




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

Srinath Kailasa <sup>1</sup>

<sup>1</sup> Department of Mathematics, University College London, UK

DOI: [10.xxxxxx/draft](https://doi.org/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Published: unpublished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](#)).

## Summary

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](#); [Ying et al., 2004](#)). 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,

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

where the potential  $\phi$  is evaluated at a set of targets,  $\{x_i\}_{i=1}^M$ , due to a set of densities,  $\{q_j\}_{j=1}^N$ , corresponding to a set of sources,  $\{y_j\}_{j=1}^N$ , and  $K(\cdot, \cdot)$  is the interaction kernel 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,

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

Indeed the FMM finds a major application in the acceleration of Boundary Element Methods (BEM) for elliptic boundary value problems ([Steinbach, 2007](#)), which can be used to model a wide range of natural phenomena.

Kernel independent variants of the FMM (kiFMMs) replace the analytical series approximations used to compress far-field interactions between clusters of source and target points, with approximation schemes that are based on kernel evaluations and extensions of the algorithm can also handle oscillatory problems specified by the Helmholtz kernel ([Engquist & Ying, 2010](#); [Ying et al., 2004](#)), with a common underlying algorithmic and software machinery that can be optimised in a kernel-independent manner.

The power of FMMs is derived from degenerate approximations of the kernel function such that (1) when evaluated between distant clusters of distant source and target points can be expressed a short sum,

$$\phi(x_i) = \sum_{p=1}^P A(x_i) B(y_j) q(y_j) \quad (3)$$

where  $P$ , which we call the ‘expansion order’, is chosen such that  $P \ll M$ ,  $P \ll N$ , and the functions  $A$  and  $B$  are determined by the particular approximation scheme of an FMM method. In the original presentation the calculation of,

$$\hat{q} = B(y_j)q(y_j)$$

corresponds to the construction of analytical expansions of the kernel function representing the potential due to the source densities at a set of source points, and the calculation,

$$\phi = A(x_i)\hat{q}$$

represents the evaluation of this potential at a set of target points. The approximation (3) can be formed when the distance between clusters of sources and targets is sufficient, or *admissible*. By splitting (1) for a given target cluster into *near* and *far* components, the latter of which are taken to be admissible and can be approximated by (3),

$$\phi(x_i) = \sum_{y_j \in \text{Near}(x_i)} K(x_i, y_j)q_j + \sum_{y_j \in \text{Far}(x_i)} K(x_i, y_j)q_j \quad (4)$$

with the near component evaluated directly using the kernel function  $K(.,.)$ . This split, in addition to a recursive procedure through a hierarchical data structure, commonly an octree in 3D, used to discretise the problem domain gives rise to the linear/log-linear complexity of the FMM, as the number of far-field interactions which are admissible is limited by a constant depending on the problem dimension, commonly referred to as the multipole to local field translation (M2L) operation. The near field operations are computed directly using the kernel function  $K(.,.)$ , commonly referred to as the particle to particle (P2P) operation. These two operations conspire to dominate runtimes in practical implementations.

## Statement of need

Previous high-performance codes for computing kiFMMs include (Malhotra & Biros, 2015; Wang et al., 2021). However, both of these efforts are packaged as opaque heavily templated C++ libraries, with brittle optimisations for the M2L and P2P operations that make it complex for users or new developers to exchange or experiment with new algorithmic or implementation ideas that improve runtime performance. Furthermore, it is not possible to easily compare whether performance has been attained by a specific algorithmic approach, or readily deploy the software on new hardware platforms due to reliance on hand written CMake based builds. Novice users are provided with Python bindings in (Wang et al., 2021), however the state of the art distributed code is only accessible via C++ (Malhotra & Biros, 2015). Software sub-components such as the hierarchical tree data structures and kernel implementations are not readily re-usable for related algorithmic work by downstream users, and underlying software used in compute kernels such as libraries for BLAS, LAPACK, or the FFT are not readily exchangeable by users to experiment with performance differences across hardware variations.

Our principle contributions with `kifmm-rs` that extend beyond current state of the art implementations are:

- A *highly portable* Rust-based data-oriented software design that allows us to easily test the impact of different algorithmic approaches, and backends for computational kernels such as BLAS libraries, for critical algorithmic sub-components such as the M2L and P2P operations.
- *State of the art* single-node performance enabled by the optimisation of BLAS based M2L field translation, based on level 3 operations with high arithmetic intensity that are

well suited to current and future hardware architectures that prioritise minimal memory movement per flop.

- The ability to process multiple right hand sides corresponding to the same particle distribution using (1), a common application in BEM.
- Full Python bindings for non-specialist users.

kifmm-rs is a core dependency for the BEM library bempt-rs (Betcke, 2024), and we present a detailed exposition behind the algorithmic and implementation approach in (Kailasa et al., 2024). Currently limited to shared memory systems, distributed memory extensions are an area of active development.

## Software design

### Rust as a platform 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.

### Data oriented design with traits

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.

### Extensibility and portability

- How do traits enable us to create an extensible software that is ready for new kernel implementations, distriuted memory implementations, and re-use of subcomponents in related algorithms?

## 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.

**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

	Apple M1 Pro	AMD 3790X
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

## Acknowledgements

Srinath Kailasa is supported by EPSRC Studentship 2417009

## References

- Betcke, M., T. & Scroggs. (2024). *Bempp-rs: Boundary element methods in rust*. <https://github.com/bempp/bempp-rs>
- Cipra, B. A. (2000). The best of the 20th century: Editors name top 10 algorithms. *SIAM News*, 33(4), 1–2.
- Engquist, B., & Ying, L. (2010). Fast directional algorithms for the helmholtz kernel. *Journal of Computational and Applied Mathematics*, 234(6), 1851–1859.
- 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](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>
- Steinbach, O. (2007). *Numerical approximation methods for elliptic boundary value problems: Finite and boundary elements*. Springer Science & Business Media.
- 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.
- 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>