Mark DiValerio - mad2799

Sanjaikanth Pillai - se9727

EE 382V - Parallel Algorithms

Summer 2020

Assignment 1

1) Any algorithm that assumes distinct elements for computing the maximum can be used with an array that has duplicate elements.

Case 1: If the algorithm has special logic that assumes elements are distinct (i.e. a loop that does not compute something that it has already computed), then the algorithm already takes into affect duplicates therefore duplicates do not affect the final maximum result.

Case 2: If the algorithm does **not** have special logic that assumes elements are distinct (i.e. setting an associative array/map using keys or setting a “maximum” variable), then the values just get overwritten with itself, and duplicates do not affect the final maximum result.

In both cases, any algorithm that finds the maximum value does not depend on ***which*** duplicate value is the maximum.

2)

```
for all i in parallel, do:
      if A[i] is odd:
          isBiggest[i] = true //each assumes its element is the biggest
      else:
        isBiggest[i] = false
for all i,j (i!=j) in parallel do:
      if(A[i] is even || A[j] is even)
        continue
      if(A[j] > A[i])
        isBiggest[i] = false
max = null
for all i in parallel do:
      if isBiggest[i]
        max = A[i]
return max
```

This algorithm finds the maximum in O(1) time. Please note that if there are no odd numbers, the max will return a null value.

3)

```
for i in [0, n-1] in parallel do:

       B[i] = A[i];


for h from 0 to logn - 1 do: // upward sweep

       for all i from 0 to n-1 in steps of 2^h+1 in parallel do:

         B[i + 2^(h+1) - 1] = B[i + 2^h - 1] + B[i + 2^(h+1) - 1]


B[n-1] = 0


for h = logn - 1 down to 0 do:

       for i from 0 to n-1 in steps of 2^(h+1) in parallel do:

         left = B[i + 2^h - 1]

         B[i + 2^h - 1] = B[i + 2^(h+1) - 1]

         B[i + 2^(h+1) - 1] = left + B[i + 2^(h+1) - 1]
```

4)

```
for i from 0 to n in parallel do:

       if(A[i] >= x && A[i] <= y)

         B[i] = 1

       else

        B[i] = 0


C = parallel_prefix(B); // cumulative sum, C contains the index of where the elements
will be put into D.


for i from 0 to n in parallel do:

  if(B[i] == 1)

         D[C[i]] = A[i]


return D
```