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
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

```python
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

```python
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)
```

A. Gómez 5 Speeding up Python
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
  1 cdef double var
  2 cdef int arr[50]

• cdef: as C function:
  1 cdef double function(double arg):
      return arg**2

• cdef class:
  1 cdef class MyClass:

• cdef struct:
  1 cdef struct my_struct:
      2 int var1
      3 double var2

• Several declarations into the same cdef
  1 cdef:
      2 int i
def double d
def void f (arg):
      return arg**2
```python
import math
import time

def function(arg, ilist):
    res = 0.0
    for i in xrange(len(ilist)):
        for j in xrange(len(ilist)):
            if (i > 0 and i < (len(ilist) - 1)):
                res += math.sqrt((ilist[i] + ilist[i - 1] + ilist[i + 1]) * arg ** 5) / 100.0
            else:
                res += math.sqrt(ilist[i] * arg ** 5) / 100.0
    return res

ilist = range(5000)
start_time = time.time()
print function(10.0, ilist)
end_time = time.time()
print "Kernel function took ",
    end_time - start_time, " seconds"
```

`> python test_python.py`
Example

def function(double arg, ilist):
    cdef double res = 0.0
    for i in xrange(len(ilist)):
        for j in xrange(len(ilist)):
            if (i>0 and i<(len(ilist)-1)):
                res += sqrt((ilist[i]+ilist[i-1]+ilist[i+1])*arg**5)/100.0
            else:
                res += sqrt((ilist[i])*arg**5)/100.0
    return res

import time
ilist = range(10000)
print myfunc1.function(10.0, ilist)
from math import sqrt

cdef double f(double arg, ilist):
    cdef double res = 0.0
    cdef long i = 0
    cdef long j = 0
    for i in xrange(len(ilist)):
        for j in xrange(len(ilist)):
            if (i > 0 and i < (len(ilist) - 1)):
                res += sqrt((ilist[i] + ilist[i - 1] + ilist[i + 1]) * arg ** 5) / 100.0
            else:
                res += sqrt((ilist[i]) * arg ** 5) / 100.0
    return res

def function(double arg, ilist):
    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

A. Gómez 10 Speeding up Python
from libc.math cimport sqrt

def function(double arg, ilist):
    return f(arg, ilist)
Example

cython/myfunc4.pyx

```python
from libc.math cimport sqrt
from libc.stdlib cimport malloc, free

cdef f(double arg, long* ilist, int len):
    cdef double res = 0.0
    cdef int i = 0, j = 0
    for i in xrange(len):
        for j in xrange(len):
            if (i>0 and i<(len-1)):
                res+=sqrt((ilist[i]+ilist[i-1]+ilist[i+1])*arg**5)/100.0
            else:
                res+=sqrt((ilist[i])*arg**5)/100.0
    return res

def function(double arg, ilist):
    nelemts = len(ilist)
    cdef long *array=<long*> malloc(nelemts * sizeof(long))
    for i in xrange(nelemts):
        array[i] = ilist[i]
    val = f(arg, array, nelemts)
    free(array)
    return val
```

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

A. Gómez 12 Speeding up Python
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

```
# 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
```

```
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
import cffi

ffi = cffi.FFI()
ffi.cdef(r'''
double f(double);
''')

C_code = ffi.verify(r'''
#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;
}
'''

print 'cffi says ', C_code.f(10.0)
import cffi
import numpy as np

def cast_matrix(matrix, ffi):
    ap = ffi.new("double* [%d]" % (matrix.shape[0]))
    for i in range(matrix.shape[0]):
        ap[i] = i
    return ap

ffi = cffi.FFI()
ffi.cdef(r''' double foo ( double ** ) ; ''')
C = ffi.verify(r''' # include <sched.h>
double foo(double** matrix){
    int num_threads, num_cores, thread_num;
    #pragma offload target(mic)
    {
        #pragma omp parallel private (thread_num)
        {
            thread_num = omp_get_thread_num () ; num_threads = omp_get_num_threads ();
            num_cores = sysconf( _SC_NPROCESSORS_ONLN );
            printf ("Hi I’m thread %d out of %d running on cpu %d\n", thread_num, num_threads, sched_getcpu ());
        }
    }
    return(0);
}
''', extra_compile_args=['-openmp',])

m = np.ones((10,10))
m_p = cast_matrix(m, ffi)
C.foo(m_p)
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/
```
© The University of Texas at Austin, 2015

This work is licensed under the Creative Commons Attribution Non-Commercial 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc/3.0/

When attributing this work, please use the following text: "HPC Python", Texas Advanced Computing Center, 2015. Available under a Creative Commons Attribution Non-Commercial 3.0 Unported License.