

**EE/CSCI 451**  
**Fall 2020**  
**Homework 2**  
Assigned: September 3, 2020  
Due: September 10, 2020, AOE  
Total Points: 100

## 1 [10 points]

Explain the following terms:

1. Race condition
2. Shared memory programming model
3. Asynchronous execution
4. Atomic Instruction
5. Correct parallel program

## 2 [20 points]

Suppose we want to compute the dot product of two vectors  $p$  and  $q$ , each of which has  $2^k$  elements.

$$\text{dot.product} = p \cdot q = \sum_{i=0}^{2^k-1} p_i \times q_i$$

The computation is executed in a binary-reduction-tree fashion. Figure 1 illustrates the computation process for  $k = 3$ . Now we want to parallelize the execution using  $w$  threads ( $w \leq 2^k$ ) and the final result is output by the thread with index 0.

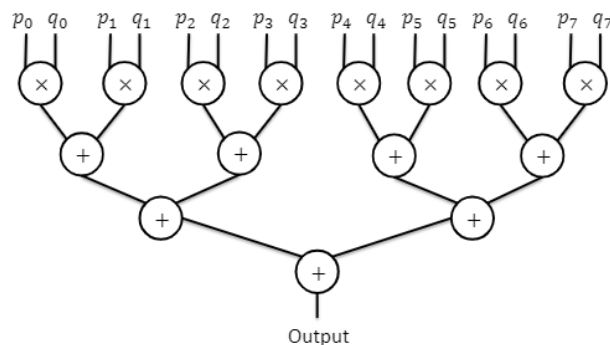


Figure 1: Computing dot product in a binary-reduction-tree fashion

1. Assuming the communication cost and synchronization cost are negligible, give a parallel solution which is scalable with respect to the number of threads. Explain why your solution is scalable.

2. Write the pseudo code of your thread function. Clearly define the meaning of variables in your pseudo code.

### 3 [40 points]

In this problem, we will sort by computing the rank of each element of an input array  $A$ . Assume the elements in array  $A$  are all distinct, i.e.  $\forall_{i \neq j} A[i] \neq A[j]$ . The rank of an element  $A[i]$  is defined as the number of elements in  $A$  less than  $A[i]$ . Denote the array to store the computed rank of elements as  $\text{Rank}[i]$ ,  $0 \leq i < n$ . After computing the rank, each element of array  $A$  is stored in the location specified by its rank.

#### 3.1 [15 points]

Write a shared memory program in Pthreads in which  $\text{Thread}(i)$ ,  $0 \leq i < n$  is responsible to compute  $\text{Rank}(i)$  and move  $A[i]$  to the correct location.

#### 3.2 [25 points]

Write a shared memory program in Pthreads in which  $\text{Thread}(i)$ ,  $0 \leq i < n$  is responsible to update the rank of all other elements based on  $A[i]$ . Also, after all the ranks have been computed, move  $A[i]$  to the correct location. You can use lock to ensure there is no data race.

### 4 [30 points]

Bellman-Ford algorithm [1] is a classic algorithm to solve single-source shortest path problem. The pseudo code of Bellman-Ford algorithm is listed in Figure 2. Assume the input graph  $G(V, E)$  is directed and each edge has a positive weight. Suppose we want to parallelize the execution of the ‘For’ loop at Line 3 using  $p$  threads ( $p = \#$  of edges in  $G$ ), with  $\text{Thread}(i, j)$  corresponding to edge  $(i, j)$  ( $0 \leq i, j < \#$  of vertices in  $G$ ).

1. What are the shared variables for the  $p$  threads?
2. Write the pseudo code of the function executed by  $\text{Thread}(i, j)$ . Your code needs to avoid any possible race condition and take care of the synchronization among the threads.

## References

- [1] “Bellman-Ford algorithm,”  
[https://en.wikipedia.org/wiki/Bellman%E2%80%93Ford\\_algorithm](https://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorithm)

### Bellman-Ford Algorithm

---

Let  $\text{edge}(i, j)$  denote the edge from vertex  $i$  to vertex  $j$

Let  $w(i, j)$  denote the weight of edge  $(i, j)$

Let  $s(i)$  denote the weight of shortest path from source to vertex  $i$

#### **Bellman-Ford ( $G(V, E)$ )**

```
1.  For each vertex  $x$  in  $V$                                 // Initialization //
    If  $x$  is source then
         $s(x) = 0$ 
        At_least_one_vertex_has_update = true
    Else
         $s(x) = \infty$ 
    End if
End for

2.  For  $k = 1$  to #_of_vertices do
    If At_least_one_vertex_has_update = true then
        At_least_one_vertex_has_update = false
3.  For each edge  $(i, j) \in E$  do
        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
    End for
    Else
        Algorithm terminates
    End if
End for
```

Figure 2: Bellman-Ford algorithm