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
学号	18329015	姓名	郝裕玮
开始日期	2021.4.19	完成日期	2021.4.20

一、实验题目

并行计数排序的实现

二、实验内容

①补充部分代码（即 Count_sort_parallel 函数，并行计数排序）如下所示（绝大部分代码已分析包含在注释中）

```
void Count_sort_parallel(int a[], int n, int thread_count) {
   int i,j,count;//i,j 用于循环
   //count 用于记录每次需排序元素的位置（通过记录有多少个比待排序元素小或相等且位置小于待排序元素的元素个数）
   int* temp=(int *)malloc(n*sizeof(int));//生成 n 个元素的 int 型数组 temp

#  pragma omp parallel num_threads(thread_count) default(none) \
      private(i, j, count) shared(n, a, temp)
      //pragma 的解释放在代码外进行文字解释
   {
#     pragma omp for
      //对于 omp for 同样放在代码外进行文字解释
      for(i=0;i<=n-1;i++){//把每个元素都进行一次计数排序
         count=0;//每次排序都要将 count 清零
         //以便统计有多少个比待排序元素小或相等且位置小于待排序元素的元素个数
         for(j=0;j<=n-1;j++){//开始统计有多少个比待排序元素小或相等且位置小于待排序元素的元素个数
            if(a[j]<a[i]||(a[j]==a[i]&&j<i)){//需要计数的两种情况
            //比待排序元素小或相等且位置小于待排序元素
               count++;//计数+1
            }
         }
         temp[count]=a[i];//将待排序元素的正确位置通过 count 放置到 temp 数组中
      }
#     pragma omp for
```

```
      for(i=0;i<=n-1;i++){
         a[i]=temp[i];//将排好序的数组 temp 内容复制到数组 a 中
         //用并行（omp for）来对串行计数排序中的 memcpy(a, temp, n*sizeof(int))进行时
间和效率上的优化
      }
   }
   free(temp);//释放 temp 数组申请的空间
}  /* Count_sort_parallel */
```

对于

```
#  pragma omp parallel num_threads(thread_count) default(none) \
      private(i, j, count) shared(n, a, temp)
```

对于预处理指令#pragma：

（1）在系统中加入预处理器指令一般是用来允许不是基本 C 语言规范部分的行为。不支持 pragma 的编译器会忽略 pragma 指令提示的那些语句，这样就允许使用 pragma 的程序在不支持它们的平台上运行。

（2）与所有的预处理器指令一样，pragma 的默认长度是一行，因此如果有一个 pragma 在一行中放不下，那么新行需要被“转义”，在前面加一个反斜杠“\”即可。

（3）# pragma omp parallel 代表之后的结构化代码块应该被多个线程并行执行。

（4）num_threads(thread_count)：对 num_threads 进行初始化。

（5）default(none)：编译器将要求我们明确在这个块中使用的每个变量和已经在块之外声明的变量的作用域。(在一个块中声明的变量都是私有的，因为它们会被分配给线程的栈。)

对于

```
#     pragma omp for
```

（1）用 parallel_指令在外部循环前创建 thread_count 个线程的集合。然后，我们不在每次内部循环执行时创建一组新的线程，而是使用一个 for 指令，告诉 OpenMP 用已有的线程组来并行化 for 循环。

接下来是对其余已完成函数的解析：

②宏定义（代码分析已全部包含在注释中）

```
#define GET_TIME(now) { \
   struct timeval t; \
   gettimeofday(&t, NULL); \
   now = t.tv_sec + t.tv_usec/1000000.0; \
}
////最后一行将 us 转换为 s，统一单位
//该结构体用于计算多线程和串行计算 pi 的运行时间
```

③主函数（绝大部分代码分析已全部包含在注释中）

```
int main(int argc, char* argv[]) {
   int n, thread_count;//数组元素个数和线程数
   int *a, *copy;//对数组 a 进行排序，最终结果复制到 copy 中
   double start, stop;//用于计算运行时间

   /* please choose terms 'n', and the threads 'thread_count' here. */
   n = 10;
   thread_count = 4;

   /* You can also get number of threads from command line */
   //if (argc != 3) Usage(argv[0]);
   //Get_args(argv, &thread_count, &n);

   /* Allocate storage and generate data for a */
   a = (int *)malloc(n*sizeof(int));//生成 n 个元素的 int 型数组 a
   Gen_data(a, n);//为 a 数组生成数据

   /* Allocate storage for copy */
   copy = (int *)malloc(n*sizeof(int));//生成 n 个元素的 int 型数组 copy

   /* Serial count sort */
   memcpy(copy, a, n*sizeof(int));//将排序好的数组 a 的结果复制到 copy 中
#  ifdef DEBUG
   Print_data(copy, n, "Original: Serial sort a");
#  endif
   GET_TIME(start);//得到串行计数排序运行的开始时间
   Count_sort_serial(copy, n);//串行计数排序
   GET_TIME(stop);//得到串行计数排序运行的结束时间
#  ifdef DEBUG
   Print_data(copy, n, "Sorted: Serial sort a");
#  endif
   if (!Check_sort(copy, n))//检查排序是否成功
      printf("Serial sort failed\n");
   printf("Serial run time: %e\n\n", stop-start);//打印串行计数排序运行时间

   /* Parallel count sort */
   memcpy(copy, a, n*sizeof(int));//将排序好的数组 a 的结果复制到 copy 中
#  ifdef DEBUG
   Print_data(copy, n, "Original: Parallel qsort a");
#  endif
   GET_TIME(start);//得到并行计数排序运行的开始时间
   Count_sort_parallel(copy, n, thread_count);//并行计数排序
   GET_TIME(stop);//得到并行计数排序运行的结束时间
#  ifdef DEBUG
```

```
   Print_data(copy, n, "Sorted: Parallel sort a");
#  endif
   if (!Check_sort(copy, n))//检查排序是否成功
      printf("Parallel sort failed\n");
   printf("Parallel run time: %e\n\n", stop-start);//打印并行计数排序运行时间

   /* qsort library */
   memcpy(copy, a, n*sizeof(int));//将排序好的数组 a 的结果复制到 copy 中
#  ifdef DEBUG
   Print_data(copy, n, "Original: Library qsort a");
#  endif
   GET_TIME(start);//得到快排运行的开始时间
   Library_qsort(copy, n);//快排
   GET_TIME(stop);//得到快排运行的结束时间
#  ifdef DEBUG
   Print_data(copy, n, "Sorted: Library qsort a");
#  endif
   if (!Check_sort(copy, n))//检查排序是否成功
      printf("Library sort failed\n");
   printf("qsort run time: %e\n", stop-start);//打印快排运行时间

   free(a);//释放 a 数组
   free(copy);//释放 copy 数组

   return 0;
}  /* main */
```

对于各种

```
#  ifdef DEBUG
#  endif
```

在工程设置里有一些设置会对该工程自动产生一系列的宏，用以控制程序的编译和运行。如果你把代码夹在#ifdef DEBUG 和对应的#endif 中间，那么这段代码只有在调试（DEBUG）下才会被编译。也就是说，如果你在 RELEASE 模式下，这些代码根本就不会存在于你的最终代码里头。

④生成数据（代码分析已全部包含在注释中）

```
/*--------------------------------------------------------------------
 * Function:  Gen_data
 * Purpose:   Generate random ints in the range 1 to n
 * In args:   n: number of elements
 * Out arg:   a: array of elements
 */

void Gen_data(int a[], int n) {//生成需要排序的数据
   int i;
```

```
   for (i = 0; i < n; i++)
      a[i] = random() % n + 1; // (double) RAND_MAX;
      //随机生成 n 个数据

#  ifdef DEBUG
   Print_data(a, n, "a");
#  endif
}  /* Gen_data */
```

⑤串行计数排序（代码分析已全部包含在注释中）

```
/*--------------------------------------------------------------------
 * Function:     Count_sort_serial
 * Purpose:      sort elements in an array using count sort
 * In args:      n: number of elements
 * In/out arg:   a: array of elements
 */

void Count_sort_serial(int a[], int n) {//与 Count_sort_parallel 内容几乎一致，不再解
释。
   int i, j, count;
   int* temp = (int *)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 */
```

⑥快速排序（代码分析已全部包含在注释中）

```
/*--------------------------------------------------------------------
 * Function:     Library_qsort
 * Purpose:      sort elements in an array using qsort library function
 * In args:      n: number of elements
 * In/out arg:   a: array of elements
 */

void Library_qsort(int a[], int n) {
```

```
   qsort(a, n, sizeof(int), My_compare);//快排函数
   //函数原型为：
void qsort(void* base,size_t num,size_t size,int (*compar)(const void*,const void*
));
   //void* base：指向要排序的数组的第一个对象的指针，将其转换为 void*。
   //size_t num：base 指向的数组中元素的数量，size_t 是无符号整数类型。
   //size_t size：数组中每个元素的大小（以字节为单位），size_t 是无符号整数类型。
   //int (*compar)(const void*,const void*)：指向比较两个元素的函数的指针。
   //该函数被重复调用 qsort 比较两个元素。它应遵循以下原型：
   //int compar (const void* p1, const void* p2);
   //<0  p1 放到 p2 之前
   //=0  p1,p2 位置不变
   //>0  p1 放到 p2 之后
}  /* Library_qsort */
```

⑦快排第四个参数的函数（代码分析已全部包含在注释中）

```
/*---------------------------------------------------------------------
 * Function:     My_compare
 * Purpose:      compare integer elements for use with qsort function
 * In args:      element a, element b
 * Return val:   positive if a > b, negative if b > a, 0 if equal
 */
int My_compare(const void* a, const void* b) {
   const int* int_a = (const int*) a;
   const int* int_b = (const int*) b;

   return (*int_a - *int_b);//具体含义可见 Library_qsort 函数中的解释
}  /* My_compare */
```

⑧Debug 时打印数组信息（代码分析已全部包含在注释中）

```
/*---------------------------------------------------------------------
 * Function:  Print_data
 * Purpose:   print an array
 * In args:   a: array of elements
 *            n: number of elements
 *            msg: name of array
 */

void Print_data(int a[], int n, char msg[]) {//debug 时打印数组相关信息
   int i;

   printf("%s = ", msg);//打印数组名
   for (i = 0; i < n; i++)
      printf("%d ", a[i]);//按顺序打印数组元素
   printf("\n");
}  /* Print_data */
```

⑨检查排序结果（代码分析已全部包含在注释中）

```
/*--------------------------------------------------------------------
 * Function:  Check_sort
 * Purpose:   Determine whether an array is sorted
 * In args:   a: array of elements
 *            n: number of elements
 * Ret val:   true if sorted, false if not sorted
 */

int  Check_sort(int a[], int n) {//检查是否排序成功
   int i;

   for (i = 1; i < n; i++)
      if (a[i-1] > a[i]) return 0;//若 a[i-1]>a[i]则证明升序排序失败
   return 1;
}  /* Check_sort */
```

⑩Get_args 和 Usage 为命令行编译程序并输入数组元素个数和线程数，与本题关系不大，不再赘述

（因为主函数中已有可以直接修改这两项值的语句）

```
   /* please choose terms 'n', and the threads 'thread_count' here. */
   n = 10;
   thread_count = 4;
```

三、实验结果

①设置数组元素个数分别为 100，1000，10000，线程数不变，均为 4：

```
======== ERROR ========

======== OUTPUT ========
Serial run time: 7.700920e-05

Parallel run time: 2.529621e-04

qsort run time: 4.386902e-05

======== REPORT ========
```

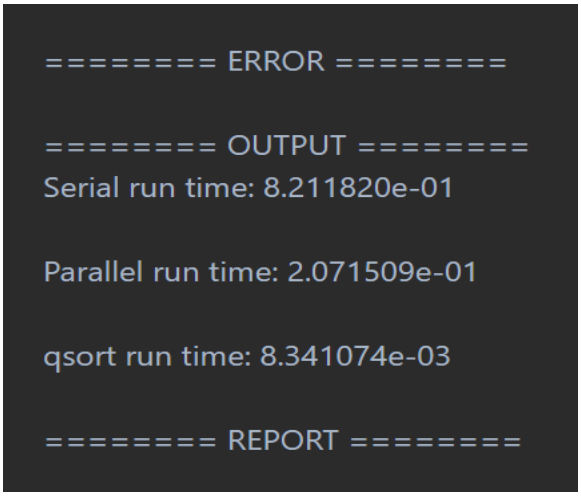
```
======== ERROR ========

======== OUTPUT ========
Serial run time: 7.587910e-03

Parallel run time: 3.319025e-03

qsort run time: 6.530285e-04

======== REPORT ========
```

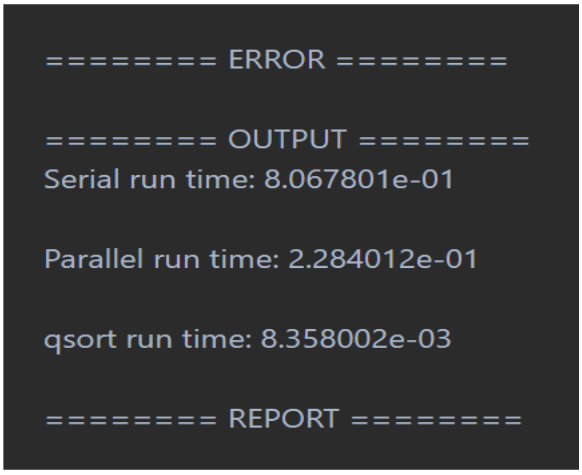


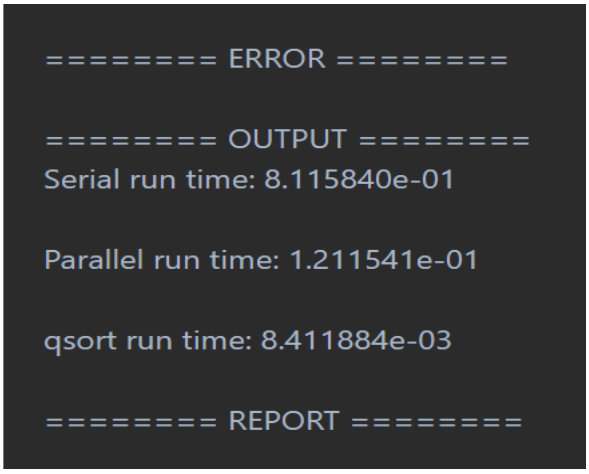
由上述结果可知，数组元素个数增加，串行，并行，快排三种排序的运行时间均会增加。

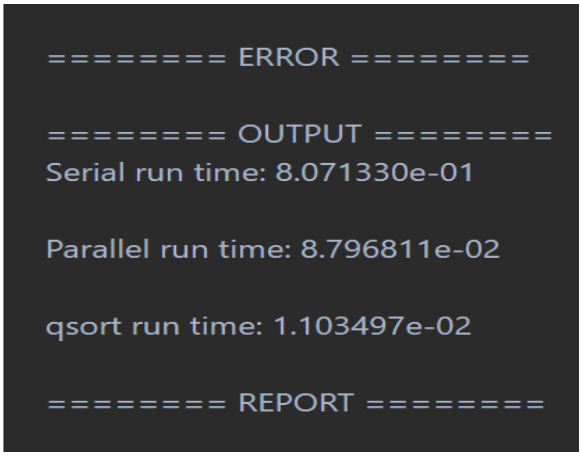
数组元素个数较小时，速率：快排>串行>并行（因为对于并行来说，OpenMP 的 parallel region 结束时，线程之间需要同步：即主线程需要等待所有其他线程完成工作之后才能继续，这个过程可以称做 barrier。该项时间在元素个数较小时对最终运行时间影响较大）

数组元素个数较大时，速率：快排>并行>串行

② 设置线程数分别为 4，8，16，数组元素个数不变，均为 10000：







由上述结果可知，线程数增加，并行的运行时间会减少。串行和快排的时间几乎不变。