Implementing Fast Multipole Methods with High-Level Interpreted Languages

Srinath Kailasa

A thesis submitted in partial fulfillment of the requirements
for the degree Master of Science

Department of Physics
University College London
August 19, 2020

Declaration

I, Srinath Kailasa, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Abstract

The Fast Multipole Method (FMM) is a numerical method to accelerate the solution of the N-Body problem, which appears in numerous contexts in science and engineering, for example in solving for gravitational or electrostatic potentials in multi-particle systems. It does so by approximating the Green's function of the system with analytic infinite series expansions (the origin of the 'multipole' in its name), and coalescing the effect of distinct distant sources together so as to greatly reduce the number of computations. The analytic expansions of the original Fast Multipole Method depend on the Green's function of the system in question (Helmholtz, Laplace etc.), and in practice a new implementation must be written for a given system.

The Kernel-Independent Fast Multipole Method (KIFMM), first presented by Ying et al. [8], is a similar approach that replaces the analytic series expansions with a continuous distribution of so called 'equivalent density' supported at discrete points on a box enclosing a set of particles. These equivalent densities are found by matching the potential they generate to those generated by the original sources at another surface in the far field. Usefully, this approach doesn't require multipole expansions of the Green's functions of a system, and therefore can be programmed in an agnostic way, hence the origin of it's name. The KIFMM is compatible with a broad class of elliptic Green's functions, and is therefore extremely useful for studying a broad range of problems with a single software implementation.

This thesis presents a well tested and extensible Python implementation of the KIFMM, with investigations made into both mathematical and computational techniques for the acceleration of the software, using the N-Body electrostatic problem as model on which to test the implementation. Python is chosen as it has emerged as a standard for scientific and data intensive computing in recent years, with a huge increase in adoption, and a well supported ecosystem of libraries and tools available for accelerating numerical codes. The wider context of this thesis is an ongoing collaboration with the ExaFMM Project [10] to produce a Python implementation of the KIFMM that sacrifices as little performance as possible. This thesis introduces the relevant theoretical background, before proceeding to discuss the strategies used in the practical implementation of the software. Key bottlenecks in the implementation are examined and addressed, with the speed and accuracy of the software benchmarked with the Laplace kernel for electrostatic problems. Finally, future avenues of investigation and software development are discussed, in the context of the wider literature.

Contents

Introduction

1.1 Overview of the Analytic FMM

1.1.1 Motivation

The paradigmatic problem of Fast Multipole Methods (**FMM**) [1] is the so called N-Body problem. This classic problem refers to the calculation of the pairwise interactions between N particles over a potentially long-range, for example in gravitational or electrostatic problems. The straightforward calculation can be written in the form of the following sum,

$$\Phi(x_j) = \sum_{i=1}^{N} w_i K(x_i, x_j) \tag{1.1}$$

Where $i, j \in [1, N]$ and $K(x, y)$ is called the Green's function, or equivalently a 'kernel function', where one is generally concerned with coordinates of particles in an $n = 2$ or 3 dimensional Hilbert space taking $x_i \in \mathbb{R}^n$. Additionally, each summand is weighted by w_i. For solving for electrostatic potential in three dimensions, which is used as the model problem throughout this thesis, this goes to,

$$\Phi(x_j) = \sum_{i=1}^{N} q_i K(x_i, x_j) \tag{1.2}$$

where q_i refers to a charge density with the kernel function,

$$K(x, y) = \frac{1}{4\pi\epsilon_0} \frac{1}{|x - y|} \tag{1.3}$$

the constant ϵ_0 is the permittivity of free space. It's easy to see how a naive direct application of this equation over N particles results in an algorithm of $O(N^2)$ complexity, therefore it's only practicable for systems of moderate size, whereas in realistic systems, one may be interested in interactions involving 10^6 to 10^8 particles.

This chapter introduces the analytic FMM, the kernel-independent version, which is the main focus of this thesis, is presented later. Though substantially different in implementation, the analytic FMM will provide the opportunity to exposit many of the key ideas behind all FMM-based algorithms, and provides a good starting point for understanding and developing upon these algorithms. First presented by Greengard [4], the analytic FMM represented a sea change for N-Body simulation. By trading off computations for error, it manages to achieve an asymptotic complexity of just $O(N)$. Additionally, it comes equipped with rigorous error bounds, making

[1] The first usage of a technical term or abbreviation listed in the glossary is highlighted throughout the text for ease of reference.

fast and accurate massive N-Body simulations feasible on available computing hardware. It's success has been such that it is regarded as one of the key developments in numerical algorithms in the twentieth century [2].

The original analytic FMM solves the electrostatic problem in two and three dimensions, this is equivalently known as the Poisson problem, represented by the differential equation,

$$\nabla^2 \phi = f \tag{1.4}$$

Where ϕ is some scalar potential to be determined, and f is a scalar source term which is usually known. For electrostatics the corresponding formulation can be derived from Gauss' law as [6],

$$\nabla^2 \phi = -\frac{q}{\epsilon_0} \tag{1.5}$$

where ϕ is the electrostatic potential, q is the charge density and ϵ_0 is the permittivity of free space. It can therefore be seen that the FMM is actually solving the Poisson problem by reformulating it as an integral equation. The ubiquity of problems of the form (1.1) in computational science has lead to diverse application of the FMM. For example, in the modeling the electrostatic interactions of charged particles in complex biological molecules at biologically relevant length scales [1]. The extension of FMMs to Helmholtz equations [11], has lead to even more applications, such as in seismic and acoustic scattering [7]. Though as the focus of this thesis is on solving the Poisson model problem for electrostatics, this is mentioned only for completeness.

The key insight that leads to the FMM's asymptotic complexity is the idea that if the field created by a distribution of charge (or mass) density is approximated to be relatively smooth in the **far field**, then it should be possible to apply some form of compression for the evaluation of contribution to local potentials due to particles in the far field. The FMM performs this compression by encoding the field contributions of particles in the far field using a multipole expansion.

For simple kernel functions and charge distributions, such as the model problem of this thesis, one can easily derive the expression for this multipole expansion by finding an series expansion of the system's Green's function. In order to generalise the discussion, an arbitrary continuous distribution of charge is considered as shown in figure 1.1, for which the potential is evaluated at some other evaluation point outside of the distribution. This can be written as follows [6],

$$\Phi(\mathbf{r}) = \frac{1}{4\pi\epsilon_0} \int \frac{1}{d} \rho(\mathbf{r}') d\tau' \tag{1.6}$$

where $\rho(\mathbf{r}')$ is a charge density, and the other symbols take their meanings from figure (1.1).

From law of the cosines,

$$d^2 = r^2 + (r')^2 - 2rr' \cos\alpha = r^2 \left[1 + \left(\frac{r'}{r} \right)^2 - 2 \left(\frac{r'}{r} \right) \cos\alpha \right] \tag{1.7}$$

$$d = r\sqrt{1+\epsilon} \tag{1.8}$$

Where,

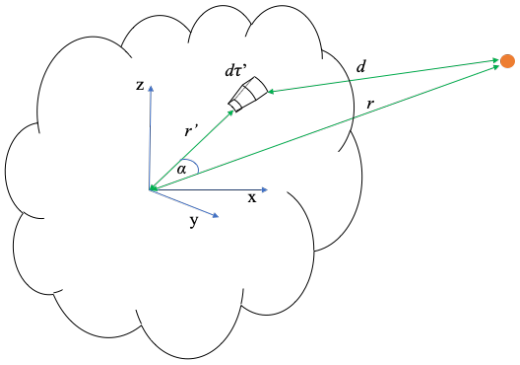

Figure 1.1: An arbitrary charge distribution, with an orange point to mark a point where the potential is being evaluated. Here, $\mathbf{r}$ is the the vector between the centre of the multipole expansion and the evaluation point, $\mathbf{r}'$ is the vector between the centre of expansion and a given volume element $d\tau'$, and d is a vector between the volume element $d\tau'$ and the evaluation point.

$$\epsilon \equiv \left(\frac{r'}{r}\right)\left(\frac{r'}{r} - 2\cos\alpha\right) \tag{1.9}$$

As ϵ is small far away from charge distribution one can expand $1/d$ binomially,

$$\frac{1}{d} = \frac{1}{r}(1+\epsilon)^{-1/2} = \frac{1}{r}\left(1 - \frac{1}{2}\epsilon + \frac{3}{8}\epsilon^2 - ...\right) \tag{1.10}$$

$$\frac{1}{d} = \frac{1}{r}\sum_{n=0}^{\infty}\left(\frac{r'}{r}\right)^n P_n(\cos\alpha) \tag{1.11}$$

Using this, the exact multipole expansion for this charge distribution is,

$$\Phi(\mathbf{r}) = \frac{1}{4\pi\epsilon_0}\sum_{n=0}^{\infty}\frac{1}{r^{n+1}}\int (r')^n P_n(\cos\alpha)\rho(\mathbf{r}')d\tau' \tag{1.12}$$

If instead one considers a charge distribution composed of N discrete charges q_i this goes to,

$$\Phi(\mathbf{r}) = \frac{1}{4\pi\epsilon_0}\sum_{i=1}^{N}\sum_{n=0}^{\infty}\frac{(r')^n q_i}{r^{n+1}}P_n(\cos\alpha) \tag{1.13}$$

Using the addition theorem for Legendre polynomials [4],

$$P_n(\cos\gamma) = \sum_{m=-n}^{n} Y_n^{-m}(\alpha,\beta)Y_n^m(\theta,\phi) \tag{1.14}$$

Where the Legendre polynomial is written in terms of spherical harmonics, where (r,θ,ϕ) and (ρ,α,β) define two spherical coordinates, and γ is the angle subtended between them. Therefore, the multipole expansion goes to,

$$\Phi(\mathbf{r}) = \frac{1}{4\pi\epsilon_0}\sum_{i=1}^{N}\sum_{n=0}^{\infty}\frac{(r')^n q_i}{r^{n+1}}P_n(\cos\alpha) \tag{1.15}$$

$$= \frac{1}{4\pi\epsilon_0}\sum_{i=1}^{N}\sum_{n=0}^{\infty}\sum_{m=-n}^{n}\frac{(r')^n q_i Y_n^{-m}(\alpha_i,\beta_i)}{r^{n+1}}Y_n^m(\theta,\phi) \tag{1.16}$$

$$= \sum_{i=1}^{N}\sum_{n=0}^{\infty}\sum_{m=-n}^{n}\frac{M_n^m}{r^{n+1}}\cdot Y_n^m(\theta,\phi) \tag{1.17}$$

This is an exact expansion, and it converges for $\frac{r'}{r} < 1$. This convergence condition means that estimating of the potential at a given evaluation point calculated using the multipole expansion is only possible in the far-field, the boundary of which is often tuned empirically for different systems as it's user defined. If instead the expansion is taken with centre at the evaluation point, one can rewrite as the multipole expansion as a 'local' expansion,

$$\Phi(\mathbf{r}) = \frac{1}{4\pi\epsilon_0}\sum_{i=1}^{N}\sum_{n=0}^{\infty}\sum_{m=-n}^{n}\frac{(r)^n q_i Y_n^{-m}(\alpha_i,\beta_i)}{r'^{n+1}}Y_n^m(\theta,\phi) \tag{1.18}$$

$$= \frac{1}{4\pi\epsilon_0}\sum_{i=1}^{N}\sum_{n=0}^{\infty}\sum_{m=-j}^{n}L_m^n\cdot Y_m^n(\theta,\phi)\cdot r^j \tag{1.19}$$

which converges when $\frac{r}{r'} < 1$. The region of convergence for both types of expansions are shown in figure (1.2).

The key point to note is that the multipole and local expansions are exact, and can be truncated as required to ensure that the asymptotic complexity of evaluating a multipole or local expansion at an evaluation point is bounded by $O(N)$. A rigorous error analysis of the analytic FMM is outside the scope of this thesis, and we defer to the literature for exact expressions for these truncation errors [4], Furthermore, exact operations exist for shifting the center of these expansions, as well as translating multipole expansion coefficients into equivalent local expansion coefficients, which are crucial in providing the improvements to asymptotic complexity[2].

1.1.2 Algorithm Structure & Analysis

The convergence condition of the multipole expansion prohibits the compression of charges from particles in the **near field**. Therefore the FMM makes use of a tree structure in a recursive algorithm, this structure is known as an Octree in three dimensions and a Quadtree in two dimensions[3]. This structure hierarchically partitions space such that each level, l, of the tree is equally partioned into $(2^d)^l$ boxes[4] over the domain of the tree, where d is the dimension, i.e. $d = 3$ in three dimensions. If one were to simply traverse the tree from the coarsest, or 'root', level

[2]Expressions for these shift operators for the three dimensional Laplace kernel considered as our model problem are provided in Appendix A.2.

[3]The usage of trees naturally leads to biological adjectives to describe it's structure, for example the coarsest level of a tree is referred to as the 'root' level, and the finest level as the 'leaf' level.

[4]Regardless of the spatial dimension, partitions of a domain are invariably referred to as boxes.

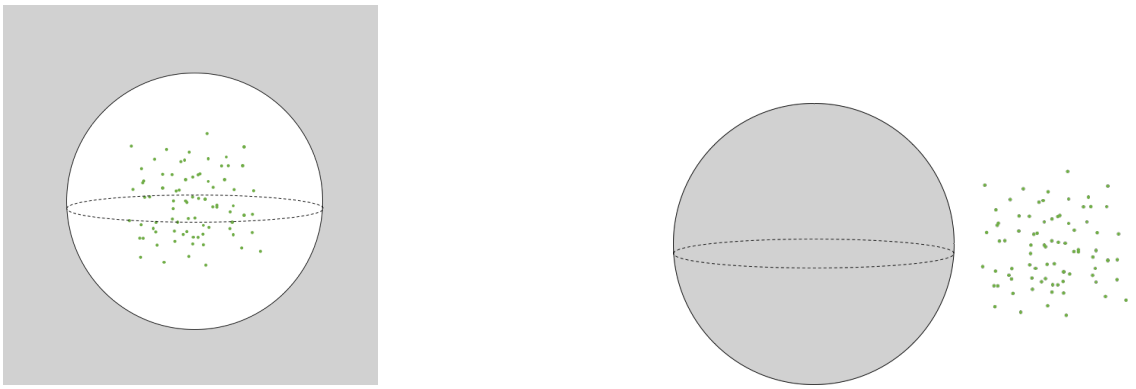

Figure 1.2: (A) A multipole expansion centered on charge distribution. (B) A local expansion, centered around a point of evaluation. Regions in which the expansions converge are shaded in grey. For the multipole expansion this is the entire domain outside of the region for $r > r'$, and for the local expansion this is the region for which $r < r'$

to the finest, or 'leaf', level and find the multipole expansion of source particles in each box of each level, one could then evaluate these multipole expansions at each particle to solve the N-Body problem. As there are $O(\log(N))$ boxes in the tree, and N particles this results in a $O(N \log(N))$ asymptotic complexity.

However the FMM reduces computational complexity further by making use of local expansions. Using the exact expressions to shift the multipole expansion coefficients[5] M_n^m to local expansion coefficients L_n^m one can approximate the the interaction between two boxes in the tree. Roughly speaking, because of the hierarchical nature of the tree, each box only needs to consider the interaction with a constant number of neighbouring boxes. Because the number of boxes $O(N)$, the FMM is bound by an $O(N)$ asymptotic complexity [7].

Using the above analysis, one can then describe the FMM algorithm in terms of two basic steps,

1. ***Upward Pass***: The tree is traversed **post-order**. Beginning at the leaf boxes, a multipole expansion is computed for each box due to the **source particles** it contains. This is also referred to as the particle-to-multipole operation, or **P2M**. Then as one moves up the tree hierarchy, the multipole expansions of a each box's parent box is computed by shifting the expansion centers of the multipole expansion of a given child box, to the center of the parent box a, in a multipole-to-multipole operation or **M2M**, and summing together all the coefficients. Following the upward pass, one obtains the multipole expansion for each box containing source particles at all levels of the hierarchical tree.

2. ***Downward Pass***: The tree is now traversed in **pre-order**, and the local expansion of each box is computed. This local expansion is the sum of two parts: (1) the local expansion of the parent box of a given box, if it exists, which is a compression of the potential due to boxes non-adjacent to a given box's parent. The parent box's local expansion centre is shifted to the center of the child box, this is also known as the local-to-local operation or **L2L**. (2) The multipole expansion of boxes which are the children of the **near neighbours**

[5]Expressions for these shift operators for the three dimensional Laplace kernel considered as our model problem are provided in Appendix A.2.

of a given box's parent but are not adjacent to the box itself. Such 'source' boxes are described as being in the **interaction list** of a given box. These multipole expansions for each source box in a given box's interaction list are translated into local expansions centered at the given box, this is also known as the multipole-to-local operation, or **M2L**. Notice that the M2L operation is only available from $l = 2$ of the tree, as in coarser levels the interaction list for all boxes is empty. The coefficients found from (1) and (2) are summed for each box. Steps (1) and (2) are repeated for each box until the leaf level. At this point, the local expansion of each leaf box is evaluated at all the **target particles** it contains. This local-to-particle, or **L2P**, operation encodes all the far field interaction of the target particles in this leaf box. This is then combined with a near field interaction, due to the source particles in the leaf box, as well as in the near neighbours, which are computed directly. As the tree is refined to the point where the leaf levels contain only a small constant number of particles, this final direct computation is of low cost.

With this specification, a more detailed analysis of the algorithm is possible, though we defer to [4] for a rigorous discussion. Firstly, as mentioned above, the tree must be refined such that the leaf boxes contain only a small constant number of particles, κ. The level of refinement n is therefore approximately taken to be $n \approx \log(N)$, where N is the number of source particles in the tree. Beginning with upward pass, at the leaf level each particle contributes to one multipole expansion. If this expansion is truncated to contain p multipole terms, the P2M operation has a complexity of $O(Np^2)$, which can be seen from (1.17), as well as the fact that the nature of a hierarchical tree means that are $O(N)$ boxes at the leaf level. The shift operators[6] M2L, L2L and M2M require p^4 operations with this truncation, so the computation of all of these are bounded by[7] $O(Np^4)$. Finally, evaluating the p^{th} degree local expansions at each target particle in the L2P operation, is bounded by $O(Np^2)$. The choice for level of refinement n, leads to $O(\kappa N)$ complexity for direct calculations at the leaf level. The whole algorithm is therefore bounded by $O(N)$. In practice, the number of expansion terms p is chosen for a prescribed relative error ϵ, using [8] $p = \log_c \epsilon$. The algorithm is illustrated in figure (1.3) in the two dimensional case, which is direct analogue of the three dimensional case which is the focus of this thesis, and a full pseudo-code specification is provided in Appendix A.1.

1.1.3 Summary

The main practical challenge in implementing software to solve the analytic FMM is the requirement of kernel-specific code to calculate the expansion coefficients of (1.17) and (1.19). For example, multiple different implementations already exist for Poison problems alone [5, 3]. In software engineering terms this is inconvenient for the purposes of studying the applications of the FMM in multiple different problem settings, as it leads to the requirement to implement problem-specific solvers for each kernel one may encounter. This leads to a productivity overhead in either designing a single, extensible library, which by definition will be complex to design. Or to otherwise maintain multiple problem-specific libraries.

[6] Expressions for these shift operators for the three dimensional Laplace kernel considered as our model problem are provided in Appendix A.2

[7] A more precise bound depends on the size of a given box's interaction list.

[8] There are optimal choices for c, with the authors of [8] specifying $c = \frac{4-\sqrt{3}}{\sqrt{3}}$ for three dimensional problems.

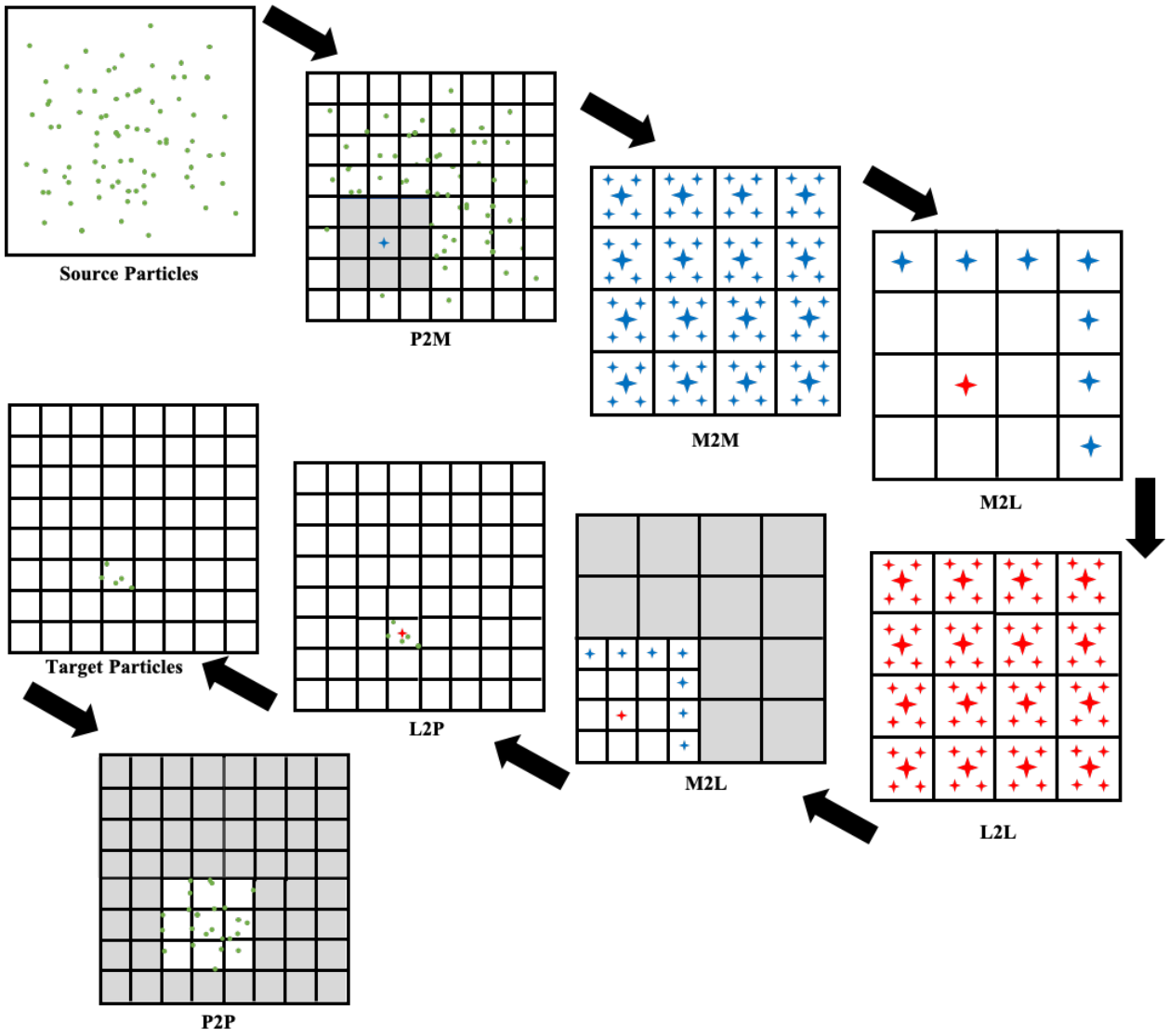

Figure 1.3: The main FMM loop in two dimensions, encapsulating the upward and downward pass. The same set of particles are used for the sources and targets, and are shown in green. Local expansions are illustrated with red stars, and multipole expansions are illustrated with blue stars. For the P2M step the grey region indicates the near field, where this multipole expansion does not converge. For the M2L and P2P steps the grey region indicates box interactions already compressed and available via the translation or direct usage of local expansions. The larger/smaller stars in the M2M and L2L steps correspond to the parent/child expansions.

The algorithm thus far described referred to as the analytic FMM is more correctly called the non-adaptive analytic FMM. The non-adaptivity refers to the assumption that all boxes at the leaf level are refined to the same degree, however as mentioned in the above analysis of the algorithm, the degree of refinement is chosen such that the number of particles in a leaf box is *constant*. Therefore, If the distribution of the particles of interest is not uniform over the computational domain, one may be needlessly refining the boxes in some regions which may even be empty. Therefore, although the asymptotic complexity of the FMM is $O(N)$, it is clear that practical implementations will suffer unless care is taken to use efficient vectorised data structures for the creation of appropriate hierarchical tree required by the algorithm.

Additionally, there is significant scope for multiple levels of parallelism in practical implementations. For example, the computations for the local and multipole coefficients as well as the application of M2L, L2L and M2M operators at a given level, are candidates for an implementation of **task-level parallelism**. In addition to parallelising of each operator application as a task, there is scope for implementing **data-level parallelism** to find the expansion coefficients. For example, Yokota and Barba [7] demonstrate how the calculation of the P2P, and M2L operators can be transferred to **GPU**s using **CUDA**. The M2L, and P2P operators represent the largest computational bottlenecks due to the number of such interactions in the FMM algorithm, therefore are a priority for acceleration. For example, in three dimensions each target box has potentially up to 189 source boxes in its interaction list with which to compute the M2L operation, and for any particularly deep tree there will be roughly $O(N)$ leaf boxes, for which interactions are calculated directly. This leads to both of these operations dominating the run-time of any FMM implementation.

In summary, the implementation of the analytical FMM is complicated by the fact that it is problem specific - which will also apply to any parallel optimisation code. This in itself provides the main motivation for developing an implementation that does not rely on explicit kernel expansions. Furthermore, the desire for developer productivity, at the expense of hyper-optimised implementations, is realised in this thesis by making use of Python, a **high level interpreted language**, for our software implementation.

1.2 Overview of the Kernel-Independent FMM

1.2.1 Motivation

First introduced by Ying et. al [8], the kernel-independent FMM (**KIFMM**) provides an algorithm that maintains the basic recursive structure and $O(N)$ asymptotic complexity of the analytic FMM, but without the requirement for the implementation of analytic expansions of the kernel function for each kernel. Instead the method relies only on kernel function evaluations. This allows software implementations to be written in an easily extensible manner for different kernels. The main difference to the analytic FMM of Section 1.1, lies in the the way that source and target densities are represented, and how the M2M, L2L and M2L operators are computed.

1.2.2 Algorithm Structure & Analysis

For the KIFMM presented in [8] the far field, $\mathcal{F}^B$, and near field, $\mathcal{N}^B$, have precise specifications. For a given box B centered at $\mathbf{c}$ with sides of length $2r$, $\mathcal{N}^B$ is a box centered at $\mathbf{c}$ with sides of length $6r$. The far field is then defined as $\mathbb{R}^d/\mathcal{N}^B$, where d is spatial dimension. Here, B is in the near field. Consider the potential in the far field $\mathcal{F}^B$, generated by a set of source particles, described by **source densities** $\{\phi_i,\ i \in I_s^B\}$ where I_s^B is the set of indices for the source particles in box B[9]. Specifying the indices for the source particles specifically to make it clear that they may be distinct from the target particles. These source particles can be equivalently described with an **upward equivalent density** distribution $\phi^{B,u}$

[9]This notation matches that used in [8] in order for ease of reference.

supported at discrete points on an **upward equivalent surface** $\mathbf{y}^{B,u}$ that encloses the set of source particles. The KIFMM relies on the assumption that the potential produced by the equivalent densities is smooth, which is guaranteed in the case that $\mathbf{y}^{B,u}$ does not overlap with the far-field $\mathcal{F}^B$ [8], furthermore the requirement that $\mathbf{y}^{B,u}$ must enclose all particles in B leads to the requirement that it must also not overlap with B. For second-order linear elliptic **PDE**s, for which the KIFMM is defined, and of which equation (1.4) is an example, the solution for the potential in the far field, which can be seen as an exterior Dirichlet problem, is guaranteed to be unique [8]. Therefore, the potentials induced by the source particles and the equivalent densities satisfy are guaranteed to be equivalent in the far field $\mathcal{F}^B$ if they coincide at the boundary of the far field $\mathcal{F}^B$, or anywhere between the boundary of the far field and the upward equivalent surface. This boundary is referred to as the **upward check surface**, $\mathbf{x}^{B,u}$, and the entire scheme is illustrated in figure (1.4A). The equality of the potentials from the source points and the equivalent density can be stated mathematically as follows,

$$\int_{\mathbf{y}^{B,u}} K(\mathbf{x},\mathbf{y})\phi^{B,u}d\mathbf{y} = \sum_{i\in I_s^B} K(\mathbf{x},\mathbf{y})\phi_i = q^{B,u} \text{ for any } \mathbf{x}\in\mathbf{x}^{B,u} \tag{1.20}$$

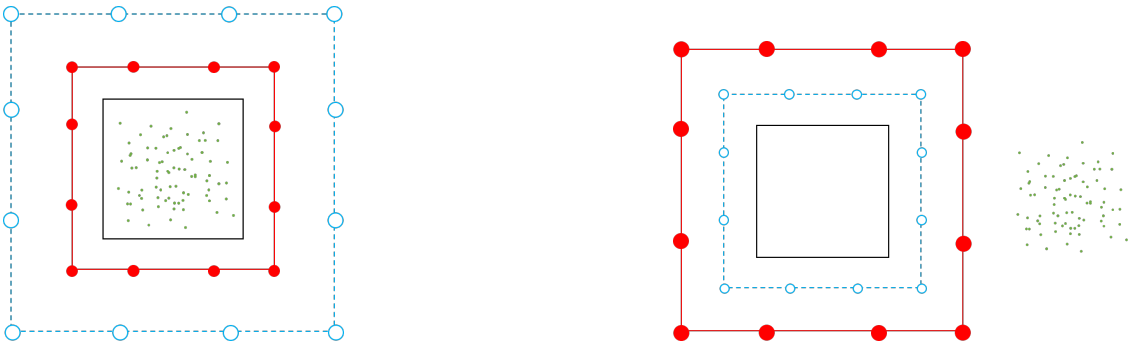

Figure 1.4: Cross section of three dimensional cubic upward/downward equivalent and check surfaces. Source points are denoted by green circles. Red solid lines denote equivalent surfaces, and blue checked lines denote check surfaces. The black solid line defines the box B. This figure is adapted directly from [8].

where the integral form of (1.1) is used for the summation of the contribution from the equivalent densities, and $q^{B,u}$ is referred to as the **upward check potential**, with the other symbols taking their previous definitions. One can define a very similar scheme for the case in which the source densities are in $\mathcal{F}^B$, as a potential induced by a **downward equivalent density** $\phi^{B,d}$ supported at discrete points on a **downward equivalent surface** $\mathbf{y}^{B,d}$. This surface needs to be located between the boundary of $\mathcal{F}^B$ and B, and again as the solution for interior Dirichlet style problems for the types of PDEs considered in this thesis is also unique [8], one can equate the potentials generated by the source points with that generated by the equivalent densities at some surface between $\mathbf{y}^{B,d}$ and B. This surface is called the **downward check surface**, $\mathbf{x}^{B,d}$, and a corresponding mathematical statement of the equality of the potentials can be written as follows,

$$\int_{\mathbf{y}^{B,d}} K(\mathbf{x},\mathbf{y})\phi^{B,d}d\mathbf{y} = \sum_{i\in I_s^{\mathcal{F}B}} K(\mathbf{x},\mathbf{y})\phi_i = q^{B,d} \text{ for any } \mathbf{x}\in\mathbf{x}^{B,d} \tag{1.21}$$

where $I_s^{\mathcal{F}B}$ represents the indices of source points in the far field of B, and $q^{B,d}$ is the **downward check potential**.

The equation (1.20) is an Fredholm integral equation, of the first kind, and it's clear that its solution $\phi^{B,u}$ can be seen to be equivalent to the multipole expansion for the source particles contained in B. Therefore, (1.20) can be seen to be correspond to the P2M operation. Similarly, the solution of (1.21) can be seen to correspond to a particle-to-local, or P2L operation. Crucially, this method of solving a set of linear equations to find equivalents of the multipole and local expansions does not depend on finding a series expansion of a kernel function, and just on its evaluation.

Using this language of equivalent densities and check surfaces, one is also able to write operations equivalent to the M2M, L2L and M2L operations. For a box A and it's parent box B, the M2M operation can be written as,

$$\int_{\mathbf{y}^{A,u}} K(\mathbf{x},\mathbf{y})\phi^{A,u}d\mathbf{y} = \int_{\mathbf{y}^{B,u}} K(\mathbf{x},\mathbf{y})\phi^{B,u}d\mathbf{y}, \text{ for any } \mathbf{x} \in \mathbf{x}^{B,u} \tag{1.22}$$

Once $\phi^{A,u}$ has been calculated for boxes at the leaf level of the tree, one can apply (1.22) to evaluate the multipole expansions for all the boxes containing source particles in the tree in a manner equivalent to the upward pass of Section 1.1. The M2M operation is illustrated in figure (1.5A).

For the downward pass, the M2L operation, between a box B and a box A in its interaction list can be written as,

$$\int_{\mathbf{y}^{A,u}} K(\mathbf{x},\mathbf{y})\phi^{A,u}d\mathbf{y} = \int_{\mathbf{y}^{B,d}} K(\mathbf{x},\mathbf{y})\phi^{B,d}d\mathbf{y}, \text{ for any } \mathbf{x} \in \mathbf{x}^{B,d} \tag{1.23}$$

This is illustrated in figure (1.6). Once the local expansions have been computed starting at level 2 of the tree, one can perform the L2L operation to transfer the local expansion of a parent box to its children, for a box A and it's child B, which can be written as,

$$\int_{\mathbf{y}^{A,d}} K(\mathbf{x},\mathbf{y})\phi^{A,d}d\mathbf{y} = \int_{\mathbf{y}^{B,d}} K(\mathbf{x},\mathbf{y})\phi^{B,d}d\mathbf{y}, \text{ for any } \mathbf{x} \in \mathbf{x}^{B,d} \tag{1.24}$$

The M2M, L2L and M2L operations are illustrated using cubic surfaces in figure (1.5) The reason for using cubic check and equivalent surfaces are for ease of integration in the case where the discretisation points chosen are from a regular Cartesian grid.

In principle, it is now possible to replace the kernel function expansions with the appropriate solutions of the integral equations (1.20), (1.22), (1.23) and (1.24). However all of the above equations are ill-conditioned, as they are ill-posed and in general infinite dimensional problems, and therefore require appropriate regularisation in order to solve. Consider the following general statement of the first-kind Fredholm equation that each of the above operations requires,

$$K\phi = q \tag{1.25}$$

This expression reflects the fact that in practical implementations these continuous integrals are discretised as matrix-vector products. Here K refers to a 'kernel matrix', which can be found via numerical quadrature, ϕ is a vector of equivalent density to be found, and q is a known vector containing the check potential. The reason for using cubic surfaces is the availability of simple quadrature rules to integrate

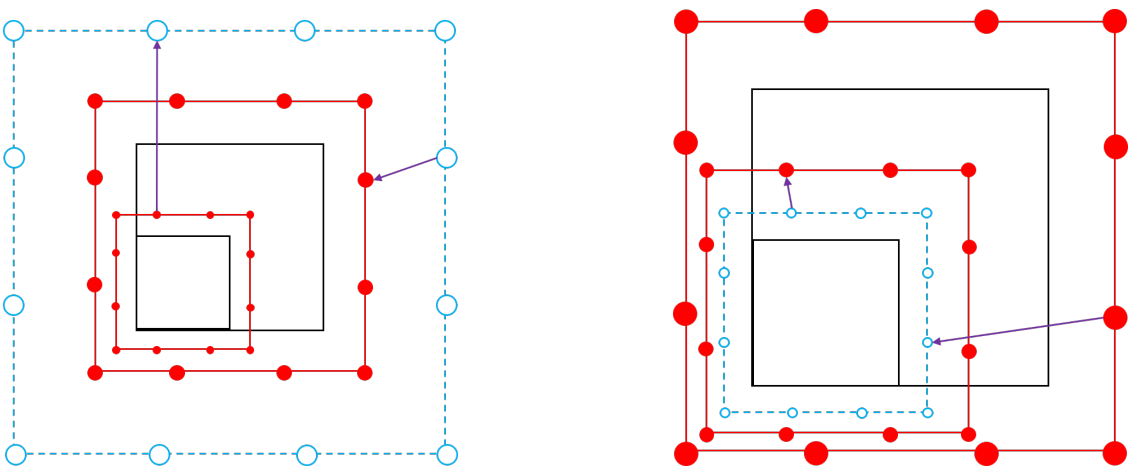

Figure 1.5: Cross section of three dimensional cubic surfaces. The (A) M2M operation and (B) L2L operation. Red solid lines denote equivalent surfaces, and blue checked lines denote check surfaces. The black solid line defines the box B. This figure is adapted directly from [8].

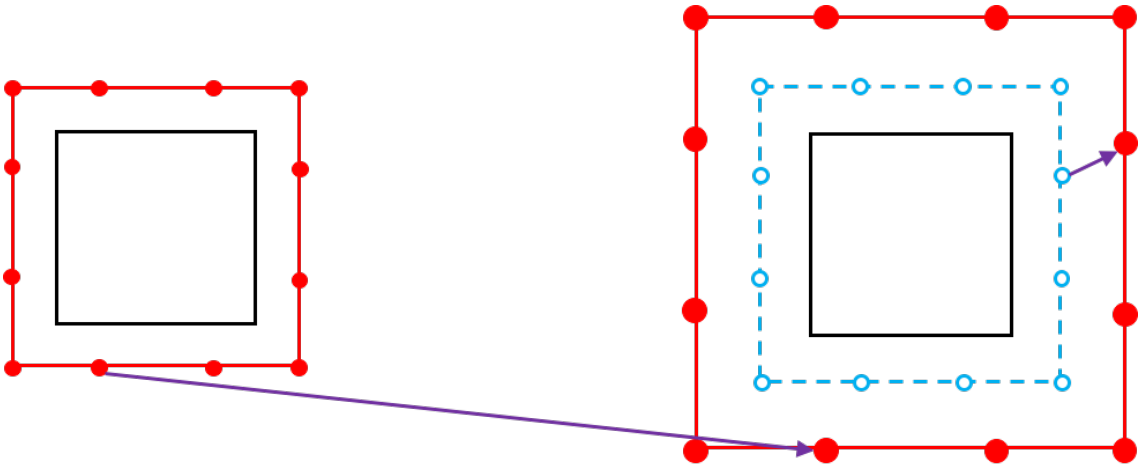

Figure 1.6: Cross section of three dimensional cubic surfaces for the M2L operation. Red solid lines denote equivalent surfaces, and blue checked lines denote check surfaces. The black solid line defines the box B.

over the faces of each surface. The specifics of the surfaces used in PyExaFMM is discussed in greater detail in Chapter 2, Section 2.3. One can use Tikhonov regularisation to write the solution,

$$\phi = (\alpha I + K^* K)^{-1} q \tag{1.26}$$

where α is an appropriate regularisation parameter, and I is the identity matrix. This equation, a second-kind Fredholm equation, can be solved in numerous ways. The authors of [8] make use of a Nystroem method, and indicate the possibility of using either Galerkin or Collocation methods. For ease of implementation, a simpler approach is to instead approximate the solution using a pseudoinverse estimated using from a Singular Value Decomposition, or **SVD**. This approach is adapted from the current C++ implementation of the KIFMM, ExaFMM-t, from the ExaFMM project [10].

Consider the SVD of a matrix A, with m rows and n columns,

$$A = U\Sigma V^* \tag{1.27}$$

where as usual U is the left singular matrix, V is the right singular matrix and Σ is a diagonal matrix whose elements are the singular values of A. We can write an approximate pseudo-inverse of A as [12],

$$A^\dagger = V\Sigma^\dagger U^* \tag{1.28}$$

here, $A^\dagger$ has n rows and m columns, and $\Sigma^\dagger$ is formed by taking the reciprocal of all the diagonal elements. Furthermore, to ensure numerical stability for this reciprocal calculation, one can choose to filter out components of V and U^* if their singular values smaller than a specified tolerance. The justification of the choices for the regularisation parameter α and the tolerance for the pseudoinverse are discussed in detail in Chapter 2, Section 2.3. In terms of the asymptotic complexity of the SVD, as the singular values/vectors are iterated through once in a straightforward manner to find the pseudoinverse, the complexity of this operation is at most $O(N)$, where N is the number of singlar values of A.

The KIFMM algorithm shares almost all of its algorithmic steps with the analytic FMM, the main novelties are the least-squares solves (1.26) to compute the required expansions of each box at each step of the algorithm. In turn these solves require the computation of the check and equivalent surfaces involved, as well as the numerical quadrature over the equivalent/source densities to compute the required check potentials. However, the the matrix to be inverted in the M2M operation as well as the L2L operation are very similar, which can be seen from figure (1.5). In fact, if the upward check surface is chosen to coincide with the downward equivalent surface, and the upward equivalent surface is chosen to coincide with the downward check surface, the matrix to be inverted for the M2M operation is the transpose of the matrix to be inverted for the L2L operation, except for a scaling factor dependent on the level of the Octree the surfaces are created in. This scaling factor is easy to identify, and for the model problem of three dimensional electrostatics with a Laplace kernel, the scaling factor between kernel matrices evaluated at two adjacent Octree levels, l, is $K^{l+1} = 2K^l$. Similarly, the required matrix inverse for the L2L and M2L operations exactly coincide except for a scaling factor. Therefore, practical implementations need to compute at most one matrix inversion which can be cached. This can then be scaled and applied as required at each step of the algorithm. Furthermore, for the Laplace kernel specifically, the scaling factors are present on both sides of the equations (1.22) and (1.24), cancelling each other out. This means that the M2M and L2L operations can also be calculated between just one parent box and its respective child boxes, and cached for later use.

Considering the asymptotic complexity of the inverse, its calculation is bound by a term proportional to the number of singular values of the kernel matrix, which in turn is dependent on the number of quadrature points chosen for the check and equivalent surfaces. Therefore as long as this number of quadrature points is kept small, the pseudoinverse does not effect the asymptotic complexity of the KIFMM. The remainder of the M2M, M2L and L2L operations are bound by the complexity of computing a matrix vector product between the calculated inverse matrix and the check potentials, which is resultantly also not impactful on the algorithm's asymptotic complexity as long as the requirement on the number of quadrature points is satisfied. As the other algorithmic analysis from Section 1.1 remains the same, we can see approximately how the $O(N)$ complexity is maintained for the KIFMM.

1.2.3 Summary

The different approach of the KIFMM leads to some different implementation challenges in comparison to the FMM. The key difficulty arises from the instability in computing the matrix inverse of the kernel matrix between the check and equivalent matrices. As mentioned, choices for the regularisation parameter α as well as the tolerance for singular values taken in the pseudoinverse are discussed with this in mind in Chapter 2, Section 2.3.

The ability to cache the M2M and L2L operators, after calculation for a single parent box and its respective child boxes, leaves the main bottleneck to precomputing all the required operators as the matrix-vector products required to compute the M2L operators. The original authors of the KIFMM accelerate the calculation of the M2L operation using a Fast Fourier Transform, or **FFT** [8]. To see why this can be done, consider the surfaces describing the M2L operation illustrated in figure (1.6), if the downward check surface of the target box, and the upward equivalent surface are chosen to be equivalent and lie on a regular Cartesian grid, then the component of the M2L operation that evaluates the action of a kernel function between these two surfaces can be considered as a convolution between the kernel function and the equivalent density. Mathematically, if the kernel function $K(x, y)$ depends only the difference between x and y as it does for these choices of surfaces for the M2L operation, the left hand side of (1.23) goes to,

$$\int_{\mathbf{y}^{A,u}} K(\mathbf{x}-\mathbf{y})\phi^{A,u}d\mathbf{y} = q^{B,d} \text{ for any } \mathbf{x} \in \mathbf{x}^{B,d} \tag{1.29}$$

Padding the empty nodes with zeros allows us to apply the Fourier convolution theorem, and solve for equivalent density as,

$$\phi^{A,u}(\mathbf{y}) = \mathcal{F}^{-1}\left[\frac{\mathcal{F}(q^{B,d}(\mathbf{x}))}{\mathcal{F}(K(\mathbf{x}))}\right] \tag{1.30}$$

The currently available major KIFMM implementations use this to accelerate the calculation of the M2L operator including ExaFMM-T and PVFMM [9, 10]. Though discussed in more detail in Chapter 2, Section 2.3, we state for the present that the expansion order p is by definition proportional to the number of quadrature points used to discretise the surface of the check and equivalent surfaces. Specifically, PyeExaFMM follows the example of ExaFMM-T and PVFMM, taking the relationship between p and number of quadrature points, n_q, to be,

$$n_q = 6(p-1)^2 + 2 \tag{1.31}$$

Meaning that each surface has $O(p^2)$ points, and therefore the M2L operation (1.23) is of $O(p^4)$ between a given target box and source box. This FFT based method accelerates the M2L calculation between a given target box and a given source box to $O(p^3 \log(p))$. In PyExaFMM we use an alternative acceleration scheme, based on low rank SVD approximations, discussed further in Chapter 2, Section 2.4. Therefore we defer to the literature for further details on FFT based acceleration methods [9].

The KIFMM presents many of the same opportunities for parallel implementation as the FMM. In terms of task-level parallelism, PyExaFMM implements parallel processing, via Python's native multiprocessing tools, for the calculation of the M2L operations. As these calculations involve dense matrix-vector products, and are also

ideal candidates to be transferred to a GPU for rapid parallel evaluation in the future. PVFMM implements task-level parallelism for the calculation of the M2M, L2L, and M2L operators with **MPI** as well as an interface with CUDA for processing the dense matrix-vector products. ExaFMM-T makes use of **OpenMP** for a similar purpose.

In summary, the key benefits of the KIFMM lie in the ability to write an implementation that is compatible with a wide class of kernel functions, without evaluating the expansion coefficients. This greatly reduces the complexity of the software design to support multiple applications of the FMM across different problem settings. Furthermore, the formulation of the key FMM operations as dense matrix-vector products makes it easy to map portions of the KIFMM to dedicated parallel hardware such as GPUs. The fact that KIFMM logic is not dependent on the form of the kernel function expansion, and just on kernel function evaluations, makes it trivial to separate the concerns of a software implementation, and write optimisation libraries for the KIFMM independently of the main loop (fig. 1.3) itself. The impact on software design and architecture is signficant, and is explored further in Chapter 2 Section 2.5. This design is the key benefit of PyExaFMM in comparison to existing KIFMM implementations, as PyExaFMM is already sacrificing a degree of performance via the choice of implementation language, we are able to implement some more effective software engineering principles to ease the burden on a user in terms of debugging and expanding upon PyExaFMM.

Strategy for Practical Implementation

2.1 Bottleneck Analysis

2.2 Efficient Tree Implementations

2.3 Operator Caching

2.4 Low-Rank Matrix Approximations using SVD

2.5 Software Design

Experiments & Results

3.1 Section 1

å

Conclusion

4.1 Section 1

Appendix

A.1 FMM Algorithm Specification

This pseudo-code is adapted from [4].

Initialisation: Choose a level of refinement of $n \approx \log_8 N$ and precision ϵ, set $p = \lceil -\log_2(\epsilon) \rceil$[1].

Upward Pass

Step 1

Form multipole expansions of potential field due to particles in each box at leaf level.

do $ibox = 1, .., 8^l$
Form p^{th} degree multipole expansion for each leaf box.
end

Step 2

Translate Multipole expansion to coarser levels from the bottom up.

do $l = n - 1, .., 0$
do $ibox = 1, .., 8^n$
Perform M2M operations.
end
end

Downward Pass

Computations at the coarsest possible level. For a given box, done by including interactions with those boxes which are well separated, and whose interactions have not been accounted for at the parent level.

Step 3

Form local expansion about center of each box at each level $l \leq n - 1$, describes field due to all particles that are not contained in the current box, it's near neighbours or it's secondary near neighbors.

[1] Different authors have different suggestions for p, the authors of [8] use $p = \log_c \epsilon$ with $c = \frac{4-\sqrt{3}}{\sqrt{3}}$

do $l = 1, ..., n-1$
  do $ibox = 1, ..., 8^l$
    Perform M2L translations.
  end
  do $1, ..., 8^l$
    Perform L2L operations.
  end
end

Step 4

After this step, local expansions are available at the leaf level. One can use this to evaluate potential at leaves from all particles in the far field.

do $ibox = 1, ..., 8^n$
  Find local expansion at leaf level, by doing M2L from interaction list.
end

Step 5

Evaluate local expansions at particle positions in all leaves

do $ibox = 1, ..., 8^n$
  For every particle in *ibox*'th box, evaluate local expansion.
end

Step 6

Compute nearest neighbors directly,
do $ibox = 1, ..., 8^n$
  For every particle in *ibox*'th box, compute potential directly with nearest neighbors.
end

Step 7

do $ibox = 1, ..., 8^n$
  Add direct and far field terms together for every particle in the *ibox*
end

A.2 Analytic FMM Operators for 3D Laplace Kernel

The expressions presented here are first derived in [4]

For l charges of strengths $q_1, ..., q_l$ located inside sphere D of radius a center at $Q = (\rho, \alpha, \beta)$, and that for points $P = (r, \theta, \phi)$ outside D potential given by the following **Multipole Expansion**,

$$\Phi(P) = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} \frac{O_n^m}{r'^{n+1}} \cdot Y_n^m(\theta', \phi')) \tag{A.1}$$

Where the points P and Q are defined such that $P - Q = (r', \theta', \phi')$. Then for any point P outside sphere D_1 of radius $a + \rho$, The multipole expansion can shifted with the following **M2M operation**,

$$\Phi(P) = \sum_{j=0}^{\infty} \sum_{k=-j}^{j} \frac{M_j^k}{r^{j+1}} \cdot Y_j^k(\theta, \phi) \tag{A.2}$$

Where,

$$M_j^k = \sum_{n=0}^{j} \sum_{m=-n}^{n} \frac{O_{j-n}^{k-m} \cdot J_m^{k-m} \cdot A_n^m \cdot A_{j-n}^{k-m} \cdot \rho^n \cdot Y_n^{-m}(\alpha, \beta)}{A_j^k} \tag{A.3}$$

and,

$$J_m^{m'} = \begin{cases} (-1)^{\min(|m'|,|m|)} & \text{if } m \cdot m' < 0 \\ 1 & \text{otherwise} \end{cases} \tag{A.4}$$

$$A_n^m = \frac{(-1)^n}{\sqrt{(n-m)! \cdot (n+1)!}} \tag{A.5}$$

For l charges of strengths $q_1, ..., q_l$ located inside sphere D_Q of radius a center at $Q = (\rho, \alpha, \beta)$, and that $\rho > (c+1)a$ with $c > 1$. The corresponding multipole expansion, converges inside sphere D_0 of radius a centered at origin. Inside D_0 the potential due to charges has the local expansion,

$$\Phi(P) = \sum_{j=0}^{\infty} \sum_{k=-j}^{j} L_j^k \cdot Y_j^k(\theta, \phi) \cdot r^j \tag{A.6}$$

where,

$$L_j^k = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} \frac{O_n^m \cdot J_k^m \cdot A_n^m \cdot A_j^k \cdot Y_{j+n}^{m-k}(\alpha, \beta)}{A_{j+n}^{m-k} \cdot \rho^{j+n+1}} \tag{A.7}$$

where A_n^m same as above,
but,

$$J_m^{m'} = \begin{cases} (-1)^{n'}(-1)^{\min(|m'|,|m|)} & \text{if } m \cdot m' < 0 \\ (-1)^{n'} & \text{otherwise} \end{cases} \tag{A.8}$$

n' refers to j, m' refers to k

and that for points $P = (r, \theta, \phi)$ outside D potential given by multipole expansion. This is also known as the multipole to local, or **M2L Operation**.

Let $Q = (\rho, \alpha, \beta)$ be the origin of a local expansion,

$$\Phi(P) = \sum_{n=0}^{p} \sum_{m=-n}^{n} O_n^m \cdot Y_n^m(\theta', \phi') \cdot r'^n \tag{A.9}$$

Where $P = (r, \theta, \phi)$ and $P - Q = (r', \theta', \phi')$.

$$\Phi(P) = \sum_{j=0}^{p} \sum_{k=-j}^{j} L_j^k \cdot Y_j^k(\theta, \phi) \cdot r^j \tag{A.10}$$

where,

$$L_j^k = \sum_{n=j}^{p} \sum_{m=-n}^{n} \frac{O_n^m \cdot J_{n-j,m-k}^m \cdot A_{n-j}^{m-k} \cdot A_j^k \cdot Y_{n-j}^{m-k}(\alpha, \beta) \cdot \rho^{n-j}}{A_j^k} \tag{A.11}$$

where A_n^m same as above,

$$J_{n,m}^{m'} = \begin{cases} (-1)^n(-1)^m & \text{if } m \cdot m' < 0 \\ (-1)^n(-1)^{m'-m} & \text{if } m \cdot m' > 0 \text{ and } |m'| < |m| \\ (-1)^n & \text{otherwise} \end{cases} \tag{A.12}$$

This operation defines the local to local, or **L2L Operation**.

Glossary

check potential The potential calculated from source particles, or an equivalent density distribution at a check surface. . 11, 12

check surface Defines the surface at which the potential caused by source points, and the equivalent density formulation coincide. . 11, 14, 15

CUDA 'Compute Unified Device Architecture', is a parallel computing API designed for the development of programs for **GPU**s . 10, 16

data-level parallelism In a multiprocessor system, data level parallelism is achieved by distributing data amongst different compute nodes to be executed upon in parallel. See **CUDA** . 10

equivalent density An equivalent representation of the potential generated by a discrete/continuous distribution. This equivalent density is supported at discrete points on an **equivalent surface**. . 10, 11

equivalent surface An equivalent surface supports equivalent density points. . 11, 14, 15

far field An set of particles in the far field are considered to be far away enough from the particle of interest that they are suitably described by a convergent multipole expansion. . 4, 8, 10, 12

FFT Fast Fourier Transform. . 15

FMM The Fast Multipole Method. . 1, 3, 4, 10, 14, 15

GPU 'Graphics Processing Unit', are specialised processors specifically designed for rapid parallel execution across large blocks of data. . 10, 16

high level interpreted language A High-Level language is one that offers an API and primitive data structures that strongly abstract from the details of the computer on which it is being run. An interpreted language is one in which codes are run directly via an *interpreter*, rather than first being compiled. In reality interpreted languages run in a 'virtual machine', which takes input code, transforms it to byte-code which is then translated into machine level instructions. This approach allows for greater portability of code, and developer productivity, at the expense of space and time overhead in running software. . 10

interaction list Boxes which are the children of the near-neighbours of the a box's parent box, but are not adjacent to the box itself. . 8, 10

KIFMM The 'Kernel Independent' Fast Multipole Method. . 1, 10, 13–16

L2L Translate from local of parent box to local expansion of child box.. 7, 8

L2P Evaluate local of leaf box at each target particle in a leaf box.. 8

M2L Translate from multipole of box A to local expansion of box B.. 8

M2M Translate from multipole of child box to multipole expansion of parent box.. 7, 8

MPI Message Passing Interface for implementing distributed memory parallelism. Each process is given it's own local memory, and can communicate and pass data and computational results to other processes being run in parallel. . 16

near field An set of particles in the near-field are considered to violate the the criteria for describing them with a convergence multipole expansion. . 6, 8–10

near neighbours Two boxes in computational tree are near neighbours if they are at the same level of refinement, and share a boundary point. . 7, 8, 20

OpenMP Open MultiProcessing is an API for implementing shared-memory parallelism. Each process is run on a separate thread, but a global memory space is shared. . 16

P2M Particle to multipole expasion operation.. 7, 8, 12

PDE Partial Differential Equation. . 11

post-order In reference to the traversal of heirarchical trees, post-order traversal is moving from the finest (leaf) level to the coarsest (root) level. . 7

pre-order In reference to the traversal of heirarchical trees, pre-order traversal is moving from the coarsest (root) level to the finest (leaf) level. . 7

source densities Source densities are associated with their respective **source particles**. In electromagnetic problems, these would correspond to charges. In gravitational problems, these would correspond to mass. . 10

source particles Particles are described as sources if they are consifered to contribute to the source field being evaluated at the target particles. They may or may not refer to the same set of particles as the target particles, for example in the classic N-Body problem, the target particles and the source particles are the same set. . 7, 8, 10, 12

SVD Singular Value Decomposition. . 13, 14

target particles Particles are described as targets if the potential due to a source field is being evaluated at these particle's positions, but they themselves are not being considered as a source for the field being evaluated. They may or may not refer to the same set of particles as the source particles, for example in the classic N-Body problem, the target particles and the source particles are the same set. . 8, 10

task-level parallelism Task-level parallelism is achieved when multiple threads or processes, running the same, or differing, code, in a multiprocessing system are executed with the same, or differing, data. . 10, 15, 16

well separated Two boxes in computational tree are near neighbours if they are at the same level of refinement, and are not near neighbours. . 20

Bibliography

[1] John A. Board et al. "Accelerated molecular dynamics simulation with the parallel fast multipole algorithm". In: *Chemical Physics Letters* 198.1 (1992), pp. 89 –94. ISSN: 0009-2614. DOI: https://doi.org/10.1016/0009-2614(92)90053-P. URL: http://www.sciencedirect.com/science/article/pii/000926149290053P.

[2] Barry A. Cipra. "The Best of the 20th Century: Editors Name Top 10 Algorithms". In: *SIAM News* 33.4 (2000).

[3] Frank Ethridge and Leslie Greengard. "A New Fast-Multipole Accelerated Poisson Solver in Two Dimensions". In: *SIAM J. Sci. Comput.* 23.3 (Mar. 2001), 741–760. ISSN: 1064-8275. DOI: 10.1137/S1064827500369967. URL: https://doi.org/10.1137/S1064827500369967.

[4] Leslie Greengard. "The Rapid Evaluation of Potential Fields in Particle Systems". PhD thesis. Yale University, 1987.

[5] Leslie Greengard and June Yub Lee. "A direct adaptive poisson solver of arbitrary order accuracy". English (US). In: *Journal of Computational Physics* 125.2 (May 1996), pp. 415–424. ISSN: 0021-9991. DOI: 10.1006/jcph.1996.0103.

[6] David J. Griffiths. *Introduction to Electrodynamics*. 4th ed. Cambridge University Press, 2017. DOI: 10.1017/9781108333511.

[7] Wen-Mei W. Hwu. *GPU Computing Gems Emerald Edition*. 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011. ISBN: 0123849888.

[8] Denis Zorin Lexing Ying George Biros. "A kernel-independent adaptive fast multipole algorithm in two and three dimensions". In: *Journal of Computational Physics* 196.2 (2004), pp. 591–626. DOI: http://dx.doi.org/10.1016/j.jcp.2003.11.021.

[9] Dhairya Malhotra and George Biros. "PVFMM: A Parallel Kernel Independent FMM for Particle and Volume Potentials". In: *Communications in Computational Physics* 18.3 (2015), 808–830. DOI: 10.4208/cicp.020215.150515sw.

[10] Lorena A. Barba Rio Yokota. *ExaFMM User's Manual*. 2011. URL: http://www.bu.edu/exafmm/files/2011/06/ExaFMM-UserManual1.pdf.

[11] Vladimir Rokhlin. "Rapid Solution of Integral Equations of Theory in Two Dimensions". In: *Journal of Computational Physics* 86.2 (Feb. 1990), pp. 414–439. DOI: 10.1016/0021-9991(90)90107-C.

[12] Lloyd N. Trefethen and David Bau. *Numerical Linear Algebra*. SIAM, 1997. ISBN: 0898713617.