利用 ctypes 并行计算 Pi

卢艳峰

3321051417

摘要

圆周率（Pi）是圆的周长与直径的比值，是一个非常重要的常数。Python 是目前最流行的语言，但由于全局解释器锁（global interpreter lock, GIL）的存在，不能够实现在多处理器上的并行性。可以利用 Python 的标准库 ctypes 调用共享库，释放 GIL，实现并行性。我们利用莱布尼茨公式计算圆周率，并且对比了纯 Python、 Python threading、ctypes 和纯 C 语言四种实现方法消耗的时间。发现通过 ctypes 可以实现并行性计算，接近 C 语言的计算效率，比纯 Python 和 Python threading 快两个数量级。并且发现 threading 不能利用多核心计算机的计算资源。论文的源代码可以从 https://github.com/LuYF-Lemon-love/SimpleProject/tree/main/ParallelComputePI 上获得。

关键字： Python ctypes C语言 共享库 线程 GIL Pi

介绍

Cpython 解释器使用了全局解释器锁（global interpreter lock, GIL）[1]，它确保同一时刻只有一个线程在执行 Python bytecode。此机制通过设置对象模型（包括 dict 等重要内置类型）针对并发访问的隐式安全简化了 CPython 实现。给整个解释器加锁使得解释器多线程运行更方便，其代价则是牺牲了在多处理器上的并行性。

不过，某些标准库或第三方库的扩展模块被设计为在执行计算密集型任务时释放 GIL。一般情况下，都是利用它们为 Python 实现 C/C++ 的扩展。常用的有标准 CPython 扩展[2]、PyBind11[3]、Cython[4]、HPy[5]、mypyc[6]、cffi、SWIG、Boost.Python[7]、cppyy[8] 和 ctypes[9]。其中，Python 的 C 扩展一般推荐使用 Cython，C++ 扩展一般使用 PyBind11[10]。但是 ctypes 由于是标准库，所以使用起来较简单。

ctypes 提供了与 C 兼容的数据类型，并允许调用 DLL 或共享库中的函数。可以使用该模块以纯 Python 形式对这些库进行封装。C 语言能够调用可移植操作系统接口（Portable Operating System Interface, POSIX）[11] 的 libpthread.so 动态链接库实现多线程。因此，Python 可以利用 ctypes 调用 C 程序实现多线程。

threading[12] 是 Python 的多线程库，由于存在 GIL，同一时刻只有一个线程可以执行 Python 代码，仅仅只能优化多个 I/O 密集型任务。

我们利用莱布尼茨公式计算圆周率，并对比了纯 Python、 Python threading、ctypes 和纯 C 语言四种实现方法消耗的时间。发现通过 ctypes 可以实现并行性计算，接近 C 语言的计算效率，比纯 Python 和 Python threading 快两个数量级。并且发现 threading 不能利用多核心计算机的计算资源。

论文的源代码可以从 https://github.com/LuYF-Lemon-love/SimpleProject/tree/main/ParallelComputePI 上获得。

方法

环境

操作系统：Ubuntu 20.04.4 LTS，64 位

处理器：Intel® Core™ i5-7500 CPU @ 3.40GHz × 4

内存：7.7 GiB

Python: 3.9.7

GCC: 9.4.0

算法

常用计算 Pi 的方法如下：

1. 蒙特卡罗法[13]

这种方法是一种基于随机数的算法，通过计算落在单位圆内的点数（M）与落在正方形内的点数（N）的比值求 Pi。该方法的精度与投入点的个数有关，投入点的数量越多，精度越高。公式如下：

$$\frac{M}{N} = \frac{piR^2}{(2R)^2} \Rightarrow pi = 4 * \frac{M}{N}$$

2. 数学公式（级数）[13]

$$\sum_{1}^{\infty}(\frac{1}{n^2}) = \frac{pi^2}{6}$$

3. 莱布尼茨公式[14]

$$\frac{pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots$$

我们最终选择方法3-莱布尼茨公式计算 Pi。并行计算 Pi 时，将莱布尼茨公式的项数平均分配给每个线程，然后再将各个线程的计算结果求和得到 Pi。

结果

我们实现了纯 Python、 Python threading、ctypes 和纯 C 语言四个版本。每个结果都是运行五次的均值。

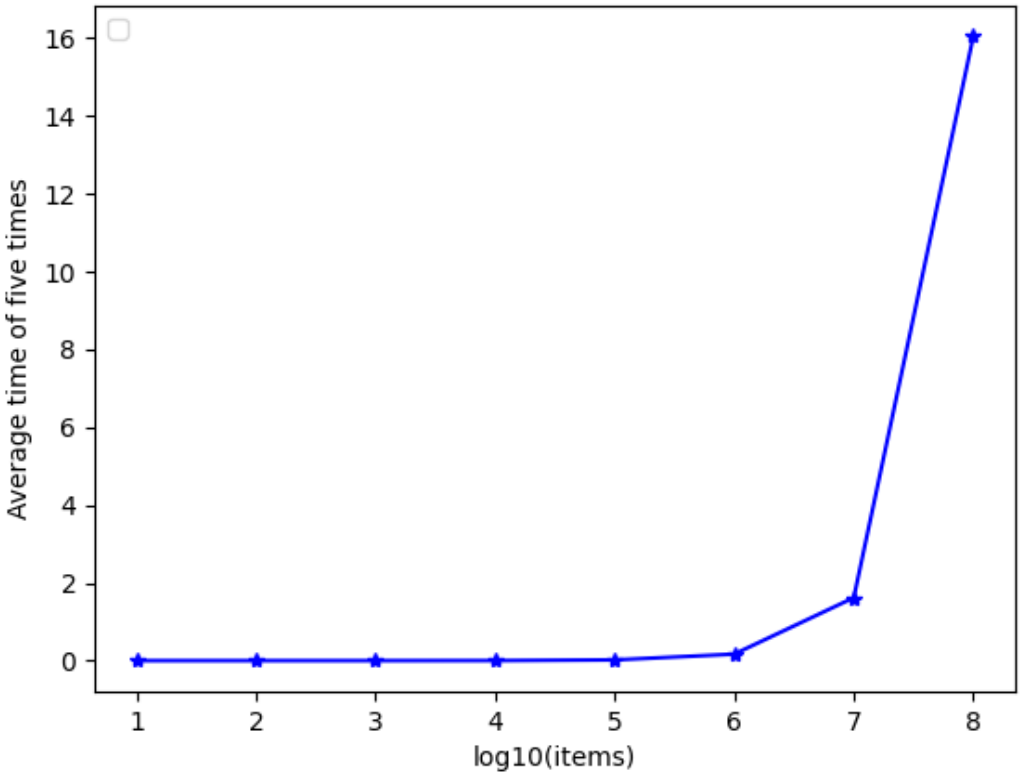

图 1：纯 Python 版本的运行结果。

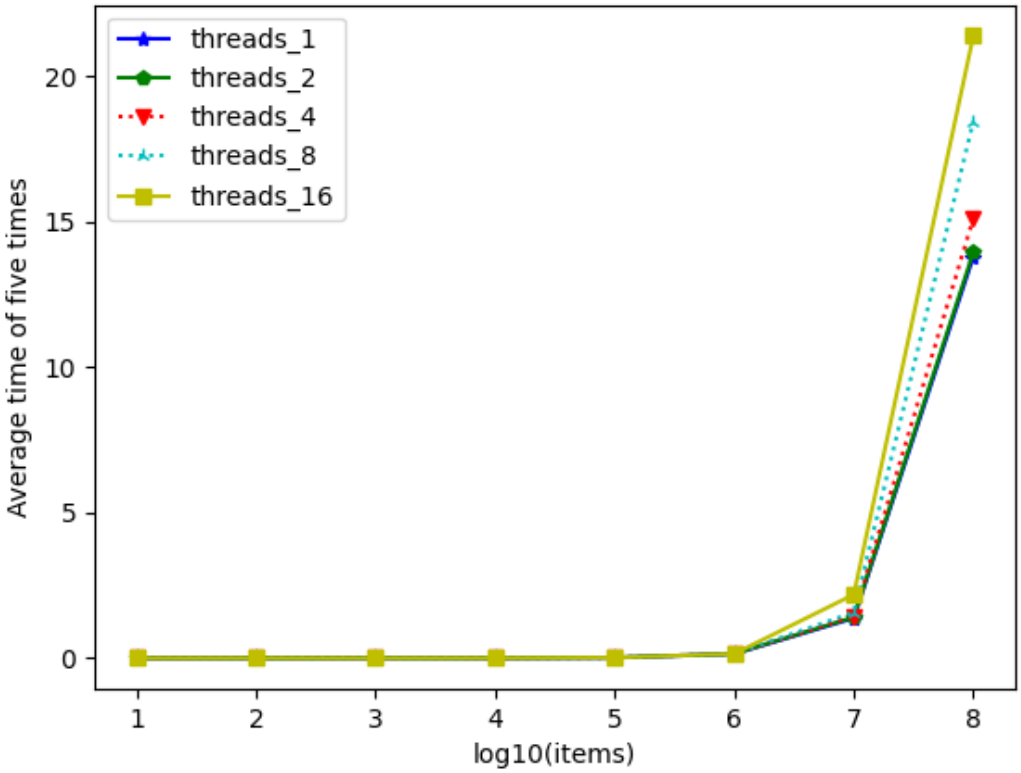

图 2：Python threading 版本的运行结果。

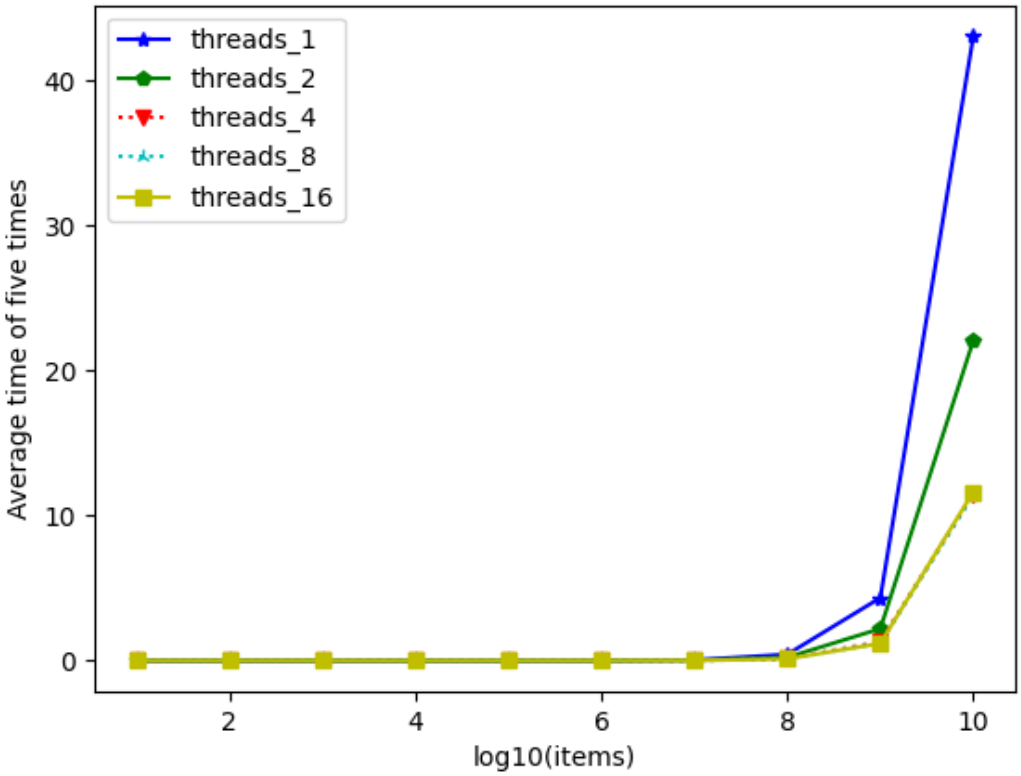

图 3：ctypes 版本的运行结果。

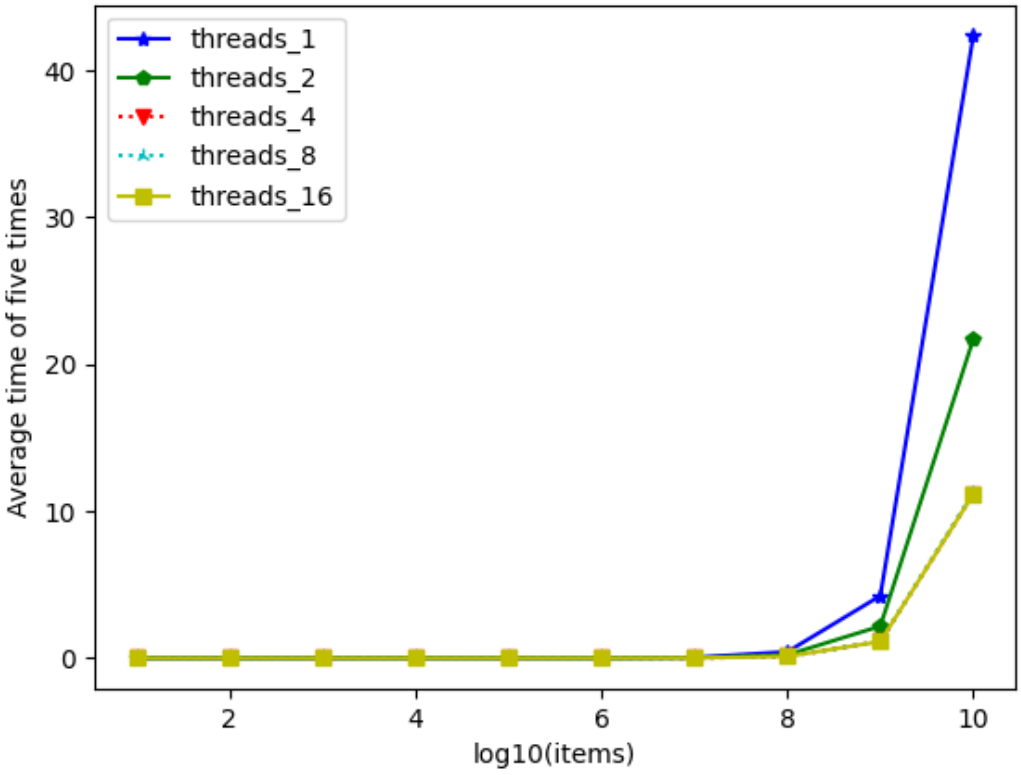

图 4：纯 C 语言版本的运行结果。

图 1和图 2的结果显示，在计算密集型任务中，Python 自带的 threading 库不会产生加速效果，甚至由于要创建线程类，产生额外的消耗资源，导致消耗时间增加。

图 3和图 4的结果显示，ctypes 版本消耗的时间和纯 C 版本大致相同，可见 ctypes 标准库的效率非常高。由于我们计算机的处理器具有 4 个核心，所以当计算密集的线程数多于核心数=4时，不会产生加速效果。

图 1和图 3的结果显示，当 ctypes 版本的线程数大于等于 4，纯 Python 版本的消耗的时间是 ctypes 版本的 100 倍，可以发现 C 语言的高效。

讨论

Python 可以通过 ctypes 调用 C 语言编译的共享库实现并行计算，效率无限接近 C 语言程序，远远高于纯 Python 程序。也发现了 threading 不能利用多核心计算机的计算资源。

如果进行计算密集型任务，请将线程数设置为计算机的核心数，这样效率最高。

由于 C++ 比 C 更富有表现力，所以我们将在未来利用 PyBind11 实现 Python 和 C++11 的混合编程。

参考

1. global interpreter lock -- 全局解释器锁 ↩
2. Python/C API Reference Manual ↩
3. PyBind11 ↩
4. Cython ↩
5. HPy ↩
6. mypyc ↩
7. Boost.Python ↩
8. cppyy ↩
9. ctypes --- Python 的外部函数库 ↩
10. 创建适用于 Python 的 C++ 扩展 ↩
11. 可移植操作系统接口 ↩
12. threading --- 基于线程的并行 ↩
13. 计算PI的几种方法 ↩ ↩
14. 并行计算多线程幂级数计算pi ↩