CORTIX: An Open Source Framework For Dynamic Network Simulations at Scale

Taha Azzaoui[1], Valmor F. de Almeida[*]

University of Massachusetts, Lowell MA: [1]Dept. of Computer Science and Mathematics, [*]Dept. of Chemical Engineering (Nuclear Program).

American meteorological society annual meeting 14 January 2020, Boston, MA

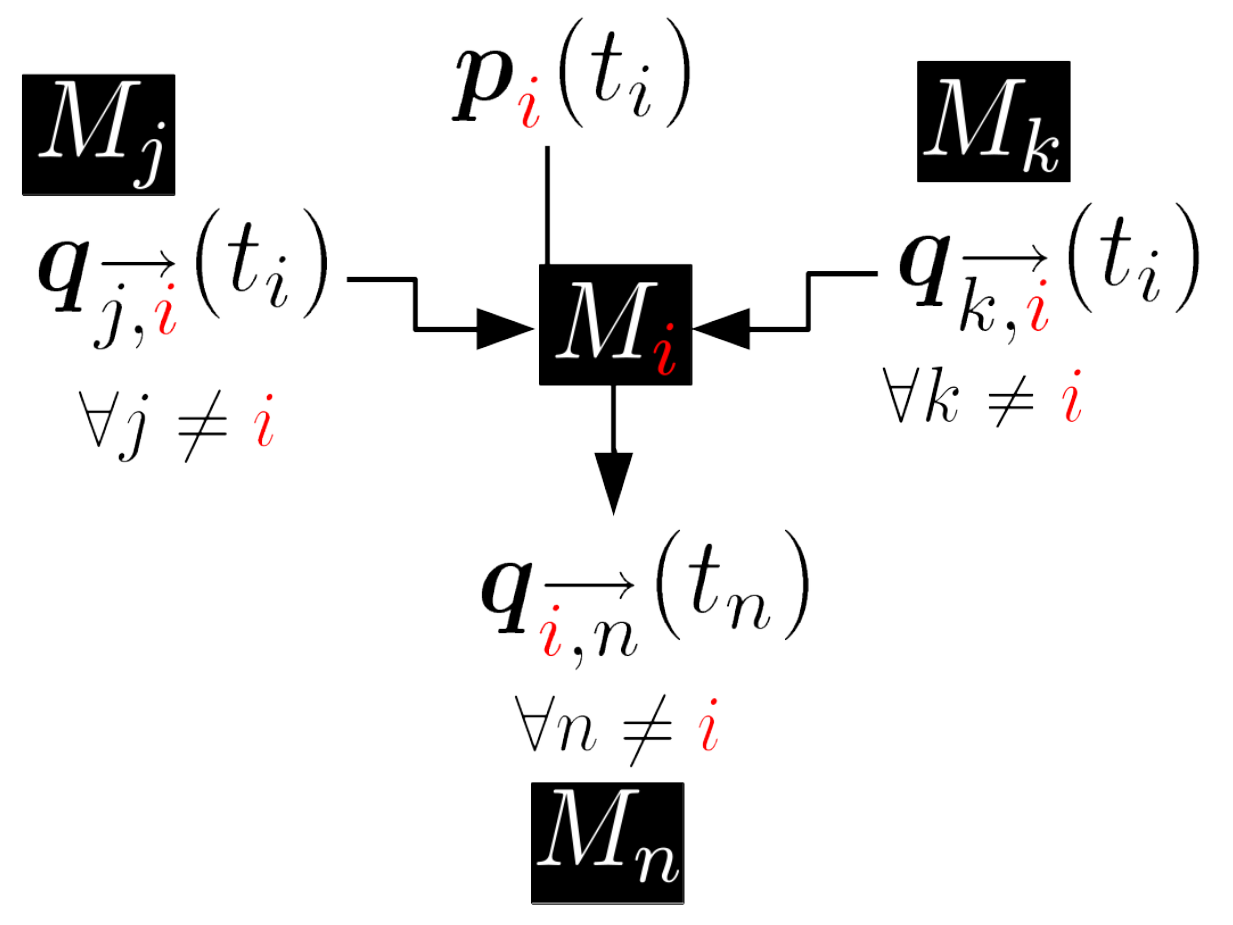
Overview

- ① Breaking complex systems up into independent parts (modules) and coupling those parts greatly simplifies the implementation of large-scale problems.
- ② Cortix provides an environment for connecting modules and passing coupling data between them.
- ③ Cortix uses the Message Passing Interface (MPI) to take advantage of massively mutli-core systems by scaling up the number of decoupled modules per simulation.

Introduction

Complex systems are often simulated via a single monolithic code which attempts to fully implement the system's governing equations and sequentially integrate them over time. Doing so, for sufficiently complex systems, can be computationally and theoretically infeasible. That is, solving millions of tightly-coupled n-dimensional ordinary differential equations is computationally expensive- prohibitively so at scale. Additionally, complex models with many moving parts are difficult to conceptualize and design entirely at once. As such, we present **Cortix**: a Python library for system-level module coupling, execution, and analysis of dynamical system models that exchange time-dependent data. Cortix enable users to decompose complex coupled models into a finite set of parts called modules. Cortix provides an environment for exchanging data between the modules that make up the overall model. Cortix is highly parallel, making use of both MPI and the Python multiprocessing library to harness the power of massively multicore systems.

Cortix Module Connectivity



- ① M_i evolves on its own time t_i .
- ② Requires a parameter vector $\boldsymbol{p}_i(t_i)$.
  - These parameters can be fed as input into modules.
- ③ Coupling vectors $\boldsymbol{q}_{\vec{j,i}}(t_i)$ allow M_i to use data from other modules in the network.
  - M_i will **wait** for time-stamped coupling data at t_i via message passing; this effectively synchronizes the whole simulation.

Module Evolution Over Time

Initialize every module $M_i\big(\boldsymbol{x}(t_0); \boldsymbol{q}(t_0), \boldsymbol{p}(t_0)\big)$ at $t_0 = 0$ in the network. For all modules M_i, $i = 1, \ldots, N$ do:

- ① Solve for $\boldsymbol{x}(t_i^{(k)})$ $\forall$ $M_i\big(\boldsymbol{x}(t_i^{(k)}); \boldsymbol{q}(t_i^{(k-1)}), \boldsymbol{p}(t_i^{(k-1)})\big)$ in parallel.
- ② Compute $\boldsymbol{q}(t_i^{(k)}), \boldsymbol{p}(t_i^{(k)})$ and exchange information: $\boldsymbol{q}_{\vec{i,j}}(t_j^{(k)})$ and $\boldsymbol{q}_{\vec{j,i}}(t_i^{(k)})$.
- ③ Advance $t_i^{(k+1)} \leftarrow t_i^{(k)} + \Delta t_i^{(k)}$ according to the configured time step $\Delta t_i^{(k)}$.

In step 2 above, message passing at different times effectively synchronizes the simulation as some modules will have to wait for information at the requested time.

Simulating Droplet Swirl

This Cortix use-case simulates the motion of a swarm of droplets in a vortex stream. It consists of two modules, namely, a `Droplet` module used to model the droplet dynamics, and a `Vortex` module used to model the effects of the surrounding air on the falling droplets. The `Droplet` module is instantiated as many times as there are droplets in the simulation while a single `Vortex` module is connected to all `Droplet` instances. The communication between modules entails a two-way data exchange between the `Vortex` module and the `Droplet` modules, where `Droplet` sends its position to `Vortex` and `Vortex` returns the air velocity to `Droplet` at the given position.

Droplet Motion Model

The equation of motion of a spherical droplet can be written as:

$$m_\mathrm{d}\, d_t\boldsymbol{v} = \boldsymbol{f}_\mathrm{d} + \boldsymbol{f}_\mathrm{b},$$

where

$$\boldsymbol{f}_\mathrm{d} = c_\mathrm{d} A\, \rho_\mathrm{f}\, \frac{||\boldsymbol{v} - \boldsymbol{v}_\mathrm{f}||}{2}\, (\boldsymbol{v} - \boldsymbol{v}_\mathrm{f}),$$

is the form drag force on the droplet,

$$\boldsymbol{f}_\mathrm{b} = (m_\mathrm{d} - m_\mathrm{f})\, g\hat{z},$$

is the buoyancy force on the droplet,

$$c_\mathrm{d}(Re) = \begin{cases} \frac{24}{Re} & Re < 0.1 \\ \left(\sqrt{\frac{24}{Re}} + 0.5407\right)^2 & 0.1 \leq Re < 6000 \\ 0.44 & Re \geq 6000 \end{cases}$$

is the drag coefficient as a function of Reynold's number, $Re = \frac{\rho_\mathrm{f}\,||\boldsymbol{v}||\,d}{\mu_\mathrm{f}}$. The mass of the droplet and its displaced fluid mass are denoted m_d and m_f, respectively. Droplet diameter, d, dynamic viscosity, μ_f, and mass density, ρ_f, of the surrounding air are provided.

Vortex Model

Here we simply use an imposed vortex circulation in analytical form given by its tangential component of velocity

$$v_\theta(r, z, t) = \left(1 - e^{\frac{-r^2}{8 r_c^2}}\right) \frac{\Gamma}{2\pi\ \max(r, r_c)} f(z)\, \big|\cos(\mu\, t)\big|,$$

and its vertical component

$$v_z(z, t) = v_h\, f(z)\, \big|\cos(\mu\, t)\big|,$$

where

$$f(z) = e^{\frac{-(h-z)}{\ell}}$$

is a vertical relaxation factor, r_c is the vortex core radius, $\Gamma = \frac{2\pi R}{v_\theta|_{r=R}}$ is the vortex circulation, R is the vortex outer radius, h is the height of the vortex, and ℓ is the relaxation length of v_z.

Results

A set of 1000 droplets of water (`Droplet` modules) are released from 500-m altitude into a `Vortex` stream of air at random positions within a square area of 250 × 250 m^2 and random droplet diameter sizes ranging from 5 mm to 8 mm; standard physical properties of both fluids are used.

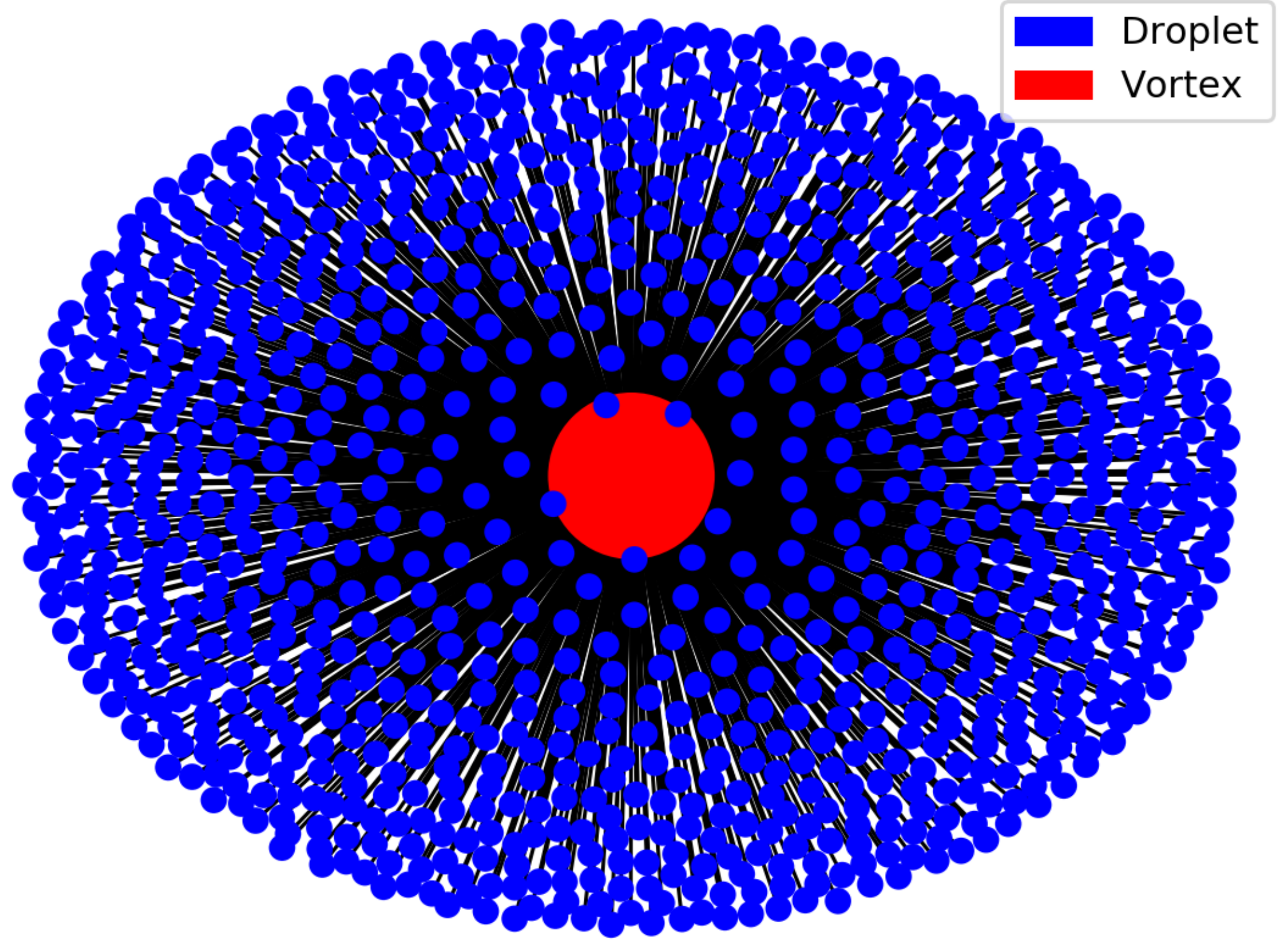


Figure 1: Cortix connectivity network for 1000 `Droplet` instances and one `Vortex` module.

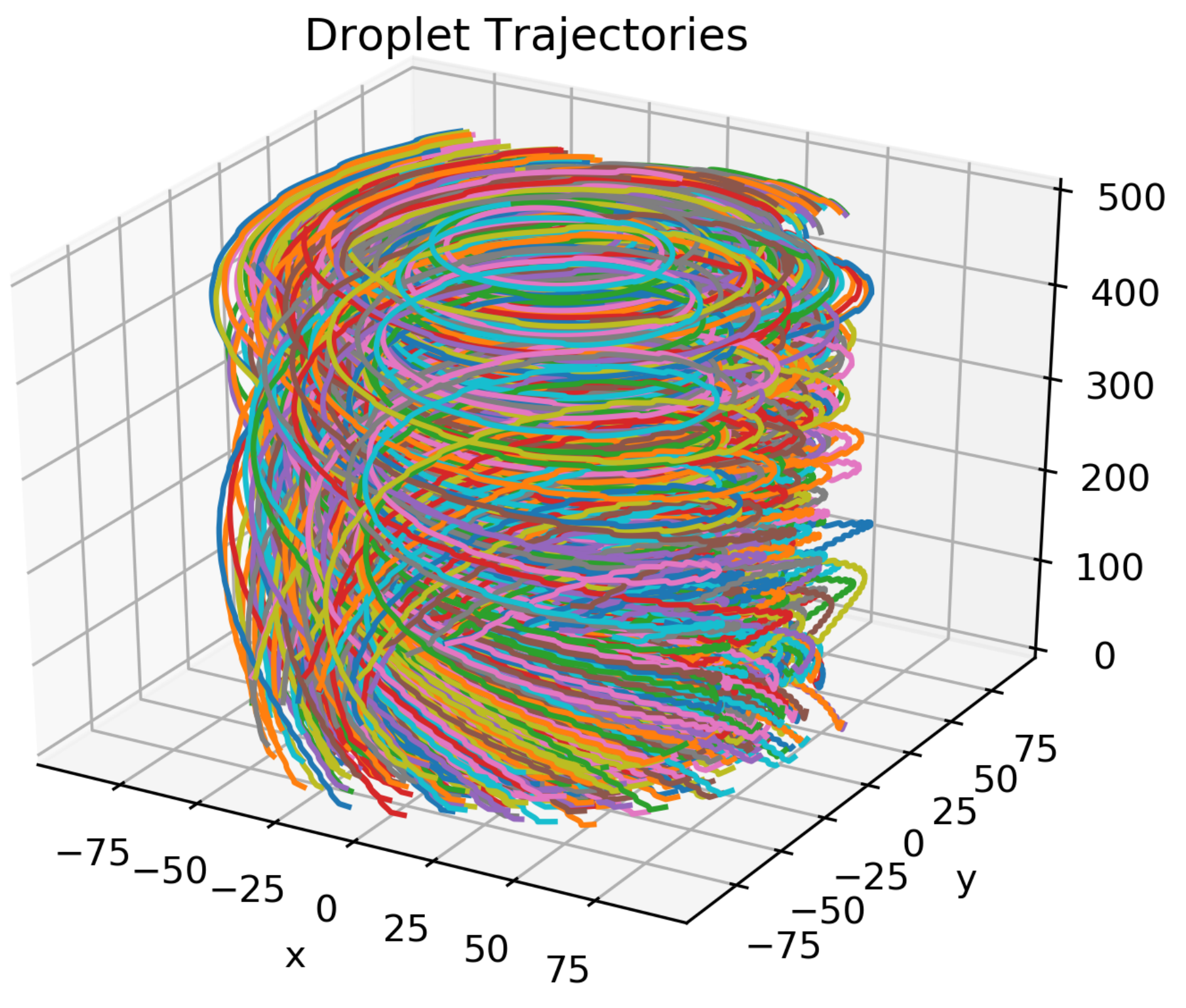


Figure 2: Trajectories of 1000 droplets released from random positions at 500-m altitude.

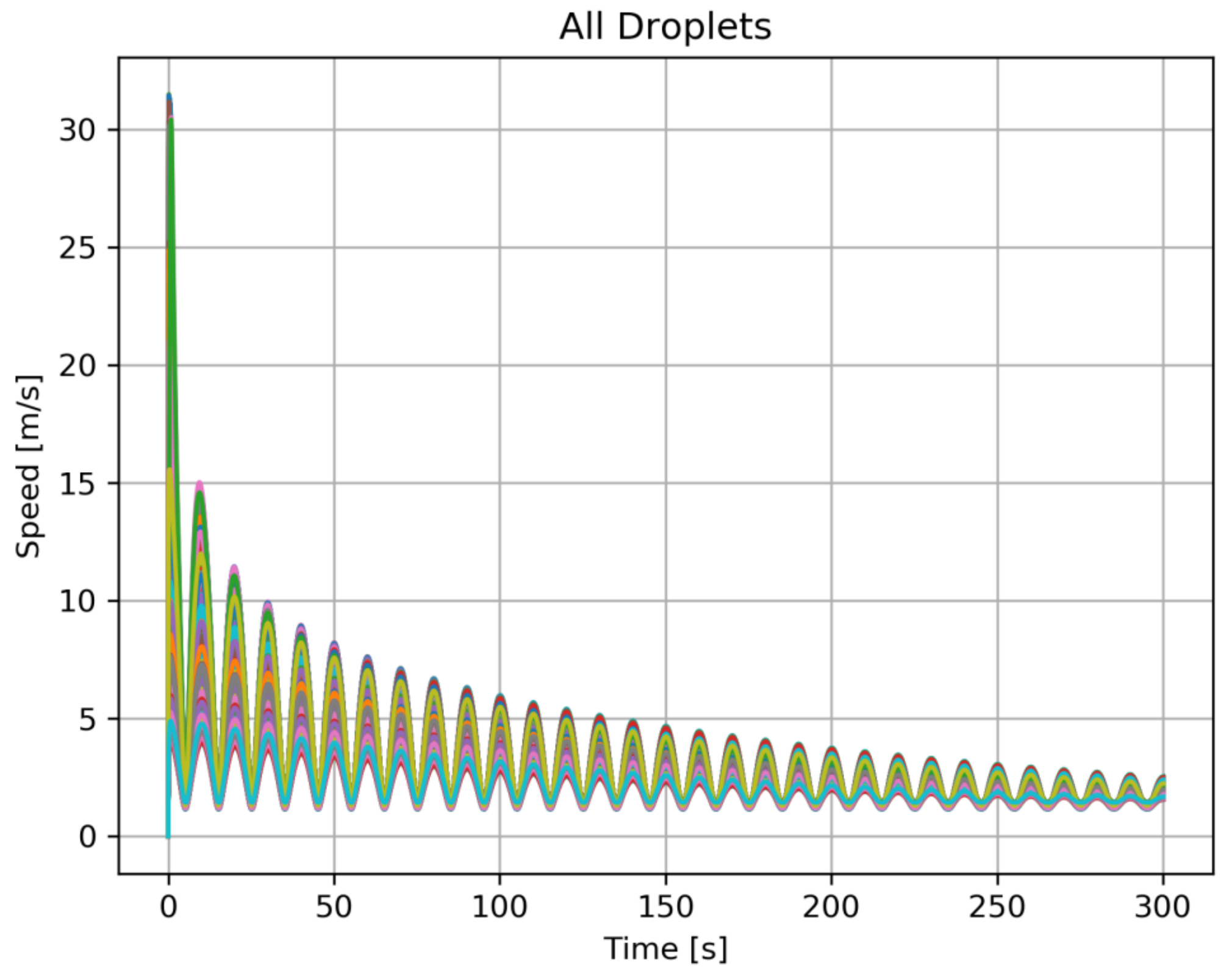


Figure 3: Speed of all droplets varying with time showing the approach to terminal velocity.

Scalability

As today's scientific workloads become increasingly parallel, a major design goal of Cortix is to be scalable from the start. By default, Cortix executes in "multiprocessing mode" which allows for rapid development and interfaces well with the Jupyter notebook environment. Cortix can also be run in "MPI mode", which makes use of the Message Passing Interface to take advantage of massively multicore systems by mapping the execution of modules across thousands of cores. This combination provides users with a lightweight environment to design and test their modules along with the flexibility to scale up to arbitrarily large HPC clusters with minimal code change. Additionally, Cortix is implemented in the Python programming language which allows users to leverage the numerous tools and packages available within the scientific Python ecosystem.

# of Droplets	Execution time (s)	# of cores
250	127	252
500	168	502
1000	346	1002
2000	1660	2002

Table 1: Droplet Simulation Performance Trend

Acknowledgments

This work was partially funded by the University of Massachusetts Lowell Francis College of Engineering and Idaho National Laboratory (INL). Access to parallel computing cycles at the INL high-performance supercomputers sponsored by Dr. Terry Todd is greatly appreciated.

- Web: cortix.org
- GitHub: github.com/dpploy/cortix