# COMP90025 Parallel and Multicore Computing

## Interconnection networks

Lachlan Andrew

School of Computing and Information Systems
The University of Melbourne

2019 Semester II

## Types of networks

Both the shared memory and distributed memory architectures require an interconnection network to connect the processor and memory or the modules respectively.

Interconnection networks can be broadly categorized into three types:

1. buses,
2. static networks and
3. switching networks.

Static networks are sometimes called point-to-point networks.

The interesting aspects to consider are the complexity of communication between the processors and memory, or between modules, and the physical cost (wiring complexity) of the network.

## Multi-bus systems

A full $b$-bus memory connection system with $p$ processors and $m$ memories uses $b\,(p+m)$ connections (not counting the buses themselves).

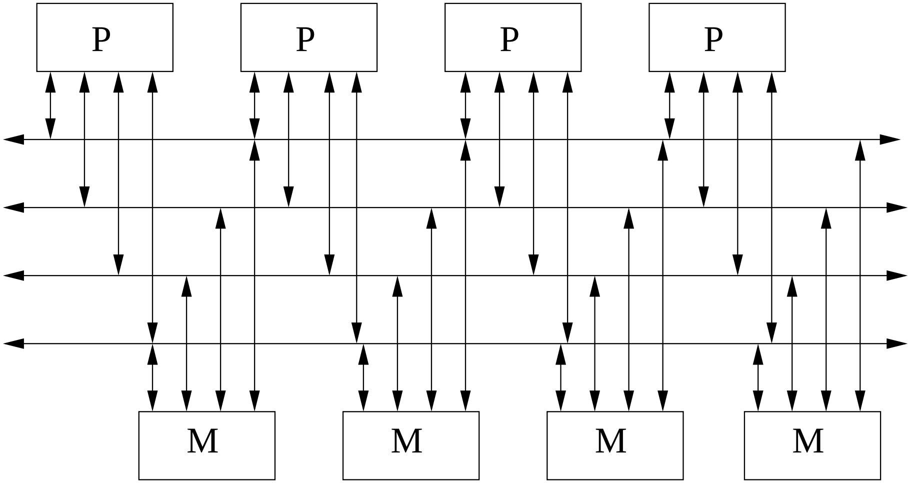


Figure: full memory connection

A partial $b$-bus memory connection system with $p$ processors and $m$ memories uses $b\,(p + \frac{m}{g})$ connections where $1 < g < b$.

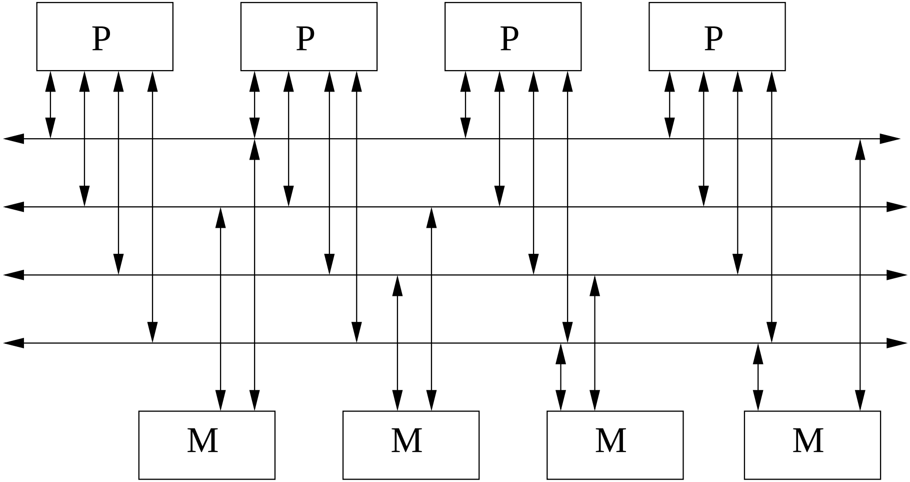


Figure: partial memory connection

A single $b$-bus memory connection system with $p$ processors and $m$ memories uses $b\,p + m$ connections.

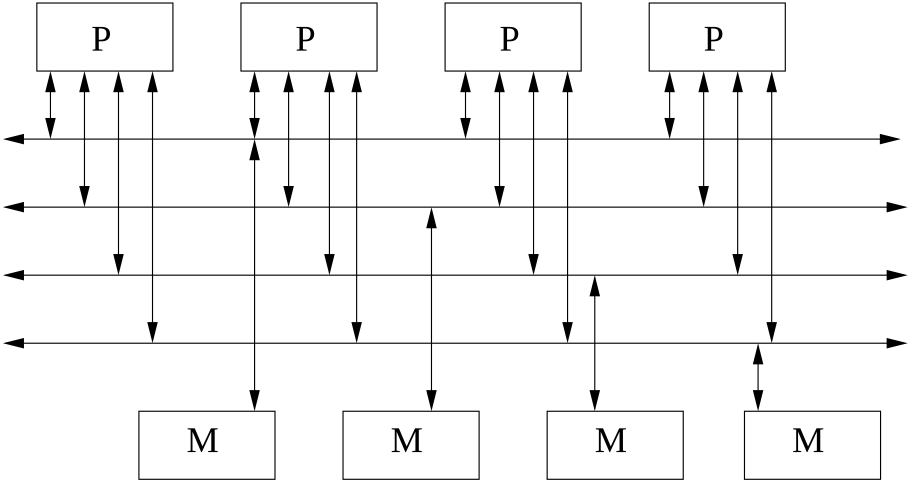


Figure: single memory connection

## Bus characteristics

Access to a bus is arbitrated by a *bus master*.

Each node on a bus has a bus master which requests access to the bus, called a *bus request*, when then node requires to use the bus. This is a global request sent to all nodes on the bus.

The node that currently has access to the bus responds with either a *bus grant* or a *bus busy* signal, which is also globally known to all bus masters.

If all $n$ nodes on a bus are continuously requesting access then a bus can supply $O(1/n)$ of its available bandwidth to each node.

The *minimum latency* on a bus, i.e. with no contention, is constant or $O(1)$ (which ignores signal propagation delay).

## Static networks

A static network has fixed point-to-point connections between modules in the system. Each connection is a dedicated communication channel between the pair of modules.

Message passing is required between the modules for any module to access the memory of any other module (for a shared addresss space machine) or for processors to communicate (in a message passing machine).

The number of “hops” that a message requires is a dominant factor determining the latency for the memory access or communication.

The bandwidth of the connections and/or the routing technology/algorithms also determine the latency.

## Communication module

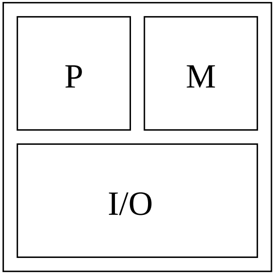


Figure: Processor, memory and communication module.

The communication module may be as straight forward as an Ethernet connection, or multiple Ethernet connections, or may be more dedicated hardware.

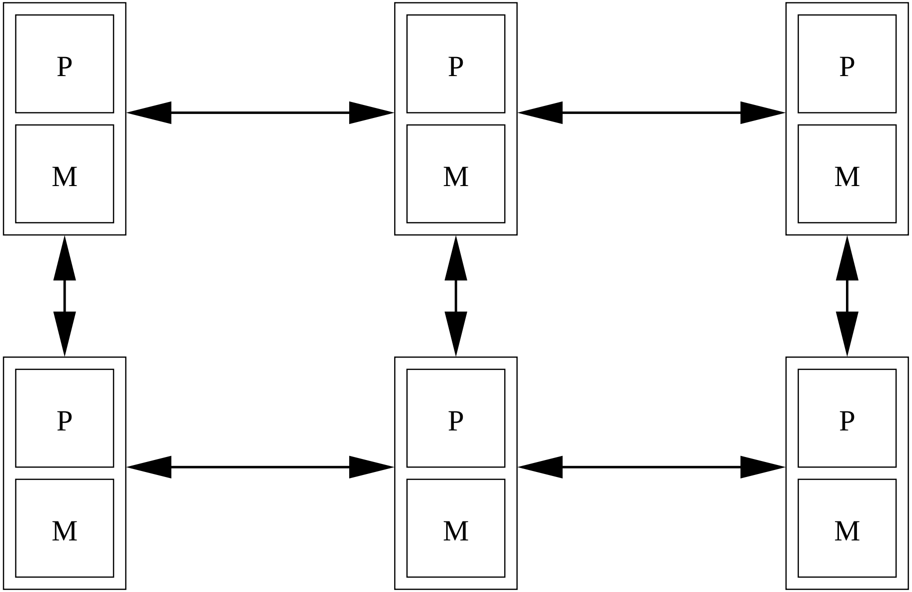


Figure: Distributed memory machine example

## Topological properties of static networks

Topological properties are used to describe a given architecture.
For static networks it is desirable to achieve a low "*cost*":
$c = d\,k$

| | |
|---|---|
| degree ($d$) | : the maximum number of edges connected to a single vertex in the graph. |
| diameter ($k$) | : the minimum number of edges a message must traverse in order to be communicated between any two vertices in the graph. |

Note that this "cost" is a compromise between the cost to build ($d$) and the communication delay ($k$), not just the cost to build.

Other properties of interest:

| | |
|---|---|
| bisection width | : The smallest number of edges that, when cut, would separate the network into two halves. Sometimes we call this edge bisection width and then also consider node bisection width. |
| planarity | : Can the network be embedded in a plane without any edges crossing. This is particular useful for integrated circuits as it eliminates the need for multiple layers. |
| symmetry | : Are all nodes topologically the same. |

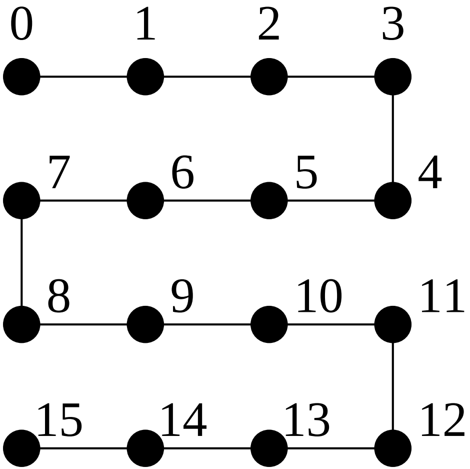


Figure: A linear array.

Degree 2 and diameter $O(n)$, has cost $O(n)$.

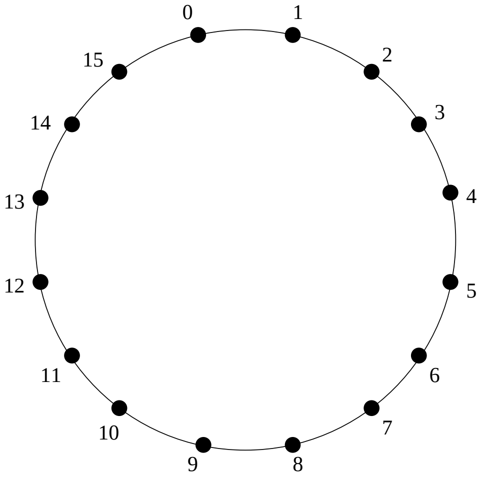


Figure: A ring network.

Generally the ring network with $n$ vertices has $k = \left\lfloor \frac{n}{2} \right\rfloor$ and $d = 2$ so the cost is $2 \left\lfloor \frac{n}{2} \right\rfloor = O(n)$.

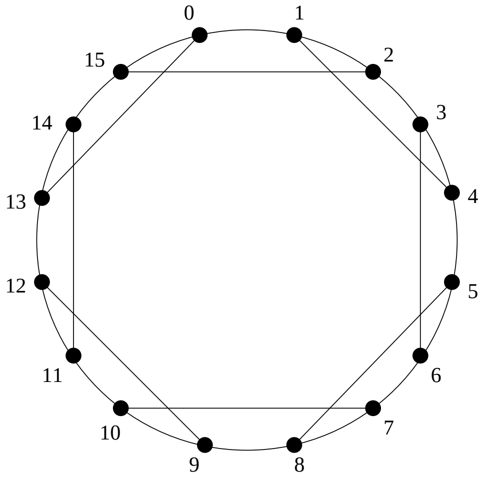


Figure: A chordal ring of degree 3.

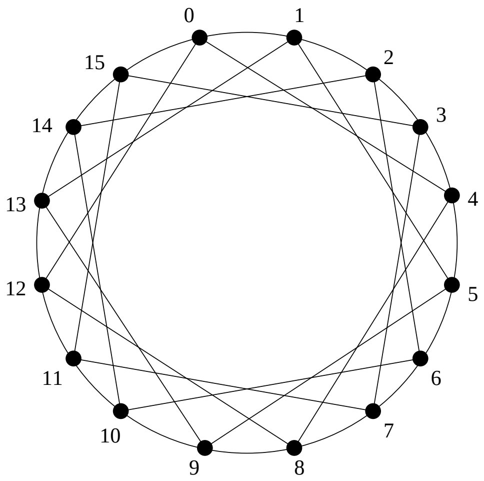


Figure: A chordal ring of degree 4.

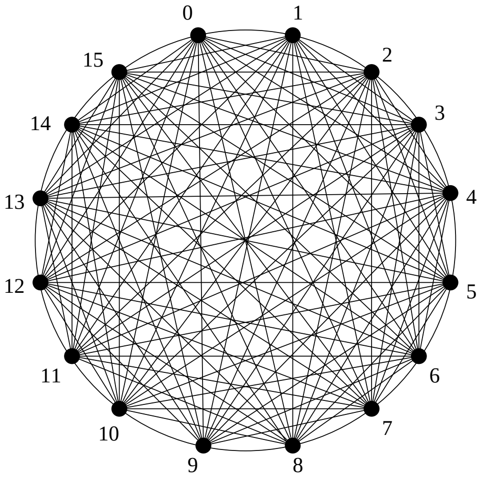


Figure: A completely connected network.

The cost of a completely connected network is $O(n)$.

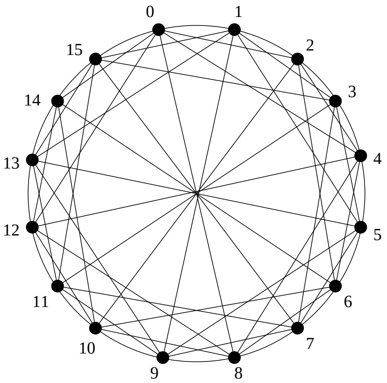
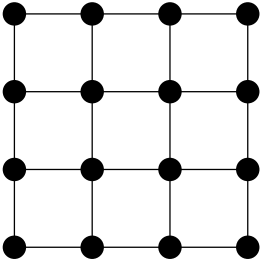


Figure: Barrel shifter.

Node $i$ is connected to node $j$, if $|j - i| = 2^r$ for some $r = 0, 1, 2, \ldots, k-1$, where $n = 2^k$. The degree is $2k - 1$ and the diameter is $k/2$. Cost is $O(\log(n)^2)$



$d = 4$

$k = 6$

$c = d\,k = 24$

Figure: A mesh network.

Generally the mesh network with $n = i \times j$ vertices has $k = i + j - 2$ and $d = 4$ so the cost is $4(i + j) - 8$ which is $8\sqrt{n} - 8 = O\left(\sqrt{n}\right)$ if $i = j$. Clearly the mesh has a smaller cost than a ring when $n$ is large.

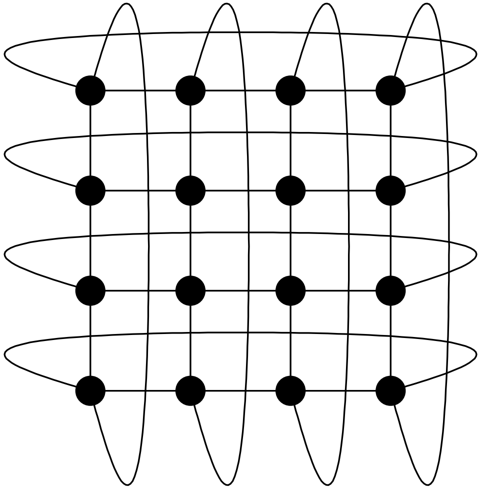
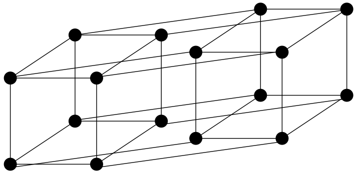


Figure: A torus network.



$d = 4$

$k = 4$

$c = d\,k = 16$

Figure: A hypercube network.

Generally the hypercube network with $n = 2^t$ vertices has $k = t$ and $d = t$ so the cost is $t^2$ or $\log_2^2 n = O\big(\log^2 n\big)$. The cost is far less than the previous two examples, for large $n$.

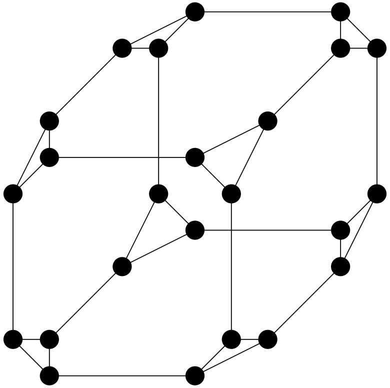
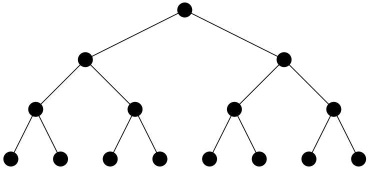


Figure: A CCC network: Cube Connected Cycles

A network called Cube Connected Cycles (CCC), is constructed from a hypercube of size $n = 2^t$ nodes by replacing each node (of degree $t$) with a cycle of length $t$ and connecting each incoming edge to one node of the cycle. Each node in the cycle now has degree 3.

## Cube Connected Cycles

- The CCC network will now have $t\,2^t$ nodes.
- The degree is 3.
- The diameter is non-trivial to show; see the literature for a proof that it is $2\,t + \lfloor \frac{t}{2} \rfloor - 2$ for $t > 3$.
  - Check by inspection that diameter is 6 for $t = 3$.
- Prove for yourself that $t = \Theta(\frac{\log n}{\log \log n})$, which gives a cost of $O(\frac{\log n}{\log \log n})$.



$d = 3$

$k = 6$

$c = d\,k = 18$

Figure: A tree network.

For a tree with degree $d$ and $n$ vertices, $k \leq 2\lfloor \log_{d-1} n \rfloor = O(\log_d n)$ (twice the height of the tree) so that the cost is $O(d \log_d n)$. Setting $d = \log n$ gives a cost of $O\left(\frac{\log^2 n}{\log \log n}\right)$ which is better than the hypercube.

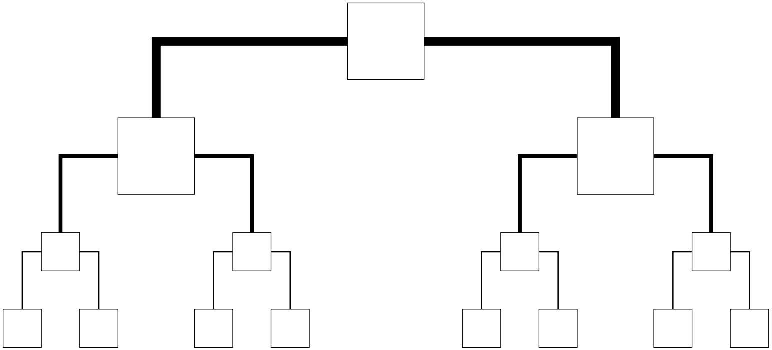


Figure: Fat Tree

The fat tree is a practical extension of the tree that has progressively more wiring towards the root of the tree. This provides for greater data flow between the subtrees.

## Data centre "fat tree"

- K-ary fat tree: three-layer topology (edge, aggregation and core)
  - each pod consists of (k/2) 2 servers + 2 layers of k/2 k-port switches
  - each edge switch connects to k/2 servers + k/2 aggr. switches
  - each aggr. switch connects to k/2 edge + k/2 core switches
  - (k/2) 2 core switches: each connects to k pods

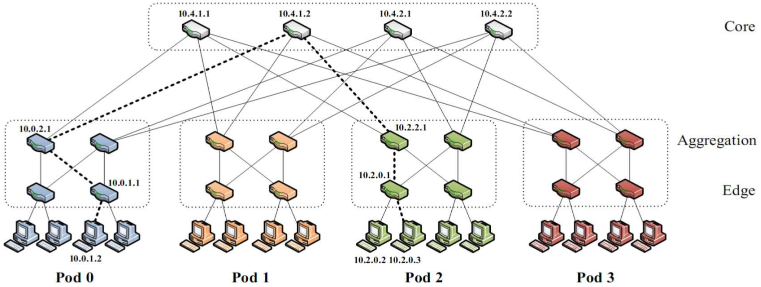


Figure: "Fat tree" with commodity switches

- Top two layers aren't processor/memory/communication modules.
- We now consider: What do these look like internally?

## Switching networks

A basic switch has two inputs, say $A$ and $B$ and two outputs, say $A'$ and $B'$. It is either in the direct setting where $A' = A$ and $B' = B$, or in the cross-over setting where $A' = B$ and $B' = A$.

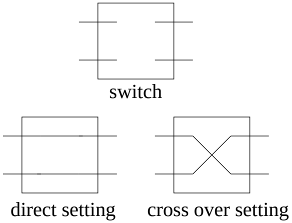


Figure: A switch.

With switching networks, the hardare cost is no longer proportional to the degree $d$. Instead, we measure the complexity as the number of switching elements.

Other settings are also possible.

A *crossbar* switching network has complexity $O\left(n^2\right)$ where $n$ is the number of inputs and outputs. The $n \times n$ crossbar is comparable to an $n$-bus with single bus memory connections.

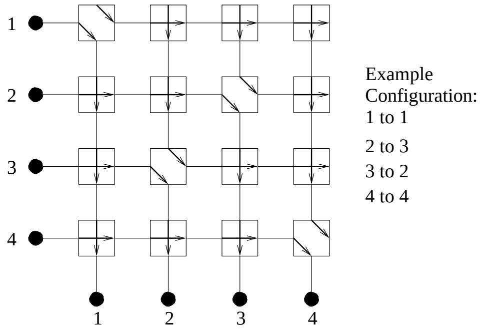


Figure: An example crossbar switching network.

The minimum latency for a crossbar is $O(1)$, i.e. any 1-to-1 mapping of inputs to outputs is possible.

## Clos Network

The Clos network is defined by integers $n$, $m$ and $r$.

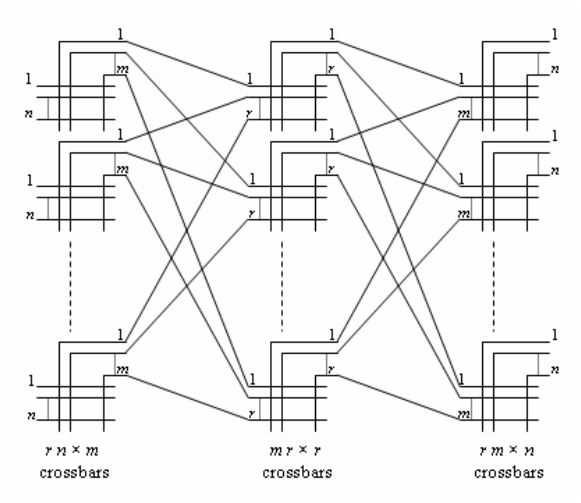


Figure: A Clos Network.

The Clos network connects $N = rn$ inputs to $N$ outputs. A straight forward crossbar would require $N^2 = r^2n^2$ switches. However the Clos network requires $2rnm + mr^2$ switches.

If $m \geq 2n - 1$, the Clos network is said to be *strict-sense nonblocking*, meaning that an unused input on an ingress switch can always be connected to an unsed output on an egress switch, *without having to re-arrange any existing connections.*

If $m \geq n$, the Clos network is said to be rearrangeably nonblocking, meaning that an unused input on an ingress switch can always be connected to an unused output on an egress switch, but for this to take place existing connections may need to be rearranged in the center stage.

If we make $m = n$ then we have $2rn^2 + nr^2$ switches or $2Nn + Nr$, which provides a lower switching complexity than $N^2$.

## Multi-stage Clos Network

Substitute each of the $r \times r$ crossbars in the center stage with a Clos network, which increases the total stages from 3 to 5.
This reduces the switching complexity of the center stage.
Further substitution can be undertaken to created Clos networks with 7, 9, 11 stages, etc.

## Beneš Switching Network

A rearrangeably nonblocking Clos network with $m = n = 2$ is generally called a Beneš Network. The number of inputs and outputs is $N = r \times n = 2r$, with $2\log_2 N - 1$ stages, each containing $\frac{N}{2}$ switches, totalling $N \log_2 N - \frac{N}{2}$ switches. An $8 \times 8$ Beneš Network is shown below.

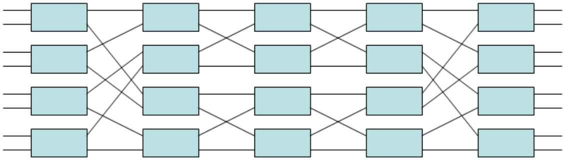


Figure: A $8 \times 8$ Beneš Switching Network.

- Here $r = 4$. Identify the two $r \times r$ "switches" in the middle, each consisting of a 3-stage Clos network.

The minimum latency for a Beneš network is again $O(1)$, i.e. any 1-to-1 mapping of inputs to outputs is still possible.

## Omega Switching Network

An *Omega* switching network has complexity $O(n \log n)$ with $n$ inputs and $n$ outputs. In general, an $n$-input Omega network requires $\log_2 n$ stages of $2 \times 2$ switches. Each stage requires $n/2$ switch modules. In total, the network uses $(n/2)\log_2 n$ switches.

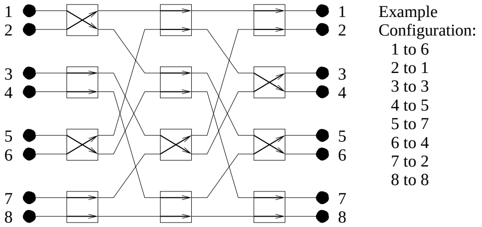


Figure: An $8 \times 8$ Omega switching network.

Uses roughly half as many $2 \times 2$ switches as a Beneš network. minimum latency increased to $O(\log n)$, i.e. some 1-to-1 I/O mappings require up to $\log n$ phases to be realized. Consider $1 \rightarrow 1$ and $5 \rightarrow 2$.

## Baseline Switching Network

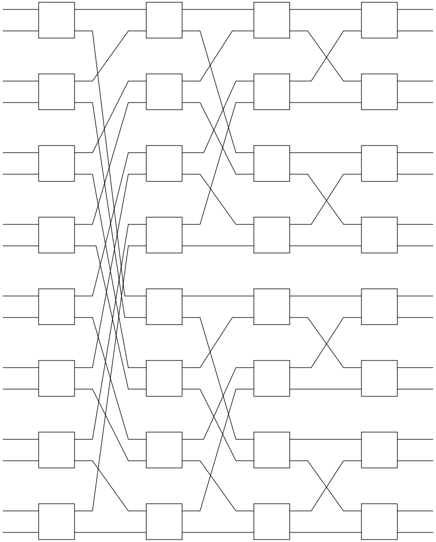


Figure: A 16 × 16 Baseline Switching Network.

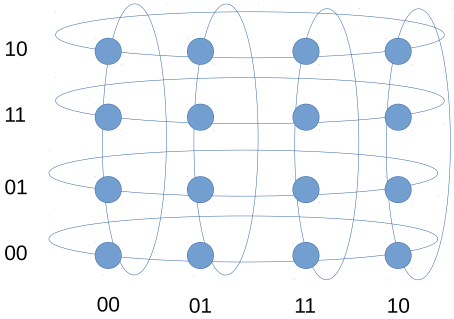
## Embedding mesh topologies in a hypercube

Any mesh whose size is a power of two (flat, torus or with partial wrap-around) can be embedded in a hypercube. That is, it can be formed by "disabling" some links in the hypercube.

- The hypercube is $\{0,1\}^d$. Each node's address is the coordinates in Euclidean space, a $d$-bit binary number
- The size of each dimension of the mesh is a power of two.
- The coordinates of each node in a $k$-dimensional mesh are a $k$-tuple, each of $\log_2(d_i)$ bits, where $d_i$ is the size in dimension $i$.
- Claim: If we partition the $d$ bits of the hypercube into $k$ groups, of sizes $d_1, \ldots, d_k$, then we can choose a numbering scheme for each coordinate $i$ such that all of the links needed to make up the mesh are present in the hypercube.
  - That is, the mesh has been *embedded* into the hypercube

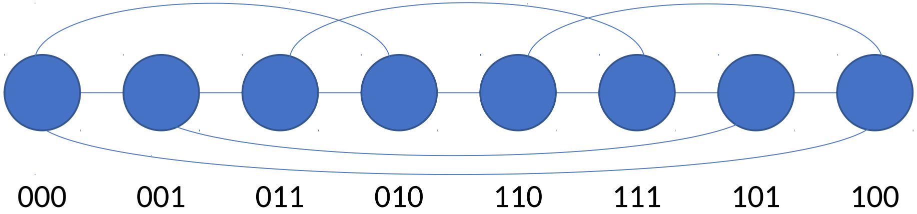
## Trivial embeddings

- A $2 \times 2$ grid—or $2 \times 2 \times 2 \times \ldots \times 2$— is already a hypercube
- A $4 \times 4 \times \cdots \times 4$ grid is exactly a hypercube if the numbering is changed slightly



## Embeddings by omitting edges

A mesh with side lengths of 8 or more requiers some edges from the hypercube to be ignored.



Similarly, linear topologies can be embedded in meshes. The 16-node linear network in an earlier slide can be seen to be part of a $4 \times 4$ mesh These embeddings require fiddling with the order of bits, so that only one bit changes between neighbours. Is that always possible?

## Gray code

- Gray code is a reordering of *d*-bit integers, for any *d*, so that only one bit changes at a time,
- Which bit changes?
  - 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0, 3
  - To find the pattern, consider the most significant bit that would change with “normal” counting.

```
int gray = 0;
int i;

for (i = 0; i < d; i++) {
    printf  ("Gray code %d\n", gray);
    int normal_change = i ^ ((i+1) % d);
    int bit = normal_change ^ (normal_change >> 1);
    gray ^= bit;
}
```

- Only one bit still changes for wrapping from the last value back to 0.
- However, the `(i+1)` in the code must change to `((i+1) % d)`