Journal of Computational Physics 258 (2014) 227–245

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

A fast nested dissection solver for Cartesian 3D elliptic problems using hierarchical matrices

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

[a] Department of Mathematics, The University of Texas at Austin, United States
[b] Department of Mathematics and ICME, Stanford University, United States

ARTICLE INFO

Article history:
Received 18 July 2012
Received in revised form 21 September 2013
Accepted 17 October 2013
Available online 24 October 2013

Keywords:
Elliptic equations
Fast algorithms
Nested dissection
Hierarchical matrices
Sparse matrix

ABSTRACT

We present a fast algorithm for solutions to linear systems arising from three dimensional elliptic problems on a regular Cartesian mesh. We follow the approach of Schmitz and Ying (2012) on combining the nested dissection matrix factorization method with hierarchical matrices in two dimensions and extend it to the three dimensional case. A theoretical linear time complexity is derived and a more practical variant with slightly worse scaling is demonstrated.

1. Introduction

In this paper we will consider the solution of an elliptic problem of form

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

on a Cartesian domain Ω in $\mathbb{R}^3$.

There are two main classes of solvers for sparse linear systems arising from elliptic partial different equations: iterative [1] and direct [2] methods. Iterative methods are arguably more competitive. However, the number of iterations to convergence can be quite large for problems where the coefficients $a(\mathbf{x})$ exhibit large contrast (large variations in magnitude for different areas of the domain) and sudden changes.

Direct methods have a much more predictable computation cost. However, in order to be efficient, one has to choose a reordering that reduces fill-in of non-zeros in the factors. Finding the optimal ordering (especially for matrices arising from problems in three dimensions) typically requires graph-theoretic approaches such as the (approximate) minimum degree algorithm [3] while a reasonably good reordering can be determined using hierarchical partitioning [4]. In three dimensions the most efficient direct methods, such as the nested dissection methods [2,5–7], have a factorization cost that scales like $\mathcal{O}(N^2)$ where N is the number of degrees of freedom.

Recently, it has been observed in [8,9] that fill-in blocks of the LDL^t factorization are highly compressible using the H-matrix [10] or hierarchical semiseparable matrix frameworks [11] and thus the calculations can be done much more efficiently. In two dimensions Xia et al. [8] showed how to combine the nested dissection method with hierarchical semiseparable matrices and we have recently used a similar approach in [12] where nested dissection with hierarchical matrix

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

0021-9991/$ – see front matter
http://dx.doi.org/10.1016/j.jcp.2013.10.030

algebra for the frontal matrix was applied to the general case of unstructured and adaptive meshes. This paper will show how to extend this line of work to three dimensional problems while maintaining almost linear $\mathcal{O}(N)$ complexity. When only single digit accuracy is required, the resulting algorithm can be used as a direct solver for quite general elliptic problems, as demonstrated by the numerical results. When better accuracy is desired, our approach can serve as highly efficiently preconditioner when combined with standard iterative solvers.

We would like to mention that fast solvers of this type have also been developed for linear systems for boundary integral equations. To name a few examples, 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 unavailable both in 2D and 3D.

The rest of the paper is organized as follows. Section 2 shows the hierarchical domain decomposition used in this paper. We describe in Section 3 the nested dissection factorization based on this domain decomposition and show how that achieves the $\mathcal{O}(N^2)$ complexity. In Section 4 we show how hierarchical matrix algebra is used to improve the algorithm by replacing the usual matrix representation and operations with the more efficient hierarchical ones. The complexity of our algorithm is discussed in Section 5 for two different approaches to determining the structure of the hierarchical matrices used. Finally, numerical results are provided in Section 6 to demonstrate the efficiency of both approaches.

2. Domain decomposition

To simplify the discussion, we assume that the domain Ω is $[0,1]^3$. A uniform Cartesian grid of size $m \times m \times m$ is introduced to discretize the problem. The hierarchical domain decomposition used here is essentially the octree structure used in most nested dissection algorithms. However, we describe this decomposition in some detail since we impose additional hierarchical structure in order to lay the foundation for the hierarchical structure placed on the frontal matrix, which is coupled to the hierarchical domain decomposition.

Here, we assume that $m = P2^Q + 1$ where P is a positive integer of $\mathcal{O}(1)$ and Q is also an integer which will turn out to be the depth of the hierarchical decomposition. Out of the $m^3 = (P2^Q + 1)^3$ nodes in total, some lie on the boundary of the domain and there are $N = (P2^Q - 1)^3$ nodes in the interior.

Eq. (1) is often discretized with a finite difference or a finite element scheme. In the finite element case, each cell of our Cartesian grid is subdivided into 6 tetrahedra. Based on this tetrahedral mesh, we can define piecewise linear basis functions $\{\phi_\kappa(\mathbf{x})\}$, one for each interior node κ. Each basis function takes the value 1 at the node κ and zero at the rest. The discrete version of (1) under this discretization is given by

$$Mu = f,$$

where u and f are vectors formed by the combination of all the u_κ and f_κ, and M is a sparse matrix with values given by

$$(M)_{\kappa\lambda} = \int_{[0,1]^3} \nabla\phi_\kappa(\mathbf{x}) \cdot a(\mathbf{x}) \nabla\phi_\lambda(\mathbf{x}) + V(\mathbf{x})\phi_\kappa(\mathbf{x})\phi_\lambda(\mathbf{x})\, d\mathbf{x}. \tag{2}$$

Due to the locality of the piecewise linear finite element functions, $(M)_{\kappa\lambda}$ is non-zero only if the nodes κ and λ are adjacent to each other.

2.1. Leaf level

We divide the domain Ω into 2^{3Q} subdomains at level Q, each labeled by

$$\mathcal{D}_{Q;i,j,k} = \left[\frac{i}{2^Q}, \frac{i+1}{2^Q}\right] \times \left[\frac{j}{2^Q}, \frac{j+1}{2^Q}\right] \times \left[\frac{k}{2^Q}, \frac{k+1}{2^Q}\right],$$

for $0 \leqslant i, j, k < 2^Q$ using $2^Q + 1$ equally spaced planes at coordinate value $\frac{0}{2^Q}, \frac{1}{2^Q}, \frac{2}{2^Q}, \ldots, \frac{2^Q}{2^Q}$ in each dimension. Some nodes will lie on the boundary of the subdomains and will be shared with their neighbors. The set of $(P+1) \times (P+1) \times (P+1)$ nodes (at the domain boundary subdomains may however contain fewer nodes) in each subdomain $\mathcal{D}_{Q;i,j,k}$ will be denoted by

$$\mathcal{V}_{Q;i,j,k}, \quad \text{where } 0 \leqslant i, j, k < 2^Q.$$

Some nodes will be contained solely within one subdomain and will be in the subdomain's *interior*, while those shared with other subdomains will be on the subdomain's *boundary*. If a node's position is divisible by P in some direction it will be on a subdomain boundary otherwise it will be in a subdomain interior. The nodes in the interior of one subdomain $\mathcal{D}_{Q;i,j,k}$ do not contribute via M to the values in the interior of another subdomain. Since the entry $M_{\kappa\lambda}$ is non-zero only when κ and λ are adjacent to each other, the boundary layer between subdomains is sufficient to decouple the nodes in the interior of each subdomain at level Q from those in another subdomain at the same level.

Another way of looking at the set of nodes that make up each subdomain can be obtained if we divide the $P2^Q + 1$ nodes $0, 1, 2, \ldots, P2^Q$ in the x-direction into $2^{Q+1} + 1$ groups of alternating sizes 1 and $P - 1$, that is node 0 in group 0,

nodes $1,\dots,P-1$ in group 1, node P in group 2, nodes $P+1,\dots,2P-1$ in group 3, until eventually node $2^Q P$ is in group 2^{Q+1}. That is, all the nodes whose position X is a multiple of P are in the even groups and the other nodes in between are in the odd groups. Now perform the same division into these alternating groups in the y- and z-directions.

A node is in even or odd groups for a particular direction depending on the divisibility of that node's position by P. Thus every node will be in a group (i,j,k) for $0 \leqslant i,j,k \leqslant 2^{Q+1}$ where i is the group from the x-direction grouping, j the group from the y-direction and k the group from the z-direction. These different sets of nodes on the last level Q are called *leaf elements* and labeled by

$$\mathcal{E}_{Q;i,j,k}, \quad \text{where } 0 \leqslant i,j,k \leqslant 2^{Q+1}.$$

Now, depending on the number of even or odd i,j,k in the element's label, we call it a *corner, segment, facet* or *volume* element. Thus an element can be

- *Corner element*, which consists of a single node and is of size 1 by 1 by 1.
- *Segment element*, which extends in 1 basis direction and is of size 1 by 1 by $P-1$ (or permutation).
- *Facet element*, which extends in 2 basis directions and is of size 1 by $P-1$ by $P-1$ (or permutation).
- *Volume element*, which extends in all 3 basis directions, and is of size $P-1$ by $P-1$ by $P-1$.

These names have geometric significance because one can also think of the various ways the planes parallel to the axes can intersect. If we consider the parity of the index (i,j,k) in turn, there are $2^3 = 8$ different types of nodes.

Element	i (mod 2)	j (mod 2)	k (mod 2)
Corner	0	0	0
x-direction segment	1	0	0
y-direction segment	0	1	0
z-direction segment	0	0	1
x-parallel facet	0	1	1
y-parallel facet	1	0	1
z-parallel facet	1	1	0
Volume	1	1	1

Each subdomain at level Q is made up of $3\times 3\times 3 = 27$ elements

$$\mathcal{V}_{Q;i,j,k} = \biguplus_{0\leqslant i',j',k'\leqslant 2} \mathcal{E}_{Q;2i+i',2j+j',2k+k'},$$

where we have used the symbol $\uplus$ to indicate a *disjoint union* so as to distinguish it from the more general union. These nodes in $\mathcal{V}_{Q;i,j,k}$ can be further classified as interior and boundary ones as follows:

- $\mathcal{I}_{Q;i,j,k}$: interior set that consists of the nodes from the single volume element.
- $\mathcal{B}_{Q;i,j,k}$: boundary set that consists of the nodes from 6 facet elements (2 parallel to each axis), 12 segment elements (4 parallel to each axis), and 8 corner elements.

More precisely, $\mathcal{V}_{Q;i,j,k} = \mathcal{I}_{Q;i,j,k} \uplus \mathcal{B}_{Q;i,j,k}$ with

$$\mathcal{I}_{Q;i,j,k} := \mathcal{E}_{Q;2i+1,2j+1,2k+1}$$

for the interior and

$$\mathcal{B}_{Q;i,j,k} := \biguplus_{\substack{0\leqslant i',j',k'\leqslant 2\\ (i',j',k')\neq(1,1,1)}} \mathcal{E}_{Q;2i+i',2j+j',2k+k'}$$

for the boundary of the subdomain. In Fig. 1 we illustrate an example of 8 overlapping leaf subdomains distinguishing between the different kinds of elements and showing how the subdomains meet. Notice that each node is plotted as a small cube in Fig. 1 in order to be able to show corner and segment elements more clearly. It should not be confused with the actual subdomain that contains these nodes.

2.2. Non-leaf levels

Based on what we introduced so far, we can start to define the node sets and elements for all other levels from bottom up. Suppose that the node sets $\mathcal{V}_{q+1;i,j,k}$, $\mathcal{I}_{q+1;i,j,k}$, $\mathcal{B}_{q+1;i,j,k}$ and the elements $\mathcal{E}_{q+1;i,j,k}$ are already defined for subdomains $\mathcal{D}_{q+1;i,j,k}$ on level $q+1$. We first introduce subdomains $\mathcal{D}_{q;i,j,k}$ on level q, defined by

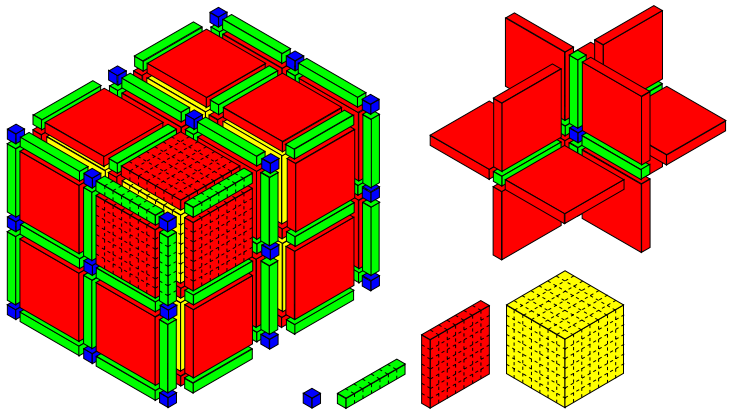


Fig. 1. Left: 8 overlapping leaf subdomains of size 9^3. Right: the interior structure. The four types of elements are shown as well.

$$\mathcal{D}_{q;i,j,k} := \bigcup_{0 \leqslant i',\, j',\, k' \leqslant 1} \mathcal{D}_{q+1;2i+i',2j+j',2k+k'}$$

for $0 \leqslant i, j, k < 2^q$. It is clear that the union of all $\mathcal{D}_{q;i,j,k}$ on level q is equal to the domain Ω.

For a fixed subdomain $\mathcal{D}_{q;i,j,k}$ on level q, its node set $\mathcal{V}_{q;i,j,k}$ is given by

$$\mathcal{V}_{q;i,j,k} := \bigcup_{0 \leqslant i',\, j',\, k' \leqslant 1} \mathcal{B}_{q+1;2i+i',2j+j',2k+k'}, \tag{3}$$

i.e., the union of the boundary set of the child subdomains. This set can be further partitioned into two subsets: interior $\mathcal{I}_{q;i,j,k}$ and boundary $\mathcal{B}_{q;i,j,k}$, depending on the location of the nodes. In terms of the elements defined on level $q+1$, the sets $\mathcal{I}_{q;i,j,k}$ and $\mathcal{B}_{q;i,j,k}$ can be written as

$$\mathcal{I}_{q;i,j,k} := \biguplus_{\substack{1 \leqslant i',\, j',\, k' \leqslant 3 \\ \text{at least one } 2}} \mathcal{E}_{q+1;4i+i',4j+j',4k+k'} \tag{4}$$

and

$$\mathcal{B}_{q;i,j,k} := \biguplus_{\substack{0 \leqslant i',\, j',\, k' \leqslant 4 \\ \text{at least one } 0 \text{ or } 4}} \mathcal{E}_{q+1;4i+i',4j+j',4k+k'}, \tag{5}$$

respectively. A simple counting shows that $\mathcal{V}_{q;i,j,k}$ consists of 117 elements from level $q+1$: 36 facets, 54 segments and 27 corners. They are allocated into $\mathcal{I}_{q;i,j,k}$ and $\mathcal{B}_{q;i,j,k}$ as follows:

- Interior $\mathcal{I}_{q;i,j,k}$ consists of the nodes from 19 elements: 12 facets, 6 segments, and 1 corner common to the 8 siblings.
- Boundary $\mathcal{B}_{q;i,j,k}$ consists of the nodes from 98 elements: 24 facets, 48 segments, and 26 corners which are shared with other subdomains.

To give an example, the elements of $\mathcal{B}_{q;i,j,k}$ on a slice parallel to the z-axis are displayed in the following table.

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

As we shall soon see, in order to support the algorithms to be described below, one needs to introduce a decomposition of $\mathcal{B}_{q;i,j,k}$ in terms of elements on its own level (level q), in addition to the one on its child level (level $q+1$). To that

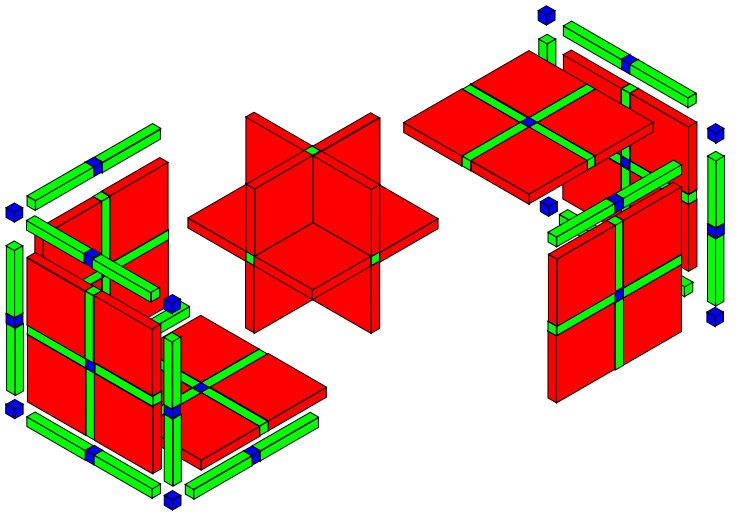


Fig. 2. The node set of a subdomain on level q is the union of the boundary sets of its children subdomains on level $q+1$. Left and right: the elements of the boundary part. Middle: the elements of the interior part. Notice that for the boundary part, elements on level $q+1$ are merged together to form elements on level q.

end, we define new elements $\mathcal{E}_{q;i,j,k}$ on level q on top of the elements on level $q+1$ as follows. Each new corner element consists of a single corner from level $q+1$, for example

$$\mathcal{E}_{q;2i,2j,2k} := \mathcal{E}_{q+1;4i,4j,4k}. \tag{6}$$

Each new segment element consists of two segments and a corner from the child level, for example

$$\mathcal{E}_{q;2i+1,2j,2k} := \mathcal{E}_{q+1;4i+1,4j,4k} \uplus \mathcal{E}_{q+1;4i+2,4j,4k} \uplus \mathcal{E}_{q+1;4i+3,4j,4k}, \tag{7}$$

and similarly for segments in the other directions. Finally, each new facet element consists of 4 facets, 4 segments, and 1 corner from the child level, for example

$$\begin{aligned}\mathcal{E}_{q;2i,2j+1,2k+1} := {} & \mathcal{E}_{q+1;4i,4j+1,4k+1} \uplus \mathcal{E}_{q+1;4i,4j+2,4k+1} \uplus \mathcal{E}_{q+1;4i,4j+3,4k+1} \\ & \uplus \mathcal{E}_{q+1;4i,4j+1,4k+2} \uplus \mathcal{E}_{q+1;4i,4j+2,4k+2} \uplus \mathcal{E}_{q+1;4i,4j+3,4k+2} \\ & \uplus \mathcal{E}_{q+1;4i,4j+1,4k+3} \uplus \mathcal{E}_{q+1;4i,4j+2,4k+3} \uplus \mathcal{E}_{q+1;4i,4j+3,4k+3}.\end{aligned} \tag{8}$$

The new facet elements parallel to the other axes are defined similarly. This combination is more complicated than the situation in two dimensions (see [12]), as there many more pieces to consider and there are more ways to order the combination of child elements in a parent facet. In Fig. 2 we show exactly how the boundary elements of the child level (level $q+1$) are combined to form larger elements one the parent level (level q).

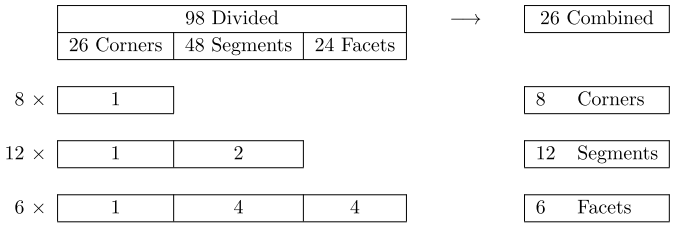
In terms of the new elements introduced on level q, the slice parallel to the z-axis that we showed earlier can be now represented as follows.

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

Given the newly introduced elements on level q, we have an alternative decomposition for a boundary set $\mathcal{B}_{q;i,j,k}$:

$$\mathcal{B}_{q;i,j,k} := \biguplus_{\substack{0\leqslant i',j',k'\leqslant 2 \\ \text{at least one 0 or 2}}} \mathcal{E}_{q;2i+i',2j+j',2k+k'}. \tag{9}$$

In this way, the 98 elements of $\mathcal{B}_{q;i,j,k}$ on level $q+1$ are combined into 26 combined elements on level q as follows.



Once we finish the definitions on level q, we can repeat the whole process on level $q-1$, and so on until we reach the top level. For each level q, we also define

$$\mathcal{I}_q := \biguplus_{i,j,k} \mathcal{I}_{q;i,j,k} \quad \text{and} \quad \mathcal{B}_q := \bigcup_{i,j,k} \mathcal{B}_{q;i,j,k}. \tag{10}$$

Using this definition, (3), (4), and (5), we have

$$\mathcal{B}_{q+1} = \mathcal{I}_q \cup \mathcal{B}_q. \tag{11}$$

Note that at level $q=0$, as there are no boundary nodes due to the Dirichlet boundary condition, we have $\mathcal{B}_1 = \mathcal{I}_0$.

3. Factorization approach with nested dissection

The nested dissection algorithm was proposed in [5] in 1973 and it quickly became the standard algorithm for solving definite sparse linear systems. Many extensions, such as the famous multifrontal methods [6], have been proposed for more general systems. We refer to the classical references [2,15,7] for more details about the nested dissection methods. Though the algorithm in this section follows the classical nested dissection method, the presentation is adapted to the hierarchical decomposition in Section 2. Section 3.1 describes the construction of the factorization for M, while Section 3.2 explains how to use the factorization to solve linear system $Mu = f$.

3.1. Construction step

The method essentially constructs and stores a block LDL^t transform of M in an efficient way. The basic tool for this process is Schur complement. For a symmetric matrix

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

the Schur complement step gives rise to a block LDL^t factorization of form

$$\begin{pmatrix} A & B^t \\ B & C \end{pmatrix} = \begin{pmatrix} I & \\ BA^{-1} & I \end{pmatrix} \begin{pmatrix} A & \\ & S \end{pmatrix} \begin{pmatrix} I & A^{-1}B^t \\ & I \end{pmatrix}, \tag{12}$$

where $S = C - BA^{-1}B^t$. In the following discussion, we shall use the notation $M : \mathcal{I} \to \mathcal{B}$ to denote that the matrix M maps a vector defined on node set $\mathcal{I}$ to a vector defined on node set $\mathcal{B}$.

Let us start from the leaf level, level Q. For each cube $\mathcal{D}_{Q;i,j,k}$, we can then consider the small matrix $M_{Q;i,j,k} : \mathcal{V}_{Q;i,j,k} \to \mathcal{V}_{Q;i,j,k}$, which is the restriction of M to the cube $\mathcal{D}_{Q;i,j,k}$, formed via

$$(M_{Q;i,j,k})_{\kappa\lambda} = \int_{\mathcal{D}_{Q;i,j,k}} \nabla\phi_\kappa(\mathbf{x}) \cdot a(\mathbf{x}) \nabla\phi_\lambda(\mathbf{x}) + V(\mathbf{x})\phi_\kappa(\mathbf{x})\phi_\lambda(\mathbf{x})\, d\mathbf{x}, \tag{13}$$

where κ and λ are restricted to the nodes in $\mathcal{V}_{Q;i,j,k}$ since all basis functions centered on nodes outside $\mathcal{D}_{Q;i,j,k}$ are zero inside $\mathcal{D}_{Q;i,j,k}$. Comparing (13) with (2), we see that the sum of $M_{Q;i,j,k}$ over all possible i, j, k (after suitable injection) is equal to the full matrix M.

Start from level Q and fix a leaf cube $\mathcal{D}_{Q;i,j,k}$ and its matrix $M_{Q;i,j,k}$. The decomposition of the nodes $\mathcal{V}_{Q;i,j,k} = \mathcal{I}_{Q;i,j,k} \uplus \mathcal{B}_{Q;i,j,k}$ leads to a 2×2 block matrix decomposition of $M_{Q;i,j,k}$ using (12)

$$M_{Q;i,j,k} = \begin{pmatrix} A_{Q;i,j,k} & B^t_{Q;i,j,k} \\ B_{Q;i,j,k} & C_{Q;i,j,k} \end{pmatrix} = L_{Q;i,j,k} \begin{pmatrix} A_{Q;i,j,k} & \\ & S_{Q;i,j,k} \end{pmatrix} L^t_{Q;i,j,k}, \tag{14}$$

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

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

With a slight abuse of notation, we can extend $L_{Q;i,j,k}$ to the whole node set by setting it to be identity on the complement of $\mathcal{V}_{Q;i,j,k}$. Since the interior node sets $\mathcal{I}_{Q;i,j,k}$ are disjoint for different blocks $\mathcal{D}_{Q;i,j,k}$, each one of $L_{Q;i,j,k}$ commutes with another distinct $L_{Q;i',j',k'}$ and as a result $L_Q := \prod_{i,j,k} L_{Q;i,j,k}$ is well defined. We will develop a suitable ordering for the rows and columns of M as we proceed. Recall from (10) that

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

Since the union of the two is the entire set of nodes for which we constructed M, we have the following 2×2 block form for M

$$M = \begin{pmatrix} A_Q & B_Q^t \\ B_Q & C_Q \end{pmatrix} = L_Q \begin{pmatrix} A_Q & \\ & S_Q \end{pmatrix} L_Q^t, \tag{15}$$

where $A_Q : \mathcal{I}_Q \to \mathcal{I}_Q$, $B_Q : \mathcal{I}_Q \to \mathcal{B}_Q$, $C_Q : \mathcal{B}_Q \to \mathcal{B}_Q$, $S_Q : \mathcal{B}_Q \to \mathcal{B}_Q$, and $S_Q = C_Q - B_Q A_Q^{-1} B_Q^t$. Comparing (15) and (14), we see that A_Q is the sum of $A_{Q;i,j,k}$ over all possible (i,j,k) after suitable injection and similarly S_Q is the sum of $S_{Q;i,j,k}$ over all possible (i,j,k).

In order to further factorize S_Q in (15), let us study the subdomains on level $Q-1$. For each subdomain $\mathcal{D}_{Q-1;i,j,k}$ at level $Q-1$, we define

$$M_{Q-1;i,j,k} : \mathcal{V}_{Q-1;i,j,k} \to \mathcal{V}_{Q-1;i,j,k}$$

to be the sum of the Schur complement matrices $S_{Q;i',j',k'} : \mathcal{B}_{Q;i',j',k'} \to \mathcal{B}_{Q;i',j',k'}$ over the eight children of subdomain $\mathcal{D}_{Q-1;i,j,k}$. Comparing this with the above statement about S_Q being the sum of $S_{Q;i',j',k'}$ over all possible (i',j',k'), we see that S_Q is equal to the sum of all $M_{Q-1;i,j,k}$ on level $Q-1$ (if we extend $M_{Q-1;i,j,k}$ by setting it to be zero outside $\mathcal{V}_{Q-1;i,j,k}$). Recall that its node set $\mathcal{V}_{Q-1;i,j,k}$ decomposes into the interior $\mathcal{I}_{Q-1;i,j,k}$ and the boundary $\mathcal{B}_{Q-1;i,j,k}$. This leads naturally to a 2×2 block form for $M_{Q-1;i,j}$

$$M_{Q-1;i,j,k} := \begin{pmatrix} A_{Q-1;i,j,k} & B_{Q-1;i,j,k}^t \\ B_{Q-1;i,j,k} & C_{Q-1;i,j,k} \end{pmatrix}, \tag{16}$$

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

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

Similarly to what we did for level Q, by recalling the relationship from (10) and (11)

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

and defining

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

we can rewrite the above computation as factoring 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} and S_{Q-1} are the sums of all possible $A_{Q-1;i,j,k}$ and $S_{Q-1;i,j,k}$, respectively, after suitable injection. As a result we get a new factorization of M:

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

Continuing in this fashion over all levels from bottom up, we eventually reach level 0 with $\mathcal{I}_0 = \mathcal{B}_1$, $\mathcal{B}_0 = \emptyset$ (due to the zero Dirichlet boundary condition), and

$$M = L_Q L_{Q-1} \cdots L_1 \begin{pmatrix} A_Q & & & & \\ & A_{Q-1} & & & \\ & & \ddots & & \\ & & & A_1 & \\ & & & & A_0 \end{pmatrix} L_1^t \cdots L_{Q-1}^t L_Q^t, \tag{17}$$

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

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

taking each of the sets $\mathcal{I}_{q;i,j,k}$ in turn for our ordering. Inverting the factorization (17) gives a factorization of M^{-1}:

$$M^{-1} = L_Q^{-t} L_{Q-1}^{-t} \cdots L_1^{-t} \begin{pmatrix} A_Q^{-1} & & & & \\ & A_{Q-1}^{-1} & & & \\ & & \ddots & & \\ & & & A_1^{-1} & \\ & & & & A_0^{-1} \end{pmatrix} L_1^{-1} \cdots L_{Q-1}^{-1} L_Q^{-1}. \tag{18}$$

We see now that creating the factorized form of M happens in two stages:

1. At the leaf level Q, for each subdomain $\mathcal{D}_{Q;i,j,k}$, we calculate $M_{Q;i,j,k}$ which is the restriction of M to $\mathcal{D}_{Q;i,j,k}$.
2. Move up level by level. For each level q and each subdomain $\mathcal{D}_{q;i,j,k}$, combine the eight matrices $S_{q+1;i',j',k'}$ of its children into the parent $M_{q;i,j,k}$.

This is detailed in Algorithm 1. In the description, we use the MATLAB notation for referring to submatrices. For example, 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 $\mathcal{X}, \mathcal{Y} \subset \mathcal{J}$ we write $G(\mathcal{X}, \mathcal{Y}) \in \mathbb{R}^{|\mathcal{X}| \times |\mathcal{Y}|}$ for the submatrix of G consisting of the rows in $\mathcal{X}$ and columns in $\mathcal{Y}$. Manipulating this matrix affects the underlying values in G.

Algorithm 1 *(Setup the factorization of M).*

for $i = 0$ to $2^Q - 1$ **do**
  for $j = 0$ to $2^Q - 1$ **do**
    for $k = 0$ to $2^Q - 1$ **do**
      Calculate the matrix $M_{Q;i,j,k}$.
      Invert $A_{Q;i,j,k}$ using dense matrix methods.
      $S_{Q;i,j,k} \leftarrow C_{Q;i,j,k} - B_{Q;i,j,k} A_{Q;i,j,k}^{-1} B_{Q;i,j,k}^t$
      Store $A_{Q;i,j,k}^{-1}$, $B_{Q;i,j,k}$ and $S_{Q;i,j,k}$
    end for
  end for
end for
for $q = Q - 1$ to 1 **do**
  for $i = 0$ to $2^q - 1$ **do**
    for $j = 0$ to $2^q - 1$ **do**
      for $k = 0$ to $2^q - 1$ **do**
        Start with zero $M_{q;i,j,k}$
        for $i' = 0, 1$ **do**
          for $j' = 0, 1$ **do**
            for $k' = 0, 1$ **do**
              Add $S_{q+1,2i+i',2j+j',2k+k'}$ to $M_{q;i,j,k}(\mathcal{B}_{q+1;2i+i',2j+j',2k+k'}, \mathcal{B}_{q+1;2i+i',2j+j',2k+k'})$
            end for
          end for
        end for
        Decompose $M_{q;i,j,k}$ into 2×2 block structure $M_{q;i,j} = \begin{pmatrix} A_{q;i,j} & B_{q;i,j}^t \\ B_{q;i,j} & C_{q;i,j} \end{pmatrix}$.
        Invert $A_{q;i,j,k}$.
        $S_{q;i,j,k} \leftarrow C_{q;i,j,k} - B_{q;i,j,k} A_{q;i,j,k}^{-1} B_{q;i,j,k}^t$
        Store $A_{q;i,j,k}^{-1}$ and $B_{q;i,j,k}$
        Store $S_{q;i,j,k}$
      end for
    end for
  end for
end for
Start with zero $M_{0;0,0,0}$
for $i' = 0, 1$ **do**
  for $j' = 0, 1$ **do**

for $k' = 0, 1$ **do**
  Add $S_{1,i',j',k'}$ to $M_{0;0,0,0}(\mathcal{B}_{1;i',j',k'}, \mathcal{B}_{1;i',j',k'})$
end for
end for
end for
Invert $A_{0;0,0,0}$.
Store $A_{0;0,0,0}$.

In terms of computational complexity, this factorization of M can be constructed in $\mathcal{O}(N^2)$ steps and solving $Mu = f$ by applying (18) to f takes $\mathcal{O}(N^{4/3})$ steps. To see this, first notice that the number of levels is $Q+1$ and the total number of unknowns is $N \simeq (P2^Q)^3 = P^3 2^{3Q}$. For each level q, let us use $s(q) \simeq P2^{Q-q}$ to denote the side length of the subdomain at this level. Since each facet has size $s(q)^2$, the largest matrices appearing at level q in the algorithm have size $\mathcal{O}(s(q)^2)$. Thus the cost of multiplying and inverting matrices for each subdomain will be $\mathcal{O}(s(q)^6)$, while the cost of a matrix–vector multiply will be $\mathcal{O}(s(q)^4)$. Thus the total cost, suppressing constants, for setting up the factorization for M and M^{-1} is

$$\sum_{q=0}^{Q} s(q)^6 2^{3q} = \sum_{q=0}^{Q} \left(P2^{Q-q}\right)^6 2^{3q} = \mathcal{O}\left(N^2\right).$$

3.2. Solution step

To solve the original $Mu = f$, we compute $u = M^{-1} f$ using (18):

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

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

Algorithm 2 *(Solving $Mu = f$).*

$u \leftarrow f$
for $q = Q$ to 1 **do**
  for $i = 0$ to $2^q - 1$ **do**
    for $j = 0$ to $2^q - 1$ **do**
      for $k = 0$ to $2^q - 1$ **do**
        $u_{\mathcal{I}_{q;i,j,k}} \leftarrow A_{q;i,j,k}^{-1} u_{\mathcal{I}_{q;i,j,k}}$
        $u_{\mathcal{B}_{q;i,j,k}} \leftarrow u_{\mathcal{B}_{q;i,j,k}} - B_{q;i,j,k} A_{q;i,j,k}^{-1} u_{\mathcal{I}_{q;i,j,k}}$
      end for
    end for
  end for
end for
$u_{\mathcal{I}_{0;0,0,0}} \leftarrow A_{0;0,0,0}^{-1} u_{\mathcal{I}_{0;0,0,0}}$
for $q = 1$ to Q **do**
  for $i = 0$ to $2^q - 1$ **do**
    for $j = 0$ to $2^q - 1$ **do**
      for $k = 0$ to $2^q - 1$ **do**
        $u_{\mathcal{I}_{q;i,j,k}} \leftarrow u_{\mathcal{I}_{q;i,j,k}} - A_{q;i,j,k}^{-1} B_{q;i,j,k}^t u_{\mathcal{B}_{q;i,j,k}}$
      end for
    end for
  end for
end for

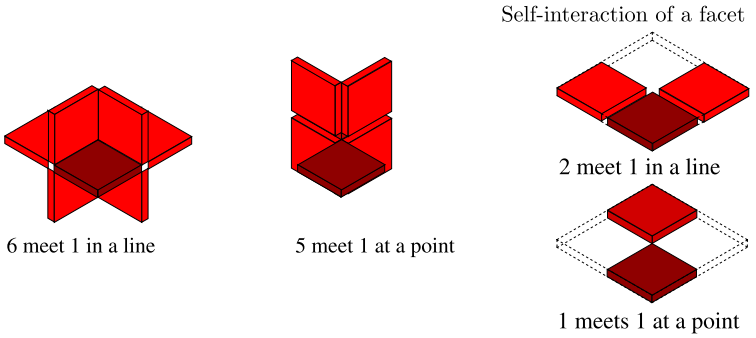


Fig. 3. Left: Examples of facets meeting in a line or at a point for the facets in $\mathcal{I}_{q;i,j,k}$. Right: self-interaction of a facet.

Following the discussion for the complexity of the matrix factorization, we see that applying M^{-1} to a vector is

$$\sum_{q=0}^{Q} s(q)^4 2^{3q} = \sum_{q=0}^{Q} \left(P 2^{Q-q}\right)^4 2^{3q} = \mathcal{O}\left(N^{4/3}\right)$$

as claimed.

4. Acceleration with hierarchical matrix algebra

The $\mathcal{O}(N^2)$ complexity of Algorithm 1 has always been the main bottleneck of the matrix factorization method with nested dissection for three dimensional problems. From the complexity analysis, we see that the main contribution comes from the operations on the matrices $A_{q;i,j,k}$, $B_{q;i,j,k}$, $C_{q;i,j,k}$, and $S_{q;i,j,k}$. Therefore, it is desirable to speed up these operations through accurate approximation which can potentially bring down the complexity. For two dimensional problems, Xia et al. [8] reduced the complexity of nested dissection method by using hierarchical semiseparable matrices [11] to represent these matrices. Due to the almost linear cost of the operations of the hierarchical semiseparable matrices, this allows the Schur complements at all levels to be performed in almost linear time.

In [12], we instead used the hierarchical matrix algebra to represent those matrices appearing in the factorization process. The main observation of the hierarchical matrix algebra is that, for a wide class of matrices, such as the Green's functions of elliptic operators, a submatrix that corresponds to the interaction between two well-separated groups of nodes (or degrees of freedom) is numerically low-rank. Based on this observation, one can then organize all nodes with a hierarchical structure and whenever two subsets of nodes are well-separated in the hierarchical structure the submatrix associated with these two subsets is stored in a low-rank factorized form. This hierarchical compression scheme allows one to store an $n \times n$ hierarchical matrix in $\mathcal{O}(n \log n)$ storage space. By exploiting the low-rank factorized form in the computation, all basic matrix operations, such as matrix–vector product, matrix–matrix addition, matrix–matrix product, and matrix-inversion, can be carried out in almost linear cost as well. In the following discussion of our algorithm, we will use these operations as black boxes. We refer the readers to Section 4.4 for a brief discussion about their implementations and to [10] for complete details.

Following our previous work in [12], we propose to represent $A_{q;i,j,k}$, $B_{q;i,j,k}$, $C_{q;i,j,k}$, and $S_{q;i,j,k}$ using the hierarchical matrix algebra. In order to do that, first we need to identify the hierarchical structure for organizing the nodes of these matrices. As shown in (4) and (5), the node set $\mathcal{I}_{q;i,j,k}$ and $\mathcal{B}_{q;i,j,k}$ of a subdomain $\mathcal{D}_{q;i,j,k}$ can be viewed as a disjoint union of elements at level $q+1$, each of which is a disjoint union of elements at level $q+2$, and so on (see (6), (7), (8)). This naturally provides the hierarchical structure for the matrices $M_{q;i,j,k}$, $A_{q;i,j,k}$, $B_{q;i,j,k}$, $C_{q;i,j,k}$, and $S_{q;i,j,k}$. For example, since $\mathcal{V}_{q;i,j,k}$ is the disjoint union of 117 elements on level $q+1$, $M_{q;,i,j,k} : \mathcal{V}_{q;i,j,k} \to \mathcal{V}_{q;i,j,k}$ can be viewed as a 117×117 block matrix. Following this, $A_{q;,i,j,k} : \mathcal{I}_{q;i,j,k} \to \mathcal{I}_{q;i,j,k}$ is a 19×19 block matrix, $B_{q;,i,j,k} : \mathcal{B}_{q;i,j,k} \to \mathcal{I}_{q;i,j,k}$ is a 98×19 block matrix, and $C_{q;i,j,k} : \mathcal{B}_{q;i,j,k} \to \mathcal{B}_{q;i,j,k}$ and $S_{q;i,j,k} : \mathcal{B}_{q;i,j,k} \to \mathcal{B}_{q;i,j,k}$ are both 98×98 block matrices.

The hierarchical matrix decomposition for these matrices depends on the choices made as to which blocks are represented by hierarchical matrices and which are represented by low rank matrices in factorized form. The choice of which blocks can be represented by low rank matrices depends on the *admissibility* criteria [10,16] used. In our case this reveals itself in the choice to represent the interaction between different facets larger than a chosen size as one low rank matrix in factorized form instead of the hierarchical matrix required by the lack of separation between the facets.

Well separated facets, such as those on the top and bottom of a cube, will always have their interaction represented in factorized form. For nearby facets, such as the 12 facet elements in the interior $\mathcal{I}_{q;i,j,k}$ of $\mathcal{D}_{q;i,j,k}$, we distinguish between two different settings (as illustrated in Fig. 3), depending on whether the minimum distance between two facets is achieved

- at a point, or
- on a line.

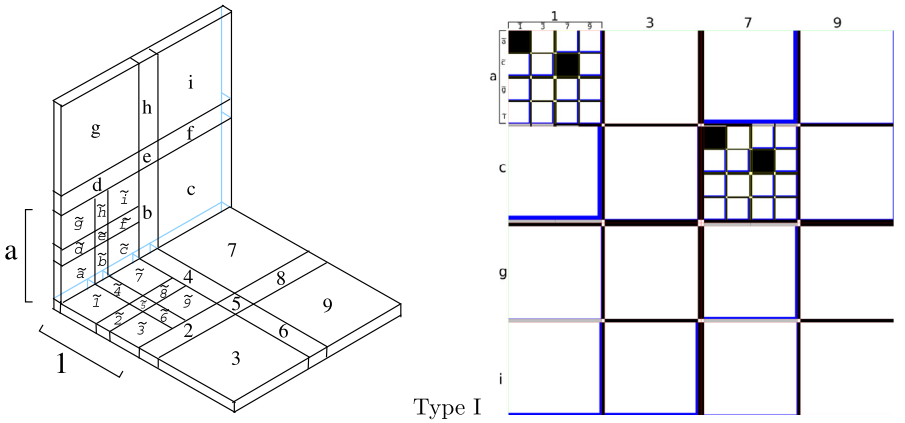


Fig. 4. Type I hierarchical division of adjacent Facets of the cube and associated interaction matrix with low rank matrices in factorized form represented by the symbol ⌜ or ⌟.

Those facets achieving minimum distance at a point will always have a low rank interaction and be represented in factorized form. On the other hand, those facets achieving minimum distance in a line will be represented in two different ways

- *Type* I: by a hierarchical matrix,
- *Type* II: in a factorized form.

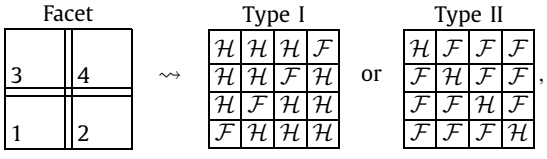
The advantage of Type I is that the growth of the maximum rank is better controlled, but at the cost of a deeper and more complex hierarchical decomposition.

4.1. Interaction between adjacent facets

In Fig. 4, we illustrate a possible Type I hierarchical division of two adjacent facets of a subdomain. This situation appears between two level $q+1$ facets in $A_{q;i,j,k}$ or two level q facets in $S_{q;i,j,k}$. The main pieces of the subdivision of a facet are the 4 sub-facets labeled $1, 3, 7, 9$ for one facet and a, c, g, i for the other. The interactions of the adjacent sub-facets (for example between 1 and a and between 7 and c) are represented hierarchically while the rest are low-rank. Considering the further subdivisions of 1 and a the sub-sub-facets show the same pattern of interaction with $\tilde{1}$ & $\tilde{a}$ and $\tilde{7}$ & $\tilde{c}$ represented densely while the others are low-rank. Depending on the sizes of the remaining sub-sub-facets one might continue further subdivision and use a hierarchical representation instead of the dense one. For Type II we would have one big factorized matrix as the two large adjacent (undivided) facets meet in a line.

4.2. Self-interaction

For the case of facet self-interaction let us consider a facet that has been divided into the 9 child elements as shown in Fig. 5 (dominated by the 4 sub-facets labeled 1, 3, 7 and 9) with subfacet 1 further divided into $\tilde{1}$ to $\tilde{9}$. If we only look at the interaction of 4 subfacets of a facet we would see the following patterns

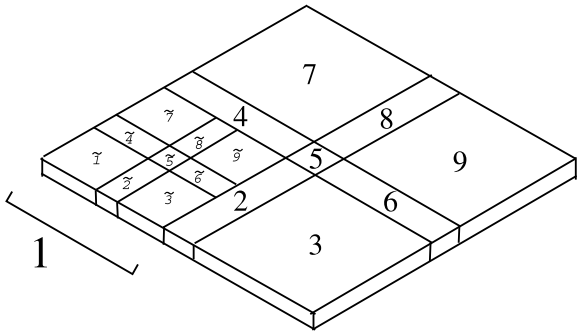


where $\mathcal{F}$ and $\mathcal{H}$ stand for factorized and hierarchical blocks, respectively.

In Type I only the interaction between facets diagonally opposite each other is represented in factorized form, while in Type II only the self-interaction is represented in hierarchical form.

4.3. Merge step of the S matrices

There is one extra complication regarding the S matrices in lines 19 and 36 of Algorithm 1. For example at line 19, the matrix $S_{q+1,2i+i',2j+j',2k+k'} : \mathcal{B}_{q+1;2i+i',2j+j',2k+k'} \to \mathcal{B}_{q+1;2i+i',2j+j',2k+k'}$ is a 98×98 block matrix, since $\mathcal{B}_{q+1;2i+i',2j+j',2k+k'}$ is the disjoint union of 98 elements at level $q+2$. However, on the right hand side, the matrix $M_{q;i,j,k} : \mathcal{V}_{q;i,j,k} \to \mathcal{V}_{q;i,j,k}$ has a block matrix structure based on the partitioning of $\mathcal{V}_{q;i,j,k}$ into elements at level $q+1$. Clearly, there is a one-level



Type I

Type II

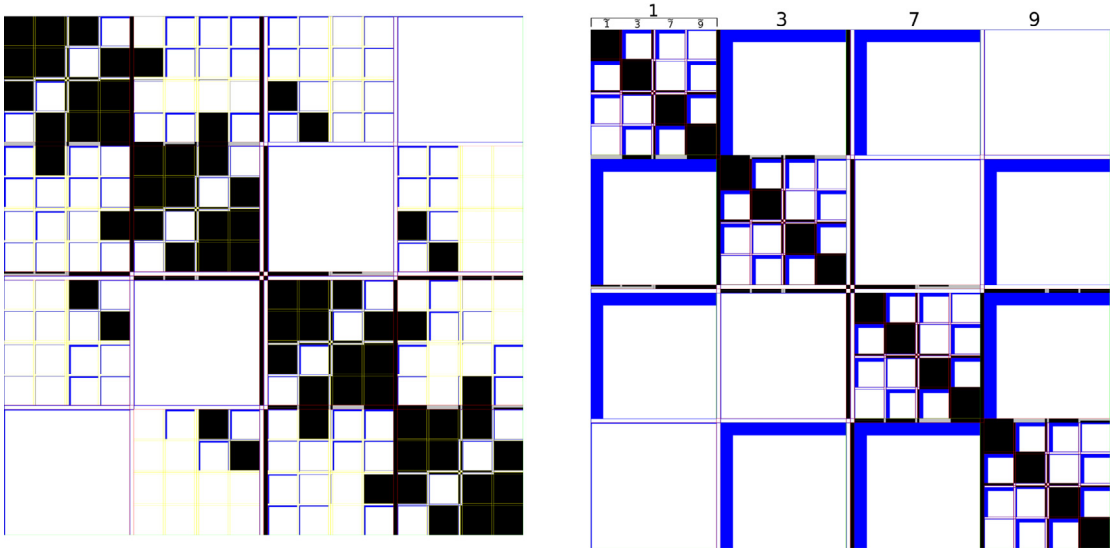


Fig. 5. Hierarchical division of self-interaction of a facet of the cube in Type I and Type II matrix decompositions. This would correspond to one part on the block diagonal of the full interaction matrix for the 6 facets (and the rest of the cube) shown in Fig. 6 and Fig. 7.

mismatch between the left-hand side and the right-hand side of this step. Therefore, one needs to reorganize the matrix $S_{q+1,2i+i',2j+j',2k+k'} : \mathcal{B}_{q+1;2i+i',2j+j',2k+k'} \to \mathcal{B}_{q+1;2i+i',2j+j',2k+k'}$ into a block structure in terms of the elements on level $q+1$. This is possible since the boundary set $\mathcal{B}_{q+1;2i+i',2j+j',2k+k'}$ can also be written as a disjoint union of 26 elements on the same level, following (9). As a result, we need to reinterpret a 98×98 block matrix corresponding to the 98 boundary elements at level $q+2$ as 26×26 submatrices corresponding to the 26 merged boundary elements at level $q+1$. The 8 corner nodes are unaffected. The segment-corner-segment merging into a parent segment is reflected in combining $3 \times 3 = 9$ submatrices into a new submatrix. Merging of 9 child elements in a plane into a parent facet corresponds to $9 \times 9 = 81$ submatrices being joined together. The node ordering we built up from the leaf level ensures that, in fact, these 9 submatrices form a contiguous 3×3 group and the 81 submatrices form a contiguous 9×9 group. Thus no rearrangement of the rows and columns of $S_{q+1,2i+i',2j+j',2k+k'}$ is required. In terms of the hierarchical matrix representation, if the new submatrix should have a hierarchical representation this is achieved by simply reinterpreting the 3×3 (in the case of a parent segment) or 9×9 (in the case of a parent facet) submatrices as part of a new hierarchical matrix decomposition. On the other hand if the new submatrix should be represented in factorized form, this conversion can be performed efficiently using standard QR and SVD [10].

From the above discussion, it is clear that the hierarchical decomposition of the geometric elements defined in Section 2 naturally corresponds to the hierarchical decomposition of the matrices for those sets of nodes discussed here.

4.4. Implementation details

We refer the reader to [10] for details on hierarchical matrix operations but present some aspects of our implementation here (as we did in [12]). The underlying dense matrix algebra is performed using BLAS and LAPACK (in particular Intel's MKL), with matrix inversion of dense matrices performed via LU-factorization.

In general, operations on hierarchical matrices proceed using two main ideas: block matrix algebra and recursion. If one were to multiply two matrices of Type II such as those illustrated in Fig. 5, the off-diagonal matrices are in factorized form

and the on-diagonal matrices are again hierarchical, so the block form of the product would contain sums of products of matrices in factorized form and matrices in hierarchical form. Since we know the hierarchical structure of the final product we can use that information to guide the calculation of each block in the product.

Matrix addition and subtraction. Consider the sum of two factorized matrices G and H with their factorizations denoted by $G \approx UV^t$ and $H \approx XY^t$. Then,

$$G + H \approx UV^t + XY^t = (U|X)(V|Y)^t.$$

Notice that the new factorized form for the sum will have an increased size compared to those for G and H. In order to prevent the size of these low rank factorizations from increasing with each addition the two new matrices need to be recompressed. So if U has width r_1 and X has width r_2, then using $(U|X) = QR$ and $(V|Y) = \tilde{Q}\tilde{R}$, the sum we seek is $QR(\tilde{Q}\tilde{R})^t = Q(R\tilde{R}^t)\tilde{Q}^t$, and we need only perform the SVD, $R\tilde{R}^t = \bar{U}\bar{\Sigma}\bar{V}^t$, 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 $\bar{U}$ and $\bar{V}$) for our truncated SVD. Thus

$$(U|X)(V|Y)^t = \underbrace{Q\bar{U}\bar{\Sigma}}_{\text{width } r'} \underbrace{(\tilde{Q}\bar{V})^t}_{\text{width } r'}.$$

The sum of two hierarchical blocks is 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. The sum of a factorized and a hierarchical block is done by decomposing the factorized block to match the substructure of the hierarchical block and proceeding recursively. The only remaining non-standard operation will be the sum of a dense and a factorized block which is done by reforming the dense version of the factorized block, then doing the dense addition and re-factorization if needed.

Matrix multiplication. For simplicity, we only consider the case where there are 4 main blocks, two on-diagonal hierarchical matrices and two off-diagonal factorized ones since these contain all the essential ingredients required for the more complex cases with multiple off diagonal blocks.

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

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

First, the off-diagonal block

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

The computation $G_{1,1}X_{1,2}$ or $(V_{1,2})^t H_{2,2}$ is multiplication of a hierarchical matrix with a dense matrix which has a small number of columns and proceeds by decomposing the dense matrix into blocks to match the substructure of the hierarchical matrix and proceeding recursively. The multiplication of a matrix in factorized form and a dense matrix proceeds using in turn the two dense matrices that make up the factorized form. The next stage involves a dense matrix with a small number of rows and proceeds in a similar fashion. Once they are done, the remaining sum is then similar to the matrix addition described above. The other off-diagonal block $G_{2,1}H_{1,1} + G_{2,2}H_{2,1}$ is done in the same way.

Next, consider the diagonal blocks. Take $G_{1,1}H_{1,1} + G_{1,2}H_{2,1}$ as an example. The first part $G_{1,1}H_{1,1}$ is done using recursion. The second part is

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

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

Matrix inversion. The inversion of hierarchical matrices is again done via recursion. Row operations are performed on the block structure of a hierarchical matrix, which requires the inversion of the matrices on the diagonal, which will again be hierarchical. Eventually the on-diagonal matrices will be dense and their inversion is performed using standard methods.

Table 1
The maximum ranks observed for the factorized low rank approximations when using the Type I and Type II hierarchical matrix decompositions for the matrices A^{-1}, B and S in Algorithm 1 for $-\Delta u = f$ with $\epsilon_r = 10^{-6}$ and $\epsilon_a = 10^{-12}$. For Type I these grow like $\mathcal{O}(\log s(q))$ while for Type II they grow like $\mathcal{O}(s(q))$.

	Type I		
A^{-1}	20	27	29
B	25	33	26
S	25	33	41
$s(q)$	15	31	63
	Type II		
A^{-1}	63	122	234
B	75	161	–
S	75	161	331
$s(q)$	15	31	63

5. Complexity analysis

Let us investigate the complexity of Algorithms 1 and 2 when hierarchical matrix algebra is used. We recall that a leaf node at level Q contains $(P+1)\times(P+1)\times(P+1)$ nodes and $N \simeq (P2^Q)^3 = P^3 2^{3Q} = \mathcal{O}(2^{3Q})$ since $P = \mathcal{O}(1)$. Here all logarithms are taken with base 2.

The cost of the hierarchical matrix operations depends on the depth of the decomposition tree and the ranks of the factorized approximations. The cost of a matrix–vector multiplication scales as $\mathcal{O}(rn\log n)$, where r is the maximum rank of the factorized parts, n is the dimension of the matrix, and $\log n$ comes from the number of subdivision levels (depth of the decomposition tree) of the hierarchical representation of the matrix involved. The cost of the matrix–matrix multiplication as well as matrix inversion scales like $\mathcal{O}(r^2 n(\log n)^2)$ instead [17].

The size of the matrices involved for a given subdomain is dominated by the number of nodes in the covered facets. Fix a cube at $\mathcal{D}_{q;i,j,k}$ at level q. For $A_{q;i,j,k}$, there are 12 facets and the matrix dimension n is of order $\mathcal{O}(s(q)^2)$. Similarly the corresponding $B_{q;i,j,k}$ and $S_{q;i,j,k}$ will also have size $n = \mathcal{O}(s(q)^2)$. The formation of the Schur complement for each subdomain involves an inversion of a hierarchical matrix, two multiplications of hierarchical matrices, and an addition of hierarchical matrices.

The difference in complexity for the Type I and Type II hierarchical matrix decompositions comes from the differing rank r of the factorized parts. We have observed in numerical experiments that the rank of the factorized parts for the Type I decomposition grows like $\mathcal{O}(\log s(q))$ while that for Type II grows like $\mathcal{O}(s(q))$ as shown in Table 1.

5.1. Type I hierarchical matrices

In this subsection we consider the complexity when using matrices of Type I such as that illustrated in Fig. 6. We have seen numerically that the rank r of the factorized parts will be approximately $\mathcal{O}(\log s(q))$ at level q. Allowing slightly stronger growth let us consider poly-logarithmic growth $\mathcal{O}(\log^\rho s(q))$ for some integer $\rho \geqslant 1$. Consider first the complexity of Algorithm 1. The cost for each subdomain is, suppressing constants,

$$\mathcal{O}\big(r^2(\log n)^2 n\big) = \mathcal{O}\big(\big(\log s(q)\big)^{2\rho}\cdot\big(\log s(q)\big)^2\cdot s(q)^2\big) = \mathcal{O}\big(\big(\log s(q)\big)^{2\rho+2}\cdot s(q)^2\big)$$

as $n = \mathcal{O}(s(q)^2)$. Summing over 2^{3q} Schur complements at each level and Q levels in total, we obtain

$$\sum_{q=0}^{Q}(Q-q)^{2\rho+2}\cdot 2^{2(Q-q)}\cdot 2^{3q} = \mathcal{O}\big(2^{3Q}\big) = \mathcal{O}(N).$$

In Algorithm 2, the dominant cost is the matrix vector multiplication in the hierarchical matrix form and for each cube on level q, it takes

$$\mathcal{O}\big(r^2(\log n)n\big) = \mathcal{O}\big(\big(\log s(q)\big)^{2\rho}\cdot\big(\log s(q)\big)\cdot s(q)^2\big) = \mathcal{O}\big(\big(\log s(q)\big)^{2\rho+1}\cdot s(q)^2\big)$$

steps. Therefore, the total cost over all subdomains and levels is equal to

$$\sum_{q=0}^{Q}(Q-q)^{2\rho+1}\cdot 2^{2(Q-q)}\cdot 2^{3q} = \mathcal{O}\big(2^{3Q}\big) = \mathcal{O}(N)$$

again. In summary, both algorithms are of linear complexity.

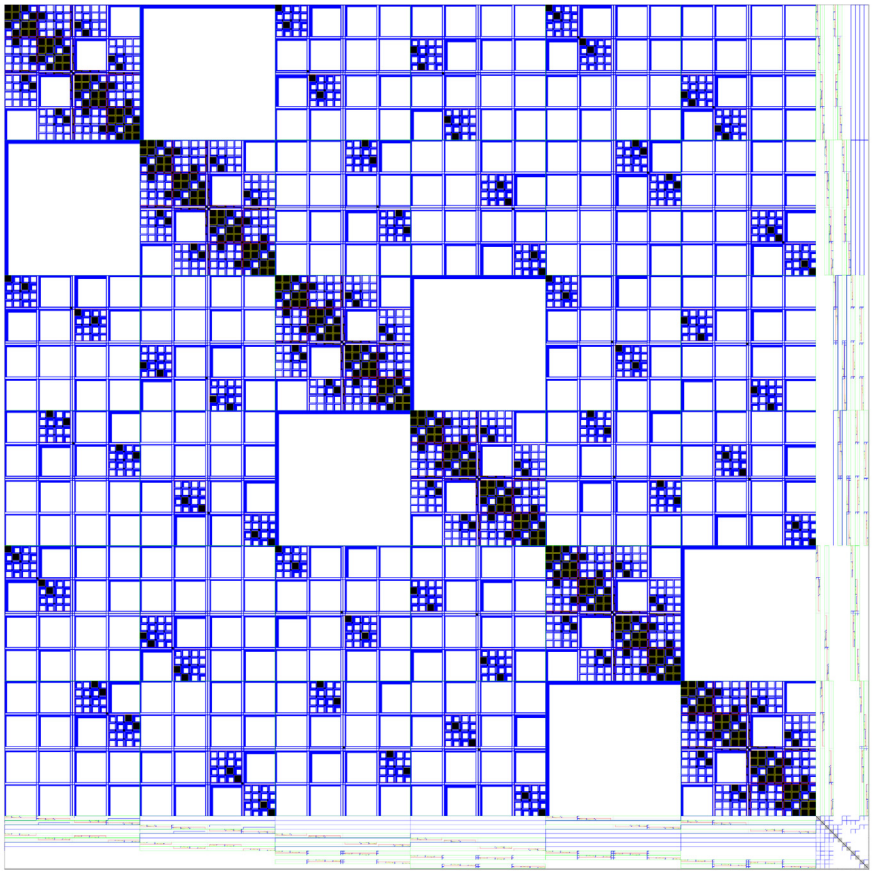


Fig. 6. Type I hierarchical division of the interaction matrix of a cube boundary (merged $S_{2;1,1,1}$) with 31×31 facets switching from hierarchical to dense matrices for size 7×7 sub-facets.

5.2. Type II hierarchical matrices

In this subsection we consider the complexity when using matrices of Type II such as that illustrated in Fig. 7. Consider first the complexity of Algorithm 1. Since the observed ranks are proportional to the segment lengths ($r \simeq s(q)$), the cost for each subdomain will be

$$\mathcal{O}\big(r^2(\log n)^2 n\big) = \mathcal{O}\big(s(q)^2 \cdot \big(\log s(q)\big)^2 \cdot s(q)^2\big) = \mathcal{O}\big(\big(\log s(q)\big)^2 \cdot s(q)^4\big).$$

Summing over 2^{3q} Schur complements at each level and Q levels in total gives us

$$\sum_{q=0}^{Q} (Q-q)^2 \cdot 2^{4(Q-q)} \cdot 2^{3q} = \mathcal{O}\big(Q^2 2^{4Q}\big) = \mathcal{O}\big(N^{4/3}\log^2 N\big).$$

Next let us consider the complexity of Algorithm 2. The matrix vector multiplication in the hierarchical matrix form for each cube takes

$$\mathcal{O}\big(r(\log n)n\big) = \mathcal{O}\big(s(q) \cdot \big(\log s(q)\big) \cdot s(q)^2\big) = \mathcal{O}\big(\log s(q) \cdot s(q)^3\big)$$

steps. Therefore, the number of steps of Algorithm 2 in Type II case is equal to

$$\sum_{q=0}^{Q} (Q-q) \cdot 2^{3(Q-q)} \cdot 2^{3q} = \mathcal{O}\big(Q\, 2^{3Q}\big) = \mathcal{O}(N \log N).$$

Comparing with the Type I hierarchical matrices, the construction cost of Type II matrix is $\mathcal{O}(N^{1/3}\log^2 N)$ times more expensive, while the solution cost is only a factor of $\mathcal{O}(\log N)$ higher. As we shall see in the numerical results displayed in the next section, the second approach turns out to have better performance in the regime we could test. Eventually the

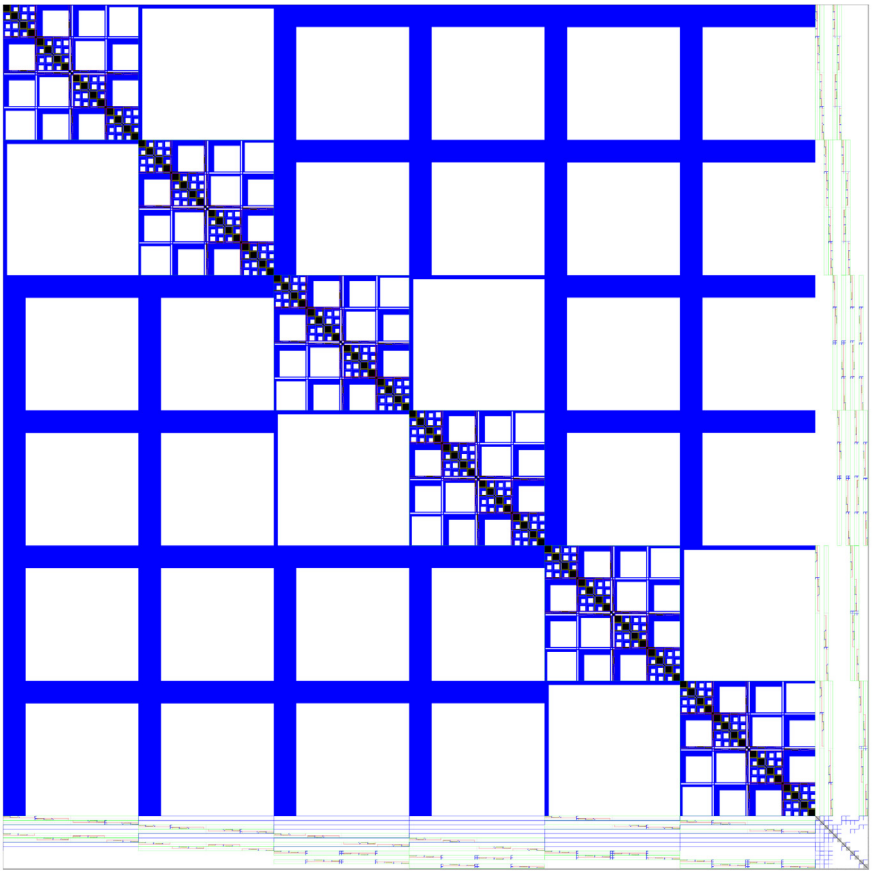


Fig. 7. Type II hierarchical division of the interaction matrix of a cube boundary (merged $S_{2;1,1,1}$) with 31×31 facets where the factorized form is used for all facet-facet interactions except self-interaction, switching from hierarchical to dense for 7×7 sub-facets.

algorithm with better scaling will provide better results but that may be for very large N (ignoring the $\log^2 N$ factor the growth rate of $\mathcal{O}(N^{4/3})$ means that if N increases by 8 then $N^{4/3}$ increases by about 16. Thus if the algorithm with worse scaling is C times faster for a particular N, it loses that advantage by the time N has increased by a factor of about C^3).

A similar analysis using the memory cost for hierarchical matrices, also given in [17] as $\mathcal{O}(rn \log n)$, shows that the matrix memory use for Type I is $\mathcal{O}(N)$ while that for Type II is $\mathcal{O}(N \log^2 N)$. Additional storage is required for information about the decomposition and nodes, for example, but this grows linearly.

6. Numerical results

All numerical tests are performed on a 2.13 GHz processor. The C++ implementation is purely serial utilizing only one core. 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 use it to 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 error over those 100 samples, for each of the following error measures

$$\text{Relative error } \frac{\|x - x^\star\|_2}{\|x^\star\|_2} \qquad \text{Energy error } \frac{\|x - x^\star\|_M}{\|x^\star\|_M} \qquad \text{Residual error } \frac{\|Mx - f\|_2}{\|M\|_2\|x^\star\|_2 + \|f\|_2},$$

where $\|x\|_M = \langle x, Mx\rangle$ is the norm induced by M.

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.

Table 2
A comparison of the Type I and Type II hierarchical matrix decomposition approach to solving $-\Delta u = f$ using $\epsilon_a = 10^{-12}$ and $\epsilon_r = 10^{-6}$ and $P = 4$. Memory use is shown relative to Type I with $N = 31^3$ and the runtimes are in seconds.

Q	N		Type I			Type II		
			Setup	Solve	Memory	Setup	Solve	Memory
3	31^3	29 791	270.61	0.78	1.0	97.23	0.44	0.9
4	63^3	250 047	3882.93	9.26	6.3	1268.40	4.92	6.6
5	127^3	2 048 383	43 359.12	89.52	55.5	15 072.42	47.12	55.0
Errors			Relative	Energy	Residual	Relative	Energy	Residual
		29 791	2.24e−07	3.12e−14	9.27e−10	4.76e−07	1.32e−13	1.91e−09
		250 047	3.77e−07	4.67e−14	3.87e−10	7.69e−07	3.55e−13	1.16e−09
		2 048 383	4.85e−07	5.98e−14	1.52e−10	1.11e−06	5.23e−13	4.90e−10

Table 3
A level by level breakdown of the time taken for the setup phase using the Type I hierarchical matrix decomposition approach to solving $-\Delta u = f$ using $\epsilon_a = 10^{-12}$ and $\epsilon_r = 10^{-6}$ and $P = 4$. The ratios between the runtimes for different N are given.

		Setup time contribution from calculations at each level			Scaling ratios	
					$31^3 \to 63^3$	$63^3 \to 127^3$
Level	Matrix format	$N = 31^3$	$N = 63^3$	$N = 127^3$	8.4	8.2
Leaf	Dense	2.23	18.63	159.40	8.4	8.6
$s = 7$	Hierarchical	16.71	141.35	1171.19	8.5	8.3
$s = 15$	Hierarchical	127.89	1351.53	12 009.16	10.6	8.9
$s = 31$	Hierarchical	**123.78**	1455.53	13 630.66	**11.8**	9.4
$s = 63$	Hierarchical	–	**915.88**	9902.98	–	**10.8**
$s = 127$	Hierarchical	–	–	**6485.64**	–	–

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.

Test 1. In this test, we solve $-\Delta u = f$ with zero Dirichlet boundary condition. In Table 2 we compare the runtimes obtained using the Type I and Type II hierarchical matrix decompositions with $\epsilon_a = 10^{-12}$ and $\epsilon_r = 10^{-6}$ and leaf level $P = 4$.

Now, the scaling observed in our experiments is close to the expected theoretical value for Type II but slightly off for Type I. One clear reason is that the zero boundary conditions tend to reduce the amount of work required and disrupt the expected scaling. If we consider the top level calculation with $s = 63$ when $N = 63^3$ we see that no calculation of S is actually required while in the corresponding level of $N = 127^3$ there are 8 calculations of S required. Even away from the top level we should not expect the runtime for each level to scale exactly by 8 because the proportion of cubes touching the domain boundary (which induce fewer calculations because of the zero boundary conditions) will not be the same when we double the number of cubes along each axis. The scaling observed for an analogous calculation without the zero boundary condition is much closer to the expected value.

In Table 3 we analyze the runtime scaling level by level for the Type I results of Table 2. We have broken down the time devoted to calculations on each level of the domain decomposition. Note that the number of levels used increases with N. We want to compare the increase in runtime with the increase in N as the problem size increases. We see that the ratio for the runtime devoted to the dense leaf level is close to that for N. The ratio for the first hierarchical level is also very good. Once we move to the next higher hierarchical level we see an increase in scaling. This is most pronounced at the last level for $N = 31^3$ (indicated in bold) which is not yet the last one for $N = 63^3$. The ratio for the same level for $63^3 \to 127^3$ is much closer to the theoretical value since we have not yet reached the last level. Note that this top level anomaly was 11.8 for the $31^3 \to 63^3$ size change but decreases to 10.8 for $63^3 \to 127^3$. Also, the new top level for each size increase contributes to the observed runtime being more than 8 times the previous size – at size $N = 63^3$ it is 24% of the runtime and at size $N = 127^3$ it is 15% – as the top level contributes a smaller and smaller fraction of the runtime for larger and larger N both these effects should diminish.

Table 4 reports the result of the same problem, but now with the side length of the leaf subdomain P equal to 8. We see that the runtime can be improved but at the cost of an increase in memory use. The difference in runtime between Type I and Type II has reduced. However, the scaling is not as good as in the earlier example. This may be because there are fewer hierarchical layers and so this example is further away from the asymptotic behavior.

Test 2. In this test, we solve $-\mathrm{div}(a(\mathbf{x})\nabla u) = f$ with a random $a(\mathbf{x})$, which is independently set at each node from a uniform distribution on $[10^{-3}, 10^3]$. In Table 5 we describe the results with again $\epsilon_a = 10^{-12}$ and $\epsilon_r = 10^{-6}$.

In the last set of results in Table 5 we use the Type I decomposition but a modified multiplication algorithm where block multiplication that results in a low rank factorized result for the sum of products is computed using a randomized

Table 4
A comparison of the Type I and Type II hierarchical matrix decomposition approach to solving $-\Delta u = f$ using $\epsilon_a = 10^{-12}$ and $\epsilon_r = 10^{-6}$ and $P = 8$. Memory use is shown relative to the Type I with $P = 4$ and $N = 31^3$ and the runtimes are in seconds.

Q	N		Type I			Type II		
			Setup	Solve	Memory	Setup	Solve	Memory
3	31^3	29 791	47.40	0.25	1.5	29.57	0.23	1.2
4	63^3	250 047	1097.78	3.05	9.2	520.45	1.87	8.4
5	127^3	2 048 383	15 369.18	30.79	77.4	7943.10	21.72	73.2

Table 5
To illustrate how we can handle non-identity $a(\mathbf{x})$ we use an $a(\mathbf{x})$ which is independently set at each node from a uniform distribution on $[10^{-3}, 10^3]$ with $\epsilon_a = 10^{-12}$ and $\epsilon_r = 10^{-6}$.

Q	N		Type I		Errors			Type II		Errors		
			Setup	Solve	Relative	Energy	Residual	Setup	Solve	Relative	Energy	Residual
3	31^3	29 791	47.26	0.26	2.75e−07	3.18e−14	8.50e−10	29.29	0.23	4.95e−07	1.74e−13	2.25e−09
4	63^3	250 047	1164.45	3.09	2.22e−06	7.57e−13	1.39e−09	539.57	2.43	1.51e−06	7.19e−13	1.56e−09
5	127^3	2 048 383	16 654.83	31.16	1.75e−05	1.48e−11	2.11e−09	8216.00	23.44	6.02e−06	6.70e−12	1.60e−09

Q	Prob. approx.		Type I		Errors		
	N		Setup	Solve	Relative	Energy	Residual
3	31^3	29 791	22.65	0.26	6.17e−07	6.83e−14	1.17e−09
4	63^3	250 047	565.35	3.11	2.07e−06	5.06e−13	1.10e−09
5	127^3	2 048 383	9709.58	31.13	1.71e−05	1.06e−11	1.80e−09

Table 6
A comparison of MUMPS and the Type II hierarchical matrix decomposition approach for *Test* 1 and *Test* 2. As before the runtimes are in seconds and the memory use is relative.

N		*Test* 1				*Test* 2			
		MUMPS		Type II		MUMPS		Type II	
		Setup	Mem.	Setup	Mem.	Setup	Mem.	Setup	Mem.
47^3	103 823	20.7	2.5			20.6	2.5		
63^3	250 047	114.5	8.3	540.7	8.4	118.3	8.3	539.6	8.5
95^3	857 375	1326.5	45.5			1389.5	45.5		
127^3	2 048 383	7775.0	153.0	7943.1	73.2	7780.5	153.0	8216.0	73.4

approach. The block multiplications involved in the Schur complement computation $S = C - BA^{-1}B^t$ have many parts to the inner sum (recall the boundary has 98 components, and the majority of entries in the full S matrix correspond to the 24 child facets) for each entry in the final product. So, for the entries in the hierarchical decomposition of S that are known to have low rank factorized form, we can circumvent the usual matrix multiplication procedure and use the idea of a probabilistic low rank approximation [18] to derive an approximation of those blocks in the product. This avoids deriving any intermediate approximations in the sum of products that the usual block matrix multiplication requires. To calculate the probabilistic low rank approximation one needs a good estimate of the expected rank but that can be derived from preliminary calculations using the existing approach. Notice that this makes the Type I approach more competitive with the Type II approach for our problem sizes.

6.1. Comparison with MUMPS

Now, in Table 6, we provide a comparison of the setup (factorization) stage with an existing multifrontal approach as implemented in MUMPS [19] using the SCOTCH ordering [20], which is based on recursive nested dissection. MUMPS was compiled for serial operation and run on one core to allow direct comparison with our results.

Note that runtime for MUMPS scales like $\mathcal{O}(N^2)$ while the memory use scales like $\mathcal{O}(N^{4/3})$ and the runtimes for random $a(\mathbf{x})$ are slightly higher than those for constant $a(\mathbf{x})$. While our approach is significantly slower for smaller sizes it is competitive at $N = 127^3$ with comparable runtimes to MUMPS and much better memory use.

7. Conclusion and future work

In this paper, we extend the work in [12] of combining nested dissection methods with hierarchical matrix algebra to the three dimensional elliptic problems on a Cartesian grid. Two approaches are proposed based on different choices of the hierarchical matrices employed. The solver using Type I hierarchical matrices has theoretically a linear complexity in

terms of the number of unknowns, while the second solver based on Type II hierarchical matrices has an $\mathcal{O}(N^{4/3}\log^2 N)$ construction cost and an $\mathcal{O}(N\log N)$ solution time for each right hand side.

Our approach can be extended to unstructured meshes of tetrahedra and more general discretization schemes as we detailed for the two dimensional case in [12]. The correct generalizations of corners and segments are required because the subdomains may not meet in precisely the same regular manner as in the Cartesian case. Not only will the lengths of segments vary on the same level but some corners (analogous to the *generalized corner* of [12]) will need to be composed of multiple nodes to recover the relationship between the leaf elements of the decomposition. Also, a *generalized segment* will be needed because the pattern of 4 subdomains of tetrahedra meeting precisely along a segment will not hold for unstructured meshes – they may meet at a node which is only in tetrahedra from 3 different subdomains.

The algorithm can be adjusted to improve performance in the situation 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 cubes containing the nodes with changed values would have to be recalculated. Similarly one could reduce the amount of re-calculation required in an h-adaptive mesh refinement system [21] as in [22] where new degrees of freedom could be incorporated incrementally and only the parts of the mesh which have been refined (and their parents in the decomposition tree) would require new calculations. The adaptive decomposition algorithm would have to be run alongside the h-refinement to distribute the new degrees of freedom and form new leaf cubes as required. Extending this idea to the degrees of freedom added and removed via a p-adaptive system one could eventually integrate with hp-adaptive systems.

We have not discussed parallelization of the calculations [23] but the hierarchical matrix algebra could be parallelized for the individual multiplications and additions. Also since all of the calculations on the same level are independent they could be performed in parallel. As long as there are more $\mathcal{D}_{q;i,j,k}$ per level than processors extensive inter-processor communication would be avoided. Only the uppermost levels would not lead to close to maximum speedup from parallelization. The possible gains of parallelization have been demonstrated by large scale parallel multifrontal solvers such as MUMPS [19].

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 [24] method. The support of the bases could also be increased at the cost of thicker border elements between subdomains.

Acknowledgements

This work was supported in part by L.Y.'s Alfred P. Sloan Research Fellowship and NSF CAREER award DMS-0846501.

References

[1] Y. Saad, Iterative Methods for Sparse Linear Systems, 2nd edition, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003.
[2] T.A. Davis, Direct Methods for Sparse Linear Systems, Fundam. Algorithms, vol. 2, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2006.
[3] P.R. Amestoy, T.A. Davis, I.S. Duff, An approximate minimum degree ordering algorithm, SIAM J. Matrix Anal. Appl. 17 (1996) 886–905.
[4] B. Hendrickson, E. Rothberg, Improving the run time and quality of nested dissection ordering, SIAM J. Sci. Comput. 20 (1998) 468–489.
[5] J.A. George, Nested dissection of a regular finite element mesh, SIAM J. Numer. Anal. 10 (1973) 345–363.
[6] I.S. Duff, J.K. Reid, The multifrontal solution of indefinite sparse symmetric linear equations, ACM Trans. Math. Softw. 9 (1983) 302–325.
[7] J.W.H. Liu, The multifrontal method for sparse matrix solution: Theory and practice, SIAM Rev. 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 J. Matrix Anal. Appl. 31 (2009) 1382–1411.
[9] P.-G. Martinsson, A fast direct solver for a class of elliptic partial differential equations, J. Sci. Comput. 38 (2009) 316–330.
[10] W. Hackbusch, L. Grasedyck, S. Börm, An introduction to hierarchical matrices, Technical Report, 21 Max-Plank-Institut für Mathematik in den Naturwissenschaften, Leipzig, 2001.
[11] J. Xia, S. Chandrasekaran, M. Gu, X.S. Li, Fast algorithms for hierarchically semiseparable matrices, Numer. Linear Algebra Appl. (2009).
[12] P.G. Schmitz, L. Ying, A fast direct solver for elliptic problems on general meshes in 2D, J. Comput. Phys. 231 (2012) 1314–1338.
[13] P.G. Martinsson, V. Rokhlin, A fast direct solver for boundary integral equations in two dimensions, J. Comput. Phys. 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 Numer. 18 (2009) 243–275.
[15] I.S. Duff, A.M. Erisman, J.K. Reid, Direct Methods for Sparse Matrices, 2nd edition, Monogr. Numer. Anal., The Clarendon Press/Oxford University Press, New York, 1989, Oxford Science Publications.
[16] W. Hackbusch, B.N. Khoromskij, R. Kriemann, Hierarchical matrices based on a weak admissibility criterion, Computing 73 (2004) 207–243.
[17] L. Grasedyck, W. Hackbusch, Construction and arithmetics of $\mathcal{H}$-matrices, Computing 70 (2003) 295–334.
[18] E. Liberty, F. Woolfe, P.-G. Martinsson, V. Rokhlin, M. Tygert, Randomized algorithms for the low-rank approximation of matrices, Proc. Natl. Acad. Sci. USA 104 (2007) 20167–20172.
[19] P.R. Amestoy, I.S. Duff, C. Vömel, Task scheduling in an asynchronous distributed memory multifrontal solver, SIAM J. Matrix Anal. Appl. 26 (2004/05) 544–565.
[20] F. Pellegrini, SCOTCH 5.1 user's guide, Technical Report, LaBRI, Université Bordeaux I, 2010.
[21] L. Demkowicz, J. Kurtz, D. Pardo, M. Paszyński, W. Rachowicz, A. Zdunek, Computing with hp-Adaptive Finite Elements, vol. 2. Frontiers: Three Dimensional Elliptic and Maxwell Problems with Applications, Chapman & Hall/CRC Appl. Math. Nonlin. Sci. Ser., Chapman & Hall/CRC, Boca Raton, FL, 2008.
[22] L. Grasedyck, W. Hackbusch, S. Le Borne, Adaptive geometrically balanced clustering of $\mathcal{H}$-matrices, Computing 73 (2004) 1–23.
[23] 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.
[24] D.N. Arnold, F. Brezzi, B. Cockburn, L.D. Marini, Unified analysis of discontinuous Galerkin methods for elliptic problems, SIAM J. Numer. Anal. 39 (2001/2002) 1749–1779.