中山大学计算机学院本科生实验报告

课程名称：　超级计算机原理与操作　　任课教师：吴迪、黄聃

年级	2019 级	专业（方向）	计算机科学与技术（超算）
学号	19335074	姓名	黄玟瑜
开始日期	2021 年 4 月 15 日星期四	完成日期	2021 年 4 月 15 日星期四

一、实验题目

使用 openmp 实现并行版本的计数排序，参考代码中提供了串行版本和运行所需的主函数。请同学们下载附件 homework4.zip 将并行的代码补充完整并完成实验报告。

二、实验内容

【实验原理】

计数排序是一个非基于比较的排序算法，它的优势在于在对一定范围内的整数排序时，它的复杂度为 O(n+k)（其中 k 是整数的范围），快于任何比较排序算法。这是一种牺牲空间换取时间的做法。

算法思想如下（排序结果为升序）：

1、输入待排序数组 a，数组 a 的元素个数 n

2、创建临时数组 temp，大小和 a 相同，用于保存结果

3、遍历数组 a 的每一个元素，对每一个元素，统计数组中小于它的元素的个数，记为 count，则该元素在数组中的正确位置的下标即 count

4、将结果 temp 写回 a 中

串行程序如下：

```
void Count_sort_serial(int a[], int n) {
   int i, j, count;
   int* temp = malloc(n*sizeof(int));

   for (i = 0; i < n; i++) {
      count = 0;
      for (j = 0; j < n; j++)
         if (a[j] < a[i])
            count++;
         else if (a[j] == a[i] && j < i)
            count++;
      temp[count] = a[i];
   }

   memcpy(a, temp, n*sizeof(int));
   free(temp);
}  /* Count_sort_serial */
```

为了不改变数组中大小相等的元素的相对位置当 a[j] == a[i] && j < i 时仍将 count+1。

【实验过程】

下面来编写并行版本的计数排序。

主要任务是遍历数组的每一个元素，对每一个元素，统计数组中小于它的元素的个数，根据统计结果将其写入保存结果的数组中的正确位置，考虑将任务平均的尽可能平均地分配给各个子线程，将保存结果的数组作为共享空间，各个线程每个线程处理分配给自己的元素并将结果写入保存结果的数组，由于写入的位置位于保存结果的数组的不同地方，因此不会出现竞争。

分析完毕后开始编写程序。

代码解释：

```
int *temp = malloc(n * sizeof(int));
```

创建保存结果的数组 temp，大小和待排序数组 a 大小相同。

```
# pragma omp parallel num_threads(thread_count)
{
   int my_n = n / thread_count;
   int my_rank = omp_get_thread_num();
   int my_first_i = my_n * my_rank;
   int my_last_i = my_first_i + my_n;
   int my_count;

   for(int i = my_first_i; i < my_last_i; i++){
      my_count = 0;
      for (int j = 0; j < n; j++){
         if (a[j] < a[i] || (a[j] == a[i] && j < i)){
            my_count++;
         }
      }
      temp[my_count] = a[i];
   }
}
```

语句# pragma omp parallel num_threads(thread_count) 声明以下部分为并行程序，将使用多个线程来运行。

my_n 为子线程需要处理的数组元素的个数。

my_rank 为该线程的线程号，当 n 个线程同时运行时，线程号为 0，1，2，…，n-1

my_first_i 为该线程处理的第一个数组元素的下标

my_last_i 为该线程处理的最后一个数组元素的下标+1，每个线程处理的数组元素的下标范围为 my_first_i 到 my_last_i-1

用于统计数组中小于它的元素的个数的变量 my_count

for 循环从 my_first_i 遍历到 my_last_i-1，对该范围内的元素依次进行处理，每次循环第一步先给 my_count 置初始值 0，再嵌套一个 for 遍历数组 a 的元素，统计其中小于当前处理的元素的个数，为了不改变数组中大小相等的元素的相对位置，当 a[j] == a[i] && j < i 时仍将 count+1，遍历完毕后 temp[my_count] = a[i];将当前处理的元素写入它在排序数组中的正确位置。

```
memcpy(a, temp, n * sizeof(int));
```

```
   free(temp);
```

将结果 temp 写回原数组 a 中，再将申请来的内存释放掉。

完整程序如下：

```
void Count_sort_parallel(int a[], int n, int thread_count) {
   int *temp = malloc(n * sizeof(int));

   # pragma omp parallel num_threads(thread_count)
   {
      int my_n = n / thread_count;
      int my_rank = omp_get_thread_num();
      int my_first_i = my_n * my_rank;
      int my_last_i = my_first_i + my_n;
      int my_count;

      for(int i = my_first_i; i < my_last_i; i++){
         my_count = 0;
         for (int j = 0; j < n; j++){
            if (a[j] < a[i] || (a[j] == a[i] && j < i)){
               my_count++;
            }
         }
         temp[my_count] = a[i];
      }
   }

   memcpy(a, temp, n * sizeof(int));
   free(temp);
}
```

三、实验结果

首先来检验该并行程序能否输出正确结果：

```
ehpc@f811ea9516cc:~$ cd Documents
ehpc@f811ea9516cc:~/Documents$ gcc -g -Wall -fopenmp -o hw4_openmp hw4_openmp.c
ehpc@f811ea9516cc:~/Documents$ ./hw4_openmp 4 20
a = 4 7 18 16 14 16 7 13 10 2 3 8 11 20 4 7 1 7 13 17
Original: Serial sort a = 4 7 18 16 14 16 7 13 10 2 3 8 11 20 4 7 1 7 13 17
Sorted: Serial sort a = 1 2 3 4 4 7 7 7 7 8 10 11 13 13 14 16 16 17 18 20
Serial run time: 5.006790e-06

Original: Parallel qsort a = 4 7 18 16 14 16 7 13 10 2 3 8 11 20 4 7 1 7 13 17
Sorted: Parallel sort a = 1 2 3 4 4 7 7 7 7 8 10 11 13 13 14 16 16 17 18 20
Parallel run time: 2.419949e-04

Original: Library qsort a = 4 7 18 16 14 16 7 13 10 2 3 8 11 20 4 7 1 7 13 17
Sorted: Library qsort a = 1 2 3 4 4 7 7 7 7 8 10 11 13 13 14 16 16 17 18 20
qsort run time: 3.814697e-06
```

线程数：4 数量：20

```
ehpc@f811ea9516cc:~/Documents$ ./hw4_openmp 4 24
a = 8 23 10 20 18 8 11 13 10 14 3 20 3 20 12 23 13 19 5 17 12 9 16 22
Original: Serial sort a = 8 23 10 20 18 8 11 13 10 14 3 20 3 20 12 23 13 19 5 17 12 9 16 22
Sorted: Serial sort a = 3 3 5 8 8 9 10 10 11 12 12 13 13 14 16 17 18 19 20 20 20 22 23 23
Serial run time: 6.914139e-06

Original: Parallel qsort a = 8 23 10 20 18 8 11 13 10 14 3 20 3 20 12 23 13 19 5 17 12 9 16 22
Sorted: Parallel sort a = 3 3 5 8 8 9 10 10 11 12 12 13 13 14 16 17 18 19 20 20 20 22 23 23
Parallel run time: 2.238750e-04

Original: Library qsort a = 8 23 10 20 18 8 11 13 10 14 3 20 3 20 12 23 13 19 5 17 12 9 16 22
Sorted: Library qsort a = 3 3 5 8 8 9 10 10 11 12 12 13 13 14 16 17 18 19 20 20 20 22 23 23
qsort run time: 4.053116e-06
```

线程数：4 数量：24

```
ehpc@f811ea9516cc:~/Documents$ ./hw4_openmp 4 40
a = 24 7 18 36 34 16 27 13 10 22 3 28 11 20 4 7 21 27 13 17 12 9 8 30 23 11 23 4 28 16 10 3 23 19 30 8 34 17 12
3
Original: Serial sort a = 24 7 18 36 34 16 27 13 10 22 3 28 11 20 4 7 21 27 13 17 12 9 8 30 23 11 23 4 28 16 10
3 23 19 30 8 34 17 12 3
Sorted: Serial sort a = 3 3 3 4 4 7 7 8 8 9 10 10 11 11 12 12 13 13 16 16 17 17 18 19 20 21 22 23 23 23 24 27 27
 28 28 30 30 34 34 36
Serial run time: 1.597404e-05

Original: Parallel qsort a = 24 7 18 36 34 16 27 13 10 22 3 28 11 20 4 7 21 27 13 17 12 9 8 30 23 11 23 4 28 16
10 3 23 19 30 8 34 17 12 3
Sorted: Parallel sort a = 3 3 3 4 4 7 7 8 8 9 10 10 11 11 12 12 13 13 16 16 17 17 18 19 20 21 22 23 23 23 24 27
27 28 28 30 30 34 34 36
Parallel run time: 2.830029e-04

Original: Library qsort a = 24 7 18 36 34 16 27 13 10 22 3 28 11 20 4 7 21 27 13 17 12 9 8 30 23 11 23 4 28 16 1
0 3 23 19 30 8 34 17 12 3
Sorted: Library qsort a = 3 3 3 4 4 7 7 8 8 9 10 10 11 11 12 12 13 13 16 16 17 17 18 19 20 21 22 23 23 23 24 27
27 28 28 30 30 34 34 36
qsort run time: 5.960464e-06
```

线程数：4 数量：40

可以看到对于随机生成的待排序数组 a，并行版本的计数排序与串行版本和库函数 qsort 排序结果相同，说明该排序算法是正确的。同时也能看出再数量级较小的情况下，串行版本的计数排序略慢于库函数 qsort 排序，并行版本的计数排序远慢于串行版本的计数排序和库函数 qsort 排序。

下面比较三个版本排序的性能。

当线程数不变，排序元素数量增加时：

```
ehpc@f811ea9516cc:~/Documents$ ./hw4_openmp 4 10
Serial run time: 1.907349e-06

Parallel run time: 3.249645e-04

qsort run time: 1.907349e-06
```

线程数：4 数量：10

```
qsort run time: 1.907349e-06
ehpc@f811ea9516cc:~/Documents$ ./hw4_openmp 4 100
Serial run time: 7.891655e-05

Parallel run time: 3.008842e-04

qsort run time: 1.311302e-05
```

线程数：4 数量：100

```
ehpc@f811ea9516cc:~/Documents$ ./hw4_openmp 4 1000
Serial run time: 7.322073e-03

Parallel run time: 2.464056e-03

qsort run time: 1.728535e-04
```

线程数：4 数量：1000

```
ehpc@f811ea9516cc:~/Documents$ ./hw4_openmp 4 10000
Serial run time: 8.245540e-01

Parallel run time: 2.115140e-01

qsort run time: 1.860142e-03
```

线程数：4 数量：10000

```
ehpc@f811ea9516cc:~/Documents$ ./hw4_openmp 4 100000
Serial run time: 7.956969e+01

Parallel run time: 1.731566e+01

qsort run time: 1.923490e-02
```

线程数：4 数量：100000

可以看到，随着排序元素数量的增加，并行版本的计数排序的排序速度逐渐超过串行版本的计数排序的排序速度，但它们仍慢于快速排序。

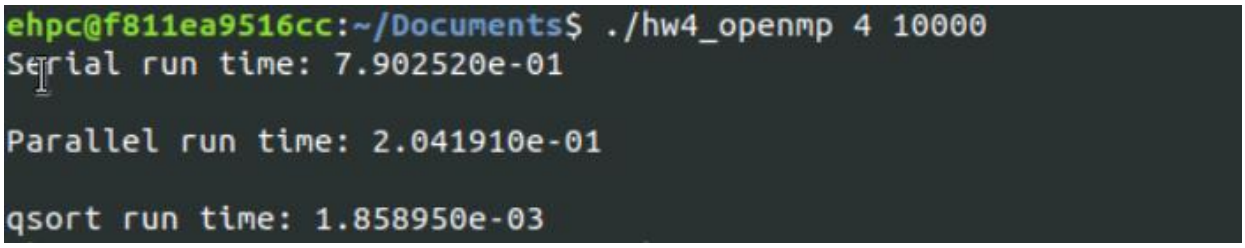
当排序元素的数量不变，线程数增加时：

```
ehpc@f811ea9516cc:~/Documents$ ./hw4_openmp 1 10000
Serial run time: 8.100049e-01

Parallel run time: 8.133080e-01

qsort run time: 1.849890e-03
```

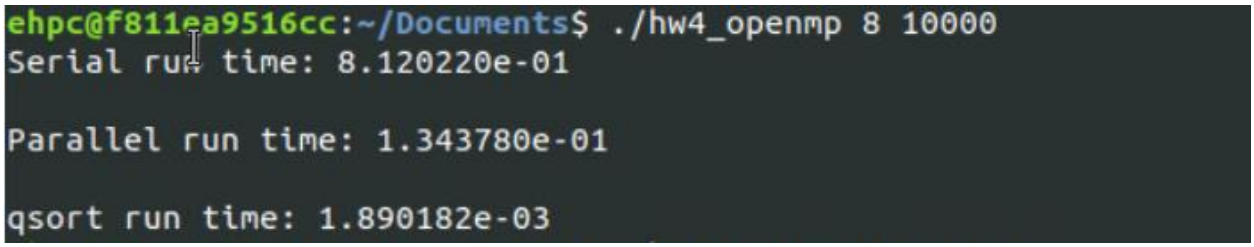
线程数：1 数量：10000

```
ehpc@f811ea9516cc:~/Documents$ ./hw4_openmp 2 10000
Serial run time: 8.044021e-01

Parallel run time: 4.088879e-01

qsort run time: 1.846075e-03
```

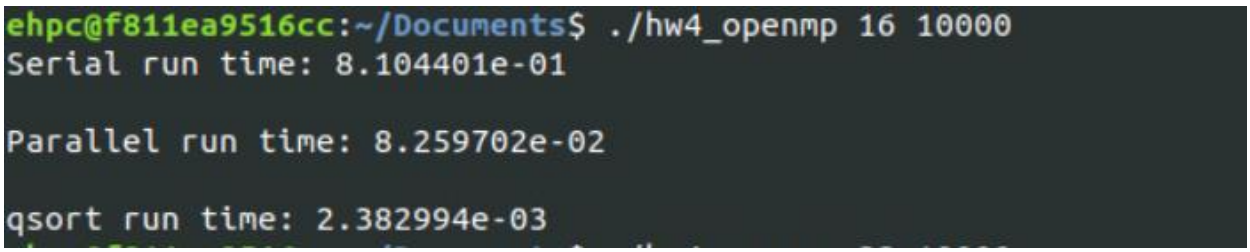
线程数：2 数量：10000



线程数：4 数量：10000



线程数：8 数量：10000



线程数：16 数量：10000

由于硬件条件限制，最多只能运行 16 个线程。可以看到当线程数为 1 时，串行版本和并行版本的计数排序速度相差不多，此后随着线程数的增加，并行版本的计数排序的排序速度以接近指数速度增加（和线程数目的增加成正比），并行版本的计数排序的排序速度和快速排序速度基本不变。

快速排序的速度仍大于串行版本的计数排序的排序速度和并行版本的计数排序的排序速度。

综合上述结果，我们可以得出结论：在处理元素的数量较小的情况下，并行程序的优势不能展现出来，甚至慢于串行程序，而在数量级较大的情况下，并行程序的优势就展现出来了，并行程序的运行速度相比于串行程序明显更快，且随着运行程序的线程数的增加，并行程序的运算速度将会显著提升，大约和线程数目的增加成正比。

此外，通过运行结果也可以看出，快速排序的排序速度始终大于计数排序，无论是串行版本的计数排序还是并行版本的计数排序，这是由算法的性质所决定的。