MERGING TWO SORTED ARRAYS

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

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

- Can we do in lesser time?

3

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.

4

THE TASK OF P_3

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

A[1]						A[8]
1	5	$\boxed{7}$	13	17	19	23

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.

Perform a binary search with A[3] in the upper array.
Get a position high=index of the largest integer smaller than 7=>high=10.

A[9]						A[16]
2	4	8	11	12	21	24

So, P_3 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).

5

THE TASK OF P_{11}

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

A[1]						A[8]
1	5	7	13	17	19	23

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

Perform a binary search with A[11] in the lower array.
Get a position high=index of the largest integer smaller than 8=>high=3.

A[9]						A[16]
2	4	$\boxed{8}$	11	12	21	24

So, P_{11} 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.

6

THE PRAM ALGORITHM

```
MERGE.LISTS (CREW PRAM):

  Given: Two sorted lists of n/2 elements each, stored in
         A[1] ··· A[n/2] and A[(n/2)+1] ··· A[n]
         The two lists and their unions have disjoint values
  Final condition: Merged list in locations A[1] ··· A[n]
  Global  A[1 ··· n]
  Local   x, low, high, index
  begin
    spawn (P1, P2, ..., Pn)
    for all Pi where 1 ≤ i ≤ n do
      { Each processor sets bounds for binary search }
      if i ≤ n/2 then
        low ← (n/2)+1
        high ← n
      else
        low ← 1
        high ← n/2
      endif
```

7

PRAM (CONTD.)

```
      { Each processor performs binary search }
      x ← A[i]
      repeat
        index ← ⌊(low + high)/2⌋
        if x < A[index] then
          high ← index − 1
        else
          low ← index + 1
        endif
      until low > high
      { Put value in correct position on merged list }
      A[high + i − n/2] ← x
    endfor
  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 $\Theta(nlogn)$ in the parallel algorithm.

8

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≤i≤t.

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

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

By definition,

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

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 $\lfloor (\frac{n}{\log n)} \rceil$ processors.

Thus,

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

$$= \Theta(\text{logn})$$

Thus reducing the number of processors from n to $\left\lfloor \frac{n}{logn} \right\rfloor$ 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 $O(\left\lceil \frac{l_i}{P} \right\rceil)$ steps.

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

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

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≤W, or P ≤W/D.

13

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.

14

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 (1,1,1,0,1,1,0,1)..The answer is 3.

15

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 (1,1,1,0,1,1,0,1)..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.

16

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, ⨂ to do this.

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

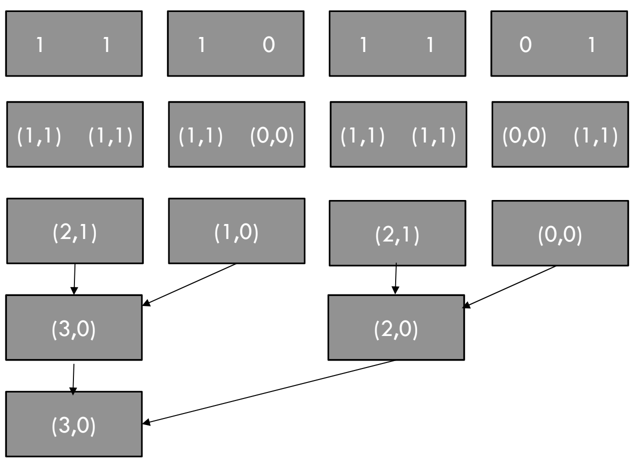
Is this operator associate?

- ((x,p) ⨂ (y,q)) ⨂(z,r)=(x+py,pq) ⨂(z,r)=(x+py+pqz,pqr)
- (x,p) ⨂ ((y,q)) ⨂(z,r))=(x,p)⨂(y+qz,qr)=(x+p(y+qz),pqr)=(x+py+pqz,pqr)

Now all the previous parallelizations can be applied ☺

17

EXAMPLE



18

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, ⊗ to do this.

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

Is this operator associate?

- $((a,x) \otimes (b,y)) \otimes (c,z)=(ay+b,xy) \otimes (c,z)=(ayz+bz+c, xyz)$
- $(a,x) \otimes ((b,y)) \otimes (c,z))=(a,x)\otimes(bz+c,yz)=(ayz+bz+c, xyz)$

Now all the previous parallelizations can be applied ☺

19

20