doi:10.1017/S1431927622002409

Microsc. Microanal. 28 (Suppl 1), 2022

Parallelizing py4DSTEM: 4D-STEM Analysis from Hours to Minutes

Alexander Rakowski[*,1], Benjamin H Savitzky[1], Steven E Zeltmann[2], Matthew Henderson[1], and Colin Ophus[1]

[1.] NCEM, Molecular Foundry, Lawrence Berkeley National Laboratory, Berkeley, CA, USA.
[2.] Department of Materials Science and Engineering, University of California, Berkeley, CA, USA.
* Corresponding author: arakowski@lbl.gov

Developments in pixelated cameras have enabled capturing the full 2D diffraction pattern at each probe position at scan speeds comparable to traditional integrating detectors. [1] Furthermore, improvements in microscope instrumentation have enabled scanning over larger sample areas. These developments have enabled a broader range of materials to be integrated and a wide range of physical phenomena to be studied.[2] However, the increased capabilities come with the burden of large dataset sizes; for example, the 4D camera can collect 200 TB hr^{-1} [3]. These large datasets, which may consist of millions of probe positions, are computationally challenging to analyze in two regards: (i) the datasets are often too large to store in system memory, and (ii) operating over a large number of probes can be slow. Overcoming these two challenges is imperative to extract the full potential from the new detector and microscope technologies.

py4DSTEM is an open-source python-based software package for analyzing 4D-STEM datasets and contains a broad set of analysis modalities, including strain and orientation mapping, virtual imaging, and differential phase-contrast imaging. [4] As currently architected, py4DSTEM primarily operates serially, analyzing a single probe position at a time, and typically utilizes only a single central processing unit (CPU) core. However, modern computers have multiple CPU cores and may also contain dedicated graphic processing units (GPUs), which are currently not used in py4DSTEM. Much of the analysis of 4D-STEM datasets is embarrassingly parallel, as the study of each probe position is often independent of the others. There is substantial scope for increasing the speed of analyzing these datasets by utilizing multiple CPU cores and GPU-powered compute.

In this work, we extend the functionality of py4DSTEM by parallelizing portions of the codebase with Dask [5]. The use of Dask allows a compute platform-agnostic parallelism which seamlessly scales from a single laptop to hundreds of nodes at high-performance computing (HPC) facilities with minimal syntactical changes to the code—particularly compared to alternative parallel and scaling programming paradigms such as OpenMP. The implementation is compatible with analyzing data in an interactive python notebook or via a python script. Crucially, Dask operates ‘lazily,’ only loading the data as required. Operating ‘lazily’ enables working on large datasets that cannot fit into system memory, which is a likely scenario when handling 4D-STEM datasets.

We exemplify the new functionality by performing Bragg disk detection, a substantial bottleneck to many 4D-STEM analysis methodologies, on a small (201x50x420x422, ~6.7 GB) and a large (83x558x1792x1920, ~300 GB) 4D-STEM dataset on multiple compute platforms: (i) single workstation (24 CPU cores, 256 GB RAM), (ii) single HPC node (32 CPU cores, 128 GB RAM), and (iii) 4 HPC nodes (128 CPU cores, 512 GB RAM). We show a 5X speed up on the small dataset using the parallelized function compared to the serial equivalent function (2 vs. 10 minutes) on a single HPC node. While the larger dataset, we saw a dramatic improvement from multiple hours when using the serial function to ~15 minutes when processing on 4 HPC nodes. The processing time for the large dataset could likely be reduced further by increasing the number of nodes. We also show that naive implementations of Dask can result in significant performance penalties. This reduction in computation time allows users to iterate parameters while examining their data or analyze other datasets more efficiently. Ultimately we demonstrate that the parallelized py4DSTEM functions are vital for studying the increasingly large datasets produced by the new electron detectors [6].

References:

[1] J Ciston et al., Microscopy and Microanalysis **25**(S2) (2019), p.1930.
[2] C Ophus, Microscopy and Microanalysis **25**(3) (2019), p. 563.
[3] SR Spurgeon et al., Nature materials **20**(3) (2021), p. 274.
[4] BH Savitzky et al., Microscopy and Microanalysis **25**(S2) (2019), p. 124.
[5] Dask: Library for dynamic task scheduling, https://dask.org
[6] Work at the Molecular Foundry was supported by the Office of Science, Office of Basic Energy Sciences, of the U.S. Department of Energy under Contract No. DEAC02-05CH11231. Use of the Center for Nanoscale Materials, an Office of Science user facility, was supported by the U.S. Department of Energy, Office of Science, Office of Basic Energy Sciences, under Contract No. DE-AC02-06CH11357. This research used resources of the National Energy Research Scientific Computing Center; a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. Work at the Molecular Foundry was supported by the Office of Science, Office of Basic Energy Sciences, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. The development of py4DSTEM was supported by the Toyota Research Institute