The Selector-Tree Network: A New Self-Routing and Nonblocking Interconnection Network

Tripti Jain, Klaus Schneider, and Anoop Bhagyanath

Embedded Systems Group
Department of Computer Science
University of Kaiserslautern

ReCoSoC, 2016
Outline

1. Introduction
2. Selector-Tree Network
3. Complexity Analysis
4. Conclusions
Interconnection Network

an idealized interconnection network:

- takes a set of $n$ input ports labeled $0, \ldots, n-1$ and
- sets up connections between them and a set of $m$ output ports $0, \ldots, m-1$
- with the connections determined by control signals
an interconnection network can be either single-stage or multi-stage

- **single-stage:**
  - the individual control boxes set up to $n$ times to get data from one node to another
  - data may have to pass through several PEs to reach its destination, e.g., mesh interconnection, hypercube, etc.

- **multi-stage:**
  - several sets of switches in parallel
  - data only needs to pass through several switches, not several processors, e.g. Benes network, Omega network, butterfly network, etc.
Interconnection Networks

establishing connections

- **self-routing**: target address determines the unique route
- **nonblocking networks**: two arbitrary components can be connected without affecting existing connections
- **rearranging networks**: two arbitrary components can be connected by rearranging existing connections
- **blocking networks**: if certain connections exclude certain other connections
## Multistage Networks

<table>
<thead>
<tr>
<th>networks</th>
<th>self-routing</th>
<th>nonblocking</th>
<th>rearranging</th>
</tr>
</thead>
<tbody>
<tr>
<td>Omega</td>
<td>✓</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>butterfly</td>
<td>✓</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>flattened butterfly</td>
<td>✓</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Benes network</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Selector-Tree Network

- a self-routing and nonblocking interconnection network
- **nonblocking**: capable of routing any permutation of its $n$ inputs to its $n$ outputs
- **self-routing**: target addresses define the conflict-free routes
- no additional set-up time
- predictable real-time behavior
**general idea:**
- first forward addresses to the right halves
- then recursively route the generated sub-permutations

**green modules (comparators):**
route to either lower or upper half

**orange modules (selectors):**
select valid inputs and forward them to outputs
Implementation of the Selector Modules

two possible implementations

- sequential implementation: minimum work
- parallel implementation: minimum time
Sequential Algorithm

\[
\text{module SeqSel([n]bool } v, [n]\text{nat } m_{\text{in}},
\text{[n/2]nat } m_{\text{out}}) \{
\text{nat } j;
\text{j = 0;}
\text{for}(i=0\ldots n-1) \{
\text{if}(v[i]) \{
\text{next}(m_{\text{out}}[j]) = m_{\text{in}}[i];
\text{next}(j) = j+1;
\}
\text{pause;}
\}
\}
\]

- \textbf{idea}: determine the \(n/2\) valid inputs and forward them in any order to its outputs
- \(m_{\text{in}}[0\ldots n-1]\): \(n\) incoming messages
- \(v[0\ldots n-1]\): valid bits
- \(m_{\text{out}}[0\ldots n/2-1]\): \(n/2\) output messages
- scan the input array \(v\) and copies an entry \(m_{\text{in}}[i]\) to \(m_{\text{out}}[j]\) whenever \(v[i]\) holds
- computation can be done in \(n\) steps with \(O(n)\) work
Parallel Algorithm – Step 1: Parallel Prefix Sum

based on parallel prefix sum

compute \( o[i] = \sum_{j=0}^{i} v[j] \)

\( o[i] \) is the number of 1s in array \( v \) left to index \( i \) or at \( i \)

can be done in \( \log(n) \) steps with \( O(n \log(n)) \) work

with a little more effort reduction to \( O(\log(n)) \) work is possible
Parallel Algorithm – Step 1: Example

\[ v = [v_0, \ldots, v_7] = [0, 0, 1, 0, 1, 1, 0, 1] \]
Parallel Algorithm – Step 2: Output Assignment

- **idea**: compute the indices that hold the valid messages
- $o[i]-1$ is the index of the output array where the valid message $m_{in}[i]$ has to be stored
- scan the array in parallel and assign $m_{in}[i]$ to $m_{out}[o[i]-1]$
- computation can be done in 1 step with $O(n)$ work

```plaintext
module ParSel([n]nat ?o, [n]bool ?v,
            [n]nat ?m_in, [n/2]nat m_out)
{
    for (i=0..n-1)
        if (v[i])
            next(m_out[o[i]-1]) = m_in[i];
}'''
```
consider the following measures:

- **depth**: length of the longest path through combinational gates
- **size**: number of gates
- **cycles**: number of hardware cycles
- **time**: Time \( (n) = \text{Cycles} \(n) \cdot \text{Depth} \(n) \)
- **work**: number of actions processed, i.e., Work \( (n) \leq \text{Cycles} \(n) \cdot \text{Size} \(n) \)
# Complexity Results

<table>
<thead>
<tr>
<th></th>
<th>SeqSel module</th>
<th>SeqSel network</th>
<th>ParSel module</th>
<th>ParSel network</th>
</tr>
</thead>
<tbody>
<tr>
<td>time cycles</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(\log(n))$</td>
<td>$O(\log(n)^2)$</td>
</tr>
<tr>
<td></td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(\log(n))$</td>
<td>$O(\log(n)^2)$</td>
</tr>
<tr>
<td>size work</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(n^2 \log(n))$</td>
<td>$O(n^2 \log(n))$</td>
</tr>
<tr>
<td></td>
<td>$O(n)$</td>
<td>$O(n \log(n))$</td>
<td>$O(n^2 \log(n))$</td>
<td>$O(n^2 \log(n))$</td>
</tr>
</tbody>
</table>
Conclusions

- a new self-routing and nonblocking interconnection network
- predictable real-time behavior
- two alternatives for selector module implementation
  - sequential
  - parallel
- sequential version is slower compared to the parallel version that requires $O(\log(n)^2)$ time
- parallel implementation is not work efficient
- two alternatives can combined to achieve optimal networks for particular sizes
Questions?