Python Short Course
Lecture 5: Extending Python

Richard P. Muller

Materials and Process Simulation Center

Spring, 2000

Extending Python

- Python is great for rapid application development
  - Little overhead in creating classes, functions, etc.
  - Can be slow at times, in suprising places
- Python is fairly easy to profile
  - time.clock() module
  - Python Profiler
- It is fairly easy to write slow features in C
  - Write the program in Python
  - Profile
  - Rewrite slow features in C
- Of course, it's never really that easy…

Profiling Python

```
def main():
  print "Hello, World"

import profile
profile.run('main()') #can also sort by time,
                      # ncalls, etc.
Hello, World
   3 function calls in 0.050 CPU seconds
Ordered by: standard name
ncalls tottime percall cumtime percall function
     1   0.000   0.000   0.000   0.000  main()
     1   0.000   0.000   0.000   0.000  ?
     1   0.500   0.500   0.500   0.500  profile
```

Example, Electrostatic Calculation

- Simple Calculation of ES Energy

```
t0 = time.clock()
n = 1000
charge,x,y,z = getBoxOfCharge(n,-10.,10.,
                        -10.,10.,-10.,10.)
t1 = time.clock()
print "Setup Time = ",t1-t0

E = calcESEnergy(n,charge,x,y,z,distance)
t2 = time.clock()
print "ES Energy, Time = ",E,t2-t1
```

Box of Charge (Random Module)

```
def getBoxOfCharge(n,xmin,xmax,ymin,ymax,zmin,zmax):
  charge,x,y,z = [],[],[],[]
  dx,dy,dz = xmax-xmin,ymax-ymin,zmax-zmin
  for i in range(n):
      charge.append(random.randint(-1,1))
      x.append(dx*random.random()-xmin)
      y.append(dy*random.random()-ymin)
      z.append(dz*random.random()-zmin)
  return charge,x,y,z
```

calcESEnergy function

```
def calcESEnergy(n,charge,x,y,z,dfunc):
 E = 0
 for i in range(n):
   qi,xi,yi,zi = charge[i],x[i],y[i],z[i]
   for j in range(i):
      qj,xj,yj,zj = charge[j],x[j],
         y[j],z[j]
      rij = dfunc(xi,yi,zi,xj,yj,zj)
      E = E + qi*qj/rij
 return E
```

Pointer to Function

Python distance function

```
def distance(xi,yi,zi,xj,yj,zj):
  return math.sqrt((xi-xj)*(xi-xj)+
                   (yi-yj)*(yi-yj) +
                   (zi-zj)*(zi-zj))
```

- Should be fast, math.sqrt() is a C function
- But we have to evaluate all of the math (xi-xj)*(xi-xj)+…
- Plus the function is called 0.5M times!

Results

- Results of python ES run:

```
% python speedtest.py
Python Version
Setup Time: 0.24
ES Energy, Time = -46.539396  23.4
```

- Profiling shows that >50% of time is spent in distance()

```
ncalls tottime percall cumtime percall function
499500 36.220  0.000   36.220  0.000   distance
     1 34.810  34.81   71.030  71.030  calcESEn
```

C Module for Distance (Speedtest.c)

```
#include "Python.h"

static PyObject *py_cdistance(PyObject *self,
                              Pyobject *args){
  double xi,yi,zi,xj,yj,zj,dist;
  PyArg_ParseTuple(args,"dddddd",&xi,&yi,&zi,
                        &xj,&yj,&zj);
  dist = c_dist_function(xi,yi,zi,xj,yj,zj);
  return Py_BuildValue("d",dist);
}
```

C module, cont.

```
static PyMethodDef Speedtest_methods[]={
  {"cdistance",py_cdistance,METH_VARARGS},
  {NULL,NULL}};

void initSpeedtest(){
  (void) Py_InitModule("Speedtest",
            Speedtest_methods);
}
```

Compiling and Using Speedtest

```
% cc -I/exec/python/include/python1.5 -c Sppedtest.c
% cc -shared Speedtest.o -o Speedtest.so
```

- Speedtest.so is now a python module, and can be imported:

```
% python
>>> from Speedtest import *
>>> cdistance(0.,0.,0.,1.,0.,0.)
1.0
```

Comparison of Python and C distance functions

```
# start program as before...

E = calcESEnergy(n,charge,x,y,z,distance)
t2 = time.clock()
Ec = calcESEnergy(n,charge,x,y,z,cdistance)
t3 = time.clock()
print "Python ES Energy, Time = ",E,t2-t1
print "C ES Energy, Time = ",Ec, t3-t2
```

Results

- Factor of two speedup!

```
% python speedtest.py
Setup time: 0.24
Python ES Energy, Time = -44.6501484596 22.31
C ES Energy, Time =      -44.6501484596 11.39
```

- Profiling Results

```
ncalls tottime percall cumtime percall function
     1  11.510  11.510  11.510  11.510 calcESEn
     1   0.300   0.300   0.710   0.710 getBoxOf
```

  - cdistance doesn't even show up in the top 15!

Converting Rest of the function to C

- I don't know how to convert a Python list to a C array
  - There's probably a way somehow
- I do know how to convert NumPy arrays to C
  - Generally when I'm really concerned about speed I'm using numpy anyway
  - Convert the lists to NumPy Arrays; this takes negligible time

```
charge = array(charge,Int)
x = array(x,Float)
etc.
```

Converting NumPy Arrays to C

- Numpy Arrays are C structures

```
typedef struct {
  PyObject_HEAD
  char *data
  int nd;
  int *dimensions, *strides;
  /* skip the rest for brevity */
} PyArrayObject;
```

This is what we want

Converting NumPy Arrays, cont.

- I normally pass in the dimensions, and then cast a pointer to the data fork:

```
PyObject *matrix;
double *mdata;
PyArg_ParseTuple(args,"Oii",&matrix,&n,&m);
mdata = ((double *)(matrix->data))
```

- You can then use mdata as a normal C array:

```
for (i=0; i<n; i++){
  for (j=0; j<m; j++){
      mdata[j+i*n] = 0.;
  }
}
```

cCalcESEnergy Wrapper

```
#include "Python.h"
#include "arrayobject.h"
#define IDATA(p) ((int *) (((PyArrayObject *)p)->data))
#define DDATA(p) ((double *) (((PyArrayObject *)p)->data))

static PyObject *py_cCalcESEnergy(PyObject *self, PyObject *args){
  int n, *q;
  PyObject *xarray, *yarray, *zarray, *qarray;
  double *x, *y, *z, energy;
  PyArg_ParseTuple(args,"i0000",&n,&qarray,&xarray,&yarray,&zarray);
  q = IDATA(qarray);
  x = DDATA(xarray);
  y = DDATA(yarray);
  z = DDATA(zarray);
  energy = c_calc_ES_energy(n,q,x,y,z);
  return Py_BuildValue("d",energy);
```

Final Results

- Final timings

Pure Python	24.65
Python/C	13.34
C called from Python	0.27

- Lessons
  - Nested loops are typically slow in Python
  - Anything called 0.5 M times bears looking at

  - Generally can speed program by rewriting only a small part of it
  - Python gives more rapid development, and thus easier introduction of new features (e.g. fast multipoles).

SWIG

- Simple Wrapper and Interface Generator
  - Automatic method of generating wrappers
  - Perl, Python, Tcl/Tk, Java, Eiffel, etc.
- IMHO more trouble than it's worth for small functions
- Important tool for functions with hundreds of entry points:
  - OpenGL module
  - VTK

References

- Web Pages
  - http://www.python.org/doc/current/ext/ext.html "Extending Python," Guido van Rossum
  - http://www.swig.org SWIG Web Page
- Books
  - Programming Python, Mark Lutz, ORA; Chapter 14 is on Extending/Embedding Python