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 5: Programming with Chapel (Part 1)

Download and unzip the file `lab5.zip` from D2L. You'll see a `lab5` directory with some program files.

Hello World

1. Three simple versions of the Hello-World program are in files, `hello[123].chpl`. Read, then compile and run these programs:

   ```
   linux> chpl -o hello1 hello1.chpl
   linux> ./hello1 -nl 1
   Hello, world!
   ...
   ```

   Note that to run a chapel program, you need to include the switch "`-nl 1`". This is because on our Linux system, the Chapel compiler is built to generate target code that is suitable for running across multiple locales. The switch "`-nl 1`" means running the program on a single locale.

   You could also use the `Makefile` to compile these programs all together:

   ```
   linux> make hello1 hello2 hello3
   linux> ./hello1 -nl 1
   Hello, world!
   ...
   ```

2. A forth version, `hello4.chpl`, contains a configurable variable, `message`. Read, then compile and run the program. Try changing the message through the command-line a couple of times:

   ```
   linux> make hello4
   linux> ./hello4 -nl 1
   Hello, world!
   linux> ./hello -nl 1 --message="Hi!"
   Hi!
   ```

3. The pair of files, `hello.chpl` and `hello-main.chpl`, contain a split version: one file defines a module, the other uses it. Again, read, then compile and run the program.

4. Finally, there are three task-based versions of the Hello-World program: `task[123].chpl`. Run these programs and see if you obverse concurrency.

Domains and Arrays

1. The files, `domain1.chpl` and `domain2.chpl`, contains some domain-related code. Read, then compile and run the programs. Change the program whatever way you want and see the results.

2. The file `mmul.c` contains a matrix multiplication program in C. Create a Chapel version of this program. Parallelizing as much as you can.

Task Queues

The file `taskQueue.chpl` contains the Chapel version of the task queue data structure that was used in Assignment 1. It is defined as a module. Read and understand the program.

1. Write a program `task-main.chpl` to use the module to do the following (in the given order):

   (a) create a task queue;

   (b) add three distinct tasks to the queue; and

   (c) remove two tasks from the queue.

   After each step, print out the content of the queue. A possible output is shown here:

   ```
   linux> ./task-main
   Init: {head = nil, tail = nil, limit = 0, length = 0}
   After 3 adds: {head = {val = 1, next = {val = 2, next = {val = 3,
     next = nil}}}, tail = {val = 3, next = nil}, limit = 0, length = 3}
   After 2 removes: {head = {val = 3, next = nil}, tail = {val = 3,
     next = nil}, limit = 0, length = 1}
   ```

2. The program `prodcons0.chpl` is a simple version of the producer-consumer program. Read and understand the program. Pay special attention to the synchronization mechanism. Compile and run this program.

   Currently, the program works only for one pair of producer and consumer. Change the program so that multiple concurrent consumers are created. The number of consumers should be controlled by a configurable constant `numCons`.