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.

# Assignment 3: Programming with Chapel

**(Due Tuesday, 2/27/18)**

This assignment is to practice programming in the new parallel programming language, Chapel. You are going to continue with Lab 6's work. Specifically, you are going to implement a matrix-multiplication program and a couple of producer-consumer programs. CS515 students need to do an extra part (see below). This assignment carries a total of 10 points.

## 1. 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 -nl 1
total = 3584 (should be 3584)
linux> ./mmul --N=16 -nl 1
total = 61440 (should be 3584)
...
linux> exit
Script done, file is mmul-script.txt
```

## 2. Producer-Consumer

The file `circQueue.chpl` contains a representation of circular queue data structure. The queue items are stored in a 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` variable declarations, especially the buffer array, which means every array element is a self-sync item, allowing only alternating reads and writes.

Your task is to write several produce-consumer programs. *Note:* You should not modify the provided `circQueue.chpl` program.

### Version 1: `prodcons1.chpl`

In this version you are to write a producer routine to add items to a queue, and a consumer routine to remove items from the same queue. The specific requirements are:

- The total number of items to add and remove is represented by a configurable constant `numItems`, with a default value of 32.
- Both routines print out a line for each item added or removed, in the following form:

  ```
  Producer added item 28 to buf[7]
  consumer removed item 21 to buf[0]
  ```

- The producer and the consumer routines must run concurrently. (When you run the program, you should see producer messages and consumer messages interleave.)

**Version 2:** `prodcons2.chpl`

In this version you are to modify the previous version to allow multiple copies of both producer and consumer routines to be created and run concurrently. Additional requirements are:

- Both the producer and consumer functions take an integer parameter, serving as an ID to allow differentiation among multiple copies of the same function. So messages from this program will look like

  ```
  Producer 2 added item 28 to buf[7]
  consumer 4 removed item 21 to buf[0]
  ```

- The numbers of producers and consumers are represented by configurable constants, `numProds` and `numCons`, respectively; both have a default value of 2.
- The workload of adding `numItems` items to the queue is evenly partitioned among the `numProds` producers; each handles `numItems/numProds` items.
- Similarly, the workload of removing `numItems` items from the queue is evenly partitioned among the `numCons` consumers; each handles `numItems/numCons` items.
- Your program should verify that both `numProds` and `numCons` evenly divide `numItems`, and reject non-conforming values.
- All producers and consumers should run concurrently.

**Version 3:** `prodcons3.chpl` [**CS515 Students**] (3 extra points for CS415 students)

In this version you are going to further enhance the program by using dynamic workload distribution. Specifically,

- There is no workload partition. All copies of the produce function will compete to add as many items to the queue as possible; each is capable of adding all `numItems` items.
- The same is true with the consumers.
- As such, there is no need to require that both `numProds` and `numCons` evenly divide `numItems`.
- However, there is a new challenge that you need to handle: each producer and consumer needs to know when to terminate.

Test your programs with different parameter values. Create a run script for showing some test cases.

## Submission

Make a `zip` file containing your programs and scripts. Use the Dropbox on the D2L site to submit your assignment file.