Max-Planck-Institut
für Mathematik
in den Naturwissenschaften
Leipzig

$\mathcal{H}^2$-matrix arithmetics in linear complexity

(revised version: September 2005)

by

Steffen Börm

Preprint no.: 47 2004

$\mathcal{H}^2$-matrix arithmetics in linear complexity

S. Börm, Leipzig

September 8, 2005

For hierarchical matrices, approximations of the matrix-matrix sum and product can be computed in almost linear complexity, and using these matrix operations it is possible to construct the matrix inverse, efficient preconditioners based on approximate factorizations or solutions of certain matrix equations.

$\mathcal{H}^2$-matrices are a variant of hierarchical matrices which allow us to perform certain operations, like the matrix-vector product, in "true" linear complexity, but until now it was not clear whether matrix arithmetic operations could also reach this, in some sense optimal, complexity.

We present algorithms that compute the best-approximation of the sum and product of two $\mathcal{H}^2$-matrices in a prescribed $\mathcal{H}^2$-matrix format, and we prove that these computations can be accomplished in linear complexity. Numerical experiments demonstrate that the new algorithms are more efficient than the well-known methods for hierarchical matrices.

MSC Subject Classification: 65F30
Keywords: Hierarchical matrices, formatted matrix operations.

1 Introduction

The discretization of partial differential or integral equations leads to large systems of linear equations, where the dimension increases with the desired accuracy. For partial differential equations, the resulting matrices are sparse, i.e., most of their entries are zero, and using this fact, very efficient techniques for storing and solving the corresponding systems have been developed. For integral equations, the situation is more complicated: since the underlying kernel functions are typically non-local, standard discretization techniques lead to densely populated matrices, which cannot be treated directly.

In order to handle dense matrices efficiently, they are approximated not by sparse, but by data-sparse representations: the panel-clustering technique [21] and the multipole expansion [22, 15, 16] replace the kernel function by separable approximations, wavelet methods [11, 25] use alternative discretization techniques in order to ensure that the resulting matrix can be approximated by a sparse matrix.

Hierarchical matrices [17, 19, 12, 6] are the algebraic counterpart of panel-clustering techniques: they work with low-rank approximations directly where the panel-clustering technique uses degenerate expansions. Due to this algebraic approach, approximate arithmetic operations on n-dimensional matrices can be performed in $\mathcal{O}(n \log^\alpha n)$ operations for $\alpha \geq 1$ [12, 14]. This is a major advantage as compared to other data-sparse representations, since the arithmetic operations can be used to construct preconditioners or even approximative solvers for partial differential [1] or integral equations [13, 5].

$\mathcal{H}^2$-matrices [20, 7] are a variant of hierarchical matrices which allow us to store certain (see [4] for a mathematically precise characterization) dense n-dimensional matrices in only $\mathcal{O}(n)$ units of storage and for which the matrix-vector product can be evaluated in $\mathcal{O}(n)$ operations. For the integral operators of classical potential theory, the construction of $\mathcal{H}^2$-matrix approximants is straightforward and leads to very efficient algorithms [8, 9, 3], and recent results [4] indicate that $\mathcal{H}^2$-matrices can also be used to represent the solution operators of elliptic partial differential equations.

Since each $\mathcal{H}^2$-matrix is also a hierarchical matrix, it is of course possible to apply the well-established algorithms for approximative matrix arithmetics, and it is even possible to convert the resulting hierarchical matrix back to an $\mathcal{H}^2$-matrix. Obviously, the complexity of algorithms derived in this way is at least as high as that of the underlying operations for hierarchical matrices, i.e., it will not be in $\mathcal{O}(n)$, but only in $\mathcal{O}(n \log^\alpha n)$.

This article presents algorithms which compute the best approximation of the sum and product of $\mathcal{H}^2$-matrices in a prescribed $\mathcal{H}^2$-matrix format in *linear* complexity, i.e., only $\mathcal{O}(n)$ operations are required for handling n-dimensional matrices. In the case of the sum of two $\mathcal{H}^2$-matrices, this is not a surprising result, and the corresponding algorithm is only included for the sake of completeness and as an introductory example. For the product, on the other hand, the existence of a linear-time algorithm is remarkable, since one would expect a higher complexity due to the fact that each of the $\mathcal{O}(n)$ blocks of the product depends on $\mathcal{O}(\log n)$ blocks in both factors. The key to the linear-time algorithm lies in the recursive structure of $\mathcal{H}^2$-matrices, which allows us to improve the efficiency by suitable pre- and postprocessing steps.

Another difference, as compared to hierarchical matrices, is that the new algorithms only require products and sums of small matrices, but no intermediate truncation operations based on singular value decompositions, so their basic structure is much simpler and their execution time much better. While the algorithms for hierarchical matrices are only efficient if intermediate approximation steps are used, the new algorithms do not require these steps and therefore compute the best approximation of the result in the prescribed format.

The matrix-matrix multiplication is a fundamental operation required in the computation of approximate inverses, approximate LU and Cholesky factorizations or when solving matrix equations, therefore it is to be hoped that the new algorithm can be applied to improve the efficiency of numerical methods in these areas.

The algorithms presented in this article compute the best approximations of sums and products of $\mathcal{H}^2$-matrices in a given block structure and for given expansion systems. In order to find error bounds, we therefore have to investigate the approximation properties of $\mathcal{H}^2$-matrix spaces. For matrices corresponding to integral operators, the necessary

bounds are well-known [8, 9, 5], but also the case of more general matrices has been investigated recently [4].

There are parallels to standard finite element theory: Cea's lemma bounds the error of the Galerkin approximation by the error of the best approximation in the given finite element space, then additional regularity assumptions have to be applied in order to control the quality of this approximation. For $\mathcal{H}^2$-matrices, the addition and multiplication algorithms compute the best approximation in the given $\mathcal{H}^2$-matrix space, and we need additional properties of the matrices in order to find error bounds for this best approximation.

Even if no a priori knowledge is available, it is possible to use algorithms which adapt the expansion systems a posteriori in order to guarantee a prescribed accuracy. This is still work in progress, but first experiments indicate that the new $\mathcal{H}^2$-matrix methods are as flexible and already more efficient than standard hierarchical matrices. These adaptive arithmetic algorithms will be addressed in a forthcoming paper.

2 $\mathcal{H}^2$-matrices

We will now briefly recall the structure of $\mathcal{H}^2$-matrices [20, 7].

2.1 Block structure

The basic idea is to split the matrix into a hierarchy of subblocks and to approximate those blocks that satisfy an *admissibility condition* in a special data-sparse format. In order to identify the admissible blocks efficiently, we introduce a hierarchy of subsets:

Definition 2.1 (Cluster tree) *Let $\mathcal{I}$ be an index set. Let $\mathcal{T}$ be a labeled tree. We denote its root by* $\operatorname{root}(\mathcal{T})$*, the label of $t \in \mathcal{T}$ by $\hat{t}$ and the set of sons by* $\operatorname{sons}(t, \mathcal{T})$ *(or just* $\operatorname{sons}(t)$ *if this does not lead to ambiguity).*

$\mathcal{T}$ is a cluster tree *for $\mathcal{I}$ if it satisfies the following conditions:*

- $\widehat{\operatorname{root}(\mathcal{T})} = \mathcal{I}$.
- *If* $\operatorname{sons}(t) \neq \emptyset$ *holds for $t \in \mathcal{T}$, we have*

$$s_1 \neq s_2 \Rightarrow \hat{s}_1 \cap \hat{s}_2 = \emptyset \text{ for all } s_1, s_2 \in \operatorname{sons}(t) \text{ and}$$
$$\hat{t} = \bigcup_{s \in \operatorname{sons}(t)} \hat{s}.$$

If $\mathcal{T}$ is a cluster tree for $\mathcal{I}$, we will denote it by $\mathcal{T}_\mathcal{I}$ and call its nodes clusters*. The set of* leaves *of $\mathcal{T}_\mathcal{I}$ is denoted by*

$$\mathcal{L}_\mathcal{I} := \{t \in \mathcal{T}_\mathcal{I} \ : \ \operatorname{sons}(t) = \emptyset\}.$$

We note that the definition implies that $\hat{t} \subseteq \mathcal{I}$ holds for all clusters in $\mathcal{T}_\mathcal{I}$ and that $\mathcal{L}_\mathcal{I}$ is a disjoint partition of $\mathcal{I}$.

Definition 2.2 (Block cluster tree) *Let $\mathcal{I}$, $\mathcal{J}$ be index sets, and let $\mathcal{T}_\mathcal{I}$ and $\mathcal{T}_\mathcal{J}$ be corresponding cluster trees. Let $\mathcal{T}$ be a labeled tree. $\mathcal{T}$ is a* block cluster tree *for $\mathcal{T}_\mathcal{I}$ and $\mathcal{T}_\mathcal{J}$ if it satisfies the following conditions:*

- $\operatorname{root}(\mathcal{T}) = (\operatorname{root}(\mathcal{T}_\mathcal{I}), \operatorname{root}(\mathcal{T}_\mathcal{J}))$.
- *Each cluster $b \in \mathcal{T}$ has the form $b = (t, s)$ for $t \in \mathcal{T}_\mathcal{I}$ and $s \in \mathcal{T}_\mathcal{J}$ and its label satisfies $\hat{b} = \hat{t} \times \hat{s}$.*
- *Let $b = (t, s) \in \mathcal{T}$. If $\operatorname{sons}(b) \neq \emptyset$, we have*

$$\operatorname{sons}(b) = \begin{cases} \{t\} \times \operatorname{sons}(s) & \text{if } \operatorname{sons}(t) = \emptyset, \operatorname{sons}(s) \neq \emptyset, \\ \operatorname{sons}(t) \times \{s\} & \text{if } \operatorname{sons}(t) \neq \emptyset, \operatorname{sons}(s) = \emptyset \\ \operatorname{sons}(t) \times \operatorname{sons}(s) & \text{otherwise.} \end{cases}$$

If $\mathcal{T}$ is a block cluster tree for $\mathcal{I}$ and $\mathcal{J}$, we will denote it by $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ and call its nodes blocks.

This definition implies that a block cluster tree for $\mathcal{I}$ and $\mathcal{J}$ is a cluster tree for the product index set $\mathcal{I} \times \mathcal{J}$.

Definition 2.3 (Admissibility) *Let $\mathcal{T}_\mathcal{I}$ and $\mathcal{T}_\mathcal{J}$ be cluster trees for $\mathcal{I}$ and $\mathcal{J}$. A predicate* $\operatorname{adm} : \mathcal{T}_\mathcal{I} \times \mathcal{T}_\mathcal{J} \to \mathbb{B} = \{\mathit{true}, \mathit{false}\}$ *is called* admissibility condition *for $\mathcal{T}_\mathcal{I}$ and $\mathcal{T}_\mathcal{J}$ if it satisfies*

$$\operatorname{adm}(t, s) \Rightarrow \operatorname{adm}(t', s) \quad \text{and} \quad \operatorname{adm}(t, s) \Rightarrow \operatorname{adm}(t, s')$$

for all $t' \in \operatorname{sons}(t)$ and all $s' \in \operatorname{sons}(s)$. This implies that all descendants of an admissible pair are also admissible.

A block cluster tree is called admissible *with respect to an admissibility condition* adm *if it satisfies*

$$b = (t, s) \in \mathcal{L}_{\mathcal{I}\times\mathcal{J}} \Rightarrow (\operatorname{adm}(t, s) \text{ or } \operatorname{sons}(t) = \emptyset = \operatorname{sons}(s)).$$

For an admissible block cluster tree, we split the set of leaves into

$$\mathcal{L}_{\mathcal{I}\times\mathcal{J},+} := \{b = (t, s) \in \mathcal{L}_{\mathcal{I}\times\mathcal{J}} \ : \ \operatorname{adm}(t, s) \text{ holds}\} \quad \text{and} \quad \mathcal{L}_{\mathcal{I}\times\mathcal{J},-} := \mathcal{L}_{\mathcal{I}\times\mathcal{J}} \setminus \mathcal{L}_{\mathcal{I}\times\mathcal{J},+},$$

i.e., into admissible and inadmissible leaves. Usually, we will not work with adm*, but only use $\mathcal{L}_{\mathcal{I}\times\mathcal{J},+}$ and $\mathcal{L}_{\mathcal{I}\times\mathcal{J},-}$.*

In practice, the admissibility condition is used to identify those blocks that can be approximated by low rank or (e.g., in the case of $\mathcal{H}^2$-matrices) more specialized formats.

Definition 2.4 (Sparsity) *Let $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ be a block cluster tree for $\mathcal{T}_\mathcal{I}$ and $\mathcal{T}_\mathcal{J}$. It is called* sparse *with* sparsity constant $C_{\mathrm{sp}} \in \mathbb{N}$ *if*

$$\#\{s' \in \mathcal{T}_\mathcal{J} \ : \ (t, s') \in \mathcal{T}_{\mathcal{I}\times\mathcal{J}}\} \leq C_{\mathrm{sp}} \quad \text{and} \quad \#\{t' \in \mathcal{T}_\mathcal{I} \ : \ (t', s) \in \mathcal{T}_{\mathcal{I}\times\mathcal{J}}\} \leq C_{\mathrm{sp}}$$

hold for all $t \in \mathcal{T}_\mathcal{I}$ and $s \in \mathcal{T}_\mathcal{J}$.

2.2 Factorized representation

The leaves of an admissible block cluster tree define a block partition of matrices $A \in \mathbb{R}^{\mathcal{I}\times\mathcal{J}}$. Typical hierarchical matrices are defined based on this partition: for all admissible blocks $b = (t, s) \in \mathcal{L}_{\mathcal{I}\times\mathcal{J},+}$, the corresponding matrix block $A|_{\hat{t}\times\hat{s}}$ is required to be of low rank and stored in an appropriate factorized form.

The $\mathcal{H}^2$-matrix format modifies this representation: we require not only that admissible blocks correspond to low-rank matrix blocks, but also that the range and image of these blocks are contained in predefined spaces.

In order to simplify the presentation, we introduce a restriction operator $\chi_t : \mathcal{I} \to \mathcal{I}$ for each $t \in \mathcal{T}_\mathcal{I}$ by

$$(\chi_t)_{ij} = \begin{cases} 1 & \text{if } i = j \in \hat{t}, \\ 0 & \text{otherwise.} \end{cases}$$

For $t \in \mathcal{T}_\mathcal{I}$, $s \in \mathcal{T}_\mathcal{J}$, the matrix $\chi_t A \chi_s \in \mathbb{R}^{\mathcal{I}\times\mathcal{J}}$ is equal to A in the sub-block $\hat{t} \times \hat{s}$ and zero everywhere else.

Definition 2.5 (Cluster basis) *Let $\mathcal{T}_\mathcal{I}$ be a cluster tree. A family $k = (k_t)_{t\in\mathcal{T}_\mathcal{I}}$ of integers is called* rank distribution. *For a given rank distribution k, a family $V = (V_t)_{t\in\mathcal{T}_\mathcal{I}}$ satisfying $V_t \in \mathbb{R}^{\mathcal{I}\times k_t}$ and $\chi_t V_t = V_t$ for all $t \in \mathcal{T}_\mathcal{I}$ is called* cluster basis *for $\mathcal{T}_\mathcal{I}$ with rank distribution k.*

We can see that this definition implies $(V_t)_{i\nu} = 0$ for all $t \in \mathcal{T}_\mathcal{I}$, $i \in \mathcal{I} \setminus \hat{t}$ and $\nu \in \{1, \ldots, k_t\}$, i.e., only matrix rows corresponding to indices in $\hat{t}$ can differ from zero.

The definition does not require the matrices V_t be of full rank, so their columns do not really form a basis, although the name "cluster basis" suggests this. This is only a practical consideration: in some applications, a system of vectors spanning the desired space can be constructed efficiently, but ensuring their linear independence would lead to unnecessary technical complications.

Definition 2.6 (Nested cluster bases) *Let $\mathcal{T}_\mathcal{I}$ be a cluster tree, and let V be a corresponding cluster basis with rank distribution k. Let $E = (E_t)_{t\in\mathcal{T}_\mathcal{I}}$ be a family of matrices satisfying $E_t \in \mathbb{R}^{k_t\times k_{t^+}}$ for each cluster $t \in \mathcal{T}_\mathcal{I}$ that has a father $t^+ \in \mathcal{T}_\mathcal{I}$. If the equation*

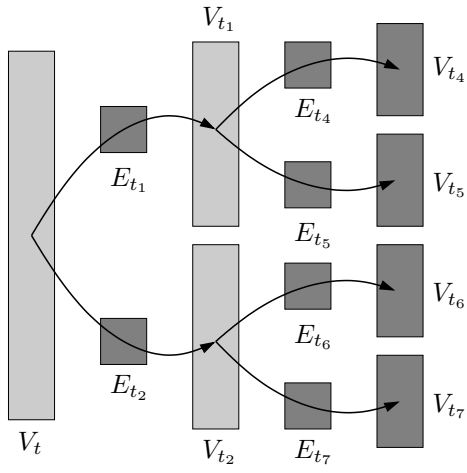
$$V_t = \sum_{t'\in\operatorname{sons}(t)} V_{t'} E_{t'} \tag{1}$$

holds for all $t \in \mathcal{T}_\mathcal{I}$ with $\operatorname{sons}(t) \neq \emptyset$, the cluster basis V is called nested *with* transfer matrices *E.*

The case $t = \operatorname{root}(\mathcal{T}_\mathcal{I})$ is only included in order to avoid the necessity of treating a special case: we can see that the definition does not require the transfer matrix for the root of $\mathcal{T}_\mathcal{I}$ to satisfy any conditions. In practice, this matrix can be ignored completely.

The nested structure is the key difference between general hierarchical matrices and $\mathcal{H}^2$-matrices [20, 7, 8], since it allows us to construct very efficient algorithms by re-using information across the entire cluster tree.

Figure 1: Representation of a nested cluster basis by transfer matrices



Definition 2.7 ($\mathcal{H}^2$-matrix) *Let $\mathcal{T}_\mathcal{I}$ and $\mathcal{T}_\mathcal{J}$ be cluster trees. Let $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ be an admissible block cluster tree. Let $A \in \mathbb{R}^{\mathcal{I}\times\mathcal{J}}$. Let V and W be nested cluster bases for $\mathcal{T}_\mathcal{I}$ and $\mathcal{T}_\mathcal{J}$ with rank distributions k and l. If we can find a matrix $S_b \in \mathbb{R}^{k_t\times l_s}$ for each $b = (t,s) \in \mathcal{L}_{\mathcal{I}\times\mathcal{J},+}$ satisfying*

$$\chi_t A \chi_s = V_t S_b (W_s)^\top, \tag{2}$$

the matrix A is called an $\mathcal{H}^2$-matrix with row cluster basis V *and* column cluster basis W*. The family $S = (S_b)_{b\in\mathcal{L}_{\mathcal{I}\times\mathcal{J},+}}$ is called the family of* coefficient matrices*.*

The set of all $\mathcal{H}^2$-matrices with row cluster basis V, column cluster basis W and block cluster tree $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ is denoted by $\mathcal{H}^2(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, V, W)$.

This definition implies that each $\mathcal{H}^2$-matrix can be written in the form

$$A = \sum_{b=(t,s)\in\mathcal{L}_{\mathcal{I}\times\mathcal{J},+}} V_t S_b (W_s)^\top + \sum_{b=(t,s)\in\mathcal{L}_{\mathcal{I}\times\mathcal{J},-}} \chi_t A \chi_s,$$

since $\mathcal{L}_{\mathcal{I}\times\mathcal{J}} = \mathcal{L}_{\mathcal{I}\times\mathcal{J},+} \dot\cup \mathcal{L}_{\mathcal{I}\times\mathcal{J},-}$ defines a partition of $\mathcal{I}\times\mathcal{J}$.

The major advantage of $\mathcal{H}^2$-matrices, as compared to standard hierarchical matrices, is that by replacing the low-rank representation $\chi_t A \chi_s = X_b Y_b^\top$ by the specialized representation (2), we can compute cluster-related quantities before performing complicated operations. Due to the nested structure, these quantities can be prepared efficiently (cf. Subsection 3.1 for an example).

Remark 2.8 (Rank distributions) *The rank k_t depends on the desired precision $\epsilon \in \mathbb{R}_{>0}$ of the matrix approximation, and the precision will usually depend on the underlying discretization error, e.g., $\epsilon \sim n^{-\beta}$ for some $\beta \in \mathbb{R}_{>0}$.*

For general integral operators and polynomial approximation, we can expect $k_t \sim \mathcal{O}(|\log\epsilon|^d)$. If multipole [16] or adaptive [3] approximations are applied, the rank is

reduced to $k_t \sim \mathcal{O}(|\log\epsilon|^{d-1})$*. For special problems and if* ϵ *is proportional to the discretization error, it is even possible to use a variable-rank approximation with* $k_t \sim \mathcal{O}(1)$ *for the majority of clusters [23, 24, 10, 9].*

For elliptic partial differential operator, the theory [1] suggests $k_t \sim \mathcal{O}(|\log\epsilon|^{d+1})$*, but practical experiments indicate that the optimal rank behaves like* $k_t \sim \mathcal{O}(|\log\epsilon|^{d-1})$*.*

The construction of $\mathcal{H}^2$-matrix approximations for discretized integral operators is relatively simple and therefore well-suited as a model problem:

Example 2.9 (Integral operators) *Let us consider the approximation of the matrix*

$$K_{ij} = \int_{\Gamma_i} \int_{\Gamma_j} g(x,y)\,dy\,dx \qquad 1 \leq i,j \leq n \tag{3}$$

resulting from a Galerkin discretization of an integral operator on a subdomain or submanifold $\Gamma \subseteq \mathbb{R}^d$ *with a kernel function* $g : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$*, using piecewise constant basis functions on a partition* $(\Gamma_i)_{i=1}^n$ *of* Γ*.*

We assume that g *is* asymptotically smooth*, i.e., that we can find constants* $c_0 \in \mathbb{R}_{>0}$ *and* $\sigma \in \mathbb{R}_{\geq 0}$ *satisfying*

$$|\partial_x^\nu \partial_y^\mu g(x,y)| \lesssim c_0^{|\nu+\mu|}(\nu+\mu)!\|x-y\|^{-\sigma-|\nu+\mu|} \tag{4}$$

for all $x, y \in \mathbb{R}^d$ *with* $x \neq y$ *and all* $\nu, \mu \in \mathbb{N}_0^d$*.*

We fix axis-parallel boxes B_t *for all* $t \in \mathcal{T}_\mathcal{I}$ *which satisfy* $\Gamma_i \subseteq B_t$ *for all* $i \in \hat{t}$ *and use*

$$\operatorname{diam}(B_t \times B_s) \leq \operatorname{dist}(B_t, B_s) \tag{5}$$

as admissibility condition.

It can be proven [8, 9] that approximating the kernel function g *locally for admissible pairs* (t,s) *of clusters by its* m*-th order Chebyshev interpolant*

$$\tilde{g}_{t,s}(x,y) = \sum_\nu \sum_\mu g(x_{t,\nu}, y_{s,\mu}) v_{t,\nu}(x) w_{s,\mu}(y)$$

(with interpolation points $(x_{t,\nu})_\nu$ *and* $(y_{s,\mu})_\mu$ *and corresponding Lagrange polynomials* $(v_{t,\nu})_\nu$ *and* $(w_{s,\mu})_\mu$*) in the domain* $B_t \times B_s$ *leads to an* $\mathcal{H}^2$*-matrix approximation* $\tilde{K}$ *which converges exponentially to* K*. A similar result can be proven for more general kernel functions and discretization schemes [5].*

2.3 Orthogonal cluster bases and best approximations

Since we intend to approximate results of arithmetic operations, we need an efficient way of finding best approximations of arbitrary matrices in a given $\mathcal{H}^2$-matrix format. This problem is especially simple if the columns of the cluster basis matrices V_t are pairwise orthonormal.

Definition 2.10 (Orthogonal cluster basis) *Let V be a cluster basis for the cluster tree $\mathcal{T}_\mathcal{I}$. It is called* orthogonal *if $V_t^\top V_t = I$ holds for all $t \in \mathcal{T}_\mathcal{I}$.*

The orthogonality implies that $V_t V_t^\top$ is an orthogonal projection onto the image of V_t, since

$$\langle V_t V_t^\top x, V_t y\rangle = \langle V_t^\top V_t V_t^\top x, y\rangle = \langle V_t^\top x, y\rangle = \langle x, V_t y\rangle$$

holds for all $x \in \mathbb{R}^\mathcal{I}$ and $y \in \mathbb{R}^{k_t}$. Therefore $V_t V_t^\top A W_s W_s^\top$ is the best approximation of a matrix block $\chi_t A \chi_s$ in the bases V_t and W_s, and

$$\tilde{A} := \sum_{b \in \mathcal{L}_{\mathcal{I}\times\mathcal{J},+}} V_t (V_t^\top A W_s) W_s^\top + \sum_{b \in \mathcal{L}_{\mathcal{I}\times\mathcal{J},-}} \chi_t A \chi_s$$

is the best approximation (in the Frobenius norm) of an arbitrary matrix $A \in \mathbb{R}^{\mathcal{I}\times\mathcal{J}}$ in the $\mathcal{H}^2$-matrix format defined by $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$, V and W.

If a non-nested cluster basis is given, an orthogonal counterpart can be constructed by simple Gram-Schmidt orthonormalization. If a nested cluster basis is given, it is possible to construct a nested orthogonal cluster basis by a modified orthonormalization algorithm in linear complexity [3]. The orthogonalization does not lead to a deterioration of the approximation properties.

3 Matrix-vector multiplication and matrix addition

The basic example for efficient algorithms for $\mathcal{H}^2$-matrices is the matrix-vector multiplication. We will only briefly recall the algorithm in order to pave the ground for the more involved techniques required for matrix arithmetics.

3.1 Matrix-vector multiplication

Let A be an $\mathcal{H}^2$-matrix with cluster bases V and W for the cluster trees $\mathcal{T}_\mathcal{I}$ and $\mathcal{T}_\mathcal{J}$ and the block cluster tree $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$. Let E and F be the families of transfer matrices for V and W.

The matrix-vector multiplication $y := Ax$ is split into four phases: First, we compute the auxiliary vectors

$$x_s := (W_s)^\top x$$

for all $s \in \mathcal{T}_\mathcal{J}$. This step is called the *forward transformation*. Then, we compute the auxiliary vectors

$$y_t := \sum_{b=(t,s)\in\mathcal{L}_{\mathcal{I}\times\mathcal{J},+}} S_b x_s$$

for all $t \in \mathcal{T}_\mathcal{I}$. This phase handles the interaction of all admissible blocks. In the third step, the *backward transformation*, we accumulate the part of the result

$$y := \sum_{t\in\mathcal{T}_\mathcal{I}} V_t y_t$$

that corresponds to all admissible blocks of the matrix. In order to complete the multiplication, we add the non-admissible parts

$$y := y + \sum_{b=(t,s)\in\mathcal{L}_{\mathcal{I}\times\mathcal{J},-}} \chi_t A \chi_s x.$$

Under standard assumptions, the second and last step can be performed in linear complexity, since they only involve relatively small matrices. In order to treat the forward transformation efficiently, we have to make use of the nested structure: $\operatorname{sons}(s) \neq \emptyset$ implies

$$x_s = W_s^\top x = \sum_{s'\in\operatorname{sons}(s)} F_{s'}^\top W_{s'}^\top x = \sum_{s'\in\operatorname{sons}(s)} F_{s'}^\top x_{s'},$$

so we can compute $x_s = W_s^\top x$ using $2k_s(\sum_{s'\in\operatorname{sons}(s)} k_{s'})$ operations instead of the $2k_s\#\hat{s}$ operations required by a naive approach, and we need to store W_s only for leaves of the cluster tree and can use the transfer matrices F for all other clusters. This leads to linear complexity for the forward transformation and for the storage requirements. The backward transformation can be treated in a similar way in order to reach linear complexity for the entire algorithm.

3.2 Split and collect operations

For the matrix-vector multiplication and similar computations, it is sufficient to exchange coefficient vectors like $\hat{x}_s$ and $\hat{y}_t$ between father and son clusters. For computations involving entire matrices, we need a counterpart for the interaction of father and son blocks.

Let V and W be nested cluster bases with transfer matrices E and F.

Algorithm 1 Split operation: Expresses a coefficient matrix S corresponding to (t, s) by a coefficient matrix S' corresponding to (t', s')

procedure Split((t, s), (t', s'), E, F, S, **var** S')
if $t' = t$ **then**
  if $s' = s$ **then**
    $S' := S$ { $t' = t$ and $s' = s$ }
  else
    $S' := SF_{s'}^\top$ { $t' = t$ and $s' \in \operatorname{sons}(s)$ }
  end if
else
  if $s' = s$ **then**
    $S' := E_{t'}S$ { $t' \in \operatorname{sons}(t)$ and $s' = s$ }
  else
    $S' := E_{t'}SF_{s'}^\top$ { $t' \in \operatorname{sons}(t)$ and $s' \in \operatorname{sons}(s)$ }
  end if
end if

The transfer from father to son is simple, since we can apply equation (1) to row and column clusters in order to get an exact representation: Let $b = (t, s) \in \mathcal{T}_\mathcal{I} \times \mathcal{T}_\mathcal{J}$, and let $S_b \in \mathbb{R}^{k_t \times l_s}$. If $\operatorname{sons}(t) \neq \emptyset$, we have

$$V_t S_b W_s^\top = \sum_{t' \in \operatorname{sons}(t)} V_{t'} E_{t'} S_b W_s^\top = \sum_{t' \in \operatorname{sons}(t)} V_{t'} S_{(t',s)} W_s^\top \tag{6}$$

for $S_{(t',s)} := E_{t'} S_b$. If $\operatorname{sons}(s) \neq \emptyset$, we can apply the same idea to get

$$V_t S_b W_s^\top = \sum_{s' \in \operatorname{sons}(s)} V_t S_b F_{s'}^\top W_{s'}^\top = \sum_{s' \in \operatorname{sons}(s)} V_t S_{(t,s')} W_{s'}^\top \tag{7}$$

for $S_{(t,s')} := S_b F_{s'}^\top$. In order to avoid special cases, we define

$$\operatorname{sons}^+(t) := \begin{cases} \operatorname{sons}(t) & \text{if } \operatorname{sons}(t) \neq \emptyset \\ \{t\} & \text{otherwise} \end{cases}$$

and introduce the Algorithm 1 that converts a matrix block $V_t S_b W_s^\top$ into the form $V_{t'} S_{b'} W_{s'}^\top$ for all $t' \in \operatorname{sons}^+(t)$ and $s' \in \operatorname{sons}^+(s)$.

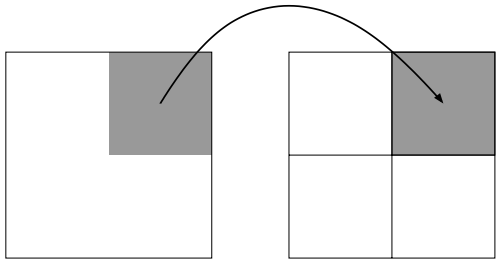
Algorithm 2 Collect operation: Approximates a coefficient matrix S' corresponding to (t', s') by a matrix S corresponding to (t, s)

```
procedure Collect((t', s'), (t, s), E, F, S', var S)
if t' = t then
  if s' = s then
    S := S + S'                                  { t' = t and s' = s }
  else
    S := S + S' F_{s'}                           { t' = t and s' ∈ sons(s) }
  end if
else
  if s' = s then
    S := S + E_{t'}^T S'                         { t' ∈ sons(t) and s' = s }
  else
    S := S + E_{t'}^T S' F_{s'}                  { t' ∈ sons(t) and s' ∈ sons(s) }
  end if
end if
```

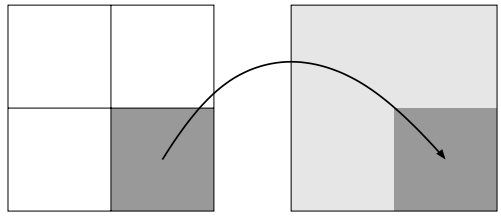
The counterpart, the transfer from son to father, is slightly more complicated from a theoretical point of view, since we can not expect to be able to express the cluster bases of the sons in terms of those of the father.

If the cluster bases are orthogonal, we can at least get the best approximation of the son blocks in the cluster bases corresponding to the father: If $t' \in \operatorname{sons}(t)$ and if $S_{(t',s)}$ is given, (1) implies that the best approximation in the cluster basis V_t and W_s can be

Figure 2: Split and collect operations



The split operation computes an exact representation of a subblock of an admissible block.



The collect operation computes only an *approximation* of a subblock in an admissible block.

expressed in the form

$$V_t V_t^\top (V_{t'} S_{(t',s)} W_s^\top) = V_t \left(\sum_{t'' \in \operatorname{sons}(t)} E_{t''}^\top V_{t''}^\top V_{t'} \right) S_{(t',s)} W_s^\top .$$

Since the sons of t correspond to disjoint subsets of $\hat{t}$, only the case $t'' = t'$ leads to a non-zero contribution to the sum, and we find

$$V_t V_t^\top (V_{t'} S_{(t',s)} W_s^\top) = V_t E_{t'}^\top S_{(t',s)} W_s^\top = V_t S_b W_s^\top$$

with $S_b := E_{t'}^\top S_{(t',s)}$. We can apply similar arguments to the column cluster basis and introduce the Algorithm 2 that approximates a matrix block (t', s') by a matrix block (t, s) for all $t' \in \operatorname{sons}^+(t)$ and all $s' \in \operatorname{sons}^+(s)$.

3.3 Matrix addition

We will now investigate the addition of two $\mathcal{H}^2$-matrices. This operation is actually quite simple, since the $\mathcal{H}^2$-matrices with a given block cluster tree and given cluster bases form a subspace and are therefore closed under addition. In order to handle general block cluster trees and non-matching cluster bases, we have to apply the correct projections.

Let $A \in \mathcal{H}^2(\mathcal{T}^A_{\mathcal{I}\times\mathcal{J}}, V^A, W^A)$ and $B \in \mathcal{H}^2(\mathcal{T}^B_{\mathcal{I}\times\mathcal{J}}, V^B, W^B)$. We denote the transfer matrices for V^A and V^B by E^A and E^B and those for W^A and W^B by F^A and F^B. We do *not* require the block cluster trees $\mathcal{T}^A_{\mathcal{I}\times\mathcal{J}}$ and $\mathcal{T}^B_{\mathcal{I}\times\mathcal{J}}$ to match, they only have to be formed corresponding to the same row and column cluster trees $\mathcal{T}_{\mathcal{I}}$ and $\mathcal{T}_{\mathcal{J}}$.

In order to keep the presentation simple, we assume that V^B and W^B are orthogonal. The algorithm can be modified to work without this assumption, but in most applications an orthonormalization of the involved cluster bases will be preferable.

Our goal is to approximate the sum $A + B$ in a prescribed space $\mathcal{M}$ of $\mathcal{H}^2$-matrices. Its best approximation in the Frobenius norm can be expressed as $\Pi(A+B)$, where Π is

Algorithm 3 Efficient computation of cluster basis products

procedure ClusterBasisProduct(t, V^B, V^A, **var** X)
if $\operatorname{sons}(t) = \emptyset$ **then**
  $X_t := (V_t^B)^\top V_t^A$
else
  $X_t := 0$;
  for $t' \in \operatorname{sons}(t)$ **do**
    ClusterBasisProduct(t', V^B, V^A, X);
    $X_t := X_t + (E_{t'}^B)^\top X_{t'} E_{t'}^A$
  end for
end if

the orthogonal projection from $\mathbb{R}^{\mathcal{I}\times\mathcal{J}}$ into the space $\mathcal{M}$, since the orthogonality implies

$$\|(A+B) - \Pi(A+B)\|_F = \inf_{M\in\mathcal{M}} \|(A+B) - M\|_F. \tag{8}$$

Since Π is a linear operator, we have $\Pi(A+B) = \Pi(A)+\Pi(B)$, so the projected addition is an associative and commutative operation, just like the standard addition.

In order to keep the presentation simple, we will only consider the situation where Π is the orthogonal projection into $\mathcal{M} := \mathcal{H}^2(\mathcal{T}^B_{\mathcal{I}\times\mathcal{J}}, V^B, W^B)$ and we will only investigate the computation of $B^{\text{new}} := \Pi(A) + B = \Pi(A+B)$, where B is overwritten with B^{new}. Due to linearity, a more general addition $D := \Pi(A+B+C)$, where Π is the projection into a general $\mathcal{H}^2$-matrix space, can be split into the steps $D^{(1)} := 0$, $D^{(2)} := \Pi(A) + D^{(1)}$, $D^{(3)} := \Pi(B) + D^{(2)}$ and $D := \Pi(C) + D^{(3)}$.

Algorithm 4 Matrix forward transformation: Computes $\tilde{S}^A_{t,s} := (V_t^B)^\top A W_s^B$ recursively

procedure MatrixForward(b, V^B, W^B, X, Y, A, **var** $\tilde{S}^A$) $\quad \{b = (t,s) \in \mathcal{T}^A_{\mathcal{I}\times\mathcal{J}}\}$
if $b \in \mathcal{L}^A_{\mathcal{I}\times\mathcal{J},+}$ **then**
  $\tilde{S}_b^A := X_t S_b^A Y_s$ $\quad \{X_t$ and Y_s computed by Algorithm 3$\}$
else if $b \in \mathcal{L}^A_{\mathcal{I}\times\mathcal{J},-}$ **then**
  $\tilde{S}_b^A := (V_t^B)^\top A W_s^B$
else
  $\tilde{S}_b^A := 0$;
  for $b' \in \operatorname{sons}(b)$ **do**
    MatrixForward(b', V^B, W^B, X, Y, A, $\tilde{S}^A$);
    Collect(b', b, E^B, F^B, $\tilde{S}^A_{b'}$, $\tilde{S}^A_b$)
  end for
end if

During the course of the computation, we can arrive at a block $b = (t,s) \in \mathcal{T}^A_{\mathcal{I}\times\mathcal{J}}$ that is an admissible leaf in $\mathcal{T}^B_{\mathcal{I}\times\mathcal{J}}$, but not in $\mathcal{T}^A_{\mathcal{I}\times\mathcal{J}}$. In this case, the best approximation of $\chi_t(A+B)\chi_s$ in the given space is given by computing the coupling matrix $\tilde{S}_b^A := (V_t^B)^\top A W_s^B$ corresponding to the best approximation of $\chi_t A \chi_s$ [2, Section 5] and adding

it to S_b^B. Similar to the forward transformation used in the matrix-vector multiplication, the auxiliary matrices $\tilde{S}_b^A$ can be computed efficiently by using the fact that V^B and W^B are nested: if $\operatorname{sons}(b) \neq \emptyset$, we can first compute $\tilde{S}_{b'}^A$ for all $b' \in \operatorname{sons}(b)$ and then use Algorithm 2 to compute $\tilde{S}_b^A$ itself.

If $\operatorname{sons}(b) = \emptyset$, the block can be admissible or non-admissible. The non-admissible case can be treated directly, but the admissible case is slightly more complicated: the matrix block is given by $\chi_t A \chi_s = V_t^A S_b^A (W_s^A)^\top$, and we have to compute

$$\tilde{S}_b^A = (V_t^B)^\top A W_s^B = (V_t^B)^\top V_t^A S_b^A (W_s^A)^\top W_s^B.$$

Since the matrices V_t^A, V_t^B have $\#\hat{t}$ rows and the matrices W_s^A, W_s^B have $\#\hat{s}$ rows, computing this product directly will lead to an inacceptable increase in the complexity. Fortunately, we can once more use the nested structure: if $\operatorname{sons}(t) \neq \emptyset$, we have

$$X_t := (V_t^B)^\top V_t^A = \sum_{t' \in \operatorname{sons}(t)} (E_{t'}^B)^\top (V_{t'}^B)^\top V_{t'}^A E_{t'}^A = \sum_{t' \in \operatorname{sons}(t)} (E_{t'}^B)^\top X_{t'} E_{t'}^A \tag{9}$$

and can compute the product matrices X_t by the efficient recursive procedure given in Algorithm 3. The same holds for $Y_s := (W_s^A)^\top W_s^B$.

Lemma 3.1 (Complexity of cluster basis products) *Let $c \in \mathbb{N}$ such that $\#\mathcal{T}_\mathcal{I} \leq c$. Let $\hat{k} \in \mathbb{N}$ such that $k_t^A, k_t^B \leq \hat{k}$ holds for all $t \in \mathcal{T}_\mathcal{I}$ and that $\#\hat{t} \leq \hat{k}$ holds for all $t \in \mathcal{L}_\mathcal{I}$. Then Algorithm 3 requires $\mathcal{O}(\hat{k}^3 c)$ operations.*

Proof. Let $t \in \mathcal{T}_\mathcal{I}$. If $\operatorname{sons}(t) = \emptyset$, we compute $X_t := (V_t^B)^\top V_t^A$ directly, and this requires $\mathcal{O}(k_t^A k_t^B \#\hat{t}) \subseteq \mathcal{O}(\hat{k}^3)$ operations. If t has a father t^+, the computation of X_{t^+} will involve adding $(E_t^B)^\top X_t E_t^A$ to X_{t^+}, and this requires $\mathcal{O}(k_t^B k_{t^+}^B k_t^A + k_{t^+}^B k_{t^+}^A k_t^A) \subseteq \mathcal{O}(\hat{k}^3)$ operations. Adding these estimates for all clusters yields the desired result. ∎

By preparing $X = (X_t)_{t \in \mathcal{T}_\mathcal{I}}$ and $Y = (Y_s)_{s \in \mathcal{T}_\mathcal{J}}$ in advance, we can compute $\tilde{S}_b^A$ efficiently even in the case $b \in \mathcal{L}_{\mathcal{I}\times\mathcal{J},+}$.

Combining Algorithm 2 with Algorithm 3 leads to Algorithm 4, the *matrix forward transformation*.

We may also find a block $b = (t, s) \in \mathcal{T}_{\mathcal{I}\times\mathcal{J}}^B$ that is an admissible leaf in $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}^A$, but not in $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}^B$. In order to compute the best approximation, we have to split the block recursively until it matches the structure of B, then transform the cluster bases, and finally add the result to the correct coefficient matrices of B. Instead of doing this instantaneously, we store S_b^A in a temporary matrix $\tilde{S}_b^B$ and handle it separately: this time, we use the nested structure of the cluster bases V^A and W^A to construct the counterpart of Algorithm 4, the recursive *matrix backward transformation*, Algorithm 5.

Lemma 3.2 (Complexity of transformations) *Let $c, b \in \mathbb{N}$ such that $\#\mathcal{T}_\mathcal{I}, \#\mathcal{T}_\mathcal{J} \leq c$ and $\#\mathcal{T}_{\mathcal{I}\times\mathcal{J}}^A, \#\mathcal{T}_{\mathcal{I}\times\mathcal{J}}^B \leq b$ holds. Let $\hat{k} \in \mathbb{N}$ such that $k_t^A, k_t^B, l_s^A, l_s^B \leq \hat{k}$ holds for all $t \in \mathcal{T}_\mathcal{I}, s \in \mathcal{T}_\mathcal{J}$ and that $\#\hat{t}, \#\hat{s} \leq \hat{k}$ holds for all $t \in \mathcal{L}_\mathcal{I}, s \in \mathcal{L}_\mathcal{I}$.*

Then the matrix forward and backward transformations require $\mathcal{O}(\hat{k}^3(b+c))$ operations.

Algorithm 5 Matrix backward transformation: Updates $B := B + V_t^A \tilde{S}_{t,s}^B (W_s^A)^\top$

procedure MatrixBackward(b, V^A, W^A, X, Y, **var** B, $\tilde{S}^B$) $\quad \{b = (t,s) \in \mathcal{T}^B_{\mathcal{I}\times\mathcal{J}}\}$
if $b \in \mathcal{L}^B_{\mathcal{I}\times\mathcal{J},+}$ **then**
  $S_b^B := S_b^B + X_t \tilde{S}_b^B Y_s$ $\quad \{X_t$ and Y_s computed by Algorithm 3$\}$
else if $b \in \mathcal{L}^B_{\mathcal{I}\times\mathcal{J},-}$ **then**
  $B := B + V_t^A \tilde{S}_b^B (W_s^A)^\top$
else
  for $b' \in \operatorname{sons}(b)$ **do**
    Split(b, b', E^A, F^A, S_b^B, $S_{b'}^B$);
    MatrixBackward(b', V^A, W^A, X, Y, B, $\tilde{S}^B$)
  end for
end if

Proof. We will only consider the forward transformation, the complexity estimate for the backward transformation follows by symmetry.

Due to Lemma 3.1, the matrices $X = (X_t)_{t\in\mathcal{T}_\mathcal{I}}$ and $Y = (Y_s)_{s\in\mathcal{T}_\mathcal{J}}$ can be computed in $\mathcal{O}(\hat{k}^3 c)$ operations, so we only have to consider the complexity of Algorithm 4.

Let $b \in \mathcal{T}_{\mathcal{I}\times\mathcal{J}}$. If $b \in \mathcal{L}_{\mathcal{I}\times\mathcal{J},+}$, we compute $X_t S_b^A Y_s$ in $\mathcal{O}(\hat{k}^3)$ operations. If $b \in \mathcal{L}_{\mathcal{I}\times\mathcal{J},-}$, we compute $(V_t^B)^\top A W_s^B$, which can also be accomplished in $\mathcal{O}(\hat{k}^3)$ operations, since $\operatorname{sons}(t) = \emptyset = \operatorname{sons}(s)$ holds by definition, implying $\#\hat{t}, \#\hat{s} \leq \hat{k}$. If b is not a leaf of $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$, we use Algorithm 2, which once more requires only $\mathcal{O}(\hat{k}^3)$ operations. Summing up over all blocks gives us a total complexity of $\mathcal{O}(b\hat{k}^3)$. Combining this with the estimate for the preparation of X and Y, we get the desired upper bound. ∎

Let us now turn to the matrix addition itself. Since the block cluster tree is defined recursively, it is straightforward to look for a recursive algorithm. Let $b = (t,s) \in \mathcal{T}^A_{\mathcal{I}\times\mathcal{J}} \cap \mathcal{T}^B_{\mathcal{I}\times\mathcal{J}}$.

If $b \in \mathcal{L}^A_{\mathcal{I}\times\mathcal{J},+}$, we can add the corresponding coefficient matrix S_b^A to the auxiliary matrix $\tilde{S}_b^B$ and handle it by the matrix backward transformation.

If $b \in \mathcal{L}^A_{\mathcal{I}\times\mathcal{J},-}$, the definition of the block cluster tree implies that $\operatorname{sons}(t) = \emptyset = \operatorname{sons}(s)$ holds and we can infer $b \in \mathcal{L}^B_{\mathcal{I}\times\mathcal{J}}$. If $b \in \mathcal{L}^B_{\mathcal{I}\times\mathcal{J},-}$, we simply add dense matrices, otherwise we have $b \in \mathcal{L}^B_{\mathcal{I}\times\mathcal{J},+}$ and use the matrix $\tilde{S}_b^A$ prepared by the matrix forward transformation.

If $b \notin \mathcal{L}^A_{\mathcal{I}\times\mathcal{J}}$, there are only two alternatives: If $b \in \mathcal{L}^B_{\mathcal{I}\times\mathcal{J}}$, this implies $b \in \mathcal{L}^B_{\mathcal{I}\times\mathcal{J},+}$ by construction, so we again use the matrix $\tilde{S}_b^A$ prepared by the matrix forward transformation. If $b \notin \mathcal{L}^B_{\mathcal{I}\times\mathcal{J}}$, we can use the definition of the block cluster tree in order to find $\operatorname{sons}(b, \mathcal{T}^A_{\mathcal{I}\times\mathcal{J}}) = \operatorname{sons}(b, \mathcal{T}^B_{\mathcal{I}\times\mathcal{J}})$ and proceed by recursion.

Since V_B and W_B are orthogonal cluster bases, our algorithm will compute the desired best approximation of the sum $A + B$.

Theorem 3.3 ($\mathcal{H}^2$-matrix addition) *Let $c, b \in \mathbb{N}$ such that $\#\mathcal{T}_\mathcal{I}, \#\mathcal{T}_\mathcal{J} \leq c$ and let $\#\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, \#\mathcal{T}'_{\mathcal{I}\times\mathcal{J}} \leq b$ holds. Let $\hat{k} \in \mathbb{N}$ such that $k_t^A, k_t^B, l_s^A, l_s^B \leq \hat{k}$ holds for all $t \in \mathcal{T}_\mathcal{I}$,*

Algorithm 6 Matrix addition

procedure RecursiveAddition(b) $\{b \in \mathcal{T}^A_{\mathcal{I}\times\mathcal{J}} \cap \mathcal{T}^B_{\mathcal{I}\times\mathcal{J}}\}$
if $\operatorname{sons}(b, \mathcal{T}^A_{\mathcal{I}\times\mathcal{J}}) = \emptyset$ **then**
  if $b \in \mathcal{L}^A_{\mathcal{I}\times\mathcal{J},+}$ **then**
    $\tilde{S}^B_b := \tilde{S}^B_b + S^A_b$ {Handled by matrix backward transformation}
  else
    if $b \in \mathcal{L}^B_{\mathcal{I}\times\mathcal{J},+}$ **then**
      $S^B_b := S^B_b + \tilde{S}^A_b$ {Prepared by matrix forward transformation}
    else
      $B := B + \chi_t A \chi_s$ {Direct addition of dense matrices}
    end if
  end if
else
  if $\operatorname{sons}(b, \mathcal{T}^B_{\mathcal{I}\times\mathcal{J}}) = \emptyset$ **then**
    $S^B_b := S^B_b + \tilde{S}^A_b$ {Prepared by matrix forward transformation}
  else
    for $b' = (t', s') \in \operatorname{sons}(b, \mathcal{T}^A_{\mathcal{I}\times\mathcal{J}})$ **do**
      RecursiveAddition(b') $\{\operatorname{sons}(b, \mathcal{T}^A_{\mathcal{I}\times\mathcal{J}}) = \operatorname{sons}(b, \mathcal{T}^B_{\mathcal{I}\times\mathcal{J}})\}$
    end for
  end if
end if

procedure MatrixAddition(A, **var** B)
ClusterBasisProduct(root($\mathcal{T}_{\mathcal{I}}$), V^B, V^A, X);
ClusterBasisProduct(root($\mathcal{T}_{\mathcal{J}}$), W^A, W^B, Y);
MatrixForward(root($\mathcal{T}^A_{\mathcal{I}\times\mathcal{J}}$), V^B, W^B, X, Y, A, $\tilde{S}^A$);
$\tilde{S}^B := 0$;
RecursiveAddition(root($\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$));
MatrixBackward(root($\mathcal{T}^B_{\mathcal{I}\times\mathcal{J}}$), V^A, W^A, X, Y, B, $\tilde{S}^B$)

$s \in \mathcal{T}_{\mathcal{J}}$ and that $\#\hat{t}, \#\hat{s} \leq \hat{k}$ holds for all $t \in \mathcal{L}_{\mathcal{I}}$, $s \in \mathcal{L}_{\mathcal{I}}$.

Then the $\mathcal{H}^2$-matrix addition requires $\mathcal{O}(\hat{k}^3(b+c))$ arithmetic operations.

Proof. Prior to the addition, we have to prepare the auxiliary matrices $\tilde{S}_b^A$ and $\tilde{S}_b^B$. Due to Lemma 3.2, this can be accomplished in $\mathcal{O}(\hat{k}^3(b+c))$ operations.

The procedure RecursiveAddition is only called for blocks $b \in \mathcal{T}^A_{\mathcal{I}\times\mathcal{J}} \cap \mathcal{T}^B_{\mathcal{I}\times\mathcal{J}}$, i.e., not more than b times. In each call, at most one matrix addition takes place, and the involved matrices have not more than $\hat{k}$ rows or columns, therefore one addition requires not more than $\mathcal{O}(\hat{k}^2)$ operations, and all additions require not more than $\mathcal{O}(\hat{k}^2 b)$ operations.

Combining this estimate with the one from Lemma 3.2 proves the desired result. ∎

Remark 3.4 (Linear complexity) *If $\mathcal{T}^A_{\mathcal{I}\times\mathcal{J}}$ and $\mathcal{T}^B_{\mathcal{I}\times\mathcal{J}}$ are sparse with sparsity constant C_{sp}, we have $b \leq C_{\mathrm{sp}} c$.*

If all clusters in the cluster tree satisfy $\#\operatorname{sons}(t) \neq 1$ and $\#\hat{t} > 0$, we find $c \leq 2n-1$ and conclude that the $\mathcal{H}^2$-matrix addition requires $\mathcal{O}(\hat{k}^3 n)$ operations, i.e., its complexity grows only linearly in the number of degrees of freedom.

Remark 3.5 (Implementation) *Storing all auxiliary matrices $\tilde{S}_b^A$ and $\tilde{S}_b^B$ requires $\mathcal{O}(\hat{k}^2 b)$ units of storage, which is on the same order as the storage required for the matrices A and B themselves.*

In practical implementations, this is not acceptable. The amount of auxiliary storage can be reduced drastically by combining matrix forward and backward transformations with the addition routine, i.e., by writing benign "Spaghetti code". Using this approach, only one auxiliary $\hat{k} \times \hat{k}$-matrix is required per level of $\mathcal{T}^A_{\mathcal{I}\times\mathcal{J}}$ or $\mathcal{T}^B_{\mathcal{I}\times\mathcal{J}}$.

The error equation (8) implies that the approximation error $\|(A+B) - \Pi(A+B)\|_F$ depends only on the choice of the space of $\mathcal{H}^2$-matrices used for the representation. Its meaning for $\mathcal{H}^2$-matrices can be compared to that of Cea's Lemma for Galerkin discretizations: if the solution can be approximated in the given space, the computed approximation will be close to the true solution. For Galerkin discretizations, Cea's Lemma is typically combined with regularity assumptions for the true solution in order to derive error estimates. For $\mathcal{H}^2$-matrices, we also need "regularity assumptions" for the true solution. In the general case, these assumptions have been investigated in [4], in the simple case of integral operators with asymptotically smooth kernel functions, the error analysis is more simple:

Example 3.6 (Integral operators) *Let us consider the case of Example 2.9. If A and B correspond to approximations of integral operators with asymptotically smooth kernel functions, so will $A+B$, i.e., $\Pi(A+B)$ will be an $\mathcal{H}^2$-matrix approximation of an asymptotically smooth kernel function, therefore the approximation error $\|(A+B) - \Pi(A+B)\|_F$ can be bounded by the error of polynomial interpolation [8] and will therefore converge exponentially if the rank is increased.*

4 Matrix multiplication

The matrix forward and backward transformations can also be applied to speed up the multiplication of $\mathcal{H}^2$-matrices. Our goal is to compute $C^{\text{new}} := C + \Pi(AB)$, where $C, C^{\text{new}} \in \mathcal{H}^2(\mathcal{T}^C_{\mathcal{I}\times\mathcal{K}}, V^C, W^C)$, $A \in \mathcal{H}^2(\mathcal{T}^A_{\mathcal{I}\times\mathcal{J}}, V^A, W^A)$ and $B \in \mathcal{H}^2(\mathcal{T}^B_{\mathcal{J}\times\mathcal{K}}, V^B, W^B)$ and Π denotes the orthogonal projection into $\mathcal{H}^2(\mathcal{T}^C_{\mathcal{I}\times\mathcal{K}}, V^C, W^C)$. As in the case of the addition, we will compute the best approximation of C^{new} in the space $\mathcal{M} = \mathcal{H}^2(\mathcal{T}^C_{\mathcal{I}\times\mathcal{K}}, V^C, W^C)$, i.e., we will have

$$\|(C + AB) - C^{\text{new}}\|_F = \|(C + AB) - \Pi(C + AB)\|_F = \inf_{M\in\mathcal{M}} \|(C + AB) - M\|_F. \quad (10)$$

Our algorithm will overwrite the representation of C with that of the result C^{new}.

We denote the transfer matrices for V^A,V^B and V^C by E^A, E^B and E^C and those for W^A,W^B and W^C by F^A, F^B and F^C.

For the sake of simplicity, we require V^C and W^C to be orthogonal.

We will again proceed recursively. In the case of the addition, only two matrices are involved, so the number of possible combinations of blocks (each can be an admissible or non-admissible leaf or not a leaf) is limited. In the case of the multiplication, the number of combinations triples, which makes writing one large procedure to handle them all inconvenient. Therefore we will split the computation into three cases: The case that the target matrix block is an admissible leaf of $\mathcal{T}^C_{\mathcal{I}\times\mathcal{K}}$, the case that it is a non-admissible leaf of $\mathcal{T}^C_{\mathcal{I}\times\mathcal{K}}$, and the case that it is not a leaf.

4.1 Admissible leaf as target

This is the most important of the three cases: experiments indicate that approximately 50% of products fall into this category. Therefore we need an especially efficient way of handling it.

We have to compute

$$S^{C,\text{new}}_{(t,r)} := S^C_{(t,r)} + (V^C_t)^\top A\chi_s B W^C_r \quad (11)$$

for $t \in \mathcal{T}_\mathcal{I}$, $s \in \mathcal{T}_\mathcal{J}$ and $r \in \mathcal{T}_\mathcal{K}$, since then the orthogonality of V^C and W^C guarantees that $S^{C,\text{new}}_{(t,r)}$ will correspond to the best approximation of the desired result.

If $(t, s) \in \mathcal{L}^A_{\mathcal{I}\times\mathcal{J},+}$, i.e., if (t, s) is an admissible leaf, we know that

$$\chi_t A\chi_s = V^A_t S^A_{(t,s)} (W^A_s)^\top$$

holds and can use this equation to re-write the operation in the form

$$S^{C,\text{new}}_{(t,r)} = S^C_{(t,r)} + (V^C_t)^\top V^A_t S^A_{(t,s)} (W^A_s)^\top B W^C_r.$$

We see that we can apply two techniques already established in the previous section: if we prepare the auxiliary matrices $X_t := (V^C_t)^\top V^A_t$ and $\tilde{S}^B_{(s,r)} := (W^A_s)^\top B W^C_r$ in advance, we find that we only have to compute

$$S^{C,\text{new}}_{(t,r)} = S^C_{(t,r)} + X_t S^A_{(t,s)} \tilde{S}^B_{(s,r)},$$

Algorithm 7 Matrix multiplication with admissible leaf as target

procedure MultiplyAdmissible(t, s, r)
if $(t,s) \in \mathcal{L}^A_{\mathcal{I}\times\mathcal{J},+} \vee (t,s) \notin \mathcal{T}^A_{\mathcal{I}\times\mathcal{J}}$ **then**
  $S^C_{(t,r)} := S^C_{(t,r)} + X_t S^A_{(t,s)} \tilde{S}^B_{(s,r)}$ {Prepared by matrix forward transformation}
else
  if $(s,r) \in \mathcal{L}^B_{\mathcal{J}\times\mathcal{K},+} \vee (s,r) \notin \mathcal{T}^B_{\mathcal{J}\times\mathcal{K}}$ **then**
    $S^C_{(t,r)} := S^C_{(t,r)} + \tilde{S}^A_{(t,s)} S^B_{(s,r)} Z_r$ {Prepared by matrix forward transformation}
  else
    if $(t,s) \in \mathcal{L}^A_{\mathcal{I}\times\mathcal{J},-} \wedge (s,r) \in \mathcal{L}^B_{\mathcal{J}\times\mathcal{K},-}$ **then**
      $S^C_{(t,r)} := S^C_{(t,r)} + (V^C_t)^\top A \chi_s B W^C_r$ {All clusters are leaves}
    else
      for $t' \in \operatorname{sons}^+(t), r' \in \operatorname{sons}^+(r)$ **do**
        $S^C_{(t',r')} := 0$; {Create artificial block (t',r')}
        for $s' \in \operatorname{sons}^+(s)$ **do**
          MultiplyAdmissible(t', s', r');
        end for
        Collect((t',r'), (t,r), E^C, F^C, $S^C_{(t',r')}$, $S^C_{(t,r)}$)
      end for
    end if
  end if
end if

and this operation involves only matrices of low dimension.

If $(s,r) \in \mathcal{L}^B_{\mathcal{J}\times\mathcal{K},+}$, i.e., if (s,r) is an admissible leaf, we can use the same argument to find

$$S^{C,\mathrm{new}}_{(t,r)} = S^C_{(t,r)} + \tilde{S}^A_{(t,s)} S^B_{(s,r)} Z_r,$$

where we use the auxiliary matrices $Z_r := (W^B_r)^\top W^C_r$ and $\tilde{S}^A_{(t,s)} := (V^A_t)^\top S^A_{(t,s)} V^B_s$.

Using the Algorithm 3 and the matrix forward transformation Algorithm 4, all auxiliary matrices can be computed efficiently.

If (t,s) and (s,r) are non-admissible leaves, we can compute their product directly by using (11).

If (t,s) or (s,r) is not a leaf, we introduce an auxiliary matrix for each combination of $t' \in \operatorname{sons}^+(t)$ and $r' \in \operatorname{sons}^+(r)$, compute the corresponding product recursively and then use Algorithm 2 to construct the best approximation for the entire block.

All of these different sub-cases are treated by Algorithm 7.

4.2 Non-admissible leaf as target

In this case, we have to compute

$$C^{\mathrm{new}} := C + \chi_t A \chi_s B \chi_r.$$

Algorithm 8 Matrix multiplication with non-admissible leaf as target

procedure MultiplyDense(t, s, r)
if $(t,s) \in \mathcal{L}^A_{\mathcal{I}\times\mathcal{J},+} \vee (t,s) \notin \mathcal{T}^A_{\mathcal{I}\times\mathcal{J}}$ **then**
  if $(s,r) \in \mathcal{L}^B_{\mathcal{J}\times\mathcal{K},+} \vee (s,r) \notin \mathcal{T}^B_{\mathcal{J}\times\mathcal{K}}$ **then**
    $\tilde{S}^C_{(t,s)} := \tilde{S}^C_{(t,s)} + S^A_{(t,s)} Y_s S^B_{(s,r)}$ {Handled by backward matrix transformation}
  else if $(s,r) \in \mathcal{L}^B_{\mathcal{J}\times\mathcal{K},-}$ **then**
    $C := C + V^A_t S^A_{(t,s)} (W^A_s)^\top B \chi_r$ {t, s and r are leaves}
  else
    for $s' \in \operatorname{sons}(s)$ **do**
      Split((t,s), (t,s'), E^A, F^A, $S^A_{(t,s)}$, $S^A_{(t,s')}$); {Create artificial block (t,s')}
      MultiplyDense(t, s', r)
    end for
  end if
else if $(t,s) \in \mathcal{L}^A_{\mathcal{I}\times\mathcal{J},-}$ **then**
  if $(s,r) \in \mathcal{L}^B_{\mathcal{J}\times\mathcal{K},+} \vee (s,r) \notin \mathcal{T}^B_{\mathcal{J}\times\mathcal{K}}$ **then**
    $C := C + \chi_t A V^B_s S^B_{(s,r)} (W^B_r)^\top$ {t, s and r are leaves}
  else
    $C := C + \chi_t A \chi_s B \chi_r$ {$(s,r) \in \mathcal{L}^B_{\mathcal{J}\times\mathcal{K},-}$}
  end if
else
  if $(s,r) \in \mathcal{L}^B_{\mathcal{J}\times\mathcal{K},+} \vee (s,r) \notin \mathcal{T}^B_{\mathcal{J}\times\mathcal{K}}$ **then**
    for $s' \in \operatorname{sons}(s)$ **do**
      Split((s,r), (s',r), E^B, F^B, $S^B_{(s,r)}$, $S^B_{(s',r)}$); {Create artificial block (s',r)}
      MultiplyDense(t, s', r)
    end for
  else
    for $s' \in \operatorname{sons}(s)$ **do**
      MultiplyDense(t, s', r)
    end for
  end if
end if

Non-admissible leaves appear only if the row cluster t and the column cluster r are leaves of $\mathcal{T}_\mathcal{I}$ and $\mathcal{T}_\mathcal{K}$, respectively, so if the block (t,s) of A or the block (s,r) of B are not leaves, their sons can only have the form (t,s') or (s',r), where $s' \in \operatorname{sons}(s)$. If, e.g., $(t,s) \in \mathcal{L}^A_{\mathcal{I}\times\mathcal{J},-}$ holds, this even implies $\operatorname{sons}(s) = \emptyset$, and in this case $(s,r) \in \mathcal{L}^B_{\mathcal{J}\times\mathcal{K}}$ has to hold.

Let us first consider the case that (t,s) and (s,r) are admissible. This implies

$$\chi_t A \chi_s = V_t^A S^A_{(t,s)} (W_s^A)^\top \quad \text{and} \quad \chi_s B \chi_r = V_s^B S^B_{(t,s)} (W_r^B)^\top.$$

We can again use Algorithm 3 to prepare auxiliary matrices $Y_s := (W_s^A)^\top V_s^B$ in advance and find

$$C^{\text{new}} = C + V_t^A S^A_{(t,s)} Y_s S^B_{(t,s)} (W_s^A)^\top.$$

As in the case of the addition, we handle this situation by storing only the auxiliary matrix

$$\tilde{S}^C_{(t,r)} := S^A_{(t,s)} Y_s S^B_{(t,s)}$$

and then using the matrix backward transformation Algorithm 5 to assemble the product after completing the recursion.

As mentioned before, $(t,s) \in \mathcal{L}^A_{\mathcal{I}\times\mathcal{J},-}$ implies $(s,r) \in \mathcal{L}^B_{\mathcal{J}\times\mathcal{K}}$, and $(s,r) \in \mathcal{L}^B_{\mathcal{J}\times\mathcal{K},-}$ implies $(t,s) \in \mathcal{L}^A_{\mathcal{I}\times\mathcal{J}}$, so we can treat these cases directly.

This leaves us with the case of non-leaf blocks. If both (t,s) and (s,r) are not leaves, their sons have the form (t,s') and (s',r) for $s' \in \operatorname{sons}(s)$, so we handle them by a recursion.

If (t,s) is admissible and (s,r) is not a leaf, we have to create artificial blocks (t,s') for all $s' \in \operatorname{sons}(s)$. Due to the definition of the admissibility condition, the artificial blocks will also be admissible and we can create them using Algorithm 1. We also have to modify our algorithms in such a way that they assume that all blocks that are not part of the corresponding block cluster trees are artificial.

Treating the situation that (s,r) is admissible and (t,s) is not a leaf similarly leads us to Algorithm 8.

4.3 Non-leaf as target

This case is similar to that of non-admissible leaves: if (t,s) and (s,r) are admissible, we multiply and store the result in the auxiliary matrix $\tilde{S}^C_{(t,r)}$, and this matrix is handled after the recursion by the matrix backward transformation.

If (t,s) is admissible and (s,r) is not, we again create an artificial block and proceed recursively. A similar procedure is applied if (s,r) is admissible and (t,s) is not.

If both (t,s) and (s,r) are not admissible, we can use a simple recursion to compute the result.

4.4 Multiplication algorithm

Now that we know how to treat all possible combinations of blocks, we can combine the Algorithms 7, 8 and 9, which perform the multiplication itself, with the Algorithms 3,

Algorithm 9 Matrix multiplication with non-leaf as target

procedure MultiplySub(t, s, r)
if $(t,s) \in \mathcal{L}^A_{\mathcal{I}\times\mathcal{J},+} \vee (t,s) \notin \mathcal{T}^A_{\mathcal{I}\times\mathcal{J}}$ **then**
  if $(s,r) \in \mathcal{L}^B_{\mathcal{J}\times\mathcal{K},+} \vee (s,r) \notin \mathcal{T}^B_{\mathcal{J}\times\mathcal{K}}$ **then**
    $\tilde{S}^C_{(t,r)} := \tilde{S}^C_{(t,s)} + S^A_{(t,s)} Y_s S^B_{(s,r)}$ {Handled by backward matrix transformation}
  else
    for $t' \in \text{sons}^+(t)$, $s' \in \text{sons}^+(s)$ **do**
      Split((t,s), (t',s'), E^A, F^A, $S^A_{(t,s)}$, $S^A_{(t',s')}$); {Create artificial block (t',s')}
      for $r' \in \text{sons}^+(r)$ **do**
        RecursiveMultiply(t', s', r')
      end for
    end for
  end if
else
  if $(s,r) \in \mathcal{L}^B_{\mathcal{J}\times\mathcal{K},+} \vee (s,r) \notin \mathcal{T}^B_{\mathcal{J}\times\mathcal{K}}$ **then**
    for $s' \in \text{sons}^+(s)$, $r' \in \text{sons}^+(r)$ **do**
      Split((s,r), (s',r'), E^A, F^A, $S^A_{(s,r)}$, $S^A_{(t',r')}$); {Create artificial block (s',r')}
      for $t' \in \text{sons}^+(t)$ **do**
        RecursiveMultiply(t', s', r')
      end for
    end for
  else
    for $t' \in \text{sons}^+(t)$, $s' \in \text{sons}^+(s)$, $r' \in \text{sons}^+(r)$ **do**
      RecursiveMultiply(t', s', r')
    end for
  end if
end if

Algorithm 10 Matrix multiplication

procedure RecursiveMultiply(t, s, r)
if $(t,r) \in \mathcal{L}^C_{\mathcal{I}\times\mathcal{K},+}$ **then**
  MultiplyAdmissible(t, s, r)
else if $(t,r) \in \mathcal{L}^C_{\mathcal{I}\times\mathcal{K},-}$ **then**
  MultiplyDense(t, s, r)
else
  MultiplySub(t, s, r)
end if

procedure MatrixMultiplication(A, B, **var** C)
ClusterBasisProduct(root($\mathcal{T}_\mathcal{I}$), V^C, V^A, X);
ClusterBasisProduct(root($\mathcal{T}_\mathcal{J}$), W^A, V^B, Y);
ClusterBasisProduct(root($\mathcal{T}_\mathcal{K}$), W^B, W^C, Z);
MatrixForward(root($\mathcal{T}^A_{\mathcal{I}\times\mathcal{J}}$), V^C, V^B, X, Y, A, $\tilde{S}^A$);
MatrixForward(root($\mathcal{T}^B_{\mathcal{J}\times\mathcal{K}}$), W^A, W^C, Y, Z, B, $\tilde{S}^B$);
$\tilde{S}^C := 0$;
RecursiveMultiply(root($\mathcal{T}_\mathcal{I}$), root($\mathcal{T}_\mathcal{J}$), root($\mathcal{T}_\mathcal{K}$));
MatrixBackward(root($\mathcal{T}^C_{\mathcal{I}\times\mathcal{K}}$), V^A, W^B, X, Z, C, $\tilde{S}^C$)

4 and 5, which handle the auxiliary matrices, to get Algorithm 10, the $\mathcal{H}^2$-matrix multiplication.

Let us consider the total complexity of this algorithm. Due to the Lemmas 3.1 and 3.2, the construction and transformation of the auxiliary matrices can be accomplished efficiently. Obviously, each single step of the recursion is rather inexpensive, so we only need to bound the number of calls to the main recursion. This means that we have to ensure that the artificial matrices created in our algorithms will not interfere with this bound.

Lemma 4.1 *If the Algorithms 7, 8 and 9 are called with $t \in \mathcal{T}_\mathcal{I}$, $s \in \mathcal{T}_\mathcal{J}$ and $r \in \mathcal{T}_\mathcal{K}$, not more than one of the blocks (t,s), (s,r) and (t,r) can be artificial.*

Proof. We proceed by induction: the Algorithm 10 starts with $t = \operatorname{root}(\mathcal{T}_\mathcal{I})$, $s = \operatorname{root}(\mathcal{T}_\mathcal{J})$ and $r = \operatorname{root}(\mathcal{T}_\mathcal{K})$. By definition, we have $(t,s) \in \mathcal{T}^A_{\mathcal{I}\times\mathcal{J}}$, $(s,r) \in \mathcal{T}^B_{\mathcal{J}\times\mathcal{K}}$ and $(t,r) \in \mathcal{T}^C_{\mathcal{I}\times\mathcal{K}}$, so in the first recursion step, all blocks are not artificial.

Now let us inspect the circumstances in which artificial blocks are created:

In Algorithm 7, this happens only when both (t,s) and (s,r) are not admissible and therefore not artificial, so their descendants can not be artificial.

In Algorithm 8, an artificial block is created if (s,r) is not a leaf. Since (t,r) is a non-admissible leaf, both (s,r) and (t,r) can not be artificial, so the same holds again for their descendants. Due to symmetry, the same holds for the case that (t,s) is not a leaf.

In Algorithm 9, an artificial descendant of (t, s) is created if (s, r) is not admissible. As in the previous cases, this implies that (s, r) and (t, r) are not artificial, therefore their descendants are not artificial. Again due to symmetry, the same statement is valid for the case that (t, s) is not admissible. ∎

Using this lemma, we can now bound the number of calls to the recursion:

Lemma 4.2 *Let $\mathcal{T}^A_{\mathcal{I}\times\mathcal{J}}$, $\mathcal{T}^B_{\mathcal{J}\times\mathcal{K}}$ and $\mathcal{T}^C_{\mathcal{I}\times\mathcal{K}}$ be sparse with sparsity constant C_{sp}. Let $c \in \mathbb{N}$ satisfy $\#\mathcal{T}_{\mathcal{I}}, \#\mathcal{T}_{\mathcal{J}}, \#\mathcal{T}_{\mathcal{K}} \leq c$.*

Then the Algorithms 7, 8 and 9 are called not more than $3C_{\mathrm{sp}}^2 c$ times during the course of Algorithm 10.

Proof. The inspection of the recursive calls in the Algorithms 7, 8 and 9 shows that one combination of clusters t, s and r appears at most in one function call.

Let $t \in \mathcal{T}_{\mathcal{I}}$. We set

$$R_t := \{(s, r) \in \mathcal{T}_{\mathcal{J}} \times \mathcal{T}_{\mathcal{K}} \ : \ (t, s, r) \text{ appears in a function call}\}.$$

We will now prove

$$\begin{aligned}
&R_t \subseteq R_t^1 \cup R_t^2 \cup R_t^3 \quad \text{for} \\
&R_t^1 := \{(s, r) \in \mathcal{T}_{\mathcal{J}} \times \mathcal{T}_{\mathcal{K}} \ : \ (t, s) \in \mathcal{T}^A_{\mathcal{I}\times\mathcal{J}}, (s, r) \in \mathcal{T}^B_{\mathcal{J}\times\mathcal{K}}\} \\
&R_t^2 := \{(s, r) \in \mathcal{T}_{\mathcal{J}} \times \mathcal{T}_{\mathcal{K}} \ : \ (t, s) \in \mathcal{T}^A_{\mathcal{I}\times\mathcal{J}}, (t, r) \in \mathcal{T}^C_{\mathcal{I}\times\mathcal{K}}\} \\
&R_t^3 := \{(s, r) \in \mathcal{T}_{\mathcal{J}} \times \mathcal{T}_{\mathcal{K}} \ : \ (s, r) \in \mathcal{T}^B_{\mathcal{J}\times\mathcal{K}}, (t, r) \in \mathcal{T}^C_{\mathcal{I}\times\mathcal{K}}\}.
\end{aligned}$$

Let $(s, r) \in R_t$. If $(t, r) \notin \mathcal{T}^C_{\mathcal{I}\times\mathcal{K}}$, Lemma 4.1 implies $(s, r) \in R_r^1$. If $(s, r) \notin \mathcal{T}^B_{\mathcal{J}\times\mathcal{K}}$, the same lemma implies $(s, r) \in R_t^2$. Otherwise, we have $(s, r) \in R_t^3$.

Due to the sparsity of $\mathcal{T}^A_{\mathcal{I}\times\mathcal{J}}$, $\mathcal{T}^B_{\mathcal{J}\times\mathcal{K}}$ and $\mathcal{T}^C_{\mathcal{I}\times\mathcal{K}}$ the estimates

$$\#R_t^1 \leq C_{\mathrm{sp}}^2, \quad \#R_t^2 \leq C_{\mathrm{sp}}^2 \quad \text{and} \quad \#R_t^3 \leq C_{\mathrm{sp}}^2$$

hold, and we can conclude $\#R_t \leq 3C_{\mathrm{sp}}^2$. Summing up over all $t \in \mathcal{T}_{\mathcal{I}}$ yields the desired estimate. ∎

Now we can prove a bound for the complexity of the $\mathcal{H}^2$-matrix multiplication.

Theorem 4.3 ($\mathcal{H}^2$-matrix multiplication) *Let $c \in \mathbb{N}$ such that $\#\mathcal{T}_{\mathcal{I}}, \#\mathcal{T}_{\mathcal{J}}, \#\mathcal{T}_{\mathcal{K}} \leq c$ holds. Let $\mathcal{T}^A_{\mathcal{I}\times\mathcal{J}}$, $\mathcal{T}^B_{\mathcal{J}\times\mathcal{K}}$ and $\mathcal{T}^C_{\mathcal{I}\times\mathcal{K}}$ be sparse with sparsity constant C_{sp}.*

Let $\hat{k} \in \mathbb{N}$ such that $k_t^C, k_t^A, l_s^A, k_s^B, l_r^B, l_r^C \leq \hat{k}$ holds for all $t \in \mathcal{T}_{\mathcal{I}}$, $s \in \mathcal{T}_{\mathcal{J}}$, $r \in \mathcal{T}_{\mathcal{K}}$ and that $\#\hat{t}, \#\hat{s}, \#\hat{r} \leq \hat{k}$ holds for all $t \in \mathcal{L}_{\mathcal{I}}$, $s \in \mathcal{L}_{\mathcal{I}}$ and $r \in \mathcal{L}_{\mathcal{I}}$.

Then the $\mathcal{H}^2$-matrix multiplication Algorithm 10 requires $\mathcal{O}(C_{\mathrm{sp}}^2 \hat{k}^3 c)$ arithmetic operations.

Proof. Due to Lemma 4.2, we have to compute not more than $3C_{\mathrm{sp}}^2 c$ products. Each of the algorithms requires only $\mathcal{O}(\hat{k}^3)$ operations per product, so the total complexity for the recursion is $\mathcal{O}(C_{\mathrm{sp}}^2 \hat{k}^3 c)$.

The Lemmas 3.1 and 3.2 imply that the preparation of X, Y, Z, $\tilde{S}^A$ and $\tilde{S}^B$ and the handling of $\tilde{S}^C$ requires $\mathcal{O}(\hat{k}^3 c)$ operations, which concludes the proof. ∎

Remark 4.4 (Linear complexity) *If $\mathcal{T}^A_{\mathcal{I}\times\mathcal{J}}$, $\mathcal{T}^B_{\mathcal{J}\times\mathcal{K}}$ and $\mathcal{T}^C_{\mathcal{I}\times\mathcal{K}}$ are sparse with sparsity constant C_{sp} and if all clusters in $\mathcal{T}_{\mathcal{I}}$, $\mathcal{T}_{\mathcal{J}}$ and $\mathcal{T}_{\mathcal{K}}$ are not empty and have either no or at least two clusters, the $\mathcal{H}^2$-matrix multiplication requires $\mathcal{O}(C_{\mathrm{sp}}^2\hat{k}^3 n)$ operations.*

Remark 4.5 (Implementation) *We can avoid storing $\tilde{S}^A$ and $\tilde{S}^B$ by combining the matrix forward transformations with the main recursion of the multiplication. This means that only auxiliary storage for X, Y, Z and $\tilde{S}^C$ is required.*

As in the case of the matrix addition, we can combine "regularity assumptions" for the exact solution $C + AB$ with the error equation (10) in order to derive error bounds for the approximative solution C^{new}. The case of general matrices can be reduced to the results presented in [4], the case of integral operators with asymptotically smooth kernel functions can again be handled by relatively simple arguments:

Example 4.6 (Integral operators) *Let us consider the case of Example 2.9. If A and B correspond to approximations of integral operators with asymptotically smooth kernel functions, the product AB will, up to discretization errors, correspond to the convolution of the kernel functions. Under standard assumptions, the convolution of asymptotically smooth kernel functions is again asymptotically smooth, and we find that the approximation error $\|AB-\Pi(AB)\|_F$ can be bounded by the exponentially convergent error of polynomial interpolation [8].*

5 Numerical experiments

We compute the $\mathcal{H}^2$-matrix approximations [8] of the classical single and double layer potential operators

$$\mathcal{V}[u](x) = \int_\Gamma \frac{u(y)}{4\pi\|x-y\|}\,dy, \qquad \mathcal{K}[u](x) = \int_\Gamma \frac{\langle x-y, n(y)\rangle\, u(y)}{4\pi\|x-y\|^3}\,dy,$$

discretized by Galerin's method using n piecewise constant basis functions. We approximate the kernel function

$$\gamma(x,y) := \frac{1}{4\pi\|x-y\|}$$

by Chebyshev interpolation of order m. The kernel function of $\mathcal{V}$ is γ, so the interpolant of γ gives rise to an $\mathcal{H}^2$-matrix approximation V of the related discrete operator. The kernel function of $\mathcal{K}$ is the normal derivative of γ, therefore we can approximate it by the normal derivative of the interpolant. This yields an $\mathcal{H}^2$-matrix approximation K of the corresponding discrete operator.

On a smooth surface, $\mathcal{V}$ and $\mathcal{K}$ correspond to asymptotically smooth kernel functions, and we expect (cf. [18, Satz 2.8]) that this will also hold for their compositions $\mathcal{V}\mathcal{V}$, $\mathcal{K}\mathcal{V}$, $\mathcal{V}\mathcal{K}$ and $\mathcal{K}\mathcal{K}$, i.e., that these compositions can be approximated efficiently by $\mathcal{H}^2$-matrices.

Up to errors introduced by the discretization process, the matrix products VV, KV, VK and KK can be expected to share the properties of their continuous counterparts,

Oper.	n	m=2		m=3		m=4		m=5	
VV	512	2.2	1.5_{-4}	1.0	8.5_{-7}	0.4	conv.	0.9	conv.
	2048	13.0	2.6_{-4}	32.3	9.6_{-6}	40.8	4.6_{-7}	47.4	conv.
	8192	66.5	4.6_{-4}	184.2	3.6_{-5}	355.2	1.6_{-6}	1187.8	1.6_{-7}
	32768	283.9	5.4_{-4}	897.0	4.9_{-5}	1919.2	2.3_{-6}	6562.9	3.1_{-7}
	131072	1196.8	5.6_{-4}	3625.6	4.7_{-5}	7918.5	3.6_{-6}	27784.2	4.5_{-7}
KV	512	2.3	2.6_{-3}	1.0	4.9_{-5}	0.4	conv.	1.0	conv.
	2048	13.8	5.4_{-3}	34.4	1.7_{-4}	41.7	5.9_{-6}	48.7	conv.
	8192	71.1	1.0_{-2}	196.7	8.5_{-4}	374.8	5.3_{-5}	1220.3	2.6_{-6}
	32768	304.3	1.6_{-2}	959.9	2.3_{-3}	1982.2	1.4_{-4}	6890.1	9.6_{-6}
	131072	1257.4	2.3_{-2}	3876.3	4.0_{-3}	8361.8	2.8_{-4}	29784.5	3.1_{-5}
VK	512	2.3	5.0_{-3}	1.0	7.8_{-6}	0.4	conv.	1.0	conv.
	2048	14.0	2.1_{-2}	35.6	4.2_{-4}	41.7	1.9_{-5}	48.6	conv.
	8192	72.7	4.2_{-2}	204.3	2.0_{-3}	395.3	1.3_{-4}	1268.2	7.1_{-6}
	32768	313.1	7.0_{-2}	1003.3	4.1_{-3}	2098.5	2.6_{-4}	7157.2	2.6_{-5}
	131072	1323.6	1.1_{-1}	4101.6	7.1_{-3}	8744.8	5.1_{-4}	30979.0	5.9_{-5}
KK	512	2.2	6.9_{-4}	1.0	9.4_{-6}	0.4	conv.	0.9	conv.
	2048	12.9	2.9_{-3}	32.4	5.2_{-5}	40.6	4.8_{-6}	47.5	conv.
	8192	66.4	6.1_{-3}	184.0	2.6_{-4}	354.4	1.8_{-5}	1154.2	9.5_{-7}
	32768	283.9	1.1_{-2}	894.0	5.5_{-4}	1881.5	3.5_{-5}	6538.1	3.4_{-6}
	131072	1169.1	1.6_{-2}	3602.0	9.6_{-4}	7839.9	6.8_{-5}	27652.0	7.9_{-6}

Table 1: Multiplying double and single layer potential on the unit sphere

i.e., the the approximations computed by our algorithm should converge exponentially if the order of the polynomial approximation is increased.

In the first example, we consider a sequence of polygonal approximations of the unit sphere $\Gamma_S = \{x \in \mathbb{R}^3 \ : \ \|x\| = 1\}$ consisting of $n = 512$, 2048, 8192, 32768 and 131072 plane triangles.

The results are collected in Table 1. For each grid, each product and each interpolation order, it contains the time[1] in seconds for computing the product by Algorithm 10 and the resulting approximation error in the operator norm, which was estimated using a power iteration.

We can see that increasing the interpolation order m will typically reduce the approximation error by a factor of 10, i.e., we observe the expected exponential convergence.

Since we are working with interpolation in three space dimensions, the rank of the cluster bases will be bounded by $k = m^3$, i.e., we expect a behaviour like $\mathcal{O}(nk^2) = \mathcal{O}(nm^6)$ in the computation time. Especially for higher interpolation orders and higher problem dimensions, this behaviour can indeed be observed.

In Table 2, we investigate the behaviour of the polynomial approximation on the

[1]On one 900 MHz UltraSPARC IIIcu processor of a SunFire 6800 computer.

Oper.	n	m=2		m=3		m=4		m=5	
VV	768	2.0	2.9_{-3}	3.8	3.5_{-4}	1.2	conv.	1.5	conv.
	3072	10.9	3.6_{-3}	32.0	6.3_{-4}	133.7	2.0_{-4}	182.1	6.6_{-5}
	12288	49.7	3.8_{-3}	176.7	7.9_{-4}	455.6	2.8_{-4}	1411.8	1.2_{-4}
	49152	208.6	4.0_{-3}	850.7	8.5_{-4}	1867.4	3.4_{-4}	7580.0	1.6_{-4}
	196608	833.0	4.0_{-3}	3692.2	8.6_{-4}	7542.0	3.6_{-4}	36077.4	1.7_{-4}
KV	768	2.2	6.3_{-2}	3.9	5.5_{-3}	1.2	conv.	1.6	conv.
	3072	11.7	7.0_{-2}	34.9	1.4_{-2}	134.6	3.2_{-3}	185.2	1.0_{-3}
	12288	53.4	8.5_{-2}	200.4	1.9_{-2}	470.0	7.2_{-3}	1510.5	3.2_{-3}
	49152	222.8	8.6_{-2}	978.0	2.1_{-2}	1922.1	8.6_{-3}	8280.3	4.2_{-3}
	196608	869.8	8.2_{-2}	4245.9	2.1_{-2}	7862.8	9.1_{-3}	40326.5	4.6_{-3}
VK	768	2.4	1.9_{-1}	3.8	4.4_{-2}	1.2	conv.	1.6	conv.
	3072	11.8	2.7_{-1}	37.7	1.1_{-1}	134.4	3.9_{-2}	185.8	2.0_{-2}
	12288	55.0	3.4_{-1}	214.5	1.6_{-1}	484.0	8.3_{-2}	1638.2	5.1_{-2}
	49152	232.3	3.7_{-1}	1059.7	2.0_{-1}	2045.5	1.2_{-1}	8951.6	8.0_{-2}
	196608	930.5	3.7_{-1}	4614.4	2.3_{-1}	8454.1	1.4_{-1}	43588.5	1.0_{-1}
KK	768	2.0	2.7_{-2}	3.8	5.1_{-3}	1.2	conv.	1.5	conv.
	3072	11.0	3.5_{-2}	32.6	1.2_{-2}	132.6	5.1_{-3}	183.2	2.6_{-3}
	12288	49.7	4.7_{-2}	185.3	1.9_{-2}	454.3	9.4_{-3}	1418.3	6.3_{-3}
	49152	206.5	5.7_{-2}	903.1	2.4_{-2}	1823.6	1.4_{-2}	7649.4	9.4_{-3}
	196608	804.8	6.3_{-2}	3945.2	2.7_{-2}	7374.3	1.8_{-2}	37011.2	1.3_{-2}

Table 2: Multiplying double and single layer potential on the unit cube

surface $\Gamma_C = \partial[-1,1]^3$ of the cube. Since it is not a smooth manifold, we expect that the smoothness of the result of the multiplication, and therefore the speed of convergence, will be reduced. This is indeed visible in the numerical experiments: we no longer observe a convergence like 10^{-m}, but only 2^{-m} for VV and KV and even slower convergence for VK and KK. The latter effect might be a consequence of the fact that the column cluster bases used for K involve the normal derivative of the kernel function, which reduces the order of smoothness even further.

The rank $k = m^3$ results from the polynomial expansion used for the approximation of the matrices V and K. Taking the special structure of the kernel functions into account, optimized cluster bases with significantly lower rank can be constructed by an efficient algorithm [3], which leads to an improved performance of both matrix-vector and matrix-matrix multiplication algorithms. Since the optimized cluster bases might not be suitable for the representation of the product, it can be necessary to construct adaptive cluster bases for the product a posteriori [7]. For Table 3, we have used optimized $\mathcal{H}^2$-matrix approximations of V and K, constructed by the algorithms given in [3] with an initial polynomial order of $m = 4$ and a truncation tolerance of $\epsilon = 10^{-3}$. Then we approximate the products VV, KV, VK and KK by four different algorithms:

- We compute them directly by Algorithm 10, using the corresponding optimized cluster bases instead of the polynomial bases.
- We compute $\mathcal{H}^2$-matrix approximations M of the products by a modified version of the algorithm presented in [7], prescribing an error tolerance of 10^{-4}. This new algorithm will be the subject of a forthcoming paper.
- We compute the products by Algorithm 10, but use the adapted cluster bases prepared for M in the previous algorithm.
- We compute the product by the standard $\mathcal{H}$-matrix multiplication algorithm [14], again prescribing an error tolerance of 10^{-4}.

We can see that using optimized cluster bases leads to significantly improved performance, but also that the optimized bases are not well suited for representing the product of matrices (the relative error is signicantly larger than in Table 1). Using optimized cluster bases, the relative error can be reduced to the desired level, but the construction of the new cluster bases takes more time than Algorithm 10. The $\mathcal{H}$-matrix multiplication algorithm is slower than all the other algorithms: it takes roughly 24 times longer than Algorithm 10 and 4 times longer than the adaptive algorithm.

We can also see that both the adaptive $\mathcal{H}^2$- and $\mathcal{H}$-matrix arithmetic algorithms indeed guarantee the prescribed accuracy of 10^{-4} for the product. When using the cluster bases from the adaptive $\mathcal{H}^2$-matrix algorithm in conjunction with Algorithm 10, its best-approximation property guarantees that the error of the latter is not larger than that of the former. We can see that this holds in practice by comparing the sixth and eighth column of Table 3.

The comparison of $\mathcal{H}$- and $\mathcal{H}^2$-matrices is not entirely fair: in the context of hierarchical matrices, a weaker admissibility condition and a weaker stopping criterion for the construction of the block cluster tree can be used, and by these advanced techniques [13] the performance of the $\mathcal{H}$-MMM can be significantly improved. This will narrow the gap between the $\mathcal{H}$- and $\mathcal{H}^2$-matrix algorithms, but the latter will still be faster.

References

[1] 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 (2003), pp. 1–28.

[2] S. Börm, *$\mathcal{H}^2$-matrices — multilevel methods for the approximation of integral operators*, Comput. Visual. Sci., 7 (2004), pp. 173–181.

[3] ——, *Approximation of integral operators by $\mathcal{H}^2$-matrices with adaptive bases*, Computing, 74 (2005), pp. 249–271.

[4] ——, *Data-sparse approximation of non-local operators by $\mathcal{H}^2$-matrices*, Preprint 44/2005, Max Planck Institute for Mathematics in the Sciences, 2005.

Oper.	n	A priori		Adaptive		A priori/new		$\mathcal{H}$-Arithmetic	
VV	768	0.7	conv.	0.7	conv.	0.7	conv.	0.7	conv.
	3072	32.1	3.1_{-3}	35.8	1.8_{-5}	32.6	1.1_{-5}	153.0	1.7_{-5}
	12288	70.6	3.5_{-3}	276.6	3.1_{-5}	72.3	3.0_{-5}	824.8	2.5_{-5}
	49152	235.4	3.6_{-3}	1343.7	4.1_{-5}	240.5	4.0_{-5}	6591.3	2.5_{-5}
	196608	807.8	3.6_{-3}	6513.3	4.7_{-5}	805.5	4.6_{-5}	29741.6	2.5_{-5}
KV	768	0.7	conv.	0.7	conv.	0.7	conv.	0.7	conv.
	3072	34.2	1.8_{-2}	37.6	2.7_{-5}	37.2	2.6_{-5}	152.9	5.5_{-5}
	12288	76.9	2.0_{-2}	270.3	3.5_{-5}	87.6	3.4_{-5}	755.5	5.4_{-5}
	49152	257.1	2.0_{-2}	1267.0	4.3_{-5}	291.6	4.2_{-5}	5688.3	5.6_{-5}
	196608	875.4	2.0_{-2}	5933.5	9.0_{-5}	989.9	9.0_{-5}	26404.6	5.5_{-5}
VK	768	0.7	conv.	0.7	conv.	0.7	conv.	0.7	conv.
	3072	39.0	1.3_{-1}	37.8	3.2_{-5}	40.2	3.1_{-5}	150.2	3.3_{-5}
	12288	89.5	2.0_{-1}	267.1	3.3_{-5}	94.4	3.3_{-5}	737.4	3.7_{-5}
	49152	291.9	2.3_{-1}	1219.5	4.5_{-5}	310.4	4.4_{-5}	5886.5	3.4_{-5}
	196608	995.9	2.5_{-1}	5734.5	8.0_{-5}	1067.0	7.8_{-5}	27303.6	3.6_{-5}
KK	768	0.7	conv.	0.7	conv.	0.7	conv.	0.7	conv.
	3072	41.1	1.3_{-2}	39.8	6.8_{-6}	44.0	6.6_{-6}	151.4	5.3_{-6}
	12288	93.0	2.2_{-2}	272.4	1.6_{-5}	100.3	1.6_{-5}	658.2	6.4_{-6}
	49152	300.6	2.7_{-2}	1327.9	3.2_{-5}	324.5	3.2_{-5}	5066.8	1.4_{-5}
	196608	1005.6	3.2_{-2}	5791.6	4.5_{-5}	1080.8	4.5_{-5}	24862.1	1.5_{-5}

Table 3: Adaptive and non-adaptive multiplication algorithms for the single and double layer potential on the cube

[5] S. Börm and L. Grasedyck, *Hybrid cross approximation of integral operators*, Preprint 68/2004, Max Planck Institute for Mathematics in the Sciences, 2004. To appear in Numerische Mathematik.

[6] S. Börm, L. Grasedyck, and W. Hackbusch, *Hierarchical Matrices*. Lecture Note 21 of the Max Planck Institute for Mathematics in the Sciences, 2003.

[7] S. Börm and W. Hackbusch, *Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices*, Computing, 69 (2002), pp. 1–35.

[8] ——, *$\mathcal{H}^2$-matrix approximation of integral operators by interpolation*, Applied Numerical Mathematics, 43 (2002), pp. 129–143.

[9] S. Börm, M. Löhndorf, and J. M. Melenk, *Approximation of integral operators by variable-order interpolation*, Numerische Mathematik, 99 (2005), pp. 605–643.

[10] S. Börm and S. A. Sauter, *BEM with linear complexity for the classical boundary integral operators*, Mathematics of Computation, 74 (2005), pp. 1139–1177.

[11] W. Dahmen and R. Schneider, *Wavelets on manifolds I: Construction and domain decomposition*, SIAM Journal of Mathematical Analysis, 31 (1999), pp. 184–230.

[12] L. Grasedyck, *Theorie und Anwendungen Hierarchischer Matrizen*, PhD thesis, Universität Kiel, 2001.

[13] ——, *Adaptive recompression of $\mathcal{H}$-matrices for BEM*, Computing, 74 (2004), pp. 205–223.

[14] L. Grasedyck and W. Hackbusch, *Construction and arithmetics of $\mathcal{H}$-matrices*, Computing, 70 (2003), pp. 295–334.

[15] L. Greengard and V. Rokhlin, *A fast algorithm for particle simulations*, Journal of Computational Physics, 73 (1987), pp. 325–348.

[16] ——, *A new version of the fast multipole method for the Laplace in three dimensions*, in Acta Numerica 1997, Cambridge University Press, 1997, pp. 229–269.

[17] W. Hackbusch, *A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: Introduction to $\mathcal{H}$-matrices*, Computing, 62 (1999), pp. 89–108.

[18] ——, *Hierarchische Matrizen — Algorithmen und Analysis*. Available online at `http://www.mis.mpg.de/scicomp/Fulltext/hmvorlesung.ps`, 2004.

[19] W. Hackbusch and B. Khoromskij, *A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part II: Application to multi-dimensional problems*, Computing, 64 (2000), pp. 21–47.

[20] W. Hackbusch, B. Khoromskij, and S. Sauter, *On $\mathcal{H}^2$-matrices*, in Lectures on Applied Mathematics, H. Bungartz, R. Hoppe, and C. Zenger, eds., Springer-Verlag, Berlin, 2000, pp. 9–29.

[21] W. Hackbusch and Z. P. Nowak, *On the fast matrix multiplication in the boundary element method by panel clustering*, Numerische Mathematik, 54 (1989), pp. 463–491.

[22] V. Rokhlin, *Rapid solution of integral equations of classical potential theory*, Journal of Computational Physics, 60 (1985), pp. 187–207.

[23] S. Sauter, *Variable order panel clustering (extended version)*, Preprint 52/1999, Max-Planck-Institut für Mathematik, Leipzig, Germany, 1999.

[24] ——, *Variable order panel clustering*, Computing, 64 (2000), pp. 223–261.

[25] J. Tausch and J. White, *Multiscale bases for the sparse representation of boundary integral operators on complex geometries*, SIAM J. Sci. Comput., 24 (2003), pp. 1610–1629.

Steffen Börm
Max-Planck-Institut für Mathematik in den Naturwissenschaften
Inselstrasse 22–26
04103 Leipzig
Germany
sbo@mis.mpg.de