**分布式与并行计算实验**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**project**：

枚举、归并、快速排序的串行与并行算法比较

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

院系：人工智能学院

姓名：石睿

学号：211300024

班级：分布式与并行计算-2023秋季学期

邮箱：[211300024@smail.nju.edu.cn](mailto:211300024@smail.nju.edu.cn)

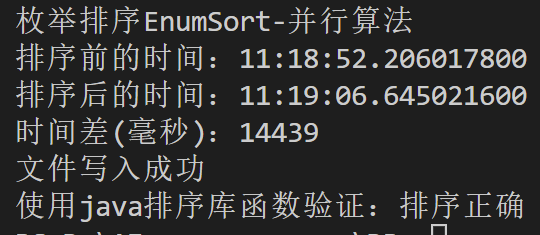
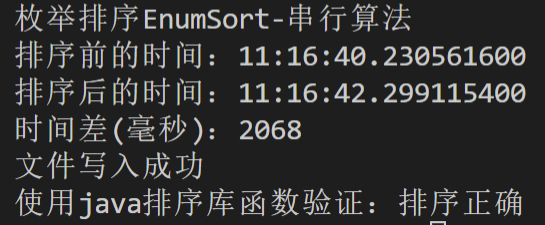
实验时间：2023.12.12

1. **实验进度**

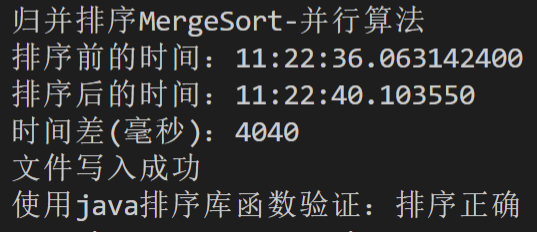
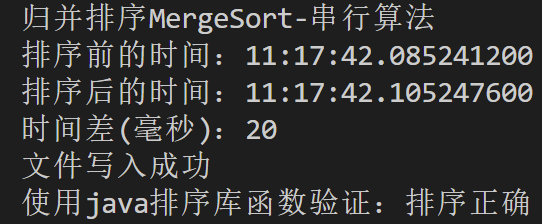
我已经完成了本次课程project的所有内容，也即分别实现快速排序、枚举排序、归并排序三种排序方法的串行与并行算法。

1. **实验结果**

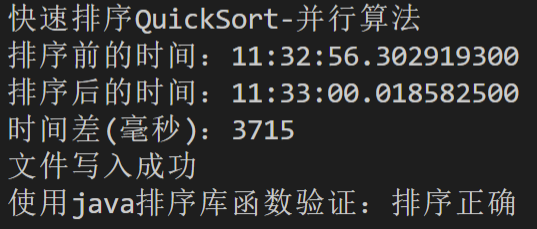
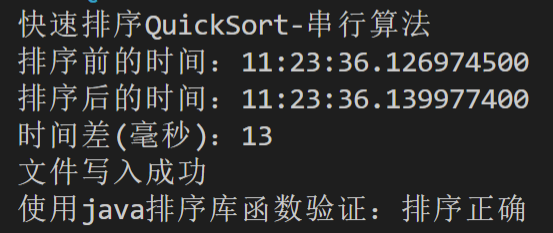
本次实验使用java完成三种串行和并行算法，**算法未优化运行结果**如下图所示（此时并行算法不额外引入任何串行部分）。各算法均运行多次，取出最符合均值的运行结果。同时由于使用了git版本控制，每次运行后orderi.txt并没有出现modified（M）标志也就是多线程运行结果符合预期（无interleave，是可串行化调度）。同时也借助java排序库函数Arrays.sort()来验证串行、并行算法正确性。具体实践细节将在第三部分详细阐释。

、

**1.1： 实现串行的枚举排序算法** **1.2： 实现并行的枚举排序算法**

****

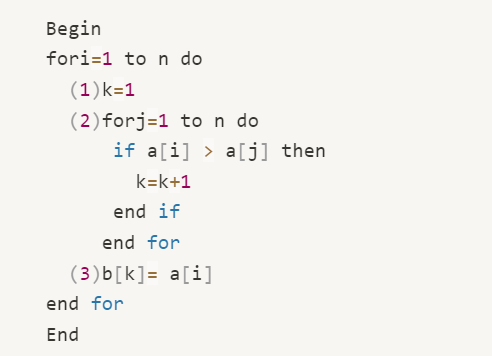
**2.1： 实现串行的归并排序算法** **2.2： 实现并行的归并排序算法**L



**3.1： 实现串行的快速排序算法** **3.2： 实现并行的快速排序算法**L

1. **技术要点**
   1. **枚举排序EnumSort**
   2. **串行算法**

枚举排序算法核心思想是对每个元素统计小于它的所有元素个数，从而得到该元素在序列中的位置。如a[1]和a[2],…,a[n]比较，记录比a[1]小的个数k，最后存放在b[k]的位置上（下标从0开始）一共n（n-1）次，复杂度下限是O(n^2)。其算法伪代码如下所示。

****

运行多次后对运行时间取平均值，可以得到串行的枚举排序所需时间约为2000ms（2.0s）（此时复杂度是O(n^2)）。

* 1. **并行算法**

上述枚举排序不是一个好的串行算法，但是其并行算法很好设计，从伪代码的角度来看加个para-do就行！因为本枚举排序串行算法是松耦合的。可以轻易地解耦成多个cpu，且各个cpu之间可以各自做事，无需频繁交互！类似的，如果串行算法中for很多，则被转化成良好的并行算法的可能性会大很多。并行枚举排序的伪代码如下所示。



以下为通过java具体实现代码。此时增加了同步障barrier，初始化定义时传入一个整形变量parties，表示只有到parties个线程同时运行到barrier.await()处后，各线程才能共同运行下去；增加了类ES，其有一个构造函数 ES(int[] arr,int[] result, int target, CyclicBarrier barrier)和运行函数public void run()。

在经过多次试验中发现，不加同步障barrier和加入同步障barrier结果相同。可能是因为main线程返回result之前，所有线程完成对result数组的赋值更新。但在逻辑上为了保持可串行化调度，是必须要加上barrier。否则在main线程返回result前，线程没有完成对result赋值，对应位置值全部为0（java对未初始化数组的定义）。

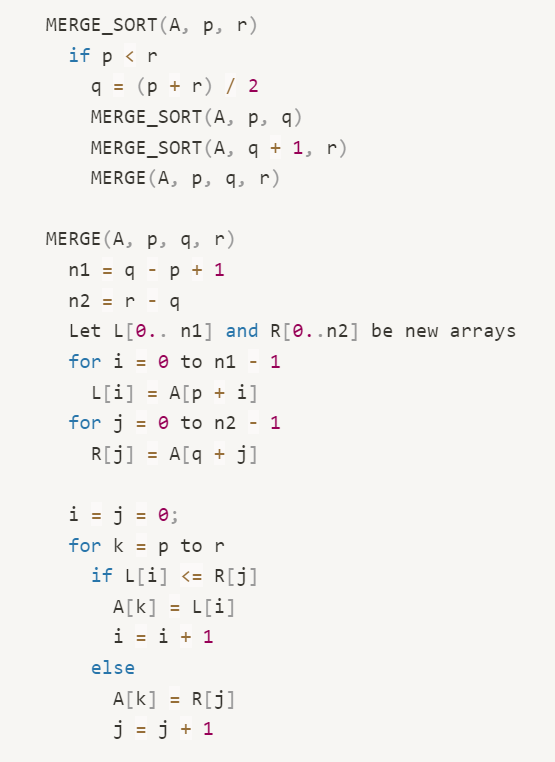


运行多次后对运行时间取平均值，可以得到符合设计逻辑（加入barrier）的并行的枚举排序所需时间约为14000ms（14s）。

对比串行和并行的枚举排序算法，可以得到并行算法约为串行算法时间消耗的7倍（在不进行任何优化的时候）。造成性能很差的原因是barrier和线程数量太多（新开arr.length）。考虑对并行枚举排序算法优化，其中barrier不可以删除。可以使部分arr中元素调用并行enum\_sort，进而减小新开线程数量以及在同步障barrier的等候时间！

* 1. **归并排序MergeSort**
  2. **串行算法**

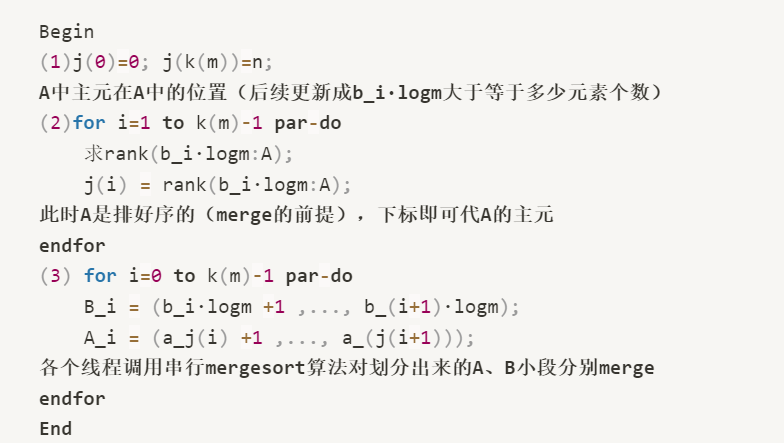
归并排序算法核心思想原数组进行折半划分，分别对两个子序列递归排序Merge\_Sort后再进行两子序列的合并Merge。复杂度下限是O(n·logn)。其算法伪代码如下所示。

****

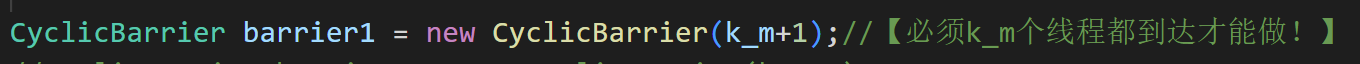
运行多次后对运行时间取平均值，可以得到串行的归并排序所需时间约为18ms（此时复杂度是O(n·logn)）。

* 1. **并行算法**

我首先尝试使用方根划分技术（Valiant算法）来完成Merge的并行化，它的复杂度为O(loglogn)，但由于划分后边界条件过多就没有继续完成了。后续实验采用对数划分技术来对Merge操作进行并行化，复杂度为O(logn)。对数划分技术的伪代码如下。



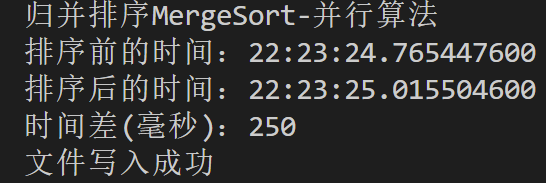
同枚举排序的并行算法设计，方根划分也需要引入同步障。必须在所有新创建的线程(k\_m)个和main线程同时到达return ans（merge后的结果元祖）前，才可以由main线程return ans。否则，main线程在新创建线程还未完成自己对ans中规定位置的merge，则会返回很多的0！

****

****

运行多次后对运行时间取平均值，可以得到并行的归并排序所需时间约为4000ms（4s）。

对比串行和并行的枚举排序算法，可以得到并行算法约为串行算法时间消耗的60倍（在不进行任何优化的时候）。造成性能很差的原因是barrier和线程数量太多。考虑对并行枚举排序算法优化，其中barrier不可以删除。可以在k\_m (A、B中划分段数)小于一定常数值的时候调用串行算法MergeSort.merge，进而减小新开线程数量以及在同步障barrier的等候时间！如在划分段数小于1000时候直接并行merge，运行时长如下所示。



* 1. **快速排序QuickSort**

**3.1 串行算法**

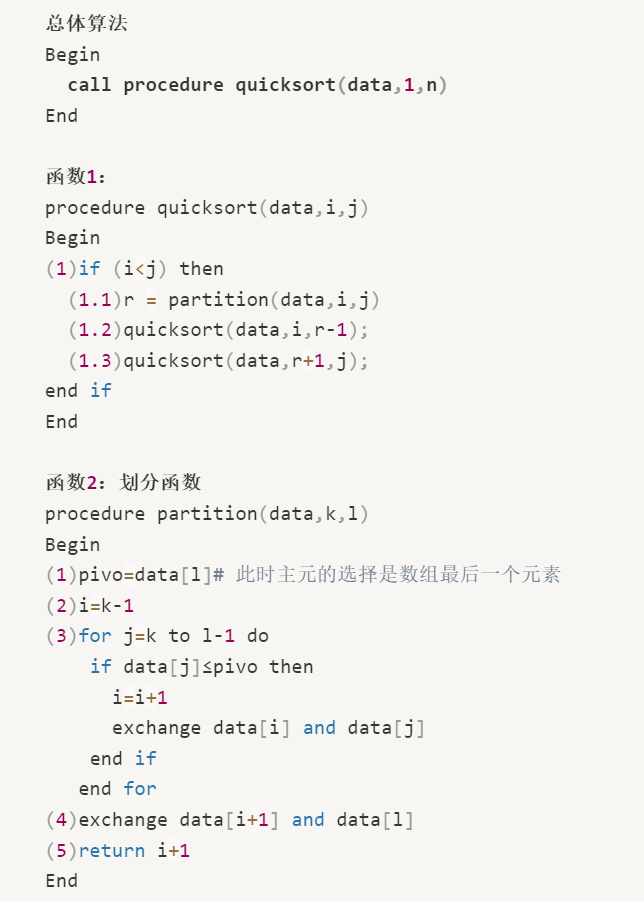
快速排序的核心步骤有以下三步：

**3.1.1** 从原始数组中随机选择主元（因为原数据分布未必均匀，取第一个、最后一个、中间元素作为主元都未必在期望上保持平均划分原数组【中位数的中位数也可以滴！保证平衡】）

**3.1.2** 依主元划分成左右两个数组，左侧小，右侧大

**3.1.3** 分别递归调用两个子数组

对应伪代码如下所示。



书上提供的伪代码partition思路不是很明确，具体实践时采用原算法课讲的partition 想法--通过双指针法来实现（复杂度也是O(n)）。

运行多次后对运行时间取平均值，可以得到串行的快速排序所需时间约为13ms（此时复杂度是O(n·logn)）。

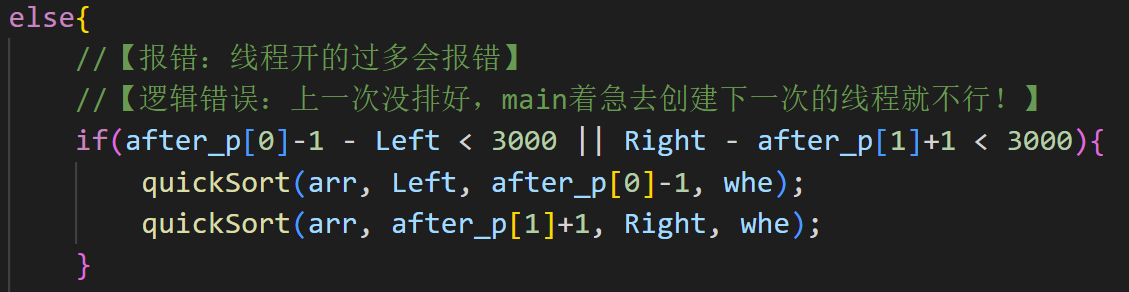
**3.2 并行算法**

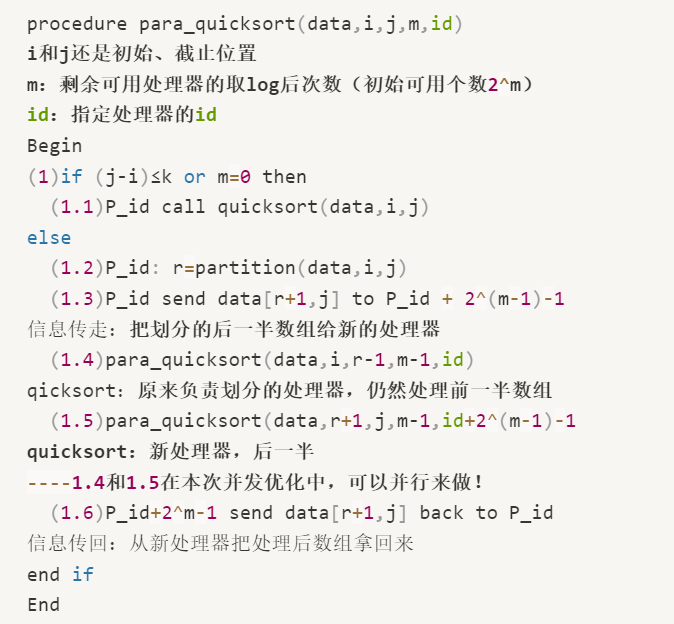
有两种方法可以实现对QuickSort的并行化：以下分别介绍其细节并实现。

**3.2.1 对每次调用的quicksort( ) 使用两处理器递归排序**

其核心想法：(1.2)quicksort(data,i,r‐1);使用一个处理器； (1.3)quicksort(data,r+1,j);——使用另外一个处理器。只做了quicksort的直接串行化并发，但真正做事的partition并没有并发，而只实现了模式的并发。

对应的时间复杂度为O(n)（可使用主定理推导出来）。此时必须限制创建线程的个数，否则会到达java允许创建新线程数量上限。运行多次后对运行时间取平均值，可以得到串行的快速排序所需时间约为3700ms(3.7s)。





**3.2.2 对partition的并行化**

对partition进行并行化的核心想法有以下四步，首先进行符号规定：

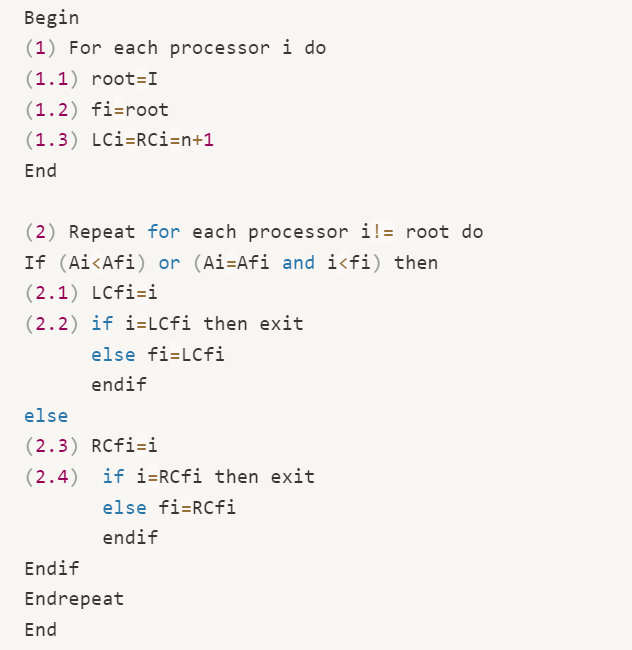
待排序的序列为(A1,…,An)，处理器Pi保存元素Ai。fi存当前主元所在的处理器号、根：主元 root、左子树：小于主元的元素 LC[1:n]、右子树：大于主元的元素 RC[1:n]

**step1**: 所有处理器把他们的处理器号写入root。

**step2**: A\_root是主元，并把root（主元所在处理器号）复制给每一个处理器i的fi

**step3:** 每个处理器对Ai和Afi做比较，把Ai写入LC\_fi 或 RC\_fi【所有处理器的fi值相同】

**step4**: 由于Ai写入LC\_fi 或 RC\_fi也是在CRCW上，最后一个写入的就是新的主元



此时算法期望复杂度为O(logn)。

* 1. **各算法未优化&优化后运行时间（保持一定并发程度）**

**未优化**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 枚举排序 | 归并排序 | 快速排序 |
| 串行 | **1660ms** | **12ms** | **11ms** |
| 并行 | **16000ms** | **4000ms** | **3700ms** |

**优化后（保证一定“并行”程度（单核处理器只能用并发模拟并行））**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 枚举排序 | 归并排序 | 快速排序 |
| 串行 | **1660ms** | **12ms** | **11ms** |
| 并行 | **16000ms** | **445ms** | **344ms** |

此时分别对归并排序和快速排序引入规模相等的串行部分（递归子数组长度小于10采用串行计算），仍保留极高的“并行”（并发）程度，但可以显著降低运行时耗。枚举排序由于没有递归，若引入串行部分计算意义不大。

考虑到并行算法多线程在单核处理器中的并行开销，并行算法性能会比串行算法有所下降。三种排序对应串行和并行的比较在前文已分析，此处就不再赘述。