CSCI 5636 “Pull Request” — Analyzing the Time and Memory Complexity of Two Numerical PDE Packages in Python

Adam Prieto
Dr. Jed Brown
Numerical Solutions to Partial Differential Equations
University of Colorado Boulder
11 December 2023

Introduction

After much thought and deliberation from Dr. Brown, I eventually decided to have my semester project focus on analyzing the time and memory complexity of two Numerical PDE packages. Each package is Python based, and designed to get aspiring PDE solvers up and running as simply as possible. However, as my PR will show, getting users up and running comes with a few drawbacks regarding system hardware requirements and installation.

Packages

To begin, I decided to test the *FiPy* and *Clawpack* python packages. Each package is uniquely suited for solving certain kinds of PDEs and produces unique solutions to its development.

A Note on Code Usage

Both FiPy and Clawpack are open source code packages. This means that users are free to access code from publicly available repositories and edit such files for their own needs. In the case of my project, I will be accessing all code from such publicly available repositories and citing where I accessed such code.

System Hardware and Software

I installed and ran all packages on an early 2015 Macbook Pro with a Dual Core i5 processor and 8GB of DDR3 memory. My Macbook Pro comes fully equipped with a running version of Bash and Python which I can access from my

computer's terminal. All code will be written in the "Sublime Text" text editor before being run on the terminal.

Installation

Although both packages are "fairly" easy to get up and running on any individual system, documentation and help can be hard to come by, so it's worth noting that each package can come with difficulty. As a Computer Science graduate student with no previous PDE solver experience, I definitely would have benefitted from additional documentation and resources on installing and running examples with each package.

FiPy

FiPy is a finite volume PDE solver designed to solve PDEs using both structured and unstructured meshes. Since the package is entirely based in Python, it's not *too* difficult for regular mathematical programmers to pick up, and the package has many applications across various STEM fields.

Although FiPy has many applications, I decided to test the package to solve CahnHilliard problems since the package is suited for using finite volume methods. Information on run time and memory usage is listed below.

Problem	Run Time	Memory Complexity
Mesh 2D	2 mins, 30 sec	6.67 GB
Cahn Hilliard Test Program	0.646 sec, 0.625 sec, 0.687 sec	3.4 MB, 7 MB, 3.5 MB

Findings above show that both memory and time complexity is pretty consistent for the provided "Test" program, and is also pretty consistent with Pyclaw's findings (listed down below). However, running Mesh 2D is where things get complicated.

When running the mesh 2D example, I repeatedly encountered a significant increase in run time and memory usage. As for why I encountered this problem, I have a few ideas. First, since FiPy is designed for flexibility and generality, the underlying abstractions and object-oriented design patterns that the developers chose will come with the possibility of encountering unforeseen problems. In this case, the overall time and memory requirements for this particular problem

proved to be too much for my macbook to handle. Other reasons for this particular hang up include FiPy's preferred interface for viewing solutions. Regardless, while FiPy offers a fairly seamless transition to getting started with Numerical PDE solvers, it does come with a potential overhead of increased resource usage and running into potentially unforeseen problems like I did.

Pyclaw

The Pyclaw PDE package is used for solving hyperbolic PDEs with a focus on solving problems in wave propagation and other conservation laws. The parent package, "Clawpack", stands for "Conservation Laws Package" and also provides a set of tools and solvers for working on unstructured grids.

Although Pyclaw also uses high resolution finite volume methods, this package is primarily geared towards solving hyperbolic PDEs. Below are relevant results for running the package on two problems.

Problem	Run Time	Memory Complexity
1d Euler Shock-Tube Problem	0.974 sec, 0.939 sec, 0.726 sec	7.4 MB, 4MB, 5.5 MB
Shu-Osher Problem	0.646 sec, 0.625 sec, 0.687 sec	3.4 MB, 7 MB, 3.5 MB

Much like FiPy, Pyclaw has pretty consistent memory and time complexity when solving PDEs using high resolution finite volume methods. This is a great indication of reliability and stability since it ensures that computational resources are utilized efficiently and aids in replicability. For professional engineers and scientists, this consistency means that researchers can accurately compare results, validate models across different platforms, and perform sensitivity analysis.

Contrasting Usage Applications

It's worth noting that Pyclaw and FiPy are two completely different packages with different focuses. As I stated above, I used FiPy to solve CahnHilliard Problems while Pyclaw was used to solve Euler equations. Other specializations of these two packages are listed below.

FiPy

1. Can be used to solve a broader class of PDEs since the package is based on higher level object oriented programming (OOP) ideas and Python integration.
2. OOP abstractions means that non-professional PDE users can *generally* pick up the package more easily.

Pyclaw

1. Numerical algorithms built into the package allow Pyclaw to scale to much larger simulations and datasets.
2. The package is particularly suited for solving Hyperbolic PDEs and related wave equations.

Conclusion

In conclusion, the overall time and memory complexity between the two selected Python packages are pretty similar, and both offer valid results for their respective equations. However, after a little digging, there are valid reasons to prefer one package over another. Reasons for each are listed below.

FiPy

1. Supports structured and unstructured grids.
2. Offers a wider selection of methods to solve PDEs

Pyclaw

1. Hyperbolic behavior from an equation like a string or wave.
2. Conservation laws like fluids and gasses.
3. Parallel computing allows mathematicians to use all CPU cores in a problem.

Sources

Code:
https://github.com/usnistgov/fipy/blob/master/examples/cahnHilliard/mesh2D.py
https://github.com/usnistgov/fipy/blob/master/examples/cahnHilliard/test.py
https://www.clawpack.org/gallery/pyclaw/gallery/shocktube.html
https://www.clawpack.org/gallery/pyclaw/gallery/shocksine.html

Websites:
-https://www.ctcms.nist.gov/fipy/
-https://github.com/cu-numpde/fall23-community-Adam-Prieto/blob/main/proposal.md