COMP 322: Fundamentals of Parallel Programming

Lecture 19: Java's ForkJoin Library

Mack Joyner
mjoyner@rice.edu

http://comp322.rice.edu

Updating all Elements in an Array

- Suppose we have a large array *a* of integers

- We wish to update each element of this array:
  - `a[i] = a[i] / (i + 1)`

- How would we write this as a parallel program using `async` and `finish`?

Recursive Decomposition

```
solve(problem)
    if problem smaller than threshold
        solveDirectly(problem)
    else
    in parallel:
        l = solve(left-half)
        r = solve(right-half)
    combine(l, r)
```

- In general, can create more than 2 sub-problems
- combine then needs to handle all the sub-problems

Update using async and finish

```
sequentialUpdate(a, lo, hi)
    for (i = lo; i < hi; i++)
        a[i] = a[i] / (i + 1)

parallelUpdate(a, lo, hi)
    if (hi - lo) < THRESHOLD
        sequentialUpdate(a, lo, hi)
    else
        mid = (lo + hi) / 2
         finish(() -> {
             async(() -> { parallelUpdate(a, lo, mid) });
             async(() -> { parallelUpdate(a, mid, hi) });
         });
```

Task Parallelism Using Standard JDK Libraries

- Thread objects (prior to JDK 5)
  - Start Runnable task t with *new Thread(t).start()*
  - Create new Thread each time parallel task needs to be done
- Executors (JDK 5)
  - Handles thread management with thread pools
- ForkJoinTasks (JDK 7) useful for divide and conquer problems
  - Implements work-stealing
- HJLib (JDK 8)

Using Java’s Fork/Join Library

We can perform recursive subdivision using the Fork/Join libraries provided in the JDK as follows:

```
public abstract class RecursiveAction extends
ForkJoinTask<Void> {
    protected abstract void compute();
    …
}
public abstract class RecursiveTask<V> extends
ForkJoinTask<V> {
    protected abstract V compute();
    …
}
```

RecursiveAction Subclass

```
class DivideTask extends RecursiveAction {
  static final int THRESHOLD = 5;
  final long[] array;
  final int lo, hi;

  DivideTask(long[] array, int lo, int hi) {
    this.array = array;
    this.lo = lo;
    this.hi = hi;
   }
   protected void compute() {…} // next slide
  }
```

compute()

```
protected void compute() {
    if (hi - lo < THRESHOLD) {
      for (int i = lo; i <= hi; ++i)
        array[i] = array[i] / (i + 1);
    } else {
      int mid = (lo + hi) >>> 1;
      invokeAll(new DivideTask(array, lo, mid),
               new DivideTask(array, mid+1, hi));
    }
   }
```

ForkJoinTask<V>

- Similar to a finish block enclosing a collection of asyncs
- Other Fork/Join methods in superclass ForkJoinTask<V>

```
class ForkJoinTask<V> extends Object
    implements Serializable, Future<V>
{
    ForkJoinTask<V> fork()   // parallel execution
    V join()         // returns result when execution completes
    V invoke()       // forks, joins, returns result
  static void invokeAll(ForkJoinTask<?> t1, ForkJoinTask<?> t2)
    …
}
```

ForkJoinTasks and Futures

- ForkJoinTasks implement the Future interface
- Acts very much like HJLib futures

```
interface Future<V> {
    V get()
    V get(long timeout, TimeUnit unit)
    boolean cancel(boolean interruptIfRunning)
    boolean isCancelled()
    boolean isDone()
}
```

Recursive Array Sum using HJlib

```java
protected double computeSum(
        final double[] xArray, final int start, final int end)
        throws SuspendableException {

    if (end - start < THRESHOLD) {

        // sequential threshold cutoff
        return seqArraySum(xArray, start, end);

    } else {
        int mid = (end + start) / 2;

        HjFuture<Double> leftFuture = future(() -> {
            return computeSum(xArray, start, mid);
        });
        HjFuture<Double> rightFuture = future(() -> {
            return computeSum(xArray, mid, end);
        });
        return leftFuture.get() + rightFuture.get();
} }
```

Recursive Array Sum using ForkJoinTasks

```
protected static class ArraySumForkJoinTask
        extends RecursiveTask<Double> {
   ...

    protected Double compute() {
        if (end - start < THRESHOLD) {
            // sequential threshold cutoff
            return seqArraySum(xArray, start, end);
        } else {
            final int mid = (end + start) / 2;
             final ArraySumForkJoinTask taskLeft =
                        new ArraySumForkJoinTask(xArray, start, mid);
             final ArraySumForkJoinTask taskRight =
                        new ArraySumForkJoinTask(xArray, mid, end);

             // Is there anything wrong with the code below?
             taskLeft.fork();
             return taskLeft.join() + taskRight.compute();
 } } }
```

Recursive Array Sum using ForkJoinTasks

```
protected static class ArraySumForkJoinTask
        extends RecursiveTask<Double> {
   ...

    protected Double compute() {
        if (end - start < THRESHOLD) {
            // sequential threshold cutoff
            return seqArraySum(xArray, start, end);
        } else {
            final int mid = (end + start) / 2;
             final ArraySumForkJoinTask taskLeft =
                        new ArraySumForkJoinTask(xArray, start, mid);
             final ArraySumForkJoinTask taskRight =
                        new ArraySumForkJoinTask(xArray, mid, end);

             taskRight.fork();
             return taskLeft.compute() + taskRight.join();

} } }
```