Models of parallel algorithms.

PRAM: CREW, CREW, EREW

NC: ("Nick's class")

Boolean circuits (DAG's made up of AND, OR, NOT gates) with
poly(n) size (number of gates) and
poly(\log(n)) depth, i.e., a.k.a. "work" and "time"

SF \{k \in \mathbb{N}\}

An algorithm is \text{AC}^k if it can be represented by a circuit as above with poly(n) gates
and O(\log^k(n)) depth.

It is \text{NC}^k if it's \text{AC}^k and in addition all gates have
"fan-in" (number of incoming edges) \leq 2.

Clearly, \text{NC}^k \subseteq \text{AC}^k \subseteq \text{NC}^{k+1}.
Adding binary numbers:

```
  1 1 1 1 0 0 0
   1 0 1 1 0
   1 0 1 1 1 0
  ______________
  1 1 0 0 1 0 1 1
```

"Carries"

To compute carry digit in column \( i \), must compute

\[
f_{i-1}( f_{i-2}( f_{i-3}( \ldots f_0(0) ) ) )
\]

where

\[
f_i = \begin{cases} 
  \emptyset & \text{if column } j \text{ is 0+0} \\
  \text{Id} & \text{if } j = 0+1 \lor 1+0 \\
  1 & \text{if } j = 1+1
\end{cases}
\]

New problem: "parallel prefix"

Suppose \( \mathcal{F} \) is a family of functions closed under composition,

\( \{ \emptyset, 1, \text{Id} \} \)

Given \( f_0, f_1, \ldots, f_{n-1} \in \mathcal{F} \),

compute \( g_i = f_i \circ f_{i-1} \circ f_{i-2} \ldots \circ f_0 \)

for all \( i \).

Alg. Define \( h_{i,k} = f_i \circ f_{i+1} \circ \ldots \circ f_{i-2k+1} \).
Observe $h_{i0} = f_x$. (Input layer)

$h_{ijk} = h_{ij,k-1} \cdot h_{i-2,j,k-1}$ (Layer $k$)

$h_{ij,\log_2(w)} = f_0 \circ f_{i-1} \circ \ldots \circ f_j$.

Adding up $n$ binary numbers, each $b$ bits long.

Obvious alg.

\[ \text{depth} \left( \log (n) \right) \]

This is $\text{NC}^2$.

\[ \begin{array}{cccc}
   1 & 1 & 1 & 1 \\
   1 & 0 & 0 & 0 & 1 & 1 \\
   0 & 0 & 1 & 1 & 1 & 1 & 1 \\
   0 & 0 & 1 & 1 & 0 & 0 & 1 \\
   1 & 0 & 1 & 0 & 0 & 1 & 1 \\
   0 & 0 & 1 & 1 & 0 & 0 & 1 \\
   0 & 0 & 1 & 1 & 0 & 0 & 1 \\
   0 & 0 & 1 & 1 & 0 & 0 & 1 \\
   0 & 0 & 1 & 1 & 0 & 0 & 1 \\
   0 & 0 & 1 & 1 & 0 & 0 & 1 \\
   0 & 0 & 1 & 1 & 0 & 0 & 1 \\
\end{array} \]
adding up $n$ numbers of $b$ bits each takes $O(\log(bn))$ time.

Integer multiplication, $O((\log n)^2)$ time

using the above algorithm to sum up partial products.

Matrix multiplication. Just comes down to computing $n$ dot products in parallel.

- Dot product is
  - $n$ multiplies in parallel $O(\log b)$
  - summing $n$ numbers of size $\leq 2b$: $O(\log(bn))$.

Overall, matrix mult is $O(\log^2(bn))$.

Matrix determinant? Next lecture.