Prof. Jingke Li (FAB 120-06, lij@pdx.edu); Class: TR 16:40-17:55 @ ASRC 230; Lab: F 10:30-11:50 @ FAB 88-10.

Lab 6: Programming with Chapel (Part 1)

Download and unzip the file `lab6.zip` from D2L. You'll see a `lab6` directory with some program files.

1. 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.

2. Domains and Arrays

1. The files, `domain1.chpl` and `domain2.chpl`, contains some domain-related code. Read the code carefully; pay attention to the connection between array `a` and domain `D`. Compile and run the programs; notice how the changes on domain `D` affect array `a`.

   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.

3. Deposit/Withdraw Program

The file `bank.c` is the same as you saw in Lab 5; it contains a sequential program performing simple deposit and withdraw operations on a bank account. Convert this program into two versions of Chapel program.

1. A sequential version, `bank1.chpl`. The five constants should be converted to configurable runtime constants, i.e. `config const`. To generate a random number, use the `Random` module. A sample is given below:

   ```
   use Random;
   var rs = new RandomStream(uint); // create a random stream of unsigned int
   var val = rs.getNext();          // get a random unsigned int
   ```

   The C formatted print statement `printf` can be converted to `writef` in Chapel almost without change, except that the integer control string is `"%i"` instead of `"%d"`.

2. A parallel version, `bank2.chpl`. Convert both C `for` loops into Chapel `forall` loops, and make further changes so that deposit and withdraw operations can interleave.

Compile and run your Chapel program to verify that they work as expected.