Scientific Computing With Python on High-Performance Heterogeneous Systems

Lorena A. Barba, *George Washington University, Washington, DC, 20052, USA*
Andreas Klöckner, *University of Illinois at Urbana-Champaign, Champaign, IL, 61801, USA*
Prabhu Ramachandran, *Indian Institute of Technology Bombay, Mumbai, 400076, India*
Rollin Thomas, *National Energy Research Scientific Computing Center, Berkeley, CA, 94720, USA*

The Python ecosystem for science has come a long way since CiSE ran a Special Issue on "Scientific Python" ten years ago (vol. 13, issue 2). We can confidently say that Python is a top language in computational science and engineering, and that the long-standing concerns about *performance* have many answers today. This Special Issue includes six articles that showcase how researchers are using Python in high-performance settings with GPU accelerators and distributed parallel systems. It is a small sample of the breadth of innovation in this field, which has expanded and matured in the last few years to offer computational scientists a choice of tools and frameworks that realize high-performance coupled with high researcher productivity.

We make some key observations of the state and trends in this field:

- Multidevice, multinode scaling is being achieved with communication paradigms like MPI that can handle direct GPU-to-GPU data motion (for instance "CUDA-aware" MPI).
- Single-device, single-node performance in Python apps is being achieved through libraries that generate and compile code specialized for the device.
- Mostly, Python users have picked their two out of three Ps, settling on productivity and performance; portability between accelerator architectures is nascent.
- High-performance I/O libraries such as HDF5 should provide GPU-aware I/O capabilities like those found already in the Python GPU ecosystem (e.g. using Apache Arrow).

1521-9615 © 2021 IEEE
Digital Object Identifier 10.1109/MCSE.2021.3088549
Date of current version 23 July 2021.

Dalcin and Fang give a status update of the last 12 years of development in the mpi4py project, an early effort that brought to Python programs the ability to use parallel distributed computing via message passing. For its 1.0 release, mpi4py was fully rewritten in Cython, offering a new degree of maintainability and extensibility. The next major release brought new features added in the MPI-3 specification; to date, nearly all features up to MPI 3.1 are supported. More recently, mpi4py implemented support for the CUDA array interface protocol, allowing "MPI+CUDA" programming in Python with CUDA-aware MPI libraries. Release 3.0 of mpi4py also added asynchronous task execution, and the upcoming 3.1 release brings support for large-size Python objects. In sum, mpi4py is now a mature project and the *de facto* option for developing MPI-based parallel applications in Python. It is used in h5py, the Python wrapper for the HDF5 library, in the open-source VTK library for image processing and visualization, and in the yt library for analyzing and visualizing volumetric data, among many others.

Bartlett, Uchytil, and Storti document two case studies of exploiting GPUs to accelerate Python code, focusing on what one can do with a single GPU. "High-productivity parallelism with Python plus packages (but without a cluster)" addresses the problem of efficiently exploiting single-node parallelism using Python. In the CPU realm, this problem is challenging enough that for many Python users the default answer is to just scale up with multiprocessing, a multinode cluster runtime, or MPI. But the authors show that for certain common classes of problems, Python already provides relatively easy access to high performance on a single-GPU device with open-source tools. Users new to GPU programming with Python should find this article a helpful pointer along the road to performance.

Integral equation methods represent an attractive approach for the solution of exterior boundary value

problems and many other boundary value problems of elliptic partial differential equations. A particular challenge with these methods is their high computational cost. In this issue, Betcke and Scroggs describe the evolution of BEMPP, a code with comprehensive functionality in this area, from a "conventional" pure C++ code base, to one that had a Python interface, to one based on pure Python coupled with code generation, highlighting in a compelling manner both advantages gained as well as potential challenges. The authors discuss specific techniques for attaining good performance as well as maintaining separation of concerns. Preliminary performance data support the exposition, making the article required reading for academic and professional users of C++ who are considering a similar evolution of code they are maintaining.

In 2016, a team from Imperial College London was nominated for the prestigious ACM Gordon Bell Prize for demonstrating the viability of Python as a platform for productive, performant, and portable HPC applications at petascale.[1] This watershed moment for HPC clearly signaled that Python can compete at the highest levels of scalability and performance. One member of that team, Freddie Witherden, contributes to this issue a considered perspective on Python as a "first-class language" for HPC ("Python at petascale with PyFR or: how I learned to stop worrying and love the snake"). He perceives the trend as being driven by three key factors. First is the increased emphasis on not just application performance, but developer and user productivity with HPC codes. Second is the growing tendency for HPC applications to rely on third-party APIs. The third factor is the increasing use of code generation for addressing performance bottlenecks. Python can address these factors and more, and in doing so puts the highest levels of performance on HPC hardware within the grasp of researchers who need it.

Firedrake[a] is a powerful Python package for solving PDEs using the finite element method. In the article "Code generation for productive portable scalable finite element simulation in Firedrake," Betteridge, Farell, and Ham highlight the advantages of a high-level software framework like Firedrake from the perspective of practical execution on large HPC machines. The authors first demonstrate the high-level pure Python interface using a simple Poisson problem in three-dimensions and then go on to consider a steady incompressible Navier–Stokes problem that they executed on three different HPC systems available in the United Kingdom. They show how the performance of the code may be tuned to the specifics of a particular platform using a host of different approaches including discretization changes as well as algorithmic changes. All of these changes are made at the highest level of the framework using a few lines of Python. They also discuss in considerable detail the scalability of the code on three different architectures. This article is a useful resource for readers to understand both how one may effectively utilize HPC hardware and also understand how good software design can facilitate this with relative ease.

In "Stencil solvers for PDEs on GPUs: An example from cosmology," Weiner explains the design principles of Pystella, a prime example of the vast potential unlocked by code generation, giving researchers access to heterogeneous high-performance systems while experiencing high productivity. Pystella is a framework for stencil solvers targeting partial differential equations (PDEs), which can serve as a backend for simulation drivers written in Python. Its motivating application was the set of nonlinearly coupled hyperbolic PDEs arising from mathematical models in cosmology. Pystella adopts OpenCL as its underlying compute language, uses the Loopy Python package for code transformation and generation, and offers message-passing parallelism via mpi4py. It provides users with a symbolic representation of scalar and vector fields, keeping track of the indexing and offsets for the shared "halo" boundary points between neighboring parallel subdomains. Expressing arbitrary stencil operations is reduced to a few lines of Python and Pystella methods, which include element-wise operations and symbolic differentiation routines. Python programmers in the target domains are granted a path to high-performance execution without having to think about the details of the target hardware and how to extract performance from it.

The authors of this Special Issue were all invited to submit, and each manuscript received three independent peer reviews. We are grateful for the contributions of our expert group of reviewers: Alex Barnett (Flatiron Institute), Steffen Boerm (Kiel University), Jed Brown (University of Colorado Boulder), Jeff Hammond (Nvidia), Kyle Mandli (Columbia University), Thomas Nicholas (Columbia University), Luke Olson (University of Illinois), Graham Pullan (Cambridge University), Andreas Schreiber (German Aerospace Center, DLR), Gopalakrishnan Shivasubramanian (Indian Institute of Technology Bombay), Laurie Stephey (National Energy Research Scientific Computing Center, NERSC), Garth N. Wells (Cambridge University), and Freddie Witherden (Texas A&M University). Without their thoughtful and timely reviews, this issue would not have been possible. Thank you!

In combination with this year's March/April issue, dedicated to Jupyter in Computational Science, this

[a]https://firedrakeproject.org

issue focuses our attention on the modern approach to enabling scientific progress with computers. We cannot escape the mounting complexity of high-performance computing, with rapidly evolving and diverse hardware accelerators. Accessing this computational power via high-productivity languages like Python, however, not only lets us manage this complexity but offers a wide door for entry in the field. A large fraction of STEM students today are learning Python as undergraduates, while the rich community activity of the open source model is helping spread these skills broadly. Python and Jupyter have become a decidedly popular combination in education and industry settings, and are now taking hold in computational science.[2] *We hope reading these articles will whet your appetite!*

REFERENCES

1. P. Vincent, F. Witherden, B. Vermeire, J. S. Park, and A. Iyer, "Towards green aviation with python at petascale," in *Proc. Int. Conf. for High Perform. Comput., Netw., Storage Anal.*, 2016, pp. 1–11, doi: 10.1109/SC.2016.1.
2. L. A. Barba, "The Python/Jupyter ecosystem: Today's problem-solving environment for computational science," *Comput. Sci. Eng.*, vol. 23, no. 3, pp. 5–9, Jul./Aug. 2021. doi: 10.1109/MCSE.2021.3074693.

LORENA A. BARBA is currently a professor of mechanical and aerospace engineering at the George Washington University, Washington, DC, USA. Her research interests include computational fluid dynamics, biophysics, and high-performance computing. Barba received the PhD degree in aeronautics from the California Institute of Technology, Pasadena, CA, USA. She is the Editor-in-Chief of *Computing in Science & Engineering* (CiSE), Co-Editor of the *CiSE Reproducible Research Track*, Associate Editor for the *ReScience* journal, Steering Committee member for the *Journal of Open Source Software*, and Editor-in-Chief of the *Journal of Open Source Education*. For more information, see https://lorenabarba.com. Contact her at labarba@gwu.edu.

ANDREAS KLÖCKNER is currently an associate professor in the scientific computing area with the Department of Computer Science, University of Illinois at Urbana-Champaign, Champaign, IL, USA. His research focuses on high-order accurate integral equation methods and fast algorithms for elliptic boundary value problems as well as code transformation for high-performance scientific computing. In support of his research, he has released numerous scientific software packages, including PyCUDA, PyOpenCL, and Loopy, among others. Contact him at andreask@illinois.edu.

PRABHU RAMACHANDRAN is a member of the faculty with the Department of Aerospace Engineering, and with the Centre for Machine Intelligence and Data Science, Indian Institute of Technology Bombay, Mumbai, India. His research interests include particle methods and meshfree methods for computational fluid dynamics, scientific computing, parallel and high-performance computing, machine learning for numerical computation, and applied scientific data visualization. He has developed many Python-based open-source projects, including Mayavi, and PySPH, among others. Contact him at prabhu@aero.iitb.ac.in.

ROLLIN THOMAS is currently a data architect at the National Energy Research Scientific Computing Center (NERSC), Lawrence Berkeley National Laboratory, Berkeley, CA, USA. He focuses on solving strategic challenges at the interface between user productivity and high-performance computing. He is the originator of NERSC's scalable Python support strategy, the creator of its Exascale Applications Readiness Program for Data, and is the chief architect of NERSC's Jupyter interface to its supercomputing resources. Prior to joining NERSC, in 2015, he was a computational astrophysicist working in the areas of numerical radiative transfer, spectroscopy, supernova cosmology, and computational infrastructure for cosmology experiments. Contact him at rcthomas@lbl.gov.