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 pythonic 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 paper we offer an overview the kernel-independent FMM algorithm, and the key tool from Python's numerical ecosystem for developing high-performance applications, Numba. We also describe the mathematical and software development techniques we used to aid performance. We conclude by benchmarking the software's accuracy, speed, and memory footprint with respect to the state of the art C++ implementation from the ExaFMM project. We report that we are able to achieve runtimes within $\mathcal{O}(10)$ of the state of the art, 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

 Published by the IEEE Computer Society © 2021 IEEE

and engineering. Consider the calculation of electrostatic potential, $\phi(x_j)$, for a given charged particle at position x_j, or 'target', due to N particles at positions x_i, or 'sources', where $i \in [1, ..., N]$, each with a charge q_i,

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

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) naively 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 expansion, which can be truncated to tune for desired accuracy. This truncation effectively allows for the user to approximate the sum in (1) with fewer calculations. However, the coefficients of a given multipole expansion are kernel-dependent in the sense that they will depend on the form of (2). Originally introduced by Ying et. al [2], the kernel-independent fast multipole method [KIFMM] represents the multipole expansions due to a cluster of charges as set of equivalent densities supported 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. This approach has the useful property of relying only on evaluations of the kernels, rather than analytic series expansions. This means that is can be programmed in a kernel-agnostic way and applied to a variety of problems. As originally developed in [2], the KIFMM 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].

Python has emerged as the de-facto standard for the majority of computational scientists. It offers the ability to rapidly prototype

In this article we begin by providing an overview of the mathematical formulation behind the KIFMM, 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

First introduced by Ying [2], the kernel-independent fast multipole method (KIFMM) ...

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

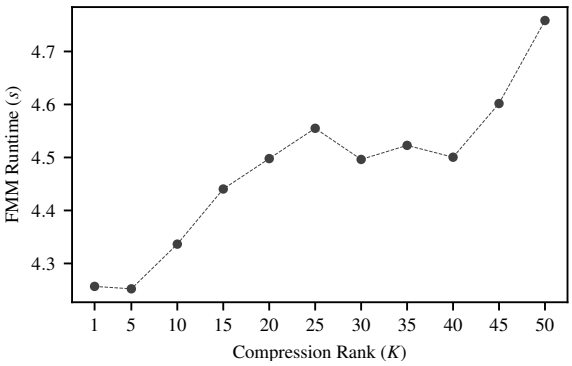

Figure 1. Compression Rank and runtime

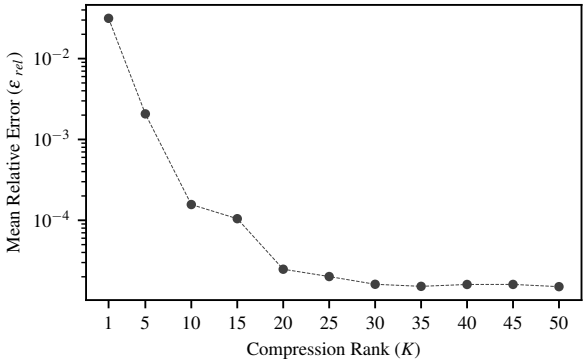

Figure 2. Compression Rank and accuracy

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.

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.

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.