Goals of the lecture

Repeated Computation of a Global Function

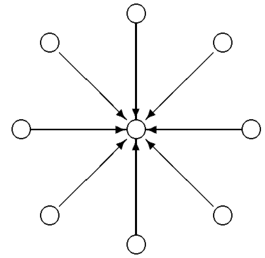
- Deadlock Detection
- Clock Synchronization
- Distributed Branch and Bound Search
- Distributed Debugging

Desirable Characteristics

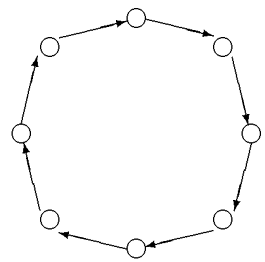
- Light Load
  - not more than k messages/time step
- High Concurrency
  - $\log_k N$ time steps
- Symmetry (Equitable Workload)
  - load balancing
  - fairness

Some Possible Approaches

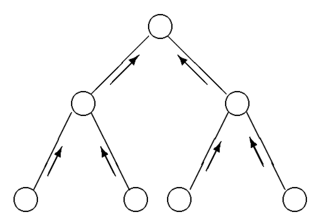
- Centralized



- Ring-based



- Hierarchical



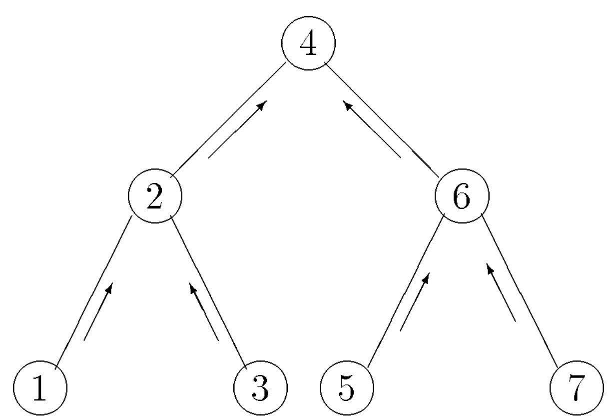
All links are logical connections

Message Flow Table

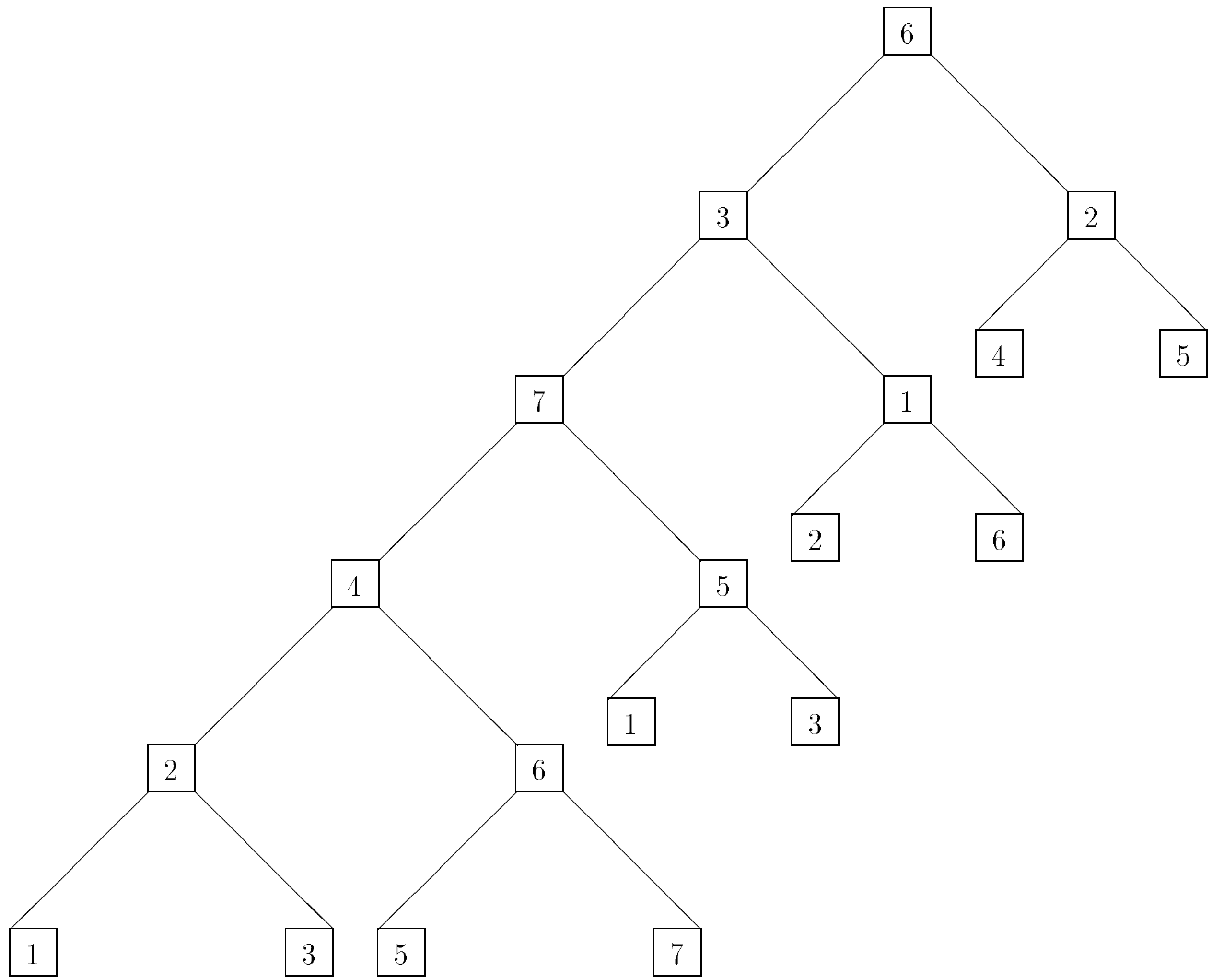
Static Hierarchy

- Number of nodes (processe) = 7

time step	Messages		
1	$1,3 \to 2$	$5,7 \to 6$	
2	$1,3 \to 2$	$5,7 \to 6$	$2,6 \to 4$
3	$1,3 \to 2$	$5,7 \to 6$	$2,6 \to 4$



Overlapping Tress



Message Flow Table

- Revolving Hierarchy
  - number of nodes = 7

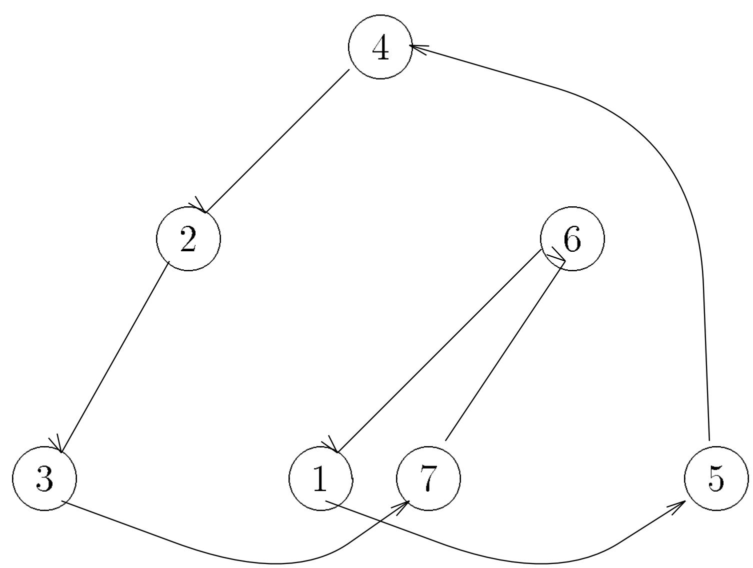
time step	Messages		idle
1	$2 \leftarrow 1,3$	$6 \leftarrow 5,7$	4
2	$4 \leftarrow 2,6$	$5 \leftarrow 1,3$	7
3	$7 \leftarrow 4,5$	$1 \leftarrow 2,6$	3
4	$3 \leftarrow 7,1$	$2 \leftarrow 4,5$	6

- Reorganization of Hierarchy
- Reuse of messages

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 5 & 1 & 7 & 2 & 6 & 3 & 4 \end{pmatrix}$$

Requirements for Desired Permutation

- Gather tree constraints
  - interior nodes of T_i = subtree of T_{i+1}
- Fairness constraints
  - No cycle of size less than N.



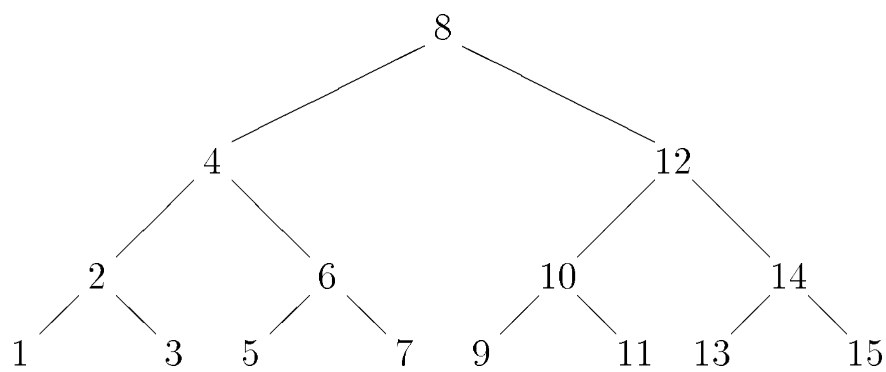
Interesting but ..

- Does there always exist such a permutation ?

- Is there a systematic method to find it ?

- Is there an efficient implementation for it ?

Method to Generate the Permutation



$next(x)$:

[

$even(x) \rightarrow \quad x' := x/2;$ (* gather tree constraint *)

$\square$

$odd(x) \wedge (x < 2^{n-1}) \rightarrow \quad x' := x + 2^{n-1};$ (*fairness constraint *)

$\square$

$odd(x) \wedge (x > 2^{n-1}) \rightarrow$

[$x = N \rightarrow \quad x' := (N-1)/2;$

$\square$

$x \neq N \rightarrow \quad y := x - 2^{n-1} + 2$

$x' := y * 2^{\lceil \log \frac{2^n}{y} - 1 \rceil}$

]

]

Implementation 1

Q: Who should I send message to at time t ?

$$
\begin{aligned}
msg(x,t) &= next^{-t}(parent(next^{t}(x))), \text{ if } next^{t}(x) \text{ is odd} \\
&= nil, \text{ otherwise}
\end{aligned}
$$

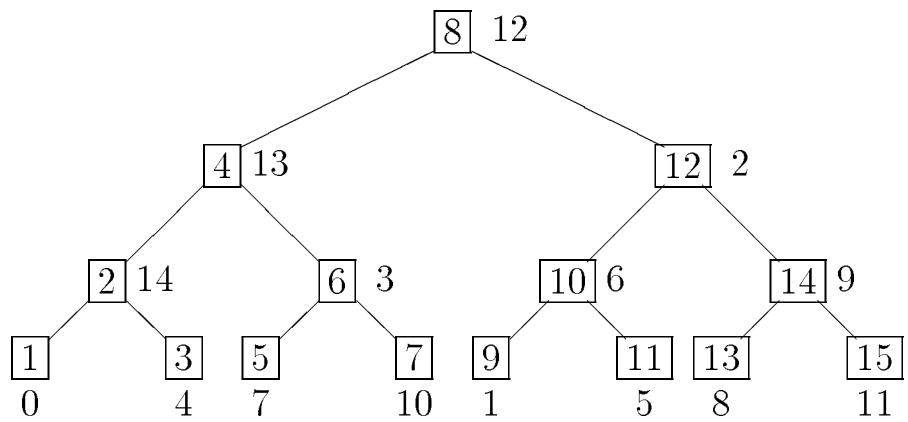
x is in-order label

$next$ is the new position function

$parent$ is the parent function for in-order labeling

parent of $x = x$ with last two bits changed to 10

Implemetation 2



$$
\begin{aligned}
msg(x,t) &= new\_parent(x+t) - t, \qquad \text{if } (x+t) \text{ is a leaf-node} \\
&= nil, \qquad \text{otherwise}
\end{aligned}
$$

- Just need to store new_parent array

Communication Required

- Communication distance set (CDS)

$$= \{new\_parent(j) - j \mid j \text{ a leaf node } \}$$

- process x will send a message to process y iff $y - x \in CDS$.
  - for $N = 15$

$$CDS = \{1, 5, 8, 10, 13, 14\}.$$

- CDS depends on the $next$ function.

Data Gathering and Broadcasting

- a process can send/receive only one message per time step
- require that the same set of messages is used for data gathering and broadcasting.
- Constraints :
  1. fairness contraints
     - equal load
  2. gather tree constraints.
     - $G(t)$ available at $t + \log N$ time step at one node.
  3. broadcast constraints.
     - $G(t)$ available at $t + 2\log N$ time step at all nodes.

Message Flow Table

time step	Messages			
0	$0 \rightarrow 7$	$4 \rightarrow 6$	$1 \rightarrow 3$	$2 \rightarrow 5$
1	$7 \rightarrow 6$	$3 \rightarrow 5$	$0 \rightarrow 2$	$1 \rightarrow 4$
2	$6 \rightarrow 5$	$2 \rightarrow 4$	$7 \rightarrow 1$	$0 \rightarrow 3$
3	$5 \rightarrow 4$	$1 \rightarrow 3$	$6 \rightarrow 0$	$7 \rightarrow 2$
4	$4 \rightarrow 3$	$0 \rightarrow 2$	$5 \rightarrow 7$	$6 \rightarrow 1$

- fairness in workload

- four times less messages than static hierarchy

Method to Generate the Permutation

$bcnext(x)$ ::

$$
\begin{array}{lll}
[& b_0 = 1 \rightarrow & x' := RS_0(x) \\
 & & \text{(* gather tree *)} \\
\Box & & \\
 & (b_0 = 0) \wedge (b_1 = 0) \rightarrow & x' := RS_1(x) \\
 & & \text{(* broadcast *)} \\
\Box & & \\
 & (b_0 = 0) \wedge (b_1 = 1) \rightarrow & x' := LS_1^a\left(\left(LS_0^b(x) + 2\right) \bmod 2^{n-1}\right); \\
 & & \text{(* fairness *)} \\
] & &
\end{array}
$$

$b_{n-1} \cdots b_0 = x$

RS_p = Right shift with p as m.s.b

LS_p = Left shift with p as l.s.b.

a = number of leading zeros

b = number of leading ones

Algorithm to find Current Minimum in the Network

- Distributed branch and bound
- Distributed simulation
  - process x : $step = 0$

$$
\begin{array}{lll}
*[& dest\_msg(x, step) \neq nil \rightarrow & send\_msg(dest\_msg(x, step), mymin) \\
 & & step := step + 1 \\
 & src\_msg(x, step) \neq nil \rightarrow & recv\_msg(src\_msg(x, step), hismin) \\
 & & \text{recompute } mymin \\
 & & step := step + 1 \\
]
\end{array}
$$

Performance of the Algorithm

- at most k messages handled by a node/time step
- the global function $G(t)$ is available at $t + \lceil \log N \rceil$ time steps.
- a throughput of one global function per times step.
- number of messages required $\sim$ half of that for static hierarcy.
- equal workload distribution

Extensions

- General N
  - use virtual nodes

- General k
  - methods to generate permutations for binary trees generalize to k-ary trees.

- asynchronous messages
  - can be used instead od synchronous messages. Nodes synchronized due to "receives".

Conclusions

- Useful for algorithms that
  - use hierarchical control
  - run for long time
- main advantages
  - equal workload distribution.
  - reduction in number of messages due to their reuse
- main disadvantages
  - requires that the communication network has more edges than static hierarchy.