**并行计算 作业**

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | 王少博 | 学号 | 181110315 |
| 总分 | 10 | 实际得分 |  |
| 作业内容（问题，思路，程序，结果，过程中遇到问题的解决方法） | | | |
| 计数排序基本思想是对于列表a中的每个元素a[i]，计算小于a[i]的元素个数，将a[i]插入到由count决定的列表下标位置中，并用临时列表覆盖原始列表。  问题：   1. **如果我们试图并行化外层循环，哪些变量为private，哪些变量为shared？**   a, n, temp为shared；i, j, count为private。  # pragma omp parallel for num\_threads(thread\_count) default(none) shared(a, n, temp) private(i, j, count) schedule(static, 2)   1. **是否存在循环携带的数据依赖性？为什么？**   不存在。由于计数排序是计算小于a[i]的元素个数，所以下一层循环不会对上一层产生影响，所以不存在循环携带的数据依赖型。   1. **编写并行化的Count\_sort。**   只需要在串行化的代码中加一句Count\_sort的预处理指令即可。并行化的Count\_sort函数的源代码如下：    程序的main函数的源代码如下：       1. **并行化的Count\_sort与串行化的Count\_sort相比，性能如何？**   ./omp后面的第一个数字是线程数、第二个数字是待排序的数组大小、第三个字母是数组的输入方式。如图1所示，当所排序的数组大小很小的时候，串行排序的时间反而比并行排序的时间慢，但是当所排序的数组大小达到1e3数量级的时候，只要线程数大于1，并行的Count\_sort比串行的Count\_sort要快3倍以上。两者的排序速度受待排序列表的数量级的影响，主要是因为并行排序存在着fork和join的时间开销。当待排序列表很小的时候，这种开销甚至会增加程序的运行时间，但是当待排序列表很大的时候，这种开销相比于整体排序时间就微不足道了，所以串行的Count\_sort比并行的Count\_sort慢3倍以上。     1. **并行化的Count\_sort与串行化的qsort库函数相比，性能如何？**   如图所示，由于Count\_sort的时间复杂度是，而qsort的时间复杂度是，所以无论线程数或者是待排序的数组的大小怎么变化，qsort的运行时间总是要比并行化的Count\_sort的运行时间要低。当待排序的数组的数据量小于1e3数量级时，qsort的排序时间要比并行化的Count\_sort的排序时间要低10到100倍；当待排序的数组的数据量大于1e3数量级时，qsort的排序时间要比并行化的Count\_sort的排序时间要低1e3数量级。  **测试结果如下：**  **./omp 10 4 g**    **./omp 10 40 g**    **./omp 10 400 g**    **./omp 10 4000 g**    **./omp 10 40000 g**    **./omp 100 40000 g**    **./omp 200 40000 g**    **./omp 1000 400000 g** | | | |
| 教师评价 | | | |
|  | | | |