EE/CSCI 451
Fall 2020
Homework 2 solution
Total Points: 100

1 [10 points]

Please refer to slides and textbook.

2 [20 points]

1. For simplicity, let us assume the number of threads w is a power of 2 and denoted as $2^m, m \leq k$. We evenly divide the input vector p into w sub-vectors, each sub-vector with length 2^{k-m}. These sub-vectors are denoted as $p_{sub_0}, \ldots, p_{sub_{w-1}}$. Similarly, we can obtain $q_{sub_0}, \ldots, q_{sub_{w-1}}$. Then, we use T hread $_i (0 \leq i < w)$ to compute the dot product of p_{sub_i} and q_{sub_i}. After each thread obtains a partial dot product, we sum up these partial dot products following the algorithm in Lecture 7 (Title: Adding in PRAM) to obtain the final result.

The time complexity for the serial execution is $O\left(2^k\right) + O\left(2^k - 1\right) = O\left(2^k\right)$. The time complexity for the parallel execution is $O\left(2^{k-m}\right) + O(\log w) = O\left(2^{k-m}\right) + O(m)$ $= O\left(2^{k-m}\right) = O\left(2^k/w\right)$. Therefore, the speedup is $\frac{O\left(2^k\right)}{O\left(2^k/w\right)} = O(w) \rightarrow$ scalable solution. 2.

```
/* Pseudo code executed by the thread with index id */
 Partial_dot_product [id] =0; // Partial_dot_product is a shared array to store partial dot products; the final result will be output as Partial_dot_product [0] by the thread with index 0
 for (i = id * 2^(k-m); i < (id + 1) * 2^(k-m);  i++)
   Partial_dot_product [id] += p_i · q_i
 end for
 barrier; // A barrier is needed here to synchronize threads
 for (i = 0; i < m; i++)
   if (id mod 2^(i+1) = 0 ) then
     Partial_dot_product [id] += Partial_dot_product [id + 2^i]
   end if
   barrier;
 end for
```

3 [40 points]

3.1 [15 points]

```
/* Pseudo code executed by Thread(i) */
Rank[i] = 0
```

```
For j = 0 to n - 1
  If j != i and A[j] < A[i]
    Rank[i] += 1
x = A[i]
Barrier
A[Rank[i]] = x
```

3.2 [25 points]

```
/* Pseudo code executed by Thread(i) */
Rank[i] = 0
Barrier
For j = 0 to n - 1
  If j != i and A[i] < A[j]
    Acquire lock Rank[j]
    Rank[j] += 1
    Release lock Rank[j]
x = A[i]
Barrier
A[Rank[i]] = x
```

4 [30 points]

1. The shared variables include '`At_least_one_vertex_has_update`' and the '`s`' array which records the shortest path lengths.

2.

```
/* Pseudo code executed by Thread(i,j) */
for (k = 0; k < # of vertices; k++)
        if At_least_one_vertex_has_update = true then
                barrier;
                At_least_one_vertex_has_update = false;
                barrier;
                Lock(s(i), s(j));
                if s(i)+w(i,j)<s(j) then
                        s(j) = s(i)+w(i,j);
                        At_least_one_vertex_has_update = true;
                end if
                Unlock(s(i), s(j));
        else then
                Return;
        end if
        barrier;
end for
```