Cython: Speeding up your Python code for research

Ryan Pepper

PhD student, CDT in Next Generation Computational Modelling

Content

- Why is Python slow compared to compiled languages?
- Why use Python for Science?
- What is Cython and how does it work?
- How do you use it?
- How fast is it?

Aside: Why is Python slow compared to compiled languages?

- Main implementation of Python, CPython is written in C
- Dynamic Typing - Creating things has more steps involved under the hood.

C:

```
int a, b, c;
a = 2;
b = 1;
c = a + b;
```

Python API:

```
PyObject *ap, *bp, *cp;
ap = Py_BuildValue("a", 2);
bp = Py_BuildValue("b", 1);
cp = PyNumber_Add(&ap,
&bp);
```

Python

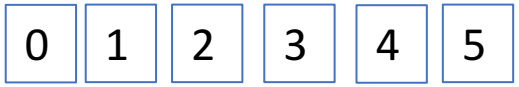
```
a = 2
b = 1
c = a + b
```

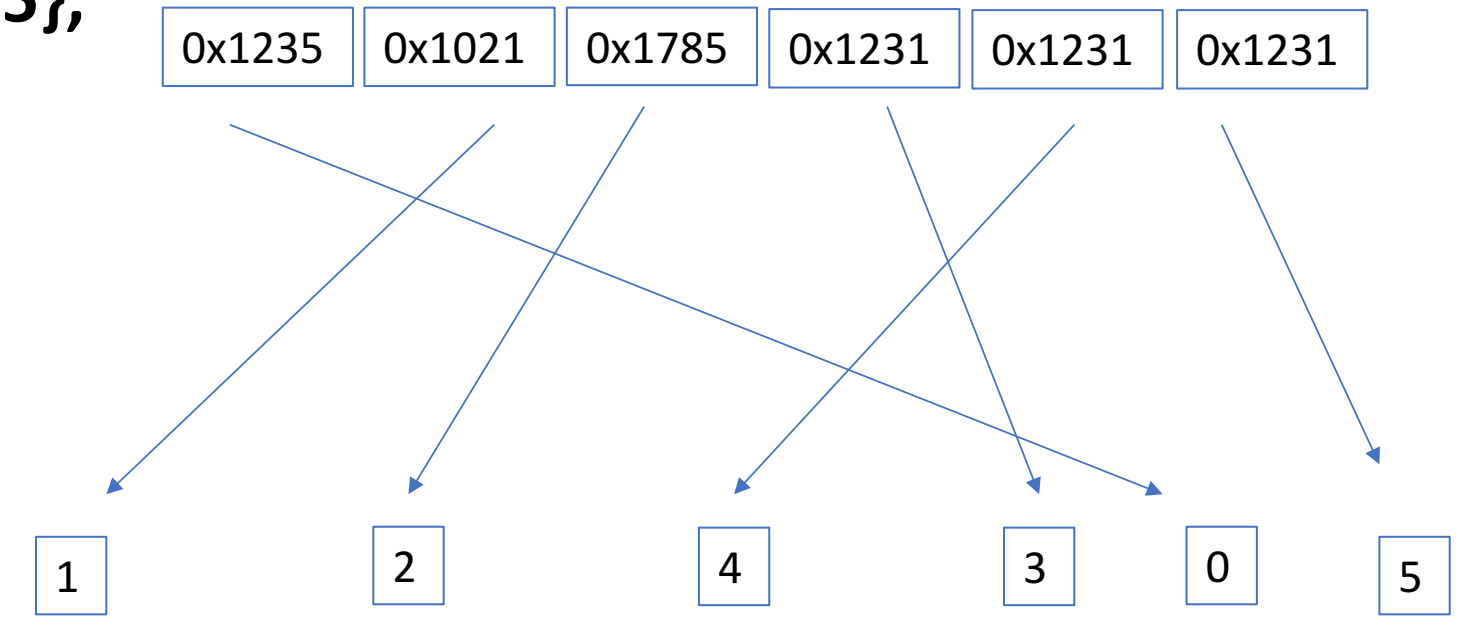
Aside: Why is Python slow compared to compiled languages?

- Memory Layout is **non-contiguous**

C:

- **int a[6] = {0, 1, 2, 3, 4, 5};**





Other stuff separates the Python Integers in Memory

- **Note: operations on Numpy arrays are faster because they ARE contiguous memory!**

Aside: Why is Python slow compared to compiled languages?

Compilers perform optimisations at compile time with optimisation flags

- Repeated operations can be removed or transformed into an operation that does the same thing – e.g. adding 2 repeatedly to a number.

C:

```
int a;
a = a + 2;
a = a + 2;
```

Better C:

```
int a;
a = a + 4;
```

GCC outputted assembler is the same for both of these sources! (test yourself with gcc –O3 -S test.c!)

The two operations are removed in favour of a faster single operation!

Why use Python then?

- It's easy to do things in (particularly loading files, manipulating data).
- Saves development time.
- We can use other tools to do things faster – people have spent a lot of time writing C/Fortran libraries and these are wrapped up into Python functions in packages such as...

Wraps BLAS, LAPACK libraries for high level matrix operations.

Various scientific libraries wrapped into modules

Computer Vision Library written in C++ with a Python Interface

Parallel Finite Element Code
Wraps C++ MPI code using a tool called SWIG

So what is Cython?

- Cython is a **cross-compiler** – it generates C code from Python which uses the Python API, which a C compiler can then compile. This gives some performance gains.
- Giving type annotations (like in C) allows Cython to further optimise the code.
- It also allows you to call directly native C functions and have them do things to data objects you can use in Python, such as numpy arrays!
- It’s an alternative to writing complicated C extensions by hand using the Python API (like Numpy does), which is hard unless you know what you’re doing and are already a competent programmer.
- You can easily write your own functions to operate on numpy arrays which will run much faster than Python code.

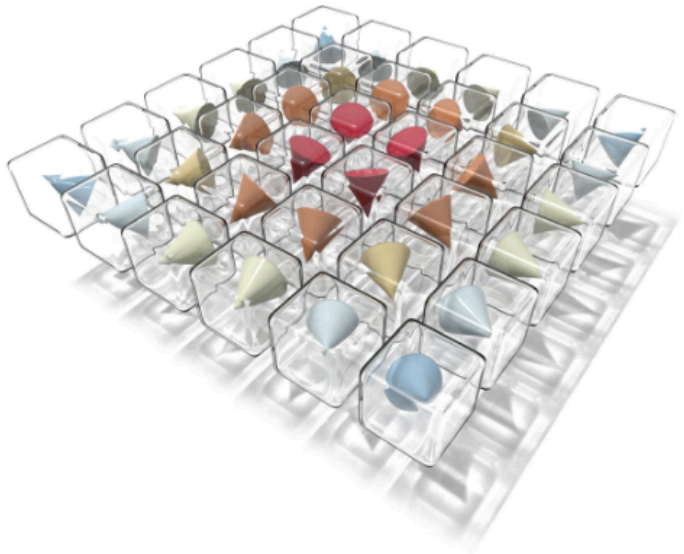
Who uses it?

Projects using Cython

- Sage math software
- lxml XML processing library
- SciPy Scientific Tools for Python
- yacairo yet another cairo binding (alpha)
- FEMhub Finite Element package
- CDS Invenio CERN Document Server Invenio
- Python-EFL Python-EFL
- Plasticity Synaptic Modification in Rate-Based Neurons (simulation package)
- GIPSY Interactive package for the processing and display of astronomical data
- Tuke an electrical design automation library
- WorldMill (now Fiona) GIS geospatial feature interface
- mpi4py MPI for Python
- petsc4py PETSc for Python
- slepc4py SLEPc for Python
- tao4py TAO for Python
- Crux Software toolkit for molecular phylogenetic inference
- Kapteyn Package Tools for astronomy (WCS, map plotting, and more)

Lots of people!

FIDIMAG



Finite differences atomistic and micromagnetic solver

David Cortés-Ortuño, Weiwei Wang, Ryan Pepper, Marc-Antonio Bisotti, Thomas Kluyver, Mark Vousden, & Hans Fangohr. (2016)

- Used in our research code to wrap finite difference operations written in C with OpenMP, and also to wrap the time integration library SUNDIALS (LLNL), and FFTW for parallelised Fourier transforms.

Real World Example

N charged bodies interacting under Coulomb potential

$$F_i = \sum_{\substack{j=0 \\ j \neq i}}^{j=N} \frac{q_i q_j}{4\pi\epsilon_0 r_{ij}^2}$$

$$dr_i = \frac{F_i dt^2}{m_i}$$

We'll show some benchmarks of various codes with parameters:

$$dt = 0.001$$

$$T = 0.1$$

$$N = 10000$$

$$m = 1$$

$$q = (-1, 1)$$

$$r_i^0 = [0, 1) \times [0, 1)$$

Speed Comparisons

Code	Time (s)	Speedup
Python	361.16	1
Cython (compile the native Python script)	324.03	1.11
Cython with Typedefs	193.59	1.86
Cython with typedefs, no bounds checking, and libc math functions	8.15	44.31
External OpenMP C code	1.41	256.141
Native C	1.68	214.98
Cython + CUDA (Geforce 750Ti)	10.63	33.97

Note: Native C code slower due to vectorised random number generation in Numpy compiled libraries
Note 2: I used CUDA for the first time yesterday, so it's probably not a fair comparison.

What is Cython not? What are the pitfalls?

- It is not magic. If your code is slow due to the algorithm you've chosen, you'd be better off choosing a better algorithm and then using Cython.
- You can crash your Python code by doing unsafe things!
- Can complicate the code base
- Not always clear where to offload things to Python – have to use profiling (not covered here for time, but a lot in the Cython documentation online).

Thanks for listening
Any questions?