1/31/2023 Models

- PRAM
- Vector RAM (VRAM)
- Circuit Models (<u>NC</u>, P)
- MP-RAM : Multiprocessor RAM

Goals for a parallel model:

- simple
- guide you (alg. designer) toward efficient algorithms
- understand how perf / alg. scales as we vary the input size
- robust across a variety of machines
- useful for understanding alg. design techniques
- can naturally express algs. in pseudocode and by extension real code.

What about the PRAM?

- perhaps <u>not</u> robust / implementable
- but alg. design techniques can be illustrated in the PRAM

- PRAM assumes - fixed (p) processors; making it rather unwieldy for coding

Work-Depth / Work-Span

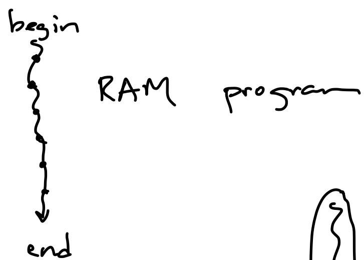
- shared random access memory
- dynamic task creation / closure.

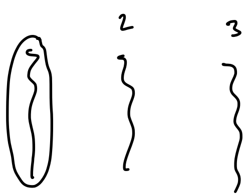
work = # operations

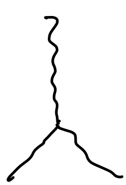
depth/span = longest set of sequential dependencies

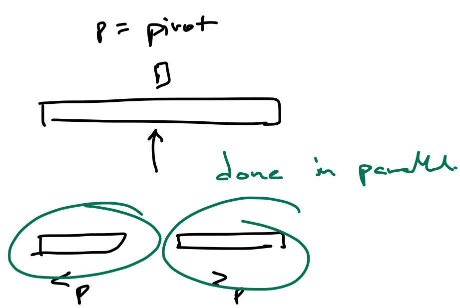
MP-RAM : Multiprocessor-RAM

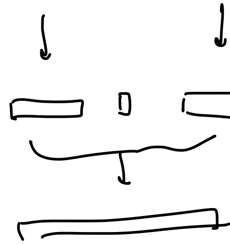
- set of dynamically evolving processes
- unbounded memory





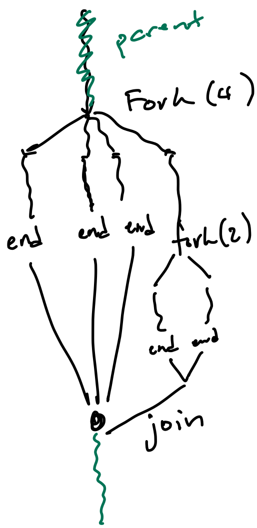


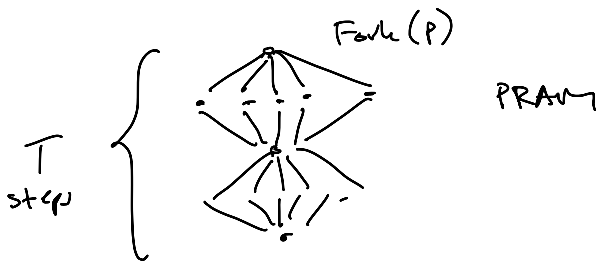




- augment with FORK(k)
  - k children processes created
  - child i gets "id" i

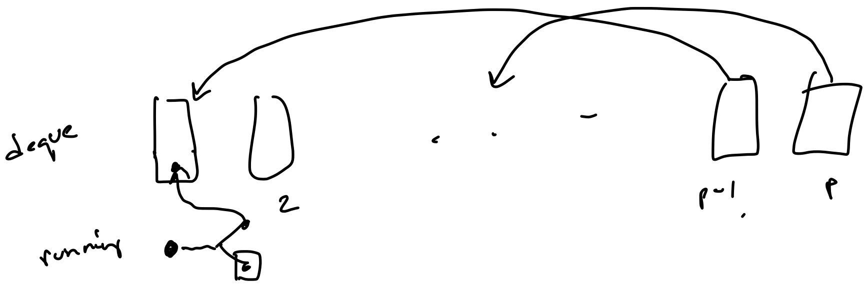
- nested forks ok!





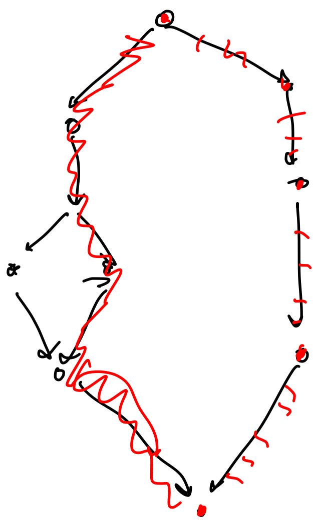
Binary-forking vs k-ary forking

work-stealing scheduler (~~B~~ Blumofe-Leiserson)



work = total number of basic ~~instructions~~ operations

depth = longest chain of sequential dependencies



W = 10

D = 5

Work = time using 1 processor

Depth = time using ∞ processors.

$$T \geq \frac{W}{P} \quad \Rightarrow \quad T \geq \max\left(\frac{W}{P}, D\right)$$

$$T \geq D$$

Brent's Thm: Given a DAG with work W, Depth D, it can be scheduled using P processors in $O\left(\frac{W}{P} + D\right)$ steps

$$\max\left(\frac{W}{P}, D\right) \leq T \leq \underbrace{O\left(\frac{W}{P} + D\right)}_{\text{work-stealing}}$$

$$\underbrace{\frac{W}{P} + D}\ \text{time}$$

$$P = \underbrace{\frac{W}{D}}_{\text{parallelism}} \qquad D + D$$

- Parallelism $= \frac{W}{D}$

  - $W = O(n \log n)$
  - $D = O(\log n)$

$\frac{W}{D} = O(n)$

---

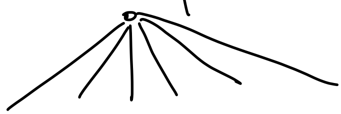
parallel for loop:

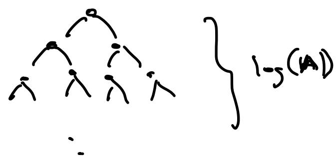
$[0, 1, 2, \ldots |A|-1]$

parfor i in $[0 : |A|]$

$B[i] \overset{:=}{\leftarrow} f(A[i])$

fork$(|A|)$





pardo, parallel do, fork : $||$

$+$ $\quad A[0 : |A|/2]$

```
SUM(A) =
  if (|A| = 1) return A[0]
  l <- sum(A[0, |A|/2]) ||
  r <- sum(A[|A|/2, |A|])
  return l + r
```

$C = A \| B$

$w(C) = w(A) + w(B)$

$D(C) = \max(D(A), D(B)) + 1$

Sum:

$w(n) = 2w(n/2) + O(1) = O(n)$

$D(n) = \max(D(n/2), D(n/2)) + O(1) \in O(\log n)$

Filter (A, f):

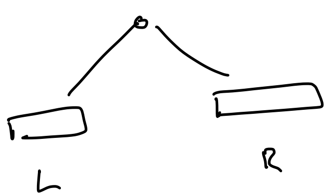
$[1, 2, 3, \ldots, 10]$

$\downarrow$

$[2, 4, 6, 8, 10]$

$W = O(n)$

$D = O(\log n)$



$O = \text{alloc}(|L| + |R|)$

Filter (A, f):

if (|A| = 1):
  if f(A[0]) return A
  else return []
endif

$$(L, R) = \text{filter}(A_1, f) \parallel \text{filter}(A_2, f)$$

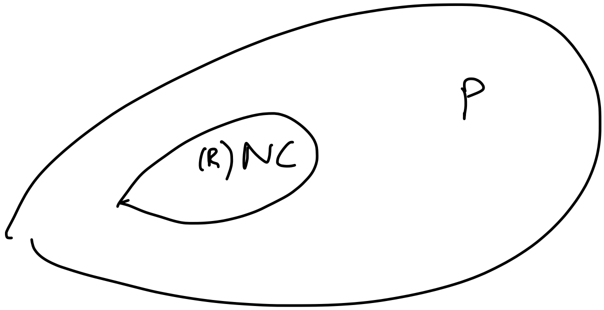
O = alloc(|L| + |R|)

{ copy L, R into O }

return O
endfun

$$D(n) = D(n/2) + O(\log n)$$

$$\in O(\log^2 n)$$



Nick's class:

- poly work
- poly-logarithmic depth

SSSP: n^3 work $\log^2 n$ depth

$n \log(n)$ work/depth

work-efficiency: parallel alg uses the same amount of work (asymptotically) as that of the best seq. algorithm.

$$\boxed{\frac{W}{P}} + \underline{\underline{O(D)}}$$