The Journal of Open Source Software

mpi4py-fft: Parallel Fast Fourier Transforms with MPI for Python

Mikael Mortensen[1], **Lisandro Dalcin**[2], **and David Elliot Keyes**[2]

1 University of Oslo, Department of Mathematics **2** King Abdullah University of Science and Technology, Extreme Computing Research Center

DOI: 10.21105/joss.01340

Software

- Review
- Repository
- Archive

Submitted: 06 March 2019
Published: 02 April 2019

Summary

The fast Fourier transform (FFT) is an algorithm that efficiently computes the discrete Fourier transform. Since the dawn of our digital society, the FFT has permeated to the heart of everyday life applications involving audio, image, and video processing. The importance of the FFT extends beyond signal processing into scientific computing because it diagonalizes the Poisson operator, which is ubiquitous in descriptions of electromagnetism, gravitation, acoustic and elastic waves, potential flow in fluids, stress in solids, Hamiltonians of the Schroedinger's equation for probability distribution of electrons in quantum mechanics, and diffusion of internal energy, chemical species, and momentum. The FFT has been named *the most important numerical algorithm of our time* by Professor Gilbert Strang (Strang, 1994).

`mpi4py-fft` (https://bitbucket.org/mpi4py/mpi4py-fft) is an open-source Python package for computing (in parallel) FFTs of possibly very large and distributed multidimensional arrays. A multidimensional FFT is computed in sequence, over all axes, one axis at the time. In order to fit in the memory of multiple processors, multidimensional arrays have to be distributed along some, but not all, of its axes. Consequently, parallel FFTs are computed with successive sequential (serial) transforms over undivided axes, combined with global array redistributions (using interprocess communication) that realign the arrays for further serial transforms.

For global redistributions, `mpi4py-fft` makes use of a new and completely generic algorithm (Dalcin, Mortensen, & Keyes, 2019) based on advanced MPI features that allows for any index sets of a multidimensional array to be distributed. It can distribute a single index set (slab decomposition), two index sets (pencil decomposition), or even more for higher-dimensional arrays. The required MPI communications are always handled under the hood by MPI for Python (Dalcin, Paz, Storti, & D'Elia, 2008). For serial FFT transforms, `mpi4py-fft` uses Cython (Behnel et al., 2011) to wrap most of the FFTW library (Frigo & Johnson, 2005) and provide support for complex-to-complex, real-to-complex, complex-to-real and real-to-real transforms.

`mpi4py-fft` is highly configurable in how it distributes and redistributes arrays. Large arrays may be globally redistributed for alignment along any given axis, whenever needed by the user. This flexibility has enabled the development of `shenfun` (Mortensen, 2018, Mortensen (2017)), which is a Python framework for solving partial differential equations (PDEs) by the spectral Galerkin method. `shenfun` is able to solve PDEs of any given dimensionality by creating tensor product bases as outer products of one-dimensional bases. This leads to large multidimensional arrays that are distributed effortlessly through `mpi4py-fft`. Throughout the `spectralDNS` (https://github.com/

spectralDNS/spectralDNS) project `shenfun` is being used extensively for Direct Numerical Simulations (DNS) of turbulent flows (Mortensen & Langtangen, 2016, Mortensen (2016), Ketcheson, Mortensen, Parsani, & Schilling (2019)), using arrays with billions of unknowns.

`mpi4py-fft` provides a flexible distributed NumPy array interface, which allows for further reuse in applications beyond the FFT. The distribution requires at least one undivided axis, but apart from that there are no restrictions nor limitations. The interface can simply be used to boost performance of global array operations through MPI.

Acknowledgements

M Mortensen acknowledges support from the 4DSpace Strategic Research Initiative at the University of Oslo.

L Dalcin and D Keyes acknowledge support from the Extreme Computing Research Center and the KAUST Supercomputing Laboratory at King Abdullah University of Science and Technology.

References

Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D. S., & Smith, K. (2011). Cython: The best of both worlds. *Comput. in Science and Engg.*, *13*(2), 31–39. doi:10.1109/MCSE.2010.118

Dalcin, L., Mortensen, M., & Keyes, D. E. (2019). Fast parallel multidimensional FFT using advanced MPI. *J. Parallel Distrib. Comput.*, *128*, 137–150. doi:10.1016/j.jpdc.2019.02.006

Dalcin, L., Paz, R., Storti, M., & D'Elia, J. (2008). MPI for Python: Performance improvements and MPI-2 extensions. *J. Parallel Distrib. Comput.*, *68*(5), 655–662. doi:10.1016/j.jpdc.2007.09.005

Frigo, M., & Johnson, S. G. (2005). The design and implementation of FFTW3. *Proceedings of the IEEE*, *93*(2), 216–231. doi:10.1109/JPROC.2004.840301

Ketcheson, D., Mortensen, M., Parsani, M., & Schilling, N. (2019). More efficient time integration for Fourier pseudo-spectral DNS of incompressible turbulence. *Int J Numer Meth Fluids*, *in press*. Retrieved from https://arxiv.org/abs/1810.10197

Mortensen, M. (2016). Massively parallel implementation in Python of a pseudo-spectral DNS code for turbulent flows. *Proceedings of EuroScipy 2015*. Retrieved from https://arxiv.org/pdf/1607.00850.pdf

Mortensen, M. (2017). Shenfun - automating the spectral Galerkin method. In B. H. Skallerud & H. I. Andersson (Eds.), *MekIT'17 - ninth national conference on computational mechanics* (pp. 273–298). Int Center for Numerical Methods in Engineering (CIMNE).

Mortensen, M. (2018). Shenfun: High performance spectral Galerkin computing platform. *J. Open Source Software*, *3*(31), 1071. doi:10.21105/joss.01071

Mortensen, M., & Langtangen, H. P. (2016). High performance Python for direct numerical simulations of turbulent flows. *Comput. Phys. Comm.*, *203*, 53–65. doi:10.1016/j.cpc.2016.02.005

Strang, G. (1994). Wavelets. *American Scientist*, *82*(3), 250–255. Retrieved from http://www.jstor.org/stable/29775194