Homework #4 CS 1645/2045

Due: Thursday, February 8, 2018 Spring 2017

Question 1:

Consider a program that consists of a large number of iterations, where the time to execute one iteration on a single processor is 1000 nsec. If this program is parallelized on two processors, each iteration would require (500 + M) nsec, where M is the time to exchange an x-byte message between the two processors. Assume that M = (50 + 10*x) nsec. This communication must occur once during each iteration.

a. What is the speedup and efficiency if x = 20 bytes
b. For what value of x (amount of communication per iteration) would executing the program on two processors result in a speedup of exactly 1? Meaning equal to serial execution.
c. For what value of x would executing the program on two processors result in an efficiency larger than 0.7.
d. Assume that it is possible to buffer data locally and group the communication such that only one message of 2x bytes is exchanged every two iterations, rather than a message of x bytes every iteration. How would this affect the speedup? Compare the speedup when x=20 bytes (and messages exchanged every two iterations) with the answer of part (a).

Question 2:

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. For example, suppose that a program has two shared linked lists (L1 and L2), each with an associated mutex (M1 and M2). If thread 0 holds the lock for L1 and subsequently waits for L2, and thread 1 holds the lock of L2 and subsequently waits for L1, they will wait forever. Here is multi-threaded example code that could lead to deadlock.

```
if (myrank % 2 == 0) {
  pthread_mutex_lock(&M1);
  insert(L1, myrank);
  pthread_mutex_lock(&M2);
  insert(L2, maxvalue(L1));
  pthread_mutex_unlock(&M2);
  pthread_mutex_unlock(&M1);
} else {
  pthread_mutex_lock(&M2);
  insert(L2, myrank+1);
  pthread_mutex_lock(&M1);
  insert(L1, minvalue(L2));
  pthread_mutex_unlock(&M1);
  pthread_mutex_unlock(&M2);
}
```

Assume minvalue and maxvalue functions return the minimum and maximum value in the linked list passed as the first argument.

a. Given the above code snippet and assuming two threads show an example of execution that would lead to a deadlock?
b. Would this still occur if we used a busy-waiting (with two flag variables) instead of mutexes?
c. How could we modify this code to ensure that a deadlock never occurred?

Question 3:
In this question we will be implementing a trapezoidal approximation method for determining the area under a curve (found in Section 3.2 of the book). In this question you will be implementing a pthread version of this approximation method.

a. There is an starter file (implemented serially) found in the git repository located here: https://github.com/bryanmills/hpc-course/tree/master/hw4
b. On comet you should be able to execute the following commands:
   `make`
   `sbatch trap.batch`
c. After your job has ran you should verify the output file. You can view your position in the queue using the following command:
   `squeue -u $USER`
d. Modify the trap.c file to parallelize the loop found in the `trap()` function. Continue to use the shared variable (`approx`) to calculate the approximation.
   i. Protect this variable using busy-waiting.
      1. Execute your parallelization using 2 threads, 50 threads and 100 threads. Note the execution time and accuracy.
   ii. Protect this variable using a mutex.
      1. Execute your parallelization using 2 threads, 50 threads and 100 threads. Note the execution time and accuracy.
e. What impact does having a local sum in the thread have upon the execution time?