COMP 322: Fundamentals of Parallel Programming

https://wiki.rice.edu/confluence/display/PARPROG/COMP322

Lecture 12: Barrier Synchronization

Vivek Sarkar
Department of Computer Science
Rice University
vsarkar@rice.edu
Announcements

• Homework 4 assigned today, due by 5pm on Wednesday, Feb 16\textsuperscript{th}
  – We will try and return graded homeworks by Feb 23\textsuperscript{rd}

• Midterm will be a 2-hour take-home written exam
  – Closed-book, closed-notes, closed-computer
  – Will be given out at lecture on Wed, Feb 23\textsuperscript{rd}
  – Must be handed in by 5pm on Friday, Feb 25\textsuperscript{th}

• No lecture on Feb 25\textsuperscript{th} since midterm is due that day
Acknowledgments for Today’s Lecture

• “Principles of Parallel Programming”, Calvin Lin & Lawrence Snyder, Addison-Wesley, 2009
  —Includes resources available at http://www.pearsonhighered.com/educator/academic/product/0,3110,0321487907,00.html

• Lecture 12 handout
Hello-Goodbye Forall Example

rank.count = 0; // rank object contains an int field, count

forall (point [i] : [0:m-1]) {
    int r;
    isolated {r = rank.count++;}
    System.out.println("Hello from task ranked " + r);
    System.out.println("Goodbye from task ranked " + r);
}

• Sample output for m = 4
  Hello from task ranked 0
  Hello from task ranked 1
  Goodbye from task ranked 0
  Hello from task ranked 2
  Goodbye from task ranked 2
  Goodbye from task ranked 1
  Hello from task ranked 3
  Goodbye from task ranked 3
Hello-Goodbye Forall Example (contd)

```java
rank.count = 0; // rank object contains an int field, count
forall (point [i] : [0:m-1]) {
    int r;
    isolated {r = rank.count++;}
    System.out.println("Hello from task ranked "+ r);
    System.out.println("Goodbye from task ranked "+ r);
}
```

• Question: how can we transform this code so as to ensure that all tasks say hello before any tasks goodbye?

• Approach 1: Replace the forall loop by two forall loops, one for the hello’s and one for the goodbye’s
  — Need to communicate local r values from one forall to the next

• Approach 2: insert a “barrier” between the hello’s and goodbye’s
  — “next” statement in HJ
Barrier Synchronization: HJ’s “next” statement

```java
rank.count = 0; // rank object contains an int field, count
forall (point [i] : [0:m-1]) {
    int r;
    isolated {r = rank.count++;}
    System.out.println("Hello from task ranked " + r);
    next; // Acts as barrier between phases 0 and 1
    System.out.println("Goodbye from task ranked " + r);
}
```

- `next`  each forall iteration suspends at next until all iterations arrive (complete previous phase), after which the phase can be advanced
  - If a forall iteration terminates before executing “next”, then the other iterations do not wait for it
  - Scope of synchronization is the closest enclosing forall statement
  - Special case of “phaser” construct (will be covered in following lectures)
Impact of barrier on scheduling forall iterations

For all iterations

Phase 0

Phase 1

Barrier Region

Modeling a next operation in the computation graph

signal edges

wait edges
One-Dimensional Iterative Averaging Example

- Initialize a one-dimensional array of \((n+2)\) double's with boundary conditions, \(\text{myVal}[0] = 0\) and \(\text{myVal}[n+1] = 1\).

- In each iteration, each interior element \(\text{myVal}[i]\) in \(1..n\) is replaced by the average of its left and right neighbors.
  - Two separate arrays are used in each iteration, one for old values and the other for the new values.

- After a sufficient number of iterations, we expect each element of the array to converge to \(\text{myVal}[i] = i/(n+1)\).
  - In this case, \(\text{myVal}[i] = (\text{myVal}[i-1]+\text{myVal}[i+1])/2\), for all \(i\) in \(1..n\).

Illustration of an intermediate step for \(n = 8\) (source: Figure 6.19 in Lin-Snyder book)

<table>
<thead>
<tr>
<th>(n=8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
</tr>
<tr>
<td>0.34</td>
</tr>
<tr>
<td>0.21</td>
</tr>
<tr>
<td>0.86</td>
</tr>
<tr>
<td>0.65</td>
</tr>
<tr>
<td>0.11</td>
</tr>
<tr>
<td>0.43</td>
</tr>
<tr>
<td>0.97</td>
</tr>
<tr>
<td>0.51</td>
</tr>
<tr>
<td>1.00</td>
</tr>
</tbody>
</table>

- Boundary value
- Interior values
- Boundary value
HJ code for One-Dimensional Iterative Averaging using nested for-forall structure (Listing 3)

1. double[] myVal = new double[n]; myVal[0] = 0; myVal[n+1] = 1;
2. double[] myNew = new double[n]; double[] temp = null;
3. int batchSize = CeilDiv(n,t); // Number of elements per task
4. for (point [iter] : [0:iterations-1]) {
5.   forall (point [i] : [0:t-1]) { // Create t tasks
6.     int start = i*batchSize + 1;
7.     for (point[j] : [start:Math.min(start+batchSize-1,n)])
8.       myNew[j] = (myVal[j-1] + myVal[j+1])/2.0;
9.   } // forall
10.  temp = myNew; myNew = myVal; myVal = temp; // swap(myNew, myVal)
11.} // for

How many tasks does this version create?
HJ code for One-Dimensional Iterative Averaging using nested forall-for-all-next structure (Listing 4)

1. double[] val1 = new double[n]; val[0] = 0; val[n+1] = 1;
2. double[] val2 = new double[n];
3. int batchSize = CeilDiv(n,t); // Number of elements per task
4. forall (point [i] : [0:t-1]) { // Create t tasks
5.   double[] myVal = val1; double myNew = val2; double[] temp = null;
6.   int start = i*batchSize + 1; int end = Math.min(start+batchSize-1,n);
7.   for (point [iter] : [0:iterations-1]) {
8.     for (point[j] : [start:end])
9.       myNew[j] = (myVal[j-1] + myVal[j+1])/2.0;
10.    next; // barrier
11.   temp = myNew; myNew = myVal; myVal = temp; // swap(myNew, myVal)
12. } // for
13.} // forall
Extension: adding a print statement between phases with Two Barriers (Listing 5)

```java
forall (point [i] : [0:t-1]) { // Create t tasks
    
    for (point [iter] : [0:iterations-1]) {
        double sum = 0;
        for (point[j] : [start:end]) {
            myNew[j] = (myVal[j-1] + myVal[j+1])/2.0;
            sum += Math.abs(myNew[j] - myVal[i]);
        }
        tSum[i] = sum;
    next; // first barrier
    if (i == 0) {
        double sum = 0;
        for(point[k]:[0:t-1]) sum += tSum[k];
        System.out.println("Sum = " + sum + " for iteration " + iter);
    }
    next; // second barrier
}
} // forall
```
Next-with-Single Statement

next `<single-stmt>` is a barrier in which single-stmt is performed exactly once after all tasks have completed the previous phase and before any task begins its next phase.

Modeling next-with-single in the Computation Graph

```
A1  A2  A3  A4
\_\_\_\_\_

signal edges

next-start

↓

single-statement

↓

next-end

wait edges

A1  A2  A3  A4
\_\_\_\_\_
```
Use of next-with-single to add a print statement between phases (Listing 6)

```
forall (point [i] : [0:t-1]) { // Create t tasks
...
  for (point [iter] : [0:iterations-1]) {
    double sum = 0;
    for (point [j] : [start:end]) {
      myNew[j] = (myVal[j-1] + myVal[j+1])/2.0;
      sum += Math.abs(myNew[j] - myVal[i]);
    }
    tSum[i] = sum;
  next { // next-with-single statement replaces two barriers
    double sum = 0; for(point[k] : [0:t-1]) sum += tSum[k];
    System.out.println("Sum = " + sum + " for iteration " + iter);
  }
...
} // forall
```