分布式与并行计算 课程Project

丁豪 人工智能学院

181220010@smail.nju.edu.cn

运行时间

- 为了减小偶然性因素造成的时间差异，对每一个算法重复运行10000次，取平均耗时。
- 考虑到“枚举排序”所需时间（$O(n^2)$）远远长于另外两者，因此“枚举排序”只运行一遍。

时间（毫秒）	快速排序	枚举排序	归并排序
串行	1.6889	2204	3.4941
并行	0.8604	277	1.5046

并行算法实现

快速排序

- 原始串行快速排序算法中有“分而治之”的递归调用部分，在每次选择pivoit并把序列按照 `小于pivoIt` 和 `大于pivoit` 分成两类后，左右两部分的递归排序可以并发执行，伪代码如下。

```
ParallelQucikSort(data, p, r)
    q = Partition(data, p, r) // partition会返回pivoit
    task[1] = ParallelQuickSort(data, p, q - 1)
    task[2] = ParallelQuickSort(data, q + 1, r)
    for j = 1 to 2 par-do
        task[i]
    return data
```

枚举排序

- 枚举排序原本会依次对每个元素进行计数的操作，我们可以使这些操作并行化，伪代码如下

```
ParallelEnumerateSort(data)
    sorted_data = new List
    for i = 1 to data.length par-do
        counter = 0
        for j = 1 to data.length do
            if data[j] < data[i]
                counter = counter + 1
        sorted_data[counter+1] = data[i]
    return sorted_data
```

归并排序

- 归并排序的并行化改进与快速排序类似，都是在“分而治之”分完之后，对两个子任务并行执行，伪代码如下

```
ParallelMergeSort(data)
    if (data.length==1)
        return data
    else
        new_data[1] = data[:data.length/2]
        new_data[2] = data[data.length/2:]
        for i = 1 to 2 par-do
            ParallelMergeSort(new_data[i])

        data = MergeData(new_data[1],new_data[2])
        return data
```

实验结果分析

- 算法中的并行使用java的Fork / Join框架实现，他会将进程使用ForkJoinPool进行管理，并自动分配到空闲的CPU核心上来运算。由于个人PC的CPU核心数量较少，所以预期至多能产生常数倍的加速效果。
- 本次实验使用的实验设备为：Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz 2.59 GHz，它具有12个计算核心。
- 实验的详细时间结果参考第一部分。可以发现在QuickSort与MergeSort中，大约获得了两倍的加速。而EnumerateSort则获得了约8倍的加速。
- 本次实验进行排序的workbench较小（总共30000个），相比于计算时间，准备、通信等时间所占的比例无法忽略。经过测试，如果人为将数据量扩大100倍到3000000个，则并行算法的加速效果可以进一步提升。同时受到CPU核心数的限制（12个），并行程度相当有限，因而最终所取得的加速效果至多只能是常数倍，无法改变渐进意义下的时间复杂度。