Python 高性能计算库——Numba

摘要：

在计算能力为王的时代，具有高性能计算的库正在被大家应用于深度学习。例如：Numpy，本文介绍了一个新的 Python 库——Numba， 在计算性能方面，它比 Numpy 表现的更好。

最近我在观看一些 SciPy2017 会议的视频，偶然发现关于 Numba 的来历--讲述了那些 C++ 横行者因为对 Gil Forsyth 和 Lorena Barba 失去信心而编写的一个库。虽然本人觉得这个想法有些不妥，但我真的很喜欢他们所教授的知识。因为我发现自己正在受益于这个库，并且从 Python 代码中获得了令人难以置信的表现，所以我觉得应该要写一些关于 Numba 库的介绍性文章，也可能会在将来添加一系列小的更多类似教程的文章。

1.那么到底什么是 Numba？

Numba 是一个库，可以在运行时将 Python 代码编译为本地机器指令，而不会强制大幅度的改变普通的 Python 代码（稍后再做说明）。翻译/魔术是使用 LLVM 编译器完成的，该编译器是相当活跃的开源社区开发的。

Numba 最初是由 Continuum Analytics 内部开发，此公司也开发了著名的 Anaconda，但现在它是开源的。核心应用领域是 math-heavy（密集数学？重型数学？）和 array-oriented（面向数组）的功能，它们在本地 Python 中相当缓慢。想象一下，在 Python 中编写一个模块，必须一个元素接着一个元素的循环遍历一个非常大的数组来执行一些计算，而不能使用向量操作来重写。这是很不好的主意，是吧？所以“通常”这类库函数是用 C / C ++或 Fortran 编写的，编译后，在 Python 中作为外部库使用。Numba 这类函数也可以写在普通的 Python 模块中，而且运行速度的差别正在逐渐缩小。

2.怎么才能 get 到 Numba 呢？

安装 Numba 的推荐方法是使用 conda 包管理

```
conda install numba
```

你也可以用 pip 来安装 Numba，但是最新版本的发布才一天之久。但是，只要你能够使用 conda，我会推荐使用它，因为它能够为你安装例如 CUDA 工具包，也许你想让你的 Python 代码 GPU 就绪（当然，这也是有可能的！）。

3. 如何使用 Numba 呢？

使用它的要求不多。基本上，你写一个自己的“普通”的 Python 函数，然后给函数定义添加一个装饰（如果你不是很熟悉装饰器，读一下关于 this 或 that）。你可以使用不同类型的装饰器，但`@jit`可能是刚开始的选择之一。其他装饰器可用于例如创建 numpy 通用功能`@vectorize`或编写将在 CUDA GPU 上执行的代码`@cuda`。我不会在这篇文章中介绍这些装饰。现在，让我们来看看基本的步骤。他们提供的代码示例是 2d 数组的求和函数，以下是代码：

```
from numba import jit
from numpy import arange
# jit decorator tells Numba to compile this function.# The argument types will be inferred by Numba when function is called.
@jit
def sum2d(arr):
    M, N = arr.shape
    result = 0.0
    for i in range(M):
        for j in range(N):
            result += arr[i,j]
    return result
a = arange(9).reshape(3,3)
print(sum2d(a))
```

正如你所看到的，Numba 装饰器被添加到函数定义中，并且 voilá这个函数将运行得很快。但是，这里带来了很有趣的注意事项：你只能使用 Numpy 和标准库里的函数来加快 Numba 速度，甚至不需要开了他们所有的特性。他们有一个相当好的文档（参考资料），

列出了所有支持的内容。见 here 是所支持 Python 的功能和 here 是所支持的 Numpy 功能。现在支持的功能可能还不太多，但我想告诉你，这就够了！请记住，Numba 不是要加快你的数据库查询或如何强化图像处理功能。他们的目标是加快面向数组的计算，我们可以使用它们库中提供的函数来解决。

4.示例和速度比较

熟练的 Python 用户永远不会使用上述代码实现 sum 功能，而是调用 numpy.sum。相反，我将向你介绍另外一个例子，为了更好地理解这个例子，也许刚开始是一个小的背景故事（如果你对这个例子的背景不感兴趣，你可以直接跳过然后直接去看代码）。

从我所学习的知识来看，我会认为自己是一个水文学家，我做的很多的一件事是模拟降雨径流过程。简单点来说：通过时间序列数据，例如雨量和空气温度，然后尝试创建模型来判断一条河流的水流量有多少。这在外行看来是非常复杂。但，对于我们来说，很简单。我们通常使用的模块迭代输入数组，并且对于每个时间步长，我们会更新一些模块内部的状态（例如，模拟土壤水分，积雪或拦截水中的树木）。在每个时间段结束时，计算水流量，这不仅取决于在同一时间步长下的雨，而且也取决于在内部模型状态（或储存）。在这种情况下，我们就需要考虑以前时间步长的状态和输出。那么你可能会看到这个问题：我们必须一段时间接一段时间的计算整个流程，而对于解决这种问题 Python 本来就是很慢的！这就是为什么大多数模块都是在 Fortran 或 C/C ++中实现的。如前所述：Python 在对于这种面向数组的计算来说是慢的。但是 Numba 允许我们在 Python 中做同样的事情，而且没有太多的性能损失。我认为至少对于模型的理解和发展，这可能会很方便。（所以我最近创建了一个名为“RRMPG ”的项目——降雨径流建模游乐场）。

Okay，现在我们来看看我们 get 到了什么。我们将使用最简单的模块之一，由 MB Fiering 在 1967 年出于教育目的开发的 ABC 模型，并将 Python 代码的速度与 Numba 优化后 Python 代码和 Fortran 实现进行比较。请注意这个模型不是我们在现实中使用的（正如名称所示），但是我认为这可能是一个不错的想法来举例。

ABC 模块是一个三个参数模块（a，b，c，习惯性命名），它只接收下雨量为输入，只有一个存储。土壤水分蒸发蒸腾损失总量（参数 b），另一部分通过土壤渗透到地下水储

存（参数 a），最后一个参数 c 代表地下水总量，离开地下变成河流。Python 中的代码，使用 Numpy 数组可能会像如下所示：

```
import numpy as np
def abc_model_py(a, b, c, rain):
    # initialize array for the stream discharge of each time step
    outflow = np.zeros((rain.size), dtype=np.float64)
    # placeholder, in which we save the storage content of the previous and
    # current timestep
    state_in = 0
    state_out = 0
    for i in range(rain.size):
        # Update the storage
        state_out = (1 - c) * state_in + a * rain[i]
        # Calculate the stream discharge
        outflow[i] = (1 - a - b) * rain[i] + c * state_out
        state_in = state_out
    return outflow
```

接下来我们使用 Numba 来实现相同的功能。

```
@jit
def abc_model_numba(a, b, c, rain):
    outflow = np.zeros((rain.size), dtype=np.float64)
    state_in = 0
    state_out = 0
    for i in range(rain.size):
        state_out = (1 - c) * state_in + a * rain[i]
        outflow[i] = (1 - a - b) * rain[i] + c * state_out
        state_in = state_out
    return outflow
```

我用随机数字作为输入来运行这些模块，这只是为了比较计算时间，而且也比较了针对 fortran 实现的时间（详见 here）。我们来看看数字：

```
py_time = %timeit -r 5 -n 10 -o abc_model_py(0.2, 0.6, 0.1, rain)
>> 6.75 s ± 11.6 ms per loop (mean ± std. dev. of 5 runs, 10 loops each)
# Measure the execution time of the Numba implementation
numba_time = %timeit -r 5 -n 10 -o abc_model_numba(0.2, 0.6, 0.1, rain)
>> 30.6 ms ± 498 µs per loop (mean ± std. dev. of 5 runs, 10 loops each)
# Measure the execution time of the Fortran implementation
fortran_time = %timeit -r 5 -n 10 -o abc_model_fortran(0.2, 0.6, 0.1, rain)
>> 31.9 ms ± 757 µs per loop (mean ± std. dev. of 5 runs, 10 loops each)
# Compare the pure Python vs Numba optimizea time
py_time.best / numba_time.best
>> 222.1521754580626
# Compare the time of the fastes numba ana fortran run
numba_time.best / fortran_time.best
>> 0.9627960721576471
```

通过添加一个装饰器，我们的计算速度比纯 Python 代码快 222 倍，甚至比 Fortran 也快很多。在计算能力决定未来的时代，Numba 一定会被更多人接受。

以上就是我的介绍，希望有人现在有动力去看看 Numba 库。我想在将来我会编写一系列小的 Numba 文章/教程，并提供更多的技术信息，让更多的人使用 Numba 库。而本文仅作为一个开始。