

Towards Exascale Multiparticle Simulations

Srinath Kailasa

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

Department of Mathematics
University College London
September, 2022

Declaration

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

Abstract

The past three decades have seen the emergence of so called ‘fast algorithms’ that are able to optimally apply and invert dense matrices that exhibit a special low-rank structure in their off-diagonal elements. Such matrices arise in numerous areas of science and engineering, from the discretised system matrices of boundary integral formulations [REFERENCE], to the dense covariance matrices that appear in machine learning applications like Kalman filtering [REFERENCE]. In the best case matrices can be stored, applied and inverted in $O(N)$, in contrast to $O(N^2)$ for storage and application, and $O(N^3)$ for inversion when computed naively.

The unification of software for the forward and inverse application of these operators in a single set of open-source libraries optimised for distributed computing environments is lacking, and is the central concern of this research project. Developing high-performance implementations of fast algorithms is challenging due to highly-technical nature of their underlying mathematical machinery, further complicated by the diversity of software and hardware environments in which research code is expected to run.

This subsidiary thesis presents current progress towards this goal. Chapter (1) introduces the Fast Multipole Method (FMM), the prototypical fast algorithm for $O(N)$ matrix vector products, and discusses implementation strategies in the context of high-performance software implementations. Chapter (2) provides a survey of the fragmented software landscape for fast algorithms, before proceeding with a case study of a Python implementation of an FMM, which attempted to bridge the gap between a familiar and ergonomic language for researchers and high-performance. The remainder of the chapter introduces Rust, our proposed solution for ergonomic and high-performance codes for computational science, and it concludes with an overview of a software output: Rusty Tree, a new Rust-based library for the construction of parallel octrees, a foundational datastructure for FMMs, as well as other fast algorithms. Chapter (3) introduces vectors for future research, specifically an introduction to fast algorithms and software for matrix inversion, and the potential pitfalls we will face in their implementation for performance, as well as an overview of a proposed investigation into the optimal mathematical implementation of field translations - a crucial component of a performant FMM.

Contents

1	The Fast Multipole Method	1
1.1	Motivation	1
1.2	Analytic to Algebraic Hierarchical Low-Rank Approximations	1
2	Designing Software for Fast Algorithms	2
2.1	The Software Landscape	2
2.2	Case Study: PyExaFMM, a Python FMM	2
2.3	Rust for High Performance Computing	2
2.4	Case Study: RustyTree, a Rust based Parallel Octree	2
3	Looking Ahead	3
3.1	Fast Direct Solvers on Distributed Memory Systems	3
3.2	Optimal Translation Operators for Fast Algorithms	3
3.3	Target Application: Maxwell Scattering	3
4	Conclusion	4
	Glossary	5

The Fast Multipole Method

- Very brief motivation behind fast particle solvers, and where they fit in the scientific landscape, why are they useful?
 - Brief survey on other uses of FMM (Kalman filtering - covariance matrices), lead into application for integral equations which is our motivating application
 - Construct example of how this fits into integral equation solvers, ie. $G_1 A G_2$, which is our motivating problem.

-

1.1 Motivation

- Brief summary of the algorithm history
 - Brief summary of the algorithm's logic - and the low rank assumption behind its power.
 - Emphasise the choices that can be made: translations, expansions, etc.

1.2 Analytic to Algebraic Hierarchical Low-Rank Approximations

- Lay out a sketch of the fast-algorithms for particle summation in terms of the flops/memory trade-off as in Yokota paper.
 - Mention other hierarchical schemes (H, H2 etc) and explain why FMM is preferred for our application.
 - Explain why our version of FMM - semi-analytical - is preferred for programming.
 - Explain what we can focus on to really accelerate in the semi-analytical FMM to program for exascale, with reference to field translations chapter.
 - Reference to software construction for fast solvers, and what works and doesn't, what's currently available - and their shortcomings, and positives.

Designing Software for Fast Algorithms

- Monograph on the complexities involved in designing software that is performant & usable for the majority of researchers who may not be software experts.

2.1 The Software Landscape

- Current software projects, what they focus on, what their pitfalls are.
 - Why are we experimenting with Python and Rust? Where does the need for this come from, what has been done in the past?

2.2 Case Study: PyExaFMM, a Python FMM

- Give an overview of the constraints that JIT compilers impose on program design.
 - Provide evidence and results from pyexafmm paper.
 - Conclude with idea that an alternative is necessary, but going back to C++ isn't the right option.

2.3 Rust for High Performance Computing

- Summary of Rust's core features for computational science
 - cargo, code organisation features, traits system, python interfacing.
 - Rebuke common misconceptions: safety (bounds checking), lack of appropriate libraries.
 - Overview of current HPC rust projects, and work groups.

2.4 Case Study: RustyTree, a Rust based Parallel Octree

- Case study for Rusty tree on different HPC environments and architectures.
 - briefly introduce algorithms (parallel sorting, tree construction)
 - talk about API, Python API, rSMPI, scaling performance
 - contrast with existing libraries, their performance, and how easy they are to install and edit.

Looking Ahead

- Towards a fully distributed fast solver infrastructure
 - Explain the context of the project, and how we plan to achieve its goals.

3.1 Fast Direct Solvers on Distributed Memory Systems

- Introduce the logic behind fast direct solvers via a short literature survey of the most popular methods.
 - Introduce RS-S and skeletonization based approaches, why these are good (proxy compression, can re-use octree data structure, work with moderate frequency oscillatory problems, straightforward to parallelize)
 - Introduce current state of the art work with Manas on proxy compression for Helmholtz problems.
 - Conclude with future plans for fast direct solver using our Galerkin discretized BIE.

3.2 Optimal Translation Operators for Fast Algorithms

- Translation operators, what are they, and what are the different approaches currently used.
 - What are the trade-offs?
 - Can I write some quick software for the quick comparison of translation operators - maybe in Python, on top of PyExaFMM?

3.3 Target Application: Maxwell Scattering

- Very brief summary of the Maxwell scattering problem, how we will form the BIE, the representation formulae we'll use, and how the integral operator will be discretised.
 - Overview of what kind of problems this would help us solve?

Conclusion

Glossary

FMM Shorthand for Fast Multipole Method. . iii