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

![Centralized Diagram]

- Ring-based

![Ring-based Diagram]

- Hierarchical

![Hierarchical Diagram]

All links are logical connections
Message Flow Table

Static Hierarchy
- Number of nodes (processes) = 7

<table>
<thead>
<tr>
<th>time step</th>
<th>Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1, 3 → 2</td>
</tr>
<tr>
<td>2</td>
<td>1, 3 → 2</td>
</tr>
<tr>
<td>3</td>
<td>1, 3 → 2</td>
</tr>
</tbody>
</table>
Overlapping Trees
Message Flow Table

- **Revolving Hierarchy**
  - number of nodes = 7

<table>
<thead>
<tr>
<th>time step</th>
<th>Messages</th>
<th>idle</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2 ← 1, 3</td>
<td>6 ← 5, 7</td>
</tr>
<tr>
<td>2</td>
<td>4 ← 2, 6</td>
<td>5 ← 1, 3</td>
</tr>
<tr>
<td>3</td>
<td>7 ← 4, 5</td>
<td>1 ← 2, 6</td>
</tr>
<tr>
<td>4</td>
<td>3 ← 7, 1</td>
<td>2 ← 4, 5</td>
</tr>
</tbody>
</table>

- **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 $T_i = \text{subtree of } T_{i+1}$

- Fairness constraints
  - No cycle of size less than $N$. 

![Diagram of tree structure]
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

\[ \text{next}(x) : \]
\[ [ \]
\[ \begin{align*}
\text{even}(x) \rightarrow & \quad x' := x/2; \, (* \text{gather tree constraint} *) \\
\text{odd}(x) \land (x < 2^{n-1}) \rightarrow & \quad x' := x + 2^{n-1}; \, (* \text{fairness constraint} *) \\
\text{odd}(x) \land (x > 2^{n-1}) \rightarrow & \quad \begin{cases}
x = N \rightarrow & x' := (N - 1)/2; \\
x \neq N \rightarrow & y := x - 2^{n-1} + 2 \\
& x' := y * 2^{\lfloor \log_2 \frac{2^n - 1}{y} \rfloor}
\end{cases}
\end{align*} \]
\[ ] \]
Implementation 1

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

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

- $x$ 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
Repeated Computation of a Global Function

Implementation 2

\[
\text{msg}(x, t) = \text{new\_parent}(x + t) - t, \quad \text{if } (x + t) \text{ is a leaf-node}
\]

\[
= \text{nil}, \quad \text{otherwise}
\]

- Just need to store \text{new\_parent} array
Communication Required

- Communication distance set (CDS)
  \[ CDS = \{new\_parent(j) - j \mid j \text{ a leaf node}\} \]

- 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 constraints
   - 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

<table>
<thead>
<tr>
<th>time step</th>
<th>Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 → 7, 4 → 6, 1 → 3, 2 → 5</td>
</tr>
<tr>
<td>1</td>
<td>7 → 6, 3 → 5, 0 → 2, 1 → 4</td>
</tr>
<tr>
<td>2</td>
<td>6 → 5, 2 → 4, 7 → 1, 0 → 3</td>
</tr>
<tr>
<td>3</td>
<td>5 → 4, 1 → 3, 6 → 0, 7 → 2</td>
</tr>
<tr>
<td>4</td>
<td>4 → 3, 0 → 2, 5 → 7, 6 → 1</td>
</tr>
</tbody>
</table>

- fairness in workload
- four times less messages than static hierarchy
Method to Generate the Permutation

\[bcnext(x) ::\]
\[\begin{align*}
 b_0 = 1 \to & \quad x' := RS_0(x) \\
 & \quad \text{(* gather tree *)}
\end{align*}\]

\[b_0 = 0 \land (b_1 = 0) \to x' := RS_1(x) \\
\quad \text{(* broadcast *)}\]

\[b_0 = 0 \land (b_1 = 1) \to x' := LS_1^{a_1}((LS_0^{b_1}(x) + 2) \mod 2^{n-1}); \\
\quad \text{(* fairness *)}\]

\[b_{n-1} \cdots b_0 = x\]
\[RS_p = \text{Right shift with } p \text{ as m.s.b}\]
\[LS_p = \text{Left shift with } p \text{ as l.s.b.}\]
\[a = \text{number of leading zeros}\]
\[b = \text{number of leading ones}\]
**Algorithm to find Current Minimum in the Network**

- Distributed branch and bound
- Distributed simulation

```
process x : step = 0

*[
    dest_msg(x, step) ≠ nil → send_msg(dest_msg(x, step), mymin)
    step := step + 1

    src_msg(x, step) ≠ nil → recv_msg(src_msg(x, step), hismin)
    recompute mymin
    step := step + 1
]
```
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 hierarchy.

- 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 of 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.