Comput Visual Sci (2008) 11:273–291
DOI 10.1007/s00791-008-0098-9

REGULAR ARTICLE

Parallel black box $\mathscr{H}$-LU preconditioning for elliptic boundary value problems

Lars Grasedyck · Ronald Kriemann · Sabine Le Borne

Received: 15 October 2007 / Accepted: 10 January 2008 / Published online: 1 April 2008

Abstract Hierarchical ($\mathscr{H}$-) matrices provide a data-sparse way to approximate fully populated matrices. The two basic steps in the construction of an $\mathscr{H}$-matrix are (a) the hierarchical construction of a matrix block partition, and (b) the blockwise approximation of matrix data by low rank matrices. In the context of finite element discretisations of elliptic boundary value problems, $\mathscr{H}$-matrices can be used for the construction of preconditioners such as approximate $\mathscr{H}$-LU factors. In this paper, we develop a new black box approach to construct the necessary partition. This new approach is based on the matrix graph of the sparse stiffness matrix and no longer requires geometric data associated with the indices like the standard clustering algorithms. The black box clustering and a subsequent $\mathscr{H}$-LU factorisation have been implemented in parallel, and we provide numerical results in which the resulting black box $\mathscr{H}$-LU factorisation is used as a preconditioner in the iterative solution of the discrete (three-dimensional) convection-diffusion equation.

Keywords Hierarchical matrices · Black box clustering · Preconditioning · LU

Mathematics Subject Classification (2000) 65F05 · 65F30 · 65F50 · 65N55

Dedicated to Wolfgang Hackbusch on the occasion of his 60th birthday.

Communicated by G. Wittum.

The work was supported in part by the US Department of Energy under Grant No. DE-FG02-04ER25649 and by the National Science Foundation under grant No. DMS-0408950.

L. Grasedyck (✉) · R. Kriemann
Max-Planck-Institute for Mathematics in the Sciences,
04103 Leipzig, Germany
e-mail: lgr@mis.mpg.de

R. Kriemann
e-mail: rok@mis.mpg.de

S. Le Borne
Department of Mathematics, Tennessee Technological University,
Box 5054, Cookeville, TN 38505, USA

1 Introduction

Hierarchical ($\mathscr{H}$-) matrices have first been introduced in 1999 [15] and since then have entered into a wide range of applications. They provide a format for the data-sparse representation of fully populated matrices. The key idea is to reorder the matrix rows and columns so that certain sub-blocks of the reordered matrix can be approximated by low-rank matrices. These low-rank matrices can be represented by a product of two rectangular matrices as follows: let $A \in \mathbb{R}^{n\times n}$ with $\operatorname{rank}(A) = k$ and $k \ll n$. Then there exist matrices $B, C \in \mathbb{R}^{n\times k}$ such that $A = BC^T$. Whereas A has n^2 entries, B and C together have $2kn$ entries which results in significant savings in storage if $k \ll n$. A new $\mathscr{H}$-matrix arithmetic has been developed which allows exact matrix–vector multiplication and approximate matrix(-matrix) operations such as addition, multiplication, inversion and LU factorisation in this format in nearly optimal complexity $\mathscr{O}(n \log^{\alpha} n)$ with a moderate parameter α [10].

In finite element methods, the stiffness matrix is sparse but its LU factors are fully populated and can be approximated by an $\mathscr{H}$-matrix. Such approximate $\mathscr{H}$-LU factors may then be used as a preconditioner in iterative methods [13,22].

In most of the previous papers on $\mathscr{H}$-matrices [3,10,13], the construction of the $\mathscr{H}$-matrix block structure is based on geometric information associated with the underlying indices. Each index is associated with its basis function and a (rectangular bounding box) of the support of the

basis function. The standard geometric clustering algorithms, which include the bisection as well as the nested dissection clustering, compute Euclidean diameters and distances based on these geometric entities in order to construct the block partition of the $\mathscr{H}$-matrix format.

In this paper, we introduce an algebraic clustering algorithm that is applicable to *sparse* matrices and only needs the matrix itself as input. A matrix graph is constructed based on the sparsity structure of the matrix, and the subsequent algebraic clustering algorithm is based on this matrix graph. We therefore obtain an algorithm for an algebraic $\mathscr{H}$-matrix construction that is similar to algebraic multigrid (AMG) techniques [5,6,14,26]. A related black box clustering approach based on heavy edge matching has also been developed in [21,24].

Given an $\mathscr{H}$-matrix format, we can convert the sparse stiffness matrix into an $\mathscr{H}$-matrix and compute its $\mathscr{H}$-LU factorisation. This yields a preconditioner to accelerate the iterative solution of the linear system of equations. We will apply the resulting black box preconditioner in the iterative solution of convection-dominated partial differential equations, providing comparisons with standard $\mathscr{H}$-LU factorisation based on geometric clusterings as well as the direct solvers PARDISO [27–29] and UMFPACK [7].

The remainder of this paper is structured as follows: Sect. 2 is devoted to preliminaries: it will provide an introduction of the model partial differential equation and a brief introduction to the construction and arithmetics of $\mathscr{H}$-matrices. Section 3 introduces the new black box clustering algorithm. It begins with a simple, motivating example and then continues with the general case. Section 4 deals with the parallel implementation of the $\mathscr{H}$-LU factorisation based on black box nested dissection clustering, and Sect. 5 provides numerical results for the new approaches in comparison with standard geometric $\mathscr{H}$-matrix techniques as well as the PARDISO and the UMFPACK solver.

This article is dedicated to Wolfgang Hackbusch on the occasion of his sixtieth birthday.

2 Preliminaries: the model problem and $\mathscr{H}$-Matrices

2.1 The finite element model problem

Throughout this paper, we consider a linear system of equations of the form $Au = b$, where A is the sparse Galerkin stiffness matrix of an invertible second order uniformly elliptic partial differential operator $\mathscr{A} : H_0^1(\Omega) \to H^{-1}(\Omega)$,

$$\mathscr{A}u = -\mathrm{div}\sigma\nabla u + b\cdot\nabla u + cu, \tag{1}$$

on a domain $\Omega \subset \mathbb{R}^d$ with L^∞-coefficients $\sigma : \Omega \to \mathbb{R}^{d\times d}$, $b : \Omega \to \mathbb{R}^d$, $c : \Omega \to \mathbb{R}$. The N-dimensional finite element space is denoted by $V_N \subset H_0^1(\Omega)$ and is spanned by a local basis $(\varphi_i)_{i\in\mathscr{I}}$ with index set $\mathscr{I} := \{1,\dots,N\}$, where the term "local" is defined as follows:

Assumption 1 *(Locality)* We assume that for the basis functions $(\varphi_i)_{i\in\mathscr{I}}$ the supports $\Theta_i = \mathrm{supp}(\varphi_i)$ are locally separated in the sense that there exist two constants C_{sep} and $n_{\min}$ so that

$$\max_{i\in\mathscr{I}} \#\left\{ j\in\mathscr{I} \;\middle|\; \mathrm{dist}(\Theta_i,\Theta_j) \le C_{\mathrm{sep}}^{-1}\,\mathrm{diam}(\Theta_i)\right\} \le n_{\min}. \tag{2}$$

The left-hand side is the maximal number of basis functions with 'relatively close' supports.

Remark 1
1. The stiffness matrix A is sparse with at most $Nn_{\min}$ non-zero entries.
2. The locality condition (2) does not require shape regularity or a K-mesh property (neighboured elements are of comparable size). On the other hand, it bounds the number of non-neighboured elements that are close to each other in $\mathbb{R}^d$.

We will define geometric entities which are required in the original $\mathscr{H}$-matrix constructions but will no longer be required for our new black box clustering approach.

Definition 1 *(Geometric entities)* Every index $i \in \mathscr{I}$ is associated with a basis function φ_i of the underlying finite element space V_N. For every i, we assign a (fixed) nodal point x_i such that

$$x_i \in \mathrm{supp}\varphi_i. \tag{3}$$

For a cluster (i.e. subset) $v \subset \mathscr{I}$ of indices, we define its support by

$$\Omega_v := \bigcup_{j\in v} \mathrm{supp}\varphi_j. \tag{4}$$

The geometric $\mathscr{H}$-matrix construction (see Subsect. 2.2) needs (upper bounds of) the diameters of these clusters as well as the distances between two such clusters (both in the Euclidean norm). Since diameters and distances can be computed much more efficiently for rectangular boxes than for arbitrarily shaped domains, we supply each cluster v with a bounding box

$$B_v = \bigotimes_{j=1}^{d} [\alpha_{v,j}, \beta_{v,j}] \tag{5}$$

that contains Ω_v, i.e. $\Omega_v \subset B_v$.

2.2 A brief introduction to $\mathscr{H}$-Matrices

In this section, we will review $\mathscr{H}$-matrices and their arithmetic. An $\mathscr{H}$-matrix provides a data-sparse approximation to a dense matrix by replacing certain blocks of the matrix

by matrices of low rank which can be stored very efficiently. The blocks which allow for such low rank representations are selected from a hierarchy of partitions organised in a so-called cluster tree.

Definition 2 *(Cluster tree)*
Let $T_{\mathscr{I}} = (V, E)$ be a tree with vertex set V and edge set E. For a vertex $v \in V$, we define the set of successors (or sons) of v as $S(v) := \{w \in V \mid (v, w) \in E\}$. Correspondingly, the predecessor (or father) of a non-root vertex v is defined as the unique vertex $F(v)$ s.t. $(F(v), v) \in E$. The tree $T_{\mathscr{I}}$ is called a cluster tree of $\mathscr{I}$ if its vertices consist of subsets of $\mathscr{I}$ and satisfy the following conditions (cf. Fig. 1 (left)):

1. $\mathscr{I} \in V$ is the root of $T_{\mathscr{I}}$, and $v \subset \mathscr{I}$, $v \neq \emptyset$, for all $v \in V$.
2. For all $v \in V$, there holds $S(v) = \emptyset$ or $v = \dot{\bigcup}_{w \in S(v)} w$.

The depth of a cluster tree, $d(T_{\mathscr{I}})$, is defined as the length of the longest path in $T_{\mathscr{I}}$. In the following, we identify V and $T_{\mathscr{I}}$, i.e. we write $v \in T_{\mathscr{I}}$ instead of $v \in V$. The nodes $v \in V$ are called clusters. The nodes with no successors are called *leaves* and define the set $\mathscr{L}(T_{\mathscr{I}}) := \{v \in T_{\mathscr{I}} \mid S(v) = \emptyset\}$.

In previous papers, several strategies have been introduced to construct a cluster tree from a given index set, e.g. bisection or nested dissection, but most of these constructions are based on the underlying geometric entities defined in Definition 1. As an example, we will review the geometric bisection clustering. Here, a cluster v with support Ω_v (4) is subdivided into two smaller clusters v_1, v_2 as follows:

1. Let Q_v denote a box that contains all nodal points $(x_i)_{i \in v}$, cf. (3). For the root cluster this could be the bounding box $Q_{\mathscr{I}} := B_{\mathscr{I}}$.
2. Subdivide the box Q_v into two boxes $Q_v = Q_1 \dot{\cup} Q_2$ of equal size.
3. Define the two successors $S(v) = \{v_1, v_2\}$ of v by

$$v_1 := \{i \in v \mid x_i \in Q_1\}, \quad v_2 := \{i \in v \mid x_i \in Q_2\}$$

and use the boxes $Q_{v_1} := Q_1$, $Q_{v_2} := Q_2$ for the further subdivision of the sons.

The subdivision is typically performed such that the resulting diameters of the boxes associated with successor clusters become as small as possible. A single step of geometric bisection is illustrated in Fig. 2 where a cluster v consisting of 17 vertices is subdivided into clusters v_1, v_2 consisting of 8 and 9 vertices lying in Q_{v_1} and Q_{v_2}, resp. Here, the subdivision into v_1 and v_2 is based on the geometric locations associated with the indices.

Given a cluster tree $T_{\mathscr{I}}$, any two clusters $s, t \in T_{\mathscr{I}}$ form a product $s \times t$, also called a *block cluster*, which can be associated with the corresponding matrix block $(A_{ij})_{i \in s, j \in t}$ (cf. Fig. 1 (right)). We will use an admissibility condition to decide whether such a block will be allowed in a block partition of the matrix A or will be further subdivided. In general, an admissibility condition is a Boolean function

$$\mathrm{Adm} : T_{\mathscr{I}} \times T_{\mathscr{I}} \to \{\mathtt{true}, \mathtt{false}\}.$$

For cluster trees based on the underlying geometry, typical admissibility conditions use geometric information, e.g. the standard admissibility condition is given by

$$\mathrm{Adm}_{\mathrm{S}}(s \times t) = \mathtt{true} \quad :\Leftrightarrow \quad \min(\mathrm{diam}(B_s), \mathrm{diam}(B_t)) \leq \eta \ \mathrm{dist}(B_s, B_t) \tag{6}$$

for some $0 < \eta$. Here, B_s, B_t are the bounding boxes (5) of the clusters s, t, resp., and the distance and diameters are computed with respect to the Euclidean norm.

Given a cluster tree $T_{\mathscr{I}}$ and an admissibility condition, we construct a hierarchy of block partitionings of the product index set $\mathscr{I} \times \mathscr{I}$. The hierarchy forms a tree structure and is organised in a *block cluster tree* $T_{\mathscr{I} \times \mathscr{I}}$:

Definition 3 *(Block cluster tree)* Let $T_{\mathscr{I}}$ be a cluster tree of the index set $\mathscr{I}$. A cluster tree $T_{\mathscr{I} \times \mathscr{I}}$ is called a block cluster tree (based upon $T_{\mathscr{I}}$) if for all $v \in T_{\mathscr{I} \times \mathscr{I}}$ there exist $s, t \in T_{\mathscr{I}}$ such that $v = s \times t$. The nodes $v \in T_{\mathscr{I} \times \mathscr{I}}$ are called block clusters.

A block cluster tree may be constructed from a given cluster tree in the canonical way defined by Algorithm 1 (cf. Fig. 1), which we will employ for all cluster trees constructed in this paper.

Algorithm 1 Canonical block cluster tree construction

```
procedure bct_construct( s, t, Adm(·), n_min )
  if Adm(s × t) = true ∨ min{#s, #t} ≤ n_min then
    S(s × t) := ∅;
  else
    for all s' ∈ S(s) do
      for all t' ∈ S(t) do
        S(s × t) := S(s × t) ∪ {bct_construct(s', t', Adm(·), n_min)};
      end for
    end for
  end if
  return s × t;
end
```

The parameter $n_{\min}$ (from Assumption 1) has to be chosen large enough to fulfil locality condition (2). For rather small blocks, the matrix arithmetic of a full matrix is more efficient than that of a structured matrix. Therefore, $n_{\min}$ should be chosen at least $n_{\min} \geq 10$, which is typically at the same time sufficient for Assumption 1.

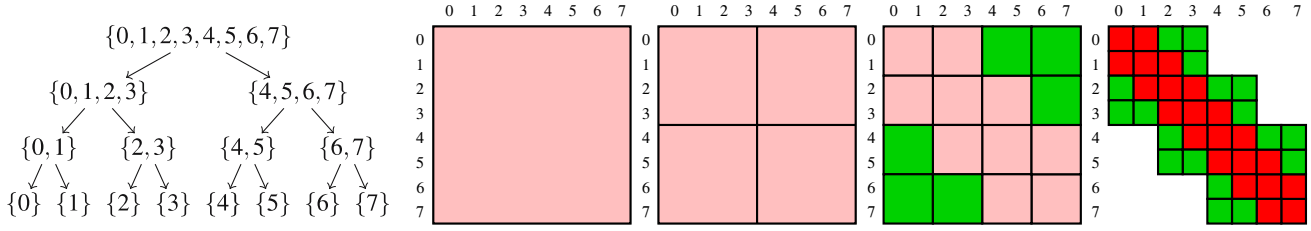

Fig. 1 *Left*: A cluster tree $T_{\mathcal{I}}$. *Right*: The four levels of the block cluster tree $T_{\mathcal{I}\times\mathcal{I}}$, where nodes that are further refined are *light red*, inadmissible leaves are *red*, and admissible leaves are *green*

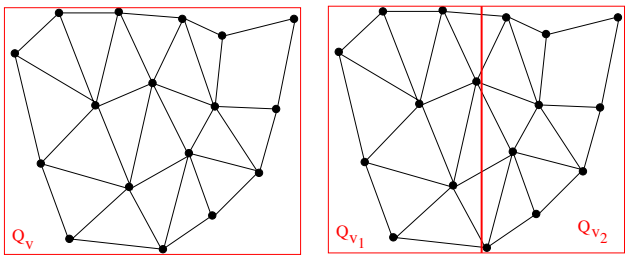

Fig. 2 Geometric bisection

The leaves of a block cluster tree obtained through this construction yield a disjoint partition of the product index set $\mathcal{I} \times \mathcal{I}$.

In Fig. 1, we provide a simple example for a cluster tree and the corresponding block cluster tree. The indices in this example correspond to the continuous, piecewise linear basis functions of a regularly refined unit interval (in lexicographical order).

Matrix blocks which correspond to admissible block clusters will be approximated in a data-sparse format by the following Rk-matrix representation.

Definition 4 *(Rk-matrix representation)* Let $k, n, m \in \mathbb{N}_0$. Let $M \in \mathbb{R}^{n\times m}$ be a matrix of at most rank k. A representation of M in factorised form

$$M = AB^T, \quad A \in \mathbb{R}^{n\times k}, B \in \mathbb{R}^{m\times k}, \tag{7}$$

with A and B stored in full matrix representation, is called an Rk-matrix representation of M, or, in short, we call M an Rk-matrix.

If the rank k is small compared to the matrix size given by n and m, we obtain considerable savings in the storage and work complexities of an Rk-matrix compared to a full matrix [10].

Finally, we can introduce the definition of a hierarchical matrix:

Definition 5 *($\mathcal{H}$-matrix)* Let $k, n_{\min} \in \mathbb{N}_0$. The set of $\mathcal{H}$-matrices induced by a block cluster tree $T := T_{\mathcal{I}\times\mathcal{I}}$ with blockwise rank k and minimum block size $n_{\min}$ is defined by

$$\mathcal{H}(T,k) := \{M \in \mathbb{R}^{\mathcal{I}\times\mathcal{I}} \mid \forall s \times t \in \mathcal{L}(T) : \operatorname{rank}(M|_{s\times t}) \le k \text{ or } \min\{\#s, \#t\} \le n_{\min}\}. \tag{8}$$

Blocks $M|_{s\times t}$ with $\operatorname{rank}(M|_{s\times t}) \le k$ are stored as Rk-matrices whereas all other blocks are stored as full matrices.

Whereas the classical $\mathcal{H}$-matrix uses a fixed rank for the Rk-blocks, it is possible to replace it by *variable* (*or adaptive*) *ranks* in order to enforce a desired relative accuracy within the individual blocks [10]. In particular, in the adaptive setting, for a given admissible block $s \times t$, we set the rank $k = k(M|_{s\times t})$ of the corresponding matrix block $M|_{s\times t}$ as follows:

$$k(M|_{s\times t}) := \min\{k' \in \mathbb{N}_0 \mid \sigma_{k'+1} \le \delta\sigma_1\} \tag{9}$$

where σ_i denotes the i-th largest singular value of $M|_{s\times t}$, and $0 < \delta < 1$ denotes the desired relative accuracy within each block.

2.3 Arithmetic of $\mathcal{H}$-matrices

Given two $\mathcal{H}$-matrices $A, B \in \mathcal{H}(T,k)$ based on the same block cluster tree T, i.e. with the same block structure, the exact sum or product of these two matrices will typically not belong to $\mathcal{H}(T,k)$. In the case of matrix addition, we have $A + B \in \mathcal{H}(T,2k)$; the rank of an exact matrix product is less obvious. We will use a truncation operator $\mathcal{T}^{\mathcal{H}}_{k\leftarrow k'}$ to define the $\mathcal{H}$-matrix addition $C := A \oplus_{\mathcal{H}} B$ and $\mathcal{H}$-matrix multiplication $C := A \otimes_{\mathcal{H}} B$ such that $C \in \mathcal{H}(T,k)$.

A truncation of a rank k' matrix R to rank $k < k'$ is defined as the best approximation with respect to the Frobenius (or spectral) norm in the set of rank k matrices. In the context of $\mathcal{H}$-matrices, we use such truncations for all admissible (rank k') blocks. Using truncated versions of the QR-decomposition and singular value decomposition, the truncation of a rank k' matrix $R \in \mathbb{R}^{n,m}$ (given in the form $R = AB^T$ where $A \in \mathbb{R}^{n,k'}$ and $B \in \mathbb{R}^{m,k'}$) to a lower rank can be computed with complexity $\mathcal{O}\left((k')^2(n+m)\right)$; further details are provided in [10]. We then define the $\mathcal{H}$-matrix addition and multiplication as follows:

$$A \oplus_{\mathcal{H}} B := \mathcal{T}^{\mathcal{H}}_{k\leftarrow 2k}(A+B);$$
$$A \otimes_{\mathcal{H}} B := \mathcal{T}^{\mathcal{H}}_{k\leftarrow k'}(A \cdot B)$$

where $k' \le c(p+1)k$ is the rank of the exact matrix product, c denotes some constant (which depends on the block

cluster tree T) and p denotes the depth (Definition 2) of the tree. Estimates that show that the $\mathscr{H}$-matrix addition and multiplication have almost optimal complexity for typical $\mathscr{H}$-structures are provided in [10] along with details on the efficient implementation of these operations. The $\mathscr{H}$-matrix addition and multiplication are operations required to define an $\mathscr{H}$-inversion as well as an $\mathscr{H}$-LU factorisation recursively in the block structure. Details on these algorithms can be found in [1,2,10,13]. We will provide a parallel version of the $\mathscr{H}$-LU factorisation (including auxiliary routines) in Sect. 4.

3 Black box clustering for sparse matrices

$\mathscr{H}$-matrices are based on a block cluster tree $T_{\mathscr{I}\times\mathscr{I}}$ that describes the hierarchical partition of a matrix into admissible and inadmissible blocks. The formatted arithmetic in the $\mathscr{H}$-matrix format requires only this partition but not the geometric information (i.e. cluster diameters and distances) by which the cluster tree $T_{\mathscr{I}}$ was built.

In some applications geometric information might not be available. Instead, only the already assembled sparse stiffness matrix A is provided. In this case, we will extract information on the connectivity of the indices from the matrix directly, or rather from its matrix graph, as is defined next, to construct a suitable partition of the index set.

Definition 6 *(Matrix graph)* The (directed) graph $\mathscr{G}(A) = (V_A, E_A)$ of a matrix $A \in \mathbb{R}^{\mathscr{I}\times\mathscr{I}}$ is defined by the vertex and edge sets

$$V_A := \mathscr{I}, \quad E_A := \{(i,j) \in \mathscr{I}\times\mathscr{I} \mid i \neq j \wedge A_{ij} \neq 0\}.$$

The restriction $\mathscr{G}(A)|_{V'}$ to a subset $V' \subseteq V_A$ is defined by $\mathscr{G}(A)|_{V'} := (V', E_A \cap V'\times V')$. Furthermore, for $v \in V_A$, we call $\#\{u \in V_A \mid (u,v) \in E_A \vee (v,u) \in E_A\}$ the *degree* of the node v.

We say that $\mathscr{G}$ is *connected* if there is a path $i{=}j_0, \ldots, j_\ell{=}j$ with $(j_\nu, j_{\nu+1}) \in E_A$ from every node $i \in \mathscr{I}$ to every other node $j \in \mathscr{I}\backslash\{i\}$. The *length* of such a path is ℓ.

Subdividing the index set then corresponds to partitioning the matrix graph $\mathscr{G}(A)$.

In order to simplify the algorithms and presentation, we use the symmetrised matrix graph

$$\begin{aligned}&\mathscr{G}_{\mathrm{sym}}(A) = (V,E), \qquad V := \mathscr{I},\\ &E := \{(i,j) \in \mathscr{I}\times\mathscr{I} \mid i \neq j \wedge (A_{ij} \neq 0 \vee A_{ji} \neq 0)\}.\end{aligned} \tag{10}$$

In Remark 5, we will comment on non-symmetric matrix graphs.

3.1 Breadth first search clustering

We will first show the close relationship between graph partitioning and geometric clustering in our model problem of a regular grid which will serve as a motivation for the subsequent black box clustering. Let A be the stiffness matrix resulting from the finite element discretisation of the Poisson problem on the (regularly triangulated) unit square $[0,1]^2$. Let h denote the grid-width. Figure 3 shows the connectivity of the indices and therefore the matrix graph $\mathscr{G}(A) = \mathscr{G}_{\mathrm{sym}}(A)$ of A.

Remark 2 Let $i, j \in V_A$ be two nodes of $\mathscr{G}(A)$ and let $x_i, x_j \in \mathbb{R}^d$ denote the nodal points associated with the indices i, j (see Definition 1). Then, $a_{ij} \neq 0$ implies $\operatorname{supp}\varphi_i \cap \operatorname{supp}\varphi_j \neq \emptyset$ which in turn implies $\|x_i - x_j\| \leq ch$, for some constant $c \geq 0$.

For our model problem with a uniform tensor product grid in $\mathbb{R}^2$ we have: $a_{ij} \neq 0 \implies \|x_i - x_j\| \leq h$. Furthermore, if i and j are connected by a path of length ℓ, we can bound the Euclidean distance by $\|x_i - x_j\| \leq \ell h$.

For an arbitrary path between i and j this bound is usually too large and only the shortest path should be considered. This example shows that in the case of a regular grid, we can estimate the distance between two nodes by computing the (shortest) path length connecting these nodes, without knowledge of the geometrical data associated with these nodes.

Motivated by this example, we will now develop a black box clustering technique which no longer requires geometric information. Instead of the geometrical (i.e. Euclidean) distance and diameter of clusters, the path lengths in the matrix-graph are used:

Definition 7 (*Distance and Diameter in a Graph*) Let $\mathscr{G}(A)$ be a connected graph and let $i, j \in V_A$ with $i = j_0, \ldots, j_\ell = j$, $(j_\nu, j_{\nu+1}) \in E_A$, $0 \leq \nu < \ell$, be the shortest path in $\mathscr{G}(A)$ from i to j. Then, we define the distance between i and j as $\operatorname{dist}_{\mathscr{G}(A)}(i,j) := \ell$. For two sets of nodes $I_1, I_2 \subseteq V_A$ the

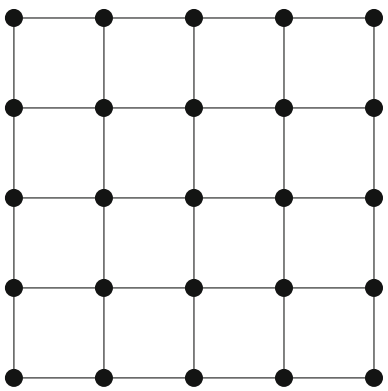

Fig. 3 Matrix graph in the model problem

distance $\mathrm{dist}_{\mathscr{G}(A)}(I_1, I_2)$ is defined as

$$\mathrm{dist}_{\mathscr{G}(A)}(I_1, I_2) := \min_{i' \in I_1, j' \in I_2} \mathrm{dist}_{\mathscr{G}(A)}(i', j').$$

Furthermore, we define the *diameter* of $\mathscr{G}(A)$ as

$$\mathrm{diam}(\mathscr{G}(A)) := \max_{i', j' \in V_A} \mathrm{dist}_{\mathscr{G}(A)}(i', j').$$

The computation of the shortest path between two nodes in a connected graph is performed in $\mathscr{O}(\#E_A)$ by *breadth first search* (or BFS), which starts with one node and successively extends the current set of nodes by the set of all adjacent nodes until the destination is reached. Since only direct neighbours are included in the next iteration step, the visited set of nodes in a BFS has a small diameter in the graph, which is the wanted property for the resulting sub-graphs during graph partitioning.

However, the straightforward approach of using a single start node for BFS may lead to an unsuitable partitioning (e.g. one sub-graph being surrounded by the other sub-graph). We avoid this situation by using two start nodes $u, v \in V_A$ which are chosen with a maximal distance. We perform the BFS algorithm for both nodes simultaneously (or rather, alternatingly). The choice of u and v ensures similar diameters of the resulting sub-graphs. In Fig. 4, this process is illustrated with u and v being the lower left (red) and the upper right (blue) node, respectively. In each step of the BFS algorithm, the initial node sets $V_v = \{v\}$ and $V_u = \{u\}$ are expanded by the set of adjacent nodes

$$V_v := V_v \cup \bigcup_{v' \in V_v} \left\{ i \in V_A \backslash (V_u \cup V_v) \mid (v', i) \in E_A \right\} \text{ and}$$

$$V_u := V_u \cup \bigcup_{u' \in V_u} \left\{ i \in V_A \backslash (V_u \cup V_v) \mid (u', i) \in E_A \right\}$$

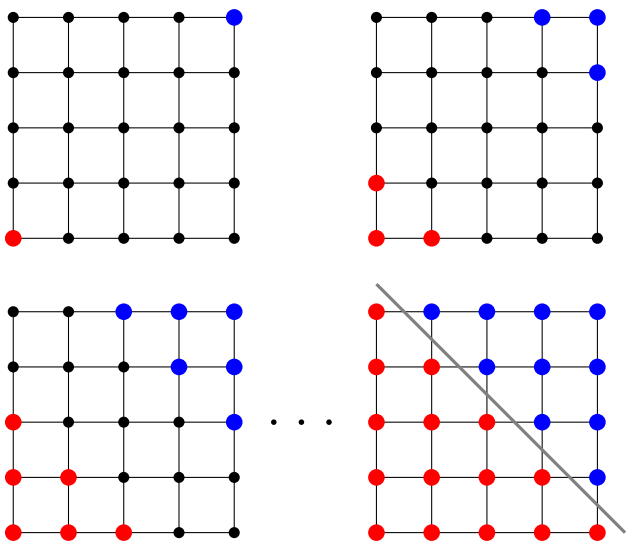

Fig. 4 Different levels of graph partitioning via BFS

until all nodes of the graph have been visited, i.e. $V_u \cup V_v = V_A$.

It should be noted, that since $\mathscr{G}(A)$ is connected, the sub-graphs $\mathscr{G}(A)|_{V_v}$ and $\mathscr{G}(A)|_{V_u}$ are also connected. Furthermore, the results of this partitioning method are typically different from the results obtained by geometrical partitioning, e.g. in the situation depicted in Fig. 4 the final separation plane is not axis aligned.

Algorithm 2 BFS graph partitioning

```
procedure bb_bfs_part( 𝒢 = (V, E), v, u )
  V' := V \ {v, u}; V_v := {v}; V_u := {u};
  while V' ≠ ∅ do
    N_v := ⋃_{w∈V_v} {w' ∈ V' | (w, w') ∈ E};
    V_v := V_v ∪ N_v; V' := V' \ N_v;
    N_u := ⋃_{w∈V_u} {w' ∈ V' | (w, w') ∈ E};
    V_u := V_u ∪ N_u; V' := V' \ N_u;
  end while;
  return {V_v, V_u};
end
```

By recursively applying Algorithm 2 to the resulting sub-graphs $\mathscr{G}(A)|_{V_v}$ and $\mathscr{G}(A)|_{V_u}$, the index set $\mathscr{I}$ is hierarchically subdivided into a cluster tree. The next steps for the example in Fig. 4 are shown in Fig. 5.

Finding two nodes in a graph with maximal distance usually requires quadratic complexity $\mathscr{O}\left(\#V_A^2\right)$. Therefore, the following heuristic approach will be used to determine nodes with a large, almost maximal distance: We choose an arbitrary node $i_0 \in V_A$ and compute via BFS a node $i_1 \in V_A$ with maximal distance to i_0, i.e. $\mathrm{dist}_{\mathscr{G}(A)}(i_0, i_1) = \max_{j \in V_A} \mathrm{dist}_{\mathscr{G}(A)}(i_0, j)$, which costs $\mathscr{O}(\#V_A)$. Afterwards, this process is repeated for i_1, i.e. a node $i_2 \in V_A$ is determined with maximal distance to i_1. This can be repeated a fixed number $n_{\mathrm{BFS}} > 0$ of steps or until the distance between i_ℓ and $i_{\ell+1}$ no longer grows. In Fig. 6, this procedure is illustrated by an example, whereas Algorithm 3 implements the described method.

Finally, the complete algorithm for partitioning a given graph based on BFS and simultaneously building a cluster tree for the given index set is provided in Algorithm 4.

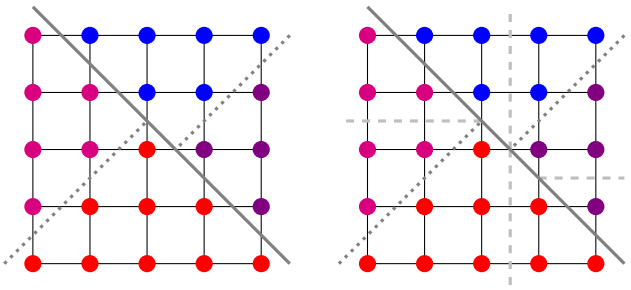

Fig. 5 Next levels of cluster tree construction

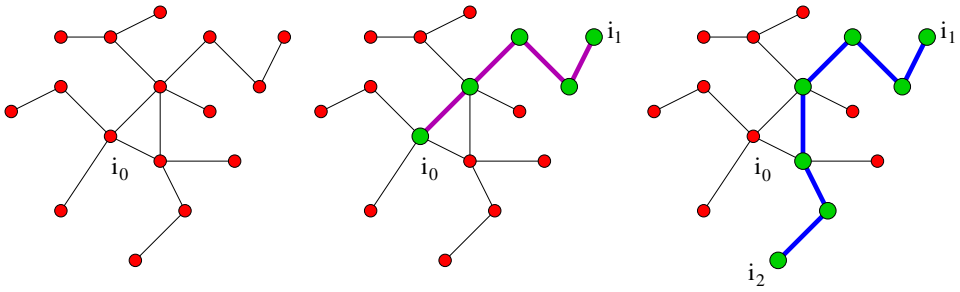

Fig. 6 Determining start nodes for BFS graph partitioning

Algorithm 3 Determine start nodes in BFS black box clustering

```
procedure bb_bfs_start( 𝒢 = (V, E) )
  choose i_0 ∈ V; determine i_1 ∈ V with dist_𝒢(i_0, i_1) = max_{j∈V} dist_𝒢(i_0, j);
  for 1 ≤ ℓ ≤ n_BFS do
    determine i_{ℓ+1} ∈ V with dist_𝒢(i_ℓ, i_{ℓ+1}) = max_{j∈V} dist_𝒢(i_ℓ, j);
    if dist_𝒢(i_{ℓ−1}, i_ℓ) = dist_𝒢(i_ℓ, i_{ℓ+1}) then
      return {i_{ℓ−1}, i_ℓ};
    end if
  end for
  return {i_ℓ, i_{ℓ+1}};
end
```

Algorithm 4 Black Box BFS-Clustering

```
procedure bb_ct_build_bfs( 𝒢 = (V, E) )
  if #V ≤ n_min then
    return cluster t := V;
  else
    {v, u} := bb_bfs_start( 𝒢 );
    {V_v, V_u} := bb_bfs_part( 𝒢, v, u )
    t_1 := bb_ct_build_bfs( 𝒢|_{V_v} );
    t_2 := bb_ct_build_bfs( 𝒢|_{V_u} );
    return cluster t := V with 𝒮(t) := {t_1, t_2};
  end if
end
```

Lemma 1 (Complexity of black box clustering) *The complexity N_{bbc} for the BFS based black box clustering in Algorithm 4 to build the cluster tree $T_{\mathscr{I}}$ for $A \in \mathbb{R}^{N\times N}$ is bounded by*

$$N_{\mathrm{bbc}} = \mathscr{O}(c\,N\ \mathrm{depth}(T_{\mathscr{I}})),$$

where c is the maximal degree of a node in $\mathscr{G}(A)$.

Proof The assertion is proved by induction over the depth of $T_{\mathscr{I}}$. For $\mathrm{depth}(T_{\mathscr{I}}) = 1$ we have one cluster and nothing is to be done. Hence, the assertion is trivially fulfilled.

Now let $\mathrm{depth}(T_{\mathscr{I}}) > 1$ and $\mathscr{G} = (V, E)$ be the graph to be partitioned. The computation of start nodes by Algorithm 3 requires a bounded number of BFS iterations each with costs $\mathscr{O}(\#E) = \mathscr{O}(c\,\#V)$. The partitioning of $\mathscr{G}$ into V_v and V_u by Algorithm 2 is performed with a single BFS iteration, again resulting in a complexity of $\mathscr{O}(c\,\#V)$.

By induction, the recursion for $\mathscr{G}|_{V_1}$ has a complexity of $\mathscr{O}\left(c\,\#V_1\ \mathrm{depth}(T_{V_1})\right)$ and the recursion for $\mathscr{G}|_{V_2}$ $\mathscr{O}\left(c\,\#V_2\ \mathrm{depth}(T_{V_2})\right)$ respectively. Therefore, the total costs are at most $\mathscr{O}(c\,\#V\ \mathrm{depth}(T_V))$.

The construction of the block cluster tree $T_{\mathscr{I}\times\mathscr{I}}$ is straightforward (see Algorithm 1) if we can provide an admissibility condition. Since the distance between nodes computed by path lengths corresponds to the geometrical distance between the position of the associated indices (at least in the regular grid chosen for the motivation example), we use the same admissibility condition (6), only now with graph-based distances and diameters as defined in Definition 7:

Definition 8 (Black box admissibility) For two clusters $s, t \in T_{\mathscr{I}}$ and a (sparse) matrix $A \in \mathbb{R}^{\mathscr{I}\times\mathscr{I}}$, we define the *standard BB-admissibility*

$$s \times t \text{ is BB-admissible} \quad :\Leftrightarrow \quad \min\{\mathrm{diam}(s), \mathrm{diam}(t)\} \le \eta\, \mathrm{dist}_{\mathscr{G}(A)}(s,t). \tag{11}$$

The *weak BB-admissibility* is defined by

$$s \times t \text{ is weakly BB-admissible} \quad :\Leftrightarrow \quad \mathrm{dist}_{\mathscr{G}(A)}(s,t) > 1$$

The weak BB-admissibility is the weakest reasonable admissibility to be used. It may happen that blocks are considered admissible but will fill in during an $\mathscr{H}$-LU decomposition.

Unfortunately, the computation of the distance between two clusters s, t as well as the diameter of a cluster s are expensive tasks, requiring quadratic complexity in the cardinality of the clusters. Fortunately, to assure admissibility, we only need to assert that $\eta\,\mathrm{dist}_{\mathscr{G}(A)}(s,t)$ is larger than $\min\{\mathrm{diam}(s), \mathrm{diam}(t)\}$. For this, it is sufficient to define an upper bound $\widetilde{\mathrm{diam}}$ of the diameters of s and t

$$\min\{\mathrm{diam}(s), \mathrm{diam}(t)\} \le \widetilde{\mathrm{diam}}.$$

and check whether

$$\widetilde{\mathrm{diam}} \le \eta\,\mathrm{dist}_{\mathscr{G}(A)}(s,t)$$

holds.

For the estimation of the diameters of s and t and therefore for the definition of $\widetilde{\mathrm{diam}}$ the following lemma is used.

Lemma 2 *Let $s \in T_{\mathscr{I}}$ be a cluster, and let $i_0 \in s$. Let $i_1 \in s$ be so that*

$$\mathrm{dist}_{\mathscr{G}(A)}(i_0, i_1) = \max_{i\in s} \mathrm{dist}_{\mathscr{G}(A)}(i_0, i).$$

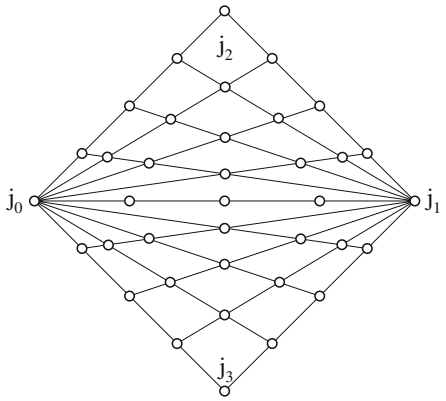

Fig. 7 Worst and best case for $\widetilde{\text{diam}}$

Furthermore, let $\text{diam}_{i_0,i_1}(s)$ *be defined as*

$$\text{diam}_{i_0,i_1}(s) := 2\,\text{dist}_{\mathcal{G}(A)}(i_0, i_1).$$

Then we can estimate

$$\text{diam}(s) \le \text{diam}_{i_0,i_1}(s) \le 2\,\text{diam}(s). \tag{12}$$

Proof

$$\begin{aligned}\text{diam}(s) &= \max_{i,j\in s} \text{dist}_{\mathcal{G}(A)}(i,j) \le 2 \max_{i\in s} \text{dist}_{\mathcal{G}(A)}(i_0, i)\\ &\le \text{diam}_{i_0,i_1}(s).\end{aligned}$$

Figure 7 shows an example for optimal and worst case estimation of the diameter of a graph by diam_{i_0,i_1}. The diameter of the presented graph is 8, whereas $\text{diam}_{j_0,j_1} = 8$ and $\text{diam}_{j_2,j_3} = 16$.

The upper bound $\text{diam}_{i_0,i_1}(s)$ from (12) has already been computed for all interior nodes $s \in T_{\mathcal{I}} \backslash \mathcal{L}(T_{\mathcal{I}})$ during Algorithm 4 by Algorithm 3, where i_0 and i_1 correspond to the last two considered nodes. For the (small) leaves $s \in \mathcal{L}(T_{\mathcal{I}})$, we can compute the exact diameter by BFS in $\mathcal{O}\left(n_{\min}^2\right)$.

Since by (12) a good estimate for the real diameter of the sub-graphs is provided, we assume

$$\begin{aligned}\widetilde{\text{diam}} &\le \min\left\{\text{diam}_{i_0,i_1}(s), \text{diam}_{i_0,i_1}(t)\right\}\\ &\le 2\,\min\{\text{diam}(s), \text{diam}(t)\}\end{aligned}$$

for the following discussion.

Finally, for the standard BB-admissibility to be satisfied we have to check if $\widetilde{\text{diam}} \le \eta\,\text{dist}(s,t)$ holds. For this, we extend the set s by all nodes with a distance less than $\frac{1}{\eta}\widetilde{\text{diam}}$ from s to obtain the surrounding $U(s)$:

$$U(s) := \left\{ i \in \mathcal{I} \mid \eta\ \text{dist}(i,s) < \widetilde{\text{diam}} \right\}.$$

Now, if $U(s) \cap t = \emptyset$, then

$$\forall i \in s,\ j \in t\ :\ dist_{\mathcal{G}(A)}(i,j) \ge \frac{1}{\eta}\widetilde{\text{diam}}$$

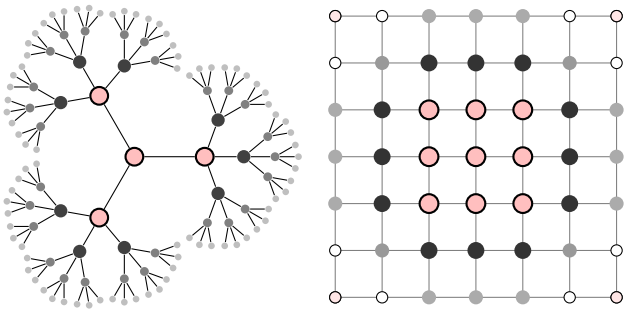

Fig. 8 Exponential (*left*) and bounded (*right*) size of surroundings

holds. This proves the standard BB admissibility for $s \times t$

$$\min\{\text{diam}(s), \text{diam}(t)\} \le \widetilde{\text{diam}} \le \eta\,\text{dist}(s,t).$$

On the other hand, for non-empty intersections $U(s) \cap t, s \times t$ is probably not admissible. This depends on the sharpness of the choice of $\widetilde{\text{diam}}$ in comparison to $\min\{\text{diam}(s), \text{diam}(t)\}$ (see also Remark 4). In such a case, $s \times t$ is regarded as inadmissible.

The complexity of the standard black box admissibility check depends on the size of the surroundings $U(s)$ and $U(t)$. In general, this can be $\mathcal{O}\left(c^{\widetilde{\text{diam}}}\#s\right) = \mathcal{O}\left(c^{\#s}\#s\right)$, where c is the maximal degree of nodes in s (see Fig. 8 (left)). In our model problem on the other hand, the size of a surrounding grows linearly with the diameter of a graph and exponential in the dimension.

Remark 3 (Complexity of the Standard BB Adm. Check [Model Problem]) Let s be a set of n^d nodes corresponding to a d-dimensional cube (Fig. 8 (right, center nodes)). Furthermore, let w.l.o.g. $\widetilde{\text{diam}} \le 2\,\text{diam}(s)$ hold. Then, for the surrounding $U(s)$ by $\widetilde{\text{diam}}$ layers around s the number of nodes can be estimated as

$$\begin{aligned}\#U(s) &\le (n + 2\widetilde{\text{diam}})^d - n^d\\ &= n^d\left((1+2d)^d - 1\right) \in \mathcal{O}\left(n^d d^d\right) = \mathcal{O}\left(\#s d^d\right).\end{aligned}$$

Therefore, the standard black box admissibility check for a block cluster $s \times t$ has complexity

$$\mathcal{O}\left(d^d \max\{\#s, \#t\}\right) \tag{13}$$

With this result we can estimate the complexity for the construction of the block cluster tree $T_{\mathcal{I}\times\mathcal{I}}$ from a given cluster tree $T_{\mathcal{I}}$ by Algorithm 1 using the standard BB-admissibility to be approximately $\mathcal{O}(N \log N)$ for the model problem. This is by a factor k less than the storage requirements for an $\mathcal{H}$-matrix based on $T_{\mathcal{I}\times\mathcal{I}}$, and by more than a factor $k^2 \log N$ less than a subsequent $\mathcal{H}$-LU decomposition [2]. Therefore, the complexities for the setup of both the cluster tree and the block cluster tree are dominated by the complexity of the $\mathcal{H}$-matrix arithmetic based on these trees. The same behaviour was observed for other problem classes.

Remark 4 (Modified standard BB admissibility) The quality of the block partitioning defined by the above described admissibility check obviously depends on the estimate of the minimal diameters of s and t. Consider for example $\widetilde{\text{diam}} \geq \delta \min\{\text{diam}(s), \text{diam}(t)\}$ with $\delta \geq 1$. Now the tested admissibility of the described procedure would become

$$\min\{\text{diam}(s), \text{diam}(t)\} \leq \frac{\eta}{\delta} \text{dist}(s,t).$$

Alternatively, the parameter η can be modified to fit the changed admissibility condition due to an overestimation of the diameter, e.g. $\eta' := \eta/\delta$.

Remark 5 The black box clustering in Algorithm 4 is based on an undirected, connected graph. This can be generalised to arbitrary graphs as follows.

1. For disconnected (but undirected) graphs, we can split the graph into maximal connected parts $\mathscr{I} = \dot{\bigcup}_{i=1,\ldots,q} V_i$. We then proceed with the clustering of the connected parts V_i.
2. For a directed graph, we again have to split the graph into maximal connected parts $\mathscr{I} = \dot{\bigcup}_{i=1,\ldots,q} V_i$, for which the clustering algorithm is applied individually. In addition to this, we apply a proper ordering of the connected components to obtain a block lower triangular form of the matrix (cf. Fig. 9). To solve such a system, only the inversion, e.g. $\mathscr{H}$-LU factorisation, of the diagonal blocks is necessary. Furthermore, the off-diagonal matrix blocks can be stored in the original sparse matrix format, thereby reducing the storage requirements.

3.2 Clustering via other graph partitioning algorithms

In the previous section, the goal of the partitioning algorithm was to simulate the geometrical partitioning by using path lengths in the graph, resulting in a similar decomposition compared to geometric bisection. Another approach for a suitable partitioning for $\mathscr{H}$-matrices is the decoupling

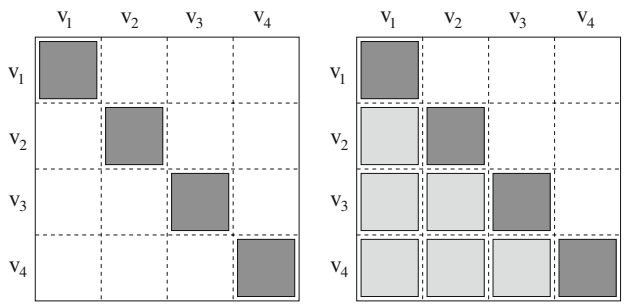

Fig. 9 Splitting into connected parts $v_1, \ldots, v_4$ and resulting matrix sparsity structure, left for an undirected and right for a directed graph

of the constructed sub index sets, e.g. the minimisation of the number of edges between sub-graphs, the so called *edge-cut*. Consider for instance a matrix $A \in \mathbb{R}^{\mathscr{I}\times\mathscr{I}}$ with the disjoint decomposition of $\mathscr{I}$ into $\mathscr{I} = I_1 \cup I_2$. If the matrix block $A_{12} \in \mathbb{R}^{I_1 \times I_2}$ contains only a few entries, this directly translates into a low rank of A_{12}, which remains low during an $\mathscr{H}$-LU factorisation. Note that we have assumed $\mathscr{G}(A)$ to be connected and hence the graphs $\mathscr{G}(A)|_{I_1}$ and $\mathscr{G}(A)|_{I_2}$ are connected by at least one edge.

Minimising the edge-cut between sub-graphs during graph partitioning is the goal of many such algorithms described in the literature, e.g. via spectral or multilevel methods (see also the Discussion in [19]), which also found their way into graph partitioning software, e.g. METIS [19], SCOTCH [25] or CHACO [17].

Other strategies are based on maximal parallel efficiency, e.g. when decomposing a grid in a parallel domain decomposition method, or trying to minimise the fill-in during LU factorisation of sparse matrices.

Not all graph partitioning algorithms are applicable in the context of $\mathscr{H}$-matrices. Strategies based on graph colouring for instance result in a decomposition of the graph with a very large number of edges between individual sub-graphs and therefore in a high rank of the corresponding matrix blocks. Also, for efficiency of the $\mathscr{H}$-matrix algorithms, the computed sub-graphs shall be of similar size.

The above mentioned multilevel algorithms minimising the edge-cut fulfil these requests. Nevertheless, they can only be considered to be heuristics even in the case of the model problem in contrast to the BFS algorithm. However, the numerical results in Sect. 5 will demonstrate the effectiveness of these graph partitioning methods.

The algorithm for constructing a cluster tree using a general graph partitioning algorithm is similar to Algorithm 4 and presented in Algorithm 5. There, "partition()" denotes the given partitioning procedure.

Algorithm 5 General Black Box Clustering

```
procedure bb_ct_build( G = (V, E) )
  if #V ≤ n_min then
    return cluster t := V;
  else
    {G_1, G_2} = partition( G );
    t_1 := bb_ct_build( G_1 );
    t_2 := bb_ct_build( G_2 );
    return cluster t := V with S(t) := {t_1, t_2};
  end if
end
```

3.3 Nested dissection

Many direct methods for sparse linear systems perform an LU factorisation of the original matrix after some reordering

of the indices in order to reduce fill-ins. A popular reordering method is the so-called *nested dissection* method which exploits the concept of separation. The idea of nested dissection has been introduced more than 30 years ago [8] and since then attracted considerable attention (see, e.g. [4,18] and the references therein). The main idea is to separate the vertices in a (matrix) graph $\mathcal{G}(A)$ into three parts: two disconnected sub-graphs G_1 and G_2 and a third one, Γ referred to as an interior boundary or (vertex) separator which contains couplings with both of the other two parts. The nodes in the sub-graphs G_1 and G_2 are numbered first and the nodes in Γ are numbered last. This process is then repeated recursively in G_1 and G_2. An illustration of the resulting sparsity pattern is shown in Fig. 10 for the first two decomposition steps.

A favourable property of such an ordering is that a subsequent LU factorisation maintains a major part of this sparsity structure, i.e. there occurs no fill-in in the large, off-diagonal zero matrix blocks. In fact, in the case of a regular three-dimensional grid, the computational complexity amounts to $\mathcal{O}\left(N^2\right)$ for a matrix $A \in \mathbb{R}^{N\times N}$ [23]. In order to obtain a (nearly) optimal complexity, we propose to approximate the nonzero, off-diagonal blocks in the $\mathcal{H}$-matrix representation and compute them using $\mathcal{H}$-matrix arithmetic (see also [11,12]).

Especially suited for nested dissection are graph partitioning algorithms trying to minimise the edge-cut between sub-graphs of comparable size. In that case, the size of the separator is also small and hence, the size of the zero off-diagonal blocks is large. Applying those algorithms to a given matrix graph is therefore the first step in a nested dissection graph partitioning.

The next step is the identification of the vertex separator, whose removal decouples the remaining sub-graphs. Such a separator is not unique and can be computed in different ways. In [19], an algorithm for computing a *vertex cover*, i.e. a subset of nodes incident to all edges, for the set of edges connecting both sub-graphs is used. For this, both end-vertices of a connecting edge can be removed from the graph and put into the vertex cover until no edge between both sub-graphs remains. Since all vertex covers must at least contain as many nodes as there are edges in the edge-cut, this simple algorithm computes a vertex cover of at most two times the size of a minimal vertex cover. Better results, e.g. smaller vertex separators have been achieved by only removing one vertex per edge from the larger of the two sub-graphs. The final method is presented in Algorithm 6.

Algorithm 6 Computing a vertex separator

```
procedure vtxsep ( G = (V, E), V_1, V_2 )
  Let E_ec := {(v,u) | (v ∈ V_1 ∧ u ∈ V_2) ∨ (u ∈ V_1 ∧ v ∈ V_2)};
  if #V_1 > #V_2 then V_large := V_1
  else V_large := V_2;
  V_vtx := ∅;
  for all (v,u) ∈ E_ec do
    W := V_large ∩ {v,u};
    V_vtx := V_vtx ∪ W;
    E_ec := E_ec \ ((W × V) ∪ (V × W));
  end for;
  return V_vtx;
end;
```

In contrast to the classical nested dissection approach, the matrices corresponding to sub-graph-to-separator couplings or to separator-to-separator couplings are not represented by dense matrices but by $\mathcal{H}$-matrices. Therefore, the node set of the vertex separator has to be further partitioned. Here the problem arises that the graph $\mathcal{G}(A)|_\Gamma$ is in general not connected, even if $\mathcal{G}(A)$ was (cf. Fig. 11). Nevertheless, the partitioning should be based on the distance in the original graph.

To achieve this, we use a modified form of the BFS partitioning algorithm, where the connectivity in the vertex separator is not defined in $\mathcal{G}(A)|_\Gamma$ but in the surrounding graph $\mathcal{G}(A)$. For the determination of the start nodes by Algorithm 3 this means, that although the BFS is performed in $\mathcal{G}(A)$, only nodes in Γ are considered to be valid start nodes. The partitioning of the graph $\mathcal{G}(A)|_\Gamma$ via Algorithm 2 follows a similar pattern: the BFS is done in $\mathcal{G}(A)$ but the sets V_v and V_u are only updated with nodes in the vertex separator.

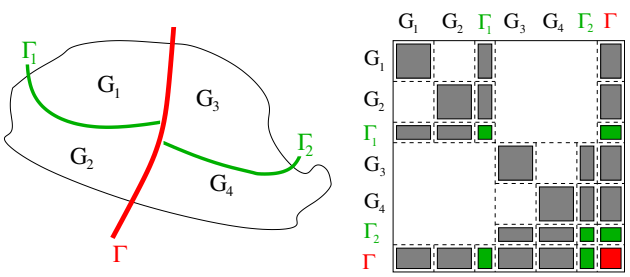

Fig. 10 Nested dissection and resulting matrix sparsity structure

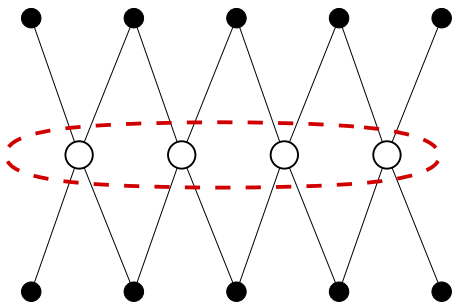

Fig. 11 Not connected separator of a connected graph

Further subdivision of the subsets Γ_1 and Γ_2 of a partitioning of Γ is again performed in a surrounding graph, which can be $\mathscr{G}(A)$ or a sufficiently large restriction of $\mathscr{G}(A)$:

Remark 6 A BFS iteration for $\mathscr{G}(A)|_\Gamma$ in $\mathscr{G}(A)$ can be stopped as soon as all nodes in the vertex separator have been visited. Let $V_{\text{vis}} \subseteq V_A$ be the final set of visited nodes during this BFS. Then V_{vis} represents a restriction of the node set V_A sufficient for computing the distance between nodes in Γ in $\mathscr{G}(A)$. Hence, for further BFS iterations, $\mathscr{G}(A)|_{V_{\text{vis}}}$ can be used instead of $\mathscr{G}(A)$. For each $V \subseteq \Gamma$ we define

$$\mathscr{G}_{\min}(V) := \mathscr{G}(A)|_{V'_{\text{vis}}}$$

to be the *minimal surrounding graph* computed during the BFS with V'_{vis} being the minimal set of visited nodes.

Algorithm 7 combines all these algorithms into a procedure for building a cluster tree for a given vertex separator. There, "bb_bfs_start_vtxsep" and "bb_bfs_part_vtxsep" are Algorithms 3 and 2, respectively, with the modifications described above.

Algorithm 7 Graph Partitioning of a Vertex Separator

```
procedure bb_ct_build_vtxsep ( V, G_sur )
  if #V ≤ n_min then
    return cluster t := V;
  else
    {v, u} := bb_bfs_start_vtxsep( V, G_sur );
    {V_v, V_u} := bb_bfs_part_vtxsep( V, G_sur, u, v )
    t_1 := bb_ct_build_vtxsep( V_v, G_min(V_v) );
    t_2 := bb_ct_build_vtxsep( V_u, G_min(V_u) );
    return cluster t := V with S(t) := {t_1, t_2};
  end if;
end;
```

Lemma 3 (Complexity of vertex separator partitioning) *Let $\mathscr{G} = (V_\mathscr{G}, E_\mathscr{G})$ be a connected graph and V_{vtx} be the computed vertex separator. Then the complexity for building a cluster tree $T(V_{\text{vtx}})$ for V_{vtx} by Algorithm 7 is*

$$\mathscr{O}\left(\#E_\mathscr{G} \operatorname{depth}(T(V_{\text{vtx}}))\right). \tag{14}$$

Proof Computing start nodes and partitioning V in $\mathscr{G}_{\text{sur}} = (V, E)$ into V_v and V_u in Algorithm 7 involves a fixed number of BFS iterations in $\mathscr{G}_{\text{sur}}$. Therefore, the costs are $\mathscr{O}(\#E)$. Furthermore, due to construction by BFS, the sum of the edges in $\mathscr{G}_{\min}(V_v)$ and $\mathscr{G}_{\min}(V_u)$ is at most $\#E$. Therefore, on each level of the cluster tree, the costs for all BFS iterations together is $\mathscr{O}(\#E)$. Since the number of levels is defined by $\operatorname{depth}(T(V_{\text{vtx}}))$, this proves (14).

Finally, we can consider the construction of the cluster tree for an index set $\mathscr{I}$. The basic procedure stays the same as in Algorithm 5, i.e. the partitioning of the index set. In addition to this, the computation of the vertex separator and the construction of its cluster tree is included. Algorithm 8 shows the final method, where "partition()" again denotes a given graph partitioning algorithm suitable for $\mathscr{H}$-matrices.

Algorithm 8 General Black Box Clustering with Nested Dissection

```
procedure bb_ct_build_nd( G = (V, E) )
  if #V ≤ n_min then
    return cluster t := V;
  else
    {G_1, G_2} = partition( G );
    V_vtx := vtxsep(G, V(G_1), V(G_2));
    t_1 := bb_ct_build_nd( G_1 );
    t_2 := bb_ct_build_nd( G_2 );
    t_3 := bb_ct_build_vtxsep( V_vtx, G_min(V_vtx) );
    return cluster t := V with S(t) := {t_1, t_2, t_3};
  end if
end
```

Lemma 4 *(Complexity of black box clustering with nested dissection) Let "partition()" be a graph partitioning algorithm to be used in Algorithm 8 with costs of at most $\mathscr{O}(\#V + \#E)$ for a graph $\mathscr{G} = (V, E)$. Then the complexity for computing a cluster tree $T_\mathscr{I}$ for a given sparse matrix $A \in \mathbb{R}^{N\times N}$ by Algorithm 8 is*

$$N_{\text{bbc,nd}} = \mathscr{O}\left(c\, N \operatorname{depth}(T_\mathscr{I})\right), \tag{15}$$

where c is the maximal degree of a node in $\mathscr{G}(A)$.

Proof The proof is similar to the proof for Lemma 1. The only difference is in the induction step, where we observe that, by assumption, the partitioning costs are $\mathscr{O}(\#V + \#E) = \mathscr{O}(c\,\#V)$. Again by induction and together with Lemma 3 the total cost for computing T_V stays within $\mathscr{O}(c\,\#V \operatorname{depth}(T_V))$.

For the construction of the block cluster tree the admissibility condition from Definition 8 has to be modified to make use of the decoupling of the indices by the vertex separator:

Definition 9 *(Nested dissection black box admissibility)* Let $\text{adm} : T \to \{\texttt{false}, \texttt{true}\}$ be a black box admissibility condition as defined in Definition 8. Then, for two clusters $s, t \in T_\mathscr{I}$ and a (sparse) matrix $A \in \mathbb{R}^{\mathscr{I}\times\mathscr{I}}$, we define the *nested dissection BB-admissibility* (or *ND-BB-admissibility*) of adm as

$$\begin{aligned} s \times t \text{is ND-BB-admissible} :\Leftrightarrow\ & \text{adm}(s\times t) = \texttt{true} \ \vee \\ & (s \neq t \wedge \text{neither } s \text{ nor } t \text{ is a vertex separator}). \end{aligned} \tag{16}$$

Remark 7 For testing the admissibility of $s \times t$ with the given admissibility condition adm in Definition 9, the distance and diameter of a cluster are defined in terms of $\mathscr{G}_{\min}(s)$ and $\mathscr{G}_{\min}(t)$.

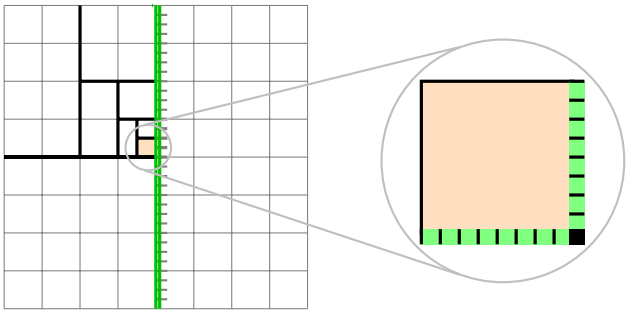

Fig. 12 Different size of clusters in vertex separator (*green*) and graph (*orange*) after eight partitioning steps in the model problem

Remark 8 Since the vertex separator between two graphs is usually much smaller than the graphs themselves, the depths of the corresponding cluster trees differ in magnitude. By following the algebraic clustering described above, the difference of the diameters of the vertex separator and the subgraphs grows rapidly by each partitioning step, leading to an unbalanced cluster tree in terms of path lengths and cluster size per tree-level. The result of this imbalance are large constants in the complexity of the $\mathscr{H}$-arithmetic, e.g. the sparsity constant C_{sp} [10].

As an example, consider the situation in the model problem depicted in Fig. 12. For simplicity, we assume that the partitioning of the index set follows the geometrical algorithm with an alternating separation axis in x and y direction. After $2\,\ell$ partitioning steps, the innermost graph has $2^{\ell-1}$ neighbouring clusters of the first vertex separator. The corresponding block clusters are inadmissible.

To avoid this situation, during the partitioning of the vertex separator *idle steps* are introduced, i.e. steps where no decomposition of the node set is performed and trivial clusters with only one son are constructed. For a graph $\mathscr{G}$ decomposed into $\mathscr{G}_1 = (V_1, E_1)$, $\mathscr{G}_2(V_2, E_2)$ and a vertex separator V_{vtx}, the depth of $T(V_{\mathrm{vtx}})$ should be equal to $p := \max\{\mathrm{depth}(T(V_1)), \mathrm{depth}(T(V_2))\}$. To achieve a size of $n_{\min}$ after p partitioning steps for V_{vtx}, the size of the node set has to be reduced in average by the factor

$$\rho(V_{\mathrm{vtx}}) := \left(\frac{n_{\min}}{\#V_{\mathrm{vtx}}}\right)^{1/p}$$

per step. An idle step is then performed, if the size of V in Algorithm 7 on level ℓ is less than $\#V_{\mathrm{vtx}}\rho(V_{\mathrm{vtx}})^{\ell}$.

4 Parallel $\mathscr{H}$-LU Factorisation

The parallelisation of the LU factorisation based on a hierarchical bipartition of the index set, e.g. as was done in Sect. 2.2 or by Algorithm 4, is mainly restricted to the parallelisation of the involved matrix multiplications [20]. Furthermore, since recursions with respect to off-diagonal blocks are necessary, the parallel degree is limited, leading to unsatisfactory results in practical applications.

Using nested dissection on the other hand, greatly increases the parallelism of the LU factorisation since parts of the matrix are decoupled and can be treated independently. This is not restricted to $\mathscr{H}$-matrices but to general matrices and was therefore exploited long ago for many applications, and is in fact one of the reasons for the popularity of this technique.

Nested dissection itself is a special version of the *domain decomposition* method, where the number of domains is restricted to two. The applicability of domain decomposition to the $\mathscr{H}$-matrix technique with a focus on parallel execution was demonstrated in [16]. One can also find a discussion of a multilevel domain decomposition approach with a recursive definition of the involved matrices in this article.

Due to these properties, we will concentrate on the parallelisation of the LU factorisation for $\mathscr{H}$-matrices based on nested dissection. Since this technique is not limited to the black box algorithms described in Sect. 3 but also applicable in the geometrical case discussed in [12], we will slightly change the notation for involved clusters following the domain decomposition naming scheme: the clusters constructed by bipartition are called *domain clusters* whereas the third cluster that decouples the two domain clusters, before called the vertex separator, is now called *interface cluster*.

For the parallelisation we assume a computer system with a distributed memory, e.g. a network of workstations. Since the memory is not shared, some form of communication is necessary between individual processors.

For simplicity, the number of processors p shall be a power of 2. Furthermore, we assume that p is much smaller than N, the number of unknowns. Also, let $\mathscr{P} = \{0, \ldots, p-1\}$ be the set of processors.

The mapping of clusters in the cluster tree $T_{\mathscr{I}}$ to subsets of processors is done recursively, where the recursion begins with the root $\mathscr{I}$ which is mapped to all processors in the set $\mathscr{P}$. For the direct successors of $\mathscr{I}$, $\mathscr{P}$ is partitioned into $\mathscr{P}_1 = \{0, \ldots, p/2-1\}$ and $\mathscr{P}_2 = \{p/2, \ldots, p-1\}$. The succeeding domain clusters t_1 and t_2 of $\mathscr{I}$ are then mapped to $\mathscr{P}_1$ and $\mathscr{P}_2$, respectively. The interface cluster on the same level as t_1 and t_2 is, on the other hand, mapped to the union of $\mathscr{P}_1$ and $\mathscr{P}_2$, and therefore to $\mathscr{P}$. This mapping is pursued until at level $\log p$ each domain cluster is associated with a distinct processor. All successors are then mapped to the CPU of their direct ancestor. An example of this mapping can be seen in Fig. 13 (left).

For the nodes of the block cluster tree $T_{\mathscr{I}\times\mathscr{I}}$ and hence the matrix blocks in the resulting $\mathscr{H}$-matrix, the described mapping of the cluster tree is again used. Diagonal blocks $t \times t \in T_{\mathscr{I}\times\mathscr{I}}$, are mapped to the set of processors associated with t. For an off-diagonal block $s \times t \in T_{\mathscr{I}\times\mathscr{I}}$, let $\mathscr{P}_t$

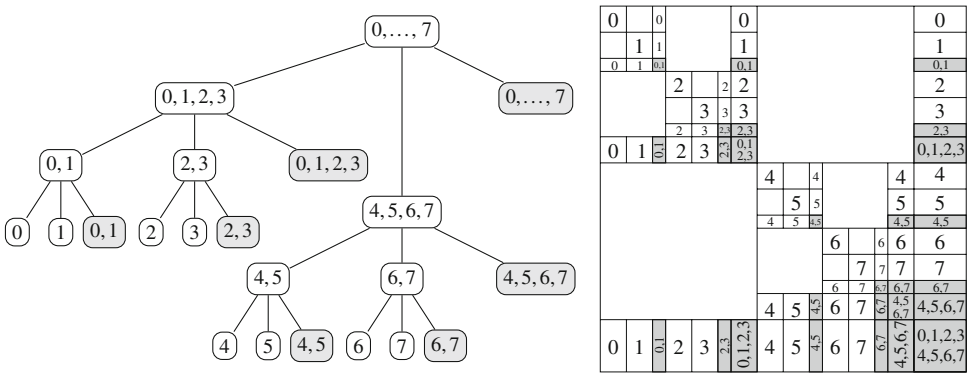

Fig. 13 Mapping of the nodes of the cluster tree $T_{\mathscr{I}}$ and corresponding mapping of the block clusters and corresponding matrix blocks to 8 processors

and $\mathscr{P}_s$ denote the corresponding processor-sets. Then $s \times t$ is assigned to the set $\mathscr{P}_t \cap \mathscr{P}_s$. In Fig. 13 (right) this was done for the block cluster tree resulting from the previous example.

The described processor layout allows two different handlings for matrix blocks corresponding to combinations of interface clusters, e.g. the matrix A_{33}. Since these matrices are handled sequentially, they can be either stored on all processors in the local processor map or just by a distinct one, the so-called *master processor* $m(A) = m(\mathscr{P}(A)) := \min \mathscr{P}(A)$. The former method slightly reduces the communication costs, whereas the latter reduces storage costs. The overall complexity is not changed.

So far it was assumed that the corresponding nodes for the mapping of the processors in the block cluster tree are available. Unfortunately, this is not always the case. Especially the off-diagonal blocks corresponding to a domain-interface coupling might be admissible and hence, not refined such that the recursive algorithm is not applicable. Therefore, for a given admissibility condition adm, e.g. (6) or (16), the modified condition adm_p is used for the construction of the block cluster tree in case of more than one processor:

$$\mathrm{adm}_p(s \times t) = \texttt{true} :\Leftrightarrow \mathrm{adm}(s \times t) = \texttt{true} \wedge \max\{\mathrm{depth}(s), \mathrm{depth}(t)\} \geq \log p \quad (17)$$

This ensures that up to the case that a block cluster is mapped to a distinct processor, nodes in the block cluster tree are available. It should be noted that this modification slightly decreases the sparsity of the resulting block cluster tree and hence, increases the complexity of the algorithms. The modified admissibility condition is also crucial for the parallel scalability of the algorithm, since unrefined blocks are handled by a single processor.

The recursive nature of the processor mapping, which indirectly also occurs at the data-distribution of the block cluster tree, repeats in the design of the matrix algorithms, of which the LU factorisation shall be discussed in detail.

For this, the matrix A is assumed to have a 3×3 block structure:

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

with submatrices corresponding to domain clusters in the blocks A_{11} and A_{22}, domain-interface coupling contained in A_{13}, A_{23}, A_{31} and A_{32} and the relation between interface indices in A_{33}. For all matrix blocks M let $\mathscr{P}(M)$ denote the set of mapped processors.

The actual LU decomposition method is split into 3 algorithms, whereby in Algorithm 9 the main procedure is shown. There, depending on the local processor number i, first the local diagonal block is decomposed. Afterwards, the off-diagonal matrix blocks are computed by calling the corresponding "solve"-functions. Each processor then calculates the local update T_i to the matrix A_{33}, which are summed up to form the global update for the interface matrix. The final result T of the summation resides on the master processor $m(A)$. This processor is also responsible for decomposing the interface matrix A_{33}.

Remark 9 The summation and distribution of the matrices T_i and A_{33} can be accomplished in $\log \#\mathscr{P}(A)$ steps by using a procedure based on a binary tree, where each master processor of a processor set $\mathscr{P}$ exchanges data with the corresponding master processors of the subsets $\mathscr{P}_1$ and $\mathscr{P}_2$ of $\mathscr{P}$ [9].

Solving the off-diagonal matrices A_{13} and A_{23} follows a similar pattern. First the local matrices are solved, followed by the update and computation of the interface matrix. Note that the arguments A and U in Algorithm 10 correspond to the matrices A_{11}, A_{31} and A_{22}, A_{32} in Algorithm 9.

The algorithm for the procedure "solve_upper" is implemented in an analogous way.

Finally, the matrix multiplication, which is used to compute the update for the interface matrix has to be modified

Algorithm 9 Parallel LU factorisation

procedure LU(i, A)
  if $i \in \mathscr{P}(A_{11})$ **then**
    LU(i, A_{11});
    solve_lower(i, A_{11}, A_{31}); solve_upper(i, A_{11}, A_{13});
    $T_i := A_{31}A_{13}$;
  else
    LU(i, A_{22});
    solve_lower(i, A_{22}, A_{32}); solve_upper(i, A_{22}, A_{23});
    $T_i := A_{32}A_{23}$;
  end if
  $T := \sum_{i \in \mathscr{P}(A)} T_i$;
  if $i = m(A)$ **then**
    $A_{33} := A_{33} - T$;
    LU(i, A_{33});
  end if
  distribute A_{33} to all processors in $\mathscr{P}$;
end;

Algorithm 10 Computation of off-diagonal matrix A during LU factorisation

procedure solve_lower(i, A, U)
  if $i \in \mathscr{P}(A_{11})$ **then**
    solve_lower(i, A_{11}, U_{11});
    $T_i := A_{11}U_{13}$;
  else
    solve_lower(i, A_{12}, U_{22});
    $T_i := A_{12}U_{23}$;
  end if
  $T = \sum_{i \in \mathscr{P}(A)} T_i$;
  if $i = m(A)$ **then**
    $A_{13} := A_{13} - T$;
    solve_lower(i, A_{13}, U_{33});
  end if
  distribute A_{13} to all processors in $\mathscr{P}$;
end;

$$(A_{11}\ A_{12}\ A_{13}) \cdot \begin{pmatrix} U_{11} & & U_{13} \\ & U_{22} & U_{23} \\ & & U_{33} \end{pmatrix}$$

to exploit the parallel distribution of the involved matrices. The resulting algorithm is similar to the computation of the off-diagonal blocks in Algorithm 10.

Algorithm 11 Parallel multiplication of off-diagonal matrices

procedure multiply(i, A, B, C)
  if $i \in \mathscr{P}(A_{11})$ **then**
    multiply(i, A_{11}, B_{11}, T_1);
  else
    multiply(i, A_{12}, B_{12}, T_2);
  end if
  $T := \sum_{i \in \mathscr{P}(A)} T_i$;
  if $i = m(A)$ **then** $C := C + T + A_{13}B_{31}$;
  distribute C to all processors in $\mathscr{P}$;
end;

$$(A_{11}\ A_{12}\ A_{13}) \cdot \begin{pmatrix} B_{11} \\ B_{21} \\ B_{31} \end{pmatrix}$$

Two things are essential for the analysis of the parallel algorithms: first, the difference in the work associated to the two diagonal sub matrices A_{11} and A_{22}, and second, the size of the vertex separator. Equal workload for the matrices A_{11} and A_{22} is ensured by cardinality balanced geometrical clustering [12] or by the black box clustering algorithms in Sect. 3.

As described in Sect. 3.3, partitioning algorithms minimising the edge cut are particularly suited for nested dissection because of a small interface between the constructed domain clusters. Unfortunately, the exact size of the interface can not be further estimated in the general case, which prohibits a detailed complexity analysis.

In the geometrical case on the other hand, a small (or minimal) interface corresponds to a reduction of the spatial dimension, e.g. the interface being a curve in $\mathbb{R}^2$ or surface in $\mathbb{R}^3$ respectively. This directly translates into the number of indices in the interface: for a domain cluster s in $\mathbb{R}^d$ with $\#s = n$ and sons $S(s) = \{s_1, s_2\}$, the minimal interface t between s_1 and s_2 is of order

$$\#t \in \mathscr{O}\left(n^{\frac{d-1}{d}}\right). \tag{18}$$

Due to the similarities between geometrical clustering and algebraic BFS based clustering for the model problem (see Sect. 3.1), the same holds in the latter case.

For the complexity analysis, we will assume an equal workload for domain clusters and interface clusters of minimal order (18).

We start by examining the matrix multiplication. Algorithm 11 consists of a recursive call, a parallel summation of matrices, a sequential multiplication of the interface-interface coupling and a broadcast of the result to all processors in the local processor set. Let n be the dimension of the matrices A and B, e.g. $A \in \mathscr{H}(T, k)$ with $\#V(T) = n$, and $p = \#\mathscr{P}(A)$. By using Remark 9 both communication parts can be accomplished in $\log p$ steps. Since the involved matrices result in a storage size of $\mathscr{O}(n \log n)$, the work for the summation and the broadcast is

$$N_{\text{sum}}(n, p) = N_{\text{broadcast}}(n, p) = \mathscr{O}(\log(p) n \log n).$$

Together with the recursive call, the final complexity can then be defined as

$$N_{\mathscr{H}\cdot\mathscr{H}}(n, p) = N_{\mathscr{H}\cdot\mathscr{H}}\left(\frac{n}{2}, \frac{p}{2}\right) + N_{\text{sum}}\left(n^{\frac{d-1}{d}}, p\right) + N_{\mathscr{H}\cdot\mathscr{H}}\left(n^{\frac{d-1}{d}}\right) + N_{\text{broadcast}}\left(n^{\frac{d-1}{d}}, p\right).$$

For clarity and simplicity, we omitted the dependency on the rank and sparsity of the matrices and concentrated on the parallel part. Also, since the multiplication of the submatrices is done in parallel, only a single term $N_{\mathscr{H}\cdot\mathscr{H}}(n/2, p/2)$ appears in the equation.

Sequential matrix multiplication has a complexity of $N_{\mathscr{H}\cdot\mathscr{H}}(n, 1) = \mathscr{O}\left(n \log^2 n\right)$ (see [12, Theorem 19]). Using this result and by solving the recursion formula, we get the

final complexity of the parallel matrix multiplication:

$$N_{\mathscr{H}\cdot\mathscr{H}}(n,p) = \mathscr{O}\left(\frac{n}{p}\log^2\frac{n}{p} + n^{\frac{d-1}{d}}\log n^{\frac{d-1}{d}}\left(\log n^{\frac{d-1}{d}} + \log p\right)\right). \tag{19}$$

A similar analysis can be done for Algorithm 10 which also contains calls to the parallel matrix multiplication. The resulting equation for the complexity of Algorithm 10 is as follows:

$$\begin{aligned} N_{\text{Solve}}(n,p) = {} & N_{\text{Solve}}\left(\frac{n}{2},\frac{p}{2}\right) + N_{\mathscr{H}\cdot\mathscr{H}}\left(\frac{n}{2},\frac{p}{2}\right) \\ & + N_{\text{sum}}\left(N^{\frac{d-1}{d}},p\right) + N_{\text{Solve}}\left(N^{\frac{d-1}{d}},1\right) \\ & + N_{\text{broadcast}}\left(N^{\frac{d-1}{d}},p\right). \end{aligned}$$

Again, parallel computations are only included once. The final equation for the computational complexity for solving the off-diagonal matrices in parallel is identical to the complexity of the matrix multiplication:

$$N_{\text{Solve}}(n,p) = \mathscr{O}\left(\frac{n}{p}\log^2\frac{n}{p} + n^{\frac{d-1}{d}}\log n^{\frac{d-1}{d}}\left(\log n^{\frac{d-1}{d}} + \log p\right)\right). \tag{20}$$

Here, the result $N_{\text{Solve}}(n,1) = \mathscr{O}\left(n\log^2 n\right)$ for the sequential procedure discussed in the proof of Corollary 20 in [12] was used.

At last, we come to the final algorithm of the parallel LU decomposition. Examining Algorithm 9, one obtains the following complexity equation

$$\begin{aligned} N_{\mathscr{H}-LU}(N,p) = {} & N_{\mathscr{H}-LU}\left(\frac{N}{2},\frac{p}{2}\right) + 2N_{\text{Solve}}\left(\frac{N}{2},\frac{p}{2}\right) \\ & + N_{\mathscr{H}\cdot\mathscr{H}}\left(\frac{N}{2},\frac{p}{2}\right) + N_{\text{sum}}\left(N^{\frac{d-1}{d}},p\right) \\ & + N_{\mathscr{H}-LU}\left(N^{\frac{d-1}{d}}\right) \\ & + N_{\text{broadcast}}\left(N^{\frac{d-1}{d}},p\right). \end{aligned}$$

Again, [12, Corollary 20] provides us with the result for the sequential $\mathscr{H}$-LU factorisation, which is $N_{\mathscr{H}-LU}(N,1) = \mathscr{O}\left(n\log^2 n\right)$. After solving the recursion and using (19) and (20), we can finally write down the complexity for computing the LU factorisation in parallel.

Corollary 1 (Parallel $\mathscr{H}$-LU factorisation) *For the complexity $N_{\mathscr{H}-LU}(N,p)$ of the parallel $\mathscr{H}$-LU factorisation there holds:*

$$\begin{aligned} & N_{\mathscr{H}-LU}(N,p) \\ & = \mathscr{O}\left(\frac{N}{p}\log^2\frac{N}{p} + N^{\frac{d-1}{d}}\log N^{\frac{d-1}{d}}\left(\log p + \log N^{\frac{d-1}{d}}\right)\right) \\ & = \mathscr{O}\left(\frac{N\log^2 N}{p} + \frac{N\log^2 N}{N^{1/d}}\right). \end{aligned} \tag{21}$$

According to (21), the complexity of the LU factorisation is equal to the complexity of the parallel matrix multiplication. The sequential part of the algorithm, and hence, the part which restricts the parallel scalability, is mainly dependent on the size of the first interface, which should be chosen optimal. Compared with this, the dependence on the number of processors is only logarithmic. We conclude that the scalability is optimal as long as $p \lesssim N^{1/d}$.

Remark 10 The same kind of analysis can also be done for the direct domain decomposition Ansatz for the parallel LU factorisation as described in [16], resulting in a complexity of

$$\begin{aligned} & N_{\mathscr{H}-LU}(N,p) \\ & = \mathscr{O}\left(\frac{N}{p}\log^2\frac{N}{p} + p^{1/d}N^{\frac{d-1}{d}}\log^2 N^{\frac{d-1}{d}}\right) \\ & = \mathscr{O}\left(\frac{N\log^2 N}{p} + p^{1/d}\frac{N\log^2 N}{N^{1/d}}\right). \end{aligned} \tag{22}$$

Comparing (21) and (22), the (partial) parallelisation of the interface as done by the nested dissection reduces the complexity by a factor of $p^{1/d}$ for the second term.

5 Numerical results

In this section, the performance of the $\mathscr{H}$-matrix arithmetics based on black box clustering shall be examined on different problems and compared to the geometrical approach. For the black box clustering, the breadth-first search based algorithm introduced in Sect. 3.1 and the graph partitioning algorithms implemented in METIS [19] and SCOTCH [25] are used. Besides the sequential method, also the parallel algorithm of Sect. 4 is tested. The tests were performed on a parallel computer system with 32 individual nodes connected by an Infiniband network. Each node was equipped with an AMD Opteron 254 processor with 2.8 GHz CPU speed.

In all tests, the $\mathscr{H}$-LU factorisation $LU \approx A$ or, if A is symmetric positive definite, the $\mathscr{H}$-Cholesky factorisation $LL^T \approx A$ is computed. The accuracy δ of the $\mathscr{H}$-arithmetics was chosen to obtain an approximation such that $\|I - (LU)^{-1}A\|_2 \leq \rho \leq 10^{-2}$. In a linear iteration method, the preconditioner $(LU)^{-1}$ therefore guarantees convergence with a convergence rate of at least ρ and hence, serves as a very good preconditioner.

Table 1 Comparison of geometrical and black box bisection clustering for the $\mathscr{H}$-Cholesky factorisation of the Poisson matrix

N	Geometric			Black box (BFS)			Black box (METIS)			Black box (SCOTCH)		
	Time (s)	Mem (MB)	δ	Time (s)	Mem (MB)	δ	Time (s)	Mem (MB)	δ	Time (s)	Mem (MB)	δ
253^2	3.8	76	$2_{10}-4$	4.9	75	$9_{10}-5$	6.6	86	$1_{10}-4$	7.1	88	$1_{10}-4$
358^2	10.0	169	$1_{10}-4$	12.9	173	$4_{10}-5$	15.7	187	$6_{10}-5$	17.7	199	$6_{10}-5$
511^2	24.1	374	$7_{10}-5$	34.1	403	$2_{10}-5$	41.7	441	$3_{10}-5$	41.3	440	$3_{10}-5$
729^2	61.1	840	$4_{10}-5$	85.8	912	$1_{10}-5$	116.1	1020	$1_{10}-5$	108.9	1,000	$1_{10}-5$
$1{,}023^2$	144.9	1,780	$2_{10}-5$	227.1	1,960	$6_{10}-6$	250.8	2,110	$8_{10}-6$	262.7	2,140	$8_{10}-6$
40^3	79.1	285	$1_{10}-3$	102.3	295	$8_{10}-4$	106.5	292	$1_{10}-3$	93.9	280	$1_{10}-3$
51^3	194.5	634	$1_{10}-3$	334.9	788	$5_{10}-4$	326.1	763	$7_{10}-4$	266.7	706	$7_{10}-4$
64^3	520.3	1,400	$1_{10}-3$	1,280.0	2,010	$3_{10}-4$	896.4	1,760	$4_{10}-4$	812.6	1,720	$4_{10}-4$
81^3	1,440.0	3,560	$5_{10}-4$	3,332.9	4,760	$2_{10}-4$	2,444.8	4,330	$2_{10}-4$	2,489.9	4,420	$2_{10}-4$
102^3	3,875.5	8,070	$4_{10}-4$	9,773.4	11,490	$1_{10}-4$	6,575.7	9,940	$2_{10}-4$	8,509.6	11,020	$1_{10}-4$

Table 2 Comparison of geometrical and black box nested dissection clustering for the $\mathscr{H}$-Cholesky factorisation of (23)

N	Geometric			Black box (BFS)		
	Time(s)	Mem (MB)	δ	Time (s)	Mem (MB)	δ
253^2	0.9	51	$1_{10}-3$	1.3	47	$3_{10}-5$
358^2	1.9	86	$4_{10}-4$	2.9	94	$2_{10}-5$
511^2	4.5	212	$2_{10}-4$	6.5	198	$9_{10}-6$
729^2	9.6	371	$1_{10}-4$	15.0	402	$5_{10}-6$
$1{,}023^2$	20.2	878	$6_{10}-5$	31.6	819	$2_{10}-6$
40^3	12.6	99	$1_{10}-2$	32.7	135	$3_{10}-4$
51^3	46.9	300	$3_{10}-3$	97.6	323	$2_{10}-4$
64^3	117.4	592	$2_{10}-3$	289.1	719	$1_{10}-4$
81^3	269.8	1,410	$1_{10}-3$	804.3	1,570	$8_{10}-5$
102^3	752.3	3,020	$1_{10}-3$	1,907.3	3,370	$6_{10}-5$

For the first numerical test, the matrix A is defined by Poisson's equation

$$-\Delta u = f \quad \text{in} \quad \Omega =]0,1[^d, \quad d \in \{2,3\}, \tag{23}$$

discretised with the finite element method using piecewise linear Ansatz functions. Since the corresponding stiffness matrix is symmetric, the $\mathscr{H}$-Cholesky factorisation $LL^T \approx A$ is computed.

Table 1 shows the results for the computation of the $\mathscr{H}$-Cholesky decomposition LL^T of the matrix A using standard geometrical bisection and black box bisection with the standard admissibility (see Definition 8).

In all cases, the geometrical approach shows the best performance in terms of execution time and (except for the smallest problem) memory consumption. The numbers for the black box clustering vary significantly with the used partitioning technique, whereby in the 2D case the BFS strategy and for the 3D problem the METIS algorithm give the best results. The required accuracy δ of the $\mathscr{H}$-arithmetics to obtain the desired accuracy of the $\mathscr{H}$-Cholesky factors is very similar for all graph partitioning methods, with a slight disadvantage for the BFS algorithm. Nevertheless, the increase in the execution time for black box clustering based $\mathscr{H}$-matrices compared to the geometrical approach never exceeds a factor of 2 for the 2D problems and 2.5 in the 3D case, demonstrating the usability of the black box technique.

Instead of standard bisection, also nested dissection (see Sect. 3.3) can be applied to the given problem. The results for these computations are presented in Table 2. While the comparison of the results for the geometrical and the black box approach show a similar picture as in the standard bisection clustering, the raw numbers clearly demonstrate the superiority of the nested dissection technique over the bisection approach. The execution time of the Cholesky factorisation is drastically reduced and the memory consumption is more than halved compared to the results in Table 1.

The second problem uses the convection-diffusion equation:

$$-\kappa\Delta u + b \cdot \nabla u = f \quad \text{in} \quad \Omega =]0,1[^d, \quad d \in \{2,3\}. \tag{24}$$

Here, the (circular) convection direction b is defined by

$$b(x) := \begin{pmatrix} 0.5 - x_2 \\ x_1 - 0.5 \end{pmatrix}.$$

In $\mathbb{R}^3$, the third component of b is zero. The value of κ is set to 10^{-3} resulting in a dominant convection.

The results for the convection–diffusion equation presented in Table 3 are similar to the Poisson problem, albeit the difference between the geometrical and the algebraical approach is smaller.[1] Again, the METIS algorithm provides the best graph partitioning strategy with a resulting runtime

[1] Timings in the last row ($N = 102^3$) were obtained—due to memory limitations—on a 20% slower CPU.

Table 3 Comparison of geometrical and black box bisection clustering for the $\mathscr{H}$-LU factorisation of (24)

N	Geometric			Black box (BFS)			Black box (METIS)			Black box (SCOTCH)		
	Time (s)	Mem (MB)	δ	Time (s)	Mem (MB)	δ	Time (s)	Mem (MB)	δ	Time (s)	Mem (MB)	δ
253^2	6.7	137	$1_{10}-4$	8.3	137	$9_{10}-5$	7.9	134	$1_{10}-4$	8.3	135	$1_{10}-4$
358^2	16.0	294	$9_{10}-5$	21.2	308	$5_{10}-5$	20.8	307	$6_{10}-5$	21.4	308	$6_{10}-5$
511^2	40.8	669	$5_{10}-5$	58.5	716	$2_{10}-5$	49.0	677	$3_{10}-5$	52.6	702	$3_{10}-5$
729^2	106.0	1,450	$3_{10}-5$	144.6	1,560	$1_{10}-5$	119.1	1,490	$2_{10}-5$	127.9	1,520	$2_{10}-5$
$1{,}023^2$	236.3	3,120	$2_{10}-5$	382.1	3,450	$7_{10}-6$	294.5	3,240	$1_{10}-5$	335.8	3,430	$1_{10}-5$
40^3	137.6	527	$1_{10}-3$	188.2	541	$6_{10}-4$	179.0	529	$7_{10}-4$	169.6	522	$7_{10}-4$
51^3	378.5	1,210	$6_{10}-4$	592.2	1,360	$5_{10}-4$	449.6	1,270	$5_{10}-4$	503.1	1,300	$5_{10}-4$
64^3	1,079.4	2,840	$5_{10}-4$	1,861.8	3,300	$3_{10}-4$	1,630.2	3,240	$3_{10}-4$	1,645.6	3,220	$3_{10}-4$
81^3	2,924.4	6,930	$3_{10}-4$	5,002.7	8,140	$2_{10}-4$	3,952.5	7,630	$2_{10}-4$	4,349.6	7,870	$2_{10}-4$
102^3	9,738.7	15,970	$2_{10}-4$	21,327.5	20,340	$1_{10}-4$	17,177.0	19,270	$1_{10}-4$	16,560.9	19,340	$1_{10}-4$

Table 4 Comparison of geometrical and black box nested dissection clustering for the $\mathscr{H}$-LU factorisation of (24)

N	Geometric			Black Box (BFS)		
	Time (s)	Mem (MB)	δ	Time (s)	Mem (MB)	δ
253^2	2.1	97	$1_{10}-4$	2.9	92	$4_{10}-5$
358^2	4.4	160	$9_{10}-5$	5.8	183	$2_{10}-5$
511^2	9.9	406	$6_{10}-5$	13.0	367	$1_{10}-5$
729^2	20.9	687	$4_{10}-5$	28.5	765	$6_{10}-6$
$1{,}023^2$	43.8	1,620	$4_{10}-5$	61.6	1,490	$3_{10}-6$
40^3	47.3	292	$1_{10}-3$	54.8	263	$4_{10}-4$
51^3	154.7	690	$5_{10}-4$	175.0	627	$2_{10}-4$
64^3	409.2	1,360	$4_{10}-4$	522.4	1,380	$1_{10}-4$
81^3	843.2	3,100	$3_{10}-4$	1,192.8	3,020	$1_{10}-4$
102^3	2,328.8	6,890	$2_{10}-4$	3,371.4	6,640	$8_{10}-5$

of the $\mathscr{H}$-arithmetics of only 20–30% slower than the corresponding geometrical variant. The accuracy δ is almost identical to the Poisson problem, demonstrating the robustness of $\mathscr{H}$-matrices.

Next, nested dissection clustering is used to compute an $\mathscr{H}$-LU preconditioner. The corresponding results are shown in Table 4. As for the Poisson problem, nested dissection significantly improves the computational efficiency and reduces the memory consumption. The difference between the geometrical and the black box approach is comparable to the Poisson case.

The previous two problems are based on a regularly refined grid for which, by construction (see Sect. 3.1), similar results for the geometrical and the black box case are expected. The third example will use a different grid with strong local refinement (see Fig. 14). The equation to be solved is again the Poisson problem.

Table 5 shows the results obtained from the computation of the Cholesky factorisation for the locally refined grid. There, especially METIS shows a very competitive performance compared to geometrical clustering. The BFS clustering technique on the other hand results in a relatively large impact in terms of runtime. The required accuracy δ is very similar for all clustering strategies.

Although the $\mathscr{H}$-matrix technique is not applicable to arbitrary matrices, the black box approach for the computation of a cluster tree brings $\mathscr{H}$-matrices near direct solvers like PARDISO [27–29] or UMFPACK [7]. Therefore, the performance of $\mathscr{H}$-matrices for the solution of linear systems compared to the performance of these direct solvers shall be examined for the convection-diffusion problem (24). For $\mathscr{H}$-matrices and for PARDISO the nested dissection approach was used in this test. The results are presented in Table 6. Missing numbers could not be computed on the given computer system.

The results show a large dependence on the spatial dimension of the problem. In 2D the UMFPACK and the PARDISO solver are very fast and consume the least amount of

Table 5 Performance of Cholesky factorisation with different clustering techniques on a locally refined grid

N	Geom.			BFS		
	Time (s)	Mem (MB)	δ	Time (s)	Mem (MB)	δ
209, 427	62.3	616	$3_{10}-5$	143.5	849	$1_{10}-5$
794, 197	335.4	2, 770	$9_{10}-6$	832.2	3, 850	$4_{10}-6$

N	METIS			SCOTCH		
	Time (s)	Mem (MB)	δ	Time (s)	Mem (MB)	δ
209,427	71.7	666	$2_{10}-5$	80.2	705	$2_{10}-5$
794,197	431.1	3,140	$6_{10}-6$	461.4	3,230	$6_{10}-6$

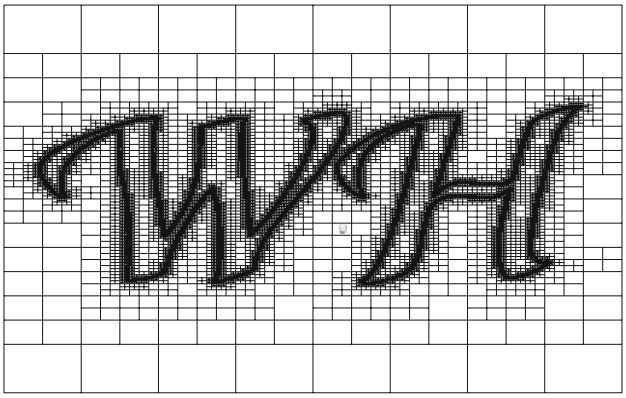

Fig. 14 Grid with local refinement

memory. $\mathscr{H}$-Matrices on the other hand need more memory but provide the better computational complexity, which results in a point of break-even at 2 million unknowns for the convection–diffusion problem. For the 3D case, PARDISO and UMFPACK have a quadratic to cubic complexity in N. Therefore, $\mathscr{H}$-matrices are faster for more than 100,000 unknowns, demonstrating the advantage of the almost linear complexity of $\mathscr{H}$-matrices (Fig. 15).

Finally, the parallel performance of the $\mathscr{H}$-arithmetic based on nested dissection shall be examined for the $\mathscr{H}$-Cholesky factorisation of the stiffness matrix for the Poisson problem (23). Figure 16 shows the parallel speedup, i.e. the ratio of the sequential and the parallel runtime, for different problem sizes. In addition to the nested dissection clustering algorithm, also a direct domain decomposition method (DD) was applied to the given problem to show the different scaling properties (cf. Remark 10). The results are presented only for the geometrical approach since the black box clustering technique gave almost identical numbers.

Table 6 Comparison of $\mathscr{H}$-LU factorisation, PARDISO and UMFPACK for (24)

N	Black box		PARDISO		UMFPACK	
	Time (s)	Mem (MB)	Time (s)	Mem (MB)	Time (s)	Mem (MB)
511^2	13.0	367	7.1	193	7.3	259
729^2	28.5	765	18.6	422	17.5	543
$1{,}023^2$	61.6	1,490	52.6	891	51.6	1,185
$1{,}447^2$	135.0	3,040	150.2	1,876	131.8	2452
$2{,}047^2$	283.6	6,150	475.1	4,048	379.8	5,292
40^3	54.8	263	41.6	275	85.5	585
51^3	175.0	627	197.9	719	710.6	2,026
64^3	522.4	1,380	941.6	1,953	4,528.1	5,364
81^3	1,192.8	3,020	4,317.5	5,207	21,321.1	16,071
102^3	3,371.4	6,640	22,191.3	13,996		

In the 2D case, the influence of the sequentially treated interface is minimal, leading to an almost perfect parallel speedup. Only for a large number of processors, the speedup deviates from the perfect behaviour. Here, the role of the sequential part increases, bounding the speedup. Fortunately, this effect decreases with larger problem sizes. For the direct domain decomposition algorithm, the interface and therefore the sequential part of the algorithm is larger, which only leads to a mediocre scaling behaviour.

Due to the decreased volume-to-surface ratio in $\mathbb{R}^3$, the interface between domain clusters is larger. Therefore, the parallel speedup of the $\mathscr{H}$-Cholesky factorisation is smaller than in the 2D case (cf. (21)). Also, the increase of the

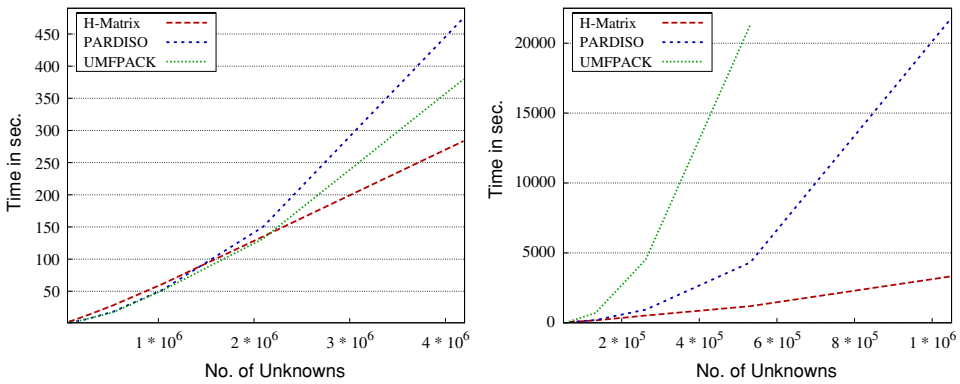

Fig. 15 Complexity of $\mathscr{H}$-LU factorisation, PARDISO and UMFPACK for (24) in $\mathbb{R}^2$ (*left*) and $\mathbb{R}^3$ (*right*)

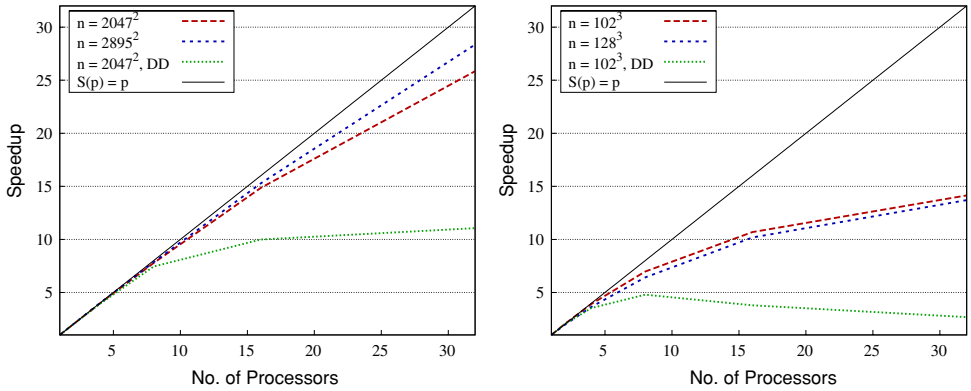

Fig. 16 Parallel speedup of the $\mathscr{H}$-Cholesky factorisation of (23) in $\mathbb{R}^2$ (*left*) and $\mathbb{R}^3$ (*right*)

speedup with a larger N is not visible. In fact, a smaller number of unknowns produces a slightly better parallel scaling behaviour. This can be explained by imbalances in the cluster sizes and therefore the work load per processor which are more pronounced in 3D due to the small number of indices per spatial direction (128 vs. 2895 in $\mathbb{R}^2$).

References

1. Bebendorf, M.: Why finite element discretizations can be factored by triangular hierarchical matrices. SIAM Num. Anal. **45**, 1472–1494 (2007)
2. Börm, S., Grasedyck, L., Hackbusch, W.: Hierarchical matrices (2003). Lecture Notes No. 21, Max-Planck-Institute for Mathematics in the Sciences, Leipzig, Germany, available online at www.mis.mpg.de/preprints/ln/, revised version June (2006)
3. Börm, S., Grasedyck, L., Hackbusch, W.: Introduction to hierarchical matrices with applications. Eng. Anal. Boundary Elements **27**, 405–422 (2003)
4. Brainman, I., Toledo, S.: Nested-dissection orderings for sparse LU with partial pivoting. SIAM J. Mat. Anal. Appl. **23**, 998–1012 (2002)
5. Brandt, A., McCormick, S., Ruge, J.: Algebraic multigrid (AMG) for sparse matrix equations. In: Sparsity and its Applications, pp. 257–284. Cambridge University Press, Cambridge (1984)
6. Bröker, O., Grote, M., Mayer, C., Reusken, A.: Robust parallel smoothing for multigrid via sparse approximate inverses. SIAM J. Sci. Comput. **23**, 1396–1417 (2001)
7. Davis, T.A.: Algorithm 832: UMFPACK, an unsymmetric-pattern multifrontal method. ACM Trans. Math. Soft. **30**(2), 196–199 (2004)
8. George, A.: Nested dissection of a regular finite element mesh. SIAM J. Numer. Anal. **10**, 345–363 (1973)
9. Grama, A., Gupta, A., Karypis, G., Kumar, V.: Introduction to Parallel Computing. Addison Wesley, Reading (2003)
10. Grasedyck, L., Hackbusch, W.: Construction and arithmetics of $\mathscr{H}$-matrices. Computing **70**, 295–334 (2003)
11. Grasedyck, L., Kriemann, R., Borne, S.L.: Parallel Black Box Domain Decomposition Based $\mathscr{H}$-LU Preconditioning. Tech. rep., Max-Planck-Institute MIS (2005)
12. Grasedyck, L., Kriemann, R., Le Borne, S.: Domain decomposition based $\mathscr{H}$-LU preconditioning. Numer. Math. (2006) (submitted)
13. Grasedyck, L., Le Borne, S.: $\mathscr{H}$-matrix preconditioners in convection-dominated problems. SIAM J. Mat. Anal. **27**, 1172–1183 (2006)
14. Haase, G., Kuhn, M., Reitzinger, S.: Parallel AMG on distributed memory computers. SIAM J. Sci.Comp. **24**(2), 410–427 (2002)
15. Hackbusch, W.: A sparse matrix arithmetic based on $\mathscr{H}$-matrices. Part I: Introduction to $\mathscr{H}$-matrices. Computing **62**, 89–108 (1999)
16. Hackbusch, W.: Direct domain decomposition using the hierarchical matrix technique. In: Herrera, I., Keyes, D., Widlund, O., Yates, R. (eds.) Domain Decomposition Methods in Science and Engineering, pp. 39–50. UNAM (2003)
17. Hendrickson, B., Leland, R.: The Chaco User's Guide: Version 2.0. Tech. Rep. SAND94–2692, Sandia National Laboratories (1994)
18. Hendrickson, B., Rothberg, E.: Improving the run time and quality of nested dissection ordering. SIAM J. Sci. Comp. **20**, 468–489 (1998)
19. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J. Sci. Comput. **20**(1), 359–392 (1999)
20. Kriemann, R.: Parallele Algorithmen für $\mathscr{H}$-Matrizen. Ph.D. thesis, Universität Kiel (2005)
21. Le Borne, S., Oliveira, S., Yang, F.: $\mathscr{H}$-matrix preconditioners for symmetric saddle-point systems from meshfree discretizations. Numer. Linear Algebra Appl. (2006) (in press)
22. Lintner, M.: The eigenvalue problem for the 2d Laplacian in $\mathscr{H}$-matrix arithmetic and application to the heat and wave equation. Computing **72**, 293–323 (2004)
23. Lipton, R.J., Rose, D.J., Tarjan, R.E.: Generalized nested dissection. SIAM J. Numer. Anal. **16**, 346–358 (1979)
24. Oliveira, S., Yang, F.: An algebraic approach for $\mathscr{H}$-matrix preconditioners. Computing **80**, 169–188 (2007)
25. Pellegrin, F.: SCOTCH 5.0 User's guide. Tech. rep., LaBRI, Université Bordeaux I (2007)
26. Ruge, R.W., Stüben, K.: Efficient solution of finite difference and finite element equations by algebraic multigrid (AMG). In: Paddon, H.H.D.J. (ed.) Multigrid Methods for Integral and Differential Equations, pp. 169–212. Clarenden Press, Oxford (1985)
27. Schenk, O., Gärtner, K.: Solving unsymmetric sparse systems of linear equations with PARDISO. J. Future Generat. Comput. Syst. **20**, 475–487 (2004)
28. Schenk, O., Gärtner, K.: On fast factorization pivoting methods for symmetric indefinite systems. Elec. Trans. Numer. Anal. **23**, 158–179 (2006)
29. Schenk, O., Gärtner, K., Fichtner, W.: Efficient sparse LU factorization with left-right looking strategy on shared memory multiprocessors. BIT **40**, 158–176 (2000)