Prof. Jingke Li (FAB 120-06, lij@pdx.edu); Class: TTh 10:00-11:20 @ UTS 205; Lab: W 12:30-13:50, 14:00-15:20.

Lab 7: Sorting Algorithms

In this lab, you are going to implement several versions of the odd-even sort algorithm. Download and unzip the file `lab7.zip` from D2L. You'll see a `lab7` directory with some program files.

Odd-Even Sort in C

1. Review the odd-even sort algorithm from the lecture notes.
2. The file `oddeven0.c` contains some starter code. Copy it to `oddeven1.c` and complete the implementation. Follow the requirements specified in the comment blocks. You should try to match the sample output shown here:

```
linux> ./oddeven1
[Init] 7 10 5 8 2 15 3 1 14 4 16 13 9 6 11 12
[t= 1] 7 10 5 8 2 15 1 3 4 14 13 16 6 9 11 12
[t= 2] 7 5 10 2 8 1 15 3 4 13 14 6 16 9 11 12
[t= 3] 5 7 2 10 1 8 3 15 4 13 6 14 9 16 11 12
[t= 4] 5 2 7 1 10 3 8 4 15 6 13 9 14 11 16 12
[t= 5] 2 5 1 7 3 10 4 8 6 15 9 13 11 14 12 16
[t= 6] 2 1 5 3 7 4 10 6 8 9 15 11 13 12 14 16
[t= 7] 1 2 3 5 4 7 6 10 8 9 11 15 12 13 14 16
[t= 8] 1 2 3 4 5 6 7 8 10 9 11 12 15 13 14 16
[t= 9] 1 2 3 4 5 6 7 8 9 10 11 12 13 15 14 16
[t=10] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
[t=11] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
[t=12] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
[t=13] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
[t=14] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
[t=15] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
[t=16] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Result verified!
```

3. Compile your program and run several times. Do you observe the array being sorted before all iterations are executed (as shown in the above example)?
4. Now improve your code by adding an early termination detection in your program. The new program should terminate as soon as the array is sorted. Here is a sample output from the new version:

```
linux> ./oddeven1
[Init] 7 10 5 8 2 15 3 1 14 4 16 13 9 6 11 12
[t= 1] 7 10 5 8 2 15 1 3 4 14 13 16 6 9 11 12
[t= 2] 7 5 10 2 8 1 15 3 4 13 14 6 16 9 11 12
[t= 3] 5 7 2 10 1 8 3 15 4 13 6 14 9 16 11 12
[t= 4] 5 2 7 1 10 3 8 4 15 6 13 9 14 11 16 12
[t= 5] 2 5 1 7 3 10 4 8 6 15 9 13 11 14 12 16
[t= 6] 2 1 5 3 7 4 10 6 8 9 15 11 13 12 14 16
[t= 7] 1 2 3 5 4 7 6 10 8 9 11 15 12 13 14 16
[t= 8] 1 2 3 4 5 6 7 8 10 9 11 12 15 13 14 16
[t= 9] 1 2 3 4 5 6 7 8 9 10 11 12 13 15 14 16
[t=10] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
[t=11] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
[t=12] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Array sorted in 12 iterations
Result verified!
```

Odd-Even Sort in Chapel

Now we want to implement the odd-even sort algorithm in Chapel.

1. Since there is no built-in random number generator in Chapel, we'll keep using the C version. Read the program `extern.c` to see how to use C functions in a Chapel program. Note that there is no need to include any C header files.
2. Write a program `oddeven2.chpl` to implement the odd-even sort algorithm. You should try to port the C program `oddeven1.c` function-by-function to the Chapel version; parallelize code whenever possible. In the first try, you should omit the early-termination code.
3. If your base implementation is successful, you should try to add early-termination code to your Chapel program.

   Hint: Some form of reduce operation might be needed. In an earlier lecture we showed an example of the `reduce` operation. The program `reduce.chpl` shows a new form of its usage — inside a `forall` construct.