# 15.11 用Cython写高性能的数组操作¶

## 问题¶

你要写高性能的操作来自NumPy之类的数组计算函数。 你已经知道了Cython这样的工具会让它变得简单，但是并不确定该怎样去做。

## 解决方案¶

作为一个例子，下面的代码演示了一个Cython函数，用来修整一个简单的一维双精度浮点数数组中元素的值。

```
# sample.pyx (Cython)

cimport cython

@cython.boundscheck(False)
@cython.wraparound(False)
cpdef clip(double[:] a, double min, double max, double[:] out):
    '''
    Clip the values in a to be between min and max. Result in out
    '''
    if min > max:
        raise ValueError("min must be <= max")
    if a.shape[0] != out.shape[0]:
        raise ValueError("input and output arrays must be the same size")
    for i in range(a.shape[0]):
        if a[i] < min:
            out[i] = min
        elif a[i] > max:
            out[i] = max
        else:
            out[i] = a[i]
```

要编译和构建这个扩展，你需要一个像下面这样的 `setup.py` 文件 （使用 `python3 setup.py build_ext --inplace` 来构建它）：

```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Distutils import build_ext

ext_modules = [
    Extension('sample',
              ['sample.pyx'])
]

setup(
  name = 'Sample app',
  cmdclass = {'build_ext': build_ext},
  ext_modules = ext_modules
)
```

你会发现结果函数确实对数组进行的修正，并且可以适用于多种类型的数组对象。例如：

```
>>> # array module example
```

```
>>> import sample
>>> import array
>>> a = array.array('d',[1,-3,4,7,2,0])
>>> a

array('d', [1.0, -3.0, 4.0, 7.0, 2.0, 0.0])
>>> sample.clip(a,1,4,a)
>>> a
array('d', [1.0, 1.0, 4.0, 4.0, 2.0, 1.0])

>>> # numpy example
>>> import numpy
>>> b = numpy.random.uniform(-10,10,size=1000000)
>>> b
array([-9.55546017,  7.45599334,  0.69248932, ...,  0.69583148,
       -3.86290931,  2.37266888])
>>> c = numpy.zeros_like(b)
>>> c
array([ 0.,  0.,  0., ...,  0.,  0.,  0.])
>>> sample.clip(b,-5,5,c)
>>> c
array([-5.        ,  5.        ,  0.69248932, ...,  0.69583148,
       -3.86290931,  2.37266888])
>>> min(c)
-5.0
>>> max(c)
5.0
>>>
```

你还会发现运行生成结果非常的快。 下面我们将本例和numpy中的已存在的 `clip()` 函数做一个性能对比：

```
>>> timeit('numpy.clip(b,-5,5,c)','from __main__ import b,c,numpy',number=1000)
8.093049556000551
>>> timeit('sample.clip(b,-5,5,c)','from __main__ import b,c,sample',
...        number=1000)
3.760528204000366
>>>
```

正如你看到的，它要快很多——这是一个很有趣的结果，因为NumPy版本的核心代码还是用C语言写的。

## 讨论¶

本节利用了Cython类型的内存视图，极大的简化了数组的操作。 `cpdef clip()` 声明了 `clip()` 同时为C级别函数以及Python级别函数。 在Cython中，这个是很重要的，因为它表示此函数调用要比其他Cython函数更加高效 （比如你想在另外一个不同的Cython函数中调用clip()）。

类型参数 `double[:] a` 和 `double[:] out` 声明这些参数为一维的双精度数组。 作为输入，它们会访问任何实现了内存视图接口的数组对象，这个在PEP 3118有详细定义。 包括了NumPy中的数组和内置的array库。

当你编写生成结果为数组的代码时，你应该遵循上面示例那样设置一个输出参数。 它会将创建输出数组的责任给调用者，不需要知道你操作的数组的具体细节 （它仅仅假设数组已经准备好了，只需要做一些小的检查比如确保数组大小是正确的）。 在像NumPy之类的库中，使用 `numpy.zeros()` 或 `numpy.zeros_like()` 创建输出数组相对而言比较容易。另外，要创建未初始化数组， 你可以使用 `numpy.empty()` 或 `numpy.empty_like()` . 如果你想覆盖数组内容作为结果的话选择这两个会比较快点。

在你的函数实现中，你只需要简单的通过下标运算和数组查找（比如a[i],out[i]等）来编写代码操作数组。 Cython会负责为你生成高效的代码。

`clip()` 定义之前的两个装饰器可以优化下性能。 `@cython.boundscheck(False)` 省去了所有的数组越界检查， 当你知道下标访问不会越界的时候可以使用它。 `@cython.wraparound(False)` 消除了相对数组尾部的负数下标的处理（类似Python列表）。 引入这两个装饰器可以极大的提升性能（测试这个例子的时候大概快了2.5倍）。

任何时候处理数组时，研究并改善底层算法同样可以极大的提示性能。 例如，考虑对 `clip()` 函数的如下修正，使用条件表达式：

```
@cython.boundscheck(False)
@cython.wraparound(False)
cpdef clip(double[:] a, double min, double max, double[:] out):
   if min > max:
      raise ValueError("min must be <= max")
   if a.shape[0] != out.shape[0]:
      raise ValueError("input and output arrays must be the same size")
   for i in range(a.shape[0]):
      out[i] = (a[i] if a[i] < max else max) if a[i] > min else min
```

实际测试结果是，这个版本的代码运行速度要快50%以上（2.44秒对比之前使用 `timeit()` 测试的3.76秒）。

到这里为止，你可能想知道这种代码怎么能跟手写C语言PK呢？ 例如，你可能写了如下的C函数并使用前面几节的技术来手写扩展：

```
void clip(double *a, int n, double min, double max, double *out) {
 double x;
 for (; n >= 0; n--, a++, out++) {
   x = *a;

   *out = x > max ? max : (x < min ? min : x);
 }
}
```

我们没有展示这个的扩展代码，但是试验之后，我们发现一个手写C扩展要比使用Cython版本的慢了大概10%。 最底下的一行比你想象的运行的快很多。

你可以对实例代码构建多个扩展。 对于某些数组操作，最好要释放GIL，这样多个线程能并行运行。 要这样做的话，需要修改代码，使用 `with nogil:` 语句：

```
@cython.boundscheck(False)
@cython.wraparound(False)
cpdef clip(double[:] a, double min, double max, double[:] out):
   if min > max:
      raise ValueError("min must be <= max")
   if a.shape[0] != out.shape[0]:
      raise ValueError("input and output arrays must be the same size")
   with nogil:
      for i in range(a.shape[0]):
         out[i] = (a[i] if a[i] < max else max) if a[i] > min else min
```

如果你想写一个操作二维数组的版本，下面是可以参考下：

```
@cython.boundscheck(False)
@cython.wraparound(False)
cpdef clip2d(double[:,:] a, double min, double max, double[:,:] out):
   if min > max:
      raise ValueError("min must be <= max")
   for n in range(a.ndim):
      if a.shape[n] != out.shape[n]:
         raise TypeError("a and out have different shapes")
   for i in range(a.shape[0]):
      for j in range(a.shape[1]):
         if a[i,j] < min:
            out[i,j] = min
         elif a[i,j] > max:
            out[i,j] = max
         else:
            out[i,j] = a[i,j]
```

希望读者不要忘了本节所有代码都不会绑定到某个特定数组库（比如NumPy）上面。 这样代码就更有灵活性。 不过，要注意的是如果处理数组要涉及到多维数组、切片、偏移和其他因素的时候情况会变得复杂起来。 这些内容已经超出本节范围，更多信息请参考 PEP 3118 ， 同时 Cython文档中关于“类型内存视图” 篇也值得一读。