kifmm-rs: A Kernel-Independent Fast Multipole Framework in Rust

Srinath Kailasa [1]

1 Department of Mathematics, University College London, UK

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

License
Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License (CC BY 4.0).

DRAFT

Summary

- High level functionality and purpose of software in the context of related work for a non-specialist audience

We present `kifmm-rs` a Rust based implementation of the kernel independent Fast Multipole Method (kiFMM), with Python bindings, that serves as a implementation framework for implementing the kiFMMs Greengard & Rokhlin (1987). The FMM is a core algorithm for scientific computing, commonly cited as one of the top algorithmic advances of the twentieth century (Cipra, 2000) due to its acceleration of the computation of N-body potential evaluation problems of the form,

$$\phi(x_i) = \sum_{j=1}^{N} K(x_i, y_j) q(y_j) \tag{1}$$

from $O(N^2)$ to $O(N)$ or $O(N \log(N))$ for interaction kernels for second order elliptic partial differential equations. Such kernels commonly arise in scientific computations, such as the Laplace kernel which models the electrostatic or gravitational potentials corresponding to a set of source points on a set of target points,

$$K(x, y) = \begin{cases} \frac{1}{2\pi} \log(\frac{1}{\|x-y\|}), & (2D) \\ \frac{1}{4\pi \|x-y\|}, & (3D) \end{cases} \tag{2}$$

Kernel independent variants of the FMM (kiFMM), replace the analytical series approximations used to compress far-field interactions between clusters of source and target points, with kernel evaluations and are suitable for a wide variety of elliptic PDE kernel, such as the Laplace and Stokes kernels, and have been extended to oscillatory kernels such as Helmholtz. This enables the generic optimisation and programming of the underlying FMM machinery in a manner that is 'kernel independent'. As a rule of thumb, kernel independent variants are preferred for higher-order approximations for performance reasons.

Our principle contributions with this software that extend beyond current state of the art implementations Wang et al. (2021) are:

- The ability to process multiple right hand sides corresponding to the same particle distribution using (1).
- State of the art performance enabled by the optimisation of BLAS based field translation, based on level 3 operations with high arithmetic intensity that are well suited to modern hardware architectures that prioritise minimal memory movement per flop.
- The use of Rust as a computational platform for high-performance scientific computing, that enables an extensible data oriented design and simple cross platform deployment.

- Full Python bindings for our software's Rust API, enabling adoption by non-specialist users.

Statement of need

- clearly illustrates the research purpose of the software and places it in the research context

Algorithmic optimisations for the FMM and its kiFMM variants are often implemented

- disparately, as one offs
- highest performance libraries are difficult to compare in terms of algorithm tricks or implementation tricks.
- If designed for a single node, difficult to extend to multi-node.
- Hard to plug/play different softwares and hardwares.

`kifmm-rs` is an

- Rust package build for speed and flexibility
- API designed to be user friendly, and easy to bind
- Simple trait based design, allow for separation of concerns and interface
- Can
  - evaluate potentials, potential gradients, for a range of compatible kernels
  - heterogenous support for critical operations
  - multi-platform deployment with Rust
  - state of the art performance on a single node.
  - design flexible, can easily extend to multi-node problems in a future release.

Combination of - speed + design + extensibility to new functionality (related algorithms)

Past and ongoing research projects fi

- where does this software fit in?
- Older FMM efforts
- Embedded within new Bempp-rs
- Cite the M2L paper (Kailasa et al., 2024)

Rust as a language for scientific computing

Rust's build system

- Simple cross platform builds.
- Standardised autovectoriser, i.e. single compiler, based on LLVM
- SIMD via Pulp.
- C ABI compatibility, enables language bindings, as well as access to open source.

Rust's trait system and data oriented design.

- What is data oriented design?
- How do Rust's traits enable us to write in structs of arrays.

Benchmarks

We benchmark our software against other leading implementations on a single node Wang et al. (2021) for the target architectures in Table [1] for achieving a given precision for a common benchmark problem of computing (1) for the three dimensional Laplace kernel (2) for a million uniformly distributed source and target points.

DRAFT

Table 1: Hardware and software used in our benchmarks, for the Apple M1 Pro we report only the specifications of its 'performance' CPU cores. We report per core cache sizes for L1/L2 and total cache size for L3. We note that the Apple M series of processors are designed with unusually large cache sizes, as well as 'unified memory' architectures enabling rapid data access across specialised hardware units such as the performance CPU cores and the specialised matrix coprocessor used for BLAS operations when run with Apple's Accelerate framework

	Apple M1 Pro	AMD 3790X
Cache Line Size	128 B	64 B
L1i/L1d	192/128 KB	32/32 KB
L2	12 MB	512 KB
L3	12 MB	134 MB
Memory	16 GB	252 GB
Max Clock Speed	3.2 GhZ	3.7 GhZ
Sockets/Cores/Threads	1/8/8	1/32/64
Architecture	Arm V8.5	x86
BLAS	Apple Accelerate	BLIS
LAPACK	Apple Accelerate	Netlib
FFT	FFTW	FFTW
Threading	Rayon	Rayon

Conclusion

- Why this represents an innovation for KiFMM software as exists.
  - embedded within a wider project, with 'users'
  - easy to extend to alternative subcomponents
  - focus on data oriented design via rust.
  - focus on easy deployment to current and emerging architectures.

-->

Acknowledgements

Srinath Kailasa is supported by EPSRC Studentship 2417009

References

Cipra, B. A. (2000). The best of the 20th century: Editors name top 10 algorithms. *SIAM News*, *33*(4), 1–2.

Greengard, L., & Rokhlin, V. (1987). A fast algorithm for particle simulations. *Journal of Computational Physics*, *73*(2), 325–348. https://doi.org/10.1016/0021-9991(87)90140-9

Kailasa, S., Betcke, T., & El-Kazdadi, S. (2024). Designing kernel independent fast multipole methods for modern architectures. *Submitted To SIAM Journal on Scientific Computing*.

Malhotra, D., & Biros, G. (2015). PVFMM: A Parallel Kernel Independent FMM for Particle and Volume Potentials. *Commun. Comput. Phys.*, *18*(3), 808–830. https://doi.org/10.4208/cicp.020215.150515sw

Wang, T., Yokota, R., & Barba, L. A. (2021). ExaFMM: A high-performance fast multipole method library with c++ and python interfaces. *Journal of Open Source Software*, *6*(61), 3145.

DRAFT

Ying, L., Biros, G., & Zorin, D. (2004). A kernel-independent adaptive fast multipole algorithm in two and three dimensions. *Journal of Computational Physics*, *196*(2), 591–626. https://doi.org/10.1016/j.jcp.2003.11.021

DRAFT