Speeding up Python

Or how to avoid recoding your (entire) application in another language

Before you Start

- Most Python projects are “fast enough”. This presentation is not for them.
- Profile your code, to identify the slow parts AKA “hot spots”.
- The results can be surprising at times.
- Don’t bother speeding up the rest of your code.

Improving your Python

- Look into using a better algorithm or datastructure next.
- EG: Sorting inside a loop is rarely a good idea, and might be better handled with a tree or SortedDict.

Add Caching

- Memorize expensive-to-compute (and recompute) values.
- Can be done relatively easily using functools.lru_cache in 3.2+.

Parallelise

- Multicore CPU’s have become common.
- Probably use threading or multiprocessing.

Threading

- CPython does not thread CPU-bound tasks well.
- CPython can thread I/O-bound tasks well.
- CPython cannot thread a mix of CPU-bound and I/O-bound well.
- Jython, IronPython and MicroPython can thread arbitrary workloads well.
- Data lives in the same threaded process. This is good for speed, bad for reliability.

Multiprocessing

- Works with most Python implementations.
- Is a bit slower than threading when threading is working well.
- Gives looser coupling than threading, so is easier to “get right”.
- Can pass data from one process to another using shared memory.

Numba

- If you’re able to isolate your performance issue to a small number of functions or methods (callables), consider using Numba.
- It’s just a decorator – simple.
- It has “object mode” (which doesn’t help much), and “nopython mode” (which can help a lot, but is more restrictive).
- nopython mode just looks like the @njit decorator

Use Python Optimizations

- https://wiki.python.org/moin/PythonSpeed/PerformanceTips
- Building an aggregate string (where fn(x) is a str):
  - Use: ‘’.join(fn(item) for item in list_)
  - Not: mystring=‘’; for item in list_: mystring += fn(item)
- Avoid .’s inside a loop.
- Avoid function and method calls.
- Avoid using globals.

Pypy

- Pypy is an alternative Python interpreter that includes a JIT out of the box.
- It’s quite a bit faster for pure python, and can be acceptably fast for C extension modules.

Cython

- Cython is a Python-to-C transpiler.
- It accepts something that almost looks like Python as input.
- It can be used to create C Extension Modules or entire programs.
- It’s a lot less error prone, and there’s a lot less boilerplate, than coding a C Extension Module by hand.
- cython3 --annotate foo.pyx: Produce a colorized HTML report; yellow lines indicate Python interaction.

Other Python → C and/or C++ transpilers

- Pythran
- Py2C
- Py14
- Py2CPP
- There may be others...

Write just your Hotspot in C or C++

- ...and call it using Cython, CFFI, Pybind11, or subprocess.
- Python's ctypes might seem like a good idea, but it can actually be a bit slow if you spend much time crossing the C ↔ Python barrier. It's also a bit brittle.
- The subprocess module can be even slower at the C ↔ Python barrier.
- CFFI and subprocess are notable for working well on CPython and Pypy.

SWIG

- SWIG can be used to interface C code with a large number of other languages, including Python.
- It can be kind of burdensome compared to Cython, if you’re only targeting Python.
- However, if you need to expose C code to a large number of languages, SWIG may be an attractive option.

Boost.Python

- Boost.Python is a library of code to facilitate C++ ↔ Python interoperability.
- It can do nice things like exporting a C++ iterator as a Python iterator.
- I haven’t tried it, but I know a lot of people like it.

Rewrite your Hotspot in Rust

- Milksnake: created by Armin Ronacher (the creator of Flask).
- rust-cpython
- PyO3: Rust bindings for CPython – a fork of rust-cpython
- Rust requires no runtime library, so I’m told it’s a pretty good fit.
- https://developers.redhat.com/blog/2017/11/16/speed-python-using-rust/

The subprocess module

- Rewrite your hotspot in any arbitrary language.
- Then use a pipe or shared memory or socket to communicate with that subprocess from your Python code.

Caveat

- You can’t use things like numba and CFFI on the same callable. You can use them in the same process, just not on the same callable.

That’s all Folks

- Questions?