Department: Head
Editor: Name, xxxx@email

PyExaFMM: Designing a highly-performant particle fast multipole solver in Python with Numba

S. Kailasa
Department of Mathematics, University College London

T. Betcke
Department of Mathematics, University College London

T. Wang
Department of Mechanical and Aerospace Engineering, The George Washington University

L. A. Barba
Department of Mechanical and Aerospace Engineering, The George Washington University

***Abstract*—PyExaFMM is a Python based kernel-independent particle fast multipole method [FMM] implementation, built on the success of the ExaFMM project, to answer the question of whether we could develop a highly-performant scientific code without resorting to a lower level language. The FMM is a good case study for understanding the maturity of Python for developing high-performance software, due its reliance on a complex heirarchical octree data structure. In this article we explore the software engineering and mathematical techniques used to extract performance for PyExaFMM, and we report that we are able to achieve runtimes within $\mathcal{O}(10)$ of the state of the art C++ implementation, with comparable accuracy for three dimensional electrostatic problems.**

THE FAST MULTIPOLE METHOD [FMM], originally developed by Greengard and Rokhlin [1], approximates the solution of the so called N body problem, in which one seeks to calculate the pairwise interactions between N objects. This problem arises in numerous contexts in physics and engineering, for example in the calculation of the resulting electrostatic potentials due to a set of charged particles. Considering the calculation of electrostatic potential as our model problem, the potential, $\phi(x_j)$, for a given charged particle at position x_j, or 'target', due to N particles at

 Published by the IEEE Computer Society

positions x_i, or 'sources', where $i \in [1, ..., N]$, each with a charge q_i, can be written as,

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

here $K(\cdot, \cdot)$ is called the Green's function, or kernel function, which for electrostatic problems in three dimensions is,

$$K(x, y) = \frac{1}{4\epsilon_0 \pi |x - y|}, \tag{2}$$

where ϵ_0 is the permittivity of free space. This kernel function is often referred to as the Laplace kernel.

Attempting to evaluate the sum in (1) directly for N target particles at positions x_j where $j \in [1, ..., N]$ due to N sources, results in algorithm of $\mathcal{O}(N^2)$ runtime complexity, however the FMM is able to approximate (1) with just $\mathcal{O}(N)$ runtime complexity, with proscribed error bounds.

The key idea behind the FMM is to encode the potential in the far field due to a cluster of particles with a representative analytic *multipole expansio*n centered on the cluster, which can be truncated to tune for desired accuracy. This truncation allows one to approximate the sum in (1) with fewer calculations. In regions centered far away from the cluster, their potentials can be encoded in a *local expansion*. Translations between multipole and local expansions can be done analytically, and are critical in the development of the $\mathcal{O}(N)$ algorithm. In the FMM, the problem domain is described by a box enclosing all targets and sources, which is hierarchically partitioned into a structure known as a quadtree in two dimensions, and an octree in three dimensions. The partioning is defined by a parameter for the maximum number of particles contained in each leaf node. The FMM then consists of two traversals through this tree. Initially during the *upward pass*, multipole expansions for for particles contained in boxes at the leaf level of the tree are formed. In order to obtain the multipole expansion the previous level of discretization, the multipole expansions of a given box's children at the leaf level have their expansion centre translated to the centre of the parent box, and the coefficients are summed. This is repeated level by level, till one is left with the multipole expansions of all boxes in the tree. Secondly, during the *downward pass*, the multipole expansions of non-adjacent boxes of a given box are defined as being in it far-field. In order to evaluate their contribution towards the potential for particles within a given box, one translates their multipole expansion to a local expansion centered at the given box. The local expansion is then transferred to the children of the given box by shifting the expansion centre. Proceeding level by level, one is left with the local expansions for each leaf box. Crucially the size of the far field not already encapsulated in the local expansion shrinks as we descend down the octree. This is because the far field of a box's parent level is already captured in the child's local expansion. These local expansions compress all of the contribution from the far field towards the potential for targets within a given leaf. Their near fields are evaluated directly using (1). The complexity of this near field evaluation is bounded by the parameter dictating the maximum number of particles per leaf. The $\mathcal{O}(N)$ complexity can be roughly seen to be the result of the ability to exactly translate between the multipole and local expansions for all boxes deemed to be in the far field, as well as the recursive procedure of the FMM. As a result of this, each box must only consider it's interaction with a bounded constant number of other boxes. As there are $\mathcal{O}(N)$ boxes in a given octree, the entire algorithm can be seen to be bounded by a runtime complexity of $\mathcal{O}(N)$.

The coefficients of a given multipole expansion are kernel-dependent in the sense that they will depend on the form of (2), software implementations have to be rewritten for each specific physical model. Fast multipole methods for general kernels [CITATION] have been developed, however there are fewer implementations that rely only on numerical kernel evaluations, rather than analytic series expansions, examples include [CITATION]. The latter approaches are often referred to as kernel-independent fast multipole methods [KIFMM]. ...

PyExaFMM utilizes the approach first presented by Ying et. al [2]. This KIFMM represents the multipole expansions due to a cluster of charges as set of equivalent densities sup-

ported on a surface enclosing the cluster, with the fields generated by the charges matched with the equivalent field via a least-squares fitting in the far field. As originally developed in [2], this method generalizes the FMM to non-oscillatory second-order elliptic partial differential equations with constant coefficients, however further work has extended the method to include oscillatory problems [3].

At present, numerous high-quality open source implementations exist to solve the analytic FMM for various kernels [CITATION], as well as the KIFMM [2] and [DARVE]. The desire for another FMM implementation lies in the disconnect between the computational skills of many users of FMM solvers, such as natural scientists and engineers, and the complex softwares developed by software specialists in compiled languages such as Fortran or C++. Despite often offering Python interfaces, it is still not feasible for many domain specialists to contribute to the software development of these existing libraries without a significant investment in acquiring new skills.

Python has emerged as a de-facto standard for the majority of computational scientists for both data analysis, and application development [CITATION], and in recent years high-performance libraries that aid in bypassing the limitations of the Python interpreter have emerged to make it a serious contender for use in developing software for high-performance computing [HPC]. The final vision being the ability to rapidly prototype an algorithm in Python, drop in high-performance libraries, and build a HPC application without necessarily being a software expert. PyExaFMM therefore represents a compromise between usability and performance, with the audience of the software in mind. Many HPC libraries in Python optimize operations on linear data structures, such as arrays. However, the FMM relies on a hierarchical tree, as such it represents a non-trivial algorithm to optimize with currently available tools. In developing PyExaFMM we aimed to demonstrate the possibility of building a HPC application purely in Python, while achieve comparable performance with state of the art compiled language implementations.

In this article we begin by providing an overview of the mathematical formulation behind the KIFMM in [2], before introducing Numba and discussing how it is used by PyExaFMM to accelerate computations, as well as program efficient data structures. We continue by discussing the mathematical and software based optimizations used by PyExaFMM for performance, and provide an overview of the software design used in order to implement optimizations effectively. We conclude with a discussion comparing the performance, in terms of accuracy, runtimes, and memory footprint, with the comparable state-of-the-art C++ implementation from the ExaFMM project, ExaFMM-T [4].

THE KERNEL-INDEPENDENT FAST MULTIPOLE METHOD

Algorithm

TECHNIQUES FOR ACHIEVING PERFORMANCE

What is Numba?

What is does, how is it useful? Where do we use it, and why there. How much difference can it make in an idealized routine. What doesn't work, why doesn't it work. Where to be careful. Programming to an (invisible) framework ...

Where is numba used heavily? Tree construction routines on Morton coordinates. Multithreading of tree construction, as well as P2M evaluation. Experiment to demonstrate the speed of kernel evaluation, with caveat that data must be pre-organised.

Precomputing Operators

Transfer vectors, hashing, HDF5, loading into memory.

Compressing the M2L step with a randomised SVD

Introduce [2]. Numerical bounds on error out of scope, but show how experiments demonstrate that the FMM error dominates the SVD error.

Software Architecture

The key is separating routines to be accelerated, and organising data ready to run. The data organisation part (and it's slowness) should be demonstrated as a bottleneck using experiment.

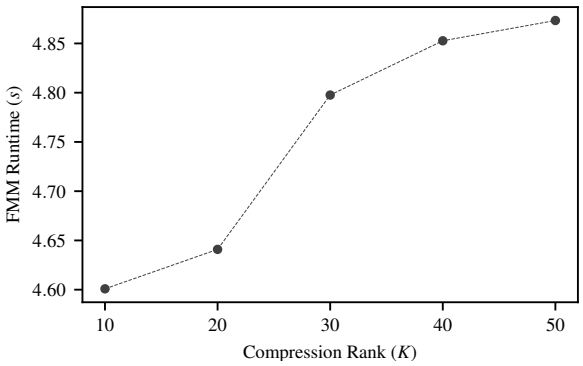

Figure 1. Compression Rank and runtime

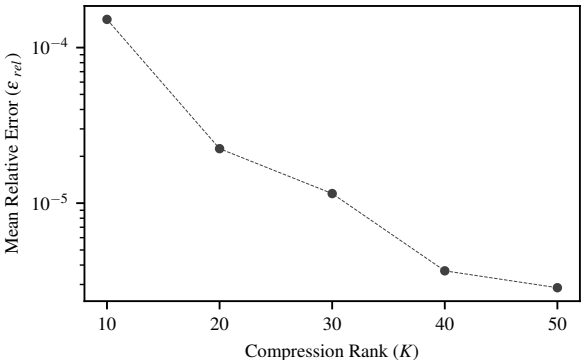

Figure 2. Compression Rank and accuracy

PERFORMANCE COMPARISON WITH STATE OF THE ART

Description of main experiments, and how they are conducted. What are the main results? Are they expected from theory? What conclusions can be drawn about developing HPC codes entirely in Python, is it worth it?

CONCLUSION

What have we learned, what will we be working on next?

ACKNOWLEDGMENT

SK is supported by EPSRC Studentship 2417009.

REFERENCES

1. L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *Journal of Computational Physics*, vol. 73, no. 2, pp. 325–348, 1987.
2. L. Ying, G. Biros, and D. Zorin, "A kernel-independent adaptive fast multipole algorithm in two and three dimensions," *Journal of Computational Physics*, vol. 196, no. 2, pp. 591–626, 2004.
3. B. Engquist and L. Ying, "Fast directional multilevel algorithms for oscillatory kernels," *SIAM Journal on Scientific Computing*, vol. 29, no. 4, pp. 1710–1737, 2007.
4. T. Wang, R. Yokota, and L. A. Barba, "Exafmm: a high-performance fast multipole method library with c++ and python interfaces," *Journal of Open Source Software*, vol. 6, no. 61, p. 3145, 2021.

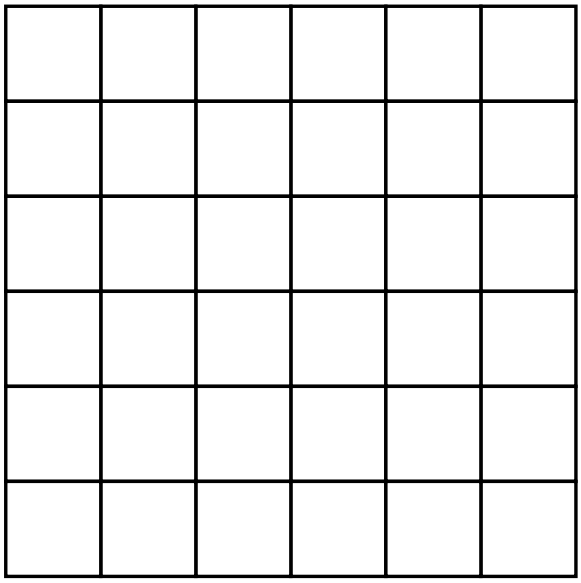

Figure 3. Compression Rank and accuracy

Srinath Kailasa is a graduate student at University College London. He is currently pursuing a PhD in Computational Mathematics, having received an MPhys in Physics (2017) and an MSc Scientific Computing (2020) from the University of Durham, and University College London respectively. His research interests are in high-performance and scientific computing. Contact him at srinath.kailasa.18@ucl.ac.uk.

Timo Betcke is a Professor of Computational Mathematics at University College London. Contact him at t.betcke@ucl.ac.uk.

Tingyu Wang is a PhD student in Mechanical Engineering at the George Washington University. Contact him at twang66@email.gwu.edu.

Lorena. A. Barba is a Professor of Mechanical and Aerospace Engineering at the George Washington University. Contact her at labarba@email.gwu.edu.

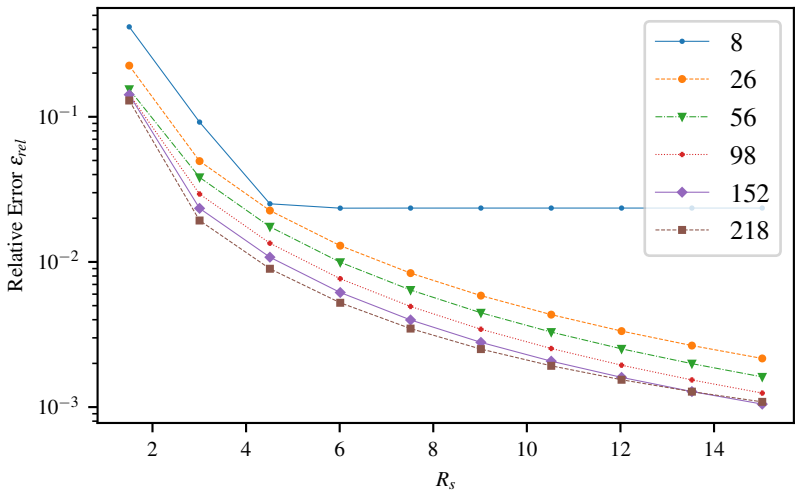

Figure 4. ExaFMM Multipole Expansion Convergence

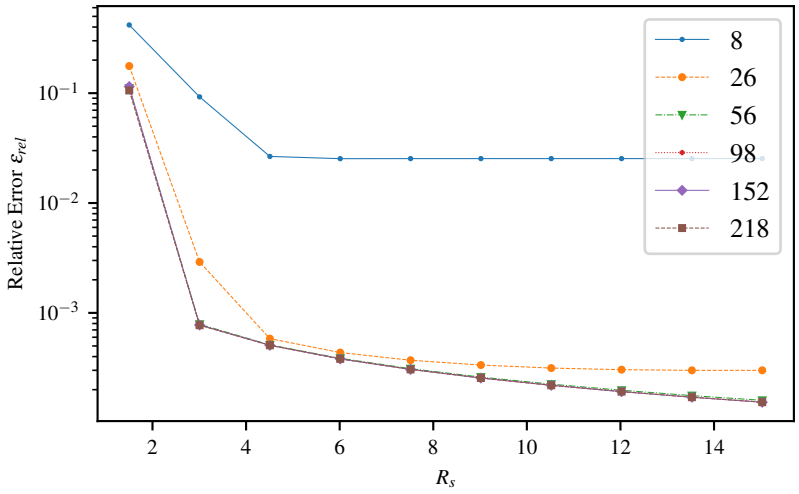

Figure 5. PyExaFMM Multipole Expansion Convergence

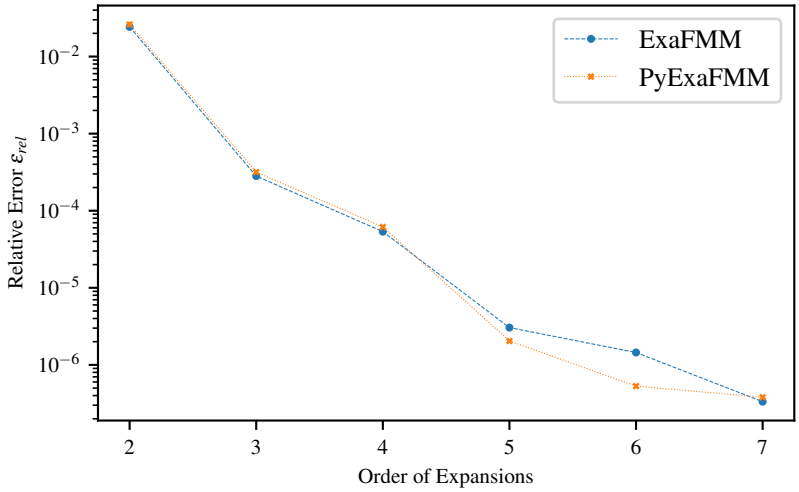

Figure 6. Potential Convergence Comparison