56-COMPL-cython

December 3, 2017

Sin Cython

Este programa genera N enteros aleatorios entre 1 y M, y una vez obtenidos los eleva al cuadrado y devuelve la suma de los cuadrados. Por tanto, calcula el cuadrado de la longitud de un vector aleatorio con coordenadas enteros en el intervalo $[1, M]$.

```
In [1]: def cuadrados(N,M):
            res = 0
            for muda in xrange(N):
                x = randint(1,M)
                res += x*x
            return res

In [2]: for n in srange(3,8):
            %time A = cuadrados(10^n,10^6)
```

```
CPU times: user 12 ms, sys: 4 ms, total: 16 ms
Wall time: 11.7 ms
CPU times: user 56 ms, sys: 36 ms, total: 92 ms
Wall time: 70.5 ms
CPU times: user 644 ms, sys: 76 ms, total: 720 ms
Wall time: 539 ms
CPU times: user 3.58 s, sys: 212 ms, total: 3.79 s
Wall time: 3.54 s
CPU times: user 33.1 s, sys: 392 ms, total: 33.5 s
Wall time: 33.1 s
```

Con Cython

Esta sección debe usar el núcleo de Python2. Efectuamos el mismo cálculo:

```
In [3]: %load_ext cython

In [4]: %%cython -a

        import math
        import random
        def cuadrados_cy(long long N, long long M):
            cdef long long res = 0
```

```
            cdef long long muda
            cdef long long x
            for muda in xrange(N):
                x = random.randint(1,M)
                res += math.pow(x,2)
            return res
```

```
Out[4]: <IPython.core.display.HTML object>
```

```
In [5]: for n in range(3,8):
            %time A = cuadrados_cy(10^n,10^6)
```

```
CPU times: user 4 ms, sys: 0 ns, total: 4 ms
Wall time: 2.1 ms
CPU times: user 16 ms, sys: 16 ms, total: 32 ms
Wall time: 20.2 ms
CPU times: user 220 ms, sys: 76 ms, total: 296 ms
Wall time: 202 ms
CPU times: user 1.7 s, sys: 144 ms, total: 1.84 s
Wall time: 1.66 s
CPU times: user 15.6 s, sys: 140 ms, total: 15.7 s
Wall time: 15.6 s
```

Optimizando el cálculo de números aleatorios:

Esta sección debe utilizar el núcleo de Sage. No funciona la opción *-a* al llamar a cython y no vemos la dependencia de Python del código.

La primera parte de la celda, hasta *def main():*, genera enteros aleatorios entre 1 y 10^6 usando librerías externas compilables en C. Este trozo se puede reutilizar.

```
In [1]: %%cython

        cdef extern from "gsl/gsl_rng.h":
           ctypedef struct gsl_rng_type:
              pass
           ctypedef struct gsl_rng:
               pass
           gsl_rng_type *gsl_rng_mt19937
           gsl_rng *gsl_rng_alloc(gsl_rng_type * T)

        cdef gsl_rng * r = gsl_rng_alloc(gsl_rng_mt19937)

        cdef extern from "gsl/gsl_randist.h":
             long int uniform "gsl_rng_uniform_int"(gsl_rng * r, unsigned long int n)

        def main():
            cdef int n
            n = uniform(r,1000000)
            return n
```

```
        cdef long f(long x):
            return x**2

        import random
        def cuadrados_cy2(int N):
            cdef long res = 0
            cdef int muda
            for muda in range(N):
                res += f(main())
            return res
```

```
In [2]: for n in srange(3,8):
            %time A = cuadrados_cy2(10^n)
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 175 µs
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 1.2 ms
CPU times: user 12 ms, sys: 0 ns, total: 12 ms
Wall time: 10.5 ms
CPU times: user 80 ms, sys: 0 ns, total: 80 ms
Wall time: 80 ms
CPU times: user 540 ms, sys: 0 ns, total: 540 ms
Wall time: 540 ms
```

Problema similar sin números aleatorios:

```
In [3]: %%cython
        def cuadrados_cy3(long long int N):
            cdef long long int res = 0
            cdef long long int k
            for k in range(N):
                res += k**2
            return res
```

```
In [4]: for n in srange(3,8):
            %time A = cuadrados_cy3(10^n)
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 39.1 µs
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 37.9 µs
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 175 µs
CPU times: user 4 ms, sys: 0 ns, total: 4 ms
Wall time: 1.45 ms
CPU times: user 16 ms, sys: 0 ns, total: 16 ms
```

```
Wall time: 13.9 ms
```

```
In [5]: def cuadrados5(N):
            res = 0
            for k in range(N):
                res += k**2
            return res

In [6]: for n in srange(3,8):
            %time A = cuadrados5(10^n)

CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 932 µs
CPU times: user 8 ms, sys: 0 ns, total: 8 ms
Wall time: 6.77 ms
CPU times: user 56 ms, sys: 0 ns, total: 56 ms
Wall time: 55.3 ms
CPU times: user 364 ms, sys: 24 ms, total: 388 ms
Wall time: 385 ms
CPU times: user 3.12 s, sys: 136 ms, total: 3.26 s
Wall time: 3.26 s
```

Hemos comprobado, de dos maneras, que es en la generación de los números aleatorios donde Python pasa la mayor parte del tiempo en este cálculo. Si optimizamos esa parte, usando una librería en C, o simplemente la suprimimos, el cálculo es mucho más rápido. Cython pierde muchísima eficiencia cuando debe ejecutar funciones de Python que son mucho más lentas que las correspondientes funciones en C.