CS121 Problem Set 3

Due: 23:59, December 4, 2024

1. Submit your solutions to Gradescope (www.gradescope.com).
2. In "Account Settings" in Gradescope, set FULL NAME to your Chinese name and enter your STUDENT ID.
3. If you submit handwritten solutions, write neatly and submit a clear scan.
4. When submitting your homework, be sure to match each of your solutions to the corresponding problem number.

1. Consider the same situation as in problem 1 of problem set 2. Give another algorithm for all-to-all broadcast which takes time $(t_s + t_w m)(p-1)$.

   Hint: Try to embed a p process ring in the tree.

2. We discussed barrier synchronization as a way to ensure all threads reach a point in the code before any of them progress beyond the point. The figure below shows a barrier for N threads implemented using locks. Explain clearly how this code works. What should the initial values of `count` and the `arrival` and `departure` locks be? Why is it necessary to use two lock variables?

```
void barrier()
{
    set_lock(arrival);
    count++;
    if (count < N)
        unset_lock(arrival);
    else
        unset_lock(departure);
    set_lock(departure);
    count--;
    if (count > 0)
        unset_lock(departure);
    else
        unset_lock(arrival);
}
```

3a. Consider the sequential code shown in the figure below. List all the dependence relationships in the code, indicating the type of dependence in each case. Draw the loop-carried dependence graph (LDG).

```
for (i=1; i<=n; i++)
  for (j=1; j<=n; j++) {
    S1: a[i][j] = a[i-1][j] + b[i][j];
    S2: b[i][j] = c[i][j]
  }
```

3b. Using the LDG derived in Q1a, describe how to obtain a parallel algorithm, explaining any modifications required to improve the efficiency. Express your parallel algorithm using OpenMP.

4. Consider the loop shown in the figure below. Indicate the loop-carried dependence relationships. By restructuring the code, is it possible to eliminate this dependence and parallelize the loop? Express your answer using OpenMP.

```
a[0] = 0;
for (i=1; i<=n; i++)
  S1: a[i] = a[i-1] + i;
```

5. One way to get a numerical approximation to π is to sum the following sequence:

$$\pi = 4(1 - 1/3 + 1/5 - 1/7 + \ldots)$$

A sequential algorithm to calculate π using this formula is given in the figure below. By analyzing the loop-carried dependences in this code, show how the algorithm may be parallelized, expressing your answer using OpenMP. Indicate clearly any modifications required to the code in order to allow parallelization and improve the efficiency.

```
double factor = 1.0;
double sum = 0.0;
for (i = 0; i < n; i++) {
    S1: sum += factor/(2*i+1);
    S2: factor = -factor;
}
pi = 4.0*sum;
```