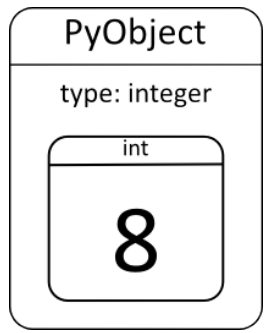
Speeding up Python

Antonio Gómez-Iglesias
agomez@tacc.utexas.edu

April 17th, 2015

Why

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



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  import time
3  def function(arg, ilist):
4    res = 0.0
5    for i in xrange(len(ilist)):
6      for j in xrange(len(ilist)):
7        if (i>0 and i<(len(ilist)-1)):
8          res+=math.sqrt((ilist[i]+ilist[i-1]
9               +ilist[i+1])*arg**5)/100.0
10       else:
11         res+=math.sqrt(ilist[i]*arg**5)/100.0
12   return res
13
14 ilist = range(5000)
15 start_time = time.time()
16 print function(10.0, ilist)
17 end_time = time.time()
18 print "Kernel function took ", \
19        end_time-start_time, " seconds"
```

```
> python test_python.py
```

Example

examples/2_cython/myfunc1.pyx

```
1  from math import sqrt
2
3  def function(double arg, ilist):
4      cdef double res = 0.0
5      for i in xrange(len(ilist)):
6        for j in xrange(len(ilist)):
7          if (i>0 and i<(len(ilist)-1)):
8            res+=sqrt((ilist[i]+ilist[i-1]
9                 +ilist[i+1])*arg**5)/100.0
10         else:
11           res+=sqrt((ilist[i])*arg**5)/100.0
12     return res
```

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

examples/2_cython/test_cython1.py

```
1  import time
2  ilist = range(10000)
3  print myfunc1.function(10.0, ilist)
```

Example

examples/2_cython/myfunc2.pyx

```
1  from math import sqrt
2
3  cdef double f(double arg, ilist):
4    cdef double res = 0.0
5    cdef long i = 0
6    cdef long j = 0
7    for i in xrange(len(ilist)):
8      for j in xrange(len(ilist)):
9          if (i>0 and i<(len(ilist)-1)):
10           res+=sqrt((ilist[i]+ilist[i-1]
11                +ilist[i+1])*arg**5)/100.0
12         else:
13           res+=sqrt((ilist[i])*arg**5)/100.0
14   return res
15
16 def function(double arg, ilist):
17   return f(arg, ilist)
```

```
> cython myfunc2.pyx
> icc -shared -fPIC -O3
      myfunc2.c -o myfunc2.so
      -I$TACC_PYTHON_INC/python2.7/
> 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, ilist):
6    cdef double res = 0.0
7    cdef long i = 0
8    cdef long j = 0
9    cdef long length = len(ilist)
10   for i in xrange(length):
11     for j in xrange(length):
12       if (i>0 and i<(len(ilist)-1)):
13         res+=sqrt((ilist[i]+ilist[i-1]+
14               ilist[i+1])*arg**5)/100.0
15       else:
16         res+=sqrt((ilist[i])*arg**5)/100.0
17   return res
18
19 def function(double arg, ilist):
20   return f(arg, ilist)
```

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

Example

examples/2_cython/myfunc4.pyx

```
1  from libc.math cimport sqrt
2  from libc.stdlib cimport malloc, free
3
4  cdef f(double arg, long* ilist, int  len):
5      cdef double res = 0.0
6      cdef int i = 0, j = 0
7      for i in xrange(len):
8        for j in xrange(len):
9          if (i>0 and i<(len-1)):
10           res+=sqrt((ilist[i]+ilist[i-1]
11                +ilist[i+1])*arg**5)/100.0
12         else:
13           res+=sqrt((ilist[i])*arg**5)/100.0
14     return res
15
16 def function(double arg, ilist):
17   nelemts = len(ilist)
18   cdef long *array=<long *> malloc(nelemts * sizeof(long))
19   for i in xrange(nelemts):
20     array[i] = ilist[i]
21   val =  f(arg, array, nelemts)
22   free(array)
23   return val
```

```
> cython myfunc4.pyx
> icc -shared -fPIC -O3
      myfunc4.c -o myfunc4.so
      -I$TACC_PYTHON_INC/python2.7/
> python test_cython4.py
```

Cython

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

Using C in your Python code

- Fine-grained control
- Great for small kernels: if you have to rewrite everything to C, why would you use Python?
- You get access to all C libraries, codes,...
- Use OpenMP, use the Intel Xeon Phi(!!),...
- Two main options:
  - You can use ctypes (not easy!)
  - cffi (easyish)

cffi

- Module that adds a C parser that understands function and structure definitions
- Interface taken from Lua
- Two models:
  - Import your shared C library
  - Write C in your Python code

Import your shared library

- Define the structure of the library you want to use (cdef)
- Import the library (dlopen)
- Cast Python objects to C datatypes (cast)
- Done

Using cffi in Stampede

- Not officially supported

```
module use /work/02658/agomez/tools/modules/
module load ffi
```

- Install cffi module:

```
export CFLAGS="-I$TACC_FFI_INC -L$TACC_FFI_LIB -lffi"
pip install cffi --user
```

Example

examples/2_cffi/c_function.c

```c
#include <math.h>
double f(double arg){
  double res = 0.0;  int i = 0;
  for (i=0; i<50000000; ++i)
    res += sqrt((i+1) * pow(arg, 5));
  return res;
}
```

```
> icc c_function.c -shared \
-fPIC -o c_function.so
```

examples/2_cffi/cffi_lib.py

```python
from cffi import FFI
ffi = FFI()
ffi.cdef (r'''
    double f (double);
    ''')
lib = ffi.dlopen ("./c_function.so")

print 'cffi says ' , lib.f (10.0)
```

Writing C

- Define the structure of the library you want to use (cdef)
- Write your code (verify)
- Cast Python objects to C datatypes (cast)
- Done

Example

examples/2_cffi/cffi_code.py

```
1  import cffi
2
3  ffi = cffi.FFI()
4  ffi.cdef(r'''
5  double f(double);
6  ''')
7
8  C_code = ffi.verify(r'''
9  #include <math.h>
10
11 double f(double arg){
12   double res = 0.0;
13   int i = 0;
14   for (i=0; i<50000000; ++i) {
15     res += sqrt((i+1)*pow(arg , 5));
16   }
17   return res;
18 }
19 ''')
20
21 print 'cffi says ', C_code.f(10.0)
```

Example

examples/2_cffi/cffi_complex.py

```
1  import cffi
2  import numpy as np
3
4  def cast_matrix(matrix, ffi):
5      ap = ffi.new("double* [%d]" % (matrix.shape[0]))
6      for i in range(matrix.shape[0]):
7          ap[i] = i
8      return ap
9
10 ffi = cffi.FFI()
11 ffi.cdef(r''' double foo(double**);''')
12 C = ffi.verify(r'''
13   #include <sched.h>
14   double foo(double** matrix){
15     int num_threads, num_cores, thread_num;
16     #pragma offload target(mic)
17     {
18       #pragma omp parallel private (thread_num)
19       {
20         thread_num = omp_get_thread_num() ; num_threads = omp_get_num_threads();
21         num_cores = sysconf( _SC_NPROCESSORS_ONLN );
22         printf ("Hi I'm thread %d out of %d running on cpu %d\n", thread_num, num_threads,
                 sched_getcpu());
23       }
24     }
25     return(0);
26   }
27 ''', extra_compile_args=['-openmp',])
28 m = np.ones ((10,10))
29 m_p = cast_matrix(m, ffi)
30 C.foo(m_p)
```

pypy

What is PyPy?

- Alternative implementation of Python
- Just-in-Time compiler
- numpy not fully supported (yet)
- You don't have to do anything to get great performance

```
module use /work/02658/agomez/tools/modules/
module load cffi
module load pypy/2.5.0
pypy examples/2_cython/test_python.py
module unuse /work/02658/agomez/tools/modules/
```

License