
Parallel and Distributed Computing

Chapter 4: Communications in Networks

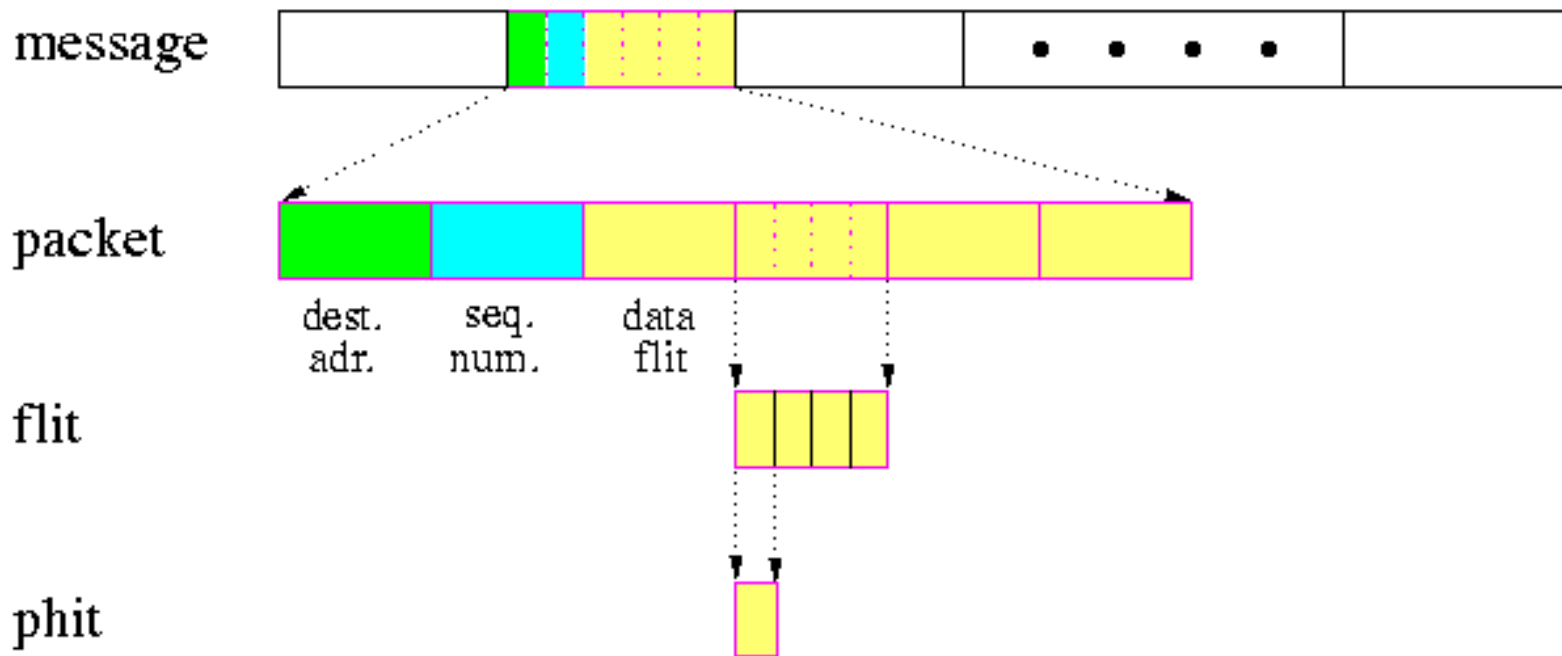
Jun Zhang

Laboratory for High Performance Computing & Computer Simulation
Department of Computer Science
University of Kentucky
Lexington, KY 40506

4.1a: Communication Units

- **Message:** the unit of communication from the programmer's perspective. Its size is limited only by the computer's memory space
- **Packet:** fixed-size small unit of communication containing routing information (e.g., a destination address) and sequencing information in its header. Its size is of order hundred or thousands of bytes or words. It consists of header flits and data flits

4.1b: Illustration of Communication Units



Flit: small unit of information at link layer, of size of a few words
Phit: the smallest physical unit of information at the physical layer, which is transferred across one physical link in one cycle

4.2a: Communication Costs in Parallel Machines

- The cost of network communication is related to:
 - 1.) programming model semantics
 - 2.) network topology
 - 3.) data handling and routing
 - 4.) communication software protocols
- Time for communicating a message between two nodes is the sum of:
 - 1.) time to prepare a message for transmission
 - 2.) time taken by the message to traverse the network to its destination

4.2b: Major Parameters in Communication Costs (I)

- **Startup time** (t_s): time required to handle a message at the sending and receiving nodes
 - 1.) prepare message (adding header, trailer, error correction information)
 - 2.) execute the routing algorithm
 - 3.) establish an interface between the local node and the router
- This latency is only incurred once for a single message transfer

4.2c: Major Parameters in Communication Costs (II)

- **Per-hop Time** (t_h): Time taken by the header of a message to travel between two directly connected nodes in the network
- It is directly related to the latency within the routing switch for determining which output buffer or channel the message should be forwarded to
- The per-hop time is also called node latency

4.2d: Major Parameters in Communication Costs (II)

- **Per-word transfer time** (t_w): Time taken for one word to traverse a link
- This time includes network and buffering overheads
- Per-word transfer time is the reciprocal of the channel bandwidth

4.3a: Message Routing Techniques

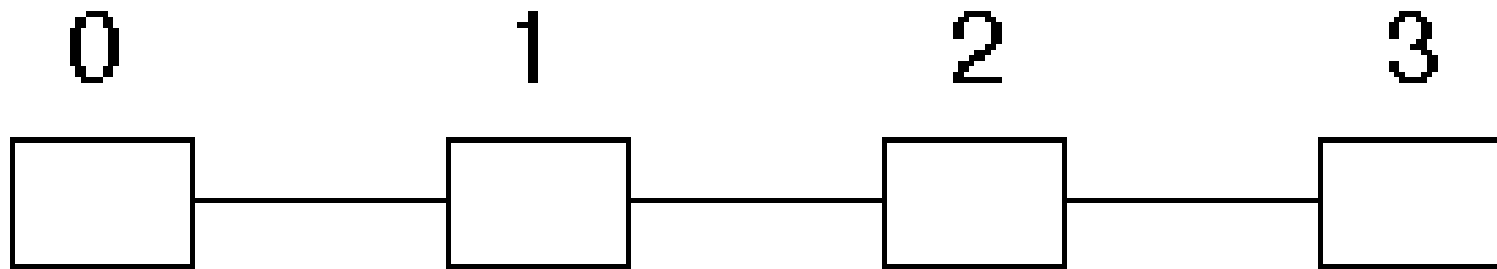
- Store-and-Forward Routing
- Packet Routing
- Cut-Through Routing

4.3b: Store-and-Forward Routing (I)

- When a message traverses a path with multiple links, Each intermediate node on the path forwards the message to the next node after it has received and stored the message
- Total communication cost for a message of size m to traverse a path of l links

$$t_{comm} = t_s + (mt_w + t_h)l$$

4.3c: Illustration of Store-and-Forward Routing



Note that the lack of parallelism in utilizing communication resources

4.3c: Store-and-Forward Routing (II)

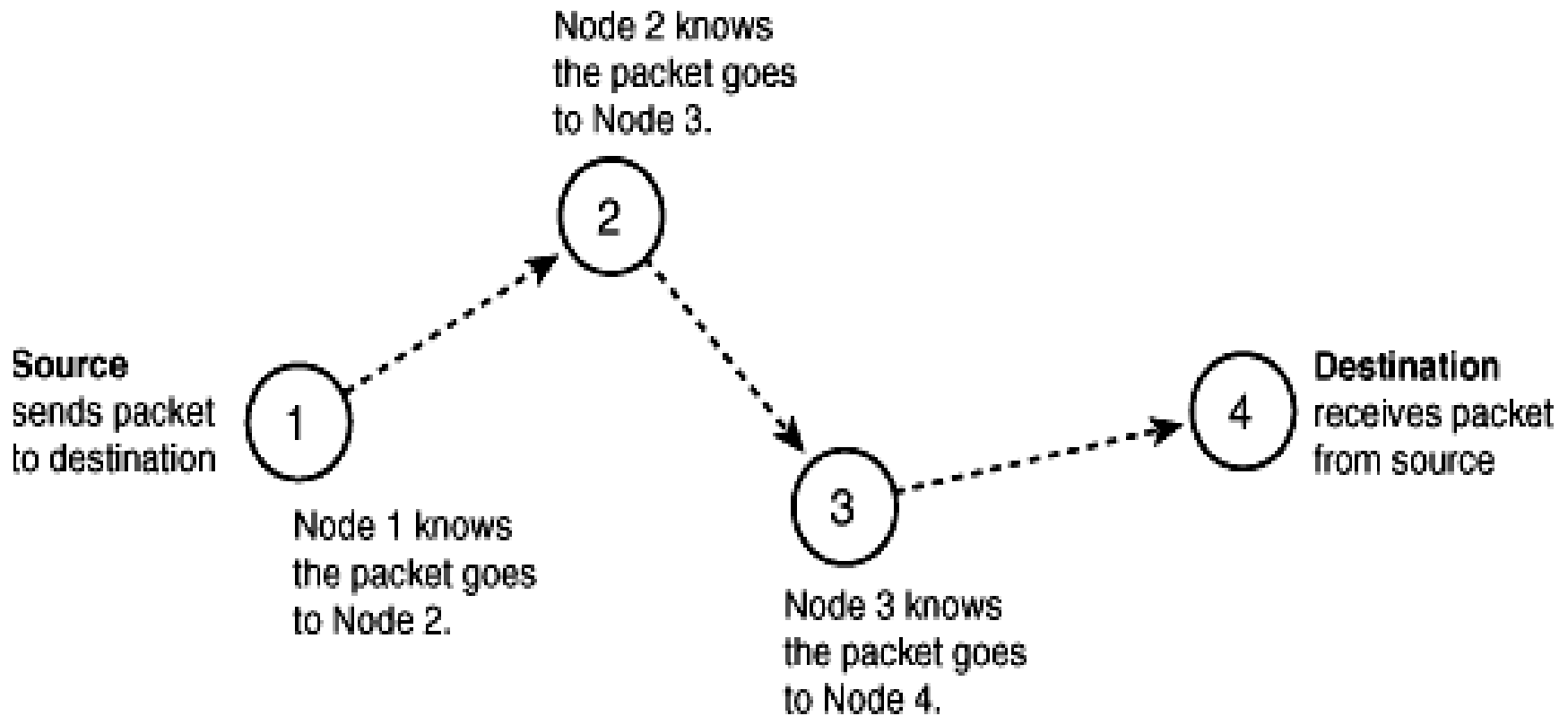
- For present generation parallel computers, t_h is very small
- The quantity mt_w is much larger. So we usually ignore the per-hop time in store-and-forward routing scheme
- The simplified formula is:

$$t_{comm} = t_s + mlt_w$$

4.4a: Packet Routing

- A long message is cut into pieces
- Message pieces are sent through the network one followed by another
- Advantages:
 - 1.) better utilization of communication resources
 - 2.) lower overhead from packet loss (errors)
 - 3.) packets may take different paths
 - 4.) better error correction capability

4.4b: Packet Routing



4.4c: Cost of Packet Routing

$$t_{comm} = t_s + t_{w1}m + t_h l + t_{w2}(r + s) + \left(\frac{m}{r} - 1 \right) t_{w2}(r + s)$$
$$= t_s + t_{w1}m + t_h l + t_{w2}m + t_{w2} \frac{r}{s} m$$

- t_s : start up time
- t_h : per- hop time
- l : number of links
- m : message size
- t_{w1} : package time
- t_{w2} : per- word transfer time
- r : size of a packet
- s : additional information in the message header

4.5a: Cut-Through Routing

- A special packet switching for parallel computers with the following properties to reduce cost:
 1. All packets go through the same path
 2. In-sequence delivery
 3. Associate error information at message level
 4. Use lean error detection mechanisms

4.5b: Flits Communication

- Cut-through routing uses flits
- A tracer is first sent from the source to the destination node to establish a connection
- Flits are sent through the path one after the other
- An immediate node forwards the flit as soon as it is received
- Buffer space is not necessary at the immediate nodes

4.5c: Communication Cost of CT Routing

$$t_{comm} = t_s + lt_h + t_w m$$

- Note that it no longer contains the product of message size and number of links
- Cut-through routing is fast for large size message and long distance communication
- For nearest neighbor communication, store-and-forward routing and cut-through routing are similar

4.6a: Routing Mechanisms for Static Networks

- A routing mechanism determines the path a message takes through the network to get from the source to the destination node
- It may use information about the state of the network
- It returns one or more paths through the network from the source to the destination node
- Two routing mechanisms: **Minimal** and **non-minimal**

4.6b: Minimal and Non-minimal Routing

- **Minimal routing** always selects one of the shortest paths between the source and destination node
- This scheme could lead to congestion in parts of the network
- **Non-minimal routing** may route the message along a longer path to avoid network congestion

4.6c: Deterministic and Adaptive Routing

- A **deterministic routing** scheme determines a unique path for a message, based on its source and destination
does not use network status
- An **adaptive routing** scheme uses information regarding the current status of network to determine the path of message
may detect a congestion spot and route message around it

4.6d: Dimension-Ordered Routing

- It assigns successive channels for traversal by a message based on a numbering scheme determined by the dimension of the channel
- **XY-Routing** for a two-dimensional mesh
- **E-Cube Routing** for a hypercube
- Both are deterministic and minimal routing techniques
- Both routing schemes are dead-lock free

4.6e: XY-Routing Scheme

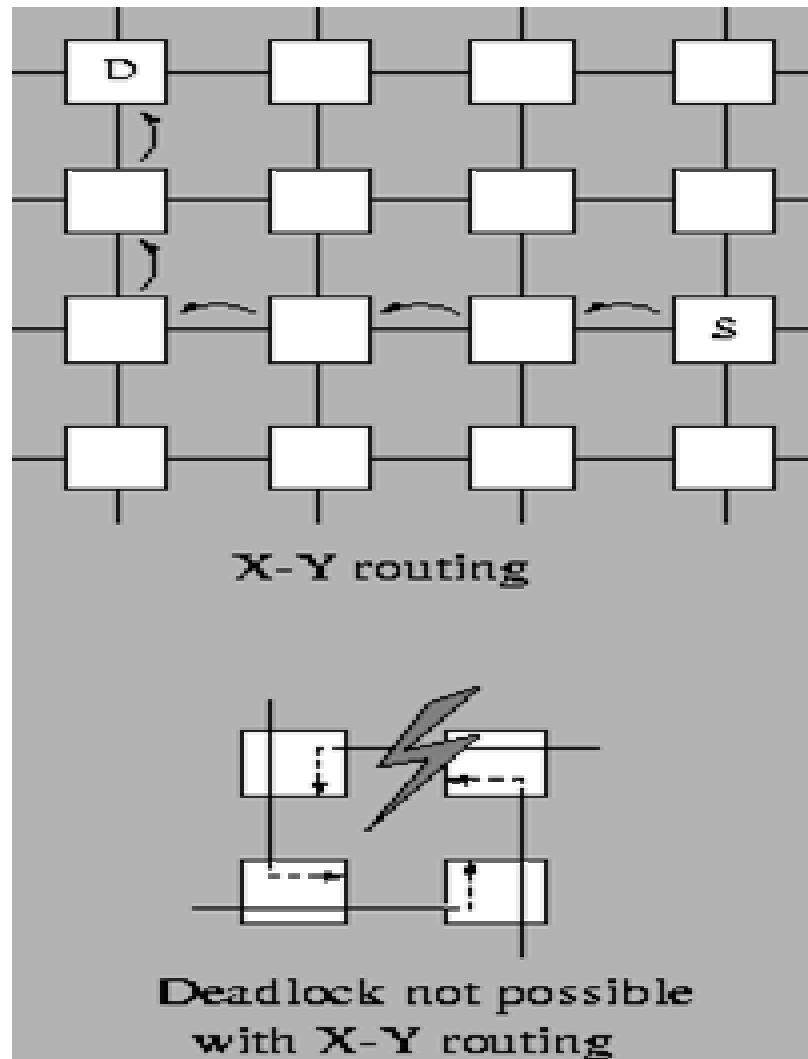
- A message is first sent along the X dimension until it reaches the column of the destination node
- Then it is routed along the Y dimension until it reaches the destination
- The length of the path is

$$| S_x - D_x | + | S_y - D_y |$$

Source node: P_{S_x, S_y}

Destination node: P_{D_x, D_y}

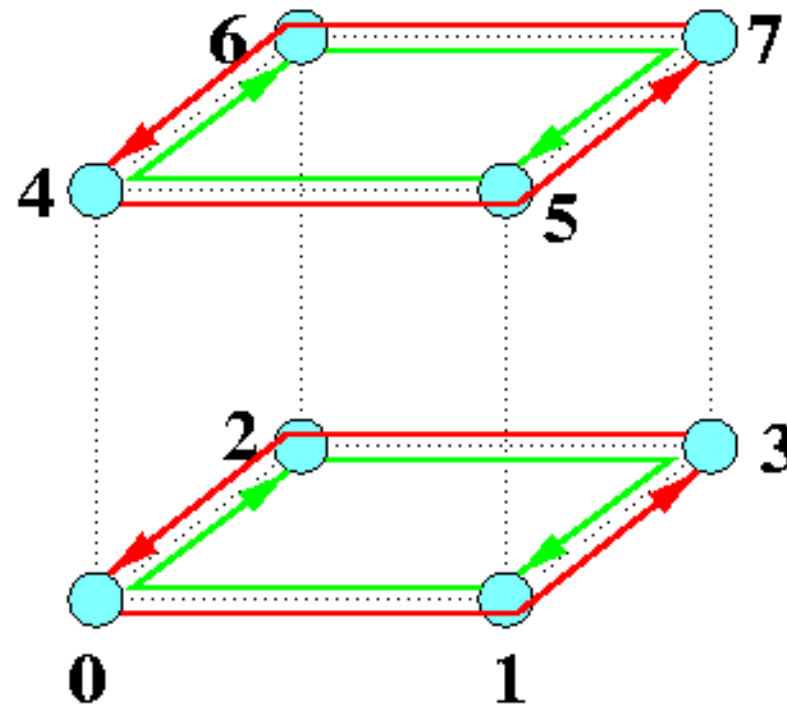
4.6f: Illustration of XY-Routing



4.6g: E-Cube Routing Scheme

- The number of links in the path is determined by the binary labels of the source and destination nodes (Hamming distance)
- Each step corrects one bit difference, starting from the least significant digit (exclusive-or operation)

4.6h: Illustration of E-Cube Routing



4.7a: Network Embedding and Graph Mapping

- Embedding one network into another is important for porting algorithms
- No need to develop algorithms for every network topology
- Process-processor mapping is not controlled by the programmer
- Mapping can be used to determine the degradation in the performance of an algorithm

4.7b: Graph Mapping

- Let $G(V, E)$ and $G'(V', E')$ be graphs
 V, V' are set of vertices
 E, E' are set of edges

- A map $H: G \rightarrow G'$ is an embedding if:

each vertex in V is mapped to a vertex in V'
and each edge in E is mapped to one or more
edges in E'

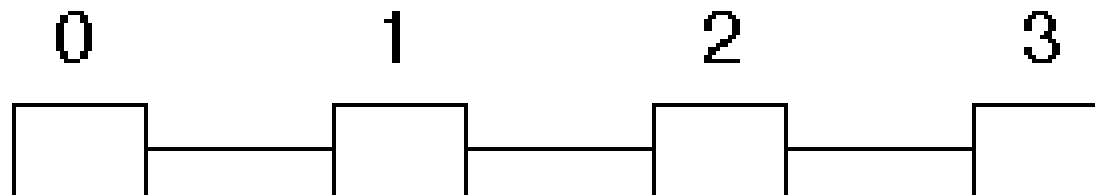
4.7c: Mapping Terminologies

- **Congestion:** Maximum number of edges of E mapped on to any edge in E'
- **Dilation:** Maximum number of edges of E' that any one edge of E is mapped onto
That is how much we stretch an edge of E
- **Expansion:** $|V'| / |V|$
- **Load:** Maximum number of vertices of E that are mapped to a single vertex of E'

4.8a: Embedding a Linear Array onto a Hypercube (I)

- Embed a linear array of 2^d processors into d -dimensional hypercube

1). Label the processors of the linear array in order from 0 to $2^d - 1$



4.8b: Embedding a Linear Array onto a Hypercube (II)

- 2. Map linear array processor K to processor $H(i, d)$ where:

$$H(0, 1) = 0,$$

$$H(1, 1) = 1, \text{ and}$$

$$H(i, x+1) = \begin{cases} H(i, x), & i < 2^x \\ 2^x + H(2^{x+1} - 1 - i, x), & i \geq 2^x \end{cases}$$

The sequence generated by the H 's is a **binary reflected Gray code (RGC)**

4.8c: Gray Code – The Easier Way

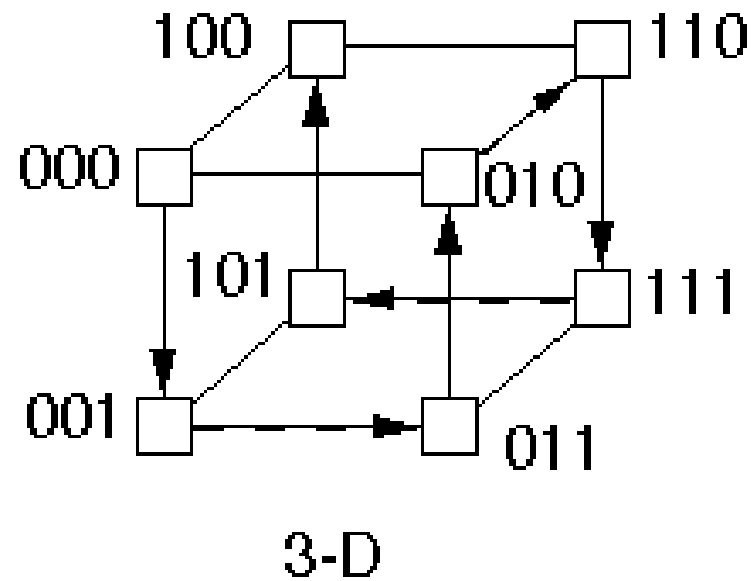
- 1-bit code is 0 then 1
- To get x -bit code:
 - copy $(x-1)$ -bit code, call it A
 - repeat the $(x-1)$ -bit code in reverse order, call it B
 - add a 0 bit in front of the elements of A
 - add a 1 bit in front of the elements of B

4.8d: Binary Reflected Gray Code

1-bit	2-bit	3-bit
0	0 0	0 0 0
1	0 1	0 0 1
	— — —	
	1 1	0 1 1
	1 0	0 1 0
		— — — —
		1 1 0
		1 1 1
		1 0 1
		1 0 0

4.8e: A Sample Mapping

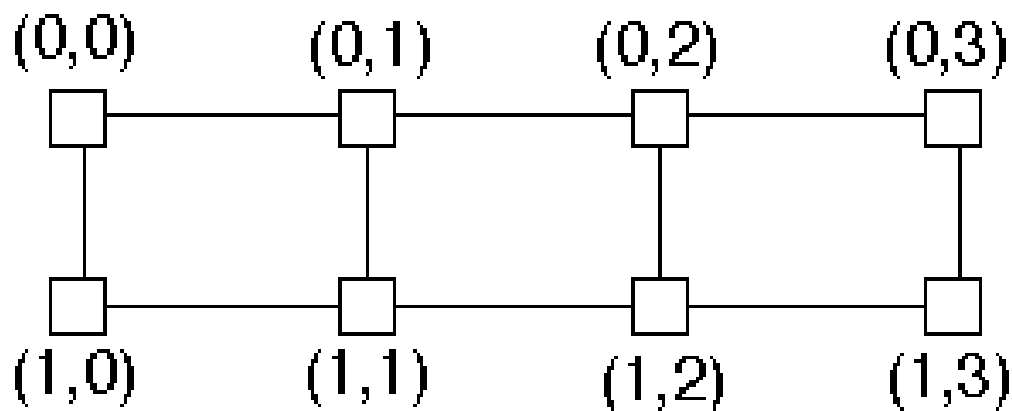
0	0	0	0
1	0	0	1
2	0	1	1
3	0	1	0
	-	-	-
4	1	1	0
5	1	1	1
6	1	0	1
<u>7</u>	1	0	0



4.9a: Embedding a Mesh into Hypercube

- A $2^s \times 2^r$ mesh can be embedded into a $s+r$ dimensional hypercube with dilation and congestion 1

1.) Label the mesh using 2 dimensional coordinates



4.9b: Embedding a Mesh into Hypercube

2.) Produce gray code for each dimension

Number	Gray Code
0	0
1	1
Number	Gray Code
0	00
1	01
2	11
3	10

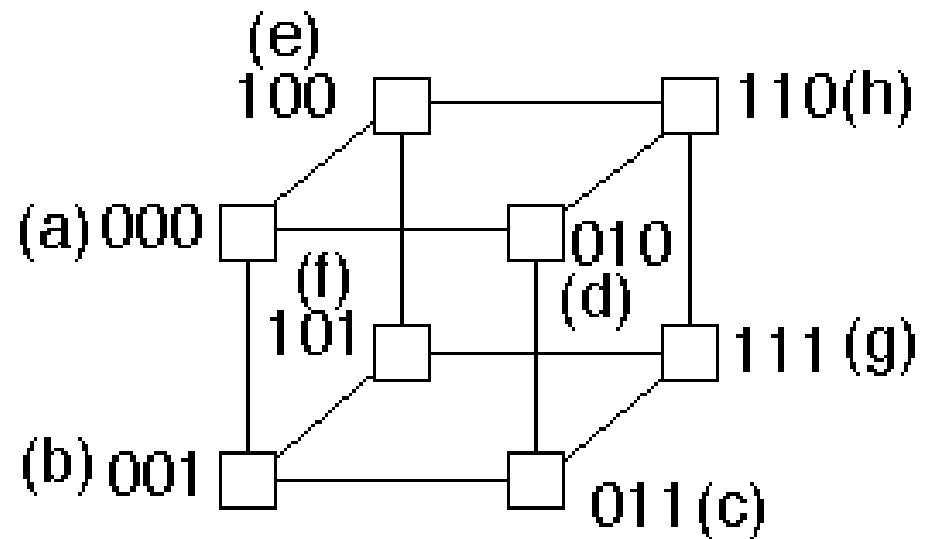
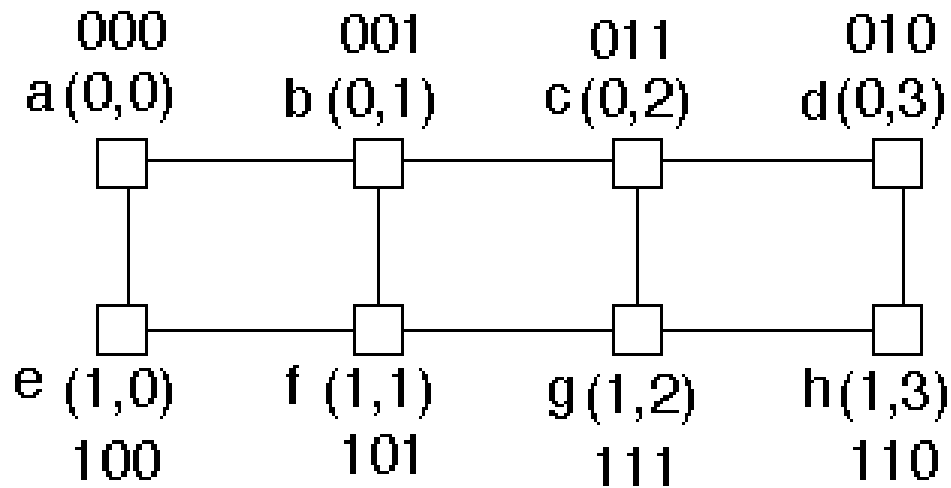
4.9c: Embedding a Mesh into Hypercube

3.) Concatenate gray codes to produce gray code for tuples

4.) Gray code indicates the processor in the Hypercube the mesh processor will be mapped onto

Tuple	Gray Code
(0,0)	0 00
(0,1)	0 01
(0,2)	0 11
(0,3)	0 10
(1,0)	1 00
(1,1)	1 01
(1,2)	1 11
(1,3)	1 10

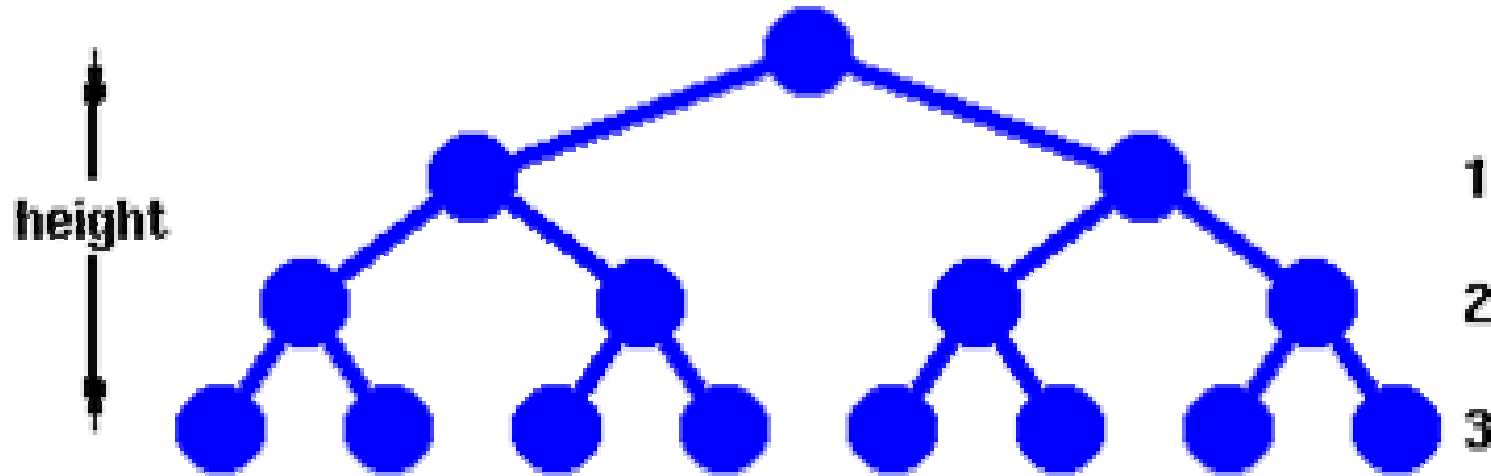
4.9d: Embedding a Mesh into Hypercube



4.10a: Embedding Complete Binary Tree to Hypercube

- A complete binary search tree is a binary search tree such that:
 - 1.) All internal nodes have two children, with one possible exception
 - 2.) All leaves occur on at most two different, but consecutive, levels
 - 3.) If a level contains leaves and internal nodes, the internal nodes must be to the left of all leaves, internal nodes with two children must be to the left of internal nodes with one child

4.10b: A Complete Binary Tree

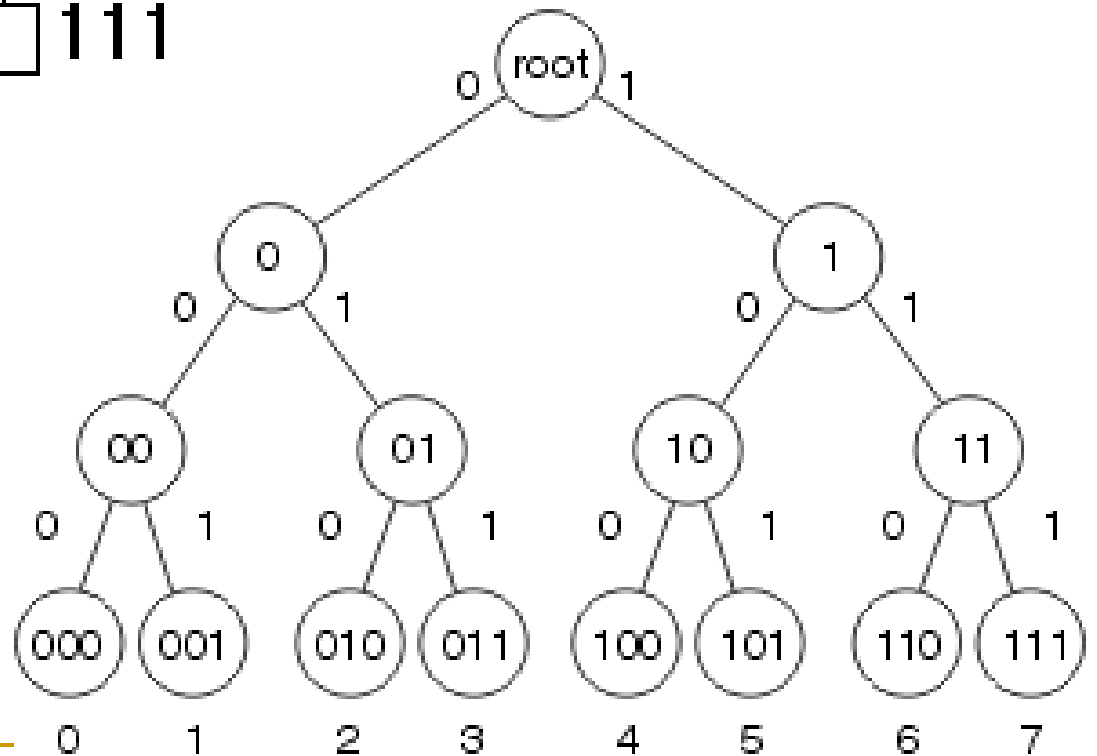
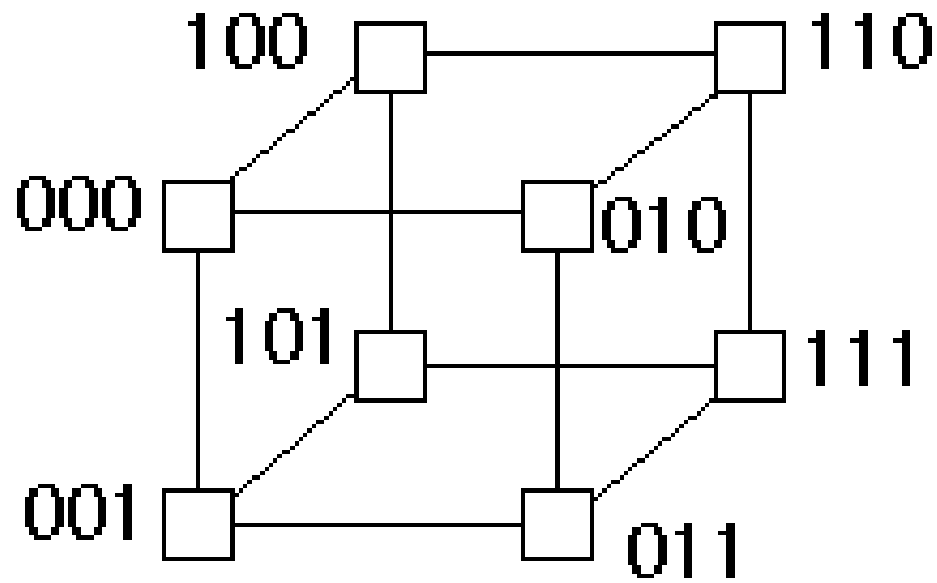


Only map the leaf nodes to a hypercube, assuming internal nodes are switches

4.10c: Mapping Algorithm

- Map any node to the root
- Map the same node to the left child, map the node with the least significant digit reversed to the right child
- For each node in the 2nd level, map the same node to its left child, map the node with the next least significant digit reversed to the right child
- Continue this way until reach the leaf level
- (The height of the binary tree equals the number of digits of the binary labels.)

4.10d: Map Binary Tree to Hypercube



4.11a: Cost-Performance Tradeoffs

- Fattened lower dimensional networks can have similar performance as higher dimensional networks
- Less complicated layouts, wire crossings, and variable wire-lengths
- Node distance effect can be minimized by using appropriate routing techniques
- Which network provides better cost/performance tradeoffs?

4.11b: Fattened Mesh and Hypercube

Fattened mesh with $(\log p)/4$ wires per channel

$$T_{mesh} = t_s + t_h \sqrt{p} / 2 + 4t_w m / (\log p)$$

Hypercube with one wire per channel

$$T_{cube} = t_s + t_h (\log p) / 2 + t_w m$$

Cut-through routing

Mesh with wraparound

4.11c: Which One is Better?

- For a fixed number of processors p with light load
- For large size message, the t_w term dominates
- When $p > 16$

$$(4t_w m / (\log p)) < (t_w m)$$

Fattened mesh with wraparound is faster than hypercube with the same number of wires