ON THE BEST APPROXIMATION OF THE HIERARCHICAL MATRIX PRODUCT

JÜRGEN DÖLZ, HELMUT HARBRECHT, AND MICHAEL D. MULTERER†

ABSTRACT. The multiplication of matrices is an important arithmetic operation in computational mathematics. In the context of hierarchical matrices, this operation can be realized by the multiplication of structured block-wise low-rank matrices, resulting in an almost linear cost. However, the computational efficiency of the algorithm is based on a recursive scheme which makes the error analysis quite involved. In this article, we propose a new algorithmic framework for the multiplication of hierarchical matrices. It improves currently known implementations by reducing the multiplication of hierarchical matrices towards finding a suitable low-rank approximation of sums of matrix-products. We propose several compression schemes to address this task. As a consequence, we are able to compute the best-approximation of hierarchical matrix products. A cost analysis shows that, under reasonable assumptions on the low-rank approximation method, the cost of the framework is almost linear with respect to the size of the matrix. Numerical experiments show that the new approach produces indeed the best-approximation of the product of hierarchical matrices for a given tolerance. They also show that the new multiplication can accomplish this task in less computation time than the established multiplication algorithm without error control.

1. INTRODUCTION

Hierarchical matrices, $\mathcal{H}$-matrices for short, historically originate from the discretization of boundary integral operators. They allow for a data sparse approximation in terms of a block-wise low-rank matrix. As first shown in [21], the major advantage of the $\mathcal{H}$-matrix representation over other data sparse formats for non-local operators is that basic operations, like addition, multiplication and inversion, can be performed with nearly linear cost. This fact enormously stimulated the research on $\mathcal{H}$-matrices, see e.g. [5, 6, 13, 17, 22, 24] and the references therein, and related hierarchical matrix formats like HSS, see [9, 30, 32], HODLR, see [1, 2], and $\mathcal{H}^2$-matrices, cf. [7, 23].

The applications for $\mathcal{H}$-matrices are manifold: They have been used for solving large scale algebraic matrix Riccati equations, cf. [18], for solving Lyapunov equations, cf. [3], for preconditioning, cf. [19, 27] and for the second moment analysis of partial differential equations with random data, cf. [10, 11], just to name a few.

In this context, the matrix-matrix multiplication of $\mathcal{H}$-matrices is an essential operation. Based on the hierarchical block structure, the matrix-matrix multiplication is performed in a recursive fashion. To that end, in each recursive call, the two factors are considered as block-matrices which are multiplied by a block-matrix product. The resulting matrix-block is then again compressed to the $\mathcal{H}$-matrix format. To limit the computational cost, the block-wise ranks for the compression are

† Michael D. Multerer was born as Michael D. Peters

usually priorily bounded by a user-defined threshold. For this thresholding procedure, no a-priori error estimates exist. This fact and the recursive structure of the matrix-matrix multiplication render the error analysis difficult, in particular, since there is no guarantee that intermediate results provide the necessary low-rank structure.

To reduce the number of these time-consuming and error-introducing truncation steps, different modifications have been proposed in the literature: In [8], instead of applying each low-rank update immediately to an $\mathcal{H}$-matrix, multiple updates are accumulated in an auxiliary low-rank matrix. This auxiliary matrix is propagated as the algorithm traverses the hierarchical structure underlying the $\mathcal{H}$-matrix. This greatly improves computation times, although the computational cost is not improved. Still, also in this approach, multiple truncation steps are performed. Thus, it does not lead to the best approximation of the low-rank block under consideration.

As an alternative, in [10], it has been proposed to directly compute the low-rank approximation of the output matrix block by using the truncated singular value decomposition, realized by means of a Krylov subspace method. This requires only the implementation of matrix-vector multiplications and is hence easy to realize. Especially, it yields the best approximation of the low-rank blocks to be computed. But contrary to expectations, it does not increase efficiency since the eigensolver converges very slowly in case of a clustering of the eigenvalues. Therefore, computing times have not been satisfactory.

In the present article, we pick up the idea from [10] and provide an algorithm that facilitates the direct computation of any matrix block in the product of two $\mathcal{H}$-matrices. This algorithm is based on a sophisticated bookkeeping technique in combination with a compression based on basic matrix-vector products. This new algorithm will naturally lead to the best approximation of the $\mathcal{H}$-matrix product within the $\mathcal{H}$-matrix format. In particular, we cover the cases of an optimal fixed rank truncation and of an optimal adaptively chosen rank based on a prescribed accuracy.

Our numerical experiments show that both, the fixed rank and the adaptive versions of the used low-rank techniques, are significantly more efficient than the traditional arithmetic with fixed rank. In particular, the numerical experiments also validate that the desired error tolerance can indeed be reached when using the adaptive algorithms.

For the actual compression of a given matrix block, any compression technique based on matrix-vector products is feasible. Exemplarily, we shall consider here the adaptive cross approximation, see [4,16], the Golub-Kahan-Lanczos bidiagonalization procedure, see [14], and the randomized range approximation, cf. [25]. We will employ these methods to either compute approximations with fixed ranks or with adaptively chosen ranks. We remark that, in the fixed rank case, a similar algorithm for randomized range approximation, has successively been applied to directly compute the approximation of the product of two HSS or HODLR matrices in [28, 29].

The rest of this article is structured as follows. In Section 2, we briefly recall the construction and structure of $\mathcal{H}$-matrices together with their matrix-vector multiplication. Section 3 is dedicated to the new framework for the matrix-matrix

multiplication. The three example algorithms for the efficient low-rank approximation are then the topic of Section 4. Section 5 is concerned with the analysis of the computational cost of the new multiplications, which shows that the new matrix-matrix multiplication has asymptotically the same computational cost as the standard matrix-matrix multiplication, i.e., almost linear in the number of degrees of freedom. Nonetheless, as the numerical results in Section 6 show, the constants in the estimates are significantly lower for the new approach, resulting in a remarkable speed-up. Finally, concluding remarks are stated in Section 7.

2. Preliminaries

The pivotal idea of hierarchial matrices is to introduce a tree structure on the cartesian product $\mathcal{I} \times \mathcal{I}$, where $\mathcal{I}$ is a suitable index set. The tree structure is then used to identify sub-blocks of the matrix which are suitable for low-rank representation. We follow the the monograph [22, Chapter 5.3 and A.2] and first recall a suitable definition of a tree.

Definition 2.1. *Let V be a non-empty finite set, call it* vertex set*, and let* child *be a mapping from V into the power set $\mathcal{P}(V)$, i.e.,* $\operatorname{child}\colon V \to \mathcal{P}(V)$*. For any $v \in V$, an element v' in* $\operatorname{child}(v)$ *is called* child*, whereas we call v the* parent *of v'. We call the structure $T(V, \operatorname{child})$ a* tree*, if the following properties hold.*

(1) There is exactly one element $r \in V$ which is not a child of a vertex, i.e.,

$$\bigcup_{v \in V} \operatorname{child}(v) = V \setminus \{r\}.$$

We call this vertex the root *of the tree.*

(2) All $v \in V$ are successors *of r, i.e., there is a $k \in \mathbb{N}_0$, such that $v \in \operatorname{child}^k(r)$. We define $\operatorname{child}^k(v)$ recursively as*

$$\operatorname{child}^0(v) = \{v\} \quad \text{and} \quad \operatorname{child}^k(v) = \bigcup_{v' \in \operatorname{child}^{k-1}(v)} \operatorname{child}(v').$$

(3) Any $v \in V \setminus \{r\}$ has exactly one parent.

Moreover, we say that the number k is the level *of v. The* depth *of a tree is the maximum of the levels of its vertices. We define the set of* leaves *of T as*

$$\mathcal{L}(T) = \{v \in V\colon \operatorname{child}(v) = \emptyset\}.$$

We remark that for any $v \in T$, there is exactly one path from r to v, see [22, Remark A.6].

Definition 2.2. *Let $\mathcal{I}$ be a finite index set. The* cluster tree $\mathcal{T}_\mathcal{I}$ *is a tree with the following properties.*

(1) $\mathcal{I} \in \mathcal{T}_\mathcal{I}$ is the root of the tree $\mathcal{T}_\mathcal{I}$,

(2) for all non-leaf elements $\tau \in \mathcal{T}_\mathcal{I}$ it holds

$$\dot{\bigcup_{\sigma \in \operatorname{child}(\tau)}} \sigma = \tau,$$

i.e., all non-leaf clusters are the disjoint union of their children,

(3) all $\tau \in \mathcal{T}_\mathcal{I}$ are non-empty.

The vertices of the cluster tree are referred to as clusters*.*

To achieve almost linear cost for the following algorithms, we shall assume that the depth of the cluster tree is bounded by $\mathcal{O}(\log(\#\mathcal{I}))$ and that the cardinality of the leaf clusters is bounded by $n_{\min}$. Various ways to construct a cluster tree fulfilling this requirement along with different kinds of other properties exist, see [22] and the references therein.

Obviously, by applying the second requirement of the definition recursively, it holds $\tau \subset \mathcal{I}$ for all $\tau \in \mathcal{T}_{\mathcal{I}}$. Consequently, the leaves of the cluster tree form a partition of $\mathcal{I}$.

Definition 2.3. *An* admissibility condition *for $\mathcal{I}$ is a mapping*

$$\operatorname{adm}\colon \mathcal{P}(\mathcal{I}) \times \mathcal{P}(\mathcal{I}) \to \{\text{true}, \text{false}\}$$

which is symmetric, i.e., for $\tau \times \sigma \in \mathcal{P}(\mathcal{I}) \times \mathcal{P}(\mathcal{I})$ it holds

$$\operatorname{adm}(\tau, \sigma) = \operatorname{adm}(\sigma, \tau),$$

and monotone, i.e., if $\operatorname{adm}(\tau, \sigma) = \text{true}$*, it holds*

$$\operatorname{adm}(\tau', \sigma') = \text{true}, \quad \textit{for all } \tau' \in \operatorname{child}(\tau), \sigma' \in \operatorname{child}(\sigma).$$

Different kinds of admissibility exist, see [22] for a thorough discussion and examples. Based on the admissibility condition and the cluster tree, the block-cluster tree forms a partition of the index set $\mathcal{I} \times \mathcal{I}$.

Algorithm 1 Construction of the block-cluster tree $\mathcal{B}$, cf. [22, Definition 5.26]

```
function BuildBlockClusterTree(block-cluster b = τ × σ)
    if adm(τ, σ) = true then
        child(b) := ∅
    else
        if τ and σ have sons then
            sons(b) := {σ' × τ' : τ' ∈ child(τ), σ' ∈ child(σ)}
            for b' ∈ child(b) do
                BuildBlockClusterTree(b')
            end for
        else
            child(b) := ∅
        end if
    end if
end function
```

Definition 2.4. *Given a cluster-tree $\mathcal{T}_{\mathcal{I}}$, the tree structure $\mathcal{B}$ constructed by Algorithm 1 invoked with $b = \mathcal{I} \times \mathcal{I}$ is referred to as* block-cluster tree.

For notational purposes, we write $p = \operatorname{depth}(\mathcal{B})$ and refer to $\mathcal{N}$ as the set of inadmissible leaves of $\mathcal{B}$ and call it the *nearfield*. In a similar fashion, we will refer to $\mathcal{F}$ as the set of admissible leaves of $\mathcal{B}$ and call it the *farfield*. We remark that the depth of the block-cluster tree is bounded by the depth of the cluster tree and that our definition of the block-cluster tree coincides with the notion of a *level-conserving block-cluster tree* from the literature.

Definition 2.5. *For a block-cluster $b = \tau \times \sigma$ and $k \leq \min\{\#\tau, \#\sigma\}$, we define the set of* low-rank matrices *as*

$$\mathcal{R}(\tau \times \sigma, k) = \big\{\mathbf{M} \in \mathbb{R}^{\tau\times\sigma} \colon \operatorname{rank}(\mathbf{M}) \leq k\big\},$$

where all elements $\mathbf{M} \in \mathcal{R}(\tau \times \sigma, k)$ are stored in low-rank representation, i.e.,

$$\mathbf{M} = \mathbf{L}_\mathbf{M}\mathbf{R}_\mathbf{M}^\intercal$$

for matrices $\mathbf{L}_\mathbf{M} \in \mathbb{R}^{\tau\times k}$ and $\mathbf{R}_\mathbf{M} \in \mathbb{R}^{\sigma\times k}$.

Obviously, a matrix in $\mathcal{R}(\tau \times \sigma, k)$ requires $k(\#\tau + \#\sigma)$ units of storage instead of $\#\tau \cdot \#\sigma$, which results in a significant storage improvement if $k \ll \min\{\#\tau, \#\sigma\}$. The same consideration holds true for the matrix-vector multiplication.

With the definition of the block-cluster tree at hand, we are in the position to introduce hierarchical matrices.

Definition 2.6. *Given a block-cluster tree $\mathcal{B}$, the set of* hierarchical matrices*, in short $\mathcal{H}$-*matrices*, of maximal block-rank k is given by*

$$\mathcal{H}(\mathcal{B}, k) := \Big\{\mathbf{H} \in \mathbb{R}^{\#\mathcal{I}\times\#\mathcal{I}} : \mathbf{H}|_{\tau\times\sigma} \in \mathcal{R}(\tau \times \sigma, k) \text{ for all } \tau \times \sigma \in \mathcal{F}\Big\}.$$

A tree structure is induced on each element of this set by the tree structure of the block-cluster tree. Note that all nearfield blocks $\mathbf{H}|_{\tau\times\sigma}$, $\tau \times \sigma \in \mathcal{N}$, are allowed to be dense matrices.

The tree structure of the block-cluster tree provides the following useful recursive block matrix structure on $\mathcal{H}$-matrices. Every matrix block $\mathbf{H}|_{\tau\times\sigma}$, corresponding to a non-leaf block-cluster $\tau \times \sigma$, has the structure

(1)
$$\mathbf{H}\big|_{\tau\times\sigma} = \begin{bmatrix} \mathbf{H}\big|_{\operatorname{child}(\tau)_1\times\operatorname{child}(\sigma)_1} & \cdots & \mathbf{H}\big|_{\operatorname{child}(\tau)_1\times\operatorname{child}(\sigma)_{\#\operatorname{child}(\sigma)}} \\ \vdots & & \vdots \\ \mathbf{H}\big|_{\operatorname{child}(\tau)_{\#\operatorname{child}(\tau)}\times\operatorname{child}(\sigma)_1} & \cdots & \mathbf{H}\big|_{\operatorname{child}(\tau)_{\#\operatorname{child}(\tau)}\times\operatorname{child}(\sigma)_{\#\operatorname{child}(\sigma)}} \end{bmatrix}.$$

If the matrix block $\mathbf{H}|_{\tau'\times\sigma'}$, $\tau' \in \operatorname{child}(\tau)$, $\sigma' \in \operatorname{child}(\sigma)$, is a leaf of $\mathcal{B}$, the matrix block is either a low-rank matrix, if $\tau' \times \sigma' \in \mathcal{F}$, or a dense matrix, if $\tau' \times \sigma' \in \mathcal{N}$. If the matrix block is not a leaf of $\mathcal{B}$, it exhibits again a similar block structure as $\mathbf{H}|_{\tau\times\sigma}$. The required ordering of the clusters relies on the order of the indices in the clusters. A possible block structure of an $\mathcal{H}$-matrix is illustrated in Figure 1. Of course, the structure may vary depending on the used cluster tree and admissibility condition.

Having the block structure (1) available, an algorithm for the matrix-vector multiplication, as listed in Algorithm 2, can easily be derived. Note that the matrix-vector multiplication for the leaf block-clusters involves either a dense matrix or a low-rank matrix. In accordance with [22, Lemma 7.17], the matrix-vector multiplication of an $\mathcal{H}$-matrix block $\mathbf{H}|_{\tau\times\sigma}$ with $\mathbf{H} \in \mathcal{H}(\mathcal{B}, k)$ can be accomplished in at most

$$N_{\mathcal{H}\cdot v}(\tau \times \sigma, k) \leq 2C_{\mathrm{sp}} \max\{k, n_{\min}\}\Big(\big(\operatorname{depth}(\#\tau) + 1\big)\#\tau + \big(\operatorname{depth}(\#\sigma) + 1\big)\#\sigma\Big)$$

operations, where the constant $C_{\mathrm{sp}} = C_{\mathrm{sp}}(\mathcal{B})$ is given as

$$C_{\mathrm{sp}}(\mathcal{B}) := \max_{\tau\in\mathcal{T}_\mathcal{I}} \#\big\{\sigma \in \mathcal{T}_\mathcal{I} \colon \tau \times \sigma \in \mathcal{B}\big\}.$$

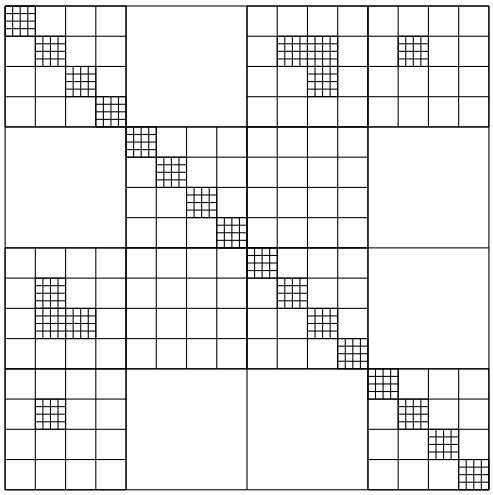


FIGURE 1. Illustration of the recursive block structure of an $\mathcal{H}$-matrix. Only the smallest blocks are allowed to be dense matrices while all other blocks are represented by low-rank matrices.

Algorithm 2 $\mathcal{H}$-matrix-vector multiplication **y+=Hx**, see [22, Equation (7.1)]

```
function HTIMESV(y|_τ, H|_{τ×σ}, x|_σ)
    if τ × σ ∉ L(B) then
        for τ' × σ' ∈ child(τ × σ) do
            HTIMESV(y|_τ', H|_{τ'×σ'}, x|_σ')
        end for
    else
        y|_τ += H|_{τ×σ} x|_σ
    end if
end function
```

Given a cluster $\tau \in \mathcal{T}_{\mathcal{I}}$, the quantity C_{sp} is an upper bound on the number of corresponding block-clusters $\tau \times \sigma \in \mathcal{B}$. Thus, it is also an upper bound for the number of corresponding matrix blocks $\mathbf{H}|_{\tau\times\sigma}$ in the tree structure of an $\mathcal{H}$-matrix corresponding to $\mathcal{B}$.

3. The multiplication of $\mathcal{H}$-matrices

Instead of restating the $\mathcal{H}$-matrix multiplication in its original form from [21], we directly introduce our new framework. The connection between the new framework and the traditional multiplication will be discussed later in this section.

We start by introducing the following *sum-expressions*, which will simplify the presentation and the analysis of the new algorithm.

Definition 3.1. *Let $\tau \times \sigma \in \mathcal{B}$. For a finite index set $\mathcal{J}_{\mathcal{R}}$, the expression*

$$\mathcal{S}_{\mathcal{R}}(\tau,\sigma) = \sum_{j\in\mathcal{J}_{\mathcal{R}}} \mathbf{A}_j \mathbf{B}_j^{\intercal},$$

is called a `sum`-expression of low-rank matrices*, if it is represented and stored as a set of factorized low-rank matrices*

$$\big\{\mathbf{A}_j\mathbf{B}_j^\intercal \in \mathcal{R}(\tau\times\sigma, k_j)\colon j\in\mathcal{J}_\mathcal{R}\big\}.$$

Similarly, for a finite index set $\mathcal{J}_\mathcal{H}$*, the expression*

$$\mathcal{S}_\mathcal{H}(\tau,\sigma) = \sum_{j\in\mathcal{J}_\mathcal{H}} \mathbf{H}_j\mathbf{K}_j$$

is called a `sum`-expression of $\mathcal{H}$-matrices*, if it is represented and stored as a set of pairs of* $\mathcal{H}$*-matrix blocks*

$$\big\{\big(\mathbf{H}_j, \mathbf{K}_j\big) = \big(\mathbf{H}|_{\tau\times\rho_j}, \mathbf{K}|_{\rho_j\times\sigma}\big)\colon \tau\times\rho_j, \rho_j\times\sigma\in\mathcal{B}, j\in\mathcal{J}_\mathcal{H}\big\},$$

with $\mathbf{H},\mathbf{K}\in\mathcal{H}(\mathcal{B},k)$ *and* $\mathbf{H}|_{\tau\times\rho_j}$*,* $\mathbf{H}|_{\tau\times\rho_j}$*,* $j\in\mathcal{J}_\mathcal{H}$*, being either dense matrices or providing the block-matrix structure* (1)*.*

The expression

$$\mathcal{S}(\tau,\sigma) = \mathcal{S}_\mathcal{R}(\tau,\sigma) + \mathcal{S}_\mathcal{H}(\tau,\sigma)$$

is called a `sum`-expression *and a combination of the two previously introduced expressions. In particular, we require that* $\mathcal{S}_\mathcal{R}$ *is stored as a* `sum`*-expression of low-rank matrices and* $\mathcal{S}_\mathcal{H}$ *is stored as a* `sum`*-expression of* $\mathcal{H}$*-matrices.*

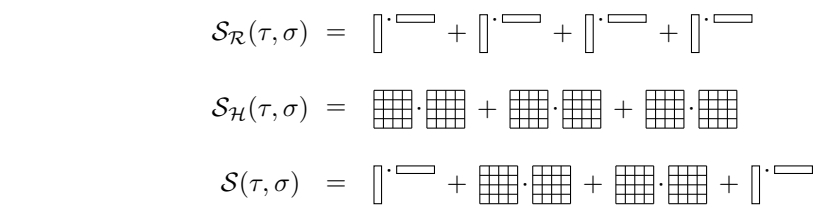


FIGURE 2. Examples for the introduced `sum`-expressions. $\mathcal{S}_\mathcal{R}(\tau,\sigma)$ is a sum of low-rank matrices, $\mathcal{S}_\mathcal{H}(\tau,\sigma)$ a sum of $\mathcal{H}$-matrix products, and $\mathcal{S}(\tau,\sigma)$ a mixture of both.

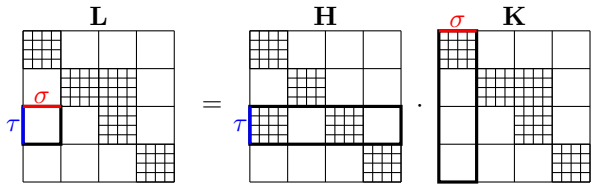
Examples of `sum`-expressions are illustrated in Figure 2. A `sum`-expressions may be considered as a kind of a queuing system to store the sum of low-rank matrices and/or $\mathcal{H}$-matrix products for subsequent operations.

We remark that the sum of two `sum`-expressions is again a `sum`-expression and shall now make use of this fact to devise an algorithm for the multiplication of $\mathcal{H}$-matrices. For simplicity, we assume that all involved $\mathcal{H}$-matrices are built upon the same block-cluster tree $\mathcal{B}$.

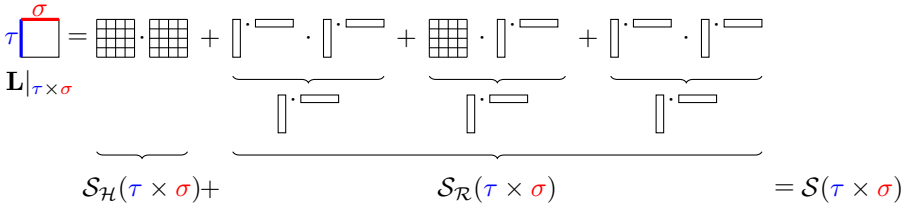
3.1. **Relation between $\mathcal{H}$-matrix products and `sum`-expressions.** We start with two $\mathcal{H}$-matrices $\mathbf{H},\mathbf{K}\in\mathcal{H}(\mathcal{B},k)$ and want to represent their product $\mathbf{L}:=\mathbf{HK}$ in $\mathcal{H}(\mathcal{B},k)$. To that end, we rewrite the $\mathcal{H}$-matrix product as a `sum`-expression

$$\mathbf{L} = \mathbf{HK} =: \mathcal{S}_\mathcal{H}(\mathcal{I},\mathcal{I}) =: \mathcal{S}(\mathcal{I},\mathcal{I}).$$

The task is now to find a suitable low-rank approximation to $\mathbf{L}|_{\tau\times\sigma}$ in $\mathcal{R}(\tau\times\sigma,k)$ for all admissible leafs $\tau\times\sigma$ of $\mathcal{B}$. If $\tau\times\sigma$ is a child of the root, i.e., $\tau\times\sigma\in\operatorname{child}(\mathcal{I}\times\mathcal{I})$,



(a) Matrix blocks of $\mathbf{H}$ and $\mathbf{K}$ which have to be taken into account for $\mathbf{L}|_{\tau\times\sigma}$.



(b) Computation of the `sum`-expression for $\mathbf{L}|_{\tau\times\sigma}$.

FIGURE 3. Illustration why a block $\mathbf{L}|_{\tau\times\sigma}$ on the coarsest level of the target $\mathcal{H}$-matrix can be represented as a `sum`-expression $\mathcal{S}(\tau,\sigma)$.

we have that

$$\begin{aligned}\mathbf{L}|_{\tau\times\sigma} &= \mathcal{S}_{\mathcal{H}}(\mathcal{I},\mathcal{I})|_{\tau\times\sigma}\\ &= \sum_{\rho\in\mathrm{child}(\mathcal{I})} \mathbf{H}|_{\tau\times\rho}\mathbf{K}|_{\rho\times\sigma}\\ &= \sum_{\substack{\rho\in\mathrm{child}(\mathcal{I}):\\ \tau\times\rho\in\mathcal{F}\\ \text{or}\\ \rho\times\sigma\in\mathcal{F}}} \mathbf{H}|_{\tau\times\rho}\mathbf{K}|_{\rho\times\sigma} + \sum_{\substack{\rho\in\mathrm{child}(\mathcal{I}):\\ \tau\times\rho\in\mathcal{B}\setminus\mathcal{F}\\ \rho\times\sigma\in\mathcal{B}\setminus\mathcal{F}}} \mathbf{H}|_{\tau\times\rho}\mathbf{K}|_{\rho\times\sigma},\end{aligned}$$

due to the block-matrix structure (1) of $\mathbf{H}$ and $\mathbf{K}$, see also Figure 3 for an illustration.

The pivotal idea is now that $\mathbf{L}|_{\tau\times\sigma}$ can be written as a `sum`-expression itself, for which we treat the two remaining sums as follows:

- The products in the first sum involve at least one low-rank matrix, such that the product in low-rank representation can easily be computed using matrix-vector multiplications. Having these low-rank matrices computed, we can store the first sum as a `sum`-expression of low-rank matrices $\mathcal{S}_{\mathcal{R}}(\tau,\sigma)$.
- Both factors of the products in the second sum correspond to inadmissible block-clusters. Thus, they are either dense matrices or $\mathcal{H}$-matrices. Since a dense matrix is just a special case of an $\mathcal{H}$-matrix, the second sum can be written as a `sum`-expression of $\mathcal{H}$-matrices $\mathcal{S}_{\mathcal{H}}(\tau,\sigma)$.

It follows that $\mathbf{L}|_{\tau\times\sigma}$ can be represented as a `sum`-expressions by setting

$$\mathbf{L}|_{\tau\times\sigma} = \mathcal{S}_{\mathcal{R}}(\tau,\sigma) + \mathcal{S}_{\mathcal{H}}(\tau,\sigma) =: \mathcal{S}(\tau,\sigma),$$

see also Figure 3 for an illustration.

We can thus represent all children of the root of the block-cluster tree by `sum`-expressions. However, we will require to represent all leaves of the block-cluster tree as `sum`-expressions. It thus remains to discuss how to represent matrix blocks $\mathbf{L}|_{\tau\times\sigma}$ when $\tau\times\sigma$ is not a child of the root.

Remark 3.2. *The representation of any $\mathbf{L}|_{\tau\times\sigma}$ in terms of `sum`-expressions is not unique. For example, assuming that $\tau\times\sigma$ is on level j, one may refine the block-matrix structure of $\mathbf{H}$ and $\mathbf{K}$ by modifying the corresponding admissibility condition. When the admissibility condition is set to* false *on all levels smaller or equal to j, one can construct a finer partitioning for $\mathbf{H}$ and $\mathbf{K}$ to which one can apply the same strategy as above to obtain a `sum`-expression for $\mathbf{L}|_{\tau\times\sigma}$. In particular, the conversions to the finer partitioning can be achieved without introducing any additional errors. However, we will show in Section 5 that we require the following more sophisticated strategy to obtain an $\mathcal{H}$-matrix multiplication in almost linear complexity.*

3.2. **Restrictions of `sum`-expressions.** The main difference between the `sum`-expressions for $\mathbf{L}$ and its restriction $\mathbf{L}|_{\tau\times\sigma}$ from the previous subsection is the presence of $\mathcal{S}_{\mathcal{R}}(\tau,\sigma)$. Given a block-cluster $\tau'\times\sigma'\in\operatorname{child}(\tau\times\sigma)$, it then holds

$$\begin{aligned}\mathbf{L}|_{\tau'\times\sigma'} &= \big(\mathbf{L}|_{\tau\times\sigma}\big)|_{\tau'\times\sigma'}\\ &= \mathcal{S}(\tau,\sigma)|_{\tau'\times\sigma'}\\ &= \mathcal{S}_{\mathcal{R}}(\tau,\sigma)|_{\tau'\times\sigma'} + \mathcal{S}_{\mathcal{H}}(\tau,\sigma)|_{\tau'\times\sigma'},\end{aligned}$$

where $\mathcal{S}_{\mathcal{H}}(\tau,\sigma)|_{\tau'\times\sigma'}$ can be rewritten as

$$\begin{aligned}\mathcal{S}_{\mathcal{H}}(\tau,\sigma)|_{\tau'\times\sigma'} &= \sum_{\substack{\rho\in\mathcal{T}_{\mathcal{I}}:\\ \tau\times\rho\in\mathcal{B}\setminus\mathcal{F}\\ \rho\times\sigma\in\mathcal{B}\setminus\mathcal{F}}} \big(\mathbf{H}|_{\tau\times\rho}\mathbf{K}|_{\rho\times\sigma}\big)\big|_{\tau'\times\sigma'}\\ &= \sum_{\substack{\rho\in\mathcal{T}_{\mathcal{I}}:\\ \tau\times\rho\in\mathcal{B}\setminus\mathcal{F}\\ \rho\times\sigma\in\mathcal{B}\setminus\mathcal{F}}} \sum_{\rho'\in\operatorname{child}(\rho)} \mathbf{H}|_{\tau'\times\rho'}\mathbf{K}|_{\rho'\times\sigma'}\\ &= \sum_{\substack{\rho'\in\mathcal{T}_{\mathcal{I}}:\\ \tau'\times\rho'\in\mathcal{B}\\ \rho'\times\sigma'\in\mathcal{B}}} \mathbf{H}|_{\tau'\times\rho'}\mathbf{K}|_{\rho'\times\sigma'}.\end{aligned}$$

Each of the products in the last sum can be treated in the same manner as for the root. Thus, $\mathcal{S}_{\mathcal{H}}(\tau,\sigma)|_{\tau'\times\sigma'}$ can be represented as a `sum`-expression

$$\mathcal{S}_{\mathcal{H}}(\tau,\sigma)|_{\tau'\times\sigma'} = \mathcal{S}(\tau',\sigma') = \mathcal{S}_{\mathcal{R}}(\tau',\sigma') + \mathcal{S}_{\mathcal{H}}(\tau',\sigma'),$$

where $\mathcal{S}_{\mathcal{R}}(\tau',\sigma')$ and $\mathcal{S}_{\mathcal{H}}(\tau',\sigma')$ may be both non-empty.

The restriction of $\mathcal{S}_{\mathcal{R}}(\tau,\sigma)$ to $\tau'\times\sigma'$ can be accomplished by the restriction of the corresponding low-rank matrices. In actual implementations, the restriction of the low-rank matrices can be realized by index-shifts, and thus without further arithmetic operations.

Since the sum of two `sum`-expressions is again a `sum`-expression, we have shown that each matrix block $\mathbf{L}|_{\tau\times\sigma}$ can be represented as a `sum`-expression. A recursive algorithm for their construction is listed in Algorithm 3. When the algorithm is initialized with $\mathcal{S}(\mathcal{I},\mathcal{I}) = \mathcal{S}_{\mathcal{H}}(\mathcal{I},\mathcal{I}) = \mathbf{HK}$ and is applied recursively to all elements

of $\mathcal{B}$, it creates a `sum`-expression for each block-cluster in $\mathcal{B}$, in particular for all leaves of the farfield and the nearfield.

Algorithm 3 Given $\mathcal{S}(\tau,\sigma)$, construct $\mathcal{S}(\tau',\sigma')$ for $\tau'\times\sigma'\in\operatorname{child}(\tau\times\sigma)$.

function $\mathcal{S}(\tau',\sigma') = \textsc{restrict}(\mathcal{S}(\tau,\sigma),\ \tau'\times\sigma')$
  Set $\mathcal{S}_{\mathcal{R}}(\tau',\sigma') = \sum_i \big(\mathbf{A}_i\mathbf{B}_i^\intercal\big)\big|_{\tau'\times\sigma'}$, given $\mathcal{S}_{\mathcal{R}}(\tau,\sigma) = \sum_i \mathbf{A}_i\mathbf{B}_i^\intercal$
  Set $\mathcal{S}_{\mathcal{H}}(\tau',\sigma')$ as empty
  for $\mathbf{H}|_{\tau\times\rho}\mathbf{K}|_{\rho\times\sigma}\in\mathcal{S}_{\mathcal{H}}(\tau,\sigma)$ **do**
    for $\rho'\in\operatorname{child}(\rho)$ **do**
      if $\tau'\times\rho'\in\mathcal{F}$ or $\rho'\times\sigma'\in\mathcal{F}$ **then**
        Compute low-rank matrix $\mathbf{A}\mathbf{B}^\intercal = \mathbf{H}|_{\tau'\times\rho'}\mathbf{K}|_{\rho'\times\sigma'}$
        Set $\mathcal{S}_{\mathcal{R}}(\tau',\sigma') = \mathcal{S}_{\mathcal{R}}(\tau',\sigma') + \mathbf{A}\mathbf{B}^\intercal$
      else
        Set $\mathcal{S}_{\mathcal{H}}(\tau',\sigma') = \mathcal{S}_{\mathcal{H}}(\tau',\sigma') + \mathbf{H}|_{\tau'\times\rho'}\mathbf{K}|_{\rho'\times\sigma'}$
      end if
    end for
  end for
  Set $\mathcal{S}(\tau',\sigma') = \mathcal{S}_{\mathcal{R}}(\tau',\sigma') + \mathcal{S}_{\mathcal{H}}(\tau',\sigma')$
end function

3.3. **$\mathcal{H}$-matrix multiplication using `sum`-expressions.** The algorithm from the previous section provides us, when applied recursively, with exact representations in terms of `sum`-expressions for each matrix block $\mathbf{L}|_{\tau\times\sigma}$ for all block-clusters $\tau\times\sigma\in\mathcal{B}$. In order to compute an $\mathcal{H}$-matrix approximation of $\mathbf{L}$, we only have to convert these `sum`-expressions to dense matrices or low-rank matrices. This leads to the $\mathcal{H}$-matrix multiplication algorithm given in Algorithm 4, which is initialized with $\mathcal{S}(\mathcal{I},\mathcal{I}) = \mathcal{S}_{\mathcal{H}}(\mathcal{I},\mathcal{I}) = \mathbf{HK}$. The EVALUATE()-routine in the algorithm computes the representation of the corresponding `sum`-expression as a full matrix, whereas the $\mathfrak{T}$()-routine is a generic low-rank approximation or *truncation* operator.

Algorithm 4 $\mathcal{H}$-matrix product: Compute $\mathbf{L}|_{\tau\times\sigma} = (\mathbf{HK})|_{\tau\times\sigma}$ from $\mathcal{S}(\tau,\sigma)$

function $\mathbf{L}|_{\tau\times\sigma} = \mathcal{H}\textsc{mult}(\mathcal{S}(\tau,\sigma))$
  if τ,σ is not a leaf **then** ▷ $\mathbf{L}|_{\tau\times\sigma}$ is an $\mathcal{H}$-matrix
    for $\tau'\times\sigma'\in\operatorname{child}(\tau\times\sigma)$ **do**
      Set $\mathcal{S}(\tau',\sigma') = \textsc{restrict}(\mathcal{S}(\tau,\sigma),\ \tau'\times\sigma')$
      $\mathbf{L}|_{\tau'\times\sigma'} = \mathcal{H}\textsc{mult}(\mathcal{S}(\tau',\sigma'))$
    end for
  else
    if $\tau\times\sigma\in\mathcal{F}$ **then** ▷ $\mathbf{L}|_{\tau\times\sigma}$ is low-rank
      $\mathbf{L}|_{\tau\times\sigma} = \mathfrak{T}(\mathcal{S}(\tau,\sigma))$
    else ▷ $\mathbf{L}|_{\tau\times\sigma}$ is a dense
      $\mathbf{L}|_{\tau\times\sigma} = \textsc{evaluate}(\mathcal{S}(\tau,\sigma))$
    end if
  end if
end function

Algorithm 5 SVD of a low-rank matrix $\mathbf{L}\mathbf{R}^\intercal$, see [22, Algorithm 2.17]

function $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\intercal$=LowRankSVD($\mathbf{L}\mathbf{R}^\intercal$)
  $\mathbf{Q_L}\mathbf{R_L}$ = QR-decomposition of $\mathbf{L}$, $\mathbf{Q_L} \in \mathbb{R}^{\tau\times\tilde{k}}$, $\mathbf{R_L} \in \mathbb{R}^{\tilde{k}\times\tilde{k}}$
  $\mathbf{Q_R}\mathbf{R_R}$ = QR-decomposition of $\mathbf{R}$, $\mathbf{Q_R} \in \mathbb{R}^{\sigma\times\tilde{k}}$, $\mathbf{R_R} \in \mathbb{R}^{\tilde{k}\times\tilde{k}}$
  $\tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^\intercal = \mathrm{SVD}(\mathbf{R_L}\mathbf{R_R}^\intercal)$
  $\mathbf{U} = \mathbf{Q_L}\tilde{\mathbf{U}}$
  $\mathbf{V} = \mathbf{Q_R}\tilde{\mathbf{V}}$
end function

The algorithm can be seen as a general framework for the multiplication of $\mathcal{H}$-matrices, although several special cases are stated in the literature using different algorithmic implementations.

3.3.1. *No truncation.* In principle, the truncation operator could act as an identity. For this implementation, it was shown in [17] that the rank $\tilde{k}$ of low-rank matrices in the product is bounded by

$$\tilde{k} \le C_{\mathrm{id}} C_{\mathrm{sp}} (p+1) k,$$

where the constant $C_{\mathrm{id}} = C_{\mathrm{id}}(\mathcal{B})$ is given by

$$\begin{aligned} C_{\mathrm{id}}(\tau\times\sigma) := \#\{\tau'\times\sigma' \colon \tau' \in \mathrm{successor}(\tau), \sigma' \in \mathrm{successor}(\sigma) \text{ such that} \\ \text{there exists } \rho' \in \mathcal{T}_{\mathcal{I}} \text{ such that} \\ \tau'\times\rho' \in \mathcal{B}, \rho'\times\sigma' \in \mathcal{B}\}, \\ C_{\mathrm{id}}(\mathcal{B}) := \max_{\tau\times\sigma\in\mathcal{L}(\mathcal{B})} C_{\mathrm{id}}(\tau\times\sigma). \end{aligned}$$

Although the rank of the product is bounded from above, the constants in the bound might be large. Hence one is interested in truncating the low-rank matrices to lower rank in a best possible way. Depending on the employed truncation operator $\mathfrak{T}$, different implementations of the multiplication evolve.

3.3.2. *Truncation with a single low-rank SVD.* Traditionally, the used truncation operators are based on the singular value decomposition, from which several implementations have evolved. The most accurate implementation is given by computing the exact product in low-rank format and truncating it to a lower rank by using a singular value decomposition for low-rank matrices as given in Algorithm 5. The number of operations for the $\mathcal{H}$-matrix multiplication, assuming $n_{\min} \le k$, is then bounded by

$$43 C_{\mathrm{id}}^3 C_{\mathrm{sp}}^3 k^3 (p+1)^3 \max\{\#\mathcal{I}, \#\mathcal{F} + \#\mathcal{N}\},$$

see [17]. However, it turns out that, for more complex block-cluster trees of practical relevance, the numerical effort for this implementation of the multiplication is quite high.

3.3.3. *Truncation with multiple low-rank SVDs — fast truncation.* Therefore, one may replace the the above truncation by the fast truncation of low-rank matrices, which aims at accelerating the truncation of sums of low-rank matrices by allowing a larger error margin. The basic idea is that in many cases

$$\mathbf{M}_n\mathbf{N}_n^\intercal = \mathfrak{T}\Big(\sum_{i=1}^{n} \mathbf{A}_i\mathbf{B}_i^\intercal\Big)$$

can be sufficiently well be approximated by computing

$$\begin{aligned}\mathbf{M}_2\mathbf{N}_2^\intercal &= \mathfrak{T}\big(\mathbf{A}_1\mathbf{B}_1^\intercal + \mathbf{A}_2\mathbf{B}_2^\intercal\big)\\ \mathbf{M}_i\mathbf{N}_i^\intercal &= \mathfrak{T}\big(\mathbf{M}_{i-1}\mathbf{N}_{i-1}^\intercal + \mathbf{A}_i\mathbf{B}_i^\intercal\big), \qquad i = 3,\dots,n.\end{aligned}$$

If the fast truncation is used as a truncation operator, the number of operations for the $\mathcal{H}$-matrix multiplication, assuming $n_{\min} \leq k$, is bounded by

$$56C_{\mathrm{sp}}^2 \max\{C_{\mathrm{id}}, C_{\mathrm{sp}}\}k^2(p+1)^2\#\mathcal{I} + 184C_{\mathrm{sp}}C_{\mathrm{id}}k^3(p+1)(\#\mathcal{F} + \#\mathcal{N}),$$

see [17]. Numerical experiments confirm that the $\mathcal{H}$-matrix multiplication using the fast truncation is indeed faster, but also slightly less accurate than the previous version of the multiplication.

3.3.4. *Truncation with accumulated updates.* Recently, in [8], a new truncation operator was introduced, to which we will refer to as truncation with accumulated updates. Therefore, we slightly modify the definition of the `sum`-expression and denote the new object by $\mathcal{S}^a$.

Definition 3.3. *For a given block-cluster $\tau \times \sigma \in \mathcal{B}$, we say that the sum of a low-rank matrix $\mathbf{AB}^\intercal \in \mathcal{R}(\tau \times \sigma, k)$ and a `sum`-expression of $\mathcal{H}$-matrices $\mathcal{S}_{\mathcal{H}}(\tau, \sigma)$ is a `sum`-*expression with accumulated updates *and write*

$$\mathcal{S}^a(\tau, \sigma) = \mathbf{AB}^\intercal + \mathcal{S}_{\mathcal{H}}(\tau, \sigma).$$

In particular, we write $\mathcal{S}^a_{\mathcal{R}}(\tau, \sigma) = \mathbf{AB}^\intercal$ and, for a second low-rank matrix $\tilde{\mathbf{A}}\tilde{\mathbf{B}}^\intercal \in \mathcal{R}(\tau \times \sigma, k)$, we define the sum of these expressions with a low-rank-matrix as

$$\begin{aligned}\mathcal{S}^a_{\mathcal{R}}(\tau, \sigma) + \tilde{\mathbf{A}}\tilde{\mathbf{B}}^\intercal &= \mathfrak{T}\big(\mathbf{AB}^\intercal + \tilde{\mathbf{A}}\tilde{\mathbf{B}}^\intercal\big)\\ \mathcal{S}^a(\tau, \sigma) + \tilde{\mathbf{A}}\tilde{\mathbf{B}}^\intercal &= \mathfrak{T}\big(\mathbf{AB}^\intercal + \tilde{\mathbf{A}}\tilde{\mathbf{B}}^\intercal\big) + \mathcal{S}_{\mathcal{H}}(\tau, \sigma),\end{aligned}$$

i.e., instead of adding the new low-rank matrix to the list of low-rank matrices in $\mathcal{S}^a(\tau, \sigma)$, we perform an addition of low-rank matrices with subsequent truncation.

Obviously, every `sum`-expression with accumulated updates is also a `sum`-expression in the sense of Definition 3.1. The key point is that the addition with low-rank matrices is treated differently. By replacing the `sum`-expressions in Algorithm 4 by `sum`-expressions with accumulated updates, we obtain the $\mathcal{H}$-matrix multiplication as stated in [8]. The number of operations for this algorithm is bounded by

$$3C_{\mathrm{mm}}C_{\mathrm{sp}}^2k^2(p+1)^2\#\mathcal{I}.$$

The constant C_{mm} consists of several other constants which exceed the scope of this article and we refer to [8] for more details. However, the numerical experiments in [8] indicate that the truncation operator with accumulated updates is faster than the fast truncation operator.

An issue of both, the fast truncation and the truncation with accumulated updates, is a situation where the product of $\mathcal{H}$-matrices has to be converted into a low-rank matrix. Here, both implementations rely on a *hierarchical approximation* of the product of the $\mathcal{H}$-matrices. That is, the product is computed in $\mathcal{H}$-matrix format and then, starting from the leaves, recursively converted into low-rank format, which is a time-consuming task and requires several intermediate truncation steps. This introduces additional truncation errors, although the truncation with accumulated updates somehow reduces the number of conversions.

A slightly different approach than the fast truncation with hierarchical approximation was proposed in [10]. There, the $\mathcal{H}$-matrix products have been truncated to low-rank matrices using an iterative eigensolver based on matrix-vector multiplications before applying the fast truncation operator. The numerical experiments prove this approach to be computationally efficient, while providing even a best approximation to the product of the $\mathcal{H}$-matrices in low-rank format.

We summarize by remarking that all of the common $\mathcal{H}$-matrix multiplication algorithms are variants of Algorithm 4, employing different truncation operators. Therefore, in order to improve the accuracy and the speed of the $\mathcal{H}$-matrix multiplication, efficient and accurate truncation operators have to be used.

Since approaches based on the singular value decomposition of dense or low-rank matrices have proven to be less promising, we focus in the following on low-rank approximation methods based on matrix-vector multiplications. The idea behind this approach is that the multiplication of a `sum`-expression $\mathcal{S}(\tau,\sigma)$ with a vector $\mathbf{v}$ of length $\#\sigma$ can be computed efficiently by

$$\mathcal{S}(\tau,\sigma)\mathbf{v} = \sum_{j\in\mathcal{J}_{\mathcal{R}}} \mathbf{A}_j\big(\mathbf{B}_j^{\intercal}\mathbf{v}\big) + \sum_{j\in\mathcal{J}_{\mathcal{H}}} \mathbf{H}_j\big(\mathbf{K}_j\mathbf{v}\big).$$

Although this idea has already been mentioned in [10], the used eigensolver in [10] seemed to be less favourable for this task. In the next section, we will discuss several approaches to compute low-rank approximations to `sum`-expressions using matrix-vector multiplications. In particular, we will discuss *adaptive* algorithms, which compute low-rank approximations to `sum`-expressions up to a prescribed error tolerance. The adaptive algorithms will allow us to compute the best-approximation of $\mathcal{H}$-matrix products.

4. Low-rank approximation schemes

In addition to the well known hierarchical approximation for the approximation of $\mathcal{H}$-matrices by low-rank matrices, we consider here three different schemes for the low-rank approximation of a given matrix. All of them can be implemented in terms of elementary matrix-vector products and are therefore well suited for the use in our new $\mathcal{H}$-matrix multiplication. In what follows, let $\mathbf{A} := \mathbf{H}|_{\tau\times\sigma} \in \mathbb{R}^{m\times n}$, $m = \#\tau$, $n = \#\sigma$, always denote a target matrix block, which might be implicitly given in terms of a `sum`-expression $\mathcal{S}(\tau,\sigma)$.

4.1. **Adaptive cross approximation.**

In the context of boundary element methods, the *adaptive cross approximation* (ACA), see [4], is frequently used to find $\mathcal{H}$-matrix approximations to system matrices. However, one can prove, see [12], that the same idea can also be applied to the product of pseudodifferential operators. Since the $\mathcal{H}$-matrix multiplication can be seen as a discrete analogon to the multiplication of pseudodifferential operators, we may use ACA to find the low-rank approximations for the admissible matrix blocks. Concretely, we approximate $\mathbf{A} = \mathbf{H}|_{\tau\times\sigma}$ by a partially pivoted Gaussian elimination, see [4] for further details. To this end, we define the vectors $\boldsymbol{\ell}_r \in \mathbb{R}^m$ and $\mathbf{u}_r \in \mathbb{R}^n$ by the iterative scheme shown in Algorithm 6, where $\mathbf{A} = [a_{i,j}]_{i,j}$ is the matrix-block under consideration.

A suitable criterion that guarantees the convergence of the algorithm is to choose the pivot element located in the (i_r, j_r)-position as the maximum element in modulus of the remainder $\mathbf{A} - \mathbf{L}_{r-1}\mathbf{U}_{r-1}$. Unfortunately, this would compromise the

Algorithm 6 Adaptive cross approximation (ACA)

for $r = 1, 2, \ldots$ **do**
  Choose the element in (i_r, j_r)-position of the Schur compolement as pivot
  $\hat{\mathbf{u}}_r = [a_{i_r,j}]_{j=1}^n - \sum_{j=1}^{r-1} [\boldsymbol{\ell}_j]_{i_r} \mathbf{u}_j$
  $\mathbf{u}_r = \hat{\mathbf{u}}_r / [\hat{\mathbf{u}}_r]_{j_r}$
  $\boldsymbol{\ell}_r = [a_{i,j_r}]_{i=1}^m - \sum_{i=1}^{r-1} [u_i]_{j_r} \boldsymbol{\ell}_i$
end for
Set $\mathbf{L}_r := [\boldsymbol{\ell}_1, \ldots, \boldsymbol{\ell}_r]$ and $\mathbf{U}_r := [\mathbf{u}_1, \ldots, \mathbf{u}_r]^\intercal$

overall cost of the approximation. Therefore, we resort to another pivoting strategy which is sufficient in most cases: we choose j_r such that $[\hat{\mathbf{u}}_r]_{j_r}$ is the largest element in modulus of the row $\hat{\mathbf{u}}_r$.

Obviously, the cost for the computation of the rank-k-approximation $\mathbf{L}_k \mathbf{U}_k$ to the block $\mathbf{A}$ is $\mathcal{O}\big(k^2(m+n)\big)$ and the storage cost is $\mathcal{O}\big(k(m+n)\big)$. In addition, if $\mathbf{A}$ is given via a `sum`-expression, we have to compute $\mathbf{e}_{i_m}^\intercal \mathbf{A}$ and $\mathbf{A}\mathbf{e}_{j_m}$ in each step, where $\mathbf{e}_i$ denotes the i-th unit vector, in order to retrieve the row and column under consideration. The respective computational cost for the multiplication of a `sum`-expression with a vector is estimated in Lemma 5.3.

4.2. **Lanczos bidiagonalization.** The second algorithm we consider for compressing a given matrix block is based on the Lanczos bidiagonalization (BiLanczos), see Algorithm 7. This procedure is equivalent to the tridiagonalization of the corresponding, symmetric Jordan-Wielandt matrix

$$\begin{bmatrix} \mathbf{0} & \mathbf{A} \\ \mathbf{A}^\intercal & \mathbf{0} \end{bmatrix},$$

cf. [15].

Algorithm 7 Golub-Kahan-Lanczos bidiagonalization

Choose a random vector $\mathbf{w}_1$ with $\|\mathbf{w}_1\| = 1$ and set $\mathbf{q}_0 = \mathbf{0}, \beta_0 = 0$
for $r = 1, 2, \ldots$ **do**
  $\mathbf{q}_r = \mathbf{A}\mathbf{w}_r - \beta_{r-1}\mathbf{q}_{r-1}$
  $\alpha_r = \|\mathbf{q}_r\|_2$
  $\mathbf{q}_r = \mathbf{q}_r / \alpha_r$
  $\mathbf{w}_{r+1} = \mathbf{A}^\intercal \mathbf{q}_r - \alpha_r \mathbf{w}_r$
  $\beta_r = \|\mathbf{w}_{r+1}\|_2$
  $\mathbf{w}_{r+1} = \mathbf{w}_{r+1} / \beta_r$
end for

The algorithm leads to a decomposition of the given matrix block according to

$$\mathbf{Q}^\intercal \mathbf{A} \mathbf{W} = \mathbf{B} := \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ & \alpha_2 & \beta 2 & & \\ & & \ddots & \ddots & \\ & & & \alpha_{n-1} & \beta_{n-1} \\ & & & & \alpha_n \end{bmatrix}$$

with orthogonal matrices $\mathbf{Q}^\intercal \mathbf{Q} = \mathbf{I}_m$ and $\mathbf{W}^\intercal \mathbf{W} = \mathbf{I}_n$, cf. [15]. Note that although the algorithm yields orthogonal vectors $\mathbf{q}_r$ and $\mathbf{w}_r$ by construction, we have to

perform additional reorthogonalization steps to obtain numerical stability. Since the algorithm, like ACA, only depends on matrix-vector multiplications, it is well suited to compress a given block $\mathbf{A}$.

Truncating the algorithm after k steps results in a low-rank approximation

$$\mathbf{A} \approx \mathbf{Q}_k \mathbf{B}_k \mathbf{W}_k^\intercal = \mathbf{U}_k \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ & \alpha_2 & \beta_2 & & \\ & & \ddots & \ddots & \\ & & & \alpha_{k-1} & \beta_{k-1} \\ & & & & \alpha_k \end{bmatrix} \mathbf{V}_k^\intercal.$$

It is then easy to compute a singular value decomposition $\mathbf{B}_k = \tilde{\mathbf{U}}\mathbf{S}\tilde{\mathbf{V}}^\intercal$ and to obtain the separable decomposition

$$\mathbf{A} \approx (\mathbf{Q}_k \tilde{\mathbf{U}}\mathbf{S})(\mathbf{W}_k \tilde{\mathbf{V}})^\intercal = \mathbf{L}_k \mathbf{U}_k.$$

As in the ACA case, the algorithm only requires two matrix-vector products with the block $\mathbf{A}$ in each step.

4.3. **Randomized low-rank approximation.** The third algorithm we consider for finding a low-rank approximation to $\mathbf{A}$ is based on successive multiplication with Gaussian random vectors. The algorithm can be motivated as follows, cf. [25]: Let $\mathbf{y}_i = \mathbf{A}\boldsymbol{\omega}_i$ for $i = 1, \dots, r$, where $\boldsymbol{\omega}_1, \dots, \boldsymbol{\omega}_r \in \mathbb{R}^n$ are independently drawn Gaussian random vectors. The collection of these random vectors is very likely to be linearly independent, whereas it is very unlikely that any linear combination of them falls in the null space of $\mathbf{A}$. As a consequence, the collection $\mathbf{y}_1, \dots, \mathbf{y}_r$ is linearly independent and spans the range of $\mathbf{A}$. Thus, by orthogonalizing $[\mathbf{y}_1, \dots, \mathbf{y}_r] = \mathbf{L}_r \mathbf{R}$, with an orthogonal matrix $\mathbf{L}_r \in \mathbb{R}^{m \times r}$, we obtain $\mathbf{A} \approx \mathbf{L}_r \mathbf{U}_r$ with $\mathbf{U}_r = \mathbf{L}_r^\intercal \mathbf{A}$, see Algorithm 8. Employing oversampling, the quality of the approximation can be increased even further. For all the details, we refer to [25]. As the previous

Algorithm 8 Randomized low-rank approximation

Set $\mathbf{L}_0 = [\,]$
for $r = 1, 2, \dots$ **do**
  Generate a Gaussian random vector $\boldsymbol{\omega} \in \mathbb{R}^n$
  $\boldsymbol{\ell}_r = (\mathbf{I} - \mathbf{L}_{r-1}\mathbf{L}_{r-1}^\intercal)\mathbf{A}\boldsymbol{\omega}$
  $\boldsymbol{\ell}_r = \boldsymbol{\ell}_r \,/\, \|\boldsymbol{\ell}_r\|_2$
  $\mathbf{L}_r = [\mathbf{L}_{r-1}, \boldsymbol{\ell}_r]$
  $\mathbf{U}_r = \mathbf{L}_r^\intercal \mathbf{A}$
end for

two algorithms, the randomized low-rank approximation requires only two matrix-vector multiplications with $\mathbf{A}$ in each step.

In contrast to the other two presented compression schemes, the randomized approximation allows for a blocked version in a straightforward manner, where instead of a single Gaussian random vector, a Gaussian matrix can be used to approximate the range of $\mathbf{A}$. Although there also exist block versions of the ACA and the BiLanczos, as well, they are know to be numerically very unstable.

Note that there exist probabilistic error estimates for the approximation of a given matrix by Algorithm 8, cf. [25]. Unfortunately, these error estimates are with

respect to the spectral norm and therefore only give insights on the largest singular value of the remainder. To have control on the actual approximation quality of a certain matrix block, we are thus rather interest in an error estimate with respect to the Frobenius norm. To that end, we propose a different, adaptive criterion which estimates the error with respect to the Frobenius norm.

4.4. **Adaptive stopping criterion.** In our implementation, the proposed compression schemes rely on the following well known adaptive stopping criterion, which aims at reflecting the approximation error with respect to the Frobenius norm. We terminate the approximation if the criterion

$$\|\boldsymbol{\ell}_{k+1}\|_2\|\mathbf{u}_{k+1}\|_2 \leq \varepsilon\|\mathbf{L}_k\mathbf{U}_k\|_F \tag{2}$$

is met for some desired accuracy $\varepsilon > 0$. This criterion can be justified as follows. We assume that the error in each step is reduced by a constant rate $0 < \vartheta < 1$, i.e.,

$$\|\mathbf{A} - \mathbf{L}_{k+1}\mathbf{U}_{k+1}\|_F \leq \vartheta\|\mathbf{A} - \mathbf{L}_k\mathbf{U}_k\|_F.$$

Then, there holds

$$\begin{aligned}\|\boldsymbol{\ell}_{k+1}\|_2\|\mathbf{u}_{k+1}\|_2 &= \|\mathbf{L}_{k+1}\mathbf{U}_{k+1} - \mathbf{L}_k\mathbf{U}_k\|_F \\ &\leq \|\mathbf{A} - \mathbf{L}_{k+1}\mathbf{U}_{k+1}\|_F + \|\mathbf{A} - \mathbf{L}_k\mathbf{U}_k\|_F \\ &\leq (1+\vartheta)\|\mathbf{A} - \mathbf{L}_k\mathbf{U}_k\|_F\end{aligned}$$

and, vice versa,

$$\begin{aligned}\|\mathbf{L}_{k+1}\mathbf{U}_{k+1} - \mathbf{L}_k\mathbf{U}_k\|_F &\geq \|\mathbf{A} - \mathbf{L}_k\mathbf{U}_k\|_F - \|\mathbf{A} - \mathbf{L}_{k+1}\mathbf{U}_{k+1}\|_F \\ &\geq (1-\vartheta)\|\mathbf{A} - \mathbf{L}_k\mathbf{U}_k\|_F.\end{aligned}$$

Therefore, the approximation error is proportional to the norm

$$\|\boldsymbol{\ell}_{k+1}\mathbf{u}_{k+1}\|_F = \|\boldsymbol{\ell}_{k+1}\|_2\|\mathbf{u}_{k+1}\|_2$$

of the update vectors, i.e.,

$$(1-\vartheta)\|\mathbf{A} - \mathbf{L}_k\mathbf{U}_k\|_F \leq \|\boldsymbol{\ell}_{k+1}\|_2\|\mathbf{u}_{k+1}\|_2 \leq (1+\vartheta)\|\mathbf{A} - \mathbf{L}_k\mathbf{U}_k\|_F.$$

Thus, together with (2), we can guarantee a relative error bound

$$\|\mathbf{A} - \mathbf{L}_k\mathbf{U}_k\|_F \leq \frac{\varepsilon}{1-\vartheta}\|\mathbf{L}_k\mathbf{U}_k\|_F \leq \frac{\varepsilon}{1-\vartheta}\|\mathbf{A}\|_F. \tag{3}$$

Based on this blockwise error estimate, it is straightforward to assess the overall error for the approximation of a given $\mathcal{H}$-matrix.

Theorem 4.1. *Let* $\mathbf{H}$ *be the uncompressed matrix and* $\tilde{\mathbf{H}}$ *be the matrix which is compressed by one of the aforementioned compression schemes. Then, with respect to the Frobenius norm, there holds the error estimate*

$$\|\mathbf{H} - \tilde{\mathbf{H}}\|_F \lesssim \varepsilon\|\mathbf{H}\|_F$$

provided that the blockwise error satisfies (3)*.*

Proof. In view of (3), we have

$$\begin{aligned}\|\mathbf{H} - \tilde{\mathbf{H}}\|_F^2 &= \sum_{\tau\times\sigma\in\mathcal{F}} \big\|\mathbf{H}|_{\tau\times\sigma} - \tilde{\mathbf{H}}|_{\tau\times\sigma}\big\|_F^2 \\ &\lesssim \sum_{\tau\times\sigma\in\mathcal{F}} \|\mathbf{H}|_{\tau\times\sigma}\|_F^2 \\ &= \varepsilon^2\|\mathbf{H}\|_F^2.\end{aligned}$$

Taking square roots on both sides yields the assertion. □

4.5. **Fixed rank approximation.** The traditional $\mathcal{H}$-matrix multiplication is based on low-rank approximations to a fixed a-priori prescribed rank. We will therefore shortly comment on the fixed rank versions of the introduced algorithms which we will later also use in the numerical experiments. Since ACA and the BiLanczos algorithm are intrinsically iterative methods, we stop the iteration whenever the prescribed rank is reached. For the randomized low-rank approximation we may use a single iteration of its blocked variant. The corresponding algorithm featuring also additional $q \in \mathbb{N}_0$ subspace iterations to increase accuracies is listed in Algorithm 9, cp. [25]. For practical purposes, when the singular values of $\mathbf{A}$ decay

Algorithm 9 Randomized rank-k approximation with subspace iterations

Generate a Gaussian random matrix $\mathbf{\Omega} \in \mathbb{R}^{n\times k}$
$\mathbf{L} = \mathbf{A\Omega}$
Orthonormalize columns of $\mathbf{L}$
for $\ell = 1, \dots, q$ **do**
  $\mathbf{U} = \mathbf{A}^\intercal\mathbf{L}$
  Orthonormalize columns of $\mathbf{U}$
  $\mathbf{L} = \mathbf{AU}$
  Orthonormalize columns of $\mathbf{L}$
end for
$\mathbf{U} = \mathbf{A}^\intercal\mathbf{L}$

sufficiently fast, a value of $q = 1$ is usually sufficient. We will therefore use $q = 1$ in the numerical experiments. For a detailed discussion on the number of subspace iterations and comments on *oversampling*, i.e., sampling at a higher rank with a subsequent truncation, we refer to [25].

5. Cost of the $\mathcal{H}$-matrix multiplication

The following section is dedicated to the cost analysis of the $\mathcal{H}$-matrix multiplication as introduced in Algorithm 4. We first estimate the cost for the computation of the `sum`-expressions and then proceed by analyzing the multiplication of a `sum`-expression with a vector. Having these estimates at our disposal, the main theorem of this section confirms that the cost of the $\mathcal{H}$-matrix multiplication scales almost linearly with the cardinality of the index set $\mathcal{I}$.

Lemma 5.1. *Given a block-cluster $\tau \times \sigma \in \mathcal{B}$ with `sum`-expression $\mathcal{S}(\tau, \sigma)$, the `sum`-expression $\mathcal{S}(\tau', \sigma')$ for any block-cluster $\tau' \times \sigma' \in \operatorname{child}(\tau \times \sigma)$ can be computed in at most*

$$N_{\text{update},\mathcal{S}}(\tau', \sigma') \leq \sum_{\substack{\rho' \in \mathcal{T}_\mathcal{I}:\\ \tau' \times \rho' \in \mathcal{B}\setminus\mathcal{F}\\ \rho' \times \sigma' \in \mathcal{F}}} kN_{\mathcal{H}\cdot v}(\tau' \times \rho', k) + \sum_{\substack{\rho' \in \mathcal{T}_\mathcal{I}:\\ \tau' \times \rho' \in \mathcal{F}\\ \rho' \times \sigma' \in \mathcal{B}\setminus\mathcal{F}}} kN_{\mathcal{H}\cdot v}(\rho' \times \sigma', k) + \sum_{\substack{\rho' \in \mathcal{T}_\mathcal{I}:\\ \tau' \times \rho' \in \mathcal{F}\\ \rho' \times \sigma' \in \mathcal{F}}} k\big(\#\rho' + \min\{\#\tau', \#\sigma'\}\big)$$

operations.

Proof. We start with recalling that $\mathcal{S}(\tau',\sigma')$ is recursively given as

$$\begin{aligned}\mathcal{S}(\tau',\sigma') &= \mathcal{S}(\tau,\sigma)|_{\tau'\times\sigma'}\\ &= \mathcal{S}_{\mathcal{R}}(\tau,\sigma)|_{\tau'\times\sigma'} + \mathcal{S}_{\mathcal{H}}(\tau,\sigma)|_{\tau'\times\sigma'}\\ &= \mathcal{S}_{\mathcal{R}}(\tau,\sigma)|_{\tau'\times\sigma'} + \sum_{\substack{\rho'\in\mathcal{T}_{\mathcal{I}}:\\ \tau'\times\rho'\in\mathcal{B}\\ \rho'\times\sigma'\in\mathcal{B}}} \mathbf{H}|_{\tau'\times\rho'}\mathbf{H}|_{\rho'\times\sigma'}, \qquad (4)\end{aligned}$$

with $\mathcal{S}_{\mathcal{R}}(\mathcal{I},\mathcal{I}) = \emptyset$. Clearly, the restriction of $\mathcal{S}_{\mathcal{R}}(\tau,\sigma)$ to $\tau'\times\sigma'$ comes for free, and we only have to look at the remaining sum. It can be decomposed into

$$\sum_{\substack{\rho'\in\mathcal{T}_{\mathcal{I}}:\\ \tau'\times\rho'\in\mathcal{B}\\ \rho'\times\sigma'\in\mathcal{B}}} \mathbf{H}|_{\tau'\times\rho'}\mathbf{H}|_{\rho'\times\sigma'} = \sum_{\substack{\rho'\in\mathcal{T}_{\mathcal{I}}:\\ \tau'\times\rho'\in\mathcal{B}\setminus\mathcal{F}\\ \rho'\times\sigma'\in\mathcal{B}\setminus\mathcal{F}}} \mathbf{H}|_{\tau'\times\rho'}\mathbf{H}|_{\rho'\times\sigma'} \qquad (5)$$

$$+ \sum_{\substack{\rho'\in\mathcal{T}_{\mathcal{I}}:\\ \tau'\times\rho'\in\mathcal{B}\setminus\mathcal{F}\\ \rho'\times\sigma'\in\mathcal{F}}} \mathbf{H}|_{\tau'\times\rho'}\mathbf{H}|_{\rho'\times\sigma'} \qquad (6)$$

$$+ \sum_{\substack{\rho'\in\mathcal{T}_{\mathcal{I}}:\\ \tau'\times\rho'\in\mathcal{F}\\ \rho'\times\sigma'\in\mathcal{B}\setminus\mathcal{F}}} \mathbf{H}|_{\tau'\times\rho'}\mathbf{H}|_{\rho'\times\sigma'} \qquad (7)$$

$$+ \sum_{\substack{\rho'\in\mathcal{T}_{\mathcal{I}}:\\ \tau'\times\rho'\in\mathcal{F}\\ \rho'\times\sigma'\in\mathcal{F}}} \mathbf{H}|_{\tau'\times\rho'}\mathbf{H}|_{\rho'\times\sigma'}. \qquad (8)$$

The products on the right-hand side in (5) are not computed, thus, no numerical effort has to be made. The products in (6), (7), and (8) must be computed and stored as low-rank matrices. The computational effort for this operation is $kN_{\mathcal{H}\cdot v}(\tau'\times\rho',k)$ for (6), $kN_{\mathcal{H}\cdot v}(\rho'\times\sigma',k)$ for (7), and $k^2(\#\rho' + \min\{\#\tau',\#\sigma'\})$ for (8). Thus, given $\mathcal{S}(\tau,\sigma)$, the computational cost to compute $\mathcal{S}(\tau',\sigma')$ is

$$\sum_{\substack{\rho'\in\mathcal{T}_{\mathcal{I}}:\\ \tau'\times\rho'\in\mathcal{B}\setminus\mathcal{F}\\ \rho'\times\sigma'\in\mathcal{F}}} kN_{\mathcal{H}\cdot v}(\tau'\times\rho',k) + \sum_{\substack{\rho'\in\mathcal{T}_{\mathcal{I}}:\\ \tau'\times\rho'\in\mathcal{F}\\ \rho'\times\sigma'\in\mathcal{B}\setminus\mathcal{F}}} kN_{\mathcal{H}\cdot v}(\rho'\times\sigma',k)$$

$$+ \sum_{\substack{\rho'\in\mathcal{T}_{\mathcal{I}}:\\ \tau'\times\rho'\in\mathcal{F}\\ \rho'\times\sigma'\in\mathcal{F}}} k\big(\#\rho' + \min\{\#\tau',\#\sigma'\}\big),$$

which proves the assertion. □

Lemma 5.2. *The $\mathcal{H}$-matrix multiplication as given by Algorithm 4 requires at most*

$$N_{\mathcal{S}}(\mathcal{B}) \le 16C_{\mathrm{sp}}^3 k\max\{k,n_{\min}\}(p+1)^2\#\mathcal{I}$$

operations for the computation of the `sum`*-expressions.*

Proof. We consider a block-cluster $\tau\times\sigma\in\mathcal{B}\setminus\mathcal{F}$ on level j. Then, the numerical effort to compute the `sum`-expression $\mathcal{S}(\tau',\sigma')$ for $\tau'\times\sigma'\in\mathrm{child}(\tau\times\sigma)$ is estimated

by Lemma 5.1, if the `sum`-expression $\mathcal{S}(\tau,\sigma)$ is known. Therefore, it is sufficient to sum over all block-clusters in $\mathcal{B}$ as follows:

$$\begin{aligned}
&\sum_{\tau\times\sigma\in\mathcal{B}} N_{\mathrm{update},\mathcal{S}}(\tau,\sigma)\\
&\quad\le \sum_{\tau\times\sigma\in\mathcal{B}}\Bigg(2\sum_{\substack{\rho\in\mathcal{T}_\mathcal{I}:\\ \tau\times\rho\in\mathcal{B}\setminus\mathcal{F}\\ \rho\times\sigma\in\mathcal{F}}} kN_{\mathcal{H}\cdot v}(\tau\times\rho,k)+\sum_{\substack{\rho\in\mathcal{T}_\mathcal{I}:\\ \tau\times\rho\in\mathcal{F}\\ \rho\times\sigma\in\mathcal{F}}} k\big(\#\rho+\min\{\#\tau,\#\sigma\}\big)\Bigg)\\
&\quad= 2\sum_{\tau\times\sigma\in\mathcal{B}}\sum_{\substack{\rho\in\mathcal{T}_\mathcal{I}:\\ \tau\times\rho\in\mathcal{B}\setminus\mathcal{F}\\ \rho\times\sigma\in\mathcal{F}}} kN_{\mathcal{H}\cdot v}(\tau\times\rho,k)+\sum_{\tau\times\sigma\in\mathcal{B}}\sum_{\substack{\rho\in\mathcal{T}_\mathcal{I}:\\ \tau\times\rho\in\mathcal{F}\\ \rho\times\sigma\in\mathcal{F}}} k\big(\#\rho+\min\{\#\tau,\#\sigma\}\big).
\end{aligned}$$

To estimate the first sum, consider

$$\begin{aligned}
\sum_{\tau\times\sigma\in\mathcal{B}}\sum_{\substack{\rho\in\mathcal{T}_\mathcal{I}:\\ \tau\times\rho\in\mathcal{B}\setminus\mathcal{F}\\ \rho\times\sigma\in\mathcal{F}}} kN_{\mathcal{H}\cdot v}(\tau\times\rho,k) &\le \sum_{\tau\times\sigma\in\mathcal{B}}\sum_{\substack{\rho\in\mathcal{T}_\mathcal{I}:\\ \tau\times\rho\in\mathcal{B}\setminus\mathcal{F}}} kN_{\mathcal{H}\cdot v}(\tau\times\rho,k)\\
&\le C_{\mathrm{sp}}\sum_{\tau\times\rho\in\mathcal{B}} kN_{\mathcal{H}\cdot v}(\tau\times\rho,k)\\
&\le 2C_{\mathrm{sp}}^2 k\max\{k,n_{\min}\}(p+1)\sum_{\tau\times\rho\in\mathcal{B}}(\#\tau+\#\rho)\\
&\le 4C_{\mathrm{sp}}^3 k\max\{k,n_{\min}\}(p+1)^2\#\mathcal{I},
\end{aligned}$$

due to the fact that

$$\sum_{\tau\times\sigma\in\mathcal{B}}(\#\tau+\#\sigma)\le 2C_{\mathrm{sp}}(p+1)\#\mathcal{I},$$

see, e.g., [17, Lemma 2.4]. Since the second sum can be estimated by

$$\begin{aligned}
\sum_{\tau\times\sigma\in\mathcal{B}}\sum_{\substack{\rho\in\mathcal{T}_\mathcal{I}:\\ \tau\times\rho\in\mathcal{F}\\ \rho\times\sigma\in\mathcal{F}}} k\big(\#\rho+\min\{\#\tau,\#\sigma\}\big) &\le \sum_{\substack{\tau\times\rho\in\mathcal{B}\\ \rho\times\sigma\in\mathcal{B}}} k\big(\#\rho+\min\{\#\tau,\#\sigma\}\big)\\
&\le \sum_{\substack{\tau\times\rho\in\mathcal{B}\\ \rho\times\sigma\in\mathcal{B}}} k\big(2\#\rho+\#\tau+\#\sigma\big)\\
&\le 2C_{\mathrm{sp}}k(p+1)\#\mathcal{I},
\end{aligned}$$

summing up yields the assertion. □

Lemma 5.3. *For any block-cluster $\tau\times\sigma\in\mathcal{B}$, the multiplication of $\mathcal{S}(\tau,\sigma)$ with a vector of length $\#\sigma$ can be accomplished in at most*

$$4C_{\mathrm{sp}}\max\{k,n_{\min}\}(p+1)\sum_{\substack{\rho\in\mathcal{T}_\mathcal{I}:\\ \tau\times\rho\in\mathcal{B}\setminus\mathcal{F}\\ \rho\times\sigma\in\mathcal{B}\setminus\mathcal{F}}}(\#\tau+\#\rho+\#\sigma)$$

operations.

Proof. We first estimate the number of elements in $\mathcal{S}_\mathcal{R}(\tau,\sigma)$ and $\mathcal{S}_\mathcal{H}(\tau,\sigma)$. Therefore, we remark that, for fixed τ, there are at most C_{sp} block-cluster pairs $\tau\times\rho$ in $\mathcal{B}$ and that the same consideration holds for σ. Thus, looking at the recursion (4), the

recursion step from $\mathcal{S}\big(\operatorname{parent}(\tau),\operatorname{parent}(\sigma)\big)$ to $\mathcal{S}(\tau,\sigma)$ adds at most C_{sp} low-rank matrices. Thus, considering that $\mathcal{S}(\mathcal{I},\mathcal{I})=\emptyset$, we have at most $C_{\mathrm{sp}}\operatorname{level}(\tau\times\sigma)$ low-rank matrices in $\mathcal{S}(\tau,\sigma)$. Summing up, the multiplication of $\mathcal{S}(\tau,\sigma)$ with a vector requires at most

$$\begin{aligned}
&C_{\mathrm{sp}}k\operatorname{level}(\tau\times\sigma)(\#\tau+\#\sigma)+\sum_{\substack{\rho\in\mathcal{T}_{\mathcal{I}}:\\ \tau\times\rho\in\mathcal{B}\setminus\mathcal{F}\\ \rho\times\sigma\in\mathcal{B}\setminus\mathcal{F}}}\Big(N_{\mathcal{H}\cdot v}(\tau\times\rho,k)+N_{\mathcal{H}\cdot v}(\rho\times\sigma,k)\Big)\\
&\qquad\le C_{\mathrm{sp}}k(p+1)(\#\tau+\#\sigma)\\
&\qquad\qquad+\sum_{\substack{\rho\in\mathcal{T}_{\mathcal{I}}:\\ \tau\times\rho\in\mathcal{B}\setminus\mathcal{F}\\ \rho\times\sigma\in\mathcal{B}\setminus\mathcal{F}}}2C_{\mathrm{sp}}\max\{k,n_{\min}\}(p+1)(\#\tau+2\#\rho+\#\sigma)\\
&\qquad\le 4C_{\mathrm{sp}}\max\{k,n_{\min}\}(p+1)\sum_{\substack{\rho\in\mathcal{T}_{\mathcal{I}}:\\ \tau\times\rho\in\mathcal{B}\setminus\mathcal{F}\\ \rho\times\sigma\in\mathcal{B}\setminus\mathcal{F}}}(\#\tau+\#\rho+\#\sigma)
\end{aligned}$$

operations. □

With the help of the previous lemmata, we are now able to state our main result, which estimates the number of operations for the $\mathcal{H}$-matrix multiplication from Algorithm 4.

Theorem 5.4. *Assuming that the range approximation scheme $\mathfrak{T}(\mathbf{M},k^\star)$ requires $\ell k^\star$, $\ell\ge 1$, matrix-vector multiplications of $\mathbf{M}$ and additionally $N_{\mathfrak{T}}(k^\star)$ operations to find a rank $k^\star$ approximation to a matrix $\mathbf{M}\in\mathbb{R}^{m\times n}$, then the $\mathcal{H}$-matrix multiplication as stated in Algorithm 4 requires at most*

$$8C_{\mathrm{sp}}^2(\ell k^\star+n_{\min}+2C_{\mathrm{sp}})k\max\{k,n_{\min}\}(p+1)^2\#\mathcal{I}+C_{\mathrm{sp}}(2\#\mathcal{I}-1)N_{\mathfrak{T}}(k^\star)$$

operations.

Proof. We start by estimating the number of operations for a single far-field block $\tau\times\sigma\in\mathcal{F}$. Using Lemma 5.3, this requires at most

$$4C_{\mathrm{sp}}\ell k^\star\max\{k,n_{\min}\}(p+1)\sum_{\substack{\rho\in\mathcal{T}_{\mathcal{I}}:\\ \tau\times\rho\in\mathcal{B}\setminus\mathcal{F}\\ \rho\times\sigma\in\mathcal{B}\setminus\mathcal{F}}}(\#\tau+\#\rho+\#\sigma)+N_{\mathfrak{T}}(k^\star)$$

operations. Summing up over all farfield blocks yields an effort of at most

$$\begin{aligned}
&\sum_{\tau\times\sigma\in\mathcal{F}}\left(4C_{\mathrm{sp}}\ell k^\star\max\{k,n_{\min}\}(p+1)\sum_{\substack{\rho\in\mathcal{T}_{\mathcal{I}}:\\ \tau\times\rho\in\mathcal{B}\setminus\mathcal{F}\\ \rho\times\sigma\in\mathcal{B}\setminus\mathcal{F}}}(\#\tau+\#\rho+\#\sigma)+N_{\mathfrak{T}}(k^\star)\right)\\
&\qquad\le 4C_{\mathrm{sp}}\ell k^\star\max\{k,n_{\min}\}(p+1)\sum_{\substack{\tau\times\rho\in\mathcal{B}\\ \rho\times\sigma\in\mathcal{B}}}(\#\tau+2\#\rho+\#\sigma)+\sum_{\tau\times\sigma\in\mathcal{F}}N_{\mathfrak{T}}(k^\star)\\
&\qquad\le 8C_{\mathrm{sp}}^2\ell k^\star k\max\{k,n_{\min}\}(p+1)^2\#\mathcal{I}+C_{\mathrm{sp}}(2\#\mathcal{I}-1)N_{\mathfrak{T}}(k^\star),
\end{aligned}$$

where we have used that $\#\mathcal{F}\le C_{\mathrm{sp}}(2\#\mathcal{I}-1)$, cf. [22, Lemma 6.11]. Assuming that a nearfield block $\tau\times\sigma\in\mathcal{N}$ is computed by applying its corresponding `sum`-expression $\mathcal{S}(\tau,\sigma)$ to an identity matrix of size at most $n_{\min}\times n_{\min}$, the nearfield

blocks of the $\mathcal{H}$-matrix product can be computed in at most

$$8C_{\mathrm{sp}}^2 k \max\{k, n_{\min}\} n_{\min} (p+1)^2 \#\mathcal{I}$$

operations. Summing up the operations for the farfield, the nearfield, and the operations of the `sum`-expressions from Lemma 5.2 yields the assertion. □

We remark that, with the convention $p = \mathrm{depth}(\mathcal{B}) = \mathcal{O}(\log(\#\mathcal{I}))$, the previous theorem states that the $\mathcal{H}$-matrix multiplication from Algorithm 4 has an almost linear cost with respect to $\#\mathcal{I}$. To that end, for given $\mathcal{I}$, we require that the blockwise ranks of the $\mathcal{H}$-matrix are bounded by a constant $k^\star$. That constant may depend on $\mathcal{I}$. According to [12], this is reasonable for $\mathcal{H}$-matrices which arise from the discretization of pseudodifferential operators. In particular, it is well known that $k^\star$ depends poly-logarithmically on $\#\mathcal{I}$ for suitable approximation accuracies ε.

6. Numerical examples

The following section is dedicated to a comparison of the new $\mathcal{H}$-matrix multiplication to the original algorithm from [21], see also Section 3.3.3. Besides the comparison between these two algorithms, we also compare different truncation operators. The considered configurations are listed in Table 1. Note that, in order to compute the dense SVD of a `sum`-expression, we compute the SVD of the `sum`-expression applied to a dense identity matrix.

	traditional multiplication	new multiplication
sum of low-rank matrices	truncation with SVD of low-rank matrices, see Algorithm 5	—
conversion of products of $\mathcal{H}$-matrix blocks to low rank	• Hierarchical approximation, see [21] • ACA, see Section 4.1 • BiLanczos, see Section 4.2 • Randomized, see Setion 4.3 • Reference: dense SVD	—
conversion of `sum`-expressions to low rank	combination of the operations above, see discussion in Section 3	• ACA, see Section 4.1 • BiLanczos, see Section 4.2 • Randomized, see Setion 4.3 • Reference: dense SVD

Table 1. Considered configurations of the $\mathcal{H}$-matrix multiplication for the numerical experiments.

We would like to compare the different configurations in terms of speed and accuracy. For computational efficiency, the traditional $\mathcal{H}$-matrix arithmetic puts an upper bound $k_{\max}$ on the truncation rank. However, for computational accuracy, one should rather put an upper bound on the truncation error. This is also referred to as ε-rank, where the truncation is with respect to the smallest possible rank such that the relative truncation error is smaller than ε. We perform the experiments for both types of bounds, with $k_{\max} = 16$ and $\varepsilon = 10^{-12}$.

6.1. **Example with exponential kernels.** For our first numerical example, we consider the Galerkin discretizations $\mathbf{K}_1$ and $\mathbf{K}_2$ of an exponential kernel

$$k_1(\mathbf{x},\mathbf{y}) = \exp(-\|\mathbf{x}-\mathbf{y}\|)$$

and a scaled exponential kernel

$$k_2(\mathbf{x},\mathbf{y}) = x_1 \exp(-\|\mathbf{x}-\mathbf{y}\|)$$

on the unit sphere with boundary Γ. This means that, for a given finite element space $V_N = \operatorname{span}\{\varphi_1,\dots,\varphi_N\}$ on Γ, the system matrices are given by

$$\big[\mathbf{K}_\ell\big]_{ij} = \int_\Gamma \int_\Gamma k_\ell(\mathbf{x},\mathbf{y})\varphi_j(\mathbf{x})\varphi_i(\mathbf{y})\,\mathrm{d}\sigma_{\mathbf{x}}\,\mathrm{d}\sigma_{\mathbf{y}}$$

for all $i,j = 1,\dots,N$ and $\ell = 1,2$.

It is then well known that $\mathbf{K}_1$, $\mathbf{K}_2$ and their product $\mathbf{K}_1\mathbf{K}_2$ is compressible by means of $\mathcal{H}$-matrices, see [12, 22]. For our numerical experiments, we assemble the matrices by using piecewise constant finite elements and adaptive cross approximation as described in [26]. The computations have been carried out on a single core of a compute server with two Intel(R) Xeon(R) E5-2698v4 CPUs with a clock rate of 2.20GHz and a main memory of 756GB. The backend for the linear algebra routines is version 3.2.8 of the software library Eigen, see [20].

Figure 4 depicts the computation times per degree of freedom for the different kinds of $\mathcal{H}$-matrix multiplication for the fixed rank truncation, whereas Figure 5 shows the computations times for the ε-rank truncation. The cost of the fixed rank truncation seems to be $\mathcal{O}(N\log(N)^2)$, in accordance with the theoretical cost estimates. We can also immediately infer that it pays off to replace the hierarchical approximation by the alternative low-rank approximation schemes to improve computation times. For the ε-rank truncation, no cost estimates are known. While the asymptotic behaviour of the computation times for the traditional multiplication seems to be in the preasymptotic regime in this case, the new multiplication scales almost linearly. Another important point to remark is that the new algorithm with ε-rank truncation seems to outperform the frequently used traditional $\mathcal{H}$-matrix multiplication in terms of computational efficiency. Therefore, we shall now look whether it is also competitive in terms of accuracy.

To estimate the error of the $\mathcal{H}$-matrix multiplication, we apply ten subspace iterations to a subspace of size 100, using the matrix-vector product

$$(\mathbf{K}_1\mathbf{K}_2)\mathbf{v} - \mathbf{K}_1(\mathbf{K}_2\mathbf{v}),$$

and compute an approximation to the Frobenius norm. Looking at the corresponding errors in Figures 6 and 7, we see that the accuracies of the fixed rank arithmetic behave similar. However, the new multiplication reaches these accuracies in a shorter computation time. For the ε-rank truncation, we observe that the traditional multiplication cannot achieve the prescribed ε-rank of $\varepsilon = 10^{-12}$. This accuracy is only achieved by the new multiplication with an appropriate low-rank approximation method. The computation times for these accuracies are even smaller than the fixed-rank version of the traditional $\mathcal{H}$-matrix multiplication. Concerning the low-rank approximation methods, it seems as if ACA and the randomized algorithm are more robust when applied to smaller matrix sizes, although this difference vanishes for large matrices.

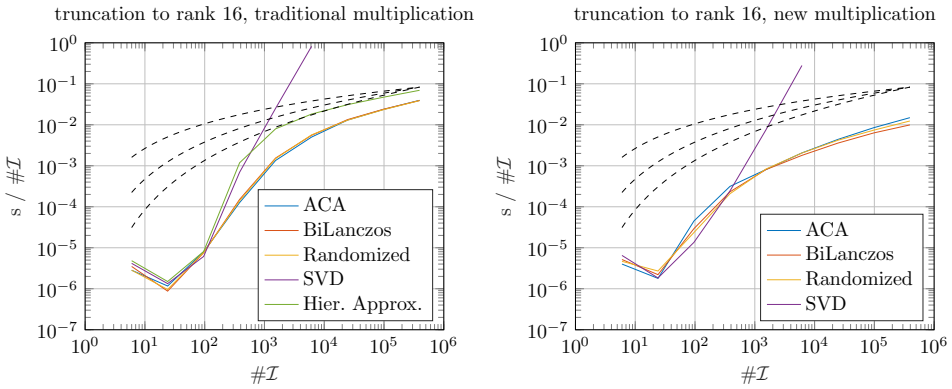


FIGURE 4. Computation times in seconds per degree of freedom for the product of the matrices occurring from the exponential kernels using fixed rank truncation with corresponding asymptotics $N \log^2 N$, $N \log^3 N$, and $N \log^4 N$.

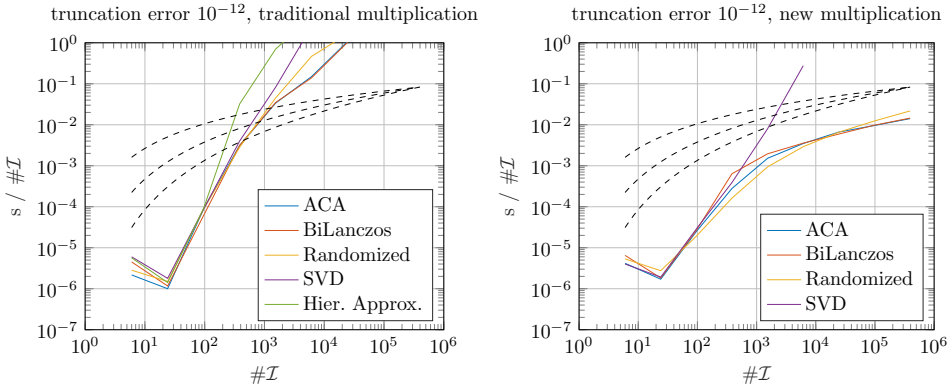


FIGURE 5. Computation times in seconds per degree of freedom for the product of the matrices occurring from the exponential kernels using ε-rank truncation with corresponding asymptotics $N \log^2 N$, $N \log^3 N$, and $N \log^4 N$.

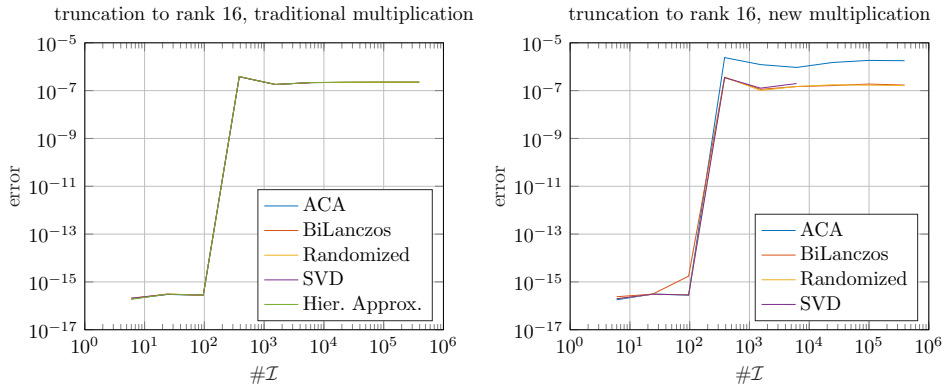


FIGURE 6. Error using fixed rank truncation for the product of the matrices occurring from the exponential kernels.

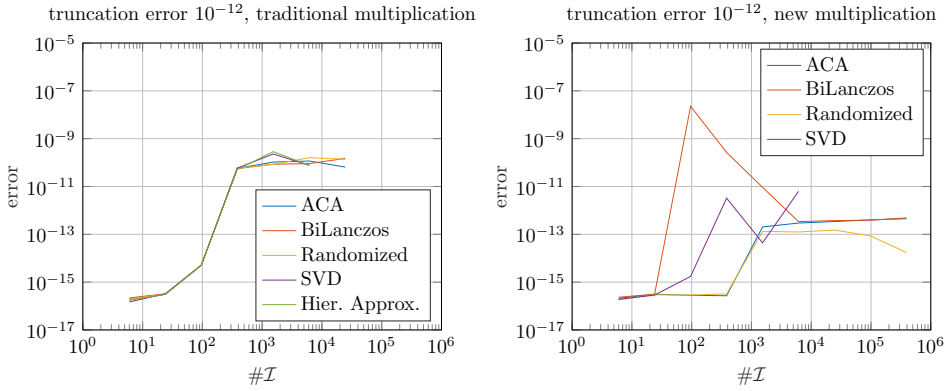


FIGURE 7. Error using ε-rank truncation for the product of the matrices occurring from the exponential kernels.

Since the approximation quality of the low-rank methods crucially depends on the decay of the singular values of the off-diagonal blocks, we repeat the experiments on a different example.

6.2. **Example with matrices from the boundary element method.** The second example is concerned with the discretized boundary integral operator

$$[\mathbf{V}]_{ij} = \int_\Gamma \int_\Gamma \frac{\varphi_j(\mathbf{x})\varphi_i(\mathbf{y})}{\|\mathbf{x}-\mathbf{y}\|} \, \mathrm{d}\sigma_{\mathbf{x}} \, \mathrm{d}\sigma_{\mathbf{y}},$$

which is frequently used in boundary element methods, see also [31]. The computation times for the operation $\mathbf{VV}$ can be found in Figures 8 and 9 and the corresponding accuracies can be found in Figures 10 and 11. We can see that the behaviour of the traditional and the new $\mathcal{H}$-matrix multiplication is in large parts the same as in the previous example, i.e., the new multiplication with ε-rank truncation can reach higher accuracies in shorter computation time than the traditional multiplication with an upper bound on the used rank. However, we shortly comment on the right figure of Figure 11. There, we see, as in the previous example, that the ACA and the randomized low-rank approximation method are more robust on small matrix sizes than the BiLanczos algorithm. We also see that the ACA and the randomized low-rank approximation method are even more robust than assembling the full matrix and applying a dense SVD to obtain the low-rank blocks.

7. Conclusion

The multiplication of hierarchical matrices is a widely used algorithm in engineering. The recursive scheme of the original implementation and the required upper threshold for the rank make an a-priori error analysis of the algorithm difficult. Although several approaches to reduce the number of error-introducing operations have been made in the literature, an algorithm providing a fast multiplication of $\mathcal{H}$-matrices with a-priori error-bounds is still not available nowadays. By introducing

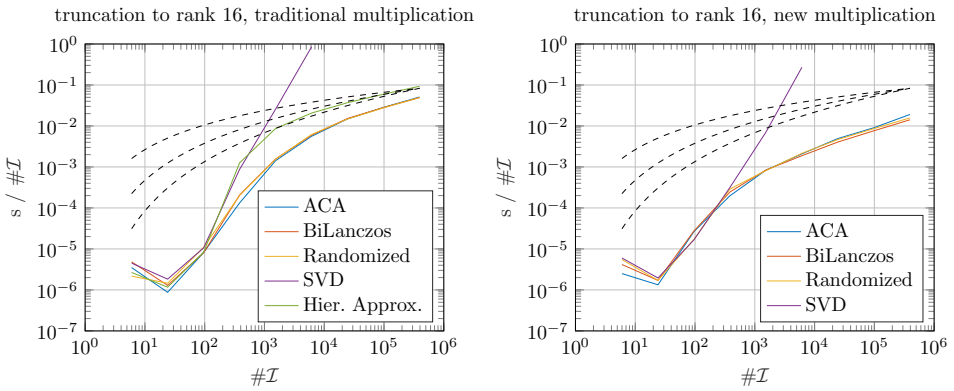


Figure 8. Computation times in seconds per degree of freedom for the product of the boundary integral operator matrices using fixed rank truncation with corresponding asymptotics $N \log^2 N$, $N \log^3 N$, and $N \log^4 N$.

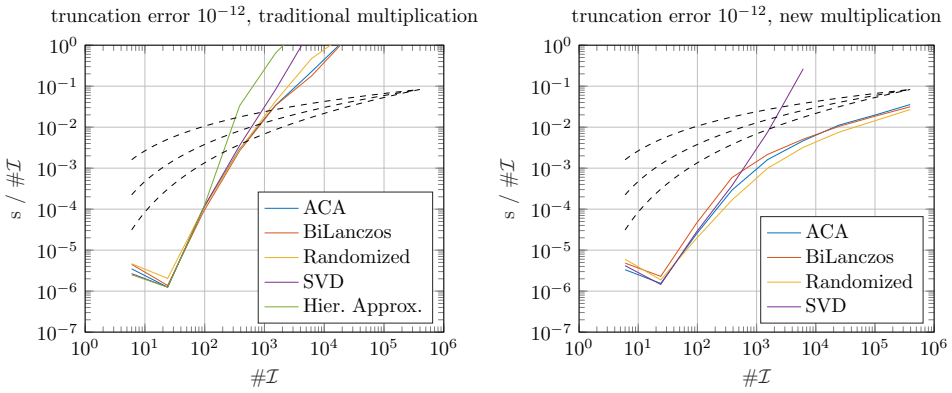


FIGURE 9. Computation times in seconds per degree of freedom for the product of the boundary integral operator matrices using ε-rank truncation with corresponding asymptotics $N \log^2 N$, $N \log^3 N$, and $N \log^4 N$.

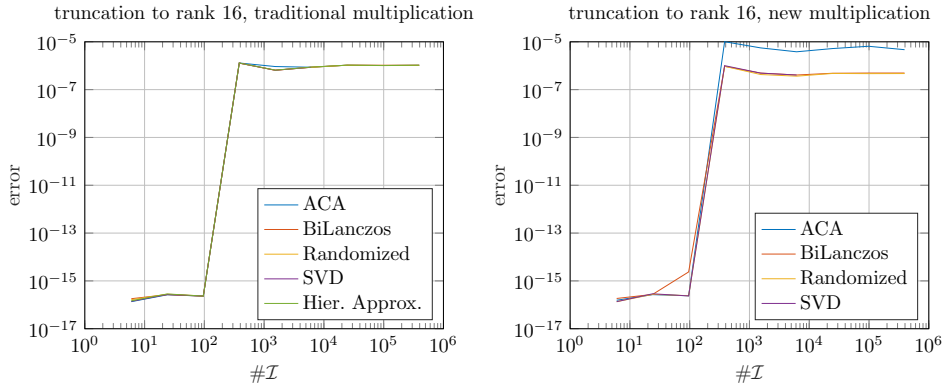


FIGURE 10. Error using fixed rank truncation for the product of the boundary integral operator matrices.

`sum`-expressions, which can be seen as a queuing system of low-rank matrices and $\mathcal{H}$-matrix products, we can postpone the error-introducing low-rank approximation until the last stage of the algorithm. We have discussed several adaptive low-rank approximation methods based on matrix-vector multiplications which make an a-priori error analysis of the $\mathcal{H}$-matrix multiplication possible. The cost analysis shows that the cost of our new $\mathcal{H}$-matrix multiplication algorithm is almost linear with respect to the size of the matrix. In particular, the numerical experiments show that the new approach can compute the best approximation of the $\mathcal{H}$-matrix product while being computationally more efficient than the traditional product.

Parallelization is an important topic on modern computer architectures. Therefore, we remark that the computation of the low-rank approximations for the target blocks is easily parallelizeable, once the necessary `sum`-expression is available. We

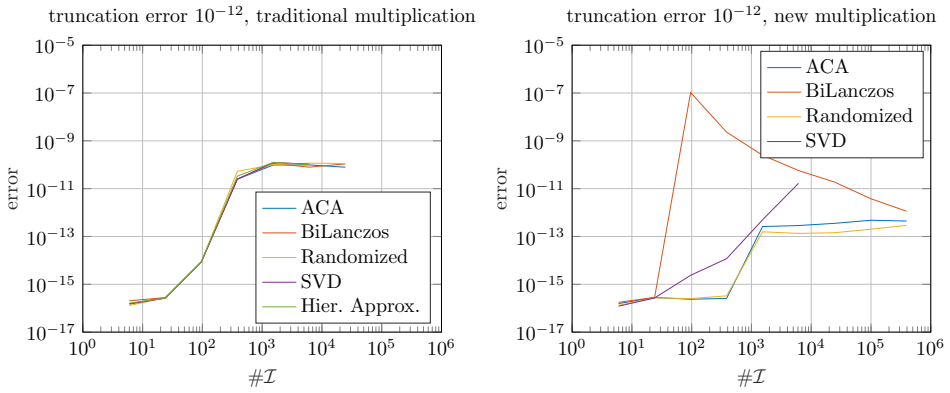


FIGURE 11. Error using ε-rank truncation for the product of the boundary integral operator matrices.

also remark that the computation of the `sum`-expressions does not require concurrent write access, such that the parallelization on a shared memory machine should be comparably easy.

ACKNOWLEDGEMENTS

The authors would like to thank Daniel Kressner for many fruitful discussions on iterative and randomized eigensolvers.

The work of Jürgen Dölz is supported by the Swiss National Science Foundation (SNSF) through the project 174987 and by the Excellence Initiative of the German Federal and State Governments and the Graduate School of Computational Engineering at TU Darmstadt.

REFERENCES

[1] S. Ambikasaran and E. Darve. An $\mathcal{O}(N \log N)$ fast direct solver for partial hierarchically semi-separable matrices: with application to radial basis function interpolation. *Journal of Scientific Computing*, 57(3):477–501, 2013.

[2] A. H. Aminfar, S. Ambikasaran, and E. Darve. A fast block low-rank dense solver with applications to finite-element matrices. *Journal of Computational Physics*, 304:170–188, 2016.

[3] U. Baur and P. Benner. Factorized solution of Lyapunov equations based on hierarchical matrix arithmetic. *Computing*, 78(3):211–234, 2006.

[4] M. Bebendorf. Approximation of boundary element matrices. *Numerische Mathematik*, 86(4):565–589, 2000.

[5] M. Bebendorf. *Hierarchical matrices. A means to efficiently solve elliptic boundary value problems*, volume 63 of *Lecture Notes in Computational Science and Engineering*. Springer, Berlin, 2008.

[6] M. Bebendorf and W. Hackbusch. Existence of $\mathcal{H}$-matrix approximants to the inverse FE-matrix of elliptic operators with L^∞-coefficients. *Numerische Mathematik*, 95(1):1–28, 2003.

[7] S. Börm. *Efficient numerical methods for non-local operators. $\mathcal{H}^2$-matrix compression, algorithms and analysis*, volume 14 of *EMS Tracts in Mathematics*. European Mathematical Society (EMS), Zürich, 2010.

[8] S. Börm. Hierarchical matrix arithmetic with accumulated updates. *Preprint*, arXiv:1703.09085, 2017.

[9] S. Chandrasekaran, M. Gu, and T. Pals. A fast ULV decomposition solver for hierarchically semiseparable representations. *SIAM Journal on Matrix Analysis and Applications*, 28(3):603–622, 2006.

[10] J. Dölz, H. Harbrecht, and M. Peters. $\mathcal{H}$-matrix accelerated second moment analysis for potentials with rough correlation. *Journal of Scientific Computing*, 65(1):387–410, 2015.
[11] J. Dölz, H. Harbrecht, and M. Peters. $\mathcal{H}$-matrix based second moment analysis for rough random fields and finite element discretizations. *SIAM Journal on Scientific Computing*, 39(4):B618–B639, 2017.
[12] J. Dölz, H. Harbrecht, and C. Schwab. Covariance regularity and $\mathcal{H}$-matrix approximation for rough random fields. *Numerische Mathematik*, 135(4):1045–1071, 2017.
[13] I. P. Gavrilyuk, W. Hackbusch, and B. N. Khoromskij. $\mathcal{H}$-matrix approximation for the operator exponential with applications. *Numerische Mathematik*, 92(1):83–111, 2002.
[14] G. H. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial and Applied Mathematics. Series B: Numerical Analysis*, 2(2):205–224, 1965.
[15] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, fourth edition, 2012.
[16] S. A. Goreinov, E. E. Tyrtyshnikov, and N. L. Zamarashkin. A theory of pseudoskeleton approximations. *Linear Algebra and its Applications*, 261(1–3):1–21, 1997.
[17] L. Grasedyck and W. Hackbusch. Construction and arithmetics of $\mathcal{H}$-matrices. *Computing*, 70(4):295–334, 2003.
[18] L. Grasedyck, W. Hackbusch, and B. N. Khoromskij. Solution of large scale algebraic matrix Riccati equations by use of hierarchical matrices. *Computing*, 70(2):121–165, 2003.
[19] L. Grasedyck, R. Kriemann, and S. Le Borne. Parallel black box $\mathcal{H}$-LU preconditioning for elliptic boundary value problems. *Computing and Visualization in Science*, 11(4–6):273–291, 2008.
[20] G. Guennebaud, B. Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.
[21] W. Hackbusch. A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: Introduction to $\mathcal{H}$-matrices. *Computing*, 62(2):89–108, 1999.
[22] W. Hackbusch. *Hierarchical Matrices: Algorithms and Analysis*. Springer, Heidelberg, 2015.
[23] W. Hackbusch, B. N. Khoromskij, and S. A. Sauter. On $\mathcal{H}^2$-matrices. In *Lectures on applied mathematics (Munich, 1999)*, pages 9–29. Springer, Berlin, 2000.
[24] W. Hackbusch and B.N. Khoromskij. A sparse $\mathcal{H}$-matrix arithmetic. II. Application to multi-dimensional problems. *Computing*, 64(1):21–47, 2000.
[25] N. Halko, P. G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, 2011.
[26] H. Harbrecht and M. Peters. Comparison of fast boundary element methods on parametric surfaces. *Computer Methods in Applied Mechanics and Engineering*, 261–262:39–55, 2013.
[27] R. Kriemann and S. Le Borne. $\mathcal{H}$-FAINV: hierarchically factored approximate inverse preconditioners. *Computing and Visualization in Science*, 17(3):135–150, 2015.
[28] P. G. Martinsson. A fast randomized algorithm for computing a hierarchically semiseparable representation of a matrix. *SIAM Journal on Matrix Analysis and Applications*, 32(4):1251–1274, 2011.
[29] P. G. Martinsson. Compressing rank-structured matrices via randomized sampling. *SIAM Journal on Scientific Computing*, 38(4):A1959–A1986, 2016.
[30] F.-H. Rouet, X. S. Li, P. Ghysels, and A. Napov. A distributed-memory package for dense hierarchically semi-separable matrix computations using randomization. *ACM Transactions on Mathematical Software*, 42(4), 2016.
[31] O. Steinbach. *Numerical Approximation Methods for Elliptic Boundary Value Problems*. Springer Science & Business, New York, 2008.
[32] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li. Fast algorithms for hierarchically semiseparable matrices. *Numerical Linear Algebra with Applications*, 17(6):953–976, 2010.

Jürgen Dölz, Technical University of Darmstadt, Graduate School CE, Dolivostrasse 15, 64293 Darmstadt, Germany
E-mail address: doelz@gsc.tu-darmstadt.de

Helmut Harbrecht, University of Basel, Department of Mathematics and Computer Science, Spiegelgasse 1, 4051 Basel, Switzerland
E-mail address: helmut.harbrecht@unibas.ch

Michael D. Multerer, born as Michael D. Peters, ETH Zurich, Department of Biosystems Science and Engineering, Mattenstrasse 26, 4058 Basel, Switzerland
E-mail address: michael.peters@bsse.ethz.ch