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

September, 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: 2

    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Unpublished Manuscripts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: 1, 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: 4

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 (**??**). 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 decades since the algorithm's first introduction.

Since the end of Dennard Scaling[1] in the 2000s, hardware design and development has focussed 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 (**??**) 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.

Software efforts for FMMs were a particular focus of research activity in the 2010s, with particularly prominent examples being the ExaFMM project [4, 26],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 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

[2]We demonstrate this with specific benchmarks in Section **??** for a selection of CPU architectures.

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, 27]. 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 software 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 the $\mathcal{H}$ and $\mathcal{H}^2$ matrix approaches, 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.

In Chapter 4 we present an investigation into the requirements of a programming platform for scientific software. 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 and radiply developing features that enable high-performance, adapting material first presented in [19].

In Chapter 3 we present a rigorous application of our software, in which we investigate optimisations for a crucial Multipole to Local (M2L) operation for kiFMMs, a critical bottleneck for FMMs in general due to its requirement for significant data re-organisation due to the implicit non-contiguous memory accesses required by this operation. This operation is of convolution type, and current state of the art approaches are based on Fast Fourier Transforms (FFTs) with an $\mathcal{O}(N \log N)$ complexity bound, requiring careful explicit vector programming to achieve practical performance. We find that despite this 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 of the approach, 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 material in this chapter is largely adapted from that first presented in [18].

Chapter 5 describes in detail the engineering approach of our software, particularly the employment of Rust's 'trait' system, as well as specific implementation details of the kiFMMs operators, and data structures some of which is adapted from its first presentation in [17]. Chapter **??** describes the extension of our software to distributed memory systems, and details communication reducing schemes for exchanging ghost information. Chapter 6 contains numerical experiments with our software in a single node as well as HPC setting, including a study of the applicability of our software to oscillatory problems described by the time independent Helmholtz equation at low to moderate wavenumbers. 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,

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

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

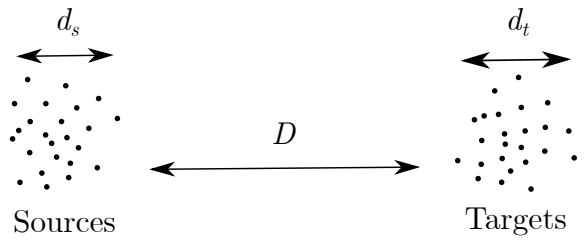

Figure 2.1: A set of source and target particle cluster, where the width of each cluster is significantly less than the distance separating them, $d_s, d_t \ll D$.

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.

- The full algorithm for uniform case

- sketch of complexity for full generic algorithm

- adaptive case, and why this might be useful.

- What extras does it entail. Will we consider it in this thesis?

- Define interaction lists.

- Define all operators.

- Give an algorithm sketch.

- Give complexity sketch.

2.2 Computational Structure

Figure 2.2 illustrates the dataflow in the FMM when evaluating the potentials due to a set of source particles at a set of target particles in its far-field. Shown this way, it's easy to see how the spatial locality of required boxes for during the P2M and M2M steps during the upward pass, and the L2L, L2P and P2P steps during the downward pass could be implemented using contiguous buffers for spatially local octant data. Indeed there are many established encodings for 'space filling curves', which can be used to encode the indices of nodes in a hierarchical tree, common examples including so called Hilbert and Morton curves [25]. The power of this, in combination with an octree data structure is that it allows for extremely efficient lookups for node data, and that of nodes adjacent to a given node, as geometric information is tied to memory layout.

In terms of the tree data structure, the leaf level operators: L2P P2M and P2P are embarrassingly parallel over each leaf. As would be the M2L for an adaptive FMM. In a single node setting on a CPU these operators can effectively handled with a combination of multi-threading and efficient vectorisation via SIMD, and similarly easy to translate into a GPU implementation. However, in this case the small data sizes involved in the the L2P, P2M and M2L for an adaptive FMM, may make GPU implementations redundant due to the memory transfer costs outweighing the increased throughput. This is enabled by the high arithmetic-intensity of leaf-level operators that involve the particle data. Consider the P2P

... P2P arithmetic intensity. It is compute bound, and scales very well with direction of processor technology.

As we have seen, it is possible to form the M2M and L2L operators as matrix-vector products for the kiFMM, and also its variants [CITE FORTIN, bbFMM]. Indeed, as these operations depend on the relative position between two boxes, we can effectively block these operations as matrix-matrix products of sets of boxes at a given level, usually siblings as their coefficient data can effectively be stored contiguously in memory alongside a localtional encoding.

... Matrix multiplication is a heavily optimised routine in both software and hardware - BLIS, gotoblas etc. - special registers in Apple/new AMD/qualcomm devices due to AI

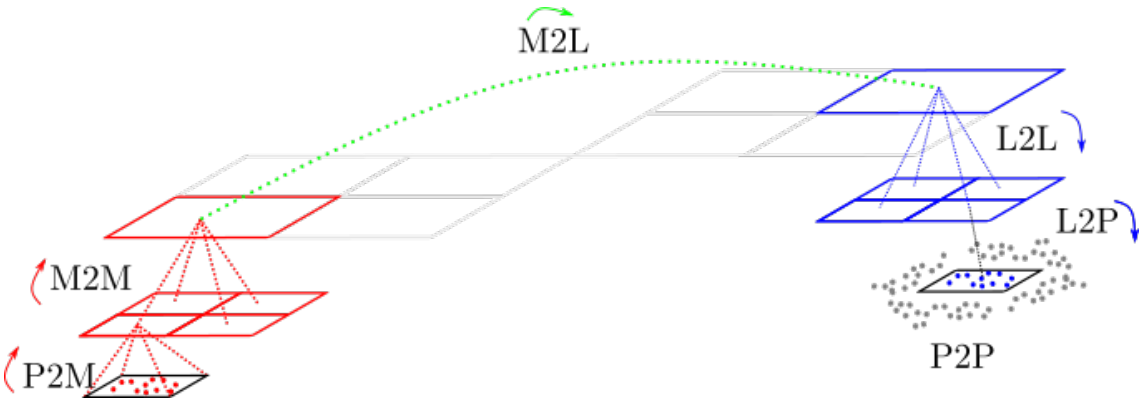

Figure 2.2: Data flow for the evaluation of the potentials due to a given set of (red) source particles in the far field of a given set of (blue) target particles, in 2D for clarity with a uniform tree of depth 2. The source particles in the near field of the target box are shown in grey.

However, the power of the FMM and its derived asymptotic complexity come from capturing long-range interactions via the M2L field translations. This naturally leads to non-contiguous memory accesses during this step, as the M2L operator is only defined for non-adjacent nodes. Indeed, despite the $O(N)$ complexity, efficiently organising data for this step alone can define the runtime of a practical implementation. This is of even greater importance in 3D, where the sizes of interaction lists are larger, as well as for oscillatory problems where the rank for a given interaction grows with box diameter. Performing efficient M2L requires a careful balance between efficiently handling the interaction lists to ensure contiguous data access, as well as algorithmic optimisations to reduce the complexity of this operator.

The expense of the FMM can therefore be considered a trade-off between compute-bound leaf level operators, and the memory bound M2L operator.

... Chandramowlishwaram result from ages ago, have things gotten worse since then for M2L?

The FMM offers further opportunities for parallelism. The two most expensive parts of the FMM, the P2P and M2L are independent from each other, and can be performed asynchronously. This is the basis of approaches which offload one of these operators, usually the P2P, to a co-processor while performing the recursive

loop.

Given two non-intersecting boxes at level l - If their interaction lists do not overlap, they can proceed independently of one another during the downward pass - Even if their interaction lists do overlap, if the appropriate metadata structure required for the M2L is setup - i.e. where to look for multipole data, perhaps a re-allocation to get data contiguous for lookup. They can still proceed independently of each other. - This has remarkable consequences for parallelising distributed FMMs. Early approaches suffered from a reliance on a global all-reduce, stemming from the fact that nodes near the root are required by almost all boxes during the downward pass. - However, as first noted by Ibeid et. al. There is (in 3D) at least an 8 fold redundancy if data is partitioned across MPI ranks such that each rank contains roots that correspond to leaves of a 'global' tree. - This means, that the FMM below the depth of the global tree, can proceed completely independently on each rank, and the communication required for the global tree is only of multipole and local coefficient data which is inherently small as only a few ranks are involved in the communication. - The second issue in distributed memory, aside from ghost communication, is achieving an appropriate load balance among processors. This is addressed in a multitude of ways, commonly by creating a cost function that is proportional to the size of each box's interaction list, followed by a repartition of the data. As the initial partition is achieved via parallel sort, the appropriate load-balanced partition again relies on a parallel sort, for which numerous communication optimal implementations are readily available.

As we can see the most significant bottlenecks in FMM implementations are related to memory transfer both vertically through the cache-hierarchy on a specific device, and horizontally among distributed devices. Specifically, the data organisation required for performing the M2L, and the communication of ghost information in a distributed setting. The remainder of the algorithm can be very efficiently parallelised with established approaches, as long as care is taken to keep the computational intensity (i.e. number of effective flops) as high as possible.

The FMM also exposes significant opportunities for deployment via asynchronous

execution and task based runtimes, as the most time consuming sections of the algorithm can be evaluated independently of each other. Additionally, during the recursive loop, as long as appropriate metadata is created, all operations are trivially parallel over each box at a given level.

2.3 Kernel Independent Fast Multipole Methods

The decades since its original presentation have seen the FMM extended with related ideas which principally differ in how they represent interactions between distant clusters of particles typically referred to as *algebraic* FMMs.

- Review of the KiFMM and variants. Black Box FMM, Analytical FMM, Data Driven Techniques.

- Motivation for use from a software engineering and computational performance perspective.

- Approximation scheme via method of fundamental solutions, why do we expect certain convergence?

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.

Previous approaches to handle this for analytical FMMs can be technically complex

However, there has been some significant results in developing 'kernel independent' approaches for oscillatory problems. The resulting schemes closely mirror the kiFMM

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

Aside from the 'kernel independent' and 'analytical' FMMs presented, the FMM exists in a universe of closely related *hierarchical* matrix methods, variants principally differ in their approximation scheme for the field, the underlying hierarchical data structure used - i.e whether or not an quad/octree is used which relies on gemoetric information, or one works directly with the matrix entries.

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]

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 compared 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.

- 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.

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 Operator (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.

- 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 - summarise BLIS/GotoBLAS, matrix mul register availability due to AI applications, and why this idea is good now.

3.1.1 Literature Review

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

- Full literature review of past approaches

- 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)

3.1.2 A New Direct Matrix Compression Based Acceleration Scheme

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

- Why it is counterintuitive

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

- The method itself, the precomputations required - metadata for displacements

- runtime metadata that's required, and the new allocations that are required for contiguous blocks of multipole data over sibling.

3.1.3 Review of FFT based Acceleration Based Schemes

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

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

- Both algorithmic and computational

3.1.4 Results

- 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. - i.e. similar software and hardware backends (they use OpenMP tasking vs Rayon for threading) - cache destroyed by granular tasking approach

- The numerical results from the paper.

- some color here, like the pirate ship.

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

3.1.5 Discussion

- Why our approach is sustainable given long term trends in hardware and software.

- Why might it be useful for Helmholtz FMM ...

- What are important trends, and what have we actually done.

- Interesting point of comparison with ScalFMM to demonstrate the importance of caching to performance, noting that direct software comparisons are not entirely fair.

3.2 Leaf Level Operators (P2P, L2P, P2M)

- How do the SIMD implementations work? Newton steps + fast inverse square root. - how are we blocking targets over threads? - 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.

3.4 Field Translation in a Distributed Setting

- Handling of ghost data, algorithms for this vs what is done now which is explained in the introduction now.

- Global/local tree split.

- Available parallelism in the downward pass

- Opportunities for kernel re-use

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

6.2.2 How High Can k Go?

- 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,

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

$$\text{Total Cost } = O\left(k^4 D_d^4 2^{4d} \sum_{l=0}^{d} 2^{-4l}\right) \tag{6.1}$$

$$= O\left(k^4 D_d^4 2^{4d} \frac{16}{15}(1 - \frac{1}{16^{d+1}})\right) \tag{6.2}$$

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 hugely different to $N \log N$ algorithm

- 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 problems 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. 21

BLAS Basic Linear Algebra Subprograms. v, 5, 21

CLP Core Level Parallelism. 2

CPU Central Processing Unit. v, 2, 3, 11

DLP Data Level Parallelism. 2

FFT Fast Fourier Transform. v, 5, 22

FLOP Floating Point Operation. 5, 21

FMM Fast Multipole Method. v, 1–5, 7–13, 15, 17, 19–23, 35, 37

GPU Graphics Processing Unit. 2, 3, 11

HPC High Performance Computing. v, 4

kiFMM kernel independent Fast Multipole Method. v, 4–6, 11, 17, 37

L2L Local to Local. 11, 35

L2P Local to Particle. 11

M2L Multipole to Local. v, 5, 12, 13, 35, 37

M2L Multipole to Particle. 11

M2M Multipole to Multipole. 11, 35

MFS Method of Fundamental Solutions. 17

MIMD Multiple Instruction Multiple Data. 2

P2M Particle to Multipole. 11

P2P Particle to Particle. 11, 12

PDE Partial Differential Equation. v, 4

SIMD Single Instruction Multiple Data. 2, 3, 11, 22, 23

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] Hans Sagan. *Space-filling curves*. Springer Science & Business Media, 2012.

[26] 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.

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

[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.