coursera

4.3 Lecture Summary

4 Dataflow Synchronization and Pipelining

4.3 One-Dimensional Iterative Averaging with Phasers

Lecture Summary: In this lecture, we revisited the barrier-based Iterative Averaging example that we studied earlier, and observed that a full barrier is not necessary since *forall* iteration i only needs to wait for iterations $i-1$ and $i+1$ to complete their current phase before iteration i can move to its next phase. This idea can be captured by phasers, if we allocate an array of phasers as follows:

```
// Allocate array of phasers
Phaser[] ph = new Phaser[n+2]; //array of phasers
for (int i = 0; i < ph.length; i++) ph[i] = new Phaser(1);

// Main computation
forall ( i: [1:n-1]) {
  for (iter: [0:nsteps-1]) {
    newX[i] = (oldX[i-1] + oldX[i+1]) / 2;
    ph[i].arrive();

    if (index > 1) ph[i-1].awaitAdvance(iter);
    if (index < n-1) ph[i + 1].awaitAdvance(iter);
    swap pointers newX and oldX;
  }
}
```

As we learned earlier, grouping/chunking of parallel iterations in a *forall* can be an important consideration for performance (due to reduced overhead). The idea of grouping of parallel iterations can be extended to *forall* loops with phasers as follows:

```
// Allocate array of phasers proportional to number of chunked tasks
Phaser[] ph = new Phaser[tasks+2]; //array of phasers
for (int i = 0; i < ph.length; i++) ph[i] = new Phaser(1);

// Main computation
forall ( i: [0:tasks-1]) {
  for (iter: [0:nsteps-1]) {
    // Compute leftmost boundary element for group
    int left = i * (n / tasks) + 1;
    myNew[left] = (myVal[left - 1] + myVal[left + 1]) / 2.0;

    // Compute rightmost boundary element for group
    int right = (i + 1) * (n / tasks);
    myNew[right] = (myVal[right - 1] + myVal[right + 1]) / 2.0;

    // Signal arrival on phaser ph AND LEFT AND RIGHT ELEMENTS ARE AV
    int index = i + 1;
```