All About Cython

http://www.cython.org

Robert Bradshaw and Craig Citro
UW Math

May 16, 2009

Outline

1. Introduction

2. Cython: The Project

3. Cython and Sage

Outline

1 Introduction

2 Cython: The Project

3 Cython and Sage

What's Cython?

Cython is a language extremely close to Python that allows you to:

- write **extremely** fast code,
- stay happily oblivious to the Python/C API,
- easily mix Python and C types, and
- use C/C++ libraries from Python with a minimal amount of pain and heartache.

Examples

```
sage: def mysum(N):
 ...:     s = 0
 ...:     for k in range(N):
 ...:         s += k
 ...:     return s

sage: time mysum(10**6)
499999500000L
Time: CPU 0.25 s, Wall: 0.25 s

sage:: def mysum2(N):
 ...:     return sum(range(N))

sage: time mysum2(10**6)
499999500000L
Time: CPU 0.19 s, Wall: 0.19 s
```

Examples

```
def mysum_c(N):
    cdef int k
    cdef long long s = 0

    for k in range(N):
        s += k
    return s
```

So we compile this bit of Cython code, and we have:

```
sage: %cython
 ...: def mysum_c(n):
 ...:     cdef int k
 ...:     cdef long long s
 ...:     s = 0
 ...:     for k in range(n):
 ...:         s += k
 ...:     return s

sage: time mysum_c(10**6)
499999500000L
Time: CPU 0.00 s, Wall: 0.00 s
```

Examples

Yeah, this one is just a **wee** bit faster:

```
sage: timeit('mysum(10**6)')
5 loops, best of 3: 255 ms per loop

sage: timeit('mysum_c(10**6)')
625 loops, best of 3: 1.23 ms per loop

sage: 255/1.23
207.317073170732
```

Examples

Of course, there are limitations:

```
sage: mysum_c(10**10)
Traceback (most recent call last):
...
OverflowError: long int too large to convert to int
```

Cython

Cython (`http://www.cython.org`) lets you:

- declare attributes for your classes with C datatypes
- declare methods to take and return C datatypes
- interface with your existing C/C++ libraries

Cython

No one wants to declare types for all of their objects, and manually allocate and deallocate our C objects – this is one of the reasons we aren't using C in the first place!

We don't have to. The Cython development model:

- Write code in Python.
- Get it working **correctly**.
- Profile the code.
- Move the inner loops to Cython.

Cython: It Works

Jason Grout:

> I spent two or three days working on this. Here is the end result: 0.24
> seconds compared to 150 seconds. Such is the power of Cython :). That's
> a speedup of a factor of 150.64/0.24=627!

This particular function, because it is so fast now, has become a regular tool in our research and has led to discovering at least one counter-example to a conjecture that was open for several months.

One `def` to rule them all . . .

There are three ways to declare a function in Cython:

- `def`: The usual Python declaration; uses Python calling conventions, and takes Python types
- `cdef`: A C declaration; uses C calling conventions, takes Python or C types
- `cpdef`: The best of both worlds

Different `defs` for different folks . . .

Let's see an example:

```
def extend_py(self, d):
    self._length += d

cdef extend_c(self, int d):
    self._length += d

cpdef extend(self, int d):
    self._length += d
```

Different defs for different folks . . .

```
In [3]: %time b.time_test(1, 10**7, 'def')
CPU times: user 1.55 s, sys: 0.00 s, total: 1.56 s
Wall time: 1.57 s

In [5]: %time b.time_test(1, 10**7, 'cdef')
CPU times: user 0.07 s, sys: 0.00 s, total: 0.07 s
Wall time: 0.07 s

In [7]: %time b.time_test(1, 10**7, 'cpdef')
CPU times: user 0.09 s, sys: 0.00 s, total: 0.09 s
Wall time: 0.09 s
```

Different defs for different folks . . .

```
In [4]: %time for _ in range(10**7): b.extend_py(1)
CPU times: user 2.74 s, sys: 0.15 s, total: 2.89 s
Wall time: 2.93 s

In [6]: %time for _ in range(10**7): b.extend(1)
CPU times: user 2.85 s, sys: 0.04 s, total: 2.89 s
Wall time: 2.92 s
```

Outline

1. Introduction
2. Cython: The Project
3. Cython and Sage

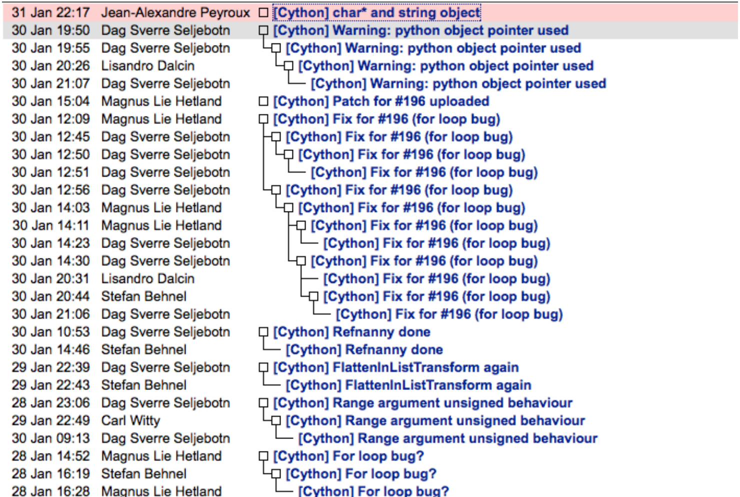
Cython

Cython is open source, freely available under the Apache License.

Web page: http://www.cython.org
Mercurial: http://hg.cython.org
Wiki: http://wiki.cython.org
Bugtracker: http://trac.cython.org/
Mailing list: cython-dev@codespeak.net

There are more than twelve Cython developers . . .

- Lead developers: Stephan Behnel, Robert Bradshaw
- Dag Sverre Seljebotn (Google Summer of Code 2008): Tight integration of Cython types and buffer types (see PEP 3118), used by Numpy and PIL
- Large, active development community:



31 Jan 22:17	Jean-Alexandre Peyroux	□ [Cython] char* and string object
30 Jan 19:50	Dag Sverre Seljebotn	□ [Cython] Warning: python object pointer used
30 Jan 19:55	Dag Sverre Seljebotn	□ [Cython] Warning: python object pointer used
30 Jan 20:26	Lisandro Dalcin	□ [Cython] Warning: python object pointer used
30 Jan 21:07	Dag Sverre Seljebotn	[Cython] Warning: python object pointer used
30 Jan 15:04	Magnus Lie Hetland	□ [Cython] Patch for #196 uploaded
30 Jan 12:09	Magnus Lie Hetland	□ [Cython] Fix for #196 (for loop bug)
30 Jan 12:45	Dag Sverre Seljebotn	□ [Cython] Fix for #196 (for loop bug)
30 Jan 12:50	Dag Sverre Seljebotn	□ [Cython] Fix for #196 (for loop bug)
30 Jan 12:51	Dag Sverre Seljebotn	[Cython] Fix for #196 (for loop bug)
30 Jan 12:56	Dag Sverre Seljebotn	□ [Cython] Fix for #196 (for loop bug)
30 Jan 14:03	Magnus Lie Hetland	□ [Cython] Fix for #196 (for loop bug)
30 Jan 14:11	Magnus Lie Hetland	□ [Cython] Fix for #196 (for loop bug)
30 Jan 14:23	Dag Sverre Seljebotn	[Cython] Fix for #196 (for loop bug)
30 Jan 14:30	Dag Sverre Seljebotn	□ [Cython] Fix for #196 (for loop bug)
30 Jan 20:31	Lisandro Dalcin	[Cython] Fix for #196 (for loop bug)
30 Jan 20:44	Stefan Behnel	□ [Cython] Fix for #196 (for loop bug)
30 Jan 21:06	Dag Sverre Seljebotn	[Cython] Fix for #196 (for loop bug)
30 Jan 10:53	Dag Sverre Seljebotn	□ [Cython] Refnanny done
30 Jan 14:46	Stefan Behnel	[Cython] Refnanny done
29 Jan 22:39	Dag Sverre Seljebotn	□ [Cython] FlattenInListTransform again
29 Jan 22:43	Stefan Behnel	[Cython] FlattenInListTransform again
28 Jan 23:06	Dag Sverre Seljebotn	□ [Cython] Range argument unsigned behaviour
29 Jan 22:49	Carl Witty	□ [Cython] Range argument unsigned behaviour
30 Jan 09:13	Dag Sverre Seljebotn	[Cython] Range argument unsigned behaviour
28 Jan 14:52	Magnus Lie Hetland	□ [Cython] For loop bug?
28 Jan 16:19	Stefan Behnel	□ [Cython] For loop bug?
28 Jan 16:28	Magnus Lie Hetland	[Cython] For loop bug?

Cython under many names

A quick history:

- Cython is a fork of the Pyrex project, started by Greg Ewing (first released in 2002)
- Began life as part of the Sage project (and originally called "SageX") in 2006, work mostly by William Stein, Martin Albrecht, and Robert Bradshaw
- Lots of outside interest, particularly from Stefan Behnel (who was maintaining another Pyrex fork, `lxml`)
- Cython first launched in 2007

Does it cook breakfast, too?

So there are still a few things not supported in Cython. Most of these are simply just a lack of developer time so far:

- ~~Closures~~
- ~~Closures~~
- ~~Clos~~ures
- Generators
- Multiple Inheritance (no plan right now . . .)
- Other various bits: `http://wiki.cython.org/Unsupported`

Would you like to know more?

There's a lot of interesting stuff I didn't get to talk about . . .

- Cython support for built-in types (`cdef list ls` . . .)
- Automatic conversion between most Python and C/C++ types (whenever it would make sense)
- Exposing Cython classes (`.pxd` files for declarations, . . .)
- Cython can also be used to interface with C++ libraries (only a small amount of black magic needed!)

Robert will talk more about these in a few minutes . . .

Outline

1 Introduction

2 Cython: The Project

3 Cython and Sage

Cython as far as the eye can see . . .

Roughly 20% of the source files in Sage are written in Cython (which accounts for about 30% of the code itself). We use Cython for several things:

- Speeding up key algorithms,
- interfacing with C/C++ libraries, and
- avoiding the Python/C API (read: saving our sanity).

Making Sage source faster . . .

To really understand what's taking time in Cython source, you often need to do serious profiling or read the generated C source code. However, it's easy to get your hands on the annotated HTML file for any file in the Sage source tree. You can simply do `sage -cython -a` on any file, and the annotated source will appear right there:

```
[craigcitro@sharma ~/three-four-two/devel/sage-main/sage/rings/pol
$ l *dense_flint*
 704 polynomial_integer_dense_flint.cpp
   4 polynomial_integer_dense_flint.pxd
  40 polynomial_integer_dense_flint.pyx
[craigcitro@sharma ~/three-four-two/devel/sage-main/sage/rings/pol
$ sa -cython -a polynomial_integer_dense_flint.pyx
[craigcitro@sharma ~/three-four-two/devel/sage-main/sage/rings/pol
$ l *dense_flint*
 700 polynomial_integer_dense_flint.c
 704 polynomial_integer_dense_flint.cpp
 684 polynomial_integer_dense_flint.html
   4 polynomial_integer_dense_flint.pxd
  40 polynomial_integer_dense_flint.pyx
```

Using .pxi and .pxd files

For most uses `.pxd` files are in, `.pxi` files are out.

Use a `.pxd` file if you want to

- Declare external functions from another library
- Declare inline functions
- Declare types

Use a `.pxi` file if you want to

- Include generic templating code (e.g. `polynomial_template.pxi`)
- Include a chunk of code textually
- Include a separate copy of the file in each module

Too much of Sage still uses `.pxi` files, because once upon a time, `.pxd`s didn't do the job.

NumPy and Buffers

Last summer Dag Sverre Seljebotn did an **awesome** job of providing **fast, simple** access to NumPy arrays, or anything else supporting the buffer interface.

fastnumpy.pyx

```
cimport numpy

def sum(x):
    cdef numpy.ndarray[int, ndim=1] arr = x
    cdef int i, s = 0
    for i in range(arr.shape[0]):
        s += arr[i]
    return s
```

This loop gets translated into pure C.

Improved C++

Some C++ niceties have been added:

- Exception catching
- (Non-pointer) functions in structs

fastnumpy.pyx

```
cdef extern from ”foo.cpp”:
    cdef struct Foo:
        cdef int foo() except +
        cdef int allocate() except +MemoryError
    cdef int raise_py_error()
    cdef int something_dangerous() except +raise_py_error
```

More to come...

Complex Numbers

The next release of Cython will have complex number support.

- With or without support from `complex.h`

mandelbrot.pyx

```
cdef extern from ”complex.h”:
    double cabs(double complex)

cdef bint in_mandelbrot(double complex c, int iter):
    cdef int i
    cdef double complex z = c
    for i in range(iter):
        z = z*z+c
        if cabs(z) > 2:
            return False
    return True

...
```

Embedding

An `--embed` option to create a `main()` method that embeds the interpreter. One then compiles to create an executable.

standalone.pyx

```
if __name__ == "__main__":
    print "Running just like a .py file would."

print "Stuff here runs to."
```

Of course, you still have to link against Python.

Closures

We finally (almost) support closures.

- The last major roadblock before 100% Python support
- Generators, lambda, etc. are just essentially closures

closure.pyx

```
%cython
def remember(x):
    def f():
        return x
    return f

sage: f = remember(3)
sage: f()
3
```

Needs more testing!

Improvements

Many more improvements...

- Newer temp allocation scheme
- Utility code generation
- Pure Python mode
- `import *` and `cimport *`
- `isinstance(...)` checks types for Extension classes
- cdivision
- Compiler directives
- Better type conversions
- Better errors, optimizations, boostrapping...

The Cython codebase is maturing enough to work on higher level stuff.

Wrapping C++

One can wrap C++ with Cython, but it's kind of hackish:

- Declare classes as structs
- Use string substitution
- Write a wrapper file

Wrapping C++

This will all change this summer thanks to **Danilo Freitas** and **Google**.

Wrapping C++

Danilo's objective is to make Cython **C++ aware** enough to natively use **STL**.
If you can wrap STL, you can wrap just about **anything**...

- Templates
- Real C++ classes and inheritance
- Function overloading
- Operator overloading

Some of this may be also available in non-C++, non-extern code.

One of the biggest questions is how to provide **Pythonic syntax** for C++ constructs.

Proposed C++ Syntax

The code below is a proposal, suggestions welcome!

foo.pxd

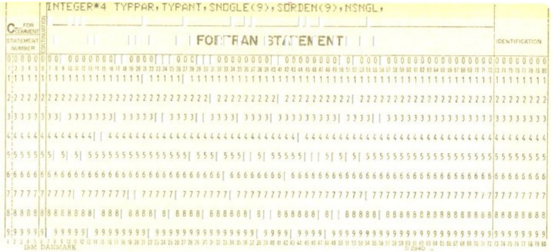
```
cdef extern from ”foo.h” namespace Foo:

    cdef cppclass MyFoo[T] (MySuperClass):
        MyFoo[T] __add__(MyFoo[T], int)
        MyFoo[T] __add__(MyFoo[T], MyFoo[T])
        T    __getitem__(MyFoo[T], int)
        void __setitem__(MyFoo[T], T, int)
```

We don’t necessarily have to construct a full model of C++, just enough to **pass it on** to the C++ compiler.

Fortran Support

There is another GSoC project by **Kurt Smith** to provide **Fortran support**.

- NumPy buffers aware
- Automatically create C bindings
- Use `f2py` to parse header files
- ...



```
INTEGER*4 TYPPAR,TYPANT,SNDGLE(9),SORDEN(9),NSNGL,
```

Future

What's in store for Cython in the long run?

- 100% python coverage and compatibility
- Type inference
- Control flow analysis
- Header file parsing (auto .pxd generation)
- Eventual inclusion into Python
- ???

Any questions?

Thanks for listening!