Writing, Running, and Analyzing Large-scale Scientific Simulations with Jupyter Notebooks

Pambayun Savira*
University of St. Thomas
Argonne National Laboratory

Thomas Marrinan†
University of St. Thomas
Argonne National Laboratory

Michael E. Papka‡
Argonne National Laboratory
Northern Illinois University

ABSTRACT

Large-scale scientific simulations typically output massive amounts of data that must be later read in for *post-hoc* visualization and analysis. With codes simulating complex phenomena at ever-increasing fidelity, writing data to disk during this traditional high-performance computing workflow has become a significant bottleneck. *In situ* workflows offer a solution to this bottleneck, whereby data is simultaneously produced and analyzed without involving disk storage. In situ analysis can increase efficiency for domain scientists who are exploring a data set or fine-tuning visualization and analysis parameters. Our work seeks to enable researchers to easily create and interactively analyze large-scale simulations through the use of Jupyter Notebooks without requiring application developers to explicitly integrate in situ libraries.

Index Terms: Computing methodologies—Modeling and simulation—Simulation types and techniques—Interactive simulation; Visualization—Visualization application domains—Scientific visualization

1 INTRODUCTION

High Performance Computing (HPC) applications leverage the enormous computing power of supercomputers. Over the last decade, the compute capabilities of supercomputers have increased by a factor of 100x. However, the corresponding data input/output (I/O) rate has only increased by 10x [3]. This prevents current and next-generation large-scale scientific simulations from saving high fidelity data even when computation is possible. In situ visualization and analysis have been identified as one potential solution to the I/O bottleneck [6]. Data size can be greatly reduced by processing simulation data as it is generated and only outputting the results of an analysis task.

Another issue with HPC is the barrier to entry for scientists. Running and analyzing simulations typically involves remote login, mastery of text-based commands, and an in-depth familiarity with the supercomputing resource itself. Glick et al. have shown that Jupyter Notebooks benefit their users by lowering this barrier [4]. Jupyter Notebooks provide a web-based development environment and file management system that supports an interactive development workflow. Additionally, users can incorporate images or other rich media, widgets, and data analysis libraries in a Jupyter Notebook. This aids in the iterative development process by enabling users to more quickly write and interactively test sections of code.

Jupyter Notebooks have also shown strong potential for in situ visualization and analysis. Ibrahim et al. created an in situ library, Ascent, that can be integrated into large-scale simulations to stream data to a Jupyter Notebook [5]. Ascent enables control to be transferred between simulation and analysis routines. While analysis is performed in a Jupyter Notebook, the simulation remains paused. Results from the analysis can then be sent back to the simulation, thus creating a feedback loop to potentially steer the simulation.

In this paper, we present an alternative in situ workflow that encompasses writing, running, and analyzing large-scale scientific simulations using Jupyter Notebooks. We show how distributed computing codes can be written in a Jupyter Notebook and then compiled into an executable file. The produced executable can then be launched from the Notebook to run across many nodes of a supercomputer. Finally, data output to stdout from the application (i.e. printed) can be piped back into the Notebook and used for in situ interactive analysis. Our approach does not require the integration of any third-party libraries. Therefore, this workflow can easily be integrated into any existing application. The user only needs to add a print statement for the data produced by the simulation.

2 IN SITU HPC WORKFLOW USING JUPYTER NOTEBOOKS

A Jupyter Notebook server consists of a web front-end application that supports many different programming language kernels on the back-end. While Python is the default and most popular programming language for Jupyter Notebooks, we also utilize the Xeus-Cling kernel for interpreting C++ code. This is important because large-scale scientific simulations typically leverage the Message Passing Interface (MPI) standard, and C and C++ are by far the most widely used programming languages for MPI projects [1].

One of the main limitations when writing distributed computing code in a Jupyter Notebook is the fact that code is only interpreted on the host machine running the Jupyter server. This means that while MPI code can be written and run in a Notebook, the parallelism will be limited by the number of cores on the single host machine.

However, creating and running an executable on the Jupyter server simply runs the executable in the terminal. Respective code in a notebook is interpreted on the host machine running the Jupyter server. The host machine can run MPI code, but it will limited on the number of cores the machine has. Thus, this is not conducive for distributed computing.

2.1 Writing Distributed Computing Applications

To remove the constraint of code being run on a single machine, our proposed workflow leverages one of the "magic" commands provided by the Xeus-Cling kernel. Magic commands can tell the interpreter to do something outside the scope of typical code. In this case, we use the `%%executable` command which will dump the code from all cells into an executable binary. The content of the cell with the magic command is used for the body of the C++ program's `main` function. We can then use another magic command (`!<executable_name> [<params...>]`) to launch an executable. In order to run our distributed MPI program across multiple nodes in a cluster, `!mpirun` can be executed.

We demonstrate this ability with a simple two-dimensional (2D) Lattice Boltzmann Methods Computational Fluid Dynamics (LBM-CFD) simulation [2]. Our implementation uses MPI with each process responsible for a sub-grid in the overall 2D domain, and is self-contained as a C++ class written with under 400 lines of code. We have written a Notebook that sets up simulation parameters (e.g. grid dimensions, fluid viscosity, barrier locations, etc.) and runs N

*e-mail: pam.savira@stthomas.edu
†e-mail: tmarrinan@stthomas.edu
‡e-mail: papka@anl.gov

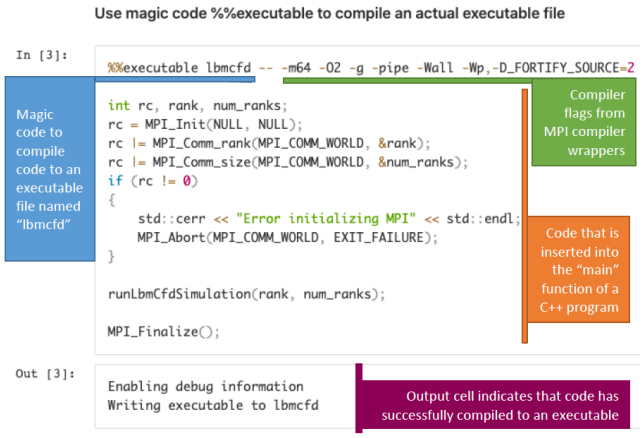

Figure 1: Using a magic command to compile a Notebook's C++ code into an executable. Note, that compiler flags can also be specified – in this instance, those used with the MPI wrapper compiler `mpicxx`.

time steps of the simulation. Fig. 1 shows the Jupyter Notebook cell with the magic `%%executable` command to compile the simulation into an executable application.

2.2 Running Applications on a HPC Cluster

While compiled MPI applications can be launched via a Jupyter Notebook using the magic command `!mpirun`, its text-based output will simply be shown on the web interface. In order to capture the distributed application's output and use it for performing in situ analysis in the Notebook, a different approach was needed. Instead of using the magic command, our workflow involves writing code to spawn a new process (in this instance `mpirun`) and piping its stdout into a buffer. Once the applications output is stored in a buffer, it can be used for further processing.

In our sample LBM-CFD simulation, we added print statements at specific points in the code. For every M time steps, we compute and print 64-bit floating point vorticity (i.e. rotational velocity) values at each cell in the 2D grid. Prior to outputting the vorticity itself, we print a unique string (e.g. `"***BEGIN OUTPUT***"`). This enables the Notebook to read stdout until it comes across the unique string and then read the next $dim_x * dim_y * 8$ bytes to get the vorticity values. This strategy enables the simulation to still be instrumented with other print statements that may be useful in other situations.

2.3 Analyzing Large-scale Simulations

Our workflow allows users to select whichever language best suites their analysis. Launching the MPI executable and reading its data from stdout is not limited to using the same language as the simulation. Once data is read into a Notebook, it can be used for any visualization or analysis task suitable for the application. This workflow of writing distributed computing application in one Notebook then launching the application and performing in situ analysis in a separate Notebook provides a user friendly mechanism for interfacing with large-scale scientific simulations.

For our sample LBM-CFD simulation, we were interested in tracking vortices (regions with high clock-wise or counter clock-wise velocity). We selected Python as a language for performing the in situ analysis due to its wealth of data analysis and visualization libraries. For each time step of vorticity values that were read in, we converted the 64-bit floating point buffer to a grayscale image and used OpenCV's *SimpleBlobDetector* to find regions with low or high values that were approximately elliptical. Fig. 2 shows the results from a single time step of our in situ analysis with the simulation running on Argonne Leadership Computing Facility's Cooley

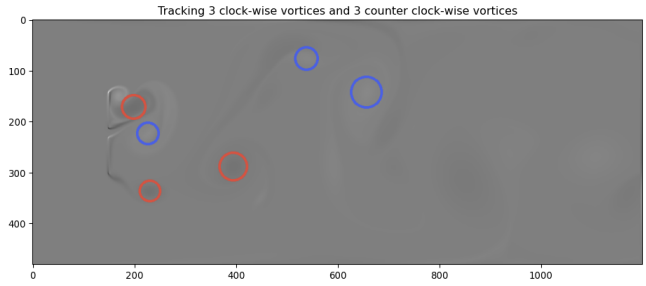

Figure 2: Vortex tracking snapshot from one time step of the LBM-CFD simulation run at 1200×480. Blue circles track clock-wise vortices, and red circles track counter clock-wise vortices.

cluster using 48 processes spread across 8 nodes. This visualization can easily be displayed in the Jupyter Notebook, resulting in an animation that plays out as the simulation is running.

3 Conclusion

We have presented an in situ workflow that leverages Jupyter Notebooks for writing, running, and analyzing large-scale scientific simulations. This workflow eliminates the need for simulations to save raw data to disk prior to performing analysis tasks. Moreover, it provides domain scientists with the easy-to-use Jupyter interface in all phases and does not require integrating complex third-party libraries. However, we do note that this workflow does have some limitations. Primarily, all data output from the simulation must be serialized due to the fact that stdout is piped into the Notebook as a single stream shared by all processes.

For future work, we look to add more interaction during the analysis phase. For example, we could add widgets to the LBM-CFD analysis Notebook to dynamically control the vortex detection parameters. This would help domain scientists select appropriate analysis parameters without continually re-running the simulation under different conditions.

Acknowledgments

This research was supported by the Argonne Leadership Computing Facility which is a U.S. Department of Energy Office of Science User Facility operated under contract DE-AC02-06CH11357.

References

[1] D. E. Bernholdt, S. Boehm, G. Bosilca, M. Gorentla Venkata, R. E. Grant, T. Naughton, H. P. Pritchard, M. Schulz, and G. R. Vallee. A survey of MPI usage in the US exascale computing project. *Concurrency and Computation: Practice and Experience*, 32(3):e4851, 2020. doi: 10.1002/cpe.4851

[2] S. Chen and G. D. Doolen. Lattice Boltzmann Method for fluid flows. *Annual Review of Fluid Mechanics*, 30(1):329–364, 1998. doi: 10.1146/annurev.fluid.30.1.329

[3] H. Childs, J. Bennet, C. Garth, and B. Hentschel. In Situ visualization for computational science. In *IEEE Computer Graphics and Applications*, pp. 76–85, 2019. doi: 10.1109/MCG.2019.2936674

[4] B. Glick and J. Mache. Jupyter notebooks and user-friendly HPC access. In *2018 IEEE/ACM Workshop on Education for High-Performance Computing (EduHPC)*, 2018. doi: 10.1109/EduHPC.2018.00005

[5] S. Ibrahim, T. Stitt, M. Larsen, and C. Harrison. Interactive in situ visualization and analysis using Ascent and Jupyter. In *Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, ISAV '19, p. 44–48, 2019. doi: 10.1145/3364228.3364232

[6] T. Peterka, D. Bard, J. Bennett, E. W. Bethel, R. Oldfield, L. Pouchard, C. Sweeney, and M. Wolf. ASCR workshop on in situ data management: Enabling scientific discovery from diverse data sources. Technical report, US DOE Office of Science, 2019.