A Model

Fixed Connection Network

- Processors Labeled $P_1, P_2, \ldots, P_N$
- Each Processor knows its Unique $ID$
- Local Control
- Local Memory
- Fixed Bi-directional Connections
- Synchronous
  - Global Clock Signals Next Phase
Operations at Each Phase

Each Time the Global Clock Ticks

- Receive Input from Neighbors
- Inspect Local Memory
- Perform Computation
- Generate Output for Neighbors
- Update Local Memory
A Model of Cooperation: Bucket Brigades

- $N$ Processors, Labeled $P_1$ to $P_N$

- Processor $P_i$ is connected to $P_{i+1}$, $i < N$ and $P_{i-1}$, $i > 0$
A Sort Algorithm

Odd-Even Transposition on Linear Array

- The Array is $X[1 : N]$
- $P_i$'s Local Variable $X$ is $X[i]$
- $P_i$'s have a Local Variables $Y$ and a Global/Singular variable $Step$
- $Step$ is initialized to Zero (0) at all $P_i$
- Compares and Exchanges are done alternately at Odd/Even - Even/Odd Pairs
Odd-Even Transposition

Algorithmic Description of Parallel Bubble Sort

At Each Clock Tick and For Each $P_i$ do {

$Step$ $++$;

if parity($i$) == parity($Step$) & $i < N$ then

Read from $P_{i+1}$ to $Y$;

$X = \min(X,Y)$

else if $i > 1$ then

Read from $P_{i-1}$ to $Y$;

$X = \max(X,Y)$;

}
Example of Parallel Bubble Sort

Sort 4 Numbers 7, 2, 3, 1 on an Array of 4 Processors

Case of 4, 3, 2, 1 Takes 4 Steps
Measuring Benefits

How Do We Measure What We Have Gained?

- Let \( T_1(N) \) be the Best Sequential Algorithm
- Let \( T_P(N) \) be the Time for Parallel Algorithm (P processors)
- The Speedup \( S_P(N) \) is \( T_1(N)/T_P(N) \)
- The Cost \( C_P(N) \) is \( P \times T_P(N) \), assuming \( P \) processors
- The Work \( W_P(N) \) is the summation of the number of steps taken by each of the processors. It is often, but not always, the same as Cost.
- The Cost Efficiency \( CE_P(N) \) (often called efficiency \( Ep(N) \)) is
  \[
  S_P(N)/P = C_1(N) / C_P(N) = T_1(N) / (P \times T_P(N))
  \]
- The Work Efficiency \( WE_P(N) \) is
  \[
  W_1(N) / W_P(N) = T_1(N) / W_P(N)
  \]
Napkin Analysis of Parallel Bubble

How'd We Do? - Well, Not Great!

- $T_I(N) = N \lg N$  
  Optical Sequential
- $T_P(N) = N$  
  Parallel Bubble
- $S_N(N) = \lg N$  
  Speedup
- $C_N(N) = W_N(N) = N^2$  
  Cost and Work
- $E_N(N) = \lg N / N$  
  Cost and Work Efficiency

But Good Relative to Sequential Bubble

$S_N(N) = N^2 / N = N$  ;  $E_N(N) = S_N(N) / N = 1$!

Really Fast Sorts — 8 —  
Charles E. Hughes — UCF
Non-Scalability of Odd-Even Sort

Assume we start with 1 processor sorting 64 values, and then try to scale up by doubling number of values (N), each time we double number of processors (P) in a ring. The cost of the parallel sort requires each processor to sort its share of values (N/P), and then do P swaps and merges. Since P processors are busy, the cost is N lg N/P. After the local sort, sets are exchanged, merged, and parts thrown away. The merge costs N/P on each of P processors, for a Cost of N, and P-1 such merges occur, for a total cost of N×(P-1).

Efficiency is then

\[ E = \frac{N \lg N}{(N \lg N/P + N \times (P-1))} = \frac{\lg N}{(P - 1 + \lg N - \lg P)} \]

First 2 columns double N as P doubles. Second three try to increase N to keep efficiency when P doubles.

<table>
<thead>
<tr>
<th>N</th>
<th>P</th>
<th>E</th>
<th>N</th>
<th>P</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>1</td>
<td>1.0000</td>
<td>64</td>
<td>1</td>
<td>1.0000</td>
</tr>
<tr>
<td>128</td>
<td>2</td>
<td>1.0000</td>
<td>4096</td>
<td>2</td>
<td>1.0000</td>
</tr>
<tr>
<td>256</td>
<td>4</td>
<td>0.8889</td>
<td>16777216</td>
<td>4</td>
<td>0.9600</td>
</tr>
<tr>
<td>512</td>
<td>8</td>
<td>0.6923</td>
<td>2.81475E+14</td>
<td>8</td>
<td>0.9231</td>
</tr>
<tr>
<td>1024</td>
<td>16</td>
<td>0.4762</td>
<td>7.92282E+28</td>
<td>16</td>
<td>0.8972</td>
</tr>
<tr>
<td>2048</td>
<td>32</td>
<td>0.2973</td>
<td>6.2771E+57</td>
<td>32</td>
<td>0.8807</td>
</tr>
<tr>
<td>4096</td>
<td>64</td>
<td>0.1739</td>
<td>3.9402E+115</td>
<td>64</td>
<td>0.8707</td>
</tr>
<tr>
<td>8192</td>
<td>128</td>
<td>0.0977</td>
<td>1.5525E+231</td>
<td>128</td>
<td>0.8649</td>
</tr>
</tbody>
</table>
Cost for Finding Max Value in a List

Given a sequence A of n elements find the largest of these elements.

**Serial Algorithm.**

Largest = A [0]

For i = 1 to n-1 do { if A [i] > Largest then Largest = A [i] }

n - 1 comparison.

**A Parallel Algorithm**
Efficiency of Binary Tree Max

Assume Full Binary Tree

- \( T_{N/2}(N) = T_{N/4}(N/2) + 1, \ N > 1 \)
  
  \( T_1(2) = 1 \)
  
  \( T_{N/2}(N) = \lg N = O(\lg N) \)

- \( C_N(N) = N \lg N = O(N \lg N) \)
  
  \( E_N(N) = N / N \lg N = O(1 / \lg N) \)

- \( W_{N/2}(N) = W_{N/4}(N/2) + N/2, \ N > 2 \)
  
  \( W_1(2) = 1 \)
  
  \( W_{N/2}(N) = N \ - \ 1 = O(N) \)

- This is optimally work efficient.
- But it is not optimally cost efficient.
Finding the Maximum by Controlled Anarchy

Step#1: Everyone’s an Optimist

12  6  15  7

We're #1  We're #1  We're #1  We're #1

Ok

12

We're #1  We're #1  We're #1  We're #1

Ok

6

We're #1  We're #1  We're #1  We're #1

Ok

15

We're #1  We're #1  We're #1  We're #1

Ok

7

We're #1  We're #1  We're #1  We're #1

Ok
This is the Meatiest Part

Step#2: Realism Sets In

12

6

15

7

Rats!
That’s All Folks

Step#3: Reporting the Answer

```
  12  6  15  7
   □ □ □ □
12
   □ □ □ □
  □ □ □ □
  6
   □ □ □ □
15
   □ □ □ □
  □ □ □ □
  7
   □ □ □ □
   □ □ □ □
```

15 is boss
Analysis of Very Fast Max

Optimal in Time, Not Work on CRCW (Concurrent Read Concurrent Write) PRAM (Parallel Random Access Machine)

- Assign $N$ processors to initialize $M$ in 1 step.

- Assign all $N^2$ processors to first statement to fill $B$ in 1 step.

- Assign all $N^2$ processors to 2nd statement to fill $M$ in 1 step.

- Assign $N$ processors to 3rd statement to select $\text{maxVal}$ in 1 step.
That Was Inefficient but Real Fast

- Can Solve Any Size Problem in 3 Steps
  But we need to make unreasonable assumptions about memory (CRCW)

- Use Lots of Processors
  Over a Million to Find Max of 1000

- We Want Fast but Not Too Expensive