Topics
- Network design issues
- Network Topology

Evaluation Criteria:
- Latency
- Bisection Bandwidth
- Contention and hot-spot behavior
- Partitionability
- Cost and scalability
- Fault tolerance

Networks
- How do we move data between processors?
- Design Options:
  - Topology (what)
  - Routing (which)
  - Physical implementation
    - Switching (circuit or packet) (how)
    - Flow control (when)

Buses
- Simple and cost-effective for small-scale multiprocessors
- Not scalable (limited bandwidth; electrical complications)
### Crossbars

- Each port has link to every other port
- Low latency and high throughput
- Cost grows as $O(N^2)$ so not very scalable.
- Difficult to arbitrate and to get all data lines into and out of a centralized crossbar.
- Used in small-scale MPs (e.g., C.mmp) and as building block for other networks (e.g., Omega).

### Rings

- Cheap: Cost is $O(N)$.
- Point-to-point wires and pipelining can be used to make them very fast.
- High overall bandwidth
- High latency $O(N)$
- Examples: KSR machine, Hector

### Trees

- Cheap: Cost is $O(N)$.
- Latency is $O(\log N)$.
- Easy to layout as planar graphs (e.g., H-Trees).
- For random permutations, root can become bottleneck.
- To avoid root being bottleneck, notion of Fat-Trees (used in CM-5)

### Hypercubes

- Also called binary n-cubes. # of nodes = $N = 2^n$.
- Latency is $O(\log N)$; Out degree of PE is $O(\log N)$
- Minimizes hops; good bisection BW; but tough to layout in 3-space
- Popular in early message-passing computers (e.g., intel iPSC, NCUBE)
- Used as direct network ==> emphasizes locality
### Multistage Logarithmic Networks

**Key Idea:** have multiple layers of switches between destinations.
- Cost is $O(N\log N)$; latency is $O(\log N)$; throughput is $O(N)$.
- Generally indirect networks.
- Many variations exist (Omega, Butterfly, Benes, ...).
- Used in many machines: BBN Butterfly, IBM RP3, ...

---

### Omega Network

- All stages are same, so can use recirculating network.
- Single path from source to destination.
- Can add extra stages and pathways to minimize collisions and increase fault tolerance.
- Can support combining. Used in IBM RP3.

---

### Butterfly Network

- Equivalent to Omega network. Easy to see routing of messages.
- Also very similar to hypercubes (direct vs. indirect though).
- Clearly see that bisection of network is $(N/2)$ channels.
- Can use higher-degree switches to reduce depth.

---

### k-ary n-cubes

- Generalization of hypercubes (k-nodes in a string)
- Total # of nodes = $N = k^n$.
- $k > 2$ reduces # of channels at bisection, thus allowing for wider channels but more hops.
Relationship of Butterflies to Hypercubes

Wiring is isomorphic
Except that Butterfly always takes \( \log n \) steps

Advantages of Low-Dimensional Nets

What can be built in VLSI is often wire-limited
LDNs are easier to layout:
• more uniform wiring density (easier to embed in 2-D or 3-D space)
• mostly local connections (e.g., grids)

Compared with HDNs (e.g., hypercubes), LDNs have:
• shorter wires (reduces hop latency)
• fewer wires (increases bandwidth given constant bisection width)
  • increased channel width is the major reason why LDNs win!

LDNs have better hot-spot throughput
• more pins per node than HDNs

Real World 2D mesh

1824 node Paragon: 16 x 114 array

Embeddings in two dimensions

Embed multiple logical dimension in one physical dimension using long wires

6 x 3 x 2
Properties of Some Topologies

<table>
<thead>
<tr>
<th>Topology</th>
<th>Degree</th>
<th>Diameter</th>
<th>Ave Dist</th>
<th>Bisection</th>
</tr>
</thead>
<tbody>
<tr>
<td>1D Array</td>
<td>2</td>
<td>N-1</td>
<td>N / 3</td>
<td>1</td>
</tr>
<tr>
<td>1D Ring</td>
<td>2</td>
<td>N/2</td>
<td>N/4</td>
<td>2</td>
</tr>
<tr>
<td>2D Mesh</td>
<td>4</td>
<td>2 (N^{1/2} - 1)</td>
<td>2/3 N^{1/2}</td>
<td>N^{1/2}</td>
</tr>
<tr>
<td>2D Torus</td>
<td>4</td>
<td>N^{1/2}</td>
<td>1/2 N^{1/2}</td>
<td>2N^{1/2}</td>
</tr>
<tr>
<td>k-ary n-cube</td>
<td>2n</td>
<td>nk/2</td>
<td>nk/4</td>
<td>nk/4</td>
</tr>
<tr>
<td>Hypercube</td>
<td>n = log N</td>
<td>n</td>
<td>n/2</td>
<td></td>
</tr>
</tbody>
</table>

All have some “bad permutations”
- many popular permutations are very bad for meshes (transpose)
- randomness in wiring or routing makes it hard to find a bad one!

Switching Alternatives

- Circuit Switching
- Packet Switching
  - Store-and-forward
  - Cut-through
    - Virtual cut-through
    - wormhole

Routing

Recall: routing algorithm determines
- which of the possible paths are used as routes
- how the route is determined
- \( R: N \times N \rightarrow C \), which at each switch maps the destination node \( n_d \) to the next channel on the route

Issues:
- Routing mechanism
  - arithmetic
  - source-based port select
  - table driven
  - general computation
- Properties of the routes
- Deadlock free

Routing Mechanism

need to select output port for each input packet
- in a few cycles
Reduce relative address of each dimension in order
- Dimension-order routing in k-ary d-cubes
- e-cube routing in n-cube
Routing Mechanism

- need to select output port for each input packet - quickly!
- Simple arithmetic in regular topologies
  - ex: $\Delta x, \Delta y$ routing in a grid
    - west ($-x$) $\Delta x < 0$
    - east ($+x$) $\Delta x > 0$
    - south ($-y$) $\Delta y < 0$
    - north ($+y$) $\Delta y > 0$
    - processor $\Delta x = 0, \Delta y = 0$
- Reduce relative addr of each dim in order
  - Dimension-order routing in k-ary d-cubes
  - e-cube routing in n-cube

Routing Mechanism (cont)

<table>
<thead>
<tr>
<th>$P_0$</th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
</tr>
</thead>
</table>

Source-based
- message header carries series of port selects
- used and stripped en route
- CRC? Packet Format?
- CS-2, Myrinet, MIT Artic

Table-driven
- message header carried index for next port at next switch
  - $o = R[i]$
- table also gives index for following hop
  - $o, I' = R[i']$
- ATM, HPPI

Properties of Routing Algorithms

Deterministic
- route determined by (source, dest), not intermediate state (i.e. traffic)

Adaptive
- route influenced by traffic along the way

Minimal
- only selects shortest paths

Deadlock free
- no traffic pattern can lead to a situation where no packets mover forward

Deadlock Freedom

How can it arise?
- necessary conditions:
  - shared resource
  - incrementally allocated
  - non-preemptible
  - think of a channel as a shared resource that is acquired incrementally
    - source buffer then dest. buffer
    - channels along a route

How do you avoid it?
- constrain how channel resources are allocated
- ex: dimension order

How do you prove that a routing algorithm is deadlock free
**Proof Technique**

Resources are logically associated with channels

Messages introduce dependences between resources as they move forward

Need to articulate possible dependences between channels

Show that there are no cycles in Channel Dependence Graph

- find a numbering of channel resources such that every legal route follows a monotonic sequence

=> no traffic pattern can lead to deadlock

Network need not be acyclic, only channel dependence graph

---

**Example: k-ary 2D array**

Theorem: x,y routing is deadlock free

**Numbering**

- +x channel \((i, y) \rightarrow (i+1, y)\) gets \(i\)
- similarly for \(-x\) with 0 as most positive edge
- +y channel \((x, j) \rightarrow (x, j+1)\) gets \(N+j\)
- similarly for \(-y\) channels

Any routing sequence: x direction, turn, y direction is increasing

---

**Channel Dependence Graph**

---

**Examples**

Why is the obvious routing on X deadlock free?

- butterfly?
- tree?
- fat tree?

Any assumptions about routing mechanism? amount of buffering?

What about wormhole routing on a ring?
Deadlock free wormhole networks?

Basic dimension-order routing doesn't work for k-ary d-cubes
- only for k-ary d-arrays (bi-directional)

Idea: add channels!
- provide multiple “virtual channels” to break dependence cycle
- good for BW too!

Don't need to add links, or xbar, only buffer resources
This adds nodes the the CDG, remove edges?

Routing Design Summary

Routing Algorithms restrict the set of routes within the topology
- simple mechanism selects turn at each hop
- arithmetic, selection, lookup

Deadlock-free if channel dependence graph is acyclic
- limit turns to eliminate dependences
- add separate channel resources to break dependences
- combination of topology, algorithm, and switch design

Deterministic vs adaptive routing