PRAM (Parallel Random Access Machine)

1. p general-purpose processors
2. Shared large random access memory
3. All processors run the same program
4. Processors use pid to index into the memory
5. Concurrent read
6. Write conflicts are resolved by variations
Other models

A hypercube (dimension = 3)

In a hypercube with p processors, each processor is connected to \( \lg(p) \) other processors

A bounded-degree network (degree = 4)

The NC class is a complexity class of all problem that are solved in polylogarithmic parallel time and polynomial processors.

- Input size = \( n \)
- Number of processors \( p(n) \in O(n^k) \), where \( k \) is some constant.
- Worst-case time \( T(n) \in O(\log^m n) \), where \( m \) is some constant. (This is called poly-log time.)

Note: The NC class is model independent, i.e., a problem that can be solved in poly-log time by PRAM should be solved in a bounded-degree network in poly-log too.
The binary Fan-In technique
- Not all steps can be parallelized
- Tournament
  - Elements are paired off and compared in “rounds”
Input: Keys $x[0], \ldots, x[n-1]$, initially in memory cells $M[0], \ldots, M[n-1]$, and integer $n$.
Output: The largest key will be left in $M[0]$.
Remarks: cells $M[n]$ to $M[2n-1]$ are initialized to $-\infty$.

```
parTournamentMax(M, n)
    int incr;
    1. Write $-\infty$ (some very small value) into $M[n+pid]$
       incr = 1;
    2. while (incr < n)
       Key big, tmp0, tmp1;
       Read $M[pid]$ into tmp0.
       Read $M[pid + incr]$ into tmp1.
       big = max(tmp0, tmp1);
       write big into $M[pid]$.
       incr = 2 * incr;
```

---

Step 1
Read $M[i]$ into tmp0

Step 2
Read $M[i+1]$ into tmp1
Big = max(tmp0, tmp1)
Write big into $M[i]$.

Step 3
Read $M[i]$ into tmp0

Step 4
Read $M[i+2]$ into tmp1
Big = max(tmp0, tmp1)
Write big into $M[i]$.

Step 5
Read $M[i]$ into tmp0

Note: Only shaded cells are to affect the final results
Correctness by induction

- At step $t$, $\text{incr} = 2^t$. Cell $M[i]$ contains the maximum of $x[i],...,x[i+\text{incr}-1]$

\[
\begin{align*}
\text{tmp0} &= \max(x[i],...,x[i+2^{t-1}-1]) \\
\text{tmp1} &= \max( x[i+2^{t-1}], ..., x[i+2^{t-1} + 2^{t-1}-1] ) \\
\text{big} &= \max(x[i],..., x[i+2^{t-1}])
\end{align*}
\]
Handling Write Conflicts

- **CREW (Concurrent Read, Exclusive Write):**
  only one processor write in a particular cell at any one step; It is illegal to have more than one processor write to one cell at the same time.

- **CRCW (Concurrent Read, Concurrent Write):**
  - Common-Write model
  - Arbitrary-Write model
  - Priority-Write model

---

**Problem**  Boolean or on $n$ Bits
Input: Bits $x_0$, …, $x_{n-1}$, as 0’s and 1’s, in $M[0]$, …, $M[n-1]$
Output: $x_0 \lor \ldots \lor x_{n-1}$ in $M[0]$.

**Solution 1 (CREW)**
Same as the binary Pan-In for finding the max of $n$ integers. It runs in $\Omega(\log n)$ time.
Problem  Boolean or on \( n \) Bits
Input: Bits \( x_0, \ldots, x_{n-1} \), as 0's and 1's, in \( M[0], \ldots, M[n-1] \)
Output: \( x_0 \lor \cdots \lor x_{n-1} \) in \( M[0] \).

Solution 2 (CRCW)
\( \text{commonWriteOr}(M, n) \)
1. \( P_i \) reads \( x_i \) from \( M[i] \);
   If \( x_i \) is 1; then \( P_i \) writes 1 in \( M[0] \).

Correctness: Since all the processors that write into \( M[0] \) write the same value, the program is a legal program under Common-Write model.
Running time: It takes just one step when \( n \) processors are available.

Input: Keys \( x[0], \ldots, x[n-1] \), initially in memory cells \( M[0], \ldots, M[n-1] \), and integer \( n > 2 \).
Output: The largest key will be left in \( M[0] \).

Remarks: Processors \( P_{ij} \) have index \( i = \text{pid}/n \), and \( j = \text{pid} - ni \). If \( i \geq j \), the processor does not work.

\( \text{fastMax}(M, n) \)
1. Compute \( i \) and \( j \) from \( \text{pid} \).
   if \( i > j \) return;
   \( P_{ij} \) reads \( x_i \) (from \( M[i] \)).

2. \( P_{ij} \) reads \( x_j \) (from \( M[j] \)).
   \( P_{ij} \) compares \( x_i \) and \( x_j \).
   Let \( k \) be the index of the smaller key (if tied).
   \( P_{ij} \) writes 1 in \( \text{loser}(k) \).
   // At this point, every key other than the largest
   // has lost a comparison

3. \( P_{i+1} \) reads \( \text{loser}(i) \) (and \( P_{0,n-1} \) reads \( \text{loser}(n-1) \)).
   The processor that read a 0 writes \( x_i \) in \( M[0] \). (\( P_{i+1} \) would write \( x_{i+1} \))
   // This processor already has the needed \( x \) in its local memory
   // from steps 1 and 2.
<table>
<thead>
<tr>
<th>Initial memory contents ( (n = 4) )</th>
<th>Input</th>
<th>Loser array</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 7 3 6 0 0 0 0</td>
<td>0 3 4 7</td>
<td></td>
</tr>
</tbody>
</table>

After Step 2

\[ \begin{align*}
P_{0,1} & \quad P_{0,2} & \quad P_{0,3} \\
1 & \quad 1 & \quad 1 \\
\end{align*} \]

After Step 3

\[ \begin{align*}
P_{1,2} \\
7 \\
\end{align*} \]