COMP 322: Fundamentals of Parallel Programming

Lecture 13: Barriers (contd), Iterative Averaging Revisited

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

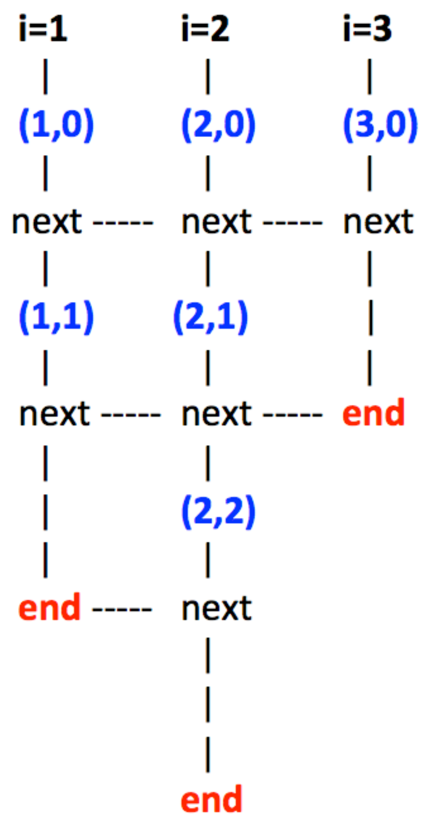
https://wiki.rice.edu/confluence/display/PARPROG/COMP322

Worksheet #12: Forall Loops and Barriers

Draw a "barrier matching" figure similar to slide 23 for the code fragment below.

```
String[] a = { "ab", "cde", "f" };
. . . int m = a.length; . . .
forallPhased (0, m-1, (i) -> {
  for (int j = 0; j < a[i].length(); j++) {
    // forall iteration i is executing phase j
    System.out.println("(" + i + "," + j + ")");
    next();
  }
});
```

Solution



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

```
forallPhased (0, m - 1, (i) -> {
 println(“Starting forall iteration ” + i);
 next(); // Acts as barrier for forall-i
 forallPhased (0, n - 1, (j) -> {
   println(“Hello from task (“ + i + “,” + j + “)”);
   next(); // Acts as barrier for forall-j
   println(“Goodbye from task (“ + i + “,” + j + “)”);
 } // forall-j
 next(); // Acts as barrier for forall-i
 println(“Ending forall iteration ” + i);
}); // forall-i
```

Observation 2: When a forall iteration terminates, other iterations do not wait for it at future barriers

```
1.  forallPhased (0, m - 1, (i) -> {
2.    forseq (0, i, (j) -> {
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.

Observation 3: Different forall iterations may perform “next” at different program points (barrier matching problem)

```
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
- next statement may even be in a method such as oddPhase1()

One-Dimensional Iterative Averaging Example Revisited

- Initialize a one-dimensional array of (n+2) double's with boundary conditions, myVal[0] = 0 and myVal[n+1] = 1.
- In each iteration, each interior element 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 myVal[i] = i/(n+1)
  - In this case, myVal[i] = (myVal[i-1]+myVal[i+1])/2, for all i in 1..n

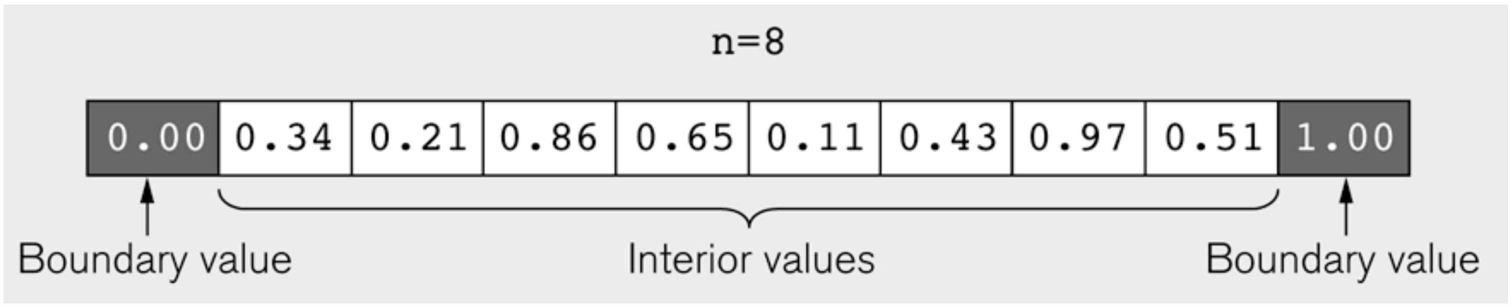


Illustration of an intermediate step for n = 8 (source: Figure 6.19 in Lin-Snyder book)

HJ code for One-Dimensional Iterative Averaging with forseq-forall structure

```
double[] myVal=new double[n+2]; myVal[n+1] = 1;
double[] myNew=new double[n+2];
forseq(0, m-1, (iter) -> {
   // Compute MyNew as function of input array MyVal
      forall(1, n, (j) -> { // Create n tasks
        myNew[j] = (myVal[j-1] + myVal[j+1])/2.0;
      }); // forall
  temp=myVal; myVal=myNew; myNew=temp;// Swap myVal & myNew;
  // myNew becomes input array for next iteration
}); // for
```

This program creates m*n async tasks

HJ code for One-Dimensional Iterative Averaging with forseq-forallChunked structure

```
double[] myVal=new double[n+2]; myVal[n+1] = 1;
double[] myNew=new double[n+2];
int nc = numWorkerThreads();
forseq(0, m-1, (iter) -> {
   // Compute MyNew as function of input array MyVal
   forallChunked(1, n, n/nc, (j) -> { // Create nc tasks
       myNew[j] = (myVal[j-1] + myVal[j+1])/2.0;
   }); // forallChunked
  temp=myVal; myVal=myNew; myNew=temp;// Swap myVal & myNew;
  // myNew becomes input array for next iteration
}); // for
```

This program creates m*nc async tasks

HJ code for One-Dimensional Iterative Averaging with forall-forseq structure and barriers

```
double[] gVal=new double[n+2]; gVal[n+1] = 1;
double[] gNew=new double[n+2];
forallPhased(1, n, (j) -> { // Create nc tasks
  // Initialize myVal and myNew as local pointers
  double[] myVal = gVal; double[] myNew = gNew;
  forseq(0, m-1, (iter) -> {
    // Compute MyNew as function of input array MyVal
    myNew[j] = (myVal[j-1] + myVal[j+1])/2.0;
    next(); // Barrier before executing next iteration of iter loop
    // Swap local pointers, myVal and myNew
    double[] temp=myVal; myVal=myNew; myNew=temp;
    // myNew becomes input array for next iteration
  }); // forseq
}); // forall
```

This program creates n async tasks, and performs _m_ barrier operations per task

HJ code for One-Dimensional Iterative Averaging with grouped forall-forseq structure and barriers

```
double[] gVal=new double[n+2]; gVal[n+1] = 1;
double[] gNew=new double[n+2];
HjRegion1D iterSpace = newRectangularRegion1D(1,m);
int nc = numWorkerThreads();
forallPhased(1, nc, (jj) -> { // Create nc tasks
  // Initialize myVal and myNew as local pointers
  double[] myVal = gVal; double[] myNew = gNew;
  forseq(0, m-1, (iter) -> {
    forseq(myGroup(jj,iterSpace,nc), (j) -> {
      // Compute MyNew as function of input array MyVal
      myNew[j] = (myVal[j-1] + myVal[j+1])/2.0;
    }); // forseq
    next(); // Barrier before executing next iteration of iter loop
    // Swap local pointers, myVal and myNew
    double[] temp=myVal; myVal=myNew; myNew=temp;
    // myNew becomes input array for next iter
  }); // forseq
}); // forall
```

This program creates nc async tasks, and performs *m* barrier operations per task

Single Program Multiple Data (SPMD) Parallel Programming Model

Basic idea

- Run the same code (program) on P workers
- Use the "rank" --- an ID ranging from 0 to (P-1) --- to determine what data is processed by which worker
    - —Hence, "single-program" and "multiple-data"
    - —Rank is equivalent to index in a top-level "forall (point[i] : [0:P-1])" loop
- Lower-level programming model than dynamic async/finish parallelism
    - —Programmer's code is essentially at the worker level (each forall iteration is like a worker), and work distribution is managed by programmer by using barriers and other synchronization constructs
    - —Harder to program but can be more efficient for restricted classes of applications (e.g. for OneDimAveraging, but not for nqueens)
- Convenient for hardware platforms that are not amenable to efficient dynamic task parallelism
    - —General-Purpose Graphics Processing Unit (GPGPU) accelerators
    - —Distributed-memory parallel machines

HJ code for One-Dimensional Iterative Averaging with grouped forall-forseq structure and barriers

```
double[] gVal=new double[n+2]; gVal[n+1] = 1;
double[] gNew=new double[n+2];
HjRegion1D iterSpace = newRectangularRegion1D(1,m);
int nc = numWorkerThreads();
forallPhased(1, nc, (jj) -> { // Create nc tasks
   // Initialize myVal and myNew as local pointers
   double[] myVal = gVal; double[] myNew = gNew;
   forseq(0, m-1, (iter) -> {
     forseq(myGroup(jj,iterSpace,nc), (j) -> {
       // Compute MyNew as function of input array MyVal
       myNew[j] = (myVal[j-1] + myVal[j+1])/2.0;
      }); // forseq
     next(); // Barrier before executing next iteration of iter loop
     // Swap local pointers, myVal and myNew
     double[] temp=myVal; myVal=myNew; myNew=temp;
     // myNew becomes input array for next iter
  }); // forseq
}); // forall
```

Instead of async-finish, this SPMD version creates one task per worker, uses myGroup() to distribute work, and use barriers to synchronize workers.