Compressible Jet Code in Python: Contaminant Diffusion

Abstract

A three-dimensional, finite-difference, compressible Navier Stokes solver is developed for the simulation of a turbulent plane jet to study the transmission of the COVID-19 virus. The solver is written using Python and Numpy and parallelized with the mpi4py library. It is used to simulate the transport of a passive tracer by a jet flow. The motivation is to simulate flows representative of coughs/sneezes carrying the COVID-19 virus to study the effectiveness of social distancing guidelines, e.g. the "six feet" rule.

Introduction

With the rise of the coronavirus (COVID-19) pandemic in the early part of 2020, there has also been a rise in efforts to understand the transmission of the virus so that effective safety measures can be implemented to slow its spread. Because the virus causes acute respiratory symptoms, airborne droplets dispersed through coughing, sneezing, talking and breathing are thought to be an important transmission vector. For this reason, understanding droplet-laden flow physics is a necessary step for designing and evaluating safety measures to limit the virus' spread. A widely circulated safety guideline recommends that people maintain at least six feet of distance between one another to avoid spreading the virus. While increasing distance is likely to decrease the chances of coming into contact with COVID-19 laden droplets, there is some uncertainty in determining how those droplets travel through the air, where they will end up, and how long they may linger airborne. The motion of the droplets are governed by complex physical phenomena which include turbulence and coupling between the flow and droplets.

A common method for learning about these complex physics is computational modelling[1]. However, the computational models for this type of problem involve large amounts of information and large separation of scales so the computations typically require massively parallel calculations to be run on high performance computers. The focus of this project is on developing a computational solver for use on a high performance computer. The solver is then used to simulate a representative flow to learn about the transport of the virus in the air due to coughs and sneezes in order to create visualizations that can be used to inspect and evaluate existing safety measures such as the six feet rule.

Methodology

The Navier-Stokes equations in conservative form (Eq. 1-2) are solved using a second order MacCormack scheme (Eq. 3-4)[2]. The scheme uses a two-step time integration that uses first-order forward differences at the first step and first-order backward differences at the second step such that the average of the steps achieves second order accuracy. The scheme has a maximum stencil width of three grid points which makes the parallelization of the grid more efficient than schemes with larger stencil widths. This will allow the code to have a good scaling behavior when running

highly parallel simulations. The code is parallelized by splitting the grid into sections along the streamwise direction. This direction was chosen as it was expected that the computational domain would be the longest in this direction given the geometry of the problem. An improvement upon this parallelization approach would be to parellelize the grid in more physical directions while minimizing the surface areas of the sections, however this implementation is beyond the scope of this project.

A rectangular prism is used for the computational domain, illustrated in Figure 1 (dimensions not to scale). Ambient pressure boundary conditions are used on the top and bottom walls and periodic boundary conditions are used in the spanwise direction. The size of the computational domain is $50h \times 25h \times 1h$ where h is the height of the jet inlet. The inlet height is chosen to approximate the diameter of a human mouth ($h = 0.04m$)[3]. The inlet is approximated as a plane jet with infinite spanwise extent. Numerical sponges are used at the top and bottom boundaries as well as the outlet boundary in order to dampen any reflected flow characteristics. To approximate the conditions of a human cough, a max inlet velocity of $10m/s$[3] is used with an approximately parabolic profile for stability considerations. Additionally, a weak coflow of about 10% of the jet inlet velocity is added to prevent spurious circulations towards the inlet and also to aid in stability of the solver. A conserved passive scalar contaminant is used to model the transport of approximate passive tracer particles such as aerosols. The scalar contaminant has a value of unity in the jet inlet and the flow is initialized with a scalar contaminant value of zero. Modeling the transport of the virus is only a good assumption for transport via aerosols which are particles small enough that they act as passive tracers. Likely, these aerosols would be less effective at spreading the virus than larger droplets which are not modeled well by a passive tracer. Droplets could be better simulated by implementation of Lagrangian particle tracking as well as two-way coupling of the flow, however these extensions are beyond the scope of the project. However, the code does provide a framework to develop these models.

Strong and weak scaling analysis is used to measure code performance and gauge technical readiness for use on an HPC cluster. The strong scaling is done on a constant grid size of $n_x = 2000$, $n_y = 30$, $n_z = 30$ for 1000 time steps. The weak scaling is done on a grid size of $n_x = 100$ per core, $n_y = 30$, $n_z = 30$ for 1000 time steps (Figure 4). Additionally, the code is profiled using Pyinstrument (Figure 5) in an effort to identify the performance bottlenecks of the code. Visualizations of filled contours are generated using matplotlib and iso-contour plots are generated using the plotly package.

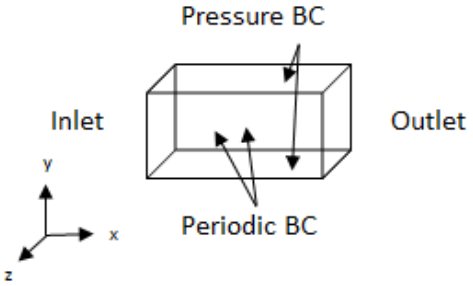


Figure 1: Schematic of the Computational Domain

Reference Fluid Properties

ρ	μ	γ	R_g	κ	D
1.022	1.81e-5	1.4	287	0.0262	1.5e-5

$$\frac{\partial U}{\partial t} = -\frac{\partial E}{\partial x} - \frac{\partial F}{\partial y} - \frac{\partial G}{\partial z} \tag{1}$$

$$U = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ E_t \\ \rho\phi \end{bmatrix} \quad E = \begin{bmatrix} \rho u \\ \rho u^2 + p - \tau_{xx} \\ \rho uv - \tau_{xy} \\ \rho uw - \tau_{xz} \\ (E_t + p)u - u\tau_{xx} - v\tau_{xy} - w\tau_{xz} + q_x \\ \rho u\phi - D\frac{\partial \phi}{\partial x} \end{bmatrix}$$

$$F = \begin{bmatrix} \rho v \\ \rho uv - \tau_{xy} \\ \rho v^2 + p - \tau_{yy} \\ \rho vw - \tau_{yz} \\ (E_t + p)v - u\tau_{xy} - v\tau_{yy} - w\tau_{yz} + q_y \\ \rho v\phi - D\frac{\partial \phi}{\partial y} \end{bmatrix} \quad G = \begin{bmatrix} \rho w \\ \rho uw - \tau_{xz} \\ \rho vw - \tau_{yz} \\ \rho w^2 - \tau_{zz} \\ (E_t + p)v - u\tau_{xz} - v\tau_{yz} - w\tau_{zz} + q_z \\ \rho w\phi - D\frac{\partial \phi}{\partial z} \end{bmatrix} \tag{2}$$

$$\text{Predictor: } \bar{U}^{t+\Delta t}_{i,j,k} = U_{i,j,k} - \frac{\Delta t}{\Delta x}\left(E^t_{i+1,j,k} - E^t_{i,j,k}\right) - \frac{\Delta t}{\Delta y}\left(F^t_{i,j+1,k} - F^t_{i,j,k}\right) - \frac{\Delta t}{\Delta z}\left(G^t_{i,j,k+1} - G^t_{i,j,k}\right) \tag{3}$$

$$\text{Corrector: } U^{t+\Delta t}_{i,j,k} = \frac{1}{2}\Big[U^t_{i,j,k} + \bar{U}^{t+\Delta t}_{i,j,k} - \frac{\Delta t}{\Delta x}\left(\bar{E}^{t+\Delta t}_{i,j,k} - \bar{E}^{t+\Delta t}_{i-1,j,k}\right) - \frac{\Delta t}{\Delta y}\left(\bar{F}^{t+\Delta t}_{i,j,k} - \bar{F}^{t+\Delta t}_{i,j-1,k}\right) - \frac{\Delta t}{\Delta z}\left(\bar{G}^{t+\Delta t}_{i,j,k} - \bar{G}^{t+\Delta t}_{i,j,k-1}\right)\Big] \tag{4}$$

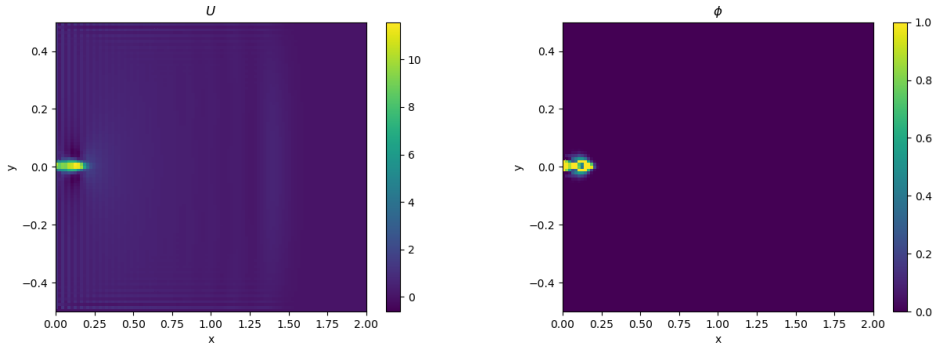
Results and Discussion

The simulations show a high concentration of the passive tracer persists well into the two meter domain, with significant mixing in the normal direction. The implications of this result are that the "six feet" rule is not effective to limit the spread of aerosols by flows from coughs or sneezes. The concentration of the scalar is high around 0.3-0.5 at the domain outlet within about half a second of the jet startup. The contours of velocity and scalar contaminant (Figure 2) show the rapid formation of small, chaotic structures, indicating turbulent mixing is an important characteristic of the flow. From the present results, it is apparent that the calculation requires more resolution to achieve a more accurate simulation of the turbulence in the flow. The three-dimensional isosurface of contaminant scalar show that there is small but seemingly coherent variation of ϕ along the z-axis (Figure 3). It is likely that a large extent is needed in the domain in the spanwise direction to capture turbulent eddies that do not show coherent motion. An investigation

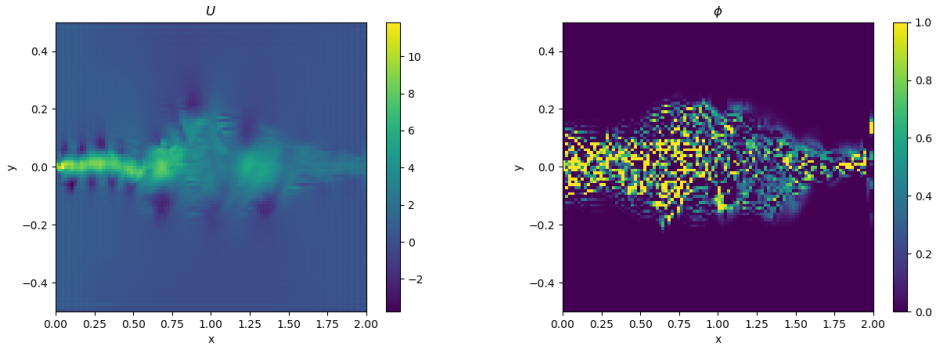
of the velocity correlation functions in this direction could provide more definitive results about whether the spanwise domain extent is sufficient to capture turbulence.

The strong scaling analysis of the code gives promising results, in particular that it is continuing to speed-up without hitting diminishing returns on up to 48 processors (the limit of processors available during this project). The weak scaling analysis shows that the code does tradeoff some efficiency to run on larger numbers of processors, dropping to approximately 40% of the serial performance level. However, because the efficiency is approximately constant at this value for 16, 32, and 48 processors, it is expected that the code would not see any more significant decreases in efficiency running on more processors.

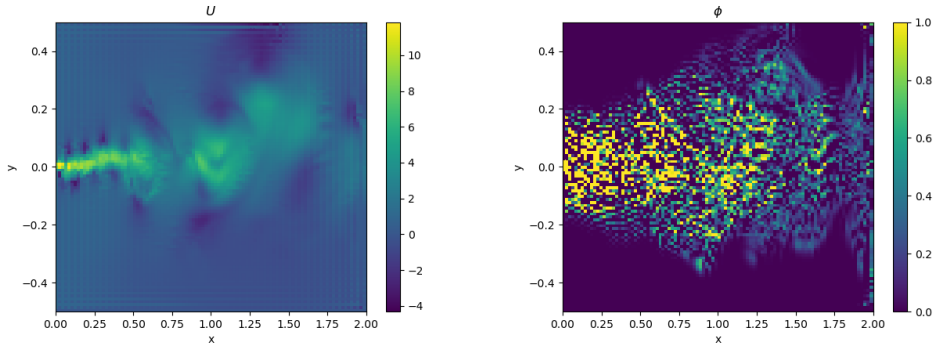
The profiling via Pyinstrument shows that the code spends most of its computation time in the numerics module computing derivatives and variable fluxes. This result is good because it signifies that the bottleneck of the code is the part that is expected to require the most computation and that the code is not hitting other spurious bottlenecks in other code modules. This result is important for identifying the areas of the code that should receive JIT compilation with the Numba package. The initialization of the solver, computation of simulation time step size, and the application of boundary conditions only amount to approximately 10% of the computation time. For larger problem sizes and longer simulation times, this value would decrease and the numerics code module would run even more often. This is fortunate because this code section is nearly fully parallel. The code is also profiled while using Numba JIT compilation on the numerics module. A small speedup of approximately 10% was observed over many time steps in longer simulations. In addition to Numba compilation, it is expected that this speedup could be increased if the code is rewritten to use Fortran memory ordering (as that is the format that the code is written in), however attempts at implementing this improvement were unsuccessful.



(a) $t = 0.03$ (b) $t = 0.03$



(c) $t = 0.50$ (d) $t = 0.50$



(e) $t = 0.88$ (f) $t = 0.88$

Figure 2: Mid-plane contours of streamwise velocity and contaminant concentration.

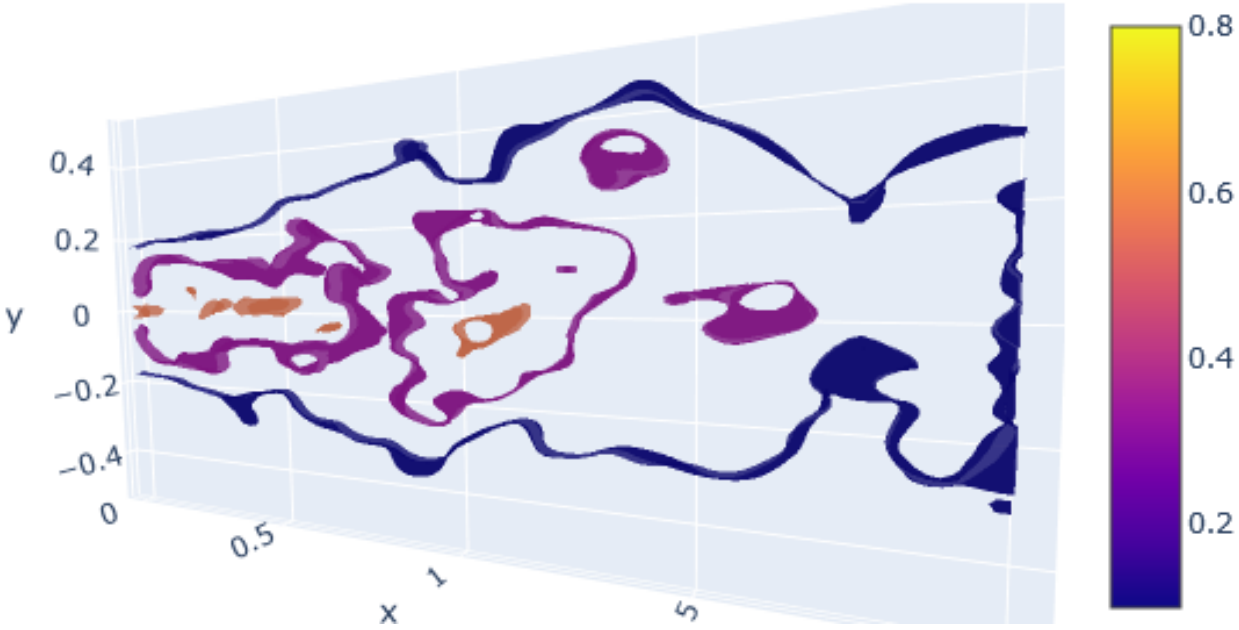
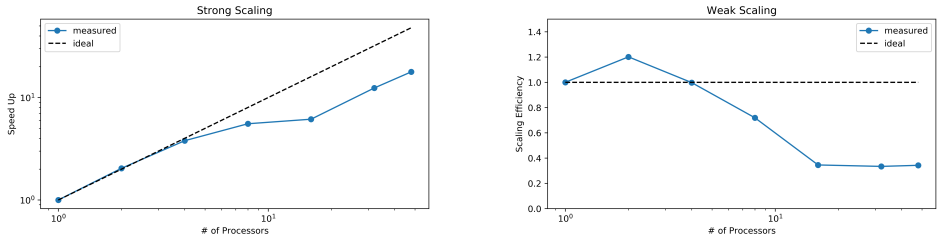


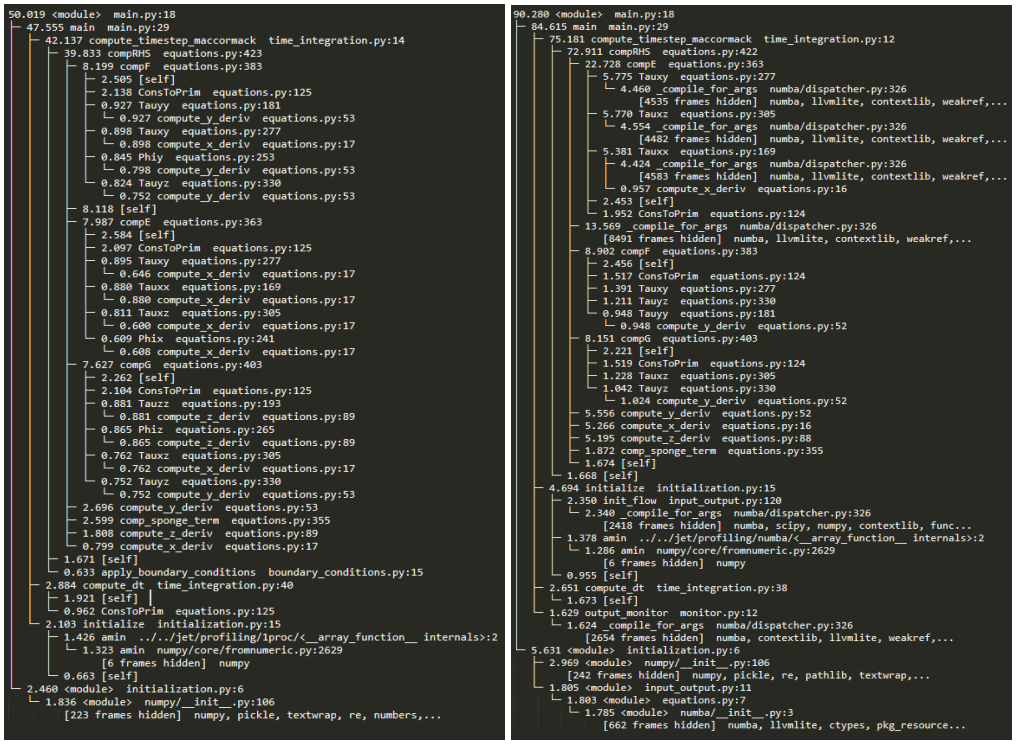
Figure 3: Iso-surface contours of $\phi = \{0.1, 0.33, 0.57\}$ at $t = 0.88$. Smoothing is applied by a Gaussian filter of width $\sigma = 3\Delta_i$.



(a) Strong scaling analysis. The size of the computational problem remains fixed as the number of processors used increases. The problem was computed on a $2000 \times 30 \times 30$ grid for 1000 steps.

(b) Weak scaling analysis. Both the number of processors and the problem size are increased resulting in a constant workload per processor. Each processor handled a $100 \times 30 \times 30$ partition of the domain for 1000 steps.

Figure 4: Parallel Scaling Analysis



(a) Without Numba JIT compilation (b) With Numba JIT compilation

Figure 5: Pyinstrument Profiling

Summary

A three-dimensional, compressible Navier-Stokes solver is written to simulate the problem of a jet flow transporting a passive scalar contaminant. The code is parallelized in the streamwise direction using mpi4py. It is found that the passive scalar remained highly concentrated in the 2 meter domain for the duration of the simulation, indicating that the six feet rule often discussed in COVID-19 healthcare guidelines would not be fully effective at preventing the spread of an aerosol spray. Fortunately, aerosols are less likely to spread the virus compared to airborne droplets. The code developed here provides an HPC-ready scalable platform for implementation of Lagrangian particle tracking that could be further used to study the transport of droplets which contain higher viral loads and have a higher likelihood of spreading COVID-19.

The code performance is evaluated using Pyinstrument profiling as well as strong and weak scaling analyses. The weak scaling of the solver apparently leveled out at 40% of serial efficiency beginning at 16 processors until 48 processors. The strong scaling of the solver shows good results, indicating that it would still scale well with access to more processors. The results of the code performance study indicate that the solver is scalable and ready to perform highly parallel calculations. The code for this project can be found at https://github.com/mpw2/me344-project The computational resources for this project are provided through the Stanford University ME344 course.

References

[1] Rajat Mittal, Rui Ni, and Jung-Hee Seo. The flow physics of covid-19. *Journal of fluid Mechanics*, 894, 2020.

[2] John David Anderson and J Wendt. *Computational fluid dynamics*, volume 206. Springer, 1995.

[3] Jianjian Wei and Yuguo Li. Human Cough as a Two-Stage Jet and Its Role in Particle Transport. *PloS one*, 12(1):e0169235–e0169235, jan 2017.

[4] Lisandro Dalcin. mpi4py, 2013.

[5] 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.

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

[7] 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.