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.

Assignment 3: Programming with Chapel

(Due Wednesday, 3/1/17)

This assignment is to practice programming in the new parallel programming language, Chapel. You are going to continue with Lab 5's work. Specifically, you are going to implement two application programs that you've already worked on, matrix-multiplication and producer-consumer. However, this time, the setting or requirements are a little different. For this assignment, both CS415 and CS515 students will do the same programming work. However, CS515 students will need to write a short report (see below). This assignment carries a total of 10 points.

Matrix Multiplication

The file `mmul.c` contains a matrix multiplication program in C. Create a Chapel version of this program, `mmul.chpl`. The specific requirements are:

- Represent the array dimension size parameter, `N`, by a configurable constant in the Chapel program. Set its default value to 8.
- Define a two-dimensional domain `D` to represent the array index set, {`0..N-1`} × {`0..N-1`}. Declare the three arrays, `a`, `b`, and `c`, over this domain.
- Parallelize *all* loops in `mmul.c`. Convert them to array operations, reduction operations, and/or parallel loops. The resulting Chapel program should have just one loop, a `forall` parallel loop for the multiplication section.
- Use the `script` command to create a run script of your program running with `N=8` (default), `N=16`, `N=32`, and `N=64`.

```
linux> script mmul-script.txt
Script started, file is mmul-script.txt
linux> ./mmul
total = 3584 (should be 3584)
linux> ./mmul --N=16
total = 61440 (should be 3584)
...
linux> exit
Script done, file is mmul-script.txt
```

Producer-Consumer

The file `circQueue1.chpl` contains an alternative representation of task queue data structure. Instead of a linked list, the queue items are stored in a circular buffer array. When the end of the buffer is reached, it continues back from the beginning.

Read and understand this program. Pay special attention to the `sync` declaration for the array, which means every array element is a self-sync item, allowing only alternating reads and writes.

A companion file `prodcons1.chpl` is a partially implemented producer-consumer program, with a single pair of producer and consumer threads.

Here are your tasks:

1. Complete the program `prodcons1.chpl` by providing the code for the `consumer()` routine. You are not allowed to modify other parts of the provided program.

   Requirements:

   - Keep track of how many items have been removed from the queue.
   - Include two `writeln` statements to print out messages that are in parallel to those in the `producer()` routine. Here is a sample:

     ```
     consumer removed <task> from queue
     consumer got endFlag, total tasks = <cnt>
     ```

   - Use the `script` command to create a run script of your program running with (1) the default setting; (2) `numTasks=40`, and (3) `buffSize=5` (with default `numTasks`).

2. The `circQueue1` task queue module can only handle a single consumer thread. However, with a small change, it can be extended to handle multiple consumer threads. Implement this extended version in a new file, `circQueue2.chpl`.

   Hint: Which variable may cause race condition when there are multiple consumer threads? Maybe you want to make it a `sync` variable?

3. Write a new producer-consumer program, `prodcons2.chpl`, which uses the `circQueue2` module to support a single producer running with multiple consumer threads.

   Requirements:

   - The number of consumers should be controlled by a configurable constant `numCons`. Set its default value to 2.
   - The `producer()` routine should add `numCons` copies of the special `endFlag` task to the queue after all regular tasks are added.
   - The `consumer()` routine should keep track of the number of tasks it has removed from the queue. Note that this is a local count, since multiple copies of this routine will be running. Before termination, it should add the local count to a global task count.
   - The `main()` function should make sure that the producer thread and the consumer threads are all running concurrently. It should print out the global task count at the end.
   - Create a run script of this program running with (1) the default setting; (2) `numCons=4`; (3) `numTasks=30` and `numCons=6`, and (4) `buffSize=5`, `numTasks=40`, and `numCons=8`.

Report (CS515 Students Only)

Write a short report answering the following questions:

1. There is no explicit checking on buffer being full or empty in `circQueue1.chpl`. How come? What happens if the buffer is full or empty?
2. There is no synchronization code in `prodcons1.chpl` at all. Why is that?
3. Is there a need to have any synchronization code in `prodcons2.chpl`? If your answer is "yes", explain what difference between these two programs that triggers the need.

Submission

Make a `zip` file containing your programs and scripts (and the report if you are a CS515 student). Use the Dropbox on the D2L site to submit your assignment file.