




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

Srinath Kailasa ^{1,2}

¹ Department of Mathematics, University College London, UK ² Department of Engineering, University of Cambridge, 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Published: unpublished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

kifmm-rs is an open-source framework for implementing Fast Multipole Methods (FMMs) in shared and distributed memory in three dimensions. FMMs accelerate the calculation of N -body potential evaluation problems that arise in computational science and engineering of the form,

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

where the potential ϕ is evaluated at a set of target particles, $\{x_i\}_{i=1}^M$, due to a set of densities, $\{q_j\}_{j=1}^N$ collocated at a set of source particles $\{y_j\}_{j=1}^N$. This evaluation can be interpreted as a matrix-vector multiplication,

$$\phi = \mathbf{K} \mathbf{q} \quad (2)$$

where ϕ is an M dimensional vector, \mathbf{K} is a dense $M \times N$ matrix and \mathbf{q} is an N dimensional vector. As \mathbf{K} is dense, the naive evaluation of (2) is of $O(M \cdot N)$ complexity. FMMs provide a way of evaluating (2) in $O(P(M + N))$ where P is a user defined parameter $P \ll M, N$ called the *expansion order*, for a range of interaction kernels $K(\cdot)$ that commonly arise from elliptic partial differential equations (PDEs). The prototypical example for which the FMM was first presented is the Laplace kernel ([Greengard & Rokhlin, 1987](#)), which describes particles interacting electrostatically or gravitationally,

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

Since their initial introduction FMMs have been developed for a wide variety of kernel functions such as the Helmholtz kernel which arises from the time-independent wave equation ([Rokhlin, 1993](#)),

$$K(x, y) = \begin{cases} \frac{i}{4} H_0^{(1)}(k|\mathbf{x} - \mathbf{y}|), & 2D \\ \frac{e^{ik|\mathbf{x}-\mathbf{y}|}}{4\pi|\mathbf{x}-\mathbf{y}|}, & 3D \end{cases} \quad (4)$$

As well as the Stokes kernel describing fluid flow ([Tornberg & Greengard, 2008](#)), and the Navier kernel used in elastostatics ([Fu et al., 1998](#)) among others. The major application of FMMs is found in the accelerated application of dense operator matrices that arise from the

27 integral formulation of partial differential equations, and as a result FMM algorithms are a
28 crucial component across simulations in many domains, from medical imaging to geophysics.

29 FMMs accelerate the evaluation of (2) by decomposing the problem domain using a hierarchical
30 tree, a quadtree in 2D and an octree in 3D, the algorithm consisting of a series of operations
31 collectively referred to as *field translations*. Evaluations of the off-diagonal blocks of the matrix
32 \mathbf{K} in (2) correspond to the ‘multipole to local’ or M2L field translation which summarise the
33 long-range interactions of a local cluster of target points, and the evaluation of the diagonal
34 blocks correspond to unavoidable direct evaluations of the kernel function for evaluating
35 near-range interactions. M2L is the most challenging optimisation bottleneck in FMMs due to
36 its memory-bound nature.

37 Many parts of an implementation are common across FMM algorithms, such as the tree setup
38 and the kernel function implementation. As a result, our framework is presented as a set
39 of modular, re-usable, sub-components of each of the key algorithmic sub-components: (i)
40 the tree in shared and distributed memory, (ii) the field translation operations and (iii) the
41 kernel evaluation. We use our framework to develop an implementation of the so-called kernel
42 independent Fast Multiple Method (kiFMM) (Ying et al., 2004), compatible with a wide variety
43 of elliptic PDE kernels with an implementation for the Laplace and Helmholtz kernels provided
44 currently. The kiFMM has favourable implementation properties due to its formulation in
45 terms of high operational-intensity BLAS operations.

46 Statement of need

47 Other notable software efforts for the FMM include PVFMM (Malhotra & Biros, 2015),
48 ExaFMM (Wang et al., 2021), ScalFMM (Blanchard et al., 2015) and TBFMM (Bramas,
49 2020). PVFMM, ScalFMM and TBFMM are fully distributed implementations of the black
50 box FMM (Fong & Darve, 2009) and kiFMM respectively. The latter two are notable for
51 being distributed using a task-based runtime, with the former using a more traditional MPI
52 based approach. ExaFMM offers a shared memory implementation of the kiFMM, with Python
53 interfaces and a simple template based design. A commonality of previous implementations is the
54 coupling of algorithmic optimisation with implementation optimisation. For example, ExaFMM
55 and PVFMM both offer field translations that are highly optimised for x64 architectures and
56 lack ARM support, with ScalFMM and TBFMM being tailored to the runtime systems they
57 are designed for.

58 Our design is data oriented, with complex behaviour composed over simple linear data structures
59 using Rust’s trait system. Traits offer a way of specifying behaviour defining contracts between
60 types that are enforced by Rust’s compiler. This enables the exposure of performance critical
61 sections in a manner that is easy to optimise in isolation. In contrast to previous software
62 efforts, our design enables a decoupling of the underlying algorithmic implementation and the
63 software optimisation. This has enabled the comparative analysis of the implementation of the
64 critical M2L field translation (Kailasa et al., 2024), and the future iterative refinement of field
65 translations in response to algorithmic and hardware advances.

66 Our principle contributions with `kifmm-rs` can therefore be summarised as:

- 67 ■ *A modular data-oriented design* which enables field translations to be implementd over
68 simple linear data structures, allowing us to easily examine the impact of different
69 algorithmic approaches and computational backends, such as BLAS libraries, for critical
70 algorithmic sub-components.
- 71 ■ *Optimisations for ARM and x86 targets.* ARM targets are increasingly common in
72 high-performance computing systems, with portability enabled by Rust’s LLVM based
73 compiler.
- 74 ■ *Competitive shared and distributed memory performance.* With state of the art M2L
75 performance (Kailasa et al., 2024), as well as a communication-optimal distributed
76 memory implementation inspired by (Ibeid et al., 2016).

77 ▪ A C API, using Rust's C ABI compatibility allowing for the construction of bindings into
 78 other languages, with full Python bindings for non-specialist users.
 79 ▪ A moderate frequency Helmholtz FMM. Helmholtz FMMs are often presented for the
 80 low-frequency case Malhotra & Biros (2015), due to the challenging data sizes involved.
 81 We present an extension of the kiFMM to the Helmholtz problem which has proven so
 82 far to be effective in single precision for relatively high wave numbers of up to $k \sim 100$.

83 kifmm-rs is currently a core library used as a part of the Bempp boundary element project
 84 (Betcke & Scroggs, 2024). A full API description is available as a part of published [document-](#)
 85 [tation](#). [Python](#), [Rust](#) and [C](#) examples can be found in the repository.

86 Acknowledgements

87 Srinath Kailasa is supported by EPSRC Studentship 2417009

88 References

- 89 Betcke, T., & Scroggs, M. (2024). *Bempp-rs: Boundary element methods in rust*. <https://github.com/bempp/bempp-rs>
 90 Blanchard, P., Bramas, B., Coulaud, O., Darve, E., Dupuy, L., Etcheverry, A., & Sylvand, G. (2015). ScalFMM: A generic parallel fast multipole library. *SIAM Conference on Computational Science and Engineering (SIAM CSE 2015)*.
 91 Bramas, B. (2020). TBFMM: A c++ generic and parallel fast multipole method library. *Journal of Open Source Software*, 5(56), 2444.
 92 Fong, W., & Darve, E. (2009). The black-box fast multipole method. *Journal of Computational Physics*, 228(23), 8712–8725.
 93 Fu, Y., Klimkowski, K. J., Rodin, G. J., Berger, E., Browne, J. C., Singer, J. K., Van De Geijn, R. A., & Vemaganti, K. S. (1998). A fast solution method for three-dimensional many-particle problems of linear elasticity. *International Journal for Numerical Methods in Engineering*, 42(7), 1215–1229.
 94 Greengard, L., & Rokhlin, V. (1987). A fast algorithm for particle simulations. *Journal of Computational Physics*, 73(2), 325–348.
 95 Ibeid, H., Yokota, R., & Keyes, D. (2016). A performance model for the communication in fast multipole methods on high-performance computing platforms. *International Journal of High Performance Computing Applications*, 30(4), 423–437. <https://doi.org/10.1177/1094342016634819>
 96 Kailasa, S., Betcke, T., & Kazdadi, S. E. (2024). M2L translation operators for kernel independent fast multipole methods on modern architectures. *arXiv Preprint arXiv:2408.07436*.
 97 Malhotra, D., & Biros, G. (2015). PVFMM: A parallel kernel independent FMM for particle and volume potentials. *Communications in Computational Physics*, 18(3), 808–830.
 98 Rokhlin, V. (1993). Diagonal forms of translation operators for the helmholtz equation in three dimensions. *Applied and Computational Harmonic Analysis*, 1(1), 82–93.
 99 Tornberg, A.-K., & Greengard, L. (2008). A fast multipole method for the three-dimensional stokes equations. *Journal of Computational Physics*, 227(3), 1613–1619.
 100 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.

- ¹¹⁹ Ying, L., Biros, G., & Zorin, D. (2004). A kernel-independent adaptive fast multipole algorithm
¹²⁰ in two and three dimensions. *Journal of Computational Physics*, 196(2), 591–626.

DRAFT