Speeding up scientific Python code using Cython

Advanced Python Summer School, Kiel



Stéfan van der Walt

September 2012

Example Code

https://python.g-node.org/wiki/cython

Kiel2012 2 / 38

• Example Code

Introduction

- Motivation
- Motivation (continued)
- Use Cases
- Tutorial Overview

From Python to Cython

Handling NumPy Arrays

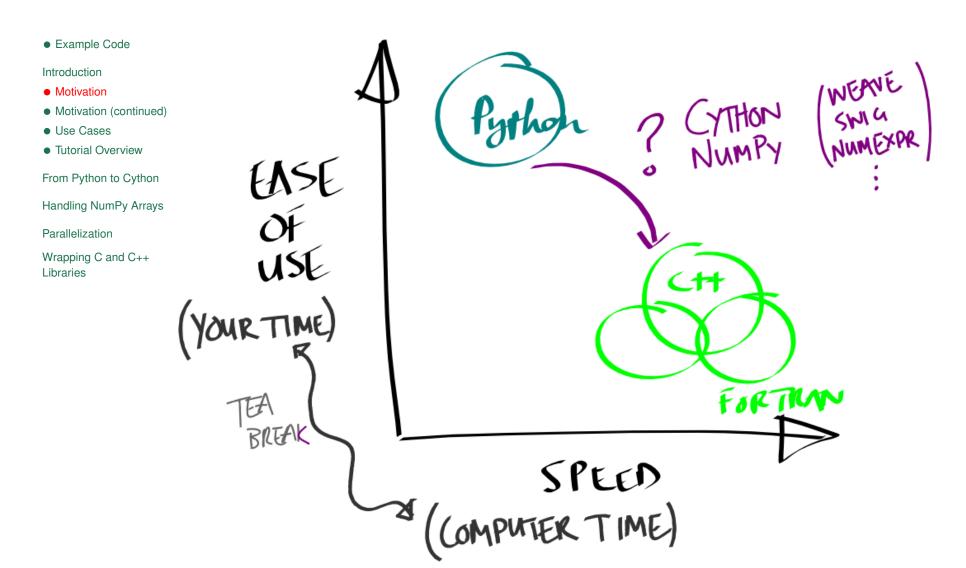
Parallelization

Wrapping C and C++
Libraries

Introduction

Kiel2012 3 / 38

Motivation



Kiel2012 4 / 38

Motivation (continued)

Example Code

Introduction

- Motivation
- Motivation (continued)
- Use Cases
- Tutorial Overview

From Python to Cython

Handling NumPy Arrays

Parallelization

Wrapping C and C++ Libraries

- Cython allows us to cross the gap
- This is good news because
 - we get to keep coding in Python (or, at least, a superset)
 - but with the speed advantage of C
- You can't have your cake and eat it. Or can you?
- Conditions / loops approx. 2–8x speed increase, 30% overall; with annotations: hundreds of times faster

Kiel2012 5 / 38

Use Cases

• Example Code

Introduction

- Motivation
- Motivation (continued)
- Use Cases
- Tutorial Overview

From Python to Cython

Handling NumPy Arrays

Parallelization

Wrapping C and C++ Libraries

- Optimize execution of Python code (profile, if possible! demo)
- Wrap existing C and C++ code
- Breaking out of the Global Interpreter Lock; openmp
- Mixing C and Python, but without the pain of the Python C API

Kiel2012 6 / 38

Tutorial Overview

Example Code

Introduction

- Motivation
- Motivation (continued)
- Use Cases
- Tutorial Overview

From Python to Cython

Handling NumPy Arrays

Parallelization

Wrapping C and C++ Libraries For this quick introduction, we'll take the following approach:

- 1. Take a piece of pure Python code and benchmark (we'll find that it is too slow)
- 2. Run the code through Cython, compile and benchmark (we'll find that it is somewhat faster)
- 3. Annotate the types and benchmark (we'll find that it is much faster)

Then we'll look at how Cython allows us to

- Work with NumPy arrays
- Use multiple threads from Python
- Wrap native C libraries

Kiel2012 7 / 38

• Example Code

Introduction

From Python to Cython

- Benchmark Python code
- More Segments
- Benchmark Python Code
- Compile the code with Cython
- Compile generated code
- Benchmark the new code
- Providing type information
- Benchmark
- Expense of Python

Function Calls

- The Last Bottlenecks
- .
- Integrating Arbitrary Functions (callbacks)

Handling NumPy Arrays

Parallelization

Wrapping C and C++ Libraries

From Python to Cython

Kiel2012 8 / 38

Benchmark Python code

• Example Code

Introduction

From Python to Cython

- Benchmark Python code
- More Segments
- Benchmark Python Code
- Compile the code with Cython
- Compile generated code
- Benchmark the new code
- Providing type information
- Benchmark
- Expense of Python

Function Calls

• The Last Bottlenecks

_

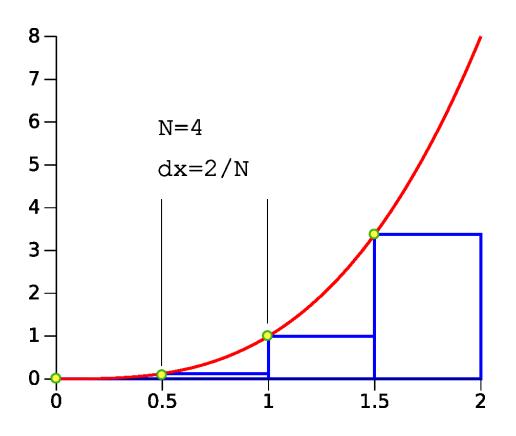
Integrating Arbitrary
 Functions (callbacks)

Handling NumPy Arrays

Parallelization

Wrapping C and C++ Libraries

Our code aims to compute (an approximation of) $\int_a^b f(x) dx$



More Segments

Example Code

Introduction

From Python to Cython

- Benchmark Python code
- More Segments
- Benchmark Python Code
- Compile the code with Cython
- Compile generated code
- Benchmark the new code
- Providing type information
- Benchmark
- Expense of Python

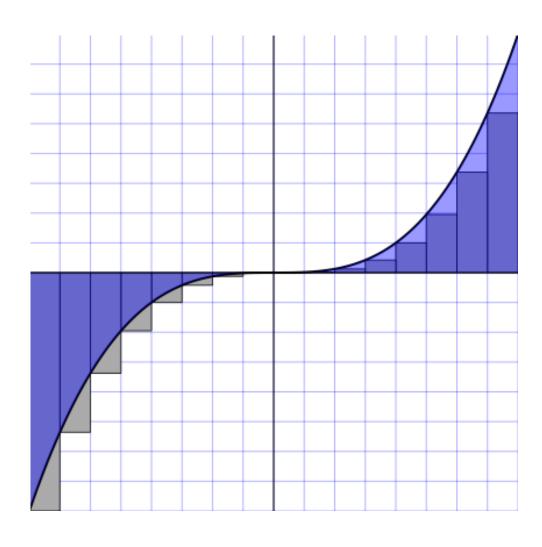
Function Calls

- The Last Bottlenecks
- Integrating Arbitrary Functions (callbacks)

Handling NumPy Arrays

Parallelization

Wrapping C and C++ Libraries



Kiel2012 10 / 38

Benchmark Python Code

```
from __future__ import division
• Example Code
Introduction
                       def f(x):
From Python to Cython
                              return x**4 - 3 * x

    Benchmark Python code

    More Segments

                       def integrate_f(a, b, N):

    Benchmark Python Code

• Compile the code with
                              """Rectangle integration of a function.
Cython
• Compile generated code
                              Parameters

    Benchmark the new code

    Providing type information

Benchmark
                              a, b: int

    Expense of Python

Function Calls
                                    Interval over which to integrate.
• The Last Bottlenecks
                              N : int
                                    Number of intervals to use in the discretisation.

    Integrating Arbitrary

Functions (callbacks)
                              11 11 11
Handling NumPy Arrays
                              s = 0
Parallelization
                              dx = (b - a) / N
Wrapping C and C++
                              for i in range(N):
Libraries
                                    s += f(a + i * dx)
                              return s * dx
```

Kiel2012 11 / 38

Compile the code with Cython

Example Code

Introduction

From Python to Cython

- Benchmark Python code
- More Segments
- Benchmark Python Code
- Compile the code with Cython
- Compile generated code
- Benchmark the new code
- Providing type information
- Benchmark
- Expense of Python

Function Calls

- The Last Bottlenecks
- Integrating Arbitrary Functions (callbacks)

Handling NumPy Arrays

Parallelization

Wrapping C and C++ Libraries

- cython filename.[py|pyx]
- What is happening behind the scenes? cython -a filename. [py | pyx]
 - Cython translates Python to C, using the Python C API (let's have a look)
- This code has some serious bottlenecks.

Kiel2012 12 / 38

Compile generated code

• Example Code

Introduction

From Python to Cython

- Benchmark Python code
- More Segments
- Benchmark Python Code
- Compile the code with Cython
- Compile generated code
- Benchmark the new code
- Providing type information
- Benchmark
- Expense of Python
 Function Calls
- The Last Bottlenecks
- Integrating Arbitrary Functions (callbacks)

Handling NumPy Arrays

Parallelization

Wrapping C and C++ Libraries

```
$ gcc -02 -fPIC -I/usr/include/python2.7
  -c integrate.c -o integrate_compiled.o

Easier yet, construct setup.py:

from distutils.core import setup
from distutils.extension import Extension
from Cython.Distutils import build_ext

setup(
  cmdclass = {'build_ext': build_ext},
  ext_modules = [
    Extension("integrate", ["integrate.pyx"]),
  ])
```

Run using python setup.py build_ext -i. This means: build extensions «in-place».

Kiel2012 13 / 38

Benchmark the new code

• Example Code

Introduction

From Python to Cython

- Benchmark Python code
- More Segments
- Benchmark Python Code
- Compile the code with Cython
- Compile generated code
- Benchmark the new code
- Providing type information
- Benchmark
- Expense of Python
 Function Calls
- The Last Bottlenecks
- Integrating Arbitrary
 Functions (callbacks)

Handling NumPy Arrays

Parallelization

Wrapping C and C++ Libraries

- Use IPython's %timeit (could do this manually using from timeit import timeit; timeit(...))
- Slight speed increase ($\approx 1.4 \times$) probably not worth it.
- Can we help Cython to do even better?
 - Yes—by giving it some clues.
 - Cython has a basic type inferencing engine, but it is very conservative for safety reasons.
 - Why does type information allow such vast speed increases?

Kiel2012 14 / 38

Providing type information

Example CodeIntroduction

From Python to Cython

- Benchmark Python code
- More Segments
- Benchmark Python Code
- Compile the code with Cython
- Compile generated code
- Benchmark the new code
- Providing type information
- Benchmark
- Expense of Python

Function Calls

- The Last Bottlenecks
- Integrating Arbitrary
 Functions (callbacks)

•

Handling NumPy Arrays

Parallelization

Wrapping C and C++
Libraries

```
from __future__ import division
def f(double x):
   return x**4 - 3 * x
def integrate_f( double a, double b, int N ):
    """Rectangle integration of a function.
    . . .
    11 11 11
     cdef:
         double s = 0
         double dx = (b - a) / N
         size_t i
    for i in range(N):
         s += f(a + i * dx)
    return s * dx
```

Kiel2012 15 / 38

Benchmark...

Kiel2012 16 / 38

Expense of Python Function Calls

• Example Code

Introduction

From Python to Cython

- Benchmark Python code
- More Segments
- Benchmark Python Code
- Compile the code with Cython
- Compile generated code
- Benchmark the new code
- Providing type information
- Benchmark
- Expense of Python

Function Calls

- The Last Bottlenecks
- Integrating Arbitrary
 Functions (callbacks)
- •

Handling NumPy Arrays

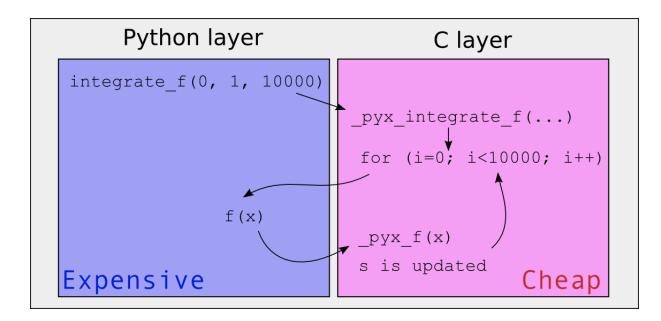
Parallelization

Wrapping C and C++ Libraries

```
def f(double x):
    return x**4 - 3 * x

def integrate_f(double a, double b, int N):
    cdef:
        double s = 0
        double dx = (b - a) / N
        size_t i

    for i in range(N):
        s += f(a + i * dx)
    return s * dx
```



The Last Bottlenecks

• Example Code

Introduction

From Python to Cython

- Benchmark Python code
- More Segments
- Benchmark Python Code
- Compile the code with Cython
- Compile generated code
- Benchmark the new code
- Providing type information
- Benchmark
- Expense of Python

Function Calls

- The Last Bottlenecks
- Integrating Arbitrary
 Functions (callbacks)

Handling NumPy Arrays

Parallelization

Wrapping C and C++ Libraries

```
# cython: cdivision=True
cdef double f(double x):
   integrate_f(double a, double b, int N):
   cdef:
       double s = 0
       double dx = (b - a) / N
       size_t i
   for i in range(N):
       s += f(a + i * dx)
   return s * dx
```

Kiel2012 18 / 38

Benchmark!

Kiel2012 19 / 38

Integrating Arbitrary Functions (callbacks)

• Example Code # cython: cdivision=True Introduction From Python to Cython Benchmark Python code cdef class Integrand: More Segments cdef double f(self, double x): Benchmark Python Code • Compile the code with raise NotImplementedError() Cython Compile generated code Benchmark the new code cdef class MyFunc(Integrand): Providing type information Benchmark cdef double f(self, double x): Expense of Python return x*x*x*x - 3 * x**Function Calls** The Last Bottlenecks Integrating Arbitrary def integrate_f(Integrand integrand, Functions (callbacks) double a, double b, int N): Handling NumPy Arrays cdef double s = 0cdef double dx = (b - a) / NParallelization Wrapping C and C++ cdef ssize_t i Libraries for i in range(N): s += integrand.f(a + i * dx)

return s * dx

Kiel2012 20 / 38

Exploring Cython Further

Kiel2012 21 / 38

Example Code

Introduction

From Python to Cython

Handling NumPy Arrays

- Declaring the MemoryView type
- Declaring the Numpy Array type
- Matrix Multiplication
- Our Own MatMul

Parallelization

Wrapping C and C++ Libraries

Handling NumPy Arrays

Kiel2012 22 / 38

Declaring the MemoryView type

• Example Code

Introduction

From Python to Cython

Handling NumPy Arrays

Declaring the MemoryView type

- Declaring the Numpy Array type
- Matrix Multiplication
- Our Own MatMul

Parallelization

Wrapping C and C++ Libraries

```
import numpy as np

def foo( double[:, ::1] arr ):
    cdef double[:, ::1] = np.zeros_like(arr)
    cdef size_t i, j
    for i in range( arr.shape[0] ):
        for j in range(arr.shape[1]):
            out[i, j] = arr[i, j] * i + j

    return np.asarray(out)
```

Kiel2012 23 / 38

Declaring the Numpy Array type

Example Code

Introduction

From Python to Cython

Handling NumPy Arrays

- Declaring the MemoryView type
- Declaring the Numpy Array type
- Matrix Multiplication
- Our Own MatMul

Parallelization

Wrapping C and C++ Libraries

An alternative to the MemoryView syntax that corresponds more closely with ndarray dtypes:

```
cimport numpy as cnp
import numpy as np

def foo( cnp.ndarray[cnp.float64, ndim=2] arr ):
    cdef cnp.ndarray[cnp.float64, ndim=2] out = np.zer
    cdef size_t i, j
    for i in range(arr.shape[0]):
        for j in range(arr.shape[1]):
        arr[i, j] = i + j
```

Different types are defined in Cython/Includes/numpy.pxd.

Kiel2012 24 / 38

Matrix Multiplication

Example Code

Introduction

From Python to Cython

Handling NumPy Arrays

- Declaring the MemoryView type
- Declaring the Numpy Array type
- Matrix Multiplication
- Our Own MatMul

Parallelization

Wrapping C and C++ Libraries

```
rows_A, cols_A = A.shape[0], A.shape[1]
rows_B, cols_B = B.shape[0], B.shape[1]
out = np.zeros(rows_A, cols_B)
# Take each row in A
for i in range(rows_A):
      And multiply by each column in B
    for j in range(cols_B):
         for k in \
              range(cols_A):
                                                     b<sub>1,2</sub>
              s = s + A[i, k] *
                                                    b<sub>2,2</sub>
                       B[k, j]
         out[i, j] = s
                                      a_{1,1} | a_{1,2}
```

Our Own MatMul

We won't even try this in pure Python (way too slow).

• Example Code

Introduction

From Python to Cython

Handling NumPy Arrays

- Declaring the
- MemoryView typeDeclaring the Numpy
- Matrix Multiplication
- Our Own MatMul

Parallelization

Array type

Wrapping C and C++ Libraries

```
def dot(double[:, ::1] A,
         double[:, ::1] B,
         double[:, ::1] out ):
    cdef:
         size_t rows_A, cols_A, rows_B, cols_B
         size_t i, j, k
         double s
    rows_A, cols_A = A.shape[0], A.shape[1]
    rows_B, cols_B = B.shape[0], B.shape[1]
    # Take each row in A
    for i in range(rows_A):
          And multiply by every column in B
        for j in range(cols_B):
             s = 0
             for k in range(cols_A):
                 s = s + A[i, k] * B[k, j]
             out[i, j] = s
```

• Example Code

Introduction

From Python to Cython

Handling NumPy Arrays

Parallelization

Parallel Loops with «prange»

•

Wrapping C and C++
Libraries

Parallelization

Kiel2012 27 / 38

Parallel Loops with «prange»

```
@cython.boundscheck(False)
• Example Code
                 @cython.wraparound(False)
Introduction
                 def pdot(double[:, ::1] A,
From Python to Cython
                            double[:, ::1] B.
Handling NumPy Arrays
Parallelization
                            double[:, ::1] out):

    Parallel Loops with

«prange»
                      cdef:
                           size_t rows_A, cols_A, rows_B, cols_B
Wrapping C and C++
                           size_t i, j, k
Libraries
                           double s
                      rows_A, cols_A = A.shape[0], A.shape[1]
                      rows_B, cols_B = B.shape[0], B.shape[1]
                      with nogil:
                           # Take each row in A
                           for i in prange (rows_A):
                                # And multiply by every column in B
                                for j in range(cols_B):
                                     s = 0
                                     for k in range(cols_A):
                                          s = s + A[i, k] * B[k, i]
```

Benchmark!

Kiel2012 29 / 38

Example Code

Introduction

From Python to Cython

Handling NumPy Arrays

Parallelization

Wrapping C and C++ Libraries

- Fortran
- External Definitions
- Build: Link Math Library
- C++ Class Wrapper
- C++ Class Wrapper
- C++ Class Wrapper
- C++ Class Wrapper
- In conclusion...

Wrapping C and C++ Libraries

Kiel2012 30 / 38

Fortran

• Example Code

Introduction

From Python to Cython

Handling NumPy Arrays

Parallelization

Wrapping C and C++ Libraries

- Fortran
- External Definitions
- Build: Link Math Library
- C++ Class Wrapper
- C++ Class Wrapper
- C++ Class Wrapper
- C++ Class Wrapper
- In conclusion...

We won't be talking about that here, but Ondrej Certik has some excellent notes:

http://fortran90.org/src/best-practices.html#interfacing-with-python

Kiel2012 31 / 38

External Definitions

In conclusion...

 Example Code Create a file, trig.pyx, with the following content: Introduction cdef extern from "math.h": From Python to Cython double cos(double x) Handling NumPy Arrays double sin(double x) Parallelization Wrapping C and C++ double tan(double x) Libraries Fortran External Definitions double M PI Build: Link Math Library • C++ Class Wrapper • C++ Class Wrapper def test_trig(): • C++ Class Wrapper print 'Some trig functions from C:', \ • C++ Class Wrapper

Kiel2012 32 / 38

cos(0), $cos(M_PI)$

Build: Link Math Library

• Example Code

Introduction

From Python to Cython

Handling NumPy Arrays

Parallelization

Wrapping C and C++ Libraries

- Fortran
- External Definitions
- Build: Link Math Library
- C++ Class Wrapper
- C++ Class Wrapper
- C++ Class Wrapper
- C++ Class Wrapper
- In conclusion...

Kiel2012 33 / 38

• Example Code

Introduction

From Python to Cython

Handling NumPy Arrays

Parallelization

Wrapping C and C++ Libraries

- Fortran
- External Definitions
- Build: Link Math Library
- C++ Class Wrapper
- C++ Class Wrapper
- C++ Class Wrapper
- C++ Class Wrapper
- In conclusion...

```
namespace geom {
    class Circle {
    public:
        Circle(double x, double y, double r);
        ~Circle();
        double getX();
        double getY();
        double getRadius();
        double getArea();
        void setCenter(double x, double y);
        void setRadius(double r);
    private:
        double x:
        double y;
        double r;
    };
```

Kiel2012 34 / 38

• Example Code

Introduction

From Python to Cython

Handling NumPy Arrays

Parallelization

Wrapping C and C++ Libraries

- Fortran
- External Definitions
- Build: Link Math Library
- C++ Class Wrapper
- C++ Class Wrapper
- C++ Class Wrapper
- C++ Class Wrapper
- In conclusion...

```
cdef extern from "Circle.h" namespace "geom":
    cdef cppclass Circle:
        Circle(double, double, double)
        double getX()
        double getY()
        double getRadius()
        double getArea()
        void setCenter(double, double)
        void setRadius(double)
```

Kiel2012 35 / 38

```
cdef class PyCircle:
                          cdef Circle *thisptr

    Example Code

Introduction
From Python to Cython
                          def __cinit__(self, double x, double y, double r):
Handling NumPy Arrays
                                self.thisptr = new Circle(x, y, r)
Parallelization
Wrapping C and C++
Libraries
                          def __dealloc__(self):
Fortran

    External Definitions

                                del self.thisptr

    Build: Link Math Library

    C++ Class Wrapper

    C++ Class Wrapper

                          @property

    C++ Class Wrapper

                          def area(self):
• C++ Class Wrapper
In conclusion...
                                return self.thisptr.getArea()
                          @property
                          def radius(self):
                                return self.thisptr.getRadius()
                          def set_radius(self, r):
                                self.thisptr.setRadius(r)
                          @property
                          def center(self):
```

Kiel2012

Example Code

Introduction

From Python to Cython

Handling NumPy Arrays

Parallelization

Wrapping C and C++ Libraries

- Fortran
- External Definitions
- Build: Link Math Library
- C++ Class Wrapper
- C++ Class Wrapper
- C++ Class Wrapper
- C++ Class Wrapper
- In conclusion...

Kiel2012 37 / 38

In conclusion...

- Build functional and tested code
- Profile
- Re-implement bottlenecks (behavior verified by tests)
- Et voilà—high-level code, low-level performance. [It's no silver bullet, but it's still pretty good.]



Kiel2012 38 / 38