浙江大學

课程综合实践
实验报告

实验名称 NumPy 向量化计算

姓名学号 陶泓宇 3200103929

实验日期 2021 年 7 月 29 日

目录

1 实验目的和要求

NumPy 是 Python 中科学计算的基础包。它是一个 Python 库，提供多维数组对象，各种派生对象（如掩码数组和矩阵），以及用于数组快速操作的各种 API，有包括数学、逻辑、形状操作、排序、选择、输　输出、离散傅立叶变换、基本线性代数，基本统计运算和随机模拟等等。

本次实验我们将借助 NumPy 实现一个支持批量处理的向量化的双线性插值，来让大家熟悉 NumPy 的向量化编程模式。

2 操作方法和实验步骤

2.1 基准代码

原始代码如下图所示：

```python
import numpy as np
from numpy import int64


def bilinear_interp_baseline(a: np.ndarray, b: np.ndarray) -> np.ndarray:
    """
    This is the baseline implementation of bilinear interpolation without vectorization.
    - a is a ND array with shape [N, H1, W1, C], dtype = int64
    - b is a ND array with shape [N, H2, W2, 2], dtype = float64
    - return a ND array with shape [N, H2, W2, C], dtype = int64
    """
    # Get axis size from ndarray shape
    N, H1, W1, C = a.shape
    N1, H2, W2, _ = b.shape
    assert N == N1

    # Do iteration
    res = np.empty((N, H2, W2, C), dtype=int64)
    for n in range(N):
        for i in range(H2):
            for j in range(W2):
                x, y = b[n, i, j]
                x_idx, y_idx = int(np.floor(x)), int(np.floor(y))
                _x, _y = x - x_idx, y - y_idx
                # For simplicity, we assume all x are in [0, H1 - 1), all y are in [0, W1 - 1)
                res[n, i, j] = a[n, x_idx, y_idx] * (1 - _x) * (1 - _y) + a[n, x_idx + 1, y_idx] * _x * (1 - _y) + \
                               a[n, x_idx, y_idx + 1] * (1 - _x) * _y + a[n, x_idx + 1, y_idx + 1] * _x * _y
    return res
```

图 1: 基准代码

2.2 numpy 优化代码

优化后的总体代码如下：

```
# TODO: Implement vectorized bilinear interpolation
res = np.empty((N, H2, W2, C),dtype=int64)
b_x = np.empty((N, H2, W2, 1), dtype=float64)
b_int_x = np.empty((N, H2, W2, 1), dtype=float64)
b_float_x = np.empty((N, H2, W2, 1), dtype=float64)
b_int_y = np.empty((N, H2, W2, 1), dtype=float64)
b_float_y = np.empty((N, H2, W2, 1), dtype=float64)
b_y = np.empty((N, H2, W2, 1), dtype=float64)
res_index_00 = np.empty((N, H2, W2, C),dtype=float64)
res_index_01 = np.empty((N, H2, W2, C),dtype=float64)
res_index_10 = np.empty((N, H2, W2, C),dtype=float64)
res_index_11 = np.empty((N, H2, W2, C),dtype=float64)

for i in range (N):
    b_x[i] = b[i].reshape(-1)[::2].reshape(1,720,1280,1)
    b_int_x[i] = b[i].reshape(-1)[::2].reshape(1,720,1280,1).astype(np.int64)
    b_float_x[i] = (b_x[i] - b_int_x[i]).astype(np.float64)
    b_y[i] = b[i].reshape(-1)[1::2].reshape(1,720,1280,1)
    b_int_y[i] = b[i].reshape(-1)[1::2].reshape(1,720,1280,1).astype(np.int64)
    b_float_y[i] = (b_y[i] - b_int_y[i]).astype(np.float64)

for i in range(N):
    res_index_00[i] = a[i][[b[i].astype(np.int).reshape(-1)[::2]],[b[i].astype(np.int).reshape(-1)[1::2]]].reshape(1,720,1280,4)
    res_index_01[i] = a[i][[b[i].astype(np.int).reshape(-1)[::2]],[b[i].astype(np.int).reshape(-1)[1::2]+1]].reshape(1,720,1280,4)
    res_index_10[i] = a[i][[b[i].astype(np.int).reshape(-1)[::2]+1],[b[i].astype(np.int).reshape(-1)[1::2]]].reshape(1,720,1280,4)
    res_index_11[i] = a[i][[b[i].astype(np.int).reshape(-1)[::2]+1],[b[i].astype(np.int).reshape(-1)[1::2]+1]].reshape(1,720,1280,4)

res_index_00 = res_index_00 * (1-b_float_x) * (1-b_float_y)
res_index_10 = res_index_10 * (b_float_x) * (1-b_float_y)
res_index_01 = res_index_01 * (1-b_float_x) * (b_float_y)
res_index_11 = res_index_11 * (b_float_x) * (b_float_y)
res = res_index_00 + res_index_10 + res_index_01 + res_index_11
return res
```

图 2: numpy 优化代码

2.3 优化思路

首先，我们通过如下代码分别获得 b 矩阵中对应 a 矩阵的 x，y 坐标矩阵：

```
b_x[i] = b[i].reshape(-1)[::2].reshape(1,720,1280,1)
b_y[i] = b[i].reshape(-1)[1::2].reshape(1,720,1280,1)
```

b[i].reshape(-1) 将 b 矩阵一维化，b[i].reshape(-1)[::2] 则将一维化以后的矩阵每隔一个取一个数组成新的 x 坐标矩阵；reshape(1,720,1280,1) 则将一维矩阵再次变回 (8,720,1280,1) 的矩阵，以便后续的乘法运算。

之后，我们再通过如下代码获得坐标矩阵的整数部分和小数部分

```
b_int_x[i] = b[i].reshape(-1)[::2].reshape(1,720,1280,1).astype(np.int64)
b_float_x[i] = (b_x[i] - b_int_x[i]).astype(np.float64)
b_int_y[i] = b[i].reshape(-1)[1::2].reshape(1,720,1280,1).astype(np.int64)
b_float_y[i] = (b_y[i] - b_int_y[i]).astype(np.float64)
```

之后，由于 numpy 的二维花式索引特性（见下图）：

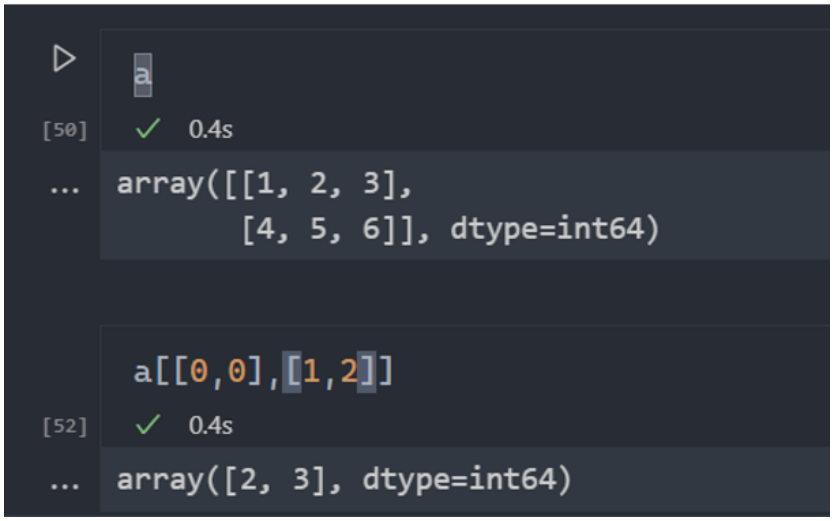

图 3: numpy 二维花式索引

我们采用如下代码以获得 b_x,b_y 矩阵对应到 a 矩阵中得到的新矩阵：

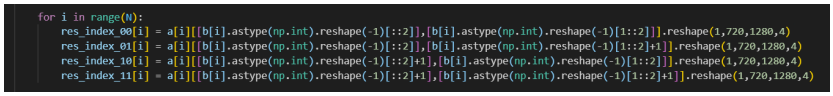

图 4: 得到新矩阵

由于二维花式索引的限制，我们采用了一个 0 到 8 的 for 循环以获得完整维度的结果矩阵；b[i].astype(np.int).reshape(-1)[::2] 表示索引到 a 矩阵对应的 x 坐标，b[i].astype(np.int).reshape(-1)[1::2] 表示索引到 a 矩阵对应的 y 坐标；reshape(1,720,1280,4) 则将一维的矩阵变为要求维数

之后的处理方法则与给出的基准代码类似

```
res_index_00=res_index_00*(1-b_float_x)*(1-b_float_y)
res_index_10=res_index_10*(b_float_x)*(1-b_float_y)
res_index_01=res_index_01*(1-b_float_x）*(b_float_y)
res_index_11=res_index_11*(b_float_x)*(b_float_y)
res=res_index_00+res_index_10+res_index_01+res_index_11
```

3 实验数据记录和处理

下面是几次运行结果的截图；总体而言，代码运行时间缩短到了 4-7 秒，加速了 20-30 倍

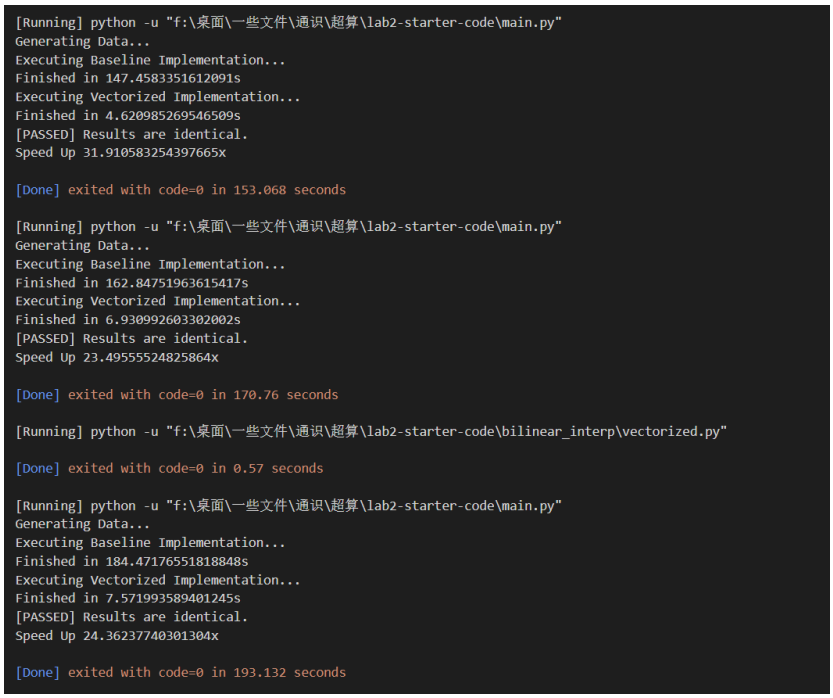

图 5: 运行结果

4 实验结果与分析

本次优化仍然使用了一次 for 循环，虽然只循环了 8 次，但仍降低了代码的运行速度；但由于没找到三维的花式索引方式，所以目前还是只能这样。