NumPy中的多维迭代器

《代码之美》　　第19章　　江涌

NumPy是Python的一个可选安装包，提供了一个功能强大的N维数组对象。NumPy提供了多种数组的数学化操作与结构化操作，使得Python能够很好地开发一些关键的并且要求运行速度很快的工程代码和科学代码。

NumPy中通过切片(slicing)的概念来实现快速结构化操作。记法为[start:stop:stride]，例如im2=im[8:2:-1,9:1:-3]。

按照slicing方式选取的新影像将与原始影像共享数据，不会生成一个副本，减少计算机资源的消耗。

关键挑战

经常需要遍历数组中的元素，在遍历中进行所需要的操作。

简单想法：用单层for循环处理一维，双层for循环处理两维……但当维数N是一个任意整数，怎么办？

递归：递归条件(recursive case)，基线条件(base case)

Copy_ND(a,b,N)　　//将N维数组b复制到N维数组a

递归实现：

```
if (N==0)
    copy memory from b to a
    return
for i=0 to size of first dimension of a and b
    ptr_b=b[i]
    Copy_ND(ptr_a,ptr_b,N-1)
    a[i]=ptr_a
```

递归算法在每次迭代中进行函数调用，容易产生速度很慢的代码；许多算法需要保存中间值用于后续的递归调用(求最大值)，这些值将被作为递归调用的参数传递，很难提供用于递归解决方案的简化工具。

因此，NumPy使用迭代来完成。迭代器(Iterator)是一种简化这些算法的抽象，包含了单个循环内遍历数组中所有元素的思想。迭代器的两个基本方法： hasnext 是否还有下一个元素;

next 返回下一个元素。

```
for x in iterobj:
    process(x)
```

数组的内存模型

邻接型数组：在内存中连续存放。

一个二维4*5数组

```
>>>p=[[1,2,3,4,5],
      [6,7,8,9,10],
      [11,12,13,14,15],
      [16,17,18,19,20]]
>>>from numpy import *
>>>pp=array(p)
>>>p1=pp[1:3,1:4]
```

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

跨度

数组中某一维的跨度(stride)：

沿着这一维，或者说数轴，从数组中的一个元素移动到下一个元素，需要跳过多少字节。

（跨度可以是负数）

pp: 第一维跨度4*5,第二维跨度4；

p1：第一维跨度4*5，第二维跨度4.

迭代器设计

迭代器循环伪码：

set up iterator

(including pointing the current value to the first value in the array)

while iterator not done:

process the current value

point the current value to the next value

设计分为三部分：

1. Moving to the next value
2. Termination
3. Setup

递进

确定按怎样的顺序来提取元素

NumPy中通过使用一组数字来模拟简单计数而实现。用N元整数组来表示当前位置，数组的形状$n_1 \times n_2 \times ... \times n_N$ (0,…,0)表示数组第一个元素,$(n_1 - 1, n_2 - 1, ..., n_N - 1)$ 表示数组的最后一个元素。

下一个元素的位置是将最后一个数字加1来得到。若第i个数字到达了n_i，那么这个数字将被设置为0，而第(i-1)个数字将增1 。比如对于$3 \times 2 \times 3$ 数组来说

$$(0,0,0)(0,0,1)(0,0,2)(0,1,0)(0,1,1)(0,1,2)(1,0,0)...(2,1,1)(2,1,2)$$

假设data是指向数组起始位置的指针，counter[N]是计数器数组，strides[N]是跨度值数组，那么下面的运算将把dataptr设置为指向数组当前值的第一个字节：

```
dataptr = (char *)data;
for (i=0; i<N; i++) dataptr += counter[i]*strides[i];
```

事实上可以在记录计数器的同时也记录指针，当计数器的第i个下标增1时，dataptr将会增加$strides[i]$；当第i个下标复位为0时，数组当前值的内存地址应该减去$(n_i-1)\times strides[i]$。

迭代器负责维护计数器与指向当前值的指针。

终止

判断迭代器何时完成及如何发出终止信号

1.附加标志变量到迭代器上，每次迭代中都进行判断，如果没有元素了就设置这个标志。

2.查找在第一维的计数器中从$n_i - 1$到0的跃迁点。

3.如果给定了数组的大小，只需记住将要进行的迭代次数。

NumPy中将迭代的总次数作为信息保存下来，以及保存一个到目前为止的迭代次数的动态计数器，当达到迭代的总次数时迭代终止。

构建

需要保存的信息：

- 整数计数器（初始设置为0）
- 下标计数器（初始设置为(0,0,…,0)）
- 判断是否基于简单的邻接内存，设置标志保存判断结果
- 为了加速回卷步骤，保存每一维
- $(n_i - 1)$，避免重复计算 $(n_i - 1) \times strides[i]$
- 跨度信息，维数信息，元素数量
- 保存系数$l_i (i = 1, ...N)$来简化整数计数器与下标计数器的转化
- 当前指针

数组中的每一项都可以用一个在0和 $n_1 \times ... \times n_N$ 之间的整数 k 或者下标计数器 $(k_1, ..., k_N)$ 来表示，这个关系可以被定义为

$$k = \sum_{i=1}^{N} k_i \left(\prod_{j=i+1}^{N} n_j \right)$$

或

$$l_1 = k$$

$$l_i = l_{i-1} \bmod \left(\prod_{j=i}^{N} n_j \right)$$

$$k_i = \left\lfloor \frac{l_i}{\prod_{j=i+1}^{N} n_j} \right\rfloor$$

coords[N]	下标计数器N维数组
dims_m1[N]	N维数组保存每一维 $n_i - 1$
strides[N]	N维数组，保存每维跨度
backstrides[N]	N维数组，回卷时需要移动的数量 $(n_i - 1) \times$ strides[i]
nd_m1	维数
dataptr	指向当前位置内存的指针
size	所有元素数量 $n_1 \times n_2 \times \ldots \times n_N$
index	整数计数器从0变到size-1
factors	计算一维下标和N维下标转换时辅助数组

记录迭代计数器

注意计数器的递进都是从最后一维增1开始，当某维大于$n_i - 1$时发生回卷，此时可能使其他维的下标也发生回卷。

于是可以从最后维开始向前循环。在当前维上判断当前下标是不是小于$n_i - 1$，如果是，则将下标位置加1，并且将当前维对应的strides[i]加到dataptr上，跳出循环；若第i维下标的计数器大于或等于$n_i - 1$，就重新设为0，且将dataptr减去该维对应的backstrides[i](回卷)，继续循环判断前一维。

C语言实现

```
for (i=it->nd_m1; i>=0; i--) {
    if (it->coords[i] < it->dims_m1[i]) {
        it->coords[i]++;
        it->dataptr += it->strides[i];
        break;
    }
    else {
        it->coords[i] = 0;
        it->dataptr -= it->backstrides[i];
    }
}
```

使用while语句

```
done = 0;
i = it->nd_ml;
while (!done || i>=0) {                          /*&&*/
    if (it->coords[i] < it->dims_m1[i]) {
       it->coords[i]++;
       it->dataptr += it->strides[i];
       done = 1; }
    else {
        it->coords[i] = 0;
        it->dataptr -= it->backstrides[i];}
    i--;
}
```

NumPy迭代器的结构

```
typedef struct {
    PyObject_HEAD
    int nd_m1;
    npy_intp index, size;
    npy_intp coords[NPY_MAXDIMS];
    npy_intp dims_m1[NPY_MAXDIMS];
    npy_intp strides[NPY_MAXDIMS];
    npy_intp backstrides[NPY_MAXDIMS];
    npy_intp factors[NPY_MAXDIMS];
    PyArrayObject *ao;
    char *dataptr;
    npy_bool contiguous;
} PyArrayIterObject;
```

指向构建迭代器的原始数组的指针

判断是否邻接数组

接口

在NumPy中

it=PyArray_IterNew(ao)

----------构建数组ao的迭代器

PyArray_ITER_NOTDONE(it)

----------判断迭代是否结束

PyArray_ITER_NEXT(it)

----------实现迭代的下一个位置

PyArray_ITER_DATA(it)

----------得到指向当前值第一个字节的指针

示例：计算N维数组中的最大值。(假设数组ao是double类型)

```
#include <float.h>
double *currval, maxval=-DBL_MAX;
PyArrayIterObject *it;
it = PyArray_IterNew(ao);
while (PyArray_ITER_NOTDONE(it)) {
    currval = (double *)PyArray_ITER_DATA(it);
    if (*currval > maxval) maxval = *currval;
    PyArray_ITER_NEXT(it);
}
```

用迭代器处理邻接数组也是很快的，但是更快的还是传统办法：

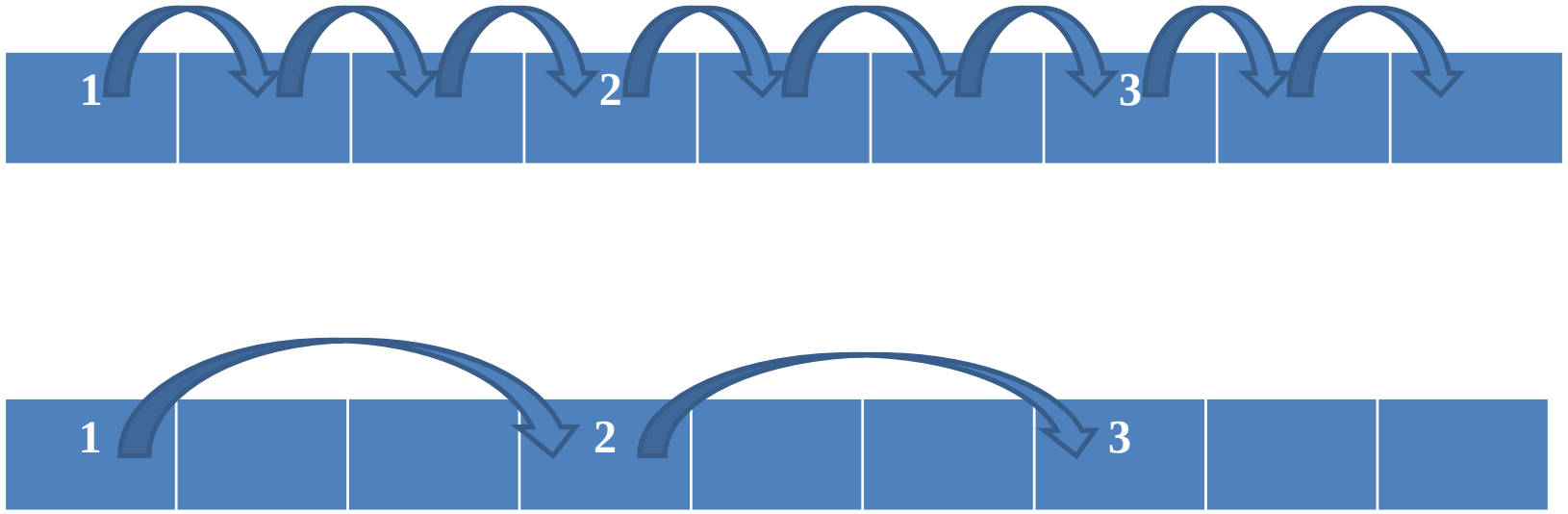
```
double *currval, maxval=-MAX_DOUBLE;
int size;
currval = (double *)PyArray_DATA(ao);
size = PyArray_SIZE(ao);
while (size--) {
    if (*currval > maxval) maxval = *currval;
    currval += 1;
}
```

迭代器的使用

排除某一维的迭代

当操作不涉及到某维时可以使迭代器跨过这维进行迭代，以获得速度提升。

例如数组a[3][3]，只想修改a(0,0),a(1,0),a(2,0)的值



通常是排除最后一维，NumPy的前身Numeric就是引入这种方法实现数学功能。

NumPy中对迭代器稍微改动

```
it=PyArray_IterAllButAxia(array,&dim)
```

dim为排除的维。当输入的维是-1时，将会自行选择最小非零跨度值的维。

另一种经常的选择就是排除有最大元素数量的维，这样将迭代次数降至最低。

实现：

1. 将迭代大小除以将要移除的维的长度；
2. 将被选择的维的元素数量设置为1:dims_m1[i]=0；
3. 将该维在backstrides中相应位置上的值设置为0；
4. 将邻接标志重新设置为0，因为现在要处理的数组在内存中将不会是邻接的。

在循环的每次迭代中，迭代器将指向数组所选择维的第一个元素。

迭代器的使用

多重迭代

数组加法需要多个迭代器，每个输入数组一个迭代器，并且输出数组一个迭代器。NumPy提供了一个多重迭代器，可以对多个迭代器同时处理。这种多迭代器经常用于自动处理NumPy的广播(broadcasting)功能。

广播能使一个(4,1)形状的数组加到一个(3)形状的数组上，得到形状(4,3)的数组，也能对一个(5,1,1)形状数组，一个(4,1)形状数组和一个(3)形状数组进行运算，得到一个(5,4,3)形状的数组。

```
>>>p=array([[1,2,3],
                 [4,5,6],
                 [7,8,9]])
>>>p1=[2,1,4]
>>>print p+p1
[[3,3,7],
  [6,6,10],
  [9,9,13]]
```

```
>>>a=array([[1],[2],[3],[4]])
>>>a1=[2,1,4]
>>>print a+a1
[[3,2,5],
  [4,3,6],
  [5,4,7],
  [6,5,8]]
```

The rules of broadcasting are:

• Arrays with fewer dimensions are treated as occupying the last dimensions of an array that has the full number of dimensions, so that all arrays have the same number of dimensions. The new, initial dimensions are filled in with 1s.

• The length of each dimension in the final broadcast shape is the greatest length of that dimension in any of the arrays.

• For each dimension, all inputs must either have the same number of elements as the broadcast result or a 1 as the number of elements.

• Arrays with a single element in a particular dimension act as if that element were virtually copied to all positions during the iteration. In effect, the element is “broadcast” to the additional positions.

修改迭代器实现广播功能：修改迭代器的形状为匹配广播的形状，用于广播维数的strides和backstrides被修改为0，这样这一维递进时，迭代器不会移动数据指针，无需复制内存。

```
PyObject *multi;
PyObject *in1, *in2;
double *i1p, *i2p, *op;
/* get in1 and in2 (assumed to be arrays of NPY_DOUBLE) */
/* first argument is the number of input arrays; the next
   (variable
   number of) arguments are the array objects */
multi = PyArray_MultiNew(2, in1, in2);
/* construct output array */
out = PyArray_SimpleNew(PyArray_MultiIter_NDI
                        PyArray_MultiIter_DIMS(multi),
                        NPY_DOUBLE);
op = PyArray_DATA(out);
```

多重迭代器

输出数组

数组当前数据指向

```
while(PyArray_MultiIter_NOTDONE(multi)) {
    /* get (pointers to) the current value in each array */
    i1p = PyArray_MultiIter_DATA(multi, 0);
    i2p = PyArray_MultiIter_DATA(multi, 1);
    /* perform the operation for this element */
    *op = *ip1 + *ip2
    op += 1;  /* Advance output array pointer */
    /* Advance all the input iterators */
    PyArray_MultiIter_NEXT(multi);
}
```

总结

迭代器是一个非常漂亮的抽象。NumPy的迭代器为Python的数组运算提供了很强大的灵活工具，并且无需考虑数组在内存中是否邻接，以切片的概念减少内存的消耗。它的优化循环与广播机制都是其漂亮的发光点。