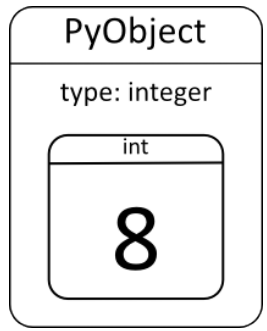
Speeding up Python

Antonio Gómez-Iglesias
agomez@tacc.utexas.edu

October 30th, 2014

Why

- Python is nice, easy, development is fast
- However, Python is slow
- The bottlenecks can be rewritten:
  - SWIG
  - Boost.Python
  - numba
  - Cython



Cython

What's Cython?

- Python with C data types
- Any* Python code is valid Cython code
- Translate the code into C/C++ code. Use it as modules
- You can call C libraries
- Code using Python values and C values can be intermixed (automatic conversions)
- The more type information you provide the better the compile

First Example

Use iPython

```
In [1]: %load_ext cythonmagic
In [2]: %%cython
        import math
        def first_cython(int arg):
            return math.sqrt(arg**9/13 + 7*arg**3 + 29)**3
In [3]: first_cython(100)
```

How Much Faster

Use iPython

```
In [1]: import math
In [2]: def first_python(arg):
            return math.sqrt(arg**9/13 + 7*arg**3 + 29)**3
In [3]: %timeit first_python(20)

In [4]: %load_ext cythonmagic

In [5]: %%cython
        import math
        def first_cython(arg):
            return math.sqrt(arg**9/13 + 7*arg**3 + 29)**3
In [6]: %timeit first_cython(20)

In [7]: %%cython
        import math
        def fast_cython(int arg):
            return math.sqrt(arg**9/13 + 7*arg**3 + 29)**3
In [8]: %timeit fast_cython(20)
```

Cython Functions

- Python functions are defined with *def*. They take Python objects as parameters and return Python objects
- C functions are defined with *cdef*. They take either Python objects or C values and return Python objects or C values
- Both can call each other within a Cython module
- Only Python functions can be called from outside the model by Python code

Type Declaration

- cdef: static typization

```
cdef double var
cdef int arr[50]
```

- cdef: as C function:

```
cdef double function(double arg):
    return arg**2
```

- cdef class:

```
cdef class MyClass:
```

- cdef struct:

```
cdef struct my_struct:
    int var1
    double var2
```

- Several declarations into the same cdef

```
cdef:
    int i
    double d
    void f (arg):
        return arg**2
```

Example

examples/2_cython/test_python.py

```
1 import math
2 def function(arg):
3     res = 0.0
4     for i in range(50000000):
5         res+=math.sqrt((i+1)*arg**5)
6     return res
7
8 print function(10.0)
```

```
> time python test_python.py
```

Example

examples/2_cython/myfunc1.pyx

```
1 import math
2
3 def function(double arg):
4     cdef double res = 0.0
5     for i in range(50000000):
6         res+=math.sqrt((i+1)*arg**5)
7     return res
```

examples/2_cython/test_cython1.py

```
1 import myfunc1
2 print myfunc1.function(10.0)
```

```
> cython myfunc1.pyx
> icc -shared -fPIC -O3
      myfunc1.c -o myfunc1.so
      -I$TACC_PYTHON_INC/python2.7/
> time python test_cython1.py
```

Example

examples/2_cython/myfunc2.pyx

```
1  import math
2
3  cdef double f(double arg):
4    cdef double res = 0.0
5    cdef int i = 0
6    for i in range(50000000):
7      res+=math.sqrt((i+1)*arg**5)
8    return res
9
10 def function(double arg):
11   return f(arg)
```

examples/2_cython/test_cython2.py

```
1  import myfunc2
2  print myfunc2.function(10.0)
```

```
> cython myfunc2.pyx
> icc -shared -fPIC -O3
      myfunc2.c -o myfunc2.so
      -I$TACC_PYTHON_INC/python2.7/
> time python test_cython2.py
```

Example

examples/2_cython/myfunc3.pyx

```
1  from libc.math cimport sqrt
2  #cdef extern from "math.h":
3  #  double sqrt(double x)
4
5  cdef double f(double arg):
6    cdef double res = 0.0
7    cdef int i = 0
8    for i in range(50000000):
9      res+=sqrt((i+1)*arg**5)
10   return res
11
12 def function(double arg):
13   return f(arg)
```

examples/2_cython/test_cython3.py

```
1  import myfunc3
2  print myfunc3.function(10.0)
```

```
> cython myfunc3.pyx
> icc -shared -fPIC -O3
      myfunc3.c -o myfunc3.so
      -I$TACC_PYTHON_INC/python2.7/
> time python test_cython3.py
```

Cython

- Easy to decorate your own code
- Lot of potential
- Iterative process
- Link to a great tutorial

License