




kokkos-fft: A shared-memory FFT for the Kokkos ecosystem

Yuuichi Asahi^{1*}, Thomas Padioleau^{1*}, Paul Zehner^{1*}, Julien Bigot^{1*}, and Damien Lebrun-Grandie^{2*}

¹ Université Paris-Saclay, UVSQ, CNRS, CEA, Maison de la Simulation, 91191, Gif-sur-Yvette, France ² Oak Ridge National Laboratory, Oak Ridge, Tennessee, US * These authors contributed equally.

DOI: [10.21105/joss.08391](https://doi.org/10.21105/joss.08391)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Daniel S. Katz](#) 

Reviewers:

- [@thivinanandh](#)
- [@VasanthRajendran](#)

Submitted: 06 June 2025

Published: 30 July 2025

License

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

In partnership with



AMERICAN
ASTRONOMICAL
SOCIETY

This article and software are linked with research article DOI [10.3847/xxxxx](https://doi.org/10.3847/xxxxx) <- [update this with the DOI from AAS once you know it.](#), published in the *Astrophysical Journal* <- The name of the AAS journal..

Summary

kokkos-fft provides a unified, performance-portable interface for Fast Fourier Transforms (FFTs) within the Kokkos ecosystem ([C. Trott et al., 2021](#)). It seamlessly integrates with leading local FFT libraries including FFTW, cuFFT, rocFFT, and oneMKL. Designed for simplicity and efficiency, kokkos-fft offers a user experience akin to `numpy.fft` for in-place and out-of-place transforms, while leveraging the raw speed of vendor-optimized libraries. A demonstration solving 2D Hasegawa-Wakatani turbulence with the Fourier spectral method illustrates how kokkos-fft can deliver significant speedups over Python-based alternatives without drastically increasing code complexity, empowering researchers to perform high-performance FFTs simply and effectively.

Statement of need

The fast Fourier transform (FFT) is a family of fundamental algorithms that is widely used in scientific computing and other areas ([Rockmore, 2000](#)). `kokkos-fft` is designed to help *Kokkos* ([C. R. Trott et al., 2022](#)) users who are:

- developing a Kokkos application that relies on FFT libraries, e.g., fluid simulation codes with periodic boundaries, plasma turbulence, etc.
- inclined to integrate in-situ signal and image processing with FFTs, e.g., spectral analyses, low pass filtering, etc.
- willing to use de facto standard FFT libraries just like `numpy.fft` ([Harris et al., 2020](#)).

kokkos-fft can benefit such users through the following features:

- A simple interface like `numpy.fft` with in-place and out-of-place transforms: Only accepts *Kokkos Views* that correspond to the `numpy.array`, to make APIs simple and safe.
- 1D, 2D, 3D standard and real FFT functions (similar to `numpy.fft`) over 1D to 8D Kokkos Views: Batched plans are automatically used if View dimension is larger than FFT dimension.
- A reusable *FFT plan* that wraps the vendor libraries for each Kokkos backend: *FFTW*, *cuFFT*, *rocFFT*, and *oneMKL* are automatically enabled based on the enabled Kokkos backend.
- Support for multiple CPU and GPU backends: FFT libraries for the enabled Kokkos backend are executed on the stream/queue used in

the [ExecutionSpace](#) where the parallel operations are performed.

- Compile time and/or runtime errors for invalid usage (e.g., View extents mismatch).

A couple of libraries that offer common APIs over performant vendor FFT libraries already exist. Relying on data structures in Python, these APIs are offered by a dedicated FFT library like FluidFFT ([Mohan et al., 2019](#)) or a more general library offering GPU acceleration like Jax ([Bradbury et al., 2018](#)). In C++, offering this kind of APIs is non-trivial because of the lack of standard data structures with extents and/or data locations. Thanks to [Kokkos Views](#) and [ExecutionSpace](#), we can offer simple and safe APIs, which is the unique feature of this library.

How to use kokkos-fft

For those who are familiar with `numpy.fft`, you may use kokkos-fft quite easily. In fact, all of the `numpy.fft` functions (`numpy.fft.<function_name>`) have an analogous counterpart in kokkos-fft (`KokkosFFT::<function_name>`), which can run on the Kokkos device. In addition, kokkos-fft supports [in-place transform](#) and [plan reuse](#) capabilities.

Let's start with a simple example to perform the 1D real to complex transform using `rfft` in kokkos-fft.

```
#include <Kokkos_Core.hpp>
#include <Kokkos_Random.hpp>
#include <KokkosFFT.hpp>
int main(int argc, char* argv[]) {
    Kokkos::ScopeGuard guard(argc, argv);
    const int n = 4;
    Kokkos::View<double*> x("x", n);
    Kokkos::View<Kokkos::complex<double>*> x_hat("x_hat", n/2+1);
    // initialize the input array with random values
    Kokkos::DefaultExecutionSpace exec;
    Kokkos::Random_XorShift64_Pool<> random_pool(/*seed=*/12345);
    Kokkos::fill_random(exec, x, random_pool, /*range=*/1.0);
    KokkosFFT::rfft(exec, x, x_hat);
    // block the current thread until all work enqueued into exec is finished
    exec.fence();
}
```

This is equivalent to the following Python code.

```
import numpy as np
x = np.random.rand(4)
x_hat = np.fft.rfft(x)
```

There are two additional arguments in the Kokkos version:

- `exec`: [Kokkos execution space instance](#) that encapsulates the underlying compute resources (e.g., CPU cores, GPU devices) where the task will be dispatched for execution.
- `x_hat`: [Kokkos Views](#) where the complex-valued FFT output will be stored. By accepting this view as an argument, the function allows the user to pre-allocate memory and optimize data placement, avoiding unnecessary allocations and copies.

Also, kokkos-fft only accepts [Kokkos Views](#) as input data. The accessibility of a View from ExecutionSpace is statically checked and will result in a compilation error if not accessible. See [documentations](#) for basic usage.

Benchmark: 2D Hasegawa-Wakatani turbulence with the Fourier spectral method

As a more scientific example, we solve a typical 2D plasma turbulence model, called the Hasegawa-Wakatani equation (Wakatani & Hasegawa, 1984) using the Fourier spectral method (see Figure 1 for the vorticity structure).

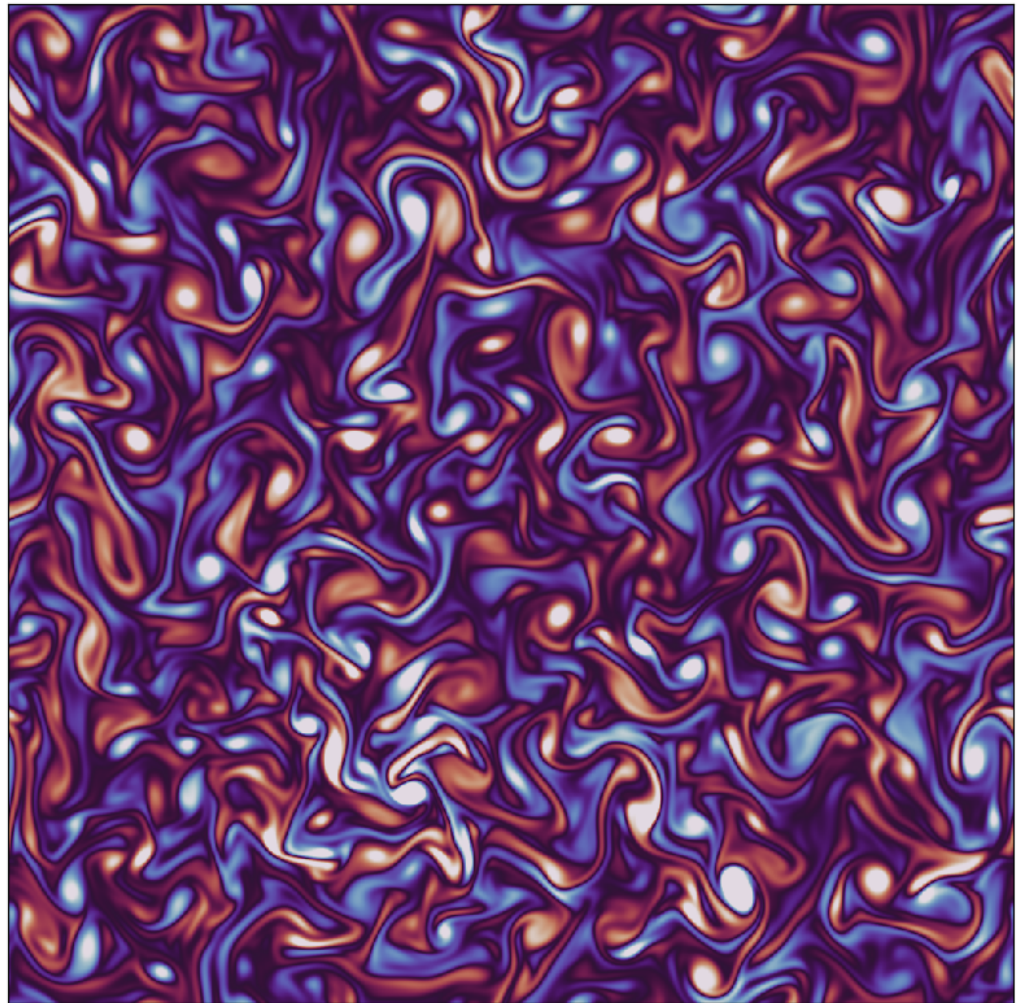


Figure 1: Vorticity.

Using Kokkos and kokkos-fft, we can easily implement the code (see [example](#)), just like Python, while getting a significant acceleration. The core computational kernel of the code is the nonlinear term which is computed with FFTs. We construct the forward and backward FFT plans once during initialization which are reused in the time evolution loops.

We have performed a benchmark of this application over multiple backends. We performed a simulation for 100 steps with a resolution of 1024×1024 while I/O is disabled. The following table shows the achieved performance.

Device	Icelake (python)	Icelake	A100	H100	MI250X	PVC
Kokkos Backend	-	OpenMP	CUDA	CUDA	HIP	SYCL
LOC	568	738	738	738	738	738
Com- piler/ver- sion	Python 3.12.3	IntelLLVM 2023.0.0	nvcc 12.2	nvcc 12.3	rocm 5.7	IntelLLVM 2024.0.2
GB/s (Theoreti- cal peak)	205	205	1555	3350	1600	3276.8
Elapsed time [s]	463	9.28	0.25	0.14	0.41	0.30

Here, the testbed includes Intel Xeon Platinum 8360Y (referred to as Icelake), NVIDIA A100 and H100 GPUs, AMD MI250X GPU (1 GCD) and Intel Data Center GPU Max 1550 (referred to as PVC). On Icelake, we use 36 cores with OpenMP parallelization. As expected, the Python version is the simplest in terms of lines of code (LOC). With Kokkos and kokkos-fft, the same logic can be implemented without significantly increasing the source code size (roughly 1.5 times longer). However, the benefit is enormous: a single and simple code that runs on multiple architectures efficiently.

Acknowledgements

This work has received support by the CExA Moonshot project of the CEA [cexa-project](#). This work was carried out using FUJITSU PRIMERGY GX2570 (Wisteria/BDEC-01) at the University of Tokyo. This work was partly supported by JHPCN project jh220036. This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. This work was also granted access to the HPC resources of CINES under the allocation 2023-cin4492 made by GENCI.

References

- Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., & Zhang, Q. (2018). *JAX: Composable transformations of Python+NumPy programs* (Version 0.3.13). <http://github.com/jax-ml/jax>
- Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., ... Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825), 357–362. <https://doi.org/10.1038/s41586-020-2649-2>
- Mohanan, A. V., Bonamy, C., & Augier, P. (2019). FluidFFT: Common API (C++ and Python) for Fast Fourier Transform HPC libraries. *Journal of Open Research Software*. <https://doi.org/10.5334/jors.238>
- Rockmore, D. N. (2000). The FFT: An algorithm the whole family can use. *Computing in Science & Engineering*, 2(1), 60–64. <https://doi.org/10.1109/5992.814659>
- Trott, C. R., Lebrun-Grandié, D., Arndt, D., Ciesko, J., Dang, V., Ellingwood, N., Gayatri, R., Harvey, E., Hollman, D. S., Ibanez, D., Liber, N., Madsen, J., Miles, J., Poliakoff, D.,

- Powell, A., Rajamanickam, S., Simberg, M., Sunderland, D., Turcksin, B., & Wilke, J. (2022). Kokkos 3: Programming model extensions for the exascale era. *IEEE Transactions on Parallel and Distributed Systems*, 33(4), 805–817. <https://doi.org/10.1109/TPDS.2021.3097283>
- Trott, C., Berger-Vergiat, L., Poliakoff, D., Rajamanickam, S., Lebrun-Grandie, D., Madsen, J., Al Awar, N., Gligoric, M., Shipman, G., & Womeldorff, G. (2021). The Kokkos EcoSystem: Comprehensive performance portability for high performance computing. *Computing in Science & Engineering*, 23(05), 10–18. <https://doi.org/10.1109/MCSE.2021.3098509>
- Wakatani, M., & Hasegawa, A. (1984). A collisional drift wave description of plasma edge turbulence. *The Physics of Fluids*, 27(3), 611–618. <https://doi.org/10.1063/1.864660>