal{E}_{q;i,j}$ (for $0 \leqslant i \leqslant 4$ and $0 \leqslant j \leqslant 2$) and share the elements $\mathcal{E}_{q;2,j}$ (for $0 \leqslant j \leqslant 2$):



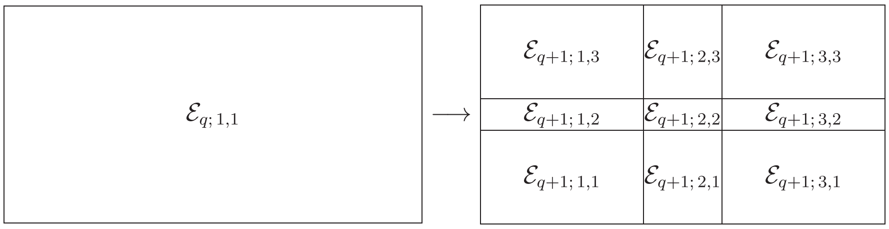
After dividing $\mathcal{D}_{q;0,0}$, we obtain:


$\mathcal{E}_{q+1;\,0,4}$
$\mathcal{E}_{q+1;\,1,4}$
$\mathcal{E}_{q+1;\,2,4}$
$\mathcal{E}_{q+1;\,3,4}$
$\mathcal{E}_{q+1;\,4,4}$
$\mathcal{E}_{q;\,3,2}$
$\mathcal{E}_{q;\,4,2}$
$\mathcal{E}_{q+1;\,0,3}$
$\mathcal{E}_{q+1;\,1,3}$
$\mathcal{E}_{q+1;\,2,3}$
$\mathcal{E}_{q+1;\,3,3}$
$\mathcal{E}_{q+1;\,4,3}^{\star}$
$\mathcal{E}_{q+1;\,0,2}$
$\mathcal{E}_{q+1;\,1,2}$
$\mathcal{E}_{q+1;\,2,2}$
$\mathcal{E}_{q+1;\,3,2}$
$\mathcal{E}_{q+1;\,4,2}^{\star}$
$\mathcal{E}_{q;\,3,1}$
$\mathcal{E}_{q;\,4,1}$
$\mathcal{E}_{q+1;\,0,1}$
$\mathcal{E}_{q+1;\,1,1}$
$\mathcal{E}_{q+1;\,2,1}$
$\mathcal{E}_{q+1;\,3,1}$
$\mathcal{E}_{q+1;\,4,1}^{\star}$
$\mathcal{E}_{q+1;\,0,0}$
$\mathcal{E}_{q+1;\,1,0}$
$\mathcal{E}_{q+1;\,2,0}$
$\mathcal{E}_{q+1;\,3,0}$
$\mathcal{E}_{q+1;\,4,0}$
$\mathcal{E}_{q;\,3,0}$
$\mathcal{E}_{q;\,4,0}$

Firstly, four new leaf corners at the lower level are inherited from the upper level:

$$\mathcal{E}_{q+1;0,0} = \mathcal{E}_{q;0,0} \quad \mathcal{E}_{q+1;4,0} = \mathcal{E}_{q;2,0} \quad \mathcal{E}_{q+1;0,4} = \mathcal{E}_{q;0,2} \quad \mathcal{E}_{q+1;4,4} = \mathcal{E}_{q;2,2}$$

The leaf facet $\mathcal{E}_{q;1,1}$ needs to be divided into 4 new facets, 4 segments and 1 corner at the center:


$\mathcal{E}_{q;\,1,1}$
$\longrightarrow$
$\mathcal{E}_{q+1;\,1,3}$
$\mathcal{E}_{q+1;\,2,3}$
$\mathcal{E}_{q+1;\,3,3}$
$\mathcal{E}_{q+1;\,1,2}$
$\mathcal{E}_{q+1;\,2,2}$
$\mathcal{E}_{q+1;\,3,2}$
$\mathcal{E}_{q+1;\,1,1}$
$\mathcal{E}_{q+1;\,2,1}$
$\mathcal{E}_{q+1;\,3,1}$

Now the new leaf blocks will have their own sets of internal vertices $\mathcal{I}_{q+1;0,0}, \ldots, \mathcal{I}_{q+1;1,1}$ and boundary vertices $\mathcal{B}_{q+1;0,0}, \ldots, \mathcal{B}_{q+1;1,1}$, so the facets are determined by

$$\begin{aligned}
\mathcal{E}_{q+1;1,1} &= \mathcal{I}_{q+1;0,0} \cap \mathcal{E}_{q;1,1}, \quad & \mathcal{E}_{q+1;3,1} &= \mathcal{I}_{q+1;1,0} \cap \mathcal{E}_{q;1,1}, \\
\mathcal{E}_{q+1;1,3} &= \mathcal{I}_{q+1;0,1} \cap \mathcal{E}_{q;1,1}, \quad & \mathcal{E}_{q+1;3,3} &= \mathcal{I}_{q+1;1,1} \cap \mathcal{E}_{q;1,1}.
\end{aligned}$$

For convenience, set:

$$\begin{aligned}
\mathcal{B}'_{q+1;0,0} &= \mathcal{B}_{q+1;0,0} \cap \mathcal{E}_{q;1,1}, \quad & \mathcal{B}'_{q+1;1,0} &= \mathcal{B}_{q+1;1,0} \cap \mathcal{E}_{q;1,1}, \\
\mathcal{B}'_{q+1;0,1} &= \mathcal{B}_{q+1;0,1} \cap \mathcal{E}_{q;1,1}, \quad & \mathcal{B}'_{q+1;1,1} &= \mathcal{B}_{q+1;1,1} \cap \mathcal{E}_{q;1,1}.
\end{aligned}$$

These are the parts of the boundaries of the new leaves that are inside $\mathcal{E}_{q;1,1}$ which will determine the new leaf elements. Then intersecting 3 at a time and taking the union (recall that our generalized corner is given where more than 2 blocks meet, and they will meet on their common boundary layers – there are 4 ways to pick 3 blocks to test and so we need to take the union of the 4 possible intersection results):

$$\begin{aligned}
\mathcal{E}_{q+1;2,2} = {} & \left(\mathcal{B}'_{q+1;0,0} \cap \mathcal{B}'_{q+1;1,0} \cap \mathcal{B}'_{q+1;0,1}\right) \quad \cup \quad \left(\mathcal{B}'_{q+1;0,0} \cap \mathcal{B}'_{q+1;1,0} \cap \mathcal{B}'_{q+1;1,1}\right) \quad \cup \quad \left(\mathcal{B}'_{q+1;0,0} \cap \mathcal{B}'_{q+1;0,1} \cap \mathcal{B}'_{q+1;1,1}\right) \\
& \cup \quad \left(\mathcal{B}'_{q+1;1,0} \cap \mathcal{B}'_{q+1;0,1} \cap \mathcal{B}'_{q+1;1,1}\right),
\end{aligned}$$

we can define the new central corner. From there the 4 new leaf segments between the new leaf facets will be determined, since we want those vertices where the 2 new leaf blocks meet along their common boundary but wish to exclude the central corner they may share. Thus:

$$\mathcal{E}_{q+1;2,1} = \left(\mathcal{B}'_{q+1;0,0} \cap \mathcal{B}'_{q+1;1,0}\right) \setminus \mathcal{E}_{q+1;2,2}, \quad \mathcal{E}_{q+1;2,3} = \left(\mathcal{B}'_{q+1;0,1} \cap \mathcal{B}'_{q+1;1,1}\right) \setminus \mathcal{E}_{q+1;2,2}$$

and

$$\mathcal{E}_{q+1;1,2} = \left(\mathcal{B}'_{q+1;0,0} \cap \mathcal{B}'_{q+1;0,1}\right) \setminus \mathcal{E}_{q+1;2,2}, \quad \mathcal{E}_{q+1;1,2} = \left(\mathcal{B}'_{q+1;1,0} \cap \mathcal{B}'_{q+1;1,1}\right) \setminus \mathcal{E}_{q+1;2,2}.$$

To determine how the segment $\mathcal{E}_{q;2,1}$ is divided into new leaf elements $\mathcal{E}_{q+1;4,1} \uplus \mathcal{E}_{q+1;4,2} \uplus \mathcal{E}_{q+1;4,3}$ we first determine the corner that will be created at the middle of the old segment:

$$\mathcal{E}_{q+1;4,2} = (\mathcal{B}_{q+1;1,0} \cap \mathcal{B}_{q+1;1,1}) \cap \mathcal{E}_{q;2,1},$$

then the new leaf segments above and below will be:

$$\mathcal{E}_{q+1;4,1} = (\mathcal{B}_{q+1;1,0} \cap \mathcal{E}_{q;2,1}) \setminus \mathcal{E}_{q+1;4,2} \quad \text{and} \quad \mathcal{E}_{q+1;4,3} = (\mathcal{B}_{q+1;1,0} \cap \mathcal{E}_{q;2,1}) \setminus \mathcal{E}_{q+1;4,2}.$$

The breakdown of the other 3 segments on the sides of $\mathcal{E}_{q;1,1}$ is similar.

A complication arises because, since segments are shared between neighboring blocks, two blocks may arrive at a different decomposition of the parent segment into segment-corner-segment. So the segment-corner-segment group in the middle between the two blocks, the elements $\mathcal{E}_{q+1;4,1}, \mathcal{E}_{q+1;4,2}$ and $\mathcal{E}_{q+1;4,3}$, has been highlighted with $\star$ and # since these elements are only completely determined by the block divisions on one side if the block on the other side is never further divided.

$\mathcal{E}_{q;0,2}$	$\mathcal{E}_{q;1,2}$	$\mathcal{E}_{q+1;4,4}$	$\mathcal{E}_{q+1;5,4}$	$\mathcal{E}_{q+1;6,4}$	$\mathcal{E}_{q+1;7,4}$	$\mathcal{E}_{q;8,4}$
		$\mathcal{E}_{q+1;4,3}{}^{\#}$	$\mathcal{E}_{q+1;5,3}$	$\mathcal{E}_{q+1;6,3}$	$\mathcal{E}_{q+1;7,3}$	$\mathcal{E}_{q;8,3}$
$\mathcal{E}_{q;0,1}$	$\mathcal{E}_{q;1,1}$	$\mathcal{E}_{q+1;4,2}{}^{\#}$	$\mathcal{E}_{q+1;5,2}$	$\mathcal{E}_{q+1;6,2}$	$\mathcal{E}_{q+1;7,2}$	$\mathcal{E}_{q;8,2}$
		$\mathcal{E}_{q+1;4,1}{}^{\#}$	$\mathcal{E}_{q+1;5,1}$	$\mathcal{E}_{q+1;6,1}$	$\mathcal{E}_{q+1;7,1}$	$\mathcal{E}_{q;8,1}$
$\mathcal{E}_{q;0,0}$	$\mathcal{E}_{q;1,0}$	$\mathcal{E}_{q+1;4,0}$	$\mathcal{E}_{q+1;5,0}$	$\mathcal{E}_{q+1;6,0}$	$\mathcal{E}_{q+1;7,0}$	$\mathcal{E}_{q;8,0}$

So we may, for example, find $\mathcal{E}^{\star}_{q+1;4,1} \neq \mathcal{E}^{\#}_{q+1;4,1}$. To resolve this, if another decomposition already exists we intersect the two tentative segments as follows:

$$\mathcal{E}_{q+1;4,1} = \mathcal{E}^{\star}_{q+1;4,1} \cap \mathcal{E}^{\#}_{q+1;4,1} \quad \text{and} \quad \mathcal{E}_{q+1;4,3} = \mathcal{E}^{\star}_{q+1;4,3} \cap \mathcal{E}^{\#}_{q+1;4,3}$$

to form the new leaf segments. The middle corner is then found from:

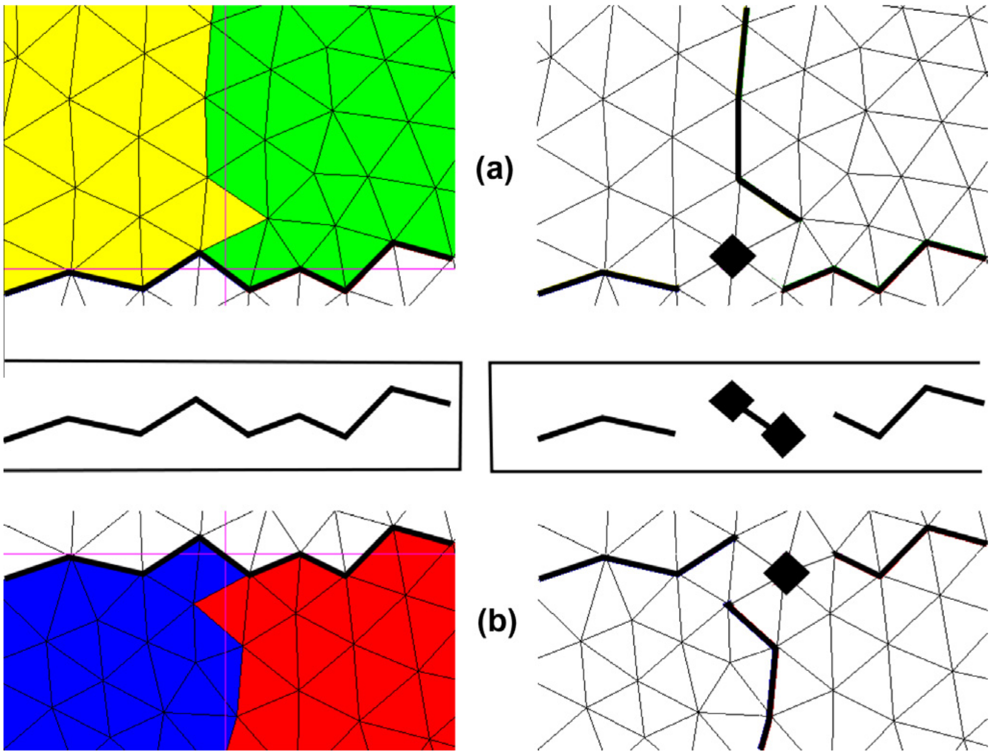


Fig. 12. The common segment (left) on the border between the two sides (a) and (b) is divided differently from the top and from the bottom, this division is reconciled by reducing the length of the child segments and increasing the central corner to 2 vertices.

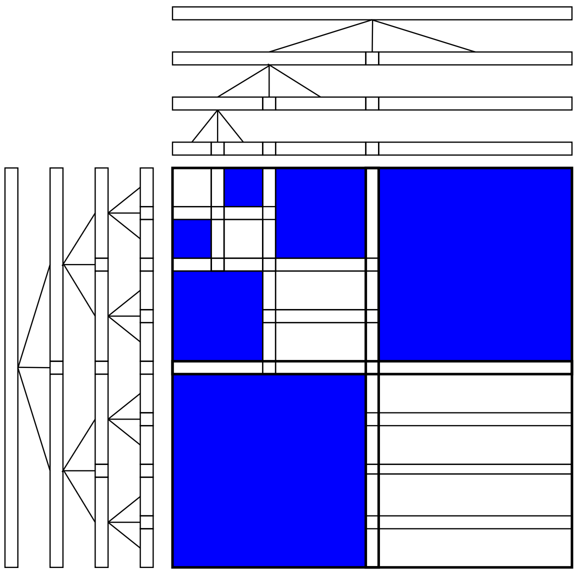


Fig. 13. Hierarchical matrix representing the interaction between segments with different subdivisions.

Table 8
Setup times for a non-uniform square using $\epsilon_a = 10^{-12}$ and $\epsilon_r = 10^{-6}$ for $-\Delta u = f$. The first 2 sets are from the non-adaptive approach with variable maximum leaf size, and the second 4 from the adaptive approach with indicated maximum leaf size.

N	Finer			Coarser			Adaptive			
	Q	Max leaf	Setup	Q	Max leaf	Setup	Maximum Leaf Size			
							200	300	350	400
14689	5	77	2.23	4	239	1.12	0.83	0.76	0.78	0.79
59201	6	86	10.52	5	269	5.61	4.03	3.62	3.71	3.78
237697	7	88	47.66	6	291	26.18	18.72	16.46	16.19	16.66
952577	8	89	206.89	7	303	116.50	85.45	76.58	73.11	74.32
3813889	9	90	897.74	8	309	514.67	386.87	347.38	308.77	335.34

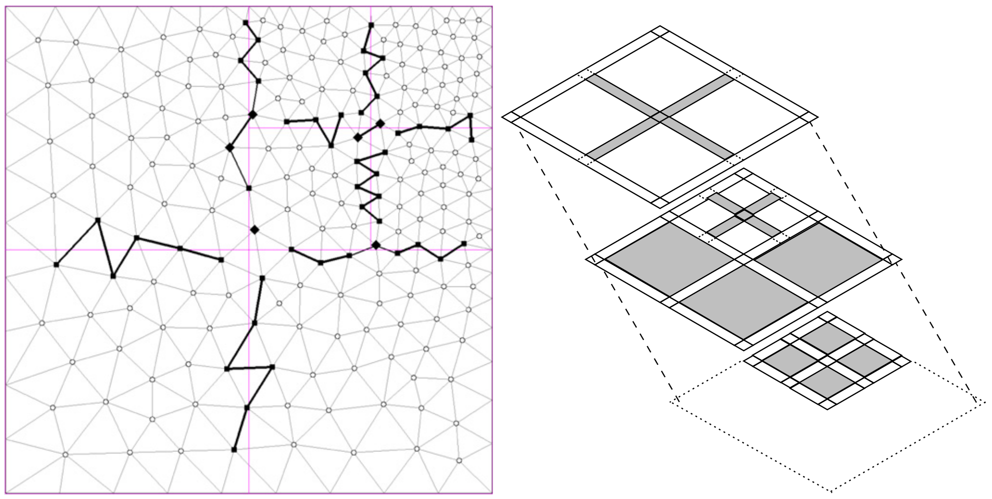


Fig. 14. An adaptive decomposition of a mesh showing leaf blocks on various levels.

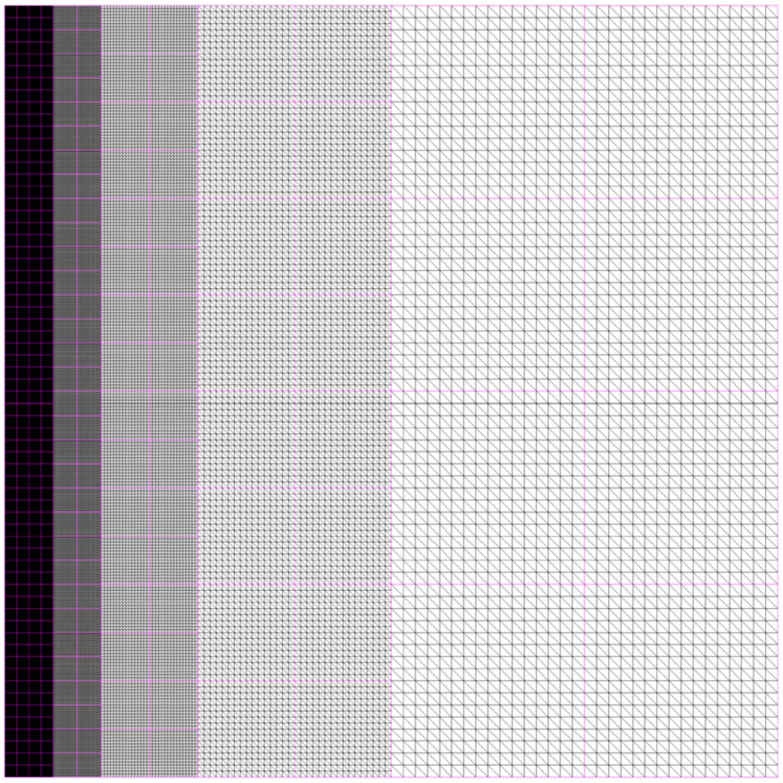
$$\mathcal{E}_{q+1;4,2} = \mathcal{E}_{q;2,1} \setminus \left(\mathcal{E}_{q+1;4,1} \uplus \mathcal{E}_{q+1;4,3}\right),$$

which has the effect of possibly increasing the size of this element:

$$\mathcal{E}_{q+1;4,2} \supseteq \mathcal{E}^{\star}_{q+1;4,2} \cup \mathcal{E}^{\#}_{q+1;4,2}.$$

This approach is clearly motivated by the idea of the generalized corner introduced in Section 4. Now all four affected child blocks which meet at this corner have consistent boundary elements. In Fig. 12 we show part of a mesh illustrating this phenomenon.

Once the adaptive decomposition is complete, elements occur at all levels, some of which are the boundaries of blocks which have since been divided, i.e. they are not leaf elements and have children. These are deleted, leaving only a consistent decomposition of all the vertices into leaf elements (which may not have similar sizes or depths in the tree). Then, as in the



		Uniform					Adaptive	
		Leaves					Leaf 289	
N	Refined	Min	Max	Q	Setup	Solve	Setup	Solve
16129		25	25	5	2.20	0.03	1.07	0.07
40386	1/2	25	81	5	2.98	0.08	2.67	0.18
88963	1/4	25	289	5	6.48	0.30	6.05	0.41
186180	1/8	9	289	6	16.46	0.67	13.00	0.87
380677	1/16	4	289	7	44.18	1.40	26.78	1.77
769734	1/32	4	1089	7	259.75	9.70	57.61	3.85
1547911	1/64	4	4225	7	5923.37	-	111.63	7.64
3104328	1/128	-	-	-	-	-	223.82	15.21

Fig. 15. Top: Uniform mesh on $[0,1]^2$ selectively refined on $[0,x] \times [0,1]$ to produce a large range of triangle size and density. Bottom: Numerical results for different values of N.

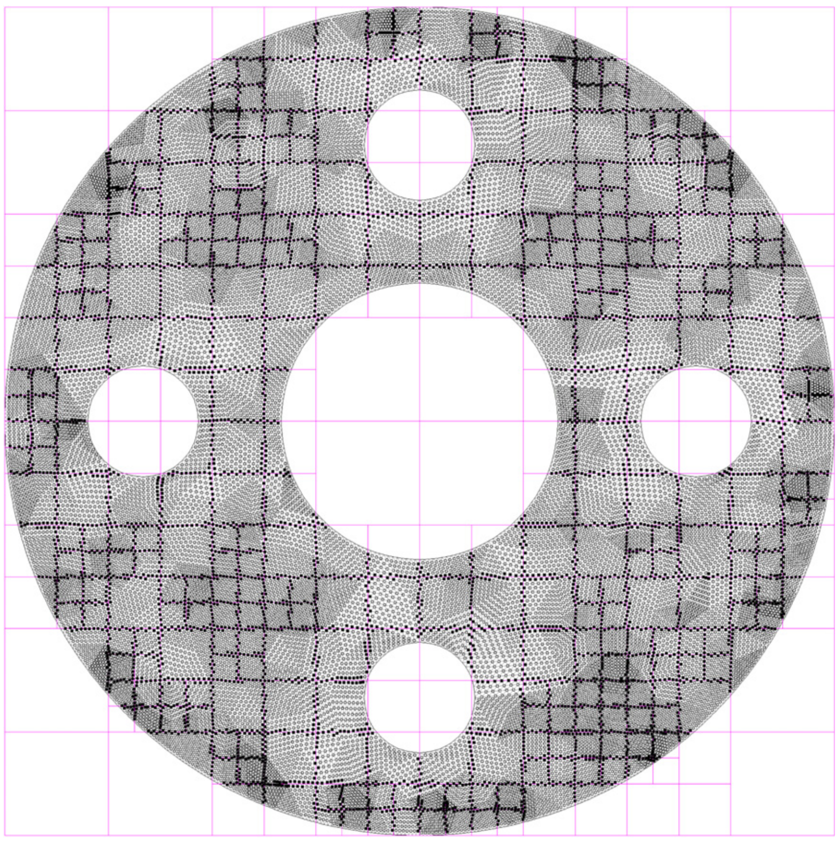
non-adaptive case, the segments and corners on all levels above the leaves are built up by taking ordered sequences of vertices for the child elements and merely copying them in the case of parent corners, or concatenating them to form the segment-corner-segment structure of the parent segment.

The factorized form of M in the adaptive case has a similar structure to the uniform case except that each level is no longer full and leaves, with their associated dense matrices, can occur on various levels. Again we proceed level by level starting from the deepest occupied level Q, and construct the products $L_q := \prod_{i,j} L_{q;i,j}$ and the factorization:

$$M = L_Q \begin{pmatrix} A_Q & \\ & S_Q \end{pmatrix} L_Q^t = L_Q L_{Q-1} \begin{pmatrix} A_Q & & \\ & A_{Q-1} & \\ & & S_{Q-1} \end{pmatrix} L_{Q-1}^t L_Q^t,$$

where $A_Q : \mathcal{I}_Q \to \mathcal{I}_Q, A_{Q-1} : \mathcal{I}_{Q-1} \to \mathcal{I}_Q$, and $S_{Q-1} : (\mathcal{I}_Q \uplus \mathcal{I}_{Q-1})^c \to (\mathcal{I}_Q \uplus \mathcal{I}_{Q-1})^c$. The remaining vertices in the block decomposition are no longer simply the boundary of level $Q-1$ but the whole domain is still the disjoint union of all the $\mathcal{I}_q$ and we get as before:

$$M = L_Q L_{Q-1} \ldots L_1 \begin{pmatrix} A_Q & & & & \\ & A_{Q-1} & & & \\ & & \ddots & & \\ & & & A_1 & \\ & & & & A_0 \end{pmatrix} L_1^t \ldots L_{Q-1}^t L_Q^t,$$



	Uniform $Q = 7$			Adaptive, leaf< 100		
N	Setup	Solve	Error	Setup	Solve	Error
442108	66.28	1.47	4.20e-06	69.59	1.58	4.70e-06
1773052	345.70	6.98	1.01e-05	298.11	7.35	1.00e-05
7101436	1335.54	29.19	2.22e-05	1194.24	29.47	2.19e-05

Fig. 16. Setup and Solve times for a punctured annulus mesh using $\epsilon_a = 10^{-12}$ and $\epsilon_r = 10^{-6}$ for $-\Delta u = f$. In the second set of results the adaptive decomposition was used.

where $A_q : \mathcal{I}_q \rightarrow \mathcal{I}_q$. As before, the matrices A_q are block diagonal once we collect all the non-empty $\mathcal{I}_{q;i,j}$ in a given order for each level q.

In the adaptive case it may happen that two segments may not be subdivisible to the same degree, and so the corresponding submatrices will be subdivided further with respect to the rows than columns (or *vice versa*) as illustrated in Fig. 13. This means that our choice of hierarchical matrix structure does not follow a common pattern among all the child $S_{q+1;i',j'}$ matrices that we wish to combine into the parent $M_{q;i,j}$ matrix. Thus when we perform the multiplications required for the Schur complement we may have to merge and split blocks so that the two hierarchical matrices we wish to multiply become compatible.

5.2. Complexity

The complexity analysis in the adaptive case is much harder because we have little control over how segment sizes are distributed and how deep the decomposition tree may be. The cost of hierarchical matrix multiplication depends on the number of steps in the hierarchical decomposition. If this grows linearly instead of logarithmically with N we expect worse performance. The actual scaling will depend on the depth of the tree in various areas and how those trees are combined. Our numerical experiments suggest that the algorithm will retain linear or almost linear scaling as long as the difference between the least and most refined areas of the mesh is no more than 10 steps which seems reasonable for most meshes. Further investigation may reveal the precise characteristics of meshes retaining good performance.

Similarly the cost of the actual decomposition process depends on the number of times a vertex appears before the leaf level and how densely populated the boundaries of the decomposition regions are compared to the interiors (each vertex has to be checked on every level it appears and the set manipulations cost $s\log s$ for sets of size s). This would usually lead to $\mathcal{O}(N\log N)$ or $\mathcal{O}(N\log^2 N)$ but if the mesh density increases too quickly near a particular point there may only be a constant number of vertices in the leaves at each level leading to a very deep decomposition and worse scaling. If this becomes a problem in practice the cost could be amortized over potentially numerous calculations on the same mesh or consolidated with an h-adaptive scheme. By integrating with the mesh subdivision process one could use the added information about the distribution of the newly created vertices to choose the two dividing lines to approximately equalize the number of vertices in each of the 4 resulting subblocks instead of simply choosing the midpoints to obtain 4 equal areas.

5.3. Numerical results

While the observed scaling remains very similar, the actual runtime depends on the choice of leaf size. Increasing leaf size (and memory consumption) improves runtime up to a certain point, and then the dense calculations start to dominate and runtime increases again. In Table 8 we show the results of the uniform and adaptive approach to solving $-\Delta u = f$ using $\epsilon_a = 10^{-12}$ and $\epsilon_r = 10^{-6}$ for a non-uniform square similar to the one in Fig. 14. The runtime scaling and value is the best for the adaptive approach with maximum leaf size 350 while for our largest example the comparable uniform and adaptive versions, with respective maximum leaf sizes 309 and 300, demonstrate a decrease of about 1/3 in runtime from 514.67 to 347.38 using the adaptive approach.

Another type of mesh for which the adaptive approach is suited is one that has been selectively refined and the ratio of largest to smallest triangles is quite large such as in the case illustrated in Fig. 15. Note there is almost perfectly linear scaling in the adaptive case as N increases from 1547911 to 3104328 while the uniform version leads to very large maximum leaves and a huge jump in runtime for those problems that did not exhaust the available memory.

Finally, in Fig. 16 we compare the adaptive and uniform approaches for a non simply-connected domain for $-\Delta u = f$ with the usual $\epsilon_a = 10^{-12}$ and $\epsilon_r = 10^{-6}$. The adaptive approach has a smaller advantage here because the triangle size is relatively uniform and only the distribution needs to be accommodated.

6. Conclusion and future work

We have presented a fast algorithm for approximate solutions of the large sparse linear systems arising from elliptic equations via the finite element method. This algorithm is asymptotically linear in runtime and memory requirements. An explicit procedure for dealing with general, quasi-uniform meshes was described, as well as an adaptive decomposition method that offers improved performance.

We have only demonstrated the approach for piecewise linear elements but one could generalize the approach to work with different discretizations such as spectral elements or different methods such as the discontinuous Galerkin [19] method. One could incorporate our adaptive decomposition method and calculation of the solution into an h-adaptive mesh refinement system [20] as in [21] so that new degrees of freedom could be incorporated incrementally, where only the parts of the mesh which have been refined (and their parents in the decomposition tree) would require new calculations. Similarly one could incrementally take into account the degrees of freedom added and removed via a p-adaptive system, and eventually hp-adaptive systems.

These ideas can be extended to other finite element bases as long as the support of the basis elements is localized. The boundary layer between blocks might have to be enlarged to ensure that there is no interaction between the internal vertices at the leaf level when the stencil support grows larger than the one neighborhood. The approach can also be extended to non-symmetric matrices (a simple test solving $-\Delta u + b(\mathbf{x}) \cdot \nabla u = f$ using slightly modified algorithms produced similar results to those for $-\Delta u = f$). One could extend to a tetrahedral mesh in three dimensions but the boundaries between the "cubes" of tetrahedra would have to be carefully managed.

We have not discussed parallelization of the calculations [22], but since all of the calculations on the same level are independent they (as well as the underlying block matrix multiplications) could be performed in parallel. Only once there are fewer blocks per level than processors would extensive inter-processor communication be required. Large scale parallel multifrontal solvers such as MUMPS [23] illustrate the possible gains of parallelization.

Another situation where the algorithm could be adjusted to improve performance is where $a(\mathbf{x})$ and/or $V(\mathbf{x})$ is perturbed locally and repeated calculations are required – a calculated factorization could be largely reused as only the parents of the blocks containing the vertices with changed values would have to be recalculated.

Acknowledgements

Phillip G. Schmitz and Lexing Ying are partially supported by the NSF Grant DMS-0846501. Lexing Ying is also partially supported by a Sloan Research Fellowship.

References

[1] T.A. Davis, Direct Methods for Sparse Linear Systems, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2006.
[2] Y. Saad, Iterative Methods for Sparse Linear Systems, second ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003.
[3] P.R. Amestoy, T.A. Davis, I.S. Duff, An approximate minimum degree ordering algorithm, SIAM Journal on Matrix Analysis and Applications 17 (1996) 886–905.
[4] B. Hendrickson, E. Rothberg, Improving the run time and quality of nested dissection ordering, SIAM Journal on Scientific Computing 20 (1998) 468–489.
[5] J.A. George, Nested dissection of a regular finite element mesh, SIAM Journal on Numerical Analysis 10 (1973) 345–363.
[6] I.S. Duff, J.K. Reid, The multifrontal solution of indefinite sparse symmetric linear equations, ACM Transactions on Mathematical Software 9 (1983) 302–325.
[7] J.W.H. Liu, The multifrontal method for sparse matrix solution: theory and practice, SIAM Review 34 (1992) 82–109.
[8] J. Xia, S. Chandrasekaran, M. Gu, X.S. Li, Superfast multifrontal method for large structured linear systems of equations, SIAM Journal on Matrix Analysis and Applications 31 (2009) 1382–1411.
[9] J. Xia, S. Chandrasekaran, M. Gu, X.S. Li, Fast algorithms for hierarchically semiseparable matrices, Numerical Linear Algebra with Applications (2009).
[10] W. Hackbusch, L. Grasedyck, S. Börm, An Introduction to Hierarchical Matrices, Technical Report 21, Max-Plank-Instituit für Mathematik in den Naturwissenschaften, Leipzig, 2001.
[11] L. Grasedyck, W. Hackbusch, Construction and arithmetics of $\mathcal{H}$-matrices, Computing 70 (2003) 295–334.
[12] P.-G. Martinsson, A fast direct solver for a class of elliptic partial differential equations, Journal of Scientific Computing 38 (2009) 316–330.
[13] P.G. Martinsson, V. Rokhlin, A fast direct solver for boundary integral equations in two dimensions, Journal of Computational Physics 205 (2005) 1–23.
[14] L. Greengard, D. Gueyffier, P.-G. Martinsson, V. Rokhlin, Fast direct solvers for integral equations in complex three-dimensional domains, Acta Numerica 18 (2009) 243–275.
[15] S. Börm, Approximation of solution operators of elliptic partial differential equations by $\mathcal{H}$- and cal H^2-matrices, Numerische Mathematik 115 (2010) 165–193.
[16] E. Liberty, F. Woolfe, P.-G. Martinsson, V. Rokhlin, M. Tygert, Randomized algorithms for the low-rank approximation of matrices, Proceedings of the National Academy of Sciences of the USA 104 (2007) 20167–20172.
[17] J. Xia, Robust and efficient multifrontal factorization for large discretized PDEs, in: M. Berry et al. (Eds.), High Performance Scientific Computing: Algorithms and Applications, Springer, Berlin, 2011. Proceedings of a Conference Held, October 11–12, 2010 Purdue University, West Lafayette, Indiana, USA.
[18] L. Grasedyck, R. Kriemann, S. Le Borne, Parallel black box $\mathcal{H}$-LU preconditioning for elliptic boundary value problems, Computing and Visualization in Science 11 (2008) 273–291.
[19] D.N. Arnold, F. Brezzi, B. Cockburn, L.D. Marini, Unified analysis of discontinuous Galerkin methods for elliptic problems, SIAM Journal on Numerical Analysis 39 (2001/02) 1749–1779.
[20] L. Demkowicz, J. Kurtz, D. Pardo, M. Paszyński, W. Rachowicz, A. Zdunek, Computing with hp-adaptive finite elements, Frontiers: Three-dimensional Elliptic and Maxwell Problems with Applications, Chapman & Hall/CRC Applied Mathematics and Nonlinear Science Series, vol. 2, Chapman & Hall/CRC, Boca Raton, FL, 2008.
[21] L. Grasedyck, W. Hackbusch, S. Le Borne, Adaptive geometrically balanced clustering of $\mathcal{H}$-matrices, Computing 73 (2004) 1–23.
[22] L. Lin, C. Yang, J. Lu, L. Ying, W. E, A Fast Parallel Algorithm for Selected Inversion of Structured Sparse Matrices with Applications to 2D Electronic Structure Calculations, Technical Report, LBNL-2677E, Lawrence Berkeley National Lab, 2009.
[23] P.R. Amestoy, I.S. Duff, C. Vömel, Task scheduling in an asynchronous distributed memory multifrontal solver, SIAM Journal on Matrix Analysis and Applications 26 (2004/2005) 544–565.