Kernels of the black-box FMM

Matthias Messner

February 28, 2013

Abstract

This document explains in detail the (CPU) kernels of the black-box FMM we use and points out differences to the paper by Fong and Darve [1].

Contents

1 Basics

We compute the interaction between two cells $X, Y \subset \mathbb{R}$. The cell Y contains all source particles $\{y_j\}_{j=1}^N$ and the cell X contains all target particles $\{x_i\}_{i=1}^M$. The goal is to compute the interactions efficiently by exploiting the Chebyshev interpolation scheme

$$\begin{aligned} f_i &= \sum_{j=1}^{N} K(x_i, y_j) w_j \quad \text{for } i = 1, \dots, M \\ &\sim \sum_{m=1}^{\ell} S_\ell(x_i, \bar{x}_m) \sum_{n=1}^{\ell} K(\bar{x}_m, \bar{y}_n) \sum_{j=1}^{N} S_\ell(y_j, \bar{y}_n) w_j \end{aligned} \tag{1}$$

with the interpolation points $\bar{x} \in X$, respectively $\bar{y} \in Y$. The interpolation polynomial is defined as

$$S_\ell(x, \bar{x}_m) = \frac{1}{\ell} + \frac{2}{\ell} \sum_{i=1}^{\ell-1} T_i(x)\, T_i(\bar{x}_m). \tag{2}$$

The individual FMM kernels can be identified in Eqn. (1) as

- P2M (particle-to-moment) at the leaf-level and M2M (moment-to-moment) at all other levels:

$$W_n = \sum_{j=1}^{N} S_\ell(y_j, \bar{y}_n) w_j \quad \text{for } n = 1, \ldots, \ell \tag{3}$$

- M2L (moment-to-local):

$$F_m = \sum_{n=1}^{\ell} K(\bar{x}_m, \bar{y}_n) W_n \quad \text{for } m = 1, \ldots \ell \tag{4}$$

- and L2P (local-to-particle) at the leaf-level and L2L (local-to-local) at all other levels:

$$f_i \sim \sum_{m=1}^{\ell} S_\ell(x_i, \bar{x}_m) F_m \quad \text{for } i = 1, \ldots, M. \tag{5}$$

For the sake of readability we omit the mapping from and to the reference interval $[-1, 1]$. Moreover, the extension of the interpolation scheme in $\mathbb{R}$ to the tensor-interpolation in $\mathbb{R}^3$ is going to be explained in detail in Sec. 3.

2 M2L — optimization based on symmetries

In this section we consider the M2L kernel. In Eqn. (4) it is presented for the one-dimensional case. Its extension to the three-dimensional case is less involving then for the other kernels. We have the cells $X, Y \subset \mathbb{R}^3$ and the M2L kernel reads in $\mathbb{R}^3$ as

$$F_m = \sum_{n=1}^{\ell^3} K(\bar{x}_m, \bar{y}_n) W_n \quad \text{for } m = 1, \ldots \ell^3. \tag{6}$$

In the following we introduce an improved version of the M2L kernel for the black-box FMM which is favorable over the approach presented in [1] in terms of precomputation time, number of required floating point operations (flops) and cache reuse. First, we show the advantage of individual SVDs, second, we exploit symmetries in the arrangement of the far-field interactions and, finally, the combination of both approaches leads to the new M2L kernel.

Far-field interactions Normally, in $\mathbb{R}^3$ the far-field is limited to the at most 26 near-field interactions of the parent-cell. This leads to at most 189 far-field interactions $I_Y = \{Y_s : 1 \leq s \leq 189\}$ (source cells) for each target cell X. Most kernel functions are homogeneous (if we scale the distance between source y and target x by a factor of α the resulting potential is scaled by α^n, where n is a constant and depends on the kernel function). Hence, the M2L kernels of all 316 possible far-field interactions for all cells need to be computed only once. In order to identify the far-field interactions we introduce translation vectors $t = (t_1, t_2, t_3)$ which give the relative position of a cell-pair (X, Y). All its components are limited to the interval $t_i \in [-3, 3]$ (near-field interactions of parent-cell) and $|t| > \sqrt{3}$ (excludes near-field interactions) must be true. This leads to $7^3 - 3^3 = 316$ different translation vectors.

Note. *If we use kernel functions which are not homogeneous, such as the Lenard-Jones potential* $K_{LJ}(r) = \frac{1}{r^{12}} - \frac{1}{r^6}$ *the only difference is that we cannot compute the M2L kernels all far-field interactions only in the reference cells (all dimensions are in* $[-1, 1]$*) and scale them on different levels but we have to compute them on each level. This affects the precomputational time and the memory requirement (both* $h - 2$ *times as large as the one for the homogeneous case).*

Table 1: Comparison of both M2L optimizations

Acc	r_{big}	r_{ave}	$\text{cost}_{\text{ave}}/\text{cost}_{\text{big}}$
3	19	4.6	0.69
5	67	11.2	0.62
7	150	22.2	0.67

2.1 Individual SVDs

We apply individual SVDs to each of the 316 far-field interactions and compare the resulting cost to the approach proposed in [1] in Tab. 1. The first column gives the accuracy defined as $(\ell, \varepsilon_{\text{SVD}}) = (Acc, 10^{-Acc})$ and the second and third column compare the low-rank r_{big} to r_{ave} (the average low-rank of all 316 far-field interactions in our approach). Finally, column four shows the ratio of the computational costs of applying them, they are defined as $\text{cost}_{\text{big}} = \mathcal{O}(316 \cdot r_{\text{big}}^2)$ and $\text{cost}_{\text{ave}} = \mathcal{O}(316 \cdot 2\ell^3 r_{\text{ave}})$. Where do these numbers come from? Fong and Darve [1] write the $t = 1, \ldots, 316$ far-field interactions as $\mathsf{K}_t \sim \mathsf{U}\mathsf{C}_t\mathsf{V}^\top$, hence, the M2L kernel consists in 316 matrix-vector products with $\mathsf{C}_t \in \mathbb{R}^{r_{\text{big}} \times r_{\text{big}}}$. We represent the far-field interactions as $\mathsf{K}_t \sim \mathsf{U}_t\mathsf{\Sigma}_t\mathsf{V}_t^\top$ with the matrices $\mathsf{U}_t, \mathsf{V}_t \in \mathbb{R}^{\ell^3 \times r_t}$. Note that the average rank is defined as $r_{\text{ave}} = (\sum_{t=1}^{316} r_t)/316$.

2.2 Symmetries

Evidently, there are symmetries in the arrangement of the far-field interactions. By exploiting them, the 316 interactions can be reduced to 16 only. How do we do that? First, by considering symmetries given by the axial planes, we obtain 8 octants as shown in Fig. 1. We use the octant with $i, j, k \geq 0$ as reference octant. In this way we reduce the 316 interactions to only $4^3 - 2^3 = 56$ different interactions. Next, we exploit

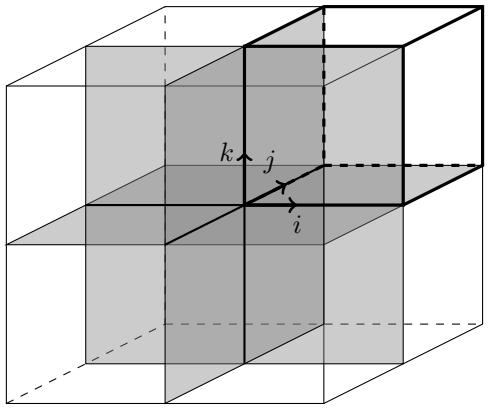
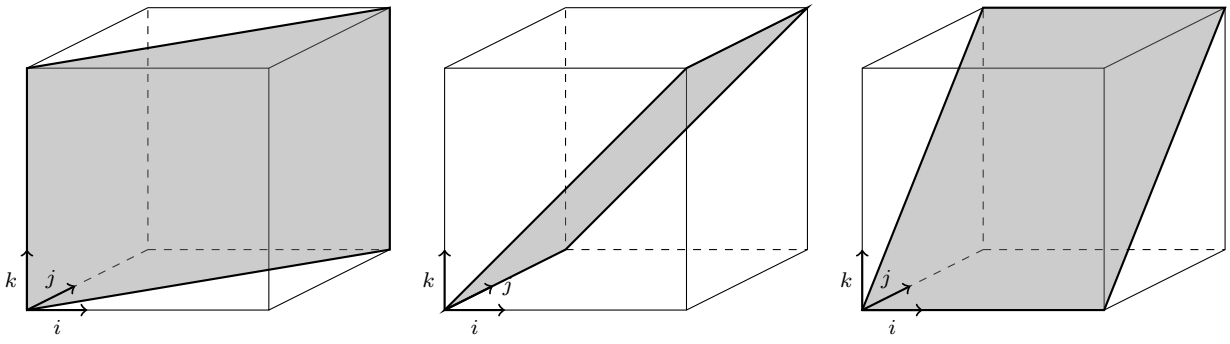


Figure 1: Box of size $7 \times 7 \times 7$ showing the reference octant with $i, j, k \geq 0$ (contains $4^3 - 2^3 = 56$ far-field interactions).

symmetries given by diagonal planes, $i = j$ in Fig. 2a, $i = k$ in Fig. 2b and $j = k$ in Fig. 2c. After combining them and requiring $j \leq i$, $k \leq i$ and $k \leq j$ we can further reduce the 56 interactions in the reference octant to 16 only in the resulting cone, shown in Fig. 3. Finally, all 316 far-field interactions can be expressed as permutations of 16, only. We write them as

$$\mathsf{K}_t = \mathsf{P}_t\mathsf{K}_{p(t)}\mathsf{P}_t^\top \tag{7}$$

where $p(t) : [1, 316] \to [1, 16]$ tells us, depending on t, which of the 16 remaining far-field interactions to use. P_t are the corresponding permutation matrices.



(a) $j \leq i$ (b) $k \leq i$ (c) $k \leq j$

Figure 2: Symmetry planes in reference octant given by $i, j, k \geq 0$.

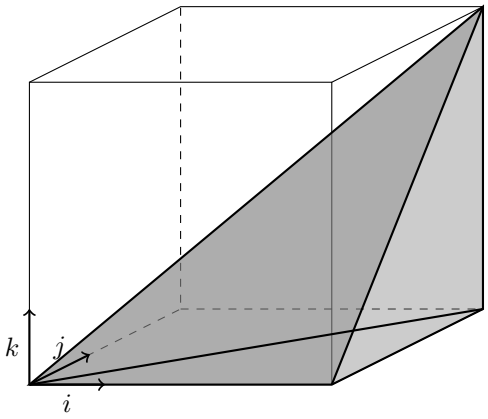


Figure 3: Cone obtained by symmetries from Fig. 2, and requiring $j \leq i$, $k \leq i$, $k \leq j$ contains 16 far-field interactions.

Note. *In this document we do not explain how to compute the permutation matrices and the function $p(t)$. We assume they are known and we know how to compute them. In fact they are part of the precomputation and need to be computed only once.*

2.3 Implementation

The setup of the 316 permutation matrices and the SVDs of the 16 remaining far-field interactions are part of the precomputation. The computation of a far-field interaction, given by t, consists of $\mathsf{f} = \mathsf{K}_t\mathsf{w}$ (in matrix notation). The vectors $\mathsf{f}, \mathsf{w} \in \mathbb{R}^{\ell^3}$ contain the local expansions $(\mathsf{f})_m = F_m$ and multipole expansions $(\mathsf{w})_n = W_n$ of a cell-pair whose relative position is given by the translation vector t. However, we do not compute K_t explicitly as in Eqn. (7) but we use it to compute

$$\mathsf{f} = \mathsf{K}_t\mathsf{w} = \mathsf{P}_t\mathsf{K}_{p(t)}\mathsf{P}_t^\top\mathsf{w} \tag{8}$$

in three steps: (1) we permute multipole expansions $\mathsf{w}_t = \mathsf{P}_t^\top\mathsf{w}$, (2) we compute permuted local expansions $\mathsf{f}_t = \mathsf{K}_{p(t)}\mathsf{w}_t$, (3) we permute permuted local expansions $\mathsf{f} = \mathsf{P}_t\mathsf{f}_t$ (the application of the permutation matrix to a vector consists of reordering the vector – no flop, no matrix-vector product).

In Algorithm 1 we present a naive (unblocked) M2L kernel. It implements the threefold application in a sequential way, hence, we end up with at most 316 matrix-vector products.

Algorithm 1 Unblocked M2L kernel (at most 316 matrix-vector products)

```
1: function UNBLOCKED M2L(target cell X and all far-field interactions I_Y)
2:     retrieve f from X
3:     for all source cells Y in I_Y do
4:         retrieve w from Y and compute t from cell-pair (X, Y)
5:         w_t ← P_t^⊤ w                          ▷ Permute multipole expansions
6:         f_t ← K_{p(t)} w_t                     ▷ Compute permuted local expansions
7:         f ← f + P_t f_t                        ▷ Permute permuted local expansions
8:     end for
9: end function
```

In Algorithm 2 we present an improved (blocked) implementation of the M2L kernel which exploits the symmetries presented in Sec. 2.2. We introduce 16 matrices $\mathsf{F}_p, \mathsf{W}_p \in \mathbb{R}^{\ell^3 \times n_c}$ (see Line 2 in Algorithm 2) whose columns store the permuted local and multipole expansions corresponding to the respective index $p(t)$ they belong to. The value $n_c = 24$ gives the maximal needed number of columns. In other words, there is no chance that one of the 16 far-field interactions is needed more often than 24 times. Moreover, we need an expansion counter $\mathsf{c} \in \mathbb{N}^{16}$ (see Line 4) indicating the number of columns (expansions) of these matrices. Then, we split the single loop into three loops. In the first one, we assemble the set of matrices $\{\mathsf{W}_{p(t)}\}$ and at its end $|\mathsf{c}| \leq 189$ and $\max(\mathsf{c}) = n_c$ is true. In the second one, we perform the 16 matrix-matrix products. And in the last one, we increment the local expansions.

Table 2: Timing results of the Algorithms 1 and 2: Required time of all M2L kernels for $2 \cdot 10^4$ uniformly distributed particles and $h = 5$ on a 2.26 GHz Intel Core 2 Duo without MKL

Acc	t_1 [s]	t_2 [s]	t_2/t_1
3	0.4498	0.2665	0.59
5	2.7170	1.7141	0.63
7	13.7341	11.1773	0.81

Algorithm 2 Blocked M2L kernel (at most 16 matrix-matrix products)

function BLOCKED M2L(target cell X and all far-field interactions I_Y)
  allocate $\{\mathsf{F}_p\}, \{\mathsf{W}_p\}$ for $p = 1, \ldots, 16$ ($\mathsf{F}_p, \mathsf{W}_p \in \mathbb{R}^{\ell^3 \times n_e}$)
  retrieve f from X
  initialize $\mathsf{c} = \mathsf{0}$ ($\mathsf{c} \in \mathbb{N}^{16}$)
  for all source cells Y in I_Y **do**
    retrieve w from Y and compute t from cell-pair (X, Y)
    column $(\mathsf{c})_{p(t)}$ of $\mathsf{W}_{p(t)}$ gets $\mathsf{P}_t^\top \mathsf{w}$ ▷ Permute multipole expansions
    increment $(\mathsf{c})_{p(t)}$
  end for
  for all $\{\mathsf{K}_p\}$ **do**
    $\mathsf{F}_p \leftarrow \mathsf{K}_p \mathsf{W}_p$ ▷ Compute permuted local expansions
  end for
  reinitialize $\mathsf{c} = \mathsf{0}$
  for all source cells Y in I_Y **do**
    compute t from cell-pair (X, Y)
    retrieve f_t from column $(\mathsf{c})_{p(t)}$ of $\mathsf{F}_{p(t)}$
    increment $(\mathsf{c})_{p(t)}$
    $\mathsf{f} \leftarrow \mathsf{f} + \mathsf{P}_t \mathsf{f}_t$ ▷ Permute permuted local expansions
  end for
end function

3 Tensor-product interpolation

The anterpolation (P2M, M2M) can be understood as the transposed operation of interpolation (L2L, L2P). In our notation they read as

$$W_n = \sum_{j=1}^{N} S_\ell(y_j, \bar{y}_m)\, w_j, \text{ for } n = 1, \ldots, \ell^3 \qquad \text{and} \qquad f_i = \sum_{m=1}^{\ell^3} S_\ell(x_i, \bar{x}_m) F_m, \text{ for } i = 1, \ldots, M \tag{9}$$

and in matrix notation as $\mathsf{w}' = \mathsf{S}^\top \mathsf{w}$ and $\mathsf{f} = \mathsf{S}\mathsf{f}'$ if $(')$ denotes the upper level. In the leaf level these kernels correspond to P2M (particle-to-moment) and L2P (local-to-particle) operations. In all other levels they correspond to the M2M (moment-to-moment) and L2L (local-to-local) operations. Depending on the level, M and N vary: in the leaf level they denote the number of source and target particles, in all other levels we have $M = N = \ell^3$. In the following, we explain the tensor-product interpolation first for the M2M and L2L kernels and then for the P2M and L2P kernels.

3.1 M2M and L2L kernels

We assume the uniform case in $\mathbb{R}^3$, i.e., each cell has 8 child cells. We address them based on Morton ordering [see 3] the local position of the child cells is given by their local position 000 until 111, if we translate this binary representation we obtain indices from 0 to 7.

Tensor-product If we consider the 1d case in Fig. 4 each cell has two child cells 0 and 1 and ℓ interpolation points. We set up an interpolation matrix $\mathsf{S}_0, \mathsf{S}_1 \in \mathbb{R}^{\ell \times \ell}$ for both of them. The entries are computed as

$$(\mathsf{S}_i)_{nm} = S_\ell(x_n, \bar{x}_m) \quad \text{with } x_n = \frac{\bar{x}_n}{2} + \frac{(-1)^{i+1}}{2} \quad \text{for } i = 1, 2 \text{ and } n, m = 1, \ldots, \ell, \tag{10}$$

with the Chebyshev roots, defined as $\bar{x}_m = \cos\left((2m-1)\pi/2\ell\right)$. Finally, the M2M and L2L kernels consist of applying both matrices S_0 and S_1 (for child 0 and child 1), respectively.

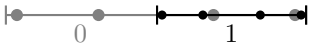


Figure 4: Interpolation points for $\ell = 4$ in $\mathbb{R}$: parent cell in gray $[-1, 1]$ and child cell in black $[0, 1]$

Next, we consider the $\mathbb{R}^3$ case (we think of Fig. 5 as the lower level of child cells): each cell has 8 child cells and the number of interpolation points in each cell is ℓ^3. The most straight forward (but also the most naive) way of implementing the M2M and L2L kernels would be to compute 8 interpolation matrices of size $\ell^3 \times \ell^3$. However, by exploiting the tensor-product approach we can construct a more efficient scheme. We

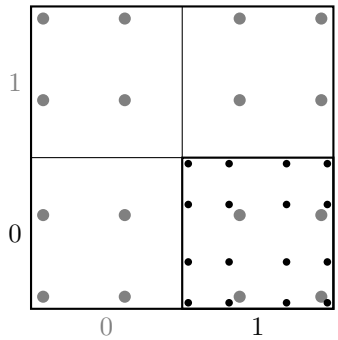


Figure 5: Tensor structure of interpolation points for $\ell = 4$ in $\mathbb{R}^3$ (figure shows the lower level of child cells): parent cell in gray $[-1, 1] \times [-1, 1] \times [-1, 1]$ and child cell in black $[0, 1] \times [-1, 0] \times [-1, 0]$

consider the child cell of index 1 (in binary of Morton notation 001) in Fig. 5. The full interpolation matrix $\mathsf{S}_{001} \in \mathbb{R}^{\ell^3 \times \ell^3}$ can be defined as a tensor product (also Kronecker product) and reads as

$$\mathsf{S}_{001} = \mathsf{S}_0 \times \mathsf{S}_0 \times \mathsf{S}_1 \tag{11}$$

Again, we do never compute S_{001} explicitly, but we exploit properties of the tensor product. In fact, it allows us to reduce the cost $\mathcal{O}(\ell^6)$ of applying the matrix S_{001} to only $\mathcal{O}(3\ell^4)$. We use Eqn. (11) to write the M2M kernel as $\mathsf{w}' = (\mathsf{S}_0 \times \mathsf{S}_0 \times \mathsf{S}_1)^\top \mathsf{w}$ and the L2L kernel as $\mathsf{f} = (\mathsf{S}_0 \times \mathsf{S}_0 \times \mathsf{S}_1)\, \mathsf{f}'$. Recall, we have constructed also the interpolation points based on the tensor product approach as $\bar{x}_m = \bar{x}_i \times \bar{x}_j \times \bar{x}_k$ with $m = k\ell^2 + j\ell + i$. In the following, we use the indices i, j, k and i', j', k' to address them (indices in x, y, z direction and $(')$ denotes the upper level). According to these indices we can reassemble the vectors w, w' and f, f' as tensors of size $\ell \times \ell \times \ell$ with the ijk-th entry being given by the m-th entry of the vectors, respectively. Finally, by using index notation we can write the M2M and L2L kernels as

$$(\mathsf{w}')_{i'j'k'} = (\mathsf{S}_0)_{kk'} (\mathsf{S}_0)_{jj'} (\mathsf{S}_1)_{ii'} (\mathsf{w})_{ijk} \quad \text{and} \quad (\mathsf{f})_{ijk} = (\mathsf{S}_0)_{kk'} (\mathsf{S}_0)_{jj'} (\mathsf{S}_1)_{ii'} (\mathsf{f}')_{i'j'k'}, \tag{12}$$

which can be implemented as three consequent matrix-matrix products followed by a permutation of the resulting matrices. Each product has a cost of $\mathcal{O}(\ell^4)$. This operation has to be repeated for each child cell.

Note. *If we have a look at Fig. 4 we recognize the symmetry of the interpolation points. Hence, it is also possible to express the matrix S_1 as a permutation of S_0. If we follow this idea we could group the 8 times 3 consecutive matrix-matrix products to 8 larger matrix-matrix products. In reality, though, the time required for the M2M and L2L is vanishingly small compared to all other kernels. That is why we did not further investigate in this optimization.*

3.2 P2M and L2P kernels

We cannot extend the tensor-product approach we presented in Sec. 3.1 for the P2M and L2P kernels. In the case of M2M and L2L kernels the evaluation points at both, the child and parent level, form a uniform

(tensor) grid. Here, this is only true for the parent level. The particles (the equivalent to the child level for the M2M and L2L kernels) are completely non-uniform.

Nonetheless, we can construct an efficient implementation. Lets stick again with the $\mathbb{R}$ case, and recall the definition of the interpolation polynomial in Eqn. (2). In order to ease the explanation we use the notation used in [2] to rewrite the interpolation polynomial as

$$S_\ell(x, \bar{x}_m) = \frac{2}{\ell} \sum_{\alpha=0}^{\ell-1}{}' T_\alpha(x) T_\alpha(\bar{x}_m) \tag{13}$$

where the prime ($'$) means that the term in the sum for $\alpha = 0$ is divided by 2. In matrix notation we can write the interpolation polynomial (hereafter interpolation matrix) $\mathsf{S} \in \mathbb{R}^{N \times \ell}$ as an outer product

$$\mathsf{S} = \mathsf{T}\bar{\mathsf{T}}^\top \quad \text{with } \mathsf{T} \in \mathbb{R}^{N \times \ell} \text{ and } \bar{\mathsf{T}} \in \mathbb{R}^{\ell \times \ell} \tag{14}$$

with the entries

$$(\mathsf{T})_{i\alpha} = T_{\alpha-1}(x_i) \qquad \text{and} \qquad (\bar{\mathsf{T}})_{m\alpha} = \frac{2\, T_{\alpha-1}(\bar{x}_m)}{\ell}. \tag{15}$$

The matrix T needs to be computed for each leaf-cell, the matrix $\bar{\mathsf{T}}$ only once. Hereafter, we will focus only on the i-th row $\mathsf{s} = (\mathsf{S})_{i:}$ of the interpolation matrix, i.e., we consider the interpolation of the particle i, only. In the outer-product form this interpolation vector reads as

$$\mathsf{s}_i = \mathsf{t}_i \bar{\mathsf{T}}^\top \quad \text{with } \mathsf{t}_i = (\mathsf{T})_{i:} \tag{16}$$

Next, we consider the $\mathbb{R}^3$ case. Since we have separated the non-uniform part (particles) and the uniform one (interpolation points) we can apply the tensor-product interpolation on each particle separately. We write the i-th interpolation vector as

$$\mathsf{s}_i = \left(\mathsf{t}_{i_0} \bar{\mathsf{T}}^\top\right) \times \left(\mathsf{t}_{i_1} \bar{\mathsf{T}}^\top\right) \times \left(\mathsf{t}_{i_2} \bar{\mathsf{T}}^\top\right) = \left(\mathsf{t}_{i_0} \times \mathsf{t}_{i_1} \times \mathsf{t}_{i_2}\right) \left(\mathsf{T}^\top \times \mathsf{T}^\top \times \mathsf{T}^\top\right) \tag{17}$$

with $x_i = (x_{i_0}, x_{i_1}, x_{i_2})$. Note, for the evaluation of the P2M and L2P kernels in Eqn. (9) we do not compute s_i explicitly, but we apply the matrices one after another (analogously to Eqn. (12) we implement them as matrix-matrix products). The cost for the P2M and L2P kernels scales like $\mathcal{O}(N\ell^3 + \ell^4)$ if we assume $M \sim N$.

Note. *The cost of only applying s_i in the 1d case is $\mathcal{O}(N\ell)$, however, the assembly of s_i scales like $\mathcal{O}(\ell^2)$ (for each particle). We can extend this to the 3d case, where the application only of s_i costs $\mathcal{O}(N\ell^3)$, however, the assembly costs $\mathcal{O}(\ell^6)$ floating point operations (for each particle).*

Force computation in the L2P kernel We define the force kernel $\boldsymbol{K} : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}^3$ as the gradient of the Laplace kernel $K : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}$, i.e.,

$$\boldsymbol{K}(x, y) = \frac{1}{4\pi} \frac{x - y}{|x - y|^2} = -\nabla_x K(x, y). \tag{18}$$

Hence, if we want to compute the forces $\boldsymbol{f}_i \in \mathbb{R}^3$ acting on the i-th target particle (we use Eqn. (1) and reformulate it) we have

$$\boldsymbol{f}_i \sim \sum_{m=1}^{\ell^3} \boldsymbol{P}_\ell(x_i, \bar{x}_m) \sum_{n=1}^{\ell^3} K(\bar{x}_m, \bar{y}_n) \sum_{j=1}^{N} S_\ell(y_j, \bar{y}_n) w_j \tag{19}$$

with $\boldsymbol{P}_\ell(x, \bar{x}) = -\nabla_x S_\ell(x, \bar{x})$. We notice that P2M, M2M, M2L and L2L are the same as for the computation as for the potential, only L2P changes. With the definition of

$$P_\ell(x, \bar{x}_m) = \frac{2}{\ell} \sum_{\alpha=1}^{\ell-1} \alpha U_{\alpha-1}(x) T_\alpha(\bar{x}_m) \quad \text{for } x \in [-1, 1] \tag{20}$$

in 1d and with the definition of s_i in Eqn. (17) we can define also the three components of the gradient of the i-th interpolation vector p_i as

$$\mathsf{p}_i = \begin{pmatrix} (\mathsf{u}_{i_0} \times \mathsf{t}_{i_1} \times \mathsf{t}_{i_2}) \left(\bar{\mathsf{U}}^\top \times \bar{\mathsf{T}}^\top \times \bar{\mathsf{T}}^\top\right) \\ (\mathsf{t}_{i_0} \times \mathsf{u}_{i_1} \times \mathsf{t}_{i_2}) \left(\bar{\mathsf{T}}^\top \times \bar{\mathsf{U}}^\top \times \bar{\mathsf{T}}^\top\right) \\ (\mathsf{t}_{i_0} \times \mathsf{t}_{i_1} \times \mathsf{u}_{i_2}) \left(\bar{\mathsf{T}}^\top \times \bar{\mathsf{T}}^\top \times \bar{\mathsf{U}}^\top\right) \end{pmatrix} \tag{21}$$

with $\mathsf{u}_{i_d} \in \mathbb{R}^{\ell-1}$ and $\bar{\mathsf{U}} \in \mathbb{R}^{\ell \times (\ell-1)}$ and the entries $(\mathsf{u}_{i_d})_\alpha = \alpha U_{\alpha-1}(x_{i_d})$ and $(\bar{\mathsf{U}})_{m\alpha} = {}^{2T_\alpha(\bar{x}_m)}/_{\ell}$. As always, p_i is never computed explicitly. All components are applied sequentially and are implemented as matrix-matrix products.

Note. *In our implementation of the P2M and L2P kernels we did not strictly follow the above derivation. But we use the following approach presented by means of the L2P kernel, which read as*

$$f_i = \sum_{n=1}^{\ell^3} S_\ell(x_i, \bar{x}_n) F_n \quad \textit{for } i = 1, \ldots, N,$$

with

$$S_\ell(x_i, \bar{x}_n) = \Big[\frac{1}{\ell} + \frac{2}{\ell} \sum_{\alpha=1}^{\ell-1} T_\alpha(x_{i_1}) T_\alpha(\bar{x}_{n_1})\Big] \Big[\frac{1}{\ell} + \frac{2}{\ell} \sum_{\beta=1}^{\ell-1} T_\beta(x_{i_2}) T_\beta(\bar{x}_{n_2})\Big] \Big[\frac{1}{\ell} + \frac{2}{\ell} \sum_{\gamma=1}^{\ell-1} T_\gamma(x_{i_3}) T_\gamma(\bar{x}_{n_3})\Big].$$

By reorganizing sums, the above equation can be expressed as

$$f_i = \frac{a + 2b_i + 4c_i + 8d_i}{\ell^3} \quad \textit{for } i = 1, \ldots, N,$$

with

$$\begin{aligned}
a &= \sum_{n=1}^{\ell^3} F_n, \\
b_i &= \sum_{\alpha=1}^{\ell-1} T_\alpha(x_{i_1}) \sum_{n=1}^{\ell^3} T_\alpha(\bar{x}_{n_1}) F_n + \sum_{\beta=1}^{\ell-1} T_\beta(x_{i_2}) \sum_{n=1}^{\ell^3} T_\beta(\bar{x}_{n_2}) F_n + \sum_{\gamma=1}^{\ell-1} T_\gamma(x_{i_3}) \sum_{n=1}^{\ell^3} T_\gamma(\bar{x}_{n_3}) F_n, \\
c_i &= \sum_{\alpha=1}^{\ell-1} T_\alpha(x_{i_1}) \sum_{\beta=1}^{\ell-1} T_\beta(x_{i_2}) \sum_{n=1}^{\ell^3} T_\alpha(\bar{x}_{n_1}) T_\beta(\bar{x}_{n_2}) F_n + \\
&\quad \sum_{\alpha=1}^{\ell-1} T_\alpha(x_{i_1}) \sum_{\gamma=1}^{\ell-1} T_\gamma(x_{i_3}) \sum_{n=1}^{\ell^3} T_\alpha(\bar{x}_{n_1}) T_\gamma(\bar{x}_{n_3}) F_n + \\
&\quad \sum_{\beta=1}^{\ell-1} T_\beta(x_{i_2}) \sum_{\gamma=1}^{\ell-1} T_\gamma(x_{i_3}) \sum_{n=1}^{\ell^3} T_\beta(\bar{x}_{n_2}) T_\gamma(\bar{x}_{n_3}) F_n, \\
d_i &= \sum_{\alpha=1}^{\ell-1} T_\alpha(x_{i_1}) \sum_{\beta=1}^{\ell-1} T_\beta(x_{i_2}) \sum_{\gamma=1}^{\ell-1} T_\gamma(x_{i_3}) \sum_{n=1}^{\ell^3} T_\alpha(\bar{x}_{n_1}) T_\beta(\bar{x}_{n_2}) T_\gamma(\bar{x}_{n_3}) F_n
\end{aligned}$$

The overall complexity does not change.

4 Splitting the outer product in the interpolation polynomial

We introduce the notation of the interpolation polynomial based on the notation from [2], the interpolation of a function $p(x)$ reads as

$$p(x) \sim \sum_{n=0}^{\ell-1} \gamma_n c_n T_n(x) \tag{22}$$

with $\gamma_0 = 1$ and $\gamma_n = 2$ for $n \geq 1$ and with

$$c_n = \frac{1}{\ell} \sum_{k=1}^{\ell} p(\bar{x}_n) T_k(\bar{x}_n) \tag{23}$$

After inserting Eqn. (23) into Eqn. (22) we end up with

$$p(x) \sim \frac{1}{\ell} \sum_{n=1}^{\ell} p(\bar{x}_n) \sum_{k=0}^{\ell-1} \gamma_k T_k(\bar{x}_n) T_k(x) \tag{24}$$

In the discrete case, i.e., we have N scattered points $\{x_i\}_{i=1}^N \subset [-1, 1]$, we write Eqn. (24) in matrix notation as

$$\mathsf{p} \sim \mathsf{T}_x \bar{\mathsf{T}}_x^\top \bar{\mathsf{p}} \tag{25}$$

with the matrices $\mathsf{T}_x \in \mathbb{R}^{N \times \ell}$ with $(\mathsf{T}_x)_{ik} = T_{k-1}(x_i)$ and $\bar{\mathsf{T}}_x \in \mathbb{R}^{\ell \times \ell}$ with $(\bar{\mathsf{T}}_x)_{nk} = {}^{\gamma_{k-1}}/_{\ell} T_{k-1}(\bar{x}_n)$. Using this notation the interpolated kernel function $K(x, y)$ reads as

$$\mathsf{K} \sim \mathsf{T}_x \bar{\mathsf{T}}_x^\top \bar{\mathsf{K}} (\mathsf{T}_y \bar{\mathsf{T}}_y^\top)^\top \tag{26}$$

With the low-rank representation $\bar{\mathsf{K}} \sim \mathsf{U}\mathsf{V}^T$ we compute $\bar{\mathsf{U}} = \bar{\mathsf{T}}_x^\top \mathsf{U}$ and $\bar{\mathsf{V}} = \bar{\mathsf{T}}_y^\top \mathsf{V}$ and write Eqn. (26) as

$$\mathsf{K} \sim \mathsf{T}_x \bar{\mathsf{U}} \bar{\mathsf{V}}^\top \mathsf{T}_y^\top \tag{27}$$

Can we still use symmetries? Recall the approach we proposed in Sec. 2. It exploits the symmetries in the arrangement of the far-field interactions. As we see from Eqn. (7) in that case we need permutation matrices P_t for the t-th far-field interaction and Eqn. (26) becomes

$$\mathsf{K}_t \sim \mathsf{T}_x \bar{\mathsf{T}}_x^\top (\mathsf{P}_t \bar{\mathsf{K}}_{p(t)} \mathsf{P}_t^\top)(\mathsf{T}_y \bar{\mathsf{T}}_y^\top)^\top. \tag{28}$$

With $\bar{\mathsf{K}}_{p(t)} \sim \mathsf{U}_{p(t)} \mathsf{V}_{p(t)}^\top$ the matrices in the outer product in Eqn. (27) become $\bar{\mathsf{U}}_t = \bar{\mathsf{T}}_x^\top \mathsf{P}_t \mathsf{U}_{p(t)}$ and $\bar{\mathsf{V}}_t = \bar{\mathsf{T}}_y^\top \mathsf{P}_t \mathsf{V}_{p(t)}$. Thus, it is not possible anymore to use the fact that due to symmetries we can express all 316 far-field interactions by permutations of 16 only.

References

[1] W Fong and E Darve. The black-box fast multipole method. *Journal of Computational Physics*, 228(23):8712–8725, 2009. ISSN 0021-9991. doi: DOI: 10.1016/j.jcp.2009.08.031. URL http://www.sciencedirect.com/science/article/pii/S0021999109004665.

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

[3] M S Warren and J K Salmon. A hashed Oct-Tree N-body algorithm. In *Proceedings of the 1993 ACM/IEEE conference on Supercomputing*, Supercomputing '93, pages 12–21, New York, NY, USA, 1993. ACM. ISBN 0-8186-4340-4. doi: http://doi.acm.org/10.1145/169627.169640. URL http://doi.acm.org/10.1145/169627.169640.