Programming Assignment:
Sort Pump Gophers

CS 377: Parallel Programming
Spring 2019

April 11, 2019

1 Administrative Details

Due: Fri, April 19, 2019

To be handed in: Your printed lab report, and printed source code.

Comments: Be sure to update the comments to indicate your name and the location of your source file.

Report: Your lab report will discuss your overall experience solving problems, any problems encountered, how solved, lessons learned, etc.

Starting Code: I've created starting code for you to copy from:
`/home/mlsmith/cs377-examples/Go/sort-pump.got`

2 Trivia / Explanation

I forgot to mention that the Go Programming Language has a mascot! It doesn't really have a name, other than the Go Gopher. You can read more about it here: https://blog.golang.org/gopher. For this reason, I named this assignment "Sort Pump Gophers". You can think of the goroutines (processes) as gophers working concurrently to sort a sequence of numbers.

3 Description

In class we discussed a parallel implementation of Bubble Sort as a process network that could sort a list of n numbers in order $O(n)$ time. Figure 1 contains a diagram depicting that process network.

The starter code lays out the functions you will need to fill in. The randNums, cell, and printSorted functions will all be run as goroutines (these are the gophers!), and the sortPump function will set up the process network. It launches the goroutines and creates the channels to wire up the process network.

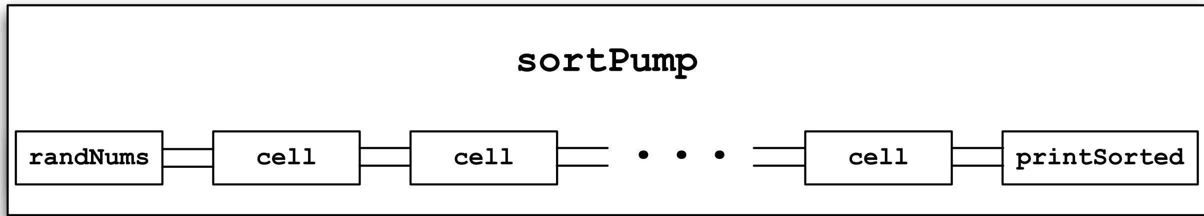

Figure 1: Process network for a sortPump. Two channels connecting each pair of processes–one to pass numbers being sorted, and the other a kill channel.

4 Advice

You will not only be writing this program using a new model of concurrency, you will be using a new language that is different enough from C and Java that you will need to consult the Go language references online frequently. That said, you should also use the two demo programs we walked through in class–dining philosophers and the sieve of Eratosthenes–for examples of Go language syntax, coding style, and how to create and use channels. It can sometimes be frustrating learning a new language, but as Computer Science majors, it should also be exciting!

Some specific pieces of advice based on my implementation experience:

- the layout of the process network for this program is most similar to the prime number sieve, so you might want to start with that program as a model.
- since each process is connected to one or two other processes via two channels, you will need to use Go's `select` statement to read safely from both channels in parallel.
- to get things started, the randNums process will generate random numbers and write them to a channel. you will find the starter code helpful in terms of the channels to use. you may assume this process generates a fixed number of random numbers, and then sends a kill signal (boolean) along the kill channel to let the first cell know it's finished.
- you will want to read up on how Go supports pseudorandom numbers. of note is that the random numbers are deterministic (for experimental reproducibility). if you want to test your program on different streams of random numbers, you will need to figure out how to initialize a different random seed each time, by reading the clock.
- each cell process will read numbers one at a time, and write the smaller of the last two numbers it's read along its output channel. when it reads a kill signal it passes the kill signal along to the next cell process. This has the effect of flushing the buffer.
- the final cell process sends its numbers to the printSorted process, until it's sent its last number. when it reads the kill signal, it passes it along as well.
- one of the challenges I had was my sortPump function returning before the process network completed its work. you will need to find a way to wait for the last number to be printed before terminating. I did that by having my sortPump function read the kill signal from the printSorted process, but there are other ways.

- one peculiarity of Go, which is really quite elegant once you get used to it, is how it distinguishes between declaring a variable while initializing it (using the `:=` operator) and just assigning a new value to a variable that has already been declared (using the `=` operator). watch out for that...
- if you have any other questions, please don't hesitate to email me or ask in class, or ask your classmates. we are a community of learners!

Finally, have fun! I look forward to reading your solutions, and reading about your experiences working on this assignment.