EE 382C/361C: Multicore Computing **Fall 2016**

Lecture 10: September 27

Lecturer: Vijay Garg *Scribe: Shriya Nair*

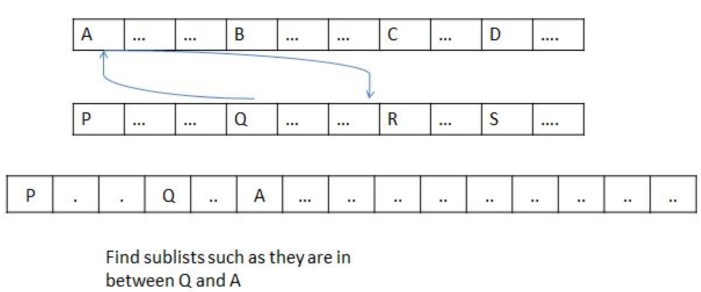
10.1 Puzzle: Sort 2 arrays

Description:
Two sorted A and B of n elements each have to be merged into a sorted array C

Solution 1: Sequential Algorithm
-Time = O(n)
-Work = O(n)
-using two pointers

Solution 2: Parallel Algorithm
- select an element, find its rank (index + position from binary search)
- Time = $O(\log n)$
- Work = $O(n \log n)$

Solution 3: Cascaded Algorithm - the Work-Optimal Solution
-Divide n array into n/log n groups each of size log n.
Number of splitters = $(n/\log n)$
Time taken to search for a splitter = $O(\log n)$
Arrows will never cross because both arrays are sorted



- Fill in only splitters
- Find sublists between and such that they are between Q and A
Max number of sublists = $O(n/\log n)$
Size of each list = $\log n$
Two lists of logn size can be merged in $O(\log n)$

step	T(n)	W(n)
Rank	$O(\log n)$	O(n)
Merging	$O(\log n)$	$O(n/\log n.\log n)$ = O(n)
total:	$O(\log n)$	O(n)

10.2 New Puzzle

Problem: Parallel Prefix sum
Description: Consider an array on size n not necessarily sorted. Compute the recurring sum. (also called scan)
Example: 1,2,3,4,5,6
Output : 1,3,6,10,15,21

Solution 1: Sequential Algorithm
T(n) = O(n)
W(n) = O(n)

10.3 Consistency Conditions

- cannot use locks on datastructures- expensive, sequentializing.
- limiting parallelization of code.
Example:

```
---push(40)----  ----pop(?)----
      ----push(30)----
```

When an element is popped will it return 40 or 30?
- Need a definition for consistency.

10.3.1 Sequential Consistency

- Defined by Lamport
Consider a method foo:

```
s.foo(arg1, arg2)
```

where..
1. s is the object
2. foo: name of the method
3. the statement is the invocation
4. arg1, arg2 are the arguments
5. s.response is the return value. Possible values: OK, value, exception

Consider two operations e,f.

Conventions:
inv(e) : invocation of e
resp(e): response generated after invoking e
proc(e) : process e is on

History of operations: (H, $<_H$) where $<_H$ is the real-time order.
Definition: e $<_H$ f = resp(e) occurred before inv(f)
Example:

```
---push(40)----
    (e)
                  ----push(30)----
                        (f)
```

Consider e, f :

```
---push(40)---
      (e)
           ---push(30)---
               (f)
```

e $\not<_H$ f && f $\not<_H$ e $\implies$ e is concurrent with f

Properties of the relation $<_H$:
1. Irreflexive : e $<_H$ f $\not\Rightarrow$ f $\not<_H$ e
2. Transitive : e $<_H$ f && f $<_H$ g $\implies$ e $<_H$ g
$\implies$ Asymmetric
Hence, (H,$<_H$) is a partially ordered set or poset

10.3.2 Process Order

e $<_H$ f are in process order iff:
proc (e) = proc (f) and resp(e) occurred before inv(f)

$<_{po} \subseteq <_H$
e $<_{po}$ f $\implies$ e $<_H$ f but the reverse may not be true

10.3.3 Sequential History

A history (H, $<_H$) is sequential if $<_H$ is a total order.
A poset (X, <) is a total order if $\forall$ distinct (x,y) belongs to X, (x $<_H$ y) or (y $<_H$ x)

Legal Sequential History:
A sequential history S is legal if it satisfies sequential specifications of the objects.

References

[1] Vijay K Garg, Introduction to Multicore Computing