Introduction to data-parallelism
and OpenCL

Sylvain Lefebvre - INRIA

Longest common subsequence

- LCSS(S, Q)
  - *Length of* longest sub-sequence

- Example:
  - AB**C**DE**F**G**H**
  - ZZ**C**Z**F**Z**H**
  - LCCS = 3
    (there can be multiple sub-seq of same length)

- Notion of prefix:
  - S = (ABCDEFGH)
  - S = ((ABCDEFG),H)
  - $S_n = (S_{n-1}, H)$

Longest common subsequence

- Property 1
  - S_n = (BONT**O**)= (BONT) , **O**
  - Q_n = (BANJ**O**) = (BANJ) , **O**

- Consider
  - LCSS(S_{n-1},Q_{n-1})

➔ LCSS((S_{n-1},**x**),(Q_{n-1},**x**)) = 1 + LCSS(S_{n-1},Q_{n-1})

Longest common subsequence

- Property 2
  - S_n = (BONT) = (BON) , **T**
  - Q_n = (BANJ) = (BAN) , **J**

- Consider
  - LCSS(S_{n-1},Q_n) , LCSS(S_n,Q_{n-1})

➔LCSS((S_{n-1},**x**),(Q_{n-1},**y**)) =

x ≠ y max(LCSS(S_{n-1},Q_n) , LCSS(S_n,Q_{n-1}))

Algorithm

LCSS(S_n , Q_m)

if (S_n = ø or Q_m = ø) { return 0 }

(S_{n-1} , x) = S_n

(Q_{m-1} , y) = Q_m

if (x == y) {

return 1 + LCSS($S_{n-1,}$ Q_{m-1})

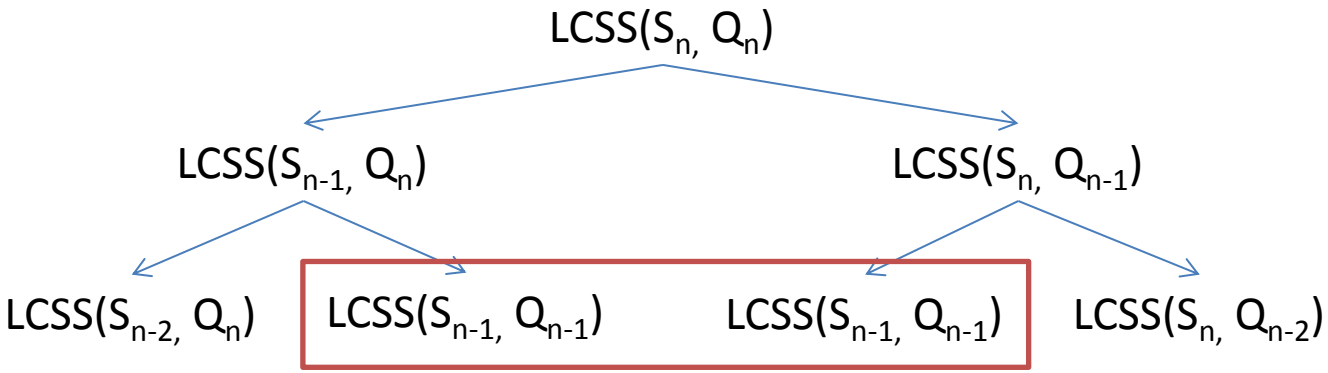
} else {

return max(LCSS($S_{n-1,}$ Q_m) , LCSS($S_{n,}$ Q_{m-1}))

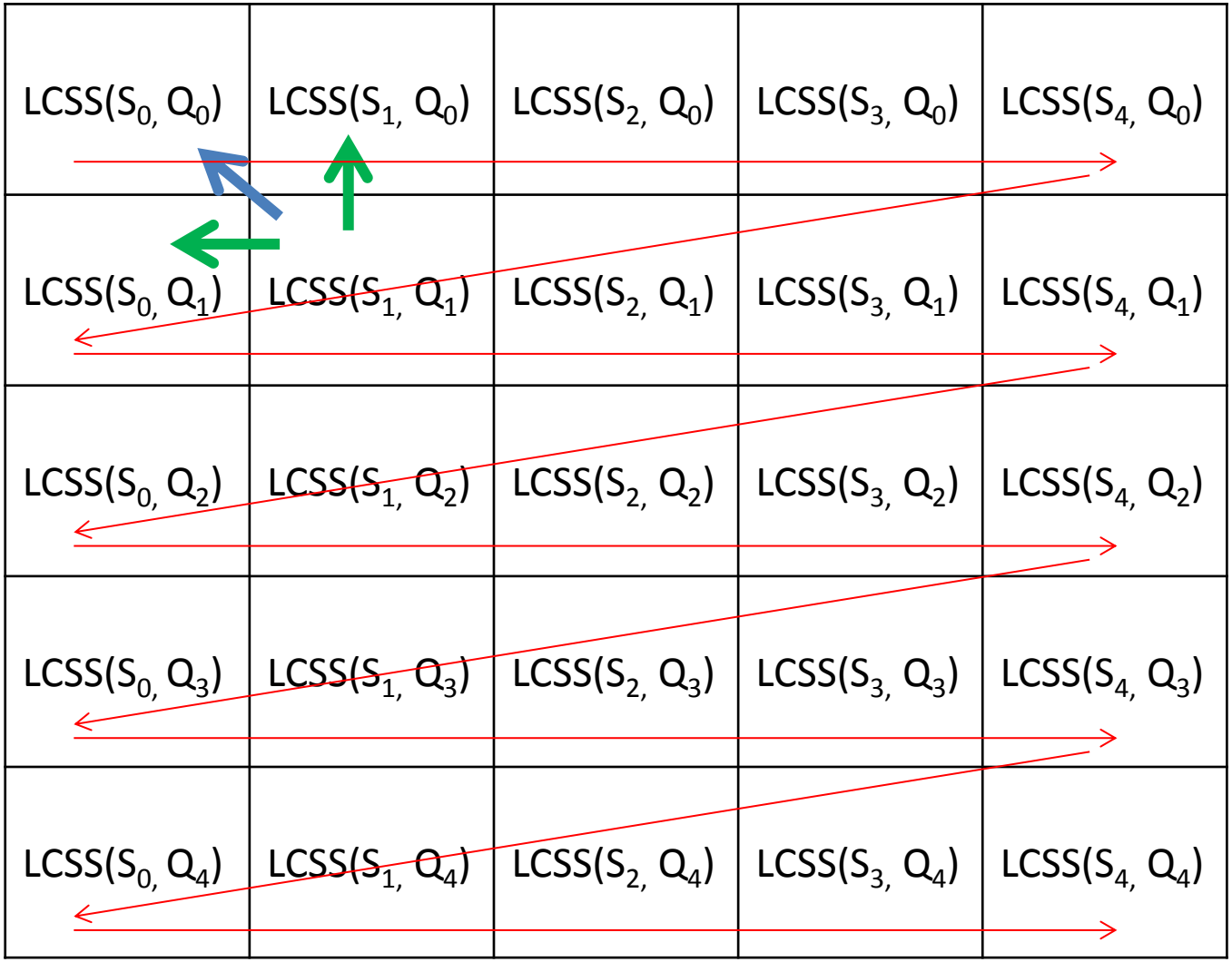
}

Dynamic programming

- General method
- Key observation:
  - Recursive calls often overlap
  - Memorize them, visit in order



Dynamic programming table



Dynamic programming table

0	**0**	**0**	**0**	**0**
0	$LCSS(S_1, Q_1)$	$LCSS(S_2, Q_1)$	$LCSS(S_3, Q_1)$	$LCSS(S_4, Q_1)$
0	$LCSS(S_1, Q_2)$	$LCSS(S_2, Q_2)$	$LCSS(S_3, Q_2)$	$LCSS(S_4, Q_2)$
0	$LCSS(S_1, Q_3)$	$LCSS(S_2, Q_3)$	$LCSS(S_3, Q_3)$	$LCSS(S_4, Q_3)$
0	$LCSS(S_1, Q_4)$	$LCSS(S_2, Q_4)$	$LCSS(S_3, Q_4)$	$LCSS(S_4, Q_4)$

Pseudo-code

```
LCSS( S_N,Q_M )
    DP = [0..N][0..M]
    for j = 0 .. M
       DP[0][j] = 0
    end
    for i = 1 .. N
         DP[i][0] = 0
         for j = 1 .. M
           if S[i] == Q[j]
             DP[i][j] = 1 + DP[i-1][j-1]
           else
            DP[i][j] = max( DP[i-1][j] , DP[i][j-1] )
           end
         end
    end
```