Quantitative Economics Workshop Paris

Introduction (and why Python?)

John Stachurski

September 2022

Today

- Introduction to scientific computing with Python
- Fixed points and job search
- Dynamic programming: theory and algorithms
- Parallelization on the GPU

Assumptions:

- You have read lectures 1-3 at

  `https://python-programming.quantecon.org/intro.html`

- some basic familiarity with programming

- https://github.com/QuantEcon/rse_comp_econ_2022

Background — Language Types

Proprietary

- Excel
- MATLAB
- STATA, etc.

Open Source

- Python
- Julia
- R

closed and stable vs open and fast moving

Background — Language Types

Low level

- C/C++
- Fortran
- Rust

High level

- Python
- Ruby
- TypeScript

Low level languages give us fine grained control

Example. $1 + 1$ in assembly

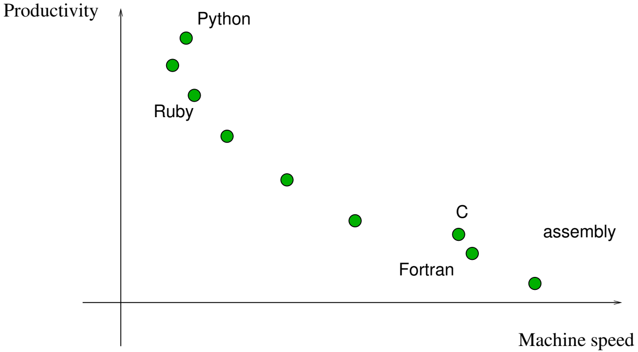
```
pushq   %rbp
movq    %rsp, %rbp
movl    $1, -12(%rbp)
movl    $1, -8(%rbp)
movl    -12(%rbp), %edx
movl    -8(%rbp), %eax
addl    %edx, %eax
movl    %eax, -4(%rbp)
movl    -4(%rbp), %eax
popq    %rbp
```

High level languages give us abstraction, automation, etc.

Example. Reading from a file in Python

```
data_file = open("data.txt")
for line in data_file:
    print(line.capitalize())
data_file.close()
```

Trade-Offs



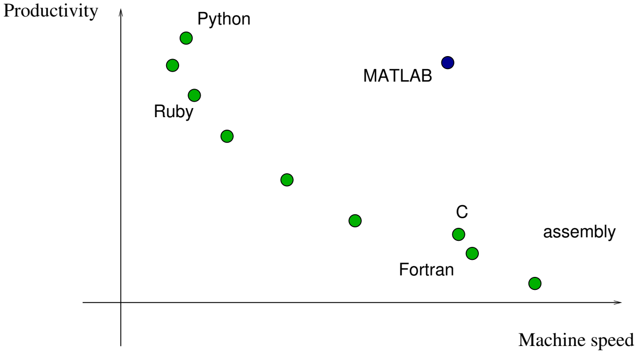
Productivity
Python
Ruby
C
assembly
Fortran
Machine speed

But what about scientific computing?

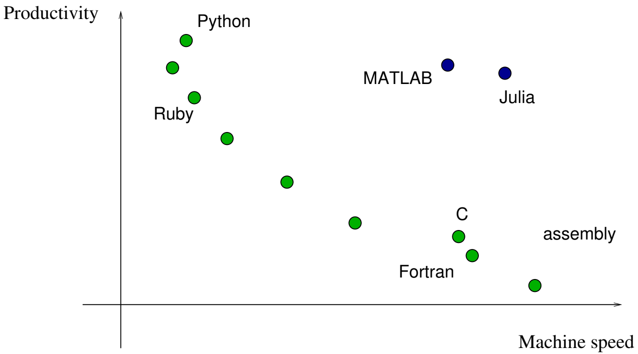
Requirements

- Productive — easy to read, write, debug, explore
- Fast computations

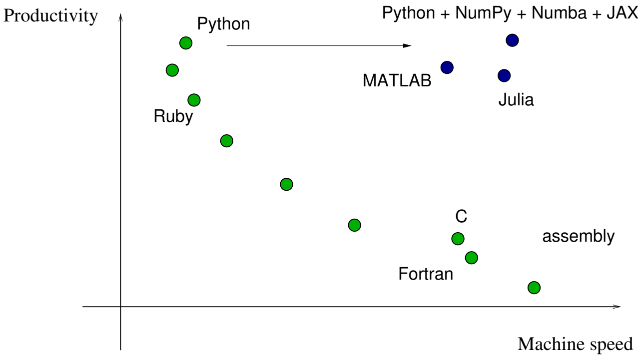
Trade-Offs


Productivity
Python
MATLAB
Ruby
C
assembly
Fortran
Machine speed

Trade-Offs

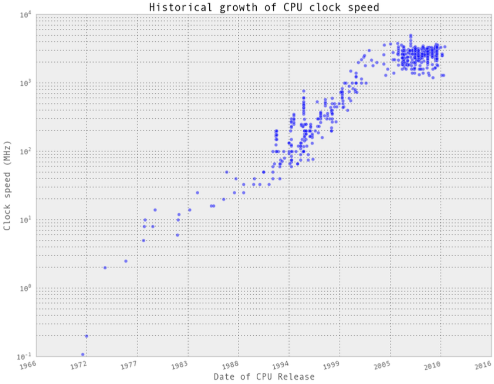

Productivity
Python
MATLAB
Julia
Ruby
C
assembly
Fortran
Machine speed

Trade-Offs

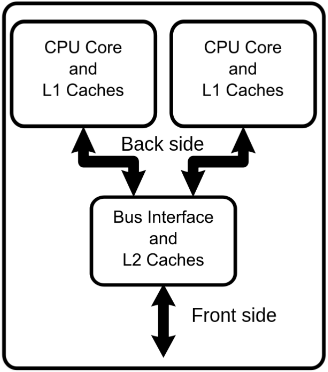

Productivity
Python
Python + NumPy + Numba + JAX
MATLAB
Julia
Ruby
C
assembly
Fortran
Machine speed

Trend 1: Parallelization

CPU frequency (clock speed) growth is slowing

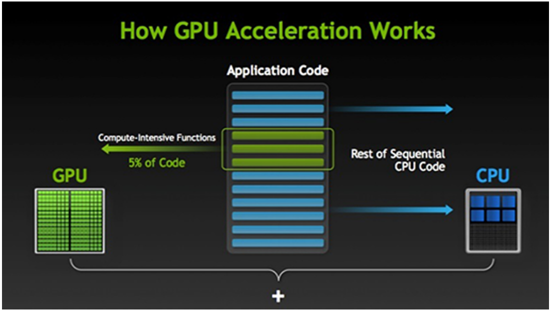


Chip makers have responded by developing multi-core processors



Source: Wikipedia

GPUs / ASICs are also becoming increasingly important



Applications: machine learning, deep learning, etc.

Trend 2: Distributed Computing

Advantages:

- run code on big machines we don't have to buy
- customized execution environments
- circumvent annoying internal IT departments

Options:

- University machines
- AWS
- Google Colab, etc.

Which Language

How about R?

- Specialized to statistics
- Easy to learn, well designed
- Huge range of estimation routines
- Significant demand for R programmers
- Popular in academia

However loosing ground to Python

Example. Chris Wiggins, Chief Data Scientist at The New York Times:

"Python has gotten sufficiently weapons grade that we don't descend into R anymore. Sorry, R people. I used to be one of you but we no longer descend into R."

Julia

Pros:

- Fast and elegant
- Many scientific routines
- Julia is written in Julia

Cons:

- Some stability issues
- Failing to achieve rapid growth

Python

- Easy to learn, well designed
- Massive scientific ecosystem
- Heavily supported by big players
- Open source
- Huge demand for tech-savvy Python programmers

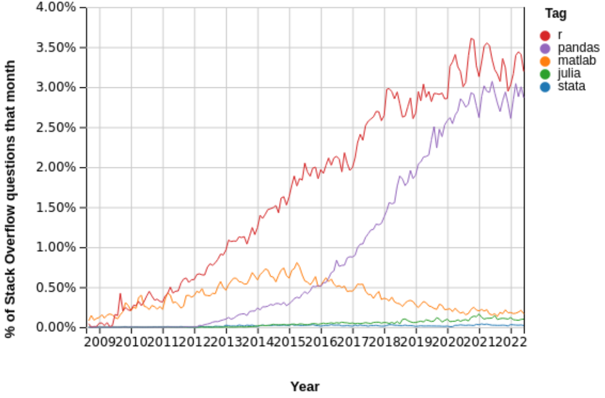
Scientific Computing

Python has strong tools in vectorization / JIT compilation / parallelization / visualization / etc.

Examples:

- SciPy, NumPy, Matplotlib, pandas
- Numba (JIT compilation, multithreading)
- Tensorflow, PyTorch (machine learning, AI)
- JAX (JIT compilation, parallelization), etc., etc.

Popularity, others vs one Python library (pandas)



Downloads / Installation / Troubleshooting

Install Python + Scientific Libs (Optional!)

- Install Anaconda from https://www.anaconda.com/
    - Select latest version
    - For your OS
    - Say “yes” at prompts
- Not plain vanilla Python

Remote options

- https://colab.research.google.com
- https://www.pythonanywhere.com/

Jupyter Notebooks

A browser based interface to Python / Julia / R / etc.

- Search for `jupyter notebook`

Useful for:

- getting started
- exploring ideas

Working with Notebooks

- Entry and execution
- Markdown
- Getting help
- Copy paste
- Edit and command mode