CBC QuantEcon Workshop

Introduction

John Stachurski

September 2022

Instructors

John Stachurski

- Researcher in asset pricing, dynamic programming, Markov process theory
- Co-founder of QuantEcon
- Based at Australian National University

Pablo Winant

- Researcher in macroeconomics and computational methods
- CREST and ESCP Business School
- Lead developer of `dolo`, `interpolation`, `NoLib`

Schedule

- September 20th - 23rd: Scientific computing with Python
- September 24th - 25th: Weekend break
- September 26th - 27th: Scientific computing with Julia

Format

- 08:30 - 10:30: Lecture
- 10:30 - 11:00: Coffee Break
- 11:00 - 13:00: Practice Sessions
- 14:30 - 16:00: Office hours

Content: Python Workshop

- Introduction to scientific computing with Python
- Dynamics (simulation, Markov chains)
- Fixed points and job search
- Dynamic programming: theory and algorithms
- Parallelization on the GPU

Assumptions:

- Some familiarity with programming / computation
- linear algebra, multivariate calculus, etc.
- have seen some dynamic programming

Resources:

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

Background

The theme of Part 1 is economic modeling Python

Why use Python for economic modeling?

In fact, why use Python for scientific computing?

Let's compare Python to some of the alternatives

Language Types

Low level

- C/C++
- Fortran
- assembly

High level

- Python
- Ruby
- TypeScript

Low level languages give us fine grained control

Example. $1 + 1$ in assembly

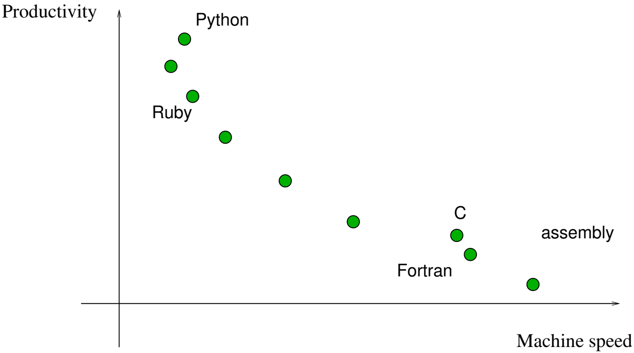
```
pushq   %rbp
movq    %rsp, %rbp
movl    $1, -12(%rbp)
movl    $1, -8(%rbp)
movl    -12(%rbp), %edx
movl    -8(%rbp), %eax
addl    %edx, %eax
movl    %eax, -4(%rbp)
movl    -4(%rbp), %eax
popq    %rbp
```

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

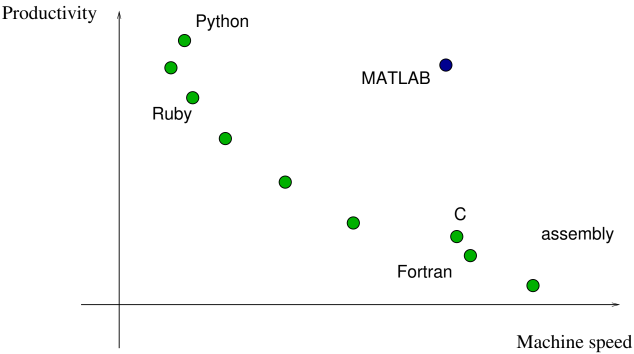

Productivity
Python
Ruby
C
assembly
Fortran
Machine speed

But what about scientific computing?

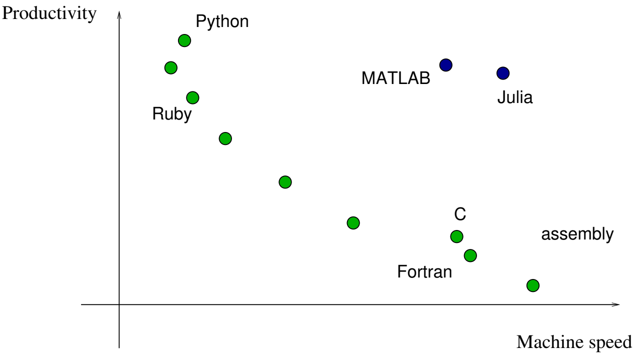
Requirements

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

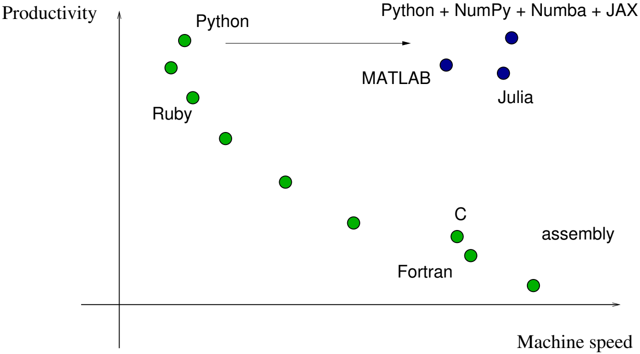
Trade-Offs


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

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

Loosing some 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:

- More niche than Python
- Yet to achieve rapid growth

Python

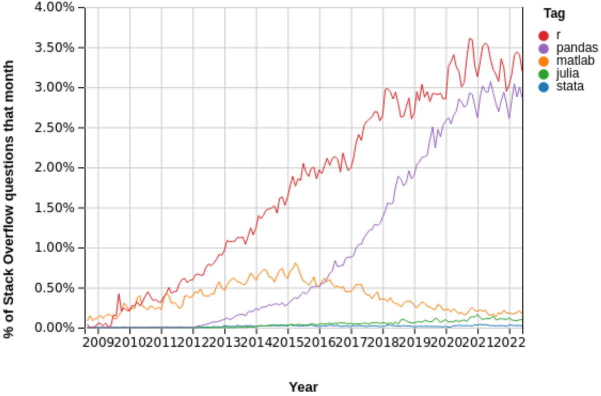
Pros:

- Massive scientific ecosystem
- Strong investment from Google, Facebook (Meta), etc.
- Huge demand for tech-savvy Python programmers

Cons:

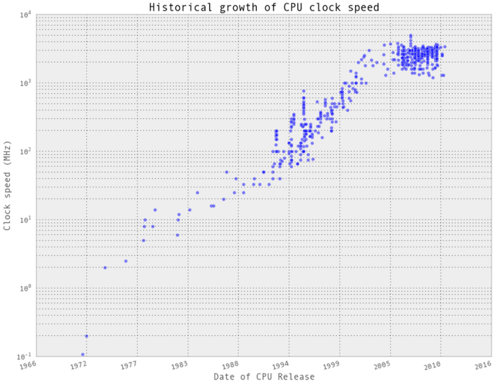
- Numerical code not as clean as Julia
- Econometrics libraries less developed than R/STATA

Popularity, others vs one Python library (pandas)

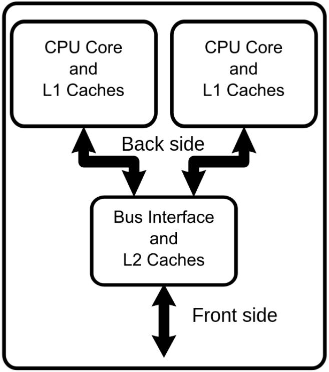


Current Trend 1: Parallelization

CPU frequency (clock speed) growth is slowing

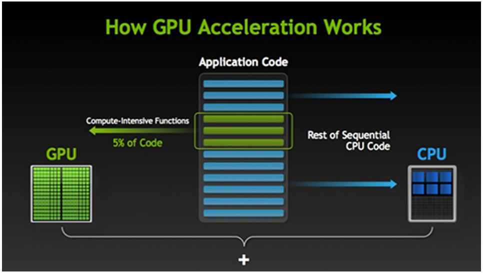


Chip makers have responded by developing multi-core processors



Source: Wikipedia

GPUs / TPUs are becoming more and more important


How GPU Acceleration Works
Application Code
Compute-Intensive Functions
5% of Code
Rest of Sequential CPU Code
GPU
CPU
+

Applications:

- machine learning, deep learning, etc.
- dynamic programming!

Trend 2: Distributed Computing

Advantages:

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

Options:

- Google Colab, PythonAnywhere, etc.
- AWS
- Supercomputers

Downloads / Installation / Remote Options

Install Python + scientific libraries (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

Notebooks

Now let's move on to

- `python_by_example.ipynb`
- `finite_markov.ipynb`
- `equilibrium.ipynb`
- `numba.ipynb`
- `european_option_numba.ipynb`