Mina Vu

Assignment 3 Report

CS515 Parallel Programming

Spring 2022

**Programming with Chapel**

For assignment 3, we were tasked with writing Chapel programs for two problems with two versions each. The first problem was the oddeven sort where we were given the sequential implementation of it in C code and must translate it to Chapel. Its second version included an early termination policy in which the program ends once the array is sorted. The second problem was the producer/consumer problem. This is a quintessential parallel programming problem that needs no explanation for this assignment. It uses a provided cqueue module in which a circular array holds items produced and to be consumed.

I was able to write complete programs for both problems for both versions. Parallelizable loops were written using forall loops. Shared variables were written with atomic variables and sync variables provided by the Chapel programming language. All programs run to completion without failures in its execution of the tasks or hanging. Configurable constants were used as instructed so that variables N, numItems, numCons, WORST, and DEBUG are changeable at the command line interface.

In my observation of the programs, the given oddeven C program initializes the array in the worst case by adding 1 to the integers so that items are n+1 to 2 instead of n to 1 like the specification said. This is because the index starts at 0. However, this code translated perfectly to the Chapel code because index there starts at 1 and is inclusive at both ends of the range. Unlike the sample outputs, I did not observe any failed oddeven sorts. The array always got sorted after all (N+1)/2 times. The early termination policy could save a few rounds of checking each odd and even pairs. As I observed, at most five rounds were saved when N was 50.

The basic producer/consumer Chapel code were very well interleaved in my observations of a few runs. Almost always, it interleaved one after another. The extended producer/consumer code also interleaved very well. In one execution with four consumers, I noticed that one consumer did not win any competition to remove an item. It was completely starved through the entire process. At first, I thought something was wrong, but once I ran the program a few more times and observed normalcy in behaviors of all consumers, I deemed that the consumer previously just had really bad luck in fighting for an item.

This assignment was daunting at first to start because I had to put it off for a little bit after completing lab 6. This meant that I had to review what I had learnt about Chapel first in order to start writing code again. Unsurprisingly, this was not too difficult of a process. The translation of oddeven C code to Chapel was a bit more difficult than my review because we had to use uint(8) bit integers. This was a bit foreign to me but I soon realized that it meant our random integers should only be in the range [0,256). Implementing this took several tries to get the casting and syntax correct. The producer/consumer problem was much easier to implement in Chapel than I had expected. I believe this is because Chapel was developed for parallelism and therefore concurrency and synchronization was made easy to implement. Overall, my experience with this assignment was very educational and not at all frustrating.