

---

# COMP 322: Fundamentals of Parallel Programming

## Lecture 12: Iterative Averaging Revisited, Single-Program Multiple-Data (SPMD) pattern

Vivek Sarkar, Eric Allen  
Department of Computer Science, Rice University

Contact email: [vsarkar@rice.edu](mailto:vsarkar@rice.edu)

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



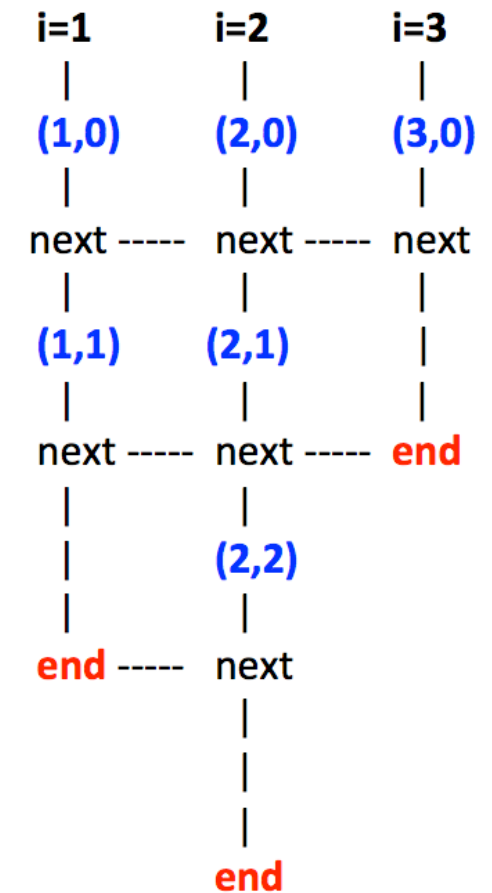
# Worksheet #11: forall Loops and Barriers

Draw a “barrier matching” figure similar to slide 13 for the code fragment below.

```

1. String[] a = { “ab”, “cde”, “f” };
2. . . . int m = a.length; . . .
3. forallPhased (0, m-1, (i) -> {
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. });
    
```

## Solution



# 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  $\text{myVal}[i] = (\text{myVal}[i-1] + \text{myVal}[i+1])/2$ , for all  $i$  in  $1..n$

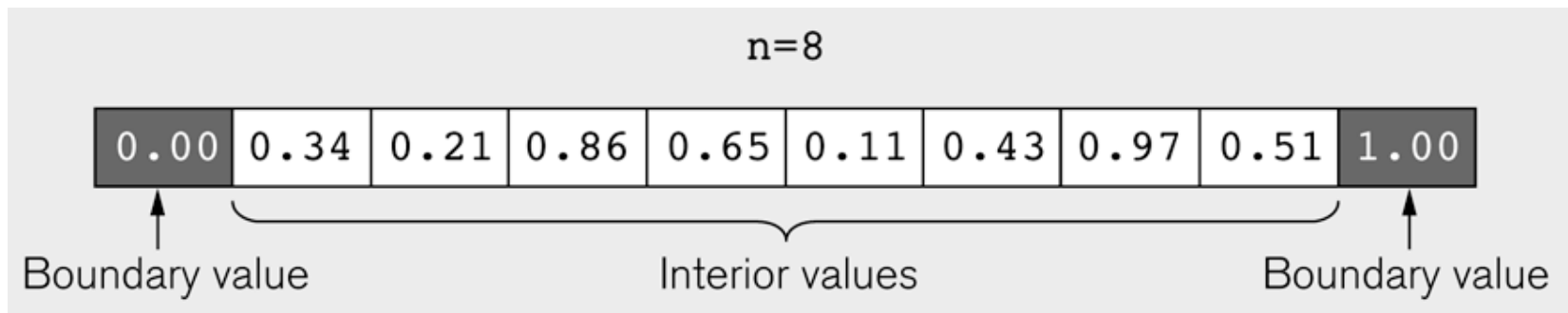


Illustration of an intermediate step for  $n = 8$  (source: Figure 6.19 in Lin-Snyder book)



# Iterative Averaging is similar to a Finite Difference solution to the One-Dimensional Heat Equation

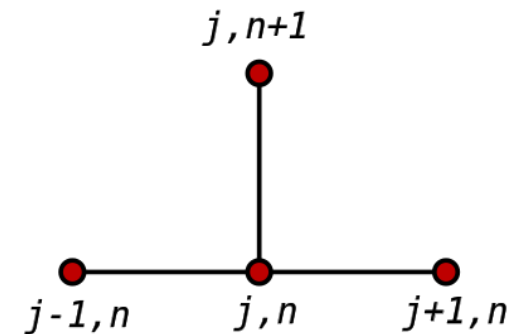
- Using a forward difference at time and a second-order central difference for the space derivative at position (FTCS) we get the recurrence equation:

$$\frac{u_j^{n+1} - u_j^n}{k} = \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{h^2}.$$

- This is an explicit method for solving the one-dimensional heat equation.
- We can obtain from the other values this way:

$$u_j^{n+1} = (1 - 2r)u_j^n + ru_{j-1}^n + ru_{j+1}^n$$

- where  $r = k/h^2$



- So, with this recurrence relation, and knowing the values at time n, one can obtain the corresponding values at time n+1.
- Source: [http://en.wikipedia.org/wiki/Finite\\_difference\\_method](http://en.wikipedia.org/wiki/Finite_difference_method)



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

---

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



# General Approach for Iteration Grouping (Loop Chunking)

---

**Parallel fork-join solution with grouped forall:**

```
for (iter : [0:iterations-1]) {  
  forall (g : [0:ng-1])  
    for(j : myGroup(g, [1:n], ng)  
      myNew[j] = (myVal[j-1] + myVal[j+1])/2;  
      Swap myNew and myVal  
    }  
}
```



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

---

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



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

```
1. double[] gVal=new double[n+2]; gVal[n+1] = 1;
2. double[] gNew=new double[n+2];
3. forallPhased(1, n, (j) -> { // Create n tasks
4.     // Initialize myVal and myNew as local pointers
5.     double[] myVal = gVal; double[] myNew = gNew;
6.     forseq(0, m-1, (iter) -> {
7.         // Compute MyNew as function of input array MyVal
8.         myNew[j] = (myVal[j-1] + myVal[j+1])/2.0;
9.         next(); // Barrier before executing next iteration of iter
    loop
10.        // Swap local pointers, myVal and myNew
11.        double[] temp=myVal; myVal=myNew; myNew=temp;
12.        // myNew becomes input array for next iteration
13.    }); // forseq
14. }); // forall
```





# General Approach for Iteration Grouping with Barriers

---

## Barrier-based solution:

```
// Note that iter-loop is inserted between forall-g and for-j loops
forall (g : [0:ng-1])
  for (iter : [0:iterations-1]) {
    for(j : myGroup(g, [1:n], ng)
      myNew[j] = (myVal[j-1] + myVal[j+1])/2;
    next; // Barrier
    Swap myNew and myVal
  } // for iter
```

Also referred to as a “single program multiple data” (SPMD) pattern



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

```
1. double[] gVal=new double[n+2]; gVal[n+1] = 1;
2. double[] gNew=new double[n+2];
3. HJRegion1D iterSpace = newRectangularRegion1D(1,n);
4. int nc = numWorkerThreads();
5. forallPhased(0, nc-1, (jj) -> { // Create nc tasks
6.     // Initialize myVal and myNew as local pointers
7.     double[] myVal = gVal; double[] myNew = gNew;
8.     foreach(0, m-1, (iter) -> {
9.         foreach(myGroup(jj,iterSpace,nc), (j) -> {
10.            // Compute MyNew as function of input array MyVal
11.            myNew[j] = (myVal[j-1] + myVal[j+1])/2.0;
12.        }); // foreach
13.    next(); // Barrier before executing next iteration of iter loop
14.    // Swap local pointers, myVal and myNew
15.    double[] temp=myVal; myVal=myNew; myNew=temp;
16.    // myNew becomes input array for next iter
17. }); // foreach
18. }); // forall
```



# 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-foreach structure and barriers

```
1. double[] gVal=new double[n+2]; gVal[n+1] = 1;
2. double[] gNew=new double[n+2];
3. HJRegion1D iterSpace = newRectangularRegion1D(1,m);
4. int nc = numworkerThreads();
5. forallPhased(1, nc, (jj) -> { // Create nc tasks
6.     // Initialize myVal and myNew as local pointers
7.     double[] myVal = gVal; double[] myNew = gNew;
8.     foreach(0, m-1, (iter) -> {
9.         foreach(myGroup(jj,iterSpace,nc), (j) -> {
10.            // Compute MyNew as function of input array MyVal
11.            myNew[j] = (myVal[j-1] + myVal[j+1])/2.0;
12.        }); // foreach
13.        next(); // Barrier before executing next iteration of iter loop
14.        // Swap local pointers, myVal and myNew
15.        double[] temp=myVal; myVal=myNew; myNew=temp;
16.        // myNew becomes input array for next iter
17.    }); // foreach
18. }); // forall
```

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

# Announcements

---

- **Reminder: Homework 2 is due by 5pm on Wednesday, February 11, 2015**
- **Worksheets can be submitted at the start of the next lecture, if needed**
- **Lecture handouts for Topic 3 should be coming soon!**
- **My office hours today will be during 3pm - 3:30pm in Duncan Hall room 3131**
  - Or meet me here after class

