分布式与并行计算课程Project

201300023 王瑾
人工智能学院

实验环境

- Visual Studio Code 1.56.2 (user setup)
- Python 3.8.0 64-bit
- Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz 2.59 GHz

运行时间

	快速排序	枚举排序	归并排序
串行	0.07782530784606934	62.880125284194946	0.11912846565246582
并行	3.591548442840576	61.56863713264465	5.522083282470703

并行算法的伪代码

> par-do something 意为为其创建新线程

- 快速排序并行算法

```
def ParallelQucikSort(data, p, r):
    q = Partition(data, p, r)
    task[1] = ParallelQuickSort(data, p, q - 1)
    task[2] = ParallelQuickSort(data, q + 1, r)
    par-do task[1]
    par-do task[2]
    return data
```

- 枚举排序并行算法

```
def ParallelEnumerateSort(data):
    sorted_data = []
    for i = 1 in range(len(data)) par-do:
        counter = 0
        for j = 1 in range(len(data)) do:
            if data[j] < data[i]:
                counter = counter + 1
        sorted_data[counter+1] = data[i]
    return sorted_data
```

- 归并排序并行算法

```
def ParallelMergeSort(data):
    if (data.length==1)
        return data
    else
        task1 = data[:data.length/2]
        task2 = data[data.length/2:]
        res1 = ParallelMergeSort(task1)
        res2 = ParallelMergeSort(task2)
        res = MergeData(res1,res2)
        return res
```

实验结果分析

- 使用的是对并行支持较差的python3.8，又由于并行处理的开销是一个先增后降的过程，在三万这个数据集规模上，三个并行算法都是对串行的负优化或者无效优化。
- 使用的是concurrent.futures中的ThreadPoolExecutor类和as_completed以及wait函数模拟并行，使用列表存储指向线程的指针，并且将函数传递给该线程分治执行。

```
from concurrent.futures import ThreadPoolExecutor, wait, as_completed
```

并行算法优化

- 对于枚举算法，直接创建三万个进程会导致线程排队，产生大量不必要的等待时间，可以人为限制线程的数目（例如控制20个以内）
- 对于快排和归并这类需要递归创建线程的并行算法，可以监控递归的层数，在层数达到一定数量后直接使用串行的快排和归并。