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, for example in the discretised system matrices of boundary integral formulations of problems from acoustics, to electromagnetics, fluid dynamics, geomechanics and even seismology. 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. We conclude with a look ahead towards a key target application for our software, the solution of large-scale electromagnetic scattering problems described by Maxwell's equations.

Contents

The Fast Multipole Method

- 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

- Give a sense of the scale of improvement that these methods provide for solving this class of problems - making integral equations practicable computational techniques is a big deal.

1 page

1.1 Motivation

- Summary of the algorithm's logic - and the low rank assumption behind its power.

- Overview of the choices that can be made: translations, expansions, etc.

2 pages

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.

- Discuss 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.

- Discuss software construction for fast solvers, and what works and doesn't, what's currently available - and its shortcomings, and positives.

2 pages

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.

- Get more examples and data on the difficulties faced by researchers for research software.

- Explain how the software goal of this research is to design software that can scale from a laptop to the latest supercomputing cluster.

1 page

2.1 The Software Landscape

- Current software projects, what they focus on, what their pitfalls are.

- Emphasise lack of integrated approach, and relatively few examples of open-source codes that are easy to build/deploy - i.e. aren't special research codes created to demonstrate a result.

- 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

- Summarise pyexafmm paper, and what it hoped to discover - Can we use JIT compilers to build cse applications? Answer, probably not.

- Give an overview of the constraints on program design.

- Conclude with idea that an alternative is necessary, but going back to C++ isn't the right option.

- List what we ideally want from a language for scientific computing. Speed is one thing, but we also want maintainability, easy testing, building on different environments, Python...

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 for numerical data.

- Highlight what actually is missing, e.g. rust-native tools (linear algebra, MPI etc) - and what's being done about it.

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).

- The novelty isn't the fact that it's a parallel octree, it's that it's one that you can use easily from Python, and deploy to different HPC environments and architectures.

Talk about the ease of writing a Python interface, and how this interoperability works. Talk about rSMPI project, it's important as this is an example that makes installation harder than it needs to be as it's a C shim - and that this is an example of a (relative) pitfall as an early adopter.

- contrast with existing libraries, their performance on different architectures, and how easy they are to install and edit - how malleable are they?

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 of different approaches?
    - Can I write some quick software for the quick comparison of translation operators - maybe in Python, on top of RustyTree? This would allow me to get some graphs to compare between approaches. If this is too much work, I will have to just compare the approaches in words.

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

- Short monograph summarising near term (translation operators, algebraic fmm) and longer term (inverse library) goals. Talk about recent achievements and results, to demonstrate that the goals are achievable in the time remaining.

Glossary

FMM Shorthand for Fast Multipole Method. . iii