Levi Overcast
DSA F21
October 17

Parallel & Distributed Computing
Homework 2

1. Exercise 4.1 from Chapter 4 in Pacheco

   4.1 When we discussed matrix-vector multiplication we assumed that both m and n, the number of rows and the number of columns, respectively, were evenly divisible by t, the number of threads. How do the formulas for the assignments change if this is not the case? The formulas as given only assume that m is divisible by t, the number of threads. Each thread loops over every value of n. And this is all to the good since array are stored contiguously in memory, so dividing individual rows by threads would add unnecessary jumps to the code. However, assuming that the number of columns is evenly divisible by the thread count is not a general solution. Suppose there were a non-zero remainder in the number of rows. Then the corresponding entries in the output vector would end up unintentionally as zeros. To fix this, simply assign all as normal except give the last thread its normal portion plus any remainder.

   ```
   my_last_row = (my_id == nthreads - 1) ? my_first_row + stride
                                         : arrlen;
   ```

2. Exercise 4.2

   4.2 If we decide to physically divide a data structure among the threads, that is, if we decide to make various members local to individual threads, we need to consider at least three issues:

   a. How are the members of the data structure used by the individual threads? b. Where and how is the data structure initialized? c. Where and how is the data structure used after its members are computed?

   What would we have to do in order to divide A and y among the threads? Dividing y wouldn't be difficult–each thread could allocate a block of memory that could be used for storing its assigned components. Presumably, we could do the same for A–each thread could allocate a block of memory for storing its assigned rows. Modify the matrix-vector multiplication program so that it distributes both of these data structures. Can you "schedule" the input and output so that the threads can read in A and print out y? How does distributing A and y affect the run-time of the matrix-vector multiplication? (Don't include input or output in your run-time.)

Shared	Number of threads	Side length of square matrix	Time
Yes	1	10000	4.19617e-05
Yes	4	10000	7.00951e-05
No	1	10000	5.79357e-05
No	4	10000	8.4877e-05

3. Exercise 4.4
   4.4 The performance of the calculation program that uses mutexes remains roughly constant once we increase the number of threads beyond the num- ber of available CPUs. What does this suggest about how the threads are scheduled on the available processors?

4. Exercise 4.5
   4.5 Modify the mutex version of the calculation program so that the critical section is in the for loop. How does the performance of this version compare to the performance of the original busy-wait version? How might we explain this?

5. Exercise 4.6
   4.6 Modify the mutex version of the calculation program so that it uses a semaphore instead of a mutex. How does the performance of this version compare with the mutex version?

6. Exercise 4.8
   4.8 If a program uses more than one mutex, and the mutexes can be acquired in different orders, the program can deadlock. That is, threads may block forever waiting to acquire one of the mutexes. As an example, suppose that a program has two shared data structures–for example, two arrays or two linked lists–each of which has an associated mutex. Further suppose that each data structure can be accessed (read or modified) after acquiring the data structure's associated mutex. a. Suppose the program is run with two threads. Further suppose that the following sequence of events occurs: What happens? b. Would this be a problem if the program used busy-waiting (with two flag variables) instead of mutexes? c. Would this be a problem if the program used semaphores instead of mutexes?