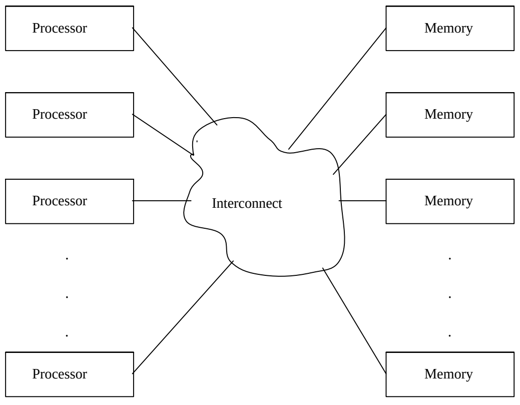
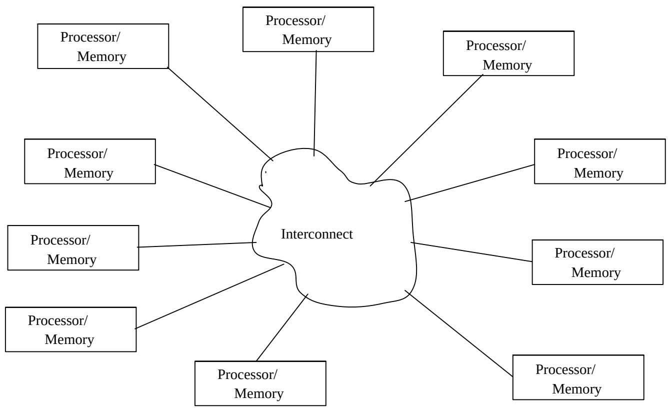
CSE 4351 Notes 5: Interconnection Networks and Routing

Two Major Approaches to Processor/Memory Interconnects:



Indirect (Dynamic)



Direct (Static)

Graph-theoretic Properties:

Bisection width = minimum number of "wires" that must be removed to get two "halves"

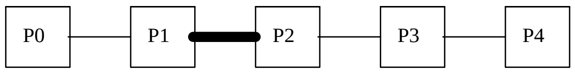
Diameter = maximum distance between any pair of processors

Bounded degree = all networks in the class have vertex-degree bounded by a constant

Symmetry = number of automorphisms, number of vertex equivalence classes (vecs)

Rules of Thumb: High bisection width and low diameter are good. Bounded degree networks are easier to build.
High symmetry - simpler code, complicated to build

Linear array



Bisection width = 1

Diameter = N - 1

Automorphisms = 2, vecs = ceil(N/2)

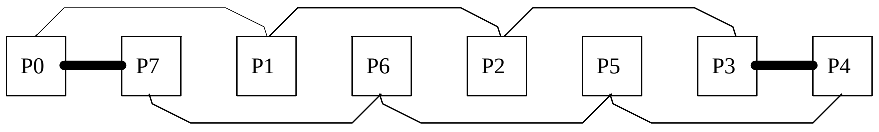
Convenient for VLSI signal processing and bit-level integer multiplication. The convolution of two polynomials h(x) = f(x)g(x) of f(x) = $a_0 + a_1x + a_2x^2 + \ldots + a_{n-1}x^{n-1}$ and g(x) = $b_0 + b_1x + b_2x^2 + \ldots + b_{n-1}x^{n-1}$ may be computed on a 2n-1 node linear array by inputting $a_{n-1}, a_{n-2}, a_{n-3} \ldots a_0$ into the left end at the *odd* numbered steps and b0, b1, b2 . . . b_{n-1} at the *odd* numbered steps. When an a and a b value arrive at a node, the values are multiplied and added to the value which will be output as a coefficient for h(x)

Trace for n = 4:

```
                                p0  p1  p2  p3  p4  p5  p6
--------------------------------------------------------------------------------------
    a0  •   a1  •   a2  •   a3
0                                                           b0  •   b1  •   b2  •   b3
--------------------------------------------------------------------------------------
        a0  •   a1  •   a2  •   a3
1                                                       b0  •   b1  •   b2  •   b3
--------------------------------------------------------------------------------------
            a0  •   a1  •   a2  •   a3
2                                                   b0  •   b1  •   b2  •   b3
--------------------------------------------------------------------------------------
                a0  •   a1  •   a2  •   a3
3                                               b0  •   b1  •   b2  •   b3
--------------------------------------------------------------------------------------
                    a0  •   a1  •   a2  •   a3
4                                           b0  •   b1  •   b2  •   b3
--------------------------------------------------------------------------------------
                        a0  •   a1  •   a2  •   a3
5                                       b0  •   b1  •   b2  •   b3
--------------------------------------------------------------------------------------
                            a0  •   a1  •   a2  •   a3
6                                   b0  •   b1  •   b2  •   b3
--------------------------------------------------------------------------------------
                                a0  •   a1  •   a2  •   a3
7                               b0  •   b1  •   b2  •   b3
--------------------------------------------------------------------------------------
                                    a0  •   a1  •   a2  •   a3
8                           b0  •   b1  •   b2  •   b3
--------------------------------------------------------------------------------------
                                        a0  •   a1  •   a2  •   a3
9                       b0  •   b1  •   b2  •   b3
--------------------------------------------------------------------------------------
                                            a0  •   a1  •   a2  •   a3
10                  b0  •   b1  •   b2  •   b3
```

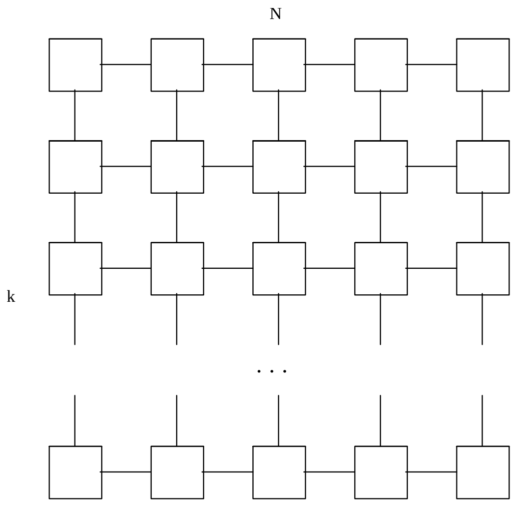
INCREDIBLY EASY TO PERFORM ROUTING!!!

Ring - symmetric version of linear array. To avoid long wraparound connection, the following arrangement may be used:



Bisection width = 2, Diameter = N/2, Automorphisms = 2N, vecs = 1

Mesh



2-d most common, can have higher dimension

Bisection width = min(k,N) if max(k, N) is even and min(k, N) + 1, otherwise. Diameter = k + N - 2

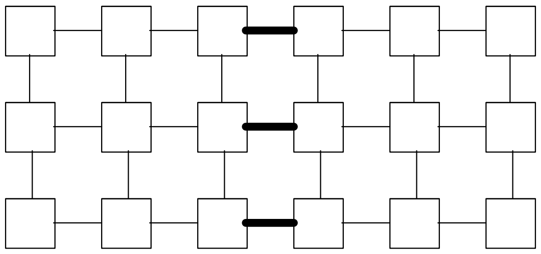
Bisection width for higher dimension array:

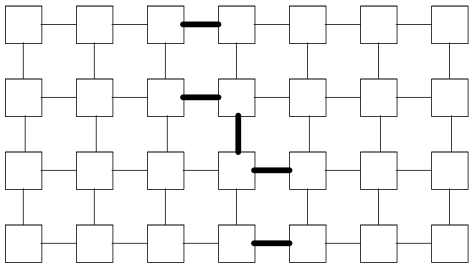
Suppose dimensions are $2 \le N_1 \le N_2 \le . . . \le N_r$.

If N_r is even, then bisection width is $N_1 \bullet N_2 \bullet . . . \bullet N_{r-1}$

If N_r is odd, then bisection width is $N_1 \bullet N_2 \bullet . . . \bullet N_{r-1}$ + bisection width of array $N_1 \times N_2 \times . . . \times N_{r-1}$

Automorphisms = 8 and vecs ≈ $N^2/8$ when N = k



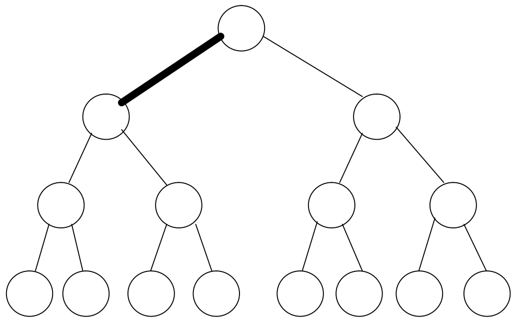


ROUTING IS MORE DIFFICULT THAN LINEAR ARRAY. BOTTLENECKS AROUND CENTER OF MESH.

Sorting can be used to handle routing (along with "perfect matching" in graphs). Can also use randomized approaches such as Valiant-Brebner routing.

2-d Torus - adds wrap-around connections between P(i,N-1) and P(i,0), also P(k-1,j) and P(0,j).
N-by-N torus (N > 4) has bisection width = if N even then 2N else 2(N+1),
diameter = if N even then N else N-1, $8N^2$ automorphisms, 1 vec

Complete Binary Tree



Bisection Width = 1 Diameter = 2*lg(N+1) - 2 Automorphisms = $2^{2^{h}-1}$, vecs = h + 1 (h=height, which is 3 in example)

Often the internal nodes are used just for communication

Leaf selection - number leaves from 0 to 2^k - 1, can switch path to leaf by sending MSB first followed by decreasing significance bits.

Butterfly

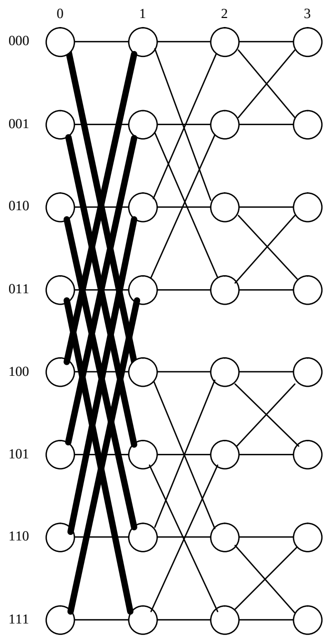
2^k rows and k+1 columns

Processors in each row are connected as a linear array

Bisection width = 2^k (just remove highest dimension edges)

Automorphisms = $2^{2^{k+1}-1}$ vecs = (k + 1)/2

Processor P[i,j] is also connected to processor in next column via complementing the (j+1)-st most significant bit



Fat tree - generalization of the butterfly that has been used in several commercial systems.

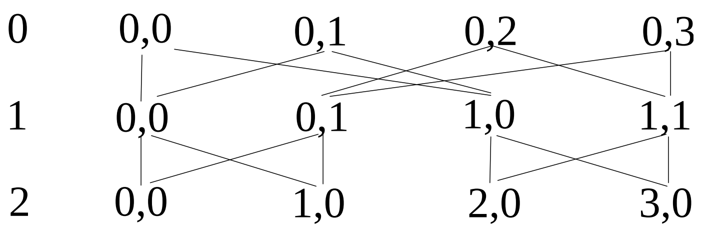
Each non-root node has two parents, but each non-leaf node has *degree* children.

The *depth* of the fat tree indicates the number of levels past the root level (level 0).

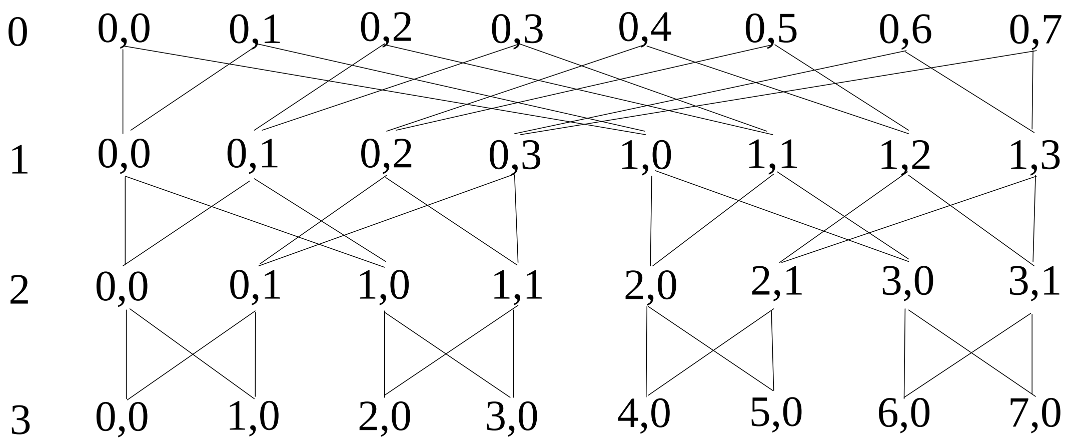
Level i ($0 \le i \le$ depth) has vertices labeled (i, j, k) where $0 \le j <$ degree^i and $0 \le k < 2^{\text{depth-i}}$.

Non-leaf vertex (i, j, k), i < depth has children (i + 1, j•degree + p, k/2) where 0 ≤ p < degree.

Example: depth = 3 and degree = 2:



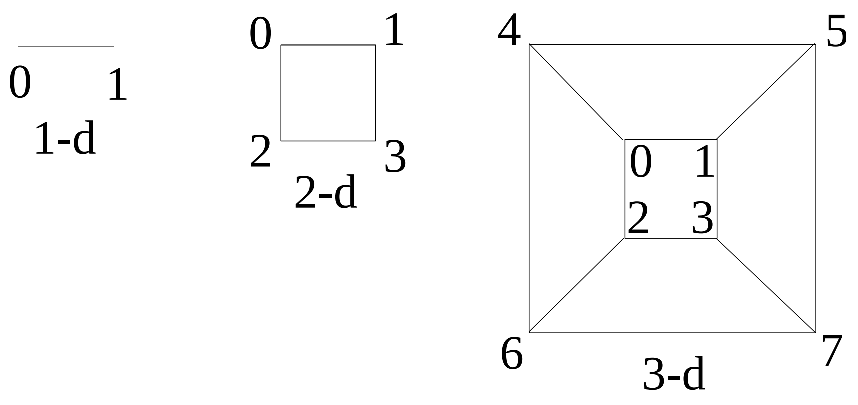
Example: depth = 4 and degree = 2:

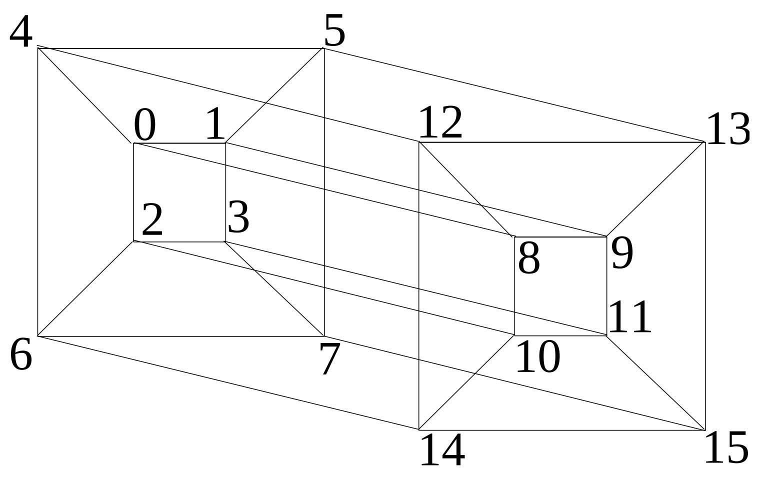


Hypercube

Squashes together each row of butterfly, giving 2^k processors

Each processor is connected to k others - by complementing exactly one bit





4-d

Note: Cube connection results from replacing each vertex by a ring of k vertices

Bisection width = $2^{k - 1}$ Automorphisms = $k!2^k$ vecs = 1

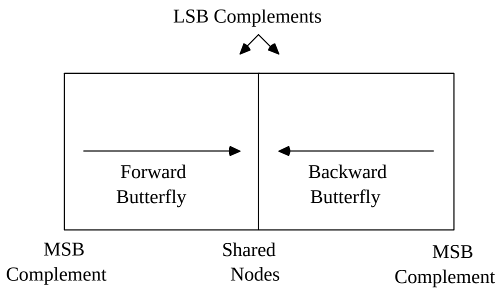
Derivation of automorphisms:

1. Any one of the 2^k vertices may be mapped to vertex 0.
2. Any dimension is equivalent to any other, since there are k dimensions there are k! permutations.

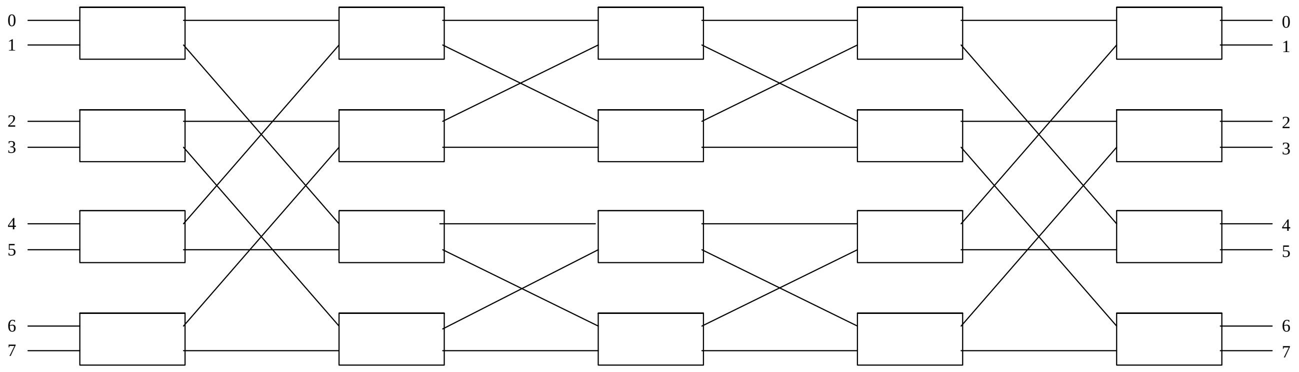
Hamiltonian cycle of processors - use reflected Gray code (address = i xor (i >>1) trick)

1-bit:	2-bits:	3-bits:	4-bits:
0	**0** 0	**0** 0 0	**0** 0 0 0
1	**0** 1	**0** 0 1	**0** 0 0 1
	----	**0** 1 1	**0** 0 1 1
	1 1	**0** 1 0	**0** 0 1 0
	1 0	--------	**0** 1 1 0
		1 1 0	**0** 1 1 1
		1 1 1	**0** 1 0 1
		1 0 1	**0** 1 0 0
		1 0 0	----------
			1 1 0 0
			1 1 0 1
			1 1 1 1
			1 1 1 0
			1 0 1 0
			1 0 1 1
			1 0 0 1
			1 0 0 0

Benes Network: A multi-stage switching network variation of the butterfly/hypercube with $(2k+1)2^k$ binary switches. We look at this network to get an initial understanding of static (permutation) routing and then examine the same concepts for hypercubes.

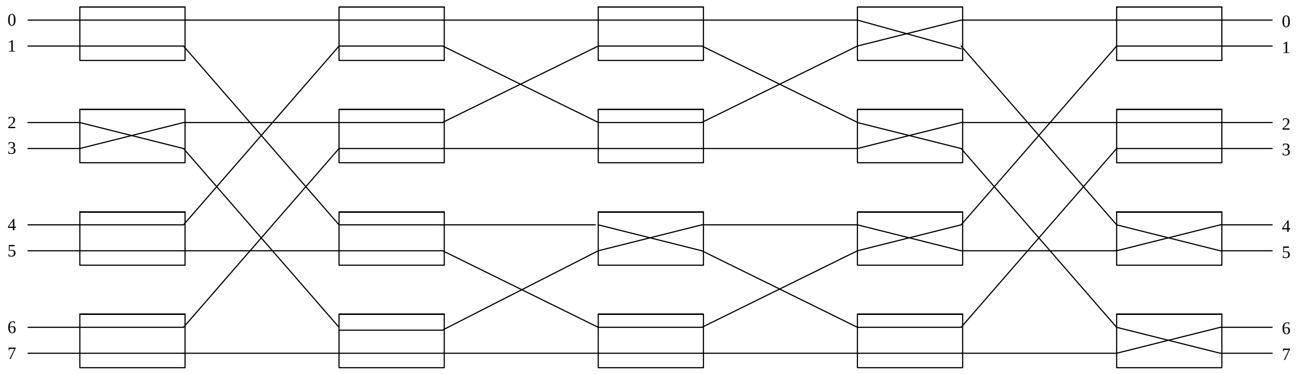


Commonly written as

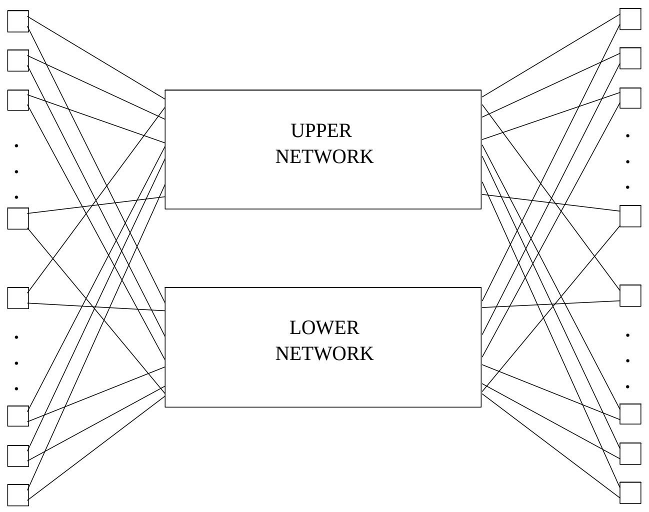


Switches can be set to give an "edge-disjoint" path for <u>any</u> permutation. For permutation

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 5 & 3 & 4 & 7 & 0 & 1 & 2 & 6 \end{pmatrix}$$ we get:



The existence of a switch setting for each permutation can be proven by viewing the Benes network as a recursive structure:



If we can successfully route a n-permutation problem to the upper and lower networks, we then have two n/2-permutation problems. 2-permutations are trivially routed. n-permutation problems are always solvable based on Hall's Matching Theorem:

A 2N-node bipartite graph G=(U,V,E) has a perfect matching if and only if for all subsets $S \subseteq U$, $|N(S)| \geq |S|$, where N(S) denotes the nodes in V that are adjacent to a node in S.

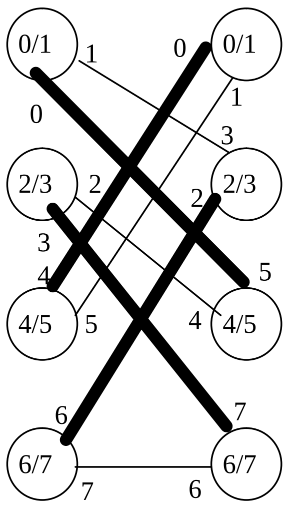
Corollary: If all vertices in a bipartite graph are incident on k edges, then there are k disjoint perfect matchings. (The set of k matchings may not be unique. The number of perfect matchings is known as the <u>permanent</u> of the binary adjacency matrix for the graph.)

Translation:

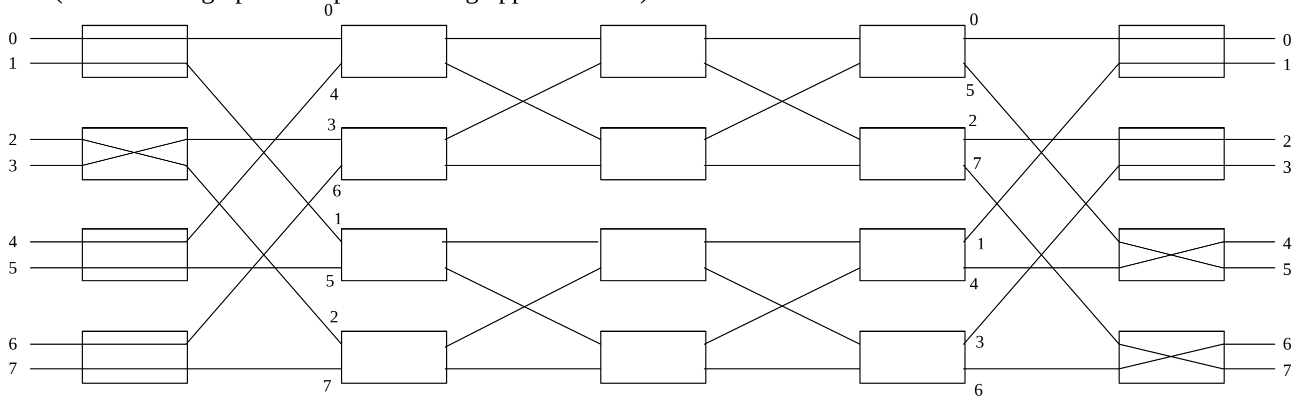
bipartite: two-colorable
U: Nodes of color 1
V: Nodes of color 2
Perfect matching: Set of edges that will include all vertices, but no vertex is on two of the edges
Condition will be satisfied by routing graph, since each node is incident to two edges.

All "outside" switches have two packets to be routed to the other side.

Example: $\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 5 & 3 & 4 & 7 & 0 & 1 & 2 & 6 \end{pmatrix}$

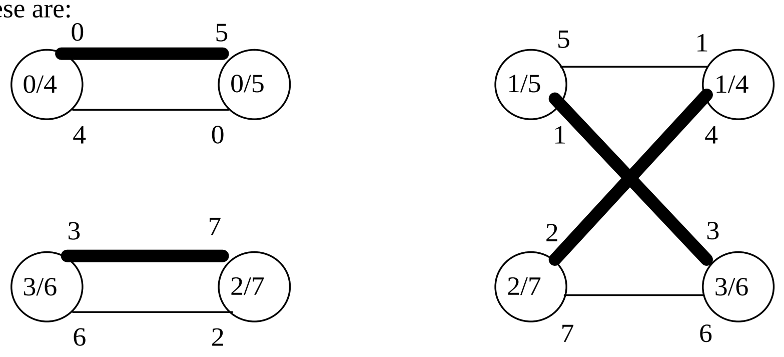


This gives (dark lines in graph correspond to using upper network)

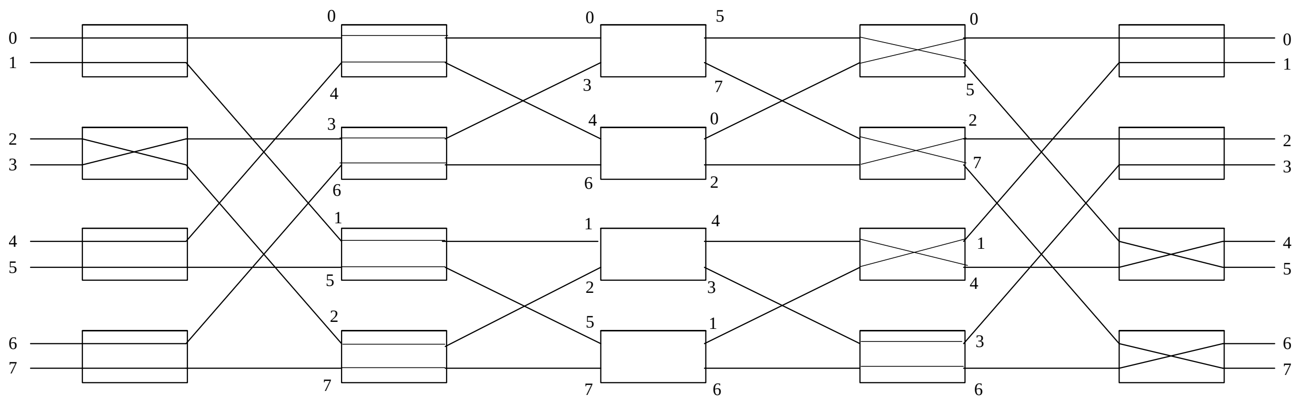


and an upper routing problem of $\begin{pmatrix} 0 & 4 & 3 & 6 \\ 5 & 0 & 7 & 2 \end{pmatrix}$ and a lower routing problem of $\begin{pmatrix} 1 & 5 & 2 & 7 \\ 3 & 1 & 4 & 6 \end{pmatrix}$

The matching problems for these are:



The recursive structure views these two problems as separate, but there is no difficulty in solving them as a single bipartite matching problem. We now have:



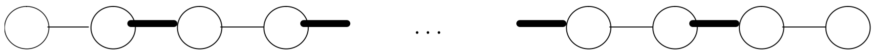
The remaining four subnetworks (single switches) are trivial.

FINDING A MAXIMUM BIPARTITE MATCHING

Even though trial-and-error is usually sufficient for small problems, the notion of an alternating path facilitates the task. Hopcroft and Karp developed an extremely efficient depth-first-search algorithm of this concept that runs in O(|E| √(|V|)) time, but it is not suitable for "hand tracing". (Even more remarkably, Micali and Vazirani obtained the same bound for matching in general graphs).

The algorithm is based on incrementally increasing the size of the matching. The algorithm starts by using an initial deficient matching (a single edge is fine or we may greedily attempt to insert each edge in the matching without backtracking by removing a vertex). We then search for a path with the following properties:

1. The starting and terminating vertices are different and are not included in the previous matching.
2. The path alternates between k+1 edges that are not in the matching and k edges that are in the matching, i.e.



—— = Edge not in previous matching

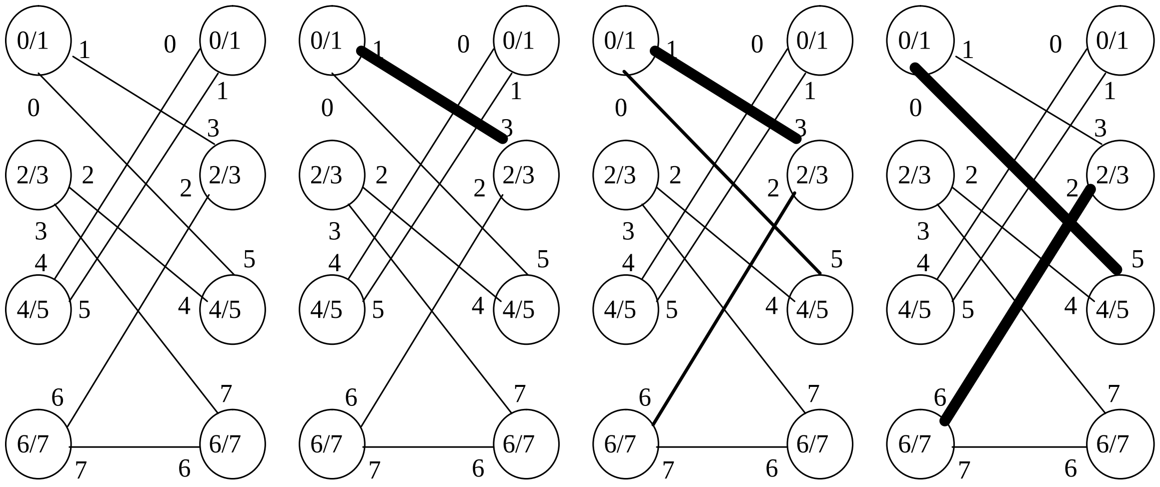
▬ = Edge included in previous matching

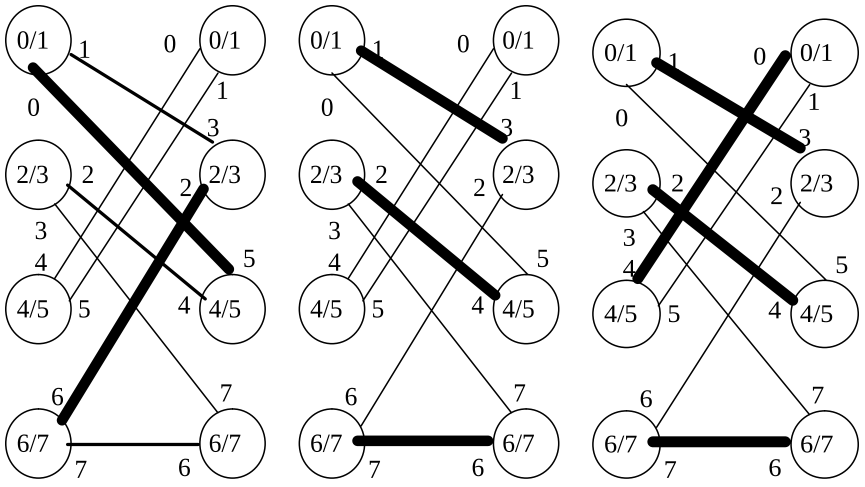
3. The new matching is obtained by removing the edges from the previous matching that are on the alternating path and then including the alternating path edges that were not in the previous matching.

NOTE: Often a simple alternating path is just a single edge between two vertices not in the previous matching!

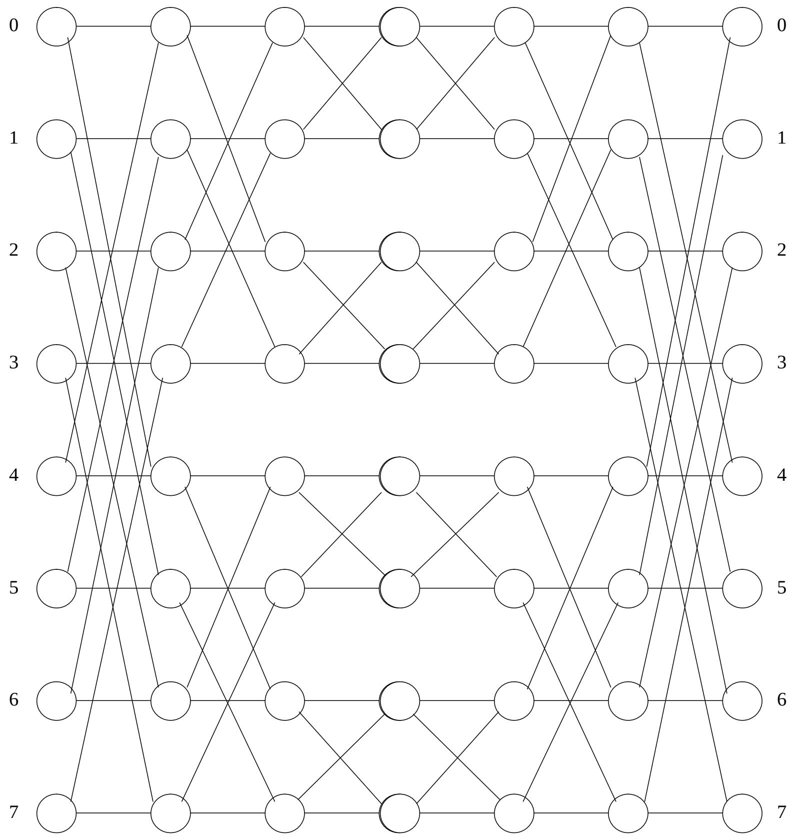
Sequential code and sample input files are in files bipartiteMatch*.

Example:

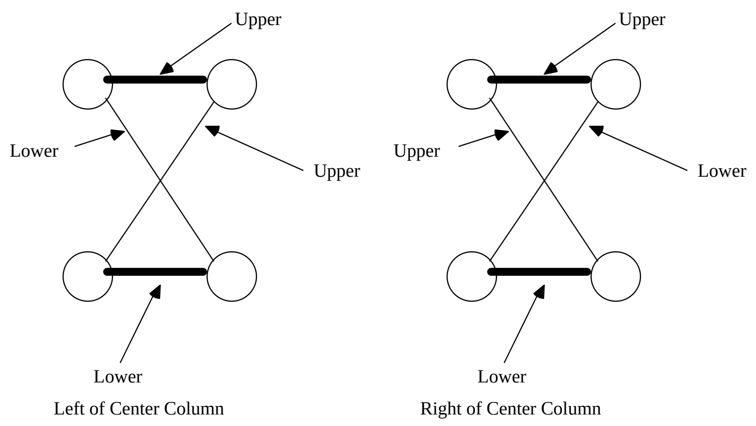




The same ideas apply to a butterfly/hypercube, but will go both ways on an edge at different times:

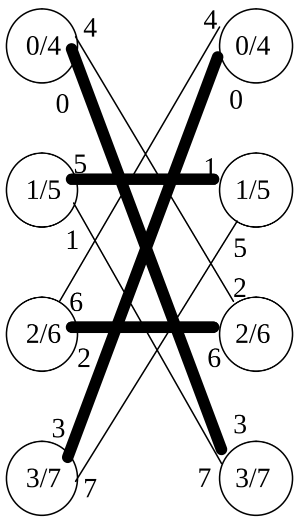


Pairs of processors act as switches in each layer. First (leftmost) level: (0,4),(1,5),(2,6),(3,7) Second layer: (0,2),(1,3),(4,6),(5,7)
Interpretation:



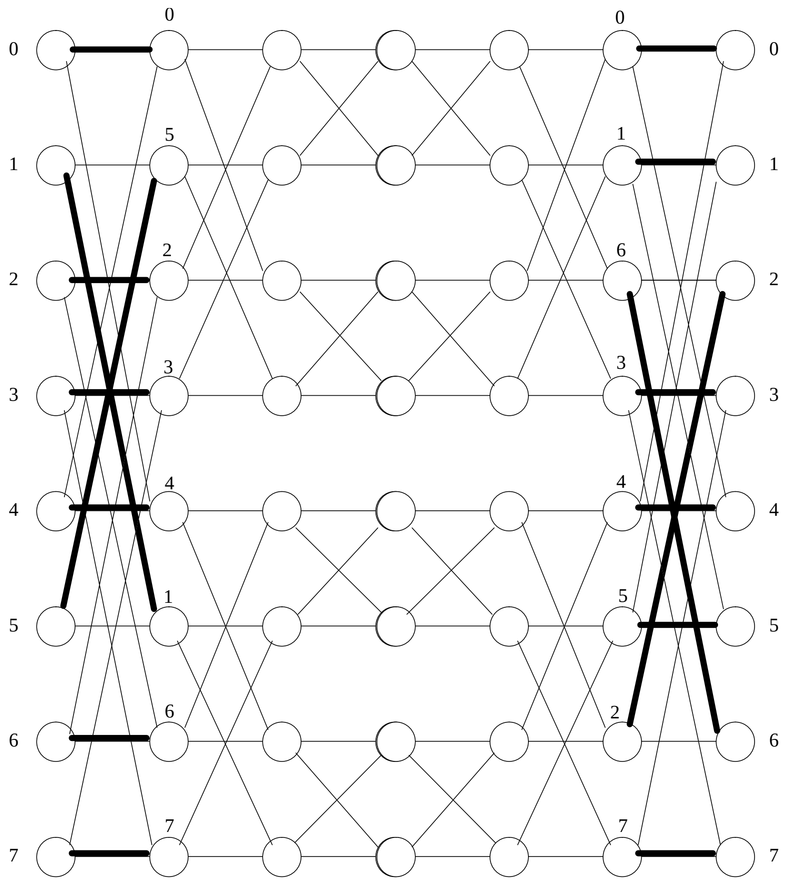
Permutation routing is again based on perfect matching. In each pair, one processor takes upper, the other takes lower. Gives a vertex-disjoint path.

Example: Route $\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 3 & 7 & 6 & 0 & 2 & 1 & 4 & 5 \end{pmatrix}$



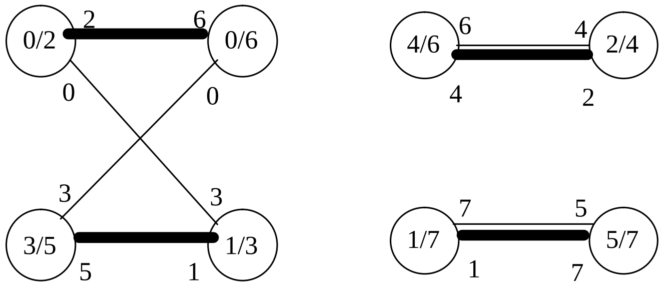
Routing Graph:

Paths:

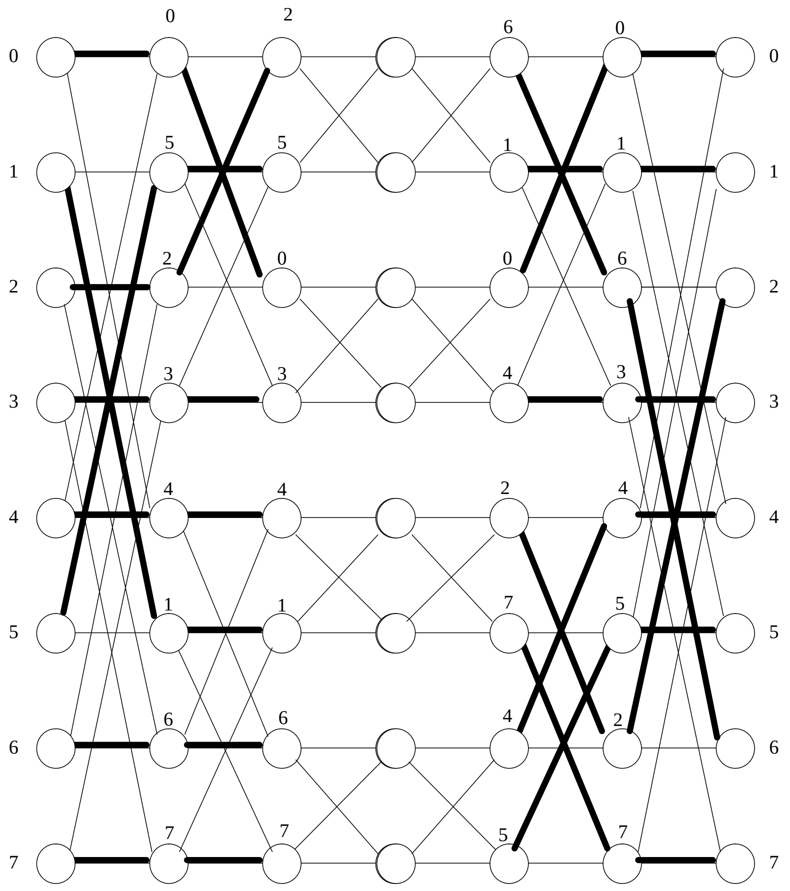


Upper routing problem is: $\begin{pmatrix} 0 & 5 & 2 & 3 \\ 3 & 1 & 6 & 0 \end{pmatrix}$. Lower routing problem is: $\begin{pmatrix} 4 & 1 & 6 & 7 \\ 2 & 7 & 4 & 5 \end{pmatrix}$.

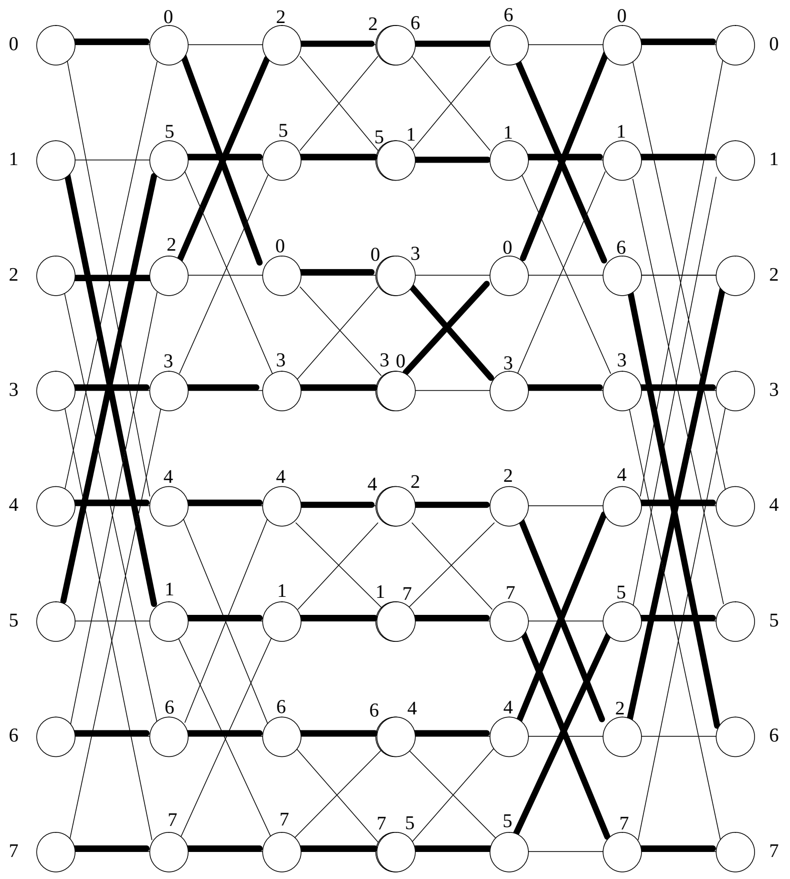
Routing graphs:



Corresponding paths:



Final paths are trivial:



Static Routing for Meshes

Given an arbitrary permutation for packets on a 2-d mesh, perfect matching (as in hypercubic networks) may be used to achieve a static routing. The algorithm will give a 3N - 3 step strategy for routing a given permutation on a N x N mesh. The result can be generalized to multidimensional meshes. The algorithm has three phases, only the first phase must be precomputed:

Phase 1: Permute the packets within each column so that at most one packet in each row is destined for each column via perfect matching

Phase 2: Route each packet within its row to the correct column

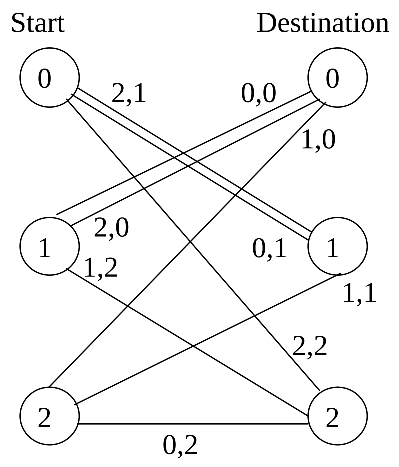
Phase 3: Route each packet within its column to its final destination

Example:

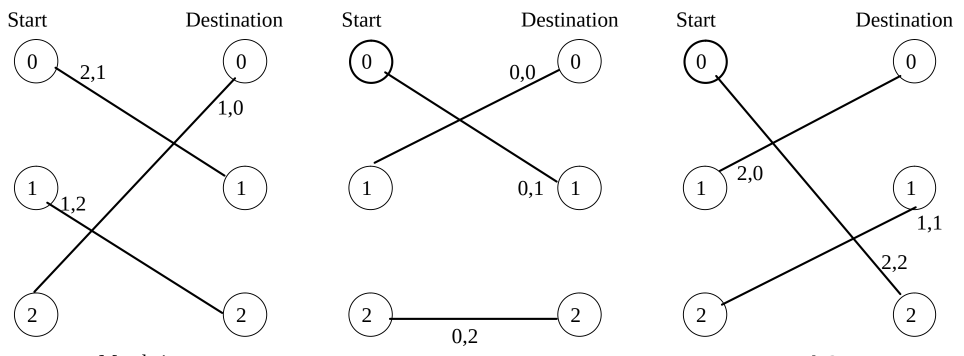
Packets are initially located as follows, with destination indicated:

	0	1	2
0	2,1	1,2	0,2
1	2,2	0,0	1,0
2	0,1	2,0	1,1

The routing graph has one edge per packet, based on the starting column and the destination column for each:



Three perfect matchings are then derived. All packets in the ith matching are routed to row i in the first phase.



Match 1 Match 2 Match 3

Thus, phase 1 will give:

	0	1	2
0	2,1	1,2	1,0
1	0,1	0,0	0,2
2	2,2	2,0	1,1

Phase 2 routes within rows according to column destinations:

	0	1	2
0	1,0	2,1	1,2
1	0,0	0,1	0,2
2	2,0	1,1	2,2

Phase 3 routes within columns according to row destinations:

	0	1	2
0	0,0	0,1	0,2
1	1,0	1,1	1,2
2	2,0	2,1	2,2

Greedy ("Shortest Path") Routing

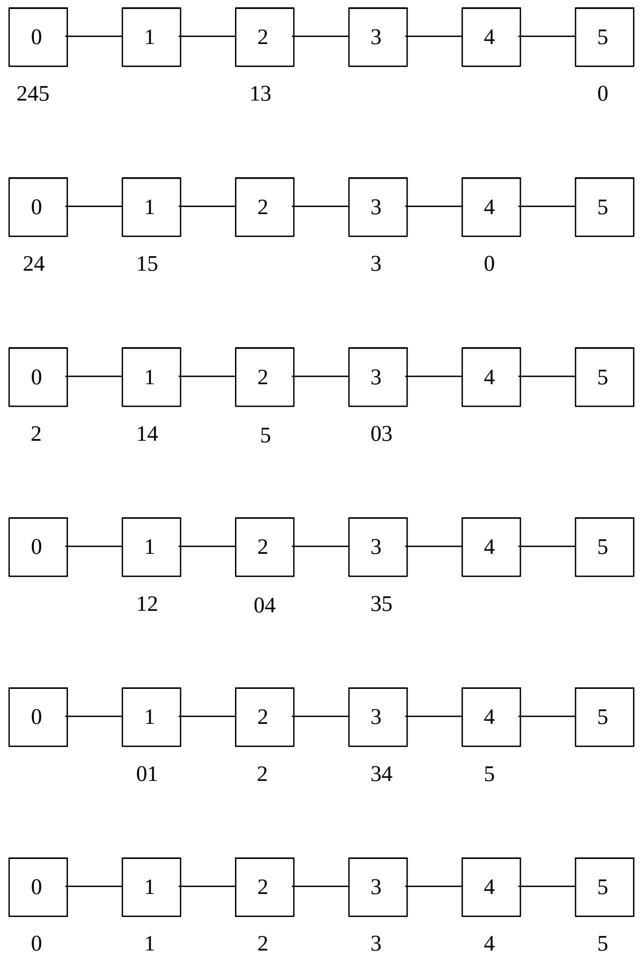
Routing decisions are made on-the-fly in a simple way

Linear Array - Completely solved by greedy approach

Special case - each processor is the destination for a single packet, but a processor may be the source for multiple packets

Inverse problem - each processor is the source for a single packet, but a processor may be the destination for multiple packets

Modification - always choose the packet that has the farthest to go, still takes no more than n - 1 steps



Mesh (n-by-n)

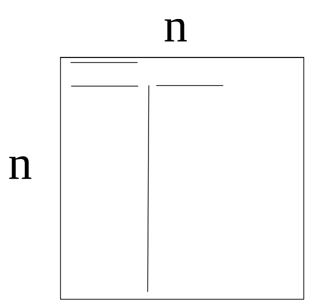
Algorithm: Linear routing in x-dimension, followed by linear routing in y-dimension:

Queueing: For a given edge, choose the packet which must go the farthest in that dimension

Routing for y-dimension takes n - 1 steps using result from linear arrays.

The linear array result also applies when each node starts with one packet, but a node may receive multiple packets - like in the x-dimension

Takes 2n - 2 steps, but may queue almost 2n/3 packets, consider the situation:



Each of the three portions of the first two rows has about n/3 elements destined for column n/3. At mesh node (2,n/3), for each element that leaves (out the bottom) two additional elements will be queued

Hypercube

Algorithm: Processor scans low-order to high-order comparing processor address and the destination address until the a mismatch is found, then send out that edge

Example: Suppose processors with addresses of form 0...0, <x> send to processors with addresses of form <y>, 0...0 (0...0, <x>, and <y> each have k bits).

PROBLEM: All √N packets must traverse processor 0...0,0...0

General solution to congestion: Valiant-Brebner Routing

1. Packet is greedily routed to a random intermediate destination.

2. Packet is greedily routed from intermediate destination to the real destination.

Switching Techniques

Circuit switching: Physical switches are set to reserve entire path for full bandwidth.

Store-and-forward (packet switching): A packet of a message is forwarded to next processor on path only after the entire packet has been received. Path is not established up front.

Virtual cut-through: If next router along path has input buffer space, then the flits in a packet will be pipelined. Otherwise, there is sufficient space to store an entire packet.

Wormhole routing: Uses pipelining, but has smaller buffers and uses flow-control to stall the flits of a packet, possibly among several routers. Prone to deadlock, so physical channels are divided into virtual channels that allow sufficient progress as long as cycle(s) of virtual channels are avoided.

Broadcasting Models:

One-to-all (ordinary broadcast) /All-to-all (‘‘gossiping’’) / Personalized (individualized messages)

Single-port/Multi-port - degree of communication concurrency for a processor

Mono-directional/Bi-directional - meaning of an edge

Example: One-to-all broadcast on k-dimensional hypercube with node 0 as root.

Diameter is lower bound on number of rounds.

```
rootWork=0;
for (receiveDim=0; receiveDim < k; receiveDim++)
{
    if bit receiveDim of rank == 1
        Set bit receiveDim of rootWork
    if rootWork==rank
        break;
}
for (i=0; i < k; i++)
    if i ≥ receiveDim
        if bit i of processorRank is 0
            Send data value to node rank + $2^i$
        else
            Receive data value from node rank - $2^i$
```

Easily adapted for arbitrary processor to be the root.

Example: All-to-all broadcast on k-dimensional hypercube using all links simultaneously

Lower bounds:

1. Diameter of hypercube = k

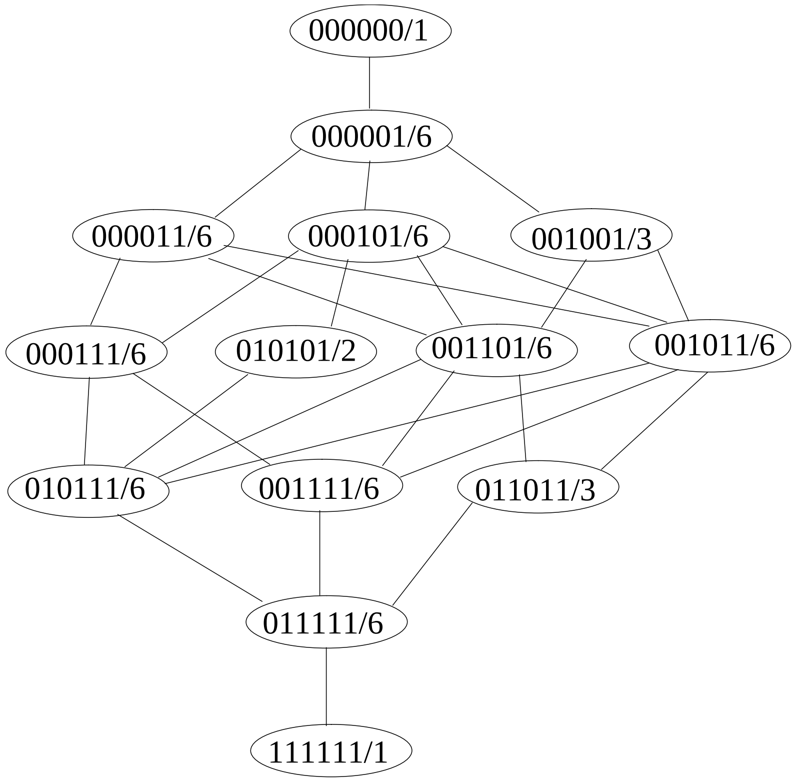
2. In a given round, a processor may receive up to k messages. Each processor needs to receive 2^k - 1 messages, so at least ceiling((2^k - 1)/k)rounds are needed.

Bound 2 is more significant

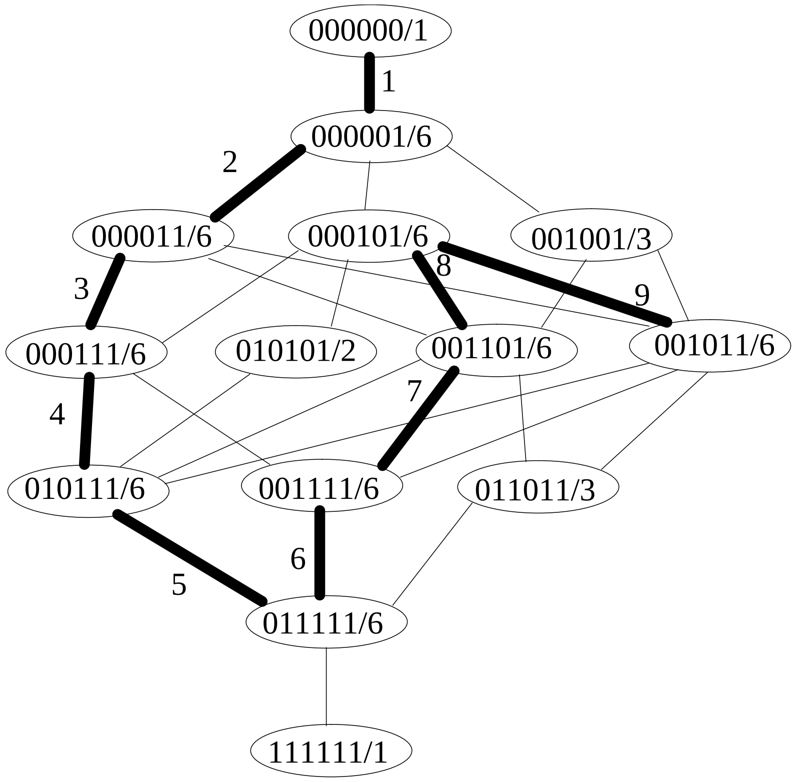
k	ceiling((2^k - 1)/k)
2	2
3	3
4	4
5	7
6	11
7	19
8	32
9	57
10	103
11	187
12	342

Algorithm: Design a restricted one-to-all broadcast tree that can be easily replicated with any processor as the root of the broadcast. The algorithm will use no more than one link in each dimension in each round of communication..

First, we construct a graph that uses the necklace notion to group together processors whose addresses are the same under bitwise rotation. Two necklaces will be connected by an edge if some processor for one necklace is adjacent to some processor for the other necklace. For example, we use k = 6:



A broadcast tree is designed for the value at processor 000000 by using depth-first search (breadth-first is also fine) to navigate among necklaces with k processors:



These 9 edges give the first 9 rounds of communication. The underlined bits emphasize that only one link for each dimension is used in each round.

Round 1:	$000000 \rightarrow 00000\underline{1}$
	$000000 \rightarrow 0000\underline{1}0$
	$000000 \rightarrow 000\underline{1}00$
	$000000 \rightarrow 00\underline{1}000$
	$000000 \rightarrow 0\underline{1}0000$
	$000000 \rightarrow \underline{1}00000$
Round 2:	$000001 \rightarrow 0000\underline{1}1$
	$000010 \rightarrow 000\underline{1}10$
	$000100 \rightarrow 00\underline{1}100$
	$001000 \rightarrow 0\underline{1}1000$
	$010000 \rightarrow \underline{1}10000$
	$100000 \rightarrow 10000\underline{1}$
Round 3:	$000011 \rightarrow 000\underline{1}11$
	$000110 \rightarrow 00\underline{1}110$
	$001100 \rightarrow 0\underline{1}1100$
	$011000 \rightarrow \underline{1}11000$
	$110000 \rightarrow 11000\underline{1}$
	$100001 \rightarrow 1000\underline{1}1$
Round 4:	$000111 \rightarrow 0\underline{1}0111$
	$001110 \rightarrow \underline{1}01110$
	$011100 \rightarrow 01110\underline{1}$
	$111000 \rightarrow 1110\underline{1}0$
	$110001 \rightarrow 110\underline{1}01$
	$100011 \rightarrow 10\underline{1}011$
Round 5:	$010111 \rightarrow 01\underline{1}111$
	$101110 \rightarrow 1\underline{1}1110$
	$011101 \rightarrow \underline{1}11101$
	$111010 \rightarrow 11101\underline{1}$
	$110101 \rightarrow 1101\underline{1}1$
	$101011 \rightarrow 101\underline{1}11$
Round 6:	$011111 \rightarrow 0\underline{0}1111$
	$111110 \rightarrow \underline{0}11110$
	$111101 \rightarrow 11110\underline{0}$
	$111011 \rightarrow 1110\underline{0}1$
	$110111 \rightarrow 110\underline{0}11$
	$101111 \rightarrow 10\underline{0}111$
Round 7:	$001111 \rightarrow 0011\underline{0}1$
	$011110 \rightarrow 011\underline{0}10$
	$111100 \rightarrow 11\underline{0}100$
	$111001 \rightarrow 1\underline{0}1001$
	$110011 \rightarrow \underline{0}10011$
	$100111 \rightarrow 10011\underline{0}$
Round 8:	$001101 \rightarrow 00\underline{0}101$
	$011010 \rightarrow 0\underline{0}1010$
	$110100 \rightarrow \underline{0}10100$
	$101001 \rightarrow 10100\underline{0}$

	$010011 \rightarrow 0100\underline{0}1$	
	$100110 \rightarrow 100\underline{0}10$	
Round 9:	$000101 \rightarrow \underline{1}00101$	
	$001010 \rightarrow 00101\underline{1}$	
	$010100 \rightarrow 0101\underline{1}0$	
	$101000 \rightarrow 101\underline{1}00$	
	$010001 \rightarrow 01\underline{1}001$	
	$100010 \rightarrow 1\underline{1}0010$	

In rounds 10 and 11, the algorithm ‘‘cleans up’’ for the necklaces with < k processors:

Round 10:	$001011 \rightarrow 0010\underline{0}1$	Necklace 001011 to necklace 001001
	$010110 \rightarrow 010\underline{0}10$	
	$101100 \rightarrow 10\underline{0}100$	
	$001011 \rightarrow 0\underline{1}1011$	Necklace 001011 to necklace 011011
	$010110 \rightarrow \underline{1}10110$	
	$101100 \rightarrow 10110\underline{1}$	
Round 11:	$000101 \rightarrow 0\underline{1}0101$	Necklace 000101 to necklace 010101
	$001010 \rightarrow \underline{1}01010$	
	$111110 \rightarrow 11111\underline{1}$	Necklace 011111 to necklace 111111

Several details that guarantee the success of the clean-up rounds have been omitted. These rounds are only needed when k is not prime. There is much flexibility in generating a solution, i.e. there are many alternate schemes.

The last detail is to replicate for broadcasts for other processors besides 000000. This is easily done by taking the bits in the address of the broadcasting processor and exclusive or'ing this address onto the sending and receiving address for each of the transmissions in all rounds.

The end result is that all links will be busy in every round except perhaps the last one.

Sorting Techniques:

<u>Odd-Even Transposition Sort</u>

Uses n/2 rounds (each with two transposition steps) to sort n values on linear array.

$1 \leftrightarrow 6\ \ 7 \leftrightarrow 5\ \ 3 \leftrightarrow 4\ \ 2 \leftrightarrow 0$
$1\ \ 6 \leftrightarrow 5\ \ 7 \leftrightarrow 3\ \ 4 \leftrightarrow 0\ \ 2$

$1 \leftrightarrow 5\ \ 6 \leftrightarrow 3\ \ 7 \leftrightarrow 0\ \ 4 \leftrightarrow 2$
$1\ \ 5 \leftrightarrow 3\ \ 6 \leftrightarrow 0\ \ 7 \leftrightarrow 2\ \ 4$

$1 \leftrightarrow 3\ \ 5 \leftrightarrow 0\ \ 6 \leftrightarrow 2\ \ 7 \leftrightarrow 4$
$1\ \ 3 \leftrightarrow 0\ \ 5 \leftrightarrow 2\ \ 6 \leftrightarrow 4\ \ 7$

$1 \leftrightarrow 0\ \ 3 \leftrightarrow 2\ \ 5 \leftrightarrow 4\ \ 6 \leftrightarrow 7$
$0\ \ 1 \leftrightarrow 2\ \ 3 \leftrightarrow 4\ \ 5 \leftrightarrow 6\ \ 7$

0 1 2 3 4 5 6 7

Mesh Sorting

Unlike other topologies, there is no single "obvious" desirable way to place an ordered sequence onto a mesh for 2 dimensions, much less higher dimensions. It is usually taken that given a good way to obtain a particular arrangement, it is not a problem to permute the arrangement (based on static routing for meshes, discussed earlier). The following "shearsort" algorithm for 2-d meshes is practical (uses odd-even transposition in rows or columns), but not optimal (misses by a logarithmic factor). The algorithm takes N (2log N +1) steps where the mesh is N x N. The output is in "snakelike" ordering for rows.

Phases 1, 3, 5, ... , 2log(N) + 1 sort all rows (in O(N) parallel time for each phase)

Odd rows are sorted so that smaller numbers are at the left. Even rows are sorted so that smaller numbers are at the right.

Phases 2, 4, 6, ..., 2log(N) sort all columns (in O(N) parallel time for each phase)

Columns are sorted with smaller numbers at the top

Example:

10	2	12	8
16	5	1	14
3	9	7	13
6	15	4	11

2	8	10	12
16	14	5	1
3	7	9	13
15	11	6	4

2	7	5	1
3	8	6	4
15	11	9	12
16	14	10	13

1	2	5	7
8	6	4	3
9	11	12	15
16	14	13	10

1	2	4	3
8	6	5	7
9	11	12	10
16	14	13	15

1	2	3	4
8	7	6	5
9	10	11	12
16	15	14	13

Why does it work? The algorithm is oblivious (since odd-even transpose is also oblivious.). By the 0-1 sorting lemma, if an oblivious algorithm will correctly sort any input sequence with 0s and 1s, then the algorithm will sort any sequence correctly. (The proof of the 0-1 sorting lemma shows that if an oblivious sort fails on some sequence, then there exists a sequence of 0s and 1s that will also make the algorithm fail.)

Problems:

1. Route the following permutation on a Benes network and a hypercube (e.g. butterfly):

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 4 & 5 & 6 & 7 & 0 & 1 & 2 & 3 \end{pmatrix}$$

2. Use Gray codes to show that a 64-node hypercube is isomorphic to a 4 x 4 x 4 torus.

3. Show how to achieve the following mesh permutation using perfect matching and linear array sorting.

3,2	3,3	2,2	2,1
3,1	4,2	1,1	1,4
4,4	1,3	2,3	1,2
4,3	2,4	3,4	4,1

4. Derive an all-to-all broadcast scheme for a 4-d hypercube similar to the one used for 6-d hypercubes.

5. How many automorphisms are there for a complete binary tree with h = 4?

6. How many vertex equivalence classes does a 4x5 torus have? A 4x5 mesh?

7. How many automorphisms does a 5-node ring have?

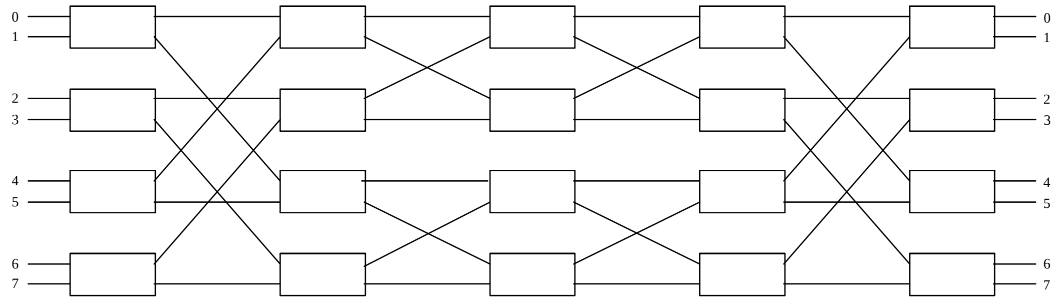
8. For purposes of this exercise only, let us define a class of networks to be *scalable* if a larger network in that class may be constructed from smaller network(s) of that class by only including additional vertices and edges. In particular, the drastic measures of removing edges or vertices are prohibited. Indicate which network classes are scalable and which are not.

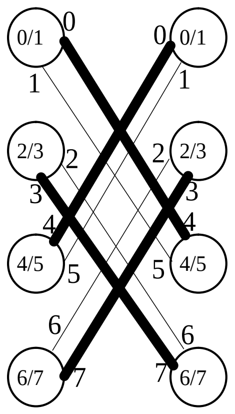
9. Consider an r-dimension array with $N = N_1 = N_r$ and N is odd. What is the bisection width?

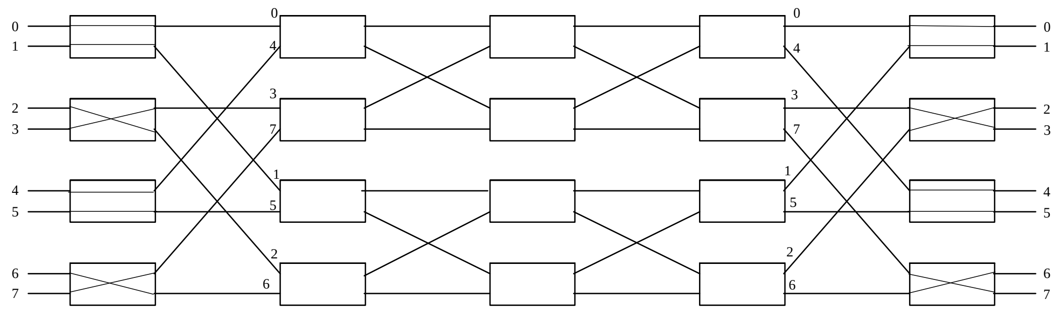
10. Draw the fat tree with depth = 2 and degree = 3.

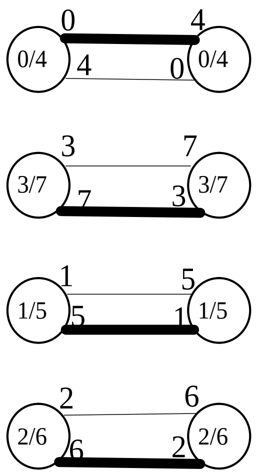
1. Route the following permutation on a Benes network and a hypercube (e.g. butterfly):

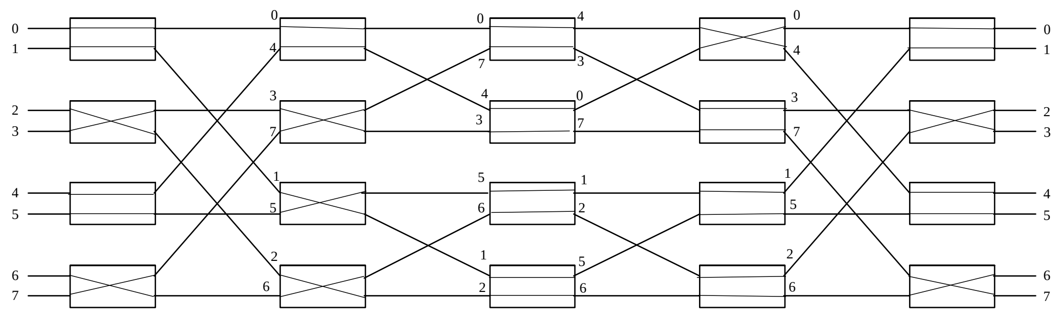
$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 4 & 5 & 6 & 7 & 0 & 1 & 2 & 3 \end{pmatrix}$$

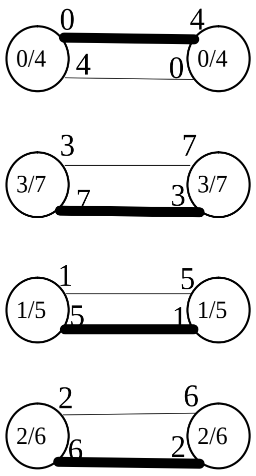



0/1
2/3
4/5
6/7
0/1
2/3
4/5
6/7

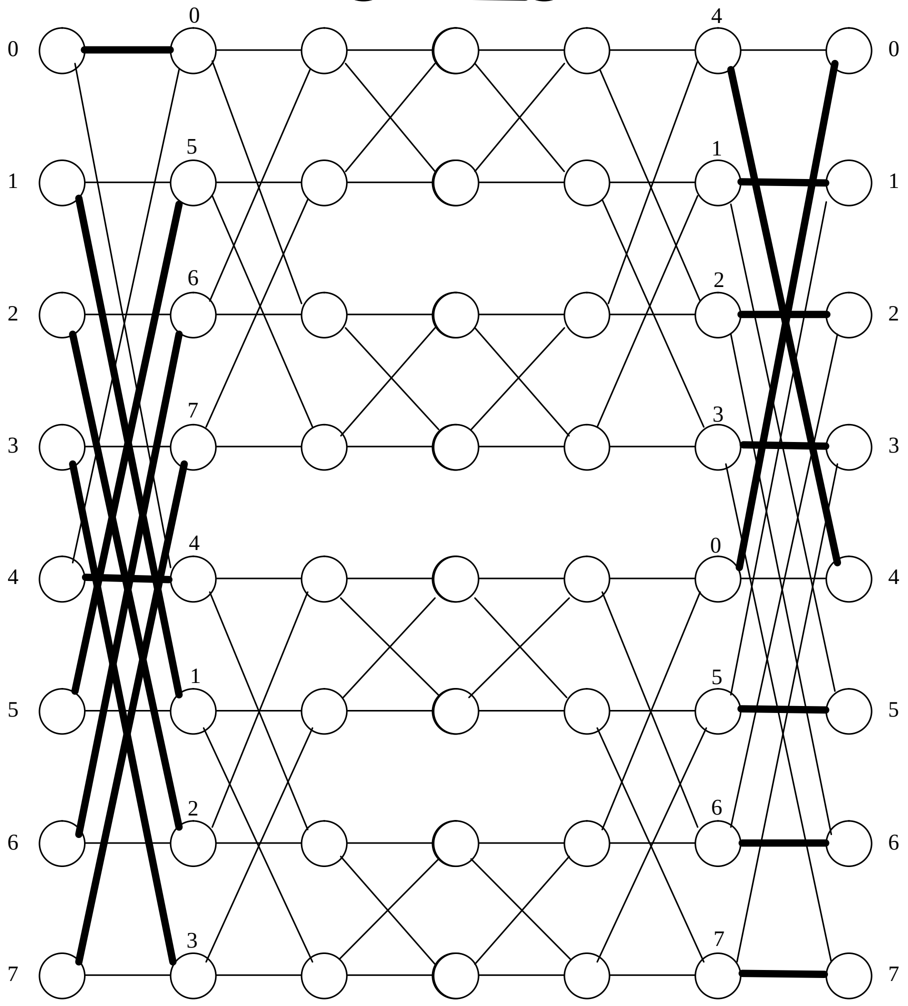



0/4
0/4
3/7
3/7
1/5
1/5
2/6
2/6

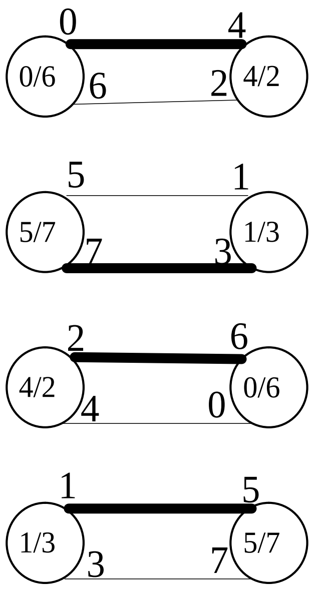


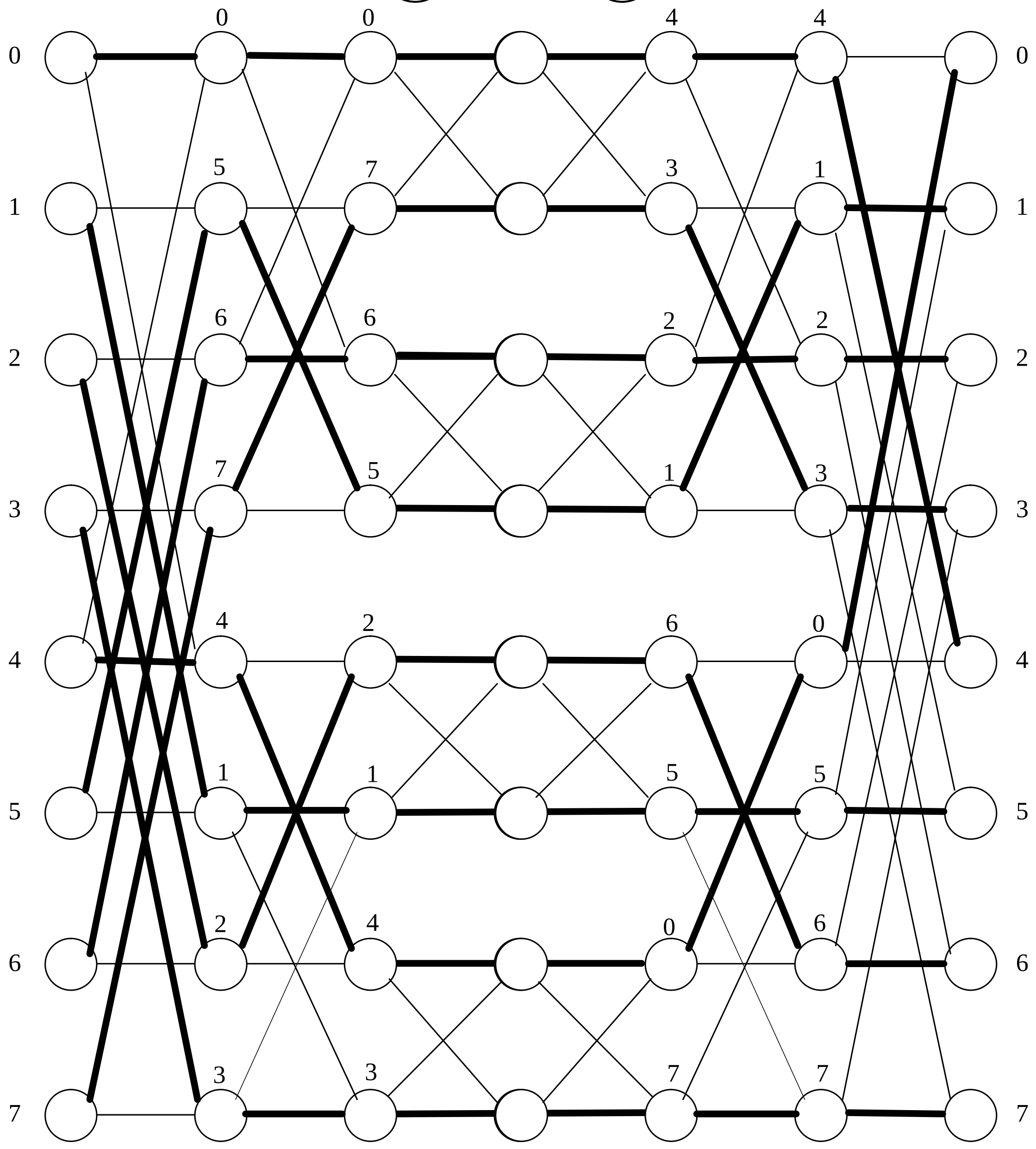


0
4
0/4
4
0
0/4
3
7
3/7
7
3
3/7
1
5
1/5
5
1
1/5
2
6
2/6
6
2
2/6



0
1
2
3
4
5
6
7
0
5
6
7
4
1
2
3
4
1
2
3
0
5
6
7
0
1
2
3
4
5
6
7


0
4
0/6
6
2
4/2
5
1
5/7
7
3
1/3
2
6
4/2
4
0
0/6
1
5
1/3
3
7
5/7


0
1
2
3
4
5
6
7
0
5
6
7
4
1
2
3
0
7
6
5
2
1
4
3
4
3
2
1
6
5
0
7
4
1
2
3
0
5
6
7
0
1
2
3
4
5
6
7

2. Use Gray codes to show that a 64-node hypercube is isomorphic to a 4 x 4 x 4 torus.

Each torus node has a three component address (x, y, z) where each component value is 0, 1, 2, or 3. To map to a hypercube, each of the three components is mapped to two bits in the hypercube address using the 2-bit Gray code:

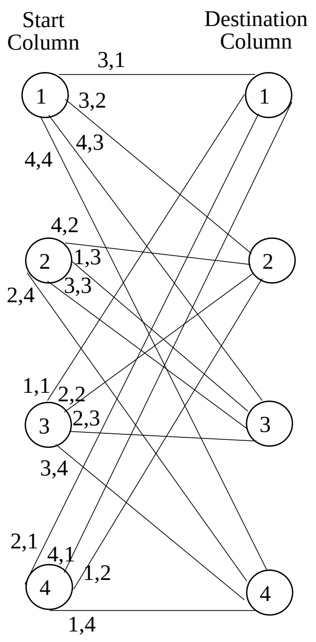
torus address component	hypercube bits
0	00
1	01
2	11
3	10

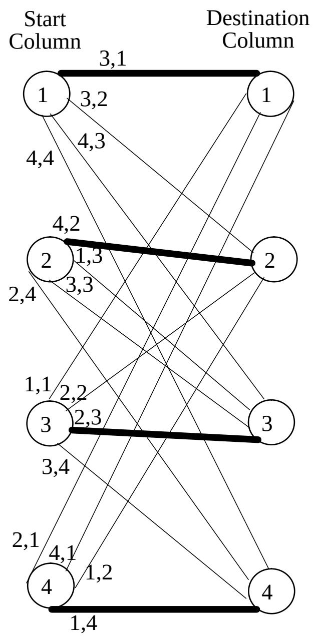
To see that adjacencies are preserved, consider torus node (1, 2, 3) and its neighbors:

torus address	hypercube address
(1, 2, 3)	011110
(0, 2, 3)	$0\underline{0}1110$
(2, 2, 3)	$\underline{1}11110$
(1, 3, 3)	$011\underline{0}10$
(1, 1, 3)	$01\underline{0}110$
(1, 2, 2)	$01111\underline{1}$
(1, 2, 0)	$0111\underline{0}0$

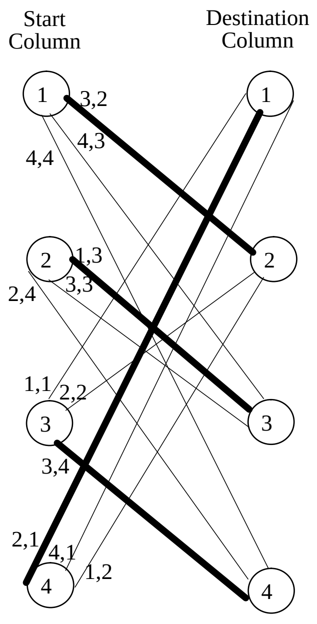
3. Show how to achieve the following mesh permutation using perfect matching and linear array sorting.

3,2	3,3	2,2	2,1
3,1	4,2	1,1	1,4
4,4	1,3	2,3	1,2
4,3	2,4	3,4	4,1

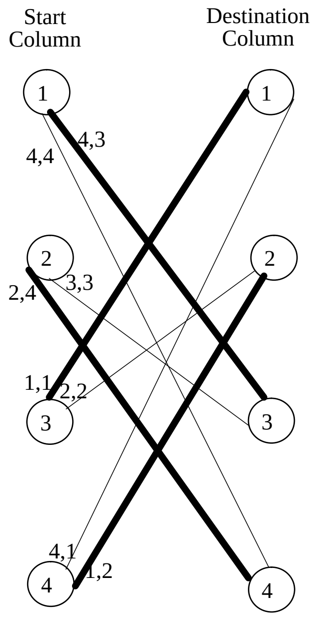




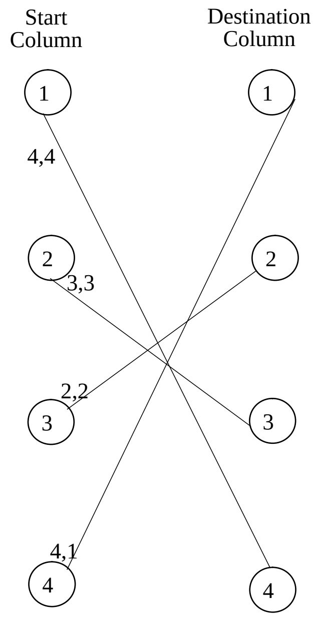
Matching 1



Matching 2



Matching 3



Matching 4

3,1	4,2	2,3	1,4
3,2	1,3	3,4	2,1
4,3	2,4	1,1	1,2
4,4	3,3	2,2	4,1

Routing within columns for four matchings

3,1	4,2	2,3	1,4
2,1	3,2	1,3	3,4
1,1	1,2	4,3	2,4
4,1	2,2	3,3	4,4

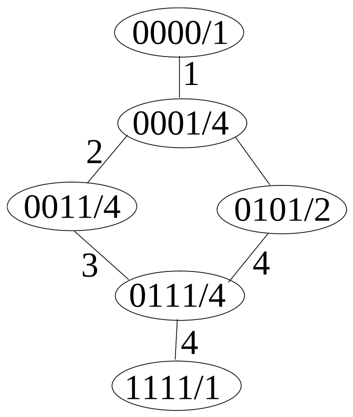
Sort within rows based on column destination

1,1	1,2	1,3	1,4
2,1	2,2	2,3	2,4
3,1	3,2	3,3	3,4
4,1	4,2	4,3	4,4

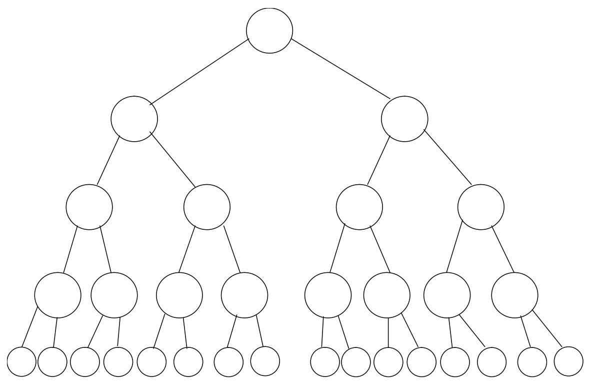
Sort within columns based on column destination

4. Derive an all-to-all broadcast scheme for a 4-d hypercube similar to the one used for 6-d hypercubes.

   Lower bounds indicate that 4 rounds are sufficient.



5. How many automorphisms are there for a complete binary tree with h = 4?



Since each parent node has two orientations for its children, there are 2^{15} = 32768 automorphisms

6. How many vertex equivalence classes does a 4x5 torus have? A 4x5 mesh?

   Torus: 1
   Mesh: 6

7. How many automorphisms does a 5-node ring have?

   10

8. . . . Indicate which network classes are scalable and which are not.

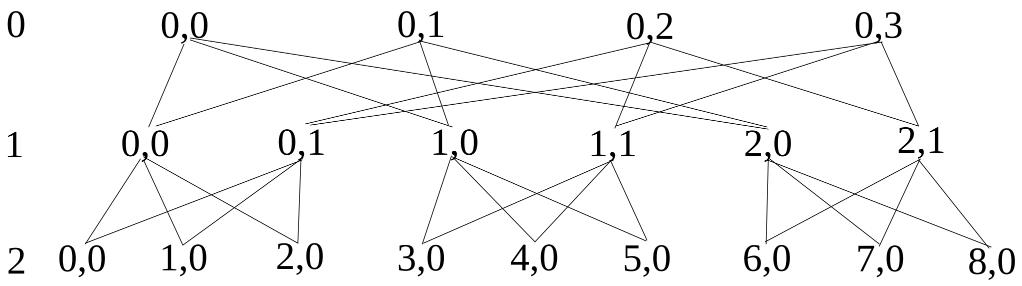
   Scalable: linear array, mesh, binary tree, butterfly, fat tree, hypercube, benes

   Not scalable: ring, torus

9. Consider an r-dimension array with $N = N_1 = N_r$ and N is odd. What is the bisection width?

   $(N^r - 1)/(N - 1)$ by using a geometric sum $(1 + N^1 + N^2 + . . . + N^{r-1})$.

10. Draw the fat tree with depth = 2 and degree = 3.


0
0,0
0,1
0,2
0,3
1
0,0
0,1
1,0
1,1
2,0
2,1
2
0,0
1,0
2,0
3,0
4,0
5,0
6,0
7,0
8,0