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

INTRODUCTION

- Large-scale scientific simulations typically output massive amounts of data that must be stored and later read in for post-hoc analysis and visualization.
- Writing data to disk 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.

MOTIVATION

- Lower the entry barrier of using High Performance Computing (HPC) for large data analysis and visualization
- Advance science discovery by enabling researchers to easily create, interact with, and analyze large-scale simulations through the use of Jupyter Notebooks
- Provide faster analysis and visualization by integrating in situ

METHODOLOGY

Write code in one notebook → **Create executable from that code** → **Launch executable from another notebook** → **Read data in from executable output and perform on-the-fly visualization / analysis notebook**

Use magic code %%executable to compile an actual executable file

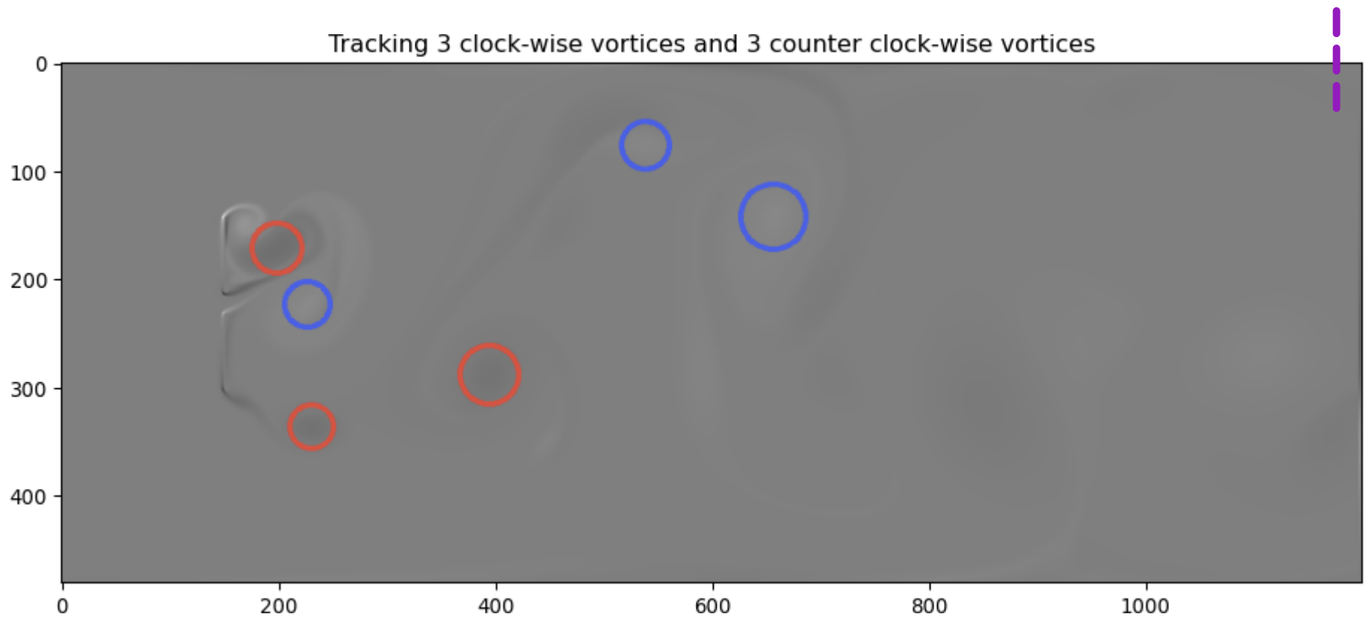
```
In [3]:
%%executable lbmcfd -- -m64 -O2 -g -pipe -Wall -Wp,-D_FORTIFY_SOURCE=2

int rc, rank, num_ranks;
rc = MPI_Init(NULL, NULL);
rc |= MPI_Comm_rank(MPI_COMM_WORLD, &rank);
rc |= MPI_Comm_size(MPI_COMM_WORLD, &num_ranks);
if (rc != 0)
{
    std::cerr << "Error initializing MPI" << std::endl;
    MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);
}

runLbmCfdSimulation(rank, num_ranks);

MPI_Finalize();

Out [3]:
Enabling debug information
Writing executable to lbmcfd
```

Magic code to compile code to an executable file named "lbmcfd"

Compiler flags from MPI compiler wrappers

Code that is inserted into the "main" function of a C++ program

Output cell indicates that code has successfully compiled to an executable



JUPYTER NOTEBOOK

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.

LBM-CFD

We demonstrate this simulation workflow with a simple two-dimensional (2D) Lattice Boltzmann Methods Computational Fluid Dynamics (LBM-CFD) simulation.

CONCLUSION

- This workflow eliminates the need for simulations to save raw data to disk prior to performing analysis tasks
- It provides domain scientists with the easy-to-use Jupyter interface in all phases
- Limitation: all data output from the simulation must be serialized
- Future work: we look to add more interaction during the analysis phase

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