114-PROBA-Generador-cython

April 23, 2018

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 5.61 ms, sys: 263 µs, total: 5.87 ms
Wall time: 6.09 ms
CPU times: user 44.7 ms, sys: 0 ns, total: 44.7 ms
Wall time: 57.6 ms
CPU times: user 477 ms, sys: 57.1 ms, total: 534 ms
Wall time: 480 ms
CPU times: user 4.39 s, sys: 130 ms, total: 4.52 s
Wall time: 4.42 s
CPU times: user 43.3 s, sys: 403 ms, total: 43.7 s
Wall time: 43.4 s
```

Con Cython

Mismo cálculo:

```
In [3]: %%cython
        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
```

```
In [4]: for n in srange(3,8):
            %time A = cuadrados_cy(10^n,10^6)
```

```
CPU times: user 2.94 ms, sys: 48 µs, total: 2.99 ms
Wall time: 2.83 ms
CPU times: user 28.7 ms, sys: 34 µs, total: 28.7 ms
Wall time: 33.3 ms
CPU times: user 259 ms, sys: 24.3 ms, total: 284 ms
Wall time: 252 ms
CPU times: user 2.47 s, sys: 167 ms, total: 2.64 s
Wall time: 2.44 s
CPU times: user 22.9 s, sys: 258 ms, total: 23.1 s
Wall time: 22.9 s
```

Optimizando el cálculo de números aleatorios:

```
In [5]: %%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 [6]: for n in srange(3,8):
            %time A = cuadrados_cy2(10^n)
```

```
CPU times: user 68 µs, sys: 1e+03 ns, total: 69 µs
Wall time: 72 µs
CPU times: user 568 µs, sys: 0 ns, total: 568 µs
Wall time: 511 µs
CPU times: user 4.82 ms, sys: 0 ns, total: 4.82 ms
Wall time: 4.83 ms
CPU times: user 48.1 ms, sys: 0 ns, total: 48.1 ms
Wall time: 48.5 ms
CPU times: user 479 ms, sys: 0 ns, total: 479 ms
Wall time: 479 ms
```

Problema similar sin números aleatorios:

```
In [8]: %%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 [9]: for n in srange(3,8):
            %time A = cuadrados_cy3(10^n)
```

```
CPU times: user 12 µs, sys: 0 ns, total: 12 µs
Wall time: 16.9 µs
CPU times: user 16 µs, sys: 1e+03 ns, total: 17 µs
Wall time: 20 µs
CPU times: user 75 µs, sys: 2 µs, total: 77 µs
Wall time: 78 µs
CPU times: user 770 µs, sys: 0 ns, total: 770 µs
Wall time: 719 µs
CPU times: user 7.01 ms, sys: 0 ns, total: 7.01 ms
Wall time: 10.2 ms
```

```
In [10]: def cuadrados5(N):
             res = 0
             for k in range(N):
                 res += k**2
             return res
```

```
In [11]: for n in srange(3,8):
             %time A = cuadrados5(10^n)

CPU times: user 643 µs, sys: 0 ns, total: 643 µs
Wall time: 487 µs
CPU times: user 1.35 ms, sys: 3.95 ms, total: 5.31 ms
Wall time: 4.26 ms
CPU times: user 25.5 ms, sys: 12 ms, total: 37.4 ms
Wall time: 37.4 ms
CPU times: user 373 ms, sys: 28.2 ms, total: 402 ms
Wall time: 387 ms
CPU times: user 3.75 s, sys: 169 ms, total: 3.92 s
Wall time: 3.88 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.