One last lecture on routing

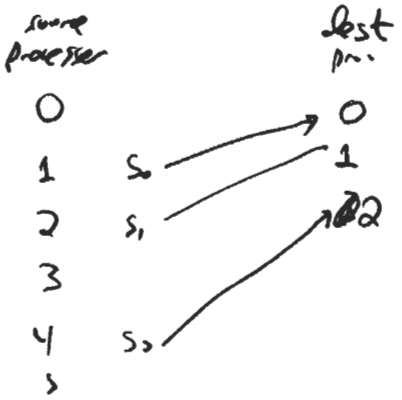
I have a magic trick too! (Nerd magic)...

First, I need to define a squish message pattern.

Given a set subset of processors $\{s_0, \ldots, s_{n-1}\}$
with $0 \le s_0 < s_1 < \cdots < s_{n-1} < P$

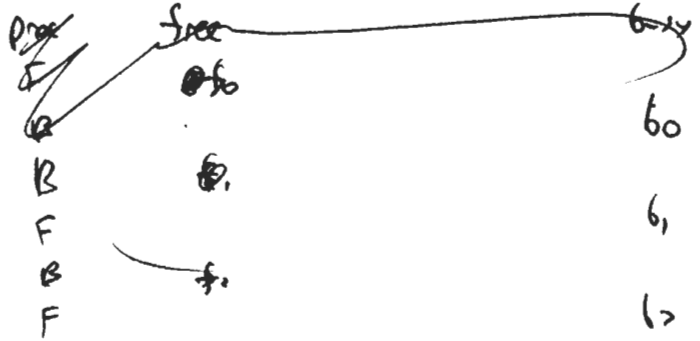
Processor # s_i sends a message to processor i.

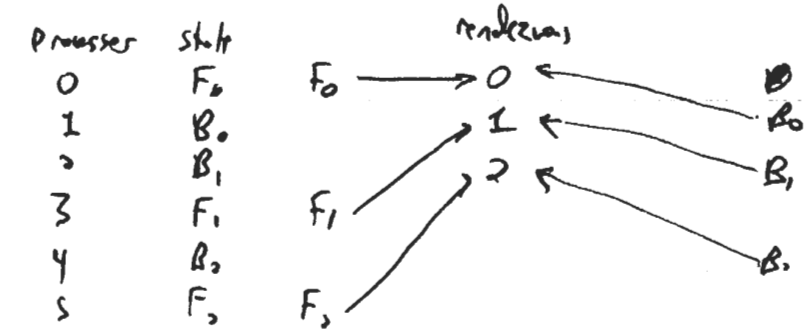
E.g.



Why would I be interested in squish?

Could use it to pair up free + busy processors





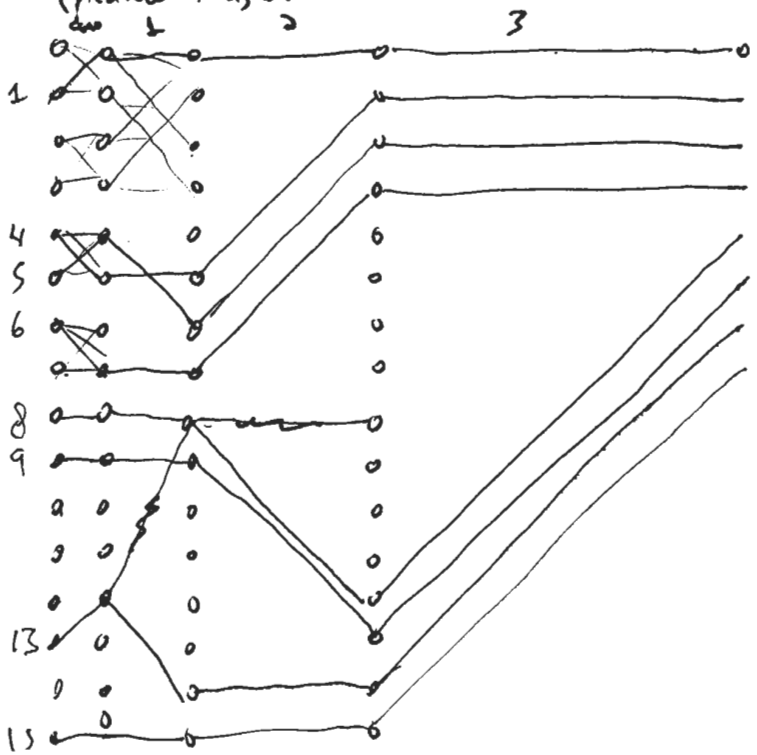
Now processor 1 knows the address of 1 free + 1 busy processor.

Could be used in a cilk implementation for example.
Or parallel "cons"

Now the magic: I have a butterfly

Now the Magic! Here is a butterfly

(predraw 4 bits wide)



pick some subset that I will

Squash e.g. 1, 4, 5, 6, 8, 9, 13, 15

$S_0 = 1$ $S_3 = 6$ $S_6 = 13$

$S_1 = 4$ $S_4 = 8$ $S_7 = 15$

$S_2 = 5$ $S_5 = 9$

1101
0110
1011

Look MA, no collisions!

~~This always~~

Thm: This magic always works

Proof: Define: a semicontraction is a 1-1 routing from $\{s_0, \ldots, s_n\}$ to $\{d_0, \ldots, d_n\}$

with $s_i \to d_i$ such that

$\forall i,j$ $\quad |s_i - s_j| \geq |d_i - d_j|$

↕ this is subtraction, not Hamming distance.

Lemma: Semicontraction routes on butterfly are conflict.

proof: by contradiction:

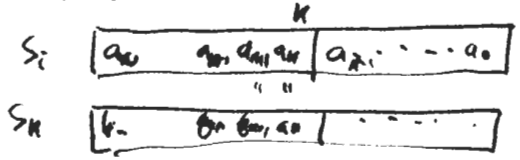
Suppose $\exists$ conflict. $\Rightarrow$ some pair must conflict

$s_i \to d_i$ conflicts with $s_j \to d_j$

Looks like

Observe $|s_i - s_j| < 2^k$

why? $s_i + s_j$ must agree on ~~both~~ high bit



but $a_i = b_i$ for $i \geq k$

so $|s_i - s_j| = |\sum a_i 2^i - \sum b_i 2^i|$

$= \sum$

$$= \left| \sum_{i=0}^{k-1} (a_i - b_i)\cdot 2^i + \sum_{i=k}^{n} (a_i - b_i)\cdot 2^i \right|$$

$$= \left| \sum_{i=0}^{k-1} (a_i - b_i)\cdot 2^i + 0 \right|$$

$$\leq \left| 2^k - 1 \right| < 2^k$$

observe $|d_i - d_j| > 2^k$

Must agree on low bits

d_i: $\ldots \; c_{k+1} \,|\, c_k \; \ldots \; c_0$

d_j: $\ldots \; e_{k+1} \; e_k \; \ldots \; e_0$

$c_i = e_i$ if $i \leq k$

$$|d_i - d_j| = \left| \sum (c_i - e_i) 2^i \right|$$

$$= \left| \sum_{i=0}^{k} (c_i - e_i)\cdot 2^i + \sum_{i=k+1}^{n} (c_i - e_i) 2^i \right|$$

$$= \left| 0 + \sum_{i=k+1}^{n} (c_i - e_i) 2^i \right|$$

some bit must be different, so $> 2^k$

$\Rightarrow$ not a semicontraction $\Rightarrow\!\Leftarrow$ ☒.

Observe that squish is a semicontraction and $\therefore$ The magic works ☒.

What can we do with this magic?
How do we implement it?

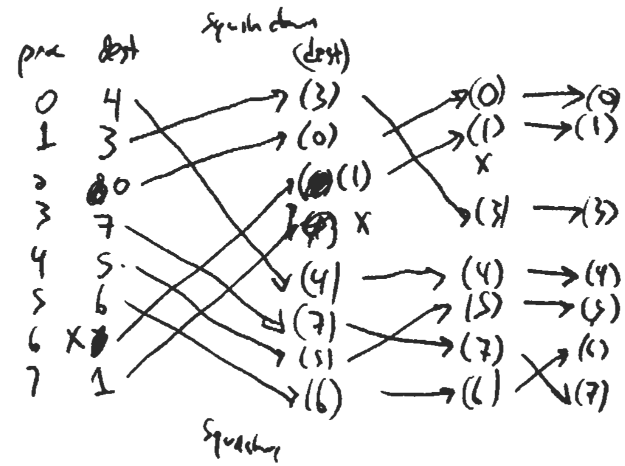
Application of magic: routing 1-1 message traffic

Algorithm:

squish down all messages with 0 in bit (n-1).

squish up all messages with 1 in bit n-1

+ recurse



How to implement SQUISH.

The problem is to compute the enumeration of the set $\{S, \ldots\}$

let $v_i = \begin{cases} 0 & \text{if } i \notin S \\ 1 & \text{if } i \in S \end{cases}$

i.e.		$\vec{v}$	$\vec{w}$
0		0	
1	S_0	1	
2	S_1	1	
3		0	
4	S_2	1	
5		0	

also scan +

Want to compute the prefix sum $\vec{w}$

w_i = the number of 1's in v before i

$$w_i = \sum_{j=0}^{i-1} v_i$$

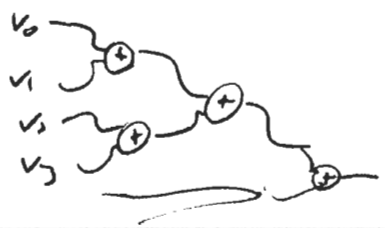
Can easily compute $\vec{w}$ in work P and $T_\infty = P$

Can we do it faster?

Yes. For example we can compute

$$w_p = v_0 + v_1 + \cdots + v_{p-1}$$

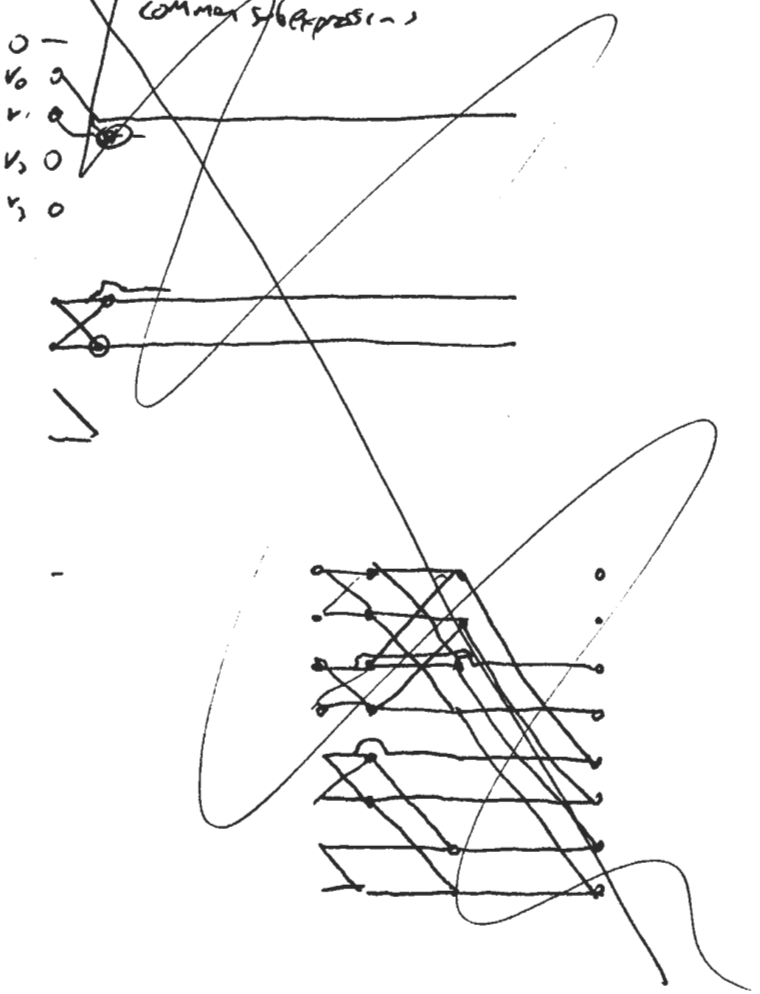
$$= ((v_0 + v_1) + (v_2 + v_3)) + ((v_4 + s_5) + \cdots)$$

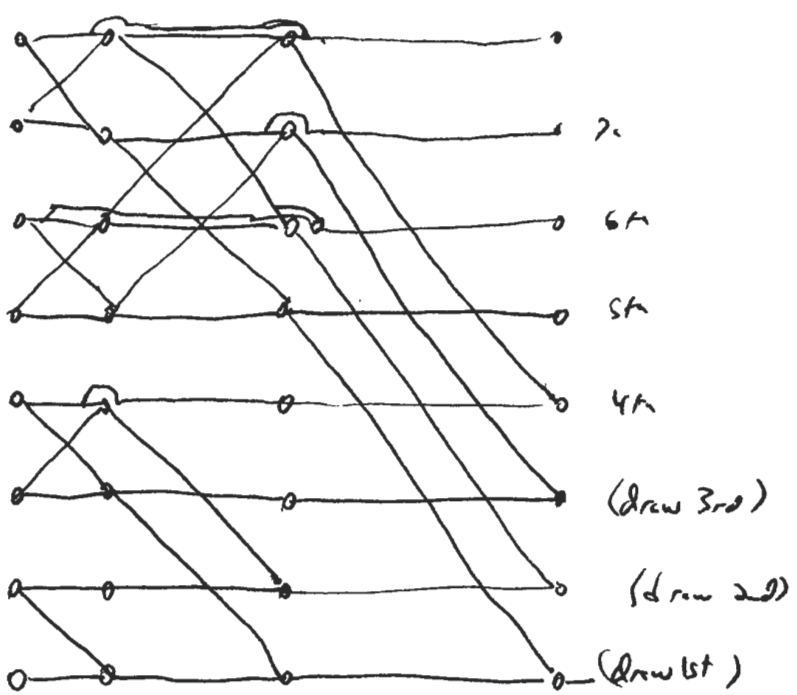


$T_\infty = \lg n$

but work?

On a hypercube! Butterfly! just do it, reusing common subexpressions





It fits on a butterfly!

Analysis: Time to do scan

depth of graph = $\lg n$

how long to add 2 numbers?

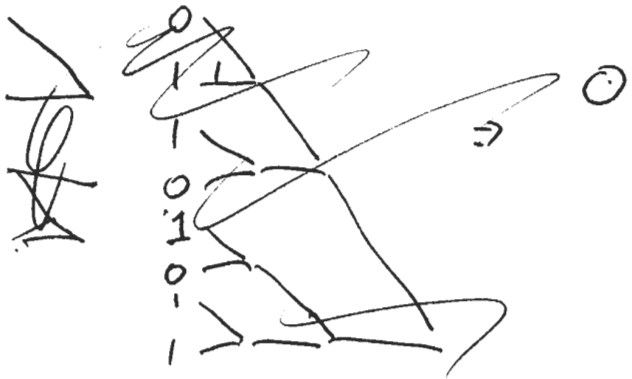
we assume the wire is $O(1)$ bits/time unit

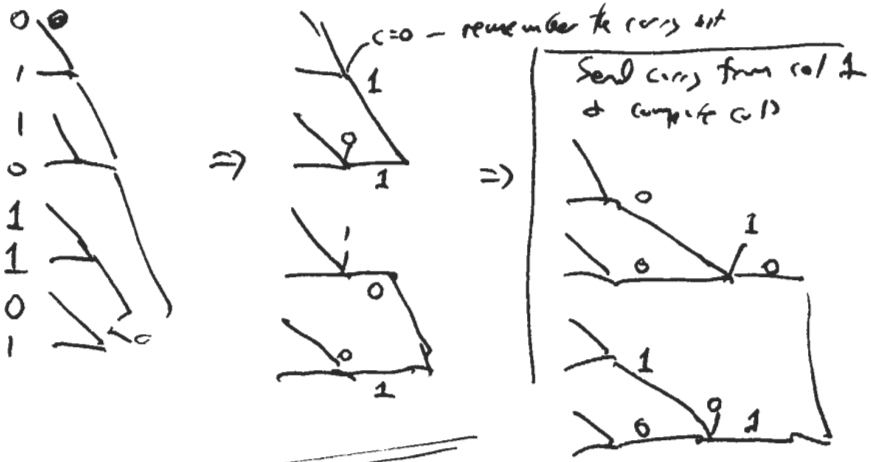
$\Rightarrow$ it takes $O(\lg n)$ time just to send the final answer out the last stage.

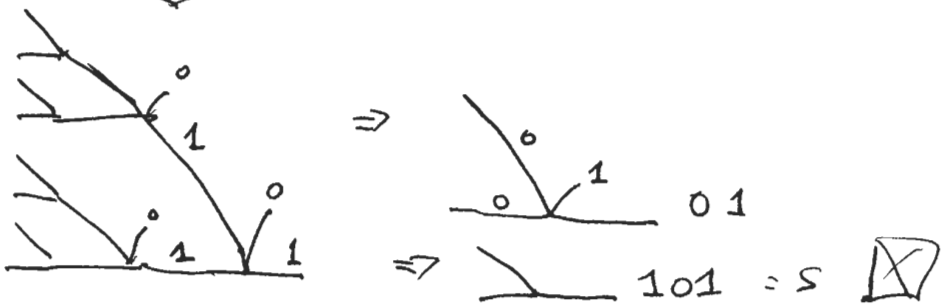
~~But, the adds can be pipelined!~~

So is it $O(\lg^2 n)$ time?

No! Bits can be pipelined.







∴ With pipelining get $O(\lg n)$ alg for /+

~~Total time to run a 1-1 routing~~

~~$T_n = O(\lg n) + T_{n/2}$~~

do 1-1 routing on n processes:

compute scan + up on 0's	$\Theta(\lg n)$
Send message is squish down	$M + \lg n$
compute scan + down on 0's	$\Theta(\lg n)$
Send message is squish up	$M + \lg n$

do 1-1 routing on two subcubes in parallel

M = message size

$M = \Omega(\lg n)$

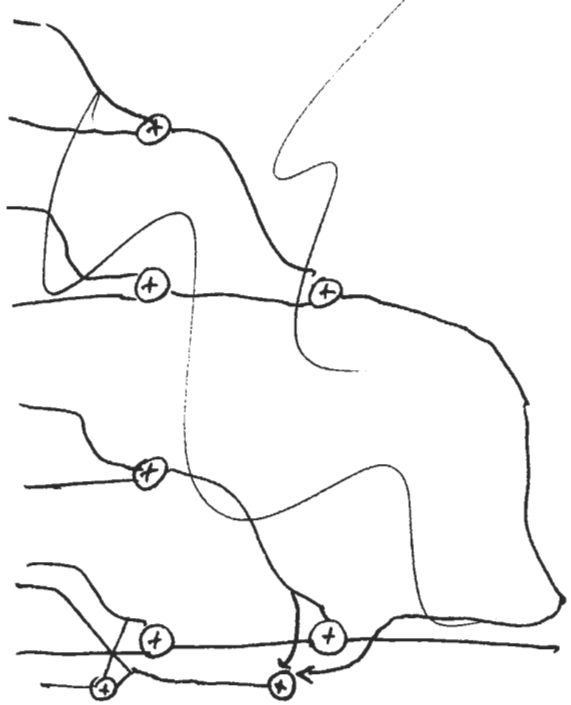
$\Theta(M) = \lg n$

$$T_n = 2M + \Theta(\lg n) + T_{n/2}$$

$$T_n = \Theta(M) + T_{n/2}$$

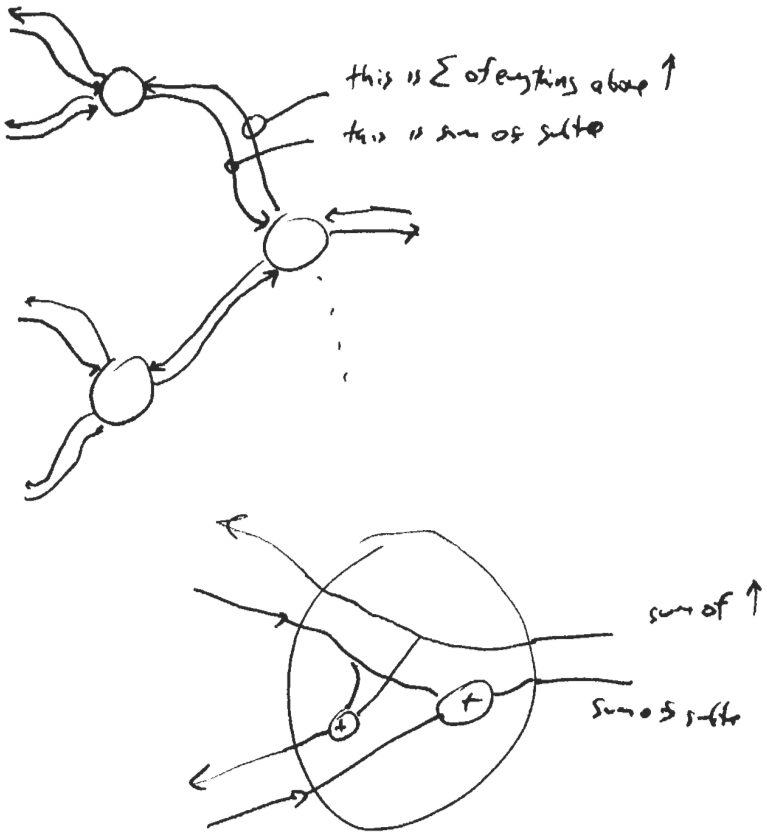
$$= O(M \lg n)$$

Another way to do parallel prefix: with two trees superimposed

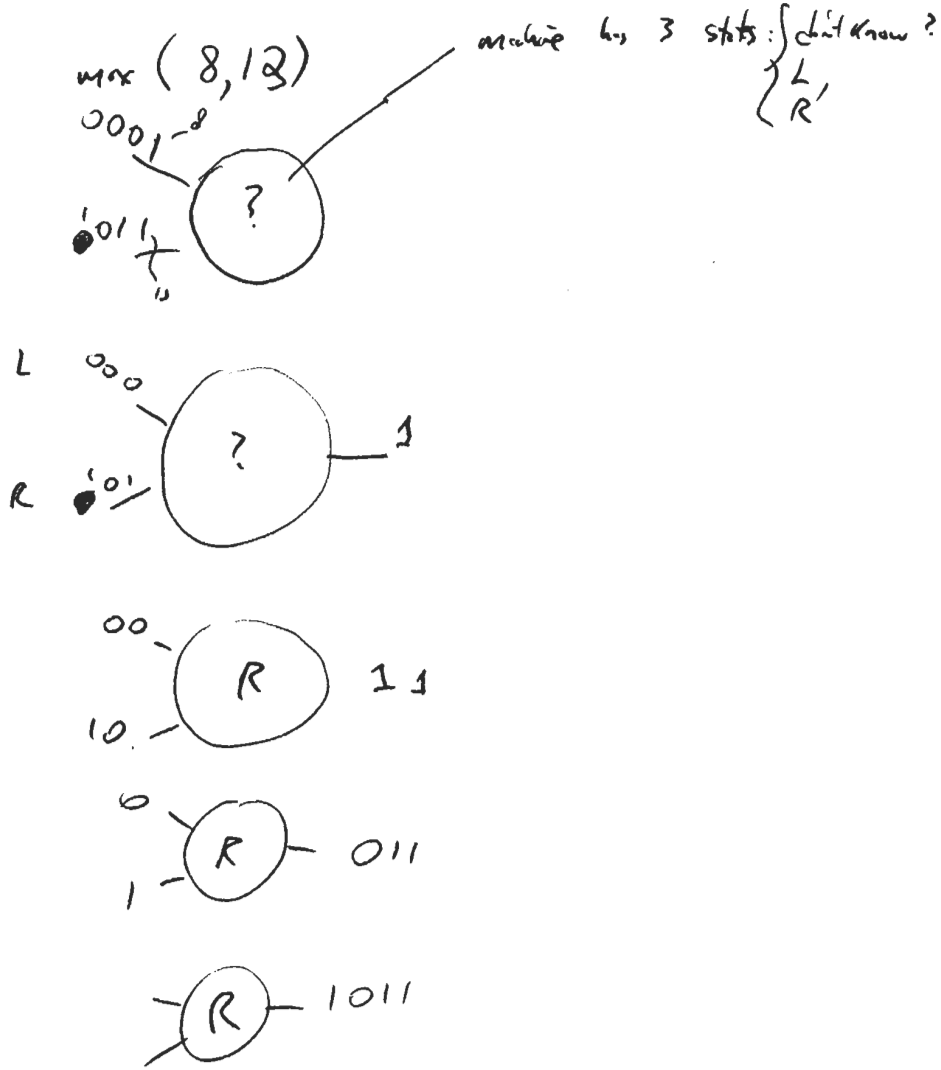


Another way to do /+
with depth $2 \lg P$ adders

but work only P
(butterfly is depth $\lg P$, work $P \lg P$)

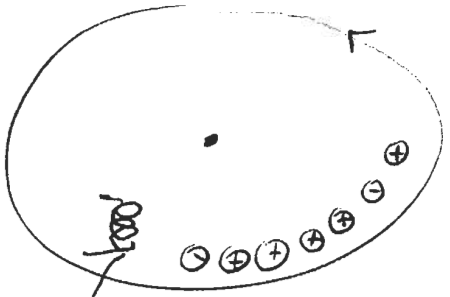


Also can bit-pipeline max; feed in most significant bits first



Disks:

A spinning platter covered with magnetic



encode info in the orientation of magnet.

To write: position an electromagnet over a spot and apply current, it switches the state

To read: in principle, ~~the same~~ goes in reverse. Sense current from the induced field.

Reality: ~~Gigantic Magnets~~

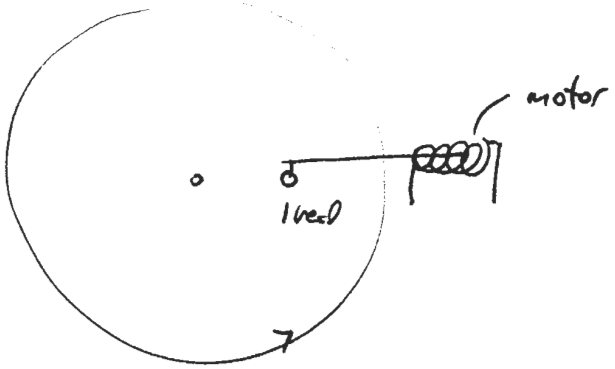
GMR – Giant Magneto Resistance a device whose resistance is a fn of magnetic fields. Very sensitive.

GMR discovered in late 80's

⇒ about 10 years to appear in disks.

About 20 ~~GB/in²~~ Gb/in²

How to position the head?



Disk spins
motor moves head radially ("seek")
⇒ can see any spot on disk

Motor is a linear induction motor

(old motors were stepper motors. Time to seek distance n was $\Theta(n)$

With linear induction motor, head accelerates to ½-way point ⇒ time $\Theta(\sqrt{n})$

today, avg seek time = ~~4.9ms~~ 4.9ms read, 5.4ms write
best seek time = ~1 ms
worst = ~9 ms
rot = 10,000 rpm (expensive + hot) 6 ms ↻
5,400 rpm 11 ms ↻
(½ that on average.

More data near edge of disk
=> trick: put your data on the outer part of the disk to reduce # of seeks.

Head "flies" above disk surface

Disk bearings use fluid-dynamics (was ball bearings – about 2 years ago)

- Quiet
- No runout (vibration, making accurate placement tight)
  Non Repeatable Runout

the disk itself "swims" in

with ball bearings heat + vibration were the disks big enemy, causing the bearings to deform.

Now – I don't know what causes disks to wear out.

~~[illegible]~~

Drive can move
~~4[illegible]~~
458-799 Mb/s (up to 200 MB/s)

Since avg seek is 5ms seek
3ms ↻
8ms

=> 16 MB/s transfers needed to keep the disk 1/2 busy!