EE/CSCI 451
Fall 2020
Homework 3 solution
Total Points: 100

1 [10 points]

Please refer to slides and textbook.

2 [30 points]

The Pthreads program discussed in class when converted into PRAM will cause multiple writes to same location C(i, k) in k-th iteration by threads $(i, 1 : n)$. We can have another for loop within each of the k iterations to serialize the thread accesses by threads $(i, 1 : n)$. The final program will take n^2 clocks.

A better approach is to rearrange the data accesses by threads to the elements in C. The program can be implemented in n clocks as shown by the program below.

```
Thread(i,j)
{
       Do k from 1 to n
        index = (k + j - 1) % (n+1) + floor((k + j - 1)/(n+1))
        C(i, index) = C(i, index) + A(i,j)*B(j, index)
       End
}
```

3 [10 points]

No, the parallel execution will not produce the same output A.
This is because the loop we are parallelizing has dependence within the loop (i.e., loop dependence). For example, in the serial version, $A[i][j-1]$ is always computed before $A[i][j]$; but in the parallel version, it is likely that $A[i][j-1]$ is computed after $A[i][j]$ due to the scheduling of threads; this is problematic because the computation of $A[i][j]$ depends on $A[i][j-1]$.

4 [20 points]

- Yes, deadlock is possible. It is likely that Processes A and B are both waiting for each other's response to complete handshake. In this scenario, both processes are blocked, resulting in infinite wait. One solution to avoid such deadlock is to swap Line 3 and Line 4 of Process A.

- No, it is not safe. Line 4 updates the value of a which was sent by Line 3. It is likely that the update is performed before a is copied into designated buffer. Line 4 is safe to be executed only after the copy operation is completed.

5 [30 points]

- Additional details can be found in Chapter 6.4.2.

```
row = get_row_id;
col = get_col_id;
a_buff[0] = a_block(row, col);
a_buff[1];
b_buff[0] = b_block(row, col);
b_buff[1];

for(i=0; i<p; i++){
        if(row%2 ==0){
        Blocking_send(a_buff[0], Process((row-1)%p, col));
                Blocking_recv(a_buff[1], Process((row+1)%p, col));
                a_buff[0] = a_buff[1];
    } else {
        Blocking_recv(a_buff[1], Process((row+1)%p, col));
        Blocking_send(a_buff[0], Process((row-1)%p, col));
                a_buff[0] = a_buff[1];
    }

    if(col%2 ==0){
        Blocking_send(b_buff[0], Process(row, (col-1)%p));
                Blocking_recv(b_buff[1], Process(row, (col+1)%p));
                a_buff[0] = a_buff[1];
    } else {
        Blocking_recv(b_buff[1], Process(row, (col+1)%p));
        Blocking_send(b_buff[0], Process(row, (col-1)%p));
                a_buff[0] = a_buff[1];
    }
    Compute_and_Update(c, a_buff[0], b_buff[0]);
}
```

- Additional details can be found in Chapter 6.5.1, Example 6.3.

```
row = get_row_id;
col = get_col_id;
a_buff[0] = a_block(row, col);
a_buff[1];
b_buff[0] = b_block(row, col);
b_buff[1];
for(i=0; i<p; i++){
        Nonblocking_send(a_buff[i%2], Process((row-1)%p, col));
        Nonblocking_send(b_buff[i%2], Process(row,(col-1)%p));
        Nonblocking_recv(a_buff[(i+1)%2], Process((row+1)%p, col));
        Nonblocking_recv(b_buff[(i+1)%2], Process(row,(col+1)%p));
        Compute_and_Update(c, a_buff[i%2], b_buff[i%2]);
        Check_nonblocking_send_receive_status();
}
```

- Total communication $= O(p^2 \times 2(\frac{n}{p})^2 \times p) = O(p \times n^2)$