Our last lecture on routing

I have a magic trick too! (Ned's magic)

First, I need to define a squash message pattern.

Given a set of processors \( \{ s_0, \ldots, s_n \} \) with \( 0 \leq s_0 < s_1 < \cdots < s_{n-1} < p \)

Processor \( s_i \) sends a message to processor \( i \).

E.g. some processors

\[
\begin{array}{c}
0 \\
1 \\
2 \\
3 \\
4 \\
5 \\
\end{array}
\quad \begin{array}{c}
0 \\
1 \\
2 \\
12 \\
5 \\
\end{array}
\]

Why would I be interested in squash?

Could we use it to pair up free and busy processors

Free

\[
\begin{array}{c}
60 \\
60 \\
60 \\
60 \\
60 \\
\end{array}
\]

Busy

\[
\begin{array}{c}
60 \\
60 \\
60 \\
60 \\
60 \\
\end{array}
\]

Now the magic: I have a butterfly

\[
\begin{array}{c}
0 \\
1 \\
2 \\
3 \\
4 \\
5 \\
\end{array}
\quad \begin{array}{c}
0 \\
1 \\
2 \\
12 \\
5 \\
\end{array}
\]

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

Could be used in a silk implementation for example. Or parallel "come"
Now the magic: Here is a butterfly
(proof of big win)

1 3
2 0
5 4
6
7
8
9
10
11
12
13

Pick some subset that I call
S: \{1, 4, 5, 6, 8, 9, 13, 15\}

$S_0 = 1, S_3 = 6, S_6 = 13$
$S_1 = 4, S_4 = 8, S_7 = 15$
$S_2 = 5, S_5 = 9$

Look MA, no collisions!

This always:

Theorem: This magic always works

Proof: Define a semicontraction is a 1-1 mapping
from \{S_0, \ldots, S_n\} to \{D_0, \ldots, D_n\}
with $S_i \rightarrow D_i$ such that

\[
|S_i - S_j| \geq |D_i - D_j|
\]

This is satisfying, not knowing, do we.

Lemma: Semicontraction rules on butterflies are conflict.

Proof by contradiction:

Suppose 3 conflicts, \Rightarrow some pair must conflict
$S_i \rightarrow D_i$ conflicts with $S_j \rightarrow D_j$

Looks like:

\[
\begin{align*}
S_i & \rightarrow D_i \\
S_j & \rightarrow D_j
\end{align*}
\]

Observe $|S_i - S_j| < 2^k$

Why? $S_i + D_j$ must agree on both low bits.

\[
S_i = \begin{bmatrix}
q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 \\
\end{bmatrix}
\]

But $a_i = b_i \forall i > n$

So

\[
|S_i - S_j| = \left| \sum a_i d_i - \sum b_i d_i \right|
\]
\[\begin{align*}
\sum_{i=0}^{k-1} (a_i - b_i) \cdot 2^i + \sum_{i=k}^{n} c_i \\
= \begin{cases} 
\sum_{i=0}^{k} (a_i - b_i) \cdot 2^i + 0 & \quad \text{if } i \leq k \\
0 + \sum_{i=k}^{n} (c_i - e_i) & \quad \text{if } i = k \end{cases}
\end{align*}\]

\[\begin{align*}
\sum_{i=0}^{k} (a_i - b_i) \cdot 2^i + 0 \\
= 0 + \sum_{i=k}^{n} (c_i - e_i)
\end{align*}\]

\[d_i - d_j > 2^n \]

\text{algorithm:}

1. squash all messages with 0 in bit (n-1).
2. squash up all messages with 1 in bit (n-1).
3. recurse

\[\text{Application of magic: reading 1-1 noise traffic}\]

\text{What can we do with this magic?}

\text{How do we implement it?}

\text{Observe that squash is a semi-contraction and squash. The magic works.}
How to Implement COUNTS.

The problem is to compute the sum

\[ \sum_{i \in S} j \]

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

\text{i.e.}
\begin{align*}
0 & \quad 0 \\
1 & \quad 1 \\
2 & \quad 1 \\
3 & \quad 0 \\
4 & \quad 1 \\
5 & \quad 0 \\
\end{align*}

\text{wait to compute the prefix sum } w

\[ w_i = \text{the number of } 1's \text{ in } v \text{ before } i \]

\[ w_i = \sum_{j=0}^{i} v_j \]

Can compute \( w \) in \( O(n) \) and \( T_{wa} = P \)

Can we do it faster?

Yes. For example, we can compute better.

\[ w_i = v_0 + v_1 + \ldots + v_i \]

\[ = ((v_0 + v_1) + (v_2 + v_3)) + ((v_4 + v_5) + \ldots) \]

\[ \text{To} = \text{log}_2 \text{n} \]

but why?

\text{On a hypercube butterfly, just do it, } x_{\text{out}}
It fits in a butterfly!

Analysis: Time to do scan +

depth of scan = $\log_2 n$

how long to add 2 numbers?

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

\( \Rightarrow \) \( O(\log n) \) time

just to get the final answer
at the last stage.
With pipelining $T_n = O(\log n)$, time $\Theta$ for

Total time to run a T-1 routing

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

Do T-1 routing on $n$ processes:

- Compute scan + up on 0's $O(\log n)$
- Send message in squash down
- Compute scan + down on 0's $O(\log n)$
- Send message in squash up

Do T-1 routing on two subcycles until

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

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

$T_n = O(M \log n)$
Another way to do 1+ with depth 2phL adder
but now only P
(Actually is depth phL, not phLph)

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

max (8, 18)

machine by 3 shifts; do I know?

L
R

sum of ⌊ 1

R

01

R

11

00

01

10

011

1011
Disks:

A spinning platter covered with medium.

Encode info in the orientation of current.

To write, position an electromagnet over a spot and apply current, it induces the store.

To read, in principle, the field goes in reverse, see current from the induced field.

Realism: Giant Magnetoresistance (GMR - Giant Magnetoresistance) is a device whose resistance is a function of magnetic fields. Very sensitive.

GMR discussed in late 80's

=7 about 10 years to spin in disks.

=40 Gb/in²

How to position the head?

Disc spins under motor head radially ("seek")

⇒ can see one spot on disk

Motor is a linear induction motor

(Old motors used stepper motors, time to seek distance in cm = \( \Theta(n) \))

With linear induction motor, head accelerates to \( \frac{1}{2} \)-way point ⇒ time \( \Theta(n) \).

deny peak time = \( \sqrt{2} \) s

4.9 m/sec, 5.4 m/sec, 6.9 m/sec

Best seek = \( \approx 1 \) m

Worst = \( \approx 9 \) m

rot = 10,000 rpm (express in hot)

5,400 rpm

(1/2 that on average)
More disk read code or disk

Usage: put your data on the

Note: put of the disk to
reduce # of seeks.

Head "Files" above disk such

Disk because we should-dynamisize

Last 6 MIL bytes - 0.1 sec

- Quiet
- No noise (without moving disk head)
- Non Repeatable Read

The low-disk "wins" in

14.9

With full spinning hot + vibration

Write the disks by enemy) ... to enemies to defeat.

Now - I don't know what comes here

to wear out.

Observe cost to work from hot +

Drive can now

458 - 799 M/Sec (up to 200 MBs)

Drive avg seek is 5ms see

\[ \frac{5}{8} \text{ ms} \]

\[ \frac{16 \text{ MB} \times \text{transfers needed to}}{\text{use the disk too busy!}} \]