LATTICE OPTIMIZATION USING JUPYTER NOTEBOOK ON HPC CLUSTERS*

H. Nishimura†, Y. Qin, C. Sun, S. James, K. Fernsler, K. Song, G. Jung
Lawrence Berkeley National Laboratory, Berkeley, California 94720, USA

Abstract

Tracy accelerator simulation library [1] was originally developed for the Advanced Light Source (ALS) design studies [2] at LBNL in the late 1980's. It was originally written in Pascal [3], later ported to C++, and then to C# [4]. It is still actively updated and currently used by the ALS Upgrade Project (ALS-U) [5] to design and to optimize the lattice. Recently, it has been reconstructed to provide ease of use and flexibility by leveraging the quickly growing Python language. This paper describes our effort of porting it to Jupyter Notebook[6] on our institutional High-Performance Computing (HPC) clusters [7].

PERFORMANCE AND PRODUCTIVITY

The most CPU-intensive part of the ALS-U design study is the optimization in the high-dimensional parameter space using the multi-objective genetic algorithm (MOGA)[8] on HPC clusters by using Tracy++ [9]. It is critical to have natively compiled code to maximize the runtime performance. But this approach may not be user-friendly enough since many scientists are not comfortable to deal with the complex tool-chain and software dependencies to compile the code. Python made this easy by providing an interactive environment and excellent scientific library support. By making C++ and Python work together properly, we can get a balance of performance and productivity.

Productivity will be further increased when Tracy++ is used in the Jupyter Notebook, an interactive and computational environment which has been well recognized in scientific and engineering fields. Other advantages include the contextual readability, reproducibility, and mobility.

For example, astropy [10] for astronomy is already a part of the Anaconda distribution [11]. There are also Jupyter Notebooks available for accelerator physics [12].

Our effort is to port Tracy++ API to Python in an automated environment, using Jupyter Notebook running on HPC clusters that hosts a Jupyter HUB [13].

TRACY++ FOR PYTHON

We started with porting the Tracy API to Python, then migrated it to the HPC cluster.

Wrapper Creation

Tracy++ for Python is a Python module that wraps the Tracy++ library. Among multiple options, we chose to use SWIG [14] which is a tool to automate the wrapper generation of C/C++ libraries for various programming languages including Python.

The early version of Tracy++ uses C++ class inheritance intensively. Consequently, SWIG tries to wrap all the Tracy++ entries. However, there are C++ features that are not compatible with Python, overriding C++ operators, function signatures, virtual functions, and default values for function parameters for example.

Some of these features are removed when Tracy was cleaned up a few years ago [15]. Although it was to improve performance, the effort of porting Tracy++ to Python also started. SWIG could not wrap the full API of Tracy++ due to the complexity. However, it worked fine with the trimmed version of it.

This time, the full version of Tracy++ was reconfigured to enhance the usability for Python. One of the notable changes is to control the visibility from Python properly. Differential Algebra [16] is one of such examples. Although it plays a significant role in the 6-dim particle propagation with radiation loss, there is no need to use it directly from Python. Therefore, it is not passed to SWIG. In the case of default values, both C++ and Python support them but differently. Therefore, after removing them in C++, Python can revive some of them if needed in the helper module.

Tracy development has been on Windows in Visual Studio where multiple projects are grouped into a unit called solution. It includes Tracy++ and related libraries, various client applications in C++ and the SWIG wrapper project. When the library is updated, the entire solution including the wrapper is also updated.

Helper Module

The wrapper creation has been entirely automated by using SWIG. However, there are some repetitive patterns required on the Python side. They are moved to a Python helper module to provide the following functions for seamless access from Python.

- Various listing and plotting functions as C++ layer cannot write nor plot to the Jupyter Notebook output cells.
- The accelerator lattice definition utilities to use Python lists efficiently.
- Conversion of corresponding numerical data types between C++ and Python NumPy [17].
- Default values of the function parameters. Some of them are the revival of those removed in the C++ layer.

* Work supported by the U.S. Department of Energy under Contract No. DE-AC02-05CH11231
† H_Nishimura@lbl.gov

Virtual Functions

A C++ class uses virtual methods so that the base class can use the algorithm defined in its derived function. However, if it's derived class is in Python, this trick stops working. This happens when a storage ring lattice is defined as a new Python class derived from the Ring class in C++.

The lattice is defined in Python and set to the data member of the base class. The derived class may need to define new methods for customized operations. Unfortunately, the base class cannot see such Python functions even by using the virtual methods. The call-back mechanism may solve this issue. However, it is not used because of its complexity. Such new operations have to be done in Python. As Tracy++ provides a rich set of building blocks, this is usually acceptable. An exception is the evaluation function of the genetic algorithm calculations done in C++. It must be supplied as a C++ function. When the evaluation algorithm reaches to the level of parameterization, it can be defined in Python as data. As such calculations are never interactive, this solution is acceptable.

Parallel Processing

Tracy++ has been used with MPI on the HPC clusters for MOGA optimizations. This parallelization happens outside of Tracy++. Therefore, it is out of the scope of this paper, although Python can play a flexible role as its job control language.

On the other hand, Tracy++ uses OpenMP [18] internally to utilize multiple CPU cores on a single PC in parallel to calculate dynamic apertures and frequency maps. As OpenMP is not visible from Python, these routines are wrapped as normal functions.

Data Access and Management

Python can access public data members of a C++ class. However, they reside in the proxy in Python. The get and set functions.

Data transfer becomes an issue if its size is too large and memory management is not automated. It requires extra caution to pass a dynamically allocated object over the language boundary. The generic containers of C++ reduce this issue. Besides, Tracy++ requires a data container created in Python, and passed to C++ to receive the output result with a significant amount of data.

At the same time, the data transfer over the language boundary should be minimized. A good example is the dynamic aperture and frequency map calculations [19]. It has been a common practice to pair their results. A new function was created to do this effectively in parallel and pass the result concisely to Python.

Simple genetic algorithm calculation is supported in Tracy++ by using OpenMP. We often use it to replace wide-range parameter scan in the high-dimensional parameter space. It performs very shallow optimization to disqualify unusable lattices, and not to keep these candidates for the best solution. Due to a large amount of output data, the result is saved to a file and later processed separately by using Pandas [20] and other statistical packages in Python. The rich availability of various packages in the Notebook makes an evaluation of output data quite efficient.

Parallel processing at the Python level is not included yet.

Portability to Linux

Tracy++ for Python is always updated on Windows as already explained. The port to Linux starts by updating the full Tracy++ system including the SWIG project on a local Linux box. This process is straightforward and confirmed on several Linux distributions of Ubuntu and RedHat families including Mint, CentOS and Scientific Linux.

JUPYTER HUB ON THE CLUSTER

LBNL Lawrencium cluster [7] is a general purpose High-Performance Computational Cluster built for scientific computing. The Lawrencium infrastructure recently provided the Jupyter Notebook service which has a user friendly, graphical interface.

Tracy++ for Python is migrated to the cluster in three phases. The first phase is to move the whole development to the cluster environment. There, a user can choose to work either from a web console which is provided by Jupyter-Hub, or a traditional terminal via ssh. This significantly reduces the entrance barrier for users who are not familiar with Linux or a cluster environment.

The second phase is to integrate the parallel capability of the cluster into the Notebook. This requires fine tuning of the build process on the system side. We are currently in this phase.

The third phase is to introduce a distributed computing at the Python level. We use the package, "MPI for Python" [21], to give the complete scalability to the high-level routines in Python accessing Tracy++. This package is also a part of Anaconda and is compatible with our work. Our future effort will be with MPI for Python.

CONCLUSION

The combination of C++ and Python worked successfully with Tracy++. Generation of the wrapper for Tracy++ for Python has been automated by using on SWIG. The wrapper is always synced and updated on Windows. Porting to a Linux PC is straightforward by using shell scripts.

It works in Jupyter Notebook smoothly with the variety of other Python packages for plotting, data analysis, and symbolic calculations.

It was very timely that the LBNL Lawrencium HPC cluster started offering the Jupyter service when there is a need for more computing power using Python. Migration to the Lawrencium HPC cluster turned out to be simple and straightforward. Also, environment software modules are used to manage users' runtime environments dynamically on Lawrencium, rather than local software management on the local PC.

ISBN 978-3-95450-182-3

Distributed computing on the HPC cluster in Python is possible by using MPI for Python. Tracy++ for Python is in the process of adopting it. As the CPU-intensive routines are all in C++, degradation should not be an issue compared to the benefit.

This approach is the right direction for the future accelerator design studies. Future C++ development will continue to use Jupyter HUB on the HPC cluster in mind.

REFERENCES

[1] H. Nishimura, PAC'01, Chicago, USA, p.3006, (2001).

[2] LBL PUB-5172 Rev. LBL, (1986).
A. Jackson, PAC'93, Washington, D.C, USA, p.1432, (1993).

[3] H. Nishimura, EPAC'88, Rome, Italy, p.803, (1989).

[4] H. Nishimura, ICAP 09, San Francisco, USA, p.326, (2009).

[5] C. Sun, et. al., IPAC 2016, Busan, Korea, p.2959, (2016).

[6] http://jupyter.org

[7] http://lrc.lbl.gov

[8] L. Yang et. al., Nucl. Instr. and Meth. A 609, p.50, (2009).

[9] C. Sun, et. al., Phys. Rev. STAB 15, 054001 (2012).

[10] http://www.astropy.org

[11] https://www.continuum.io

[12] https://jupyter.radiasoft.org

[13] https://github.com/jupyterhub/jupyterhub

[14] http://swig.org

[15] H. Nishimura, et. al., IPAC 2015, Richmond, VA, USA, p.1192, (2015).

[16] M.Berz, SSC-152, (1988).
Leo Michelotti, IEEE PAC89, p.839, (1989).
N. Malitskey et. al., SSCL-659, (1994).

[17] http://www.numpy.org

[18] http://www.openmp.org

[19] H. S. Dumas and J. Laskar, Phys. Rev. Lett. 70, 2975. (1993).

[20] http://pandas.pydata.org

[21] http://pythonhosted.org/mpi4py