向量化计算

代码

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

def bilinear_interp_vectorized(a: np.ndarray, b: np.ndarray) -> np.ndarray:
    """
        This is the vectorized implementation of bilinear interpolation.
        - 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

    # TODO: Implement vectorized bilinear interpolation
    res = np.empty((N, H2, W2, C), dtype=int64)

    padding = np.empty((N, H2, W2, 2))
    padding[:, :, :, :] = b

    xy = np.array(np.meshgrid(np.arange(H2), np.arange(W2)), dtype=np.float32)
    x = xy[0].flatten().astype(np.int)
    y = xy[1].flatten().astype(np.int)

    index = padding[:, x, y, :]
    clip_index = np.floor(index).astype(np.int)

    clip_index = clip_index.reshape((-1, 2))
    index = index.reshape((-1, 2))
    a = a.reshape((-1, C))

    _xy = index - clip_index
    n = np.array(np.meshgrid(np.arange(W2*H2), np.arange(N)), dtype=np.int32)
    n = n[1].flatten().astype(np.int)

    result = (1-_xy[:, 0, None]) * (1-_xy[:, 1, None]) * a[clip_index[:, 0]*W1 + clip_index[:, 1] + n[:]*W1*H1, :] + \
             _xy[:, 0, None] * (1 - _xy[:, 1, None]) * a[(clip_index[:, 0]+1)*W1 + clip_index[:, 1] + n[:]*W1*H1, :] + \
             (1-_xy[:, 0, None]) * _xy[:, 1, None] * a[clip_index[:, 0]*W1 + (clip_index[:, 1]+1) + n[:]*W1*H1, :] + \
             _xy[:, 0, None] * _xy[:, 1, None] * a[(clip_index[:, 0]+1)*W1 + (clip_index[:, 1]+1) + n[:]*W1*H1, :]

    res[:, :, :, :] = result.reshape((N1, W2, H2, -1)).transpose(0, 2, 1, 3)
    return res
```

思路

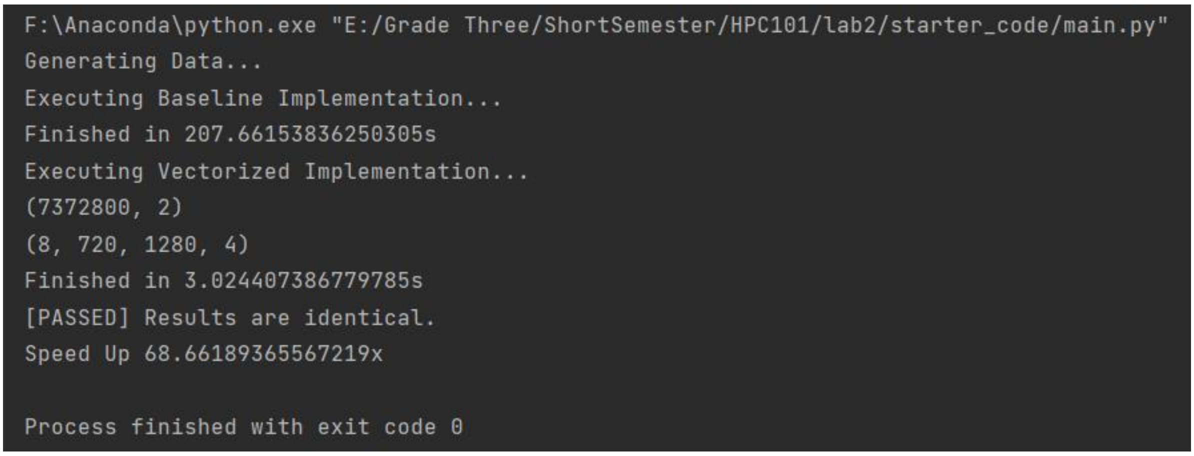
首先，先利用meshgrid生成三维的矩阵坐标xy，利用xy[0] xy[1]遍历b数组中XY轴的坐标，获得每个点的值，再对值做向下取整得到该位置对应的a数组中的值的坐标；然后将数组全部reshape成二维的，方便检索；之后做batchsize的向量化，生成值为00..00111...11...22..22..33..33..44..55..66..77形状的数组，代表不同的batchsize，在检索下标时乘上W1和H1就能区分不同的Batch size；最后通过reshape，转成正确的形状进行输出。

正确性和加速比

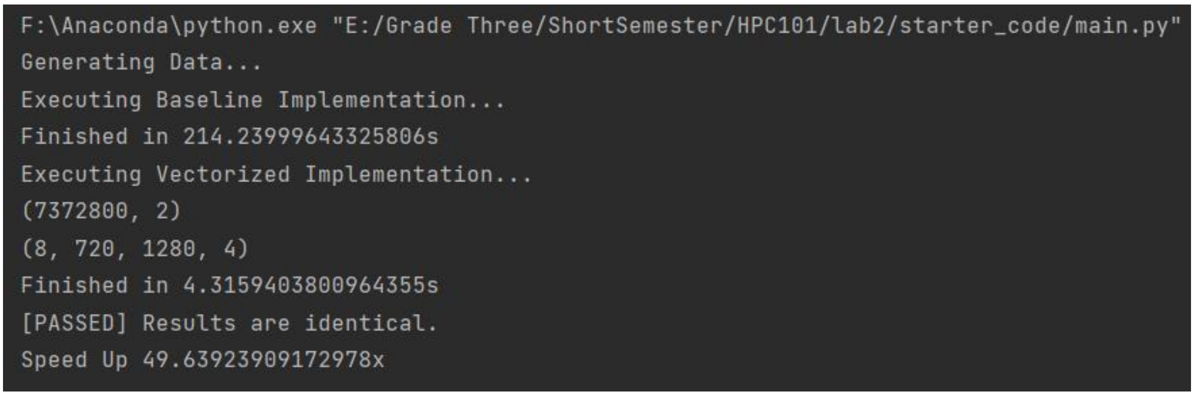
试验次数	加速比
1	68.66
2	49.64
3	39.59
4	47.50
5	49.94

平均加速比为51.066

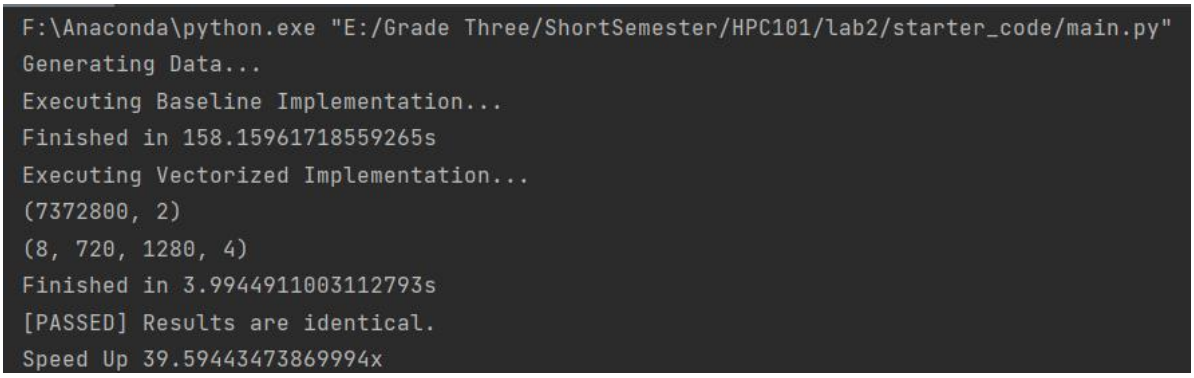
实验截图：



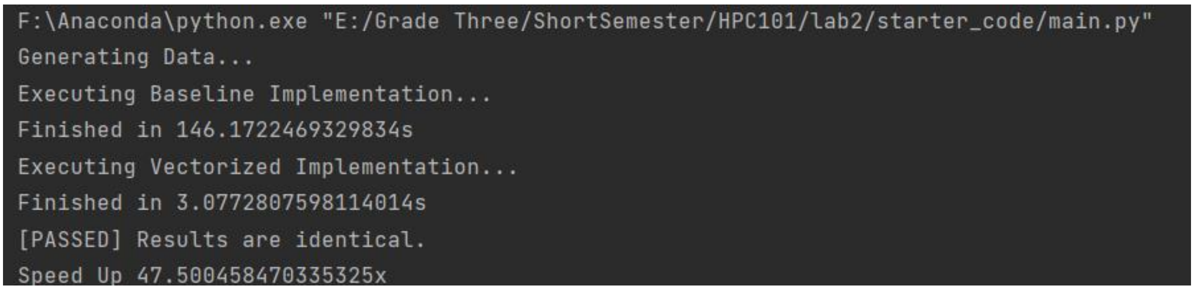
```
F:\Anaconda\python.exe "E:/Grade Three/ShortSemester/HPC101/lab2/starter_code/main.py"
Generating Data...
Executing Baseline Implementation...
Finished in 207.66153836250305s
Executing Vectorized Implementation...
(7372800, 2)
(8, 720, 1280, 4)
Finished in 3.024407386779785s
[PASSED] Results are identical.
Speed Up 68.66189365567219x

Process finished with exit code 0
```



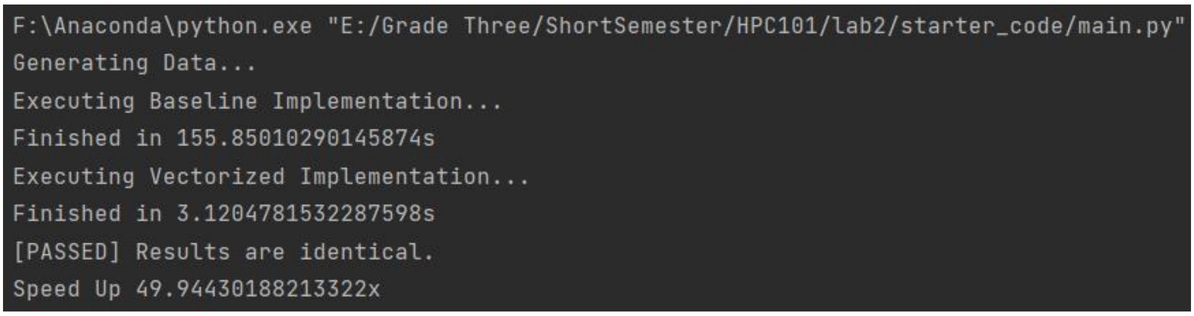
```
F:\Anaconda\python.exe "E:/Grade Three/ShortSemester/HPC101/lab2/starter_code/main.py"
Generating Data...
Executing Baseline Implementation...
Finished in 214.23999643325806s
Executing Vectorized Implementation...
(7372800, 2)
(8, 720, 1280, 4)
Finished in 4.3159403800964355s
[PASSED] Results are identical.
Speed Up 49.63923909172978x
```



```
F:\Anaconda\python.exe "E:/Grade Three/ShortSemester/HPC101/lab2/starter_code/main.py"
Generating Data...
Executing Baseline Implementation...
Finished in 158.15961718559265s
Executing Vectorized Implementation...
(7372800, 2)
(8, 720, 1280, 4)
Finished in 3.9944911003112793s
[PASSED] Results are identical.
Speed Up 39.59443473869994x
```



```
F:\Anaconda\python.exe "E:/Grade Three/ShortSemester/HPC101/lab2/starter_code/main.py"
Generating Data...
Executing Baseline Implementation...
Finished in 146.1722469329834s
Executing Vectorized Implementation...
Finished in 3.0772807598114014s
[PASSED] Results are identical.
Speed Up 47.500458470335325x
```



```
F:\Anaconda\python.exe "E:/Grade Three/ShortSemester/HPC101/lab2/starter_code/main.py"
Generating Data...
Executing Baseline Implementation...
Finished in 155.85010290145874s
Executing Vectorized Implementation...
Finished in 3.1204781532287598s
[PASSED] Results are identical.
Speed Up 49.94430188213322x
```