DOI: 10.21105/joss.03145

Software

- Review
- Repository
- Archive

Editor: Jack Poulson

Reviewers:

- @berenger-eu
- @pitsianis

Submitted: 05 March 2021
Published: 30 May 2021

ExaFMM: a high-performance fast multipole method library with C++ and Python interfaces

Tingyu Wang[1], **Rio Yokota**[2], **and Lorena A. Barba**[1]

1 The George Washington University **2** Tokyo Institute of Technology

Summary

ExaFMM is an open-source library for fast multipole algorithms, providing high-performance evaluation of N-body problems in three dimensions, with C++ and Python interfaces. This new implementation is the most recent of many across a decade of work in our research group. Our goal for all these years has been to produce reusable, standard code for this intricate algorithm. The new header-only C/C++ implementation is easier to extend, still competitive with state-of-the-art codes, and includes a `pybind11` Python interface (Jakob et al., 2017).

The fast multipole method (FMM) was introduced more than 30 years ago (Greengard & Rokhlin, 1987) as a means of reducing the complexity of N-body problems from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$ using hierarchical approximations of long-range interactions. Two major variants of hierarchical N-body algorithms exist: treecodes and FMM. (Algebraic analogues also exist, such as H-matrix methods.) Both were originally developed for fast evaluation of the gravitational potential field, but now have found many applications in different fields. For example, the integral formulation of problems modeled by elliptic partial differential equations can be reinterpreted as N-body interactions. In this way, N-body algorithms are applicable to acoustics, electromagenetics, fluid dynamics, and more. The present version of ExaFMM implements the so-called kernel-independent variant of FMM (Ying et al., 2004). It supports computing both the potential and forces of Laplace, low-frequency Helmholtz and modified Helmholtz (a.k.a. Yukawa) kernels. Users can add other non-oscillatory kernels with modest programming effort.

Statement of Need

Over the past three decades, a plethora of fast N-body implementations have emerged. We will mention a few notable ones for context. `Bonsai` (Bédorf et al., 2012) is a gravitational treecode that runs entirely on GPU hardware. `ChaNGa` (Jetley et al., 2008) is also a treecode that uses `Charm++` to automate dynamic load balancing. `ScalFMM` (Agullo et al., 2014) implements the black-box FMM, a kernel-independent variant based on interpolation. It comes with an option to use `StarPU` runtime system to handle heterogeneous task scheduling. `TBFMM` (Bramas, 2020) is a task-based FMM library that features a generic C++ design to support various types of tree structures and kernels, through heavy use of C++ templates. `PVFMM` (Malhotra & Biros, 2015) can compute both particle and volume potentials using a kernel-independent FMM, KIFMM (Ying et al., 2004).

The first version of ExaFMM focused on low-accuracy optimizations and implemented a dual-tree traversal (Barba & Yokota, 2012; Yokota, 2013; Yokota & Barba, 2012). It was GPU-enabled using CUDA, parallel with MPI and exploited multithreading using OpenMP. Despite all these efforts, it has remained a challenge in the FMM community to have a well-established open-source software package, analogous to FFTW for the fast Fourier transform, delivering compelling performance with a standard and easy-to-use interface.

The "alpha" version of ExaFMM is long and complex, and hard to maintain. Its length is partly due to a focus on fast execution, which led to specialized treatment of low-p evaluations and extensive hand-coded optimizations. This emphasis on achieving high performance led to poor reusability and maintainability. The current version uses the kernel-independent variant of the method for higher performance at high accuracy (higher truncation order p). Keeping focus on maintainability, it stays close to standards, with clean function interfaces, shallow inheritance and conservative use of classes. Instead of a fixation with being the fastest of all, we focus on "ballpark" competitive performance. Above all, the Python API and ability to call it from Jupyter notebooks should make this software valuable for many applications.

Features of the software design

`exafmm-t` is designed to be standard and lean. First, it only uses C++ STL containers and depends on mature math libraries: BLAS, LAPACK and FFTW3. Second, `exafmm-t` is moderately object-oriented, namely, the usage of encapsulation, inheritance and polymorphism is conservative or even minimal in the code. The core library consists of around 6,000 lines of code, which is an order of magnitude shorter than many other FMM packages.

`exafmm-t` is concise but highly optimized. To achieve competitive performance, our work combines techniques and optimizations from several past efforts. On top of multi-threading using OpenMP, we further speed up the P2P operator (near-range interactions) using SIMD vectorization with SSE/AVX/AVX-512 compatibility; we apply the cache optimization proposed in PVFMM (Malhotra & Biros, 2015) to improve the performance of M2L operators (far-range interactions). In addition, `exafmm-t` also allows users to pre-compute and store translation operators, which benefits applications that require iterative FMM evaluations. The single-node performance of `exafmm-t` is on par with the state-of-the-art packages that we mentioned above. We ran a benchmark that solves a Laplace N-body problem with 1 million randomly distributed particles on a workstation with a 14-core Intel i9-7940X CPU. It took 0.95 and 1.48 seconds to obtain 7 and 10 digits of accuracy on the potential, respectively.

`exafmm-t` is also relatively easy to extend. Adding a new kernel only requires users to create a derived `FMM` class and provide the kernel function. Last but not least, it offers high-level Python APIs to support Python applications. Thanks to `pybind11`, most STL containers can be automatically converted to Python-native data structures. Since Python uses duck typing, we have to expose overloaded functions to different Python objects. To avoid naming collisions and keep a clean interface, we chose to create a Python module for each kernel under `exafmm-t`'s Python package, instead of adding suffixes to function and class names to identify types.

Application

We have recently integrated `exafmm-t` with `Bempp-cl`, an open-source boundary element method (BEM) package in Python (Betcke & Scroggs, 2021), whose predecessor, `BEM++` (Śmigaj et al., 2015), has enabled many acoustic and electromagnetic applications. In BEM applications, computations are dominated by the dense matrix-vector multiplication (mat-vec) in each iteration. `exafmm-t` reduces both time and memory cost of mat-vec to a linear complexity, thus makes `Bempp-cl` feasible to solve large-scale problems. In an upcoming paper, we demonstrate the capabilities and performance of Bempp-Exafmm on biomolecular electrostatics simulations, including solving problems at the scale of a virus (Wang et al., 2021). The showcase calculation in that paper (submitted) obtains the surface electrostatic potential of a Zika virus, modeled with 1.6 million atoms, 10 million boundary elements (30M points), at a runtime of 1.5 hours on 1 CPU node.

Acknowledgements

This software builds on efforts over more than a decade. It directly received support from grants to LAB in both the UK and the US, including EPSRC Grant EP/E033083/1, and NSF Grants OCI-0946441, NSF CAREER OAC-1149784, and CCF-1747669. Other support includes faculty start-up funds at Boston University and George Washington University, and NVIDIA via hardware donations.

References

Agullo, E., Bramas, B., Coulaud, O., Darve, E., Messner, M., & Takahashi, T. (2014). Task-based FMM for multicore architectures. *SIAM Journal on Scientific Computing*, *36*(1), C66–C93. https://doi.org/10.1137/130915662

Barba, L. A., & Yokota, R. (2012). *ExaFMM: An open source library for Fast Multipole Methods aimed towards Exascale systems*. Poster on **Figshare**, under CC-BY license, https://dx.doi.org/10.6084/m9.figshare.92166.v1. https://doi.org/10.6084/m9.figshare.92166.v1

Betcke, T., & Scroggs, M. W. (2021). Bempp-cl: A fast python based just-in-time compiling boundary element library. *Journal of Open Source Software*, *6*(59), 2879. https://doi.org/10.21105/joss.02879

Bédorf, J., Gaburov, E., & Portegies Zwart, S. (2012). A sparse octree gravitational N-body code that runs entirely on the GPU processor. *Journal of Computational Physics*, *231*(7), 2825–2839. https://doi.org/10.1016/j.jcp.2011.12.024

Bramas, B. (2020). TBFMM: A C++ generic and parallel fast multipole method library. *Journal of Open Source Software*, *5*(56), 2444. https://doi.org/10.21105/joss.02444

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

Jakob, W., Rhinelander, J., & Moldovan, D. (2017). *pybind11 – seamless operability between c++11 and python*.

Jetley, P., Gioachin, F., Mendes, C., Kale, L. V., & Quinn, T. (2008). Massively parallel cosmological simulations with ChaNGa. *2008 IEEE International Symposium on Parallel and Distributed Processing*, 1–12. https://doi.org/10.1109/IPDPS.2008.4536319

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

Śmigaj, W., Betcke, T., Arridge, S., Phillips, J., & Schweiger, M. (2015). Solving Boundary Integral Problems with BEM++. *ACM Transactions on Mathematical Software*, *41*(2), 1–40. https://doi.org/10.1145/2590830

Wang, T., Cooper, C. D., Betcke, T., & Barba, L. A. (2021). *High-productivity, high-performance workflow for virus-scale electrostatic simulations with Bempp-Exafmm*. http://arxiv.org/abs/2103.01048

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

Yokota, R. (2013). An FMM Based on Dual Tree Traversal for Many-Core Architectures. *Journal of Algorithms & Computational Technology*, *7*(3), 301–324. https://doi.org/10.1260/1748-3018.7.3.301

Yokota, R., & Barba, L. A. (2012). A tuned and scalable fast multipole method as a pre-eminent algorithm for exascale systems. *The International Journal of High-Performance Computing Applications*. https://doi.org/10.1177/1094342011429952