PRAM ALGORITHMS:
BRENT’S LAW
MERGING TWO SORTED ARRAYS

An optimal RAM algorithm creates the merged list one element at a time.

- Requires at most n-1 comparisons to merge two sorted lists of n/2 elements.
- Time complexity $\Theta(n)$

Can we do in lesser time?

PARALLEL MERGE

Consider two sorted lists of distinct elements of size n/2.

We spawn n processors, one for each element of the list to be merged.

In parallel, the processors perform binary search of the corresponding elements in the other half of the array.

- Element in the lower half of the array performs a binary search in the upper half.
- Element in the upper half of the array performs a binary search in the lower half.
THE TASK OF P₃

A[i=3] is larger than i-1=(3-1)=2 elements in the lower array (lower wrt. Index)


Thus, 7 is larger than 2 elements in the lower array, and larger than (high-n/2)=10-8=2 elements in the upper array.

So, P₃ can calculate the position of 7 in the merged list, ie. after (i-1)+(high-n/2), thus the position is (i+high-n/2).

THE TASK OF P₁₁

A[i=11]=8 is larger than i-(n/2+1)=(11-9)=2 elements in the upper array (lower wrt. Index)


Thus, 8 is larger than 2 elements in the upper array, and larger than high=3 elements in the upper array.

So, P₁₁ can calculate the position of 8 in the merged list, ie. after (i-n/2-1)+(high), thus the position is (i+high-n/2).

Thus the same expression is used to place the elements in their proper position in the merged list.
THE PRAM ALGORITHM

MERGE LISTS (CREW PRAM):
Given: Two sorted lists of $n/2$ elements each, stored in
The two lists and their unions have disjoint values
Final condition: Merged list in locations $A[1] \ldots A[n]$
Global $A[1 \ldots n]$
Local $i, low, high, index$
begin
spawn ($P_1, P_2, \ldots, P_n$)
for all $P_i$ where $1 \leq i \leq n$ do
(Each processor sets bounds for binary search)
if $i \leq n/2$ then
low $\leftarrow (i/2) + 1$
else
low $\leftarrow 1$
high $\leftarrow n/2$
endif
endfor

{ Each processor performs binary search }$x \leftarrow A[i]$
repeat
index $\leftarrow (low + high)/2$
if $x < A[index]$ then
high $\leftarrow index - 1$
else
low $\leftarrow index + 1$
endif
until $low > high$
{ Put value in correct position on merged list }$A[high + i \times n/2] \leftarrow x$
end

Note that the final writing into the array is done by the processors
without any conflict. All the locations are distinct.
Also note that the total number of operations performed have increased
from that in a sequential algorithm $\Theta(n)$ to $\Theta(n \log n)$ in the parallel
algorithm.
COST-OPTIMAL SOLUTIONS

We have seen examples of PRAM algorithms which are not cost optimal.

Is there a cost-optimal parallel reduction algorithm that has also the same time complexity?

BRENT’S THEOREM (1974)

Assume a parallel computer where each processor can perform an operation in unit time.

Further, assume that the computer has exactly enough processors to exploit the maximum concurrency in an algorithm with $M$ operations, such that $T$ time steps suffice.

Brent's Theorem say that a similar computer with fewer processes, $P$, can perform the algorithm in time, $T_P \leq T + (M - T)/P$
BRENT’S THEOREM (PROOF)

Let $s_i$ denote the number of computational operations performed by the parallel algorithm $A$ at step $i$, where $1 \leq i \leq t$.

By definition $\sum_{i=1}^{T} s_i = M$.

Thus, using $p$ processors we can simulate step $i$ in time $\left\lfloor \frac{s_i}{p} \right\rfloor$.

By definition,

$$T_p = \sum_{i=1}^{T} \left\lfloor \frac{s_i}{p} \right\rfloor \leq \sum_{i=1}^{T} \frac{s_i + p - 1}{p} = \sum_{i=1}^{T} \frac{p}{p} + \sum_{i=1}^{T} \frac{s_i - 1}{p} = T + \frac{M - T}{p}.$$

Note this reduction is work-preserving, meaning that the total work does not change.

Also, note $p$ is lesser than the initial number of processors, which is manifested by the increase in the time required.

APPLICATION TO PARALLEL REDUCTION

We know of a solution with large number of processors, which takes $\Theta(n)$ time.

Let us reduce the number of processors to $\left\lfloor \frac{n}{\log n} \right\rfloor$ processors.

Thus,

$$T_p \leq \log n + \frac{(n-1) - \log n}{\frac{n}{\log n}} = \Theta \left( \log n + \log n - \frac{\log n - \log^2 n}{n} \right) = \Theta(\log n)$$

Thus reducing the number of processors from $n$ to $\left\lfloor \frac{n}{\log n} \right\rfloor$ does not change the complexity of the parallel algorithm.

If the total number of operations performed by the parallel algorithm is the same as an optimal sequential algorithm, then a cost optimal parallel algorithm does exist.
AN ORDER ANALYSIS: WORK-DEPTH MODEL

Let $l_i$ denote the computation in the $i^{th}$ level.

Thus, by assigning $\left\lceil \frac{l_i}{P} \right\rceil$ operations to each of the $P$ processors in the PRAM, the operations for level $i$ can be performed in $O\left(\frac{l_i}{P}\right)$ steps.

Summing the time over all the $D$ (Depth) levels,

$$T_{PRAM}(W, D, P) = O\left(\sum_{i=1}^{D} \left\lceil \frac{l_i}{P} \right\rceil \right) = O\left(\sum_{i=1}^{D} \left\lceil \frac{l_i}{P} \right\rceil \right) = O\left(\frac{1}{P} \sum_{i=1}^{D} l_i + D\right) = O\left(\frac{W}{P} + D\right).$$

Note: $W$ is the total work done by the sequential algorithm, which we have assumed is the same.

The total work performed by the PRAM is $O(W+PD)$.

A cost optimal solution thus can be obtained if $PD \leq W$, or $P \leq \frac{W}{D}$.

EXERCISES

Now think of the cost optimal solutions that we discussed, like reduction, prefix sum, suffix sum, pointer jumping, tree traversal etc. in the light of Brent’s Law.
NON-OBVIOUS APPLICATIONS OF PREFIX SUM

Suppose, we have an array of 0’s and 1’s, and we want to determine how many 1’s begin the array.
* Ex (1,1,0,1,0,1).The answer is 3.

NON-OBVIOUS APPLICATIONS OF PARALLEL SCAN / REDUCTIONS

Suppose, we have an array of 0’s and 1’s, and we want to determine how many 1’s begin the array.
* Ex (1,1,0,1,0,1).The answer is 3.

It may be non-intuitive to think of an associative operator which we might use here!

However, there seems to be a common trick, which we can try to learn.
THE TRICK

Let us define for any segment of the array by the notation \((x,p)\)

* \(x\) denotes the number of leading 1's
* \(p\) denotes whether the segment contains only 1's.

Thus, each element \(a_i\) is replaced by \((a_i,a_i)\).

How do we combine, \((x,p)\) and \((y,q)\)?

Let us define an operator, \(\otimes\) to do this.

It is intuitive that \((x,p) \otimes (y,q) = (x + py, pq)\). Why?

Is this operator associative?

* \(((x,p) \otimes (y,q)) \otimes (z,r)) = (x + py + pqz, pqr)\)
* \((x,p) \otimes ((y,q) \otimes (z,r)) = (x + p(y + qz), pqr) = (x + py + pqz, pqr)\)

Now all the previous parallelizations can be applied 😊

EXAMPLE

```
1 1 1 0 1 1 0 1
(1,1) (1,1) (1,1) (0,0) (1,1) (0,0) (1,1)
(2,1) (1,0) (2,1) (0,0)
(3,0) (2,0)
(3,0)
```
CAN YOU EVALUATE A POLYNOMIAL IN PARALLEL USING A SIMILAR METHOD?

Consider a polynomial: \(a_0x^{n-1} + a_1x^{n-2} + \ldots + a_{n-2}x + a_{n-1}\).

Each segment also denotes a polynomial. Say, the first two coefficients denoted \(a_0x + a_1\).

Let us consider \((p,y)\) to denote a segment.
- \(p\) denotes the value of the segment’s polynomial evaluated for \(x\).
- \(y\) denotes the value of \(x^n\), where \(n\) is the length of the segment.

Thus, each element \(a_i\) is replaced by \((a_i,x)\).

How do we combine, \((p,y)\) and \((q,z)\)?

Let us define an operator, \(\otimes\) to do this.

It is intuitive that \((p,y) \otimes (q,z) = (pz + q, yz)\). Why?

Is this operator associative?
- \([(a,x) \otimes (b,y)] \otimes (c,z) = (ayz + bx, xyz)\)
- \((a,x) \otimes [(b,y) \otimes (c,z)] = (ayz + bx + c, xyz)\)

Now all the previous parallelizations can be applied.

ANNOUNCEMENTS:

NO CLASS ON 13TH AUGUST.

QUIZ ON 14TH AUGUST, 2015 AT 4:30 PM
-- SYLLABUS (TILL THIS POINT)

IF YOU AGREE, WE CAN SWAP THE ABOVE TOO!