Modern Research Software For Fast Multipole Methods

Srinath Kailasa

A thesis submitted in partial fulfillment of the requirements

for the degree of Doctor of Philosophy

Department of Mathematics

University College London

October, 2024

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.

Acknowledgements

The completion of this thesis simply wouldn't have been possible without the strong prevailing winds of emotional support from my family the frequency of fun with my friends, and the sympathetic and dedicated teachers and colleagues I met at UCL and across the world. Thank you *all* for giving me this opportunity, I'm excited for what the future brings.

నేను ఇది మా అమ్మ కోసం రాశాను

UCL Research Paper Declaration Form

Published Manuscripts

1. PyExaFMM: an exercise in designing high-performance software with Python and Numba.

    a) DOI: 10.1109/MCSE.2023.3258288

    b) Journal: Computing in Science & Engineering

    c) Publisher: IEEE

    d) Date of Publication: Sept-Oct 2022

    e) Authors: Srinath Kailasa, Tingyu Wang, Lorena A. Barba, Timo Betcke

    f) Peer Reviewed: Yes

    g) Copyright Retained: No

    h) ArXiv: 10.48550/arXiv.2303.08394

    i) Associated Thesis Chapters: 4

    j) Statement of Contribution

        i. Srinath Kailasa was the lead author and responsible for the direction of this research and the preparation of the manuscript.

        ii. Tingyu Wang offered expert guidance as on fast multipole method implementations, and critical feedback of the manuscript.

        iii. Lorena A. Barba served as an advisor, offering valuable insights into the structure of the manuscript, suggesting improvements to enhance clarity and impact of the work.

        iv. Timo Betcke provided significant advisory support, contributing to the design and framing of the research, interpretation of the results and provided critical feedback of the manuscript.

Unpublished Manuscripts

1. M2L Translation Operators for Kernel Independent Fast Multipole Methods on Modern Architectures.

a) Intended Journal: SIAM Journal on Scientific Computing

b) Authors: Srinath Kailasa, Timo Betcke, Sarah El-Kazdadi

c) ArXiv: 10.48550/arXiv.2408.07436

d) Stage of Publication: Submitted

e) Associated Thesis Chapters: 3, 6

f) Statement of Contribution

i. Srinath Kailasa was the lead author and responsible for the direction of this research and the preparation of the manuscript.

ii. Timo Betcke provided significant advisory support aid with interpretation of the results and provided critical feedback of the manuscript.

iii. Sarah El-Kazdadi provided expert guidance on the usage of explicit vector programming, which was critical for achieving the final results presented.

2. kiFMM-rs: A Kernel-Independent Fast Multipole Framework in Rust

a) DOI: TODO

b) Intended Journal: Journal of Open Source Software

c) Authors: Srinath Kailasa

d) Stage of Publication: Submitted

e) Associated Thesis Chapters: 5

e-Signatures confirming that the information above is accurate

Candidate:

Date:

Supervisor/Senior Author:

Date:

Abstract

This thesis is concerned with the development of a software platform for the kernel independent Fast Multipole Method (kiFMM), a variant of the widely applied Fast Multipole Method (FMM) algorithm which finds far reaching application across computational science. Indeed, for certain dense rank-structured matrices, such as those that arise from the boundary integral formulation of elliptic PDEs, the FMM and its variants accelerate the computation of a matrix vector product from $O(N^2)$ to just $O(N)$ in the best case.

We demonstrate the efficacy of our software's flexible design by contrasting implementations of a key bottleneck known as the Multipole to Local (M2L) field translation, and present a new highly optimised approach based on direct matrix compression techniques and BLAS operations, which we contrast with the current state of the art approach based on FFTs for kiFMMs. We show that we are able to achieve highly-competitive runtimes for three dimensional problems described by the Laplace kernel with respect to the state of the art, and often faster depending on the available hardware, with a simplified approach. Our approach is well suited to the direction of development of hardware architectures, and demonstrates the importance of re-considering the design of algorithm implementations to reflect underlying hardware features, as well as the enabling power of research software for algorithm development.

The software itself is written in Rust, a modern systems programming language, with features that enable a data oriented approach to design and simple deployment to common CPU architectures. We describe our design and show how it allows us to extend our software to problems described by the Helmholtz kernel, at low frequencies, as well as in a distributed memory setting, where emphasis has been placed on retaining a simple user interface and installation suitable for non-software experts, while remaining modular enough to remain open to open-source contribution from specialists. We conclude with both single node and HPC benchmarks, demonstrating the scalability of our software as well as its state of the art performance.

Impact Statement

This thesis establishes norms and practices for developing practical implementations of the kernel independent Fast Multipole Method (kiFMM), which will be of significant utility to the developers specialising in this and related algorithms. During this research we re-visited established codes for the kiFMM, identified software construction techniques that can lead to more flexible implementations that allow users to experiment, exchange, and build upon critical algorithmic subcomponents, computational backends, and problem settings - which are often missing from competing implementations which focus achieving specific benchmarks. For example, the flexibility of the software presented in this thesis allows for the critical evaluation of key algorithmic subcomponents, such as the 'multipole to local' (M2L) operator which we presented in [18].

As the primary outputs are open-source software libraries [19, 17] which are embedded within existing open-source efforts, most significantly the Bempp project, with an existing user-base the software outputs of this thesis are likely to have a wide ranging impact in academia and industry influenced by the demand for these softwares. Furthermore, the adoption and promotion of Rust for this project, and within our group, establishes further the utility of this relatively new language for achieving high-performance in scientific codes, which in recent years has been the subject of growing interest in the wider high-performance scientific computing community.

Contents

Introduction

Since its introduction in the late 1980s by Greengard and Rokhlin [15], the Fast Multipole Method (FMM) has become a hallmark algorithm of scientific computing often cited as one of the 'top 10' algorithmic advances of the past century [8]. The problem it addresses was originally motivated by N particle simulations in which the interactions are *global* but with a strongly decaying property. Motivating examples being N particles interacting via gravity or electrostatic forces. In these cases, as well as for interactions delineated by particular interaction *kernels*, interactions between distant *clusters* of particles can be represented by truncated series expansions. This is indeed where the name for the original presentation originated, as multipole expansions derived from the fundamental solution of the Poisson equation, often called the *Laplace kernel* in FMM literature were used to form these truncated series expansions,

$$K(\mathbf{x}-\mathbf{y}) = \begin{cases} = -\frac{1}{2\pi}\log(|\mathbf{x}-\mathbf{y}|),\ d=2 \\ = \frac{1}{4\pi|\mathbf{x}-\mathbf{y}|},\ d{=}3 \end{cases} \tag{1.1}$$

where $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ and d is the spatial dimension. Furthermore, by using a hierarchical discretisation for the problem domain, increasingly distant interactions can be captured while still using truncated sums to express the potential due to particles contained within each subdomain in the hierarchy. With this, the FMM is able to reduce the $\mathcal{O}(N^2)$ operations required to evaluate the potentials at each of N particles due to all other particles into an algorithm requiring just $\mathcal{O}(N)$ for problems described by the Laplace kernel (1.1). The crucial advantage of the FMM is that

it comes equipped with rigorous error estimates, which guarantee exponential convergence with increasing numbers of series terms used in the truncated expansion, such that the problem could be evaluated to any desired accuracy while retaining the $\mathcal{O}(N)$ complexity bound for number of operations.

In the preceding decades, the FMM has been extended with numerous variants that utilise a similar algorithmic framework, often with associated software efforts. Despite this, unlocking the highest available performance for practical FMM implementations remains an active area of research. Principally this can be attributed to the dramatic changes in the landscape of computing technologies in the time since the algorithm's first introduction.

Since the end of Dennard Scaling[1] in the 2000s, hardware design and development has had a renewed focus on enhancing parallelism. Hierarchical levels of parallelism are now available to programmers representing a significant departure from the Single Instruction Single Data (SISD) systems which existed at the time of the FMM's introduction. From true hardware parallelism available via Core Level Parallelism (CLP), where multiple Central Processing Unit (CPU) cores are able to independently execute tasks or threads simultaneously in a Multiple Instruction Multiple Data (MIMD) fashion, to Thread Level Parallelism (TLP), extended by technologies such as *hyperthreading*, whereby each physical core is able to run multiple threads sharing an address space simultaneously, which are optimally scheduled by the operating system. Data Level Parallelism (DLP), whereby the same operation is applied to multiple data elements simultaneously is made available to programmers via the Single Instruction Multiple Data (SIMD) execution model represented by *vector instruction sets* available to take advantage of special hardware registers on modern CPUs, as well as via the the Single Instruction Multiple Threads (SIMT) execution model of modern GPUs.

The importance of simultaneously exploiting multiple levels of parallelism for

[1]First articulated in 1974 by Robert Dennard, Dennard scaling described how as transistors were miniaturised their power density remarkably was able to be maintained as a constant. This held true until the mid 2000s, at which point physical limits on heat dissipation and leakage current lead to power efficiency gains via miniaturisation plateauing despite the steady increase in miniaturisation described by Moore's law, marking the end of Dennard scaling. This in turn lead to the growth of multicore processors and specialised hardware accelerators, as a way to increase available computing performance without increasing power consumption.

achieving performance with modern software, and the increasing disparity between memory bandwidth and compute resources [11], makes carefully organising memory access through the deep hierarchies of modern memory technologies increasingly the key bottleneck to unlocking performance in both a shared memory and distributed setting to ensure that memory movements do not dominate runtimes. In this context the expense of pairwise evaluations of (1.1) addressed by the FMM are increasingly of less relative importance for even moderately sized problems involving hundreds of thousands of source and target particles in shared memory[2], as the direct evaluation is embarrassingly parallel over each target particle with significant opportunity for memory re-use and well suited to the SIMD or SIMT execution models of modern CPUs or GPUs, respectively. Despite the $\mathcal{O}(N)$ complexity bound offered by the FMM, the operators from which it is composed contain practically significant constants and implicit non-contiguous memory accesses, making achieving practical performance challenging particularly in three dimensions.

The algorithm on its own was considered of importance, though its relative importance was disputed throughout the 1990s due to large constants in its associated operators, despite its optimal complexity. [CITE kurzak, aluru]. Some even questioned whether for key applications such as molecular dynamics whether the FMM could be competetive until unphysical particle sizes up to 10^60 in comparison to established techniques based on tree codes or Ewald summations due to these hidden factors in complexity.

However, a series of landmark software efforts throughout the 2000s and 2010s revitalised the FMM, and closely related variatn algorithms as a *practical tool* for achieving the scale required to take advantage of the latest High Performance Computing (HPC) systems. Particularly prominent examples being the ExaFMM project [4, 25],and PVFMM [22]. Indeed, software for the FMM and closely related methods have cumulatively been the recipients of three ACM Gordon Bell awards [6]. Particular recent focii have been to examine how heterogenous architectures can be taken advantage of [22], whereby CPUs and GPUs are used in concert for data

[2]We demonstrate this with specific benchmarks in Chapter 6 for a selection of CPU architectures.

organisation and CPU bound components of the algorithm respectively, as well as taking advantage of existing parallel runtimes for achieving task-based parallelism to high effect especially in a distributed memory setting [7, 1].

The collective weakness of existing software efforts however is their brittleness. The complexity for developing high-performance software in a research setting results in softwares that are strongly adapted for a particular algorithmic approach, hardware architecture, or runtime system, often with the goal of achieving a particular benchmark or demonstrating the utility of a particular technique. Relatively little software is under active maintenance, with clear technical documentation on how performance was achieved, and fewer still open their subcomponents for extension, have downstream users and simple user interfaces. As a result, it is difficult to compare algorithmic and software optimisations taken by different implementations, and contrast their relative merits, as well as to see how particular approaches adapt to advances in available hardware and software.

Achieving the greatest available performance, apart from inherently being of scientific interest, enables larger and more detailed scientific simulations. The FMM has been deployed as a core numerical method in the solution of elliptic Partial Differential Equations (PDEs) when formulated as boundary or volume integral equations in combination with iterative methods, therefore acts as a crucial subcomponent in derived solvers applied to a vast array of problems in computational physics, from acoustics [16], and electromagnetics [10] to fluid dynamics [10], and biomolecular electrostatics [29, 26]. Generalised variants of the FMM have also been applied in other areas requiring fast kernel summation arising in computational statistics, such as in the widely applied Kalman filter [21], modelling Gaussian processes [2], and machine learning [14, 23]. Indeed, faster N particle kernel summations, of which the FMM is an example, have been identified as a key benchmark operation for optimisation for in HPC due to their broad utility [3], demonstrating the importance of developing performant open-source software for FMMs.

However scientific software that hopes to achieve widespread adoption must also focus on *usability* in addition to performance. Which in this context means a soft-

ware that is easy to deploy on current and emerging hardware and software environments, can be interfaced by common programming languages, and is open to extension to new algorithmic approaches and implementations, while remaining highly-performant. This in turn makes the structure of the software itself an object of study.

The focus of this thesis can therefore be summarised as the development of a *platform* for developing FMMs, in particular the kernel independent Fast Multipole Method (kiFMM), a 'black box' variant of the FMM which doesn't rely on explicit series expansions of the fundamental solution and only on its evaluation, that thrives even after the conclusion of this research project. Our software consists of re-usable subcomponents, and acts as a useful tool for algorithmic investigation while being capable of state of the art performance. We demonstrate this through studies into both the design as well as the application of our software in investigating the implementation of algorithmic subcomponents of the kiFMM.

We begin in Chapter 2 by reviewing the literature on algorithmic and software developments for FMMs, and related methods such as $\mathcal{H}$ matrices, with a particular focus on the kiFMM currently implemented by our software. We review the computational structure general of these algorithms, and identify the requirements and bottlenecks for fast implementations for non-oscillatory and oscillatory problems, as well as in a distributed memory setting. We also review recent software developments, and describe the current open-source landscape.

In Chapter 3 we present an application of our software through an investigation of optimisations for the field translations for the kiFMM, devoting the most focus to the Multipole to Local (M2L) operation, a critical bottleneck for FMMs due to its requirement for significant data re-organisation due to the implicit non-contiguous memory accesses required by this operation. As the M2L is of convolution type current state of the art approaches are based on Fast Fourier Transforms (FFTs) with an $\mathcal{O}(N \log N)$ complexity bound, however they require careful explicit vector programming to achieve practical performance. Remarkably, we find that we are able to construct a highly competitive scheme based on direct matrix compression

techniques and Basic Linear Algebra Subprograms (BLAS) operations due to the superior memory re-use, despite it requiring a greater number of FLOPs. Indeed, our scheme's reliance on BLAS operations, and comparatively simple algorithmic structure and implementation, make it well suited to direction of development of computer hardware, and depending on the underlying hardware can even offer faster performance than the current state of the art approach. The remainder of the chapter describes the formulation of the other operators from which the kiFMM is composed in both a single node, and present our approach for how these operators are extended to a distributed setting, and the simplified communication reducing strategies we use.

The requirements of software engineering in science, with a focus on rapid iteration, small teams, and requirements for high performance, make the selection of an appropriate programming environment crucial to the success of scientific software projects. In Chapter 4 we present an investigation on scientific programming environments. Discussing the trade-off between high and low-level languages. Specifically our language of choice Rust, a modern systems-level programming language rapidly growing in popularity in science and engineering, is evaluated with respect to our previous approach that used Python, the de-facto high-level language of choice for high-level scientific computing, with an increasing community of developers focussed on providing features that enable high-performance.

Chapter 5 describes in detail the engineering approach of our software, particularly the employment of Rust's 'trait' system for the implementation of 'data oriented design', which is a software design philosophy that emphasises that business logic should minimise memory movements, and that data structures should be as close to contiguously layed out as possible. We present the utility of our design via a set of case studies that demonstrate its flexibility. In particular we examine how traits enable a flexible approach for switching between differing implementations of operators (Section 5.3), single and multi node settings (Section 5.4) and indeed different algorithms for the FMM (Section 5.5).

Chapter 6 contains benchmark experiments for our software, on a single node for

Laplace and Helmholtz problems as well as in a HPC setting for Laplace problems, for which we find we are able to scale our software to the order of X unknowns. Additionally, though the kiFMM is presented for non-oscillatory problems, or oscillatory problems at 'low' frequencies, we examine in Section **??** just how high frequencies can be taken, given a highly performant kiFMM implementation, finding that we are able to compute even highly-oscillatory problems in practically useful times, despite losing out on optimal scaling presented in other methods. We conclude with a reflection on our results and suggestions for future avenues of investigation in Chapter 7.

Review of Fast Multipole Methods

2.1 Fast Multipole Methods

We use the case of evaluating electrostatic potentials to motivate the FMM for non-oscillatory problems, mirroring the original presentation [15]. Consider the electric field, $\mathbf{E}$ due to a static charge distribution $q(\mathbf{y})$ which is supported over some finite domain $\mathbf{y} \in \Omega \subset \mathbb{R}^d$. It can be defined in terms of a scalar potential ϕ.

$$\mathbf{E} = -\nabla\phi$$

which itself can be seen to satisfy Poisson's equation,

$$\begin{cases} -\Delta\phi(\mathbf{x}) = q(\mathbf{x}), & \text{for } x \in \mathbb{R}^d \\ \lim\limits_{|x|\to\infty} u(\mathbf{x}) = 0 \end{cases}$$

where $d = 2, 3$ in problems of interest.

We can write the evaluation of the potential at a point $\mathbf{x}$ as a convolution of the source with the fundamental solution of the Poisson equation,

such that,

$$\phi(\mathbf{x}) = \int_{\mathbb{R}^d} K(\mathbf{x}-\mathbf{y})q(\mathbf{y})d\mathbf{y}, \;\; \mathbf{x} \in \mathbb{R}^d \tag{2.1}$$

Under an appropriate discretisation, where care is taken to appropriately handle the singularity in the Laplace kernel (1.1), we see that this integral corresponds to

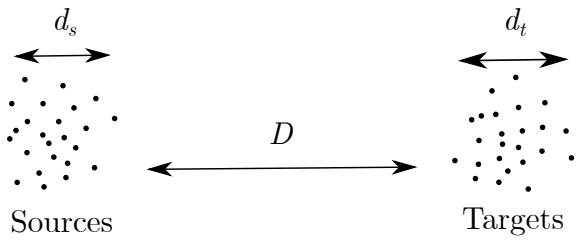

Figure 2.1: A set of source and target particle cluster, where the width of each cluster is significantly less than the distance separating them, $d_s, d_t \ll D$.

a matrix vector multiplication, where the matrix is *dense*, i.e. it consists only of non-zero entries.

As we are principally concerned with the simpler problem of evaluating the potential due to a discrete charge distribution, with N charges we can replace $q(\mathbf{y})$ with $\{q(\mathbf{y}_j)\}_{j=1}^N$ associated with *source particles* $\{\mathbf{y}_j\}_{j=1}^N \in \mathbb{R}^d$, the integral for potential evaluated at M *target particles*, $\{\mathbf{x}_i\}_{i=1}^M \in \mathbb{R}^d$ becomes a discrete sum,

$$\phi(\mathbf{x}_i) = \sum_{j=1}^{N} K(\mathbf{x}_i - \mathbf{y}_j) q(\mathbf{y}_j), \;\; i = 1, ..., M \tag{2.2}$$

where we can handle the singularity by setting,

$$K(\mathbf{x}_i - \mathbf{y}_j) = \begin{cases} 0, \;\; \mathbf{x}_i = \mathbf{y}_j \\ K(\mathbf{x}_i - \mathbf{y}_j), \;\; \text{otherwise} \end{cases} \tag{2.3}$$

We see that the sum (2.2) corresponds to a dense matrix vector multiplication,

$$\phi = Kq \tag{2.4}$$

Naively computed this requires $\mathcal{O}(MN)$ operations, where in general the source and target particles may correspond to the same set. The FMM relies on a *degenerate* approximation of the interaction kernel when subsets, or *clusters*, of source and target particles are sufficiently separated as sketched in Figure 2.1. Following the discussion in [18] the sum (2.2) can be written as,

$$\phi(\mathbf{x}_i) \approx \sum_{p=1}^{P} \sum_{j=1}^{N} A_p(\mathbf{x}_i) B_p(\mathbf{y}_j) q(\mathbf{y}_j), \;\; i = 1, ..., M \tag{2.5}$$

where we call P the expansion order, taken such that $P \ll N$, $P \ll M$. The

functions A_p and B_p are defined by the approximation scheme used by a particular approach for the FMM, in the original presentation the calculation,

$$\hat{q}_p = \sum_{j=1}^{N} B_p(\mathbf{y}_j) q(\mathbf{y}_j) \tag{2.6}$$

Corresponded to the coefficients of an order P multipole expansion due to the source charges. Following which the potential is approximated by,

$$\phi(\mathbf{x}_i) \approx \sum_{p=1}^{P} A_p(\mathbf{x}_i)\hat{q}_p, \;\; i = 1, ..., M \tag{2.7}$$

at the target particles. The approximation of the potential with this scheme can be seen to require $\mathcal{O}(P(M+N))$ operations. The accuracy of this approximation scheme, and the error bounds provided by the FMM, depends on the distance between the source and target clusters remaining large relative to their width. This condition is often referred to as an *admissibility condition* in the literature. FMMs therefore split the sum (2.2) into *near* and *far* components when considering arbitrary clusters of source and target particles,

$$\phi(\mathbf{x}_i) = \sum_{\mathbf{y}_j \in \text{Near}(\mathbf{x}_i)} K(\mathbf{x}_i, \mathbf{y}_j) q(\mathbf{y}_j) + \sum_{\mathbf{y}_j \in \text{Far}(\mathbf{x}_i)} K(\mathbf{x}_i, \mathbf{y}_j) q(\mathbf{y}_j), \;\; i = 1, .., M \tag{2.8}$$

In cases where a source and target cluster can be considered *admissable*, i.e. the source cluster is considered in the *far field* of the target cluster such that each $\mathbf{y}_j \in \text{Far}(\mathbf{x}_j)$, we apply the approximation (2.5). However, when a source and target cluster are *inadmissable*, such that the source cluster is considered in the *near field* of a target cluster such that each $\mathbf{y}_j \in \text{Near}(\mathbf{x}_j)$ we are left to evaluate the sum directly via (2.2).

The notion of admissability is made more concrete by reference to a data structure chosen to discretise the problem. For the FMM quadtrees and octrees are commonly used in two and three dimensions respectively. These are data structures in which a d-dimensional bounding box is used to cover the source and target particles, and is recursively divided into 2^d 'child' boxes. This process can be either

'adaptive' or 'uniform'. In the former case, the box is divided until a user defined threshold defining the maximum number of points per terminal leaf box is reached, which can lead to adjacent boxes of differing sizes and is able to closely model extreme particle distributions. In the latter case, boxes are divided such that each leaf box is of the same size, specified by a user defined parameter controlling the maximum depth of the tree.

- In order to achieve its $\mathcal{O}(N)$ complexity the FMM is structured to reduce to a minimum the number of sums evaluated between inadmissable clusters.

- Algorithm sketch, note on adaptivity and how it's defined and achieved (weak and strong)

- sketch of complexity for full generic algorithm

- The power of the FMM's original insight was the split of near and far interactions, and the usage of 'interaction lists' to describe the interactions relative to each box. Furthermore the organisation into a hierarchical algorithm provide the tools for achieving linear complexity for non-oscillatory problem key to which is the ability to translate between multipole and local expansion representations. These three features (i) the construction of interaction lists and (ii) the low-rank approximation scheme used to express the field due to a collection of charges and (iii) the methods used to translate between these operators principallly define all existing FMM variants, which retain a common recursive algorithmic structure.

- From a computational perspective, the FMM offers even more features that combine with those above that result in algorithmic variants, the main ones being approaches to (i) tree construction, and resulting data layout and access schemes (ii) the approach to parallelisation and distribution across nodes (iii) the approach to mapping algorithmic subcomponents to take advantage of high-performance software and hardware. This has resulted in a diverse set of algorithmic approaches, some of which we review in Section 2.6, all with their own benefits and trade-offs between implementational and algorithmic complexities.

- There are large lurking constants in FMM.

- Aluru, some arguments that the original FMM was not O(N) for Laplace -

Kurzak paper, in the 00s some people even thought that FMM only competetive with Ewald/treecodes at 10e62 particles, completely unfeasible - disproved by a series of landmark codes and developments, which showed that *particular* FMMs could harness petascale compute resources.

- Give a break down table for each operator in analytical, and kernel independent cases (bbFMM and kiFMM) without acceleration, in 3D, and memory accesses total

- Have a breakdown table of each approach *with acceleration* in 3D, and memory accesses (total) required

- Despite most schemes in principle having a high flop/byte ratio, in practice the M2L by construction is memory limited, due to the non-contiguous nature of interaction lists.

- In addition to practical significance of each approach due to requirements of data access on modern CPUs, Kurzak / Dongarra paper reference on how rules have changed.

- This was addressed in a series of papers in the 90s/2000s to address this with new forms of translation operators, that reduced the asymptotic complexity of the far-field evaluations, some of which we will cover in chapter on field translations.

- Principally, can be grouped into two broad camps: analytical and algebraic. The latter of which coincide with the H matrix approach.

- Distinguishing features math: compression scheme for low rank sub-blocks - alg vs analytical schemes. admissability - algebraic - ACA, SVD, QR, Randomised, Krylov. - interpolation basis - MFS, Cheb - analytical - expansions - Sph Harm, Cartesian (Taylor) - compression - diagonalisation, plane wave/exponential - hierarchical block structure (HSS/HODLR/H/H2). - FMM is an instance of H2. analytical FMM is H2 but unrolled loop.

- Distinguishing features CS - Data access - tree (ORB vs Octree) - Octree: - pointer based - bottom up - ORB scheme, bounding boxes (square etc) - Abduljabbar - hardware - GPU support, het, CPU only - SIMD - interaction list data access - linearisation of the tree. - ghost exchange in distributed memory. - software - asych - runtime model - threading model - software platform - language and platform

support, performance. - ability to support all the variants listed above, and be flexible to swap them out and compare them as toolbox for understanding the FMM.

- Combinatorial explosion of FMM variants, very little work on unifying the work that has been done. Unclear of which the best approaches even are. People find an approach and stick to it. Especially in the context of rapidly changing hardware environment, where many previous acceleration schemes based on optimal flops may no longer be optimal.

- Heuristically, the best approach in practice is likely to be one that maximises flop/bytes, while simulatneously linearising interaction list data access to take advantage of modern cache hierarchies, must also be tunable in terms of building up cache.

- The actual expansion scheme is also of relevance, as will reflect the nature of the flops - e.g. are we computing spherical harmonics and other special functions - however, the consequence of this is likely to matter less than admissability choices, and linearisation of data access schemes.

- Have ambikasaran diagram here for matrix decomposition variants as it's very clear.

2.2 Computational Structure

- The data flow is shown in Figure 2.2 for all the operators for a uniform FMM in two dimensions with two levels taken in the hierarchical tree.

- In this section we map this data flow to the available parallelism on shared and distributed memory systems.

- Tree construction

- bottom up vs top down, complexities of each approach - Morton vs Hilbert spatial decompositions - ORB as an alternative approach. - What are the positives and trade-offs of different spatial encodings? How much do I think it matters? - linked list vs linear - linear us, pvfmm - exafmm-t as an example of linked list variant.

- Interaction list access - Originally, based on linked lists - Anderson, - Move towards masking techniques, bebendorf GPU and Gumerov, as GPUs became a thing. - High degree of success with index pointer based techniques, us, pvfmm and exafmm.

- Runtime, Thread and Data level parallelism - threading model - tbb/rayon approach on work stealing vs openmp fork-join - data level - achieved easily if formulated with modern BLAS libraries for critical operators.

- Formulation of translation operators. - analytical (unrolled) formulations - relies on special functions, multithreading possible but manual - can be written as BLAS - algebraic - BLAS

Ghost data exchange of interaction lists - MPI - point to point vs collective vs neighbourhood collective - Runtime systems - BSP (ie MPI) vs task based

- implications for caching based on granularty of tasks.

How can parallelism best be expressed?

- Unroll the FMM loop - linearise data access for interaction lists - ensure that as many key operations as possible can use those implementations with high flop/byte ratio and take advantage of SIMD i.e. open source FFTW/BLAS/LAPACK.

What does this mean in terms of the operators?

- Data dependency at coarsest granularty is over levels, reminiscent of V list cycle

in multigrid approaches. - Picking this level of granularty gives the greatest flexibility to an implementer to explore options. - Furthermore, the P2P and recursive scheme are completely data independent and should be treated as such.

- Ensuring data locality for translations - i.e. it's clear that M2M and L2L can be batched over sets of siblings. - but can also batch over multiple sets of siblings at each level.

- the challenge ise the M2L, where the number of accesses is as no order of magnitude larger than M2M and L2L. - P2P, standard techniques can be adapted due to natural parallelism over target particles. The challenge is ensuring that SIMD is used appropriately for inverse square roots, and special functions, to fully utilise cache and CPU. - tree and admissibility condition should be separated from the data access method - i.e. no matter what these are, level-wise data access is always linear over contiguous blocks.

- Parallel model that allows for tunable cache utilisation - this means taking advantage of 'free lunches' from open-source software for high performance, relying on BLAS/LAPACK as much as possible as well as FFT if utilised. - SIMD special functions, and inverse roots - i.e. avoid parallel models where cache coherence could be destroyed for any reason.

- Removing as much as possible into the pre-computation step. - i.e. separate out required data into that evaluated at runtime, and that which can be stored and cached.

- Distributed memory - Foundational work by Warren and Salmon. - naively, due to global data dependency over each level, high levels of the tree are required across the entire domain. - BUT the amount of data is actually extreemly small, just the multipole coefficients for long range interactions. The P2P charge data is by definition extremely local, and can be handled effectively even with point to point communication patterns which are simple. - modern MPI implementations can do all-reduce style operations in log(P), talk a little about this. - MPI 3.0 and above introduce neighbourhood communication - Note that, when viewed hierarchically, data exchange is local in terms of morton keys. So maps well to hierarchical

neighbourhood collective communication patterns

- Software environment that allows us to mix and match all of the above, and can be a tool for both high performance simulation and algorithmic exploration.

At the end of this we have a skeleton for a high performacne FMM/hierarchical matrix

- Component diagram for high performance abstract FMM:

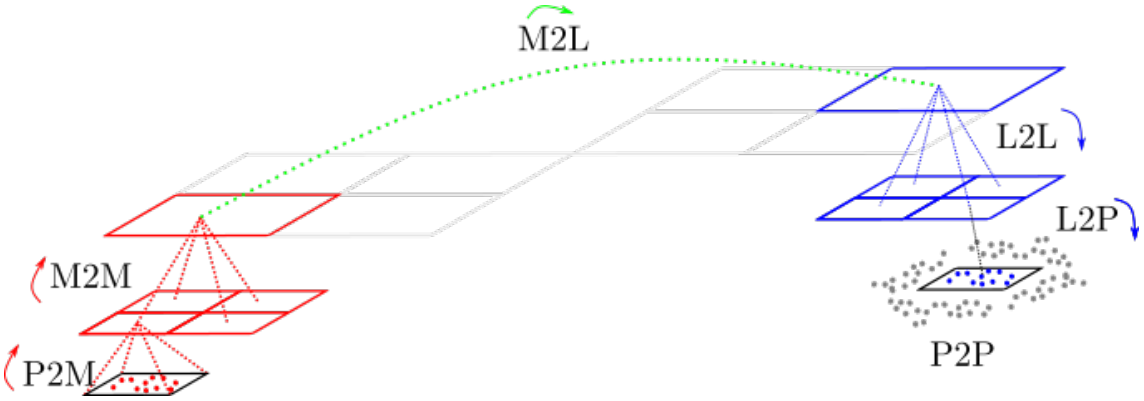

Figure 2.2: Data flow for the evaluation of the potentials due to a given set of (red) source particles in the far field of a given set of (blue) target particles, in two dimensions for clarity with a uniform tree of depth two. The source particles in the near field of the target box are shown in grey.

- Many past works have alluded to this feature of the FMM, though it is rarely expressed as such explicitly in the literature.

- Early examples include the works of Chandramowlishwaram and co-workers, who develop performance models characterising the kiFMM on various hardwares, and acknowledge the trade-off between the M2L and P2P as the key characteristic for FMM performance, and controlled by the tree depth.

- Deeper trees, synonymous with fewer particles per leaf node, and therefore smaller U-lists for P2P, but with more M2L.

- Shallower trees, synonymous with larger P2P and fewer M2L.

- Since then many works have focussed on expressing the data dependencies explicitly, and exposing them to special runtimes. - Proponent of task based runtime systems, as not restricted to specific algorithmic ordering of tasks removes artificial syncs, expose more native concurrency, and shorten critical path. The latter tends to restrict operations to a specific order. DAG-based dynamic runtime engines can remove artifactual synchronizations in the form of subroutine boundaries, remove

artifactual orderings in the form of pre-scheduled loops, expose the native concurrency, and shorten the critical path. StarPU, Charm++ and Legion being popular runtimes.

- However as we can see there are remarkably few intra-level synchrnoisations required for the FMM, meaning that the overhead of a runtime system in comparison to ordinary multithreading based appraoches

- The principal drawback of losing this control is that though NUMA aware approaches do exist, it is significantly hardware to control data locality, which i§s critical for performance on modern architectures.

- Furthermore, most critically, the two most expensive operations of the FMM, the P2P and M2L operators, *have no data dependencies* Meaning that there is amply opportunity to develop fully asynchronous implementations of these two operations, and given the optimal SIMD/SIMT structure of the P2P operation the obvious choice of operator to deploy to GPU in a heterogenous implementation.

2.3 Kernel Independent Fast Multipole Methods

As mentioned the functions A_p and B_p in (2.5) corresponded to explicit expansions of the Green's function in the original formulation of the FMM.

- This approach has benefits, principally it will have a very low pre-computation time for all the operators, and there have been translation operators developed for a wide range of common PDE kernels [CITE AVAILABLE KERNELS]. However, this approach also contains significant trade-offs. Very often kernel expansions rely on the evaluation of special functions for the translation operators, which can be expensive at runtime especially when large expansion orders are used.

- From a practical perspective, each kernel implementation may require kernel specific fine-tuning to achieve high performance, making a unified software framework that can tackle a range of kernels a daunting task.

- Variants of the FMM , known as 'kernel Independent', take an alternative approach to explicitly constructing kernel expansions. Instead of constructing explicit expansions, they use proxies to represent the field due to the charges contained in each box, their defining feature being that they result in schemes which only rely on kernel evaluations, while remaining compatible with a wide range of elliptic PDE kernels. Notable examples include [Darve paper], [Rokhlin and Martinsson paper in 1D].

- From a software perspective, this leaves a smaller surface area of code optimisation, data organisation for the application of operators and the kernel evaluations, which together determine the performance.

- We describe in detail the approach taken in [20] as it's the method which we implement in our software.

- The principal features of this approach are its usage of equivalent charges placed on surfaces that enclose the box, and the method of fundamental solutions as its approximation scheme.

- As the scheme relies on an analysis based uniqueness argument, the kiFMM and similar schemes with some analysis used in their specification are occasionally

referred to as 'semi-analytical' FMMs. Though this term garners scattered use through the literature, and it is relatively common to refer to any method which does not rely on explicit analytical expansions of the interaction kernel as an 'algebraic' method.

- Review all translation operators

- How is pinv constructed? What limits the accuracy of these factors in the original kiFMM, how was this rectified by kiFMM (pvFMM)

- Comment on storage requirements for Laplace and Helmholtz kernels in 3D.

- Comment on what can be precomputed and cached.

- Comment on specification of the operators as matrix-vector operations.

- Brief comment on M2L acceleration of the original spec of this method, ie. why was SVD based compression and BLAS abandoned and FFT favoured?

- Expected convergence behaviour, are the multipoles also exponentially converging with expansion order?

One of the big advantages of using outer and inner sphere approximations is the simplicity of the process of combining them. This is in contrast to the complicated formulas that must be used if the approximations are based on spherical harmonics.

2.4 Oscillatory Fast Multipole Methods

The crucial feature of the Laplace kernel (1.1) is the fact that far-field interactions (2.8) can be considered 'low rank', and therefore amenable to compression. Importantly for (1.1) the rank of a given interaction between two boxes is scale invariant, and only depends on their relative positions.

However, for problems described by the Helmholtz kernel,

$$K(\mathbf{x}-\mathbf{y}) = \begin{cases} \frac{i}{4}H_0^{(1)}(k|\mathbf{x}-\mathbf{y}|),\ d=2 \\ \frac{e^{ik|\mathbf{x}-\mathbf{y}|}}{4\pi|\mathbf{x}-\mathbf{y}|},\ d=3 \end{cases} \tag{2.9}$$

where d is the spatial dimension, $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$, k is the wavenumber, $H_0^{(1)}$ is the Hankel function of the first kind of order 0. The rank of interactions is no longer scale invariant, and indeed grows with box size. To see why this may be, consider the case for $d = 3$. From theorem 2.11 in [9], we can express the Helmholtz kernel as a separable series,

$$\frac{e^{ik|\mathbf{x}-\mathbf{y}|}}{4\pi|\mathbf{x}-\mathbf{y}|} = ik\sum_{p=0}^{\infty}\sum_{-p}^{p} h_p^{(1)}(k|\mathbf{x}|)Y_p^m(\frac{\mathbf{x}}{|\mathbf{x}|})j_p(k|\mathbf{y}|)\overline{Y_p^m(\frac{\mathbf{y}}{|\mathbf{y}|})} \tag{2.10}$$

Where k is the wavenumber, Y_p^m, for $m = -n, ..., p$ $p = 0, 1, ...$ are set of orthonormal spherical harmonics, and $|\mathbf{x}| > |\mathbf{y}|$, j_p is the spherical Bessel function of order p and $h_p^{(1)}$ is the spherical Hankel function of the first kind of order p.

In which case, an expression of the form (2.5) for the Helmholtz potential evaluated at a set of M target particles due to a set of N source particles

$$\phi(\mathbf{x}_i) \approx \sum_{p=1}^{P}\sum_{m=-p}^{p}\sum_{j=1}^{N} A_p(x_i)B_p(y_j)q(y_j), i = 1, ..., M \tag{2.11}$$

where we've truncated the expansion to P terms, known as the expansion order, and A and B are functions of the target and source particle positions only, respectively. We see that in this case for increasing expansion order, the number of terms in the sum grows quadratically. Though a demonstration is out of scope for this thesis, we mention that the number of terms P required to observe convergence in

the above sum is proportional to kD

$$P \approx kD \tag{2.12}$$

Some work for schemes that rely on the MFS is presented in [5]. Therefore, in the FMM for oscillatory problems interactions between boxes can be seen to have ranks growing quadratically, proportionally to $(kD)^2$, for increasing box size in 3D.

For schemes based on MFS as the kiFMM used in our software, the quadratic relationship between rank growth and box size is observed by noting that the condition for convergence (2.12) corresponds to a fixed number of points per square wavelength (in 3D), therefore the rank, which is proportional to the number of points used to discretise the equivalent surfaces, can be seen to grow quadratically with increasing box size.

- If one were to use an ordinary scheme for the FMM, this would correspond to a doubling of the expansion order with each level

In order to handle this growth in rank while retaining a fast algorithm, analytical approaches were introduced that rely on exponential expansions and diagonal forms

... Some more specifics here about what these enable, i.e. they don't change the rank, but in this representation which is relatively cheap to get to, the multipole to local translation operators are diagonal and therefore cheap to compute.

- Much of the algorithmic structure remains the same.

- The 'wide-band' FMM introduced by Cheng and co-workers, split of when and where to use each representation.

- complexity estimate for analytical high frequency FMMs

... There has also been some work [engquist and ying, second darve and messner paper] on extending kernel Independent methods to handle oscillatory problems, which have been demonstrated to achieve the same complexity.

- How do these work? Based on a cone-admissability condition.

- However, as we've seen modern architectures offer favourable environment for direct computations, and therefore we are able to get away with relatively shallow octree data structures. As the octrees are shallow, the compounding effect of rank

growth with decreasing level/increasing box size, is bounded. If one could vary the expansion order by level, in order to remain in the convergence regime, one could continue to use the ordinary kiFMM, however you would lose the $N \log N$ algorithmic scaling.

- We have an implementation that allows us to vary P by level, therefore can keep observed accuracy constant while increasing P as boxes get larger.

- Some kind of experiment showing the convergence graph with increasing P, in the geometric then convergence regimes.

- Scaling graph vs $\mathcal{O}(N)$ $\mathcal{O}(N \log N)$ - need to check. However, the actual complexity will increase quartically with k as growing number of terms with level,

e.g. M2M

Diameter at level l is

$$D_l = D_d 2^{d-l}$$

where d is the depth of the octree.

$$\text{Cost at level } l = O(N_l^2) = O((kD_l)^4)$$

Where the rank of the M2M matrices is N_l at level l and D_l is the box diameter at level l and k is the wave number

Written in terms of the diameter of the finest boxes at depth d,

$$\text{Cost at level } l \ = O((kD_d 2^{d-l})^4) = O(k^4 D_d^4 2^{4(d-l)})$$

$$\text{Total Cost } = O\left(k^4 D_d^4 2^{4d} \sum_{l=0}^{d} 2^{-4l}\right) \tag{2.13}$$

$$= O\left(k^4 D_d^4 2^{4d} \frac{16}{15}(1 - \frac{1}{16^{d+1}})\right) \tag{2.14}$$

Scales quartically with k and exponentially with depth d - which is used to balance P2P cost.

Consider that for $\sim N$ leaf boxes, the depth of the tree is given by $d = \log_8 N$,

$$d = \log_8 N = \frac{1}{3} \log_2 N$$

Substituting into the complexity estimate and ignoring small terms,

$$\text{Total Cost } = O\left(\frac{16}{15} k^4 D_d^4 N^{4/3}\right)$$

- For a fixed k, and D_d, we have an $O(N^{4/3})$ algorithm for computing all M2M, as other operators M2L and L2L will result in very similar complexity estimates as the M2L will only differ by a constant. So cost of FMM with this scheme of increasing expansion order by level is of $O(N^{4/3})$, but constants are determined by kD which scales quartically, so only manageable for relatively shallow trees and moderate wavenumbers.

- In terms of complexity not as good as specialised algorithm, however as the runtime performance will largely be driven by quality of data access schemes, and kernel evaluations, we consider to what extent the machinery of the kiFMM can be re-used for moderate frequency problems. How high can wavenumber be increased, before the lack of linear scaling leads to poor overall runtimes? We are able to test this as in our implementation we are able to tune expansion order by level. This is something we explore in Section 6.2.

2.5 Distributed Memory Fast Multipole Methods

Supercomputers are usually defined as massively parallel machines characterised by hundreds to thousands of individual nodes comprising of individual hardware pieces which communicate via a network. In this context memory movements, now between compute nodes, are of even greater practical relevance

- Network topologies, NVLink and other RDMA technologies that are emerging. - Rate of improvement in bandwidth/latency on network vs DRAM.

Minimising communication is crucial - Ibeid communication costs, and how these can be simplified.

- How are FMMs distributed for distributed memory?

- Focus is on the maximal reduction in communication.

- what communication can and cannot be avoided?

- How the local/global split in terms of tree gives rise to optimal communication scheme.

- What simplifying assumptions can we take for most pre-exascale systems?

- Avoid sorting of Morton keys/point data, the local/global split gives us a way to statically partition tree across available resources - simplifying assumption if work with ncpu = pow(8). - Not restricted to this, but makes threading simpler for local FMMs.

- Optimal implementation of MPI primitives for common data sizes.

- What will probably not work approaching exascale? - the gather operation over all processes required for multiple steps of this algorithm - ghost exchange, multipoles at local root on nominated processor. How can these problems be addressed? Do they even matter for the problem sizes we're concerned with?

- Bandwidth and latency complexity estimates for communication approaches - Bandwidth - total data transfer, transfer per process, scaling with PARFOR - latency - Number of communication steps, - scaling with P

2.6 Algorithm Zoo

Having described the key intuition behind the FMM for oscillatory and non-oscillatory kernels, as well as our variant of interest the kiFMM of Ying and co-workers. We use this section to describe a few of the vast numbers of variant approaches to compute the FMM, and unify the literature on FMM with that of the closely related *hierarchical matrices*, or $\mathcal{H}$ matrices for short.

The FMM and related methods principally vary with respect to their admissability criterion, method of field approximation, and underlying hierarchical data structure used.

Often FMMs and related methods are grouped roughly into three categories

Analytic, where kernel dependent expansions are used to approximate the fields. Original method, extended since then to a range of PDE kernels, including oscillatory problems described by time harmonic maxwell and helmholtz equations. For example originally, a multipole expansion was used. In 2D these take the form of simple coefficients, in 3D for Laplace already have to deal with spherical harmonic basis.

Semi-Analytic, examples include the kiFMM and the bbFMM, here only kernel evaluations are used however analytical properties are required in the construction of the fictitious surfaces on which the methods rely. examples include the kiFMM of [20] and also the black box FMM [12].

Algebraic, purely rely on kernel evaluation, Martinsson and Rokhlin 2011 [24]

- Analytical FMMs - different expansion representations, and their impact on complexity, table from yokota paper, - comment on complexity vs real implementation (special functions computation, for some kernels there are simplifying approaches e.g. Gumerov real Laplace expansions)

- Comment on lack of unified comparison in the literature

The line between 'algebraic' and 'semi-analytic' is fuzzy in practice, as the nature of the computations have an extremely close correspondence, and in the example of the kiFMM of [20] equivalent up to a choice of discretisation to the purely 'algebraic'

$\mathcal{H}^2$ matrix scheme.

When it comes to practical performance there is a gap in the literature in terms of a direct comparison between these rough approaches. Many of them achieve optimal asymptotic complexities, however practical performance depends principally on components that appear as constants in complexity estimates - related to memory accesses, and optimal vectorisation and parallelisation where possible.

Indeed, the FMM can be seen to be a special case of the more general hierarchical matrices, or $\mathcal{H}$ matrices for short.

- Generally considered that low-order analytical and high-order algebraic approaches, for performance however this is not actually known.

- analytical requires evaluation of special functions, what does this look like on modern architectures? Surely this is not a preferred operation, and will not be going in to the future.

Common $\mathcal{H}$ matrix formats are summarised in table [Ambikasaran table], indeed we see that the FMM is actually of class $\mathcal{H}^2$. Though often written about in different contexts, and by different geographically disparate communities, algebraic variants of the the FMM can be seen to be equivalent to methods for $\mathcal{H}^2$ matrices, the principal difference being that algebraic FMMs are commonly defined using quad/octrees whereas $\mathcal{H}^2$ matrices rely on the more generic 'cluster tree' approach that works directly with matrix entries.

- How are analytical expansions defined, 2D Laplace example, note and references on 3D laplace example. - private note on how these derivations are found.

- Note on admissibility, how it defines a broad class of FMM matrices, table of related matrix types with different rank structure and admissability criterion

- HSS, HBS, $\mathcal{H}$, $\mathcal{H}^2$ exactly where does the FMM fit in.

- What is the expected complexity of matrix vector product, how do they differ (nested vs non-nested bases).

- How are off diagonals approximated? Linear algebra. e.g. SVD/ACA/CUR decompositions vs analytical expansions.

- Relative costs of computing these, precomputation is expensive in general com-

pared to FMM which uses analytical expansions and minimal precomputation costs.

- Alternative to geometry captured by oct/quadtree. ORB could also be used for the FMM.

- But rely on cluster tree and cluster block trees over the matrix indices.

This is achieved with a hierarchical discretisation of the problem domain, often a *quadtree* in two dimensions and correspondingly an *octree* in three dimensions.

- trees define admissability from geometry for FMM, note on alternative approaches such as ORB.

- can control interaction list in different ways by using control parameter in ORB.

- Control of the interaction list is an optimisation used in stencil based approaches, more recent efforts like the Yesypenko paper, older approaches like Gumerov and the 8,4,2 method mentioned in the Yokota summary paper.

- These data structures are generated by creating a bounding box that covers the source and target particles, which without loss of generality may correspond to the same set. This box is then recursively sub-divided into *child boxes* of equal size.

- There is relatively little work directly contrasting the relative merits of different approaches for constructing FMMs. Yokota et. al [28] provide some analysis, attempting to bound the vast literature on FMM and related methods, however stop short of rigorous benchmarks. Indeed, direct software benchmarks between different groups and approaches can be flawed for numerous reasons, principally whether or not a given implementation was optimised for optimal hardware use or simply for demonstrative purposes. Furthermore, Yokota et. al only compare their analytical implementation of the FMM for Laplace kernels in 3D with an implementation for HSS matrices, which though of the same asymptotic complexity, have completely different scaling properties in 3D.

- A general rule of thumb has so far been that algebraic methods perform better at high order, meanwhile analytical methods are more suitable for low order. It's understandable to see where this point of view comes from. The translation operators for analytical FMMs take the form of very short sums, potentially involving special functions. The practical evaluation of which though highly optimised, are

inherently expensive in terms of FLOPs for high order evaluations. On the other hand, kernel independent approaches rely only on the evaluation of matrix blocks that are constructed via kernel evaluation, and therefore are easy to express as BLAS operations.

- As a point of direct comparison, we present the scaling on a single node of computing the Laplace problem in 3D on a range of modern softwares. We compute the problem on the surface of a sphere, taking increasingly fine discretisations of its surface, a sphere is chosen as a range of modern FMM and $\mathcal{H}^2$ matrix software is optimised for Boundary Element Method (BEM) applications

Alternative N-body approaches, there has been limited work except a landmark study [13], again the asmpyotic costs of these competing approaches is important

- Asymptotic costs of competing approaches include $\mathcal{O}(N \log N)$ for the FFT and $\mathcal{O}(N)$ for multigrid

- The FFT has been shown to have $\mathcal{O}(P^{1/d})$ communication complexity, with $\mathcal{O}(() \log P)$ for multigrid. Recently the FMM has also been shown to have $\mathcal{O}(\log P)$ communication complexity, and therefore the trade-offs between different approaches are significantly harder to contrast.

- FFT is generally preferred for problems with uniform resolution, multiscale features are somewhat better handled by the FMM and multigrid in theory. However, interpolation of non-uniform features can be handled efficiently with effective SIMD and caching optimisations.

- But the most important costs are related to data handling, and unique again to a specific implementation.

Table 2.1: Comparison of Expansion Types, adapted from [27] and expanded with more recent variants and estimates.

Approximation Scheme (+ M2L Scheme)	**Storage**	**Naive Arithmetic**
Cartesian Taylor	$\mathcal{O}(P^3)$	$\mathcal{O}(P^6)$
Cartesian Chebyshev (+ Direct Compression)	$\mathcal{O}(P^3)$	$\mathcal{O}(P^6)$
Spherical Harmonics	$\mathcal{O}(P^2)$	$\mathcal{O}(P^4)$
Spherical Harmonics (+ 'point and shoot')	$\mathcal{O}(P^2)$	$\mathcal{O}(P^3)$
Spherical Harmonics (+ FFT)	$\mathcal{O}(P^2)$	$\mathcal{O}(P^2 \log P)$
Planewave	$\mathcal{O}(P^2)$	$\mathcal{O}(P^3)$
Equivalent Charges	$\mathcal{O}(P^2)$	$\mathcal{O}(P^4)$
Equivalent Charges (+ FFT)	$\mathcal{O}(P^3)$	$\mathcal{O}(P^3 \log P)$
Equivalent Charges (+ Direct Compression)	$\mathcal{O}(P^2)$	$\mathcal{O}(k^2 \cdot P^2)$

2.7 Available Software

Despite the intensity of developments over the last decade, the software landscape is fragmented for FMMs and related methods.

- What software exists, and which approaches do they use?

The lack of re-usable subcomponents slows down algorithmic innovation. For example, there are numerous implementations of SIMD vectorised Green's functions, community software building has been poor.

We show the performance of some of the main implementations below for Laplace and Helmholtz.

Additionally, implementations are designed to demonstrate the performance of specific approaches and algorithms. It is exceptionally hard to swap algorithmic approaches, hardware and software backends. For example ExaFMM-T / ExaFMM are re-implementations of the entire FMM algorithm, where much of the underlying machinery for algorithm deployment is identical - it's just the metadata required for the operators and the approximation scheme which is different. Yet these are both long, overlapping, and complex libraries.

- No software aims to make any guarantee about performance, but neither do they expose subcomponents to users. At least one fo these should be done so that community efforts can begin in earnest, and software lives beyond a PhD research project.

To the best of our knowledge, no other open-source FMM software have the ability to vary expansion order by level.

High Performance Field Translations for the kiFMM

The discussion in this chapter, including figures and diagrams, is adapted from the material first presented in [18]

3.1 The Multipole to Local Translation (M2L)

- From chandrowlishwaram 2010 to now, the M2L has become the key bottleneck in terms of optimising kiFMM implementations. Cost of DRAM access hasn't scaled as quickly as available flops.

- Note here on the computational structure of the M2L problem. How it's poorly matched to modern CPU/GPU.

- We've already observed the M2L operator to be of convolution type, and therefore amenable to FFT acceleration if using regular grids like in the kiFMM.

- This has optimal complexity, but the low arithmetic intensity of the internal Hadamard product is difficult to optimise out.

- We postulate that direct matrix compression techniques, with specially designed hardware features that optimise for BLAS, as well as randomised methods for matrix compression - reducing pre-computation time, can result in highly competitive runtimes. Very high arithmetic intensity

- Introduce this subchapter

3.1.1 Literature Review

- use this section to introduce idea of transfer vectors, reflection and rotational symmetry

- Full literature review of past approaches

- Dense and Analytical approaches

- The historical push for this originally resulted in point and shoot/diagonal forms ('new' FMM paper)

- Where past efforts have been focussed, and why? (Original paper dismissed direct matrix compression)

- how this is achieved in practice (i.e. what computations are needed, not the implementation details)

- Explanation of the FFT method, and why it was able to achieve high performance.

- Why this may not be completely appropriate, low arithmetic intensity (maybe estimate?)

- How PVFMM makes it work, with very high arithmetic intensities, and special structure.

- Some criticism here of that approach.

- Require a special implementation for each architecture, intricate to maintain, requires passing mutable pointers over threads. Difficult to replicate, ours is the only re-implementation of this scheme in the open-source.

- Give the gist here, the actual details can be shoved in the appendix as it's not really a part of the discussion.

- Numerical compressoin of low rank blocks, approaches - ie. how is SVD handled for aspect ratio - randomised SVD - estimating cutoff rank - power iterations, and practical considerations - multithreading, where to use rSVD, limitation due to internal QR required, potentially alternative schemes - krylov-schur

- Alternative recompression scheme based on QR

$$A = BC^T, \text{ with } B \in \mathbb{R}^{m\times K}, C \in \mathbb{R}^{n\times K} \tag{3.1}$$

where $K > k$ target rank, but still much smaller than m, n.

1. Compute Economic QR (cheaper than det SVD) $B = Q_B R_B$, $C = Q_C R_C$.
2. Compute truncated SVD $T_k(R_B, R_C^T) = \tilde{U}_k \Sigma_k \tilde{V}_k$
3. Set $U_k = Q_B \tilde{U}_k$, $V_k = Q_C \tilde{V}_k$, return $T_k(A) := U_k \Sigma_k V_k^T$

Complexity is $O((m+n)K^2)$, but smaller constant than SVD.

When A does not have rank k but can be well approximated by a rank - k matrix, it is advisable to choose the oversampling parameter p larger than 0 in the rSVD (alg 2 in Kressner review). The following result shows that the resulting error will not be far away from the best approximation error $sigma\_k + 1$ in expectation, with resepct to the the random matrix Ω,

- Theorem, Let $k \geq 2$ and $p \geq 2$ be chosen such that $k + 1 \leq \min\{m, n\}$. Then the rank-k approximation $\tilde{A}$ satisfies

$$\mathbb{E}\|A - \tilde{A}\| \leq \left(2 + \frac{4\sqrt{(k+p)\min\{m,n\}}}{p-1}\right)\sigma_{k+1} \tag{3.2}$$

The bound improves significantly when performing a few steps of subspace iteration after Step 2 in Algorithm2, which requires a few additional block matrix-vector multiplications with AT and A. In practice, this may not be needed. As the following example shows, the observed approximation error is much better than predicted by Theorem above

Can here demonstrate compression of Laplace operator with different oversampling paremeters and observe how close erorr is to σ_{k+1}, use this to justify lack of power iterations.

- Need Singular value distributions for Helmholtz kernel to justify the parameters for compression, that investigation needs to be in this section. Similar to the Darve

paper, i.e. to justify the approximate rank for the rSVD.

- Why might this be preferred, or advantageous, what are its constraints

- structure of modern CPU and GPU

- emerging CPU architectures with specialised units for matrix multiplication - examples of CPUs, apple M series, Qualcomm snapdragon

- implications for matmuls on GPU (low-precision)

- Using numerical compression schemes for low-rank blocks has a long history in the H matrix community - ACA, 'adaptive' expansion order schemes. - can be seen to correspond to 'variable expansion order FMM' schemes of the 90s/00s.

3.1.2 A New Direct Compression Based Acceleration Scheme

- Pioneering work by messner et. al. took these schemes and specifically focussed on computational aspects, the SVD is an expensive algorithm, worked on re-ordering the computation to reduce expense - we build on this - Furthermore they worked on re-organising application to improve caching, and our approach extends this.

- Precomputations required for Laplace and Helmholtz, required storage. - Why 'storage' is perhaps irrelevant - shallow trees, and anyways data movement rather than storage is critical.

- Approaches for BLAS based field translation in some more detail than in the paper.

- Essentially, we extend the idea of Messner et. al by completely unrolling the M2L loop via precomputatioan of critical metadata.

- Explanation of what metadata is required, what we mean by unrolling.

- Demonstrate how this is actually a general variant of what's been done in other approaches.

- This specification is very suggestive of an approach that relies on linear data structures and preserves cache-coherence.

- Suggests that a simple method using BLAS and multithreading can be highly effective, in contrast to the runtime systems experimented with in the past decades, simply because the cache structure of these computations is simple, and runtimes

may destroy this.

- Algorithm itself, take from paper

- Caching experiment vs ScalFMM. Why software comparisons can be contrived, due to the vast differences in implementation details -e.g. kernel evaluations, but can directly compare the M2L runtimes alone for different expansion orders and tree levels. - cache destroyed by granular tasking approach

- The numerical results from the paper for Precomputations as well as M2L application cost. Need the same results for Helmholtz, but first need to find optimal parameters.

- Comment and discussion from the paper can be lifted here.

- Interesting point of comparison with ScalFMM to demonstrate the importance of caching to performance, noting that direct software comparisons are not entirely fair, but we are relying on same compiler version, threading model, and BLAS versions. The only difference being the organisation of the computation.

- Time just the M2L for ScalFMM, multithreading enabled, on threadripper. Single and double precision if possible, at different expansion orders, for increasing tree depth,

Kressner discussion on approaches for low rank matrix factorisations (non-hierarchical)

- Martin Stoll Krylov schur - Levitt and Martinsson Linear Complexity Black Box Compression - Halko, randomised linear algebra - ACA - SVD - Other methods from paper.

3.2 Leaf Level Operators (P2P, L2P, P2M)

- NOTE this is work taken from the green-kernels library, implemented within our group for green kernel evaluation. - 6k loc. - specialised for a couple of instruction sets + generic autovectorised implementation. - How do the SIMD implementations work? Newton steps + fast inverse square root. - sleef for special functions

From FMM side - how are we blocking targets over threads? - what are the implications of ultrafast direct calculations on CPU, and potential for even faster on GPU (low-precision). - i.e. majority of computation can be handled directly

quite effectively. Minimise M2L, and makes direct matrix compression techniques more attractive. Could handle these rapidly on the CPU, and especially in H100-like systems with UMA, offload expensive direct computation to GPU. - how could we translate our current codes to a GPU for P2P, what are the trade-offs? Would memory transfer mean that it's never worth it?

3.3 Parent to Child Operators (M2M, L2L)

- Formulation as BLAS3 - specifically I want to show the exact way that this is done - i.e. using morton-like (at the level of a level) encoding to lookup siblings/sets of siblings at once and apply BLAS3. - block sizes determined heuristically for a given architecture, but could in principle be estimate from available L2/L3 cache sizes. - storage, i.e. relative between parent and child, especially note that extra required if different expansion order taken per level.

3.4 Field Translation in a Distributed Setting

- Communication intensive MPI FMM phases are related to ghost data communication for the M2L and P2P (All other operators are inherently local).

- Ibeid et. al. provide estimates of communication complexity based on halos of both of these operators

- From Ibeid, add a derivation.

- How are these achieved in practice? Global/Local split, introduced by Abduljabbar and Yokota.

- Suggestive of hierarchical communication pattern.

- However, this is perhaps over-complicated.

- Domains of each processor are known via all-reduce. Therefore a-priori know exactly where all elements of locally contained interaction lists lie at problem setup.

- Given massive core-count, and relatively small total node count on pre-exascale systems like Archer 2 which are likely to persist. Global/local split can be further simplified - just choose a number of nominated nodes (even just one) for the global

tree computation. Instead of hierarchical exchange, calculate required data exchange as a precomputation step, and can then tune chunk-wise data exchange over all data using neighbourhood communicators. Much simpler to setup than multiple function calls at each level of the hierarchy, and allows for finer tuning of data exchange.

- Note, each AMD Epyc node on Archer 2 can easily handle problem sizes O(1e7) points in 0.5s in double precision.

Modern Programming Environments for Science

The discussion in this chapter, including figures and diagrams, is adapted from the material first presented in [19].

4.1 Requirements for Research Software

- Requirements and constraints on research software development.

- As an example FMM softwares used in recent benchmark studies (ExaFMM variants, PVFMM) have been constructed during the course of doctoral or post-doctoral projects. This entails a significant 'key man' risk, in which when the project owner completes their course of research the project enters a decay state and is no longer actively maintained and developed. New developers, unfamiliar with the code bases which can grow to thousands of lines of code, and often written without reference to standard software engineering paradigms for designing and managing large code bases (continuous integration, software diagrams, and simple decoupled interfaces) will find it challenging to build upon existing advances, and resort to developing new code-bases from scratch, rediscovering implementation details that are often critical in achieving practical performance.

- In seeking to avoid this cycle we envisioned a project built in Python, which maximises the maintainability of a project due to its simple syntax and language construction. New developers who, as a standard, are often educated in Python in the natural sciences and engineering, will hopefully be familiar with the language in

order to gain productivity as fast as possible. However, as we demonstrated in our paper ... This itself imposes significant constraints on performance, which is balanced by the 'usability' of the language, making it just as challenging as developing a complex code in a compile language.

- Modern compiled languages offer tools that enable developer productivity. Examples include Go, Rust, ...

- The complexity of methods leads to complex code surface areas which are difficult to maintain especially in an academic setting with few resources for professional software engineering practice.

- The diversity of hardware and software backends leads to increasing difficulty for projects to experiment with and incorporate computational advances.

- Hardware and software complexity, and gap between a one-off coding project and extensible maintainable software tooling.

- Review developments in computer hardware and software that make this easier to be more productive, but also more challenging to wrap together over time.

- Emerging and future trends, exemplified by the step change in compiled langauges in the new generation and the interest in Rust and similar langauges. The mojo project and what this says about the future.

4.2 Low Level or High Level? Balancing Simplicity with Performance

- Summary of Python paper results, in summary complex algorithms necessitate complex code in order to achieve performance - specifically the requirement for programmers to be in charge of memory and for hot sections manually vectorise etc. Writing everything in a high-level language obfuscates the application code from the sections critical to performance

- Review of why this was thought to be a good idea, and why it might be worth trying again in the future.

- What problems does this paper address, wrt to the literature?

- Brief review of motivation and reasoning behind Rust, and which features we take advantage of

- Build system, paragraph explaining it and contrast with build systems wrt to competitors.

- Trait system, though shares similarities with C++21 concepts, a part of the type systems - organisation of shared behaviour without loss of performance, bottom up organisation.

- Why the two language problem isn't really a problem with modern compiled langauges, note on the succesful projects that manage this and how effective just writing C interfaces can be, if using another language.

Software Design

5.1 Data Oriented Design with Rust Traits

- Motivation, and review, DOD book.

- How do traits enable data oriented design.

- Overview of the design of the software.

- Why is this good for the future? Well, it leaves open extension to other approaches for any individual subcomponent.

- An example of this is the genericity over data type, kernel implementation, and field translation method, with a space for the kind of tree data structure.

- Exactly how is decoupling achieved with trait interfaces? - decoupling of implementation from abstraction.

5.2 FMM Software As A Framework

- Want to encourage as much code re-use as possible, and minimal structural rewrites.

- The re-implementation of critical subcomponents should be avoided. A step towards this is the development low-level C interfaces which enable the construction of higher level interfaces in compatible languages.

- We've made a start to this with a low-level interface to the principal API of the FMM software.

- We also want to be able to deploy on as wide a range of target hardware as possible, and leave open extension to future systems, enabled by design, referencing diagram.

- High level diagram of how software components fit together

- Code generation for multiple targets enabled by Rust's llvm based compiler.

- C ABI as a compatiblity layer to other projects, success with this in developing Python wrappers and integration with NGBEM

- Flexible backends enabled by RLST package for BLAS and Lapack.

We discuss the features of our software by examining three important test-cases, that are critical to its performance.

5.3 Case Study: A Trait Based M2L

- How do metadata computations work for a configurable M2L implementation? What does the high-level framework expect of an M2L implementation?

- What does this look like in practice? Exactly what traits are there, how can re-implement them for an alternative M2L implementation?

This is incredibly compelling for an FMM software, as it can serve as a testbed for extension and algorithmic experimentation as well as comparison. E.g. we could attempt to use our framework to directly compare analytical and kiFMM methods, re-using the same kernel/lapack/blas backends, tree data structure, the only difference would be the translation operator implementations allowing for a fair comparison.

- In principal, how could one also implement a new FMM, a new tree or a new operator for e.g. GPU?

5.4 Case Study: High Performance Trees

- Exactly how are Morton encodings done, and what are the drawbacks and alternatives. Hilbert encodings, ORB. How much difference do any of these things make?

- Tree Construction approach and algorithms - Morton encoding via lookup tables - neighbour finding - interaction list construction (fast)

- What did we end up doing, and what is the justification for these being good enough. - weakly adaptive vs fully adaptive FMM. - why?

Important implementation details - construction of interaction lists, neighbour finding. - construction of Morton encodings. - rapid data access, lookup tables/index pointers - trade-offs of approach in shared and distributed memory - e.g. adaptive vs weakly adaptive trees. - problems with load balancing approach etc

- How am I storing key data? I'm not using a pure Z order in the data layout, I'm storing by level in Morton order for ease of lookup of contiguous sibling data

- How are multi/single node trees designed?

- How are ghosts handled in distributed setting? How could this be further optimised via blocking algorithms on very large systems.

- Compare to ghost handling of competing softwares, why this is so much simpler and often good enough.

- basically, have a very shallow struct, with trait interfaces that define the trees/fmm trees. This means that the actual tree is incredibly abstract, and flexible. Being able to query it like a single node tree means that kernel code is largely unchanged for operators.

5.5 Case Study: FMM Metadata

- Arguably the most important part of setting up the calculation to be fast is calculating metadata effectively, i.e. need to move as much of the work away from the runtime as possible.

- Most important pieces here are figuring out how the interaction lists correspond to runtime data structures in M2L.

- Metadata for ghosts, some have to be done at runtime.

Numerical Experiments

6.1 Laplace

6.1.1 Single Node

6.1.2 Multi Node

- Weak scaling, communication vs computation time breakdown.

    - Load balance discussion, does it matter for the distributions tested?

    - Discussion on impacts of bandwidth and latency, and potential for async.

    - Trade-off sort methods.

6.2 Helmholtz

6.2.1 Single Node

Simple benchmark for low k

- A graph of p vs level for each given accuracy, e.g. pick a few in single and double, likely to be easiest and most appropriate for single precision, and low double precision.

- Big colorful plot of HF helmholtz, use same parameters as in the Lexing Ying/Engquist paper, can't directly compare runtimes - but these are considered high frequencies in 3D.

- Comment, with optimal K evaluations appropriately using SIMD, relatively shallow trees, especially in single precision, can model very high frequency prob-

lems in practically useful runtimes, though perhaps (need to check) lose asymptotic scaling.

Conclusion

In this thesis we have presented progress on the development and design of a software framework for kernel independent Fast Multipole methods. We've documented outputs towards the broader goal of a sustainable framework which can be extended, with re-usable subcomponents. This research performed necessitated a significant investigation into the optimal programming environment for high-performance scientific computing that enabled high productivity within the constraints of academic software development. The resultant software enabled a new investigation of the critical M2L field translation operation, a key bottleneck in the kiFMM algorithm, and the development of a highly competitive approach well suited to emerging trends in computer hardware.

Due to the high-performance the FMM operator kernels for both the Laplace and low-frequency Helmholtz kernels demonstrated in this work as well as the creation of trees, we are in a good position to extend our software to a distributed setting. The decoupling of operator kernels from their implementation via the design of our software also enables future extensions to a heterogenous platforms in which batch BLAS for the M2L, and SIMT for the P2P operations

- Direction of travel of hardware, how this could effect exactly how the algorithm is set up

- specifically handling as much as possible on a GPU, data organisation in a unified memory context will be cheap.

List of Acronyms

BEM Boundary Element Method. 28

BLAS Basic Linear Algebra Subprograms. v, 6, 28

CLP Core Level Parallelism. 2

CPU Central Processing Unit. v, 2–4

DLP Data Level Parallelism. 2

FFT Fast Fourier Transform. v, 5, 28

FLOP Floating Point Operation. 6, 28

FMM Fast Multipole Method. v, 1–6, 8–11, 21, 23, 25–28, 30, 46

GPU Graphics Processing Unit. 2, 3

HPC High Performance Computing. v, 3, 4, 7

kiFMM kernel independent Fast Multipole Method. v, 5–7, 21, 23, 25, 46

L2L Local to Local. 23

M2L Multipole to Local. v, 5, 23, 46

M2M Multipole to Multipole. 22, 23

MFS Method of Fundamental Solutions. 21

MIMD Multiple Instruction Multiple Data. 2

PDE Partial Differential Equation. v, 4

SIMD Single Instruction Multiple Data. 2, 3, 28, 30

SIMT Single Instruction Multiple Threads. 2, 3

SISD Single Instruction Single Data. 2

TLP Thread Level Parallelism. 2

Appendix

Bibliography

[1] Emmanuel Agullo et al. “Task-based FMM for multicore architectures”. In: *SIAM Journal on Scientific Computing* 36.1 (2014), pp. C66–C93.

[2] Sivaram Ambikasaran et al. “Fast direct methods for Gaussian processes and the analysis of NASA Kepler mission data”. In: *arXiv preprint arXiv:1403.6015* (2014).

[3] Krste Asanovic et al. “The landscape of parallel computing research: A view from berkeley”. In: (2006).

[4] Lorena A Barba and Rio Yokota. “ExaFMM: An open source library for Fast Multipole Methods aimed towards Exascale systems”. In: *Boston: Boston University. Retrieved from barbagroup: http://barbagroup. bu. Edu* (2011).

[5] Alex H Barnett and Timo Betcke. “Stability and convergence of the method of fundamental solutions for Helmholtz problems on analytic domains”. In: *Journal of Computational Physics* 227.14 (2008), pp. 7003–7026.

[6] Gordon Bell et al. “A look back on 30 years of the Gordon Bell Prize”. In: *The International Journal of High Performance Computing Applications* 31.6 (2017), pp. 469–484.

[7] Bérenger Bramas. “TBFMM: A C++ generic and parallel fast multipole method library”. In: *Journal of Open Source Software* 5.56 (2020), p. 2444.

[8] Barry A Cipra. “The best of the 20th century: Editors name top 10 algorithms”. In: *SIAM news* 33.4 (2000), pp. 1–2.

[9] D Colton. *Inverse Acoustic and Electromagnetic Scattering Theory*. 1998.

[10] Eric Darve and Pascal Havé. "A fast multipole method for Maxwell equations stable at all frequencies". In: *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 362.1816 (2004), pp. 603–628.

[11] Jack Dongarra et al. "With extreme computing, the rules have changed". In: *Computing in Science & Engineering* 19.3 (2017), pp. 52–62.

[12] William Fong and Eric Darve. "The black-box fast multipole method". In: *Journal of Computational Physics* 228.23 (2009), pp. 8712–8725.

[13] Amir Gholami et al. "FFT, FMM, or multigrid? A comparative study of state-of-the-art Poisson solvers for uniform and nonuniform grids in the unit cube". In: *SIAM Journal on Scientific Computing* 38.3 (2016), pp. C280–C306.

[14] Alexander Gray and Andrew Moore. "N-Body'problems in statistical learning". In: *Advances in neural information processing systems* 13 (2000).

[15] Leslie Greengard and Vladimir Rokhlin. "A fast algorithm for particle simulations". In: *Journal of computational physics* 73.2 (1987), pp. 325–348.

[16] Sijia Hao, Per-Gunnar Martinsson, and Patrick Young. "An efficient and highly accurate solver for multi-body acoustic scattering problems involving rotationally symmetric scatterers". In: *Computers & Mathematics with Applications* 69.4 (2015), pp. 304–318.

[17] Srinath Kailasa. "kifmm-rs: A Kernel-Independent Fast Multipole Framework in Rust". Submitted to Journal of Open Source Software. 2024.

[18] Srinath Kailasa, Timo Betcke, and Sarah El Kazdadi. *M2L Translation Operators for Kernel Independent Fast Multipole Methods on Modern Architectures.* 2024. arXiv: `2408.07436 [cs.CE]`. URL: `https://arxiv.org/abs/2408.07436`.

[19] Srinath Kailasa et al. "PyExaFMM: an exercise in designing high-performance software with Python and Numba". In: *Computing in Science & Engineering* 24.5 (2022), pp. 77–84.

[20] 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.

[21] Judith Yue Li et al. "A Kalman filter powered by-matrices for quasi-continuous data assimilation problems". In: *Water Resources Research* 50.5 (2014), pp. 3734–3749.

[22] Dhairya Malhotra and George Biros. "PVFMM: A parallel kernel independent FMM for particle and volume potentials". In: *Communications in Computational Physics* 18.3 (2015), pp. 808–830.

[23] William B March and George Biros. "Far-field compression for fast kernel summation methods in high dimensions". In: *Applied and Computational Harmonic Analysis* 43.1 (2017), pp. 39–75.

[24] Per-Gunnar Martinsson and Vladimir Rokhlin. "An accelerated kernel-independent fast multipole method in one dimension". In: *SIAM Journal on Scientific Computing* 29.3 (2007), pp. 1160–1178.

[25] Tingyu Wang, Rio Yokota, and Lorena A Barba. "ExaFMM: a high-performance fast multipole method library with C++ and Python interfaces". In: *Journal of Open Source Software* 6.61 (2021), p. 3145.

[26] Tingyu Wang et al. "High-productivity, high-performance workflow for virus-scale electrostatic simulations with Bempp-Exafmm". In: *arXiv preprint arXiv:2103.01048* (2021).

[27] R. Yokota. "An FMM based on dual tree traversal for many-core architectures". In: *Journal of Algorithms and Computational Technology* 7.3 (2013), pp. 301–324. ISSN: 17483018. DOI: 10.1260/1748-3018.7.3.301. arXiv: 1209.3516.

[28] Rio Yokota, Huda Ibeid, and David Keyes. "Fast multipole method as a matrix-free hierarchical low-rank approximation". In: *International Workshop*

on Eigenvalue Problems: Algorithms, Software and Applications in Petascale Computing. Springer. 2015, pp. 267–286.

[29] Rio Yokota et al. “Biomolecular electrostatics using a fast multipole BEM on up to 512 GPUs and a billion unknowns”. In: *Computer Physics Communications* 182.6 (2011), pp. 1272–1283.