Computational Methods for Quantitative Economics

John Stachurski

April 2023

Topics

- Discussion of scientific computing
- Computing equilibria
- Option pricing with Python
- High dimensional problems

Assumptions:

- basic econ/computer/maths/stats
- some programming?

Aims:

- Discuss options
- Review trends
- Learn techniques

Resources

- https://github.com/QuantEcon/kobe_comp_econ_2023

Trends

What are the major trends in scientific computing?

- what's driving them?
- how can we benefit?

Trend 1: Proprietary → Open Source

Proprietary

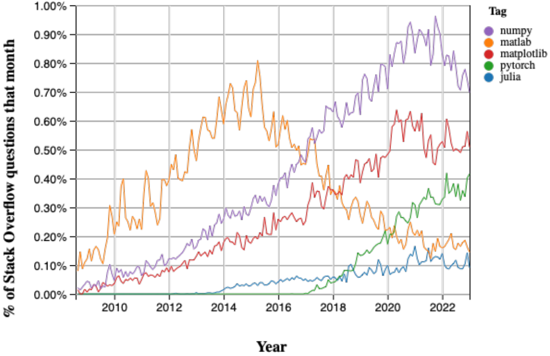
- Excel
- MATLAB, Mathematica
- STATA, Eviews, SPSS.

Open Source / Open Standard

- Python
- Julia
- R

closed and stable vs open and fast moving

Popularity:


1.00%
0.90%
0.80%
0.70%
0.60%
0.50%
0.40%
0.30%
0.20%
0.10%
0.00%
% of Stack Overflow questions that month
2010
2012
2014
2016
2018
2020
2022
Year
Tag
numpy
matlab
matplotlib
pytorch
julia

Trend 2: Low Level $\rightarrow$ High Level

Low level

- C/C++
- Fortran
- Assembly

High level

- Python
- Javascript
- PHP

Low level languages give us control

- control CPU
- control memory

High level languages give us

- abstraction
- automation
- flexibility, etc.

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
```

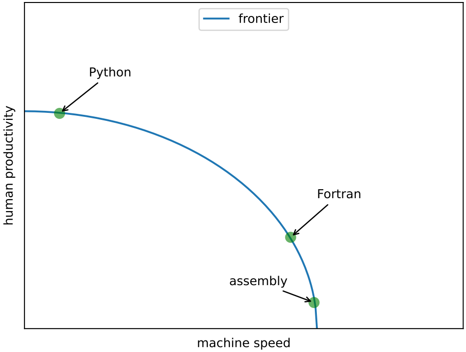
Example. $1 + 1$ in C

```c
#include <stdio.h>
int main() {
    int sum = 1 + 1;
    printf("1 + 1 = %d\n", sum);
    return 0;
}
```

Example. $1 + 1$ in Python

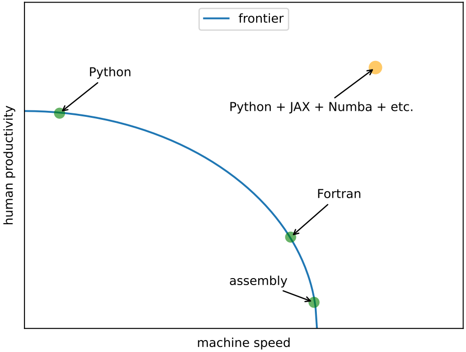
```
sum = 1 + 1
print("1 + 1 = ", sum)
```

Trade-Offs:


frontier
Python
Fortran
assembly
human productivity
machine speed

New trend — a shifting frontier

Trade-offs:


frontier
Python
Python + JAX + Numba + etc.
Fortran
assembly
human productivity
machine speed

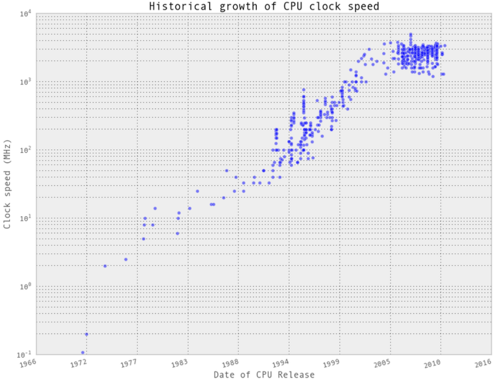
Example. What platforms/languages does OpenAI use?

In order (according to repo stats):

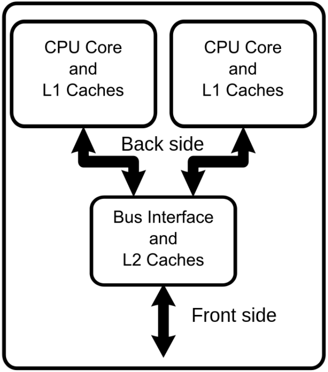
1. Python
2. C++
3. Javascript
4. Jupyter notebooks
5. Ruby

Trend 3: Parallelization

CPU frequency (clock speed) growth is slowing

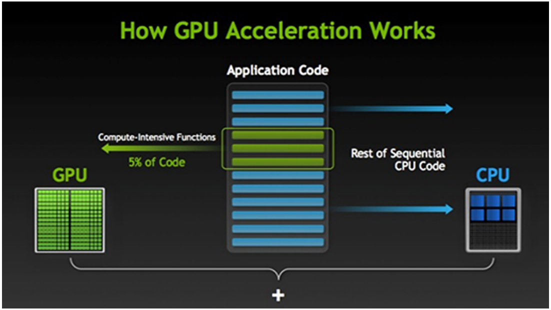


Chip makers have responded by developing multi-core processors



Source: Wikipedia

GPUs are becoming increasingly important



Applications: machine learning, deep learning, etc.

Support for Parallelization

While scientific computing environments best support parallelization?

- Most have some support
- but which make it easy to harness its power?

Current winner:

- Google JAX (Python library)

Which Language

How about R?

- Specialized to statistics
- Huge range of estimation routines
- Popular in academia
- Loosing some ground to Python (AI, machine learning)

Julia

Pros:

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

Cons:

- Low rates of investment in some important libraries

Python

- Easy to learn, well designed
- Massive scientific ecosystem
- Heavily supported by big players
- Strong support for parallel computing
- Huge demand for tech-savvy Python programmers

Accessing Python

Option 1: Via a service (remote option)

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

Option 2: Local install (Python + scientific libs)

- Install Anaconda from https://www.anaconda.com/
  - Select latest version
- Not plain vanilla Python

How to Interact with Python?

Many options:

- **write** with VS Code / Emacs / Vim
- **run** with base Python, IPython, etc.

Or do both with **Jupyter notebooks / Jupyter lab**

- for simplicity we focus only on the last option

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