Department: Head
Editor: Name, xxxx@email

# PyExaFMM: Designing a high-performance 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*—The particle fast multipole method [FMM] is a good case study for understanding the efficacy of Python for developing high-performance software for non-trivial algorithms, due its reliance on a hierarchical tree data structure. In this paper we explore the mathematical and software design techniques used to extract performance for PyExaFMM, a Python based FMM solver accelerated with Numba, designed to be run on single-node multicore architectures. We report that we achieve runtimes within $\mathcal{O}(N)$ of the state of the art C++ implementation, with comparable accuracy and memory footprint for three dimensional problems in double precision.**

## WE INTRODUCE PYEXAFMM[1]

- What's the context of PyExaFMM? - Compiled language implementations exist - Numba JIT compiler for Python can translate hot loops code to efficient machine code. - Python is well understood in SC community. - FMM is a benchmark for complex SC software built using Python alone. - This paper consists of: - Brief intro to FMM algorithm (¡1000 words) - Math methods, software design and parallelization strategies for achieving performance with Numba. ( 2500 words) - Benchmarks wrt comparable C++ module ( 1500 words)

[1]https://www.github.com/exafmm/pyexafmm

## FAST MULTIPOLE METHODS

Larger figure

  

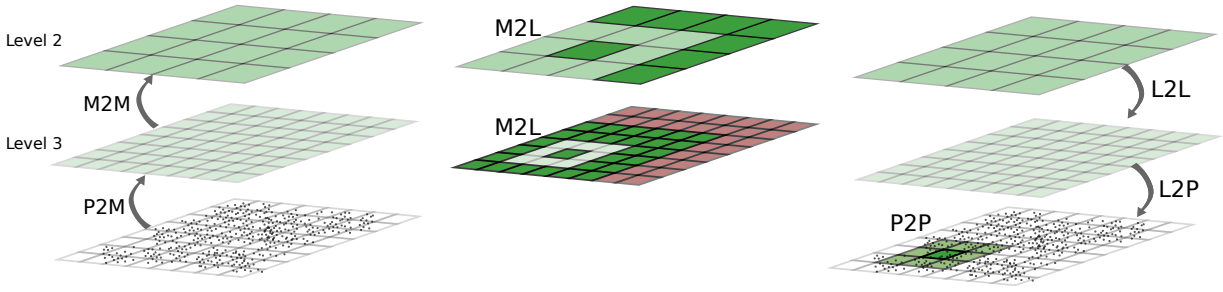


**Figure 1.** algorithm

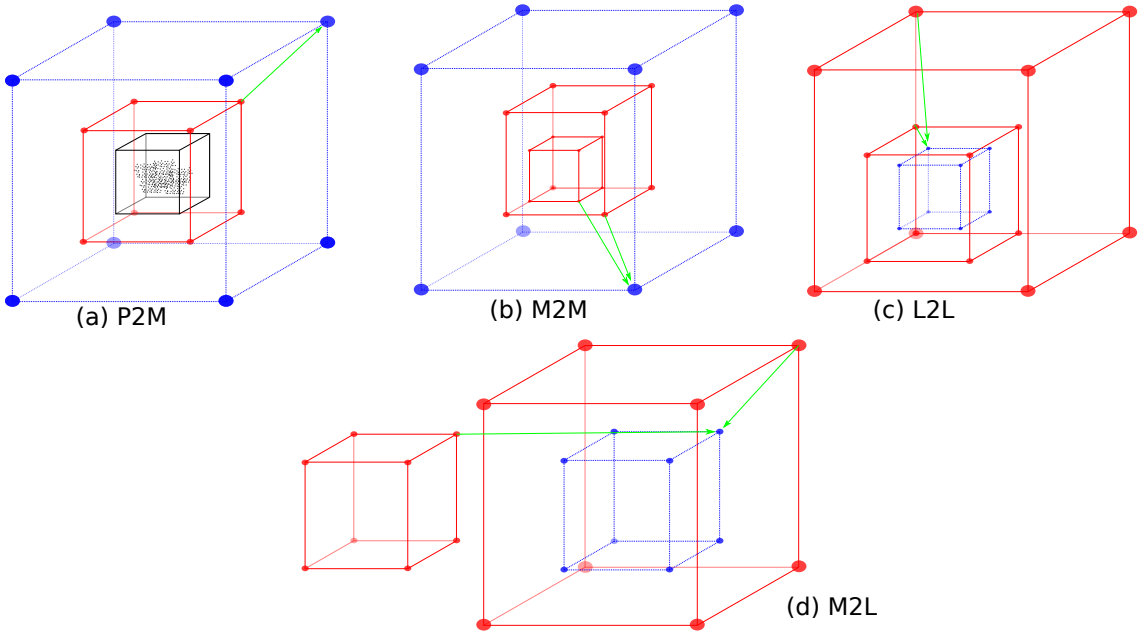


**Figure 2.** The operators of the KIFMM. Equivalent surfaces are shown in red, check surfaces in blue, and the charged points in black. Surfaces are plotted with 8 quadrature points, one at each vertex.

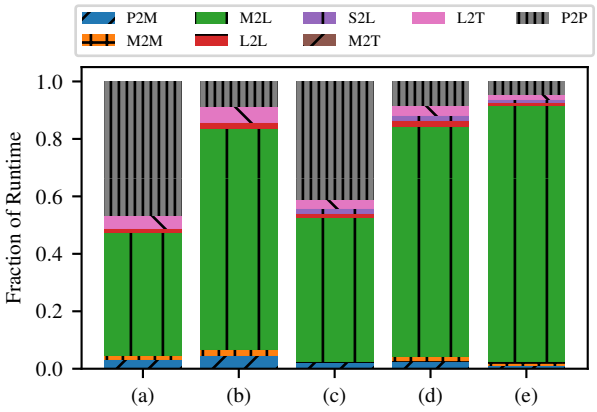


**Figure 3.** Foo bar

**Table 1. Relative error, runtime and peak memory consumption in comparison to the SOTA. Experiments run with $N = 1,000,000$ points tested in two geometries: (1) distributed randomly in a cubic unit box, (2) distributed randomly on the surface of a sphere with unit radius, leading to $M$ leaves in their respective geometries, with a maximum of $150$ points per leaf, multipole and local expansions of order $p$, and a compression rank $k$ for PyExaFMM. Charge densities are chosen in the interval $[0, 1)$. Runtimes exclude tree building time. Reported to 3 significant figures after a single run.**

| | | | | Runtime | | Peak Memory | | Relative Error | |
|---|---|---|---|---|---|---|---|---|---|
| $k$ | $M$ | $p$ | Geometry | PyExaFMM | ExaFMM-T | PyExaFMM | ExaFMM-T | PyExaFMM | ExaFMM-T |
| 50 | 17,017 | 4 | Sphere | $10.6 \pm 0.1$ s | $0.41 \pm 0.04$ s | 2.96 GB | 2.34 GB | 1.00e-4 | 8.75e-5 |
| | 32,768 | | Random | $13.2 \pm 0.2$ s | $0.41 \pm 0.05$ s | 4.93 GB | 2.98 GB | 8.75e-5 | 7.66e-5 |
| | | 6 | Sphere | $41.0 \pm 0.7$ s | $1.48 \pm 0.01$ s | 3.04 GB | 2.80 GB | 2.41e-6 | 1.67e-6 |
| | | | Random | $71 \pm 4$ s | $1.66 \pm 0.04$ s | 4.93 GB | 3.55 GB | 1.59e-6 | 3.41e-6 |
| | | 8 | Sphere | $57.0 \pm 0.1$ s | $1.78 \pm 0.04$ s | 3.09 GB | 3.22 GB | 2.00e-6 | 2.86e-6 |
| | | | Random | $131 \pm 2$ s | $2.11 \pm 0.06$ s | 4.93 GB | 3.88 GB | 1.71e-6 | 3.84e-6 |
| | | 10 | Sphere | $57.0 \pm 0.1$ s | $1.78 \pm 0.04$ s | 3.09 GB | 3.22 GB | 2.00e-6 | 2.86e-6 |
| | | | Random | $131 \pm 2$ s | $2.11 \pm 0.06$ s | 4.93 GB | 3.88 GB | 1.71e-6 | 3.84e-6 |
| 150 | 17,017 | 4 | Sphere | $10.6 \pm 0.1$ s | $0.41 \pm 0.04$ s | 2.96 GB | 2.34 GB | 1.00e-4 | 8.75e-5 |
| | 32,768 | | Random | $13.2 \pm 0.2$ s | $0.41 \pm 0.05$ s | 4.93 GB | 2.98 GB | 8.75e-5 | 7.66e-5 |
| | | 6 | Sphere | $33.8 \pm 0.8$ s | $1.28 \pm 0.03$ s | 3.00 GB | 2.53 GB | 4.55e-6 | 4.15e-6 |
| | | | Random | $65 \pm 2$ s | $1.46 \pm 0.04$ s | 4.93 GB | 3.32 GB | 2.81e-6 | 3.91e-6 |
| | | 8 | Sphere | $41.0 \pm 0.7$ s | $1.48 \pm 0.01$ s | 3.04 GB | 2.80 GB | 2.41e-6 | 1.67e-6 |
| | | | Random | $71 \pm 4$ s | $1.66 \pm 0.04$ s | 4.93 GB | 3.55 GB | 1.59e-6 | 3.41e-6 |
| | | 10 | Sphere | $57.0 \pm 0.1$ s | $1.78 \pm 0.04$ s | 3.09 GB | 3.22 GB | 2.00e-6 | 2.86e-6 |
| | | | Random | $131 \pm 2$ s | $2.11 \pm 0.06$ s | 4.93 GB | 3.88 GB | 1.71e-6 | 3.84e-6 |

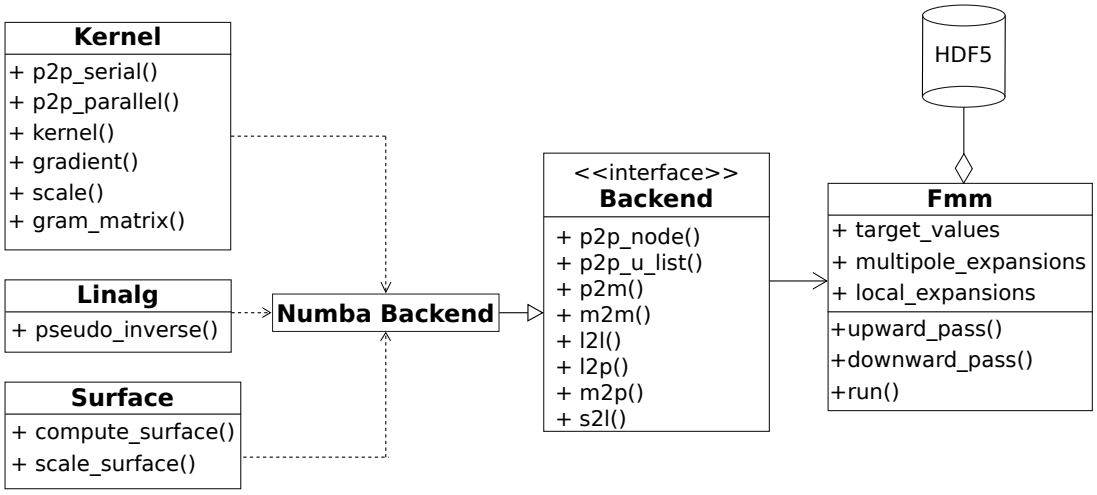


**Figure 4.** Simplified UML model of all PyExaFMM components. Trees and operators are precomputed and stored in the HDF5 database. Except for the 'Fmm' object which acts as the user interface, all other components are modules consisting of simple functions.