Optimized M2L Kernels for the Chebyshev Interpolation based Fast Multipole Method

Matthias Messner, Berenger Bramas, Olivier Coulaud, Eric Darve

November 20, 2012

Abstract

A fast multipole method (FMM) for asymptotically smooth kernel functions ($1/r$, $1/r^4$, Gauss and Stokes kernels, radial basis functions, etc.) based on a Chebyshev interpolation scheme has been introduced in [5]. The method has been extended to oscillatory kernels (e.g., Helmholtz kernel) in [12]. Beside its generality this FMM turns out to be favorable due to its easy implementation and its high performance based on intensive use of highly optimized BLAS libraries. However, one of its bottlenecks is the precomputation of the multiple-to-local (M2L) operator, and its higher number of floating point operations (flops) compared to other FMM formulations. Here, we present several optimizations for that operator, which is known to be the costliest FMM operator. The most efficient ones do not only reduce the precomputation time by a factor up to 340 but they also speed up the matrix-vector product. We conclude with comparisons and numerical validations of all presented optimizations.

1 Introduction

The fast multipole method (FMM) is a method first designed in [7] to reduce the cost of matrix-vector products from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$ or $\mathcal{O}(N \log N)$ depending on the underlying kernel function. Most FMM variants have been developed and optimized for specific kernel functions [13, 8, 3, 2]. However, some have a formulation that is independent of the kernel function [14, 10, 6, 5, 12]. The paper at hand addresses the optimization of one of these formulations, the so called black-box FMM (**bbFMM**), presented in [5]. It is based on the approximation of the kernel function via a Chebyshev interpolation and is a black-box scheme for kernel functions that are asymptotically smooth, e.g., $1/(r^2 + c^2)^{n/2}$ with $r = |x-y|$, c a given parameter and $n \in \mathbb{N}$. The bbFMM has been extended to the directional FMM (**dFMM**) for oscillatory kernels in [12]. It is suitable for any kernel function of the type $g(r)e^{\imath kr}$ where $g(r)$ is an asymptotically smooth function ($\imath^2 = -1$ is the imaginary unit and k the wavenumber).

The main idea of the FMM is to properly separate near-field ($|x - y| \to 0$) and far-field ($|x - y| \to \infty$). The near-field is evaluated directly and the far-field can be approximated and thus computed efficiently. In this paper, we will denote M2L the operator that transforms a multipole expansion into a local expansion. The M2L operator is the costliest step in the method, since it needs to be applied to all clusters in the interaction list, that is 189 times for each cluster for a typical configuration in bbFMM. In this paper we focus on various optimizations of this operator for both the bbFMM and the dFMM.

First, we address the optimization proposed in [5, 12]. In that paper, the M2L operator between a cluster and its interaction list is further compressed using a singular value decomposition (SVD). The key idea is that the SVD provides the smallest possible rank given an error ε (therefore leading to the smallest computational cost). In [5], a single SVD is used to compress **all the M2L operators at a given level** using a single singular vector basis. However, the singular values of individual M2L operators decay at different rates; it is often the case for example that cluster pairs that are separated by a small distance have higher rank than pairs that are separated by a greater distance. If we denote w the diameter (or width) of a cluster and d the distance between two clusters, then the smallest distance corresponds to $d_{\min} = 2w$ while the largest is $d_{\max} = 3\sqrt{3}w$. The ratio $d_{\max}/d_{\min} = 3\sqrt{3}/2$ is in fact relatively large. This can be taken advantage of to further compress the M2L operator on a **cluster-pair basis** leading to an even smaller computational cost than the original algorithm [5]. This point is investigated in details in this paper.

Another bottleneck in the FMM of [5] is the precomputation time of the M2L operators. We therefore introduce a new set of optimizations that exploit symmetries in the arrangement of the M2L operators. For example, for bbFMM, this allows us to express all M2L operators ($316 = 7^3 - 3^3$) as permutations of a subset with only 16 unique M2L operators. These permutations correspond to reflections across various planes (for example $x = 0$, $y = 0$, $z = 0$, etc). Table 5 for example reports reductions in precomputation time by a factor of about 50–200x.

Besides drastically reducing precomputation time and memory requirement this approach paves also the road for various algorithmic improvements. Modern processors use memory cache to enhance performance and, as a result, schemes that block data (thereby reducing the amount of data traffic between main memory and the floating point units) can result in much higher performance. In our case, the use of symmetries allows calling highly optimized matrix-matrix product implementations (instead of matrix-vector) that have better cache reuse. Overall, this may result in acceleration by a factor 4–6x for smooth kernels (Tables 6–8). We also present results using oscillatory kernels of the type mentioned above ($g(r)e^{\imath kr}$). In that case, the acceleration is more modest and ranges from about 1.2 to 2.7x (Table 9–11).

In this paper we therefore focus both on performance and precomputation time. Both are important factors for the FMM: the precomputation is crucial if we are interested in only one matrix-vector product, while in other cases, such as the solution of a linear system, the fast application of matrix-vector products is central.

The paper is organized as follows. In Sec. 2, we briefly recall the bbFMM and the dFMM and introduce notations that are needed for explanations later in this paper. In Sec. 3, we address the separation of near- and far-field, introduce the notion of transfer vector to uniquely identify M2L operators, and describe how the kernel (smooth or oscillatory) affects the interaction list. We start Sec. 4 with a brief recall of the known optimization of the M2L operator (see [5, 12]) and suggest further improvements. Then, we present a new set of optimizations and explain them in details. These include using a low-rank approximation of M2L for individual interactions (individual cluster pairs), along with the definition of symmetries and permutations and how they are used to reduce the computational cost. Finally, in Sec. 5, we present numerical results. We compare all variants for bbFMM and dFMM with three datasets corresponding to a sphere, an oblate spheroid, and a prolate spheroid. The measured running time of the FMM, along with the precomputation time, and its accuracy as a function of various parameters are reported. The efficiency of the algorithms as a function of the target accuracy is considered. We also performed tests using two different kinds of linear algebra libraries for the matrix-matrix products and other operations (an implementation of BLAS and LAPACK vs the Intel MKL library).

2 Pairwise particle interactions

The problem statement reads as follows. Assume, the cells X and Y contain source $\{y_j\}_{j=1}^N$ and target particles $\{x_i\}_{i=1}^M$, respectively. Compute the interactions

$$f_i = \sum_{j=1}^{N} K(x_i, y_j) w_j \quad \text{for } i, \ldots, M. \tag{1}$$

The kernel function $K(x, y)$ describes the influence of the source particles onto the target particles. The cost of directly evaluating the summation in Eqn. (1) grows like $\mathcal{O}(MN)$ which becomes prohibitive as $M, N \to \infty$ and it is why we need a fast summation scheme.

2.1 Fast summation schemes based on Chebyshev interpolation

For a detailed derivation and error analysis of the FMM based on Chebyshev interpolation we refer the reader to [5, 12]. We adapt most of their notations and repeat only concepts which are necessary to understand explanations hereafter.

Let the function $f : \mathbb{R}^3 \to \mathbb{C}$ be approximated by a Chebyshev interpolation scheme as

$$f(x) \sim \sum_{|\alpha|\leq 3\ell} S_\ell(x, \bar{x}_\alpha) f(\bar{x}_\alpha) \tag{2}$$

with the 3-dimensional multi-index $\alpha = (\alpha_1, \alpha_2, \alpha_3)$ and $|\alpha| = \max(\alpha_1, \alpha_2, \alpha_3)$ with $\alpha_i \in (1, \ldots, \ell)$. The interpolation points $\bar{x}_\alpha = (\bar{x}_{\alpha_1}, \bar{x}_{\alpha_2}, \bar{x}_{\alpha_3})$ are given by the tensor-product of the Chebyshev roots $\bar{x}_{\alpha_i}$ of the Chebyshev polynomial of first kind $T_\ell(x) = \cos(\arccos x)$ with $x \in [-1, 1]$. The interpolation operator reads as

$$S_\ell(x, \bar{x}_\alpha) = S_\ell(x_1, \bar{x}_{\alpha_1})\, S_\ell(x_2, \bar{x}_{\alpha_2})\, S_\ell(x_3, \bar{x}_{\alpha_3}). \tag{3}$$

For the interpolation on arbitrary intervals, we need the affine mapping $\Phi : [-1, 1] \to [a, b]$. We omit it hereafter for the sake of readability.

2.1.1 Black–box FMM (bbFMM)

If two cells X and Y are well separated, we know from [5] that *asymptotically smooth kernel functions* can be interpolated as

$$K(x, y) \sim \sum_{|\alpha|\leq\ell} S_\ell(x, \bar{x}_\alpha) \sum_{|\beta|\leq\ell} K(\bar{x}_\alpha, \bar{y}_\beta)\; S_\ell(y, \bar{y}_\beta). \tag{4}$$

We insert the above approximation into Eqn. (1) and obtain

$$f_i \sim \sum_{|\alpha|\leq\ell} S_\ell(x_i, \bar{x}_\alpha) \sum_{|\beta|\leq\ell} K(\bar{x}_\alpha, \bar{y}_\beta) \sum_{j=1}^{N} S_\ell(y_j, \bar{y}_\beta)\; w_j \tag{5}$$

which we split up in a three-stage fast summation scheme.

1. Particle to moment (P2M) or moment to moment (M2M) operator: equivalent source values are anterpolated at the interpolation points $\bar{y}_\beta \in Y$ by

$$W_\beta = \sum_{j=1}^{N} S(y_j, \bar{y}_\beta)\, w_j \quad \text{for} \quad |\beta| \leq \ell. \tag{6}$$

2. Moment to local operator (M2L): target values are evaluated at the interpolation points $\bar{x}_\alpha \in X$ by

$$F_\alpha = \sum_{|\beta|\leq\ell} K(\bar{x}_\alpha, \bar{y}_\beta)\, W_\beta \quad \text{for} \quad |\alpha| \leq \ell. \tag{7}$$

3. Local to local (L2L) or local to particle (L2P) operator: target values are interpolated at final points $x_i \in X$ by

$$f_i \sim \sum_{|\alpha|\leq\ell} S(x_i, \bar{x}_\alpha)\, F_\alpha \quad \text{for} \quad i = 1, \ldots, M. \tag{8}$$

Recall, the cells X and Y are well separated and thus all contributions of f_i can be computed via the above presented fast summation scheme (no direct summation is necessary).

2.1.2 Directional FMM (dFMM)

Whenever we deal with *oscillatory kernel functions*, such as the Helmholtz kernel, the wavenumber k comes into play. Depending on the diameter of the cells X and Y and the wavenumber they are either in the low-frequency or in the high-frequency regime. In the low-frequency regime the fast summation schemes of the dFMM and the bbFMM are the same. In the high-frequency regime the fast summation scheme becomes directional. From [12] we know that any oscillatory kernel function $K(x,y) = G(x,y)e^{\imath k|x-y|}$, where $G(x,y)$ is an asymptotically smooth function, can be rewritten as

$$K(x,y) = K^u(x,y)e^{\imath ku\cdot(x-y)} \quad \text{with} \quad K^u(x,y) = G(x,y)e^{\imath k(|x-y|-u\cdot(x-y))}. \tag{9}$$

We assume that the cells X and Y of width w are centered at c_x and c_y and c_y lies in a cone of direction u being centered at c_x (think of the domain around X being virtually subdivided in cones given by directional unit vectors $\{u_c\}_{c=1}^C$, where C is determined by their aperture). If the cell pair (X,Y) satisfies the *separation* criterion $\mathcal{O}(kw^2)$ and the *cone-aperture* criterion $\mathcal{O}(1/kw)$, the error of the Chebyshev interpolation of the kernel function

$$K^u(x,y) \sim \sum_{|\alpha|\le\ell} S_\ell(x,\bar{x}_\alpha) \sum_{|\beta|\le\ell} K^u(\bar{x}_\alpha,\bar{y}_\beta)\, S_\ell(y,\bar{y}_\beta) \tag{10}$$

decays exponentially in the interpolation order ℓ (independent of the wavenumber k; see [11, 12]). We insert the above interpolated kernel function in Eqn. (1) and obtain

$$f_i \sim e^{\imath ku\cdot x_i} \sum_{|\alpha|\le\ell} S_\ell(x_i,\bar{x}_\alpha)e^{-\imath ku\cdot\bar{x}_\alpha} \sum_{|\beta|\le\ell} K(\bar{x}_\alpha,\bar{y}_\beta)e^{\imath ku\cdot\bar{y}_\beta} \sum_{j=1}^{N} S_\ell(y_j,\bar{y}_\beta)e^{-\imath ku\cdot y_j}\, w_j \tag{11}$$

for all $i = 1,\dots,M$. Similarly as with the bbFMM, a three-stage fast summation scheme for oscillatory kernels in the high-frequency regime can be constructed.

1. Particle to moment (P2M) or moment to moment (M2M) operator: equivalent source values are anterpolated at the interpolation points $\bar{y}_\beta \in Y$ by

$$W_\beta^u = e^{\imath ku\cdot\bar{y}_\beta} \sum_{j=1}^{N} S(y_j,\bar{y}_\beta)\, e^{-\imath ku\cdot y_j}\, w_j \quad \text{for} \quad |\beta| \le \ell. \tag{12}$$

2. Moment to local operator (M2L): target values are evaluated at the interpolation points $\bar{x}_\alpha \in X$ by

$$F_\alpha^u = \sum_{|\beta|\le\ell} K(\bar{x}_\alpha,\bar{y}_\beta)\, W_\beta^u \quad \text{for} \quad |\alpha| \le \ell. \tag{13}$$

3. Local to local (L2L) or local to particle (L2P) operator: target values are interpolated at final points $x_i \in X$ by

$$f_i \sim e^{\imath ku\cdot x_i} \sum_{|\alpha|\le\ell} S(x_i,\bar{x}_\alpha)\, e^{-\imath ku\cdot\bar{x}_\alpha}\, F_\alpha^u \quad \text{for} \quad i = 1,\dots,M. \tag{14}$$

Even though the bbFMM and the dFMM are here presented as single-level schemes, they are usually implemented as multilevel schemes. Strictly speaking, the steps one and three of both schemes are the P2M and L2P operators. Let us recall briefly on the directional M2M and L2L operators of dFMM: based on the criterion $\mathcal{O}(1/kw)$, the aperture of the cones at the upper level is about half the aperture at the lower level. Due to a nested cone construction along octree levels, we preserve the accuracy of the Chebyshev interpolation within the multilevel scheme. For a detailed description of all operators we refer to [5, 12]). Note, the similarity of the M2L operators (step two of both schemes) of bbFMM and dFMM. In fact, the only difference in their implementation is that in the bbFMM case we have one loop over all cell pairs, whereas in the dFMM case we have two loops: the outer loop over all existing cones of direction $\{u_c\}_{c=1}^C$ and the inner loop over all cell pairs lying in the current cone. In this paper, we focus on the M2L operators and their efficient numerical treatment.

3 M2L operators

The first step of any FMM consists in a proper separation of near- and far-field. After that, the near-field is evaluated directly and the far-field efficiently using a fast summation scheme. In this section, we focus on the first step. The union of near- and far-field of a target cell X is spatially restricted to the near-field of its parent cell. Algorithm 1 explains how these interactions are computed for dFMM [12]. The recursive partitioning starts with the two root cells X and Y of the octrees for source and target particles. If a pair of cells satisfies the separation criterion in the high- or low-frequency regime, Y is a far-field interaction of X. Else, if they are at the leaf level of the octree, Y is a near-field interaction of X. If none is true, the cell is subdivided and the tests are repeated recursively. In line 3 in Alg. 1 we use the term directional far-field,

Algorithm 1 Separate near- and far-field in the low- and high-frequency regime

1: **function** SeparateNearAndFarField(X, Y)
2:     **if** (X, Y) are admissible in the high-frequency regime **then**
3:         add Y to the directional far-field of X **return**
4:     **else if** (X, Y) are admissible in the low-frequency regime **then**
5:         add Y to the far-field of X **return**
6:     **else if** (X, Y) are leaves **then**
7:         add Y to the near-field of X **return**
8:     **else**
9:         **for** all $X_{\text{child}} \in X$ and all $Y_{\text{child}} \in Y$ **do**
10:             SeparateNearAndFarField($X_{\text{child}}, Y_{\text{child}}$)
11:         **end for**
12:     **end if**
13: **end function**

a concept explained in detail in [12]: In the high-frequency regime the far-field is subdivided into cones of direction u needed by the directional kernel function $K^u(x, y)$. Each source cell Y is assigned to a cone and there are as many directional far-fields as there are cones.

3.1 Transfer vectors

In order to address interactions uniquely we introduce transfer vectors $t = (t_1, t_2, t_3)$ with $t \in \mathbb{Z}^3$. They describe the relative positioning of two cells X and Y and are computed as $t = {}^{(c_x - c_y)}/_{w}$ where c_x and c_y denote the centers of the cells X and Y and w their width. In the following we use transfer vectors to uniquely identify the M2L operator that computes the interaction between a target cell X and a source cell Y. In matrix notation an M2L operator reads as K_t of size $\ell^3 \times \ell^3$ and the entries are computed as

$$(\mathsf{K}_t)_{m(\alpha)n(\beta)} = K(\bar{x}_\alpha, \bar{y}_\beta) \tag{15}$$

with the interpolation points $\bar{x}_\alpha \in X$ and $\bar{y}_\beta \in Y$. Bijective mappings

$$m, n : \{1, \ldots, \ell\} \times \{1, \ldots, \ell\} \times \{1, \ldots, \ell\} \to \{1, \ldots, \ell^3\} \tag{16}$$

with $m^{-1}(m(\alpha)) = \alpha$ and $n^{-1}(n(\beta)) = \beta$ provide unique indices to map back and forth between multi-indices of $\bar{x}_\alpha$ and $\bar{y}_\beta$ and rows and columns of K_t. We choose them to be $m(\alpha) = (\alpha_1 - 1) + (\alpha_2 - 1)\ell + (\alpha_3 - 1)\ell^2 + 1$ and same for $n(\beta)$.

3.2 Interaction lists

Very commonly fast multipole methods are used for translation invariant kernel functions $K(x, y) = K(x + v, y + v)$ for any $v \in \mathbb{R}^3$. Because of that and because of the regular arrangement of interpolation points $\bar{x}$ and $\bar{y}$ in uniform octrees it is sufficient to identify unique transfer vectors at each level of the octree and to compute the respective M2L operators. In the following we refer to such sets of unique transfer vectors as interaction lists $T \subset \mathbb{Z}^3$.

If we consider *asymptotically smooth kernel functions* the near-field is limited to transfer vectors satisfying $|t| \leq \sqrt{3}$; it leads to $3^3 = 27$ near-field interactions (see [5]). In a multi-level scheme, these 27 near-field interactions contain all $6^3 = 216$ near- and far-field interactions of its eight child-cells. Far-field interactions are given by transfer-vectors that satisfy $|t| > \sqrt{3}$. This leads to a maximum of $6^3 - 3^2 = 189$ interactions per cell and we end up with the usual overall complexity of $\mathcal{O}(N)$ of fast multipole methods for asymptotically smooth kernel functions. The union of all possible far-field interactions of the eight child cells gives $7^3 - 3^3 = 316$ interactions. That is also the largest possible number of different M2L operators have to be computed per octree level. Most asymptotically smooth kernel functions happen also to be homogeneous $K(\alpha r) = \alpha^n K(r)$ of degree n. In other words, if we scale the distance $r = |x - y|$ between source and target by a factor of α the resulting potential is scaled by α^n, where n is a constant that depends on the kernel function. The advantage of homogeneous kernel functions is that the M2L operators need to be precomputed only at one level and can simply be scaled and used on other levels. This affects the precomputation time and the required memory.

If we consider *oscillatory kernel functions* we need to distinguish between the low- and high-frequency regime [12]. The admissibility criteria in the low-frequency regime are the same as those for asymptotically smooth kernel functions. However, in the high-frequency regime the threshold distance between near-field and far-field is $\mathcal{O}(kw^2)$; nonetheless, as shown in [12] we end up with the usual complexity of $\mathcal{O}(N \log N)$ of fast multipole methods for oscillatory kernel functions. It depends on the wavenumber k (a property of the kernel function). Thus, the size of near- and far-field is not a priori known as it is in the case of asymptotically smooth kernel functions. Table 1 summarizes the number of near and far-field interactions to be computed depending on different kernel functions.

Table 1: Number of near- and far-field interactions depends on the kernel function

type of kernel function	cells in near-field	cells in far-field
smooth	≤ 27 per leaf	≤ 316 per level
smooth and homogeneous	≤ 27 per leaf	≤ 316 for all levels
oscillatory	depends on k	depends on k

4 Optimizing the M2L operators

In all fast multipole methods the M2L operator adds the largest contribution to the overall computational cost: for bbFMM it grows like $\mathcal{O}(N)$ and for dFMM like $\mathcal{O}(N \log N)$. In this section, we first briefly recall the optimization that was used up to now and suggest an improvement. Then, we present a new set of optimizations that exploit the symmetries in the arrangement of the M2L operators.

4.1 Single low-rank approximation (SA)

In the following we explain the basics of the optimization used in [5, 12] and we refer to it as the SA variant hereafter. The idea is based on the fact that all M2L operators K_t with $t \in T$ can be assembled as a single big matrix in two ways: either as a row of matrices $\mathsf{K}^{(\text{row})} = [\mathsf{K}_1, \dots, \mathsf{K}_t, \dots, \mathsf{K}_{|T|}]$ or as a column of matrices $\mathsf{K}^{(\text{col})} = [\mathsf{K}_1; \dots; \mathsf{K}_t; \dots; \mathsf{K}_{|T|}]$ of M2L operators. The cardinality $|T|$ gives the number of transfer vectors in the interaction list T. Next, both big matrices are compressed using truncated singular value decompositions (SVD) of accuracy ε as

$$\mathsf{K}^{(\text{row})} \sim \mathsf{U\Sigma V}^* \quad \text{and} \quad \mathsf{K}^{(\text{col})} \sim \mathsf{A\Gamma B}^* \tag{17}$$

with the unitary matrices U, B of size $\ell \times r$ and V, A of size $|T|\ell^3 \times r$ and the r singular values in $\mathsf{\Sigma}, \mathsf{\Gamma}$. With a few algebraic transformations each M2L operator can be expressed as

$$\mathsf{K}_t \sim \mathsf{U C}_t \mathsf{B}^* \quad \text{where} \quad \mathsf{C}_t = \mathsf{U}^* \mathsf{K}_t \mathsf{B} \quad \text{of size } r \times r \text{ is computed as} \quad \mathsf{C}_t = \mathsf{\Sigma V}_t^* \mathsf{B} \quad \text{or} \quad \mathsf{C}_t = \mathsf{U}^* \mathsf{A}_t \mathsf{\Gamma}. \tag{18}$$

The advantage of this representation is that the cost of applying the M2L operator gets reduced from applying a matrix of size $\ell^3 \times \ell^3$ to a matrix of only $r \times r$. Moreover, less memory is required. However, the precomputation time grows cubically with the accuracy of the method due to the complexity of the SVD. In [12] the SVD has been substituted by the adaptive cross approximation (ACA) followed by a truncated SVD [1]. The precomputation time has been cut down drastically due to the linear complexity of the ACA.

4.1.1 SA with recompression (SArcmp)

If we use the SA variant the achieved low-rank r is the same for all M2L operators given by C_t. To a large extent, r is determined by the greatest individual low-rank of the M2L operators. This means that most of the matrices C_t of size $r \times r$ have effectively a smaller low-rank $r_t \leq r$. We exploit this fact by individually approximating them as

$$\mathsf{C}_t \sim \bar{\mathsf{U}}_t \bar{\mathsf{V}}_t^* \quad \text{with } \bar{\mathsf{U}}_t, \bar{\mathsf{V}}_t \text{ of size } r \times r_t \text{ and the constraint} \quad r_t < r/2. \tag{19}$$

Without the constraint the low-rank representation is less efficient than the original representation. The effects of the recompression are studied in Sec. 5.2.

4.2 Individual low-rank approximation (IA)

As opposed to the SA approach, an individual low-rank approximation of the M2L operators as

$$\mathsf{K}_t \sim \mathsf{U}_t \mathsf{V}_t^* \quad \text{with } \mathsf{U}_t \mathsf{V}_t \text{ of size } \ell^3 \times r_t \tag{20}$$

directly leads to the optimal low-rank representation of each of them. As in the previous section, the approximation can be performed by either a truncated SVD or the ACA followed by a truncated SVD (note, the rank r_t in the Eqns. (19) and (20) might be similar but has not to be the same). All these variants (SA, SArcmp and IA) still require the approximation and storage of all M2L operators. In terms of time and memory however, it would be desirable to come up with a method that requires the approximation and the storage of a subset of operators only. Let us present a set of optimizations that fulfill these two requests.

4.2.1 Symmetries and permutations

Here, we illustrate how the full set of M2L operators can be expressed by a subset only. The idea is based on symmetries in the arrangement of M2L operators and exploits the uniform distribution of the interpolation points. We start by presenting the idea using a model example. After generalizing this idea, we demonstrate that M2L operators can be expressed as permutations of others.

Model example The target cell X in Fig. 1 has three interactions Y_t with the transfer vectors $t \in \{(2,1),(1,2),(-2,1)\}$. We choose the reference domain to be given by $t_1 \geq t_2 \geq 0$. The goal is to express the M2L operators of all interactions via M2L operators of interactions that lie in the reference domain only. In our example this is the interaction with the transfer vector $(2,1)$. The two transfer vectors $(1,2)$ and $(-2,1)$ can be shown to be reflections along the lines given by $t_1 = 0$ and $t_1 = t_2$, respectively. We easily find that any $t \in \mathbb{Z}^2$ can be expressed as a reflection (or a combination of reflections) of transfer vectors that satisfy $t_1 \geq t_2 \geq 0$.

We claim that any reflection of a transfer vector corresponds to a permutation of the respective M2L operator. Recall that the evaluation of $K(\bar{x}_\alpha, \bar{y}_\beta)$ gives the entry from row $m(\alpha)$ and column $n(\beta)$ of the M2L operator. $\mathsf{K}_{(2,1)}$ is the only M2L operator whose transfer vector satisfies $t_1 \geq t_2 \geq 0$. The multi-indices are constructed as presented in Fig. 1. As can be checked, the entry $(\mathsf{K}_{(2,1)})_{mn}$ is not the same as $(\mathsf{K}_{(1,2)})_{mn}$ or $(\mathsf{K}_{(-2,1)})_{mn}$. However, if we use the permuted multi-indices from Fig. 2a for $\mathsf{K}_{(-2,1)}$ or those from Fig. 2b for $\mathsf{K}_{(1,2)}$ they are the same. The logic behind this can be summarized as follows.

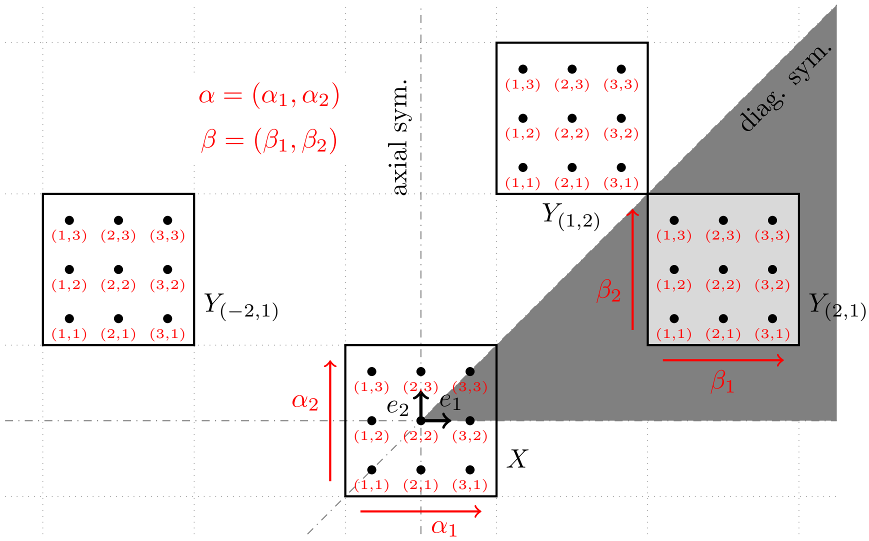


Figure 1: Axial and diagonal symmetries of interactions. The interpolation points $\bar{x}_\alpha$ and $\bar{y}_\beta$ are indexed by the multi-indices α and β, respectively (interpolation order $\ell = 3$). The only transfer vector that satisfies $t_1 \geq t_2 \geq 0$ is $t = (2, 1)$. In that case, we claim that $\mathsf{K}_{(2,1)}$ is the only M2L operator we need to compute. The direction of the arrows indicates the growth of the respective multi-index component.

- If an *axial symmetry* is given by $t_1 = 0$ as shown in Fig. 2a, we invert the corresponding component of the multi-index as
$$\alpha \leftarrow (\ell - (\alpha_1 - 1), \alpha_2) \quad \text{and} \quad \beta \leftarrow (\ell - (\beta_1 - 1), \beta_2). \tag{21}$$

- If the *diagonal symmetry* is given by $t_1 = t_2$ as shown in Fig. 2b, we swap the corresponding components as
$$\alpha \leftarrow (\alpha_2, \alpha_1) \quad \text{and} \quad \beta \leftarrow (\beta_2, \beta_1). \tag{22}$$

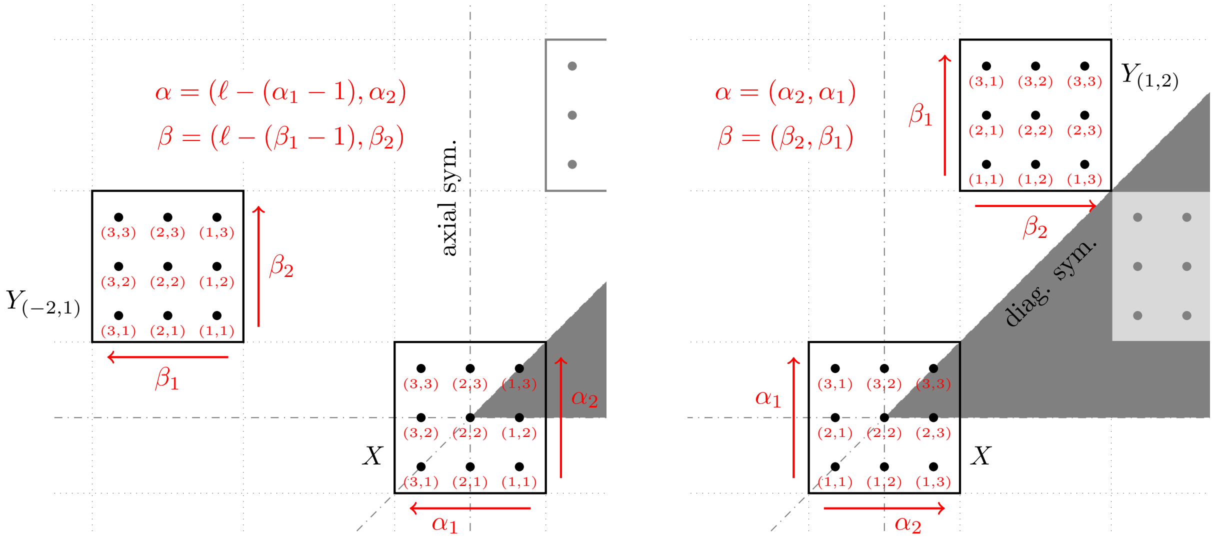
Sometimes it is necessary to combine axial and diagonal permutations. Take as example the transfer vector $(-1, 2)$: we need to flip it along $t_1 = 0$ and then along $t_1 = t_2$ to get $(2, 1)$. Note that reflections are non commutative, i.e., the order of their application matters. This is also true for permutations of the M2L operators.

Generalization Let us extend the above concept to the three dimensional case. We start by introducing three axial and three diagonal symmetries in $\mathbb{Z}^3$.

- *Axial symmetry planes* are given by $t_1 = 0$, $t_2 = 0$ and $t_3 = 0$ (see Fig. 3). Each of the three planes divides $\mathbb{Z}^3$ in two parts, i.e., the negative part $t_i < 0$ and the positive part $t_i \geq 0$. By combining all three planes $\mathbb{Z}^3$ is divided into octants. In the following we use $\mathbb{Z}^3_+$, i.e., the octant with $t_1, t_2, t_3 \geq 0$ as reference octant.

- *Diagonal symmetry planes* are given by $t_1 = t_2$, $t_1 = t_3$ and $t_2 = t_3$ (see Fig. 4). In $\mathbb{Z}^3$ there are six diagonal symmetries; however, we restrict ourselves to the symmetries affecting the reference octant $\mathbb{Z}^3_+$.

By combining the three diagonal symmetries and the three axial symmetries we obtain the cone shown in Fig. 5. We refer to it as $\mathbb{Z}^3_{\text{sym}}$; it is given by
$$\mathbb{Z}^3_{\text{sym}} = \left\{ \mathbb{Z}^3_{\text{sym}} \subset \mathbb{Z}^3 : t_1 \geq t_2 \geq t_3 \geq 0 \text{ with } t \in \mathbb{Z}^3 \right\}. \tag{23}$$



(a) Invert the component α_1 and β_1 due to the axial symmetry $t_1 = 0$.

(b) Swap the components $\alpha_1 \leftrightarrow \alpha_2$ and $\beta_1 \leftrightarrow \beta_2$ due to the diagonal symmetry $t_1 = t_2$.

Figure 2: The direction of the arrows indicates the growth of the respective multi-index component such that the M2L operators $\mathsf{K}_{(-2,1)}$ and $\mathsf{K}_{(1,2)}$ become the same as $\mathsf{K}_{(2,1)}$. In other words, the mapping from the arrows in Fig. 1 to the arrows here is analog to the mapping of the multi-indices and corresponds to the permutations of $\mathsf{K}_{(2,1)}$ such that it coincides with $\mathsf{K}_{(-2,1)}$ and $\mathsf{K}_{(1,2)}$.

By its means we can identify the subset of transfer vectors $T_{\text{sym}} \subset T \subset \mathbb{Z}^3$ as

$$T_{\text{sym}} = T \cap \mathbb{Z}^3_{\text{sym}} \tag{24}$$

such that all others $T \backslash T_{\text{sym}}$ can be expressed as reflections of this fundamental set. Next, we claim that these symmetries are also useful for M2L operators.

Permutation matrices Any reflection of a transfer vector along a symmetry plane determines the permutation of its associated M2L operator as

$$\mathsf{K}_t = \mathsf{P}_t \mathsf{K}_{p(t)} \mathsf{P}_t^\top. \tag{25}$$

The permutation matrix P_t depends on the transfer vector $t \in T$. We also need the surjective mapping $p : T \to T_{\text{sym}}$; it associates every transfer vector in T to exactly one in T_{sym}. The left application of P_t, essentially, corresponds to the permutation of α and its right application to the permutation of β, affecting rows (respectively columns) of the original matrix $\mathsf{K}_{p(t)}$. Note, the permutation matrices P_t depend only on the transfer vector t. How do we construct them? For some t we introduce axial and diagonal permutations π_t^{A} and π_t^{D} that read as follows.

- *Axial symmetries*: multi-index permutations are computed as

$$\pi_t^{\mathrm{A}}(\alpha_1, \alpha_2, \alpha_3) = (\bar{\alpha}_1, \bar{\alpha}_2, \bar{\alpha}_3) \quad \text{with} \quad \bar{\alpha}_i = \begin{cases} \alpha_i & \text{if } t_i \geq 0, \\ \ell - (\alpha_i - 1) & \text{else.} \end{cases} \tag{26}$$

  There exist 8 different possibilities that correspond to the octants presented in Fig. 3. Note, $\pi_t^A(\alpha) = \alpha$ is only true for transfer vectors with $t_1, t_2, t_3 \geq 0$.

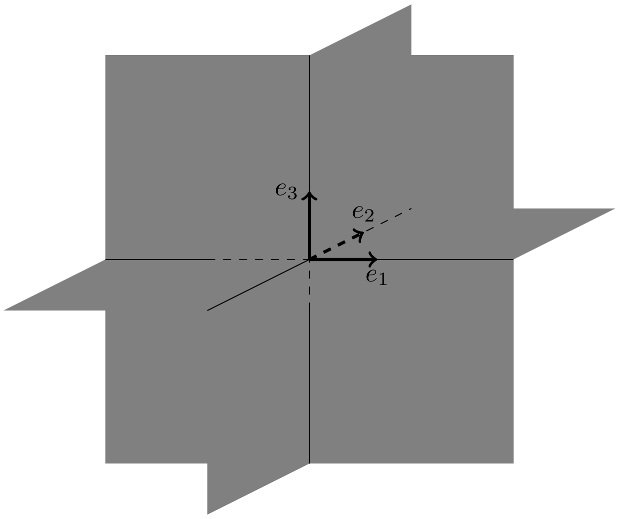
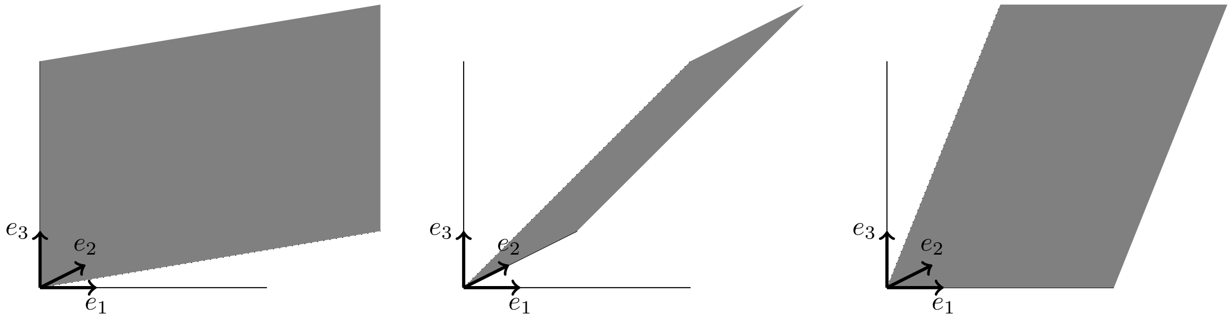


Figure 3: Three axial symmetry planes split $\mathbb{Z}^3$ in octants. The reference octant is given by $t_1, t_2, t_3 \geq 0$.



(a) $t_1 = t_2$

(b) $t_1 = t_3$

(c) $t_2 = t_3$

Figure 4: Three diagonal symmetry planes in the reference octant.

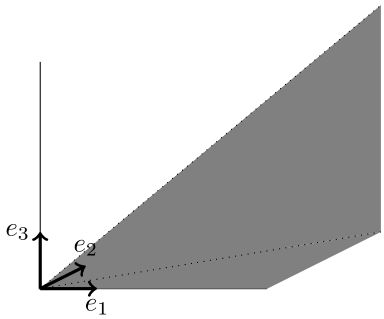


Figure 5: The final cone $\mathbb{Z}^3_{\text{sym}}$ $(t_1 \geq t_2 \geq t_3 \geq 0)$ is obtained by combining axial and diagonal symmetries.

- *Diagonal symmetries*: multi-index permutations are computed as

$$\pi_t^{\mathrm{D}}(\alpha_1, \alpha_2, \alpha_3) = (\alpha_i, \alpha_j, \alpha_k) \quad \text{such that} \quad |t_i| \geq |t_j| \geq |t_k|. \tag{27}$$

  There exist 6 different possibilities that correspond to the 6 different cones if we consider Fig. 5. Note again, $\pi_t^{\mathrm{D}}(\alpha) = \alpha$ is only true for transfer vectors with $t_1 \geq t_2 \geq t_3 \geq 0$.

Given these multi-index permutations and the mapping functions $m(\alpha)$ and $n(\beta)$ we can define a permutation matrix P_t of size $\ell^3 \times \ell^3$. Its entries are 0 except in column j the entry $i = m(\pi_t(m^{-1}(j)))$ is 1. Let us go through the computation of this index: first, we compute the multi-index $\alpha = m^{-1}(j)$, then, we permute the multi-index $\bar{\alpha} = \pi_t(\alpha)$ and last, we compute the row-index $i = m(\bar{\alpha})$. Permutation matrices may be written as

$$\mathsf{P}_t = \left(e_{m(\pi_t(m^{-1}(0)))}, e_{m(\pi_t(m^{-1}(1)))}, \ldots, e_{m(\pi_t(m^{-1}((\ell-1)^3)))}\right), \tag{28}$$

where e_j denotes a column unit vector of length ℓ^3 with 1 in the jth position and 0 elsewhere. Permutation matrices are orthogonal $\mathsf{P}_t \mathsf{P}_t^\top = \mathsf{I}$, hence, the inverse exists and can be written as $\mathsf{P}_t^{-1} = \mathsf{P}_t^\top$. Note that the combination of permutations is non commutative. Given these permutations π_t^{A} and π_t^{D} we setup $\mathsf{P}_t^{\mathrm{A}}$ and $\mathsf{P}_t^{\mathrm{D}}$ and construct the permutation matrix as

$$\mathsf{P}_t = \mathsf{P}_t^{\mathrm{D}}\, \mathsf{P}_t^{\mathrm{A}}. \tag{29}$$

The permutation for the multi-index β is the same.

4.2.2 IA with symmetries (IAsym)

By exploiting the above introduced symmetries we end up with an optimization we refer to as the IAsym variant. We individually approximate and store only M2L operators with $t \in T_{\mathrm{sym}}$ and express all others via permutations as shown Eqn. (25). The IAsym variant for an arbitrary transfer vector $t \in T$ consists of the following three steps.

1. Permute multipole expansions
$$\mathsf{w}_t = \mathsf{P}_t^\top \mathsf{w} \tag{30}$$
2. Compute permuted local expansions
$$\mathsf{f}_t = \mathsf{K}_{p(t)} \mathsf{w}_t \tag{31}$$
3. Un-permute local expansions
$$\mathsf{f} = \mathsf{P}_t \mathsf{f}_t \tag{32}$$

Note that the permutation matrix is not applied to the actual M2L operator (remains unchanged as can be seen in step 2). Its application is implemented as a reordering of vector entries (step 1 and 3). Depending on whether the M2L operator exist in its full-rank or in its low-rank representation (see Eqn. (20)) the application corresponds to one or two matrix-vector products. In the following we introduce a blocking scheme that leads to a faster execution on a computer.

4.2.3 IAsym with blocking (IAblk)

We know from Sec. 4.2.1 that based on the consideration of symmetries and permutations, many interactions share the same M2L operators. This paves the road for blocking schemes. Essentially, the idea is to substitute many matrix-vector products by a few matrix-matrix products. Blocking schemes do not change the overall complexity of the algorithm, but they allow for the use of highly optimized matrix-matrix product implementations. Such achieve much higher peak performances than optimized matrix-vector product implementations due to better cache reuse and less memory traffic [4, 9].

In our concrete case, we use the blocking scheme to block multipole and local expansions. Instead of permuting them and applying the M2L operators individually (matrix-vector products), we assemble

Algorithm 2 Blocking scheme with $|T_{\text{sym}}|$ matrix-matrix products

function BlockedM2L(target cell X and all far-field interactions I_Y)
    allocate F_p and W_p for $p = 1, \dots, |T|_{\text{sym}}$
    retrieve f from X
    set all $c_p = 0$
    for all source cells Y in I_Y **do**
        retrieve w from Y and compute t from cell-pair (X, Y)
        column $c_{p(t)}$ of $\mathsf{W}_{p(t)}$ gets $\mathsf{P}_t^\top \mathsf{w}$ ▷ Permute multipole expansions
        increment $c_{p(t)}$
    end for
    for all $\{\mathsf{K}_p\}$ **do**
        $\mathsf{F}_p \leftarrow \mathsf{K}_p \mathsf{W}_p$ ▷ Compute permuted local expansions
    end for
    set all $c_p = 0$
    for all source cells Y in I_Y **do**
        compute t from cell-pair (X, Y)
        retrieve f_t from column $c_{p(t)}$ of $\mathsf{F}_{p(t)}$
        increment $c_{p(t)}$
        $\mathsf{f} \leftarrow \mathsf{f} + \mathsf{P}_t \mathsf{f}_t$ ▷ Permute permuted local expansions
    end for
end function

those that share the same M2L operator as distinct matrices to whom we apply the M2L operators then (matrix-matrix products). Algorithm 2 explains this in details. We need the matrices W_p and F_p of size $\ell^2 \times n_p$ for $p = 1, \dots, |T_{\text{sym}}|$. Their columns store the permuted multipole and the resulting (also permuted) local expansions. The values for n_p indicate how many interactions in T share the same M2L operator of interactions in T_{sym}, in other words, $n_1 + \cdots + n_p + \cdots + n_{|T_{\text{sym}}|} = |T|$ is true. In the case of bbFMM this is a priori known, since the full interaction list is a priori given (see Sec. 3.2). That is not the case for dFMM and the values for n_p have to be determined during a precomputation step. We also need counters c_p to indicate the position of the currently processed expansions in W_p and F_p. As opposed to IAsym, here we split up the single loop over all interactions into three loops. In the first one, we assemble the set of matrices W_p. At the end $c_p \le n_p$ is true for all p. In the second loop, we perform at most $|T_{\text{sym}}|$ matrix-matrix products. And in the last loop, we increment the local expansion with the expansions from all F_p.

Blocking along multiple target cells Algorithm 2 proposes to use the blocking scheme for all interactions of only *one* target cell. In the worst case no M2L operator is shared and the algorithm coincides with IAsym. Moreover, the size of the matrices W_p and F_p might vary to a large extent. That is why we pursued the blocking idea further to come up with a more efficient scheme. Instead of using individual n_p we choose it to be a constant n for all $p = 1, \dots, |T_{\text{sym}}|$. Then we keep on blocking expansions using interactions lists of *multiple* (as opposed to one) target cells. Once $c_p = n$ is true for some p, we apply the M2L operator as $\mathsf{F}_p = \mathsf{K}_p \mathsf{W}_p$ where $\mathsf{W}_p, \mathsf{F}_p$ are both of size $\ell^3 \times n$. In our numerical studies we use this blocking scheme with $n = 128$.

5 Numerical results

In the previous sections we introduced various optimizations of the M2L operators for bbFMM and dFMM. As representative kernel functions we use

$$\text{the Laplace kernel} \quad K(x,y) = \frac{1}{4\pi|x-y|} \quad \text{and the Helmholtz kernel} \quad K(x,y) = \frac{e^{\imath k|x-y|}}{4\pi|x-y|}. \tag{33}$$

In the numerical studies, hereafter, we use the same parameter setting as are used in the respective publications and for dFMM we use the wavenumber $k = 1$. All computations are performed on a single CPU of a Intel Core i7–2760QM CPU @ 2.40GHz × 8 with 8GB shared memory. We used the compiler gcc 4.6.3 with the flags "-O2 -ffast-math". The source files (C++) of the program we used can be downloaded via `http://github.com/burgerbua/dfmm`; they are distributed under the "BSD 2-Clause" license.

M2L optimizations We show and analyze results for the following eight variants:

1. NA: the full interaction list T is represented by full-rank (not approximated) M2L operators
2. NAsym: the reduced interaction list T_{sym} is represented by full-rank (not approximated) M2L operators
3. NAblk: same as NAsym but with additional blocking of multipole and local expansions
4. SA: the variant presented in [5] and briefly sketched in Sec. 4.1
5. SArcmp: same as SA but with additional recompression of all C_t
6. IA: same as NA but with low-rank (individually approximated) M2L operators
7. IAsym: same as NAsym but with low-rank (individually approximated) M2L operators
8. IAblk: same as NAblk but with low-rank (individually approximated) M2L operators (see Alg. 2)

Moreover, we study two different low-rank approximation schemes for the SA and IA variants: on one hand we use a truncated singular value decomposition (SVD) and on the other hand the adaptive cross approximation (ACA) followed by a truncated SVD [1].

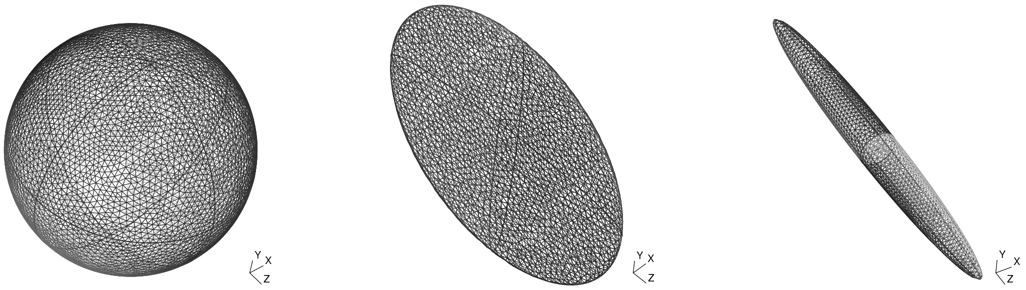
Example geometries We use the three benchmark examples shown in Fig. 6 and described in the listing below. The depth of the octree is chosen such that near- and far-field are balanced, i.e., we want the fastest possible matrix-vector product.

1. The *sphere* from Fig. 6a is contained in the bounding-box $64 \times 64 \times 64$; 168 931 particles are randomly scattered on its surface. The octree has 6 levels.
2. The *oblate sphere* from Fig. 6b is contained in the bounding-box $6.4 \times 64 \times 64$; 125 931 particles are randomly scattered on its surface. The octree has 6 levels.
3. The *prolate sphere* from Fig. 6c is contained in the bounding-box $6.4 \times 6.4 \times 64$; 119 698 particles are randomly scattered on its surface. The octree has 7 levels.

In approximative sense, the sphere is a three-dimensional, the oblate sphere a two-dimensional and the prolate sphere a one-dimensional object in $\mathbb{R}^3$. We choose these three geometries to study their influence on the performance on the dFMM. Table 2 shows the size of the full and the reduced interaction lists ($|T|$ and $|T_{\text{sym}}|$) per level (lf stands for low-frequency, hf for high-frequency regime) for all three geometries. The size of the interaction lists clearly grows with the dimensionality of the geometry. We report on the impact of this behavior later.

	sphere				oblate sphere				prolate sphere			
	3(hf)	4(hf)	5(hf)	6(hf)	3(hf)	4(hf)	5(hf)	6(hf)	4(hf)	5(hf)	6(hf)	7(lf)
$\lvert T\rvert$	668	18710	2666	418	60	2336	1502	400	214	738	382	424
$\lvert T_{\text{sym}}\rvert$	20	518	93	21	6	203	89	21	35	61	21	22

Table 2: Size of interactions lists per level for dFMM (hf and lf stands for high- and low-frequency regime, respectively)



(a) sphere (b) oblate spheroid (c) prolate spheroid

Figure 6: The example geometries are centered at $(0, 0, 0)$

5.1 Accuracy of the method

Both, the bbFMM and the dFMM, have two approximations: 1) the interpolation of the kernel functions determined by interpolation order ℓ, and 2) the subsequent low-rank approximation of the M2L operators determined by the target accuracy ε. The final relative error is a result of both approximations. We compute it as

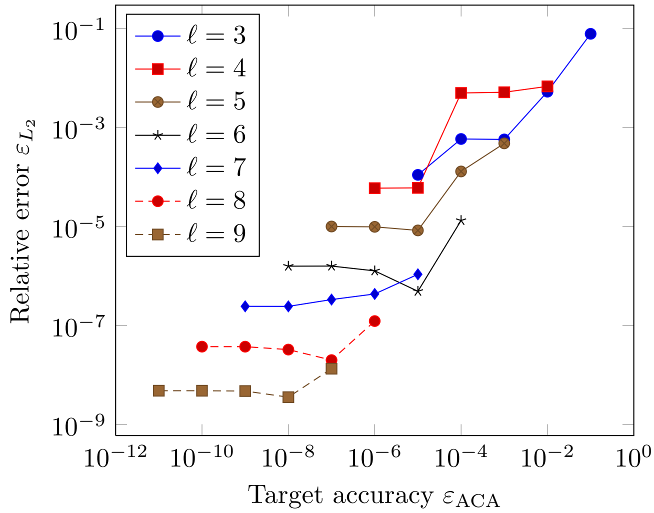
$$\varepsilon_{L_2} = \left(\frac{\sum_{i \in M} |f_i - \bar{f}_i|^2}{\sum_{i \in M} |f_i|} \right)^{1/2} \tag{34}$$

where M is the number of particles x in an arbitrary reference cluster at the leaf level; f and $\bar{f}$ are the exact and approximate results, respectively. In Fig. 7 we compare achieved accuracies for the bbFMM and the dFMM with the IAblk variant (other variants produce identical results). Both plots show the behavior of the relative error ε_{L_2} depending on the interpolation order ℓ and the target accuracy ε. Evident is the matching result between the left and right figure. All curves show an initial plateau and then, after a sharp knee a slope of roughly 1. The knee occurs approximately at $(\ell, \varepsilon) = (Acc, 10^{-Acc})$. In the rest of the paper we use this convention to describe the accuracy Acc of bbFMM and dFMM. The low-rank approximations for the computations whose accuracies are shown in Fig. 7 were conducted with the ACA followed by a truncated SVD. By just using a truncated SVD we obtain identical results.

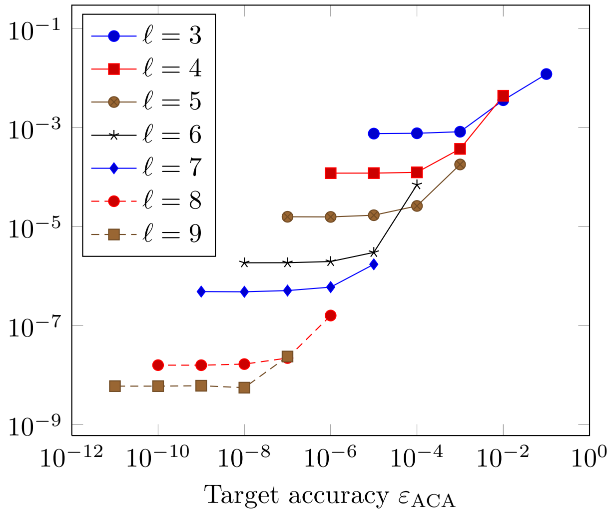
5.2 Reducing the cost with the SArcmp variant

The cost of applying an approximate M2L operator mainly depends on its rank k. In Tab. 3 we compare the average rank of M2L operators for the SA and the IA variants at all levels that have expansions. Let us explain how we computed the average rank for SA. Recall, when we use that variant, all M2L operators from one interaction list posses the same rank; the bbFMM and the dFMM in the low-frequency regime have one and the dFMM in the high-frequency regime has potentially multiple (directional) interaction lists per level. Thus, the average rank per level is the average of all ranks used in all interactions at that level.

The application of one M2L operator from SA and IA requires $\mathcal{O}(k^2)$, respectively $\mathcal{O}(2k\ell^3)$ operations. Note, the ranks k of the M2L operators are different; for a given accuracy Acc they are normally significantly lower for IA than for SA. This can be seen in Tab. 3. The large ranks at level 7 (first level in the low-frequency regime) of SA are noteworthy. There, the lower bound for the separation criterion is given by the usual low-frequency criterion saying that non touching cells are admissible. Hence, the smallest possible transfer



(a) bbFMM for the smooth kernel function



(b) dFMM for the oscillatory kernel function

Figure 7: Accuracies for the prolate sphere from Fig. 6c. The target accuracy $\varepsilon_{\mathrm{ACA}}$ refers to the accuracy of the approximate M2L operators (see Eqn. (20)). Here, we used the ACA followed by a truncated SVD.

	SA				IA			
Acc	4(hf)	5(hf)	6(hf)	7(lf)	4(hf)	5(hf)	6(hf)	7(lf)
3	9.8	12.3	12.8	19	5.4	5.7	5.7	5.0
5	21.7	30.8	39.2	71	11.3	12.3	13.5	12.6
7	38.2	58.6	80.0	163	18.9	21.5	24.7	24.1
9	57.8	96.5	138.7	296	28.6	33.2	39.7	40.1

Table 3: Comparison of average ranks k for the SA and IA variants of the dFMM (prolate sphere)

vectors have a length of $\min_{t\in T}|t| = 2$. The slowly decaying singular values of associated M2L operators are responsible for the large ranks. On the other hand, the upper bound for the separation criterion coincides with the lower bound of level 6 (parent level), which is in the high-frequency regime. Hence, the largest possible transfer vectors have a length of $\max_{t\in T}|t| \sim 4k$. M2L operators whose transfer vectors are in that range have much faster decaying singular values. This fact explains the efficiency of SArcmp (individual recompression of each M2L operator).

In Tab. 4 we analyze the cost savings of SArcmp compared to SA. The left values in each column multiplied by 10^6 give the overall number of floating point operations per level for SA. The right values (in brackets) indicate the respective cost ratio of SArcmp to SA. The recompression reduces the cost remarkably (see also Fig. 8). At the low-frequency level 7, SArcmp reduces the cost by more than a factor of 2. This is almost twice as much as in high frequency levels. For the impact on timing results we refer to Sec. 5.4.

	cost(SA)/10^6 (cost(SArcmp)/ cost(SA))							
Acc	4(hf)		5(hf)		6(hf)		7(lf)	
3	0.8	(0.98)	20.3	(0.93)	31.5	(0.93)	204.3	(0.62)
5	4.9	(0.97)	161.4	(0.89)	339.3	(0.86)	2889.3	(0.47)
7	19.3	(0.97)	687.8	(0.88)	1590.2	(0.83)	15420.0	(0.40)
9	55.0	(0.97)	2138.2	(0.87)	5237.6	(0.83)	51505.7	(0.38)

Table 4: Comparison of cost in terms of floating point operations for SA and SArcmp (prolate sphere)

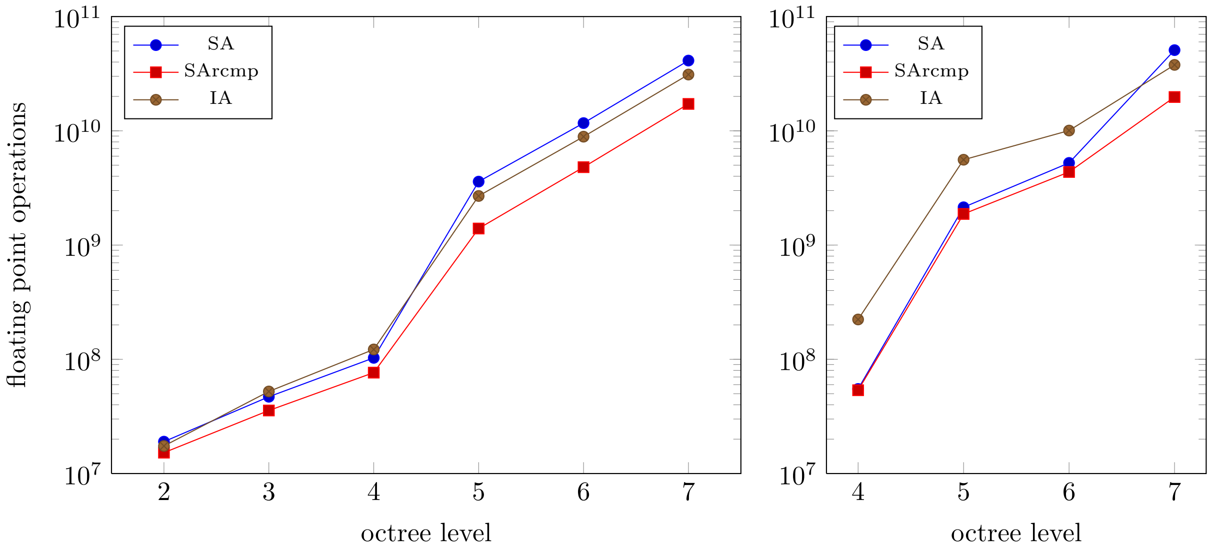
In Fig. 8 we compare the cost of SA, SArcmp and IA for bbFMM and dFMM for the prolate sphere and accuracy $Acc = 9$. The bbFMM in Fig. 8a has expansions at the levels $2-7$. The reason for the jump between level 4 and 5 is because the levels $2-4$ have a maximum of $|T| = 16$ and the levels $5-7$ a maximum of $|T| = 316$ (common maximum size for bbFMM) possible M2L operators. The jump in the case of dFMM in Fig. 8b at level 5 can be explained in the same way: if we look at Tab. 2 we see that $|T|$ is about twice as large as it is at the other levels. The levels $4, 5, 6$ of dFMM are in the high-frequency regime. There, the cost of IA is approximately $12, 5, 3$ times greater than the cost of SA. However, at level 7 (low-frequency regime) of dFMM and at the levels $5-7$ of bbFMM the cost of IA is only about $2/3$ compared to SA. The reason is the size of the interaction lists $|T|$. The larger they become the larger the span between the smallest and the largest rank and that favors individual approximations. The SArcmp is computationally the least expensive variant. Similar results are obtained for all other accuracies.

5.3 Speeding up the precomputation with IA variants

The bottleneck of SA and SArcmp is the relatively large precomputation time. This is a minor issue in the case of homogeneous kernel functions. In that case the approximated M2L operators can be stored on the disk and loaded if needed for further computations. However, if we deal with non-homogeneous kernel functions, such as the Helmholtz kernel, IAsym is the way to go. In Tab. 5 we compare the precomputation time of SA, IA and IAsym (we do not report on SArcmp; due to the additional recompression its precomputation time is higher than the one for SA). For the low-rank approximation we use a truncated SVD or the ACA followed by a truncated SVD. In both cases we end up with the same rank. We get remarkable speedups in the precomputation. Let us look at an extreme example: for the sphere and an accuracy $Acc = 7$ we have $t_{\mathrm{SVD}} = 4740.9\,\mathrm{s}$ for SA versus $t_{\mathrm{ACA}} = 1.8\,\mathrm{s}$ for IAsym. This corresponds to a speedup greater than 2600. For the oblate and prolate sphere we obtain similar results.

5.4 Comparison of all M2L variants

In the previous sections we have revealed the impact of SArcmp and IAsym on the computational cost and the precomputation time of the M2L operators, respectively. In this section we compare all variants and focus on their impact on memory requirement and running time (of applying M2L operators during the



(a) bbFMM

(b) dFMM

Figure 8: Comparison of the M2L cost (floating point operations) growth per level for the SA, SArcmp and IA variants of bbFMM and dFMM ($Acc = 9$ and prolate sphere)

actual matrix-vector product). Tables 6–8 (respectively, 9–11) present running times of the M2L operators for bbFMM (respectively, dFMM) and all three geometries. The upper set of rows reports on results obtained with a BLAS and LAPACK implementation (libblas 1.2.20110419-2, which is sub-optimal in our case). The lower set shows the times obtained with the Intel MKL [9] (Intel MKL 10.3.11.339 (intel64)), which proved faster for our purpose.

Impact of using symmetries on the memory requirement Missing times in the tables indicate that the required memory exceeded the available memory of 8 GB. Clearly, the NA variant (no approximation, no use of symmetries) requires the most memory. On the other hand, IAsym and IAblk are the most memory friendly variants. Both require only memory for storing the low-rank representations of M2L operators from T_{sym} (see Tab. 2 for a comparison of T and T_{sym}).

Impact of the blocking scheme on runtime Bold numbers indicate the fastest variants. Two results attract our attention. 1) If we look at the times in the Tabs. 6–11 we notice that in four cases (bbFMM with the sphere, the oblate sphere and the prolate sphere and dFMM with the prolate sphere) IAblk, and in two cases (dFMM with the sphere and the oblate sphere) SArcmp is fastest. To be more general, IAblk wins at levels having non-directional expansions (all levels of bbFMM and low-frequency levels of dFMM) and SArcmp at levels having directional expansions (high-frequency levels of dFMM). Why is that? The reason follows from the cost studies in Sec. 5.2. Let us take for example $Acc = 5$. Recall, we measure the computational cost based on the size of the approximate M2L operators, ie., $\mathcal{O}(k^2)$ for SA and $\mathcal{O}(2k\ell^2)$ for IA. The respective ranks k are given in Tab. 3. The ratio of these costs for SA to IA is 0.46 at the high-frequency level 6 and 1.60 at the low-frequency level 7. As a matter of fact, at high-frequency levels wins SA and at low-frequency levels IA. Even savings of 47% at the low-frequency level due to the recompression of SArcmp (see Tab. 4) are not sufficient to outweigh the advantage of the blocking scheme of IAblk. If there is no low-frequency level, such as for the sphere in Tab. 9 and the oblate sphere in Tab. 10, the SArcmp outperforms all other variants. For example, if we would repeat the computations for the prolate sphere with an octree of depth 6 (no low-frequency level) the resulting timing patterns would follow those from

Acc	SA		IA		IAsym	
	t_{SVD} [s]	t_{ACA} [s]	t_{SVD} [s]	t_{ACA} [s]	t_{SVD} [s]	t_{ACA} [s]
sphere						
3	7.0	4.6	6.2	1.9	0.2	0.1
4	75.0	20.2	37.0	4.9	1.1	0.2
5	317.2	69.1	188.8	12.4	5.6	0.4
6	1336.4	197.0	790.5	28.6	22.9	0.9
7	4740.9	435.9	-	-	84.0	1.8
oblate sphere						
3	1.4	0.9	1.2	0.4	0.1	0.0
4	13.4	3.8	7.1	1.1	0.5	0.1
5	59.7	13.5	37.0	2.7	2.8	0.2
6	299.4	39.4	150.1	6.4	11.1	0.5
7	1021.2	97.6	551.0	13.3	40.9	0.9
8	-	217.7	1751.4	29.0	129.3	2.0
9	-	444.6	-	-	369.2	3.9
prolate sphere						
3	0.6	0.7	0.8	0.3	0.1	0.0
4	7.5	4.1	5.3	0.7	0.4	0.1
5	64.7	18.3	31.0	2.1	2.5	0.1
6	358.2	73,3	199.4	5.0	15.7	0.4
7	1374.8	204.3	808.9	11.6	66.0	0.8
8	-	549.3	-	25.0	322.3	1.7
9	-	1273.2	-	50.2	807.4	3.7

Table 5: Precomputation times (SVD versus ACA) for the SA, IA and IAsym variants. Missing numbers mean that the available memory of 8 GB has not been sufficient for the respective computation.

the sphere and the oblate sphere (the overall application time would increase too, since the tree-depth is based on our choice of balancing near- and far-field, i.e., the shortest overall application time). 2) Evident is the wide margin in the speedup of variants that use blocking and those which do not. If we use the MKL (as opposed to libblas) for NA, NAsym, SA, SArcmp, IA and IAsym we end up with $1.5 - 2$ times faster application times. However, if we use the MKL for NAblk and IAblk we achieve $3 - 4$ times faster times. Even though these speedups are greatest with the MKL library, it highlights the benefits of the blocking scheme presented in Alg. 2.

Varying growth of application times In Fig. 9 we visualize the different growth rates of M2L application times for the bbFMM with increasing accuracies *Acc*. We are interested in the growth rates due to algorithmic changes. That is why we only study those variants that do not use blocking. Since no approximation is involved the times for NAsym grows the fastest. The times for SA grow slower but still faster than those for IAsym. SArcmp features the slowest growth, it is the optimal variant in terms of computational cost (see Tab. 4).

6 Conclusion

The fast multipole method based on Chebyshev interpolation, first presented in [5] for smooth kernel functions (bbFMM) and extended in [12] to oscillatory ones (dFMM), is a convenient-to-use algorithm due to its easy

Acc	NA	NAsym	NAblk	SA	SArcmp	IA	IAsym	IAblk
				libblas 1.2.20110419-2				
3	0.7	0.8	0.4	0.5	**0.3**	0.4	0.4	**0.3**
4	3.6	3.8	1.8	1.4	**0.9**	1.3	1.4	**0.7**
5	14.2	12.8	5.9	4.6	**2.2**	3.4	3.6	**1.8**
6	42.8	38.2	16.5	11.5	**4.8**	9.0	8.7	**4.2**
7	102.7	101.7	40.7	25.5	**9.6**	20.2	18.4	**8.6**
8	229.3	234.2	89.7	47.8	**18.3**	40.7	35.9	**16.2**
9	-	484.5	180.0	83.8	**30.0**	74.0	68.6	**28.4**
				Intel MKL 10.3.11.339 (intel64)				
3	0.3	0.3	0.2	0.2	0.4	0.4	0.4	**0.2**
4	1.8	1.2	0.6	0.6	0.7	0.7	0.8	**0.4**
5	9.4	6.3	1.9	2.0	1.4	2.0	2.0	**1.0**
6	28.2	14.1	4.8	6.9	2.6	5.0	4.0	**1.7**
7	72.6	57.7	12.0	19.3	5.8	12.7	8.2	**3.5**
8	127.6	117.0	25.6	34.3	12.0	24.6	16.2	**6.0**
9	-	260.5	50.8	60.4	20.6	44.6	32.9	**9.9**

Table 6: M2L timings for the bbFMM (sphere). In this table and below as well, bold numbers correspond to the smallest entry in a row. In some cases, two columns use bold font when the running times are sufficiently close that the difference is not significant.

Acc	NA	NAsym	NAblk	SA	SArcmp	IA	IAsym	IAblk
				libblas 1.2.20110419-2				
3	0.3	0.4	0.2	0.2	**0.1**	0.2	0.2	**0.1**
4	1.6	1.6	0.8	0.8	**0.4**	0.6	0.6	**0.3**
5	6.4	5.8	2.6	1.9	**0.9**	1.6	1.6	**0.8**
6	19.1	16.7	7.4	5.4	**2.1**	4.0	3.9	**1.9**
7	46.8	44.8	18.3	11.2	**4.5**	9.0	9.0	**3.8**
8	102.8	101.9	40.1	21.3	**8.4**	18.2	16.5	**7.2**
9	-	212.2	81.1	37.0	**13.5**	32.4	30.4	**12.8**
				Intel MKL 10.3.11.339 (intel64)				
3	0.1	0.1	0.1	0.1	0.2	0.2	0.2	**0.1**
4	0.7	0.5	0.3	0.3	0.3	0.3	0.4	**0.2**
5	4.5	2.0	0.9	1.0	0.7	0.9	0.9	**0.4**
6	13.1	6.7	2.2	3.2	1.2	2.6	1.8	**0.8**
7	33.7	27.4	5.4	8.2	2.7	5.7	4.4	**1.6**
8	57.9	52.7	11.5	14.6	5.2	10.9	7.2	**2.7**
9	117.4	118.2	22.9	27.3	9.1	20.0	14.2	**4.4**

Table 7: M2L timings for bbFMM (oblate sphere)

Acc	NA	NAsym	NAblk	SA	SArcmp	IA	IAsym	IAblk
libblas 1.2.20110419-2								
3	0.2	0.2	0.1	0.1	**0.1**	0.1	0.1	**0.1**
4	1.0	1.1	0.5	0.4	**0.3**	0.4	0.4	**0.2**
5	4.0	4.0	1.9	1.2	**0.6**	1.2	1.0	**0.6**
6	12.0	11.8	4.7	3.4	**1.3**	2.6	2.4	**1.4**
7	29.3	31.4	11.7	6.9	**2.8**	7.2	5.2	**2.9**
8	68.4	71.4	25.5	13.1	**5.2**	11.6	10.8	**4.9**
9	131.7	137.3	50.7	22.3	**8.6**	21.1	19.9	**8.7**
Intel MKL 10.3.11.339 (intel64)								
3	0.1	0.1	0.1	0.1	0.1	0.1	0.1	**0.1**
4	0.4	0.3	0.2	0.2	0.2	0.2	0.2	**0.1**
5	2.9	1.3	0.6	0.6	0.4	0.6	0.6	**0.3**
6	8.2	4.2	1.4	2.1	0.7	1.5	1.1	**0.5**
7	20.5	16.3	3.4	5.9	1.7	3.7	2.4	**1.0**
8	35.7	34.1	7.2	9.1	3.7	7.1	5.5	**1.7**
9	73.3	73.8	14.4	17.6	5.9	12.8	11.2	**2.8**

Table 8: M2L timings for bbFMM (prolate sphere)

Acc	NA	NAsym	NAblk	SA	SArcmp	IA	IAsym	IAblk
libblas 1.2.20110419-2								
3	6.3	5.9	4.8	2.0	**2.0**	3.2	2.9	3.5
4	25.7	25.1	20.1	4.3	**3.9**	8.0	8.1	8.3
5	113.0	89.5	71.4	7.4	**7.1**	19.8	19.4	19.7
6	-	275.9	202.2	14.5	**11.6**	43.0	42.8	40.4
7	-	-	-	-	-	-	86.5	78.1
Intel MKL 10.3.11.339 (intel64)								
3	4.8	3.9	2.3	1.6	**1.7**	3.2	2.4	2.2
4	21.5	17.1	7.5	3.3	**3.4**	6.7	5.9	4.8
5	109.1	61.5	22.8	6.3	**6.3**	10.6	14.0	10.0
6	-	162.2	58.8	11.6	**9.8**	35.1	29.7	17.6
7	-	-	-	-	-	-	60.8	30.8

Table 9: M2L timings for dFMM (sphere); high-frequency leaf level

Acc	NA	NAsym	NAblk	SA	SArcmp	IA	IAsym	IAblk
libblas 1.2.20110419-2								
3	1.8	1.8	1.6	0.6	**0.6**	0.9	0.9	1.1
4	8.9	8.6	6.8	1.3	**1.1**	2.7	2.6	2.7
5	31.4	30.6	24.8	2.8	**2.3**	6.9	6.8	6.9
6	91.8	95.9	71.6	5.5	**4.1**	15.9	15.9	14.8
7	-	228.4	176.6	9.6	**7.3**	32.1	32.3	28.9
8	-	-	-	16.4	**11.2**	59.2	59.6	52.7
9	-	-	-	25.5	**21.9**	-	105.1	90.1
Intel MKL 10.3.11.339 (intel64)								
3	1.4	1.0	0.7	0.4	**0.4**	0.7	0.8	0.6
4	7.2	5.0	2.4	0.9	**1.0**	2.0	1.8	1.3
5	25.8	20.7	7.9	2.2	**1.9**	5.5	4.5	3.1
6	62.5	56.7	20.4	4.3	**3.4**	12.6	10.3	5.9
7	-	141.3	48.8	7.8	**5.5**	24.7	21.8	11.0
8	-	-	-	13.6	**9.5**	45.3	42.0	18.8
9	-	-	-	21.0	**14.7**	-	79.6	30.9

Table 10: M2L timings for dFMM (oblate sphere); high-frequency leaf level

Acc	NA	NAsym	NAblk	SA	SArcmp	IA	IAsym	IAblk
libblas 1.2.20110419-2								
3	0.6	0.6	0.5	0.3	**0.2**	0.3	0.3	0.3
4	3.2	2.8	2.3	1.0	**0.5**	0.9	0.9	1.0
5	11.9	10.0	8.7	2.7	**1.2**	2.4	2.4	2.5
6	32.5	30.8	25.1	6.5	**2.6**	6.0	5.6	5.7
7	80.3	77.0	61.1	13.2	**5.1**	12.1	11.8	11.1
8	-	-	-	24.8	**9.1**	24.3	23.3	21.2
9	-	-	-	47.5	**18.6**	43.5	43.5	37.2
Intel MKL 10.3.11.339 (intel64)								
3	0.3	0.3	0.2	0.2	**0.2**	0.2	0.2	**0.2**
4	2.1	1.4	0.7	0.5	**0.4**	0.6	0.6	**0.4**
5	8.7	5.8	2.5	2.0	**1.0**	2.0	1.5	**1.0**
6	20.2	17.9	6.7	5.1	**2.0**	4.6	3.4	**2.0**
7	48.4	46.5	16.3	10.3	**4.0**	9.6	7.5	**3.8**
8	-	-	-	16.6	**7.2**	18.3	15.8	**7.0**
9	-	-	-	31.6	**14.8**	33.7	32.1	**11.6**

Table 11: M2L timings for dFMM (prolate sphere); low-frequency leaf level

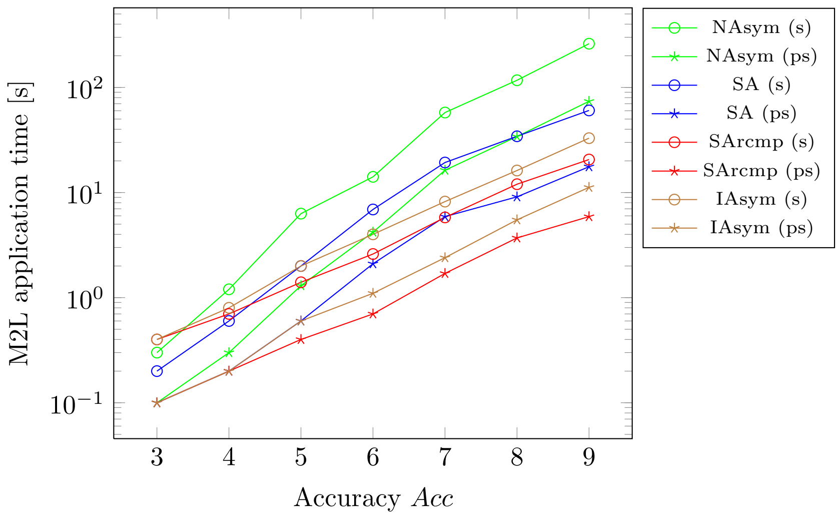


Figure 9: Running times versus accuracy for NAsym, SA, SArcmp and IAsym for bbFMM taken from the Tabs. 6, 7 and 8; (s) stands for sphere and (ps) for prolate sphere

implementation and its kernel function independence. In this paper we investigated algorithms to reduce the running time of the M2L operator. We proposed several optimizations and studied their respective strengths and weaknesses.

On one hand we proposed SArcmp, which uses an individual recompression of the suboptimally approximated M2L operators obtained via SA (the variant presented in [5]). We have shown that this new variant reduces the computational cost noticeably. In some settings it even provides the fastest M2L application times. On the other hand we also proposed a new set of optimizations based on an individual low-rank approximation of the M2L operators; we refer to them as IA variants. As opposed to SA they directly lead to the optimal low-rank representation for each operator. The overall number of flops is greater than for SArcmp (which is strictly a lower bound on the number of flops). However, the advantage of the individual treatment of the M2L operators is that we can exploit symmetries in their arrangement. This means that all operators can be expressed as permutations of a subset. For example, in the case of the bbFMM (in which the full interaction list has a constant size), we need to approximate and store only 16 instead of 316 operators. The remaining ones can be expressed as permutations thereof. This has a great impact on the precomputation time and the memory requirement. Moreover it allows to express (again in the case of the bbFMM) the at most 189 matrix-vector products (applications of the M2L operators) as at most 16 **matrix-matrix** products. We referred to this approach as the IAblk variant. It can then take advantage of highly optimized implementations of matrix-matrix operations (e.g., the MKL [9]).

Let us conclude by comparing SArcmp and IAblk, the two variants that have the fastest running times. IAblk wins if we compare precomputation time, required memory and runtime at levels having non-directional expansions (bbFMM and low-frequency levels in dFMM). SArcmp wins only if we compare the runtime at levels having directional expansions (high-frequency levels in dFMM). However, in order to identify the optimal variant we have to distinguish two potential uses of the FMM as a numerical scheme to perform fast matrix-vector products. 1) If we are interested in the result of a single matrix-vector product, a quick precomputation is essential. However, 2) if we are looking for the iterative solution of an equation system (many matrix-vector products), a fast running time of the M2L operator is crucial. Let us explain this with an example. We take **dFMM** (with MKL) with accuracy $Acc = 5$ for the sphere. IAblk wins if we are interested in the former use. The precomputation takes 0.4 s versus 69.1 s (for SArcmp) and the M2L application takes 10.0 s versus 6.3 s, which sums up to 10.4 s versus 75.4 s. All other operators (P2P, P2M, M2M, L2L and L2P) have nearly the same runtime in both cases, and their runtimes are negligible compared to M2L. Looking at the latter use, SArcmp starts being faster if the iterative solution requires more than 19 matrix-vector products. For higher accuracies this threshold rises, e.g., for $Acc = 6$ it lies at 26 matrix-vector products. In the case of **bbFMM**, IAblk is always optimal.

References

[1] M. Bebendorf. Hierarchical LU decomposition based preconditioners for BEM. *Computing*, 74:225–247, 2005.

[2] H. Cheng, W. Y. Crutchfield, Z. Gimbutas, L. Greengard, J. F. Ethridge, J. Huang, V. Rokhlin, N. Yarvin, and J. Zhao. A wideband fast multipole method for the Helmholtz equation in three dimensions. *Journal of Computational Physics*, 216(1):300–325, 2006.

[3] E. Darve and P. Havé. Efficient fast multipole method for low-frequency scattering. *Journal of Computational Physics*, 197(1):341–363, 2004.

[4] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. S. Duff. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Softw.*, 16(1):1–17, March 1990.

[5] W. Fong and E. Darve. The black-box fast multipole method. *Journal of Computational Physics*, 228 (23):8712–8725, 2009.

[6] Z. Gimbutas and V. Rokhlin. A generalized fast multipole method for nonoscillatory kernels. *SIAM Journal on Scientific Computing*, 24(3):796–817, March 2002.

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

[8] L. Greengard and V. Rokhlin. A new version of the fast multipole method for the Laplace equation in three dimensions. *Acta Numerica*, 6:229–269, 1997.

[9] Intel. Intel Math Kernel Library (Intel MKL) 10.3, 2012. URL `http://software.intel.com/en-us/articles/intel-mkl/`. [Online; accessed 28-August-2012].

[10] P. G. Martinsson and V. Rokhlin. An accelerated kernel-independent fast multipole method in one dimension. *SIAM Journal on Scientific Computing*, 29(3):1160–1178, May 2007.

[11] J. C. Mason and D. C. Handscomb. *Chebyshev Polynomials*. Chapman & Hall/CRC, 2003.

[12] M. Messner, M. Schanz, and E. Darve. Fast directional multilevel summation for oscillatory kernels based on chebyshev interpolation. *Journal of Computational Physics*, 231(4):1175 – 1196, 2012.

[13] V. Rokhlin. Diagonal forms of translation operators for the Helmholtz equation in three dimensions. *Applied and Computational Harmonic Analysis*, 1:82–93, 1993.

[14] L. Ying, G. Biros, and D. Zorin. A kernel-independent adaptive fast multipole algorithm in two and three dimensions. *Journal of Computational Physics*, 196(2):591 – 626, 2004.