Using Python for Improved Productivity in HPC and Data Science Applications: the Time is Now

Jess Woods[1], Matthew Baker[1], Mathialakan Thavappiragasam[1], Ada Sedova[1], Oscar Hernandez[1], and Vivek Sarkar[2]

[1] *Oak Ridge National Laboratory*
[2] *Georgia Institute of Technology*

1 Introduction

Developer productivity for scientific software is essential in order to address changing timelines in scientific computing needs and an increasing set of science problems that can benefit from large-scale computational approaches. Generally, computational approaches in science were considered to be long-standing and less time-sensitive; this made them worthy of multi-year efforts as exemplified by DOE SciDAC projects. There is now a newfound urgency to analyze massive quantities of data and perform large calculations for societal benefits. Such research efforts include the biological and environmental sciences, where the complexity and multiple scales of systems, together with growing dataset sizes from high-throughput experimental methods, have created major bottlenecks [1]. Urgent needs include research in sustainability, genomics [2] and healthcare research such as recent responses to the COVID-19 pandemic [3]. In addition, the emergence of data-driven and AI-driven solutions to these urgent problems has placed a higher premium on the use of highly optimized, parallel software.

Improving developer productivity has been a long-standing challenge that is deeply impacted by emerging hardware platforms, workforce expertise, and programming tools. We are currently at a critical point in all three aspects of this challenge. In terms of hardware platforms, the "free ride" of automatic improvements in efficiency and cost from Moore's Law and Dennard Scaling has ended. Sustained improvements now require specialized, post-Moore accelerators and radically different computer architectures. Therefore, high-performance computing (HPC) software development, which has always been the realm of specialized experts, is now becoming even more highly specialized. However, the development of scientific software for domain science is not performed by HPC experts, but by domain scientists with highly specialized knowledge of their modeled systems and the structure of their data. In addition, the current state of programming tools for scientific computing is inadequate for rapid creation of new applications on new hardware platforms, especially for computational scientists who are not HPC experts. In this paper, we advocate for the newly emerging approach to developing scientific software: an HPC-based Python ecosystem.

Python is a widely used high-productivity language, second only to JavaScript in popularity, but far better suited to scientific software than JavaScript. It allows for a fast code production schedule, even for domain scientists who are not HPC experts. The Python community has worked extensively to circumvent Python's performance obstacles by introducing a rich collection of high performance libraries. Python serves several different purposes in HPC scientific software development. It can be used as a wrapper or "glue" to combine different languages and traditional HPC libraries, and to develop workflows and use data science and machine learning toolkits. There are many Python libraries that can be quickly parallelized or offloaded. Python itself can be extended with another programming model for parallelism, for example, CuPy to accelerate NumPy and Dask with asynchronous task-parallelism across multiple GPUs or nodes. Additionally, Python tools like Jupyter Notebook enable interactive programming to steer computations in ways beyond traditional file I/O [4].

2 Motivating Use Cases

This section summarizes multiple use cases where Python has been essential to productivity in rapidly developing scientific software solutions to urgent problems.

2.1 Molecular Docking for Drug Discovery

To examine developer productivity in a recent and urgent use case, we look at the use of Python in a current research effort at the Oak Ridge National Laboratory (ORNL) to develop antiviral therapeutics against the SAR-COV-2 virus [5, 6]. This includes the use of the Summit supercomputer to screen drug candidates for the SARS-CoV-2 virus with molecular docking calculations. Screening datasets of millions to billions of ligands against the viral proteins is a big data problem and a time-sensitive endeavor. Engineering the deployment of this large, but intuitively parallel, computation in a reasonable amount of time is an HPC problem. However, these types of calculations were not designed to be treated by HPC approaches, and thus are file-based, and require novel software solutions in order to make this type of scaling possible and rapidly attainable. Screening billions of compounds experimentally is impossible, but such a computational screen is attainable and has been performed in under 24 hours on Summit. Here we how describe the use of Python helped to dramatically and rapidly increase productivity in this effort.

The usefulness of Python for this project can be observed in three main areas: data pre-processing, workflow management, and post-processing. Each area benefits from HPC tools and libraries to exploit parallelism within and across nodes. During the initial data processing, the program must be able to ingest data quickly, from billions of separate files that are available from compound databases. As distributed, the input ligand data files are small text files distributed in a collection of compressed tarballs and must be converted into a format that the docking program can use. Therefore, before docking 1 billion ligands, all of these files had to be extracted from their tarballs, converted, and repackaged in new tarballs. The Python ecosystem already has utilities to perform all of these tasks. The `tarfile` library can read, manipulate, and create new tarfiles entirely in memory. With the dataset on a network file system, minimizing the amount of file operations is critical to performance. The `joblib` library allows for parallel tasks to be executed within a node, spawning a separate interpreter on each CPU core to preprocess the text files and then accumulate them into new output tarballs. To distribute work across multiple nodes in an HPC cluster, a library such as mpi4py was used. This harnessed HPC-familiar MPI frameworks in Python and did so in a way compatible with many Python libraries. Additionally, since the preprocessing script was a Python script, this allowed the conversion of this script from a standalone program into a loadable module with minimal effort, greatly speeding up the conversion process. Python also provides a means to quickly prototype a framework for launching executables and creating an easy workflow on Summit. This includes pre-staging data to the nonvolatile memory (NVMe) on each node, before computations are run. Python was used to wrap Autodock-GPU [7] to run one docking per NV100 GPU and seven Power9 cores of Summit. Lastly, after the simulations are run, the data is post-processed. Output from AutoDock can also be used to train and test machine learning (ML) models that predict more accurate binding affinities from the docked poses. Previous non-Python implementations of these types of models were single-core and not scalable to big data problems. Python allows for easy scalability. The cuML library, an NVIDIA RAPIDS project, can be used to offload ML tasks to GPUs, and parallelization can also be enabled with Dask [8].

2.2 Bringing GPU Acceleration to Predictive Biology

Currently the genome bottleneck in predictive biology creates a technology gap for both basic biosciences and bioengineering applications [2]. We are limited in our ability to translate the vast amount of gene sequencing data into meaningful predictions of biological function. The ability to computationally model protein structure from sequences using GPU acceleration is attractive, as is the modeling of protein-protein interactions. Many of the cutting-edge solutions currently available use Python because of user-friendliness, ability to handle heterogeneous data structures, and the powerful modules available for AI applications. In addition, Python bindings allow users to use their software as an API, thus being able to develop new applications using it. For example, we are currently working on a program that can arrive at a final folded protein structure from a set of inferred contact distances using an in-house code that calls the OpenMM [9, 10] molecular mechanics program [11]. OpenMM is able to efficiently use the GPU with highly optimized OpenCL and CUDA kernels, but is

called from a Python program that makes it easy to create one's own simulation recipes. In order to make such software work well on HPC, the top Python wrapper must call underlying code that uses relevant compilers and linkers to use the HPC facilities, such as C++ with CUDA. In addition, the ability of Python to easily call external programs with a subprocess allowed for the model building steps for this program to be executed with a different code and seamlessly incorporated into the program. Currently, we are exploring the use of directive-based (OpenMP/OpenACC) offloading to GPU with Python/Cython code bases, including the LightDock [12] protein docking program. Code developers must devote extra effort to make these codes usable in HPC in an optimized way, however, and thus success in porting existing Python programs to HPC may vary depending on the modularity and structure of the original code.

2.3 Abstract Algebra Operations for Big Integers

Integers larger than the int64 type, used in applications like cosmology, hash tables, probability simulations, and math sequence exploration, do not fit nicely into traditional NumPy arrays and often become a computational bottleneck. We did some recent work on a Python library for abstract algebra operations (additions, matrix multiplications, etc.) that support big integers. Python made the complex abstract algebra program easier to understand, prototype, and modify. Libraries like SymPy, mentioned below, provided application-specific functions. The Dask library allowed us to easily partition our arrays, perform data-parallel work, and reduce our arrays in unique user-defined ways. It also made our operations scalable within a single node and across multiple nodes.

3 HPC Python State of the Art

3.1 Python Compilation

Python's reference interpretation is CPython, which compiles Python code into bytecode before interpreting it. This is unoptimized and can cause performance drag. PyPy [13] is a faster interpreter that uses just-in-time compilation instead. It enables a much higher performance of pure Python code, but cannot use external libraries written in the Python C API, like NumPy. An alternative solution to the interpreter issue is cross-compilation to other languages. Cython [14], for example, compiles Python to C and C++. Cython is a statically compiled language that is technically a superset of Python, with performance that comes closer to that of C. Numba [15] is a just-in-time compiler that can be invoked with simple decorators. Using LLVM, it compiles Python to machine code. Numba also supports automatic parallelization on CPUs and GPUs.

3.2 Libraries and Frameworks

NumPy [16, 17] supports efficient implementations of large, multi-dimensional arrays and mathematical functions for operating on these arrays. It is perhaps the most widely used Python library for scientific computing. Python does not have its own array structures, so many of the popular scientific libraries for HPC are built upon NumPy. This includes SciPy [18, 19], for scientific and technical computing, SymPy, for symbolic computation, and matlibplot, for plotting. Pandas, which provides more data structures for easy data analysis, manipulation and I/O, also depends on Numpy. However, NumPy cannot process data exceeding local memory. Dask [20], a library for parallel and distributed computation, extends Python, NumPy, and Pandas data structures to enable "out of core" computations. Dask's built-in schedulers allow easy portability between personal machines and clusters of various scale. Dask also includes a convenient visualization dashboard. RAPIDS [21] is an open-source library for running Python data science and analytics on NVIDIA GPUs, with minimal changes. Legate [22], a separate NumPy implementation, has a similar approach. It accelerates and distributes NumPy programs to machines of any scale. CuPy [23] is a matrix library that is highly compatible with, and can even act as drop-in replacement for NumPy, while providing the advantage of GPU-accelerated computing. Other general-purpose parallel and distributed computing

systems include pySpark, a Python API for Apache Spark [24], and Ray [25], a process-based execution framework with an API based on dynamic task graphs. Ray is packaged with domain-specific libraries for machine learning.

3.3 HPC Communication

Like most HPC languages, Python can use the MPI library. The most notable implementations is the mpi4py package [26, 27]. MPI for Python (mpi4py) is the most complete MPI library in Python, with the closest C syntax match. Similarly, a Python interface for UCX [28], a low-level high-performance networking framework, is provided in UCX-py which supports a client server model to establish communications. The Process Management Interface - Exascale (PMIx) interface [29] defines interactions between applications and the system management stack.

3.4 Related Technologies

The Jupyter Notebook [30] is an open-source web application that allows interactive Python use, in addition to many other languages. Users can create and share live code, equations, visualizations, and narrative text. It is an evolution of the program previously called IPython, a command shell for interactive computing. The Jupyter Notebook is able to support parallel and distributed computing through normal Python libraries, either simple libraries built into Python, or more sophisticated packages such as Dask. Julia [31] is a programming language designed with HPC in mind, circumventing many of Python's original performance roadblocks. It has high-level syntax, dynamic typing, just-in-time compilation, and supports interactive use. It has built in support for concurrent, parallel, and distributed computing, as well as direct calling of C and Fortran code. Arkouda [32] is a distributed computation framework for exploratory data analysis. It has a Python front end, NumPy-like syntax and a Chapel back-end. It can be used interactively to issue massively parallel computations on distributed data.

4 Looking to the Future

Python is poised to improve HPC developer productivity in many more ways in the future. For example, an underlying science problem in computational seismology is being solved by the Python package SeisFlows in the SPECFEM project with 3D seismic data processed by Spark [33]. NERSC is using Python (CuPy, specifically) to make their supercomputer hardware and its accompanying GPUs more accessible to (non-computer) scientists working in their experimental and observational data facilities [34]. The Library for Evolutionary Algorithms in Python (LEAP) toolkit provides an accessible and parallel way to run evolutionary algorithms, a type of machine learning, for researchers, engineers, and educators alike [35, 36]. In fact, there are many scientific domains, from natural language processing [37] to high-energy physics and environmental science [38], that are benefiting from Python development in tandem with AI and HPC. At ORNL, current research directions include ensuring that the Python ecosystem and libraries (like Dask and RAPIDS) are available on the Summit supercomputer and portable to different systems. Often, a library is created as a vendor-specific solution (like CuPy from NVIDIA), and not always easily portable to other vendor platforms. Scalability, especially across different nodes, is also an important consideration. The ability to increase the amount of available data, whether through data generations (simulations, etc.) or through better data storage and Python manipulation, is also important research, especially for machine learning. Clearly, Python provides a productive and efficient solution to the issue of improving scientific software development.

Acknowledgments

The authors would like to thank Omar Demerdash, and Jeremy Smith's team at ORNL, who are the domain scientists for the bioscience use-case Python-related technologies discussed in this paper.

References

[1] Zachary D Stephens, Skylar Y Lee, Faraz Faghri, Roy H Campbell, Chengxiang Zhai, Miles J Efron, Ravishankar Iyer, Michael C Schatz, Saurabh Sinha, and Gene E Robinson. Big data: astronomical or genomical? *PLoS biology*, 13(7):e1002195, 2015.

[2] DOE. Breaking the bottleneck of genomes: Understanding gene function across taxa. `https://genomicscience.energy.gov/genefunction/Breaking_the_Bottleneck_2019_Web.pdf`, 2019.

[3] The covid-19 high performance computing consortium. `https://covid19-hpc-consortium.org/`, 2020.

[4] Zahra Ronaghi, Rollin Thomas, Jack Deslippe, Stephen Bailey, Doga Gursoy, Theodore Kisner, Reijo Keskitalo, and Julian Borrill. Python in the nersc exascale science applications program for data. In *Proceedings of the 7th Workshop on Python for High-Performance and Scientific Computing*, pages 1–10, 2017.

[5] Jeremy Smith. Drug discovery for covid-19. `https://www.olcf.ornl.gov/cv19-project/drug-discovery-for-covid-19/`, 2020.

[6] Geetika Gupta. Racing the clock, covid killer sought among a billion molecules. `https://covid19-hpc-consortium.org/`, 2020.

[7] L. Solis-Vasquez, D. Santos-Martins, A. Koch, and S. Forli. Evaluating the energy efficiency of opencl-accelerated autodock molecular docking. In *2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pages 162–166, 2020.

[8] Matthew Rocklin. Dask: Parallel computation with blocked algorithms and task scheduling. In *Proceedings of the 14th python in science conference*, number 130-136. Citeseer, 2015.

[9] Peter Eastman, Mark S Friedrichs, John D Chodera, Randall J Radmer, Christopher M Bruns, Joy P Ku, Kyle A Beauchamp, Thomas J Lane, Lee-Ping Wang, Diwakar Shukla, et al. Openmm 4: a reusable, extensible, hardware independent library for high performance molecular simulation. *Journal of chemical theory and computation*, 9(1):461–469, 2013.

[10] Peter Eastman, Jason Swails, John D Chodera, Robert T McGibbon, Yutong Zhao, Kyle A Beauchamp, Lee-Ping Wang, Andrew C Simmonett, Matthew P Harrigan, Chaya D Stern, et al. Openmm 7: Rapid development of high performance algorithms for molecular dynamics. *PLoS computational biology*, 13(7):e1005659, 2017.

[11] Ada Sedova, Jess Woods, Mathialakan Thavappiragasam, John Ossyra, and Matthew Baker. openfoldtools: Open source protein structure modeling tools. `https://github.com/BSDExabio/openFoldTools`, 2020.

[12] Brian Jiménez-García, Jorge Roel-Touris, Miguel Romero-Durana, Miquel Vidal, Daniel Jiménez-González, and Juan Fernández-Recio. Lightdock: a new multi-scale approach to protein–protein docking. *Bioinformatics*, 34(1):49–55, 2018.

[13] The PyPy Project. Pypy documentation. `https://doc.pypy.org/en/latest/index.html`, 2020.

[14] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D.S. Seljebotn, and K. Smith. Cython: The best of both worlds. *Computing in Science Engineering*, 13(2):31 –39, 2011. ISSN 1521-9615. doi: 10.1109/MCSE.2010.118.

[15] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pages 1–6, 2015.

[16] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22, 2011.

[17] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[18] The pandas development team. pandas-dev/pandas: Pandas. `https://doi.org/10.5281/zenodo.3509134`, 2020.

[19] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010. doi: 10.25080/Majora-92bf1922-00a.

[20] Dask Development Team. *Dask: Library for dynamic task scheduling*, 2016. URL `https://dask.org`.

[21] RAPIDS Development Team. *RAPIDS: Collection of Libraries for End to End GPU Data Science*, 2018. URL `https://rapids.ai`.

[22] Michael Bauer and Michael Garland. Legate numpy: Accelerated and distributed array computing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362290. doi: 10.1145/3295500.3356175. URL `https://doi.org/10.1145/3295500.3356175`.

[23] Ryosuke Okuta, Yuya Unno, Daisuke Nishino, Shohei Hido, and Crissman Loomis. Cupy: A numpy-compatible library for nvidia gpu calculations. In *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS)*, 2017. URL `http://learningsys.org/nips17/assets/papers/paper_16.pdf`.

[24] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al. Apache spark: a unified engine for big data processing. *Communications of the ACM*, 59(11):56–65, 2016.

[25] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. Ray: A distributed framework for emerging {AI} applications. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 561–577, 2018.

[26] Lisandro D Dalcin, Rodrigo R Paz, Pablo A Kler, and Alejandro Cosimo. Parallel distributed computing using python. *Advances in Water Resources*, 34(9):1124–1139, 2011.

[27] Lisandro Dalcín, Rodrigo Paz, and Mario Storti. Mpi for python. *Journal of Parallel and Distributed Computing*, 65(9):1108–1115, 2005.

[28] P. Shamis, M. G. Venkata, M. G. Lopez, M. B. Baker, O. Hernandez, Y. Itigin, M. Dubman, G. Shainer, R. L. Graham, L. Liss, Y. Shahar, S. Potluri, D. Rossetti, D. Becker, D. Poole, C. Lamb, S. Kumar, C. Stunkel, G. Bosilca, and A. Bouteiller. Ucx: An open source framework for hpc network apis and beyond. In *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, pages 40–43, 2015.

[29] Ralph H Castain, Joshua Hursey, Aurelien Bouteiller, and David Solt. Pmix: process management for exascale environments. *Parallel Computing*, 79:9–29, 2018.

[30] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing. Jupyter notebooks – a publishing format for reproducible computational workflows. In F. Loizides and B. Schmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87 – 90. IOS Press, 2016.

[31] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017.

[32] Michael Merrill, William Reus, and Timothy Neumann. Arkouda: Numpy-like arrays at massive scale backed by chapel. `https://github.com/mhmerrill/arkouda`, 2019.

[33] C. Chen, Y. Yan, L. Huang, and L. Qian. Implementing a distributed volumetric data analytics toolkit on apache spark. In *2017 New York Scientific Data Summit (NYSDS)*, pages 1–8, 2017.

[34] Nick Becker. Rapids makes gpus more accessible for python users at the national energy research scientific computing center. https://medium.com/rapids-ai/rapids-makes-gpus-more-accessible-for-python-users-at-the-national-energy-research-scientific-ea8561edc3a1, 2020.

[35] Mark Coletti, Eric Scott, and Jeffrey Bassett. Library for evolutionary algorithms in python (leap). In *Genetic and Evolutionary Computation Conference Companion (GECCO'20 Companion)*. ACM New York, NY, USA, 2020.

[36] Mark Coletti, Alex Fafard, and David Page. Troubleshooting deep-learner training data problems using an evolutionary algorithm on summit. *IBM Journal of Research and Development*, 64(3/4): 1–12, 2019.

[37] Aleksandr Drozd, Anna Gladkova, and Satoshi Matsuoka. Python, performance, and natural language processing. In *Proceedings of the 5th Workshop on Python for High-Performance and Scientific Computing*, pages 1–10, 2015.

[38] Rick Stevens, Valerie Taylor, Jeff Nichols, Arthur Barney Maccabe, Katherine Yelick, and David Brown. Ai for science. Technical report, Argonne National Lab.(ANL), Argonne, IL (United States), 2020.