High Performance Python

Week 3: Cython

What is Cython

- Cython compiles Python code into a C module.
- To do this efficiently you provide Cython with type information of your variables.

Using Cython

- Step 1: Write the code in Python
- Step 2: Copy the Python code you want to another .pyx file, and modify a setup script.
- Step 3: Add scalar types
- Step 4: Add vector/numpy types

Example

```
import numpy as np
import time
from reduce import reduce as creduce

def reduce( arr ):
     output = 0
     N,M = arr.shape
     for i in range(N):
          for j in range(M):
               output += arr[i,j]**2

     return output

if __name__ == '__main__':

     #a random array of numbers from 0-1
     arr = np.random.random( [1000,1000] )

     sTime = time.time()
     s = reduce( arr )
     print 'Execution time:',time.time()-sTime

     sTime = time.time()
     s = creduce( arr )
     print 'Execution time:',time.time()-sTime
```

Phase 0

Create a new .pyx file, and copy the function into it

```
def reduce( arr ):
     output = 0
     N,M = arr.shape
     for i in range(N):
          for j in range(M):
               output += arr[i,j]**2

     return output
```

Create a setup file

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

import numpy

ext = Extension("reduce", ["reduce_p0.pyx"],
                include_dirs = [numpy.get_include()])

setup(ext_modules=[ext],
      cmdclass = {'build_ext': build_ext})
```

Compile

```
~$ python reduce_setup.py build_ext --inplace
~$ python reduce_run.py
Execution time: 1.45264196396
Execution time: 1.35613322258
```

Phase 1

Add scalar types

```
def reduce( arr ):
     cdef double output = 0
     cdef int N, M, i, j
     N,M = arr.shape
     for i in range(N):
          for j in range(M):
               output += arr[i,j]**2

     return output
```

Compile

```
~$ python reduce_setup.py build_ext --inplace
~$ python reduce_run.py
Execution time: 1.44950199127
Execution time: 2.13104391098
```

Phase 2

Add numpy types

```
import numpy as np
cimport numpy as np

def reduce(np.ndarray[np.float64_t, ndim=2]  arr ):
     cdef double output = 0
     cdef int N, M, i, j
     #need to unpack the implied python tuple
     N = arr.shape[0]
     M = arr.shape[1]
     for i in range(N):
          for j in range(M):
               output += arr[i,j]**2

     return output
```

Compile

```
~$ python reduce_setup.py build_ext --inplace
~$ python reduce_run.py
Execution time: 1.48388695717
Execution time: 0.0024619102478
```

How to know what to Change

Cython has a built in method to show you what C code is generated by each portion of python code. This can help in finding trouble spots

```
~$ cython -a reduce_p1.pyx
~$ cython -a reduce_p2.pyx
```

This will generate some html files, which look like:

reduce_p1.pyx

Generated by Cython 0.18 on Tue Apr 9 22:55:08 2013

Raw output: reduce_p1.c

```
 1: import numpy as np
 2: cimport numpy as np
 3:
 4: def reduce( arr ):
 5:     cdef double output = 0
 6:     cdef int N, M, i, j
 7:     N,M = arr.shape
 8:     for i in range(N):
 9:             for j in range(M):
10:                     output += arr[i,j]**2
11:
12:     return output
```

reduce_p1.pyx

Generated by Cython 0.18 on Tue Apr 9 22:55:08 2013

Raw output: reduce_p1.c

```
 1: import numpy as np
 2: cimport numpy as np
 3:
 4: def reduce( arr ):
 5:     cdef double output = 0
 6:     cdef int N, M, i, j
 7:     N,M = arr.shape
 8:     for i in range(N):
 9:             for j in range(M):
10:                     output += arr[i,j]**2
```

```
    /* "reduce_p1.pyx":10
*     for i in range(N):
*             for j in range(M):
*                     output += arr[i,j]**2           # <<<<<<<<<<<<<<
*
*     return output
*/
    __pyx_t_1 = PyFloat_FromDouble(__pyx_v_output); if (unlikely(!__pyx_t_1)) {__pyx_filename = __pyx_f[0]; __pyx_lineno = 10;
    __Pyx_GOTREF(__pyx_t_1);
    __pyx_t_3 = PyInt_FromLong(__pyx_v_i); if (unlikely(!__pyx_t_3)) {__pyx_filename = __pyx_f[0]; __pyx_lineno = 10; __pyx_cli
    __Pyx_GOTREF(__pyx_t_3);
    __pyx_t_2 = PyInt_FromLong(__pyx_v_j); if (unlikely(!__pyx_t_2)) {__pyx_filename = __pyx_f[0]; __pyx_lineno = 10; __pyx_cli
    __Pyx_GOTREF(__pyx_t_2);
    __pyx_t_4 = PyTuple_New(2); if (unlikely(!__pyx_t_4)) {__pyx_filename = __pyx_f[0]; __pyx_lineno = 10; __pyx_clineno = __LI
    __Pyx_GOTREF(__pyx_t_4);
    PyTuple_SET_ITEM(__pyx_t_4, 0, __pyx_t_3);
    __Pyx_GIVEREF(__pyx_t_3);
    PyTuple_SET_ITEM(__pyx_t_4, 1, __pyx_t_2);
    __Pyx_GIVEREF(__pyx_t_2);
    __pyx_t_3 = 0;
    __pyx_t_2 = 0;
    __pyx_t_2 = PyObject_GetItem(__pyx_v_arr, ((PyObject *)__pyx_t_4)); if (!__pyx_t_2) {__pyx_filename = __pyx_f[0]; __pyx_lin
    __Pyx_GOTREF(__pyx_t_2);
    __Pyx_DECREF(((PyObject *)__pyx_t_4)); __pyx_t_4 = 0;
    __pyx_t_4 = PyNumber_Power(__pyx_t_2, __pyx_int_2, Py_None); if (unlikely(!__pyx_t_4)) {__pyx_filename = __pyx_f[0]; __pyx_
    __Pyx_GOTREF(__pyx_t_4);
    __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
    __pyx_t_2 = PyNumber_InPlaceAdd(__pyx_t_1, __pyx_t_4); if (unlikely(!__pyx_t_2)) {__pyx_filename = __pyx_f[0]; __pyx_lineno
    __Pyx_GOTREF(__pyx_t_2);
    __Pyx_DECREF(__pyx_t_1); __pyx_t_1 = 0;
    __Pyx_DECREF(__pyx_t_4); __pyx_t_4 = 0;
    __pyx_t_10 = __pyx_PyFloat_AsDouble(__pyx_t_2); if (unlikely((__pyx_t_10 == (double)-1) && PyErr_Occurred())) {__pyx_filena
    __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
    __pyx_v_output = __pyx_t_10;
  }
}
```

```
11:
12:     return output
```

Calls to Python objects require lots of C code

reduce_p2.pyx

```
Generated by Cython 0.18 on Tue Apr 9 22:56:18 2013

Raw output: reduce_p2.c

 1: import numpy as np
 2: cimport numpy as np
 3:
 4: def reduce(np.ndarray[np.float64_t, ndim=2]  arr ):
 5:     cdef double output = 0
 6:     cdef int N, M, i, j
 7:     #need to unpack the implied python tuple
 8:     N = arr.shape[0]
 9:     M = arr.shape[1]
10:     for i in range(N):
11:             for j in range(M):
12:                     output += arr[i,j]**2
13:
14:     return output
```

`Def` and `return` interfaces with your python code when you import the module.

Use `cdef` for functions that don't need to interface with python for additional speedup.