Prof. Jingke Li (FAB 120-06, lij@pdx.edu); Classes: M 16:40-18:30, W 16:40-17:20, Labs: W 17:25-18:55, all @ FAB 88-10.

Lab 9: Programming with Chapel (Part 2)

1 Domain Maps and Locales

Read and understand the demo programs `domMap1.chpl` and `domMap2.chpl`. Compile and run them. Change some mapping parameters and observe the effects.

Write a program, `locales.chpl`, to print out the id, name, and logical-cores of all the processors used in the program's execution. Here is a sample run of this program:

```
linux> ./locales -nl 4
Locale 0: emperor.cs.pdx.edu (with 8 logical cores)
Locale 1: dege.cs.pdx.edu (with 8 logical cores)
Locale 2: chinstrap.cs.pdx.edu (with 8 logical cores)
Locale 3: african.cs.pdx.edu (with 8 logical cores)
```

2 File I/O

1. Read and understand the demo program `fileIO.chpl`. Compile and run it. Now assume the input file `"input"` contains single-byte integers. Modifiy the program to read and write these integers. (*Hint:* There should be 64 single-byte integers.)

2. Write a coarse-grained version of file I/O program, `fileIO2.chpl`. In this program, create a worker rouine to be run on each locale:

   ```
   for loc in Locales do
     on loc do worker();
   ```

   The worker routines run concurrently. Each worker routine declares a local array of size `N/numLocales` (where `N` is the integer count of the input file); it then opens a channel to the input file, and reads a proper section from the file to the array. View the input file as having `numLocales` equal-sized sections; since locale id starts from 0, the worker routine on locale `k` should read the `k+1`-th section.

   Afterwards, the worker routine performs similar actions for the output — opens a channel to the output file and writes its array to the corresponding section in the file.

3 Odd-Even Sort

The file `oddeven.chpl` contains an implement of the odd-even sort algorithm (shared-memory version), as discussed in class.

1. Convert the program to a distributed-memory version, `oddeven-dm.chpl`, by introducing domain maps. Add the following code to verify that array `a` is indeed partitioned across the locales:

   ```
   write("Locale:");
   for i in a.domain do
     writef("%2i", a[i].locale.id);
   writeln();
   ```

2. Write a new shared-memory version, `oddeven2.chpl`, to add early-termination to the algorithm.

3. Write another version, `oddeven3.chpl`, to read array data from a file, and to print the result to another file. This version may be based on either `oddeven.chpl` or `oddeven2.chpl`.

4 Jacobi and Gauss-Seidel

(Optional) Read and understand the Jacobi iteration program, `jacobi.chpl`. Now write a Gauss-Seidel iteration program, *i.e.*, using only a single array to hold data. (*Hint:* You may use a local variable inside the `forall` loop to temporarily hold some data to help catching the changes.)

Submission

As usual, write a short report, in plain text or pdf, summarize your work with this lab. Submit the report and the following program files, `fileIO2.chpl`, `oddeven[23]*.chpl` and `gauss-seidel.chpl` (if you have), through the "Lab9" submission folder on D2L (under th "Activities/Assignments" tab). You should submit your work before the week-end, *i.e.* Sunday 3/8.