Fundamentals of Performance Tuning for Python Applications

Alexey Svyatkovskiy

Princeton University

October 25, 2017

Outline

1. Programming languages
2. Ways to boost the performance of Python applications
3. Main part: Hands-on exercises

The Big Three: a basis set of features

C/C++	Java/JVM	Python
compiles to machine code	compiles to VM; optimizes at runtime	interpreted bytecode; runtime type-checks

The Big Three: a basis set of features

C/C++	Java/JVM	Python
compiles to machine code	compiles to VM; optimizes at runtime	interpreted bytecode; runtime type-checks
fastest; reproducible performance	slow start and intermittent pauses, but relatively high throughput	quick to start, but terrible throughput

The Big Three: a basis set of features

C/C++	Java/JVM	Python
compiles to machine code	compiles to VM; optimizes at runtime	interpreted bytecode; runtime type-checks
fastest; reproducible performance	slow start and intermittent pauses, but relatively high throughput	quick to start, but terrible throughput
bare metal and high abstractions coexist	choice of language, some high, some low	generally only one way to do things
manual memory management (including smart pointers)	generational garbage collectors, tunable to improve throughput	simple reference counting garbage collector
language superset of C, others through FFI	hard to break out of the VM	excellent interface to other languages

What happens in the compilation step?

- Whole program is interpreted and turned into machine instructions (possibly for a virtual machine).
- All variables are interpreted as belonging to specific types: `int`, `string`, `MissileController`...
- Uses of these variables are checked for validity:
  - can't pass a `MissileController` into the cosine function;
  - can't call `launchAllMissiles()` on a `string`.

Interpreted languages do none of these things; you find out about misuses of variables at runtime (can be good, can be bad).

Python

- Powerful built-ins
- Object oriented
- Rich libraries
- Dynamic typing

There are two slightly inconsistent versions of python in the wild, python 2.x and python 3.x Within the 2.x series (currently 2.7.12) features were added from time to time. Eventually we'll all have to move to python 3 (currently at 3.5).

Purpose of the course

- Discuss fundamentals of performance tuning for Python applications
- Learning how to use tools like Numpy, Cython, Tensorflow
- Interface Python with compiled code with Ctypes
- Parallelizing code with Multiprocessing and Joblib libraries
- In addition, it discuss how to take advantage of Princeton University computing resources
- Demo: Numba and PyCUDA - Python applications on GPU

Role of computing in Science and Engineering

Computation has joined experiment and theory to become the third component of science. Nowadays, even social scientist need computing resources to carry out their research and extract meaningful insights.

Three stages of scientific computing might be identified as:

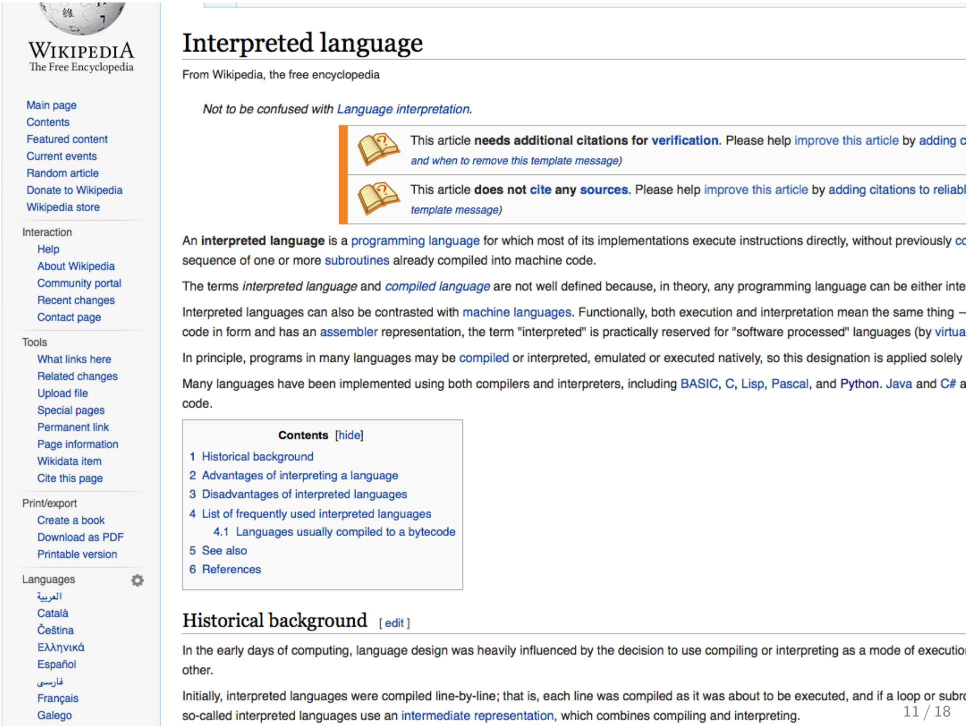
- model building: produces equations
- simulation/experiment: produces data
- analysis and visualization

Performance tuning is essential to achieve accurate results in a timely manner.

Python development environments

For this course, we will use Jupyter notebook and occasionally I will resort to the command line and vim text editor.
You can refer to the following wikipedia entry to learn about various development environments you can use with Python: `http://en.wikipedia.org/wiki/Integrated_development_environment`. An integrated development environment (IDE) includes language-aware editors, compilers/interpreters, build systems, debuggers, etc.

- IDLE is an example of a Python-specific IDE.
- Microsoft Visual Studio and Xcode (for OS X) each support several programming languages.
- The best examples of multi-language cross-platform IDEs are Eclipse and Netbeans, though the latter does not currently support Python.



Interpreted language

From Wikipedia, the free encyclopedia

Not to be confused with Language interpretation.

This article **needs additional citations for verification**. Please help improve this article by adding c
and when to remove this template message)

This article **does not cite any sources**. Please help improve this article by adding citations to reliabl
template message)

An **interpreted language** is a programming language for which most of its implementations execute instructions directly, without previously co
sequence of one or more subroutines already compiled into machine code.

The terms *interpreted language* and *compiled language* are not well defined because, in theory, any programming language can be either inte

Interpreted languages can also be contrasted with machine languages. Functionally, both execution and interpretation mean the same thing –
code in form and has an assembler representation, the term "interpreted" is practically reserved for "software processed" languages (by virtua

In principle, programs in many languages may be compiled or interpreted, emulated or executed natively, so this designation is applied solely

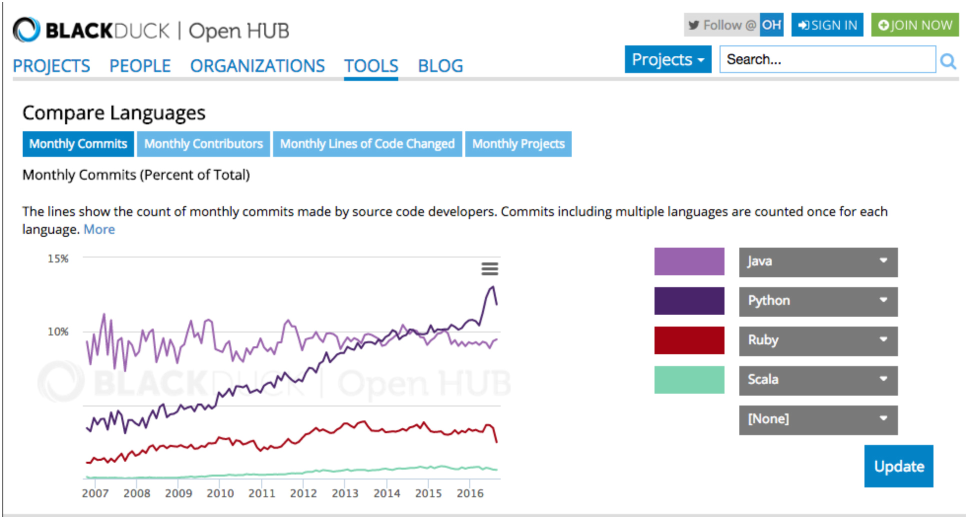
Many languages have been implemented using both compilers and interpreters, including BASIC, C, Lisp, Pascal, and Python. Java and C# a
code.

Contents [hide]

Historical background [edit]

In the early days of computing, language design was heavily influenced by the decision to use compiling or interpreting as a mode of executio
other.

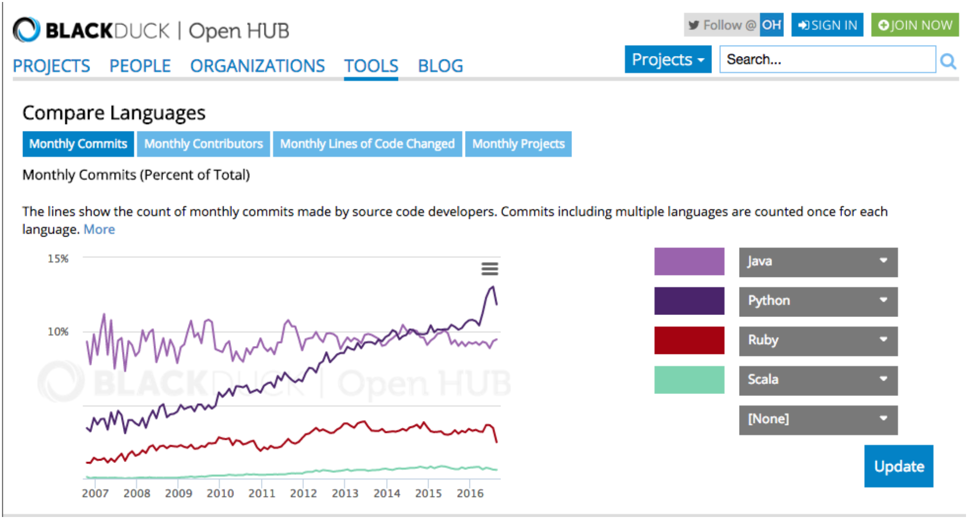
Initially, interpreted languages were compiled line-by-line; that is, each line was compiled as it was about to be executed, and if a loop or subro
so-called interpreted languages use an intermediate representation, which combines compiling and interpreting.



Compare Languages

Monthly Commits (Percent of Total)

The lines show the count of monthly commits made by source code developers. Commits including multiple languages are counted once for each language. More


BLACKDUCK | Open HUB

Follow @OH · SIGN IN · JOIN NOW

PROJECTS · PEOPLE · ORGANIZATIONS · TOOLS · BLOG · Projects · Search...

Compare Languages

Monthly Commits · Monthly Contributors · Monthly Lines of Code Changed · Monthly Projects

Monthly Commits (Percent of Total)

The lines show the count of monthly commits made by source code developers. Commits including multiple languages are counted once for each language. More

NumPy

Why numpy arrays can accelerate your applications?

- A NumPy ndarray array is described by number of dimensions, shape, data type, and the actual data stored in it. The data is stored in a homogeneous and contiguous blocks of memory, at a particular address in system memory RAM
- Spatial locality in memory access patterns results in significant performance gains, notably thanks to the CPU cache.
- Data elements are stored contiguously in memory, so that NumPy can take advantage of vectorized instructions on modern CPUs, like Intel's SSE and AVX, AMD's XOP
- NumPy can be linked to highly optimized linear algebra libraries like BLAS and LAPACK, for example through the Intel Math Kernel Library (MKL).
- Storing data in a contiguous blocks of memory ensures that the architecture of modern CPUs is used optimally, in terms of memory access patterns, CPU cache, and vectorized instructions

Cython

Cython

What is TensorFlow?

- Open source Machine Learning library
- Especially useful for Deep Learning
- For research and industry
- Apache 2.0 license

Data Flow graphs

Computation is defined as a directed acyclic graph (DAG) to optimize an objective function

- Graph is defined in high-level language (Python, C++, go)
- Graph is compiled and optimized
- Graph is executed (in parts or fully) on available low level devices (CPU, GPU)
- Data (tensors) flow through the graph
- TensorFlow can compute gradients automatically

Python Hands-on Exercises