Lab6 fork join框架实现归并排序

1. fork join框架简介

Fork/Jion框架含有两个重要的类，ForkJoinTask类以及ForkJoinPool类：

ForkJoinTask类用于创建ForkJoin任务，它有两个子类

* RecursiveAction：用于没有返回结果的任务。
* RecursiveTask ：用于有返回结果的任务。

在本例归并排序中，我们需要继承的是有返回结果的RecursiveTask类，ForkJoinTask类中含有compute()函数，当需要分割的任务不满足大小要求时，就通过该函数将任务分割成两个子任务，子任务同样经过compute()函数判断，当任务足够小时，在本例中执行将数字添加到排序数组中，当操作完成时，使用fork()函数启动一个新的fork/join任务，结束后执行join()函数获取任务结果，最后合并得到的两个数组。

ForkJoinPool类作为线程池，管理执行ForkJoinTask任务的线程，线程由双端队列deque实现，ForkJoinTask分割出的子任务从队列头部放入当前线程中，当线程任务队列为空时，线程随机从其它线程队列的尾部窃取任务，在本例中，使用无参构造函数ForkJoinPool pool=new ForkJoinPool();以默认配置创建了一个线程数等于cpu线程数的线程池。它含有两个数组:

* ForkJoinTask数组：负责存放程序提交给ForkJoinPool的任务。
* ForkJoinWorkerThread数组： 负责执行ForkJoinTask数组里的任务。

1. 类概述

Main主类：用于输入输出并且创建ForkJoin任务的线程池

Merge类：继承RecursiveTask类，含有构造函数、用于合并数组的MergeArray()函数，以及继承RecursiveTask类改写的compute()函数

1. 代码分析

主要通过compute()函数来实现归并

如果任务足够小，则将数字单独放在一个数组里，然后按照顺序合并，如果任务不是足够小，那么将该数组分成两个部分生成两个ForkJoinTask，在获得两个结果后，合并两个数组。

**protected** ArrayList<Integer> compute()

{

**boolean** cansort=**this**.end==**this**.begin;

ArrayList<Integer> sorted = **new** ArrayList<Integer>();

**if**(cansort)

{

sorted.add(tosort.get(begin));

}

**else**

{

**int** middle=(begin+end)/2;

Merge left=**new** Merge(begin,middle,tosort);

Merge right=**new** Merge(middle+1,end,tosort);

left.fork();

right.fork();

ArrayList<Integer> larray=left.join();

ArrayList<Integer> rarray=right.join();

sorted=**this**.MergeArray(larray, rarray);

}

**return** sorted;

}

其中tosort储存着原始数组，始终保持不变，通过Merge类构造函数达到截取数组足够小的某一部分（在本例中是一个数）放入sorted数组中，然后返回sorted，当得到两个结果时，合并两个数组。

1. 代码改进

用invokeAll()来代替fork()、join()

使用

Merge left=**new** Merge(begin,middle,tosort);

Merge right=**new** Merge(middle+1,end,tosort);

left.fork();

right.fork();

ArrayList<Integer> larray=left.join();

ArrayList<Integer> rarray=right.join();

执行compute()方法的线程也是一个Worker线程，当该线程对两个子任务执行fork()方法时，该Worker线程将会把任务分配给两个线程，然后停止运行，这样就浪费了一个线程，因此，可以将上述代码改为

Merge left=**new** Merge(begin,middle,tosort);

Merge right=**new** Merge(middle+1,end,tosort);

*invokeAll*(left,right);

ArrayList<Integer> larray=left.join();

ArrayList<Integer> rarray=right.join();

这样可以提高线程使用的效率

1. 源码（改进后）

**package** lab7;

**import** java.util.\*;

**import** java.util.concurrent.ExecutionException;

**import** java.util.concurrent.ForkJoinPool;

**import** java.util.concurrent.Future;

**import** java.util.concurrent.RecursiveTask;

**public** **class** Main {

**public** **static** **void** main(String[] arg)

{

Scanner out=**new** Scanner(System.***in***);

ArrayList<Integer> tosort=**new** ArrayList<Integer>();

System.***out***.println("请输入数组大小");

**int** thesize=out.nextInt();

**for**(**int** i=0;i<thesize;i++)

{

tosort.add(out.nextInt());

}

ForkJoinPool pool=**new** ForkJoinPool();

Merge tomerge=**new** Merge(0,thesize-1,tosort);

Future<ArrayList<Integer>> result=pool.submit(tomerge);

**for**(**int** i=0;i<thesize;i++)

{

**try** {

System.***out***.print(result.get().get(i)+" ");

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (ExecutionException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

}

**class** Merge **extends** RecursiveTask<ArrayList<Integer>>

{

/\*\*

\*

\*/

**private** **static** **final** **long** ***serialVersionUID*** = 6739027449294465706L;

ArrayList<Integer> tosort=**new** ArrayList<Integer>();

**int** begin;

**int** end;

Merge(**int** begin,**int** end,ArrayList<Integer> tosort)

{

**this**.begin=begin;

**this**.end=end;

**this**.tosort=tosort;

}

**public** ArrayList<Integer> MergeArray(ArrayList<Integer> a,ArrayList<Integer> b)

{

ArrayList<Integer> result=**new** ArrayList<Integer>();

**while**(!a.isEmpty()&&!b.isEmpty())

{

**if**(a.get(0)<b.get(0))

{

result.add(a.get(0));

a.remove(0);

}

**else**

{

result.add(b.get(0));

b.remove(0);

}

}

**while**(!a.isEmpty())

{

result.add(a.get(0));

a.remove(0);

}

**while**(!b.isEmpty())

{

result.add(b.get(0));

b.remove(0);

}

**return** result;

}

@Override

**protected** ArrayList<Integer> compute()

{

**boolean** cansort=**this**.end==**this**.begin;

ArrayList<Integer> sorted = **new** ArrayList<Integer>();

**if**(cansort)

{

sorted.add(tosort.get(begin));

}

**else**

{

**int** middle=(begin+end)/2;

Merge left=**new** Merge(begin,middle,tosort);

Merge right=**new** Merge(middle+1,end,tosort);

*invokeAll*(left,right);

ArrayList<Integer> larray=left.join();

ArrayList<Integer> rarray=right.join();

sorted=**this**.MergeArray(larray, rarray);

}

**return** sorted;

}

}