Data Structures, Optimal Choice of Parameters, and Complexity Results for Generalized Multilevel Fast Multipole Methods in d Dimensions

Nail A. Gumerov,[*] Ramani Duraiswami,[†] and Eugene A. Borovikov[‡]

Perceptual Interfaces and Reality Laboratory,

Institute for Advanced Computer Studies,

University of Maryland, College Park, Maryland, 20742.

Abstract

We present an overview of the Fast Multipole Method, explain the use of optimal data structures and present complexity results for the algorithm. We explain how octree structures and bit interleaving can be simply used to create efficient versions of the multipole algorithm in d dimensions. We then present simulations that demonstrate various aspects of the algorithm, including optimal selection of the clustering parameter, the influence of the error bound on the complexity, and others. The use of these optimal parameters results in a many-fold speed-up of the FMM, and prove very useful in practice.

This report also serves to introduce the background necessary to learn and use the generalized FMM code we have developed.

[*]URL: `http://www.umiacs.umd.edu/~gumerov`; Electronic address: `gumerov@umiacs.umd.edu`

[†]URL: `http://www.umiacs.umd.edu/~ramani`; Electronic address: `ramani@umiacs.umd.edu`

[‡]URL: `http://www.umiacs.umd.edu/~yab`; Electronic address: `yab@umiacs.umd.edu`

Contents

I. Introduction and scope of present work 4

A. The FMM Algorithm 5

II. Multilevel FMM 9

A. Statement of the problem and definitions 9

B. Setting up the hierarchical data structure 14

1. Generalized octrees (2^d trees) 14
2. Data hierarchies 17
3. Hierarchical spatial domains 18
4. Size of the neighborhood 20

C. MLFMM Procedure 22

1. Upward Pass 22
2. Downward Pass 24
3. Final Summation 26

D. Reduced S$|$R-translation scheme 26

1. Reduced scheme for 1-neighborhoods 26
2. Reduced scheme for 2-neighborhoods 28
3. Reduced scheme for k-neighborhoods 30

III. Data structures and efficient implementation 32

A. Indexing 33

B. Spatial Ordering 36

1. Scaling 36
2. Ordering in 1-Dimension (binary ordering) 37
3. Ordering in d-dimensions 39

C. Structuring data sets 44

1. Ordering of d-dimensional data 45
2. Determination of the threshold level 46
3. Search procedures and operations on point sets 46

IV. Complexity of the MLFMM and optimization of the algorithm 47

A. Complexity of the MLFMM procedure 47
1. Regular mesh 47
2. Non-uniform data and use of data hierarchies 50
B. Optimization of the MLFMM 51

V. Numerical Experiments 60
A. Regular Mesh of Data Points 60
B. Random Distributions 73

VI. Conclusions 77
Acknowledgments 80

References 80

VII. Appendix A 82
A. Translation Operators 82
1. R|R-operator 83
2. S|S-operator 83
3. S|R-operator 84
B. S-expansion Error 84
C. Translation Errors 85
1. R|R-translation error 85
2. S|S-translation error 86
3. S|R-translation error 87

VIII. Appendix B 88
A. S-expansion Error 88
B. S|R-translation Error 89
C. MLFMM Error in Asymptotic Model 89

I. INTRODUCTION AND SCOPE OF PRESENT WORK

Since the work of Greengard and Rokhlin (1987) [8], the Fast Multipole Method, has been established as a very efficient algorithm that enables the solution of many practical problems that were hitherto unsolvable. In the sense that it speeds up particular dense-matrix vector multiplications, it is similar to the FFT,[1] and in addition it is complementary to the FFT in that it often works for those problems to which the FFT cannot be applied. The FMM is an "analysis-based" transform that represents a fundamentally new way for computing particular dense-matrix vector products, and has been included in a list of the top ten numerical algorithms invented in the 20th century [9].

Originally this method was developed for the fast summation of the potential fields generated by a large number of sources (charges), such as those arising in gravitational or electrostatic potential problems, that are described by the Laplace equation in 2 or 3 dimensions. This lead to the name for the algorithm. Later, this method was extended to other potential problems, such as those arising in the solution of the Helmholtz [10, 11] and/or Maxwell equations [12]. The FMM has also found application in many other problems, e.g. in statistics [13–15], chemistry [16], interpolation of scattered data [17] as a method for fast summation of particular types of radial-basis functions [18, 19].

Despite its great promise and reasonably wide research application, in the authors' opinion, the FMM is not as widely used an algorithm as it should be, and is considered by many to be hard to implement. In part this may be due to the fact that this is a truly inter-disciplinary algorithm. It requires an understanding of both the properties of particular special functions such as the translation properties multipole solutions of classical equations of physics, and at the same time requires an appreciation of tree data-structures, and efficient algorithms for search. Further, most people implementing the algorithm are interested in solving a particular problem, and not in generalizing the algorithm, or explicitly setting forth the details of the algorithm in a manner that is easy for readers to implement.

In contrast in this report we will discuss the FMM in a general setting, and treat it as a method for the acceleration of particular matrix vector products, where we will not consider matrices that

[1] Unlike the FFT, for most applications of the FMM there is no fast algorithm for inverting the transform, i.e., solving for the coefficients.

arise in particular applications. Further we present a prescription for implementing data-structures that ensure efficient implementation of the FMM, and establish a clean description and notation for the algorithm. Such a description is the proper setting to determine optimal versions of the FMM for specific problems (variation of the algorithm with problem dimension, with clustered data, and for particular types of functions ϕ). These issues are both crucial to the implementation of the algorithm, and its practical complexity.

Further, these issues are also a significant hurdle for those not familiar with data-structures, such as engineers, applied mathematicians and physicists involved in scientific computation. Of course, some of these details can also be gleaned from the standard texts on spatial data-structures [1, 2], albeit not in the context of the FMM. A final issue is that our descriptions are not restricted to 2 or 3 dimensional problems, as is usual, but are presented in the d-dimensional context. This allows the same “shell” of the algorithm to be used for multiple problems, with only the translation and the function routines having to be changed.

This report will only deal with what we refer to as the “regular multilevel FMM,” that employs regular hierarchical data structures in the form of quad-trees in 2-D, octrees in 3-D, and their higher dimensional generalizations. In a later report we hope to deal with a new adaptive FMM method we have developed that achieves even better performance, by working with the point distributions of both the source data and evaluation data, generalizing the multilevel adaptive FMM work that has been presented in the literature (e.g., Cheng at al, 1999).

A. The FMM Algorithm

We first present a short informal description of the FMM algorithm for matrix-vector multiplication, before introducing the algorithm more formally. Consider the sum, or matrix vector product

$$v_j = v(\mathbf{y}_j) = \sum_{i=1}^{N} u_i \phi_i(\mathbf{y}_j), \quad j = 1, ..., M, \qquad [\mathbf{\Phi}] \{\mathbf{u}\} = \{\mathbf{v}\}. \tag{1}$$

Direct evaluation of the product requires $O(MN)$ operations.

In the FMM, we assume that the functions ϕ that constitute the matrix can be expanded as local (regular) series or multipole (singular) series that are centered at locations $\mathbf{y}_*$ and $\mathbf{x}_*$ as follows:

$$\phi(\mathbf{y}) = \sum_{q=0}^{p-1} a_q(\mathbf{y}_*) R_q\left(\mathbf{y} - \mathbf{y}_*\right) + \epsilon\left(p\right), \quad \phi(\mathbf{y}) = \sum_{q=0}^{p-1} b_q(\mathbf{x}_*) S_q\left(\mathbf{y} - \mathbf{x}_*\right) + \epsilon\left(p\right),$$

where R_q and S_q are local (regular) and multipole (singular) basis functions, $\mathbf{x}_*$ and $\mathbf{y}_*$ are expansion centers and a_q, b_q are the expansion coefficients.

The Middleman method, which is applicable only to functions that can be represented by a single uniform expansion everywhere (say an expansion in terms of the R basis above). In this case we can perform the summation efficiently by first performing a p term expansion for each of the N functions, ϕ_i, at a point $\mathbf{x}_*$ in the domain (e.g., near the center) requiring $O(Np)$ operations.

$$v(\mathbf{y}_j) = \sum_{i=1}^{N} u_i \phi_i(\mathbf{y}_j) = \sum_{i=1}^{N} u_i \sum_{q=0}^{p-1} c_{qi} R_q\left(\mathbf{y}_j - \mathbf{x}_*\right), \quad j = 1, ..., M, \tag{2}$$

We consolidate the N series into one p term series, by rearranging the order of summation, and summing all the coefficients over the index i requiring $O(Np)$ additions:

$$v(\mathbf{y}_j) = \sum_{i=1}^{N} u_i \phi_i(\mathbf{y}_j) = \sum_{q=0}^{p-1} \left[\sum_{i=1}^{N} u_i c_{qi}\right] R_q\left(\mathbf{y}_j - \mathbf{x}_*\right) = \sum_{q=0}^{p-1} C_q R_q\left(\mathbf{y}_j - \mathbf{x}_*\right) = \mathbf{C} \circ \mathbf{R}(\mathbf{y}_j - \mathbf{x}_*).$$

The single consolidated p terms series can be evaluated at all the M evaluation points of the domain in $O(Mp)$ operations. The total number of operations required is then $O(Mp + Np) \simeq O(Np)$ for $N \sim M$. The truncation number p depends on the desired accuracy alone, and is independent of M, N.

We can recognize that the trick that allows the order of summation to be changed, will work with any representation that has scalar product structure. We can represent the above summation order trick as

$$v(\mathbf{y}_j) = \sum_{i=1}^{N} u_i \phi_i(\mathbf{y}_j) = \mathbf{C} \circ \mathbf{R}(\mathbf{y}_j - \mathbf{x}_*) \quad j = 1, ..., M, \quad \mathbf{C} = \sum_{i=1}^{N} u_i \mathbf{A}\left(\mathbf{x}_i - \mathbf{x}_*\right), \tag{3}$$

where $\mathbf{x}_*$ is an arbitrary point.

Unfortunately such single series are not usually available, and we may have many local or far-field expansions. This leads to the idea of the FMM. In the FMM we construct the R and S expansions, around centers of expansions, and add the notion of translating series. We assume we have translation operators that relate the expansion coefficients in different bases, e.g.

$$
\begin{aligned}
a_q(\mathbf{y}_{*2}) &= (R|R)\,(\mathbf{y}_{*2}, \mathbf{y}_{*1})\,[a_r(\mathbf{y}_{*1})]\,, \\
b_q(\mathbf{x}_{*2}) &= (S|S)\,(\mathbf{x}_{*2}, \mathbf{x}_{*1})\,[b_r(\mathbf{x}_{*1})]\,, \\
a_q(\mathbf{y}_{*}) &= (S|R)\,(\mathbf{y}_{*}, \mathbf{x}_{*})\,[b_r(\mathbf{y}_{*})]\,,
\end{aligned}
$$

where $(R|R)\,,(S|S)\,,$ and $(S|R)$ denote translation operators that transform the coefficients between respective bases.

Given these series representations and translation operators, the FMM proceeds as follows. First the space is partitioned into boxes at various levels, and outer S expansions computed about box centers at the finest level, for points within the box. These expansions are consolidated, and they are translated $S|S$ using translations in an *upward pass* up the hierarchy. The coefficients of these box-centered expansions at each level are stored. In the *downward pass*, the consolidated S expansions are expanded as local R expansions about boxes in the evaluation hierarchy, using the $S|R$ translation, for boxes for which the expansion is valid (it is in the domain of validity of the particular S expansion). At finer levels, the R expansions at the higher levels are $R|R$ translated to the new box centers and to these are added the coefficients of the $S|R$ translations from boxes at finer levels of the source hierarchy, which were excluded at the previous level(s). At the finest level of the evaluation hierarchy we have R expansions about the box centers, and very few points for which valid expansions could not be constructed. These are evaluated directly and added to the R expansions evaluated at the evaluation points. Schematically the MLFMM, straightforward and Middleman methods for matrix-vector multiplication are shown in Figure 1.

The FMM is thus a method of grouping and translation of functions generated by each source to reduce the asymptotic complexity of finding the sum (3) approximately. The straightforward computation of that sum requires $O(MN)$ operations, whereas the FMM seeks to approximately compute these in $O(M+N)$ or $O(M+N\log N)$ operations, by using the factorization and translation properties of $\phi_i(\mathbf{y})$. To achieve this, the original (classical) FMM utilizes a grouping based on 2^d-tree space subdivision. In the following we provide methods for doing this when both the evaluation points and sources lie in the $d-$dimensional hypercube $D\times\ D\times\ ...\times\ D$. We will set up the notation and framework to discuss optimizations of the algorithm possible, and characterize the complexity of these versions of the algorithm.

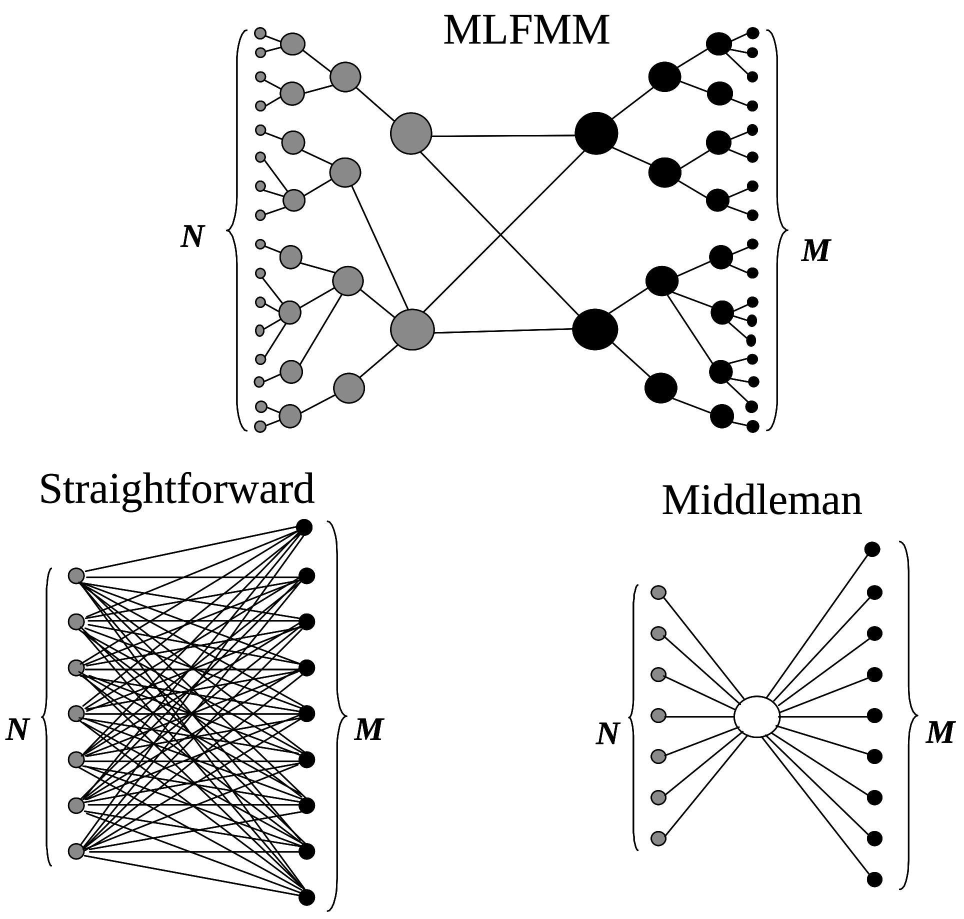


FIG. 1: Schematic of different methods that can be used for multiplication of $M \times N$ dense matrix by a vector of length N. In the straightforward method the number of operations is $O(MN)$, because of the N sources is evaluated at the M evaluation points. The number of operations (connecting lines) can be reduced to $O(M+N)$ for the MLFMM and Middleman methods. The Middleman method can be used in a limited number of cases when there exists a uniformly valid expansion for the entire computational domain. The MLFMM is more general, since it also can be to the case of functions for which expansions only in subdomains are applicable. The scheme for MLFMM shows only S$|$R translations on the coarsest level, S$|$R translations from source hierarchy to the evaluation hierarchy at finer levels are not shown for simplicity of illustration.

II. MULTILEVEL FMM

We present a formal description of the basic spatial grouping operations involved in the implementation of the fast multipole method and consider efficient methods for their implementation using 2^d-tree data-structures.

A. Statement of the problem and definitions

The FMM can be considered as a method for achieving fast multiplication of vectors with dense matrices that have special structure, e.g., their elements can be written as $\phi_{ji} = \phi(\mathbf{x}_i, \mathbf{y}_j)$,where $\mathbf{x}_i$ and $\mathbf{y}_j$ are points in d-dimensional Euclidean space. These points are usually called the "set of sources", $\mathbb{X}$, and the set of "evaluation points" or "targets" $\mathbb{Y}$:

$$\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}, \quad \mathbf{x}_i \in \mathbb{R}^d, \quad i = 1, ..., N, \tag{4}$$

$$\mathbb{Y} = \{\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_M\}, \quad \mathbf{y}_j \in \mathbb{R}^d, \quad j = 1, ..., M. \tag{5}$$

In general one seeks to evaluate the sum, or matrix-vector product

$$v_j = v(\mathbf{y}_j) = \sum_{i=1}^{N} u_i \phi_i(\mathbf{y}_j), \quad j = 1, ..., M, \tag{6}$$

where u_i are scalars (can be complex), and the contribution of a single source located at $\mathbf{x}_i \in \mathbb{R}^d$ is described by a function

$$\phi_i(\mathbf{y}) : \mathbb{R}^d \to \mathbb{R}, \quad \mathbf{y} \in \mathbb{R}^d, \quad i = 1, ..., N. \tag{7}$$

Common examples of functions ϕ_i arising in FMM applications are $\phi_i(\mathbf{y}) = |\mathbf{y} - \mathbf{x}_i|^{-1}$ (the fundamental solution of the 3D Laplace equation) or $\phi_i(\mathbf{y}) = f(|\mathbf{y} - \mathbf{x}_i|)$ (a radial basis function in d-dimensions).

For use with the FMM the functions $\phi_i(\mathbf{y})$ must have the following properties:

- **Local expansion** (also called inner or regular expansion). The function $\phi_i(\mathbf{y})$ can be evaluated directly or via the series representation near an arbitrary spatial point $\mathbf{x}_* \neq \mathbf{x}_i$ as

$$\phi_i(\mathbf{y}) = \mathbf{A}_i(\mathbf{x}_*) \circ \mathbf{R}(\mathbf{y} - \mathbf{x}_*), \quad |\mathbf{y} - \mathbf{x}_*| \leqslant r_c |\mathbf{x}_i - \mathbf{x}_*|, \quad i = 1, ..., N. \tag{8}$$

Here the series is valid in the domain $|\mathbf{y}-\mathbf{x}_*| \leqslant r_c |\mathbf{x}_i - \mathbf{x}_*|$ (see Fig. 2), where $0 < r_c < 1$ is some real number. In general r_c depends on the function ϕ_i and its properties. $\mathbf{R}$ and $\mathbf{A}$ are the basis functions and the series coefficients, represented as tensor objects in p-dimensional space $\mathbb{R}^p$ (for example, vectors of length p) and $\circ$ denotes a contraction operation between these objects, distributive with addition (for example, the scalar product of vectors of length p):

$$(u_{i_1}\mathbf{A}_{i_1} + u_{i_2}\mathbf{A}_{i_2}) \circ \mathbf{R} = u_{i_1}\mathbf{A}_{i_1} \circ \mathbf{R} + u_{i_2}\mathbf{A}_{i_2} \circ \mathbf{R}, \quad i_1, i_2 = 1, ..., N. \tag{9}$$

Concerning the tensor function $\mathbf{R}(\mathbf{y}-\mathbf{x}_*)$, which can be interpreted as a set of basis functions, we assume that it is regular at $\mathbf{y} = \mathbf{x}_*$, and that the local expansion is valid inside a d-dimensional sphere centered at $\mathbf{y} = \mathbf{x}_*$ with radius $r_c |\mathbf{x}_i - \mathbf{x}_*|$. The tensor object $\mathbf{A}_i(\mathbf{x}_*)$,can be interpreted as a set of expansion coefficients for the basis.

- **Far field expansion** (also called outer, singular, or multipole expansion). Any function $\phi_i(\mathbf{y})$ has a complementary expansion valid outside a d-dimensional sphere centered at $\mathbf{y} = \mathbf{x}_*$ with radius $R_c |\mathbf{x}_i - \mathbf{x}_*|$:

$$\phi_i(\mathbf{y}) = \mathbf{B}_i(\mathbf{x}_*) \circ \mathbf{S}(\mathbf{y}-\mathbf{x}_*), \quad |\mathbf{y}-\mathbf{x}_*| \geqslant R_c |\mathbf{x}_i - \mathbf{x}_*|, \tag{10}$$

where $R_c > 1$ is a real number similar to r_c, and the tensor function $\mathbf{S}(\mathbf{y}-\mathbf{x}_*)$ provides a basis for the outer domain, and $\circ$ is an operation similar to that in Eq. (8), that is distributive with addition (see Eq. (9)). Even though for many physical fields, such as the Green's function for Laplace's equation, the function $\mathbf{S}(\mathbf{y}-\mathbf{x}_*)$ is singular at $\mathbf{y} = \mathbf{x}_*$, this condition is not necessary. In particular we can have $\mathbf{S} = \mathbf{R}$.

The domains of validity of the expansions are shown in Figure 2.

- **Translations.**

The function $\phi_i(\mathbf{y})$ may be expressed both as a far-field and a near-field expansion (series) as in Equations (8) and (10) that are centered at a particular location $\mathbf{x}_*$.The function can also be expressed as in terms of a basis centered at another center of expansion $\mathbf{x}_{*1}$. Both representations evaluate to the same value in their domains of validity. The conversion of one representation, in one

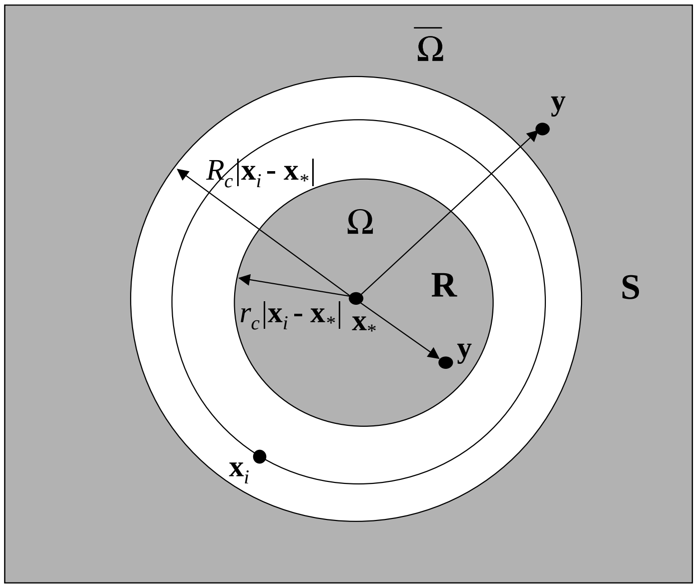


FIG. 2: Domains of validity of the regular (R, local) and singular (S, far field) expansions.

coordinate system with a particular center, to another representation in another coordinate system, with another center is termed as a **translation**. The translation operator is a *linear* operator that can be defined either in terms of the functions being translated or in terms of the coefficients of their representations in a suitable local basis. The definition of the operator $\mathcal{T}$ in terms of functions is

$$\widehat{\psi}(\mathbf{y}) = \mathcal{T}(\mathbf{t})\,[\psi(\mathbf{y})] = \psi(\mathbf{y}+\mathbf{t}), \tag{11}$$

where $\mathbf{t}$ is the translation vector, and $\widehat{\psi}$ is translation transform of ψ. For functions expandable over the bases $\mathbf{R}$ and $\mathbf{S}$ the action of the translation operators can also be represented by the action of linear transforms on the space of coefficients. Depending on the expansion basis we consider the following three types of translations, which are employed in the multilevel FMM:

1. **Local-to-local** (see Fig. 3)**.** Consider the local expansion (8) near the point $\mathbf{x}_{*1}$, which is valid for any $\mathbf{y} \in \Omega_{1i}$, $\Omega_{1i} : |\mathbf{y}-\mathbf{x}_{*1}| \leqslant r_c\,|\mathbf{x}_i-\mathbf{x}_{*1}|$. If we choose a new center for the local expansion, $\mathbf{x}_{*2} \in \Omega_{1i}$, for $\mathbf{y} \in \Omega_{2i} \subset \Omega_{1i}$, where Ω_{2i} is a sphere,

$$\Omega_{2i} : |\mathbf{y}-\mathbf{x}_{*2}| \leqslant r_c\,|\mathbf{x}_i-\mathbf{x}_{*1}| - |\mathbf{x}_{*1}-\mathbf{x}_{*2}|\,, \tag{12}$$

then the set of expansion coefficients transforms as

$$\mathbf{A}_i\left(\mathbf{x}_{*2}\right) = (\mathbf{R}|\mathbf{R})\left(\mathbf{x}_{*2}-\mathbf{x}_{*1}\right)\left[\mathbf{A}_i\left(\mathbf{x}_{*1}\right)\right], \tag{13}$$

where $(\mathbf{R}|\mathbf{R})\left(\mathbf{x}_{*2}-\mathbf{x}_{*1}\right)$ is the local-to-local translation operator (or regular-to-regular, denoted by the symbol $\mathbf{R}|\mathbf{R}$). Note that since $0<r_c<1$, we have

$$r_c\left|\mathbf{x}_i-\mathbf{x}_{*1}\right|=r_c\left|\mathbf{x}_i-\mathbf{x}_{*2}+\left(\mathbf{x}_{*2}-\mathbf{x}_{*1}\right)\right|\leqslant r_c\left|\mathbf{x}_i-\mathbf{x}_{*2}\right|+r_c\left|\mathbf{x}_{*1}-\mathbf{x}_{*2}\right|<r_c\left|\mathbf{x}_i-\mathbf{x}_{*2}\right|+\left|\mathbf{x}_{*1}-\mathbf{x}_{*2}\right|.$$

Therefore, $\left|\mathbf{x}_{*1}-\mathbf{x}_{*2}\right|>r_c\left|\mathbf{x}_i-\mathbf{x}_{*1}\right|-r_c\left|\mathbf{x}_i-\mathbf{x}_{*2}\right|$, and the condition $\mathbf{y}\in\Omega_{2i}\subset\Omega_{1i}$ yields

$$\left|\mathbf{y}-\mathbf{x}_{*2}\right|\leqslant r_c\left|\mathbf{x}_i-\mathbf{x}_{*1}\right|-\left|\mathbf{x}_{*1}-\mathbf{x}_{*2}\right|<r_c\left|\mathbf{x}_i-\mathbf{x}_{*1}\right|-r_c\left|\mathbf{x}_i-\mathbf{x}_{*1}\right|+r_c\left|\mathbf{x}_i-\mathbf{x}_{*2}\right|=r_c\left|\mathbf{x}_i-\mathbf{x}_{*2}\right|.$$

The condition $\mathbf{y}\in\Omega_{2i}\subset\Omega_{1i}$ is sufficient for validity of the local expansion (8) near point $\mathbf{x}_i=\mathbf{x}_{*2}$.

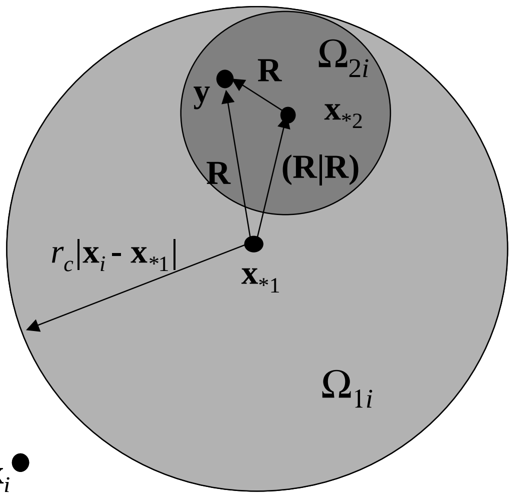


FIG. 3: Local-to-local (or regular-to-regular) translation.

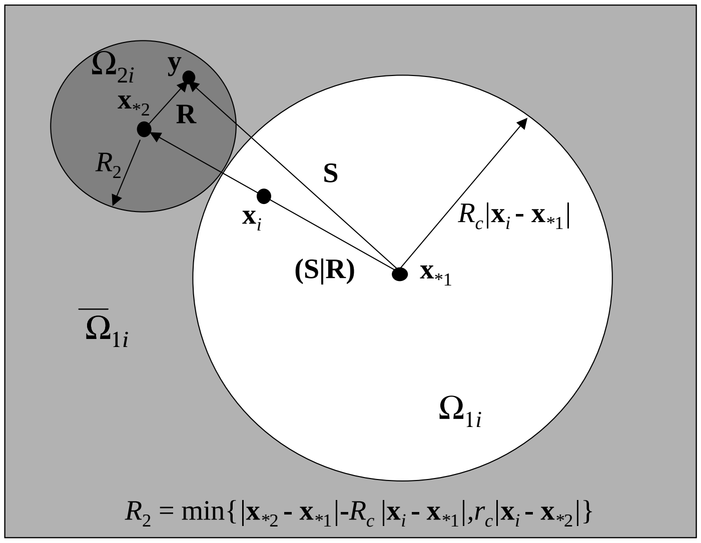
2. **Far-to-local** (see 4). Similarly, consider the far-field expansion (10) near the point $\mathbf{x}_{*1}$, which is valid for any $\mathbf{y}\in\overline{\Omega}_{1i}$, $\overline{\Omega}_{1i}:\left|\mathbf{y}-\mathbf{x}_{*1}\right|\geqslant R_c\left|\mathbf{x}_i-\mathbf{x}_{*1}\right|$ and select a center for a local expansion at $\mathbf{x}_{*2}\in\overline{\Omega}_{1i}$. Then for $\mathbf{y}\in\Omega_{2i}\subset\overline{\Omega}_{1i}$, where Ω_{2i} is the sphere,

$$\Omega_{2i}:\left|\mathbf{y}-\mathbf{x}_{*2}\right|\leqslant\min(\left|\mathbf{x}_{*2}-\mathbf{x}_{*1}\right|-R_c\left|\mathbf{x}_i-\mathbf{x}_{*1}\right|,r_c\left|\mathbf{x}_i-\mathbf{x}_{*2}\right|), \quad (14)$$

the set of expansion coefficients transforms as

$$\mathbf{A}_i\left(\mathbf{x}_{*2}\right)=(\mathbf{S}|\mathbf{R})\left(\mathbf{x}_{*2}-\mathbf{x}_{*1}\right)\left[\mathbf{B}_i\left(\mathbf{x}_{*1}\right)\right], \quad (15)$$

where $(\mathbf{S}|\mathbf{R})(\mathbf{x}_{*2}-\mathbf{x}_{*1})$ is the *far-to-local translation operator* (or singular-to-regular, denoted by the symbol $\mathbf{S}|\mathbf{R}$). Note that the condition $|\mathbf{y}-\mathbf{x}_{*2}| \leqslant |\mathbf{x}_{*2}-\mathbf{x}_{*1}| - R_c|\mathbf{x}_i-\mathbf{x}_{*1}|$ provides that $\Omega_{2i} \subset \overline{\Omega}_{1i}$, and therefore the far field expansion is valid in Ω_{2i}. The condition $|\mathbf{y}-\mathbf{x}_{*2}| \leqslant r_c|\mathbf{x}_i-\mathbf{x}_{*2}|$ ensures that the local expansion is valid in Ω_{2i}.



Ω_{2i}
$\mathbf{y}$
$\mathbf{x}_{*2}$
$\mathbf{R}$
R_2
$\mathbf{S}$
$\mathbf{x}_i$
$R_c|\mathbf{x}_i - \mathbf{x}_{*1}|$
$(\mathbf{S}|\mathbf{R})$
$\mathbf{x}_{*1}$
$\overline{\Omega}_{1i}$
Ω_{1i}
$R_2 = \min\{|\mathbf{x}_{*2} - \mathbf{x}_{*1}| - R_c|\mathbf{x}_i - \mathbf{x}_{*1}|, r_c|\mathbf{x}_i - \mathbf{x}_{*2}|\}$

FIG. 4: Far-to-local (or singular-to-regular) translation.

3. **Far-to-far** (see Fig. 5)**.** Finally, consider the far field expansion (10) near the point $\mathbf{x}_{*1}$, which is valid for any $\mathbf{y} \in \overline{\Omega}_{1i}$, $\overline{\Omega}_{1i} : |\mathbf{y}-\mathbf{x}_{*1}| \geqslant R_c|\mathbf{x}_i-\mathbf{x}_{*1}|$ and select a center $\mathbf{x}_{*2} \in \Omega_{2i}$ for another far field expansion, where Ω_{2i} is a sphere that includes Ω_{1i}, $\Omega_{2i} \supset \Omega_{1i}$. The far field expansion near $\mathbf{x}_{*1}$ can be translated to the far field expansion near $\mathbf{x}_{*2}$, if the evaluation point $\mathbf{y} \in \overline{\Omega}_{2i}$, where $\overline{\Omega}_{2i}$ is the external region of sphere Ω_{2i}, such that :

$$\overline{\Omega}_{2i} : |\mathbf{y}-\mathbf{x}_{*2}| \geqslant R_c|\mathbf{x}_{*2}-\mathbf{x}_{*1}| + R_c|\mathbf{x}_i-\mathbf{x}_{*1}|. \tag{16}$$

The set of expansion coefficients then translates as

$$\mathbf{B}_i(\mathbf{x}_{*2}) = (\mathbf{S}|\mathbf{S})(\mathbf{x}_{*2}-\mathbf{x}_{*1})[\mathbf{B}_i(\mathbf{x}_{*1})], \tag{17}$$

where $(\mathbf{S}|\mathbf{S})(\mathbf{x}_{*2}-\mathbf{x}_{*1})$ is the far-to-far field (or singular-to-singular, denoted by the symbol $\mathbf{S}|\mathbf{S}$) translation operator. Note that

$$R_c|\mathbf{x}_{*2}-\mathbf{x}_{*1}| + R_c|\mathbf{x}_i-\mathbf{x}_{*1}| \geqslant R_c|\mathbf{x}_{*2}-\mathbf{x}_{*1}-(\mathbf{x}_i-\mathbf{x}_{*1})| = R_c|\mathbf{x}_{*2}-\mathbf{x}_i|.$$

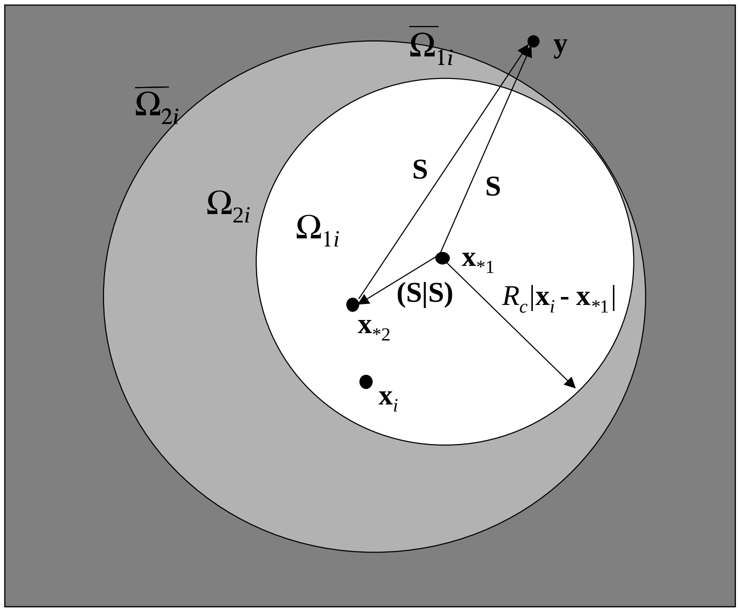


FIG. 5: Far-to-far (or singular-to-singular) translation.

Therefore, the definition of Ω_{2i} provides $|\mathbf{y}-\mathbf{x}_{*2}| \geqslant R_c |\mathbf{x}_{*2}-\mathbf{x}_i|$, and the condition for the validity of the far field expansion near a new center is satisfied.

As mentioned above the translation operators are linear, which means that

$$u_i\mathbf{C}_i\left(\mathbf{x}_{*2}\right)+u_k\mathbf{C}_k\left(\mathbf{x}_{*2}\right)=\left(\mathbf{E}|\mathbf{F}\right)\left(\mathbf{x}_{*2}-\mathbf{x}_{*1}\right)\left[u_i\mathbf{D}_i\left(\mathbf{x}_{*1}\right)+u_k\mathbf{D}_k\left(\mathbf{x}_{*1}\right)\right], \quad \mathbf{E},\mathbf{F}=\mathbf{R},\mathbf{S}, \quad (18)$$

where $\mathbf{D}_i$ and $\mathbf{D}_k$ are the expansion coefficients for expansions centered at $\mathbf{x}_{*1}$ with respect to the basis $\mathbf{E}$, $\mathbf{C}_i$ and $\mathbf{C}_k$ are the expansion coefficients for expansions centered at $\mathbf{x}_{*2}$ with respect to the basis $\mathbf{F}$, and u_i, u_k are constants.

B. Setting up the hierarchical data structure

1. Generalized octrees (2^d trees)

One of the most important properties of d-dimensional Euclidean space ($\mathbb{R}^d$) is that it can be subdivided into rectangular boxes (we are mostly concerned with cubes). In practice, the problems we are concerned with are posed on finite domains, which can then be enclosed in a bounding box. We assign this bounding box to level 0, in a hierarchical division scheme. The level 0 box can be subdivided into 2^d smaller boxes of equal size by dividing each side in half. All boxes of this size are assigned to level 1. Repeating this procedure, we produce a sequence of boxes at level 2, level 3, and so on. While this process of sub-division could continue for ever, in practice we would stop at some finite level $l_{\max}$, which is determined by some criterion (e.g., that there are at least s particles in a box at the finest level). Figure 6, left, illustrates this for $d = 2$. By the process of division we obtain a 2^d-tree (see Figure 7), in which each node corresponds to a box. Any two nodes at different levels are connected in the tree if the box corresponding to the first node at the finer level is obtained by subdivision of the box corresponding to the second node at the coarser level. At level l of a 2^d-tree we have 2^{ld} boxes, with each node having the index as n, with n ranging from 0 to $2^{ld} - 1$. Therefore any box in a 2^d-tree can be characterized by the pair (n, l) .

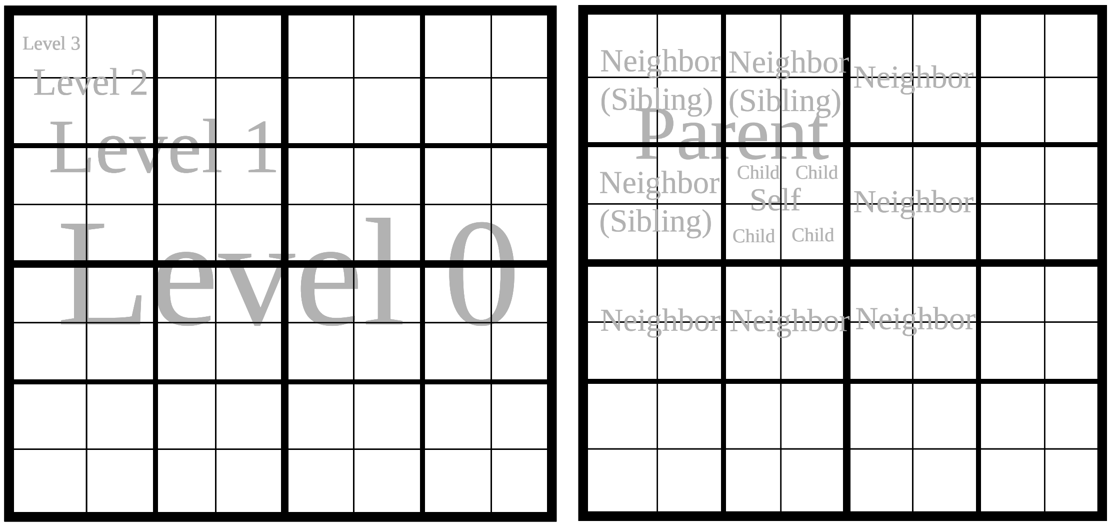


FIG. 6: The left graph shows levels in quad-tree space subdivision. The right graph shows children, parent, siblings, and neighbors of the box marked as "self".

A 2^d-tree graph clearly displays "parent-child" relationships, where the "children" boxes at level $l + 1$ are obtained by subdivision of a "parent" box at level l. For a 2^d-tree with $l_{\max}$ levels, any box at level $l \geqslant 1$ has exactly one parent, and any box at level $l \leqslant l_{\max} - 1$ has exactly 2^d children. So we can define operations $Parent\,(n, l)$, which returns the index of the parent box, and $ChildrenAll(n, l)$ that returns the indexes of the children boxes. The children of the same parent are called "siblings". Each box at a given level $l \geqslant 1$ has $2^d - 1$ siblings.

In the FMM we are also interested in neighbor relationships between boxes. These are determined exclusively by the relative spatial locations of the boxes, and not by their locations in the tree. We call two different boxes “neighbors” (or 1-neighbors) if their boundaries have at least one common point. We also define 2-neighbors, 3-neighbors, and so on. Two different boxes are 2-neighbors if they are not 1-neighbors, but they have at least one common 1-neighbor. Two different boxes are 3-neighbors if they are not 1-neighbors, and not 2-neighbors, while at least one of the neighbors of each box is a 2-neighbor of the other box, and so on up to the “k-neighborhood”, $k = 1, 2, ...$ By induction the k-neighborhood is a union of two sets: the ($k - 1$)-neighborhood of the box and all its k-neighbors. We also use the terminology “power of a set” and “power of a neighborhood” to denote the number of boxes in a particular set and number of boxes in a particular neighborhood, respectively. For example, the power of the 0-neighborhood is 1.

The number of neighbors that a given box has in a finite 2^d-tree space subdivision depends on its location relative to the boundary of the domain (the boundaries of the box at level 0). For example a box at level $l \geqslant 1$ in a quadtree situated at the corner of the largest box has only three neighbors, while a box situated far from the boundaries (indicated as “self” in Figure 6, right) has 8 neighbors. The number of neighbors depends on the dimension d. In the general d-dimensional case the minimum and maximum numbers of neighbors are

$$N_{\min}^{(Neighbors)}(d) = 2^d - 1, \quad N_{\max}^{(Neighbors)}(d) = 3^d - 1. \tag{19}$$

The minimum number of neighbors is achieved for a box in the corner, for which all neighbors are children of the same parent (siblings). Since the number of siblings is $2^d - 1$ this provides the minimum number of neighbors . The maximum number of neighbors is for a box located far from the boundary. Consider a box not on the boundary at a higher level. It has right and left neighbors in each dimension, and can be considered the central box of a cube divided into $3\times 3\times \ldots \times 3 = 3^d$ sub-boxes, which is the power of the 1-neighborhood. Excluding the box itself from this count we obtain the number of its neighbors as in Eq. (19).

Equations (19) show that the number of neighbors for large d far exceeds the number of siblings. The neighbor relationships are not easily determined by position on the 2^d-tree graph (see Figure 7) and potentially any two boxes at the same level could be neighbors. On this graph the neighbors can be close to each other (siblings), or very far apart, so that a connecting path between them may have to go through a higher node (even through the node at level 0). For further consideration we can introduce operation $NeighborsAll^{(k)}(n, l)$ which returns indexes of all k-neighbors of the box (n, l).

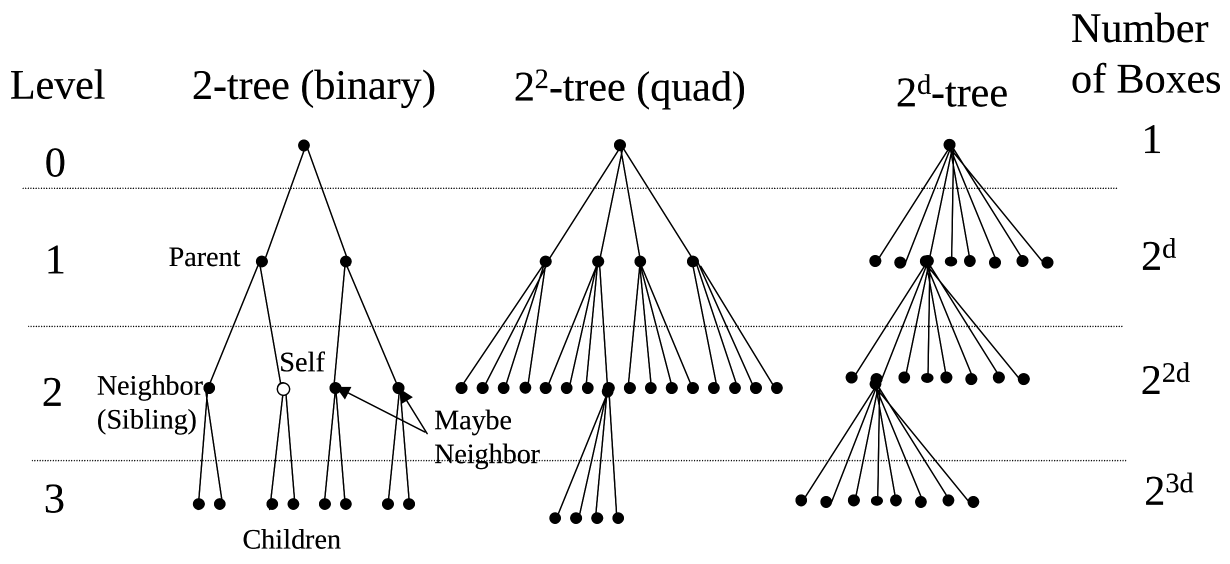


FIG. 7: 2^d-trees and terminology.

The example above shows that the power of k-neighborhood depends on the level and on the location of the box. The maximum value of the power of k-neighborhood is

$$MaxPower(k\text{-}neighborhood) = (2k+1)^d. \tag{20}$$

This shows that at fixed d the power depends polynomially on k, while at fixed k it depends exponentially on the dimension d.

2. *Data hierarchies*

The 2^d-trees provide a space subdivision without any consideration for the distribution of the source and evaluation data points (4) and (5). These data can be structured with 2^d-trees and organized in the X-data hierarchy (or source hierarchy) and the Y-data hierarchy (or evaluation/target hierarchy) according to the coordinates of the source and evaluation points. We prescribe to each source or evaluation point the index of the box (n, l) to which it belongs, so that X and Y are sets of indices (n, l).

For each data hierarchy we define the operations $Parent(n, l)$, $Children(n, l)$, and $Neighbors^{(k)}(n, l)$. The operation $Parent(n, l)$ is the same for both hierarchies, since the parent of each box already contain points of the hierarchy. The other two operations return the sets of children and k-neighbor boxes at levels $l+1$ and l, respectively, that contain the points from the particular hierarchies. To discriminate between the two sets, we denote them as $Children(X;n,l)$, and $Neighbors^{(k)}(X;n,l)$ for the X-hierarchy and $Children(Y;n,l)$, and $Neighbors^{(k)}(Y;n,l)$ for the Y-hierarchy.

3. *Hierarchical spatial domains*

We define notation here that will permit a succinct description of the FMM algorithm, and further allow for its optimization. By optimization we mean the selection of parameters, e.g., one of the parameters to be chosen is the number of points, s, that are contained at the finest level $(l_{\max})$ in a non-empty box.

We define the following four spatial domains that are used in the FMM. These can be defined for each box with index $n = 0, ..., 2^{ld}-1$ at level $l = 0, ..., l_{\max}$, and have fractal structure[2]:

- $E_1(n,l) \subset \mathbb{R}^d$ denotes spatial points *inside* the box (n,l).
- $E_2^{(k)}(n,l) \subset \mathbb{R}^d$ denotes spatial points in the k-neighborhood ($k = 1, 2, ...$) of box (n,l).
- $E_3^{(k)}(n,l) = E_1(0,0) \backslash E_2^{(k)}(n,l)$ denotes spatial points *outside* the k-neighborhood ($k = 1, 2, ...$) of box (n,l).
- $E_4^{(k)}(n,l) = E_2^{(k)}(Parent(n,l), l-1) \backslash E_2^{(k)}(n,l)$ denotes spatial points in the k-neighborhood of the parent box $(Parent(n,l), l-1)$,which do not belong to the k-neighborhood of the parent box itself.

We, thus associate with each other the sets of boxes at level l that constitute each of the domains $E_\alpha(n,l)$, $\alpha = 1, ..., 4$, which we denote as $I_\alpha(n,l)$. Boxes $I_\alpha(n,l) \cap X$ and $I_\alpha(n,l) \cap Y$ belong to the X and Y hierarchies, respectively. Figure 8 illustrates these domains in the case $d = 2$ and $k = 1$. Each FMM algorithm could in principle be based on one of these k-neighborhoods, though most FMM algorithms published thus far have used 1-neighborhoods.

[2] By fractal we mean that these structures have the same shape at different levels of the hierarchy.

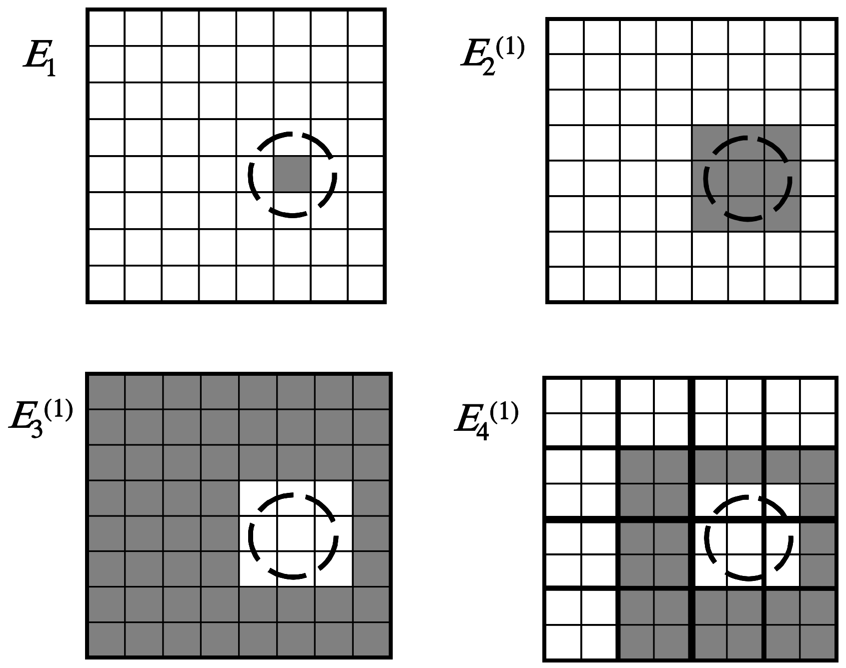


FIG. 8: The domains used for construction of hierarchical reexpansion procedure in FMM ($d = 2, k = 1$). A circle separates the box for which domains are drawn.

To choose $l_{\min}$, the level from which we start the FMM, for an implementation based on a $k-$neighborhood, we note that at $l_{\min}$ there should be at least one non-empty box outside the $E_2^{(k)}(n, l_{\min})$ neighborhood, while at level $l_{\min} - 1$ the domain $E_1(0, 0)$ resides completely inside the $E_2^{(k)}(n, l_{\min} - 1)$ domain. This happens if $2^{l_{\min}-1} < k + 1$. Thus $l_{\min}$ can be calculated as the integer part of $1 + \log_2(k + 1)$:

$$l_{\min} = [1 + \log_2(k + 1)]\,. \tag{21}$$

For $k = 1$ this results in $l_{\min} = 2$.

Based on these domains, the following functions can be defined for each box:

$$v_{n,l}^{(m)}(\mathbf{y}) = \sum_{\mathbf{x}_i \in E_m(n,l)} u_i \phi_i(\mathbf{y}), \quad m = 1, ..., 4. \tag{22}$$

Note that since the domains $E_2(n, l)$ and $E_3(n, l)$ are complementary, we have from (6) and (22):

$$v(\mathbf{y}) = v_{n,l}^{(2)}(\mathbf{y}) + v_{n,l}^{(3)}(\mathbf{y}), \tag{23}$$

for arbitrary l and n.

4. Size of the neighborhood

The size of the neighborhood, k, must be determined before running the MLFMM procedure. The choice is based on the space dimensionality and parameters r_c and R_c, which specify the regions of expansion validity.

1. The S-expansion (10) near the center of the nth box at level l for $\mathbf{x}_i \in E_1(n,l)$ is valid for any $\mathbf{y}$ in the domain $E_3^{(k)}(n,l)$. In d-dimensional space the maximum distance from the center of the unit box to its boundary is $d^{1/2}/2$ and the minimum distance from the center to the boundary of its k-neighborhood domain ($E_3^{(k)}(n,l)$) is $(2k+1)/2$. Therefore k should be selected so that
$$k \geqslant \frac{1}{2}\left(R_c d^{1/2} - 1\right). \tag{24}$$
For example, for $d = 3$, and $R_c = 1.5$ this condition yields $k \geqslant 0.799...$ so $k = 1$ can be used, while for $d = 3$, $R_c = 2$ we have $k \geqslant 1.232...$ and the minimum integer k that satisfies Eq. (24) is 2.

2. The R-expansion (8) near the center of the nth box at level l for $\mathbf{x}_i \in E_3^{(k)}(n,l)$ is valid for any $\mathbf{y}$ from the domain $E_1(n,l)$. A similar calculation as that lead to Eq. (24) leads to the following selection criteria for k:
$$k \geqslant \frac{1}{2}\left(\frac{1}{r_c} d^{1/2} - 1\right). \tag{25}$$
Therefore these two requirements will be satisfied simultaneously if
$$k \geqslant \frac{1}{2}\left[\max\left(\frac{1}{r_c}, R_c\right) d^{1/2} - 1\right]. \tag{26}$$

3. The $S|S$-translation (17) of the S-expansion from the center of the nth box at level l for $\mathbf{x}_i \in E_1(n,l)$ to the center of its parent box preserves the validity of the S-expansion for any $\mathbf{y}$ from the domain $E_3^{(k)}(Parent(n,l), l-1)$. Eq. (16) shows that this condition is satisfied if Eq. (24) holds.

4. The $R|R$-translation (13) of the R-expansion from the center of the nth box at level l for $\mathbf{x}_i \in E_1(n,l)$ to the centers of its children preserves the validity of the R-expansion for any $\mathbf{y}$ from the domain $E_1(ChildrenAll(n,l), l+1)$. Eq. (12) shows that this condition is satisfied if Eq. (25) holds.

5. The $S|R$-translation (15) of the S-expansion from the center of the mth box at level l which belongs to $E_4^{(k)}(n,l)$ for $\mathbf{x}_i \in E_1(m,l)$ to the center of the box $E_1(n,l)$ provides a valid R-expansion for any $\mathbf{y}$ from the domain $E_1(n,l)$. If the size of the box at level l is 1, then the minimum distance between the centers of boxes (m,l) (say $\mathbf{x}_{*1}$) and (n,l) (say $\mathbf{x}_{*2}$), is $k+1$. The maximum $|\mathbf{x}_i - \mathbf{x}_{*1}|$ is $d^{1/2}/2$, the minimum $|\mathbf{x}_i - \mathbf{x}_{*2}|$ is $(2k+1)/2$, and the maximum $|\mathbf{y} - \mathbf{x}_{*2}|$ is $d^{1/2}/2$. Thus, condition (14) will be satisfied if

$$d^{1/2} \leqslant \min\left\{2k+2-R_c d^{1/2}, (2k+1)\, r_c\right\}. \tag{27}$$

This also can be rewritten as

$$k \geqslant \frac{1}{2}\max\left\{\left[(R_c+1)\, d^{1/2}-2\right], \left[\frac{1}{r_c}d^{1/2}-1\right]\right\}. \tag{28}$$

Combining Eq. (28) and Eq. (26) we obtain the following general condition for the selection of the neighborhood size

$$k \geqslant \frac{1}{2}\max\left\{\left[(R_c+1)\, d^{1/2}-2\right], \left[\max\left(\frac{1}{r_c}, R_c\right) d^{1/2}-1\right]\right\}. \tag{29}$$

For example, for $d=3$, $R_c=1.5$, $r_c=1$, this condition yields $k \geqslant 1.165...$, so $k=2$ is the minimal k that satisfies all the requirements. The first requirement yields $k=1$ for the same situation, so Eq. (29) is more restrictive.

It is also useful to consider the relation between the size of the neighborhood and the dimensionality of the problem. Setting the maximum value of R_c and the minimum of r_c (both unity) we obtain from Eq. (29):

$$k > d^{1/2}-1. \tag{30}$$

This shows that 1-neighborhoods are suitable for problems with dimensionality $d=1,2,3.$ [3] For $r_c < 1$ and $R_c > 1$ and $4 \leqslant d < 9$ the minimum allowable neighborhood is the 2-neighborhood, for $9 \leqslant d < 16$ one should operate with at least 3-neighborhoods, and so on.

It is also useful to have an idea of the acceptable R_c and r_c for given k and d. If we assume that $\max\left(r_c^{-1}, R_c\right) = R_c$ then Eq. (29) and the condition $R_c > 1$ yield for $d \geqslant 1$:

$$1 < R_c \leqslant \frac{2(k+1)}{d^{1/2}} - 1. \tag{31}$$

This shows, e.g., that

[3] Thus it is fortunate that all FMM studies reported in the literature have been in dimensionalities $d \leq 3$.

Neighborhood type (k)	Dimensionality (d)	Convergence radius, R_c
1	1	≤ 3
1	2	$\leq 2^{3/2}-1 \approx 1.8284$
1	3	$\leqslant 4/3^{1/2}-1 \approx 1.3094$

If the expansion (10) holds for larger R_c than is permitted by the minimum k for a given dimensionality, then the size of the neighborhood should be increased. E.g., for $d=3$ one can increase k to 2 to extend R_c to $R_c \leqslant 6/3^{1/2}-1 \approx 2.4641$. Note that this type of neighborhood was also considered by Greengard in his dissertation [6], where the MLFMM was developed for 3D Laplace equation, but has not been used by others since.

C. MLFMM Procedure

Assuming that condition (29) holds, so that all the translations required for the FMM can be performed. The FMM procedure consists of an Upward Pass, which is performed for each box at level $l_{\max}$ up to the level $l_{\min}$ of the X-hierarchy and uses the S-expansions for these boxes, in the two step Downward Pass, which is performed for each box from level $l_{\min}$ down to level $l_{\max}$ of the Y-hierarchy and uses R-expansions for boxes of this hierarchy, and the Final Summation. Translations within the X-hierarchy are $S|S$-reexpansions, within the Y-hierarchy are $R|R$-reexpansions, and translations from the boxes of the X-hierarchy to the boxes of the Y-hierarchy are the $S|R$-reexpansions.

1. Upward Pass

Step 1. For each box $(n, l_{\max})$ in the X-hierarchy, generate the coefficients of the S-expansion , $\mathbf{B}_i\left(\mathbf{x}_c^{(n,l_{\max})}\right)$, for the function $v_{n,l_{\max}}^{(1)}(\mathbf{y})$ centered at the box-center, $\mathbf{x}_c^{(n,l_{\max})}$. Multiply these with the associated scalar in the vector being multiplied, and consolidate the coefficients of all source points in the box to determine the expansion coefficients $\mathbf{C}^{(n,l_{\max})}$ corresponding to that box,

$$\mathbf{C}^{(n,l_{\max})} = \sum_{\mathbf{x}_i \in E_1(n,l_{\max})} u_i \mathbf{B}_i\left(\mathbf{x}_c^{(n,l_{\max})}\right), \quad (n, l_{\max}) \in X, \tag{32}$$
$$\left[v_{n,l_{\max}}^{(1)}(\mathbf{y}) = \mathbf{C}^{(n,l_{\max})} \circ \mathbf{S}(\mathbf{y}-\mathbf{x}_c^{(n,l_{\max})})\right].$$

Note that since the S-expansion is valid outside a region containing the center, because of (24) the expansion (32) for the nth box is valid in the domain $E_3^{(k)}(n, l_{\max})$ (see Fig. 8).

Step 2. Repeat for $l = l_{\max} - 1, ..., l_{\min}$. For each box (n, l) in the X-hierarchy recursively determine the expansion coefficients $\mathbf{C}^{(n,l)}$ of the function $v_{n,l}^{(1)}(\mathbf{y})$ by reexpanding $v_{Children(X;n,l),l+1}^{(1)}(\mathbf{y})$ near the center of box (n, l) and summing up the contribution of all the child boxes:

$$\mathbf{C}^{(n,l)} = \sum_{n' \in Children(X;n,l)} (\mathbf{S}|\mathbf{S}) \left(\mathbf{x}_c^{(n',l+1)} - \mathbf{x}_c^{(n,l)} \right) \mathbf{C}^{(n',l+1)}, \quad (n, l) \in X, \tag{33}$$

$$\left[v_{n,l}^{(1)}(\mathbf{y}) = \mathbf{C}^{(n,l)} \circ \mathbf{S}(\mathbf{y} - \mathbf{x}_c^{(n,l)}) \right].$$

For the nth box, this expansion is valid in the domain $E_3^{(k)}(n, l)$ which is a subdomain of $E_3^{(k)}(Children(X; n, l), l + 1)$, and the far-to-far translation is applicable (see the requirement #3 above and Eq. (24)). Figure 9 illustrates this for the case $d = 2$, $k = 1$. Indeed the spheres that enclose each child box are themselves enclosed by the larger sphere around the parent box. Thus for the domain $E_3^{(k)}(n, l)$, shaded in dark gray (see Fig. 5), $(\mathbf{S}|\mathbf{S})$-translation is applicable.

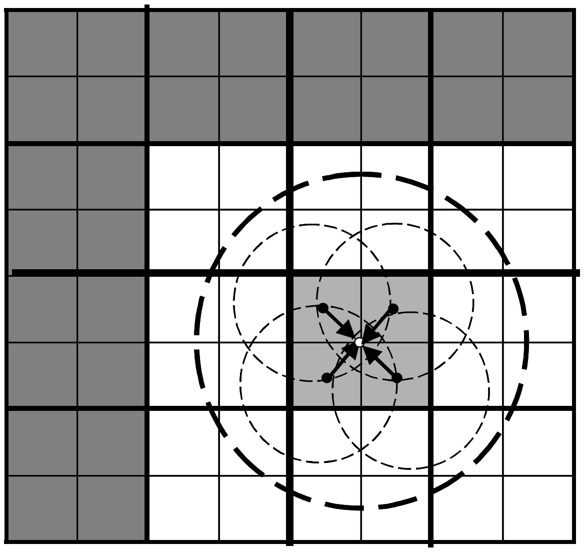


FIG. 9: Step 2 of the FMM upward pass. The S-expansion near the center of each box can be obtained by $(\mathbf{S}|\mathbf{S})$-translations of expansions centered at centers of its children.

2. *Downward Pass*

The downward pass applies steps 1 and 2 below for each of the levels $l = l_{\min}, ..., l_{\max}$.

Step 1. In this step we form the coefficients, $\widetilde{\mathbf{D}}^{(n,l)}$, of the regular expansion for the function $v_{n,l}^{(4)}(\mathbf{y})$ about the center of box $(n,l) \in Y$. To build the local expansion near the center of each box at level l,the coefficients $\mathbf{C}^{(n',l)}, n' \in I_4^{(k)}(n,l) \cap X$ should be $(\mathbf{S}|\mathbf{R})$- translated to the center of the box. Thus we have

$$\widetilde{\mathbf{D}}^{(n,l)} = \sum_{n' \in I_4^{(k)}(n,l) \cap X} (\mathbf{S}|\mathbf{R}) \left(\mathbf{x}_c^{(n',l)} - \mathbf{x}_c^{(n,l)} \right) \mathbf{C}^{(n',l)}, \quad (n,l) \in Y, \tag{34}$$
$$\left[v_{n,l}^{(4)}(\mathbf{y}) = \widetilde{\mathbf{D}}^{(n,l)} \circ \mathbf{R}(\mathbf{y} - \mathbf{x}_c^{(n,l)}) \right].$$

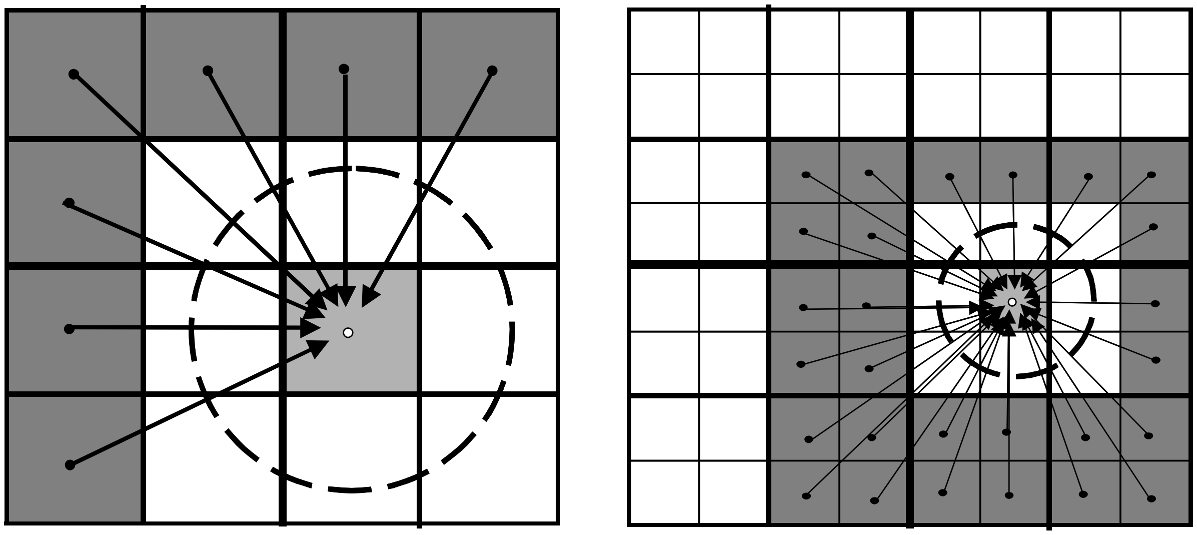


FIG. 10: Step 1 of the downward pass of the FMM. The coefficients of singular expansion corresponding to dark gray boxes are **(S|R)**-translated to the center of the light gray box. Figures illustrate this step for quadtree at levels 2 and 3.

Condition (27) ensures that the far-to-local translation (15) is applicable. Figure 10 illustrates this step for $d = 2$ and $k = 1$.

Step 2. Assuming that for $l = l_{\min}$

$$\mathbf{D}^{(n,l_{\min})} = \widetilde{\mathbf{D}}^{(n,l_{\min})} \quad (n, l_{\min}) \in Y, \quad \left[v^{(3)}_{n,l_{\min}}(\mathbf{y}) = v^{(4)}_{n,l_{\min}}(\mathbf{y})\right], \tag{35}$$

we form the coefficients of the regular expansion $\mathbf{D}^{(n,l)}$ for the function $v^{(3)}_{n,l}(\mathbf{y})$ about the box center $(n,l) \in Y$, by adding $\widetilde{\mathbf{D}}^{(n,l)}$ to the coefficients obtained by $(\mathbf{R}|\mathbf{R})$- translation of $v^{(3)}_{Parent(n,l),l-1}(\mathbf{y})$ from the parent box to the center of the child box (n,l):

$$\mathbf{D}^{(n,l)} = \widetilde{\mathbf{D}}^{(n,l)} + (\mathbf{R}|\mathbf{R})\left(\mathbf{x}_c^{(n',l-1)} - \mathbf{x}_c^{(n,l)}\right)\mathbf{D}^{(n',l-1)}, \quad n' = Parent(n,l), \quad (n,l) \in Y,$$
$$\left[v^{(3)}_{n,l}(\mathbf{y}) = \mathbf{D}^{(n,l)} \circ \mathbf{R}(\mathbf{y} - \mathbf{x}_c^{(n,l)})\right], \quad l = l_{\min}+1, ..., l_{\max}. \tag{36}$$

For the nth box, this expansion is valid in the domain $E_1(n,l)$ which is a subdomain of

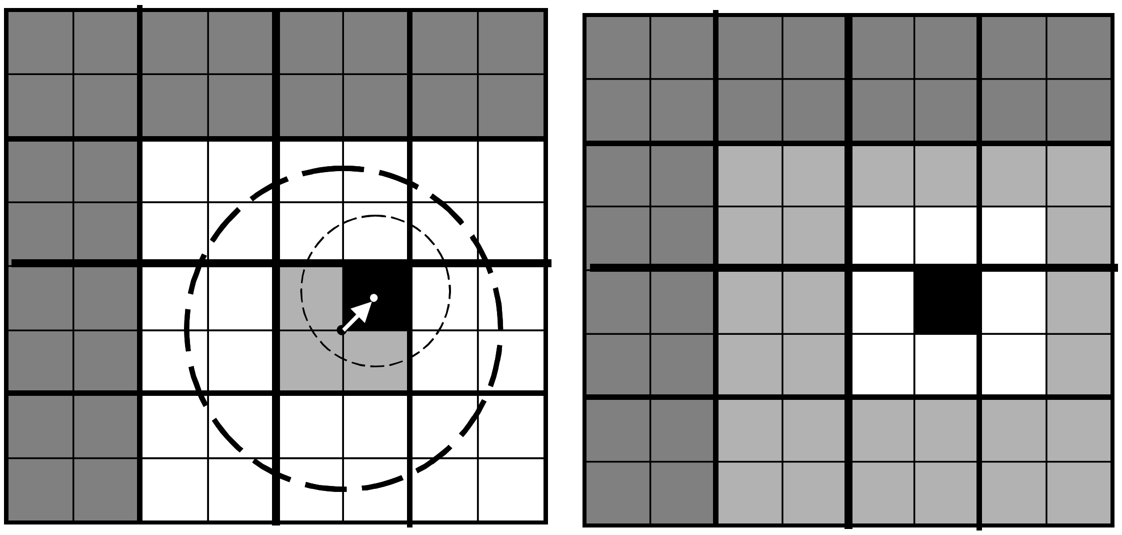


FIG. 11: Step 2 of the downward pass of the FMM. On the left figire the coefficients of the parent box (light gray) are locally translated to the center of the black box. On the right figure contribution of the light grey boxes is added to the sum of the dark boxes to repeat the structure at the finer hierarchical level.

$E_1(Parent(n,l), l-1)$, and the local-to-local translation is allowed (see the requirement #4 above and Eq. (25)). Figure 11 illustrates this for $d = 2$, $k = 1$. Indeed the smaller sphere is located completely inside the larger sphere), and junction of domains $E_3^{(k)}(n,l)$ and $E_4^{(k)}(n,l+1)$ produces $E_3^{(k)}(n,l+1)$:

$$E_3^{(k)}(n,l+1) = E_3^{(k)}(n,l) \cup E_4^{(k)}(n,l+1). \tag{37}$$

3. *Final Summation*

As soon as coefficients $\mathbf{D}^{(n,l_{\max})}$ are determined, the total sum $v(\mathbf{y}_j)$ can be computed for any point $\mathbf{y}_j \in E_1(0,0)$ using Eq. (23), where $v^{(2)}_{n,l}(\mathbf{y})$ can be computed in a straightforward way, using Eq. (22). Thus

$$v(\mathbf{y}_j) = \sum_{\mathbf{x}_i \in E_2(n,l_{\max})} \phi_i(\mathbf{y}_j) + \mathbf{D}^{(n,l_{\max})} \circ \mathbf{R}(\mathbf{y}_j - \mathbf{x}_c^{(n,l_{\max})}), \quad \mathbf{y}_j \in E_1(n, l_{\max}). \tag{38}$$

D. Reduced S|R-translation scheme

Step 1 in the downward pass is the most expensive step in the algorithm since it requires a large number of translations (from each box $n' \in I_4^{(k)}(n,l) \cap X$). The maximum number of such translations per box can be evaluated using Eq. (20) as

$$\begin{aligned} \max N_{S|R} &= 2^d \max Power(k - neighborhood) - \max Power(k - neighborhood) \\ &= \left(2^d - 1\right)(2k+1)^d, \end{aligned} \tag{39}$$

where the first term in the difference represents the number of boxes at level l that belongs to the k-neighborhood of the parent box and the second term is the number of boxes in the k-neighborhood of the box itself.

We can substantially reduce the number of translations if we note that some translations are performed from all the children boxes of a parent. For such boxes we can use the already known S-expansion coefficients from the coarser level. However, such an operation reduction trick requires an additional analysis to ensure that the expansion domains are valid.

1. *Reduced scheme for 1-neighborhoods*

Let us subdivide the set $I_4^{(k)}(n,l)$ into two sets

$$\begin{aligned} I_4^{(1)}(n,l) &= I_{41}^{(1)}(n,l) \cup I_{42}^{(1)}(n,l), \\ Parent\left(I_{41}^{(1)}(n,l)\right) &\cap Parent\left\{Neighbor^{(1)}(n,l)\right\} \neq \emptyset, \\ Parent\left(I_{42}^{(1)}(n,l)\right) &\cap Parent\left\{Neighbor^{(1)}(n,l)\right\} = \emptyset. \end{aligned} \tag{40}$$

where $I_{41}^{(1)}(n,l)$ is the set of boxes at level l whose parent boxes include boxes from the set $\left\{Neighbor^{(1)}(n,l)\right\}$, and $I_{42}^{(1)}(n,l)$ is the set of boxes whose parents do not include the neighbor boxes of (n,l) (see Figure 12). Thus all boxes in the set $I_4^{(1)}(n,l)$ can be grouped according to their parents. The set $Parent\left(I_{42}^{(1)}(n,l)\right)$ is a set of boxes of the coarser level, $l-1$, that are located in the domain $E_4^{(1)}(n,l)$, but is separated from the box (n,l). Therefore, instead of Eq. (34) we can write

$$\begin{aligned}\widetilde{\mathbf{D}}^{(n,l)} &= \sum_{n'\in I_{41}^{(1)}(n,l)\cap X} (\mathbf{S}|\mathbf{R})\left(\mathbf{x}_c^{(n',l)}-\mathbf{x}_c^{(n,l)}\right)\mathbf{C}^{(n',l)} + \\ &\sum_{n'\in Parent\left(I_{42}^{(1)}(n,l)\right)\cap X} (\mathbf{S}|\mathbf{R})\left(\mathbf{x}_c^{(n',l-1)}-\mathbf{x}_c^{(n,l)}\right)\mathbf{C}^{(n',l-1)}, \quad (n,l)\in Y.\end{aligned} \tag{41}$$

Note that for level $l=2$ the latter sum in Eq. (41) should be set to zero, since the set $Parent\left(I_{42}^{(1)}(n,l)\right)$ is empty.

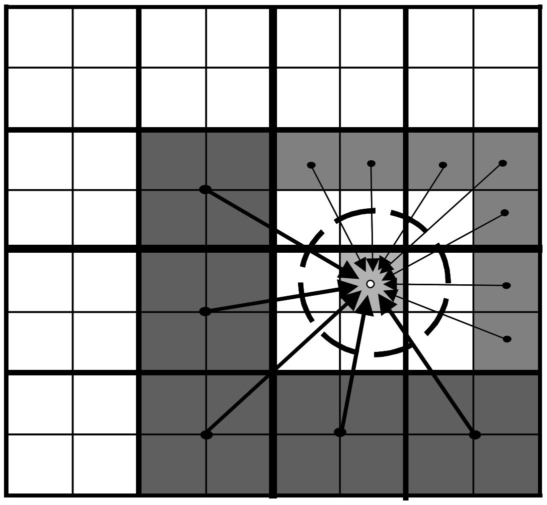


FIG. 12: Reduced scheme with 1-neighborhoods for the step 1 of the downward pass of the FMM. The coefficients of the singular expansion corresponding to dark gray boxes are **(S|R)**-translated to the center of the light gray box. For optimization the parent level coefficients can be used for boxes shown in deep dark gray. This step is illustrated for a quadtree at levels 2 and 3.

Consider now the requirement for the validity of the $S|R$-translation, Eq. (15), for such a reduced scheme. If the size of the box at level l is 1, then the minimum distance between the centers of boxes $(m,l-1)$, which is the 1-neighbor of $Parent\,(n,l)$ (say $\mathbf{x}_{*1}$) and (n,l) (say

$\mathbf{x}_{*2}$), is $\left\{\left[\frac{1}{2}(d-1)^{1/2}\right]^2+\left[\frac{5}{2}\right]^2\right\}^{1/2}$. The maximum distance $|\mathbf{x}_i-\mathbf{x}_{*1}|$ is $d^{1/2}$, the minimum distance $|\mathbf{x}_i-\mathbf{x}_{*2}|$ is $3/2$ and the maximum distance $|\mathbf{y}-\mathbf{x}_{*2}|$ is $d^{1/2}/2$. The condition (14) will thus be satisfied if

$$d^{1/2}\leqslant\min((d+24)^{1/2}-2R_c d^{1/2},3r_c). \tag{42}$$

Even for $R_c\to 1$ this requirement can be satisfied only if $3d^{1/2}<(d+24)^{1/2}$, i.e. for $d=1$ and $d=2$. Hence the reduced scheme for 1-neighborhood is applicable only for these low dimensions. For $d=1$ it permits us to make two $S|R$-translations instead of three for the regular translation scheme with $1<R_c\leqslant 2$. For $d=2$ it permits us to make twelve $S|R$-translations instead of the 27 needed for the regular translation scheme with $1<R_c\leqslant\left(13^{1/2}-1\right)/2\approx 1.3028$. Note that the reduction of the range of possible convergence radii R_c (the maximum R_c in the reduced scheme is 2 instead of 3 for $d=1$ and 1.3028 instead of 1.8284 for $d=2$) is a cost that one has to pay for the use of the reduced scheme.

2. *Reduced scheme for 2-neighborhoods*

Figure 13 illustrates the idea of the reduced translation scheme for 2-neighborhoods. Instead of translating expansions from each gray box to the black box in the left figure, one can reduce the number of translations by translating expansions from the centers of the parent boxes (shown in darker gray) and smaller number of boxes at the same level (shown in lighter gray).

Again the total sum can be subdivided into two parts corresponding to the boxes at the same level and the boxes at the parent level. The major issue here is determining the domains of validity of the $S|R$-translation (15). Extending the geometrical consideration to this case, we find that for unit size boxes at level l, the minimum distance between the centers of boxes $(n',l-1)$, which is the 2-neighbor of $Parent\,(n,l)$, say $\mathbf{x}_{*1}$, and (n,l), say $\mathbf{x}_{*2}$, is $\left\{\left[\frac{1}{2}(d-1)^{1/2}\right]^2+\left(\frac{7}{2}\right)^2\right\}^{1/2}$. The maximum distance $|\mathbf{x}_i-\mathbf{x}_{*1}|$ is $d^{1/2}$, the minimum distance $|\mathbf{x}_i-\mathbf{x}_{*2}|$ is $5/2$ and the maximum distance $|\mathbf{y}-\mathbf{x}_{*2}|$ is $d^{1/2}/2$. The condition (14) will be satisfied if

$$d^{1/2}\leqslant\min((d+48)^{1/2}-2R_c d^{1/2},5r_c). \tag{43}$$

The maximum possible space dimension can be found by letting $R_c\to 1$, which yields $3d^{1/2}<(d+48)^{1/2}$, and that the reduced translation scheme is valid for $d<6$. The maximum number of

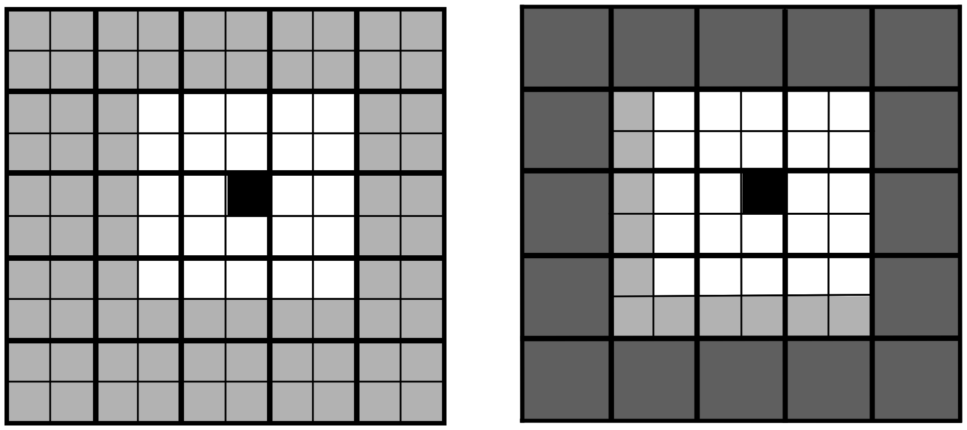


FIG. 13: Regular (left) and reduced (right) schemes with 2-neighborhoods for step 1 of the downward pass of the FMM. The coefficients of the singular expansion corresponding to the gray boxes are **(S|R)**-translated to the center of the black box. For optimization the parent level coefficients can be used for boxes shown in deep dark gray.

translations using the reduced scheme is $5^d - 3^d + 2^d 3^d - 5^d = 3^d \left(2^d - 1\right)$, which is the same as the number of translations with the regular scheme with 1-neighborhoods, Eq. (39), and less than the regular scheme with 2-neighborhoods by $(5/3)^d$ times. This can be a substantial saving for larger $d,$ e.g. for $d = 3$ since this reduces the number of translations from 875 to 189, or more than 4.6 times.

As in the case with 1-neighborhoods, the reduction in the number of translations is accompanied by a reduction in the allowed dimensionality (the maximum d is 5 for the reduced scheme and 8 for the regular scheme), and the maximum radius of expansion validity, R_c (e.g. at $d = 3$ we have $R_c \leqslant 2.4641$ for the regular scheme, see Eq. (31), and $R_c \leqslant \left(17^{1/2} - 1\right)/2 \approx 1.5616$ for the reduced scheme). However, we note that this range is larger than the range of allowable R_c for the regular scheme using 1-neighborhoods, for which we have $R_c \leqslant 1.3094.$ Taking into account that the number of translations is the same for the reduced scheme with 2-neighborhoods and regular scheme with 1-neighborhoods, we find that better convergence properties for the same amount of operations can be achieved if the reduced scheme is used.

3. Reduced scheme for k-neighborhoods

When we use FMM algorithms based on larger neighborhoods (k) the difference between the k-neighborhood of the parent box (level $l-1$) and the box itself (level l) can be large enough, so that we can group boxes not only of the parent level, but also boxes belonging to levels $l-2, l-3$, and so on. Figure 14 illustrates three possible schemes of translations with 3-neighborhoods, the regular (left), reduced with maximum size boxes at the parent level (center) and the maximum size boxes at the parent of parent (or grandparent) level.

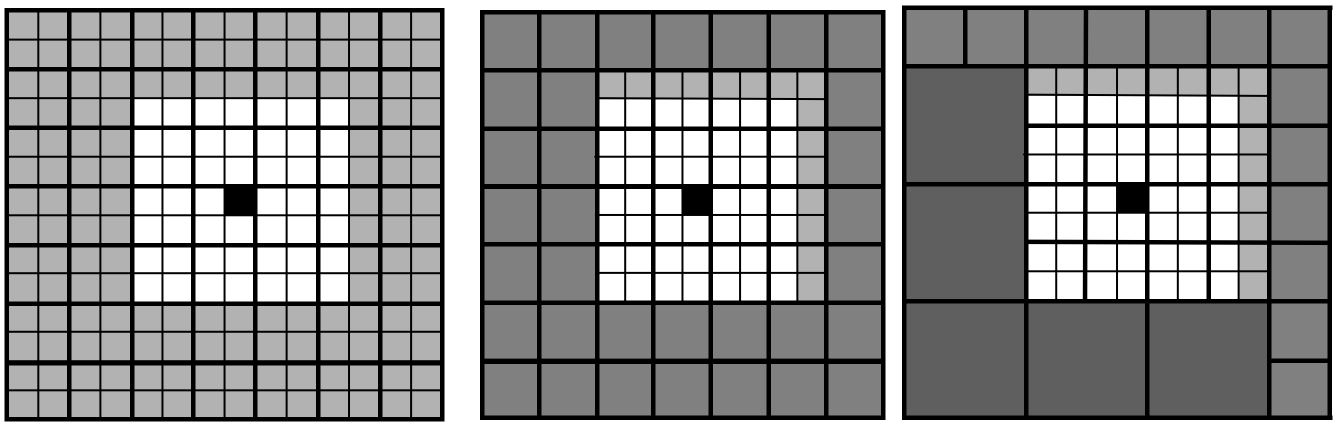


FIG. 14: Regular (left) and reduced to parent level (center) and to grandparent level (right) schemes with 3-neighborhoods for the step 1 of the downward pass of the FMM.

To check the validity of such schemes, first we note that the minimum distance from the unit size box (n,l) to the boundary of its $E_2^{(k)}(n,l)$ domain is k, while the distance to the boundary of its $E_4^{(k)}(n,l)$ domain is $2k+1$. The difference is $k+1$ and so the box of level l' can fit in this space if $l-l' \leqslant \log_2(k+1)$. The size of the box at this level is $2^{l-l'} = 2^{[\log_2(k+1)]}$ and we can consider limitations for the space dimensionality and the convergence radius for the $S|R-$translation to be performed from the box of size 2^m, $0 \leqslant m \leqslant l-l'$ located right near the boundary of $E_2^{(k)}(n,l)$.

Denoting the center of this box as $\mathbf{x}_{*1}$ and the center of box (n,l) as $\mathbf{x}_{*2}$ we find that the minimum distance between the centers is $\left\{\left[\frac{1}{2}(2^m-1)(d-1)^{1/2}\right]^2 + \left(\frac{1}{2}2^m+k+\frac{1}{2}\right)^2\right\}^{1/2}$. The maximum distance $|\mathbf{x}_i - \mathbf{x}_{*1}|$ is $\frac{1}{2}2^m\, d^{1/2}$, the minimum distance $|\mathbf{x}_i - \mathbf{x}_{*2}|$ is $(2k+1)/2$ and the maximum distance $|\mathbf{y}-\mathbf{x}_{*2}|$ is $d^{1/2}/2$. Condition (14) will be satisfied if

$$d^{1/2} \leqslant \min(\left[(d-1)(2^m-1)^2 + (2^m+2k+1)^2\right]^{1/2} - 2^m R_c d^{1/2}, (2k+1)\, r_c). \qquad (44)$$

First, we can consider the dimensionality limits. Assuming $R_c \to 1$, we obtain

$$d < \frac{2(k+1)(k+2^m)}{2^{2m}+1}, \quad k = 1, 2, ..., \quad m = 0, ..., [\log_2 k]. \tag{45}$$

The allowable dimension as a function of m, the size of the box allowed, $d(m)$, at fixed k decays monotonically. We also recall that the maximum m that can be achieved is $m = \log_2 k$. For a reduced scheme using this value we obtain

$$d < \frac{4k(k+1)}{k^2+1}, \tag{46}$$

which shows that in this case one should not expect that the reduced scheme will work for dimensions larger then $d = 3$. Therefore, for problems in larger dimensions one should select $m < \log_2 k$.

Furthermore, we can obtain the following relation for the convergence radius as a function of d, k, and m :

$$R_c \leqslant 2^{-m} \left\{ \left[(2^m - 1)^2 + \frac{4(k+1)(k+2^m)}{d} \right]^{1/2} - 1 \right\} \quad k = 1, 2, ..., \quad m = 0, ..., [\log_2 k]. \tag{47}$$

This relation shows that the range of allowable R_c decreases when m and d increase, and increases when k increases. In terms of reduction of the number of operations for given d and R_c, k and m should be selected to achieve the minimum possible k and maximum possible m.

We also note that for larger k there exist different possibilities for reduction of the number of operations. Such schemes can be composed from boxes of levels $l, l-1, l-2$, and so on. The general trend is that larger d and R_c can be treated when the scheme is designed so that smaller boxes are located closer to the k-neighborhood of the box (n, l) and larger boxes are located closer to the boundary of the domain $E_4^{(k)}(n, l)$.

The problem of efficient computational treatment of cases with larger dimensionality is still an open one, since an increase in d leads to an increase of k, and the power of the k-neighborhood (20) increases exponentially with d. Such cases can be treated however, if the number of sources in such neighborhoods is much smaller than their maximum power, which is typical for problems in higher dimensions. What is also typical for high d problems is that points usually cluster in subspaces of lower dimensionality. In this case special grouping techniques that may include local rotations can be considered as candidates for reduction of the number of operations.

III. DATA STRUCTURES AND EFFICIENT IMPLEMENTATION

Our goal is to achieve the matrix-vector multiplication, or sum, in Eq. (6) in $O(N)$ or $O(N \log N)$ operations. Accordingly all methods used to perform indexing and searching for parent, children, and neighbors in the data hierarchies should be consistent with this. It is obvious that methods based on naive traversal algorithms have asymptotic complexity $O(N^2)$ and are not allowed since this would defeat the purpose of the FMM. There is also the issue of memory complexity. If the size of the problem is small enough, then the neighbor search procedures on trees can be $O(1)$ and the complexity of the MLFMM procedure can be $O(N)$. More restrictive memory requirements bring the complexity of such operations to $O(\log N)$ and the complexity of the MLFMM procedure to $O(N \log N)$. Reduction of this complexity to $O(N)$ can be achieved in some cases using hashing (this technique however depends on the properties of the source and evaluation data sets and may not always result in savings) [20].

From the preceding discussion, 2^d-tree data structures are rather natural for use with MLFMM. Since, the regions of expansion validity are specified in terms of Euclidean distance, subdivision of space into d-dimensional cubes is convenient for range evaluation between points. We note that in the spatial data-structure literature, the data-structures used most often for higher dimensional spaces are k-d trees (e.g., see [1, 2]). Such structures could also be employed in the MLFMM, especially for cases when expansions are tensor products of expansions with respect to each coordinate, however no such attempts have been reported to our knowledge. We also can remark that 2^d-tree data structures can be easily generated from k-d data structures, so methods based on k-d trees can be used for the MLFMM. The relative merits of these and other spatial data-structures for the FMM remain a subject for investigation.

The main technique for working with 2^d-trees (and k-d trees) is the bit-interleaving technique (perhaps, first mentioned by Peano in 1890 [3], see more details and the bibliography in [1, 2]) which we apply in d-dimensions. This technique enables $O(1)$, or constant, algorithms for parent and sibling search and $O(\log N)$ algorithms for neighbor and children search. Using the bit interleaving technique the time complexity for the MLFMM is provide $O(N \log N)$ in case we wish to minimize the amount of memory used. If we are able to store the occupancy maps for the given data sets we can obtain $O(N)$ complexity.

While these algorithms are well known in the spatial data-structures community they have not been described in the context of the FMM before, and it is a lack of such a clear exposition that has held back their wider use.

A. Indexing

To index the boxes in a more efficient way we can, for example, do the following. Each box in the tree can be identified by assigning it a unique index among the 2^d children of its parent box, and by knowing the index of its parent in the set of its grandparent's children, and so on. For reasons that will be clear in the next section we index the 2^d children of a particular parent box using the numbers $0, 1, ..., 2^d - 1$. Then the index of a box can be written as the string

$$String\,(n, l) = (N_1, N_2, ..., N_l)\,, \quad N_j = 0, ..., 2^d - 1, \quad j = 1, ..., l, \tag{48}$$

where l is the level at which the indexed box is located and N_j is the index of a box at level j containing that box. We drop N_0 from the indexing, since it is the only box at level 0, and has no parents. We can assign the index 0 to this box.. For example, in two dimensions for the quad-tree we have the numbering shown in Figure 15. The smaller black box will have indexing string (3,1,2) and the larger black box will have indexing string (2,3). From the construction it is clear that each box can be described by such a string, and each string uniquely determines the box.

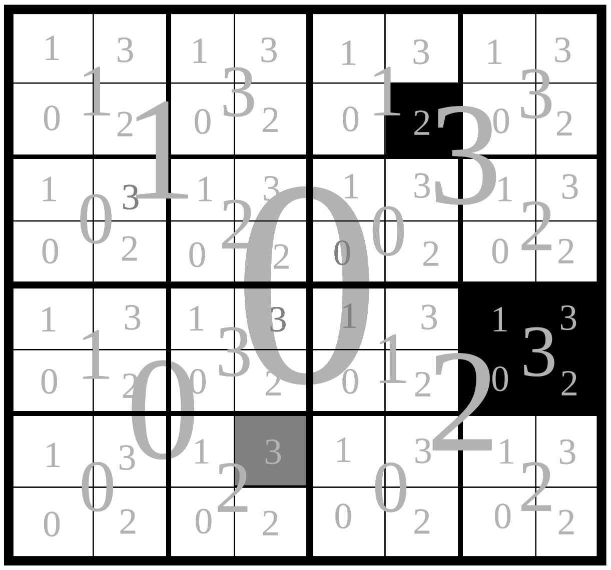


FIG. 15: Hierarchical numbering in quad-tree.

The indexing string can be converted to a single number as follows:

$$n = \left(2^d\right)^{l-1} \cdot N_1 + \left(2^d\right)^{l-2} \cdot N_2 + ... + 2^d \cdot N_{l-1} + N_l. \tag{49}$$

Note that this index depends on the level l at which the box is considered and unless this information is included, different boxes could be described by the same index. For example, boxes with strings (0,2,3) and (2,3) map to the same index, $n = 11$, but they are different. The box (0,2,3) is the small grey box and (2,3) is the larger black box in Figure 15. The unique index of any box can be represented by the pair:

$$UniversalIndex = (n, l). \tag{50}$$

We could instead say "box 11 at level 2" (this is the larger black box in Figure 15) or "box 11 at level 3" (this is the smaller gray box in Figure 15). We also have the box with index 0 at each level, which is located in the left bottom corner. "Box 0 at level 0" refers to the largest box in the 2^d-tree.

The string could be mapped in a different way, so that all boxes map to a unique index, instead of a pair. However, storing the level number, does not increase the memory or time complexity, since anyway the MLFMM loops go through the level hierarchy, and one always has level value.

If the indexing at each level is performed in a consistent way (for example in Figure 15 we always assign 0 to the child at the bottom left corner of the parent, 1 to the child in the left upper corner, 2 to the child in the right bottom corner, and 3 to the child in the right upper corner; for quad-trees this can be also called 'Z-order' following [4]) then we call such a indexing scheme "hierarchical." A consistent hierarchical scheme has the following desirable properties.

1. *Determining the Parent*: Consider a box at level l of the 2^d-tree, whose index is given by Eq. (49). The parent of this box is

$$Parent(n) = \left(2^d\right)^{l-2} \cdot N_1 + \left(2^d\right)^{l-3} \cdot N_2 + ... + N_{l-1}. \tag{51}$$

To obtain this index there is no need to know whether $N_1, N_2,$ and so on are zeros or not. We also do not need to know l, since this index is produced from string simply by dropping the last element:

$$Parent\left(N_1, N_2, ..., N_{l-1}, N_l\right) = \left(N_1, N_2, ..., N_{l-1}\right). \tag{52}$$

This means that function $Parent$ in such a numbering system is simple and level independent. For example at $d = 2$ for box index 11 the parent always will be $Parent(11) =$

2 *independent of the level being considered.* Obtaining the parent's index in the universal numbering system in Equation (50) is also simple, since the level of the parent is $l-1$. Therefore,

$$Parent\,(n,l) = (Parent(n), l-1)\,. \tag{53}$$

2. *Determining the Children*: For the function $ChildrenAll$ as well we do not need to know the level. Indeed, to get the indices of all 2^d children of a box represented by the string (48), we need simply add one more element to the string, which runs from 0 to 2^d-1, to list all the children:

$$ChildrenAll\,(N_1, N_2, ..., N_l) = \{(N_1, N_2, ..., N_l, N_{l+1})\}\,, \quad N_{l+1} = 0, ..., 2^d-1, \tag{54}$$

or

$$ChildrenAll(n) = \left\{ \left(2^d\right)^l \cdot N_1 + \left(2^d\right)^{l-1} \cdot N_2 + ... + \left(2^d\right) \cdot N_l + N_{l+1} \right\}, \quad N_{l+1} = 0, ..., 2^d-1. \tag{55}$$

For the universal numbering system (50), the operation of finding the children is simply the calculation of the children numbers and assigning their level to $l+1$:

$$ChildrenAll(n,l) = (ChildrenAll(n), l+1)\,. \tag{56}$$

Note that

$$Parent(n) = \left[n/2^d\right], \tag{57}$$

$$ChildrenAll(n) = \left\{2^d \cdot n + j\right\}, \quad j = 0, ..., 2^d-1, \tag{58}$$

where [] means integer part.

The use of 2^d-trees makes obtaining parent and children indices very convenient. Indeed the above operations are nothing but shift operations in the *bit representation* of n. Performing a right bit-shift operation on n by d-bits one can obtain the index of the parent. One can list all indices of the children boxes of n by a left bit-shift operation on n by d-bits and adding all possible combinations of d-bits.

B. Spatial Ordering

The above method of indexing provides a simple and natural way for representing a 2^d-tree graph structure and easy $O(1)$ algorithms to determine parent-children (and therefore sibling) relationships. However, we still do not have a way for determining neighbors. Further the MLFMM algorithm requires finding of the box center for a given index (n, l) and the box index to which a given spatial point $\mathbf{x}$ belongs. To do this a spatial ordering in d-dimensional space should be introduced. We provide below such an ordering and $O(1)$ algorithms for these operations.

1. Scaling

As assumed above, the part of the d-dimensional space we are interested in can be enclosed within a bounding box with dimensions $D_1 \times D_2 \times ... \times D_d$. In problems in physical space of dimensions ($d = 1, 2, 3$) we usually have isotropy of directions and can enclose that box in a cube of size $D \times ... \times D$, where

$$D = \max_d D_d, \tag{59}$$

with one corner assigned the minimum values of Cartesian coordinates:

$$\mathbf{x}_{\min} = (x_{1,\min}, ..., x_{d,\min}). \tag{60}$$

This cube then can be mapped to the unit cube $[0, 1] \times ... \times [0, 1]$ by a shift of the origin and scaling:

$$\overline{\mathbf{x}} = \frac{\mathbf{x} - \mathbf{x}_{\min}}{D}, \tag{61}$$

where the $\mathbf{x}$ are the true Cartesian coordinates of any point in the cube, and $\overline{\mathbf{x}}$ are normalized coordinates of the point. If a 2^d-tree data structure is applied to a case where each dimension has its own scale D_j (such problems are typical in parametric spaces) the mapping of the original box

$$[x_{1,\min}, x_{1,\max}] \times ... \times [x_{d,\min}, x_{d,\max}], \quad x_{j,\max} - x_{j,\min} = D_j, \quad j = 1, ..., d \tag{62}$$

to the unit cube $[0, 1] \times ... \times [0, 1]$ can be also easily performed by scaling along each dimension as:

$$\overline{x}_j = \frac{x_j - x_{j,\min}}{D_j}, \quad j = 1, ..., d, \quad \overline{\mathbf{x}} = (\overline{x}_1, ..., \overline{x}_d).$$

In the sequel we will work only with the unit cube, assuming that, if necessary, such a scaling has already been performed, and, that the point $\mathbf{x}$ in the original d-dimensional space can be found given $\overline{\mathbf{x}} \in [0,1] \times ... \times [0,1]$.

2. *Ordering in 1-Dimension (binary ordering)*

Let us consider first the case $d = 1$, where our 2^d-tree becomes a binary tree (see Figure 6). In the one-dimensional case all the points $\overline{x} \in [0,1]$ are naturally ordered and can be represented in the decimal system as

$$\overline{x} = (0.a_1 a_2 a_3 ...)_{10}\,, \quad a_j = 0, ..., 9; \quad j = 1, 2, ... \tag{63}$$

Note that the point $\overline{x} = 1$ can also be written as

$$\overline{x} = 1 = (0.999999....)_{10} \tag{64}$$

which we consider to be two equivalent representations. The latter representation reflects the fact that $\overline{x} = 1$ is a limiting point of sequence 0,0.9,0.99, ■.

We also can represent any point $\overline{x} \in [0,1]$ in the binary system as

$$\overline{x} = (0.b_1 b_2 b_3 ...)_2\,, \quad b_j = 0, 1; \quad j = 1, 2, ... \tag{65}$$

and we can write the point $\overline{x} = 1$ in the binary system as

$$\overline{x} = 1 = (0.111111....)_2\,. \tag{66}$$

Even though the introduced indexing system for the boxes in the case $d = 1$, results in a rather trivial result, since all the boxes are already ordered by their indices at a given level l from 0 to 2^{l-1}, and there is a straightforward correspondence between box indices and coordinates of points, we still consider the derivation of the neighbor, parent and children and other relationships in detail, so as to conveniently extend them to the general d-dimensional case.

a. Finding the index of the box containing a given point. Consider the relation between the coordinate of a point and the index of the box where the point is located. We note that the size of a box at each level is 1 placed at the position equal to the level number after the decimal in its binary record as shown in the table below.

Level	Box Size (dec)	Box Size (bin)
0	1	1
1	0.5	0.1
2	0.25	0.01
3	0.125	0.001
...	...	...

If we consider level 1 where there are two boxes:

$$(0.0b_1b_2b_3...)_2 \in Box\left((0)\right), \quad (0.1b_1b_2b_3...)_2 \in Box\left((1)\right), \quad \forall b_j = 0,1; \quad j = 1,2,..., \tag{67}$$

where $Box(0)$ and $Box(1)$ denote sets of spatial points that belong to boxes with indices (0) and (1), respectively (at this point we will use binary strings for indexing) . At level 2 we will have

$$\begin{aligned}&(0.00b_1b_2b_3...)_2 \in Box\left((0,0)\right), \quad (0.01b_1b_2b_3...)_2 \in Box\left((0,1)\right), \\ &(0.10b_1b_2b_3...)_2 \in Box\left((1,0)\right), \quad (0.11b_1b_2b_3...)_2 \in Box\left((1,1)\right), \\ &\qquad\qquad \forall b_j = 0,1; \quad j = 1,2,...,\end{aligned} \tag{68}$$

This process can be continued. At the lth level we obtain

$$(0.N_1N_2...N_lb_1b_2b_3...)_2 \in Box\left((N_1,N_2,...,N_l)\right), \quad \forall b_j = 0,1; \quad j = 1,2,... \tag{69}$$

Therefore to find the index of the box at level l to which the given point belongs we need simply shift the binary number representing this point by l positions and take the integer part of this number:

$$(0.N_1N_2...N_lb_1b_2b_3...)_2 \to (N_1N_2...N_l.b_1b_2b_3...)_2\,; \quad N_1N_2...N_l = \left[(N_1N_2...N_l.b_1b_2b_3...)_2\right]. \tag{70}$$

This procedure also can be written as

$$(n,l) = \left[2^l \cdot \overline{x}\right]. \tag{71}$$

b. Finding the center of a given box. The relation between the coordinate of the point and the box index can be also used to find the coordinate of the center for given box index. Indeed, if the box index is $N_1N_2...N_l$ then at level l we use l-bit shift to obtain

$$n = (N_1N_2...N_l)_2 \to (0.N_1N_2...N_l)_2\,,$$

then we add 1 as extra digit, so we have for the center of the box at level l :

$$\overline{x}_c(n,l) = (0.N_1N_2...N_l1)_2\,. \tag{72}$$

Indeed, any point with coordinates $(0.N_1N_2...N_l)_2 \leqslant \overline{x} \leqslant (0.N_1N_2...N_l111111....)_2$ belongs to this box. This procedure also can be written in the form:

$$\overline{x}_c(n,l) = 2^{-l}\cdot\left(n+2^{-1}\right), \tag{73}$$

since addition of one at position $l+1$ after the point in the binary system is the same as addition of 2^{-l-1}.

c. Finding neighbors. In a binary tree each box has two k-neighbors, except the boxes that are closer that separated from the boundaries by less than $k-1$ boxes. Since at level l all boxes are ordered, we can find indices for all the k-neighbors using the function

$$NeighborAll^{(k)}(n,l) = \{(n-k,l),(n+k,l)\}\,. \tag{74}$$

For the binary tree the k-neighbors have indices that are $\pm k$ of the given box index. If the neighbor index at level l computes to a value larger than 2^l-1 or smaller than 0 we drop this box from the neighbor list.

3. *Ordering in d-dimensions*

Coordinates of a point $\overline{\mathbf{x}} = (\overline{x}_1,...,\overline{x}_d)$ in the d-dimensional unit cube can be represented in the binary form

$$\overline{x}_m = (0.b_{m1}b_{m2}b_{m3}...)_2\,,\quad b_{mj}=0,1;\quad j=1,2,...,\quad m=1,...,d. \tag{75}$$

Instead of having d indices characterizing each point we can form a single binary index that represent the same point by an ordered mixing of the digits in the above binary representation (this is also called *bit interleaving*), so we can write:

$$\overline{\mathbf{x}} = (0.b_{11}b_{21}...b_{d1}b_{12}b_{22}...b_{d2}...b_{1j}b_{2j}...b_{dj}...)_2\,. \tag{76}$$

This can be rewritten in the system with base 2^d:

$$\overline{\mathbf{x}} = (0.N_1N_2N_3...N_j...)_{2^d}\,,\quad N_j=(b_{1j}b_{2j}...b_{dj})_2\,,\quad j=1,2,...,\quad N_j=0,...,2^d-1. \tag{77}$$

An example of converting 3 dimensional coordinates to octal and binary indices is shown in Figure 16.

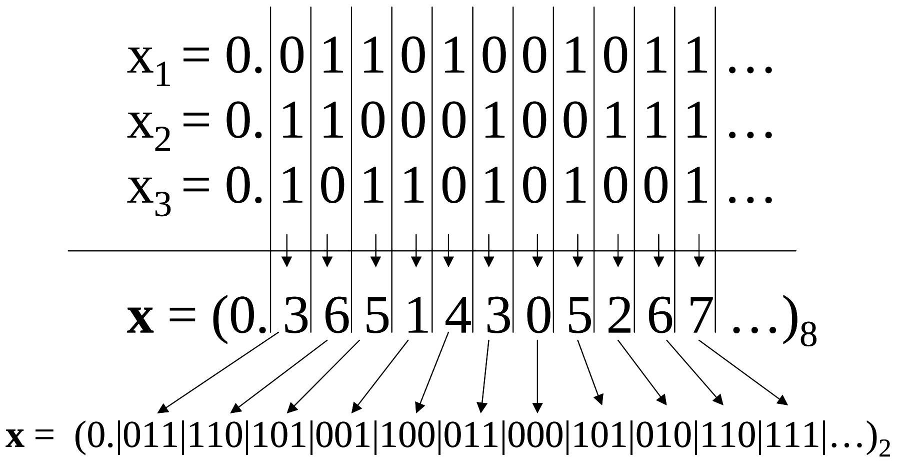

x1 = 0. 0 1 1 0 1 0 0 1 0 1 1 ...
x2 = 0. 1 1 0 0 0 1 0 0 1 1 1 ...
x3 = 0. 1 0 1 1 0 1 0 1 0 0 1 ...
x = (0. 3 6 5 1 4 3 0 5 2 6 7 ...)8
x = (0.|011|110|101|001|100|011|000|101|010|110|111|...)2

FIG. 16: Example of converting of coordinates in 3 dimensions to a single octimal or binary number.

a. Finding the index of the box containing a given point. Consider a relation between the coordinate of point, which now is a single number $x \in [0,1]$ and the index of the box in the 2^d-tree where this point is located. We use the convention of ordered indexing of boxes in the hierarchical structure. 2^d children of any box will be indexed according coordinate order. Since the children boxes are obtained by division of each side of the parent box in 2, we assign 0 to the box with the smaller center coordinate and 1 to the other box. In d dimensions, 2^d combinations are produced by d binary coordinates. So any set of d coordinates can be interpreted as a binary string, which then can be converted to a single index in the binary or some counting system, e.g., with the base 2^d:

$$(b_1, b_2, ..., b_d) \to (b_1 b_2 ... b_d)_2 = N_{2^d}. \tag{78}$$

Examples of such an ordering for $d = 2$ and $d = 3$ are shown in Figure 17.

Obviously such an ordering is consistent for all levels of the hierarchical structure, since it can be performed for children of each box. Therefore the functions $Parent$ and $ChildrenAll$ introduced above can be used, and they are not level dependent.

Now we can show that the box index that contains a given spatial point can be found using the same method as for the binary tree with slight modification. The size of the boxes at each level is nothing but 1 placed at the position equal to the level number after the point in its binary record as shown in the table for binary tree. At level 1, where we have 2^d boxes, the binary record determines

$$\overline{\mathbf{x}} = (0.b_{11}b_{21}...b_{d1}b_{12}b_{22}...b_{d2}...b_{1j}b_{2j}...b_{dj}...)_2 \in Box\left((b_{11}b_{21}...b_{d1})_2\right) = Box\left((N_1)_{2^d}\right), \tag{79}$$

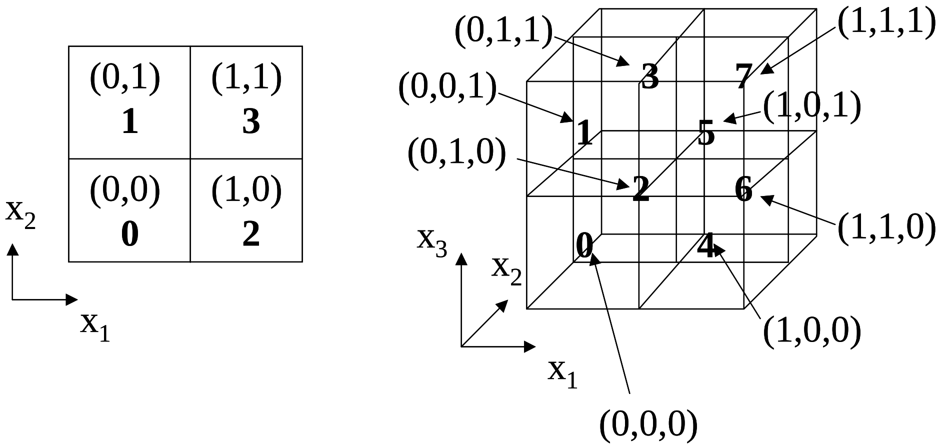


FIG. 17: Ordering of children boxes in quad-tree and in oct-tree.

Indeed, for each coordinate the first digit only determines the box at level 1, which is exactly equal to the mixed coordinate by accepted convention of ordered indexing of children. At level 2 the same happens with the second digit of each coordinate. At level l we have using 2^d-based system and string representation of the box index:

$$(0.N_1N_2...N_lc_1c_2c_3...)_{2^d} \in Box\left((N_1,N_2,...,N_l)_{2^d}\right), \quad \forall c_j = 0,...,2^d-1; \quad j=1,2,... \tag{80}$$

Therefore to find the index of the box at level l, to which the given point belongs we need simply shift the 2^d-index representing this point by l positions and take the integer part of this index:

$$(0.N_1N_2...N_lc_1c_2c_3...)_{2^d} \rightarrow (N_1N_2...N_l.c_1c_2c_3...)_{2^d}; \quad N_1N_2...N_l = [(N_1N_2...N_l.b_1b_2b_3...)_{2^d}]. \tag{81}$$

This procedure also can be performed in the binary system by a $d \cdot l$ left bit shift:

$$(0.b_{11}b_{21}...b_{d1}b_{12}b_{22}...b_{d2}...b_{1l}b_{2l}...b_{dl}b...)_2 \rightarrow (b_{11}b_{21}...b_{d1}b_{12}b_{22}...b_{d2}...b_{1l}b_{2l}...b_{dl}.b...)_2; \tag{82}$$

$$index = (b_{11}b_{21}...b_{d1}b_{12}b_{22}...b_{d2}...b_{1l}b_{2l}...b_{dl})_2.$$

In another counting system this can be obtained by multiplication of the coordinate of the point by 2^{dl} and taking the integer part. So

$$(n,l) = \left[2^{dl} \cdot \overline{\mathbf{x}}\right]. \tag{83}$$

This example shows that in contrast to the 1-dimensional case, in the d-dimensional case, the advantage in conversion of the index to binary form is substantial, since this enables performing of bit interleaving to produce a single multiindex consistent with the 2^d-tree data structure. Such procedure is natural in computations since anyway all indices are represented finally in binary format. So the algorithm of finding the box index for a given spatial point is $O(1)$ algorithm for any index of dimensions. We also note that the algorithm does not require conversion to octal or other 2^d -based system, since, as a binary representation of coordinates and bitshift procedures are available.

b. Finding the center of a given box. The relation between the coordinate of a point and its box index enables easy finding of the coordinates of the center for a given box index. To do this we first convert the box index at level l into binary form

$$n = (b_{11}b_{21}...b_{d1}b_{12}b_{22}...b_{d2}...b_{1l}b_{2l}...b_{dl})_2 \,. \tag{84}$$

Then we decompose this index to d coordinate indices (this is also called *bit deinterleaving*):

$$\begin{aligned} n_1 &= (b_{11}b_{12}...b_{1l})_2 \,. \\ n_2 &= (b_{21}b_{22}...b_{2l})_2 \,. \\ &... \\ n_d &= (b_{d1}b_{d2}...b_{dl})_2 \,. \end{aligned} \tag{85}$$

This is a simple operation since the bit string (84) should be rewritten in the form of a matrix $d \times l$ column by column. Note that because some b's can be zero we need to check the length of the bitstring n and complete it by adding zeros before the first non-zero b to achieve a length $dl,$ or we can simply fill the matrix starting with the last element b_{dl}, then putting $b_{d-1,l}$ in the same column above b_{dl} and so on.

Further conversion from the indices to the coordinate values is similar to the procedures in binary tree (72) and (73). The coordinates of the box center in binary form are

$$\overline{x}_m (n, l) = (0.b_{k1}b_{k2}...b_{kl}1)_2 \,, \quad m = 1, ..., d. \tag{86}$$

or in a form that does not depend on the counting system:

$$\overline{x}_m (n, l) = 2^{-l} \cdot \left(n_l + \frac{1}{2} \right), \quad k = 1, ..., d. \tag{87}$$

c. Finding the neighbors. The procedure for finding the k-neighbors of a given box at level l in the 2^d-tree with ordered hierarchical indexing can be reduced to the procedure of finding neighbors with respect to each dimension. Such procedure is described above for binary tree and in general case we need just slightly modify it. First we perform bit deinterleaving according Eqs. (84) and (85). Then for each coordinate index we generate the k-indices:

$$n_m^+ = n_m + k, \quad n_m^- = n_m - k, \quad m = 1, ..., d, \tag{88}$$

and check if any of these indices is out of range $[0, 2^l - 1]$. If so such index should be dropped from the list of indices generating k-neighbor list. So for each dimension m of d-dimensional space we will have a set, $Neighbor_m$, consisting of 3 or 2 (if one of the indices $index_m^{\pm}$ is dropped) indices:

$$Neighbor_m = \begin{cases} \{n_m^-, n_m, n_m^+\}, & n_m \neq 0, 2^l - 1 \\ \{n_m, n_m^+\}, & n_m = 0. \\ \{n_m^-, n_m\}, & n_m = 2^l - 1. \end{cases} \qquad m = 1, ..., d. \tag{89}$$

The set of k-neighbor generating indices is then

$$\nu = (\nu_1, ..., \nu_d), \quad \nu_m \in s_m, \quad m = 1, ..., d. \tag{90}$$

where each ν_m can be any element of s_m (89), except the case when all $\nu_m = n_m$ simultaneously for all $k = 1, ..., d$, since this case corresponds to the box itself. For a box situated far from the boundary of the domain we have therefore $3^d - 1$ possible combinations of $(\nu_1, ..., \nu_d)$, and each of them corresponds to a neighbor.

Note that n_m is obtained from the bit deinterleaving procedure in binary form. Thus the operations of finding $n_m^{\pm}$ are also convenient to perform in binary form to obtain binary format for each n_m, $m = 1, ..., d$. This yields:

$$\nu_1 = (\nu_{11}\nu_{12}...\nu_{1l})_2, \quad \nu_2 = (\nu_{21}\nu_{22}...\nu_{2l})_2, \quad ..., \quad \nu_d = (\nu_{d1}\nu_{d2}...\nu_{dl})_2, \tag{91}$$

where $\nu_{mj} = 0, 1$ are the bits of ν_m. The interleaved bit strings produce the neighbor indices:

$$NeighborAll^{(k)}(n, l) = \{(\nu_{11}\nu_{21}...\nu_{d1}\nu_{12}\nu_{22}...\nu_{d2}...\nu_{1l}\nu_{2l}...\nu_{dl})_2\}, \quad \nu_m \in s_m, \quad m = 1, ..., d. \tag{92}$$

Note that the lengths of bitstrings $(\nu_{m1}\nu_{m2}...\nu_{ml})_2$ for different m can be different because the several first bits can be zero, $\nu_{m1} = 0, \nu_{m2} = 0, ...$ In this case either each string should be completed with zeros to length l, or the formation of the neighbor index can start from the last digit ν_{dl} assuming that 0 corresponds to the absent bits.

C. Structuring data sets

The 2^d-tree hierarchical space subdivision enable organization of infinite (continuum) data sets. However, in practice even large data sets are finite. Scaling and mapping finite d-dimensional data sets into a unit d-dimensional cube yields a set $\mathbb{W}$ of N *different* points inside the cube:

$$\mathbb{W} = \{\overline{\mathbf{x}}_1, \overline{\mathbf{x}}_2, ..., \overline{\mathbf{x}}_N\}, \quad \overline{\mathbf{x}}_i \in (0,1)\times...\times(0,1) \subset \mathbb{R}^d, \quad \overline{\mathbf{x}}_i \neq \overline{\mathbf{x}}_j, \quad i \neq j, \quad i,j = 1,...,N, \tag{93}$$

where the scaling can be made in such way that no data points are located on the cube boundaries.

Because the number of points we consider is finite, there always exists a level of space subdivision L_1, at which some boxes do not contain points from $\mathbb{W}$. Indeed, for $2^{Ld} > N$, or

$$L_1 > \frac{1}{d}\log_2 N \tag{94}$$

the number of boxes is larger then the number of points. There also exists a finite L_2, such that at levels $L_2, L_2+1, L_2+2, ...$ all boxes will contain not more than one point. This is easy to prove, since if we consider the minimum distance between the points from $\mathbb{W}$:

$$\overline{D}_{\min} = \min_{i\neq j} |\overline{\mathbf{x}}_i - \overline{\mathbf{x}}_j|, \quad i,j = 1,...,N, \tag{95}$$

where $|\overline{\mathbf{x}}_i - \overline{\mathbf{x}}_j|$ is the Euclidean distance, then L_2 can be determined from the requirement that the main diagonal of the box at this level $d^{1/2}2^{-L_2}$ be smaller than $\overline{D}_{\min}$, or

$$L_2 > \log_2 \frac{d^{1/2}}{\overline{D}_{\min}}. \tag{96}$$

At some $l_{\max}$ (this can be efficiently found using an algorithm provided below) we will have a situation, where each box at such a level will contain not more than s data points $(1 \leqslant s \leqslant N)$, while at level $l = l_{\max} - 1$ there exists at least one box containing more than s data points (assuming that the total number of data points $N \geqslant 2$). We call s the *grouping* or *clustering parameter*, and $l_{\max}$ as the *threshold level* and will provide $O(N)$ algorithm for its determination. Note that for the MLFMM procedure with k-neighborhoods another clustering parameter, q, might be more appropriate for determination of $l_{\max}$. This parameter is the number of source points in the k-neighborhood of the evaluation point. So at level $l = l_{\max} - 1$ there exist at least one box containing an evaluation point, whose k-neighborhood contains more than q sources, while at level $l = l_{\max}$ there are no such boxes. Determination of q requires both data sets and more complicated procedure than determination of s, while it can be performed for $O(N \log N)$ operations.

1. *Ordering of d-dimensional data*

We can introduce a "correspondence vector", which is a vector of length N where the ith component is equal to the index of the box at level $l \geqslant l_{\max}$ of the 2^d-tree. We denote this vector as $\mathbf{v}$:

$$\mathbf{v} = (v_1, v_2, ..., v_N), \quad v_i = Index\left(\overline{\mathbf{x}}_i, l\right), \quad i = 1, ..., N, \quad l \geqslant l_{\max}, \tag{97}$$

where $Index$ can be determined using the bit-interleaving technique. The array $\mathbf{v}$ can then be sorted in non-descending order as

$$(v_1, v_2, ..., v_N) \to (v_{i_1}, v_{i_2}, ..., v_{i_N}), \quad v_{i_1} \leqslant v_{i_2} \leqslant ... \leqslant v_{i_N}. \tag{98}$$

Such a sorting requires $O(N \log N)$ operations using standard sorting algorithms and provides the permutation index (or "permutation vector" or "pointer vector") of length N :

$$\mathbf{ind} = \left(i_1, i_2, ..., i_N\right), \tag{99}$$

that can be stored in the memory. To save the memory the array $\mathbf{v}$ should not be rewritten and stored again, since $\mathbf{Ind}$ is a pointer and

$$\mathbf{v}(i) = v_i, \quad \mathbf{ind}\left(j\right) = i_j, \quad \mathbf{v}(\mathbf{ind}\left(j\right)) = \mathbf{v}(i_j) = v_{i_j}, \quad i, j = 1, ..., N, \tag{100}$$

so that

$$\mathbf{v}(\mathbf{ind}) = (v_{i_1}, v_{i_2}, ..., v_{i_N}). \tag{101}$$

At level $l \geqslant l_{\max}$ there may exist $i \neq j$ such that $v_i = v_j$ and the order of these elements in the sorted list can be arbitrary. We will fix this order once for all time, in other words we assume that a permutation index exists and does not change even though two subsequent elements in the list can be identical.

To machine precision each coordinate of the data point is represented with $Bit_{\max}$ bits. This means that there is no sense in using more than $Bit_{\max}$ levels of space subdivision – if two points have identical d coordinates in terms of $Bit_{\max}$-truncation that they can be considered as identical. We assume that $l_{\max} \leqslant Bit_{\max}$. Note that operation $Parent$ in the present hierarchical indexing system preserves the non-descending order, so once data points are sorted at the maximum resolution level $Bit_{\max}$ and permutation index is fixed this operation should not be repeated and can be performed once *before* the level $l_{\max}$ for given set is determined.

2. Determination of the threshold level

To determine the threshold level $l_{\max}$, for example, the following $O(N)$ algorithm can be used:

```
$i = 0$, $m = s$, while $m < N$
    $i = i + 1$, $m = m + 1$;
    $a = Interleaved(v(\mathbf{ind}(i)))$;
    $b = Interleaved(v(\mathbf{ind}(m)))$;
    $j = Bit_{\max} + 1$
    while $a \neq b$
        $j = j - 1$;
        $a = Parent(a)$;
        $b = Parent(b)$;
        $l_{\max} = \max(l_{\max}, j)$;
    end;
end;
```

The idea of this algorithm is rather simple and it exploits the fact that the array $\left\{Index(\overline{\mathbf{x}}_{\mathbf{ind}(i)}, Bit_{\max}), \quad i = 1, ..., N\right\}$ is sorted (ordered). At level $l_{\max}$ only s subsequent data points may have the same bit strings. The level independent operation $Parent$ can be performed several times to find the level at which two points differ.

3. Search procedures and operations on point sets

We also assume that some standard functions for working with the sets, such as the difference of two sets, $\mathbb{C} = \mathbb{A}\backslash\mathbb{B}$, intersection, $\mathbb{C} = \mathbb{A} \cap \mathbb{B}$, and union, $\mathbb{C} = \mathbb{A} \cup \mathbb{B}$ are available as library programs. Note that for ordered sets such procedures are much faster then for arbitrary sets since they do not require a step for sorting each set preceding an operation on that set. As a result of the initial data sorting we also have fast standard search procedures in sorted lists with complexity $O(\log N)$.

We also mention that the complexity of the set intersection procedure of a small set of power p_w and large set of power P_w is $O(p_w \log P_w)$, since one can look for each element of the smaller set in the larger set, and such search has $O(\log P_w)$ complexity. This yields in $O(\log N)$ complexity for $Neighbors^{(k)}(W;n,l)$ and $Children(W;n,l)$ procedures, for given W-data hierarchy ($W = X, Y$). Indeed, we have

$$\begin{aligned} Neighbors^{(k)}(W;n,l) &= NeighborsAll^{(k)}(n,l) \cap W, \quad W = X, Y, \\ Children(W;n,l) &= ChildrenAll(n,l) \cap W, \quad W = X, Y, \end{aligned} \tag{102}$$

and because the procedures $NeighborsAll^{(k)}(n,l)$ and $ChildrenAll(n,l)$ have $O(1)$ complexity and return $O(1)$ elements the complexity of the above operations is $O(\log N)$ for $W = X$, and $O(\log M)$ for $W = Y$. As was mentioned earlier, if the memory is sufficient for the problem to be solved, the search procedure can be $O(1)$,which makes the functions $Neighbors^{(k)}(W;n,l)$ and $Children(W;n,l)$, $O(1)$ procedures.

IV. COMPLEXITY OF THE MLFMM AND OPTIMIZATION OF THE ALGORITHM

Strictly speaking the complexity of the MLFMM is at least $O(N \log N)$ if the initial step for ordering of spatial data is required. However, in practice the step that must be repeated is not the step setting the data structure, but the MLFMM core procedure. Further this step is much more costly and despite potentially it can be performed for $O(N+M)$ operations, the constant in this asymptotic estimate is normally much larger than $\log N$ for practical values of N. To obtain an understanding of the algorithm complexity, and as a basis for the development of optimization schemes, we consider the complexity of each step of the MLFMM.

A. Complexity of the MLFMM procedure

1. Regular mesh

The actual number of operations in the FMM procedure depends on the particular distributions of the sources and evaluation points. To evaluate the number of operations we consider ‘the worst’ case when at the finest level of subdivision $l_{\max}$ each box contains s source points. In this case there will be no empty boxes and we will have no savings by skipping such boxes. In this worst case the number of boxes at the $l_{\max}$th level will be $2^{l_{\max}d}$ and since each box contains s sources, the total number of sources will be

$$N = 2^{l_{\max}d}s. \tag{103}$$

For simplicity we assume that each box at the finest level contains t evaluation points, so

$$M = 2^{l_{\max}d}t. \tag{104}$$

Let the length of the vector of expansion coefficients be p and the cost of obtaining the expansion coefficients $\mathbf{B}$ (see Eq. (32)) for each source be

$$CostB\left(p\right) = O(p). \tag{105}$$

Therefore the cost of the first step of the upward pass (32) will be

$$CostUpward_1 = NCostB\left(p\right) = O\left(Np\right). \tag{106}$$

This provides all the coefficients $\mathbf{C}^{(n,l_{\max})}$.

Denote by $CostSS(p)$ the computational complexity of a single $S|S$-translation of the vector of expansion coefficients. Since each box contains 2^d children and there are 2^{ld} boxes at level l, the number of operations to obtain all coefficients $\mathbf{C}^{(n,l)}$ from level $l = l_{\max} - 1$ to level $l = l_{\min}$, or, in other words, the cost of the step 2 of the upward pass (33) will be

$$\begin{aligned} CostUpward_2 &= 2^d\left(2^{(l_{\max}-1)d} + 2^{(l_{\max}-2)d} + ... + 2^{l_{\min}d}\right) CostSS(p) \\ &= \frac{2^d}{2^d-1}\left(2^{l_{\max}d} - 2^{l_{\min}d}\right) CostSS(p) < \frac{2^{(l_{\max}+1)d}}{2^d-1} CostSS(p). \end{aligned} \tag{107}$$

In the downward pass we denote the cost of a single $S|R$-translation of p expansion coefficients as $CostSR(p)$ and the cost of $R|R$-translation as $CostRR(p)$. For purposes of estimation we will take the number of boxes in the $E_4^{(k)}\left(n,l\right)$ neighborhood of a given box (n,l) as if it is not a box near the boundary. We denote $P_4^{(k,m)}\left(d\right)$ the maximum number of boxes from which centers $S|R$-translation is performed to the center of a given box for dimensionality d. Note that the use of reduced schemes for $S|R$ translations reduces $P_4^{(k)}$. According to Eq. (39) we have

$$P_4^{(k,m)}\left(d\right) \leqslant \left(2^d-1\right)\left(2k+1\right)^d, \tag{108}$$

where the upper limit is reached when a regular translation scheme is used and $P_4^{(k)}$ can be a^d $(a > 1)$ times smaller than this value if a reduced scheme of translation is employed. In this case the cost of the first step in the downward pass (see Eq. (32)) will be

$$\begin{aligned} CostDownward_1 &< P_4^{(k,m)}\left(d\right)\left(2^{l_{\min}d} + 2^{(l_{\min}+1)d} + ... + 2^{l_{\max}d}\right) CostSR(p) \\ &= P_4^{(k,m)}\left(d\right) \quad \frac{2^d}{2^d-1}\left(2^{l_{\max}d} - 2^{(l_{\min}-1)d}\right) CostSR(p) < \frac{2^{(l_{\max}+1)d}}{2^d-1} P_4^{(k,m)}\left(d\right) CostSR(p). \end{aligned} \tag{109}$$

Since each box has only one parent the cost of the second step in the downward pass (36) that is performed for levels $l_{\min}+1,...,l_{\max}$ is

$$\begin{aligned} CostDownward_2 &= \left(2^{(l_{\min}+1)d}+2^{(l_{\min}+1)d}+...+2^{l_{\max}d}\right) CostRR(p) \\ &= \frac{2^d}{2^d-1}\left(2^{l_{\max}d}-2^{l_{\min}d}\right) CostRR(p) < \frac{2^{(l_{\max}+1)d}}{2^d-1} CostRR(p). \end{aligned} \tag{110}$$

To evaluate the sum at M points that in the worst case occupy different boxes we need to sum up to $(2k+1)^d s$ sources in the k-neighborhood of each point (the first term in the right hand side of Eq. (38), see also Eq. (20)) and compute a scalar product of two vectors of length P (the second term in the right hand side of Eq. (38)). This yields

$$CostEvaluation = M\left[s(2k+1)^d CostFunc + p\right], \tag{111}$$

where $CostFunc$ is the cost of evaluation of function $\phi_i(\mathbf{x})$ at one point $\mathbf{x}$. Thus, the total complexity of the MLFMM procedure on the preset data is

$$\begin{aligned} CostMLFMM &< NCostB(p) + M\left[s(2k+1)^d CostFunc + p\right] \\ &+ \frac{2^{(l_{\max}+1)d}}{2^d-1}\left[CostSS(p) + P_4^{(k,m)}(d)\ CostSR(p) + CostRR(p)\right]. \end{aligned} \tag{112}$$

We note now that factor $2^{l_{\max}d}$ can be expressed as N/s or M/t (see Eqs. (103) and (104)). We also note that the upward pass is performed with respect to the source hierarchy, so the number of operations there should be proportional to the number of sources, N. At the same time the downward pass is performed with respect to boxes of the target hierarchy, and the number of operation in this pass should be proportional to M. Using this reasoning we can rewrite Eq. (112) in the form:

$$\begin{aligned} CostMLFMM &< NCostB(p) + M\left[s(2k+1)^d CostFunc + p\right] \\ &+ \frac{2^d}{2^d-1}\left[\frac{N}{s} CostSS(p) + \frac{M}{t} P_4^{(k,m)}(d)\ CostSR(p) + \frac{M}{t} CostRR(p)\right]. \end{aligned} \tag{113}$$

Assuming now that the parameters p, k, s, t and d are all $O(1)$, we find that the cost of the MLFMM procedure is

$$CostMLFMM = O(N+M). \tag{114}$$

Usually we assume that $N \sim M$ and in this case we have $CostMLFMM = O(N)$.

2. Non-uniform data and use of data hierarchies

In the case of a regular mesh the MLFMM algorithm goes through all the boxes and there is no need to search for neighbors or children in the data hierarchies. Thus for this case we do not even need to perform the initial step of setting up the data structures necessary and can use the functions $NeighborsAll^{(k)}$ and $ChildrenAll$ to get the k-neighbors and the children. As pointed out in the previous section, these procedures have constant, or $O(1)$ complexity. These provide an $O(N+M)$ algorithm for the case of a regular mesh. Such a complexity also applies for the case of arbitrary data sets, since one simply can assign zero expansion coefficients $\mathbf{C}^{(n,l)}$ to boxes that do not contain source points and not build $\widetilde{\mathbf{D}}^{(n,l)}$ and $\mathbf{D}^{(n,l)}$ coefficients for boxes without evaluation points.

The first simple step to algorithm adaptivity is skipping boxes that do not contain sources (in the upward pass and for $S|R$-translations from such boxes) or do not contain evaluation points (in the downward pass), in other words the use of X and Y data hierarchies. The use of these hierarchies increases the complexity of operations $Neighbors^{(k)}$ and $Children$ to $O(\log N)$ or $O(\log M)$. It is reasonable to use these methods if the cost of search is lower than the cost of translation operations, which can be avoided. Therefore, minimal requirements for use of the data hierarchies for reduction of the computational complexity is then

$$\log N \lesssim CostSS(p), \quad \log N \lesssim CostSR(p), \quad \log M \lesssim CostRR(p). \tag{115}$$

Usually these conditions are satisfied (say for $CostSS(p) \sim 100$ we have $N \lesssim 2^{100}$), and even stronger inequalities hold:

$$\log N \ll CostSS(p), \quad \log N \ll CostSR(p), \quad \log M \ll CostRR(p). \tag{116}$$

so the algorithms utilizing search in data hierarchies are in practice more efficient than the algorithm that go through all the boxes. However in this case formally we obtain from Eq. (113):

$$\begin{aligned} CostMLFMM &\lesssim NCostB\,(p) + M\left[s(2k+1)^d CostFunc + p\right] \\ &+ \frac{2^d}{2^d-1}\left[\frac{N}{s}\log N + \frac{M}{t}\,P_4^{(k,m)}\,(d)\log N + \frac{M}{t}\log M + \left(\frac{N}{s} - N_{ss}\right) CostSS(p)\right] \\ &+ \frac{2^d}{2^d-1}\left\{\left(\frac{M}{t} - M_d\right)\left[\left(\,P_4^{(k,m)}\,(d) - N_{sr}^{(k)}\right)\,CostSR(p) + CostRR(p)\right]\right\}, \end{aligned} \tag{117}$$

where N_{ss} and $N_{sr}^{(k)}$ are the numbers of boxes that do not belong to the X-hierarchy, and so skipped when the $S|S$ and $S|R$ translations are performed, and M_d is the number of boxes that do not belong to the Y-hierarchy, and so skipped in the downward pass. The asymptotic complexity of the $MLFMM$ is then

$$CostMLFMM = O(N \log N + M \log N + M \log M), \tag{118}$$

since we fix p and let N and M be asymptotically large.

We should understand that this estimate for the computational complexity is valid only in the intermediate region defined by the conditions (115). If the conditions (116) hold, the overhead from the search in the data hierarchy is small, and the savings that arise from skipping empty boxes can be substantial. One can also think about an algorithm that switches off the search in data hierarchies and employs the non-adaptive scheme if the conditions (115) are violated. The algorithm with such a switch then has the asymptotic complexity of Eq. (114).

B. Optimization of the MLFMM

a. Grouping parameter optimization. Consider the complexity of the MLFMM for the regular mesh given by Eq. (113). Assume that the costs of all the translation operations are about the same, with $CostTrans(p)$ the cost of a single translation. This cost can be taken to be equal to $CostSR(p)$, since there are more of these translations, and they affect the cost function the most ($P_4^{(k,m)}(d) \gg 1$). So,

$$CostSS(p) \sim CostRR(p) \sim CostSR(p) = CostTrans(p). \tag{119}$$

Assuming that $P_4^{(k,m)}(d) \gg 1$, and taking into account that $N/s = M/t$, we simplify Eq. (113) as

$$CostMLFMM \lesssim NCostB(p) + Mp + sM(2k+1)^d CostFunc + \frac{N}{s} \frac{2^d \, P_4^{(k,m)}(d)}{2^d - 1} CostTrans(p). \tag{120}$$

For fixed $N, M, p, k,$ and $d,$ and a given translation scheme that determines $P_4^{(k,m)}(d)$, the computational complexity of the MLFMM will be a function of the grouping parameter s alone. The dependence $CostMLFMM(s)$ is rather simple, since this is a superposition of a linear function and a hyperbola, which has a unique positive minimum, s_{opt}, that can be found from by setting d$CostMLFMM/$d$s = 0$. This leads to

$$s_{opt} = \left[\frac{2^d \, P_4^{(k,m)}(d) \, N}{(2^d - 1)(2k+1)^d M} \frac{CostTrans(p)}{CostFunc} \right]^{1/2}. \tag{121}$$

Substituting Eq. (121) into Eq. (120) we obtain cost of the optimized MLFMM as

$$CostMLFMM_{opt} \lesssim NCostB(p) + Mp + 2 \left[\frac{2^d \, P_4^{(k,m)}(d)\,(2k+1)^d}{2^d - 1} MN \, CostTrans(p) \, CostFunc \right]^{1/2}. \tag{122}$$

Note that at optimum s, the cost of direct summation of sources from the neighborhoods of the evaluation points at the finest level is equal to the cost of accounting for contribution of the points outside these neighborhoods. In other words the cost of the final summation (38) is equal to the sum of the upward and downward pass costs.

For fixed k and d, Eq. (122) yields the following asymptotic complexity for the algorithm:

$$CostMLFMM_{opt} = O\left(Mp + Np + (MN \, CostTrans(p))^{1/2} \right). \tag{123}$$

If the term in square brackets of Eq. (122) is very small, we have the following evaluation for the theoretical minimum of the MLFMM complexity:

$$\min CostFMM = NCostB(p) + Mp = O(Mp + Np). \tag{124}$$

It is important to notice that the complexity of the optimized MLFMM (122) depends on the translation cost to the power $1/2$. This means that for any translation cost, such that

$$CostTrans(p) = O\left(p^2\right), \tag{125}$$

(which is the typical complexity for matrix based translation operators), the asymptotic complexity of the optimized MLFMM algorithm is

$$CostMLFMM_{opt} = O\left(Mp + Np\right). \tag{126}$$

Thus, to reduce the influence of p on the complexity, it is important to perform this optimization. The non-optimized algorithm has a complexity of

$$CostMLFMM = O\left(Mp + Np + N \; CostTrans(p)\right), \tag{127}$$

which yields, e.g. an $O(Np^2)$ algorithm when $CostTrans(p) = O(p^2)$ and $M \sim N$. For many problems, e.g., the solution of scattering problems, p can be of order 10^2; and this can be substantial.

Assuming equality holds in Eq. (108) and simplifying the expression (121) for the optimum grouping parameter, we obtain it for the regular $S|R$-translation scheme as

$$s_{opt} = \left[2^d \frac{N}{M} \frac{CostTrans(p)}{CostFunc}\right]^{1/2}. \tag{128}$$

This formula shows an interesting fact: s_{opt} for such a scheme does *not* depend on k and is proportional to $2^{d/2}$. The cost of the optimized FMM can be found from (122) and (124):

$$CostMLFMM_{opt} \lesssim \min CostFMM + 2(2k+1)^d \left[2^d MNCostTrans(p)CostFunc\right]^{1/2}. \tag{129}$$

It is also interesting to consider the optimum value of the grouping parameter s and the asymptotic complexity of the MLFMM procedure if the search procedures needed have a complexity of $O(\log N)$. The optimum value can be easily determined from Eqs. (121) and (122) if we add an $\beta \log N$ term to the cost of translation, so

$$s_{opt} = \left[\frac{2^d\, P_4^{(k,m)}(d)\, N}{(2^d-1)\,(2k+1)^d M} \frac{CostTrans(p) + \beta \log N}{CostFunc}\right]^{1/2}, \tag{130}$$

$$\begin{aligned} CostMLFMM_{opt} &\lesssim NCostB(p) + Mp + \\ &2\left\{\frac{2^d\, P_4^{(k,m)}(d)\,(2k+1)^d}{2^d-1} MN\left[CostTrans(p) + a\log N\right] CostFunc\right\}^{1/2}. \end{aligned} \tag{131}$$

Here β is some parameter that depends on the distribution of the data and the spatial dimension of the problem, and represents an average additional cost for each translation due to the neighbor search procedure. If Eq. (116) holds, of course such addition can be omitted. For larger N, if $CostTrans(p) \sim \beta \log N$ and we have some intermediate asymptotics of the type (118) for a non-optimized MLFMM. The optimized scheme shows for $M \sim N$

$$CostMLFMM_{opt} = O\left(N \log^{1/2} N\right). \tag{132}$$

Of course if the size of the problem becomes larger, then the conditions (115) may be violated. In this case one should determine the trade-off between computational and memory complexity, switching to $O(N)$ scheme with larger memory requirements or stay with the $O\left(N \log^{1/2} N\right)$ procedure and save memory.

b. Multiparametric optimization Even for fixed N, M,and, d optimization of the MLFMM can be a non-trivial task, due to interdependence of the parameters, p, s, k, r_c, and R_c, and availability of different reduced translation schemes, with some internal parameters, such as m in Eq. (47). In this paper we formulate such a problem in more or less general terms. Relations between p, s, R_c and r_c appear in the MLFMM from factorization of functions ϕ_i where p is the truncation number that provides required error bounds (say absolute error less than ϵ). Approximations with truncated series are performed inside or outside the convergence spheres specified by numbers $R_c\delta$ and $r_c\delta$, where δ is the distance scale that has minimum at the scale of the finest subdivision level, i.e. $\delta = 2^{-l_{\max}}D = (N/s)^{-1/d}\,D$. This yields constraints of type

$$F_1\left(p, r_c, s\right) < \epsilon, \quad F_2\left(p, R_c, s\right) < \epsilon, \tag{133}$$

where F_1 and F_2 depend on functions ϕ_i, expansion bases, dimensionality of the space, etc. Together with constraint (47) and

$$1 > r_c \geqslant \frac{d^{1/2}}{2k+1}, \tag{134}$$

that follows from Eq. (44) the problem then is to minimize function $CostMLFMM\left(p, s, k\right)$ within the specified constraints. We do not attempt do this here, leaving it a subject for future work.

c. 1-D example of optimization within specified error bounds Below, as an example, we consider the following problem for the multiparametric optimization of the FMM for a one dimensional function (see Eq. (7)):

$$\phi_i(y) = \frac{1}{y - x_i}, \quad i = 1, ..., N, \quad y \in [0,1], \quad x_i \in [0,1]. \tag{135}$$

This function can be factorized in a series of regular and singular basis functions as

$$\phi_i(y) = \sum_{\alpha=0}^{\infty} \begin{cases} A_{i\alpha}(x_*)R_\alpha\left(y - x_*\right), & |y - x_*| < |x_i - x_*| \\ B_{i\alpha}(x_*)S_\alpha\left(y - x_*\right), & |y - x_*| > |x_i - x_*| \end{cases}, \tag{136}$$

where

$$\begin{aligned} R_\alpha\left(y - x_*\right) &= \left(y - x_*\right)^\alpha, \quad S_\alpha\left(y - x_*\right) = \left(y - x_*\right)^{-\alpha-1}, \\ A_{i\alpha}(x_*) &= -S_\alpha\left(x_i - x_*\right), \quad B_{i\alpha}(x_*) = R_\alpha\left(x_i - x_*\right). \end{aligned} \tag{137}$$

Comparing these series with Eqs (8) and (10) we see that the convergence radii are $r_c < 1$ and $R_c > 1$, and that these can be selected as close to 1 as we want for infinite series. If instead of infinite series we use p-truncated series then we introduce the truncation error expansion as

$$\phi_i(y) = \sum_{\alpha=0}^{p-1} \begin{cases} A_{i\alpha}(x_*) R_\alpha(y - x_*), & |y - x_*| < |x_i - x_*| \\ B_{i\alpha}(x_*) S_\alpha(y - x_*), & |y - x_*| > |x_i - x_*| \end{cases} + error_p \left(\frac{|y - x_*|}{|x_i - x_*|} \right). \quad (138)$$

Assume that at the finest level

$$|x_i - x_*| \leqslant r, \quad |y - x_*| \geqslant R. \quad (139)$$

Then the error of the S-expansion of a single source is (see Appendix A)

$$S_{error} = \left(\frac{r}{R} \right)^p \frac{1}{R - r}. \quad (140)$$

Let us use k-neighborhoods and let the size of the box at the finest level is $2^{-l_{\max}}$, then

$$R = 2^{-l_{\max}} \left(\frac{1}{2} + k \right), \quad r = 2^{-l_{\max}} \frac{1}{2}, \quad (141)$$

and so

$$S_{error} = \frac{2^{l_{\max}}}{k\,(2k+1)^p}. \quad (142)$$

The error introduced by the translation operators depends on the method of computation of translations. For example, if we use $p \times p$ matrix translation operators, the error introduced by the S$|$S and R$|$R translations will be zero in the considered case, while the S$|$R translation operator for the k-neighborhood and the m-reduced S$|$R-translation scheme introduces an error (see Appendices A and B for details) for a single translation, which can be bounded as

$$(S|R)_{error} \leqslant 2^{l_{\max}+1} \frac{2^m + k}{k} \left(\frac{2^m}{2^m + 2k} \right)^p. \quad (143)$$

The total error of the MLFMM can be estimated as

$$\epsilon = N \cdot SingleSourceError = N\left[(S|R)_{error} + (S)_{error} \right] \leqslant 5N \cdot 2^{l_{\max}} \left(\frac{2^m}{2^m + 2k} \right)^p, \quad (144)$$

where we took into account that $2^m \leqslant k$ and the fact that the maximum $(S)_{error}$ and $(S|R)_{error}$ is achieved for expansion and translation of sources at the finest level. If we relate the maximum level of subdivision to the grouping parameter as $2^{l_{\max}} = N/s$, then the estimate of the MLFMM error becomes

$$\epsilon \leqslant 5\frac{N^2}{s}\left(\frac{2^m}{2^m+2k}\right)^p = F\left(p,s,k,m;N\right). \tag{145}$$

Assuming in Eq. (120) $N = M$, $CostB\left(p\right) = p$, $d = 1$, $CostFunc = 1$, $CostTrans(p) = p^2$, we have

$$CostMLFMM \lesssim 2N\left[p + s\left(k+\frac{1}{2}\right) + \frac{p^2}{s}P_4^{(k,m)}\left(1\right)\right]. \tag{146}$$

Thus for given N and ϵ one should find $p, s, k,$and m at which the cost function $CostMLFMM\left(p,s,k,m;N\right)$ reaches its minimum, subject to the constraint $\epsilon \leqslant F\left(p,s,k,m;N\right)$. This is a typical constrained multiparametric optimization problem, that can be solved using a variety of known optimization algorithms. In the above example we can explicitly express p via other parameters of Eq. (145)

$$p = \left[\log\frac{2^m+2k}{2^m}\right]^{-1}\log\frac{5N^2}{\epsilon s}. \tag{147}$$

Substituting this expression into Eq. (146) we obtain the function $CostMLFMM\left(s,k,m;N,\epsilon\right)$. At fixed k, m, N and ϵ this function of s has a minimum at some $s = s_{opt}$. Figure 18 shows that such a minimum exists for different k and m. Note that this figure also shows that the best neighborhood and S|R-translation scheme for this case is realized at $k = 1$ and $m = 0$. However, such qualitative conclusions should be made with a caution, since the error bound obtained is rather rough.

We can make several conclusions about the complexity of the MLFMM for given error bounds. If ϵ, k, m and s are fixed parameters that do not depend on N then the length of the translation vector p increases with N as $O(\log N)$. This yields for $CostTrans(p) = p^2$

$$CostMLFMM = O\left(N\log^2 N\right). \tag{148}$$

Of course for cheaper translation costs this complexity can be improved. However, the cost of the MLFMM is bounded by $O\left(N\log N\right)$ due to the Np term in Eq. (147). If s varies with N in such a way so that we always have its optimal value, $s = s_{opt}\left(N\right)$, then the asymptotic complexity of the MLFMM for given error bounds can be estimated as

$$CostMLFMM_{opt} = O\left(N\log N\right). \tag{149}$$

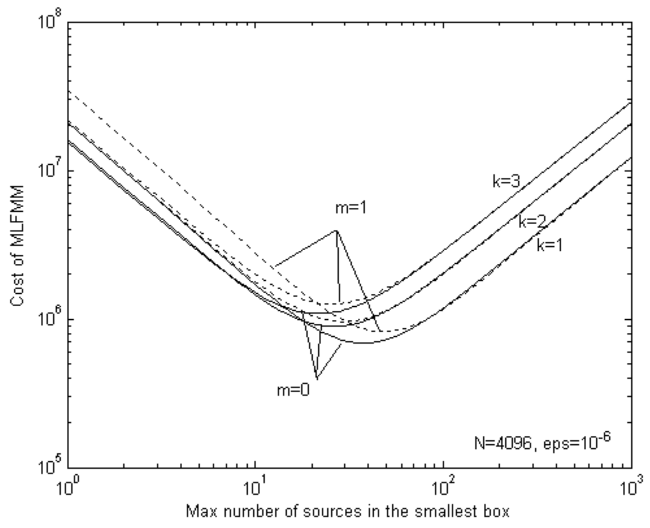


FIG. 18: Dependence of the complexity of the MLFMM on the grouping parameter, s, for the 1-dimensional example. Different curves correspond to different k-neighborhoods, and the type of the S$|$R-translation scheme used, m (curves for $m = 0$ are shown by solid lines and for $m = 1$ – by the dashed lines). Calculation were performed using Eqns (146) and (147).

This is not difficult to show, if we notice that at large N we should have

$$s_{opt}(N) = O(\log N), \quad p = O(\log N). \tag{150}$$

Our numerical studies below show that while the theoretical estimates can provide a guidance and insight for multiparametric optimization, the real optimal values depend on details such as particular program implementation, data, processor, memory, and other factors. Also the theory usually substantially overestimate the error bounds and actual errors are much smaller than their theoretical bounds. At this point we suggest to run multiparametric optimization routines on actual working FMM codes with some a posteriori estimation of actual error (say by comparison with straightforward matrix-vector multiplication) for smaller size problems and further scaling of the complexity and optimal parameter dependences on larger scale problems.

d. Asymptotic model for multiparametric constrained optimization The example considered above shows opportunities for more general analysis and conclusions about optimal choice of the MLFMM parameters in asymptotic case of large N and small ϵ. Note that Eq. (147) can be rewritten in the form

$$p = \left[\log \frac{2^m + 2k}{2^m}\right]^{-1} \left(\log \frac{5N^2}{\epsilon} - \log s\right). \tag{151}$$

In case

$$\log \frac{5N^2}{\epsilon} \gg \log s \tag{152}$$

the dependence of p on s can be neglected. In this case the dependence of the MLFMM cost on s is simple, and so the optimal s can be found independently on p from , e.g. from Eq. (121) or Eq. (130). Then the cost of the MLFMM at optimal s, fixed N, M, ϵ and given d can be considered as a function of k and m only (see Eqs (122) and (131)), since p can be considered as a function of these parameters.

Eq. (151) with omitted term proportional to $\log s$ provides such a function, $p(k, m)$, for the 1-D example considered. In general case of d-dimensions we can extend such type of dependence on special class of functions, which truncation error decays exponentially with p (kind of expansions, which converge as geometric progressions as in the example). Assuming r_c and R_c to be close to 1 we can find that the largest error can be introduced by the S$|$R-translation from point $\mathbf{x}_{*1}$ to $\mathbf{x}_{*2}$ as exponent (see Appendix B)

$$\epsilon rr_p = C\sigma^{-p}, \quad \sigma = \frac{\min\left(|\mathbf{x}_{*2} - \mathbf{x}_{*1}| - |\mathbf{y} - \mathbf{x}_{*2}|\right)}{\max|\mathbf{x}_i - \mathbf{x}_{*1}|} > 1, \tag{153}$$

where $\mathbf{x}_i$ belongs to the box centered at $\mathbf{x}_{*1}$ and $\mathbf{y}$ belongs to the box centered at $\mathbf{x}_{*2}$. The value of C also can depend on σ and the box size, but does not depend on p. The total error of the MLFMM can be then estimated similarly to Eq. (144), so we have

$$\epsilon \sim NC\sigma^p \tag{154}$$

and

$$p = \frac{1}{\log \sigma} \log\left(\frac{NC}{\epsilon}\right) \sim \frac{\gamma}{\log \sigma}, \quad \gamma = O\left(\log \frac{N}{\epsilon}\right). \tag{155}$$

where γ depends on N and ϵ. In the present asymptotic model we neglect dependence of a on s and σ using arguments of type (152), or assuming

$$\log \frac{N}{\epsilon} \gg \log \sigma, \quad \log \frac{N^2}{\epsilon} \gg \log s. \tag{156}$$

Using estimates for k-neighborhood and the reduced schemes (see discussion before Eq. (44)), we obtain from Eq. (153) the following expression for σ as a function of k and m :

$$\sigma\left(k,m\right)=\frac{\left[\left(d-1\right)\left(2^{m}-1\right)^{2}+\left(2^{m}+2k+1\right)^{2}\right]^{1/2}-d^{1/2}}{2^{m}d^{1/2}}>1. \tag{157}$$

This formula simplifies for non-reduced S|R-translation schemes as

$$\sigma\left(k,0\right)=\frac{2\left(k+1\right)-d^{1/2}}{d^{1/2}}>1. \tag{158}$$

With the known dependences $\sigma\left(k,m\right)$ and $p\left(\sigma\right)$, Eq. (122) for the MLFMM cost optimized with respect to s turns into the following function of k and m :

$$CostMLFMM_{opt}\left(k,m\right)\lesssim NCostB\left(\frac{\gamma}{\log\sigma}\right)+M\frac{\gamma}{\log\sigma} \tag{159}$$
$$+2\left[\frac{2^{d}P_{4}^{(k,m)}\left(d\right)\left(2k+1\right)^{d}}{2^{d}-1}MN\,CostTrans\left(\frac{\gamma}{\log\sigma}\right)CostFunc\right]^{1/2}.$$

This function then can be also optimized to determine the optimum k and $m.$ Consider a simplified example, when $M=\lambda N$, $CostTrans\left(p\right)=p^{2}$, $CostFunc=1$, and $CostB\left(p\right)=p.$ In this case we have

$$CostMLFMM_{opt}\left(k,m\right)\lesssim\frac{N\gamma}{\log\sigma\left(k,m\right)}\left\{1+\lambda+2\left[\lambda\frac{2^{d}P_{4}^{(k,m)}\left(d\right)\left(2k+1\right)^{d}}{\left(2^{d}-1\right)}\right]^{1/2}\right\}. \tag{160}$$

The optimum parameter sets (k_{opt},m_{opt}) for some values of d and λ are provided in the table below

d	1	1	1	1	2	2	3	4	5
λ	1	20	200	$10^{\pm3}$	1	$10^{\pm3}$	1	1	1
k_{opt}	1	2	3	4	1	2	1	2	2
m_{opt}	0	0	0	0	0	0	0	1	0

As it is seen the balance between the term responsible for the overall translation cost and the term that is responsible for expansion and convolution of the coefficients and basis functions depends on $\lambda,$ which in its turn influences the minimum of the cost function (note that λ and λ^{-1} provide the same optimal sets (k_{opt},m_{opt})). This means that special attention for optimization should be paid when the number of sources and evaluation points are substantially different. This balance can be also controlled by the translation and function evaluation costs and parameter $\gamma,$

in case $CostTrans\left(p\right) \neq p^2$. It is also noticeable that the reduced S|R-translation scheme can achieve the best performance within the specified error bounds. We found that this is the case for $d = 4$, where $m_{opt} = 1$, and did not go with analysis of this example case for dimensions larger than 5.

V. NUMERICAL EXPERIMENTS

The above algorithms for setting hierarchical data structure of 2^d-trees were implemented using Matlab and C++. We also implemented a general MLFMM algorithm in C++ to confirm the above estimates. Our implementation attempted to minimize the memory used, so for determination of nonzero neighbors and children we used $O(\log N)$ standard binary search routines. Numerical experiments were carried out for regular, uniformly random, and non-uniform data point distributions. In our experiments we varied several parameters, such as the number of points, the grouping parameter that determine the finest level of the hierarchical space subdivision, the dimensionality of the space, the size of the neighborhood, the type of the $S|R$-translation scheme and the cost of translation operations.

As a test-case for performing the comparisons we applied the FMM to the computation of a matrix-vector product with the functions $\phi_i\left(\mathbf{y}\right) = \left|\mathbf{y} - \mathbf{x}_i\right|^2$, $\mathbf{y}, \mathbf{x}_i \in \mathbb{R}^d$ and corresponding factorization of the square of distance in d-dimensional space. This function is convenient for tests since it provides exact finite factorization (degenerate kernel), and also enables computation and evaluation of errors. A good property of this function for tests also comes from the fact that it is regular everywhere in the computational domain and a method, that we call "Middleman" can be used for computation, which realizes computation with a minimum cost (124).

Our experiments were performed on a PC with an Intel Pentium III 933 MHz processor, and 256 MB memory (several examples with larger number of points were computed with 1.28 GB RAM). The results and some analysis of the computational experiments are presented below.

A. Regular Mesh of Data Points

First we performed tests with the regular multilevel FMM with $N = 2^{dl_{\max}}$ sources distributed regularly and uniformly, so at level $l_{\max}$ in a 2^d-tree hierarchical space subdivision each box contained only one source. The number of evaluation points was selected to be equal, $M = N$. Even though for the regular mesh neighbor and children search procedures are not necessary, we did not change the algorithm, so that $O(\log N)$ and $O(\log M)$ overhead for search in source and target data hierarchies was needed for these computations.

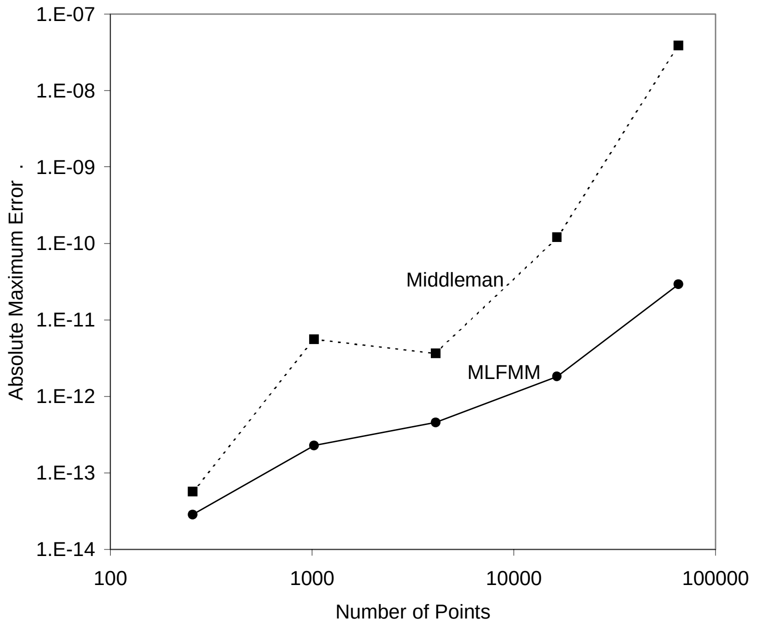


FIG. 19: A dependence of the absolute maximum error (with respect to the conventional method) on the number of points for MLFMM and Middleman method. Dimensionality of the problem $d = 2$, size of neighborhoods $k = 1$, reduced S|R-translation scheme, computations with double precision.

The accuracy of the FMM method was checked against straightforward computation of the matrix-vector product. In Figure 19 some results of such testing are presented. The absolute maximum error (assuming that $u_i = 1$, $i = 1, ..., N$) in the result was found as

$$\epsilon = \max_j \left| \varphi_{FMM}(y_j) - \varphi_{Conventional}(y_j) \right| . \tag{161}$$

For computations with double precision the error is small enough and it grows with increase in the number of operations. Since the factorization of the test function was exact this provides an idea of accuracy of the method itself, independent from the accuracy of the translation operations, which have their own error if the factorization is approximate (e.g. based on truncation of infinite series). Note that the accuracy of the FMM in our tests was higher than in the Middleman method, which can be related to the fact that translations in the FMM are performed over smaller distances, and the machine error growth is slower.

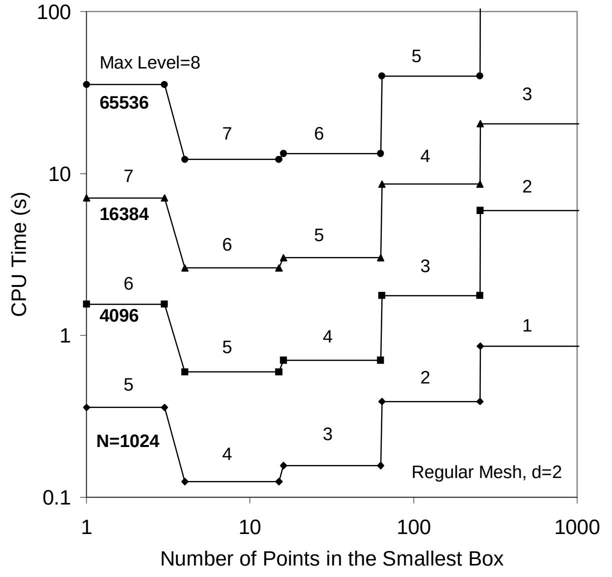


FIG. 20: CPU Time vs the number of points in the smallest box of the hierarchical space subdivision (grouping parameter s) for the multilevel FMM (Pentium III, 933 MHz, 256 MB RAM). Each staircase curve correspond to the number of points in computational domain N indicated near corresponding curve. Numbers near curves show the maximum level of the space subdivision realized at corresponding s. $d = 2, k = 1$,reduced S$|$R-translation scheme.

Figure 20 shows the CPU time required for the FMM found as a result of three series of computations for two-dimensional case ($d = 2$) with $N = 2^{10}, 2^{12}, 2^{14}$ and 2^{16} points. In these computations we varied the grouping parameter s. Because the distribution was regular, the maximum level of subdivision was constant at variations of the grouping parameter s between 2^{dl} and $2^{d(l+1)} - 1$, $l = 0, 1, ..., l_{\max} - 1$. Consequently, the number of operations for such variations was the same and the CPU time did not depend on s. For $s = 2^{dl}$, $l = 1, ..., l_{\max}$ we have jumps that correspond to change of the maximum level of the space subdivision. The conventional (straightforward) computation of the matrix-vector product corresponds to $s = 2^{dl_{\max}} = N$.

This figure shows also the heavy dependence of the CPU time on the grouping parameter and existence of a single minimum of the CPU time as a function of s. This is consistent with the results of the theoretical analysis of the computational cost of the FMM for a regular mesh (see Eq. (121) and associated assumptions above). Figure 20 also shows that the optimal value of the grouping parameter in the range of computations does not depend on N. However, for larger N

some dependence may occur for algorithms using binary search procedures in sorted lists such as provided by Eq. (130).

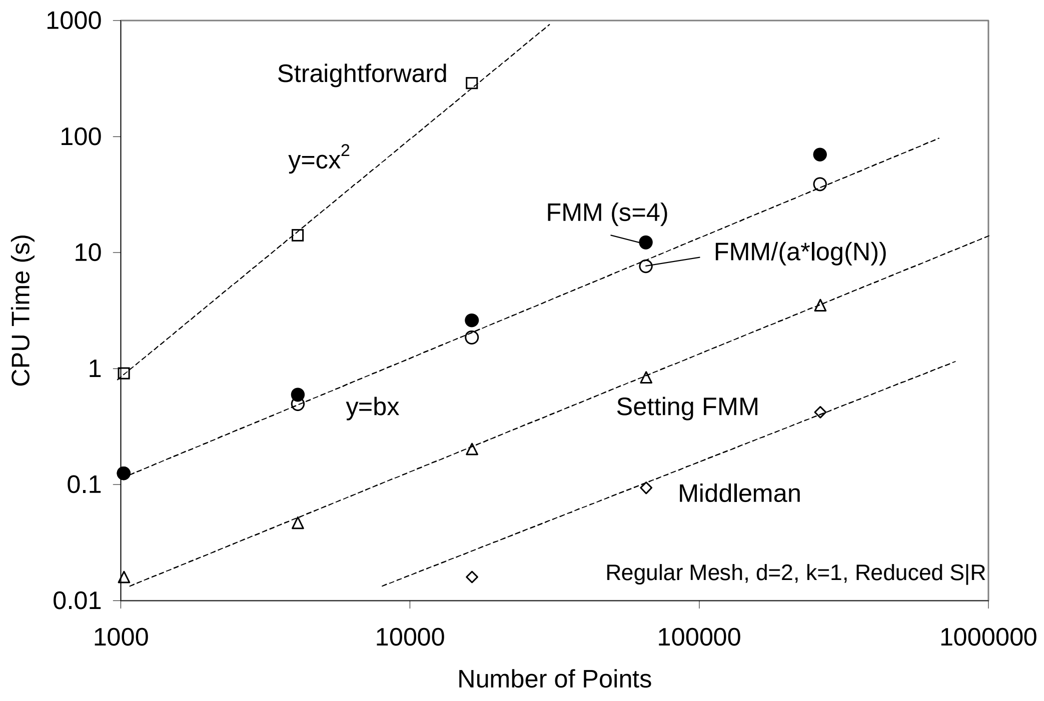


FIG. 21: Dependence of the CPU Time on the number of points, N, for computation of matrix-vector product using straightforward method (the open squares), multilevel FMM with the grouping parameter $s = 4$ (the filled circles) and the Middleman method (the open diamonds). The cost of setting the data structure required for initializing of the FMM is indicated by the open triangles. The open circles show the CPU time for the FMM scaled proportionally to $\log N$. Quadratic and linear dependences, which in logarithmic coordinates are represented by straight lines, are shown by the dashed lines. Computations are performed on a 933 MHz Pentium III, 256 MB RAM.

Figure 21 demonstrate dependence of the CPU time required by different methods to compute a matrix-vector product on a regular mesh on the number of points N. As it is expected the conventional (straightforward) method has complexity $O(N^2)$. In logarithmic coordinates this fact is reflected that the results are close to the straight line with slope 2. The FMM requires about the same time as the conventional method for $N \sim 10^2$ and far outperforms the conventional method at large N and a good choice of the grouping parameter s (in this case 100 times faster for

$10^4 < N < 10^5$). For the FMM the dependence of the CPU time on N is close to linear at low N and shows systematic deviation from the linear dependence at larger N and fixed $s.$ For fixed s the asymptotic complexity of the FMM is of order $O(N \log N)$ according Eq. (118). To check this prediction we scaled the CPU time consumed by the FMM by a factor proportional to $1/logN$. The dependence of this scaled time on N is close to linear, which shows that the version of the FMM used for numerical tests is a $O(N \log N)$ algorithm. If at large N the optimal s depends on N and the computations are always performed with the optimal $s(N),$ equation (132) shows that the asymptotic behavior of the FMM should be $O(N \log^{1/2} N)$. However in the present study we found that in the range $N < 10^6$ for the regular mesh ($d = 2$) the optimal $s = 4$ for the reduced S$|$R-translation scheme and so the asymptotic complexity of the FMM at larger N can be validated on tests with $N > 10^6$, that should be performed on workstations with larger RAM.

Note that for evaluation of the efficiency of the FMM we separated the costs of the performing of initial data setting, which should have $O(N \log N)$ complexity, but with much smaller constant than the cost of the FMM procedure itself. Figure 21 demonstrates that indeed this cost is a small portion (10% or so for the present case). In addition for multiple computations with the same spatial data points this procedure need be called only one time. As is seen from our results the CPU time required for this step grows almost linearly with N, which shows that

$$CostSetting \sim aN + bN \log N, \tag{162}$$

is the complexity realized in the range of N investigated, with $b \log N \ll a.$

The curve for the best performance that can be achieved by the Middleman method shows a linear dependence of the CPU time on N, as expected from Eq. (124) (the point corresponding to this method at $N = 16384$ shown in Figure 21 is not very accurate, which can be explained by the fact that the CPU time was measured with an accuracy of 15 ms). Comparison of this graph with the curves for the FMM shows that the overhead of the FMM arising from the translations and search procedures in the present case exceeds the cost of the initial expansion and evaluation by 100 times (for optimal choice of the grouping parameter and $N < 10^5$). At larger N, because of the nonlinear growth of the asymptotic complexity of the FMM with N, this ratio increases.

The graphs shown in Figures 22 - 24 demonstrate some results of study of the influence of the cost of a single translation on the CPU time. For this study we artificially varied the cost of translation by adding to the bodies of functions computing $(S|S)$, $(S|R)$, and $(R|R)$ translations Q

additional multiplications. So a single translation cost became $CostTrans(p) + Q$. The parameter Q was varied between 1 and 10^5.

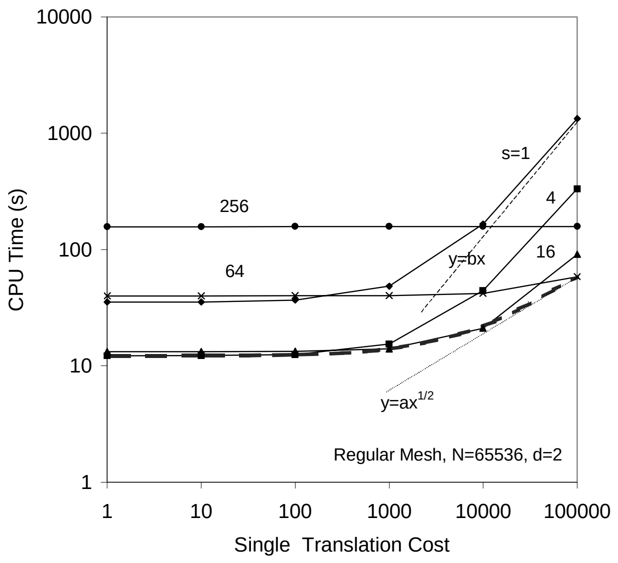


FIG. 22: Dependences of the CPU time for multilevel FMM on the cost of a single translation at various values of the grouping parameter s. (933 MHz Pentium III, 256 MB RAM). The thick dashed curve shows a dependence of the minimum time on the cost of a single translation. The neighborhoos and dimensionality are the same as in Figure 20

In the test matrix-vector computations the actual $CostTrans(p)$ was small (of the order of 10 multiplications). Figure 22 shows that addition to this cost up to 100 multiplications almost did not effect the CPU time. Since Q is much larger than the real cost, the artificial $CostTrans(p) \approx Q$. Increase of Q for low grouping parameters s leads to substantial increase in the computational time. Asymptotically this is a linear growth proportional to Q so these dependences at larger Q in logarithmic coordinates are represented by straight lines with the slope 1. The fact that curves with lower s show stronger dependence on Q is explainable, since lower s results in larger number of hierarchical levels of space subdivision, and therefore in larger number of translations. In contrast, at large s the relative contribution of the cost of all translations to the cost of the FMM is smaller compared to the cost of straightforward summations, so the curves with larger s are less sensitive to the cost of translation. It is interesting to consider the behavior of the curve that connects points providing the fastest computation time at a given Q. In Figure 22 these points correspond to $s = 4...15$ for $Q \leqslant 10^2$, to $s = 16...63$ for $10^2 < Q \leqslant 10^4$ and $s = 64...255$ for $Q = 10^5$ (see Figure 23). For these points the total FMM CPU time almost does not depend on Q for $Q \leqslant 10^2$ and then starts to grow. Eq. (122) shows that at optimal selection of the grouping parameter s and large translation costs the computational complexity should be proportional to $Q^{1/2}$. This agrees well with the results obtained in numerical experiments, since the CPU time at optimal s approaches the asymptote, which has in the logarithmic coordinates slope $1/2$. This asymptote is shown in Figure 22. Figure 23 also shows the theoretical prediction that $s_{opt} \sim Q^{1/2}$ at large Q. The line corresponding to this dependence crosses the vertical bars at $Q \geqslant 10^2$ which shows that the results of the computations are in agreement with the theory.

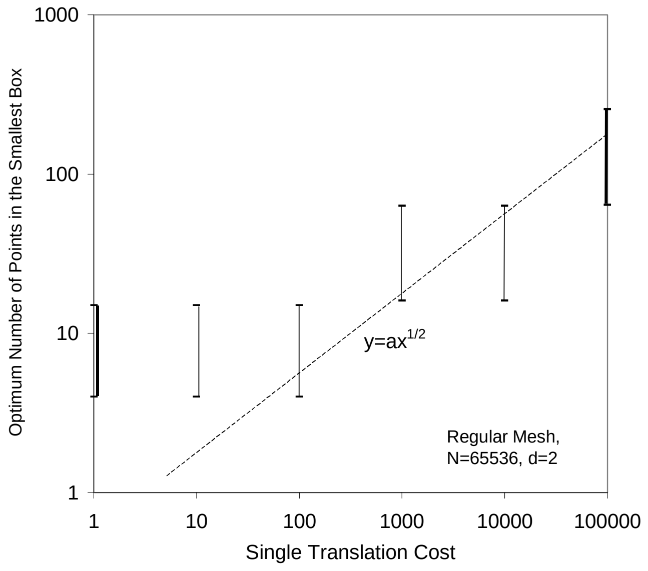


FIG. 23: Dependence of the optimal ranges of the grouping parameter s on the cost of a single translation (shown by the vertical bars). The dashed line shows the theoretical prediction for the optimal s. The dimension and neighborhoods are the same as in Figure 20.

Figure 24 demonstrates that at fixed grouping parameter s, dependencies of the CPU time on the number of points are qualitatively different. At low Q the cost of logarithmic search procedures starts to dominate for larger N and the FMM algorithm should be considered as $O(N \log N)$ or $O(N \log^{1/2} N)$ (if s is chosen $\sim \log N$). At high Q (formally at $Q \gg \log N$), however, the cost of single translation dominates over $\log N$ terms and the asymptotic complexity of the algorithm is $O(N)$. Of course, for any fixed Q there will be found some N such that $Q \ll \log N$ and

$O(N \log N)$ asymptotics should hold anyway. From a practical point of view anyway N's are limited by computational resources and so the condition $Q \gg \log N$ may hold in many practical cases, and the MLFMM can be considered as $O(N)$ algorithm (see also the discussion near Eq. (115)).

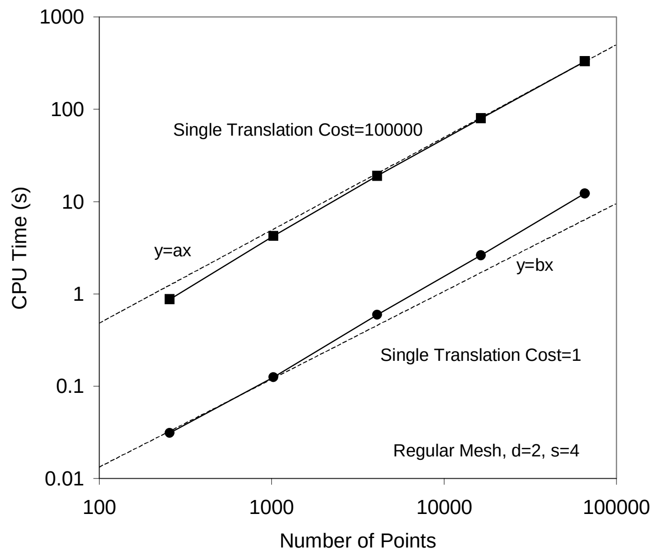


FIG. 24: Dependence of the CPU time for the multilevel FMM on the number of data points at small and large costs of a single translation (933 MHz Pentium III, 256 MB RAM).

Figures 25 and 26 illustrate the influence of the size of the neighborhood and the type of the translation scheme (reduced, $m > 0$, or non-reduced, $m = 0$, see Eq. (47) and around) on the CPU time. First we note that according Eq. (128) the size of the neighborhood, k, does not influence the optimum s for the regular mesh and the non-reduced scheme of translation. We checked this fact numerically and found that it holds when we varied k between 1 and 3. The optimum value of s for the reduced S$|$R-translation scheme may be smaller than for the non-reduced scheme, due to $P_4^{(k,m)}(d)$ at $m > 0$ is always smaller than $P_4^{(k,m)}(d)$ at $m = 0$ and s_{opt} depends on $P_4^{(k,m)}(d)$ according Eq. (56). We also checked this fact numerically for $k = 1, 2, 3$ and the reduced scheme with $m = 1$ at varying single translation costs Q and found that for low Q the optimum value of s indeed is smaller for the reduced scheme.

These figures show that the CPU time can increase several times for the same computations with different sizes of the neighborhood, and depend on the S$|$R-translation scheme used. Eq.

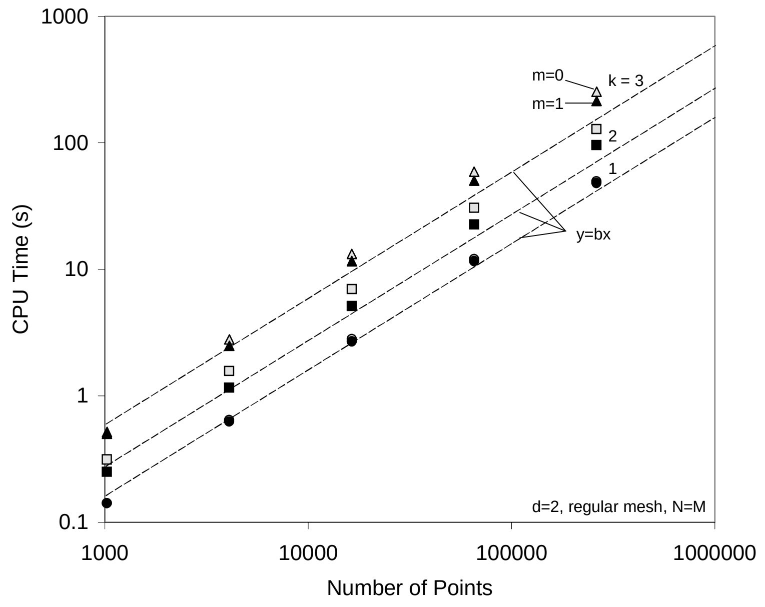


FIG. 25: Dependence of the CPU time on the number of points, N, for computation of the matrix-vector product using multilevel FMM with different sizes of neighborhoods, k (circles: $k = 1$, squares: $k = 2$, and triangles: $k = 3$) and S$|$R-translation scheme (non-reduced, $m = 0$, all boxes in the E4 neighborhood are of the same level, shown by the open circles, squares, and triangles, and reduced, $m = 1$, maximum box size in the E4 neighborhood of parent level, shown by the filled circles, squares, and triangles). The cost of a single translation, Q, is low, $Q < 100$, and the grouping parameter s is optimal for each computation ($s = 4$ for $m = 1$ and $s = 16$ for $m = 0$). The dashed lines show linear complexity. Computations are performed on 933 MHz Pentium III, 1.28 GB RAM.

(122) provides the following asymptotics for the ratio of the MLFMM complexity at different k and m when the parameter s is selected in an optimal way:

$$\left.\frac{CostMLFMM_{opt}\left(k_1,m_1\right)}{CostMLFMM_{opt}\left(k_2,m_2\right)}\right|_{CostTrans\to\infty} \sim \left[\frac{P_4^{(k_1,m_1)}\left(d\right)\left(2k_1+1\right)^d}{P_4^{(k_2,m_2)}\left(d\right)\left(2k_2+1\right)^d}\right]^{1/2} \tag{163}$$

particularly for $m_1 = m_2 = 0$, we have

$$\left.\frac{CostMLFMM_{opt}\left(k_1,0\right)}{CostMLFMM_{opt}\left(k_2,0\right)}\right|_{CostTrans\to\infty} \sim \left(\frac{2k_1+1}{2k_2+1}\right)^d. \tag{164}$$

The CPU time ratios evaluated using Eq. (163) are shown in Figure 26 by horizontal lines. It is seen that these predictions more or less agree with the numerical results and can be used for

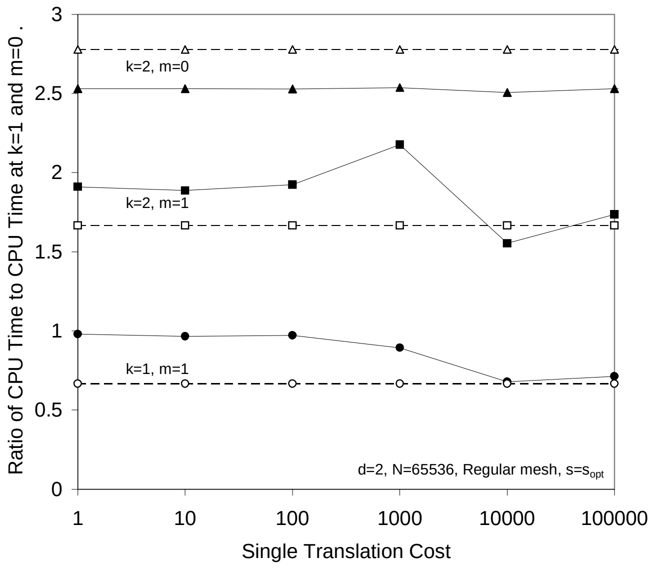


FIG. 26: Dependence of the ratio of CPU times on the cost of a single translation, Q, for different sizes of the neighborhood, k, and different S|R-translation schemes, with $m = 0$ and $m = 1$. The CPU times are normalized with respect to the CPU time obtained for $k = 1$ and $m = 0$. The horisontal lines show the theoretical prediction for large Q. The optimal value of the grouping parameter depends on m and Q and for each computation this optimal value was used. Computations are performed on 933 MHz Pentium III, 1.28 GB RAM.

scaling and predictions of the algorithm complexity. Note that despite $P_4^{(1,0)} = P_4^{(2,1)}$ the CPU time for computations with ($k = 1$, $m = 0$) and ($k = 2$, $m = 1$) differ due to additional multiplier $[(2k_1 + 1)/(2k_2 + 1)]^{d/2}$ in Eq. (163), which is due to the larger number of sources in the neighborhood of the evaluation box at the final summation stage.

Figure 27 demonstrates dependence of the CPU time on the number of points N for various dimensions d. It is clear that the CPU time increases with d. In this computations we used 1-neighborhoods with regular S|R-translation scheme ($m = 0$) which is valid for dimensions $d = 1, 2, 3$. Computations with larger dimensions require larger size of neighborhoods.

Figure 28 shows dependences of the CPU time on d at fixed N and various s. Estimation (129) shows that the number of operations at fixed N and fixed or optimal s grow with d exponentially as a^d. Such dependence is well seen on the graph and can be used for scaling and predictions of the algorithm performance.

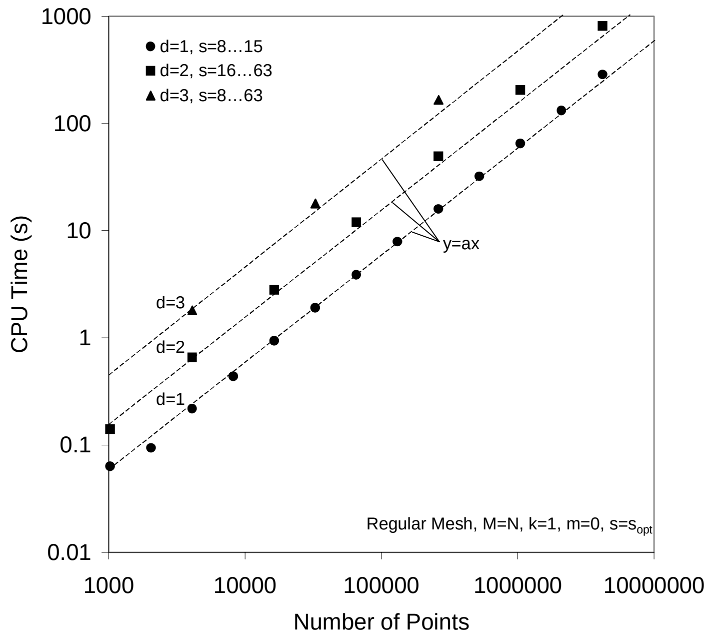


FIG. 27: Dependence of the CPU time for the multilevel FMM on the number of points N in the regular mesh for optimal values of the grouping parameter s for various dimensions of the space d (Pentium III, 933 MHz, 1.28 GB RAM).

Figure 29 demonstrates the dependence of the absolute error on the truncation number for the 1-D example (135) (see discussion below this equation). Since these functions are singular at $y = x_i$, we selected the evaluation points to be on a regular grid shifted from a regular grid of source points of the same size (so the source and evaluation points are interleaved). The absolute error was computed by comparison of the results obtained by the MLFMM and by straightforward matrix-vector multiplication in double precision arithmetic. It is seen that this error is several orders of magnitude smaller than the theoretical error bound provided by Eq. (145). However, the theoretical and computed slopes of the error curves in the semilogarithmic coordinates agree well. This slope is determined by the parameters k and m. For larger k the truncation number p can be several times smaller to achieve the same computational error.

However, because an increase of k leads to an increase in the power of the neighborhood, the optimal set of parameters that provides the lowest CPU times is not obvious, and can only be found by a multiparametric optimization procedure. We performed multiple runs and used some standard optimization routines to determine such sets of parameters for several cases. The results

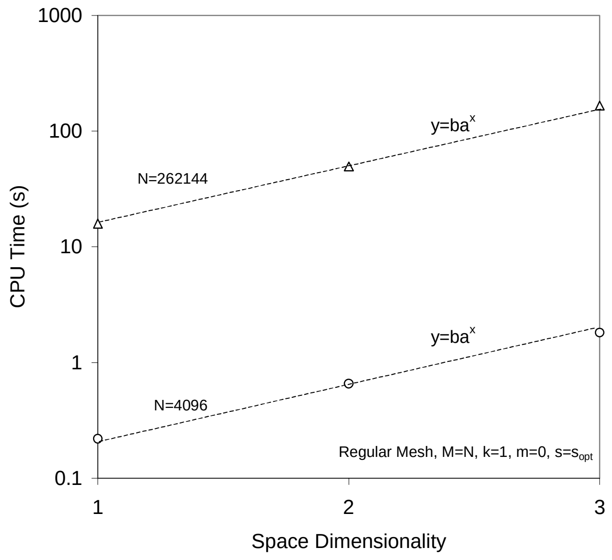


FIG. 28: Dependence of the CPU time on the space dimension d for the multilevel FMM, at optimal values of the grouping parameter s, and at two different values of points N in the regular mesh. The dashed lines show exponentials in semi-logarithmic axes used (Pentium III, 933 MHz, 1.25 GB RAM).

for $N = M = 4096$ and for a specified error of computation $\epsilon < 10^{-10}$ are shown in the table.

k	m	s	p	Actual error	CPU time (s)
1	1	32...63	42	$9.37 \cdot 10^{-11}$	0.156
1	0	32...63	29	$9.82 \cdot 10^{-11}$	0.156
2	1	32...63	27	$8.37 \cdot 10^{-11}$	0.187
2	0	32...63	20	$8.91 \cdot 10^{-11}$	0.187
3	1	32...63	19	$8.55 \cdot 10^{-11}$	0.234
3	0	32...63	16	$8.28 \cdot 10^{-11}$	0.234

It is seen that the optimal grouping parameter, s, for all cases appeared to be in the range 32...63 (because in the regular mesh for $d = 1$ there is no difference between the computations with s varying between 2^l and $2^{l+1} - 1$, $l = 0, 1, ..., l_{\max} - 1$). The optimal p depends on k and m and reduces with increasing k, and increases with m at fixed k. It is interesting that in the example considered, and for the data used, the growth of the optimal p with m is almost compensated by the reduction in the number of S$|$R-translations. Thus, the schemes with $m = 0$ and $m = 1$ have the same performance despite having different p. The best scheme for these N and ϵ appear to be

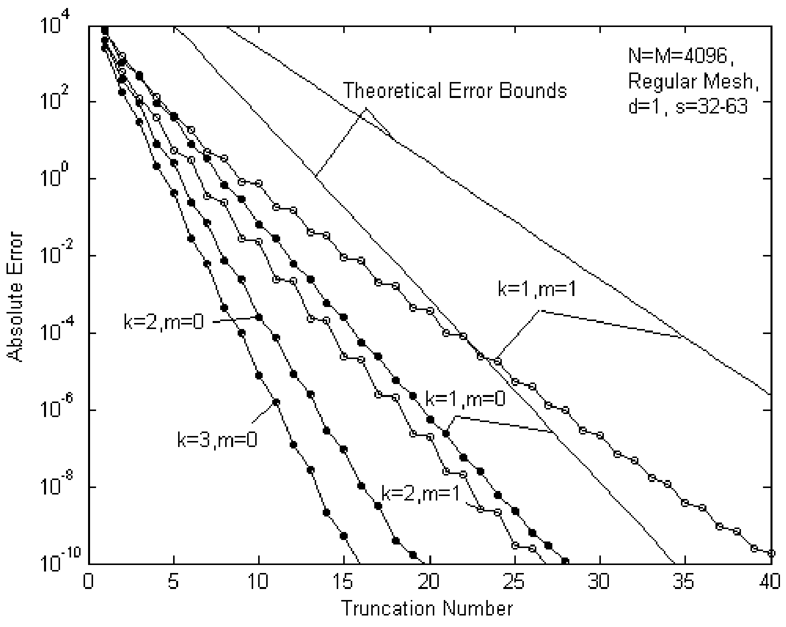


FIG. 29: Dependence of the absolute error, ϵ on the truncation number, p, for a 1-dimensional problem. Different curves correspond to different values of the parameters k and m characterizing the neighborhoods used. The curves shown by open ($m = 1$) and filled ($m = 0$) circles correspond to the actual computations. The solid lines show theoretical error bounds predicted by Eq. (145) for $k = 1$ and $m = 0$ and 1.

that with $k = 1$. However we note that this result changes for larger dimensions (simply because any scheme with $k = 1$ works only for $d \leqslant 3$ as discussed above).

Finally, we performed some tests with the regular meshes to verify the prediction of the asymptotic theory for multiparametric optimization (that at large or small ratios $\lambda = M/N$ the optimal neighborhoods should be observed at larger k and $m = 0$, for $d = 1$). For this purpose we generated M evaluation points in a coarse regular grid whose nodes were different from the source locations on a fine regular mesh. For $10^{-3} < \lambda < 10^3$ we found that the scheme with $k = 1$ and $m = 0$ provides the best performance in terms of the speed of computation at a given accuracy. For $\lambda < 10^{-3}$ we observed that indeed the minimum CPU time is achieved for larger k. One of the optimization examples at $N = 2^{20} = 1048578$ and $M = 2^7 = 128$ ($\lambda = 2^{-13} \sim 10^{-4}$) is shown in the table below

k	m	s	p	Actual error,	CPU time (s)
4	0	1024...2047	12	$1.04{\cdot}10^{-6}$	1.719
5	0	512...1023	11	$1.11{\cdot}10^{-6}$	1.734
3	0	1024...2047	13	$3.45{\cdot}10^{-6}$	1.734
2	0	2048...4095	16	$1.78{\cdot}10^{-6}$	1.766
5	1	512...1023	13	$3.57{\cdot}10^{-7}$	1.812
3	1	2048...4095	15	$7.24{\cdot}10^{-7}$	1.828
4	1	2048...4095	15	$6.82{\cdot}10^{-7}$	1.875
1	0	1024...2047	23	$2.52{\cdot}10^{-6}$	1.922
2	1	1024...2047	23	$4.40{\cdot}10^{-7}$	1.984
1	1	2048...4095	33	$1.99{\cdot}10^{-6}$	2.250

In this numerical experiment we imposed an optimization constraint $ActualError < 4 \cdot 10^{-6}$ and found that the optimum s and p that minimize the CPU time, for specified k and m.Here k varied in range from 1 to 5 and m took the values 0 or 1. The table shows that the CPU times are quite close for different k and m. In any case, the table ordered with respect to the CPU time shows that, for this example, the schemes with larger k outperform the scheme with $k = 1$ and $m = 0$ both in terms of the speed of computation and accuracy. This optimization example qualitatively agrees with the theoretical prediction. Quantitative differences (that the effect is observed at smaller λ than prescribed by the theory) may be attributed to the fact that some constants that were dropped in the simplified theoretical example, e.g. we assumed that $CostTranslation(p) = p^2$, while using $CostTranslation(p) = cp^2$ we would obtain different optimal parameters for the same λ).

B. Random Distributions

To understand the performance of the multilevel FMM when the data points are distributed irregularly we conducted a series of numerical experiments for uniform random distributions. To compare these results with those obtained for regular meshes we selected first a simplified case, when $N = M$ and sets of the source and evaluation points are the same, $\mathbb{X} = \mathbb{Y}$. Figures 30 - 31 demonstrate peculiarities of this case.

In Figure 30 the dark circles show the CPU time required for matrix-vector product computation using the FMM for a uniform random distribution of $N = 4096$ points. Computations were

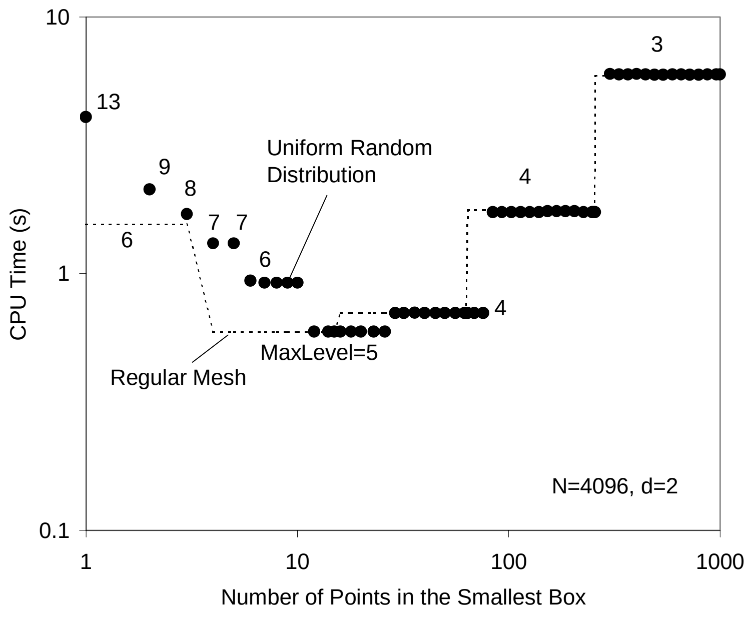


FIG. 30: Dependence of the CPU time for the multilevel FMM on the grouping parameter s for uniformly distributed data points. The filled circles correspond to a random distribution and the dashed line corresponds to a regular mesh. The numbers near the lines and the circles show the maximum level of the hierarchical space subdivision. $d = 2$, $k = 1$. Reduced S|R-translation scheme.

performed for the same data set, but with different values of the grouping parameter s. This dependence have several noticeable features. First, it is obviously seen that the CPU time reaches a minimum at s from some range. Second, that the range of optimal s is shifted towards larger values compared to the similar range for the regular distribution, discussed in the previous section. Third, that at very small s, such as $s = 1$, the CPU time for the random data is substantially larger than for data distributed on a regular mesh. Fourth, that at larger s performance of the algorithm for the random distribution is almost the same as for the regular mesh.

All these peculiarities are explainable if we indicate near the points the maximum level of the hierarchical space subdivision $l_{\max}$. It is clear that the CPU time for a fixed distribution depends not on the grouping parameter, but rather on $l_{\max}$ (which in turn is determined by s). Indeed, if two different s determine the same $l_{\max}$ the computational time should be the same for the same data set. At small values of s the maximum level of subdivision can be several times larger for random distribution than for the same number of points distributed on the regular mesh. This is clear, since the minimum distance between two points is smaller for the random distribution. Therefore smaller boxes are required to separate random points than regular points. This increases the CPU time at small s due to increasing number of translation (and neighbor search) operations. Also this explains a shift of ranges corresponding to the same $l_{\max}$ towards the larger values of s for random distributions.

If we compare the computations with the same $l_{\max}$ for random and regular distributions (such as for $l_{\max} = 6$ as shown in Figure 30), we can see that the time required for a random distribution is smaller than for a regular mesh of data points. This is also understandable for $l_{\max} > l_{\max opt}$, where $l_{\max opt}$ is the optimal maximum level of the space subdivision, at which the computational complexity is minimal, and which corresponds to s_{opt} (in the case shown in Figure 30 we have $l_{\max opt} = 5$). Indeed, for $l_{\max} > l_{\max opt}$ increase of the number of data points in the smallest box is efficient, since the cost of translations at level $l_{\max} > l_{\max opt}$ is higher than the cost of straightforward summations in the neighborhood of each evaluation point. Thus, at $l_{\max} > l_{\max opt}$ for random distributions we efficiently trade the cost of translation at larger $l_{\max}$ for straightforward evaluations which yield the CPU time reduction.

At optimal level $l_{\max opt}$ the cost of translations is approximately equal to the cost of straightforward summations in the E_2 neighborhoods. Therefore, redistribution of points should not substantially affect the computational complexity of the algorithm. This is nicely supported by the results of our numerical experiments, where we found that the optimal CPU time for a given number of points almost does not depend on their spatial distribution, as well that $l_{\max opt}$ does not depend on the particular distribution (while depending on other parameters, such as space dimensionality, type of the neighborhoods, and the cost of translation) (see Figure 30).

At $l_{\max} < l_{\max opt}$ the number of points in the boxes for uniform distributions is large enough. So the average number of operations per box is approximately the same as for the regular distribution. In some tests we observed CPU time differences for $l_{\max} < l_{\max opt}$ for regular mesh and random distributions, but these differences were relatively small. This is also seen in Figure 31, which shows dependence of the CPU time on $l_{\max}$. The curves here depend on the data point distributions. It is seen that there is a substantial difference between the dependence for regular and random distributions at $l_{\max} > l_{\max opt}$. The CPU time at the optimum level $l_{\max} = l_{\max opt}$ does not depend on distributions.

Figures 32-33 demonstrate the computation for uniform random distributions of N source and

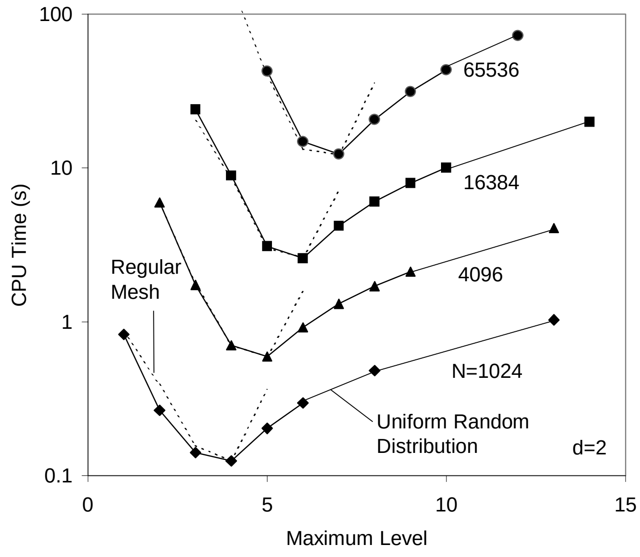


FIG. 31: Dependence of the CPU time for the regular multilevel FMM on the maximum level of hierarchical space subdivision, for random and regular uniform disributions of N data points. Dimensionality and neighborhoods are the same as in Figure 30.

M evaluation data points in the same domain when N and M substantially differ. Figure 32 shows that computations with $l_{\max opt}(M)$ provide lower CPU times.

The dependence $l_{\max opt}$ on M is shown in Figure 33. This is a logarithmic dependence,

$$l_{\max opt}(M) = [a \log M]. \tag{165}$$

We also noted in computations that the range of s_{opt} corresponding $l_{\max opt}$ depends on M and decreases with the growth of M. Such behavior is expected, since for very low M straightforward evaluation requires $O(NM)$ operations. In the limiting case $M = 1$ this evaluation should me more efficient than any other algorithm involving function reexpansions and translations. So at $M \to 1$ we should have $s_{opt} \in \left[N/2^d, N\right]$ and $l_{\max opt}(1) = 0$. At larger M, the procedure of hierarchical space subdivision becomes more and more efficient. At fixed N this leads to growth of $l_{\max opt}$ with M. Eq. (121) provides that $s_{opt} \sim M^{-1/2}$ if the cost of translations does not depend on M.

Finally, we performed a series of computations for non-uniform source and evaluation point distributions, such as shown in Figure 34. In this case there exist clusters of source and evaluation points and optimum parameters for the FMM can substantially differ from those for uniform

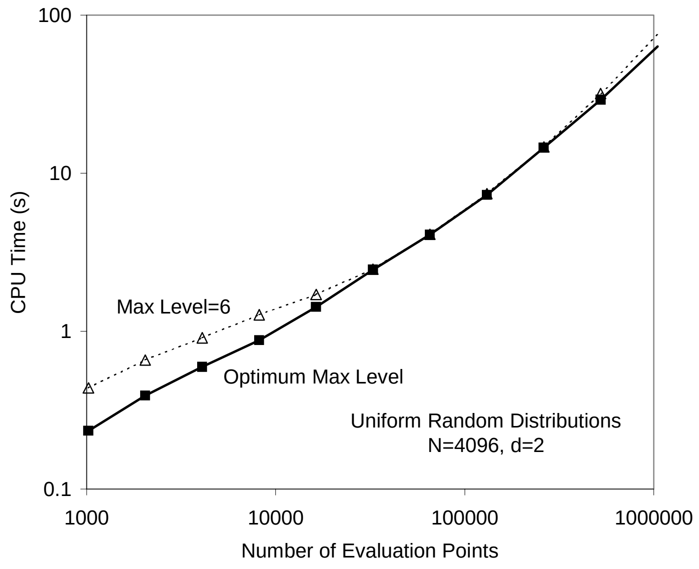


FIG. 32: Dependence of the CPU time on the number of evaluation points, M, when the number of source points, N, is fixed ($N = 4096$). Both sets have uniform random distributions within the same box. The filled squares and the solid line show computations using optimum maximum level of space subdivision, $l_{\max opt}$, while the light triangles and the dashed line show computations with fixed maximum level $l_{\max} = 6$. The dimension of the problem and the neighborhoods are the same as in Figure 30.

distributions.

Figure 35 shows the dependence of the CPU time for uniform and nonuniform distributions of the same amount of data points. Due to high clustering, the nonuniform distribution shows substantially different ranges for the optimum value of the grouping parameter s. One also can note that the minimum CPU time for this nonuniform distribution is smaller than that for the uniform distribution. We hope to present more detailed analysis of the FMM optimization and behavior for nonuniform distributions in a separate paper, where fully adaptive versions of the MLFMM will be considered and compared with the regular MLFMM.

VI. CONCLUSIONS

On the basis of theoretical analysis, we developed a $O(N \log N)$ multilevel FMM algorithm that uses 2^d-tree hierarchical space subdivision and general formulation in terms of requirements

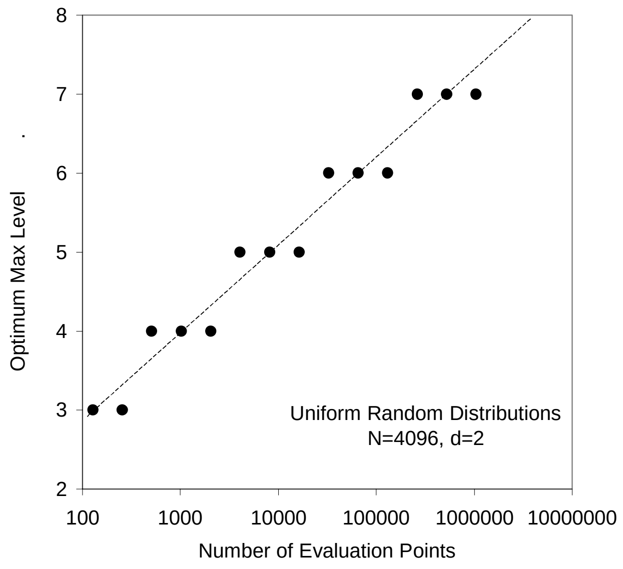


FIG. 33: Dependence of the optimum maximum level of hierarchical space subdivision on the number of evaluation points for a fixed number of the source points. All points are uniformly distributed inside the same box. $d = 2, k = 1$, reduced S$|$R-translation scheme.

for functions, for which the FMM can be employed. Numerical experiments show good performance of this algorithm and substantial speed up of computations compared to conventional $O(N^2)$ methods. Theoretical considerations shows however that $O(N \log N)$ represents some intermediate asymptotics, since $\log N$ in asymptotics is dictated by memory saving methods for search in sorted lists, and should be bounded by the cost of translation. Strictly speaking the MLFMM can be considered the $O(N)$ method.

We found also that the optimal selection of the grouping parameter is very important for efficiency of the regular multilevel FMM. This parameter can depend on many factors, such as number of the source and evaluation points, cost of single translation, space dimensionality, size of the neighborhood and scheme of the S$|$R-translation and data point distributions.

The complexity of the MLFMM at optimum choice of the grouping parameter depends on the length of the vector of expansion coefficients p as $O\left(\min\left(p, [CostTrans(p)]^{1/2}\right)\right)$. We obtained this result theoretically and confirmed in numerical experiments. For $CostTrans(p) = O(p^2)$ the dependence of the optimized MLFMM complexity on p is linear.

In case of function factorization with infinite series with exponential decay of the error with the

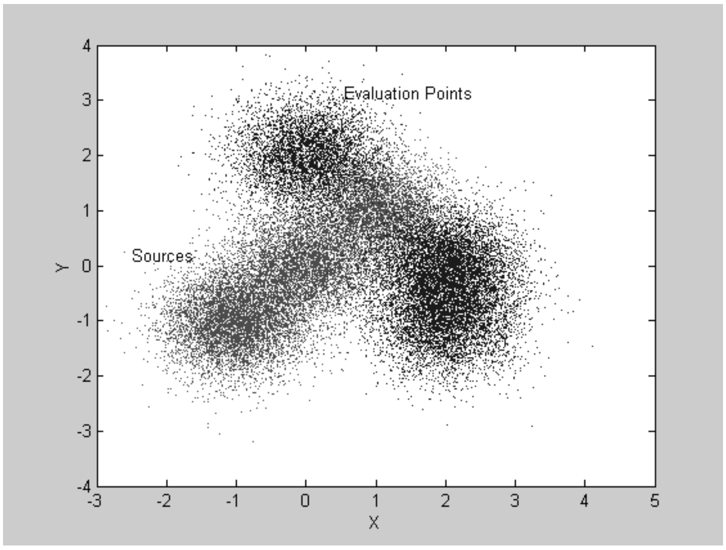


FIG. 34: Example of a non uniform distribution for $N = 16384$, source points, and $M = 16384$, evaluation points. Points were generated using a sum of six Gaussians with different centers and standard deviation. $d = 2$.

truncation number p the complexity of optimized MLFMM that performs computations within the specified error bounds is $O(N \log N)$. This is due to increase of p with N. In computations with controlled error the size of the optimum neighborhoods depends on several factors (dimension, translation cost, etc.). This includes the ratio of the number of the source and evaluation points, $\lambda = M/N$. At large and small λ substantial variations of the size of optimum neighborhood can be observed.

We found that theoretical estimations of the algorithm performance and its qualitative behavior agree well with numerical experiments. The theory also provides insight and explanation of the computational results. This allows us to use the theory developed for prediction and optimization of the MLFMM in multiple dimensions.

Finally, we should mention that the data structures considered in the present study are not the only ones for use in the FMM. Also the base framework provided in this study can be modified to turn the method in a fully adaptive scheme that we are going to present in a separate study.

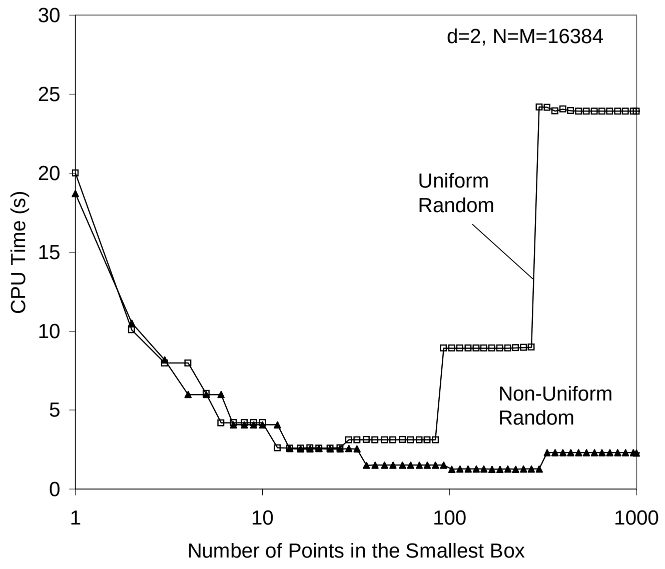


FIG. 35: Dependence of the CPU time (933 MHz, Pentium III, 256 MB RAM) required for the multilevel FMM for computations on random uniform (open squares) and non-uniform (filled triangles) data point distributions. The non-uniform distribution is shown in Figure 34. 1-neighborhoods and reduced S|R-translation scheme are used.

Acknowledgments

We would like to gratefully acknowledge the support of NSF grants 0086075, 0219681, and internal funds from the Institute for Advanced Computer Studies at the University of Maryland. We would also like to thank Prof. Hanan Samet for discussions on spatial data-structures, and Profs. Larry Davis for allowing us to offer a graduate course on the Fast Multipole Method, and Prof. Joseph JaJa for providing us internal UMIACS support for work on this problem.

[1] Hanan Samet, “Applications of Spatial Data Structures,” Addison-Wesley, 1990.

[2] Hanan Samet, “The Design and Analysis of Spatial Data Structures,” Addison-Wesley, 1994.

[3] G. Peano, “Sur une courbe qui remplit toute une aire plaine,” Mathematische Annalen 36, 1890, 157-160.

[4] J.A. Orenstein & T.H. Merret, “A class of data structures for associative searching”, Proceedings of

the Third ACM SIGAT-SIGMOD Symposium on Principles of Database Systems, Waterloo, 1984, 181-190.

[5] H.Cheng, L. Greengard & V. Rokhlin, “A fast adaptive multipole algorithm in three dimensions,” J. Comp. Physics 155, 1999, 468-498.

[6] L. Greengard, “The Rapid Evaluation of Potential Fields in Particle Systems,” MIT Press, Cambridge, MA, 1988.

[7] N.A. Gumerov & R. Duraiswami, “Fast, Exact, and Stable Computation of Multipole Translation and Rotation Coefficients for the 3-D Helmholtz Equation,” University of Maryland, Institute for Advanced Computer Studies, Technical Report UMIACS TR 2001-44, 2001.

[8] L. Greengard and V. Rokhlin, “A Fast Algorithm for Particle Simulations,” *J. Comput. Phys.*, 73, December 1987, pages 325348.135, 280-292 (1997).

[9] J.J. Dongarra and F. Sullivan, “ The top 10 algorithms.” *Computing in Science & Engineering*, **2** p. 22-23, 2000.

[10] Eric Darve, The fast multipole method: Numerical Implementation, Journal of Computational Physics 160, 195-240, 2000.

[11] Eric Darve, The fast multipole method I: error analysis and asymptotic complexity, SIAM J. Num. An., vol 38, pp. 98-128, 2000.

[12] W.C. Chew, J.M. Jin, E. Michielssen, J. Song, *Fast and Efficient Algorithms in Computational Electromagnetics*, Artech House, 2001.

[13] A. Elgammal, R. Duraiswami, and L. Davis, “Efficient Kernel Density Estimation Using the Fast Gauss Transform with Applications to Color Modeling and Tracking” IEEE Trans. PAMI (accepted).

[14] J. Strain. The fast Gauss transform with variable scales. SIAM J. Sci. Comput., vol. 12, pp. 1131–1139, 1991.

[15] L. Greengard and J. Strain. The fast gauss transform. *SIAM Journal on Scientific and Statistical Computing*, 12(1):79–94, 1991.

[16] A.H. Boschitsch, M.O. Fenley, W.K. Olson, “A Fast adaptive multipole algorithm for calculating screened Coulomb (Yukawa) Interactions," J. Comput. Phys., vol. 151, 212-241, 1999.

[17] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, T. R. Evans, “Reconstruction and Representation of 3D Objects with Radial Basis Functions,” Proc. ACM Siggraph pp. 67-76, August 2001.

[18] F. Chen and D. Suter. Using a fast multipole method to accelerate the evaluation of splines. IEEE Computational Science and Engineering, 5(3):24–31, July-September 1998.

[19] R. K. Beatson, J. B. Cherrie, and D. L. Ragozin, “Fast evaluation of radial basis functions: Methods for four-dimensional polyharmonic splines,” SIAM J. Math. Anal., vol. 32, no. 6, pp. 1272-1310, 2001.

[20] T.H. Cormen, C.E. Leiserson, R.L. Rivest, *Introduction to Algorithms*, MIT Press, 1990.

VII. APPENDIX A

Below we provide some expressions for the matrix translation operators used for the 1-D example used in the multiparametric optimization of the MLFMM algorithm with the function (135). We also evaluate the error bounds for these operators and for the S-expansion.

A. Translation Operators

The matrix translation operators can be found from a reexpansion of the basis functions, e.g.,

$$R_n(\mathbf{y}-\mathbf{x}_{*1}) = \sum_{m=0}^{\infty} (R|R)_{mn}(\mathbf{t}) R_m(\mathbf{y}-\mathbf{x}_{*2}), \quad \mathbf{t} = \mathbf{x}_{*2} - \mathbf{x}_{*1}, \tag{166}$$

where the coefficients $(R|R)_{mn}(\mathbf{t})$ form the elements of the matrix $(\mathbf{R}|\mathbf{R})(\mathbf{t})$. Indeed, if a function is specified by its expansion coefficients A_n near the expansion center $\mathbf{x}_{*1}$, and $\widehat{A}_n$ near the expansion center $\mathbf{x}_{*2}$, such that

$$\psi(\mathbf{y}) = \sum_{n=0}^{\infty} A_n R_n(\mathbf{y}-\mathbf{x}_{*1}) = \sum_{n=0}^{\infty} \widehat{A}_n R_n(\mathbf{y}-\mathbf{x}_{*2}), \tag{167}$$

then we have

$$\sum_{n=0}^{\infty} A_n R_n(\mathbf{y}-\mathbf{x}_{*1}) = \sum_{n=0}^{\infty} A_n \sum_{m=0}^{\infty} (R|R)_{mn}(\mathbf{t}) R_m(\mathbf{y}-\mathbf{x}_{*2}) = \sum_{m=0}^{\infty} \left[\sum_{n=0}^{\infty} (R|R)_{mn}(\mathbf{t}) A_n\right] R_m(\mathbf{y}-\mathbf{x}_{*2}), \tag{168}$$

and the translated coefficients can be computed as

$$\widehat{A}_m = \sum_{n=0}^{\infty} (R|R)_{mn}(\mathbf{t}) A_n, \tag{169}$$

or in matrix form as

$$\widehat{\mathbf{A}} = (\mathbf{R}|\mathbf{R})(\mathbf{t})\,\mathbf{A}. \tag{170}$$

This is the form specified by Eq. (13). Similar expressions can be obtained for the S|S and S|R operators.

1. *R|R-operator*

For the present example, with the basis functions given by Eq. (137), we have

$$R_n(y+t) = (y+t)^n = \sum_{m=0}^{n} \frac{n!}{m!\,(n-m)!} t^{n-m} y^m = \sum_{m=0}^{n} \frac{n!}{m!\,(n-m)!} t^{n-m} R_m(y)\,. \tag{171}$$

Therefore

$$(R|R)_{mn}(t) = \begin{cases} 0, & m > n \\ \frac{n!}{m!(n-m)!} t^{n-m}, & m \leqslant n \end{cases}. \tag{172}$$

The matrix of the R|R-operator is then

$$(\mathbf{R}|\mathbf{R})(t) = (R|R)_{mn}(t) = \begin{pmatrix} 1 & t & t^2 & t^3 & ... \\ 0 & 1 & 2t & 3t^2 & ... \\ 0 & 0 & 1 & 3t & ... \\ 0 & 0 & 0 & 1 & ... \\ ... & ... & ... & ... & ... \end{pmatrix}. \tag{173}$$

2. *S|S-operator*

Similarly, we have for the S|S-operator, with the basis (137):

$$\begin{aligned} S_n(y+t) &= (y+t)^{-n-1} = y^{-n-1}\left[1+\frac{t}{y}\right]^{-n-1} = \sum_{m=0}^{\infty} \frac{(-1)^m (m+n)!}{m!n!} t^m y^{-n-m-1} \\ &= \sum_{m=0}^{\infty} \frac{(-1)^m (m+n)!}{m!n!} t^m S_{n+m}(y) = \sum_{m=n}^{\infty} \frac{(-1)^{m-n}\, m!}{n!\,(m-n)!} t^{m-n} S_m(y)\,, \end{aligned} \tag{174}$$

Therefore,

$$(S|S)_{mn}(t) = \begin{cases} 0, & m < n \\ \frac{(-1)^{m-n} m!}{n!(m-n)!} t^{m-n}, & m \geqslant n. \end{cases}. \tag{175}$$

This yields the following matrix form:

$$(\mathbf{S}|\mathbf{S})\,(t) = (S|S)_{mn}\,(t) = \begin{pmatrix} 1 & 0 & 0 & 0 & ... \\ -t & 1 & 0 & 0 & ... \\ t^2 & -2t & 1 & 0 & ... \\ -t^3 & 3t^2 & -3t & 1 & ... \\ ... & ... & ... & ... & ... \end{pmatrix}. \tag{176}$$

3. S|R-operator

For the S|R-translation of the basis (137) we have

$$S_n\,(y{+}t) \;=\; (t+y)^{-n-1} = t^{-n-1}\left[1+\frac{y}{t}\right]^{-n-1} = \sum_{m=0}^{\infty}\frac{(-1)^m\,(m+n)!}{m!n!}t^{-n-m-1}y^m \tag{177}$$
$$= \sum_{m=0}^{\infty}\frac{(-1)^m\,(m+n)!}{m!n!}t^{-n-m-1}R_m(y).$$

So

$$(S|R)_{mn}(t) = \frac{(-1)^m\,(m+n)!}{m!n!t^{n+m+1}}, \tag{178}$$
$$(\mathbf{S}|\mathbf{R})(t) = \begin{pmatrix} t^{-1} & t^{-2} & t^{-3} & ... \\ -t^{-2} & -2t^{-3} & -3t^{-4} & ... \\ t^{-3} & 3t^{-4} & 6t^{-5} & ... \\ ... & ... & ... & ... \end{pmatrix}$$

B. S-expansion Error

Consider S-expansion of function (135) in infinite series

$$\psi\,(y) \;=\; \frac{1}{y-x_i} = \sum_{m=0}^{\infty}(x_i-x_{*1})^m\,S_m\,(y{-}x_{*1}) = \sum_{m=0}^{\infty}(x_i-x_{*1})^m\,(y-x_{*1})^{-m-1} \tag{179}$$
$$= \sum_{m=0}^{p-1}(x_i-x_{*1})^m\,(y-x_{*1})^{-m-1} + \sum_{m=p}^{\infty}(x_i-x_{*1})^m\,(y-x_{*1})^{-m-1}$$
$$= \sum_{m=0}^{p-1}(x_i-x_{*1})^m\,(y-x_{*1})^{-m-1} + \frac{1}{y-x_i}\left(\frac{x_i-x_{*1}}{y-x_{*1}}\right)^p.$$

Thus, the error of representation of this function with the first p terms of its series is

$$S_{error} = \left| \frac{1}{y - x_i} \left(\frac{x_i - x_{*1}}{y - x_{*1}} \right)^p \right| \tag{180}$$

Assume

$$|x_i - x_{*1}| \leqslant r, \quad |y - x_{*1}| \geqslant R, \tag{181}$$

then the error of the S-expansion of a single source is

$$S_{error} \leqslant \left(\frac{r}{R} \right)^p \frac{1}{R - r}. \tag{182}$$

since

$$R \leqslant |y - x_{*1}| = |y - x_i + x_i - x_{*1}| \leqslant |y - x_i| + |x_i - x_{*1}| \leqslant |y - x_i| + r, \tag{183}$$

so

$$|y - x_i| \geqslant R - r. \tag{184}$$

C. Translation Errors

For exact translation of the coefficients representing the function one should multiply the vector of coefficients by an infinite matrix. If the matrix is truncated by the size of the vector, the matrix translation operators introduce a translation error. In the example considered, however, the R|R and S|S translation operators introduce zero additional error, and the error due to the translation comes only from truncation of the S|R-operator. Below we show this fact and evaluate the error bounds for the S|R-translation.

1. R|R-translation error

Consider a function represented by p terms of its regular expansion near the center x_{*1} :

$$\psi(y) = \sum_{n=0}^{p-1} A_n R_n (y - x_{*1}), \quad R_n (y - x_{*1}) = (y - x_{*1})^n . \tag{185}$$

The function $\psi(y)$ can be also be represented near a new center of expansion x_{*2} as

$$\psi(y) = \sum_{m=0}^{\infty} \widehat{A}_m R_m (y - x_{*2}), \quad \widehat{A}_m = \sum_{n=0}^{p-1} (R|R)_{mn} (x_{*2} - x_{*1}) A_n. \tag{186}$$

Consider the difference between this function and function by the following truncated series near center x_{*2}:

$$\psi^p(y) = \sum_{m=0}^{p-1} \widehat{A}_m^p R_m\left(y - x_{*2}\right), \quad \widehat{A}_m^p = \sum_{n=0}^{p-1} (R|R)_{mn}\left(x_{*2} - x_{*1}\right) A_n, \quad m = 0, ..., p-1, \tag{187}$$

where the coefficients $\widehat{A}_m^p$ are obtained by multiplication of the vector of coefficients by the truncated translation matrix.

Then, using the fact that the R|R-translation matrix is triangular, $(R|R)_{mn} = 0$ for $m > n$, we have

$$\psi(y) - \psi^p(y) = \sum_{m=0}^{\infty} \widehat{A}_m R_m\left(y - x_{*2}\right) - \sum_{n=0}^{p-1} \widehat{A}_m^p R_m\left(y - x_{*2}\right) \tag{188}$$

$$= \sum_{m=0}^{\infty}\sum_{n=0}^{p-1} (R|R)_{mn}\left(x_{*2} - x_{*1}\right) A_n R_m\left(y - x_{*2}\right) - \sum_{m=0}^{p-1}\sum_{n=0}^{p-1} (R|R)_{mn}\left(x_{*2} - x_{*1}\right) A_n R_m\left(y - x_{*2}\right)$$

$$= \sum_{n=0}^{p-1}\sum_{m=0}^{n} (R|R)_{mn}\left(x_{*2} - x_{*1}\right) A_n R_m\left(y - x_{*2}\right) - \sum_{n=0}^{p-1}\sum_{m=0}^{n} (R|R)_{mn}\left(x_{*2} - x_{*1}\right) A_n R_m\left(y - x_{*2}\right)$$

$$= 0. \tag{189}$$

So the R|R-translation with the truncated matrix is exact. Indeed, since the function (185) is a polynomial of degree $p-1$, and the function (187) is also a polynomial of the same degree, the R|R-translation matrix relates the coefficients of these polynomials exactly, and so the R|R-operator does not introduce any additional error to the error of representation of function $\psi(y)$ by the finite series.

2. *S|S-translation error*

From Eqs. (169) and (175) we have for the S|S-translation

$$\widehat{A}_m = \sum_{n=0}^{m} (S|S)_{mn}(t) A_n, \quad m = 0, 1, ... \tag{190}$$

This shows that the first p coefficients $\widehat{A}_m$, $m = 0, 1, ..., p-1$ are exactly determined by the first p coefficients A_n, $n = 0, 1, ..., p-1$ by applying to them the $p \times p$ truncated S|S-matrix (which is a consequence of the fact that in our case the S|S-translation matrix is lower triangular). If the series

$$\psi(y) = \sum_{m=0}^{\infty} \widehat{A}_m S_m\left(y - x_{*2}\right) \tag{191}$$

is truncated by p terms after translation, then we need only first p coefficients $\widehat{A}_m$, which are computed exactly. The error of truncation of the series is then equal to the error of performing the S expansion, and so the truncated S$|$S-operator does not introduce any additional error.

3. S$|$R-translation error

Consider the representation of the function (135) via infinite series

$$\begin{aligned}\psi(y) &= \frac{1}{y-x_i} = \sum_{m=0}^{\infty} (x_i - x_{*1})^m S_m(y-x_{*1}) \\ &= \sum_{m=0}^{\infty}\sum_{n=0}^{\infty} (x_i - x_{*1})^m (S|R)_{nm}(x_{*2}-x_{*1}) R_n(y-x_{*2}) = \sum_{m=0}^{\infty}\sum_{n=0}^{\infty} c_{mn},\end{aligned} \tag{192}$$

where from Eq. (178) we have

$$c_{mn} = \frac{(-1)^m (m+n)!}{m!n!(x_{*2}-x_{*1})^{n+m+1}} (x_i - x_{*1})^m (y - x_{*2})^n . \tag{193}$$

If the series are truncated by the first p-terms and the $p\times p$ matrix is used for translation, then the function approximating $\psi(y)$ has the form

$$\psi^p(y) = \sum_{m=0}^{p-1}\sum_{n=0}^{p-1} c_{mn}. \tag{194}$$

Thus, the combined expansion/translation error of approximation of a single source can be evaluated as

$$\begin{aligned}(S|R)_{error} &= |\psi(y) - \psi^p(y)| = \left|\sum_{m=0}^{\infty}\sum_{n=0}^{\infty} c_{mn} - \sum_{m=0}^{p-1}\sum_{n=0}^{p-1} c_{mn}\right| \\ &= \left|\sum_{m=0}^{p-1}\sum_{n=0}^{p-1} c_{mn} + \sum_{m=0}^{p-1}\sum_{n=p}^{\infty} c_{mn} + \sum_{m=p}^{\infty}\sum_{n=0}^{\infty} c_{mn} - \sum_{m=0}^{p-1}\sum_{n=0}^{p-1} c_{mn}\right| \\ &= \left|\sum_{m=0}^{p-1}\sum_{n=p}^{\infty} c_{mn} + \sum_{m=p}^{\infty}\sum_{n=0}^{\infty} c_{mn}\right| \leqslant \sum_{m=0}^{p-1}\sum_{n=p}^{\infty} |c_{mn}| + \sum_{m=p}^{\infty}\sum_{n=0}^{\infty} |c_{mn}| \\ &\leqslant \sum_{m=0}^{\infty}\sum_{n=p}^{\infty} |c_{mn}| + \sum_{m=p}^{\infty}\sum_{n=0}^{\infty} |c_{mn}| = \sum_{m=p}^{\infty}\sum_{n=0}^{\infty} (|c_{nm}| + |c_{mn}|) .\end{aligned} \tag{195}$$

Denote

$$\theta_1 = \frac{\max|x_i - x_{*1}|}{|x_{*1} - x_{*2}|}, \quad \theta_2 = \frac{\max|y - x_{*2}|}{|x_{*1} - x_{*2}|}, \quad t = x_{*1} - x_{*2}. \tag{196}$$

Then using the following series

$$\frac{1}{(1-\alpha)^{n+1}} = 1+(n+1)\alpha+\frac{(n+1)(n+2)}{2!}\alpha^2+\ldots = \sum_{m=0}^{\infty}\frac{(m+n)!}{m!n!}\alpha^m, \quad |\alpha|<1, \tag{197}$$

and Eq. (193), we can sum up the series in Eq. (195) as follows

$$\begin{aligned}
(S|R)_{error} &\leqslant \frac{1}{|t|}\sum_{m=p}^{\infty}\sum_{n=0}^{\infty}\frac{(m+n)!}{m!n!}\left[\frac{|x_i-x_{*1}|^n|y-x_{*2}|^m+|x_i-x_{*1}|^m|y-x_{*2}|^n}{|t|^{m+n}}\right] \\
&\leqslant \frac{1}{|t|}\sum_{m=p}^{\infty}\sum_{n=0}^{\infty}\frac{(m+n)!}{m!n!}\left(\theta_1^n\theta_2^m+\theta_1^m\theta_2^n\right) \\
&= \frac{1}{|t|}\sum_{m=p}^{\infty}\left\{\sum_{n=0}^{\infty}\frac{(m+n)!}{m!n!}\theta_1^n\theta_2^m+\sum_{n=0}^{\infty}\frac{(m+n)!}{m!n!}\theta_1^m\theta_2^n\right\} \\
&= \frac{1}{|t|}\sum_{m=p}^{\infty}\left\{\theta_2^m\sum_{n=0}^{\infty}\frac{(m+n)!}{m!n!}\theta_1^n+\theta_1^m\sum_{n=0}^{\infty}\frac{(m+n)!}{m!n!}\theta_2^n\right\} \\
&= \frac{1}{|t|}\sum_{m=p}^{\infty}\left[\frac{\theta_2^m}{(1-\theta_1)^{m+1}}+\frac{\theta_1^m}{(1-\theta_2)^{m+1}}\right] \\
&= \frac{1}{|t|}\left\{\frac{1}{1-\theta_1}\sum_{m=p}^{\infty}\frac{\theta_2^m}{(1-\theta_1)^m}+\frac{1}{1-\theta_2}\sum_{m=p}^{\infty}\frac{\theta_1^m}{(1-\theta_2)^m}\right\} \\
&= \frac{1}{|t|}\left\{\frac{1}{1-\theta_1}\frac{\theta_2^p}{(1-\theta_1)^p}\frac{1}{1-\frac{\theta_2}{1-\theta_1}}+\frac{1}{1-\theta_2}\frac{\theta_1^p}{(1-\theta_2)^p}\frac{1}{1-\frac{\theta_1}{1-\theta_2}}\right\} \\
&= \frac{1}{1-\theta_1-\theta_2}\frac{1}{|t|}\left[\frac{\theta_2^p}{(1-\theta_1)^p}+\frac{\theta_1^p}{(1-\theta_2)^p}\right].
\end{aligned} \tag{198}$$

VIII. APPENDIX B

Below we provide some error bounds for the MLFMM that are used in the main text.

A. S-expansion Error

If we are using k-neighborhoods and the size of the box at the finest level is $2^{-l_{\max}}$, then for the function (135), the S-expansion error of a single source is bounded according Eq. (**??**), where

$$R = 2^{-l_{\max}}\left(\frac{1}{2}+k\right), \quad r = 2^{-l_{\max}}\frac{1}{2}. \tag{199}$$

so that

$$S_{error} \leqslant \frac{2^{l_{\max}}}{k(2k+1)^p}. \tag{200}$$

For a given level l, $l \leqslant l_{\max}$ we have the expansion error

$$S_{error}^{(l)} = \frac{2^l}{k\left(2k+1\right)^p} \leqslant \frac{2^{l_{\max}}}{k\left(2k+1\right)^p} \leqslant S_{error}. \tag{201}$$

B. S|R-translation Error

Consider now the $(S|R)_{error}$ when using k-neighborhoods and m-reduced scheme of translation. At the finest level we have for a single translation

$$\begin{aligned} |x_i - y| &\geqslant 2^{-l_{\max}} k, \quad |x_i - x_{*1}| \leqslant 2^{-l_{\max}} \frac{1}{2} 2^m, \quad |y - x_{*2}| \leqslant 2^{-l_{\max}} \frac{1}{2}, \\ |t| &= |x_{*1} - x_{*2}| \geqslant 2^{-l_{\max}} \left[\frac{1}{2}\left(2^m+1\right) + k\right]. \end{aligned} \tag{202}$$

The absolute total error of the MLFMM is bounded by expansion and translation errors of all sources, so

$$\begin{aligned} AbsTotalError &= N \cdot AbsSingleSourceError \\ &= N\left[(S|R)_{error} + (S)_{error}\right] \\ &\leqslant N\left[2 \cdot 2^{l_{\max}} \frac{2^m+k}{k}\left(\frac{2^m}{2^m+2k}\right)^p + \frac{2^{l_{\max}}}{k\left(2k+1\right)^p}\right] \\ &= \frac{1}{k} N \cdot 2^{l_{\max}}\left[2\left(2^m+k\right)\left(\frac{2^m}{2^m+2k}\right)^p + \frac{1}{\left(2k+1\right)^p}\right] \\ &\leqslant \frac{1}{k} N \cdot 2^{l_{\max}}\left[2\left(2^m+k\right)\left(\frac{2^m}{2^m+2k}\right)^p + \left(\frac{2^m}{2^m+2k}\right)^p\right] \\ &= \frac{1}{k} N \cdot 2^{l_{\max}}\left(\frac{2^m}{2^m+2k}\right)^p\left[2\left(2^m+k\right)+1\right] \\ &= N \cdot 2^{l_{\max}} \frac{2\left(2^m+k\right)+1}{k}\left(\frac{2^m}{2^m+2k}\right)^p \leqslant N \cdot 2^{l_{\max}} \frac{2\left(k+k\right)+1}{k}\left(\frac{2^m}{2^m+2k}\right)^p \\ &\leqslant N \cdot 2^{l_{\max}} \frac{2\left(k+k\right)+k}{k}\left(\frac{2^m}{2^m+2k}\right)^p = 5N \cdot 2^{l_{\max}}\left(\frac{2^m}{2^m+2k}\right)^p. \end{aligned} \tag{203}$$

Here we used the fact that

$$k \geqslant 2^m \geqslant 1. \tag{204}$$

C. MLFMM Error in Asymptotic Model

The error bounds for the 1-D example can be used for analysis of multidimensional cases where the functions $\psi_i(\mathbf{y})$ singular at $\mathbf{y} = \mathbf{x}_i$ can be expanded in power series with respect to $|\mathbf{y} - \mathbf{x}_i|$

(e.g. the estimations of Appendix A hold also for 2-D case, when $\mathbf{y}$ and $\mathbf{x}_i$ are treated as complex numbers). The 1-D example shows that there are three power functions (one for the S-expansion error and two for the S|R-translation error) which specify the overall error. If we specify

$$\sigma_0 = \frac{\min|\mathbf{y}-\mathbf{x}_{*1}|}{\max|\mathbf{x}_i-\mathbf{x}_{*1}|}, \quad \theta_1 = \frac{\max|\mathbf{x}_i-\mathbf{x}_{*1}|}{|\mathbf{x}_{*1}-\mathbf{x}_{*2}|}, \quad \theta_2 = \frac{\max|\mathbf{y}-\mathbf{x}_{*2}|}{|\mathbf{x}_{*1}-\mathbf{x}_{*2}|}, \tag{205}$$

then the three errors are of type

$$\epsilon_0 = C_0\sigma_0^{-p}, \quad \epsilon_1 = C_1\sigma_1^{-p}, \quad \epsilon_2 = C_2\sigma_2^{-p}, \tag{206}$$

where ϵ_0 represents S-expansion error, while ϵ_1 and ϵ_2 represent S|R-translation error and related to θ_1 and θ_2 according Eq. (198) as

$$\sigma_1 = \frac{1-\theta_1}{\theta_2}, \quad \sigma_2 = \frac{1-\theta_2}{\theta_1}. \tag{207}$$

Since the total error can be evaluated as

$$AbsTotalError = N \cdot AbsSingleSourceError = N\left(\epsilon_0+\epsilon_1+\epsilon_2\right), \tag{208}$$

it is bounded as

$$AbsTotalError \leqslant 3N\max(\epsilon_0,\epsilon_1,\epsilon_2). \tag{209}$$

Since ϵ_0, ϵ_1 and ϵ_2 are exponential functions, we can also estimate this as

$$AbsTotalError \leqslant NC\sigma^{-p}, \quad \max(\epsilon_0,\epsilon_1,\epsilon_2) \tag{210}$$

where

$$\sigma = \min\left(\sigma_0,\sigma_1,\sigma_2\right), \tag{211}$$

and C does not depend on p (because we select the slowest decaying function).

Let us compare first σ_2 and σ_0. We have

$$\begin{aligned}\sigma_0-\sigma_2 &= \frac{\min|\mathbf{y}-\mathbf{x}_{*1}|}{\max|\mathbf{x}_i-\mathbf{x}_{*1}|} - \frac{\min|\mathbf{x}_{*1}-\mathbf{x}_{*2}|-\max|\mathbf{y}-\mathbf{x}_{*2}|}{\max|\mathbf{x}_i-\mathbf{x}_{*1}|} &(212)\\ &= \frac{\min|\mathbf{y}-\mathbf{x}_{*1}|+\max|\mathbf{y}-\mathbf{x}_{*2}|-\min|\mathbf{x}_{*1}-\mathbf{x}_{*2}|}{\max|\mathbf{x}_i-\mathbf{x}_{*1}|} \geqslant 0. &(213)\end{aligned}$$

Indeed due to the triangle inequality, we have

$$|\mathbf{x}_{*1}-\mathbf{x}_{*2}| = |\mathbf{y}-\mathbf{x}_{*1}-(\mathbf{y}-\mathbf{x}_{*2})| \leqslant |\mathbf{y}-\mathbf{x}_{*1}|+|\mathbf{y}-\mathbf{x}_{*2}|, \tag{214}$$

and

$$\min|\mathbf{x}_{*1}-\mathbf{x}_{*2}| \leqslant \min\left(|\mathbf{y}-\mathbf{x}_{*1}|+|\mathbf{y}-\mathbf{x}_{*2}|\right) \leqslant \min|\mathbf{y}-\mathbf{x}_{*1}|+\max|\mathbf{y}-\mathbf{x}_{*2}|\,. \qquad (215)$$

So $\sigma=\min(\sigma_1,\sigma_2)$ and the error is determined not by the expansion error, but by the translation error. Then we compare σ_2 and σ_1 :

$$\sigma_1-\sigma_2=\frac{1-\theta_1}{\theta_2}-\frac{1-\theta_2}{\theta_1}=\frac{(\theta_1-\theta_2)-\left(\theta_1^2-\theta_2^2\right)}{\theta_1\theta_2}=\frac{(\theta_1-\theta_2)\left(1-\theta_1-\theta_2\right)}{\theta_1\theta_2}. \qquad (216)$$

By definition of the regions of validity of the expansions, the multiplier $1-\theta_1-\theta_2$ is always positive. Indeed, using Eq. (14) with any $R_c>1$ we can see that

$$1-\theta_1-\theta_2=\frac{|\mathbf{x}_{*1}-\mathbf{x}_{*2}|-\max\left(|\mathbf{x}_i-\mathbf{x}_{*1}|+|\mathbf{y}-\mathbf{x}_{*2}|\right)}{|\mathbf{x}_{*1}-\mathbf{x}_{*2}|}>0. \qquad (217)$$

Then we also can see that $\theta_1-\theta_2\geqslant 0$. Indeed,

$$\theta_1-\theta_2=\frac{\max|\mathbf{x}_i-\mathbf{x}_{*1}|-\max|\mathbf{y}-\mathbf{x}_{*2}|}{|\mathbf{x}_{*1}-\mathbf{x}_{*2}|}\geqslant 0, \qquad (218)$$

because the size of the box from which the S|R-translation is performed is the same as the size of the box to which S|R-translation is performed (if the boxes are at the same level, $m=0$) or larger (for $m>0$, when we use the reduced scheme of translation from boxes of the parent or coarser level).

Therefore we see that

$$\sigma=\sigma_2=\frac{1-\theta_2}{\theta_1}=\frac{|\mathbf{x}_{*2}-\mathbf{x}_{*1}|-\max|\mathbf{y}-\mathbf{x}_{*2}|}{\max|\mathbf{x}_i-\mathbf{x}_{*1}|}=\frac{\min\left(|\mathbf{x}_{*2}-\mathbf{x}_{*1}|-|\mathbf{y}-\mathbf{x}_{*2}|\right)}{\max|\mathbf{x}_i-\mathbf{x}_{*1}|}. \qquad (219)$$