COMP 322: Fundamentals of Parallel Programming

Lecture 12: Barrier synchronization in forall loops

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

https://wiki.rice.edu/confluence/display/PARPROG/COMP322
Solution to Worksheet #11: One-dimensional Iterative Averaging Example

1) Assuming \( n=9 \) and the input array below, perform one iteration of the iterative averaging example by only filling in the blanks for odd values of \( j \) in the \( \text{myNew}[\cdot] \) array. Recall that the computation is “\( \text{myNew}[j] = (\text{myVal}[j-1] + \text{myVal}[j+1])/2.0; \)”

<table>
<thead>
<tr>
<th>Index, ( j )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{myVal} )</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
<td>0</td>
<td>0.4</td>
<td>0</td>
<td>0.6</td>
<td>0</td>
<td>0.8</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( \text{myNew} )</td>
<td>0</td>
<td>0.1</td>
<td>0.2</td>
<td>0.3</td>
<td>0.4</td>
<td>0.5</td>
<td>0.6</td>
<td>0.7</td>
<td>0.8</td>
<td>0.9</td>
<td>1</td>
</tr>
</tbody>
</table>

2) Will the contents of \( \text{myVal}[\cdot] \) and \( \text{myNew}[\cdot] \) change in further iterations, after \( \text{myNew} \) above in 1) becomes \( \text{myVal}[\cdot] \) in the next iteration?

No, this represents the converged value (equilibrium/fixpoint).
HJ code for One-Dimensional Iterative Averaging using nested for-finish-forasync structure (Recap)

1. for (point [iter] : [0:m-1]) {
2.   // Compute MyNew as function of input array MyVal
3.   finish forasync (point [j] : [1:n]) { // Create n tasks
4.       myNew[j] = (myVal[j-1] + myVal[j+1])/2.0;
5.   } // finish forasync
6.   temp=myVal; myVal=myNew; myNew=temp; // Swap myVal & myNew;
7.   // myNew becomes input array for next iteration
8. } // for

Question: How many async tasks does this program create as a function of m and n?

Answer: m*n. Can we do better with chunking?
Example: HJ code for One-Dimensional Iterative Averaging with chunked for-finish-forasync-for structure (Recap)

1. int nc = Runtime.getNumOfWorkers();
2. for (point [iter] : [0:m-1]) {
3.     // Compute MyNew as function of input array MyVal
4.     finish forasync (point [jj] : [0:nc-1]) {
5.         for (point [j] : getChunk([1:n],nc,jj)) {
6.             myNew[j] = (myVal[j-1] + myVal[j+1])/2.0;
7.         } // finish forasync
8.     temp=myVal; myVal=myNew; myNew=temp; // Swap myVal & myNew;
9.     } // forasync
10. // myNew becomes input array for next iteration
11. } // for

Question: How many async tasks does this program create as a function of m, n, and nc?

Answer: m*nc. But we can do even better with “forall” loops and “barrier” synchronization.
Outline of Today’s Lecture

• **Barrier Synchronization in Forall Loops**

**Acknowledgments**

• COMP 322 Module 1 handout, Sections 10.1, 10.2, 10.4.
HJ’s forall statement = finish + forasync + barriers

Goal 1 (minor): replace common finish-forasync idiom by forall e.g., replace

\[
\text{finish forasync (point } [I,J] : [0:N-1,0:N-1]) \\
\text{for (point}[K] : [0:N-1]) \\
\]

by

\[
\text{forall (point } [I,J] : [0:N-1,0:N-1]) \\
\text{for (point}[K] : [0:N-1]) \\
\]

Goal 2 (major): Also support “barrier” synchronization

- Caveat: forall is only supported on the work-sharing runtime because of barrier synchronization
Hello-Goodbye Forall Example (Listing 33)

forall (point[i] : [0:m-1]) {
    int sq = i*i;
    System.out.println("Hello from task with square = \" + sq);
    System.out.println("Goodbye from task with square = \" + sq);
}

• Sample output for m = 4
  Hello from task with square = 0
  Hello from task with square = 1
  Goodbye from task with square = 0
  Hello from task with square = 4
  Goodbye from task with square = 4
  Goodbye from task with square = 1
  Hello from task with square = 9
  Goodbye from task with square = 9
Hello-Goodbye Forall Example (contd)

forall (point[i] : [0:m-1]) {
    int sq = i*i;
    System.out.println("Hello from task with square = “ + sq);
    System.out.println("Goodbye from task with square = “ + sq);
}

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

• Statements in red below will need to be moved to solve this problem

    Hello from task with square = 0
    Hello from task with square = 1
    Goodbye from task with square = 0
    Hello from task with square = 4
    Goodbye from task with square = 4
    Goodbye from task with square = 1
    Hello from task with square = 9
    Goodbye from task with square = 9
Hello-Goodbye Forall Example (contd)

1. forall (point[i] : [0:m-1]) {
2.   int sq = i*i;
3.   System.out.println("Hello from task with square = " + sq);
4.   System.out.println("Goodbye from task with square = " + sq);
5. }

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

• Approach 1: Replace the forall loop by two forall loops, one for the hello’s and one for the goodbye’s

—Problem: Need to communicate local sq values from one forall to the next

1. // APPROACH 1
2. forall (point[i] : [0:m-1]) {
3.   int sq = i*i;
4.   System.out.println("Hello from task with square = " + sq);
5. }
6. forall (point[i] : [0:m-1]) {
7.   System.out.println("Goodbye from task with square = " + sq);
8. }
Hello-Goodbye Forall Example (contd)

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

- Approach 2: insert a “barrier” between the hello’s and goodbye’s
  —“next” statement in HJ’s forall loops

1. // APPROACH 2
2. forall (point[i] : [0:m-1]) {
3.     int sq = i*i;
4.     System.out.println("Hello from task with square = " + sq);
5.     next; // Barrier
6.     System.out.println("Goodbye from task with square = " + sq);
7. }

- 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 next is the closest enclosing forall statement
  —Special case of “phaser” construct (will be covered later in class)
Impact of barrier on scheduling forall iterations

Modeling a next operation in the computation graph

Barrier Region

Phase 0

Phase 1

Forall iterations

Phase 0

Phase 1

time

SIG

idle

WAIT

i=0

i=1

i=2

i=3

A1

A2

A3

A4

signal edges

wait edges
Observation 1: Scope of synchronization for “next” is closest enclosing forall statement

```java
1. forall (point [i] : [0:m-1]) {
2.   System.out.println("Starting forall iteration " + i);
3.   next; // Acts as barrier for forall-i
4.   forall (point [j] : [0:n-1]) {
5.     System.out.println("Hello from task (" + i + "," + j + ")");
6.   }
7.   next; // Acts as barrier for forall-j
8.   System.out.println("Goodbye from task (" + i + "," + j + ")");
9. }
10. // forall-j
11. next; // Acts as barrier for forall-i
12. System.out.println("Ending forall iteration " + i);
13. } // forall-i
```
Observation 2: If a forall iteration terminates before “next”, then other iterations do not wait for it

1. `forall (point[i] : [0:m-1]) {`
2. `for (point[j] : [0:i]) {
3.     // Forall iteration i is executing phase j
4.     System.out.println("(" + i + "," + j + ")");
5.     next;
6. }
7. }

- Outer forall-i loop has m iterations, 0…m-1
- Inner sequential j loop has i+1 iterations, 0…i
- Line 4 prints (task,phase) = (i, j) before performing a next operation.
- Iteration i = 0 of the forall-i loop prints (0, 0), performs a next, and then terminates. Iteration i = 1 of the forall-i loop prints (1,0), performs a next, prints (1,1), performs a next, and then terminates. And so on.
Illustration of previous example

- Iteration $i=0$ of the forall-$i$ loop prints $(0, 0)$ in Phase 0, performs a next, and then ends Phase 1 by terminating.

- Iteration $i=1$ of the forall-$i$ loop prints $(1, 0)$ in Phase 0, performs a next, prints $(1,1)$ in Phase 1, performs a next, and then ends Phase 2 by terminating.

- And so on until iteration $i=8$ ends an empty Phase 8 by terminating.

Interesting figure. Try out another one in Worksheet 12!
Observation 3: Different forall iterations may perform “next” at different program points (barrier matching problem)

1. \texttt{forall (point}[i]\texttt{: [0:m-1])} \{ \\
2. \quad \texttt{if} (i \% 2 == 1) \{ // i is odd \\
3. \quad \quad \texttt{oddPhase0}(i); \\
4. \quad \quad \texttt{next;} \\
5. \quad \quad \texttt{oddPhase1}(i); \\
6. \quad \} \texttt{else} \{ // i is even \\
7. \quad \quad \texttt{evenPhase0}(i); \\
8. \quad \quad \texttt{next;} \\
9. \quad \texttt{evenPhase1}(i); \\
10. \} // if-else \\
11. \} // forall

- Barrier operation synchronizes odd-numbered iterations at line 4 with even-numbered iterations in line 8
- next statement may even be in a method such as oddPhase1()
One-Dimensional Iterative Averaging with Barrier Synchronization

1. `double[] gVal = new double[n+2]; double[] gNew = new double[n+2]; gVal[n+1] = 1;`
2. `int nc = Runtime.getRuntime().getNumWorkers();`
3. `forall (point [jj]:[0:nc-1]) { // Chunked forall is now the outermost loop`
4. `double[] myVal = gVal; double[] myNew = gNew; // Local copy of myVal/myNew pointers`
5. `for (point [iter]:[0:m-1]) {`
   6. `// Compute MyNew as function of input array MyVal`
   7. `for (point [j]:getChunk([1:n],nc,jj)) // Iterate within chunk`
   8. `myNew[j] = (myVal[j-1] + myVal[j+1])/2.0;`
   9. `next; // Barrier before executing next iteration of iter loop`
10. `// Swap myVal and myNew (each forall iterations swaps its pointers in local vars)`
11. `double[] temp = myVal; myVal = myNew; myNew = temp;`
12. `// myNew becomes input array for next iter`
13. `}` // for
14. `}` // forall

- Use of barrier reduces number of async tasks created to just `nc`
- However, these `nc` tasks perform `nc * m` barrier operations
  — Good trade-off since, barrier operations have lower overhead than task creation if number of chunks ≤ number of workers
1) Draw a “barrier matching” figure similar to slide 14 for the code fragment below.

1. String[] a = { “ab”, “cde”, “f” };  
2. . . . int m = a.length; . . .  
3. forall (point[i] : [0:m-1]) {  
4. for (int j = 0; j < a[i].length(); j++) {  
5. // forall iteration i is executing phase j  
6. System.out.println("(" + i + "," + j + ")");  
7. next;  
8. }  
9. }