Research group
Discrete Geometries of Mathematics and Physics
www.mathischeap.com/dgmp

Discrete Geometries of Mathematics and Physics
创新实验

实验指导书

Python并行编程基础

Yi Zhang (张仪)
: www.mathischeap.com
@: zhangyi_aero@hotmail.com
: https://github.com/mathischeap

1 实验基本信息

1.1 实验名称

Python并行编程基础

1.2 实验指导老师联系方式

张仪(www.mathischeap.com), 数学与计算科学学院，花江慧谷4#-408A

- 手机： 199 7417 1867
- email: zhangyi_aero@hotmail.com
- QQ: 360 788 903

1.3 所属课程名称

创新实验

2 实验的目的、意义、要求

2.1 实验目的

掌握能力：“能够使用Python进行基础的并行编程从而实现并行运算。”

2.2 实验意义

- 学习理解什么是并行运算以及并行运算的重要性。
- 学会如何使用Python进行基础的并行编程。
- 从实践中探索并行编程带来的计算速度提升。

2.3 实验要求

- 具备基础Python编程（包含阅读文档、自行查阅开源函数使用方法等）能力。
- 完成实验报告并发回你的程序。你可以通过查询网络（或询问chatgpt等）搜集信息，但不允许从网络直接复制或要求chatgpt生成代码或实验报告等。

2.4 实验主页

你可以访问`https://mathischeap.com/contents/teaching/innoexp/basic_parallel_programming_with_python/main#course-innoexp-ppp-basic`查看有关本实验的所有信息。

3 实验步骤

3.1 实验背景

现代计算机处理器都向着多核发展。我们的个人计算机往往都已经是4核、6核、8核甚至更高。以因特尔公司生产的面向个人用户的消费级处理器Core-14900k为例，它拥有24核心32线程。AMD公司的最强消费级CPU，Ryzen R9 9950x也拥有16核心32线程。而AMD公司最新发布的面向企业用户的服务器CPU EPYC 9965更是拥有192核心384线程的超大规模。在拥有如此多核心的情况下，“如何使用好它们、让它们能够彼此协同工作以降低运算时间”显得越发重要。如果无法做到这一点，那么即便你拥有再好的硬件条件，你也无法充分利用它们。这将造成巨大的资金、算力和时间浪费，如图1所示。

图1是一台高性能服务器在某个时刻的CPU使用情况。这台服务器显然具备很强的硬件能力：它使用了拥有64个逻辑处理器的因特尔志强金牌（Intel Xeon Gold）系列CPU。而且，显然它正在执行一个繁重的计算任务。这使得它的第2个逻辑处理器处于满负荷运行状态。然而，对于第2个逻辑处理器面临的巨大压力，其他63个处理器显得漠不关心、有“摸鱼”的嫌疑。显然，这不是我们想要看到的。出现这种情况很有可能是因为程序的问题：它没有进行并行编程。使得无论任务多么繁重，它都只能调用一个核心按部就班地依次完成繁重任务中的各个子任务。试想一下，如果我们可以把剩下63个处理器都用起来，那么原来需要1小时才能得到结果的计算任务可以（在理想情况下）1分钟内完成，这将极大地提高我们的工作效率。这种潜力毫无疑问是十分诱人的，也是我们学习**并行编程**[1]的最重要原因。

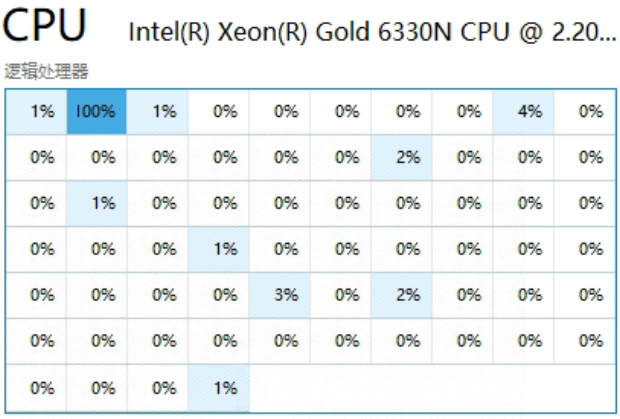

图 1: “一核有难，多核围观。”

3.2 使用工具: Python [2, 3]

当前流行的编程语言高达几百种，从中选出适合我们的、功能强大的且短时间内不会被淘汰的编程语言并非易事。不过，用别人的选择作为参考永远都不会是一件坏事。图2是一个全球编程语言流行程度的排名。

Oct 2024	Oct 2023	Change	Programming Language	Ratings	Change
1	1		Python	21.90%	+7.08%
2	3	↑	C++	11.60%	+0.93%
3	4	↑	Java	10.51%	+1.59%
4	2	↓	C	8.38%	-3.70%
5	5		C#	5.62%	-2.09%
6	6		JavaScript	3.54%	+0.64%
7	7		Visual Basic	2.35%	+0.22%
8	11	↑	Go	2.02%	+0.65%
9	16	⇈	Fortran	1.80%	+0.78%
10	13	↑	Delphi/Object Pascal	1.68%	+0.38%
11	9	↓	SQL	1.64%	-0.15%
12	14	↑	MATLAB	1.48%	+0.22%
13	20	⇈	Rust	1.45%	+0.53%
14	12	↓	Scratch	1.41%	+0.05%
15	8	⇊	PHP	1.21%	-0.69%

图 2: 全球编程语言流行程度实时排名。

从图2可以看到，排名第一的Python的流行程度在大幅领先的基础上又取得了巨大的提升（7%），达到了惊人的21.90%。这与Python的开源、易用性、功能多样性、跨平台应用能力有密不可分的关系。可以这么说，如果你在二十一世纪只能学习一个编程语言，那么Python一定是你的第一选择。因为，对于某个领域，可能存在一门比Python好的语言。但是，在其他很多领域，Python一定会

比它更好。或者说，你能够认为Python样样不精通，但它样样都行、是一个能力均衡的全能型选手。然而，即便Python的流行程度如此之大，它还是广受诟病。而针对它的负面讨论几乎都是集中在它的计算速度上。有学者做过测试，运行同样的算法，Python的速度是C/C++的$\frac{1}{70}$左右，而能耗是后者的70倍上下。计算速度偏慢是Python公认的缺点。也正是这个缺点，更加彰显出了并行运算对Python的重要性。如果我们能够使用Python执行较好的并行运算，从而充分利用手中的算力资源，这能够在很大程度上弥补其计算速度慢的缺陷。故此，**我们有了充足的理由去学习如何使用Python进行并行编程。**

3.3 具体实验过程

用Python实现并行计算有多种途径。有些途径较为基础，其效果有限、使用范围较小、且自由度较低但逻辑较为清楚、需要调用的函数简单、使用的语法易于掌握等等。我们称这些并行编程途径为**基于Python的基础并行编程方法**。另一些进阶的并行编程方法，相对于基础方法，其优劣势发生互换。而本实验着重于基础的并行编程方法。

具体而言，通过本实验，我们将一起执行以下实验步骤。

3.3.1 实验步骤一：认识Python的两个基础并行运算库

Python考虑到了并行运算的需求。所以随着Python的发展，两个基础的并行运算库被添加到了Python中去，它们是

- threading
- multiprocessing

各位同学可以打开自己的电脑，在Python内运行命令:

```
>>> import threading
>>> import multiprocessing
>>> print(threading)
>>> print(multiprocessing)
```

我们将看到threading和multiprocessing库被正常安装了。通过查阅网络资料、文档，我们能够学习这两个并行库的基本功能和区别，从而对它们形成初步的认识。

3.3.2 实验步骤二：学习threading和multiprocessing库的样板程序

在基本认识了threading和multiprocessing库之后，我们将提供几个使用它们进行并行编程的样板程序。通过学习这些样板程序，我们能够分析它们的语法逻辑和使用方法，从而为自己上手应用打好基础。

threading库的一个样板程序：

```
import threading
import time

# A sample function to print squares
def print_squares(thread_name, numbers):
    for number in numbers:
        print(thread_name, number, number**2, time.time())
        time.sleep(1)
```

```
# Creating 3 threads that execute the same function with different parameters
thread1 = threading.Thread(target=print_squares, args=("thread1", [1, 2, 3, 4, 5]))

thread2 = threading.Thread(target=print_squares, args=("thread2", [6, 7, 8, 9, 10]))
thread3 = threading.Thread(target=print_squares, args=("thread3", [11, 12, 13, 14, 15]))

# Start the threads
thread1.start()
thread2.start()
thread3.start()

# Join the threads before moving further
thread1.join()
thread2.join()
thread3.join()
```

multiprocessing库的一个样板程序：

```
import time
from multiprocessing import Pool


def factorial(numbers):
  for number in numbers:
    print(number)
    time.sleep(3)

def mp_map():
  with Pool(4) as p:
    results = p.map(factorial, [(1, 2, 3, 4, 5), [6, 7, 8, 9, 10], [11, 12, 13, 14, 15], 'abcde'])
  return results


if __name__ == '__main__':

  results = mp_map()

  print(results)
```

3.3.3 实验步骤三：使用threading和multiprocessing进行并行编程解决实际问题

在完成前面两个步骤后，我将根据各位的理解程度，发布适合的实际编程问题。各位同学可以实际上手编程，用并行的手段来解决这些问题，从而最终掌握threading和multiprocessing库的使用方法。视具体进展情况，此实验步骤的持续时间可能发生变法。

3.3.4 实验步骤四：完成实验报告

最终，我们将把整个实验过程总结成一份完整且充实的实验报告。

4 总结

这是一个面向真实应用场景的实验；我尝试将实际编程工作中可能遇到的棘手问题用简单的形式展现出来。通过本实验，我相信同学能够取得真正的收获，并从积极主动的思维中获得成果，进而认识到并行编程的意义和背后的根本逻辑。最后，我希望本实验能增大同学们对编程的兴趣。也许最终“编程确实很有趣”能够成为我们的共识。祝编程愉快, happy coding.

References

[1] P. Pacheco, M. Malensek, An Introduction to Parallel Programming, 2nd ed, 2023.

[2] S. Heldens, A. Sclocco, H. Dreuning, B. van Werkhoven, P. Hijma, J. Maassen, R. V. van Nieuwpoort, litstudy: A python package for literature reviews, SoftwareX 20 (2022) 101207.

[3] A. Rayhan, D. Gross, The rise of python: A survey of recent research.