3.3 Lecture Summary

3 Loop Parallelism

3.3 Barriers in Parallel Loops

Lecture Summary: In this lecture, we learned the *barrier* construct through a simple example that began with the following *forall* parallel loop (in pseudocode):

```
1  forall (i : [0:n-1]) {
2          myId = lookup(i); // convert int to a string
3          print HELLO, myId;
4          print BYE, myId;
5  }
```

We discussed the fact that the HELLO's and BYE's from different *forall* iterations may be interleaved in the printed output, e.g., some HELLO's may follow some BYE's. Then, we showed how inserting a barrier between the two print statements could ensure that all HELLO's would be printed before any BYE's.

Thus, barriers extend a parallel loop by dividing its execution into a sequence of *phases*. While it may be possible to write a separate *forall* loop for each phase, it is both more convenient and more efficient to instead insert barriers in a single *forall* loop, e.g., we would need to create an intermediate data structure to communicate the myId values from one *forall* to another *forall* if we split the above *forall* into two (using the notation *next*) loops. Barriers are a fundamental construct for parallel loops that are used in a majority of real-world parallel applications.

Mark as completed