Modern Research Software For Fast Multipole Methods

Srinath Kailasa

A thesis submitted in partial fulfillment of the requirements for the degree 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 wind of emotional support from my family the regularity 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, మీరు లేకుండా ఇది జరిగేది కాదు |

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Unpublished Manuscripts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 Partial Differential Equations (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 Fast Fourier Transforms (FFT) 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 [4].

As the primary outputs are open-source software libraries [5, 3] 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.

Acronyms

BLAS	Basic Linear Algebra Subprograms. v
CPU	Central Processing Unit. v
FFT	Fast Fourier Transforms. v, 4
FMM	Fast Multipole Method. v, 1–8, 20
HPC	High Performance Computing. v
kiFMM	kernel independent Fast Multipole Method. v, 3, 4, 20
M2L	Multipole to Local. v, 4, 20
PDEs	Partial Differential Equations. v
SIMD	Single Instruction Multiple Data. 2
SIMT	Single Instruction Multiple Threads. 2

Contents

Introduction

1.1 Motivation

Since its introduction in the late 1980s by Greengard and Rokhlin [2], 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 [1]. 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 such cases and 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 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 these N problem 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.

Despite being a well established algorithm, with numerous variants [CITE VARIANTS] and software efforts [CITE software] over the preceding decades, 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 mid 2000s, hardware design and development has focussed on enhancing parallelism

Heterogenous computing ...

The direct evaluations of (1.1) can be seen to be embarrassingly parallel over each target particle, and are therefore well represented in hardware by the SIMD and SIMT paradigms. Remaining open questions are about how heterogenous hardware, modern runtimes which can break up and optimally schedule subroutines of the FMM, and memory bound subroutines can be optimally designed and deployed so as to reflect hardware developments are

Speed for speed's sake, despite being an interesting proposition from a computational point of view, isn't without application.

- Why might faster simulations be helpful? - Unlock more detailed scientific simulations - Establish norms for implementation that reflects the capabilities of new hardwares

Indeed, faster particle simulations, of which the FMM is an example, have been identified as a key operation for optimisation for the exascale era [BERKELEY seven

[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.

dwarf] due to their broad utility across scientific computing.

Software efforts for FMMs were a particular focus of activity in the 2010s, with prominent examples being the ExaFMM project [CITATION], ScalFMM [CITATION] and PVFMM [CITATION]. Indeed, FMM software have cumulatively been the recipients of numerous Gordon Bell awards in the past decade [CITATIONS]. The collective weakness of existing software efforts however is their brittleness, due to the challenges of achieving high performance in a distributed memory setting and for three dimensional problems software, largely developed in a research setting, are rarely maintained with little documentation beyond published performance metrics, and optimisations designed to take advantage of or showcase a particular hardware or algorithmic approach. Therefore an additional focus of this thesis is the development of a *platform* for developing FMMs and its variants, for which much of the underlying machinery can be re-used. Modular design is critical to encouraging open-source contributions, and software that thrives even after the completion of a doctoral research project or a grant. A dual emphasis both on performance and design results in

- Something people can pick up and use on pretty much any device, especially non-specialists - Something which can be maintained even after a research project has ended. - Something that is being actively used by the community, resulting in iterative progress.

1.2 Thesis Structure

In Chapter 2 we perform a literature review of methods and software for modern Fast Multipole Methods, with a specific focus on so called 'kernel independent' or 'black box' FMMs in Sections ... and ... which are the focus of our implementation efforts. We review related ideas which share many features of the kiFMMs in Section ..., such as the $\mathcal{H}$ and $\mathcal{H}^2$ matrix approaches. We move on to a review of the kiFMMs computational structure in Section ..., where we provide estimates of the computational complexities of its operators, and identify the parallelism available in the algorithm with respect to that provided by modern hardware. We conclude with a review of past software efforts for FMMs, and place our contribution within this context.

A major effort of this thesis was designing a *platform* for kiFMMs. Whereby, one is free to experiment with the implementation of subcomponents in a highly modular way, while retaining performance and the use of the remainder of the library. Therefore a significant early investigation was into appropriate tooling environments for scientific software, first presented in [5]. We present this investigation in Chapter 3, where we document our experience with Python as an alternative for achieving low-level performance as well as our chosen platform Rust, a relatively new language emerging as a contender for performant and productive research software.

Chapter 4 details a rigorous application of our framework, where we investigated optimisations for the crucial M2L field translation, recently presented in [4]. We find non-intuitively that direct matrix compression techniques for admissable blocks can be highly competitive with state of the art optimal schemes based on FFT for three dimensional problems described by the Laplace kernel.

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. In Chapter 6 we discuss the design and implementation of our software framework for distributed memory systems, detailing communication reducing schemes for the communication of ghost information.

Chapter 7 contains numerical experiments with our software in a single node (Section ...) as well as HPC (Section ...) setting, including a study of the applicability of our software to problems described by Helmholtz problems with low to moderate wavenumbers.

We conclude with a reflection on our results and suggestions for future investigations in Chapter 8.

Review of Fast Multipole Methods

Portions of the discussion in Sections 2.1 and 2.2 of this chapter are adapted from the material first presented in [4]

2.1 Analytical Fast Multipole Methods

As in the original presentation, we use the case of evaluating electrostatic potentials to motivate the FMM. Consider the electric field, E due to a 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.

$$E = -\nabla \phi$$

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,

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

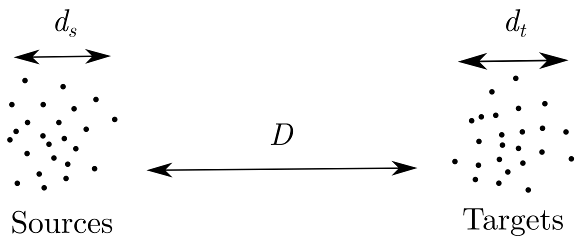

Figure 2.1: A set of source and target particle cluster, where the width of each cluster is significantly less than the distance separating them, $d_s, d_t \ll D$.

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 a matrix vector multiplication, where the matrix is *dense*, i.e. it consists 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. The FMM relies on a *degenerate* approximation of the interaction kernel when clusters of source and target particles are sufficiently separated, as sketched in Figure 2.1. Following the discussion in [4] 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 FMM 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).

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

- Overview of FMM algorithm for non-Oscillatory kernels.

- A brief not on computing changes in the past decades, and which part of the algorithm are a little redundant.

- A reflection on the key hidden constants in complexity, and why this may no longer be the most significant factor in modern implementations.

- A note on FMM software that is available, its positives and negatives, and why this is a little different.

- Conclude with why this thesis exists, to tackle these problems.

2.2 Kernel Independent Fast Multipole Methods

- 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.

- Data flow during the KiFMM.

- Performance characteristics and features of the kiFMM.

- Reflection on the kiFMM and modern software and hardware

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.

- Black Box vs Analytical

2.3 Oscillatory Fast Multipole Methods

- helmholtz fmm

- high frequency helmholtz FMM, out of scope of the thesis but can mention that they exist. Especially in the context of kiFMM approaches, what is the key difference? When does it apply (the directional low rank condition), where is there an additional loop? Why might this result in greater complexity

2.4 Related Ideas

- H Matrix and H2 matrices, and wider setting of the FMM and related problems.

    - Abduljabbar thesis contains a nice summary I can read.

2.5 The Fast Multipole Method's Computational Structure

- Parallelism levels in computing (ILP (Pipelining, Superscalar, Speculative execution), Data level (SIMD, GPU), Thread level TLP (multithreading, simultaneous multithreading and hyper threading), Process level (symettric and asymettrixc multitprocessing), task level, Distributed Parallelism e.g. MPI and MapReduce)

    - Only some of these are relevant for scientific computing

    - Examine FMM data flow and relate to levels of Parallelism and which will be taken advantage of by us, and which are yet to be examined.

    - What is the trend in hardware and why is the FMM a good kernel for scaling in future computer systems?

    - What are the principal difficulties we will encounter? Data organisation, and communication costs in a distributed setting.

    - What about good FMM software? Specialised kernels and substructures are required to be generically interfaced.

    - What parts of this are addressed by this thesis and where?

2.6 Review of Software Approaches

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

    - What are they optimised for, and what kind of performance do they promise?

    - What are the trade-offs of each software

- Hardware targetted by each available software, what's missing?

- What's available, and what are the shortcomings?

- And then

Modern Programming Environments for Science

The discussion in this chapter, including figures and diagrams, is adapted from the material first presented in [5].

3.1 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.

3.2 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

- Review of data oriented design, how this can be enabled with traits.

Performant Multipole To Local Field Translation

The discussion in this chapter, including figures and diagrams, is adapted from the material first presented in [4]

- Review of approaches, success and failures, and what works in the context of modern software and hardware systems.

- Can include section reviewing PVFMM approach

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

- Our approach, and why it works with reference to the software and hardware currently available.

- Articulate the significance of this result

- 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.

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.

- Diagram for principal traits and how they link together in the final 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.

5.2 FMM Software As A Framework

- Want to encourage as much code re-use as possible.

- 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

- Explain the diagrams (will need zoomed in diagrams for things like M2L data)

- Decouple implementation from

- 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.

5.3 High Performance Trees

- Exactly how are Morton encodings done, and what are the drawbacks and alternatives.

- ORB (how does it work)

- 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.

Important implementation details - construction of interaction lists, neighbour finding. - construction of Morton encodings. - trade-offs of approach in shared and distributed memory - e.g. adaptive vs weakly adaptive trees. - problems with load balancing approach etc - justification - simplicity/works vs complex/private

5.4 High Performance FMM Operators

- FMM data flow, mapping its tree structure to a data flow diagram.

- Go through each operator of kiFMM and how it's been optimised, at a high level for M2L operators.

5.4.1 Point to Multipole (P2M)

- Formulation in a blocked manner

5.4.2 Multipole to Multipole (M2M) and Local to Local (L2L)

- Formulation as BLAS3

5.4.3 Point to Point (P2P)

- Computational challenge (SIMD/SIMD), and why this is the easiest to optimise.

5.4.4 Multipole to Local (M2L)

- Description of computational challenge due to memory layout proscribed by Morton encoding.

    - Formulation of the problem, and required precomputations.

    - Data structure required to be flexible over this.

How future FMMs may look, shallow trees with large P2P.

The Fast Multipole Method for HPC

6.1 The Distributed Fast Multipole Method

6.2 Ghost Communication

6.3 Extending the Software's Design

- 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?

Numerical Experiments

7.1 Single Node

7.1.1 Laplace

7.1.2 Helmholtz

7.2 Multi Node

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

Appendix

Bibliography

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

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

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

[4] 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`.

[5] 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.