Quick introduction to Cython

Prabhu Ramachandran

Enthought Inc.
Mumbai

SciPy India,
Mumbai
Dec. 28, 2012

Outline

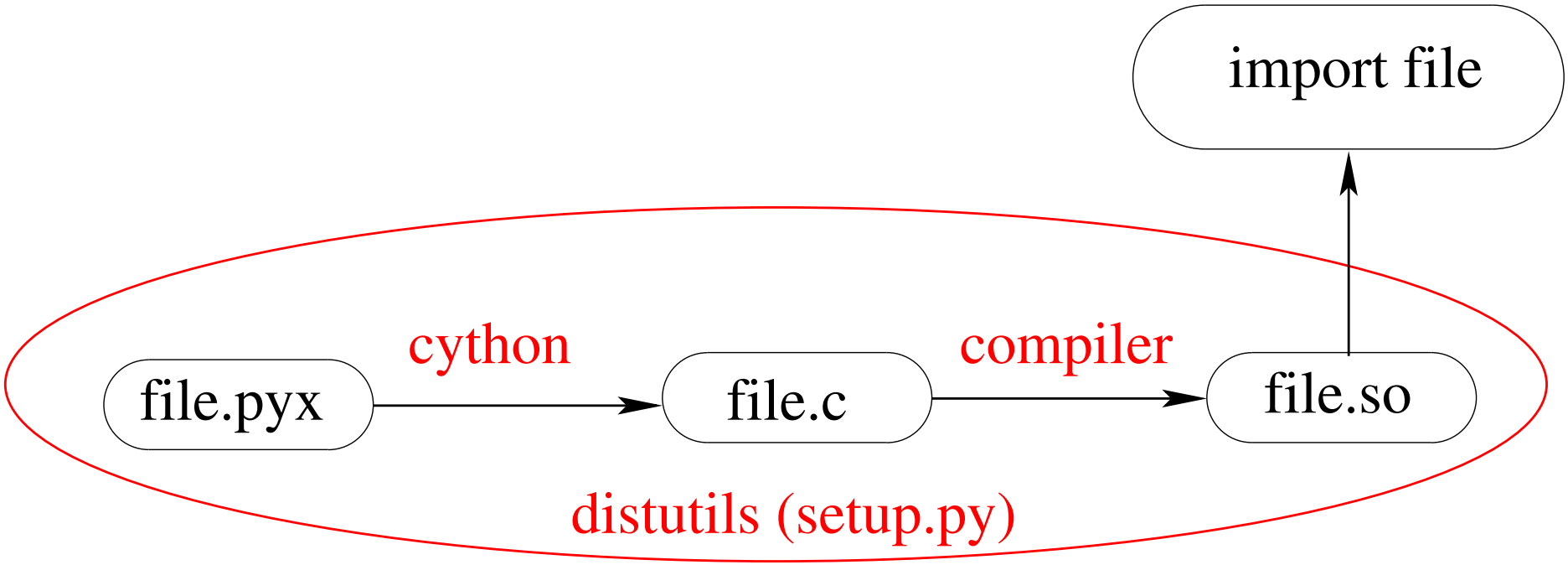
Motivation

- Pure Python can be very slow
- Not trivial to write C-extensions
- Interfacing to C libraries not too easy

Introduction

- Python-based language
- Optional static typing
- Call external C libraries
- Python to C compiler
- “Cython is Python with C-data-types”
- `cython.org`
- `docs.cython.org`
- 500x speedup is possible!

Basic structure



Simple example

```
# -- hello.pyx --
def hello(name):
    print("Hello %s!"%name)
# EOF

$ ipython
In []: import pyximport
In []: pyximport.install()
In []: import hello
In []: hello.hello('PyCon')
Hello PyCon!
```

Simple setup.py

```
# -- setup.py --
from setuptools import setup
from Cython.Distutils import build_ext
from numpy.distutils.extension import \
    Extension
setup(
    cmdclass = {'build_ext': build_ext},
    ext_modules = [Extension("hello",
                             ["hello.pyx"])
                   ]
)
# EOF
```

Compiling the code

```
$ python setup.py build_ext --inplace
# ...
$ ipython
In []: import hello
In []: hello.hello('PyCon')
Hello PyCon!
```

Meaningful example: Phase 0

```
# -- quad0.pyx --
from math import sin
def f(x):
    return sin(x**2)

def integrate(a, b, N):
    s = 0
    dx = float(b-a)/N
    for i in range(N):
        s += f(a+i*dx)
    return s * dx
```

Phase 1

```
# -- quad1.pyx --
from math import sin

def f(double x):
    return sin(x**2)

def integrate(double a, double b, int N):
    cdef int i
    cdef double s, dx
    s = 0.0
    dx = float(b-a)/N
    for i in range(N):
        s += f(a+i*dx)
    return s * dx
```

Timing

```
In []: import quad0
In []: %timeit quad0.integrate(0,1,10000)
100 loops, best of 3: 3.45 ms per loop

In []: import quad1
In []: %timeit quad1.integrate(0,1,10000)
100 loops, best of 3: 2.24 ms per loop
```

Not much.

Cython annotation

```
$ cython -a quad1.pyx
```

```
$ firefox quad1.html
```

Very handy!

15 m

Phase 2

```
# -- quad2.pyx --
from math import sin
cdef double f(double x) except *:
    return sin(x**2)
# ...
```

Could also use **cpdef** instead

```
In []: %timeit quad2.integrate(0,1,10000)
1000 loops, best of 3: 1.25 ms per loop
```

Looks like calling a Python function is slow.

Phase 3: external C functions

Calling `math.h`

```
# -- quad3.pyx --
cdef extern from "math.h":
    double sin(double)

cdef double f(double x):
    return sin(x**2)
# ...

In []: %timeit quad3.integrate(0,1,10000)
1000 loops, best of 3: 276 us per loop
```

Extension classes

```
cdef class Function:
    cpdef double eval(self, double x) except *:
        return 0

cdef class SinOfSquareFunction(Function):
    cpdef double eval(self, double x) except *:
        return sin(x*x)

def integrate(Function f, double a, double b,
              int N):
    # ...
    for i in range(N):
        s += f.eval(a+i*dx)
    return s * dx
```

Extension classes

```
cdef class Function:
    cpdef double eval(self, double x) except *:
        return 0

cdef class SinOfSquareFunction(Function):
    cpdef double eval(self, double x) except *:
        return sin(x*x)

def integrate(Function f, double a, double b,
              int N):
    # ...
    for i in range(N):
        s += f.eval(a+i*dx)
    return s * dx
```

Extension classes

- Can be extended in Python
- No multiple inheritance
- Cannot subclass Python class in Cython
- Can have typed attributes
- More info: `docs.cython.org`

25 m

NumPy support

```
import numpy as np
cimport numpy as np

# Declare numpy arrays.
cdef np.ndarray arr

# Better still
cdef np.ndarray[np.int64_t, ndim=2] arr
```

Plot the Mandelbrot set

- Consider points in the complex plane. For each point, c, calculate $z_{n+1} = z_n^2 + c$, with $z_0 = 0$.
- Note that if $z_i > 2$ the solution will diverge.
- We would like to plot the number of iterations a point diverges in.
- We are given `w, h` as width and height of image.
- Region of interest is [-2, -1.4] to [0.8, 1.4].

Naive solution

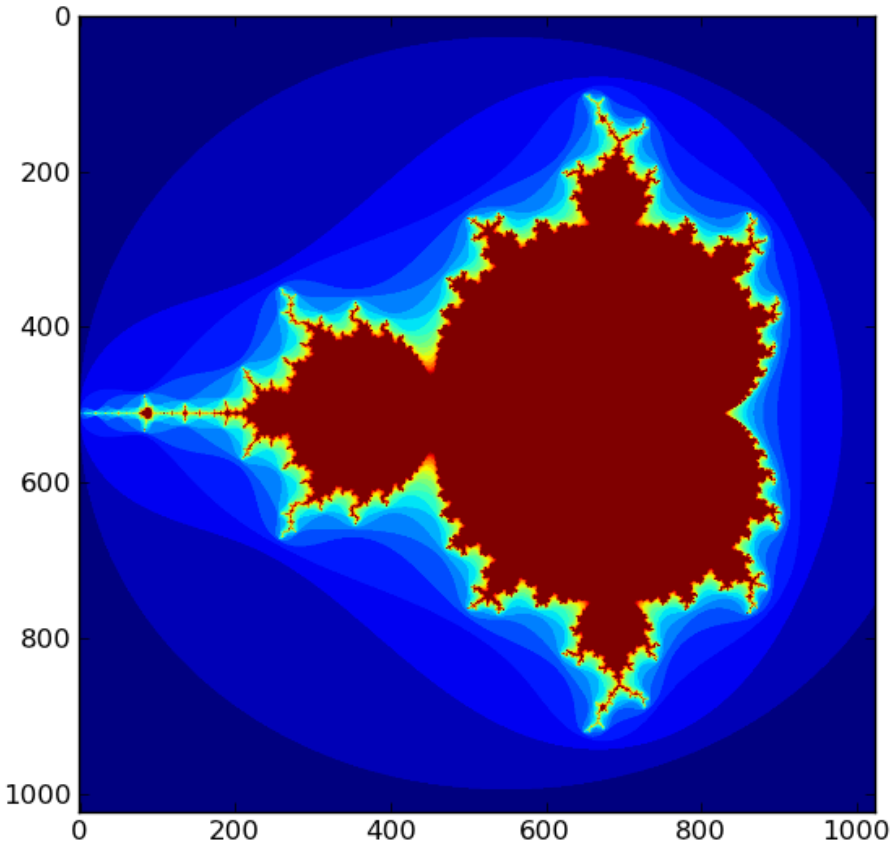
```
def mandel_py(h, w, maxit=20):
    """Returns the Mandelbrot set."""
    x, y = np.ogrid[-2:0.8:w*1j, -1.4:1.4:h*1j]
    c = x+y*1j
    output = np.zeros((w, h), dtype=int) + maxit
    for i in range(h):
        for j in range(w):
            z = c[i,j]
            c0 = c[i,j]
            for k in xrange(maxit):
                z = z**2 + c0
                if z*z.conjugate() > 4.0:
                    output[i, j] = k
                    break
    return output.T
```

NumPy solution

```
def mandel_np(h, w, maxit=20):
    """Returns the Mandelbrot set."""
    x, y = np.ogrid[-2:0.8:w*1j, -1.4:1.4:h*1j]
    c = x+y*1j
    z = c
    output = np.zeros((w, h), dtype=int) + maxit
    for i in xrange(maxit):
        z = z**2 + c
        diverged = z*z.conj() > 4
        c[diverged] = 0
        z[diverged] = 0
        output[diverged] = i
    return output.T
```

The result

```
In []: from mandel import mandel_np
In []: o = mandel_np(1024,1024)
In []: imshow(o)
```



Cython example I

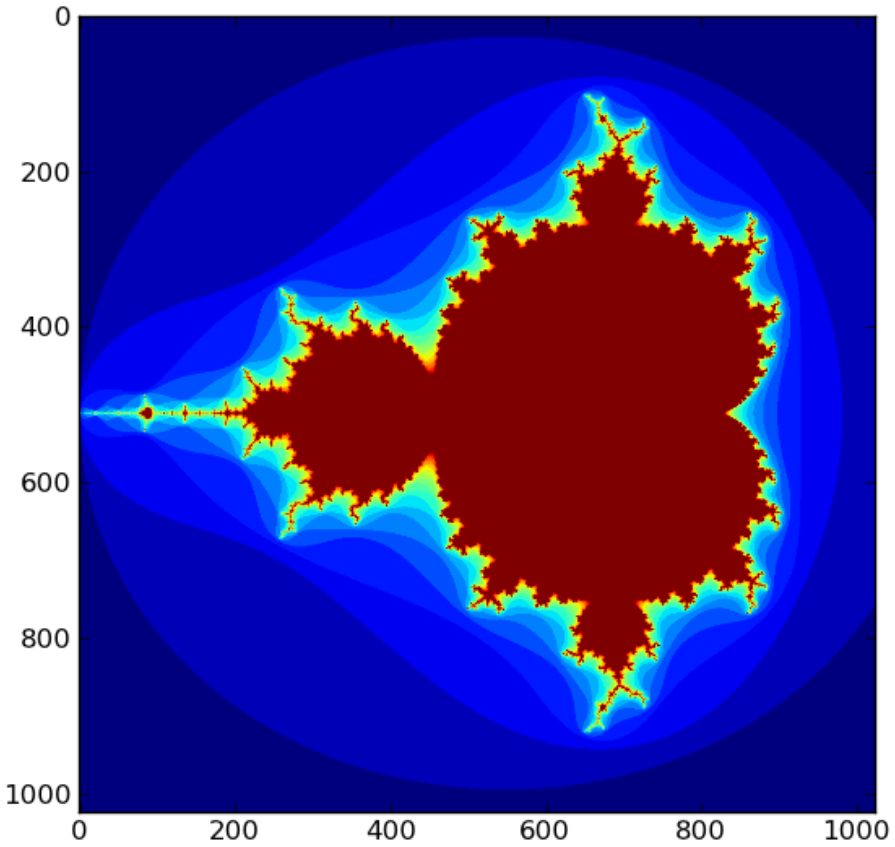
```
import numpy as np

cimport cython
cimport numpy as np
@cython.boundscheck(False)
def mandel_cy(int h, int w, int maxit=20):
    """Mandelbrot set ..."""
    cdef np.ndarray[np.complex128_t, ndim=2] c
    cdef np.ndarray[np.int64_t, ndim=2] output
    cdef int i, j, k
    cdef double complex c0, z
```

Cython example II

```
x, y = np.ogrid[-2:0.8:w*1j, -1.4:1.4:h*1j]
c = x+y*1j
output = np.zeros((w, h), dtype=int) + maxit
for i in range(h):
    for j in range(w):
        z = c[i,j]
        c0 = c[i,j]
        for k in xrange(maxit):
            z = z**2 + c0
            if (z*z.conjugate()).real > 4.0:
                output[i, j] = k
                break
return output.T
```

The result

```
In []: from mandel_cy import mandel_cy
In []: o = mandel_cy(1024,1024)
In []: imshow(o)
```



Timing

```
In []: %timeit mandel_py(256, 256)
1 loops, best of 3: 2.96 s per loop

In []: %timeit mandel_np(256,256)
10 loops, best of 3: 42.5 ms per loop

In []: %timeit mandel_cy(256,256)
100 loops, best of 3: 3.95 ms per loop
```

A whopping 700x speedup!

Additional points

- `.pxd` files
- Compiler directives:
  - `nonecheck`
  - `boundscheck`

35 m