知乎 首发于 Python加速度

别再说Python慢(数组篇)

李小文
算法工程师

2 人赞同了该文章

4种方法提升Python数组的效率

项目地址:

https://github.com/tushushu/flying-python
github.com

1. Python的列表为什么慢

组里存放了三个指针，分别指向了这三个元素。那么相比其他语言的数组而言，为什么Python的列表会慢呢？原因主要是以下两个：

1. Python是动态类型语言，意味着类型检查要耗费额外的时间。

2. Python或者说Cpython没有JIT优化器。

2. 如何用Python执行快速的数组计算

目前比较主流的解决方案有如下几种：

1. Numpy - Numpy的array更像是C/C++的数组，数据类型一致，而且array的方法(如sum)都是用C来实现的。

2. Numba - 使用JIT技术，优化Numpy的性能。无论是调用Numpy的方法，还是使用for循环遍历Numpy数组，都可以得到性能提升。

3. Numexpr - 避免Numpy为中间结果分配内存，优化Numpy性能，主要用于大数组的表达式计算。

4. Cython - 为Python编写C/C++扩展。

接下来通过两个例子来演示如何通过这四种工具

3. 数组求平方和

```
arr = [x for x in range(10000)]
```

3.1 for循环

```
def sqr_sum(arr):
    total = 0
    for x in arr:
        total += x ** 2
    return total
```

3.2 Numpy

```
import numpy as np
def sqr_sum(arr):
    return (arr ** 2).sum()

arr = np.array(arr)
print("The result is:", sqr_sum(arr))
%timeit sqr_sum(arr)
The result is: 333283335000
9.66 µs ± 275 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

3.3 Numba

```
from numba import jit
@jit(nopython=True)
def sqr_sum(arr):
    return (arr ** 2).sum()

arr = np.array(arr)
print("The result is:", sqr_sum(arr))
%timeit sqr_sum(arr)
The result is: 333283335000
3.39 µs ± 57.2 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

3.4 Numexpr

```
import numexpr as ne
def sqr_sum(arr):
    return ne.evaluate("sum(arr * arr)")

arr = np.array(arr)
print("The result is:", sqr_sum(arr))
%timeit sqr_sum(arr)
The result is: 333283335000
14.9 µs ± 144 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

```
%load_ext cython
%%cython
cimport numpy as np
ctypedef np.int_t DTYPE_t

def sqr_sum(np.ndarray[DTYPE_t] arr):
    cdef:
        DTYPE_t total = 0
        DTYPE_t x
        int i = 0
        int n = len(arr)
    while i < n:
        total += arr[i] ** 2
        i += 1
    return total
arr = np.array(arr, dtype="int")
print("The result is:", sqr_sum(arr))
%timeit sqr_sum(arr)
The result is: 333283335000
5.51 µs ± 62.4 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

4. 数组变换

```
arr = [x for x in range(1000000)]
```

4.1 for循环

```
def transform(arr):
    return [x * 2 + 1 for x in arr]

print("The result is:", transform(arr)[:5], "...")
%timeit transform(arr)
The result is: [1, 3, 5, 7, 9] ...
84.5 ms ± 381 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

4.2 Numpy

```
    return arr * 2 + 1

arr = np.array(arr)
print("The result is:", transform(arr)[:5], "...")
%timeit transform(arr)
The result is: [1 3 5 7 9] ...
803 µs ± 11.4 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

4.3 Numba

```
from numba import jit
@jit(nopython=True)
def transform(arr):
    return arr * 2 + 1

arr = np.array(arr)
print("The result is:", transform(arr)[:5], "...")
%timeit transform(arr)
The result is: [1 3 5 7 9] ...
498 µs ± 8.71 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

4.4 Numexpr

```
import numexpr as ne
def transform(arr):
    return ne.evaluate("arr * 2 + 1")

arr = np.array(arr)
print("The result is:", transform(arr)[:5], "...")
%timeit transform(arr)
The result is: [1 3 5 7 9] ...
369 µs ± 13.2 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

4.5 Cython

```
%load_ext cython
```

```
cimport numpy as np
ctypedef np.int_t DTYPE_t

def transform(np.ndarray[DTYPE_t] arr):
    cdef:
        np.ndarray[DTYPE_t] new_arr = np.empty_like(arr)
        int i = 0
        int n = len(arr)
    while i < n:
        new_arr[i] = arr[i] * 2 + 1
        i += 1
    return new_arr
arr = np.array(arr)
print("The result is:", transform(arr)[:5], "...")
%timeit transform(arr)
The result is: [1 3 5 7 9] ...
887 µs ± 29.3 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

5. 参考文章

How does python have different data types in an array?
Why are Python Programs often slower than the Equivalent Program Written in C or C++?
How Fast Numpy Really is and Why?

编辑于 2020-05-13

Python 科学计算

文章被以下专栏收录

Python加速度
让Python飞起来

进入专栏

知乎 首发于 **Python加速度**

100 个 Python 小例子

Pytho... 发表于Pytho...

还没有评论

写下你的评论...

▲ 赞同 2 ▼ 添加评论 分享 喜欢 收藏 ...