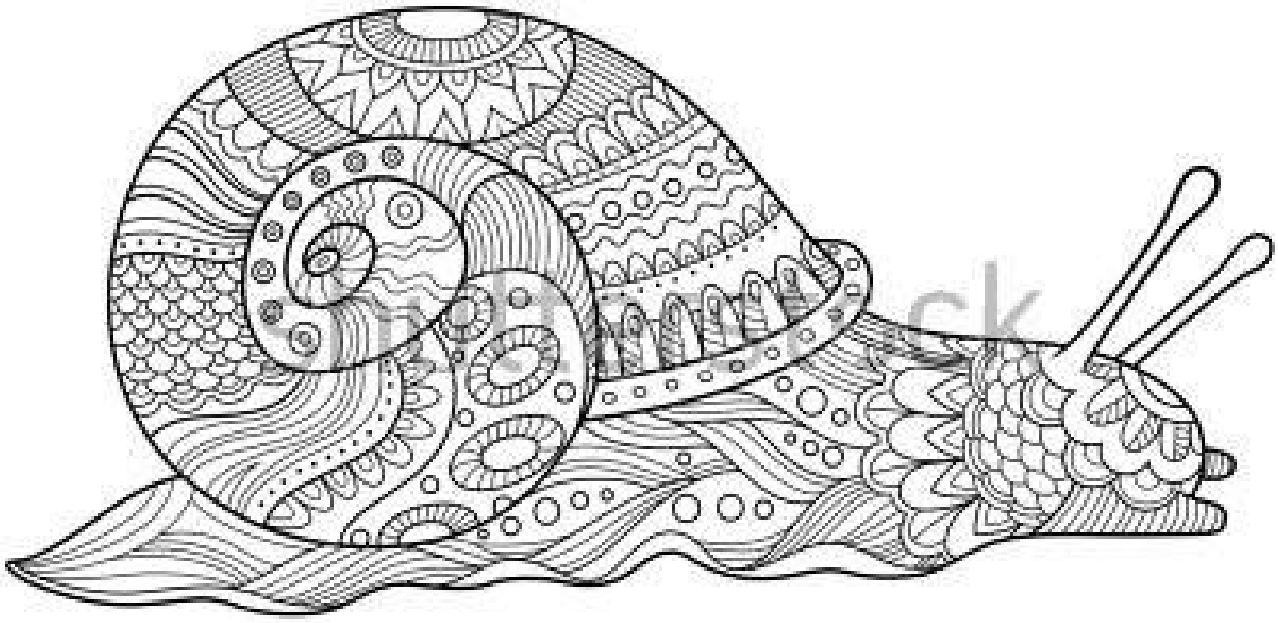


Python Numba慢如蜗牛?

li112721...
上海财经大学 经济学硕士

关注他

8 人赞同了该文章

请允许我"标题党"一把, 因为确实碰到了"慢如蜗牛"的例子.

"@stencil"

英文名的意思是"模板, 漏板"

功能类似于filter, conv, conv2

输出变量与输入变量的形状相同(都是数组)

Problem 11 - Project Euler
projecteuler.net

题目的意思是: 已知一个20*20的矩阵, 找出连续4个元素乘积的最大值, 方向可以是横竖撇捺4种.

写了一下代码:

```
import numpy as np
from numba import stencil, njit


data = []
for row in open("euler011.txt", "r"):
    data.append(list(map(int, row.split(" "))))
data = np.array(data)

@stencil
def kernel_col(a):
    return a[0, 0] * a[-1, 0] * a[-2, 0] * a[-3, 0]

@stencil
def kernel_row(a):
    return a[0, 0] * a[0, -1] * a[0, -2] * a[0, -3]

@stencil
def kernel_diag1(a):
    return a[0, 0] * a[-1, -1] * a[-2, -2] * a[-3, -3]

@stencil
def kernel_diag2(a):
    return a[0, 0] * a[-1, 1] * a[-2, 2] * a[-3, 3]

def euler011_stencil():
    out_col = kernel_col(data).max()
    out_row = kernel_row(data).max()
    out_diag1 = kernel_diag1(data).max()
    out_diag2 = kernel_diag2(data).max()
    return max(out_col, out_row, out_diag1, out_diag2)
```

然后就写了另外两个版本(纯Python版与njit版)进行对比测试.

纯Python版:

```
def euler011_loop():
    res = 0
    for i in range(20):
        for j in range(3, 20):
            num = data[i, j] * data[i, j-1] * data[i, j-2] * data[i, j-3]
            if num > res:
                res = num
    for j in range(20):
        for i in range(3, 20):
            num = data[i, j] * data[i - 1, j] * data[i - 2, j] * data[i - 3, j]
            if num > res:
                res = num
    for j in range(3, 20):
        for i in range(3, 20):
            num = data[i, j] * data[i - 1, j - 1] * data[i - 2, j - 2] * data[i - 3, j
            if num > res:
                res = num
    for i in range(3, 20):
        for j in range(17):
            num = data[i, j] * data[i - 1, j + 1] * data[i - 2, j + 2] * data[i - 3, j
            if num > res:
                res = num
    return res
```

njit版:

```
@njit
def euler011_njit():
    res = 0
    for i in range(20):
        for j in range(3, 20):
            num = data[i, j] * data[i, j-1] * data[i, j-2] * data[i, j-3]
            if num > res:
                res = num
```

```
                res = num
    for j in range(3, 20):
        for i in range(3, 20):
            num = data[i, j] * data[i - 1, j - 1] * data[i - 2, j - 2] * data[i - 3, j
            if num > res:
                res = num
    for i in range(3, 20):
        for j in range(17):
            num = data[i, j] * data[i - 1, j + 1] * data[i - 2, j + 2] * data[i - 3, j
            if num > res:
                res = num
    return res
```

运行检查结果:

```
print(euler011_stencil())
print(euler011_loop())
print(euler011_njit())

70600674
70600674
70600674
```

OK: 结果一致.

测试一下性能:

```
In [8]: %timeit euler011_stencil()
702 ms ± 6.22 ms per loop (mean ± std. dev. of 7 runs, 1 loop
each)

In [9]: %timeit euler011_loop()
1.36 ms ± 1.59 µs per loop (mean ± std. dev. of 7 runs, 1000
loops each)

In [10]: %timeit euler011_njit()
1.5 µs ± 6.45 ns per loop (mean ± std. dev. of 7 runs, 1000000
loops each)
```

2 "@njit"版的的运行时间是1.5 微秒, 速度是纯Python代码速度的906倍, 这是numba让我满意的地方.

3 "@njit"和"@stencil"都来自于Numba库, 前者速度是后者的468000倍!

为什么?

有读者能给我一个解释吗?

是我用错了? 还是"@stencil"不成熟, 有bug?

封面图片有两层含义.

1 它是模板画, 如同"@stencil"字面意思.

2 "@stencil"速度太慢了, 如同蜗牛一下(有可能是因为我用得不好, 希望有人能指点一下)

stencil的文档:

1.11. Using the @stencil decorator - Numba...
numba.pydata.org

总结:

1 Numba的"@njit"是一个Python提速的利器.

2 Numba的"@stencil"在实战中非常慢, 在查明原因之前, 不敢用了.

关于Numba的其他文章:

haitao：[高性能实战案例] MATLAB,Julia, Python(Numba)
zhuanlan.zhihu.com

更新:

做了一下实验, 发现确实评论区里面的曹洪洋所说, "@stencil"有一个固定的启动时间.

```
import numpy as np
from numba import stencil, njit


#data = []
#for row in open("euler011.txt", "r"):
#    data.append(list(map(int, row.split(" "))))
#data = np.array(data)

np.random.seed(seed=41)
data = np.random.randint(1, 100, size=(2000, 2000))

@stencil
def kernel_col(a):
    return a[0, 0] * a[-1, 0] * a[-2, 0] * a[-3, 0]

@stencil
def kernel_row(a):
    return a[0, 0] * a[0, -1] * a[0, -2] * a[0, -3]

@stencil
def kernel_diag1(a):
    return a[0, 0] * a[-1, -1] * a[-2, -2] * a[-3, -3]

@stencil
def kernel_diag2(a):
    return a[0, 0] * a[-1, 1] * a[-2, 2] * a[-3, 3]

def euler011_stencil(data):
```

```
    return max(out_col, out_row, out_diag1, out_diag2)


def euler011_loop(data):
    res = 0
    N = data.shape[0]
    for i in range(N):
        for j in range(3, N):
            num = data[i, j] * data[i, j-1] * data[i, j-2] * data[i, j-3]
            if num > res:
                res = num
    for j in range(N):
        for i in range(3, N):
            num = data[i, j] * data[i - 1, j] * data[i - 2, j] * data[i - 3, j]
            if num > res:
                res = num
    for j in range(3, N):
        for i in range(3, N):
            num = data[i, j] * data[i - 1, j - 1] * data[i - 2, j - 2] * data[i - 3, j
            if num > res:
                res = num
    for i in range(3, N):
        for j in range(N - 3):
            num = data[i, j] * data[i - 1, j + 1] * data[i - 2, j + 2] * data[i - 3, j
            if num > res:
                res = num
    return res


@njit
def euler011_njit(data):
    res = 0
    N = data.shape[0]
    for i in range(N):
        for j in range(3, N):
            num = data[i, j] * data[i, j-1] * data[i, j-2] * data[i, j-3]
            if num > res:
                res = num
    for j in range(N):
        for i in range(3, N):
            num = data[i, j] * data[i - 1, j] * data[i - 2, j] * data[i - 3, j]
```

```
            num = data[i, j] * data[i - 1, j - 1] * data[i - 2, j - 2] * data[i - 3, j
            if num > res:
                res = num
    for i in range(3, N):
        for j in range(N - 3):
            num = data[i, j] * data[i - 1, j + 1] * data[i - 2, j + 2] * data[i - 3, j
            if num > res:
                res = num
    return res

print(euler011_stencil(data))
#print(euler011_loop(data))
print(euler011_njit(data))
```

data为4*4时:

```
In [6]: %timeit euler011_stencil(data)
708 ms ± 4.95 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

In [7]: %timeit euler011_loop(data)
14.3 µs ± 39 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```



```
In [9]: %timeit euler011_njit(data)
322 ns ± 0.581 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

data为20*20时:

```
In [8]: %timeit euler011_stencil()
702 ms ± 6.22 ms per loop (mean ± std. dev. of 7 runs, 1 loop
each)

In [9]: %timeit euler011_loop()
1.36 ms ± 1.59 µs per loop (mean ± std. dev. of 7 runs, 1000
loops each)

In [10]: %timeit euler011_njit()
1.5 µs ± 6.45 ns per loop (mean ± std. dev. of 7 runs, 1000000
loops each)
```

```
In [12]: %timeit euler011_loop(data)
155 ms ± 188 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)

In [13]: %timeit euler011_njit(data)
263 µs ± 718 ns per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```


data为2000*2000时:

```
In [8]: %timeit euler011_stencil(data)
774 ms ± 3.91 ms per loop (mean ± std. dev. of 7 runs, 1 loop
each)

In [9]: %timeit euler011_njit(data)
70.5 ms ± 363 µs per loop (mean ± std. dev. of 7 runs, 10 loops
each)
```


data为4000*4000时:

```
In [15]: %timeit euler011_stencil(data)
1.02 s ± 2.62 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

In [16]: %timeit euler011_njit(data)
324 ms ± 1.51 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```


data为8000*8000时:

```
In [18]: %timeit euler011_stencil(data)
1.95 s ± 9.87 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

In [19]: %timeit euler011_njit(data)
1.57 s ± 4.01 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```


data为16000*16000时:

```
In [21]: %timeit euler011_stencil(data)
5.7 s ± 17.1 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

In [22]: %timeit euler011_njit(data)
11.5 s ± 150 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

"@stencil"有固定的启动时间(调用4次, 大概700ms), 因此, 小数据下, 使用"@njit"更快, 大数据下(10^10次级别), "@stencil"更快.

创作不易, 请大家"素质三连": 点赞, 收藏, 分享.

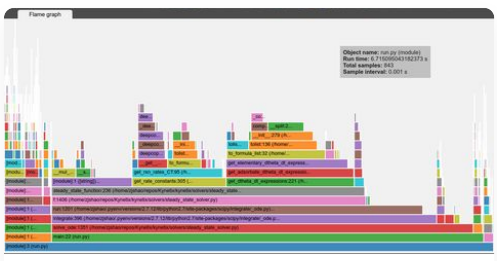
编辑于 2020-12-09

Python　高性能计算

文章被以下专栏收录

MATLAB Python 机器学习
分享数据分析，机器学习与编程的心得体会

推荐阅读



python

前言最近
python进
写，RGB
一开始用
时间效率

8 条评论 ⇌ 切换为时间排序

写下你的评论...

曹洪洋 2019-12-06

使用@stencil时，每次都需要固定的“启动”时间（接近200ms），哪怕是2×2的矩阵也是如此，把data改成更大的矩阵，比如5000×5000，这时优势就体现出来了，可以看到耗时并没有增加多少

4

li1127217ye (作者) 回复 曹洪洋 2019-12-06

感谢答疑。实验证实了

赞

曹洪洋 回复 li1127217ye (作者) 2019-12-06

不客气。其实这4个stencil可以合在一起，而且合起来后会快一点。

@stencil
def kernel(a):
return max(a[0, 0] * a[-1, 0] * a[-2, 0] * a[-3, 0], a[0, 0] * a[0, -1] * a[0, -2] * a[0, -3], a[0, 0] * a[-1, -1] * a[-2, -2] * a[-3, -3], a[0, 0] * a[-1, 1] * a[-2, 2] * a[-3, 3])

赞

li1127217ye (作者) 回复 曹洪洋 2019-12-06

我一开始也是这么做的，后来发现边界问题(越界的为0)，有可能漏掉边界上的最大值。

赞

li1127217ye (作者) 回复 曹洪洋 2019-12-06

如果用小矩阵，很容易观测到这样做有问题，由于它的边界处理的方法。

赞

曹洪洋 回复 li1127217ye (作者) 2019-12-07

确实。用for循环的版本倒是可以合并一下的，if语句需要稍微修改一下，这个看个人习惯了。分开写的好处是和用stencil对比可以更清楚看出对应关系

赞

li1127217ye (作者) 回复 Falccm 2019-12-07

对，我以前用MATLAB也是这么做的。但是还是不通用。如果要求其他复杂运算的话，就不行了。

赞