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 linear system matrices of boundary integral formulations of problems from acoustics and electromagnetics to 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. We propose the creation of a unified solver infrastructure that can demonstrates good weak scaling from local workstations to upcoming exascale clusters. 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 achieving 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

1.1 Motivation

The motivation behind the development of the original fast multipole method (FMM), was the calculation of N-body problems,

$$\phi_j = \sum_{i=1}^{N} K(x_i, x_j) q_i \tag{1.1}$$

Consider electrostatics, or gravitation, where q_i is a point charge or mass, and $K(x, y) = \frac{1}{4\pi|x-y|}$ is the Laplace kernel. Similar sums appear in the discretised form of boundary integral equation (BIE) formulations for elliptic partial differential equations (PDEs), which are the example that motivates our research. Generically, an integral equation formulation can be written as,

$$a(x)u(x) + b(x) \int_{\Omega} K(x, y)c(y)u(y)dy = f(x), \;\; x \in \Omega \subset \mathbb{R}^d \tag{1.2}$$

where the dimension $d = 2$ or 3. The functions $a(x)$, $b(x)$ and $c(y)$ are given and linked to the parameters of a problem, $K(x, y)$ is some known kernel function and $f(x)$ is a known right hand side, $K(x, y)$ is associated with the PDE - either its Green's function, or the derivative. This is a very general formula, and includes common problems such as the Laplace equation, Lippman-Schwinger equation and the Helmholtz equation in the low frequency case. Upon discretisation using some appropriate method, for example the Nyström or Galerkin methods, we obtain a linear system of the form,

$$\mathsf{K}u = f \tag{1.3}$$

The defining feature of this linear system is that K is *dense*, with non-zero off-diagonal elements. Such problems are *globally data dependent*, in the sense that the calculation at each element in the discrete system depends on all other elements.

This density made numerical methods based on boundary integral equations prohibitively expensive prior to the discovery of so called 'fast algorithms', of which the FMM is the prototypical example. The naive computational complexity of storing a dense matrix, or calculating its matrix vector product is $O(N^2)$, and the complexity of finding its inverse is $O(N^3)$, where N is the number unknowns. This cost should have made the construction of numerical software based on boundary integral formulations moot, however the FMM demonstrated that the rapid decay

behaviour exhibited by certain classes of kernel function can be exploited to reduce the asymptotic cost of handling dense matrices in formulations that result from them.

In the best case the matrix vector product described by (1.1) and (1.3) can be computed in just $O(N)$ FLOPS and stored with $O(N)$ memory. Given the wide applicability of boundary integral equations to natural sciences, from acoustics [10, 4] and electrostatics [9] to electromagnetics [3] fluid dynamics [8] and earth science [2]. The FMM can be seen to have dramatically brought within reach large scale simulations of a wide class of scientific and engineering problems.

Furthermore, the FMM can also be applied to volume integral equations [7]. Indeed, as (1.1) shares its form with the kernel summations often found in statistical applications, the FMM has found uses in and computational statistics [1], machine learning [5] and Kalman filtering [6].

Despite their different origins and formulations, these example applications are united by their global data dependency. Resolving this is the key challenge in developing software for fast algorithms. The rapid decay behaviour between physically distant interactions required by these algorithms is often referred to as 'low rank', in which cases a compressed representation may be sought

...Figure illustrating low-rank assumption.

...FMM logic. It relies on a recursive data structure to hierarchically partition a domain of interest. In two dimensions, we use a quadtree (FIGURE FOR QUADTREE), and in three dimensions we use an octree (FIGURE FOR OCTREE). Denoting the set of points contained within a node of a tree as a 'cluster', the basic logic of the FMM, regardless of its flavour (see section 1.2), involves a hierarchical recursion through the tree, and approximating interactions between distant clusters where a low-rank assumption can be applied. This is the how the FMM, and other fast-algorithms, can

1.2 From 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 FMM, and what works and doesn't, what's currently available - and its shortcomings, and positives.

- Discuss software challenges for high performance FMM, teeing up remaining sections of paper.

$\sim$ 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.

Appendix

A.1 Fast Multipole Method Algorithm

FMM Algorithm logic as pseudocode.

Glossary

FLOPS Floating Point Operations per Second, a measure of computer performance. . 2

FMM Shorthand for the Fast Multipole Method. . iii, 1

Bibliography

[1] Sivaram Ambikasaran et al. "Large-scale stochastic linear inversion using hierarchical matrices". In: *Computational Geosciences* 17.6 (2013), pp. 913–927.

[2] Stéphanie Chaillat, Marc Bonnet, and Jean-François Semblat. "A multi-level fast multipole BEM for 3-D elastodynamics in the frequency domain". In: *Computer Methods in Applied Mechanics and Engineering* 197.49-50 (2008), pp. 4233–4249.

[3] 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.

[4] 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.

[5] Dongryeol Lee, Richard Vuduc, and Alexander G Gray. "A distributed kernel summation framework for general-dimension machine learning". In: *Proceedings of the 2012 SIAM International Conference on Data Mining*. SIAM. 2012, pp. 391–402.

[6] 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.

[7] 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.

[8] Abtin Rahimian et al. "Petascale direct numerical simulation of blood flow on 200k cores and heterogeneous architectures". In: *SC'10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE. 2010, pp. 1–11.

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

[10] William R Wolf and Sanjiva K Lele. "Aeroacoustic integrals accelerated by fast multipole method". In: *AIAA journal* 49.7 (2011), pp. 1466–1477.