计算机系统体系结构 Project3

517030910326 王孝诚

2019.10.31

实验环境

Windows 10 下使用 VMWare Workstation 15 Player 创建和运行虚拟机，虚拟机环境是 Linux 发行版 Ubuntu16.04.6 LTS。

1 多线程排序应用

1.1 设计思路

这里我们需要将一个待排序数组分为两个子待排序数组，分别在一个单独线程中用普通的排序算法将其排序，最后再用第三个线程将两个排序后的数组归并。要实现划分操作，可以考虑将待排序数组存为全局变量，用指定的上下标实现切分。

1.2 核心代码解释

在 main 中，create 两个 sort 子线程，传递给他们排序算法（这里使用了冒泡排序），然后逐一 join；再 create 归并的线程 merge_thread，传递归并算法。

```
pthread_t sort_threads[thread_num];
for(int i = 0;i < thread_num; i++){
    int c = count++;
    pthread_create(&sort_threads[i],NULL,bubbleSort,&c);
}

for (int i = 0; i < thread_num; i++){
    pthread_join(sort_threads[i], NULL);
}

pthread_t merge_thread;
pthread_create(&merge_thread, NULL, merge, NULL);
pthread_join(merge_thread, NULL);
```

每个排序线程操作的上下界通过传入的变量控制。

```
void *bubbleSort(void *arg){
    int current_thread = atoi(arg);
    int low=current_thread*lenth/thread_num;
    int high=(current_thread+1)*lenth/thread_num;

    /*** 排序 ***/
}
```

2 JAVA Fork-Join 排序应用

2.1 设计思路

这道题练习使用了 Java 的 fork-join 并行 API，来实现两个分而治之的排序算法：快速排序和归并排序。划分线程的思想和前面多线程排序的是类似的，但这里需要实现对一个任务的递归调用，需要使各排序算法继承 RecursiveAction 这个类并重写其中的 compute 方法，思路和 4.5.2.1 节中 SumTask 类继承 RecursiveTask 类似。

2.2 核心代码解释

在 ForkJoinApp 的 main 函数中创建线程池，然后分别创建两个排序算法的实例并且 invoke。

```
public static void main(String[] args){
    ForkJoinPool pool = new ForkJoinPool();

    /* 初始化array1和array2 */

    // quick
    QuickSort quickSort = new QuickSort(array, 0, SIZE);
    pool.invoke(quickSort);

    // Merge
    MergeSort mergeSort = new MergeSort(array1, 0, SIZE);
    pool.invoke(mergeSort);

    /* 输出结果 */
}
```

在算法实现中，继承了 RecursiveAction 类，从而通过重写其 compute 方法实现多线程的递归调用。同时设置了阈值，当子问题规模较小时，不再划分而直接只用普通排序算法排序。

```
//quick sort
protected void compute(){
    if(high - low > THRESHOLD){
        int mr = array[low];
        int pos = low;
        for(int i=low+1;i<high;i++){
            if(array[i]<mr){
                int tmp = array[i];
                array[i] = mr;
                array[pos++] = tmp;
            }
        }

        QuickSort leftSort = new QuickSort(array, low, pos);
        QuickSort rightSort = new QuickSort(array, pos+1, high);
        invokeAll(leftSort,rightSort);

    }else{
        bubbleSort();
    }
}
```

```
22
23  // merge sort
24  @Override
25  protected void compute(){
26      if(this.low < this.high - THRESHOLD){
27          int mid = (low+high)/2;
28          MergeSort leftSort = new MergeSort(this.array, this.low, mid);
29          MergeSort rightSort = new MergeSort(this.array, mid, this.high);
30          invokeAll(leftSort,rightSort);
31          merge(this.low,this.high,mid);
32      }else{
33          bubbleSort();
34      }
35
36  }
```

3 实验结果

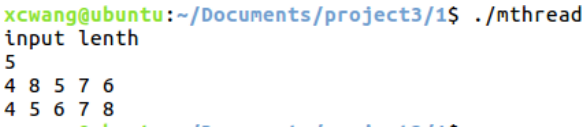

Figure 1: Multi-threaded Sort Application

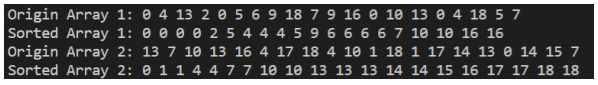

Figure 2: Fork-Join Sorting Application