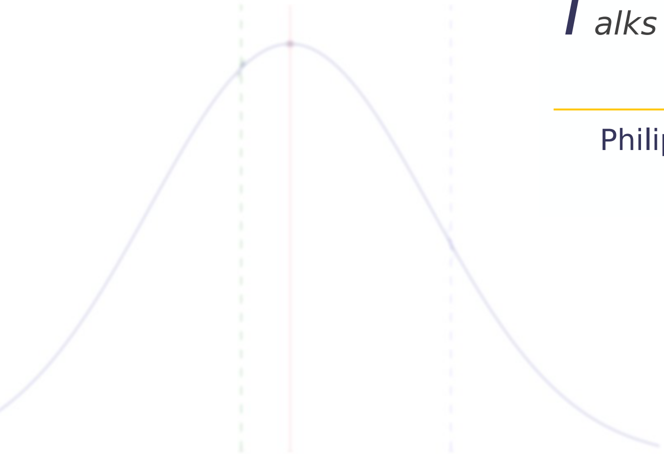


Talks

Philipp Eisenhauer

Material available on

Visit us!

UNIVERSITÄT BONN

High-performance computing using Python

Philipp Eisenhauer

I draw on the material presented in:

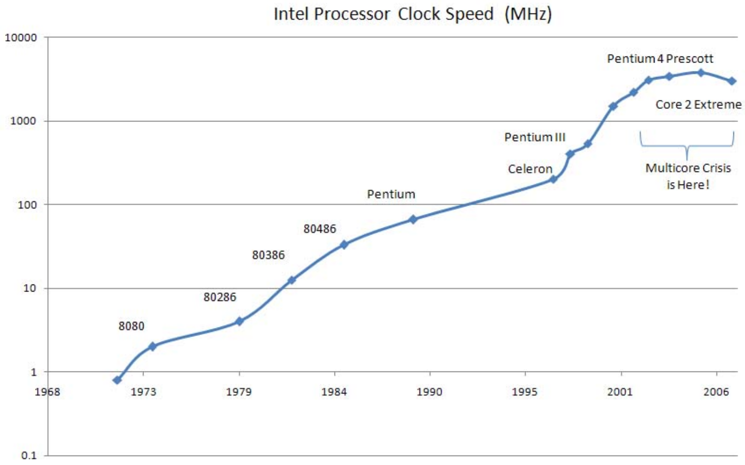
- Gorelick, M., & Ozsvald, I. (2014). *High performance python*.
- Lanaro, G. (2017). *Python high performance*.

Basic architecture

- computing units, e.g. CPU and GPU
- memory units, e.g. RAM and hard disk
- connections

Main properties of computing unit

- number of operations in one cycle, e.g. vectorization
- how many cycles in one second


Intel Processor Clock Speed (MHz)
10000
1000
100
10
1
0.1
1968
1973
1979
1984
1990
1995
2001
2006
8080
80286
80386
80486
Pentium
Celeron
Pentium III
Pentium 4 Prescott
Core 2 Extreme
Multicore Crisis is Here!

Global interpreter lock

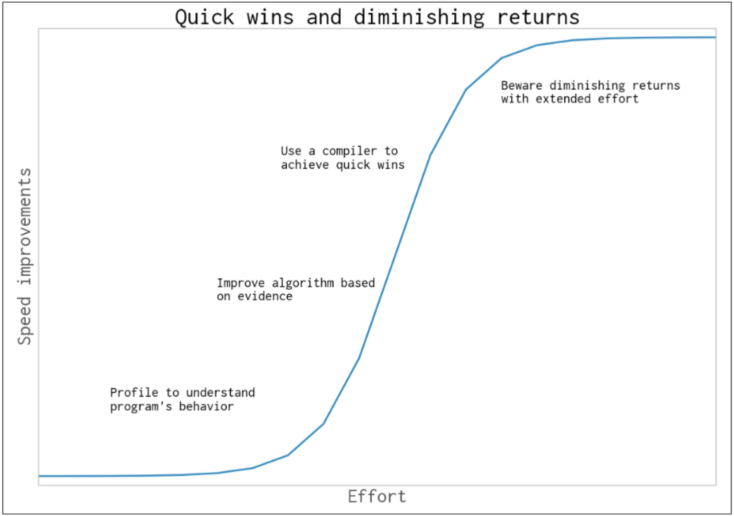
The GIL makes sure that a Python process can only run one instruction at a time, regardless of the number of cores it is currently using. This means that even though some Python code has access to multiple cores at a time, only one core is running a Python instruction at any given time.

Profiling

- *Premature optimization is the root of all evil.*
- focus on readability
- set up development environment
- flesh out testing harness
- tackle performance bottlenecks

Points of attack

- pure Python
- high-performance libraries, SciPy Stack
- compilation to faster language
- parallel computing
- distributed computing

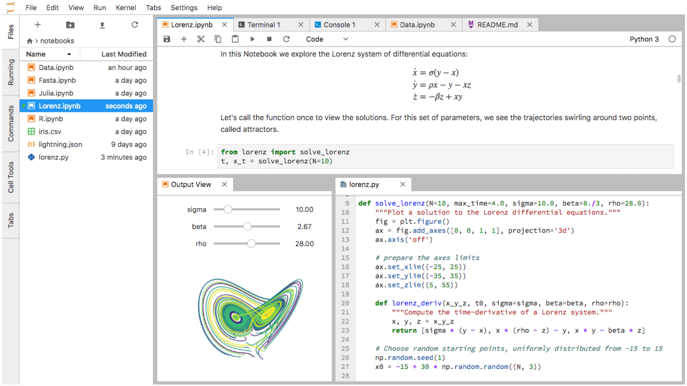

Quick wins and diminishing returns
Beware diminishing returns
with extended effort
Use a compiler to
achieve quick wins
Improve algorithm based
on evidence
Profile to understand
program's behavior
Speed improvements
Effort

High-performance libraries

Vectorization

- Vectorization is when a CPU is provided with multiple pieces of data at a time and is able to operate on all of them at once. This sort of CPU instruction is known as SIMD (Single Instruction, Multiple Data).

Practical illustration

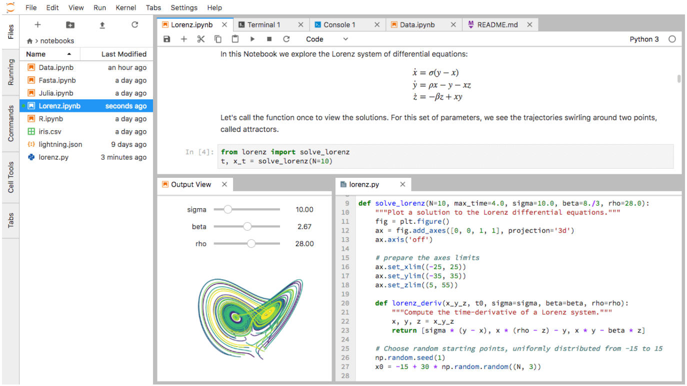


Compilation to faster language

Alternatives

- ahead-of-time, e.g. f2py, Cython
- just-in-time, e.g. numba

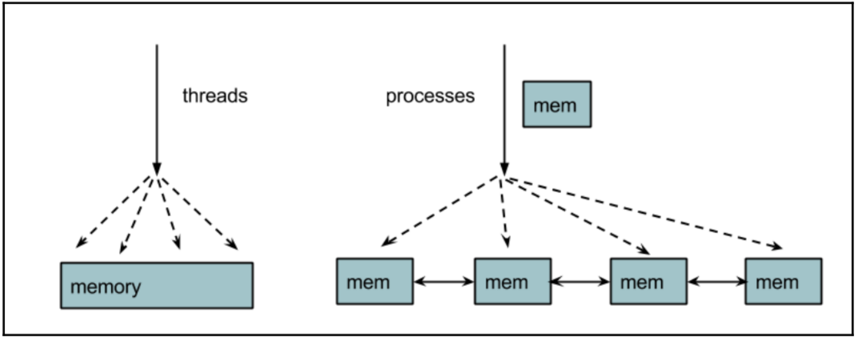
Practical illustration



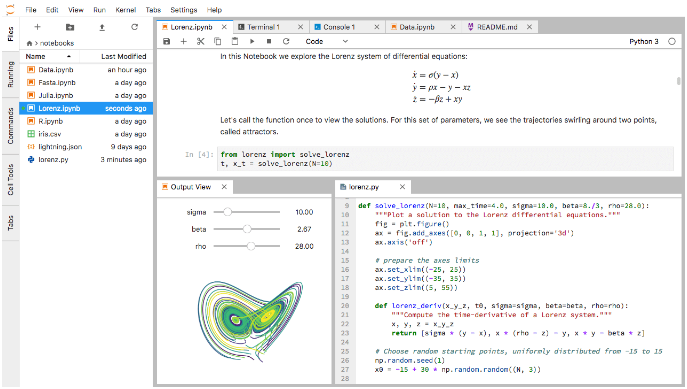
Parallel computing

Memory access

- shared memory
- distributed memory


threads
memory
processes
mem
mem
mem
mem
mem

Practical illustration



Distributed computing

Challenges to parallelization

- algorithm limitations
- bottlenecks
- startup overhead
- communication

Frameworks

- https://dask.org
- https://joblib.readthedocs.io
- https://mpi4py.readthedocs.io

Communications

- point-to-point communications
- collective communications
- dynamic process management

Resources

Textbooks

- Lanaro, G. (2017). *Python high performance*.
- Gorelick, M., & Ozsvald, I. (2014). *High performance python*.
- Hager, G., & Wellein, G. (2011). *Introduction to high performance computing for scientists and engineers*.

Appendix

References

Gorelick, M., & Ozsvald, I. (2014). *High performance python*.
Hager, G., & Wellein, G. (2011). *Introduction to high performance computing for scientists and engineers*.
Lanaro, G. (2017). *Python high performance*.