并行处理实验报告

姓名：孙旭东　　**学号：**151220097

2017 年 12 月 10 日

1　并行算法伪代码

以下分别是快速排序，归并排序，枚举排序三种排序的伪代码:

Algorithm 1 并行快速排序

Require: 无序的$data$数组，n数组大小

Ensure: 有序的$data$数组

function QuickSort($data, left, right$)
    if $left < right$ **then**
        $pivotIndex \leftarrow$ Partition(data, left, right)
        QuickSort($data, left, pivotIndex - 1$)
        QuickSort($data, pivotIndex + 1, right$)
    end if
    return $data$
end function
function ParallelQuickSort($data, left, right$)
    if $n < 1500$ **then**
        QuickSort($data, left, right$)
    else
        $pivotIndex \leftarrow$ Partition(data, left, right)
        $t \leftarrow new\ thread$
        t call ParallelQuickSort($data, left, pivotIndex - 1$)
        ParallelQuickSort($data, pivotIndex + 1, right$)
    end if
    return $data$
end function

Algorithm 2 并行枚举排序

Require: 无序的$data$数组，n数组大小

Ensure: 有序的$data$数组

function EnumkSort($data, left, right$)
    if $left < right$ **then**
        get the rank of every element
        put every element into right position according to their rank
    end if
    return $data$
end function
function ParallelEnumSort($data, left, right$)
    split $data$ into k parts: $data_1, data_2, ...data_k$
    $t_1, t_2, ...t_{k-1} \leftarrow k-1$ *new threads*
    for $1 \leq i \leq k-1$ **do**
        t_i call EnumSort($data_i, left_i, right_i$)
    end for
    EnumSort($data_k, left_k, right_k$)
    return $data$
end function

Algorithm 3 并行归并排序

Require: 无序的$data$数组，n数组大小

Ensure: 有序的$data$数组

```
1: function MergeSort(data, left, right)
2:     if left < right then
3:         middle ← (left + right)/2
4:         MergeSort(data, left, middle)
5:         MergeSort(data, middle + 1, right)
6:         Merge(data, left, middle, right)
7:     end if
8:     return data
9: end function
10: function ParallelMergeSort(data, left, right)
11:     if n < 1500 then
12:         MergeSort(data, left, right)
13:     else
14:         middle ← (left + right)/2
15:         t ← new thread
16:         t call ParallelMergeSort(data, left, middle)
17:         ParallelMergeSort(data, middle + 1, right)
18:         Merge(data, left, middle, right)
19:     end if
20:     return data
21: end function
```

算法描述：

并行的快排算法在进行过partition之后会把分割的两部分交给两个不同的线程来处理，同时在待排序数组长度小于一定阈值(1500)之后就直接进行串行排序。

并行的枚举排序算法会会首先将原数组等分成k份(k等于设备允许同时运行的线程数)，然后在每个线程上单独进行串行枚举排序。

并行的归并排序类似于快速排序，在数据量较大时分为两个线程来分别处理数组前半部分和后半部分，最后合在一起串行归并。

2 性能分析

以下是六种算法在random.txt上的10次运行的运行时间的平均数：

表 1: 各算法在random.txt上的表现

	快速排序	枚举排序	归并排序
串行	$11ms$	$39ms$	$7ms$
并行	$9ms$	$34ms$	$10ms$

可以看出各并行算法对于效率提高并不显著，尤其是归并排序的并行算法甚至比串行更慢，可以考虑到这是因为并行算法中线程调度的开销较大而且数据量较小导致的结果。

我尝试换了一个更大的测试数据：10w个无序的数据，在仿照之前实验重新测试，结果如下：

表 2: 各算法在10w数据量上的表现

	快速排序	枚举排序	归并排序
串行	$65ms$	$32577ms$	$46ms$
并行	$44ms$	$7150ms$	$39ms$

可以看出在更大的数据上并行算法相比于串行算法有了比较明显的效率提升，尤其是并行枚举算法运行时间仅仅有串行的1/4不到。
接下来换成大小为100w的大数据集进行测试，由于在这个数据量下枚举排序效率太低，所以不再对枚举排序进行测试。测试结果如下：

表 3: 各算法在100w数据量上的表现

	快速排序	归并排序
串行	$294ms$	$267ms$
并行	$201ms$	$189ms$

在100w的数据之下，并行算法的效率提升十分显著。

3 技术要点

本次实验我是使用java语言来进行实验的。由于要编写多线程程序，所以使用到了java的Thread类。在编写程序过程中我发现线程太多并不一定是好事，因为庞大的调度开销和IO开销会拖垮整个程序，所以需要过多线程的并行算法理论上很美好但是在自己的电脑上运行效率很低。所以我设置同时运行的线程不超过8个(因为自己的电脑是8核处理器)，在对线程进行数量控制之后效率得到明显的提升。
除此之外我还注意到在数据量很小时多线程并发相比于串行效率并不高，所以在快速排序和归并排序中我设置了一个阈值(1500)，在数据量小于阈值时直接进行串行排序，这对程序运行效率也有显著的提升。
在处理枚举排序的并行化时，我注意到枚举排序的特点是每两个元素都会进行比较，程序各个部分之间的耦合度较高，很难做出简单的并行划分，如果采取每个线程处理一个元素，理论上可以得到很高的效率，但是由于需要的线程数过多，反而成了负担。所以我最终决定在一开始将数据分成8份然后

被分别由8个线程独自处理，最后进行汇总。在实践中这种做法可以很稳定的降低程序运行时间开销。

4 实验结果分析

通过分析比较以上的实验数据，可以发现，在数据量小的情况下，并行算法并不比串行算法高效太多，甚至因为自身的IO开销和调度开销可能还会比串行算法慢，然而在数据量很大的情况下，并行算法的优势逐渐显现，其自身开销被庞大的数据量分担，所以可以比较高效的执行。

一个不容忽视的现象是数据量较少时归并排序并行时间可能会比串行还长，这个现象发生的原因和归并排序本身有关。首先归并排序是一个稳定的高效的排序算法，数组被依次划分为$1/2, 1/4, 1/8, ...$随着时间推移，数组的大小很快降到1500以下，这时程序会采用串行算法，也就是说并行的归并排序承担了多线程的额外开销但是并无法得到很高的并发度。随着数据量的上升，当数据达到100w时并行归并排序已经很明显优于串行了，这是因为在这种情况下多线程额外开销和庞大的数据计算比已经很小了，而庞大的数据又可以被多线程分解成一组一组的小数据，优点得到了放大缺点得到了修补，所以发挥良好。

另外，并行的枚举排序算法在较大数据量时效果是最显著的，这也是由于其自身的特性。枚举排序效率远低于快速排序和归并排序，一开始就划分成8份的做法使得原本$O(n^2)$的运算量(在系数层面上)被大大减少，所以最慢的枚举排序反而是并行后效果提升最显著的。

通过上述的实验和分析，也可以看到一个比价复合直觉的现象：原本高效的算法并行化后效率的提升往往不如原本低效的算法并行化后的效率提升显著。