COMP 322: Fundamentals of Parallel Programming

Lecture 10: Java’s ForkJoin Library

Mack Joyner and Zoran Budimlić
{mjoyner, zoran}@rice.edu

http://comp322.rice.edu
Worksheet #9: Classifying different versions of parallel search algorithm

Enter “YES” or “NO”, as appropriate, in each box below

<table>
<thead>
<tr>
<th>Example: String Search variation</th>
<th>Data Race Free?</th>
<th>Functionally Deterministic?</th>
<th>Structurally Deterministic?</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1: Count of all occurrences</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>V2: Existence of an occurrence</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>V3: Index of any occurrence</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>V4: Optimized existence of an occurrence: do not create more async tasks after occurrence is found</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>V5: Optimized index of any occurrence: do not create more async tasks after occurrence is found</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
</tbody>
</table>
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 \texttt{async} and \texttt{finish}?
Recursive Decomposition

```python
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

1. `sequentialUpdate(a, lo, hi)`
2. \[\text{for } (i = lo; i < hi; i++)\]
3. \[a[i] = a[i] / (i + 1)\]
4.
5. `parallelUpdate(a, lo, hi)`
6. \[\text{if } (hi - lo) < \text{THRESHOLD}\]
7. `sequentialUpdate(a, lo, hi)`
8. `else`
9. \[\text{mid} = (lo + hi) / 2\]
10. `finish`
11. \[\text{async } \text{parallelUpdate}(a, lo, mid)\]
12. \[\text{async } \text{parallelUpdate}(a, mid, hi)\]
Task Parallelism Using Standard JDK Libraries

• Thread objects (prior to JDK 5)
  • Start Runnable task t with \texttt{new Thread(t).start()}
  • Create newThread each time asynchronous task needs to be done

• Executors (JDK 5)
  • Handles thread management with thread pools
  • Use \texttt{execute(t)} to start a task t with no return value
  • \texttt{ExecutorService} allows for tasks with return values (futures)

• ForkJoinTasks (JDK 7) useful for divide and conquer problems
  • \texttt{Implements work-stealing}

• HJLib, Java streams (JDK 8)
Using Java’s Fork/Join Library

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

```java
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

1. class DivideTask extends RecursiveAction {
2.     static final int THRESHOLD = 5;
3.     final long[] array;
4.     final int lo, hi;
5. 
6.     DivideTask(long[] array, int lo, int hi) {
7.         this.array = array;
8.         this.lo = lo;
9.         this.hi = hi;
10.     }
11.     protected void compute() {...} // next slide
12. }


compute()

1. protected void compute() {
2.     if (hi - lo < THRESHOLD) {
3.         for (int i = lo; i <= hi; ++i)
4.             array[i] = array[i] / (i + 1);
5.     } else {
6.         int mid = (lo + hi) >>> 1;
7.         invokeAll(new DivideTask(array, lo, mid),
8.                 new DivideTask(array, mid+1, hi));
9.     }
10. }
ForkJoinTask<\textgreater V\textless>

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

```java
class ForkJoinTask<\textgreater V\textless> extends Object
    implements Serializable, Future<\textgreater V\textless>
{
    ForkJoinTask<\textgreater V\textless> fork()  // asynchronously executes
    \textgreater V join()  // returns result when execution completes
    \textgreater V invoke()  // forks, joins, returns result
    static void invokeAll(ForkJoinTask<\textgreater ?\textless> t1, ForkJoinTask<\textgreater ?\textless> t2)
    ...
}
```
ForkJoinTasks and Futures

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

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

• Because ForkJoinTasks are Futures, they are the values returned from `fork()`

• We can obtain the result of a ForkJoinTask using `join()` or `get()`

• When calling `invoke` or `invokeAll`, we never get a handle on the future explicitly
  • Similar to `finish/async` blocks in HJLib
Recursive Array Sum using HJlib

1. protected double computeSum(
2.     final double[] xArray, final int start, final int end)
3.     throws SuspendableException {

5.     if (end - start < \text{THRESHOLD}) {

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

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

13.         HjFuture<Double> leftFuture = future(() -> {
14.             return computeSum(xArray, start, mid);
15.         });
16.         HjFuture<Double> rightFuture = future(() -> {
17.             return computeSum(xArray, mid, end);
18.         });
19.         return leftFuture.get() + rightFuture.get();
20.    } }
Recursive Array Sum using ForkJoinTasks

1. **protected static class** ArraySumForkJoinTask
2. 
   extends RecursiveTask<Double> {

3. ... 

4. **protected** Double **compute**() { 
5.     if (end - start < THRESHOLD) { 
6.         // sequential threshold cutoff 
7.         return seqArraySum(xArray, start, end); 
8.     } else { 
9.         final int mid = (end + start) / 2; 
10.        final ArraySumForkJoinTask taskLeft = 
11.            new ArraySumForkJoinTask(xArray, start, mid); 
12.        final ArraySumForkJoinTask taskRight = 
13.            new ArraySumForkJoinTask(xArray, mid, end); 
14. 
15.            taskRight.fork(); 
16.            return taskLeft.compute() + taskRight.join(); 
17. 
18.            // What is wrong with the code below? 
19.            // taskLeft.**fork**(); 
20.            // **return** taskLeft.**join**() + taskRight.**compute**(); 
21. } } }
Announcements & Reminders

• HW2 is available and due by 11:59pm on Wednesday, Feb 6th
• Quiz for Unit 2 (topics 2.1 - 2.6) is available on Canvas, and due by 11:59pm on Monday, February 11th
• See course web site for all work assignments and due dates
• Use Piazza (public or private posts, as appropriate) for all communications re. COMP 322
• See Office Hours link on course web site for latest office hours schedule.