COMP 322: Fundamentals of Parallel Programming

Lecture 32: Barrier Synchronization with Phasers

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

http://comp322.rice.edu
Barrier Synchronization: Hello-Goodbye Forall Example (Pseudocode)

forall (0, m - 1, (i) -> {
    int sq = i*i; // NOTE: video used lookup(i) instead
    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 (0, m - 1, (i) - > {
    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)

forall (0, m - 1, (i) -> {
    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?
• Approach 1: Replace the forall loop by two forall loops, one for the hello’s and one for the goodbye’s

— What’s the problem here?

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

```java
forall (0, m - 1, (i) -> {
    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?

• *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 first forall to the second

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

forall (0, m - 1, (i) -> {
    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?

• Approach 2: Replace the forall loop by two forall loops, one for the hello’s and one for the goodbye’s

— What’s the problem here?

1. // APPROACH 2
2. int[] sq = new int[m];
3. forall (0, m - 1, (i) -> {
4.    sq[i] = i*i;
5.    System.out.println("Hello from task with square = " + sq[i]);
6. });
7. forall (0, m - 1, (i) -> {
8.    System.out.println("Goodbye from task with square = " + sq[i]);
9. });
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, without having to change the local variable?

• Approach 3: insert a “barrier” ("next" statement) between the hello’s and goodbye’s

1. // APPROACH 3
2. forallPhased (0, m - 1, (i) -> {
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 forallPhased iteration waits at barrier until all iterations arrive (previous phase is completed), after which the next phase can start
  — Scope of next is the closest enclosing forallPhased statement
  — If a forallPhased iteration terminates before executing “next”, then the other iterations don’t wait for it
Impact of barrier on scheduling forallPhased iterations

Four forallPhased iterations, each with a next() barrier

Phase 0

Phase 1

\[
\text{next()} = \text{SIG} + \text{WAIT}
\]

\[
i = 0 \quad \text{//A1}
\]

\[
i = 1 \quad \text{//A2}
\]

\[
i = 2 \quad \text{//A3}
\]

\[
i = 3 \quad \text{//A4}
\]

next() operation is modeled in the Computation Graph using \textit{signal} and \textit{wait} edges
forallPhased API’s in HJlib

http://www.cs.rice.edu/~vs3/hjlib/doc/edu/rice/hj/Module0.html

- static void `forallPhased` (int s0, int e0, edu.rice.hj.api.HjProcedure<java.lang.Integer> body)
- static <T> void `forallPhased` (java.lang.Iterable<T> iterable, edu.rice.hj.api.HjProcedure<T> body)
- static void `next`()

**NOTE:**
- All `forallPhased` API’s include an implicit finish at the end (just like a regular `forall`)
- Calls to `next()` are only permitted in `forallPhased()`, not in `forall()`
Observation 1: Scope of synchronization for “next” barrier is its closest enclosing forallPhased statement

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

```java
forallPhased (0, m - 1, (i) -> {
    forseq (0, i, (j) -> {
        //forall iteration i is executing phase j
        System.out.println("(" + i + "," + j + ")");
        next();
    }); //forseq-j
}); //forall-i
```

- 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.
### Barrier Matching for previous example

<table>
<thead>
<tr>
<th>i=0</th>
<th>i=1</th>
<th>i=2</th>
<th>i=3</th>
<th>i=4</th>
<th>i=5</th>
<th>i=6</th>
<th>i=7</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,0)</td>
<td>(1,0)</td>
<td>(2,0)</td>
<td>(3,0)</td>
<td>(4,0)</td>
<td>(5,0)</td>
<td>(6,0)</td>
<td>(7,0)</td>
</tr>
<tr>
<td>next</td>
<td>next</td>
<td>next</td>
<td>next</td>
<td>next</td>
<td>next</td>
<td>next</td>
<td>next</td>
</tr>
<tr>
<td>(1,1)</td>
<td>(2,1)</td>
<td>(3,1)</td>
<td>(4,1)</td>
<td>(5,1)</td>
<td>(6,1)</td>
<td>(7,1)</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td>next</td>
<td>next</td>
<td>next</td>
<td>next</td>
<td>next</td>
<td>next</td>
<td>next</td>
</tr>
<tr>
<td>(2,2)</td>
<td>(3,2)</td>
<td>(4,2)</td>
<td>(5,2)</td>
<td>(6,2)</td>
<td>(7,2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>end</td>
<td>next</td>
<td>next</td>
<td>next</td>
<td>next</td>
<td>next</td>
<td>next</td>
<td>next</td>
</tr>
<tr>
<td>(3,3)</td>
<td>(4,3)</td>
<td>(5,3)</td>
<td>(6,3)</td>
<td>(7,3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>end</td>
<td>next</td>
<td>next</td>
<td>next</td>
<td>next</td>
<td>next</td>
<td>next</td>
<td>next</td>
</tr>
<tr>
<td>(4,4)</td>
<td>(5,4)</td>
<td>(6,4)</td>
<td>(7,4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>end</td>
<td>next</td>
<td>next</td>
<td>next</td>
<td>next</td>
<td>next</td>
<td>next</td>
<td>next</td>
</tr>
<tr>
<td>(5,5)</td>
<td>(6,5)</td>
<td>(7,5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>end</td>
<td>next</td>
<td>next</td>
<td>next</td>
<td>next</td>
<td>next</td>
<td>next</td>
<td>next</td>
</tr>
<tr>
<td>(6,6)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>end</td>
<td>next</td>
<td>next</td>
<td>next</td>
<td>next</td>
<td>next</td>
<td>next</td>
<td>next</td>
</tr>
</tbody>
</table>

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

$i=0 \ldots 7$ are forall iterations

(i,j) = printin output

next = barrier operation

end = termination of a forall iteration

Phase 0

Phase 1

Phase 2

Phase 3

Phase 4

Phase 5

Phase 6

Phase 7

Phase 8
Observation 3: Different forallPhased iterations may perform “next” at different program points

1. `forallPhased (0, m-1, (i) -> {
2.    if (i % 2 == 1) { // i is odd
3.       oddPhase0(i);
4.       next();
5.       oddPhase1(i);
6.    } else { // i is even
7.       evenPhase0(i);
8.       next();
9.       evenPhase1(i);
10.  } // if-else
11. }); // forall

- Barrier operation synchronizes odd-numbered iterations at line 4 with even-numbered iterations in line 8
- One reason why barriers are “less structured” than finish, async, future
Parallelizing loops in Matrix Multiplication example using forall

1. // Parallel version using forall
2. forall(0, n-1, 0, n-1, (i, j) -> {
3.     c[i][j] = 0;
4. });
5. forall(0, n-1, 0, n-1, (i, j) -> {
6.     forseq(0, n-1, (k) -> {
7.         c[i][j] += a[i][k] * b[k][j];
8.     });
9. });
10. // Print first element of output matrix
11. println(c[0][0]);

\[ c[i,j] = \sum_{0 \leq k < n} a[i,k] \times b[k,j] \]
Parallelizing loops in Matrix Multiplication example using forall

1. // Parallel version using forall
2. forallPhased(0, n-1, 0, n-1, (i, j) -> {
3. c[i][j] = 0;
4. next();
5. forseq(0, n-1, (k) -> {
6. c[i][j] += a[i][k] * b[k][j];
7. });
8. });
9. // Print first element of output matrix
10. println(c[0][0]);

\[ c[i,j] = \sum_{0 \leq k < n} a[i,k] \times b[k,j] \]
Announcements & Reminders

• Quiz #7 is due Wednesday, April 6th at 11:59pm
• Hw #5 CP 1 is due Friday, April 8th at 11:59pm